

ETH Lecture 401-0674-00L Numerical Methods for (Partial) Differential Equations

Numerical Methods for (Partial) Differential Equations

Prof. R. Hiptmair, SAM, ETH Zurich

Spring Term 2023

(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this lecture document



Always under construction!

The online version will always be work in progress and subject to change.

(Nevertheless, structure and main contents can be expected to be stable)

Contents

0 Introduction	8
0.1 General Information	16
0.1.1 Goals	16
0.1.2 Teaching Model	16
0.1.2.1 Course Videos	17
0.1.2.2 Following the Course	24
0.1.2.3 Learning Model	25
0.1.3 Homework assignments	26
0.1.4 Information on Examinations	27
0.2 Coding Projects	28
0.2.1 Requests	28
0.3 Prerequisites for this Course	29
0.3.1 Tools from Linear Algebra	29
0.3.1.1 Basic Notations	30
0.3.1.2 Linear and Bilinear Forms	31
0.3.1.3 Norms and Inner Products	32
0.3.1.4 Diagonalization of Matrices	34
0.3.2 Tools from Real Analysis	35
0.3.2.1 Calculus in 1D	36
0.3.2.2 Differentiation in Multiple Dimensions	36
0.3.2.3 Spaces of Continuously Differentiable Functions	38
0.3.2.4 Norms on Function Spaces	40
0.3.2.5 Multi-Dimensional Integration	41
0.3.3 Required Elementary Numerical Methods	44
0.4 Mathematical Modelling with Partial Differential Equations	44
0.4.1 PDEs: Basic Notions	44
0.4.2 Electromagnetics: Eddy Current Problem	45
0.4.3 Viscous Fluid Flow	46
0.4.4 Micromagnetics	46
0.4.5 Reaction-Diffusion: Phase Separation	47
0.4.6 Activator-Inhibitor Models: Fur Patterns	48
0.4.7 Quantitative Finance: Black-Scholes Equation	48
0.4.8 Quantum Mechanics: Electronic Schrödinger Equation	49
0.4.9 Rarefied Gas Dynamics: Boltzmann Equation	50
0.4.10 Wave Propagation: Helmholtz equation	51
1 Second-Order Scalar Elliptic Boundary Value Problems	54
1.1 Preface	54
1.2 Equilibrium Models: Examples	56
1.2.1 Elastic Membranes	56

1.2.1.1	Configuration Spaces	56
1.2.1.2	Equilibrium Conditions	59
1.2.1.3	Smoothness of Displacements	62
1.2.2	Electrostatic Fields	65
1.2.3	Quadratic Minimization Problems	68
1.2.3.1	Definition	69
1.2.3.2	Existence and Uniqueness of Minimizers	72
1.2.3.3	Energy Norm	74
1.2.3.4	A Continuity Condition for the Linear Functional ℓ	75
1.3	Sobolev spaces	80
1.3.1	Function Spaces for Energy Minimization	81
1.3.2	The Function Space $L^2(\Omega)$	82
1.3.3	Supplement: Quadratic Minimization Problems on Hilbert Spaces	84
1.3.4	The Sobolev Space $H^1(\Omega)$	87
1.4	Linear Variational Problems	96
1.4.1	Quadratic Variational Calculus	97
1.4.2	Linear Second-Order Elliptic Variational Problems	99
1.4.3	Stability of Second-Order Elliptic Linear Variational Problems	100
1.5	Equilibrium Models: Boundary Value Problems	103
1.5.1	Two-Point Boundary Value Problems	103
1.5.2	Integration by parts in higher dimensions	106
1.5.3	Linear Scalar Second-order Elliptic Partial Differential Equations	109
1.6	Diffusion Models (Stationary Heat Conduction)	115
1.7	Boundary Conditions	118
1.8	Second-Order Elliptic Variational Problems	120
1.9	Essential and Natural Boundary Conditions	129
2	Finite Element Methods (FEM)	136
2.1	Introduction	137
2.1.1	Discretization	138
2.1.2	Focus: Variational Formulations of BVPs	138
2.1.3	Preview	140
2.2	Principles of Galerkin Discretization	141
2.2.1	Galerkin Discretization: First Step	142
2.2.2	Galerkin Discretization: Second Step	144
2.2.3	Galerkin Matrices	147
2.3	Case Study: Linear FEM for Two-Point Boundary Value Problems	150
2.3.1	Step I: Choice of Discrete Trial/Test Space	150
2.3.2	Step II: Choice of Basis	152
2.3.3	Formulas for Galerkin Matrices and Right-Hand-Side Vectors	153
2.4	Case Study: Triangular Linear FEM in Two Dimensions	160
2.4.1	Meshes in 2D: Triangulations	161
2.4.2	Linear Finite Element Space	163
2.4.3	Nodal Basis Functions	165
2.4.4	Sparse Galerkin Matrix	168
2.4.5	Computation of Galerkin Matrix	170
2.4.5.1	Local Computations	171
2.4.5.2	Assembly of Full Galerkin Matrix	175
2.4.6	Computation of Right-Hand Side Vector	181
2.5	Building Blocks of General Finite Element Methods	188
2.5.1	Meshes	188

2.5.2	Polynomials	190
2.5.3	Basis Functions	192
2.6	Lagrangian Finite Element Methods	197
2.6.1	Simplicial Lagrangian FEM	198
2.6.2	Tensor-Product Lagrangian FEM	201
2.7	Implementation of Finite Element Methods	207
2.7.1	Mesh Generation and Mesh File Format	209
2.7.2	Mesh Information and Mesh Data Structures	216
2.7.2.1	LEHRFEM++ Mesh: Container Layer	218
2.7.2.2	LEHRFEM++ Mesh: Topology Layer	219
2.7.2.3	LEHRFEM++ Mesh: Geometry Layer	226
2.7.3	Vectors and Matrices	228
2.7.4	Assembly Algorithms	230
2.7.4.1	Assembly: Localization	230
2.7.4.2	Assembly: Index Mappings	232
2.7.4.3	Distribute Assembly Schemes	237
2.7.4.4	Assembly: Linear Algebra Perspective	246
2.7.5	Local Computations	249
2.7.5.1	Analytic Formulas for Entries of Element Matrices	249
2.7.5.2	Local Quadrature	252
2.7.6	Treatment of Essential Boundary Conditions	262
2.8	Parametric Finite Element Methods	271
2.8.1	Affine Equivalence	271
2.8.2	Example: Quadrilateral Lagrangian Finite Elements	276
2.8.3	Transformation Techniques	279
2.8.4	Application of Parametric FEM: Boundary Approximation	292
3	FEM: Convergence and Accuracy	300
3.1	Abstract Galerkin Error Estimates	301
3.1.1	Setting	301
3.1.2	Galerkin Discretization Error	302
3.1.3	Galerkin Orthogonality (GO)	304
3.1.4	Refinement	305
3.2	Empirical (Asymptotic) Convergence of Lagrangian FEM	309
3.2.1	Asymptotic Convergence	310
3.2.2	Algebraic and Exponential Convergence	311
3.2.3	Convergence of FEM: Numerical Experiments	314
3.3	A Priori (Asymptotic) Finite Element Error Estimates	324
3.3.1	Estimates for Linear Interpolation in 1D	325
3.3.2	Error Estimates for Linear Interpolation in 2D	329
3.3.3	The Sobolev Scale of Function Spaces	336
3.3.4	Anisotropic Interpolation Error Estimates	338
3.3.5	General Approximation Error Estimates for Lagrangian FEM: A Survey	341
3.4	Elliptic Regularity Theory	350
3.5	Variational Crimes	355
3.5.1	The Impact of Numerical Quadrature	357
3.5.2	Approximation of the Boundary	358
3.6	FEM: Duality Techniques for Error Estimation	362
3.6.1	Linear Output Functionals	362
3.6.2	Case Study: Computation of Boundary Fluxes with FEM	366
3.6.3	Lagrangian FEM: L^2 -Estimates	372

3.7	Discrete Maximum Principle	377
3.7.1	Maximum Principle for Scalar 2nd-Order Elliptic BVPs	377
3.7.2	Maximum Principle for Piecewise Linear Lagrangian FEM	379
3.8	Validation and Debugging of Finite Element Codes	387
4	Beyond FEM: Alternative Discretizations	394
4.1	Finite-Difference Methods (FDMs)	394
4.1.1	Finite-Difference Method for Two-Point Boundary-Value Problems	395
4.1.2	Finite-Difference Method in Two Dimensions	400
4.1.3	Finite Differences and Finite Elements	404
4.2	Finite-Volume Methods (FVMs)	408
4.2.1	Discrete Balance Laws	409
4.2.2	Dual Meshes	410
4.2.3	Relationship of FEM and FVM	414
4.3	Spectral Galerkin Methods	418
4.4	Collocation Methods	427
4.4.1	Spectral Collocation Method	429
4.4.2	Spline Collocation Methods	431
5	Non-Linear Elliptic Boundary Value Problems	435
5.1	Non-linear Elastic Membrane Models	435
5.1.1	Thin Elastic String Model	435
5.1.1.1	Configuration Space (Recalled)	435
5.1.1.2	Mass-Spring Model	435
5.1.1.3	Continuum Limit	435
5.1.2	Thin Membrane Model	435
5.1.3	Taut Membrane Limit	435
5.2	Variational Approach	435
5.2.1	Virtual Work Equation	435
5.2.2	Non-Linear Variational Equations	435
5.2.3	Elastic Membrane Boundary Value Problem	435
5.3	Ritz-Galerkin Discretization	435
5.3.1	Abstract Galerkin Discretization on Non-Linear Variational Problems	435
5.3.2	Newton's Method in Function Space	435
5.3.3	Galerkin Discretization of Linearized Variational Problems	435
6	Numerical Integration – Single Step Methods	436
6.1	Initial-Value Problems (IVPs) for Ordinary Differential Equations (ODEs)	437
6.1.1	Ordinary Differential Equations (ODEs)	437
6.1.2	Mathematical Modeling with Ordinary Differential Equations: Examples	439
6.1.3	Theory of Initial-Value-Problems (IVPs)	444
6.1.4	Evolution Operators	448
6.2	Introduction: Polygonal Approximation Methods	452
6.2.1	Explicit Euler method	453
6.2.2	Implicit Euler method	455
6.2.3	Implicit midpoint method	457
6.3	General Single-Step Methods	458
6.3.1	Definition	459
6.3.2	(Asymptotic) Convergence of Single-Step Methods	463
6.4	Explicit Runge-Kutta Single-Step Methods (RKSSMs)	472
6.5	Adaptive Stepsize Control	479
6.5.1	The Need for Timestep Adaptation	479

6.5.2	Local-in-Time Stepsize Control	481
6.5.3	Embedded Runge-Kutta Methods	489
7	Single-Step Methods for Stiff Initial-Value Problems	495
7.1	Model Problem Analysis	496
7.2	Stiff Initial-Value Problems	509
7.3	Implicit Runge-Kutta Single-Step Methods	514
7.3.1	The Implicit Euler Method for Stiff IVPs	515
7.3.2	Collocation Single-Step Methods	516
7.3.3	General Implicit Runge-Kutta Single-Step Methods (RK-SSMs)	520
7.3.4	Model Problem Analysis for Implicit Runge-Kutta Single-Step Methods (IRK-SSMs)	522
7.4	Semi-Implicit Runge-Kutta Methods	528
7.5	Splitting Methods	532
8	Structure-Preserving Numerical Integration	538
9	Second-Order Linear Evolution Problems	539
9.1	Time-Dependent Boundary Value Problems	539
9.2	Parabolic Initial-Boundary Value Problems	541
9.2.1	Heat Equation	541
9.2.2	Heat Equation: Spatial Variational Formulation	545
9.2.3	Stability of Parabolic Evolution Problems	548
9.2.4	Spatial Semi-Discretization: Method of Lines	552
9.2.5	Recalled from Section 6.1: Ordinary Differential Equations	554
9.2.6	Recalled from § 10.2.2.7: Single-Step Methods for Numerical Integration	557
9.2.6.1	Abstract Single-Step Methods for ODEs	558
9.2.6.2	Simple Single-Step Methods	558
9.2.6.3	Single-Step Method for Initial-Value Problems	561
9.2.6.4	Collocation Single-Step Methods	563
9.2.6.5	Runge-Kutta Single-Step Methods	567
9.2.7	Timestepping for Method-of-Lines ODE	568
9.2.7.1	Single-Step Methods Applied to MOL ODE	569
9.2.7.2	Stability	570
9.2.8	Fully Discrete Method of Lines: Convergence	582
9.3	Linear Wave Equations	587
9.3.1	Models for Vibrating Membrane	588
9.3.2	Wave Propagation	593
9.3.3	Method of Lines for Wave Propagation	598
9.3.4	Timestepping for Semi-Discrete Wave Equations	600
9.3.5	The Courant-Friedrichs-Levy (CFL) Condition	607
10	Convection-Diffusion Problems	616
10.1	Heat conduction in a fluid	617
10.1.1	Modeling Fluid Flow	617
10.1.2	Heat Convection and Diffusion	619
10.1.3	Incompressible Fluids	622
10.1.4	Time-Dependent (Transient) Heat Flow in a Fluid	625
10.2	Stationary Convection-Diffusion Problems: Numerical Treatment	627
10.2.1	Singular Perturbation	629
10.2.2	Upwinding	632
10.2.2.1	Upwind Quadrature	638
10.2.2.2	Streamline Diffusion	642

10.3 Discretization of Time-Dependent (Transient) Convection-Diffusion IBVPs	648
10.3.1 Convection-Diffusion IBVPs: Method of Lines	649
10.3.2 Transport Equation	651
10.3.3 Lagrangian Split-Step Method	654
10.3.3.1 Split-Step Timestepping, <i>cf.</i> Section 7.5	655
10.3.3.2 Particle Method for Pure Transport (Advection)	657
10.3.3.3 Particle Mesh Method (PMM)	659
10.3.4 Semi-Lagrangian Method	664
11 Numerical Methods for Conservation Laws	672
11.1 Conservation Laws: Examples	673
11.1.1 Linear Advection	673
11.1.2 Traffic Modeling [BD11]	678
11.1.2.1 Particle Model	678
11.1.2.2 Continuum Traffic Model	681
11.1.3 Inviscid Gas Flow	683
11.2 Scalar Conservation Laws in 1D	686
11.2.1 Integral and Differential Form	686
11.2.2 Characteristics	689
11.2.3 Weak Solutions	695
11.2.4 Jump Conditions	697
11.2.5 Riemann Problem	698
11.2.6 Entropy Condition	706
11.2.7 Properties of Entropy Solutions	709
11.3 Conservative Finite Volume (FV) Discretization	712
11.3.1 Prologue: Finite-Difference Method (FDM)	712
11.3.2 Spatially Semi-Discrete Conservation Form	717
11.3.3 Discrete Conservation Property	720
11.3.4 Numerical Flux Functions	723
11.3.4.1 Central Flux	723
11.3.4.2 Lax-Friedrichs/Rusanov Flux	727
11.3.4.3 Upwind Flux	730
11.3.4.4 Godunov Flux	735
11.3.5 Monotone Schemes	742
11.4 Timestepping for Finite-Volume Methods	747
11.4.1 Fully Discrete Evolutions	747
11.4.2 CFL-Condition	750
11.4.3 Linear Stability Analysis	755
11.4.4 Convergence of Fully Discrete FV Method	762
11.5 Higher-Order Conservative Finite-Volume Schemes	767
11.5.1 Piecewise Linear Reconstruction	768
11.5.2 Slope Limiting	776
11.5.3 MUSCL Scheme	781
11.6 Outlook: Systems of Conservation Laws	784
12 Finite Elements for the Stokes Equation	786
Index	787
Acronyms	799
Symbols	800
Examples	802
Codes	808

Chapter 0

Introduction

Contents

0.1	General Information	16
0.1.1	Goals	16
0.1.2	Teaching Model	16
0.1.3	Homework assignments	26
0.1.4	Information on Examinations	27
0.2	Coding Projects	28
0.2.1	Requests	28
0.3	Prerequisites for this Course	29
0.3.1	Tools from Linear Algebra	29
0.3.2	Tools from Real Analysis	35
0.3.3	Required Elementary Numerical Methods	44
0.4	Mathematical Modelling with Partial Differential Equations	44
0.4.1	PDEs: Basic Notions	44
0.4.2	Electromagnetics: Eddy Current Problem	45
0.4.3	Viscous Fluid Flow	46
0.4.4	Micromagnetics	46
0.4.5	Reaction-Diffusion: Phase Separation	47
0.4.6	Activator-Inhibitor Models: Fur Patterns	48
0.4.7	Quantitative Finance: Black-Scholes Equation	48
0.4.8	Quantum Mechanics: Electronic Schrödinger Equation	49
0.4.9	Rarefied Gas Dynamics: Boltzmann Equation	50
0.4.10	Wave Propagation: Helmholtz equation	51

0.1 General Information

0.1.1 Goals

The focus of this course is **neither**

on the rigorous **numerical analysis** of partial differential equations (as covered in 401-3651-00V: *Numerical methods for elliptic and parabolic partial differential equations*)

nor

on providing **mere recipes** for applying methods or even software packages.

Flavor of this course

This course combines *emphasis on algorithms* with their sound mathematical motivation and derivation, rigorous, but not too formal.

It aims to impart an “*intuitive understanding*” of numerical methods, their properties, potential, and limitations.

The main *skills* to be acquired in this course are the following.

- ◆ Ability to *implement* advanced numerical methods for the solution of partial differential equations in C++ efficiently (, based on numerical libraries, of course)
- ◆ Ability to *modify* and *adapt* numerical algorithms guided by awareness of their mathematical foundations
- ◆ Ability to *select* and *assess* numerical methods in light of the predictions of theory
- ◆ Ability to *identify features* of a PDE (= partial differential equation) based model that are relevant for the selection and performance of a numerical algorithm
- ◆ Ability to *understand research publications* on theoretical and practical aspects of numerical methods for partial differential equations.

0.1.2 Teaching Model

This course will depart from the usual academic teaching arrangement centering around classes taught by a lecturer addressing an audience in a lecture hall.

A **flipped-classroom** course

This course will follow the **flipped-classroom** paradigm:

Learning by **self-study** guided by

instruction videos
tablet notes

lecture notes

interactive
Q&A sessions

homework
tutorial classes

All the course material will be published online through the course Moodle Page. All notes jotted down by the lecturer during the creation of videos or during the Q&A sessions will be made available as PDF.

0.1.2.1 Course Videos

In the flipped-classroom teaching model regular lectures will partly be replaced with pre-recorded videos. These videos are not commercial-grade clips, but resemble video recordings from a standard classroom setting; they convey the development of the material on a tablet accompanied by the lecturer’s voice.



Course Videos

The videos will be published through the course Moodle page and you can play them right in Moodle using the embedded video player.

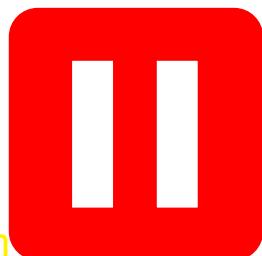
Every video comes with a PDF containing the **tablet notes** taken during the creation of the video. However, the PDF may have been *corrected, updated, or supplemented* later.

The course videos in MP4-format can also be accessed through a password-protected **Polybox folder** and then played locally, using software like **VLC**.



Video tutorials for some of the homework problems are available. However, they may be outdated and they will not be updated in the future, because according to poll data from 2019 only a small fraction of the students watched them. Instead effort will be put into creating comprehensive master solutions.

§0.1.2.2 (“Pause” and “fast forward”) Videos have *two big advantages*:



You can stop a video at any time, whenever

- you need more time to think,
- you want to look up related information,
- you want to work for yourself.

Make use of this possibility!

Fig. 2

The video portal also allows you to play the videos at *1.5× speed*. This can be useful, if the current topic is very clear to you. You can also *skip* entire *parts* using the scroll bar. The same functionality (fast playing and skipping) is offered by most video players, for instance the **VLC media player**.

§0.1.2.3 (Review questions) Most lecture units (corresponding to a video) are accompanied with a list of review questions. You should try to answer them off the top of your head *without consulting any written material* shortly after you have finished studying the unit .

If you are utterly clueless about how to approach a review question, you probably need to resume studying some of the unit’s topics.

§0.1.2.4 (List of available tutorial videos) This is the list of available video tutorials as of January 13, 2023:



Necessary corrections and updates of the lecture document will sometimes lead to changes in the numbering of paragraphs and formulas, which, of course, cannot be applied to the recorded videos.

However, these changes will be taken into account into the tablet notes supplied for every video.

1. Video tutorial for Section 1.2.1: Elastic Membranes: (44 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.2.1.30
2. Video tutorial for Section 1.2.2: Electrostatic Fields: (13 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.2.2.17
3. Video tutorial for Section 1.2.3: Quadratic Minimization Problems: (48 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.2.3.50

4.  Video tutorial for Section 1.3: Sobolev Spaces: (47 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.3.4.29
5.  Video tutorial for Section 1.4: Linear Variational Problems: (47 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.4.3.8
6.  Video tutorial for Section 1.5: Equilibrium Models: Boundary Value Problems: (50 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.5.3.19
7.  Video tutorial for Section 1.6: Diffusion Models: Stationary Heat Conduction: (12 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.6.0.10
8.  Video tutorial for Section 1.7: Boundary Conditions: (16 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.7.0.12
9.  Video tutorial for Section 1.8: Second-Order Elliptic Variational Problems: (32 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.8.0.32
10.  Video tutorial for Section 1.9: Essential and Natural Boundary Conditions: (36 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 1.9.0.13
11.  Video tutorial for Section 2.2: Principles of Galerkin Discretization: (47 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.2.3.10
12.  Video tutorial for Section 2.3: Case Study: Linear FEM for Two-Point Boundary Value Problems: (49 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.3.3.17
13.  Video tutorial for Section 2.4: Case Study: Triangular Linear FEM in Two Dimensions (I): (53 minutes) [Download link](#), [tablet notes](#)
14.  Video tutorial for Section 2.4: Case Study: Triangular Linear FEM in Two Dimensions (II): (61 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.4.6.13
15.  Video tutorial for Section 2.5: Building Blocks of General Finite Element Methods: (39 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.5.3.9

16.  Video tutorial for Section 2.6: Lagrangian Finite Element Methods: (43 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.6.2.13
17.  Video tutorial for Section 2.7.2: Mesh Information and Mesh Data Structures: (39 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 10.1.1.8 and Quizz 2.7.2.24
18.  Video tutorial for Section 2.7.4: Assembly Algorithms: (46 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.7.4.47
19.  Video tutorial for Section 2.7.5: Local Computations: (48 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.7.5.43
20.  Video tutorial for Section 2.7.6: Treatment of Essential Boundary Conditions: (38 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.7.6.21
21.  Video tutorial for Section 2.8: Parametric Finite Element Methods (I): (45 minutes) [Download link](#), [tablet notes](#)
22.  Video tutorial for Section 2.8: Parametric Finite Element Methods (II): (56 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 2.8.4.8
23.  Video tutorial for Section 3.1: Abstract Galerkin Error Estimates: (38 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.3.5.29
24.  Video tutorial for Section 3.2: Empirical (Asymptotic) Convergence of FEM: (53 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.2.3.14
25.  Video tutorial for Section 3.3: A Priori (Asymptotic) Finite Element Error Estimates (I): (21 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.3.5.29
26.  Video tutorial for Section 3.3: A Priori (Asymptotic) Finite Element Error Estimates (II): (35 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.3.5.29
27.  Video tutorial for Section 3.3: A Priori (Asymptotic) Finite Element Error Estimates (III): (35 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.3.5.29

-
- 28.  Video tutorial for Section 3.4: Elliptic Regularity Theory: (21 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.4.0.12
 - 29.  Video tutorial for Section 3.5: Variational Crimes: (30 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.5.2.2
 - 30.  Video tutorial for Section 3.6.1: Linear Output Functionals: (28 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.6.1.13
 - 31.  Video tutorial for Section 3.6.2: Case Study: Computation of Boundary Fluxes with FEM: (18 minutes) [Download link](#), [tablet notes](#)
➤ Quizz 3.6.2.13
 - 32.  Video tutorial for Section 3.6.3: Lagrangian FEM: L^2 -Estimates: (21 minutes) [Download link](#), [tablet notes](#)
 - 33.  Video tutorial for Section 3.7: Discrete Maximum Principle: (39 minutes) [Download link](#), [tablet notes](#)
 - 34.  Video tutorial for Section 3.8: Validation and Debugging of Finite Element Codes: (26 minutes) [Download link](#), [tablet notes](#)
 - 35.  Video tutorial for Section 6.1: Initial-Value Problems (IVPs) for Ordinary Differential Equations (ODEs): (63 minutes) [Download link](#), [tablet notes](#)
 - 36.  Video tutorial for Section 6.2: Introduction: Polygonal Approximation Methods: (32 minutes) [Download link](#), [tablet notes](#)
 - 37.  Video tutorial for Section 6.3: General Single-Step Methods: (31 minutes) [Download link](#), [tablet notes](#)
 - 38.  Video tutorial for Section 6.3.2: (Asymptotic) Convergence of Single-Step Methods: (39 minutes) [Download link](#), [tablet notes](#)
 - 39.  Video tutorial for Section 6.4: Explicit Runge-Kutta Single-Step Methods (RKSSMs): (46 minutes) [Download link](#), [tablet notes](#)
 - 40.  Video tutorial for Section 6.5: Adaptive Stepsize Control: (56 minutes) [Download link](#), [tablet notes](#)
 - 41.  Video tutorial for Section 7.1: Model Problem Analysis: (68 minutes) [Download link](#), [tablet notes](#)
 - 42.  Video tutorial for Section 7.2: Stiff Initial-Value Problems: (39 minutes) [Download link](#), [tablet notes](#)

-
- 43.  Video tutorial for Section 7.3: Implicit Runge-Kutta Single-Step Methods: (78 minutes) [Download link](#), [tablet notes](#)
 - 44.  Video tutorial for Section 7.4: Semi-Implicit Runge-Kutta Methods: (23 minutes) [Download link](#), [tablet notes](#)
 - 45.  Video tutorial for Section 7.5: Splitting Methods: (34 minutes) [Download link](#), [tablet notes](#)
 - 46.  Video tutorial for Section 9.2.1: Heat Equation: (18 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.2.1.14
 - 47.  Video tutorial for Section 9.2.2: Heat Equation: Spatial Variational Formulation: (15 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.2.2.10
 - 48.  Video tutorial for Section 9.2.3: Stability of Parabolic Evolution Problems: (20 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.2.3.13
 - 49.  Video tutorial for Section 9.2.4: Spatial Semi-Discretization: Method of Lines: (13 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.2.4.7
 - 50.  Video tutorial for Section 9.2.5: Ordinary Differential Equations: (20 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.2.5.15
 - 51.  Video tutorial for Section 9.2.6: Single-Step Methods for Numerical Integration: (48 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.2.6.31
 - 52.  Video tutorial for Section 9.2.7: Timestepping for Method-of-Lines ODE: (57 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.2.7.52
 - 53.  Video tutorial for Section 9.2.8: Fully Discrete Method of Lines: Convergence: (35 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.2.8.14
 - 54.  Video tutorial for Section 9.3.1: Models for Vibrating Membranes: (24 minutes) [Download link](#), [tablet notes](#)
 - Quizz 9.3.1.20
 - 55.  Video tutorial for Section 9.3.2: Wave Propagation: (33 minutes) [Download link](#), [tablet notes](#)

> Quizz 9.3.2.17

56.  Video tutorial for Section 9.3.3: Method of Lines for Wave Propagation: (13 minutes) [Download link](#), [tablet notes](#)
- > Quizz 9.3.3.9
57.  Video tutorial for Section 9.3.4: Timestepping for Semi-Discrete Wave Equations: (43 minutes) [Download link](#), [tablet notes](#)
- > Quizz 9.3.4.21
58.  Video tutorial for Section 9.3.5: The Courant-Friedrichs-Levy (CFL) Condition: (46 minutes) [Download link](#), [tablet notes](#)
- > Quizz 9.3.5.15
59.  Video tutorial for 10.1.1: Modeling Fluid Flow: (7 minutes) [Download link](#), [tablet notes](#)
60.  Video tutorial for Section 10.1.2: Heat Convection and Diffusion: (6 minutes) [Download link](#), [tablet notes](#)
61.  Video tutorial for Section 10.1.3: Incompressible Fluids: (10 minutes) [Download link](#), [tablet notes](#)
62.  Video tutorial for Section 10.1.4: Time-Dependent (Transient) Heat Flow in a Fluid: (5 minutes) [Download link](#), [tablet notes](#)
63.  Video tutorial for Section 10.2.1: Singular Perturbation: (15 minutes) [Download link](#), [tablet notes](#)
64.  Video tutorial for Section 10.2.2: Upwinding: (14 minutes) [Download link](#), [tablet notes](#)
65.  Video tutorial for Section 10.2.2.1: Upwind Quadrature: (17 minutes) [Download link](#), [tablet notes](#)
66.  Video tutorial for Section 10.2.2.2: Streamline Diffusion: (19 minutes) [Download link](#), [tablet notes](#)
67.  Video tutorial for Section 10.3.1: Convection-Diffusion IBVPs: Method of Lines : (13 minutes) [Download link](#), [tablet notes](#)
68.  Video tutorial for Section 10.3.2: Transport Equation: (8 minutes) [Download link](#), [tablet notes](#)
69.  Video tutorial for Section 10.3.3: Lagrangian Split-Step Method: (23 minutes) [Download link](#), [tablet notes](#)
70.  Video tutorial for Section 10.3.4: Semi-Lagrangian Method: (18 minutes) [Download link](#), [tablet notes](#)

71.  Video tutorial for Section 11.1: Conservation Laws: Examples: (50 minutes) [Download link](#), [tablet notes](#)
72.  Video tutorial for Section 11.2.1: Scalar Conservation Laws in 1D: Integral and Differential Form: (18 minutes) [Download link](#), [tablet notes](#)
73.  Video tutorial for Section 11.2.2: Scalar Conservation Laws in 1D: Characteristics: (33 minutes) [Download link](#), [tablet notes](#)
74.  Video tutorial for Section 11.2.3, Section 11.2.5: Scalar Conservation Laws in 1D: Weak Solutions, Jump Conditions, and the Riemann Problem: (54 minutes) [Download link](#), [tablet notes](#)
75.  Video tutorial for Section 11.2.6, Section 11.2.7: Scalar Conservation Laws in 1D: Entropy Conditions and Properties of Solutions: (26 minutes) [Download link](#), [tablet notes](#)
76.  Video tutorial for Section 11.3.1, Section 11.3.3: FV: Spatially Semi-Discrete Conservation Form: (58 minutes) [Download link](#), [tablet notes](#)
77.  Video tutorial for Section 11.3.4: Numerical Flux Functions: (52 minutes) [Download link](#), [tablet notes](#)
78.  Video tutorial for Section 11.3.5: Monotone Schemes: (28 minutes) [Download link](#), [tablet notes](#)
79.  Video tutorial for Section 11.4.1, Section 11.4.2: Timestepping: Fully Discrete Evolutions and CFL-Condition: (49 minutes) [Download link](#), [tablet notes](#)
80.  Video tutorial for Section 11.4.3: Timestepping: Linear Stability Analysis: (36 minutes) [Download link](#), [tablet notes](#)
81.  Video tutorial for Section 11.4.4: Convergence of Fully Discrete FV Method: (29 minutes) [Download link](#), [tablet notes](#)
82.  Video tutorial for Section 11.5: Higher-Order Conservative Finite-Volume Schemes: (61 minutes) [Download link](#), [tablet notes](#)

↓

§0.1.2.5 (Reporting errors) As the PDF documents and tablet notes for this course will always be in a state of flux, they will inevitably and invariably contain quite a few small errors, mainly typos and omissions.

Please report errors in the lecture material through the dedicated forum on the **Course Moodle Page!**

When reporting an error, *please specify the section and the number of the paragraph, remark, equation, etc. where it hides.* You need not give a page number.

↓

0.1.2.2 Following the Course

Weekly study assignments

- For every week there is a list of course units and associated videos published on the course Moodle Page.
- The corresponding contents **must** be studied in that same week.

§0.1.2.7 (How to organize your learning)

- ☛ **Develop a routine:** Plan *fixed slots*, with a total duration of *four hours*, for studying for the course material in your weekly calendar. This does not include homework.
- ☛ **Choose a stable setting,** where you can really concentrate (quiet area, headphones, coffee, etc.)
- ☛ **Take breaks,** when concentration is declining, usually after 20 to 45 minutes, but *avoid online distractions* during breaks.

You must not procrastinate!



Do not put off studying for this course. Dependencies between the topics will make it very hard to catch up.

§0.1.2.8 (“Personalized learning”) The flipped classroom model allows students to pursue their preferred ways of studying. The following approaches can be tried.

- **Traditional:** You watch the assigned videos similar to attending a conventional classroom lecture. Afterwards digest the material based on the tablet notes and/or the lecture document. Finally, answer the review questions and look up more information in the lecture document.
- **Reading-centered:** You work through the unit reading the tablet notes, and, sometimes, related sections of the lecture document. You occasionally watch parts of the videos, in case some considerations and arguments have not become clear to you already.



Collaborative studying is encouraged:

- You may watch course videos together with classmates.
- You may meet to discuss course units.
- You may solve homework problems in a group assigning different parts to different members.
- ☛ Explaining to others is a great way to deepen understanding.
- ☛ It is easy to sustain motivation in a peer study group.

Fig. 3

0.1.2.3 Learning Model

This course will take

hard work — perseverance — patience

For most students understanding the course contents will advance in three stages:

- ① You understand the **first 30%** when watching videos and studying the course material
- ② You understand the **next 30%** when trying to solve the homework problems
- ③ You understand the **further 30%** when preparing for the main exam
- ④ You might understand the **missing 10%** when applying the methods in projects, research, or on the job.

§0.1.2.9 (Expected workload) For this course you can earn **10 ECTS credits**. Though a very loose relationship, this roughly indicates a total workload of 300 hours:

$$300 \text{ hours} = \underbrace{200 \text{ hours}}_{\text{during term}} + \underbrace{100 \text{ hours}}_{\text{exam preparation}} .$$

This indicates that you should brace for an

$$\text{average workload} \approx 12 - 15 \text{ hours per week.}$$

I recommend a rather even split between

- watching videos and/or studying the course material: \approx 4-5 hours/week,
- and solving homework problems: \approx 5-7 hours/week.
- attending Q&A sessions and tutorials \approx 3 hours/week.

All these are averages and the workload may vary between different weeks. □

0.1.3 Homework assignments

Every week a number of homework problems will be selected from the [NumPDE Homework Problem Collection](#). The selection will be published on the course Moodle page and the problems should be worked right after they have been submitted. Most of the homework problems will involve both implementation and theoretical parts. Please note that the problem collection is being extended throughout the semester. Thus, make sure that you [obtain the most current version](#) every week.

We expect the average student to take 5-8 hours to solve the homework problems, not taking into account time spent on debugging codes for programming assignments.

The problems come with plenty of hints. A master solution will also be made available, but it is foolish to read the master solution parallel to working on a problem sheet, because *trying to find the solution on one's own is essential for developing problem solving skills*, though it may occasionally be frustrating.

Some or all of the problems of an assignment sheet will be discussed in the tutorial classes at least one week after the problems have been assigned.

Your tutors are happy to examine your solutions and give you feedback : You may either hand them your solution papers during the tutorial session (put your name on every sheet and clearly mark the problems you want to be inspected) or [upload](#) a scan/photo through the Moodle upload interface for the course, see the course Moodle page for details. You are encouraged to hand in incomplete and wrong solutions, so that you can receive valuable feedback even on incomplete or failed attempts.

§0.1.3.1 (Reporting errors in homework problems) Similarly to the lecture document, also the homework problems will occasionally be affected by small errors. In addition, we have to admit that some solutions are incomplete, too short, confusing, or even wrong. In this case the participants of the course are requested to

Please report flaws and shortcomings in homework problems through *dedicated fora* on the course Moodle page.

Please specify the exact number of the affected sub-problem and provide a short description.

0.1.4 Information on Examinations

§0.1.4.1 (Examinations during the teaching period) From the ETH course directory for 401-0674-00L (Numerical Methods for Partial Differential Equations)

A 30-minute mid-term exam and a 30-minute end term exam (*non-mandatory*) will be held during the teaching period on dates specified in the beginning of the semester. The grades of these interim examinations will be taken into account through a *bonus* of up to 30% for the final grade. The dates for the term exams will be communicated in the beginning of the course.

That final grade is computed according to the formula

$$G := 0.25 \cdot [4 \cdot \max\{G_s, 0.85G_s + 0.15g_m, 0.85G_s + 0.15g_e, 0.7G_s + 0.15g_m + 0.15g_e\}] , \quad (0.1.4.2)$$

$G_s \hat{=} \text{grade in main exam}$, $g_m \hat{=} \text{mid-term grade}$, $g_e \hat{=} \text{end-term grade}$,

where $\lceil x \rceil$ designates the smallest integer $\geq x$. Both will be closed book examinations on paper. The dates of the exams will be communicated in the beginning of the term and published on the course webpage.

§0.1.4.3 (Main examination during exam session)

- ◆ Three-hour written examination involving coding problems to be done at the computer. The date of the exam will be set and communicated by the ETH exam office, and will also be published on the course webpage.
- ◆ The coding part of the exam has to be done in Linux environment.
- ◆ Subjects of examination:
 - All topics that have been addressed in a video listed on the course Moodle page or in any assigned homework problem

The lecture document contains much more material than covered in class. All these extra topics are not relevant for the exam.

- ◆ The lecture document (as PDF), the LEHRFEM++ documentation (HTML), the documentation of EIGEN (HTML), and the online C++ REFERENCE PAGES (HTML) will be available during the examination. The corresponding final version of the lecture document will be made available at least two weeks before the exam.
- ◆ No other materials may be used during the exam.
- ◆ The homework problem collection cannot be accessed during the exam.
- ◆ The exam questions will be asked in English.
- ◆ In case you come to the conclusion that you have too little time to prepare for the main exam a few weeks before the exam, contemplate withdrawing in order not to squander an attempt.

§0.1.4.4 (Repeating the main exam)

- Bonus points earned in term exams in last year's course can be taken into account for this course's main exam.
- If you want to take this option, please [declare this intention](#) by email to the course organizers before the mid-term exam. Otherwise, your bonus will be based on the results of this year's term exams.

0.2 Coding Projects

Many homework assignment will involve a substantial implementation component:

- The programming language used in this course is **C++** in its modern version, currently C++17. C++ programming skills will be taken for granted. For comprehensive information on all aspects of C++ please refer to the [C++ reference](#) pages.



- Most codes for this course rely on the **EIGEN** C++ linear algebra template library, which comes copious documentation. This library is also used in the course Numerical Methods for CSE and [[Hip19](#), ??] gives a brief introduction.
- A special C++ finite element library has been developed for this course: **LEHRFEM++**, which has also been endowed with a detailed [documentation](#).
- The coding projects accompanying the course will be done in the framework of a **cmake**-based build system.

You are supposed to develop and run codes on your own computer or a machine in an ETH computer lab. Development and testing of homework codes is done under the **Linux** operating system using the **GNU** or **LLVM/CLANG** C++ compilers. Thus, it is advisable to work on a Linux (virtual) machine in order to ensure smooth compilation. However, with suitable adjustments the codes can also be compiled and run on Mac OS X and Microsoft Windows.

0.2.1 Requests

The lecturers very much welcome and, putting it even more strongly, rather depend on feedback and suggestions of the students taking the course for continuous improvement of the course contents and presentation. Therefore all participants are strongly encouraged to get involved actively and contribute in the following ways:

§0.2.1.1 (Reporting errors) As the documents for this course will always be in a state of flux, they will inevitably and invariably be afflicted with small errors, mainly typos and omissions.



For error reporting we use the [Discuna](#) online collaboration platform that runs in the browser.

Discuna allows to attach various types of annotations to shared PDF documents, see [instruction video](#).

Please report errors in the lecture material through the [DISCUNA NumPDE Community](#) for the current semester to which various course-related documents have already been uploaded.

In the beginning of the teaching period you receive a [join link](#) of the form <https://app.discuna.com/<JOIN CODE>>. Open the link in a web browser and it will take you to the [DISCUNA](#) community page.

To report an error,

1. select the corresponding PDF document (chapter of the lecture document or homework problem) in the left sidebar,
2. press the prominent white-on-blue +-button in the right sidebar,
3. click on the displayed PDF where the error is located,
4. then in the pop-up window choose the “Error” category,
5. and add a title and,
6. if the title does not tell everything, a short description.

In case you cannot or do not want to link an error to a particular point in the PDF, you may just click on the title page of the respective chapter. Then, please precisely specify the concerned section and the number of the paragraph, remark, equation etc. *Do not give page numbers* as they may change with updates to the documents.

Note that chapter PDFs and homework problem files will gradually be added to the [DISCUNA](#) NumPDE community. Hence, the final chapters will not be accessible in the beginning of the course. □

§0.2.1.2 (Pointing out technical problems) The [DISCUNA NumPDE Community](#) is equipped with a [chat channel “Technical Problems”](#). In case you encounter a problem affecting the videos, the course web pages, or the PDF documents supplied online, say, severely distorted or missing audio tracks or a faulty link, instantly post a comment to this channel with a short description of the problem. You can do this after clicking on the channel name in the left sidebar in the community □

§0.2.1.3 (Providing comments and suggestions) The [chat channel “General Comments”](#) of the [DISCUNA NumPDE Community](#) is meant for letting the lecturer know about weaknesses of the contents, structure, and presentation of the course and how they can be remedied. Your statements should be constructive and address specific parts or aspects of the course.

Regularly, students attending the course remark that they have found online resources like instruction videos that they think present some of the course material in a much clearer and better structured way. It is important that you tell the lecturer about those online resources so that he can include pointers to them and get inspiration. Use the “General Comments” channel also for this purpose. State clearly, which part of the course you are referring to, and briefly explain why the online resource is superior or a valuable supplement. □

0.3 Prerequisites for this Course

0.3.1 Tools from Linear Algebra

Mastery of basic concepts of linear algebra as they are taught in a first-semester introductory course is assumed.



Supplementary literature. All linear algebra techniques needed for this course are covered in

the textbook [K. NIPP AND D. STOFFER, *Lineare Algebra*, vdf Hochschulverlag, Zürich, 5 ed., 2002. (German)]

These basic concepts include vector spaces, in particular \mathbb{R}^n and \mathbb{C}^n , $n \in \mathbb{N}$, subspaces, affine spaces, linear independence, bases, matrices and vectors, and linear systems of equations. Since “affine spaces” are not in the limelight in linear algebra, let us recall their definition:

Definition 0.3.1.1. Affine (sub)space

Given a vector space V , a subspace $V_0 \subset V$, and some $\hat{v} \in V$, we call the set

$$\hat{V} := \hat{v} + V_0 := \{v = \hat{v} + v_0 : v_0 \in V_0\} \subset V$$

an **affine (sub)space** (of V).

Let us also recall, what you should know about bases:

Definition 0.3.1.2. Basis of a finite dimensional vector space

Let V be a real vector space. A finite subset $\{b^1, \dots, b^N\} \subset V$, $N \in \mathbb{N}$, is a **basis** of V , if for **every** $v \in V$ there are **unique** coefficients $\mu_\ell \in \mathbb{R}$, $\ell \in \{1, \dots, N\}$, such that $v = \sum_{\ell=1}^N \mu_\ell b^\ell$. Then N agrees with the **dimension** of V .

0.3.1.1 Basic Notations

We employ the notations for matrices and vectors from [Hip19, ??]. By default, all vectors in \mathbb{R}^n or \mathbb{C}^n are regarded as column vectors. We write a, b, \dots for “small vectors” $\in \mathbb{R}^n$, which often contain the coordinates of points. “Large” vectors of basis expansion coefficients $\in \mathbb{R}^N$, $N \in \mathbb{N}$, are designated by $\vec{\mu}, \vec{\eta}, \dots$. Vector components are usually singled out as follows, with indices invariably starting from 1:

$$\vec{\mu} = [\mu_1 \ \mu_2 \ \mu_3 \ \cdots \ \mu_N]^\top \in \mathbb{R}^N \quad \blacktriangleright \quad (\vec{\mu})_j := \mu_j, \quad j = 1, \dots, N.$$

Special vectors are the zero vector $\mathbf{0} \in \mathbb{R}^n$, the “large” unit vectors $\vec{e}^k \in \mathbb{R}^n$, $k \in \{1, \dots, n\}$, defined by

$$(\vec{e}^k)_j = \delta_{jk} := \begin{cases} 1 & , \text{if } j = k, \\ 0 & , \text{if } j \neq k \end{cases} \quad (\text{Kronecker symbol}),$$

and the column vector $\mathbf{1}$ with entries all $= 1$.

Bold capital letters like $\mathbf{A}, \mathbf{B}, \dots$ are reserved for matrices $\in \mathbb{R}^{n,m}$ or $\in \mathbb{C}^{n,m}$. We use \mathbf{A}^\top to denote the transposed matrix, and \mathbf{A}^{-1} for the inverse of a regular square matrix. For a matrix $\mathbf{A} \in \mathbb{R}^{n,m}$ we write

- $(\mathbf{A})_{i,j}$ for the entry in row i and column j , $1 \leq i \leq n$, $1 \leq j \leq m$

$$\mathbf{A} = \left[(\mathbf{A})_{i,j} \right]_{\substack{i=1,\dots,n \\ j=1,\dots,m}} \quad (i \doteq \text{row index}, j \doteq \text{row index}),$$

- $(\mathbf{A})_{:,j} \in \mathbb{R}^m$ for its j -th column (a column vector, of course), $1 \leq j \leq m$,
- $(\mathbf{A})_{i,:}$ for its i -th row (to be read as row vector, of course), $1 \leq i \leq n$.

Note that vectors of any kind are simply matrices with a single column or row. Sporadically, we also use the “dot product notation”:

$$\vec{\mu} \cdot \vec{\eta} := \vec{\mu}^\top \vec{\eta} \in \mathbb{R}, \quad \vec{\mu}, \vec{\eta} \in \mathbb{R}^N.$$

The field \mathbb{R} may be replaced with \mathbb{C} , but then complex conjugation has to be inserted in the right places.

0.3.1.2 Linear and Bilinear Forms

In linear algebra we learned about linear mappings between vector spaces. A special case is that of the image space being \mathbb{R} .

Definition 0.3.1.3. Linear forms

Given a vector space V over \mathbb{R} , a **linear form/linear functional** (LF) ℓ is a mapping $\ell : V \mapsto \mathbb{R}$ that satisfies

$$\ell(\alpha u + \beta v) = \alpha \ell(u) + \beta \ell(v) \quad \forall u, v \in V, \forall \alpha, \beta \in \mathbb{R}.$$

Linear algebra also examines multi-linear forms, mappings from $V \times V \times \cdots \times V \mapsto \mathbb{R}$, which are linear in each argument. We will not need this notion in full generality, but such mappings taking two arguments will play a central role.

Definition 0.3.1.4. (Bi-)linear forms

Given an \mathbb{R} -vector space V , a **bilinear form** (BLF) a on V is a mapping $a : V \times V \mapsto \mathbb{R}$, for which

$$a(\alpha_1 v_1 + \beta_1 u_1, \alpha_2 v_2 + \beta_2 u_2) = \\ \alpha_1 \alpha_2 a(v_1, v_2) + \alpha_1 \beta_2 a(v_1, u_2) + \beta_1 \alpha_2 a(u_1, v_2) + \beta_1 \beta_2 a(u_1, u_2)$$

for all $u_i, v_i \in V, \alpha_i, \beta_i \in \mathbb{R}, i = 1, 2$.

☞ notation: For bilinear forms we write $a(\cdot, \cdot)$, $b(\cdot, \cdot)$, etc; note the special font.

EXAMPLE 0.3.1.5 (Linear forms on \mathbb{R}^n) For a fixed column vector $\vec{\alpha} \in \mathbb{R}^n, n \in \mathbb{N}$, the mapping

$$\mathbb{R}^n \rightarrow \mathbb{R}, \quad \vec{\xi} \mapsto \vec{\alpha}^\top \vec{\xi}, \tag{0.3.1.6}$$

is a linear form on \mathbb{R}^n . The converse is also true: every linear form ℓ on \mathbb{R}^n can be written in the form (0.3.1.6) with some vector $\vec{\alpha} \in \mathbb{R}^n$. Its components are given by letting ℓ act on the unit vectors: $(\vec{\alpha})_j = \ell(\vec{e}^j), j = 1, \dots, n$

EXAMPLE 0.3.1.7 (Bilinear forms on \mathbb{R}^n) For any square matrix $\mathbf{A} \in \mathbb{R}^{n,n}, n \in \mathbb{N}$, the mapping

$$\mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}, \quad (\vec{\xi}, \vec{\eta}) \mapsto \vec{\xi}^\top \mathbf{A} \vec{\eta}, \tag{0.3.1.8}$$

represents a bilinear form on \mathbb{R}^n . In fact, for every bilinear form a on \mathbb{R}^n we can find a square matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ such that $a(\vec{\xi}, \vec{\eta}) = \vec{\xi}^\top \mathbf{A} \vec{\eta}$. The entries of \mathbf{A} can be computed by plugging unit vectors into a

$$a(\vec{\xi}, \vec{\eta}) = \vec{\xi}^\top \mathbf{A} \vec{\eta} \Leftrightarrow (\mathbf{A})_{i,j} = a(\vec{e}^i, \vec{e}^j), \quad i, j \in \{1, \dots, n\}.$$

Every n -dimensional real vector space V can be identified with \mathbb{R}^n through considering a particular basis. Thus, from the previous two examples we learn that, with respect to a chosen basis, every linear form on V can be represented by a column vector and every bilinear form on V allows the description by a square matrix.

Remark 0.3.1.9 (Linear forms from bilinear forms) Let a be a bilinear form on \mathbb{R}^n . Let us fix a vector $\vec{\xi} \in \mathbb{R}^n$. Then the mapping

$$\mathbb{R}^n \rightarrow \mathbb{R}, \quad \vec{\eta} \mapsto a(\vec{\xi}, \vec{\eta}),$$

is a linear form on \mathbb{R}^n . This is immediate from Def. 0.3.1.3 and Def. 0.3.1.4. \square

0.3.1.3 Norms and Inner Products

You should know the following concept generalizing the length of a vector, see also [Hip19, ??].

Definition 0.3.1.10. Norm (on a vector space) → [Hip19, ??]

A **norm** $\|\cdot\|_V$ on an \mathbb{R} -vector space V is a mapping $\|\cdot\|_V : V \mapsto \mathbb{R}_0^+$, such that

$$(\text{definiteness}) \quad \|v\|_V = 0 \iff v = 0 \quad \forall v \in V \quad (\text{N1})$$

$$(\text{homogeneity}) \quad \|\lambda v\|_V = |\lambda| \|v\|_V \quad \forall \lambda \in \mathbb{R}, \forall v \in V, \quad (\text{N2})$$

$$(\text{triangle inequality}) \quad \|w + v\|_V \leq \|w\|_V + \|v\|_V \quad \forall w, v \in V. \quad (\text{N3})$$

If the mapping $V \rightarrow \mathbb{R}_0^+$ still satisfies (N2) and (N3), but fails on (N1), then it is called a **semi-norm**.

§0.3.1.11 (Norms on \mathbb{R}^n) The most important norms on the vector space \mathbb{R}^n are the

$$\text{maximum norm:} \quad \|\vec{\xi}\|_\infty := \max \left\{ \left| \left(\vec{\xi} \right)_j \right|, j = 1, \dots, n \right\}, \quad (0.3.1.12)$$

$$\text{Euclidean norm:} \quad \|\vec{\xi}\|_2 := \left(\sum_{j=1}^n \left| \left(\vec{\xi} \right)_j \right|^2 \right)^{\frac{1}{2}}, \quad (0.3.1.13)$$

$$\text{1-norm:} \quad \|\vec{\xi}\|_1 := \sum_{j=1}^n \left| \left(\vec{\xi} \right)_j \right|. \quad (0.3.1.14)$$

The Euclidean norm is sometimes called **2-norm**. Frequently, for small vectors, we drop the subscript 2 from the notation for the Euclidean norm. \square

We will mainly be concerned with norms spawned by inner products, which are defined next.

Definition 0.3.1.15. Symmetric bilinear form

A bilinear form (→ Def. 0.3.1.4) a on a vector space V is **symmetric**, if and only if

$$a(u, v) = a(v, u) \quad \forall u, v \in V.$$

Important classes of bilinear forms “have signs”:

Definition 0.3.1.16. Positive (semi-)definite bilinear form

A bilinear form $a : V \times V \rightarrow \mathbb{R}$ on a real vector space V is called

$$\begin{aligned} \text{positive semi-definite} &: \Leftrightarrow a(v, v) \geq 0 \quad \forall v \in V, \\ \text{positive definite} &: \Leftrightarrow a(v, v) > 0 \quad \forall v \in V \setminus \{\mathbf{0}\}. \end{aligned}$$

The one-to-one relationship of square matrices and bilinear forms on \mathbb{R}^n provides the notion of being positive definite for matrices.

Definition 0.3.1.17. Positive definite matrix

A square matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ is **positive definite**, if

$$\vec{\eta}^\top \mathbf{A} \vec{\eta} > 0 \quad \text{for all } \vec{\eta} \in \mathbb{R}^n \setminus \{\mathbf{0}\}.$$

We abbreviate the property of a bilinear form or square matrix to be symmetric and positive definite at the same time as **s.p.d.**.

Definition 0.3.1.18. Inner product

A symmetric positive definite (s.p.d.) bilinear form on a real vector space V is also called an **inner product** or **scalar product** on V .

Inner products allow the following famous elementary estimate.

Theorem 0.3.1.19. Cauchy-Schwarz inequality

If a is a **symmetric positive semi-definite** bilinear form on the real vector space V , then

$$|a(u, v)| \leq a(u, u)^{\frac{1}{2}} a(v, v)^{\frac{1}{2}}. \quad (0.3.1.20)$$

Proof. Since a is symmetric and positive semi-definite,

$$0 \leq a(u + \tau v, u + \tau v) = a(u, u) + 2\tau a(u, v) + \tau^2 a(v, v) \quad \forall u, v \in V, \forall \tau \in \mathbb{R}.$$

① If $a(v, v) = 0$, the above inequality can only hold for all τ , if $a(u, v) = 0$ as well.

② If $a(v, v) \neq 0$, then set $\tau = -\frac{a(u, v)}{a(v, v)}$, which implies

$$0 \leq a(u, u) - \frac{a(u, v)^2}{a(v, v)} \Rightarrow a(u, v)^2 \leq a(u, u)a(v, v),$$

and finished the proof. □

Theorem 0.3.1.21. Norms from inner products

If a is an inner product (= symmetric positive definite bilinear form, Def. 0.3.1.18) on the real vector space V , then

$$\|\cdot\|_a : V \rightarrow \mathbb{R}, \quad \|v\| := a(v, v)^{\frac{1}{2}}, \quad (0.3.1.22)$$

defines a **norm** (\rightarrow Def. 0.3.1.10) on V .

Note that using $\|\cdot\|_a$ the Cauchy-Schwarz inequality can be rewritten as:

$$|a(u, v)| \leq \|u\|_a \|v\|_a \quad \forall u, v \in V. \quad (0.3.1.23)$$

Proof. (of Thm. 0.3.1.21)

- ① The property (N1) is implied by the positive definite property of a .
- ② Homogeneity (N2) follows from the linearity of a in both arguments.
- ③ The triangle inequality is a consequence of linearity, symmetry, and the Cauchy-Schwarz inequality (0.3.1.23):

$$\|u + v\|_a^2 = a(u + v, u + v) \stackrel{\text{symmetry}}{=} a(u, u) + 2a(u, v) + a(v, v) \stackrel{(0.3.1.23)}{\leq} \|u\|_a^2 + 2\|u\|_a\|v\|_a + \|v\|_a^2.$$

Then apply the binomial formula on the right. \square

As we have seen in Ex. 0.3.1.7, on \mathbb{R}^n square matrices and bilinear forms are in a one-to-one relationship. Thus properties of bilinear forms induce corresponding properties of matrices:

Definition 0.3.1.24. Symmetric positive definite matrices

A matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ is **symmetric positive definite (s.p.d.)**, if

$$\mathbf{A}^\top = \mathbf{A} \quad \text{and} \quad \vec{\xi}^\top \mathbf{A} \vec{\xi} > 0 \quad \forall \vec{\xi} \in \mathbb{R}^n \setminus \{\mathbf{0}\}.$$

0.3.1.4 Diagonalization of Matrices

Diagonalization is a central concern in linear algebra. It amounts to finding a basis of \mathbb{R}^n such that $\mathbf{A} \in \mathbb{R}^{n,n}$ becomes diagonal with respect to this basis.

$$\mathbf{AS} = \mathbf{SD}, \quad \mathbf{D} \in \mathbb{R}^{n,n} \text{ diagonal} \quad , \quad \mathbf{S} \in \mathbb{R}^{n,n} \text{ regular/invertible.} \quad (0.3.1.25)$$

In general real matrices cannot be diagonalized according to (0.3.1.25), but often they can be diagonalized in \mathbb{C} . There is one important class of matrices for which diagonalization in \mathbb{R} is always possible:

Theorem 0.3.1.26. Real diagonalization of symmetric matrices

For **every** symmetric matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, that is, $\mathbf{A}^\top = \mathbf{A}$, we can find a **diagonal** matrix $\mathbf{D} \in \mathbb{R}^{n,n}$ and an **orthogonal** matrix $\mathbf{Q} \in \mathbb{R}^{n,n}$ such that

$$\mathbf{Q}^\top \mathbf{A} \mathbf{Q} = \mathbf{D}. \quad (0.3.1.27)$$

This result can be extended to more general inner products: If $\mathbf{A} = \mathbf{A}^\top \in \mathbb{R}^{n,n}$ and $\mathbf{B} \in \mathbb{R}^{n,n}$ is s.p.d. (\rightarrow Def. 0.3.1.17), then we can find a regular matrix $\mathbf{S} \in \mathbb{R}^{n,n}$ such that

$$\mathbf{AS} = \mathbf{BSD} \quad , \quad \mathbf{D} \text{ diagonal} \quad , \quad \mathbf{S}^\top \mathbf{BS} = \mathbf{I}. \quad (0.3.1.28)$$

Review question(s) 0.3.1.29 (Tools from linear algebra)

(Q0.3.1.29.A) [Bilinear forms] Let V be a real vector space and $b : V \times V \rightarrow \mathbb{R}$ a bilinear form on V . Which statements about b are true?

- A) $b(\mathbf{x}, \mathbf{y}) = b(\mathbf{y}, \mathbf{x}) \quad \forall \mathbf{x}, \mathbf{y} \in V,$

True

False

B) $b(\alpha \mathbf{x}, \alpha \mathbf{y}) = \alpha^2 b(\mathbf{x}, \mathbf{y}) \quad \forall \mathbf{x}, \mathbf{y} \in V, \quad \forall \alpha \in \mathbb{R},$

True False

C) $b(\mathbf{x} + \mathbf{y}, \mathbf{x} - \mathbf{y}) = b(\mathbf{x}, \mathbf{x}) - b(\mathbf{y}, \mathbf{y}) \quad \forall \mathbf{x}, \mathbf{y} \in V$

True False

D) $b(\mathbf{x} + \alpha \mathbf{y}, \mathbf{z}) = \mathbf{x}, \mathbf{z} + \alpha b(\mathbf{y}, \mathbf{z}) \quad \forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in V, \quad \forall \alpha \in \mathbb{R},$

True False

E) $b\left(\sum_{i=1}^n \mathbf{x}_i, \sum_{i=1}^n \mathbf{y}_i\right) = \sum_{i=1}^n b(\mathbf{x}_i, \mathbf{y}_i) \quad \forall \mathbf{x}_i, \mathbf{y}_i \in V, \quad n \in \mathbb{N},$

True False

(Q0.3.1.29.B) [Symmetric positive-definite matrices] The matrix $\mathbf{A} \in \mathbb{R}^{m,n}$, $m, n \in \mathbb{N}$, is known to be **symmetric, positive-definite (s.p.d.)**. Which of the following conclusions can be drawn?

A) $m = n,$

True False

B) $\mathbf{A} = \mathbf{A}^\top,$

True False

C) $(\mathbf{A})_{i,j} \geq 0 \quad \forall i \in \{1, \dots, m\}, j \in \{1, \dots, n\},$

True False

D) $\mathbf{x}^\top \mathbf{A} \mathbf{x} \geq 0 \quad \forall \mathbf{x} \in \mathbb{R}^n,$

True False

E) $(\mathbf{A})_{i,i} > 0 \quad \forall i \in \{1, \dots, m\},$

True False

△

0.3.2 Tools from Real Analysis

The reader is expected to be familiar with basic concepts like continuity, differentiability, monotonicity, and curvature of functions $\mathbb{R} \mapsto \mathbb{R}$. Of course, she or he should also be able to differentiate and integrate functions in one dimension symbolically.



Supplementary literature. Most facts from real analysis needed for this course can be found in

M. STRUWE, *Analysis für Informatiker*. Lecture notes, ETH Zürich, 2009. [Link](#)

For the sake of brevity we will often resort to Landau symbols (“O”-notation): If f, g are two \mathbb{R} -valued functions defined in a neighborhood of 0 , then

$$f(\epsilon) = O(g(\epsilon)) \quad \text{for } \epsilon \rightarrow 0 \iff \exists C > 0 : |f(\epsilon)| \leq C|g(\epsilon)| \quad \forall \epsilon : |\epsilon| < \epsilon_0 , \quad (0.3.2.1)$$

for some $\epsilon_0 > 0$. Similarly, if f, g are defined for large arguments, then

$$f(x) = O(g(x)) \quad \text{for } x \rightarrow \infty \iff \exists C > 0 : |f(x)| \leq C|g(x)| \quad \forall x > x_0 ,$$

for a $x_0 \in \mathbb{R}$. A similar notation $f(n) = O(g(n))$ is used for functions defined on \mathbb{N}/\mathbb{Z} . For vector valued functions the modulus $|\cdot|$ has to be replaced with a suitable norm $\|\cdot\|$.

0.3.2.1 Calculus in 1D

From your introductory analysis course or even secondary school you should remember the notions of continuity for differentiability functions $\mathbb{R} \mapsto \mathbb{R}$. You should know how to compute higher derivatives for many elementary functions and how to apply the **chain rule** and the **product rule**.

☞ Notation: In 1D we write f' , f'' , and $f^{(k)}$ for the first, second, and k -th derivative, $k \in \mathbb{N}_0$, of a function $f : I \subset \mathbb{R} \rightarrow \mathbb{R}$.

Important is the 1D Taylor formula with integral remainder term that can be proved by repeated integration by parts:

Theorem 0.3.2.2. Taylor's theorem

If $f :]a, b[\rightarrow \mathbb{R}$, $a < b$, is $k+1$ -times continuously differentiable and $x \in]a, b[$, then

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \cdots + \frac{1}{k!}f^{(k)}(x)h^k + R(x; h) , \quad (0.3.2.3a)$$

with integral **remainder term**

$$R(x; h) = \int_x^{x+h} \frac{f^{(k+1)}(\xi)}{k!} (x-\xi)^k d\xi , \quad (0.3.2.3b)$$

for all $h : x+h \in]a, b[$.

The Taylor theorem can also be written with the remainder term in Lagrange form:

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \cdots + \frac{1}{k!}f^{(k)}(x)h^k + R(x; h) , \\ R(x; h) &= \frac{f^{(k+1)}(\zeta)}{(k+1)!} h^{k+1} \quad \text{for some } \zeta = \zeta(x; h) \in [\min\{x, x+h\}, \max\{x, x+h\}] . \end{aligned} \quad (0.3.2.4)$$

When we are solely interested in the decay of the remainder term as $h \rightarrow 0$, we often write

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \cdots + \frac{1}{k!}f^{(k)}(x)h^k + O(h^{k+1}) \quad \text{for } h \rightarrow 0 . \quad (0.3.2.5)$$

0.3.2.2 Differentiation in Multiple Dimensions

§0.3.2.6 (Derivatives) Let V, W be normed vector spaces, $f : D \subset V \rightarrow W$ differentiable on the open subset D . Then the **derivative** of f is a mapping

$$Df : D \rightarrow \mathcal{L}(V, W) := \{\text{linear mappings } V \mapsto W\} . \quad (0.3.2.7)$$

The derivative provides a local affine linear approximation of f :

$$f(v) \approx f(v_0) + Df(v_0)(v - v_0) . \quad (0.3.2.8)$$

Also $\mathcal{L}(V, W)$ is a normed vector space. Thus we can make sense of the derivative of Df , D^2f , as a mapping

$$D^2f : D \rightarrow \{\text{linear mappings } V \times V \mapsto W\} . \quad (0.3.2.9)$$

This can be iterated until we reach the k -th derivative

$$D^k f : D \rightarrow \{\text{linear mappings } \underbrace{V \times \cdots \times V}_{k \text{ times}} \mapsto W\} . \quad (0.3.2.10)$$

□

§0.3.2.11 (Partial derivatives [Str09, Sect. 7.1]) In (0.4.1.3) we already used the concept of partial derivatives. The **partial derivative** of a function $f : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}$ of d independent variables x_1, \dots, x_d ($f = f(x_1, \dots, x_d)$) in an interior point $\mathbf{x} = [x_1, \dots, x_d] \in \Omega$ with respect to x_j , $j = 1, \dots, d$, is defined as

$$\frac{\partial f}{\partial x_j}(\mathbf{x}) := \lim_{h \rightarrow 0} \frac{f(x_1, \dots, x_{j-1}, \mathbf{x}_j + h, x_{j+1}, \dots, x_d) - f(x_1, \dots, x_{j-1}, \mathbf{x}_j, x_{j+1}, \dots, x_d)}{h} , \quad (0.3.2.12)$$

if the limit exists. In other words, the partial derivative with respect to x_j is obtained by differentiating the function $x_j \mapsto f(x_1, \dots, x_{j-1}, x_j, x_{j+1}, \dots, x_d)$ as a function $\mathbb{R} \mapsto \mathbb{R}$, regarding all the other independent variables as mere parameters. Higher-order partial derivatives are simply defined by nesting the above definition, for instance

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}) := \frac{\partial}{\partial x_j} \left(\frac{\partial f}{\partial x_i} \right)(\mathbf{x}) \quad 1 \leq i, j \leq d . \quad (0.3.2.13)$$

A fundamental result about higher order partial derivatives is that their order does not matter in general:

Theorem 0.3.2.14. Partial derivatives commute

If all second partial derivatives of $f : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}^n$ are continuous functions on the open domain Ω , then

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}) = \frac{\partial^2 f}{\partial x_j \partial x_i}(\mathbf{x}) , \quad 1 \leq i, j \leq d , \quad \mathbf{x} \in \Omega .$$

The assertion of the theorem generalizes to all higher-order partial derivatives:

$$\frac{\partial^j u_i}{\partial x_{k_1} \cdots \partial x_{k_j}}(\mathbf{x}) , \quad k_\ell \in \{1, \dots, d\} , k_\ell \in \{1, \dots, j\} ,$$

is invariant with respect to permutations of k_1, \dots, k_j , if u is at least j -times continuously differentiable. □

§0.3.2.15 (The Jacobian) For a differentiable function $u : D \subset \mathbb{R}^n \rightarrow \mathbb{R}^d$ its derivative $Du(\mathbf{x})$ has matrix representation, the so-called **Jacobian**:

$$Du(\mathbf{x}) \leftrightarrow Du(\mathbf{x}) = \left[\frac{\partial u_i}{\partial x_j}(\mathbf{x}) \right]_{i,j=1}^n = \begin{bmatrix} \frac{\partial u_1}{\partial x_1}(\mathbf{x}) & \frac{\partial u_1}{\partial x_2}(\mathbf{x}) & \cdots & \cdots & \frac{\partial u_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial u_2}{\partial x_1}(\mathbf{x}) & & & & \frac{\partial u_2}{\partial x_n}(\mathbf{x}) \\ \vdots & & & & \vdots \\ \frac{\partial u_n}{\partial x_1}(\mathbf{x}) & \frac{\partial u_n}{\partial x_2}(\mathbf{x}) & \cdots & \cdots & \frac{\partial u_n}{\partial x_n}(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{n,n} . \quad (0.3.2.16)$$

□

§0.3.2.17 (The Hessian) A matrix representation can also be given for the second derivative of a real-valued function $u : D \subset \mathbb{R}^d \rightarrow \mathbb{R}$. It is known as **Hessian**

$$\mathbf{H}u(\mathbf{x}) = \mathbf{D}^2u(\mathbf{x}) = \left[\frac{\partial^2 u}{\partial x_i \partial x_j}(\mathbf{x}) \right]_{i,j=1}^d \in \mathbb{R}^{d,d}. \quad (0.3.2.18)$$

□

§0.3.2.19 (Differential operators) Differential operators are *special linear combinations of partial derivatives*. As such they spawn **linear operators** on spaces of differentiable functions defined on a domain $\Omega \subset \mathbb{R}^d$.

Important **first-order** differential operators are

- the **gradient**, defined for differentiable scalar functions $u : \Omega \rightarrow \mathbb{R}$, is the column vector

$$\mathbf{grad} u(\mathbf{x}) := \begin{bmatrix} \frac{\partial u}{\partial x_1} \\ \vdots \\ \frac{\partial u}{\partial x_d} \end{bmatrix} \in \mathbb{R}^d, \quad \mathbf{x} \in \Omega,$$

see Suppl. 1.2.1.21 for more details.

- the **divergence**, defined for differentiable **vector fields** $\mathbf{u} : \Omega \rightarrow \mathbb{R}^d$, is the scalar function

$$\operatorname{div} \mathbf{u}(\mathbf{x}) := \frac{\partial u_1}{\partial x_1}(\mathbf{x}) + \cdots + \frac{\partial u_d}{\partial x_d}(\mathbf{x}) \in \mathbb{R}, \quad \mathbf{x} \in \Omega,$$

refer to Suppl. 1.5.2.3.

- the **rotation**, defined for $d = 3$ and differentiable **vector fields** $\mathbf{u} = [u_1, u_2, u_3]^\top : \Omega \rightarrow \mathbb{R}^3$, is the column vector

$$\mathbf{curl} \mathbf{u}(\mathbf{x}) := \begin{bmatrix} \frac{\partial u_3}{\partial x_2} - \frac{\partial u_2}{\partial x_3} \\ \frac{\partial u_1}{\partial x_3} - \frac{\partial u_3}{\partial x_1} \\ \frac{\partial u_2}{\partial x_1} - \frac{\partial u_1}{\partial x_2} \end{bmatrix} \in \mathbb{R}^3, \quad \mathbf{x} \in \Omega.$$

These operators have distinct properties that account for their prominent occurrence in many mathematical models. For instance, from Thm. 0.3.2.14 we conclude by straightforward computations

$$\boxed{\mathbf{curl} \circ \mathbf{grad} = 0} \quad , \quad \boxed{\operatorname{div} \circ \mathbf{curl} = 0}. \quad (0.3.2.20)$$

A key **second-order** differential operator is the **Laplacian**, defined for a twice differentiable scalar function $u : \Omega \rightarrow \mathbb{R}$ as

$$\Delta u(\mathbf{x}) := \operatorname{div} \mathbf{grad} u(\mathbf{x}) = \frac{\partial^2 u}{\partial x_1^2}(\mathbf{x}) + \cdots + \frac{\partial^2 u}{\partial x_d^2}(\mathbf{x}), \quad \mathbf{x} \in \Omega.$$

□

0.3.2.3 Spaces of Continuously Differentiable Functions

Throughout we will need vector spaces of functions with particular properties, mainly concerning their smoothness. For the sake of concise presentation we have to introduce certain compact notations.

Given a set (domain) $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, we introduce the space of real-valued **continuous functions** on Ω

$$C^0(\Omega) := \{v : \Omega \rightarrow \mathbb{R} : v \text{ is continuous}\} .$$

If Ω is a closed set we say that the functions in $C^0(\Omega)$ are “continuous up to the boundary”. For an open set Ω the space $C^0(\Omega)$ can even contain unbounded functions!

If $I \subset \mathbb{R}$ is an (open or closed) interval we write

$$C^k(I) := \left\{ v : I \rightarrow \mathbb{R} : v^{(j)} := \frac{d^j v}{dx^j} \text{ exists and is continuous for all } j \in \{1, \dots, k\} \right\} ,$$

for the space of k -times, $k \in \mathbb{N}_0$, **continuously differentiable** functions on I . This definition can be extended to d -variate functions on $\Omega \subset \mathbb{R}^d$:

$$C^k(\Omega) := \left\{ v : \Omega \rightarrow \mathbb{R} : \frac{\partial^{|\alpha|} v}{\partial^{\alpha_1} x_1 \cdots \partial^{\alpha_d} x_d} \text{ exists and is continuous for all } \alpha \in \mathbb{N}_0^d, |\alpha| \leq k \right\} .$$

We can even set $k \leftarrow \infty$ and obtain the space $C^\infty(\Omega)$ of infinitely many times differentiable functions, also dubbed **smooth** functions.

§0.3.2.21 (Piecewise continuously differentiable functions) We will also need spaces of functions that are smooth locally:

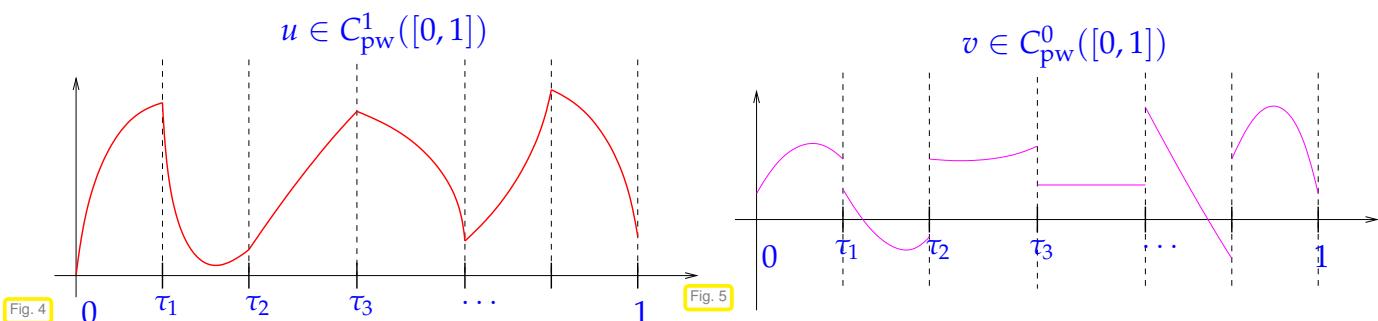
Definition 0.3.2.22. Piecewise continuously differentiable functions

For a **closed** domain $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, we define

$$C_{\text{pw}}^k(\Omega) := \{v \in C^{k-1}(\Omega) : v|_{\overline{\Omega}_j} \in C^k(\overline{\Omega}_j), j = 1, \dots, m\} ,$$

where $\{\Omega_j\}_{j=1}^m$, $m \in \mathbb{N}$, is a partition of Ω in the sense that $\Omega_k \cap \Omega_i = \emptyset$, if $i \neq k$, and $\Omega = \overline{\Omega}_1 \cup \dots \cup \overline{\Omega}_m$.

Let us take a closer look at such functions in one dimension, where Ω is an interval $[a, b]$. In this case functions in $C_{\text{pw}}^k([a, b])$ are **globally** C^{k-1} and piecewise k -times continuously differentiable functions on $[a, b] \subset \mathbb{R}$: For each $v \in C_{\text{pw}}^k([a, b])$ there is a finite partition $\{a = \tau_0 < \tau_1 < \dots < \tau_m = b\}$ such that $v|_{[\tau_{i-1}, \tau_i]}$ can be extended to a function $\in C^k([\tau_{i-1}, \tau_i])$. $C_{\text{pw}}^0([a, b]) \doteq$ piecewise continuous functions with only a finite number of discontinuities.



The following result is an immediate consequence of the definition

Corollary 0.3.2.23. Derivative of piecewise smooth functions

For $k \in \mathbb{N}$ differentiation of piecewise continuously differentiable functions on the interval $[a, b]$, $a < b$, yields other piecewise continuously differentiable functions:

$$u \in C_{\text{pw}}^k([a, b]) \quad \Rightarrow \quad \frac{du}{dx} \in C_{\text{pw}}^{k-1}([a, b]).$$

It also goes without saying that functions in $C_{\text{pw}}^0(\Omega)$ can be integrated over Ω and that

$$f \in C_{\text{pw}}^0(\Omega): \quad \int_{\Omega} f(x) dx = \sum_{j=1}^m \int_{\Omega_j} f(x) dx.$$

§0.3.2.24 (Functions with zero restriction to the boundary) For closed $\Omega \subset \mathbb{R}^d$, functions vanishing on the boundary of Ω represent special subspaces of $C_{\text{pw}}^k(\Omega)$ for any k :

$$C_{\text{pw},0}^k(\Omega) := \left\{ v \in C_{\text{pw}}^k(\Omega): v|_{\partial\Omega} = 0 \right\}.$$

Here, by $v|_{\partial\Omega}$ we mean the restriction of the function v to the boundary $\partial\Omega$.

0.3.2.4 Norms on Function Spaces

To investigate errors for solutions produced by numerical methods we need rigorous ways to measure the distance of functions. The right tools are norms on vector spaces of functions, which are special incarnations of the concept of “norm” introduced in Def. 0.3.1.10.

Here, we recall important norms on function spaces, cf. [Hip19, ??], [Hip19, ??], [Hip19, ??]:

Definition 0.3.2.25. Supremum norm

The **supremum norm** of an (essentially) bounded function $\mathbf{u} : \Omega \mapsto \mathbb{R}^n$ is defined as

$$\|\mathbf{u}\|_{\infty} (= \|\mathbf{u}\|_{L^{\infty}(\Omega)}) := \sup_{x \in \Omega} \|\mathbf{u}(x)\|, \quad \mathbf{u} \in (L^{\infty}(\Omega))^n. \quad (0.3.2.26)$$

- ◆ $L^{\infty}(\Omega)$ denotes the vector space of essentially bounded functions. It is the instance for $p = \infty$ of an L^p -space.
- ◆ The notation $\|\cdot\|_{\infty}$ hints at the relationship between the supremum norm of functions and the maximum norm for vectors in \mathbb{R}^n as defined in (0.3.1.12).
- ◆ For $n = 1$ the Euclidean vector norm in the definition reduces to the modulus $|\mathbf{u}(x)|$.
- ◆ The norm $\|\mathbf{u} - \mathbf{v}\|_{L^{\infty}(\Omega)}$ measures the maximal distance of point values of the functions \mathbf{u} and \mathbf{v} .

Many important norms on function spaces are based on integration, thus capturing non-local, large-scale properties.

Definition 0.3.2.27. Mean square norm/ L^2 -norm, see Def. 1.3.2.3

For a function $\mathbf{u} \in (C_{\text{pw}}^0(\Omega))^n$ the **mean square norm/ L^2 -norm** is given by

$$\|\mathbf{u}\|_0 (= \|\mathbf{u}\|_{L^2(\Omega)}) := \left(\int_{\Omega} \|\mathbf{u}(x)\|^2 dx \right)^{1/2}, \quad \mathbf{u} \in (L^2(\Omega))^n.$$

- ◆ $L^2(\Omega)$ designates the vector space of square integrable functions, a special specimen of an L^p -space (for $p = 2$) and a **Hilbert space**. We are going to revisit this notion in more detail later in this course in Rem. 1.3.3.9.
- ◆ The “0” in the notation $\|\cdot\|_0$ refers to the absence of derivatives in the definition of the norm.
- ◆ Obviously, the L^2 -norm is **weaker** than the supremum norm:

$$\|v\|_{L^2([a,b])} \leq \sqrt{|b-a|} \|v\|_{L^\infty([a,b])} \quad \forall v \in C_{pw}^0([a,b]).$$

In fact, we can find functions with arbitrarily small $L^2(\Omega)$ -norm that have huge values in some points. This will turn out to be a very important insight is will be discussed in Ex. 1.3.2.5.

- ◆ Parlance: $\|\mathbf{u} - \mathbf{v}\|_{L^2(\Omega)}$ is called the mean square distance in engineering sciences.

0.3.2.5 Multi-Dimensional Integration

The integral of a continuous function $u : \Omega \rightarrow \mathbb{R}$ on a “reasonably smooth” domain $\Omega \subset \mathbb{R}^d$ can be understood in the sense of **Riemann summation**.

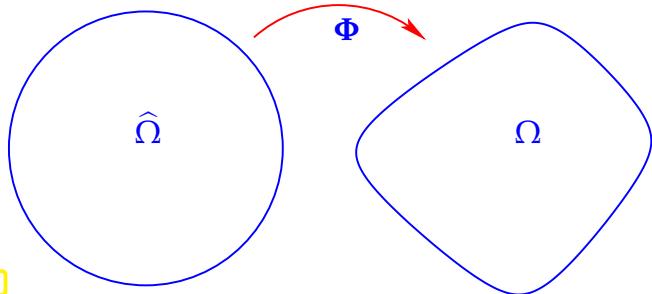
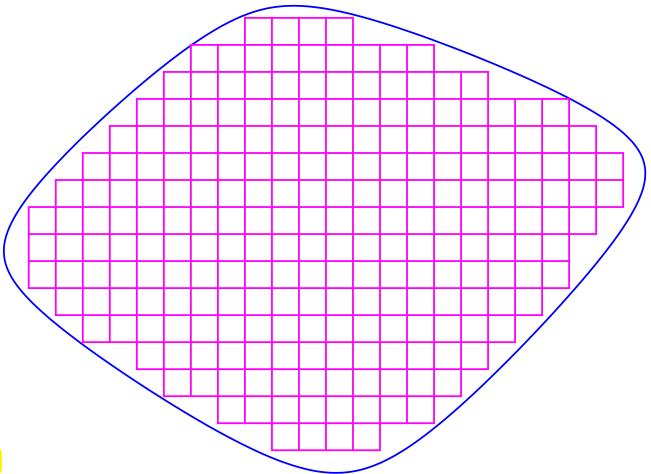
For $d = 2$ fill Ω with tiny axes-parallel squares Q_ℓ , $\ell = 1, \dots, n$, see figure, and write x_ℓ for their centers and define

$$I_n := \sum_{\ell=1}^n f(x_\ell) |Q_\ell|, \quad (0.3.2.28)$$

where $|Q_\ell|$ is the volume of the small square Q_ℓ . Then define the integral as the limit

$$\int_{\Omega} f(x) dx := \lim_{n \rightarrow \infty} I_n.$$

Fig. 6



Remember two facts:

- ◆ Locally, the function Φ can be well approximated by affine-linear map given through its linearization: for $x \in \Omega_0$ (\rightarrow § 0.3.2.6)

$$\Phi(x + h) \approx \Phi(x) + D\Phi(x)h \quad \text{for small } h \in \mathbb{R}^d, \quad (0.3.2.29)$$

where $D\Phi(x) \in \mathbb{R}^{d,d}$ is Jacobian of Φ in x (\rightarrow § 0.3.2.15).

- ◆ An affine-linear map $L : \mathbb{R}^d \mapsto \mathbb{R}^d$, $x \mapsto Tx + t$, $T \in \mathbb{R}^{d,d}$, $t \in \mathbb{R}^d$ involves a change of volume controlled by the determinant of its matrix:

$$\text{Vol}(L(V)) = |\det T| \text{ Vol}(V) \quad \forall \text{ “volumes” } V \subset \mathbb{R}^d. \quad (0.3.2.30)$$

Now, put the pieces together; the definition of the integral by Riemann summation, the possibility of a local affine-linear approximation of Φ , and the change of volume of tiny hypercubes when they are transported by Φ . This reasoning eventually leads to the following result:

Theorem 0.3.2.31. Transformation formula for integrals

Given two domains $\widehat{\Omega}, \Omega \subset \mathbb{R}^d$ and a continuously differentiable mapping $\Phi : \widehat{\Omega} \rightarrow \overline{\Omega}$, then

$$\int_{\Omega} f(x) dx = \int_{\widehat{\Omega}} f(\Phi(\hat{x})) |\det D\Phi(\hat{x})| d\hat{x} \quad (0.3.2.32)$$

for any integrable function $f : \Omega \rightarrow \mathbb{R}$.

§0.3.2.33 (Surface integrals) An embedded C^k -surface Σ in \mathbb{R}^d is a subset that, locally, is the image of a “parameter domain” $\widehat{\Omega} \subset \mathbb{R}^{d-1}$:

$$\forall x \in \Sigma: \exists \text{neighborhood } U(x) \subset \mathbb{R}^d, \widehat{\Omega} \subset \mathbb{R}^{d-1}: \Sigma \cap U(x) = \Phi(\widehat{\Omega}) \text{ with } \Phi \in C^k(\mathbb{R}^{d-1}, \mathbb{R}^d).$$

Let us assume that the surface Σ can be *parameterized* by a single function $\Phi : \widehat{\Omega} \subset \mathbb{R}^{d-1} \rightarrow \mathbb{R}^d$: $\Sigma = \Phi(\widehat{\Omega})$. Then the **surface integral** of a function $\varphi : \Sigma \rightarrow \mathbb{R}$ is defined as

$$\int_{\Sigma} \varphi(x) dS(x) := \int_{\widehat{\Omega}} \varphi(\Phi(\hat{x})) \sqrt{\det(D\Phi(\hat{x})^T D\Phi(\hat{x}))} d\hat{x}, \quad (0.3.2.34)$$

where $D\Phi(\hat{x}) \in \mathbb{R}^{d,d-1}$ is the Jacobian of Φ , see (0.3.2.16). \square

Review question(s) 0.3.2.35 (Tools from real analysis)

(Q0.3.2.35.A) [An important differential operator] Let $\Omega \subset \mathbb{R}^d$ a d -dimensional domain and consider a smooth function $u : \Omega \rightarrow \mathbb{R}$, $u = u(x)$, $x = [x_1, \dots, x_d]^T$. What is Δu ?

A) A differential operator called the **Laplacian**,

True

False

B) the so-called **Dirac operator**,

True

False

C) $\Delta u = \sum_{i=1}^d \left(\frac{\partial u}{\partial x_i} \right)^2,$

True

False

D) $\Delta u = \sum_{i=1}^d \frac{\partial^2 u}{\partial x_i^2},$

True

False

E) $\Delta u = \operatorname{div} \operatorname{grad} u,$

True

False

F) $\Delta u = \operatorname{grad} \operatorname{div} u$

True

False

(Q0.3.2.35.B) [Gauss theorem, see Thm. 1.5.2.4] Which of the following mathematical statements are known as or equivalent to the so-called **Gauss theorem/divergence theorem**? Here $\mathbf{u} : \Omega \rightarrow \mathbb{R}^d$ is a vector field defined on $\Omega \subset \mathbb{R}^d$, $f : \Omega \rightarrow \mathbb{R}$ a real-valued function, and $\mathbf{n} : \partial\Omega \rightarrow \mathbb{R}^d$ stands for the exterior unit normal vector field.

A) $\operatorname{div} \operatorname{curl} \mathbf{u} = 0$

True

False

B) $\operatorname{div}(f\mathbf{u}) = (\operatorname{grad} f)^\top \mathbf{u} + f \operatorname{div} \mathbf{u},$

True

False

C) $\int_{\Omega} \operatorname{div} \mathbf{u}(\mathbf{x}) d\mathbf{x} = \int_{\partial\Omega} \mathbf{u}(\mathbf{x})^\top \mathbf{n}(\mathbf{x}) dS,$

True

False

D) $\int_{\Omega} \frac{\partial f}{\partial x_i}(\mathbf{x}) d\mathbf{x} = \int_{\partial\Omega} f(\mathbf{x})(\mathbf{n}(\mathbf{x}))_i dS.$

True

False

(Q0.3.2.35.C) [A gradient] What is the gradient of the **Euclidean norm** function $\mathbf{x} \in \mathbb{R}^d \mapsto \|\mathbf{x}\|_2$?

A) $\operatorname{grad}\{\mathbf{x} \mapsto \|\mathbf{x}\|_2\} = 2\|\mathbf{x}\|_2$ for all $\mathbf{x} \in \mathbb{R}^d$?

True

False

B) $\operatorname{grad}\{\mathbf{x} \mapsto \|\mathbf{x}\|_2\} = \mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^d$?

True

False

C) $\operatorname{grad}\{\mathbf{x} \mapsto \|\mathbf{x}\|_2\} = \frac{\mathbf{x}}{\|\mathbf{x}\|_2}$ for all $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$?

True

False

D) $\operatorname{grad}\{\mathbf{x} \mapsto \|\mathbf{x}\|_2\} = \|\mathbf{x}\|_2 \mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^d$?

True

False

△

0.3.3 Required Elementary Numerical Methods

This course builds upon the ETH lecture 401-0663-00L **Numerical Methods for CSE**, see [Hip19]. In particular, the following topics from computational mathematics provide fundamental tools for this course

- Techniques for handling **sparse matrices** and **sparse linear systems**, see [Hip19, ??].
- **Numerical quadrature**, concepts and methods as introduced in [Hip19, ??].
- Numerical method for solving initial value problems for ordinary differential equations (**numerical integration**), in particular **stiff** initial value problems as discussed in [Hip19, ??].

Lecture notes for the course “Numerical Methods for CSE are available” for download [here](#).

0.4 Mathematical Modelling with Partial Differential Equations

§0.4.0.1 (Continuum models) Partial differential equations (**PDEs**) are at the core of most mathematical models arising from a **continuum approach**, where the configuration or state of a system is described by means of a **function** on a (**multi-dimensional**) **domain** $\Omega \subset \mathbb{R}^d$, $d \geq 1$. In fact, in one dimension, for $d = 1$, the models will involve ordinary differential equations (ODEs) rather than partial differential equations. Yet, in many cases the dimension d can be regarded as a parameter for a family of models and the case $d = 1$ is not really special and shares many traits with models for the genuinely multi-dimensional setting $d > 1$.

Therefore, the title of the course is a slight misnomer; a more appropriate title would be

Numerical Methods for Continuum Models with Local Interactions

but this sounds a bit clumsy, doesn't it? □

Next, notations used for stating partial differential equations will be explained. Then a few examples of mathematical models based on PDEs will be presented in a cursory way, in order to convey their diversity and wide scope.

0.4.1 PDEs: Basic Notions

§0.4.1.1 (Formal notion of a partial differential equation (**PDE**))

A partial differential equation for an unknown function $\mathbf{u} = [u_1, \dots, u_n]^\top : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}^n$, $d, n \in \mathbb{N}$, depending on the independent variables x_1, \dots, x_d ($\mathbf{u} = \mathbf{u}(x_1, \dots, x_d)$) has the form

$$F(\mathbf{u}, D\mathbf{u}, D^2\mathbf{u}, \dots, D^m\mathbf{u}) = \mathbf{0}, \quad (0.4.1.2)$$

where F is a general function and $D^j\mathbf{u}$ denotes a tensor of dimension $n \times d \times \dots \times d$ with nd^j entries defined as (→ § 0.3.2.11)

$$(D^j\mathbf{u}(x))_{i,k_1,\dots,k_j} := \frac{\partial^j u_i}{\partial x_{k_1} \dots \partial x_{k_j}}(x), \quad k_\ell \in \{1, \dots, d\}, \ell \in \{1, \dots, j\}. \quad (0.4.1.3)$$

☞ notation: we write x_1, \dots, x_d for the independent “spatial” variables, $\mathbf{x} = [x_1, \dots, x_d]^\top \in \mathbb{R}^d$

Note that for $j = 1$ the derivative $D\mathbf{u}$ boils down to the classical **Jacobian** of \mathbf{u} as already defined in § 0.3.2.15

$$D\mathbf{u}(\mathbf{x}) = \left[\frac{\partial u_i}{\partial x_j}(\mathbf{x}) \right]_{i,j=1}^n = \begin{bmatrix} \frac{\partial u_1}{\partial x_1}(\mathbf{x}) & \frac{\partial u_1}{\partial x_2}(\mathbf{x}) & \dots & \dots & \frac{\partial u_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial u_2}{\partial x_1}(\mathbf{x}) & & & & \frac{\partial u_2}{\partial x_n}(\mathbf{x}) \\ \vdots & & & & \vdots \\ \frac{\partial u_n}{\partial x_1}(\mathbf{x}) & \frac{\partial u_n}{\partial x_2}(\mathbf{x}) & \dots & \dots & \frac{\partial u_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}. \quad (0.3.2.16)$$

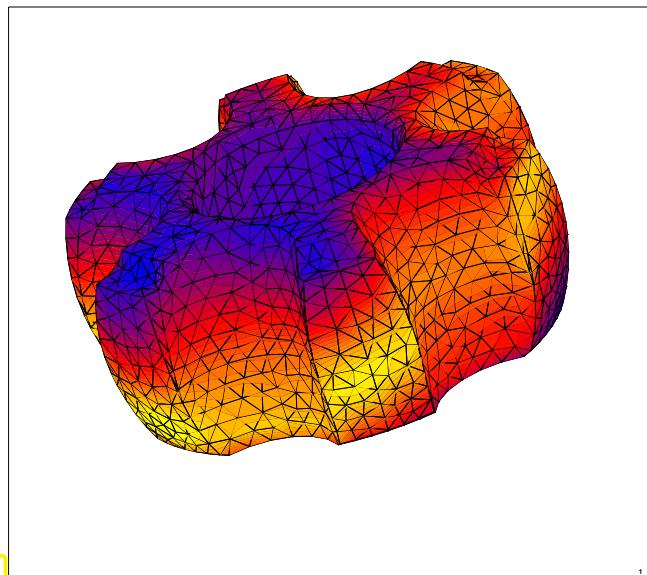
For $j = 1$ and $n = 1$ the matrix $D^2\mathbf{u}$ agrees with the **Hessian** $H\mathbf{u}$ of the scalar valued function $u = u_1$:

$$H\mathbf{u}(\mathbf{x}) = D^2\mathbf{u}(\mathbf{x}) = \left[\frac{\partial^2 u}{\partial x_i \partial x_j}(\mathbf{x}) \right]_{i,j=1}^d \in \mathbb{R}^{d,d}. \quad (0.3.2.18)$$

Note that $D^j\mathbf{u}$ may not be well defined in some $\mathbf{x} \in \Omega$ in case \mathbf{u} is not “sufficiently smooth” (j -times differentiable). □

0.4.2 Electromagnetics: Eddy Current Problem

A model for the behavior of low-frequency electromagnetic fields with harmonic dependence on time:



Eddy current model [AV10]

$$\begin{aligned}\operatorname{curl} \mathbf{E} &= -i\omega\mu(\mathbf{x})\mathbf{H} && \text{in } \Omega, \\ \operatorname{curl} \mathbf{H} &= \sigma(\mathbf{x})\mathbf{E} + \mathbf{j}_s && \text{in } \Omega, \\ \mathbf{E} \times \mathbf{n} &= 0 && \text{on } \partial\Omega.\end{aligned}\quad (0.4.2.1)$$

$$\begin{aligned}\omega > 0 &\hat{=} \text{angular frequency} \\ \sigma = \sigma(\mathbf{x}) \geq 0 &\hat{=} \text{conductivity} \\ \mu = \mu(\mathbf{x}) > 0 &\hat{=} \text{magnetic permeability} \\ \mathbf{E} : \Omega \mapsto \mathbb{C}^3 &\hat{=} \text{electric field} \\ \mathbf{H} : \Omega \mapsto \mathbb{C}^3 &\hat{=} \text{magnetic field}\end{aligned}$$

(Known: σ, μ , unknowns: \mathbf{E}, \mathbf{H} : **complex fields!**)

▷ induction heating simulation: surface electric field
(boundary element simulation [HO04])

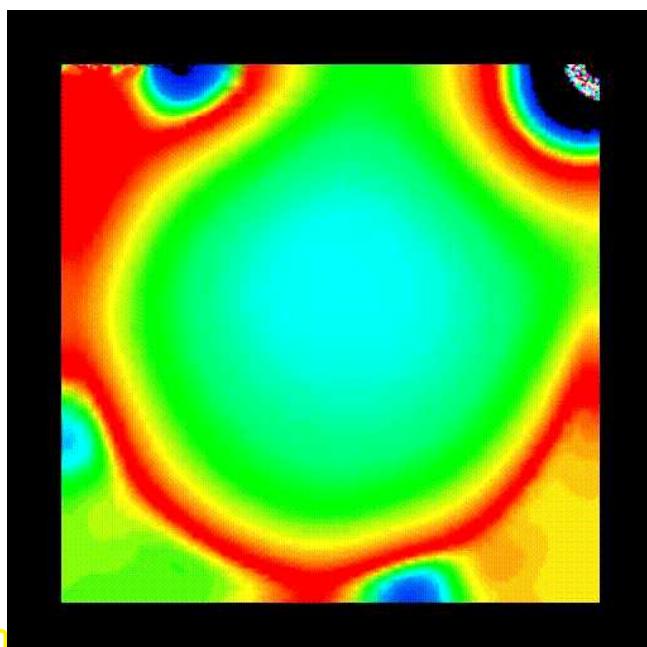
Remark 0.4.2.2 (Truncation of unbounded domain) Generically, the electromagnetic equations are posed on the **unbounded domain** $\Omega = \mathbb{R}^3$ and have to be supplemented by the **decay conditions**

$$\mathbf{E}(\mathbf{x}) \rightarrow 0 \quad \text{uniformly for } \|\mathbf{x}\| \rightarrow \infty. \quad (0.4.2.3)$$

In practice, (0.4.2.3) is often approximated by switching to a bounded domain $\Omega \subset \mathbb{R}^3$ and imposing vanishing tangential components of the electric field \mathbf{E} on the boundary $\partial\Omega$, as it is done in (0.4.2.1). □

Remark 0.4.2.4 (Degenerate elliptic boundary value problem) The eddy current equations in frequency domain (0.4.2.1) belong to the class of degenerate second-order **elliptic boundary value problems**. They are called degenerate, because \mathbf{E} is not uniquely determined where $\sigma \equiv 0$. To see this recall (0.3.2.20): In regions where $\sigma \equiv 0$ we can add any gradient to \mathbf{E} and it will still be a solution of (0.4.2.1). □

0.4.3 Viscous Fluid Flow



(Stationary, incompressible) **Navier-Stokes equations**:

$$\begin{aligned}-\nu\Delta\mathbf{u} + \mathbf{D}\mathbf{u} \cdot \mathbf{u} + \operatorname{grad} p &= \mathbf{f} && \text{in } \Omega, \\ \operatorname{div} \mathbf{u} &= 0 && \text{in } \Omega, \\ \mathbf{u} &= 0 && \text{on } \partial\Omega.\end{aligned}\quad (0.4.3.1)$$

$$\begin{aligned}\nu &\hat{=} \text{dynamic viscosity} \\ \mathbf{f} : \Omega \mapsto \mathbb{R}^3 &\hat{=} \text{given external force field} \\ \mathbf{u} : \Omega \mapsto \mathbb{R}^3 &\hat{=} \text{velocity field (unknown)} \\ p : \Omega \mapsto \mathbb{R} &\hat{=} \text{pressure (unknown)}\end{aligned}$$

If the convective term $\mathbf{D}\mathbf{u} \cdot \mathbf{u}$ is omitted we obtain the **Stokes-equations**, see Chapter 12

▷ **Lid driven cavity flow**, pressure distribution (finite element simulation with **FEATFLOW**)

The Navier-Stokes equations (0.4.3.1) describe the motion of viscous ("sticky") fluid under external forces. The boundary conditions mean that the fluid sticks to the wall of the container Ω (no-slip boundary condition).

ary conditions). The equations (0.4.3.1) provide the fundamental model in computational fluid dynamics (CFD).

0.4.4 Micromagnetics

Micromagnetics deals with the evolution of the time-dependent magnetization $\mathbf{m} = \mathbf{m}(\mathbf{x}, t)$, of a ferromagnetic material under the influence of an external magnetic field. The main quasi-stationary model are the [Landau-Livshits-Gilbert equations](#) here given in scaled (non-dimensional) form, see [Pro01]:

$$\begin{aligned} \frac{\partial \mathbf{m}}{\partial t} - \mathbf{m} \times \frac{d\mathcal{E}(\mathbf{m}, \psi(\mathbf{m}))}{d\mathbf{m}} - \alpha \mathbf{m} \times (\mathbf{m} \times \frac{d\mathcal{E}(\mathbf{m}, \psi(\mathbf{m}))}{d\mathbf{m}}) &= 0 && \text{in } \Omega \times [0, T], \\ -\Delta \psi + \operatorname{div} \mathbf{m} &= 0 && \text{in } \mathbb{R}^3 \times [0, T]. \\ |\psi(\mathbf{x})| &= O(|\mathbf{x}|^{-1}) \quad \text{for } |\mathbf{x}| \rightarrow \infty, \\ \mathbf{m}(\cdot, 0) &= \mathbf{m}_0(\cdot) && \text{in } \Omega. \end{aligned} \quad (0.4.4.1)$$

with scaled Gibbs free energy

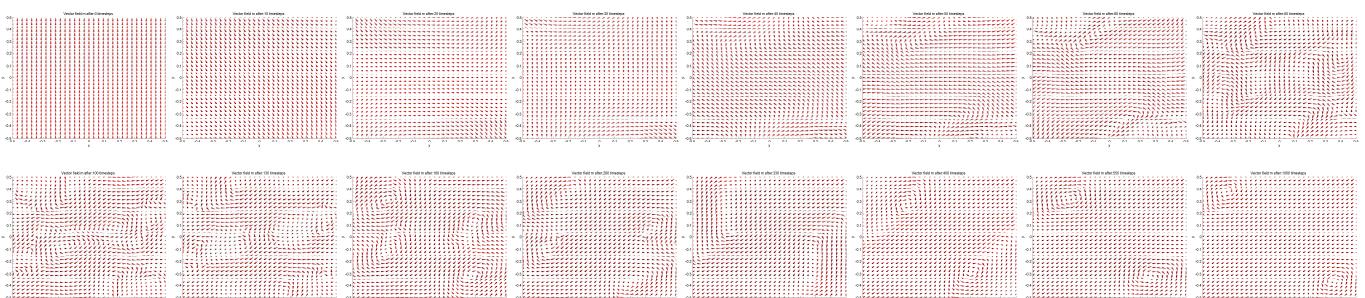
$$\mathcal{E}(\mathbf{m}, \psi) = \frac{1}{2} \int_{\Omega} \eta |\operatorname{grad} \mathbf{m}|^2 + Q(1 - (\mathbf{d} \cdot \mathbf{m})^2) - 2\mathbf{H}_0 \cdot \mathbf{m} \, dx + \frac{1}{2} \int_{\mathbb{R}^3} |\operatorname{grad} \psi|^2 \, dx.$$

The fields and coefficients occurring in the model are

$$\begin{aligned} \mathbf{m} : \Omega \times [0, T] \rightarrow \mathbb{S}^2 &\triangleq \text{magnetization (direction field, } \|\mathbf{m}\| = 1, \text{ if } \|\mathbf{m}_0\| = 1); \\ &\quad \text{(the unknown of the model)} \\ \psi : \mathbb{R}^3 \rightarrow \mathbb{R} &\triangleq \text{magnetic scalar potential} \\ \alpha > 0 &\triangleq \text{damping parameter} \\ Q > 0, \mathbf{d} \in \mathbb{R}^3 &\triangleq \text{strength/direction of material anisotropy} \\ \mathbf{m}_0 &\triangleq \text{initial magnetization} \end{aligned}$$

The equations (0.4.4.1) describe a parabolic *gradient flow system* for the Gibbs free energy on the manifold of director fields, that is, vector fields with modulus 1.

Flipping of magnetization, computed by means of a [finite element simulation](#) [Cor03], more details about finite element method (FEM) are given in Chapter 2.



We observe the formation of vortices, which finally disappear at the upper left and the lower right corners. In the final state, the elementary magnets tend to point in the same direction.

0.4.5 Reaction-Diffusion: Phase Separation

The [Cahn-Hilliard equation](#) is a PDE of mathematical physics which describes the process of phase separation, by which the two components of a binary fluid spontaneously separate and form domains pure in each component. u is the concentration of one phase of the fluid, with $u = \pm 1$ indicating domains. Here

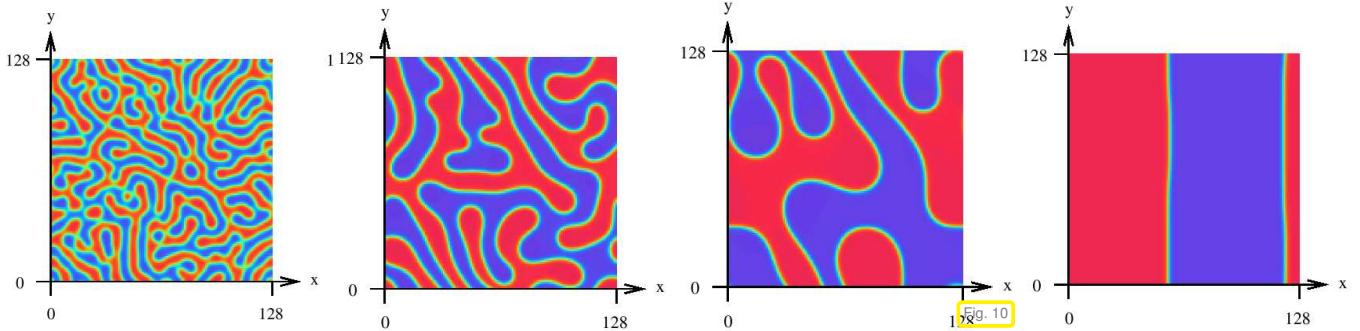
we give a boundary value evolution problem for the Cahn-Hilliard equation in scaled (non-dimensional) form:

$$\begin{aligned} \frac{du}{dt} - \alpha \Delta(u^3 - u - \gamma \Delta u) &= 0 && \text{in } \Omega \times]0, T[, \\ u(\cdot, 0) &= u_0 && \text{in } \Omega . \\ \mathbf{grad}(u^3 - u - \gamma \Delta u) \cdot \mathbf{n} &= 0 && \text{on } \partial\Omega , \\ \frac{1}{\Gamma_s} \frac{\partial u}{\partial t} + \mathbf{grad} u \cdot \mathbf{n} + \sigma \Delta u &= 0 && \text{on } \partial\Omega . \end{aligned} \quad (0.4.5.1)$$

- $u = u(x, t)$ $\hat{=}$ time-dependent concentration (unknown)
- $\alpha > 0$ $\hat{=}$ known diffusion coefficient
- $\gamma > 0$ $\hat{=}$ known diffusion length

The equations (0.4.5.1) describe a gradient flow system with “mass conservation” for the chemical potential.

Evolution snapshots (finite difference discretization, [Ken+00]):



0.4.6 Activator-Inhibitor Models: Fur Patterns

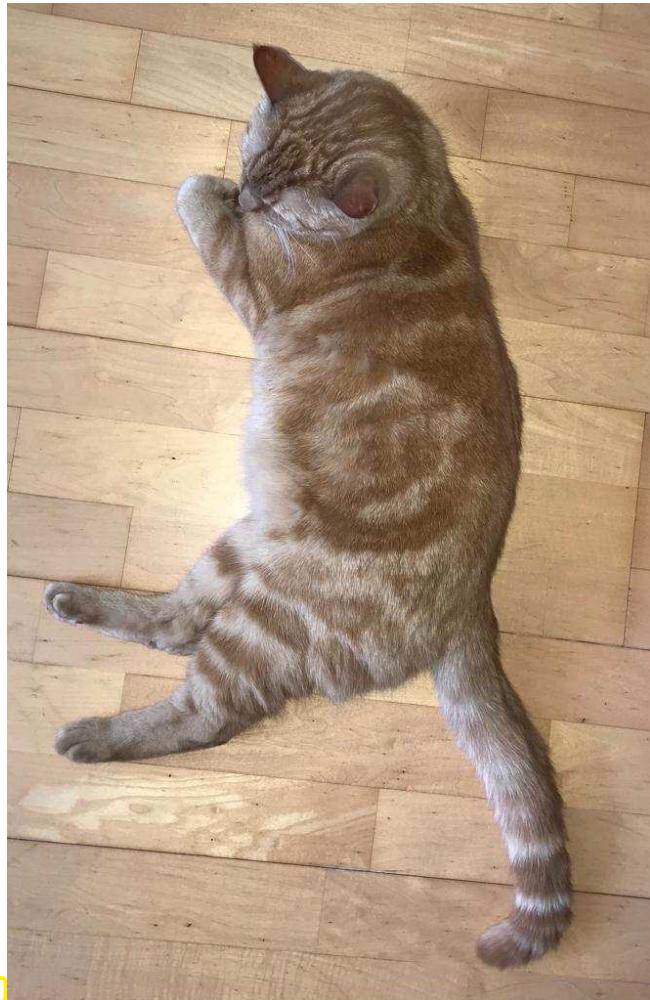


Fig. 11

◀ Cats come equipped with an elaborate PDE solver in order to generate fur patterns that are perfectly adapted to their environment.

Fur patterns of various feline species can be obtained as solutions of non-linear parabolic evolution problems known as **activator-inhibitor reaction-diffusion models** for the concentrations $u = u(\mathbf{x}, t)$, $v = v(\mathbf{x}, t)$ of signaling compounds [SW03]:

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta u &= \gamma f(u, v) , \\ \frac{\partial v}{\partial t} - \Delta v &= \gamma g(u, v) \end{aligned} \quad \text{in } \Omega ,$$

with functions

$$\begin{aligned} f(u, v) &:= a - u - h(u, v) , \\ g(u, v) &:= \alpha(b - v) - h(u, v) , \\ h(u, v) &:= \frac{uv}{1 + u + Ku^2} , \end{aligned}$$

where γ, α, a, b, K are parameters controlling the details of the pattern formation process. Initial conditions and boundary conditions have been omitted above.

0.4.7 Quantitative Finance: Black-Scholes Equation

The task of **option pricing** for European options leads to the **Black-Scholes equation** on \mathbb{R}_+^d [AP05]:

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{1}{2} \sum_{i,j=1}^d q_{ij} x_i x_j \frac{\partial^2 u}{\partial x_i \partial x_j} + r \sum_{i=1}^d x_i \frac{\partial u}{\partial x_i} - ru &= 0 \quad \text{in } \mathbb{R}_+^d \times [0, T] , \\ + \text{"exact boundary values" imposed on } \partial \mathbb{R}_+^d , \quad (0.4.7.1) \end{aligned}$$

$$u(T, \cdot) = g(\cdot) \quad \text{in } \Omega .$$

◆ $d \in \mathbb{N}$ = no. of underlying stocks, $\mathbf{x} = (x_1, \dots, x_d)^T$, $x_i \leftrightarrow$ price of stock # i

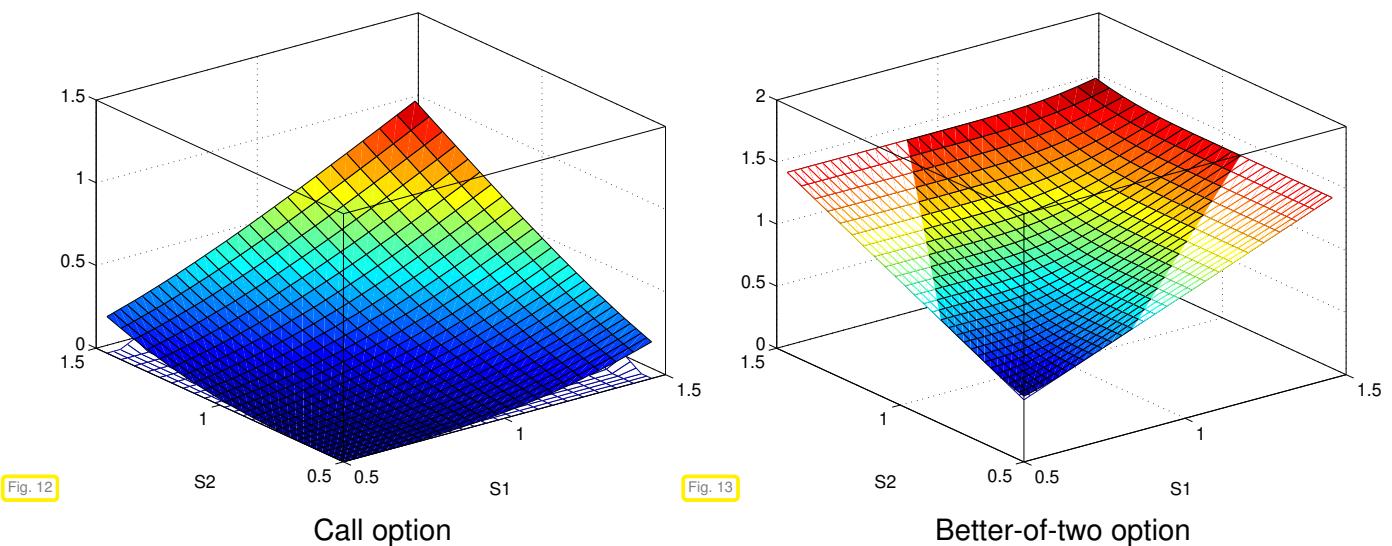
◆ Unknown $u = u(\mathbf{x}, t) \hat{=} \text{option price at time } t \text{ given stock prices } \mathbf{x}$:

$$u(t, \mathbf{x}) = \mathbb{E}(\exp(-r(t-T))g(S_T) | S_t = \mathbf{x}) ,$$

with payoff function $g : \mathbb{R}_+^d \mapsto \mathbb{R}$.

◆ Coefficients $r > 0$ = interest rate, $(q_{ij})_{i,j=1}^d$ = s.p.d. covariance matrix

This is a **high-dimensional** degenerate parabolic initial-boundary value problem. The Stock price fluctuations are modelled by means of a Wiener process (log-normal distribution) $S_i(t) = \exp(rt + X_t^i)$. Here we give numerical simulations in $d = 2$ with linear **finite elements** on tensor product mesh (MATLAB computations, C. Winter, SAM, ETH Zürich):



The payoff functions used in the computations are

$$\text{call option: } g(S_1, S_2) := \max\{S_1 + S_2 - K, 0\} , \quad \text{with strike price } K ,$$

$$\text{better-of-two option: } g(S_1, S_2) := \max\{S_1, S_2\} .$$

More details \Rightarrow course “Computational Methods for Quantitative Finance” (Ch. Schwab)

0.4.8 Quantum Mechanics: Electronic Schrödinger Equation

The following equations formulate an elliptic **eigenvalue problem** obtained from the Born-Oppenheimer approximation of the Schrödinger equation, the fundamental governing equation for quantum phenomena. Its solutions describe the cloud of electrons around for a molecule at different excited states.

$$\left(-\frac{1}{2}\Delta + \sum_{i=1}^N \sum_{j=1}^P \frac{Z_j}{|x_i - r_j|} + \sum_{i=1}^N \sum_{j>i}^N \frac{1}{|x_i - x_j|} \right) u = \lambda u , \quad (0.4.8.1)$$

+ exponential decay of u for $|x| \rightarrow \infty$.

- ◆ N = number of electrons
- ◆ P = number of nuclei (with charges $Z_j \in \mathbb{N}$ and positions $r_j \in \mathbb{R}^3$)
- ◆ Unknown: $u = u(x_1, \dots, x_N) \neq 0$, $x_i \in \mathbb{R}^3$! \rightarrow probability density $|u|^2$
- ◆ Unknown: eigenvalue λ = state energy

► High-dimensional elliptic eigenvalue problem on \mathbb{R}^{3N} !

Numerical simulation: states ($N, P = 1$) computed with **spectral sparse grid Galerkin method** [GH05]:

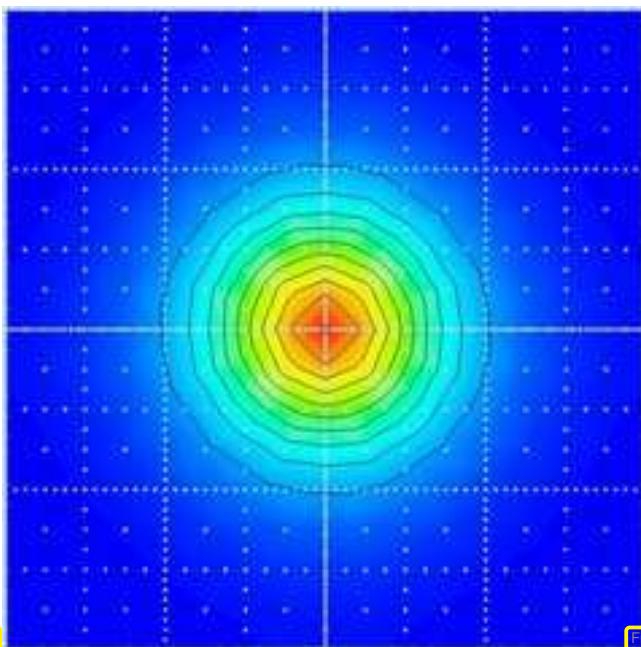


Fig. 14

Symmetric ground state

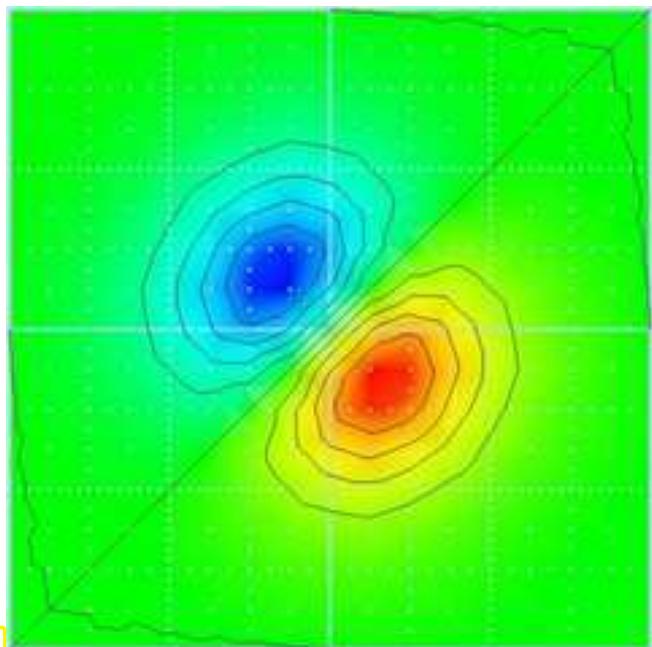


Fig. 15

Anti-symmetric ground state

0.4.9 Rarefied Gas Dynamics: Boltzmann Equation

The state of a rarefied gas occupying the bounded region of space $\Omega \subset \mathbb{R}^3$ can be described by a **density function** $f = f(\mathbf{x}, \mathbf{v}, t)$, which is a function of space (\mathbf{x}), of velocity (\mathbf{v}), and time (t). Its meaning is the following: the integral

$$\int_{B_x} \int_{B_v} f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} d\mathbf{x}, \quad B_x \subset \Omega, \quad B_v \subset \mathbb{R}^3,$$

yields the number of gas molecules located inside B_x and travelling with a velocity in B_v at time t . The evolution of the density function is governed by the **Boltzmann equation**

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \mathbf{grad}_{\mathbf{x}} f = Q(f, f) \quad \text{in } \Omega \times \mathbb{R}^3, \quad (0.4.9.1)$$

supplemented with the **inflow boundary conditions**

$$u(\mathbf{x}, \mathbf{v}, t) = g(\mathbf{x}, \mathbf{v}, t) \quad \text{for } \mathbf{x} \in \partial\Omega, \quad \mathbf{v} \cdot \mathbf{n}(\mathbf{x}) < 0, \quad (0.4.9.2)$$

where g are given boundary data. The **collision operator** is given by

$$Q(f, g)(\mathbf{x}, \mathbf{v}, t) := \int_{\mathbb{R}^3} \int_{S^2} B(\|\mathbf{v} - \mathbf{v}_*\|, \cos \theta) (h'_* f' - h_* f) d\sigma d\mathbf{v}_*, \quad (0.4.9.3)$$

$$\begin{aligned} f &:= f(\mathbf{x}, \mathbf{v}, t), & h_* &:= h(\mathbf{x}, \mathbf{v}_*, t), & f' &:= f(\mathbf{x}, \mathbf{v}', t), & h'_* &:= h(\mathbf{x}, \mathbf{v}'_*, t), \\ \mathbf{v}' &:= \frac{1}{2}(\mathbf{v} + \mathbf{v}_* + \|\mathbf{v} - \mathbf{v}_*\|), & \mathbf{v}'_* &:= \frac{1}{2}(\mathbf{v} + \mathbf{v}_* - \|\mathbf{v} - \mathbf{v}_*\| \boldsymbol{\sigma}). \end{aligned}$$

The function $B : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is the so-called **collision kernel**, and S^2 stands for the unit sphere. The angle θ is enclosed by the two velocities \mathbf{v} and \mathbf{v}' .

Note: The problem (0.4.9.1) is moderately high-dimensional, since it is posed on a seven-dimensional unbounded domain $\Omega \times \mathbb{R} \times \mathbb{R}$.

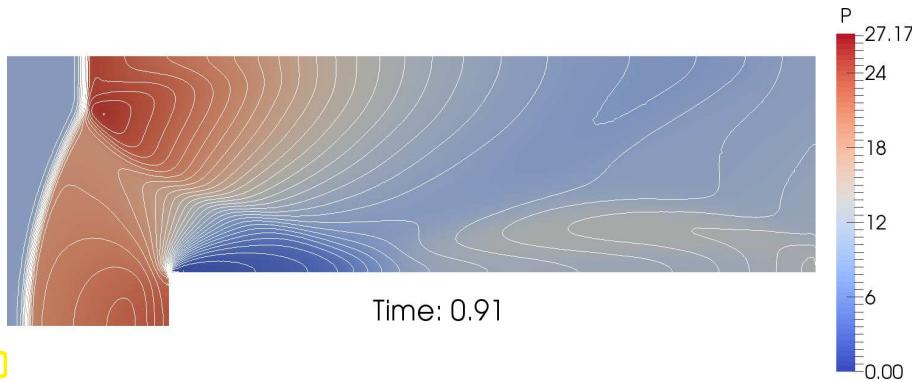


Fig. 16

Mach-3 CFD benchmark problem. Inflow on left boundary, specular reflective wall, outflow boundary conditions on the right. Computation with least squares finite elements in space and spectral polynomial approximation in velocity, see [GHP17].

0.4.10 Wave Propagation: Helmholtz equation

Time-harmonic acoustic waves are described by the spatio-temporal behavior of sound pressure $\mathbf{o} = \mathbf{p}(\mathbf{x}, t)$. In linear media without sources it satisfies the homogeneous **Helmholtz equation**

$$\Delta \mathbf{p} + k^2 n(\mathbf{x}) \mathbf{p} = 0 \quad \text{in } \Omega , \quad (0.4.10.1)$$

where $k > 0$ is the wave number (inversely proportional to the frequency), and $n = n(\mathbf{x})$ is a dimensionless spatially varying refractive index of the medium.

Often the Helmholtz equation is posed on unbounded domain, for instance $\Omega = \mathbb{R}^3$. In this case we need a **radiation condition** at ∞ :

$$\lim_{\|\mathbf{x}\| \rightarrow \infty} \|\mathbf{x}\| \left(\frac{\partial p_s}{\partial r}(\mathbf{x}) - ik p_s(\mathbf{x}) \right) = 0 \quad \text{uniformly ,} \quad (0.4.10.2)$$

where $p_s := \mathbf{p} - \mathbf{p}_{\text{inc}}$ is the scattered field, the difference of the pressure field \mathbf{p} and another pressure field \mathbf{p}_{inc} that belongs to an incident exciting acoustic wave.

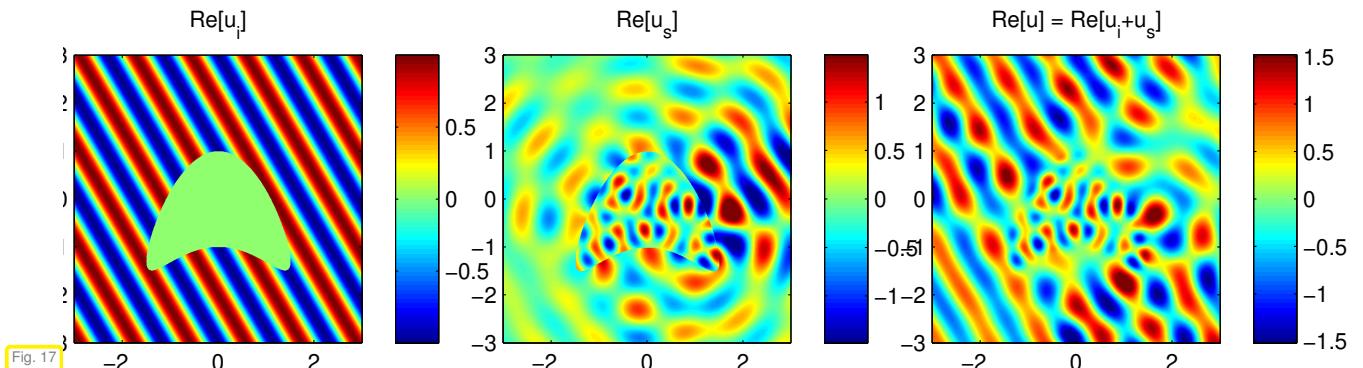


Fig. 17

Left: incident wave; middle: scattered field, right: total pressure \mathbf{p} (real parts). Computation with method of particular solutions, using the software **MPSPACK**, see [BB10].

Review question(s) 0.4.10.3.

(Q0.4.10.3.A) The following is a PDE for a vector field $\mathbf{u} : \Omega \rightarrow \mathbb{R}^2$:

$$\mathbf{grad} \mathbf{div} \mathbf{u} = [f_1, f_2, f_3]^T , \quad f_i : \Omega \rightarrow \mathbb{R} . \quad (0.4.10.4)$$

Write this PDE in detail for the components of \mathbf{u} .

(Q0.4.10.3.B) Compute the Hessian according to (0.3.2.18) for the function $\mathbf{u}(x_1, x_2) = \exp(x_1^2 + x_2^2)$.

(Q0.4.10.3.C) Compute the rotation $\mathbf{curl} \mathbf{u}$ and the divergence $\mathbf{div} \mathbf{u}$ for the vector field $\mathbf{u}(\mathbf{x}) = -\frac{\mathbf{x}}{\|\mathbf{x}\|^2}$, $\mathbf{x} \in \mathbb{R}^3 \setminus \{\mathbf{0}\}$.

Hint: Write \mathbf{u} in components and compute partial derivatives.

(Q0.4.10.3.D) Compute the Laplacian of the function $\mathbf{u}(x_1, x_2) = \sin(x_1) \cos(x_2)$. What do you observe?

(Q0.4.10.3.E) If $f \in C^1(\mathbb{R})$, what is the gradient of $\mathbf{x} \mapsto f(\|\mathbf{x}\|)$.

△

Bibliography

- [AP05] Y. Achdou and O. Pironneau. *Computational methods for option pricing*. Frontiers in Applied Mathematics. Philadelphia: SIAM, 2005 (cit. on p. 48).
- [AV10] A. Alonso-Rodriguez and A. Valli. *Eddy Current Approximation of Maxwell Equations*. Vol. 4. Modelling, Simulation & Applications. Milan: Springer, 2010 (cit. on p. 45).
- [BB10] A.H. Barnett and T. Betcke. “An exponentially convergent nonpolynomial finite element method for time-harmonic scattering from polygons”. In: *SIAM J. Sci. Comp.* 32.3 (2010), pp. 1417–1441 (cit. on p. 51).
- [Cor03] P. Corboz. “Mixed implicit timestepping for micromagnetism”. Semesterarbeit RW/CSE. Zürich, Switzerland: SAM, ETH Zürich, 2003 (cit. on p. 47).
- [GH05] M. Griebel and J. Hamaekers. *Saparse grids for the Schrödinger equation*. Preprint 0504. Bonn, Germany: INS, University of Bonn, 2005 (cit. on p. 49).
- [GHP17] P. Grohs, R. Hiptmair, and S. Pintarelli. “Tensor-Product Discretization for the Spatially Inhomogeneous and Transient Boltzmann Equation in Two Dimensions”. In: *SMAI J. Comp. Math.* 3 (2017), pp. 219–248. DOI: [10.5802/smai-jcm.26](https://doi.org/10.5802/smai-jcm.26) (cit. on p. 51).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 28, 30, 32, 40, 44).
- [HO04] R. Hiptmair and J. Ostrowski. “Coupled Boundary Element Scheme for Eddy Current Computation”. In: *J. Engr. Math.* 51.3 (2004), pp. 231–250 (cit. on p. 45).
- [Ken+00] R. Kenzler, F. Eurich, P. Maass, B. Rinn, J. Schropp, E. Bohl, and W. Dietrich. *Phase separation in confined geometries: Solving the Cahn-Hilliard equation with generic boundary conditions*. Report. Fachbereich für Mathematik, Universität Konstanz, 2000 (cit. on p. 47).
- [Pro01] A. Prohl. *Computational micromagnetism*. Advances in Numerical Mathematics. Stuttgart: B.G. Teubner, 2001 (cit. on p. 46).
- [SW03] Evelyn Sander and Thomas Wanner. “Pattern formation in a nonlinear model for animal coats”. In: *J. Differential Equations* 191.1 (2003), pp. 143–174. DOI: [10.1016/S0022-0396\(02\)00156-0](https://doi.org/10.1016/S0022-0396(02)00156-0) (cit. on p. 48).
- [Str09] M. Struwe. *Analysis für Informatiker*. Lecture notes, ETH Zürich. 2009 (cit. on p. 37).

Chapter 1

Second-Order Scalar Elliptic Boundary Value Problems

1.1 Preface

Supplement 1.1.0.1 (“Boundary value problem”) The term “boundary value problems” occurring in the title of this chapter alludes to the classical (“strong”) form of writing down most PDE-based mathematical models for stationary, that is, time-independent phenomena. We have seen this strong form in the examples of Section 0.4. In this classical form the differential equation posed on a so-called spatial domain $\Omega \subset \mathbb{R}^d$ is supplemented by equations that the restriction of the solution to the boundary $\partial\Omega$ of Ω has to satisfy:

Boundary value problem (BVP)

Given a partial differential operator \mathcal{L} , a **domain** $\Omega \subset \mathbb{R}^d$, a boundary differential operator \mathcal{B} , **boundary data** g , and a **source term** f , seek a function $u : \Omega \mapsto \mathbb{R}^n$ such that

$$\begin{aligned}\mathcal{L}(u) &= f \quad \text{in } \Omega, \\ \mathcal{B}(u) &= g \quad \text{on part of (or all) boundary } \partial\Omega.\end{aligned}\tag{1.1.0.2}$$

Eventually, we will also state the mathematical models in the form (1.1.0.2), but this will not be our starting point.

Terminology: A boundary value problem is called **scalar** $\Leftrightarrow n = 1$.
(In this case the unknown is a real valued function)

Supplement 1.1.0.3 (“Elliptic”) The attribute “elliptic” in the title of the chapter stems from a traditional classification: Mathematical theory of PDEs [PIR06] distinguishes three main classes of boundary value problems (BVPs) for partial differential equations (PDE):

- **Elliptic BVPs** (\Rightarrow “equilibrium problems”, solutions are minimizers of an “energy”).
- **Parabolic initial boundary value problems** (IBVPs) (\Rightarrow evolution towards equilibrium, see Section 9.2)
- **Hyperbolic IBVPs**, among them wave propagation problems and conservation laws (\Rightarrow transport/propagation, see Chapter 11)

The rigorous mathematical definition of “elliptic” is complicated and often fails to reveal fundamental properties of, e.g., solutions that are intuitively clear against the backdrop of the physics modelled by a certain

PDE. So we skip any further discussion of “elliptic” and focus on concrete models.

§1.1.0.4 (Outline) In the spirit of this course the first section of this chapter presents physical models whose configuration space is an infinite-dimensional function space and for which the relevant solution state fulfills an **equilibrium condition**, which gives rise to linear **variational equations**.

Subsequently, Section 1.3 ventures into the realm of **Sobolev spaces**, which provide the framework for rigorous mathematical investigation of variational equations. However, we will approach Sobolev spaces as “spaces of physically meaningful solutions” or “spaces of solutions with finite energy”. From this perspective dealing with Sobolev spaces will be reduced to dealing with their norms.

In Section 1.6, we change tack and consider a physical phenomenon (heat conduction) where modelling naturally leads to partial differential equations. On this occasion, we embark on a general discussion of **boundary conditions** in Section 1.7. Then the fundamental class of second-order elliptic boundary value problems is introduced.

In Section 1.7 in the context of stationary heat conduction we introduce the whole range of standard boundary conditions for 2nd-order elliptic boundary value problems. The discussion of various **variational formulations** will be resumed in Section 1.9.

Contents

1.1	Preface	54
1.2	Equilibrium Models: Examples	56
1.2.1	Elastic Membranes	56
1.2.2	Electrostatic Fields	65
1.2.3	Quadratic Minimization Problems	68
1.3	Sobolev spaces	80
1.3.1	Function Spaces for Energy Minimization	81
1.3.2	The Function Space $L^2(\Omega)$	82
1.3.3	Supplement: Quadratic Minimization Problems on Hilbert Spaces	84
1.3.4	The Sobolev Space $H^1(\Omega)$	87
1.4	Linear Variational Problems	96
1.4.1	Quadratic Variational Calculus	97
1.4.2	Linear Second-Order Elliptic Variational Problems	99
1.4.3	Stability of Second-Order Elliptic Linear Variational Problems	100
1.5	Equilibrium Models: Boundary Value Problems	103
1.5.1	Two-Point Boundary Value Problems	103
1.5.2	Integration by parts in higher dimensions	106
1.5.3	Linear Scalar Second-order Elliptic Partial Differential Equations	109
1.6	Diffusion Models (Stationary Heat Conduction)	115
1.7	Boundary Conditions	118
1.8	Second-Order Elliptic Variational Problems	120
1.9	Essential and Natural Boundary Conditions	129



Supplementary literature. An excellent *mathematical* introduction to partial differential equa-

tions is Evans’ book [Eva98]. Chapter 2 gives a very good idea about fundamental properties of various simple PDEs. Chapters 6 and 7 fit the scope of this chapter, but go way beyond it in terms of mathematical depth.

1.2 Equilibrium Models: Examples

In this section we examine mathematical models for two different physical systems, one from mechanics the other from electromagnetics. Both are stationary, that is, they do not change with time, and both are governed by a minimal-energy principle. These characteristics are central for the notion of “elliptic” as adopted in this course, see also Suppl. 1.1.0.3.

1.2.1 Elastic Membranes



Video tutorial for Section 1.2.1: Elastic Membranes: (44 minutes) [Download link](#), tablet notes

We consider a simple mechanical system in a “one-dimensional” and “two-dimensional” version. In one dimension it boils down to a pinned rubber band/elastic string deformed by vertical loading, that is, under the influence of a force field acting in a particular transversal direction. Think of gravity. The force will effect a deformation of the rubber band.

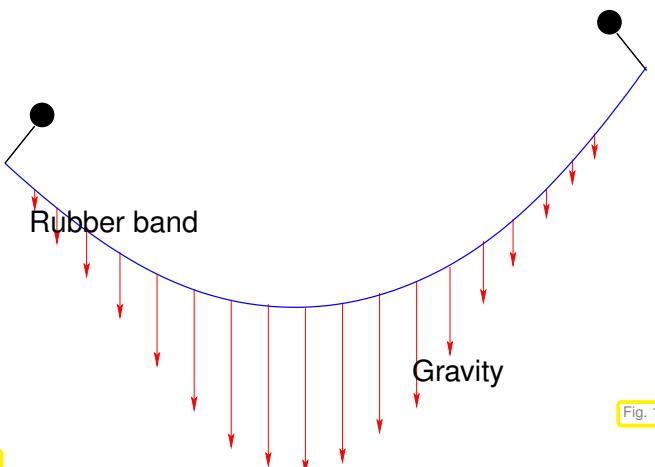


Fig. 18



Fig. 19

An example for the two-dimensional case is the taut skin of a drum clamped in a rigid frame. ▷

Again, if a vertical force is exerted on the drum membrane, it will change its shape.



Fig. 20

Our eventual goal is the **computation of the shape** of the rubber band or membrane, assuming that information about the forces, relevant properties of the materials, and the geometric situation is available. It goes without saying, that a suitable mathematical model will be instrumental.

1.2.1.1 Configuration Spaces

The first decision to be made when developing a mathematical model is what objects to use to describe state of the system under consideration.

Notion 1.2.1.1. Configuration space

The **configuration space** of a mathematical model is a set, each of whose elements completely describe all relevant aspects of a state of the modeled physical system.

EXAMPLE 1.2.1.2 (Low-dimensional configuration spaces) Many macroscopic “lumped” models merely use a few real numbers to describe the state of the system:

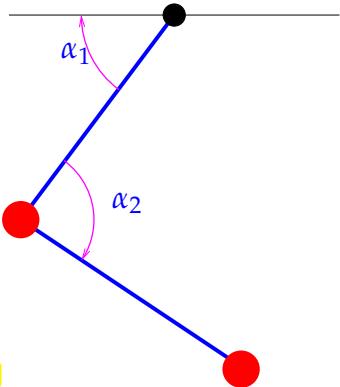


Fig. 21

Double pendulum: state described by two angles, configuration space $[0, 2\pi] \times [0, 2\pi]$.

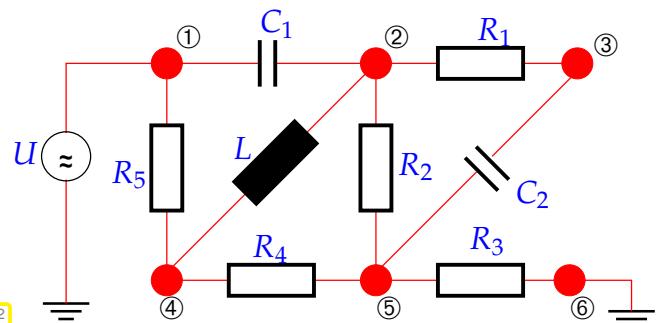


Fig. 22

Electrical circuit [Hip19, ??]: state described by branch currents and nodal potentials, configuration space \mathbb{R}^{n+m} .

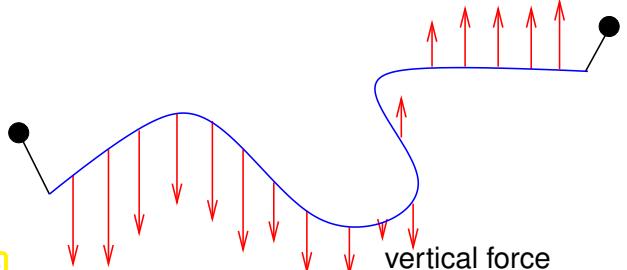
§1.2.1.3 (Configuration space for elastic string) Remember that we have assumed “vertical” forcing, that is, all forces acting on the elastic string are parallel. In this case the following assumption is very natural:

Assumption 1.2.1.4. Warp-free/loop-free shape of string

Each force line intersects the elastic string at most once.

Impossible shape: warped string

Fig. 23



This makes it possible to choose a Cartesian **coordinate system** with its x_2 -axis parallel to the force direction.

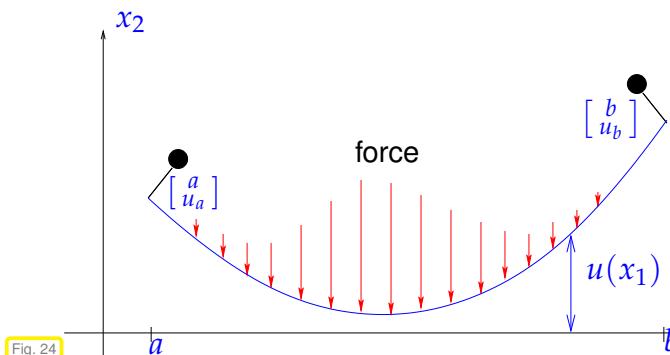


Fig. 24

Then the shape of the elastic string can be described by the **graph** of a real-valued function

$$u : [a, b] \rightarrow \mathbb{R},$$

over the **domain** $[a, b]$, $a < b$, which is a finite interval in 1D:

$$\text{string} = \left\{ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2 : x_2 = u(x_1), a \leq x_1 \leq b \right\}.$$

The value $u(x_1)$ is the **displacement** of the string from the abscissa. Hence, it must have the units of length: $[u] = 1\text{m}$.

Further evident properties of u :

- ◆ Of course, the function u must be **continuous**, because the string must not have gaps.

- ◆ The positions of the pins fix $u(a)$ and $u(b)$ and give **boundary conditions** (BDC)

$$u(a) = u_a \quad , \quad u(b) = u_b \quad \text{for given } u_a, u_b \in \mathbb{R} . \quad (1.2.1.5)$$

Basic configuration space for elastic string model

The configuration space for the elastic string model under vertical loading is the infinite-dimensional **affine function space**

$$\widehat{V}_S := \{u : [a, b] \rightarrow \mathbb{R} \cdot 1\text{m}, u \text{ continuous, } u(a) = u_a, u(b) = u_b\} . \quad (1.2.1.7)$$

Recall Def. 0.3.1.1: In the concrete case of \widehat{V}_S the “hold-all” space V is $C^0([a, b])$, the subspace V_0 is $C_0^0([a, b])$ and \widehat{v} is any function $\in V$ satisfying the pinning conditions (1.2.1.5). \square

§1.2.1.8 (Configuration space for membrane under vertical loading) Remember that all loading forces are parallel. We assume that each force line intersects the frame on which the membrane is mounted at most once. Then the membrane will not attain a warped shape, folding back over itself.

Assumption 1.2.1.9. Non-warped shape of 2D membrane

Each force line intersects the membrane at most once.

Shape of membrane
 \Updownarrow
 Graph of $u : \Omega \mapsto \mathbb{R}$

“membrane” on spatial domain $\Omega =]0, 1[^2$ \triangleright
 $(-\cdots- \hat{=} \text{ rigid frame with known geometry})$

Physical units: $[u] = 1\text{m}$

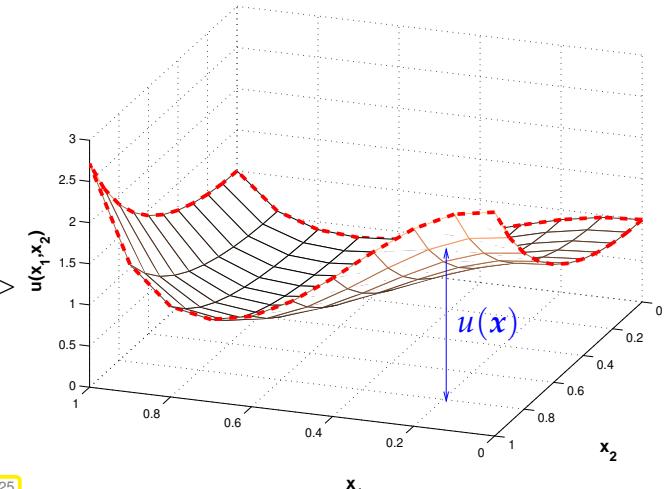


Fig. 25

☞ Notation: $x, p, y, \dots \hat{=} \text{ small vectors, coordinate vectors of points} (\rightarrow \text{Section 0.3.1.1})$

The function value $u(\mathbf{x})$ gives the displacement of the membrane over the point \mathbf{x} in the x_1/x_2 -plane. Thus, again, $u(\mathbf{x})$ must have units of length: $[u(\mathbf{x})] = 1\text{m}$.

Also the shape of the frame can be described by a function g defined on the boundary of the domain,

$$\text{frame} = \left\{ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^3 : \mathbf{x} := \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \partial\Omega, x_3 = g(\mathbf{x}) \right\} .$$

☞ Notation: $\partial\Omega \hat{=} \text{ boundary of domain } \Omega \subset \mathbb{R}^d$, e.g., in 1D $\partial[a, b] = \{a, b\}$

As in § 1.2.1.3 natural properties of u and g are:

- ◆ Both $u : \overline{\Omega} \rightarrow \mathbb{R}$ and $g : \partial\Omega \rightarrow \mathbb{R}$ are **continuous**, since neither the membrane may be ripped nor the frame be broken.
- ◆ As the membrane is firmly attached to the frame, u and g agree on $\partial\Omega$, which gives the **boundary conditions**

$$u(\mathbf{x}) = g(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \partial\Omega \quad \Leftrightarrow \quad u|_{\partial\Omega} = g \quad \text{on } \partial\Omega . \quad (1.2.1.10)$$

Configuration space for membrane model

Under vertical loading the configuration space for membrane shapes is the infinite-dimensional **affine function space**

$$\widehat{V}_M = \{u : \overline{\Omega} \rightarrow \mathbb{R} \text{ · 1m continuous, } u|_{\partial\Omega} = g\} \quad (1.2.1.12)$$

Connecting to Def. 0.3.1.1 of an affine (sub-)space, we have as “hold-all” space $V = C^0(\overline{\Omega})$, as subspace $V_0 = C_0^0(\overline{\Omega})$, and $\widehat{v} \in V$ has to satisfy (1.2.1.10).

Remark 1.2.1.13 (Continuity up to the boundary) The reader may have noticed that we put a bar on top of Ω in (1.2.1.12). It indicates that we consider u on the **closure** of the domain: $\overline{\Omega} := \Omega \cup \partial\Omega$.

To understand, why this is important, observe that $x \rightarrow \frac{1}{x}$ is a continuous function on $]0, 1[$, but not on $[0, 1]$, and that $\overline{]0, 1[} = [0, 1]$. Defining a continuous function on the closure of a domain implies continuity up to the boundary and rules out blow-up $|u(x)| \rightarrow \infty$ as $x \rightarrow \partial\Omega$.

§1.2.1.14 (Spatial domains) As explained in § 1.2.1.8 the configuration space for the membrane is a space of functions defined on a spatial domain Ω . In one dimension this was a connected interval $[a, b]$ and there is not much more to say about it, but in higher dimensions, the boundaries of domains can have special properties, which may affect the well-posedness of boundary value problems and features of their solutions.

For the remainder of this course we make the following general assumptions on spatial domains $\Omega \subset \mathbb{R}^d$:  $d = 1, 2, 3 \triangleq$ “dimension” of domain

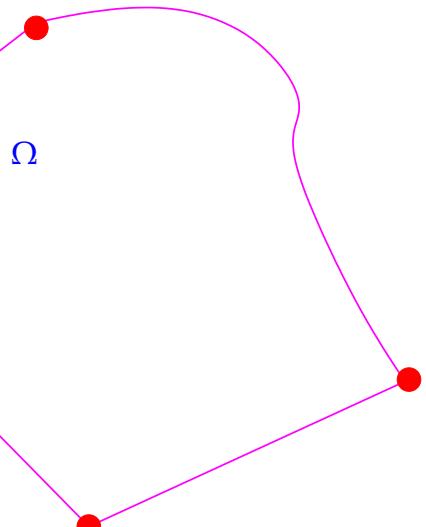
- ◆ Ω is **bounded**

$$\text{diam}(\Omega) := \sup\{\|x - y\| : x, y \in \Omega\} < \infty,$$

- ◆ Ω is **connected**: any two points in Ω can be connected by a continuous curve $\subset \Omega$,
- ◆ Ω has piecewise smooth boundary $\partial\Omega$. 

(“curvilinear polygon/polyhedron”)

For $d = 2$ we can distinguish corners (•) and edges (—) of the boundary $\partial\Omega$.



We point out that this class of domains is what can usually be represented in computer codes.

1.2.1.2 Equilibrium Conditions

Of course, for prescribed boundary conditions and given loading elastic strings and membranes will attain a unique particular shape corresponding to one element of the configuration space. A key aspect of the mathematical model must be to specify a selection criterion for that element.

§1.2.1.15 (Force density) We need a way to incorporate the loading force into the mathematical model. We do by another scalar-valued function $f : \Omega \rightarrow \mathbb{R}$, where $\Omega =]a, b[$ in the case of the 1D elastic string model.

The function f is a (vertical) **force density**, and

- in 1D has units of force per length: $[f(x)] = \frac{\text{N}}{\text{m}}$,

- in 2D has units of force per area: $[f(x)] = \frac{N}{m^2}$.

For instance in 2D, if $D \subset \Omega$ denotes a sub-domain, then $\int_D f(x) dx$ gives the total force acting on the part of the membrane over D .

Note that the force density f need not be continuous but only integrable. Hence, piecewise continuity is sufficient, $f \in C_{\text{pw}}^0(\bar{\Omega})$, see Def. 0.3.2.22. \square

Nature is economical and prefers “optimal” states, which gives us the desired selection criterion:

Equilibrium condition for stationary mechanical systems

The elastic string/membrane attain that shape that achieves a minimal potential energy.

§1.2.1.17 (Potential energies of elastic string/membrane) The above selection criterion will only be useful, if we find concrete formulas for the potential energies. We postpone their derivation to Chapter 5.

1D: For an elastic string under vertical loading by $f \in C_{\text{pw}}^0([a, b])$ whose shape is given by $u \in \widehat{V}_S$ (\rightarrow (1.2.1.7)) the total potential energy is

$$J_S(u) := \int_a^b \frac{1}{2} \sigma(x) \left| \frac{du}{dx}(x) \right|^2 - f(x) u(x) dx . \quad (1.2.1.18)$$



Here, the **stiffness** $\sigma = \sigma(x)$ potentially dependent on the position x , $[\sigma(x)] = 1\text{N}$. This ensures that J has units of energy: $[J(u)] = 1\text{J}$.

2D: For a clamped membrane loaded with a force density f and with displacement $u \in \widehat{V}_M$ (\rightarrow (1.2.1.12)) we find for the total potential energy

Here, the **stiffness** coefficient $\sigma = \sigma(x)$ has units $[\sigma(x)] = \frac{N}{m}$, which leads to $[J_M(u)] = 1J$.

To see the close relationship between ((Q1.2.1.30.F)) and (1.2.1.19), note that by the definition of the gradient

$$\sigma(x) \|\mathbf{grad} u(x)\|^2 = \sigma(x_1, x_2) \left| \frac{\partial u}{\partial x_1}(x_1, x_2) \right|^2 + \sigma(x_1, x_2) \left| \frac{\partial u}{\partial x_2}(x_1, x_2) \right|^2, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

Physics teaches us that the stiffness should be uniformly positive (UP):

$$\exists \sigma_0 > 0: \quad \sigma(x) \geq \sigma_0 \quad \forall x \in \Omega. \quad (1.2.1.20)$$

The stiffness is an example for a **macroscopic** material coefficient.

Note that apart from (1.2.1.20) the main property we expect from σ is that it is integrable. Therefore, the stiffness may only be piecewise continuous: $\sigma \in C_{\text{pw}}^0(\bar{\Omega})$ is enough and we may read (1.2.1.20) as holding “almost everywhere”

Supplement 1.2.1.21 (The gradient, see also § 0.3.2.19) A central concept in calculus: recall the definition of the **gradient** of a function $F : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$, $F(\mathbf{x}) = F(x_1, \dots, x_d)$, see [Str09, Kap. 7], [Hip19, ??]:

$$\mathbf{grad} F(\mathbf{x}) := \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \vdots \\ \frac{\partial F}{\partial x_d} \end{bmatrix} \in \mathbb{R}^d, \quad \mathbf{x} \in \Omega.$$

Note: the gradient at \mathbf{x} is a column vector of first *partial derivatives*,
read $\mathbf{grad} F(\mathbf{x})$ as $(\mathbf{grad} F)(\mathbf{x})$; $\mathbf{grad} F$ is a vector-valued function $\Omega \mapsto \mathbb{R}^d$.

Also in use (but not in this course) is the “ ∇ -notation”: $\nabla F(\mathbf{x}) := \mathbf{grad} F(\mathbf{x})$.

Obviously, as a straightforward consequence of the mean value theorem, a vanishing gradient means that the function has be constant:

$$F \in C^1(\Omega) \quad \text{and} \quad \mathbf{grad} F(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \Omega \quad \Rightarrow \quad \exists c \in \mathbb{R}: \quad F(\mathbf{x}) = c \quad \forall \mathbf{x} \in \Omega. \quad (1.2.1.22)$$

↓

Now we can quantitatively characterize the equilibrium shape based on the qualitative equilibrium condition:

Equilibrium shape

The equilibrium shape for an elastic string (S) or elastic membrane under vertical loading is

$$u = \underset{v \in \widehat{V}_*}{\operatorname{argmin}} J_*(v), \quad * = S, M, \quad (1.2.1.24)$$

with configuration spaces $\widehat{V}_S/\widehat{V}_M$ from (1.2.1.7)/(1.2.1.12) and total potential energy functionals J_S/J_M defined in ((Q1.2.1.30.F))/(1.2.1.19).

Remark 1.2.1.25 (Non-dimensional equations) By fixing reference values for the basic physical units occurring in a model (“**scaling**”), one can switch to a **non-dimensional** form of the model equations.

In the case of the elastic string model the basic units are

- unit of length **1m**,
- unit of force **1N**.

Thus, non-dimensional equations arise from fixing a reference length ℓ_0 and a reference force f_0 .

Below, following a (bad) habit of mathematicians, physical units will routinely be dropped, which tacitly assumes a priori scaling.

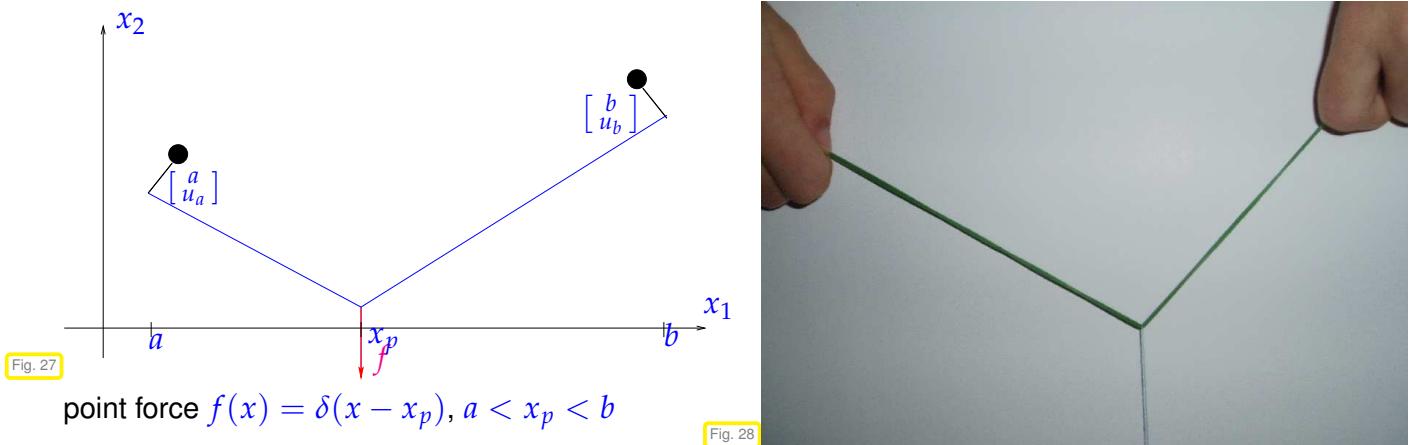
Note: Scaling is convenient, but is actually not required for numerical simulation and SI units can be kept for all quantities, owing to the fact that proper implementations of numerical methods should be *scale-invariant*. The code should always produce the same result regardless of chosen physical units (, if potential under-/overflow of floating point numbers is neglected [Hip19, ??]).

↓

1.2.1.3 Smoothness of Displacements

The alert reader will have detected some amount of sloppiness in the statement of (1.2.1.24): The displacement functions $u \in C^0(\bar{\Omega})$ have to be differentiated (partially) in order to evaluate the potential energies. Thus, these may not be defined for functions that are merely continuous. Are we forced to restrict the configuration spaces to subspaces of $C^1(\bar{\Omega})$?

EXAMPLE 1.2.1.26 (Point loading of elastic string) We pull at an elastic string at a single point, that is we use the concentrated force density $f(x) = \delta(x - x_p)$, where δ is the delta distribution and $x_p \in]a, b[$.



Point loading engenders a solution with a kink! These solutions are physically meaningful and **must not** be excluded by too stringent smoothness requirements for displacements u . □

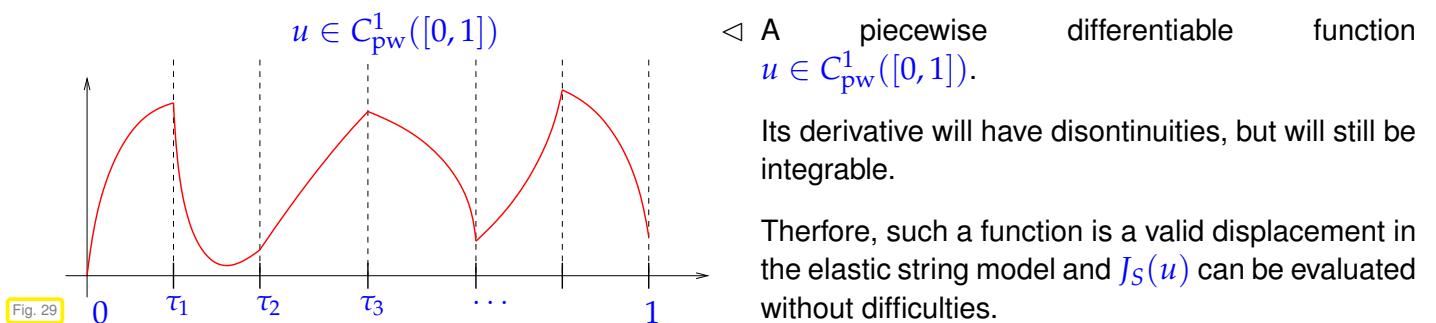
At second glance we realize that kinks of u do not pose a problem for the computation of the potential energies $J_S(u)$ and $J_M(u)$, because the (partial) derivatives of u need not be continuous, they merely have to be integrable which is compatible with a few discontinuities. We can admit displacements that fail to be differentiable at isolated points, a class of functions that we introduced as *piecewise* continuously differentiable in Def. 0.3.2.22.

Definition 0.3.2.22. Piecewise continuously differentiable functions

For a **closed** domain $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, we define

$$C_{\text{pw}}^k(\Omega) := \{v \in C^{k-1}(\Omega): v|_{\bar{\Omega}_j} \in C^k(\bar{\Omega}_j), j = 1, \dots, m\},$$

where $\{\Omega_j\}_{j=1}^m$, $m \in \mathbb{N}$, is a partition of Ω in the sense that $\Omega_k \cap \Omega_i = \emptyset$, if $i \neq k$, and $\Omega = \bar{\Omega}_1 \cup \dots \cup \bar{\Omega}_m$.



A function $u \in C_{\text{pw}}^1(\overline{\Omega})$, where Ω is the unit disk:

$$\Omega := \{x \in \mathbb{R}^2 : \|x\| < 1\}.$$

This function is an eligible displacement for a membrane mounted on a flat circular frame, because $J_M(u)$ can be computed by piecewise integration of the squared norm of the discontinuous gradient $\mathbf{grad} u$.

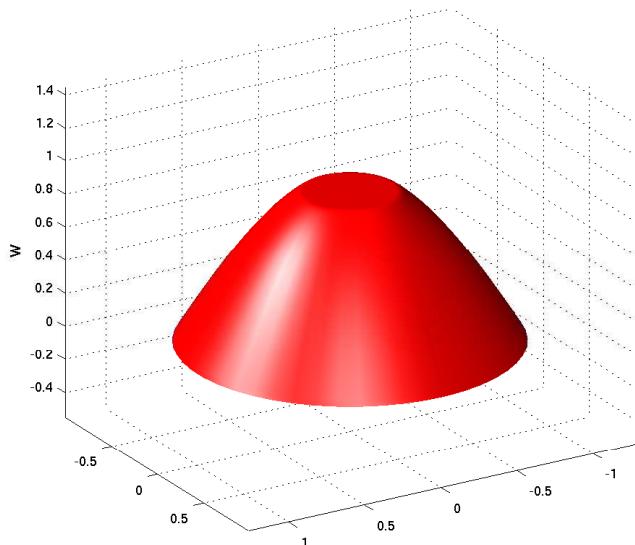


Fig. 30

These considerations yield the following upgraded definition of the configuration spaces, see § 0.3.2.21, Def. 0.3.2.22, for notations.

Energy-compatible configuration spaces

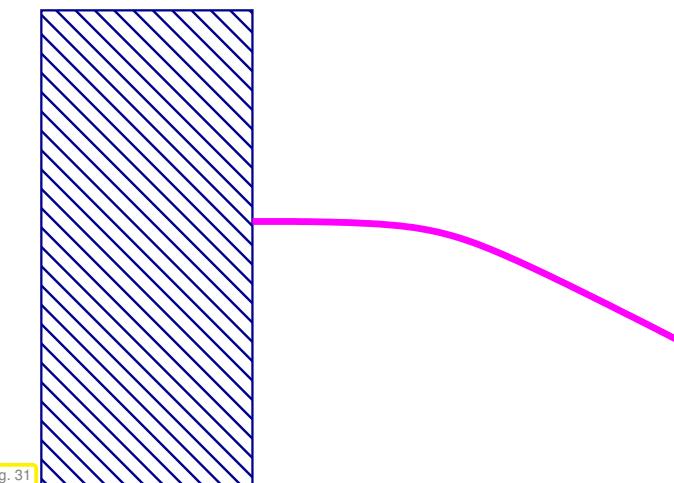
For vertical loading the following configuration spaces for displacements u take into account prescribed boundary conditions and allow the computation of potential energies

$$\text{1D (elastic string): } \widehat{V}_S := \left\{ u \in C_{\text{pw}}^1([a, b]), u(a) = u_a, u(b) = u_b \right\}, \quad (1.2.1.28)$$

$$\text{2D (membrane): } \widehat{V}_M := \left\{ u \in C_{\text{pw}}^1(\overline{\Omega}), u|_{\partial\Omega} = g \right\}. \quad (1.2.1.29)$$

Review question(s) 1.2.1.30 (Elastic membranes)

(Q1.2.1.30.A)



A flexible thin beam is mounted on a wall perpendicularly. What is a suitable configuration space for it, if the lateral extension (thickness) of the beam can be ignored?

(Q1.2.1.30.B) What is a suitable configuration space for an elastic balloon filled with pressurized gas?

(Q1.2.1.30.C) The total potential energy of an elastic membrane under small “vertical” displacement over a base plane $\Omega \subset \mathbb{R}^2$ is given by the formula

$$J_M(u) := \int_{\Omega} \frac{1}{2} \sigma(x) \|\mathbf{grad} u(x)\|^2 - f(x) u(x) \, dx. \quad (1.2.1.19)$$

What is the meaning of the quantities σ , u , and f and what are their physical units?

(Q1.2.1.30.D) Describe the physical principle that allows us to determine the equilibrium shape of a static mechanical system. Is that equilibrium shape always unique?

(Q1.2.1.30.E) Compute the **elastic energy** of an elastic string whose shape is given by the graph of a displacement function $u : [0\text{m}, 1\text{m}] \rightarrow \mathbb{R} \cdot 1\text{m}$ for

1. $u(x) := x(1\text{m} - x)\text{m}^{-1}$,
2. $u(x) := 1\text{m} \cdot \sin(x \cdot 1\text{m}^{-1})$,
3. $u(x) := \begin{cases} x & \text{for } 0\text{m} \leq x \leq 0.4\text{m} \\ \frac{2}{3}\text{m} - \frac{2}{3}x & \text{for } 0.4\text{m} < x \leq 1\text{m} \end{cases}$.

The stiffness is $\sigma(x) = 0.5\text{N}$. Note that the pure elastic energy does not depend on an external force field.

(Q1.2.1.30.F) Assume a non-dimensional graph model for an elastic string, whose shape is described by a function $u : [0, 1] \rightarrow \mathbb{R}$ and whose stiffness is constant $\sigma \equiv 1$. We know that $u(x) = \alpha x(1 - x)$ with a parameter $\alpha \in \mathbb{R}$. For which parameter value does the total potential energy

$$J_S(u) := \int_0^1 \frac{1}{2}\sigma(x) \left| \frac{du}{dx}(x) \right|^2 - f(x)u(x) dx .$$

attain a minimum, if $f \equiv 1$.

(Q1.2.1.30.G) In a non-dimensional model, the shape of a membrane over the base plane $\Omega \subset \mathbb{R}^2$ is described by the displacement function $u : \Omega \rightarrow \mathbb{R}$. We consider the two situations

1. $\Omega = [0, 1]^2$, $u(x) = \sin(\pi x_1) \sin(\pi x_2)$,
2. $\Omega = \{x \in \mathbb{R}^2 : \|x\| \leq 1\}$, $u(x) = 1 - \|x\|^2$.

The stiffness of the membrane is constant $\sigma \equiv 1$. Compute the elastic energies of the membranes.

(Q1.2.1.30.H) We consider a non-dimensional model for an elastic string with uniform stiffness $\sigma \equiv 1$ and displacement $u : [0, 1] \rightarrow \mathbb{R}$. Find a displacement function $u \in C^0([0, 1])$ for which the elastic energy

$$J_S(u) := \int_0^1 \frac{1}{2}\sigma(x) \left| \frac{du}{dx}(x) \right|^2 dx$$

of the string becomes ∞ .

Hint. Look for a suitable “ \sqrt -shaped” function.

△

1.2.2 Electrostatic Fields



Video tutorial for Section 1.2.2: Electrostatic Fields: (13 minutes) [Download link](#), [tablet notes](#)

In this section we see another important example of a mathematical model

- ◆ whose configuration space is a space of functions on a spatial domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$,
- ◆ and governed by a minimal potential energy principle as equilibrium condition.

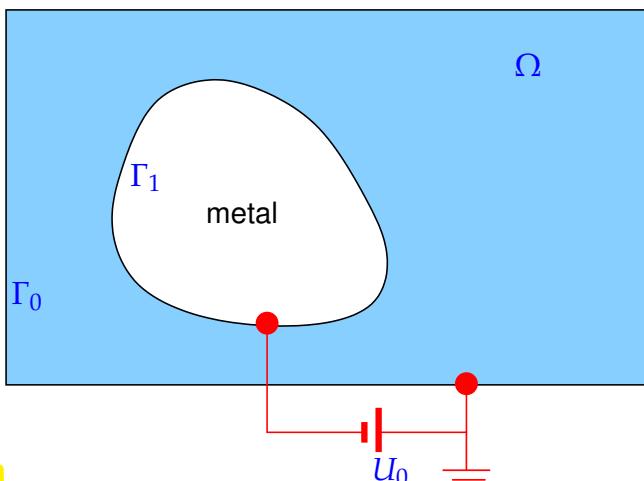
Concretely, we study field models arising from electrostatics.

A typical arrangement for an electrostatic setup is sketched beside

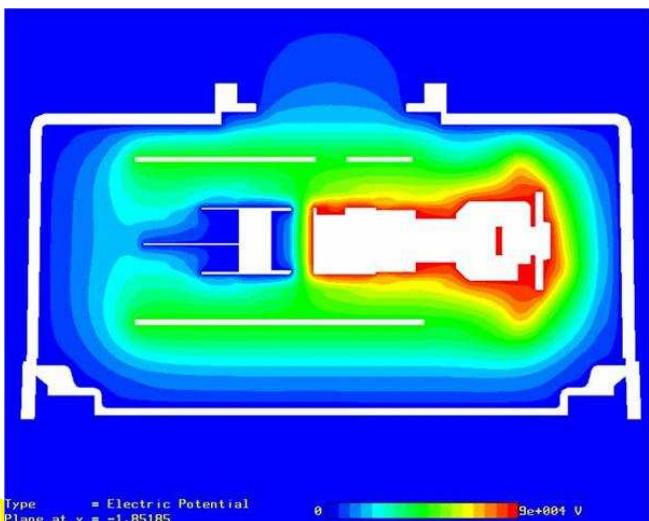
- ◆ chunk of metal in conducting box
- ◆ prescribed voltage drop metal—box

We seek the **electric field** $\mathbf{E} : \Omega \mapsto \mathbb{R}^d$ in $\Omega \subset \mathbb{R}^d$, $d = 2, 3$.

($\Omega \triangleq$ blue region \triangleright)

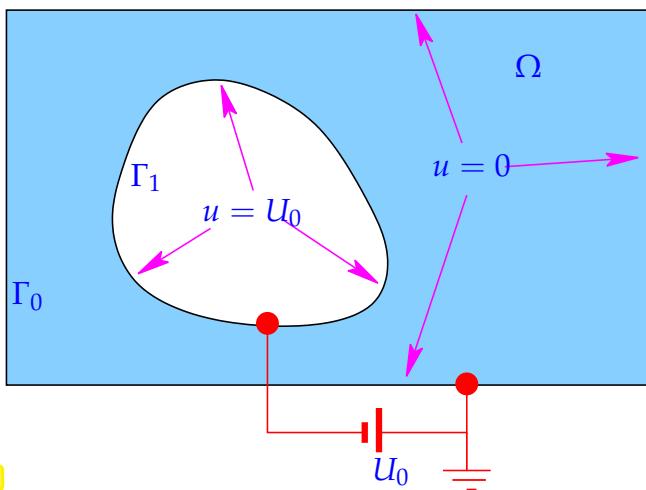


§1.2.2.1 (Scalar potential representation) In electrostatics computations do not need to tackle the electric field \mathbf{E} directly, thanks to a fundamental constraint on \mathbf{E} .



§1.2.2.3 (Boundary conditions for electrostatic potential) In order to characterize the configuration space for electrostatic field problems completely, we have to identify proper boundary conditions for the scalar potential u . To do so, we have to appeal to physics.

Recall that in electrostatics surfaces of conducting bodies are **equipotential surfaces**



Thus in the situation of Fig. 32 the electric potential must satisfy the following **boundary conditions**

$$\begin{aligned} u &= 0 && \text{on } \Gamma_0, \\ u &= U_0 && \text{on } \Gamma_1. \end{aligned} \quad (1.2.2.4)$$

which implies the **preliminary** configuration space

$$\widehat{\mathcal{V}}_E = \left\{ u \in C^0(\bar{\Omega}), u \text{ satisfies (1.2.2.4)} \right\}.$$

This choice may be problematic: What is the meaning of $\mathbf{grad} u$. We are going to address this shortly.

In the remainder of this section we write $u = U$ to designate the boundary conditions (1.2.2.4).

§1.2.2.5 (Electrostatic field energy) Electromagnetic field theory gives us an expression for the energy content of the electric field:

Electromagnetic field energy: (electrostatic setting with scalar potential u)

$$J_E(u) = \frac{1}{2} \int_{\Omega} (\epsilon(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} u(x) dx , \quad (1.2.2.6)$$

where $\epsilon : \Omega \mapsto \mathbb{R}^{3,3}$ is the dielectric tensor, $\epsilon(x)$ is symmetric, with units $[\epsilon] = \frac{\text{As}}{\text{Vm}}$.

Note that in terms of partial derivatives and matrix components $\epsilon(x) = [\epsilon_{ij}]_{i,j=1}^3$ we have

$$(\epsilon(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} u(x) = \sum_{i=1}^3 \sum_{j=1}^3 \epsilon_{ij}(x) \frac{\partial u}{\partial x_i}(x) \frac{\partial u}{\partial x_j}(x) .$$

§1.2.2.7 (Dielectric tensor) The dielectric tensor $\epsilon = \epsilon(x)$ is a matrix-valued function describing the macroscopic electric properties of the material at x . We list a few properties of ϵ :

- The dielectric tensor need not be continuous, $\sigma \in (C_{pw}^0(\bar{\Omega}))^{3,3}$ is enough and, actually, very common when different materials abut at sharp interfaces.
- Symmetry of the dielectric tensor, a matrix valued function on the spatial domain, can always be assumed: if $\epsilon(x)$ was not symmetric, then replacing it with $\frac{1}{2}(\epsilon(x)^T + \epsilon(x))$ will yield exactly the same field energy.
- A fundamental property of the dielectric tensor (for “normal” materials) is uniform positivity:

$$\exists 0 < \epsilon^- \leq \epsilon^+ < \infty: \epsilon^- \|z\|^2 \leq (\epsilon(x)z) \cdot z \leq \epsilon^+ \|z\|^2 \quad \forall z \in \mathbb{R}^3, \forall x \in \Omega . \quad (1.2.2.8)$$

If this was not satisfied, negative field energies could result.

Terminology: (1.2.2.8) \Leftrightarrow ϵ is bounded and uniformly positive definite (UPD)

□

The positivity property of ϵ is so fundamental and shared by many other material coefficient functions that it has been given a special name, cf. Def. 0.3.1.16.

Definition 1.2.2.9. Uniformly positive (definite) tensor field

An matrix-valued function $\mathbf{A} : \Omega \mapsto \mathbb{R}^{n,n}$, $n \in \mathbb{N}$, is called uniformly positive definite, if

$$\exists \alpha^- > 0: (\mathbf{A}(x)z) \cdot z \geq \alpha^- \|z\|^2 \quad \forall z \in \mathbb{R}^n \quad (1.2.2.10)$$

for almost all $x \in \Omega$, that is, only with the exception of a set of volume zero.

□

Remark 1.2.2.11. If $\mathbf{A}(x)$ is symmetric, then we have the equivalence, cf. [Hip19, ??],

$$(1.2.2.10) \Leftrightarrow \mathbf{A}(x) \text{ s.p.d. } (\rightarrow [\text{Hip19, ??}]) \text{ and } \lambda_{\min}(\mathbf{A}(x)) \geq \alpha^- ,$$

where λ_{\min} stands for the smallest eigenvalue of a matrix.

□

Now it has become much clearer what smoothness we have to impose on scalar potentials. Like in Section 1.2.1 we have to shrink the configuration space and restrict it to scalar potentials u for which the electrostatic field energy $J_E(u)$ can be computed. Again this leads to a space of continuous and piecewise continuously differentiable functions.

Configuration space for electrostatic field problems

As configuration space for scalar potentials we use the affine space

$$\widehat{V}_E := \left\{ u \in C_{\text{pw}}^1(\overline{\Omega}), u \text{ satisfies (1.2.2.4)} \right\}. \quad (1.2.2.13)$$

Using the notations of Def. 0.3.1.1, in this case we have $V = C_{\text{pw}}^1(\overline{\Omega})$, $V_0 = C_{\text{pw},0}^1(\overline{\Omega})$.

§1.2.2.14 (Electrostatic equilibrium condition) Electromagnetic field theory in the form of Maxwell's equation provides a criterion for selecting a unique electric scalar potential from the configuration space \widehat{V}_E . Again, energy minimization is the fundamental principle.

Equilibrium condition for electrostatics

Electrostatic field problem solved by potential $u_* \in \widehat{V}_E$, for which the field energy $J_E(u)$ becomes minimal:

$$u_* = \underset{u \in \widehat{V}_E}{\operatorname{argmin}} J_E(u). \quad (1.2.2.16)$$

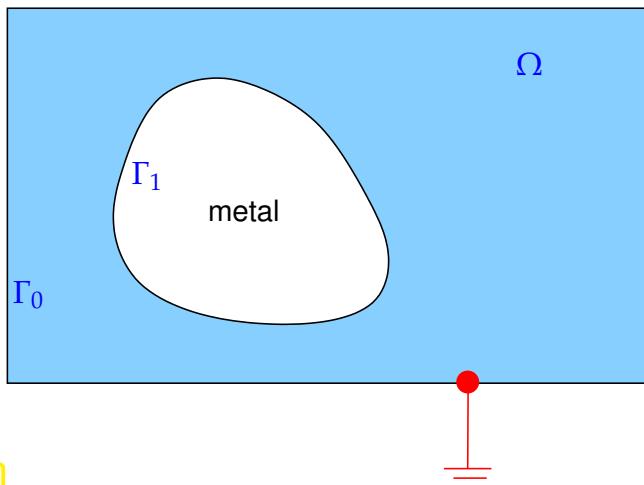
Review question(s) 1.2.2.17 (Electrostatic fields)

(Q1.2.2.17.A) Explain, why surfaces of conducting bodies are equipotential surfaces in electrostatics.

(Q1.2.2.17.B) What are the physical units for the electric scalar potential u , the electric field \mathbf{E} , the dielectric tensor \mathbf{e} , and the electromagnetic field energy

$$J_E(u) = \frac{1}{2} \int_{\Omega} (\mathbf{e}(x) \operatorname{grad} u(x)) \cdot \operatorname{grad} u(x) \, dx ? \quad (1.2.2.6)$$

(Q1.2.2.17.C) [Floating potentials] What is a suitable configuration space for describing electrostatics in the following situation:



▷ A metallic (conducting) body is located inside a (grounded) metal box, but it is not connected to any wire, which means that its scalar potential is not known a priori.
This situation is often referred to as **floating potential**.

Fig. 35

(Q1.2.2.17.D) What does it mean that a 3×3 -matrix-valued function $\mathbf{M} : \Omega \rightarrow \mathbb{R}^{3,3}$, $\Omega \subset \mathbb{R}^d$, is uniformly positive definite?

(Q1.2.2.17.E) The matrix-valued function $\mathbf{x} \mapsto \alpha(\mathbf{x})\mathbf{I}_3$, $\mathbf{x} \in \mathbb{R}^3$, \mathbf{I}_3 the 3×3 identity matrix, is uniformly positive definite. What does this imply for the function $\alpha : \mathbb{R}^3 \rightarrow \mathbb{R}$?

(Q1.2.2.17.F) Compute the electric field belonging to the electric scalar potential $u : \Omega \rightarrow \mathbb{R}$, $u(\mathbf{x}) = 1 - \|\mathbf{x}\|^2$, $\Omega := \{\mathbf{x} \in \mathbb{R}^3 : \|\mathbf{x}\| \leq 1\}$.

(Q1.2.2.17.G) Given a domain $\Omega \subset \mathbb{R}^2$ a tensor field $\mathbf{A} : \overline{\Omega} \rightarrow \mathbb{R}^2$ is defined as

$$\mathbf{A}(\mathbf{x}) = \begin{bmatrix} 1 & \alpha(\mathbf{x}) \\ \alpha(\mathbf{x}) & 1 \end{bmatrix} \quad \text{with} \quad \alpha \in C^0(\overline{\Omega}).$$

Find a necessary and sufficient condition for α so that \mathbf{A} is uniformly positive definite.

△

1.2.3 Quadratic Minimization Problems



Video tutorial for Section 1.2.3: Quadratic Minimization Problems: (48 minutes)
[Download link](#), [tablet notes](#)

Recall the minimization problems arising from the equilibrium conditions:

Section 1.2.1.2 [1D, string]

$$u_* = \operatorname{argmin}_{\substack{u \in C_{pw}^1([a,b]) \\ u(a)=u_a \ u(b)=u_b}} \underbrace{\int_a^b \frac{1}{2}\sigma(x) \left| \frac{du}{dx}(x) \right|^2 - f(x)u(x) dx}_{=: J_S(u), \text{ see } ((Q1.2.1.30.F))} \quad (1.2.3.1a)$$

Section 1.2.1.2 [2D membrane]

$$u_* = \operatorname{argmin}_{\substack{u \in C_{pw}^1(\overline{\Omega}) \\ u=g \text{ on } \partial\Omega}} \underbrace{\int_{\Omega} \frac{1}{2}\sigma(\mathbf{x}) \|\mathbf{grad} u(\mathbf{x})\|^2 - f(\mathbf{x})u(\mathbf{x}) d\mathbf{x}}_{=: J_M(u), \text{ see } (1.2.1.19)}, \quad (1.2.3.1b)$$

Section 1.2.2 [2D, 3D electrostatics]

$$u_* = \operatorname{argmin}_{\substack{u \in C_{pw}^1(\overline{\Omega}) \\ u=U \text{ on } \partial\Omega}} \underbrace{\int_{\Omega} \frac{1}{2}(\epsilon(\mathbf{x}) \mathbf{grad} u(\mathbf{x})) \cdot \mathbf{grad} u(\mathbf{x}) d\mathbf{x}}_{=: J_E(u), \text{ see } (1.2.2.6)}. \quad (1.2.3.1c)$$

Obviously, these minimization problems are rather similar. In this section we will discuss their structure and preliminary results about existence and uniqueness of solutions.

1.2.3.1 Definition

From linear algebra and Section 0.3.1.2 we need the following concepts:

Definition 0.3.1.3. Linear forms

Given a vector space V over \mathbb{R} , a linear form/functional (LF) ℓ is a mapping $\ell : V \mapsto \mathbb{R}$ that satisfies

$$\ell(\alpha u + \beta v) = \alpha \ell(u) + \beta \ell(v) \quad \forall u, v \in V, \forall \alpha, \beta \in \mathbb{R}.$$

Definition 0.3.1.4. (Bi-)linear forms

Given an \mathbb{R} -vector space V , a bilinear form (BLF) a on V is a mapping $a : V \times V \mapsto \mathbb{R}$, for which

$$a(\alpha_1 v_1 + \beta_1 u_1, \alpha_2 v_2 + \beta_2 u_2) = \alpha_1 \alpha_2 a(v_1, v_2) + \alpha_1 \beta_2 a(v_1, u_2) + \beta_1 \alpha_2 a(u_1, v_2) + \beta_1 \beta_2 a(u_1, u_2)$$

for all $u_i, v_i \in V, \alpha_i, \beta_i \in \mathbb{R}, i = 1, 2$.

Bilinear forms and linear forms are key building blocks for the energy functionals J_S , J_M , and J_E from (1.2.3.1). They all match the following type of functional:

Definition 1.2.3.2. Quadratic functional

A quadratic functional on a *real vector space* V_0 is a mapping $J : V_0 \mapsto \mathbb{R}$ of the form

$$J(u) := \frac{1}{2}a(u, u) - \ell(u) + c, \quad u \in V_0, \quad (1.2.3.3)$$

where $a : V_0 \times V_0 \mapsto \mathbb{R}$ is a **symmetric** bilinear form (\rightarrow Def. 0.3.1.4), $\ell : V_0 \mapsto \mathbb{R}$ a linear form, and $c \in \mathbb{R}$.

Recall from Def. 0.3.1.15 that a bilinear form $a : V_0 \times V_0 \mapsto \mathbb{R}$ is **symmetric**, if

$$a(u, v) = a(v, u) \quad \forall u, v \in V_0. \quad (1.2.3.4)$$

EXAMPLE 1.2.3.5 (Quadratic functionals on \mathbb{R}^N) As we have already seen in Ex. 0.3.1.5, if $V_0 = \mathbb{R}^N$ (finite-dimensional case), then a quadratic functional has the general representation

$$J(\vec{\eta}) = \frac{1}{2}\vec{\eta}^\top \mathbf{A}\vec{\eta} - \vec{\beta}^\top \vec{\eta} + c, \quad \mathbf{A} = \mathbf{A}^\top \in \mathbb{R}^{N,N}, \quad \vec{\beta} \in \mathbb{R}^N, \quad c \in \mathbb{R}. \quad (1.2.3.6)$$

Reminder: quadratic functionals of this forms occur in derivation of steepest descent and conjugate gradient methods for linear systems of equations, see [Hip19, ??].

We will continue the discussion of quadratic functionals on \mathbb{R}^n in [Hip19, ??].

§1.2.3.7 (Energies are quadratic functionals) Now, let us identify the bilinear forms a and linear forms ℓ contributing to the specific quadratic energy functionals in (1.2.3.1):

- For $J_S(v) = \int_a^b \frac{1}{2}\sigma(x) \left| \frac{dv}{dx}(x) \right|^2 - f(x)v(x) dx$ from (1.2.3.1a):

$$a(w, v) = \int_a^b \sigma(x) \frac{dw}{dx}(x) \frac{dv}{dx}(x) dx, \quad \ell(v) = \int_a^b f(x)v(x) dx. \quad (1.2.3.8)$$

- For $J_M(v) = \int_{\Omega} \frac{1}{2}\sigma(x) \|\mathbf{grad} v(x)\|^2 - f(x)v(x) dx$ from (1.2.3.1b):

$$a(w, v) = \int_{\Omega} \sigma(x) \mathbf{grad} w(x) \cdot \mathbf{grad} v(x) dx, \quad \ell(v) = \int_{\Omega} f(x)v(x) dx. \quad (1.2.3.9)$$

- For $J_E(v) = \int_{\Omega} \frac{1}{2}(\epsilon(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} u(x) dx$ from (1.2.3.1c):

$$a(w, v) = \int_{\Omega} \mathbf{grad} w(x)^T \epsilon(x) \mathbf{grad} v(x) dx, \quad \ell(v) = 0. \quad (1.2.3.10)$$

Obviously, (1.2.3.1) is all about minimizing quadratic functionals. We introduce a name for this kind a minimization problems.

Definition 1.2.3.11. Quadratic minimization problem

A minimization problem

$$w_* = \underset{w \in V_0}{\operatorname{argmin}} J(w)$$

is called a **quadratic minimization problem**, if J is a *quadratic functional* on a real vector space V_0 .

§1.2.3.12 (Offset functions) Objection! Neither (1.2.3.1a), nor (1.2.3.1b) nor (1.2.3.1c) are genuine quadratic minimization problems in the sense of Def. 1.2.3.11, because they are posed over affine spaces instead of vector spaces. Just notice that the configuration spaces may not contain the zero function.

Definition 1.2.3.13. Affine space

A subset $\widehat{W} \subset V$ of a vector space V is called an **affine (sub-)space**, if

$$\widehat{W} = V_0 + w_0 := \{w \in V : w = v_0 + w_0, v_0 \in V_0\},$$

for some **subspace** $V_0 \subset V$ and some $w_0 \in V$.

By a simple trick we can recover proper quadratic minimization problems when asked to minimize a quadratic functional over an affine subspace. This so-called **offset function trick** is suggested by the very definition of an affine subspace. First, observe that for a quadratic functional $J(v) := \frac{1}{2}a(v, v) - \ell(v)$ according to Def. 1.2.3.2 and defined on a vector space V we have

$$\begin{aligned} J(u + u_0) &= \frac{1}{2}a(u + u_0, u + u_0) - \ell(u + u_0) + c \\ &= \frac{1}{2}a(u, u) + \frac{1}{2}a(u, u_0) + \frac{1}{2}a(u_0, u) + \frac{1}{2}a(u_0, u_0) - \ell(u) - \ell(u_0) + c \\ &= \frac{1}{2}a(u, u) + \underbrace{a(u, u_0) - \ell(u)}_{=: \tilde{\ell}(u)} + \underbrace{\frac{1}{2}a(u_0, u_0) - \ell(u_0) + c}_{=: \tilde{c}}, \end{aligned} \tag{1.2.3.14}$$

due to the bilinearity of a and the linearity of ℓ . Hence, the minimizer of J over the affine subspace $u_0 + V_0$, $V_0 \subset V$ a subspace of V , can be computed as follows:

$$\underset{u \in u_0 + V_0}{\operatorname{argmin}} J(u) = u_0 + \underset{v \in V_0}{\operatorname{argmin}} J(v + u_0) = u_0 + \underset{v \in V_0}{\operatorname{argmin}} \tilde{J}(v), \tag{1.2.3.15}$$

$$\text{with } \tilde{J}(v) := \frac{1}{2}a(v, v) + \tilde{\ell}(v) + \tilde{c}, \tag{1.2.3.16}$$

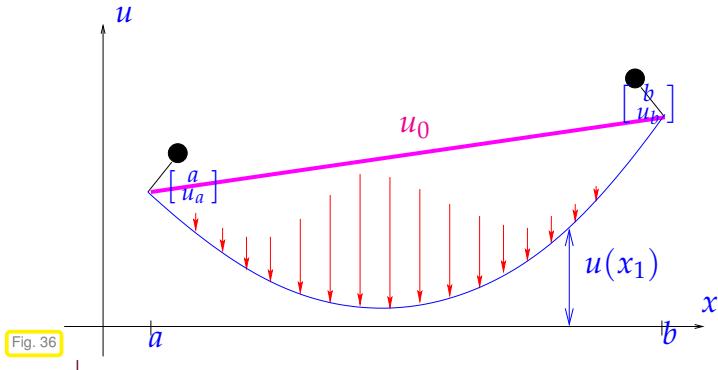
and the functional $\tilde{\ell}$, and $\tilde{c} \in \mathbb{R}$ as defined in (1.2.3.14). We have ended up with a quadratic minimization problem over a genuine vector space, here V_0 , compliant with Def. 1.2.3.11.

Thus, in the sequel we can focus on quadratic minimization problems posed on genuine vector spaces as introduced in Def. 1.2.3.11. \square

EXAMPLE 1.2.3.17 (Offset function for elastic string model) In Section 1.2.1.3 we learned that for the 1D elastic string model minimization of the potential energy J_S has to be performed over the affine space

$$\begin{aligned} \widehat{V}_S &:= \left\{ u \in C_{\text{pw}}^1([a, b]), u(a) = u_a, u(b) = u_b \right\} \\ &\quad C_{\text{pw}, 0}^1([a, b]) + u_0, \end{aligned}$$

where u_0 is **any** function in V_S .



$$\triangleleft \quad u_0 \hat{=} \text{ —}$$

Of course, u_0 should be as simple as possible.

Clearly the simplest choice is an affine linear function whose graph, a line segment, connects the pinning points:

$$u_0(x) = u_a \frac{b-x}{b-a} + u_b \frac{x-a}{b-a}. \quad (1.2.3.18)$$

Summing up, for the minimization problems considered so far the underlying vector spaces and modified linear forms $\tilde{\ell}$ read, with bilinear forms a as in (1.2.3.8)–(1.2.3.10):

- ♦ For elastic string model \leftrightarrow (1.2.3.1a): with u_0 from (1.2.3.18)

$$V_0 = C_{\text{pw},0}^1([a,b]), \quad \tilde{\ell}(v) := \int_a^b f(x)v(x) dx - a(u_0, v), \quad (1.2.3.19)$$

$$(1.2.3.20)$$

- ♦ For membrane model \leftrightarrow (1.2.3.1b): with $u_0 \in C_{\text{pw}}^1(\bar{\Omega})$ with $u_0|_{\partial\Omega} = g$:

$$V_0 = C_{\text{pw},0}^1(\bar{\Omega}), \quad \tilde{\ell}(v) = \int_{\Omega} f(x)v(x) dx - a(u_0, v), \quad (1.2.3.21)$$

$$(1.2.3.22)$$

- ♦ For the electrostatic model from Section 1.2.2: with $u_0 \in C_{\text{pw}}^1(\bar{\Omega})$ with $u_0|_{\partial\Omega} = U$:

$$V_0 = C_{\text{pw},0}^1(\bar{\Omega}), \quad \tilde{\ell}(v) = -a(u_0, v). \quad (1.2.3.23)$$

The bilinear forms a are different in each case and given by (1.2.3.8)–(1.2.3.9).

1.2.3.2 Existence and Uniqueness of Minimizers

Our models would be highly dubious, if they gave rise to quadratic minimization problems that failed to have unique solutions. We first tackle this issue in an abstract context. To begin with recall an important property of a bilinear form, which yields a *necessary* condition for the existence of a minimizer of a quadratic functional.

Definition 1.2.3.24. Positive semi-definite bilinear form \rightarrow Def. 0.3.1.16

A (symmetric) bilinear form $a : V_0 \times V_0 \mapsto \mathbb{R}$ on a real vector space V_0 is **positive semi-definite**, if

$$a(u, u) \geq 0 \quad \forall u \in V_0. \quad (1.2.3.25)$$

Necessity of (1.2.3.25) for the existence of a minimizer can be concluded as follows: In case (1.2.3.25) fails to hold there is a $u \in V_0$ for which $a(u, u) < 0$. Hence, for a quadratic functional $J : V_0 \rightarrow \mathbb{R}$ as in (1.2.3.3) we find

$$J(tu) = \frac{1}{2} t^2 \underbrace{a(u, u)}_{<0} - t \ell(u) + c, \quad t \in \mathbb{R},$$

such that $J(tu) \rightarrow -\infty$ for $t \rightarrow \infty$: J has no minimum. The next corollary summarizes this insight.

Corollary 1.2.3.26. Necessary condition for existence of minimizer

The quadratic functional $J(v) := \frac{1}{2}a(v, v) - \ell(v)$ (\rightarrow Def. 1.2.3.2) on a vector space V_0 is bounded from below, only if the bilinear form $a : V_0 \times V_0 \rightarrow \mathbb{R}$ is **positive semi-definite**.

Next, let us tackle the issue of **uniqueness** of the minimizer of the quadratic functional J from (1.2.3.3). There is a *necessary and sufficient* condition in terms of a simple property of a , see also Thm. 1.2.3.31 below.

Definition 1.2.3.27. Positive definite bilinear form \rightarrow Def. 0.3.1.16

A (symmetric) bilinear form $a : V_0 \times V_0 \mapsto \mathbb{R}$ on a real vector space V_0 is **positive definite**, if

$$u \in V_0 \setminus \{0\} \iff a(u, u) > 0.$$

EXAMPLE 1.2.3.28 (Positive definite matrices) As in Ex. 0.3.1.7, for the special case $V_0 = \mathbb{R}^N$ any matrix $\mathbf{A} \in \mathbb{R}^{N,N}$ induces a bilinear form via

$$a(\mathbf{u}, \mathbf{v}) := \mathbf{u}^T \mathbf{A} \mathbf{v} = (\mathbf{A}\mathbf{v}) \cdot \mathbf{u}, \quad \mathbf{u}, \mathbf{v} \in \mathbb{R}^N. \quad (1.2.3.29)$$

This connects the concept of a symmetric positive definite bilinear form to the more familiar concept of s.p.d. matrices (\rightarrow [Hip19, ??])

A s.p.d. (\rightarrow Def. 0.3.1.24) \Leftrightarrow a from (1.2.3.29) is symmetric, positive definite.

□

EXAMPLE 1.2.3.30 (Quadratic functionals with positive definite bilinear form in 2D) Worth remembering is the analogy between quadratic functionals with positive definite bilinear form and parabolas opening upwards:

$$\begin{aligned} J(v) &= \frac{1}{2}a(v, v) - \ell(v) \\ \uparrow &\quad \uparrow & \uparrow \\ f(x) &= \frac{1}{2}ax^2 - bx \end{aligned}$$

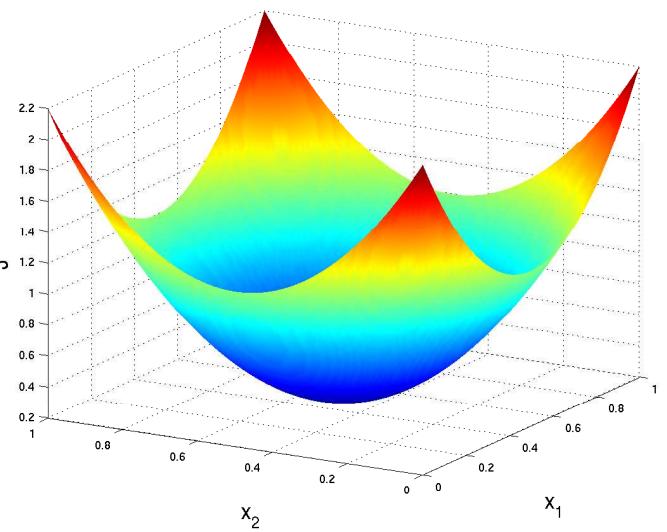
with $a > 0$!

Graphs of quadratic functionals with s.p.d. bilinear forms are “higher-dimensional” upward-open paraboloids.

graph of quadratic functional $\mathbb{R}^2 \mapsto \mathbb{R}$

Geometric intuition suggests unique global and local minimum.

Fig. 37



Theorem 1.2.3.31. Uniqueness of solutions of quadratic minimization problems

If the bilinear form $a : V_0 \times V_0 \mapsto \mathbb{R}$ is **positive definite** (\rightarrow Def. 1.2.3.27), then any solution of

$$u_* = \underset{u \in V_0}{\operatorname{argmin}} J(u) , \quad J(u) := \frac{1}{2}a(u, u) - \ell(u) + c ,$$

is unique for any linear form $\ell : V_0 \mapsto \mathbb{R}$.

Proof. (indirect) Assume that both $u_* \in V_0$ and $w_* \in V_0$, $u_* \neq w_*$ are **global minimizers** of J on V_0 .

- ① $\varphi(t) := J(tu_* + (1-t)w_*)$ has **two distinct global minima** in $t = 0, 1$.
- ② However $\varphi(t) = \frac{1}{2}t^2 \underbrace{a(u_* - w_*, u_* - w_*)}_{>0} + t \dots$ is a non-degenerate parabola opening towards $+\infty$, which clearly has a unique global minimum at its apex.

Contradiction between ① and ② \Rightarrow assumption wrong. \square

Supplement 1.2.3.32 (Convexity of quadratic functionals) Under the assumptions of the theorem, the quadratic functional J is **convex**, cf. [Hip19, ??]:

Definition 1.2.3.33. Convexity of a real-valued function \rightarrow [Str09, Def. 5.5.2]

A function $F : V_0 \rightarrow \mathbb{R}$ on a vector space V_0 is called **convex**, if

$$F(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda F(\mathbf{x}) + (1-\lambda)F(\mathbf{y}) \quad \forall \mathbf{x}, \mathbf{y} \in V_0 . \quad (1.2.3.34)$$

A function is **concave**, if its negative is convex.

It is well known that a twice continuously differentiable function $F : \mathbb{R} \rightarrow \mathbb{R}$ is convex if and only if its second derivative F'' is non-negative everywhere. Thus, the convexity of a quadratic functional based on a positive definite quadratic bilinear form is easily seen by considering the second derivative of the function

$$\varphi(t) := J(u + tv) \Rightarrow \dot{\varphi}(t) = a(v, v) > 0 , \text{ if } v \neq 0 .$$

\square

1.2.3.3 Energy Norm

Recall that s.p.d. bilinear forms induce norms:

Theorem 0.3.1.21. Norms from inner products

If a is an inner product (= symmetric positive definite bilinear form, Def. 0.3.1.18) on the real vector space V , then

$$\|\cdot\|_a : V \rightarrow \mathbb{R} , \quad \|v\| := a(v, v)^{\frac{1}{2}} , \quad (0.3.1.22)$$

defines a **norm** (\rightarrow Def. 0.3.1.10) on V .

We have seen that the s.p.d. property of the bilinear form is necessary for the existence of a unique minimizer of a quadratic minimization problem. Thus there is natural norm associated with any meaningful quadratic minimization problem:

Definition 1.2.3.35. Energy norm, cf. [Hip19, ??]

Let \mathbf{a} be the symmetric positive definite bilinear form $\mathbf{a} : V_0 \times V_0 \mapsto \mathbb{R}$ (\rightarrow Def. 0.3.1.16) underlying a quadratic functional J . Then the related **energy norm** is

$$\|u\|_{\mathbf{a}} := (\mathbf{a}(u, u))^{1/2}, \quad u \in V_0.$$

Of course, the term “energy norm” is inspired by the quadratic potential energy functionals as they occur in many mathematical models.

Do the potential energy functionals from Section 1.2.1 and Section 1.2.2 really induce energy norms, that is, are they symmetric positive definite? We discuss this for the case of electrostatics.

§1.2.3.36 (Positive definite bilinear form for electrostatic variational problem) We have to establish that

$$\mathbf{a}(u, v) := \int_{\Omega} (\epsilon(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx$$

is positive definite on the function space $V_0 := C_{pw,0}^1(\bar{\Omega})$. Two arguments will confirm this:

①: Since ϵ bounded and uniformly positive definite (\rightarrow Def. 1.2.2.9)

$$\exists 0 < \epsilon^- \leq \epsilon^+ < \infty: \epsilon^- \|\mathbf{z}\|^2 \leq (\epsilon(x)\mathbf{z}) \cdot \mathbf{z} \leq \epsilon^+ \|\mathbf{z}\|^2 \quad \forall \mathbf{z} \in \mathbb{R}^3, \forall x \in \Omega, \quad (1.2.2.8)$$

we infer the bounds

$$0 \leq \epsilon^- \int_{\Omega} \|\mathbf{grad} u(x)\|^2 dx \leq \mathbf{a}(u, u) \leq \epsilon^+ \int_{\Omega} \|\mathbf{grad} u(x)\|^2 dx \quad \forall u. \quad (1.2.3.37)$$

Hence, it is sufficient to examine the simpler bilinear form

$$\mathbf{d}(u, v) := \int_{\Omega} \mathbf{grad} u(x) \cdot \mathbf{grad} v(x) dx, \quad u, v \in C_{pw,0}^1(\bar{\Omega}). \quad (1.2.3.38)$$

$$\begin{aligned} \textcircled{2}: \quad \text{Obviously} \quad \mathbf{d}(u, u) = 0 &\Rightarrow \mathbf{grad} u = 0 \xrightarrow{(1.2.1.22)} u \equiv \text{const in } \Omega \\ \text{Observe:} \quad u = 0 \text{ on } \partial\Omega &\Rightarrow \underbrace{u = 0}_{\text{green arrow}} \end{aligned}$$

Zero boundary conditions are essential; otherwise one could add constants to the arguments of \mathbf{a} without changing its value. \square

1.2.3.4 A Continuity Condition for the Linear Functional ℓ

In Cor. 1.2.3.26 and Thm. 1.2.3.31 we found necessary conditions on the bilinear form \mathbf{a} for the existence of a minimizer of an abstract quadratic minimization problem. Now we answer the questions whether the linear functional ℓ also has to satisfy some conditions.

As before, we consider a quadratic minimization problem (\rightarrow Def. 1.2.3.11) for a quadratic functional (\rightarrow Def. 1.2.3.2)

$$J : V_0 \mapsto \mathbb{R}, \quad J(u) = \frac{1}{2} \mathbf{a}(u, u) - \ell(u),$$

based on a symmetric positive definite (s.p.d.) bilinear form \mathbf{a} \rightarrow Def. 1.2.3.27.

The following insight demonstrates that the linear form ℓ has to match the bilinear form $a(\cdot, \cdot)$ to make possible to existence of a minimizer of J :

Lemma 1.2.3.39. Boundedness condition on linear form

The quadratic functional J (\rightarrow Def. 1.2.3.2)

$$J(u) := \frac{1}{2}a(u, u) - \ell(u) + c, \quad u \in V_0, \quad (1.2.3.3)$$

based on a symmetric positive definite bilinear form a (\rightarrow Def. 1.2.3.27) is bounded from below on V_0 , if and only if

$$\exists C > 0: |\ell(u)| \leq C\|u\|_a \quad \forall u \in V_0, \quad (1.2.3.40)$$

where $\|\cdot\|_a$ is the energy norm induced by a , see Def. 1.2.3.35.

Remark 1.2.3.41. Assertion (1.2.3.40) is written in a way customary in numerical analysis and the theory of partial differential equations. It should be read as “there is a constant $C > 0$ such that for all $u \in V_0$ the estimate $|\ell(u)| \leq C\|u\|_a$ holds true.”. In logic the quantors would be arranged differently:

$$(1.2.3.40) \Leftrightarrow \exists C > 0: \{\forall u \in V_0: |\ell(u)| \leq C\|u\|_a\}.$$

□

Proof. (of Lemma 1.2.3.39)

① Condition (1.2.3.40) ensures that J is bounded from below:

$$J(u) = \frac{1}{2}a(u, u) - \ell(u) \geq \frac{1}{2}\|u\|_a^2 - C\|u\|_a \geq -\frac{1}{2}C^2.$$

We arrive at this conclusion, since the parabola $p(\xi) = \frac{1}{2}\xi^2 - C\xi$ has a global minimum in $\xi = C$.

② The proof of the other implication is *indirect* (proof by contradiction). Assume that (1.2.3.40) does not hold. Then, for every $n \in \mathbb{N}$, we can find $u_n \in V_0$ such that

$$\ell(u_n) \geq n\|u_n\|_a.$$

By rescaling $u_n \leftarrow \frac{u_n}{\|u_n\|_a}$, we can assume without loss of generality that $\|u_n\|_a = 1$, which implies

$$J(u_n) \leq \frac{1}{2} - n \rightarrow -\infty \quad \text{for } n \rightarrow \infty.$$

Hence J can attain values below any threshold.

□

Parlance: In mathematical terms (1.2.3.40) means that ℓ is **continuous** w.r.t. the energy norm $\|\cdot\|_a$.

Definition 1.2.3.42. Continuity of a linear form and bilinear form

Consider a normed vector space V_0 with norm $\|\cdot\|$. A linear form $\ell : V_0 \rightarrow \mathbb{R}$ (\rightarrow Def. 0.3.1.4) is **continuous** or **bounded** on V_0 , if

$$\exists C > 0: |\ell(v)| \leq C\|v\| \quad \forall v \in V_0.$$

A bilinear form $a : V_0 \times V_0 \rightarrow \mathbb{R}$ (\rightarrow Def. 0.3.1.4) on V_0 is **continuous**, if

$$\exists K > 0: |a(u, v)| \leq K\|u\|\|v\| \quad \forall u, v \in V_0.$$

Remark 1.2.3.43 (Existence of minimizers in finite dimensions) Beside uniqueness, existence of minimizers of quadratic minimization problems (\rightarrow Def. 1.2.3.11) with positive definite bilinear form a is a key issue. In a finite dimensional setting this is not a moot point, see Fig. 37 for a “visual proof”. We do not even need any assumption on ℓ !

Theorem 1.2.3.44. Existence of unique minimizer in finite dimensions

Let $J(u) := \frac{1}{2}a(u, u) - \ell(u) + c$ with symmetric positive definite (\rightarrow Def. 1.2.3.27) bilinear form $a : V_0 \times V_0 \rightarrow \mathbb{R}$ (\rightarrow Def. 0.3.1.4), linear form $\ell : V_0 \rightarrow \mathbb{R}$, $c \in \mathbb{R}$, be a quadratic functional on the vector space V_0 .

If V_0 has **finite dimension**, then the quadratic minimization problem (\rightarrow Def. 1.2.3.11)

$$u_* = \underset{u \in V_0}{\operatorname{argmin}} J(u)$$

always possesses a unique solution.

However, as we will see below infinite dimensional spaces hold a lot of surprises and existence of solutions of quadratic minimization problems becomes a subtle issue, even if the bilinear form is positive definite. \lrcorner

EXAMPLE 1.2.3.45 (Point load) Now we inspect a striking case of quadratic minimization problem with a linear form that fails to satisfy the continuity condition (1.2.3.40). Of course, this can happen only in an infinite-dimensional setting.

We consider energy minimization for a homogeneous 2D membrane, that is, the stiffness σ is constant and we will simply set it to 1. Let us assume that a needle is poked at the membrane: loading by a force f “concentrated in a point y ”, often denoted by $f = \delta_y$, $y \in \Omega$, where δ is the so-called **Dirac delta function** (delta distribution).

Then, in light of the property

$$\int_{\Omega} \delta_y(x) v(x) dx = v(y) \quad \forall v \in C^0(\bar{\Omega})$$

of the Dirac delta function, the quadratic minimization problem (1.2.3.1b) for the potential energy takes the special form

$$u_* = \underset{\substack{u \in C_{\text{pw}}^1(\bar{\Omega}) \\ u=g \text{ on } \partial\Omega}}{\operatorname{argmin}} \underbrace{\int_{\Omega} \frac{1}{2} \sigma(x) \|\mathbf{grad} u(x)\|^2 dx}_{=: \frac{1}{2} a(u, u)} - \underbrace{u(y)}_{=:\ell(u)}. \quad (1.2.3.46)$$

By formal arguments we will now make the startling discovery that the linear functional

$$\ell : C_{\text{pw}}^1(\bar{\Omega}) \rightarrow \mathbb{R}, \quad v \mapsto v(y)$$

is **not bounded** with respect to the energy norm

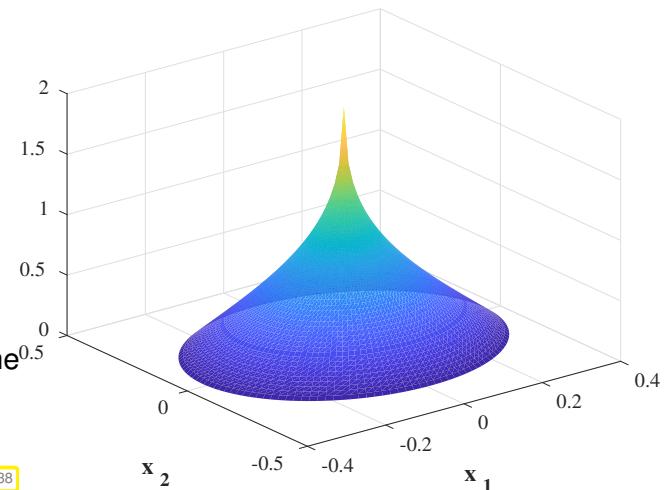
$$\|v\|_a := \left(\int_{\Omega} \|\mathbf{grad} v\|^2 dx \right)^{\frac{1}{2}}, \quad v \in C_{\text{pw}}^1(\bar{\Omega}).$$

For $\epsilon > 0$ consider the function

$$v_\epsilon(x) := \begin{cases} \log |\log \|x\|| & , \text{ if } \|x\| > \epsilon , \\ \log |\log \epsilon| & , \text{ if } \|x\| \leq \epsilon , \end{cases}$$

on the disk domain $\Omega = \{x \in \mathbb{R}^2 : \|x\| < \frac{1}{\epsilon}\}$.

By definition $v_\epsilon \in C_{pw}^1(\Omega)$, since we have cut off the peak at $x = 0$.



First, we express this function in **polar coordinates** (r, φ)

$$x_1 = r \cos \varphi , \quad x_2 = r \sin \varphi , \quad (1.2.3.47)$$

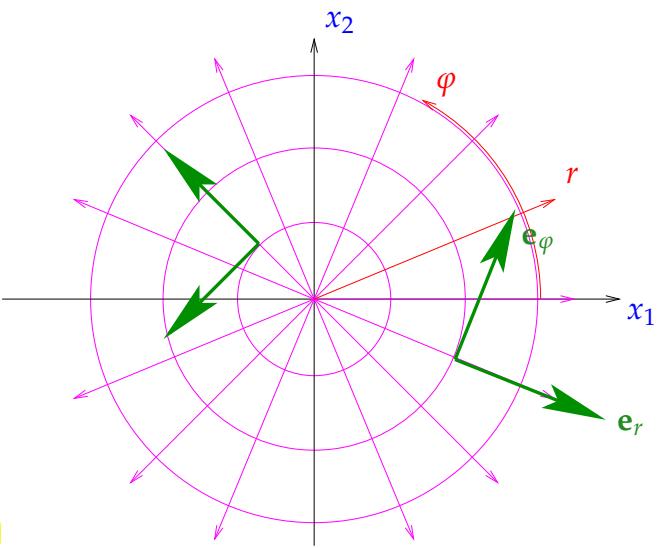
$$\Rightarrow v_\epsilon(r, \varphi) = \log |\log r|, \quad r > \epsilon .$$

Then we recall the expression for the gradient in polar coordinates

$$\mathbf{grad} v_\epsilon(r, \varphi) = \frac{\partial v_\epsilon}{\partial r}(r, \varphi) \mathbf{e}_r + \frac{1}{r} \frac{\partial v_\epsilon}{\partial \varphi}(r, \varphi) \mathbf{e}_\varphi , \quad (1.2.3.48)$$

where \mathbf{e}_r and \mathbf{e}_φ are orthogonal unit vectors in the polar coordinate directions.

Fig. 39



Also recall integration in polar coordinates, see [Str09, Bsp. 8.5.3]:

$$\int_{\Omega} f(x) dx = \int_0^{1/\epsilon} \int_0^{2\pi} f(r, \varphi) r d\varphi dr . \quad (1.2.3.49)$$

Using polar coordinates and (1.2.3.49), we compute $\|v\|_a$ by evaluating an improper integral,

$$\begin{aligned} \|v_\epsilon\|_a^2 &= \int_{\Omega} \|\mathbf{grad} v(x)\|^2 dx = \int_{\epsilon}^{1/\epsilon} \int_0^{2\pi} \left\| -\frac{1}{\log r} \mathbf{e}_r \right\|^2 r d\varphi dr \leq 2\pi \int_0^{1/\epsilon} \frac{1}{\log^2 r} \cdot \frac{1}{r} dr \\ &= 2\pi [-1/\log r]_0^{1/\epsilon} = \frac{2\pi}{\log \epsilon} = 2\pi < \infty . \end{aligned}$$

This is allowed, because the improper integral has a finite value. This means that v_ϵ has “finite elastic energy” for any $\epsilon > 0$.

Yet, $v_\epsilon(0) = \log |\log \epsilon| \rightarrow \infty$ as $\epsilon \rightarrow 0$. In 2D functions with arbitrarily small energy can attain arbitrarily big values in isolated points!

This is the mathematics behind the observation that a needle can easily prick a taut membrane: a point load leads to configurations with “infinite elastic energy”. Of course, this does not correspond to “real physics”, but indicates that point loads are outside the scope of the simple linear continuum membrane model.



Review question(s) 1.2.3.50 (Quadratic minimization problems)

(Q1.2.3.50.A) [Fundamental concepts] Let V be a real vector space. Give the formal mathematical definitions of the following concepts:

- of a **linear form** ℓ on V ,
- of a **bilinear form** a on V ,
- of a **positive definite** bilinear form a on V ,
- and of a **quadratic functional** on V .

(Q1.2.3.50.B) [Electrostatic field energy as quadratic functional] The electrostatic field energy

$$J_E(u) := \int_{\Omega} \frac{1}{2} (\epsilon(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} u(x) \, dx .$$

is a quadratic functional. On which space is it defined and what are the involved bilinear form and linear form according to the abstract definition of a quadratic functional?

(Q1.2.3.50.C) [Quadratic functional in two dimensions] The general form a quadratic functional on \mathbb{R}^N , $N \in \mathbb{N}$, is

$$J(\vec{\eta}) = \frac{1}{2} \vec{\eta}^\top \mathbf{A} \vec{\eta} - \vec{\beta}^\top \vec{\eta} + c , \quad \mathbf{A} = \mathbf{A}^\top \in \mathbb{R}^{N,N} , \quad \vec{\beta} \in \mathbb{R}^N , \quad c \in \mathbb{R} . \quad (1.2.3.6)$$

Write down the matrix \mathbf{A} , the vector $\vec{\beta}$, and the number c concretely for the quadratic functional

$$x \in \mathbb{R}^2 \mapsto \|x\|^2 - (x_1 - 1) .$$

(Q1.2.3.50.D) For what values of $\alpha, \beta, \gamma \in \mathbb{R}$ does the quadratic functional

$$J : \mathbb{R}^2 \rightarrow \mathbb{R} , \quad J(x) := x^\top \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} x - \gamma x_1 + \alpha ,$$

possess a unique minimizer.

(Q1.2.3.50.E) [Energy norm] What is the energy norm induced by a symmetric positive definite bilinear form on a vector space V ?

(Q1.2.3.50.F) [Bounded/continuous linear functional] What does it mean that a linear functional ℓ on a vector space V_0 is **bounded/continuous** with respect to a norm $\|\cdot\|$ on V_0 ?

(Q1.2.3.50.G) Show that every linear functional on $V_0 := \mathbb{R}^2$ is bounded with respect to the Euclidean norm on \mathbb{R}^2 .

(Q1.2.3.50.H) [Unbounded point evaluation] Rephrase the following statement in formal mathematical terms:

Point evaluation is an unbounded linear functional on $C^1([0,1]^2)$ with respect to the energy norm

$$\|v\|_a := \left(\int_{[0,1]^2} \|\mathbf{grad} v(x)\|^2 dx \right)^{\frac{1}{2}}.$$

(Q1.2.3.50.I) [Extracting components of a quadratic functional] Outline the implementation of the class **QuadFunc** meant to extract the building blocks of a quadratic functional according to the following definition.

Definition 1.2.3.2. Quadratic functional

A **quadratic functional** on a *real vector space* V_0 is a mapping $J : V_0 \mapsto \mathbb{R}$ of the form

$$J(u) := \frac{1}{2}a(u, u) - \ell(u) + c, \quad u \in V_0, \quad (1.2.3.3)$$

where $a : V_0 \times V_0 \mapsto \mathbb{R}$ is a **symmetric** bilinear form (\rightarrow Def. 0.3.1.4), $\ell : V_0 \mapsto \mathbb{R}$ a linear form, and $c \in \mathbb{R}$.

```
template <VECTOR, QF_FUNCTOR>
class QuadFunc {
public:
    QuadFunc(QF_FUNCTOR &qf_functor) : qf_functor_(qf_functor) {}
    double a(const VECTOR &u, const VECTOR &v) const;
    double ell(const VECTOR &v) const;
private:
    QF_FUNCTOR &qf_functor_;
};
```

The type **QF_FUNCTOR** must provide an evaluation operator

```
double operator () (const VECTOR &u) const;
```

which gives access to $J(u) \in \mathbb{R}$ for any argument vector $u \in V_0$. An object of such a type is passed to the constructor of the class. Based on that object, the methods `a()` and `ell()` should allow the evaluation of $a(u, v)$ and $\ell(v)$.

△

1.3 Sobolev spaces



Video tutorial for Section 1.3: Sobolev Spaces: (47 minutes) [Download link](#), [tablet notes](#)

EXAMPLE 1.3.0.1 (Non-existence of solutions of positive definite quadratic minimization problem)

We consider the quadratic functional

$$J(u) := \int_0^1 \frac{1}{2}u^2(x) - u(x) dx = \frac{1}{2} \int_0^1 \{(u(x) - 1)^2 - 1\} dx,$$

on the space of continuous functions which attain value zero at the endpoints of the interval: $V_0 := C_0^0([0, 1])$. The functional J fits the abstract form from Def. 1.2.3.2 with

$$a(u, v) = \int_0^1 u(x)v(x) dx , \quad \ell(v) = \int_0^1 v(x) dx \quad u, v \in V_0. \quad (1.3.0.2)$$

The quadratic minimization problem involving J is “very nice”: all necessary conditions for existence and uniqueness of minimizers from Cor. 1.2.3.26, Thm. 1.2.3.31, and Lemma 1.2.3.39 are satisfied.

Lemma 1.3.0.3.

The bilinear form $a(\cdot, \cdot)$ from (1.3.0.2) is symmetric and positive definite on V_0 and the linear form $\ell(\cdot)$ is continuous w.r.t. the energy norm induced by $a(\cdot, \cdot)$.

Proof. The s.p.d. property of $a(\cdot, \cdot)$ is obvious, because $a(v, v) = \int_0^1 v^2(x) dx$.

The continuity of ℓ w.r.t the energy norm $\|\cdot\|_a$ (\rightarrow Def. 1.2.3.35) follows from the Cauchy-Schwarz inequality Thm. 0.3.1.19:

$$|\ell(v)| = |a(\mathbf{1}, v)| \leq \|\mathbf{1}\|_a \|v\|_a = \|v\|_a \quad \forall v \in C^0([a, b]).$$

□

Now we study minimizers of J : The function $\varphi(\xi) = \frac{1}{2}\xi^2 - \xi = \frac{1}{2}\xi(1 - 2\xi) = \frac{1}{2}(\xi - 1)^2 - \frac{1}{2}$ has a global minimum at $\xi = 1$ and $\varphi(\xi) - \varphi(1) = \frac{1}{2}(\xi - 1)^2$.

► $|\eta - 1| > |\xi - 1| \Rightarrow \varphi(\eta) > \varphi(\xi).$

Assume that $u_* \in V_0$ is a global minimizer of J . Then

$$w(x) := \min\{1, 2 \max\{u_*(x), 0\}\}, \quad 0 \leq x \leq 1,$$

is another function $\in C_0^0([0, 1])$, which satisfies

$$\begin{aligned} u(x) \neq 1 &\Rightarrow |w(x) - 1| < |u_*(x) - 1| \\ &\Rightarrow J(w) < J(u_*) ! \end{aligned}$$

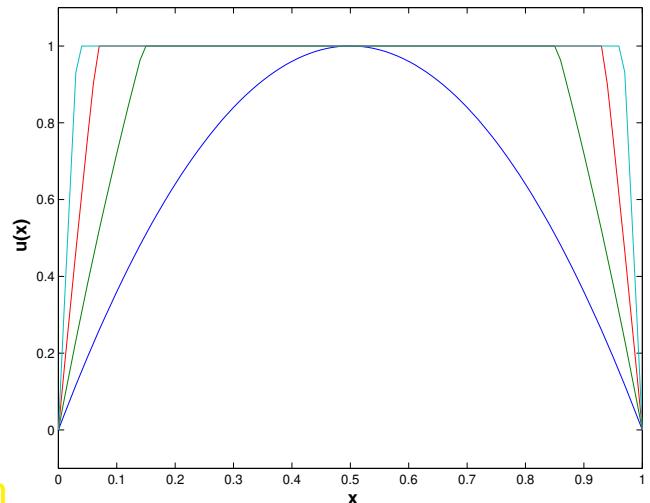


Fig. 41

Hence, whenever we think we have found a minimizer $\in C_0^0([0, 1])$, the formula provides another eligible function for which the value of the functional is even smaller! Therefore we can find sequences $(u_n)_n$ of functions in $C_0^0([0, 1])$ for which $J(u_n)$ tends to the minimum, whereas the sequence itself has no limit in $C_0^0([0, 1])$.

The problem in this example seems to be that we have chosen “too small” a function space, c.f. Section 1.3 below. □

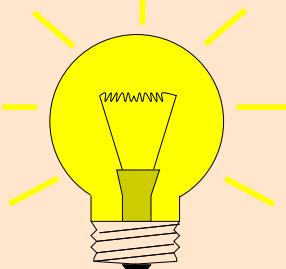
§1.3.0.4 (Preview) Mathematical theory is much concerned about proving existence of suitably defined solutions for minimization problems. As demonstrated in Ex. 1.3.0.1 this can encounter profound problems.

In this section we will learn about a class of *abstract function spaces* that has been devised to deal with the question of existence of solutions of quadratic minimization problems like (1.2.3.1b) and (1.2.3.1c). We

can only catch a glimpse of the considerations; thorough investigation is done in the mathematical field of *functional analysis*. \square

1.3.1 Function Spaces for Energy Minimization

Now we return to the question of how to choose the right function space for a linear variational problem. Bounded energy norm will be the linchpin of the argument:



Guideline: for a quadratic minimization problem (\rightarrow Def. 1.2.3.11) with

- ◆ symmetric positive definite (s.p.d.) bilinear form a ,
- ◆ a linear form ℓ that is continuous w.r.t. $\|\cdot\|_a$, see (1.2.3.40),

posed over a function space follow the advice:

consider it on the largest space of functions
for which a still makes sense !

(and which complies with boundary conditions)

In the concrete case of quadratic minimization problems like (1.2.3.1b) (minimization of potential energy of a membrane) and (1.2.3.1c) (minimization of the energy of an electrostatic field) we arrive at the following recommendation:

Choose “ $V_0 := \{ \text{functions } v \text{ on } \Omega : a(v, v) < \infty \}$ ”

Then V_0 is called the **energy (function) space** generated by the energy norm $\|\cdot\|_a$.

➤ “Reasoning turned upside down”: now we first look at the quadratic functional J or, equivalently, the bilinear form a (and potential boundary conditions), they determine the function space on which the minimization problem/variational problem is posed!

§1.3.1.1 (Switching to a larger set of numbers) Long ago you already witnessed an application of the policy of considering a minimization problem on a larger space in order to obtain solutions.

Consider the minimization problem:

$$\text{Find } x^* \in \operatorname{argmin}_{x \in \mathbb{Q}} J(x) \quad \text{for} \quad J(x) := |x^2 - 2|.$$

Obviously, there is no solution: for every $x^* \in \mathbb{Q}$ we find another one, for which J returns a smaller value. The remedy is to switch from \mathbb{Q} to \mathbb{R} : minimizers $\in \mathbb{R}$ obviously exist. \square

1.3.2 The Function Space $L^2(\Omega)$

Consider the quadratic functional (related to J from Ex. 1.3.0.1)

$$J(u) := \int_{\Omega} \frac{1}{2} |u(x)|^2 - u(x) \, dx . \quad \left(u \in C_{pw}^0(\Omega) ? \right) \quad (1.3.2.1)$$

Employing the reasoning of Section 1.2.1.3 we conclude that the space of piecewise continuous functions might provide the right setting for treating this functional. Now we follow the above recipe, which suggests that we choose

► $V_0 := \{ v : \Omega \mapsto \mathbb{R} \text{ integrable: } \int_{\Omega} |v(x)|^2 \, dx < \infty \}$ (1.3.2.2)

1. Second-Order Scalar Elliptic Boundary Value Problems, 1.3. Sobolev spaces

73

Definition 1.3.2.3. Space $L^2(\Omega)$ → **Def. 0.3.2.27**

The function space defined in (1.3.2.2) is the **space of square-integrable functions** on Ω and denoted by $L^2(\Omega)$.

It is a normed space with norm

$$\left(\|v\|_0 := \right) \|v\|_{L^2(\Omega)} := \left(\int_{\Omega} |v(x)|^2 dx \right)^{1/2}.$$

Notation: $L^2(\Omega)$ ← superscript “2”, because square in the definition of norm $\|\cdot\|_0$

We point out the straightforward inclusion $C_{pw}^0(\Omega) \subset L^2(\Omega)$.

Remark 1.3.2.4 (Mathematical notion of $L^2(\Omega)$) Here, we do **not** make an attempt to provide a rigorous mathematical definition of $L^2(\Omega)$. This is done the *measure theory* and involves quotient spaces; a rather accessible presentation is given in [Rud86, Ch. 3].

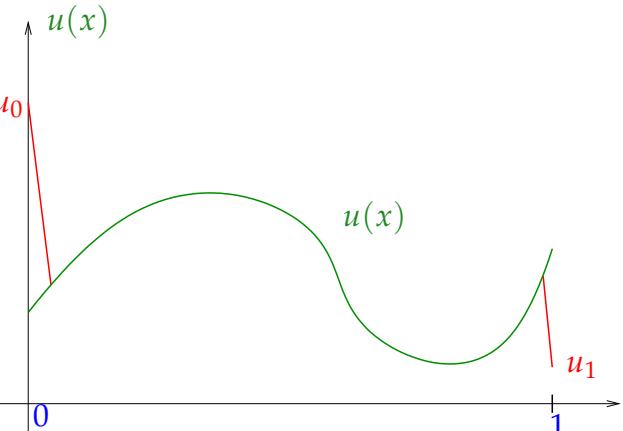
EXAMPLE 1.3.2.5 (Boundary conditions and $L^2(\Omega)$) Recall that in Ex. 1.3.0.1 we had intended to use a space of functions vanishing at the endpoints of the interval. Yet, no boundary conditions are implied by Eq. (1.3.2.2): It seems that $L^2([0,1])$ is not the right space for the minimization problem of Ex. 1.3.0.1, because apparently the boundary conditions have been forgotten.

Now we pursue a more sophisticated reasoning, which will finally tell us that fixing the values in $x = 0, 1$ for functions in $L^2([0,1])$ does not make sense.

Consider $u \in C^0([0,1])$ and try to impose **any** boundary values $u_0, u_1 \in \mathbb{R}$ by “altering” u in the following way:

(red parts of the graph belong to \tilde{u}_n)

Fig. 42



$$\tilde{u}_n(x) := \begin{cases} u(x) + (1-nx)(u_0 - u(0)) & , \text{ for } 0 \leq x \leq \frac{1}{n}, \\ u(x) & , \text{ for } \frac{1}{n} < x < 1 - \frac{1}{n}, \\ u(x) - n(1 - \frac{1}{n} - x)(u_1 - u(1)) & , \text{ for } 1 - \frac{1}{n} < x \leq 1, \end{cases} \quad n \in \mathbb{N}.$$

► $\tilde{u}_n(0) = u_0$, $\tilde{u}_n(1) = u_1$, $\|\tilde{u}_n - u\|_{L^2([0,1])}^2 = \frac{1}{3n}(u_0 + u_1 - u(0) - u(1)) \rightarrow 0$ for $n \rightarrow \infty$.

► Measured in the L^2 -norm, tiny perturbations of a function $u \in L^2([0,1])$ can make it attain any value at $x = 0$ and $x = 1$:

$$\forall u \in L^2([0,1]), \quad \forall u_a, u_b \in \mathbb{R}, \quad \forall \epsilon > 0: \quad \exists \tilde{u} \in L^2([0,1]): \quad \begin{cases} \tilde{u}(0) = u_a, \tilde{u}(1) = u_b, \\ \|\tilde{u} - u\|_{L^2([0,1])} \leq \epsilon. \end{cases}$$

► Mathematically, this means that the space $V = \{u \in L^2([0,1]) : u(0) = u(1) = 0\}$ is *not* closed in the energy space $L^2([0,1])$, meaning that there exist functions which are not in V but which can be arbitrarily well approximated by elements of V . Ex. 1.3.0.1 makes this concrete: the solution is approximated better and better but it is never reached because the trial space is too small.

Boundary conditions cannot be imposed in $L^2(\Omega)$!

This example has taught us an important lesson:

Energy-based function spaces and boundary conditions

Given a function space V_0 “defined” through its energy norm, only boundary conditions compatible with the energy norm can be imposed on functions $\in V_0$.

1.3.3 Supplement: Quadratic Minimization Problems on Hilbert Spaces

In this section we let you catch a glimpse of the rigorous functional analytic treatment of minimization problems on infinite dimensional spaces. We review the mathematical arguments that confirm the existence of minimizers of quadratic minimization problems

$$u_* = \operatorname{argmin}_{v \in V_0} J(v), \quad J(v) = \frac{1}{2}a(v, v) - \ell(v), \quad (1.3.3.1)$$

where

- ◆ V_0 is a real vector space, possibly of infinite dimension,
- ◆ $a : V_0 \times V_0 \rightarrow \mathbb{R}$ is a **symmetric positive definite** bilinear form (\rightarrow Def. 1.2.3.27) inducing an energy norm (\rightarrow Def. 1.2.3.35) $\|\cdot\|_a$ on V_0 ,
- ◆ $\ell : V_0 \rightarrow \mathbb{R}$ is a linear form, which is **bounded** with respect to the energy norm (\rightarrow Def. 1.2.3.42).

§1.3.3.2 (Completeness of normed vector spaces) The entire theory is based on a key matching condition for the space V_0 and its energy norm. To express this, we need a fundamental concept from analysis:

Definition 1.3.3.3. Cauchy sequence \rightarrow [Str09, Def. 3.5.1]

Consider a normed vector space V_0 equipped with a norm $\|\cdot\|$ (\rightarrow Def. 0.3.1.10). A sequence $(v_n)_{n \in \mathbb{N}}$ of vectors of V_0 is called a **Cauchy sequence**, if

$$\forall \epsilon > 0: \exists n = n(\epsilon) \in \mathbb{N}: \|v_k - v_m\| \leq \epsilon \quad \forall k, m \geq n.$$

Clearly, every convergent sequence is a Cauchy sequence. The converse is true only in exceptional cases, which are of enormous importance in mathematical modelling, however, which has earned them a particular name.

Definition 1.3.3.4. Complete normed vector spaces and Hilbert spaces \rightarrow [Wer95, Def. I.1.2 & V.1.4]

A normed vector space is called **complete**, if every Cauchy sequence converges. A complete normed vector space is known as **Banach space**.

If the norm of a complete normed vector space V_0 is an energy norm (\rightarrow Def. 1.2.3.35) associated with a symmetric positive definite bilinear form (\rightarrow Def. 1.2.3.27), then V_0 is called a **Hilbert space**.

EXAMPLE 1.3.3.5 (Important Banach spaces and Hilbert spaces)

- The real numbers \mathbb{R} equipped with the modulus as norm $|\cdot|$ are complete.
- Every *finite dimensional* normed real or complex vector space is complete.
- For every bounded (\Rightarrow compact) domain $\Omega \subset \mathbb{R}^d$ the space $C^0(\bar{\Omega})$, equipped with the supremum norm $\|\cdot\|_\infty$ (\rightarrow Def. 0.3.2.25) is a Banach space [Wer95, I.1 Bsp. (c)].
- For any domain $\Omega \subset \mathbb{R}^d$ the function space $L^2(\Omega)$ (\rightarrow Def. 1.3.2.3) is a Hilbert space [Wer95, I.1 Bsp. (h)].

□

The main existence theorem is given next.

Theorem 1.3.3.6. Existence of minimizers in Hilbert spaces

On a real *Hilbert space* V_0 with (energy) norm $\|\cdot\|_a$ for any $\|\cdot\|_a$ -bounded linear functional $\ell : V_0 \rightarrow \mathbb{R}$ the quadratic minimization problem

$$u_* = \operatorname{argmin}_{v \in V_0} J(v), \quad J(v) := \frac{1}{2} \|v\|_a^2 - \ell(v), \quad (1.3.3.7)$$

has a unique solution.

Proof. Owing to the assumptions on ℓ , by Lemma 1.2.3.39 the quadratic functional J is bounded from below. Hence, there is a *minimizing sequence* $(v_n)_{n \in \mathbb{N}}$, which satisfies

$$|J(v_n) - \mu| \leq 1/n \quad \text{where} \quad \mu := \inf_{v \in V_0} J(v). \quad (1.3.3.8)$$

Write $a(\cdot, \cdot)$ for the bilinear form spawning $\|\cdot\|_a$, that is $\|v\|_a^2 = a(v, v)$. From (bi-)linearity it is immediate that

$$\frac{1}{2}(J(v) + J(w)) - J\left(\frac{1}{2}(v + w)\right) = \frac{1}{4}\left(a(v, v) + a(w, w) - 2a\left(\frac{1}{2}(v + w), \frac{1}{2}(v + w)\right)\right) = \frac{1}{8}\|v - w\|_a^2.$$

This implies

$$\begin{aligned} \frac{1}{8}\|v_k - v_m\|_a^2 &\leq \frac{1}{2}\left(\underbrace{J(v_k)}_{\leq \mu + 1/k} + \underbrace{J(v_m)}_{\leq \mu + 1/m}\right) - \underbrace{J\left(\frac{1}{2}(v_k + v_m)\right)}_{\geq \mu} \\ (1.3.3.8) \quad &\leq \frac{1}{2}(1/k + 1/m) \leq \max\{1/k, 1/m\}. \end{aligned}$$

Hence, $(v_n)_{n \in \mathbb{N}}$ is a *Cauchy sequence* (\rightarrow Def. 1.3.3.3) and

$$u_* := \lim_{n \rightarrow \infty} v_n \in V_0$$

exists and satisfies

$$J(u_*) = \inf_{v \in V_0} J(v).$$

In other words, the limit u_* is a global minimizer of J on V_0 . Its uniqueness is established by the arguments of the proof of Thm. 1.2.3.31. □

Remark 1.3.3.9 (Quadratic minimization problem in $L^2(\Omega)$) Since $L^2(\Omega)$ is a Hilbert space, the previous theorem guarantees that the quadratic minimization problem for the quadratic functional from (1.3.2.1) on the function space $V_0 = L^2(\Omega)$, $\Omega :=]0, 1[$, possesses a solution.

Conversely, though the minimization problem of Ex. 1.3.0.1 was considered on the Banach space $V_0 := C_0^0([0,1])$, the bilinear form in the quadratic functional failed to be related to the (supremum) norm, which rules out the application of Thm. 1.3.3.6. \square

§1.3.3.10 (Completion of a normed vector space) The powerful Thm. 1.3.3.6 is available only in Hilbert spaces, which makes it very desirable to put quadratic minimization problems in a Hilbert space setting. Surprisingly, this can always be achieved by the procedure of **completion**.

Completion can be used to “fill the pores” of any normed vector spaces with potential limits of Cauchy sequences so that the resulting augmented space is complete in the sense of Def. 1.3.3.4.

Theorem 1.3.3.11. Completion of a normed vector space

For every normed vector space V_0 there is a unique (up to isomorphism) complete vector space \tilde{V}_0 that contains V_0 as a **dense subspace**.

Definition 1.3.3.12. Dense subset

A subset $U \subset V_0$ is said to be **dense** in a normed vector space V_0 , if every element of V_0 is the limit of a sequence in U .

EXAMPLE 1.3.3.13 (\mathbb{R} by completion of \mathbb{Q}) Every reader has seen an example of completion already in secondary school: the real numbers \mathbb{R} are obtained by “completing” the rational numbers \mathbb{Q} . This amounts to “filling the gaps in \mathbb{Q} ”. Thus many equations that fail to possess a solution in \mathbb{Q} can be solved in \mathbb{R} , see § 1.3.1.1. \square

Hence, when tackling the minimization of a quadratic functional (1.3.3.1) with positive definite bilinear form a on a vector space V_0 , we can first switch to the completion \tilde{V}_0 of V_0 with respect to the energy norm $\|\cdot\|_a$ induced by a . Then, Thm. 1.3.3.6 will ensure the existence of a unique minimizer in \tilde{V}_0 .

What will we get when we apply the completion trick to the quadratic functional of Ex. 1.3.0.1, for which $V_0 = C_0^0([0,1])$ and $a(u,v) = \int_0^1 u(x)v(x) dx$? The next theorem gives an answer.

Theorem 1.3.3.14. $L^2(\Omega)$ by completion

For any domain $\Omega \subset \mathbb{R}^d$ the completion of $C_0^0(\bar{\Omega})$ equipped with the norm $\|\cdot\|_{L^2(\Omega)}$ is the function space $L^2(\Omega)$.

As a consequence, when we resort to completion in Ex. 1.3.0.1, we end up in $L^2([0,1])$ and inevitably loose the boundary conditions, cf. Ex. 1.3.2.5, but get the unique solution $u \equiv 1$. \square

Remark 1.3.3.15 (Boundary conditions and density) In Ex. 1.3.2.5 we have seen that we cannot impose boundary conditions in $L^2(\Omega)$. This is evident from Thm. 1.3.3.14: Since functions that vanish on the boundary $\partial\Omega$ are dense in $L^2(\Omega)$, any $v \in L^2(\Omega)$ can be approximated to arbitrary precision (in $L^2(\Omega)$ -norm, of course) by a function, which is zero on $\partial\Omega$. This was, what the sequence of functions \tilde{u}_n , $n \in \mathbb{N}$, did in Ex. 1.3.2.5. \square

1.3.4 The Sobolev Space $H^1(\Omega)$

Now consider a quadratic minimization problem for the functional, c.f. (1.2.3.1b),

$$J(u) := \int_{\Omega} \frac{1}{2} \|\mathbf{grad} u\|^2 - f(x)u(x) dx \quad (u \in C_{pw,0}^1(\Omega) ?) \quad (1.3.4.1)$$

What is the natural function space for this minimization problem? According to Section 1.2.1.3 we should opt for $C_{\text{pw}}^1(\bar{\Omega})$. Now, again, we follow the above recipe, which suggests that we choose

► $V_0 := \{v : \Omega \mapsto \mathbb{R} \text{ integrable: } v = 0 \text{ on } \partial\Omega, \int_{\Omega} |\mathbf{grad} v(x)|^2 dx < \infty\}$ (1.3.4.2)

Definition 1.3.4.3. Sobolev space $H_0^1(\Omega)$

The space of integrable functions on Ω with square integrable gradient that vanish on the boundary $\partial\Omega$,

$$V_0 := \{v : \Omega \mapsto \mathbb{R} \text{ integrable: } v = 0 \text{ on } \partial\Omega, \int_{\Omega} |\mathbf{grad} v(x)|^2 dx < \infty\}, \quad (1.3.4.2)$$

is the **Sobolev space** $H_0^1(\Omega)$ with norm

$$|v|_{H^1(\Omega)} := \left(\int_{\Omega} \|\mathbf{grad} v\|^2 dx \right)^{1/2}.$$

☞ Notation: $H_0^1(\Omega)$ ← superscript “1”, because first derivatives occur in norm
← subscript “0”, because zero on $\partial\Omega$

☞ Notation: Alternative notations for H^1 -norm: $|v|_{H^1(\Omega)} = |v|_{1,\Omega} = |v|_1$, where the last notation is used, when the domain is clear from the context.

Note: $|\cdot|_{H^1(\Omega)}$ is the **energy norm** (\rightarrow Def. 1.2.3.35) associated with the bilinear form in the quadratic functional J from (1.3.4.1), cf. (1.2.3.3).

Remark 1.3.4.4 (Boundary conditions in $H_0^1(\Omega)$) The example Ex. 1.3.2.5 conveyed why imposing boundary conditions on functions in $L^2(\Omega)$ does not make sense.

Yet, in (1.3.4.2) zero boundary conditions are required for v ! This might not be possible and Def. 1.3.4.3 might be wrong.

We focus on $\Omega =]0, 1[$ as in Ex. 1.3.2.5 and wonder, whether any boundary values in $x = 0, 1$ can be imposed on a function with a negligible change of its $|\cdot|_{H^1(\Omega)}$ -norm. We try to reason parallel to Ex. 1.3.2.5. Consider $u \in C^1([0, 1])$ and attempt to enforce boundary values $u_0, u_1 \in \mathbb{R}$ by “altering” u , see Fig. 42: as before, for $n \in \mathbb{N}$ set

$$\tilde{u}_n(x) = \begin{cases} u(x) + (1 - nx)(u_0 - u(0)) & , \text{ for } 0 \leq x \leq \frac{1}{n}, \\ u(x) & , \text{ for } \frac{1}{n} < x < 1 - \frac{1}{n}, \\ u(x) - n(1 - \frac{1}{n} - x)(u_1 - u(1)) & , \text{ for } 1 - \frac{1}{n} < x \leq 1. \end{cases}$$

► $\tilde{u}_n(0) = u_0, \tilde{u}_n(1) = u_1$, BUT $|\tilde{u}_n - u|_{H^1(]0,1[)}^2 = n(u_0 + u_1 - u(0) - u(1)) \rightarrow \infty$ for $n \rightarrow \infty$.

► Enforcing boundary values at $x = 0$ and $x = 1$ cannot be done without significantly changing the “energy” of the function: Def. 1.3.4.3 seems to be correct.

There is an important lesson to be learned from Ex. 1.3.2.5 and the above considerations for $H^1(]0, 1[)$:

Boundary conditions in energy spaces

The energy norm that generated an energy function space also determines what kind of boundary conditions can be imposed on functions in that space.

§1.3.4.6 (Point evaluation functional) At this point let us view the point-load case of Ex. 1.2.3.45 from the perspective of Sobolev spaces. The function v that we examined in that example satisfies $|v|_{H^1(\Omega)} < \infty$, though it is unbounded. We conclude, that $H^1(\Omega)$ “contains” unbounded functions! An equivalent statement is made in the following corollary:

Corollary 1.3.4.7. Point evaluation on $H^1(\Omega)$

For $d \geq 2$ and a domain $\Omega \subset \mathbb{R}^d$ the point evaluation $v \mapsto v(y)$, $y \in \Omega$, is **not a continuous linear form on $H^1(\Omega)$.**

Often the solutions of the quadratic minimization problems (1.2.3.1b), (1.2.3.1c) are to satisfy non-zero boundary conditions. They belong to an affine space $u_0 + V_0$ for a suitable offset function u_0 , see § 1.2.3.12. This affine space will be contained in a larger Sobolev space, which arises from $H_0^1(\Omega)$ by dispensing with the requirement “ $v = 0$ on $\partial\Omega$ ”.

Definition 1.3.4.8. Sobolev space $H^1(\Omega)$

The Sobolev space

$$H^1(\Omega) := \{v : \Omega \mapsto \mathbb{R} \text{ integrable: } \int_{\Omega} |\operatorname{grad} v(x)|^2 dx < \infty\}$$

is a normed function space with norm

$$\|v\|_{H^1(\Omega)}^2 := \|v\|_0^2 + |v|_{H^1(\Omega)}^2 .$$

► $H^1(\Omega)$ is the “maximal function space” on which both J_S, J_M and J_E from (1.2.3.1b), (1.2.3.1c) are defined.

It is the natural function space, in which to state the energy minimization problems (1.2.3.1a)–(1.2.3.1b):

Section 1.2.1.2 [1D, string]

$$u_* = \operatorname{argmin}_{\substack{v \in H^1([a,b]) \\ v(a)=u_a \ v(b)=u_b}} \underbrace{\int_a^b \frac{1}{2} \sigma(x) \left| \frac{dv}{dx}(x) \right|^2 - f(x)v(x) dx}_{=: J_S(u), \text{ see ((Q1.2.1.30.F))}} \quad (1.3.4.9a)$$

Section 1.2.1.2 [2D membrane]

$$u_* = \operatorname{argmin}_{\substack{v \in H^1(\Omega) \\ v=g \text{ on } \partial\Omega}} \underbrace{\int_{\Omega} \frac{1}{2} \sigma(x) \|\operatorname{grad} v(x)\|^2 - f(x)v(x) dx}_{=: J_M(u), \text{ see (1.2.1.19)}} , \quad (1.3.4.9b)$$

Section 1.2.2 [2D, 3D electrostatics]

$$u_* = \operatorname{argmin}_{\substack{v \in H^1(\Omega) \\ v=U \text{ on } \partial\Omega}} \underbrace{\int_{\Omega} \frac{1}{2} (\epsilon(x) \operatorname{grad} v(x)) \cdot \operatorname{grad} v(x) dx}_{=: J_E(u), \text{ see (1.2.2.6)}} . \quad (1.3.4.9c)$$

In all these cases we have $V_0 = H_0^1(\Omega)$ ($\Omega =]a, b[$ for (1.3.4.9a)).

Supplement 1.3.4.10 ($H^1(\Omega)$ through completion) In § 1.3.3.10 we elaborated how one can build a complete function space as suitable setting for a quadratic minimization problem. In fact, the heuristic construction of $H_0^1(\Omega)$ and $H^1(\Omega)$ given above fits this technique.

Theorem 1.3.4.11. Sobolev spaces by completion

For domains as described in § 1.2.1.14 the function space $H^1(\Omega)$ can be obtained through completion (\rightarrow Thm. 1.3.3.11) of $C^\infty(\overline{\Omega})$ equipped with the norm $\|\cdot\|_{H^1(\Omega)}$.

For any domain $\Omega \subset \mathbb{R}^d$, the space $H_0^1(\Omega)$ arises from the completion of $C_0^\infty(\Omega)$ under the norm $\|\cdot\|_{H^1(\Omega)}$.

As a consequence, the spaces of smooth functions $C^\infty(\overline{\Omega})$ and $C_0^\infty(\Omega)$ are dense (\rightarrow Def. 1.3.3.12) in $H^1(\Omega)$ and $H_0^1(\Omega)$, respectively. \square

Remark 1.3.4.12 ($|\cdot|_{H^1(\Omega)}$ -seminorm) Note that $|\cdot|_{H^1(\Omega)}$ alone is no longer a norm on $H^1(\Omega)$, because for $v \equiv \text{const}$ obviously $|v|_{H^1(\Omega)} = 0$, which violates (N1) of Def. 0.3.1.10. This justifies the part $\|\cdot\|_0^2$ in the above definition of the norm $\|\cdot\|_{H^1(\Omega)}$.

Yet, $|\cdot|_{H^1(\Omega)}$ still satisfies (N2) and (N3) and qualifies as a **semi-norm** on $H^1(\Omega)$. \square

§1.3.4.13 (Quadratic minimization problems on $H^1(\Omega)$) Lemma 1.2.3.39 tells us that a quadratic functional with s.p.d. bilinear form a is bounded from below, if its linear form ℓ satisfies the continuity (1.2.3.40). Now, we discuss this for the quadratic functional J from (1.3.4.1) in lieu of J_M and J_E .

$$J(u) := \int_{\Omega} \frac{1}{2} \|\mathbf{grad} u\|^2 - f(x)u(x) \, dx \quad u \in H_0^1(\Omega) . \quad (1.3.4.1)$$

This quadratic functional J involves the linear form

$$\ell(u) := \int_{\Omega} f(x)u(x) \, dx . \quad (1.3.4.14)$$

The load function f need not be continuous, integrability is enough. Hence, $f \in C_{\text{pw}}^0(\Omega)$ should be admitted.

Crucial question: Is ℓ as given in (1.3.4.14) continuous on $H_0^1(\Omega)$?
 \Updownarrow (c.f. (1.2.3.40))

$$\exists C > 0: |\ell(u)| \leq C|u|_{H^1(\Omega)} \quad \forall u \in H_0^1(\Omega) ? .$$

(Again, recall Lemma 1.2.3.39 to appreciate the importance of this continuity: it is a necessary condition for the existence of a minimizer.)

To answer the question, we use the Cauchy-Schwarz inequality (CSI) of Thm. 0.3.1.19 in its special version for integrals

$$\left| \int_{\Omega} u(x)v(x) \, dx \right| \leq \left(\int_{\Omega} |u(x)|^2 \, dx \right)^{1/2} \left(\int_{\Omega} |v(x)|^2 \, dx \right)^{1/2} = \|u\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} \quad (1.3.4.15)$$

for all $u, v \in L^2(\Omega)$, which implies

$$|\ell(u)| = \left| \int_{\Omega} f(x) u(x) dx \right| \leq \left(\int_{\Omega} |f(x)|^2 dx \right)^{1/2} \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2} = \underbrace{\|f\|_0}_{<\infty} \|u\|_0. \quad (1.3.4.16)$$

This reduces the problem to bounding $\|u\|_0$ in terms of $|u|_{H^1(\Omega)}$.

Theorem 1.3.4.17. First Poincaré-Friedrichs inequality

If $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, is bounded, then

$$\|u\|_0 \leq \text{diam}(\Omega) \|\mathbf{grad} u\|_0 \quad \forall u \in H_0^1(\Omega).$$

Proof. The proof employs a powerful technique in the theoretical treatment of function spaces: exploit density of smooth functions (which, by itself, is a deep result).

It boils down to the insight:

In order to establish inequalities between continuous functionals on Sobolev spaces of functions on Ω it often suffices to show the target inequality for smooth functions in $C_0^\infty(\Omega)$ or $C^\infty(\Omega)$, respectively.

☞ notation: $C_0^\infty(\Omega) \hat{=} \text{smooth functions with (compact) support inside } \Omega$

In the concrete case (note the zero boundary values inherent in the definition of $H_0^1(\Omega)$) we have to establish the first Poincaré-Friedrichs inequality for functions $u \in C_0^\infty(\Omega)$ only.

1D: For the sake of lucidity the proof is first elaborated for $d = 1$, $\Omega = [0, 1]$. It merely employs elementary results from calculus throughout, namely the Cauchy-Schwarz inequality (1.3.4.15) and the fundamental theorem of calculus [Str09, Satz 6.3.4],

$$\int_a^b F'(x) dx = F(b) - F(a). \quad (1.3.4.18)$$

We infer that

$$\begin{aligned} \forall u \in C_0^\infty([0, 1]): \quad u(x) &= \underbrace{u(0)}_{=0} + \int_0^x \frac{du}{dx}(\tau) d\tau, \quad 0 \leq x \leq 1. \\ \Rightarrow \|u\|_0^2 &= \int_0^1 \left| \int_0^x \frac{du}{dx}(\tau) d\tau \right|^2 dx \stackrel{(1.3.4.15)}{\leq} \int_0^1 \left(\int_0^x 1 d\tau \cdot \int_0^x \left| \frac{du}{dx}(\tau) \right|^2 d\tau \right) dx \\ &\leq \int_0^1 \left(\int_0^1 1 d\tau \cdot \int_0^1 \left| \frac{du}{dx}(\tau) \right|^2 d\tau \right) dx \leq \int_0^1 \left| \frac{du}{dx}(\tau) \right|^2 d\tau = \left\| \frac{du}{dx} \right\|_0^2. \end{aligned}$$

Taking the square root finishes the proof in 1D.

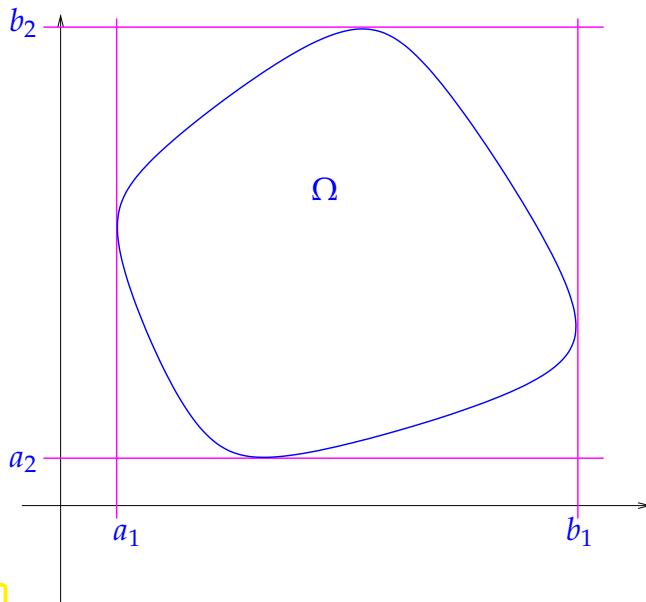


Fig. 43

Exploiting the density of smooth compactly supported functions in $H_0^1(\Omega)$, we pick $\tilde{u} \in C_0^\infty(\Omega)$ and define

$$\tilde{u}(x) = \begin{cases} u(x) & \text{for } x \in \Omega, \\ 0 & \text{outside } \Omega. \end{cases}$$

It goes without saying that $\tilde{u} \in C_0^\infty(\mathbb{R}^2)$. By the fundamental theorem of calculus (1.5.1.6) applied in x_1 -direction, we get

$$u\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \underbrace{u\left(\begin{bmatrix} a_1 \\ x_2 \end{bmatrix}\right)}_{=0} + \int_{a_1}^{x_1} \frac{\partial \tilde{u}}{\partial x_1}\left(\begin{bmatrix} \xi \\ x_2 \end{bmatrix}\right) d\xi \quad \forall x := \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2.$$

We apply this integral representation formula and then the Cauchy-Schwarz inequality (1.3.4.15):

$$\begin{aligned} \int_{\Omega} |u(x)|^2 dx &= \int_{\Omega} \left| \int_{a_1}^{x_1} \mathbf{1} \cdot \frac{\partial \tilde{u}}{\partial x_1}\left(\begin{bmatrix} \xi \\ x_2 \end{bmatrix}\right) d\xi \right|^2 dx \\ &\leq \int_{a_2}^{b_2} \int_{a_1}^{b_1} |x_1 - a_1| \int_{a_1}^{x_1} \left| \frac{\partial \tilde{u}}{\partial x_1}\left(\begin{bmatrix} \xi \\ x_2 \end{bmatrix}\right) \right|^2 d\xi dx_1 dx_2 \\ &\leq |b_1 - a_1| \int_{a_2}^{b_2} \int_{a_1}^{b_1} \int_{a_1}^{a_2} \|\mathbf{grad} u\left(\begin{bmatrix} \xi \\ x_2 \end{bmatrix}\right)\|^2 d\xi dx_1 dx_2 \\ &\leq |b_1 - a_1|^2 \int_{a_2}^{b_2} \int_{a_1}^{b_1} \|\mathbf{grad} u\left(\begin{bmatrix} \xi \\ x_2 \end{bmatrix}\right)\| d\xi dx_2 = |b_1 - a_1|^2 \int_{\Omega} \|\mathbf{grad} u(x)\|^2 dx. \end{aligned}$$

Thus, we have shown $\|u\|_{L^2(\Omega)} \leq |b_1 - a_1| \|u\|_{H^1(\Omega)}$.

The elementary proof in higher dimensions can be found in [Hac92, Sect. 6.2.2] and in even greater generality in [Eva98, Sect. 5.6.1]. \square

Corollary 1.3.4.19. Admissible loading/source functions linear 2nd-order elliptic problems

If $f \in L^2(\Omega)$, then $\ell(v) := \int_{\Omega} f(x)v(x) dx$ is a continuous linear functional on $H_0^1(\Omega)$.

As in Section 1.2.3.4 in this lemma ‘‘continuity’’ has to be read as

$$\exists C > 0: |\ell(u)| \leq C \|u\|_{H^1(\Omega)} \quad \forall u \in H_0^1(\Omega). \quad (1.2.3.40)$$

How to “work with” Sobolev spaces

Most concrete results about Sobolev spaces boil down to relationships between their norms. The spaces themselves remain intangible, but the norms are very concrete and can be computed and manipulated as demonstrated above.

Do not be afraid of Sobolev spaces!

It is only the norms that matter for us, the ‘spaces’ are irrelevant!

Sobolev spaces = “concept of convenience”: the minimization problem seeks its own function space.

Minimization problem

$$\underline{u} = \operatorname{argmin}_{v: \Omega \rightarrow \mathbb{R}} J(v)$$

“Maximal” function space
on which J is defined
(Sobolev space)

“seek” \leftrightarrow in more rigorous terms: completion with respect to energy norm, see § 1.3.3.10. □

Remark 1.3.4.21 (Justification for teaching Sobolev spaces) Then, why do you bother me with these uncanny “Sobolev spaces” after all ?

- ◆ Anyone involved in CSE must be able to understand mathematical publications on numerical methods for PDEs. Those regularly resort to the concept of Sobolev spaces to express their findings
We will also need the following spaces (see [19]):

$$V_0 = H_{0,\Gamma_\tau}(\operatorname{curl}^0; \Omega) = \left\{ \underline{v} \in V \mid \operatorname{curl} \underline{v} = 0 \right\} \quad (3)$$

$$H_{0,\Gamma_\nu}(\operatorname{div}^0, \Omega, \epsilon) = \left\{ \underline{v} \in L^2(\Omega)^3 \mid \operatorname{div} \epsilon \underline{v} = 0, \epsilon \underline{v} \cdot \underline{n}|_{\Gamma_\nu} = 0 \right\} \quad (4)$$

$$H_1 = \epsilon^{-1} \operatorname{curl}(H_{0,\Gamma_\nu}(\operatorname{curl}, \Omega)) \subset H_{0,\Gamma_\nu}(\operatorname{div}^0, \Omega, \epsilon) \quad (5)$$

$$V_1 = V \cap H_1 \quad (6)$$

$$\mathbb{H} = \mathbb{H}(\Omega, \Gamma_\tau, \epsilon) = H_{0,\Gamma_\tau}(\operatorname{curl}^0, \Omega) \cap H_{0,\Gamma_\nu}(\operatorname{div}^0, \Omega, \epsilon) \quad (7)$$

$$H_{0,\Gamma_\tau}^1(\Omega) = \left\{ \phi \in L^2(\Omega) \mid \operatorname{grad} \phi \in L^2(\Omega)^3, \phi|_{\Gamma_\tau} = 0 \right\}. \quad (8)$$

We will indicate by $\|\cdot\|_{0,\Omega}$ the norm in H corresponding to $(\cdot, \cdot)_{0,\Omega}$, by $(\cdot, \cdot)_{\operatorname{curl},\Omega}$ and $\|\cdot\|_{\operatorname{curl},\Omega}$ the standard inner product and norm in $H(\operatorname{curl}, \Omega)$, respectively, and by $\|\cdot\|_{s,\Omega}$, $0 < s \leq 1$, the natural norm in $H^s(\Omega)$ or $H^s(\Omega)^3$. For $s = 1$ we will also use the natural seminorm $\|\cdot\|_{1,\Omega}$ [16]. Finally, we define the following inner products and norms in H and V :

Fig. 44

- ◆ The statement that a function belongs to a certain Sobolev space can be regarded as a concise way of describing quite a few of its essential properties.

The next § will elucidate the second point. □

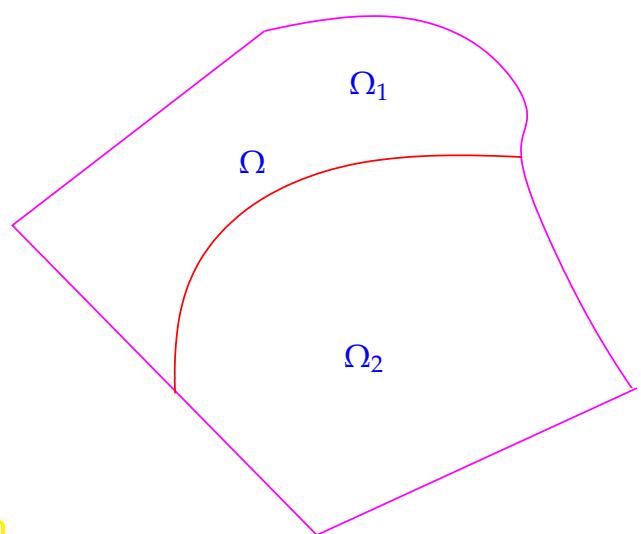
§1.3.4.22 (Built-in continuity of functions in $H^1(\Omega)$) For the elastic membrane models of Section 1.2.1 it was clear that the displacement functions must be continuous. In fact, this property is also built into the energy space $H^1(\Omega)$:

Theorem 1.3.4.23. Compatibility conditions for piecewise smooth functions in $H^1(\Omega)$

Let Ω be partitioned into sub-domains Ω_1 and Ω_2 . A function u that is continuously differentiable in the closures (*) of both sub-domains, belongs to $H^1(\Omega)$, if and only if u is continuous on Ω .

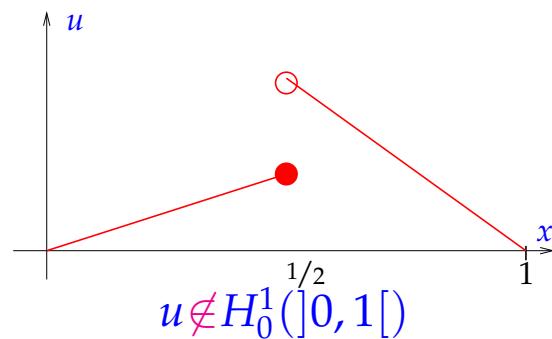
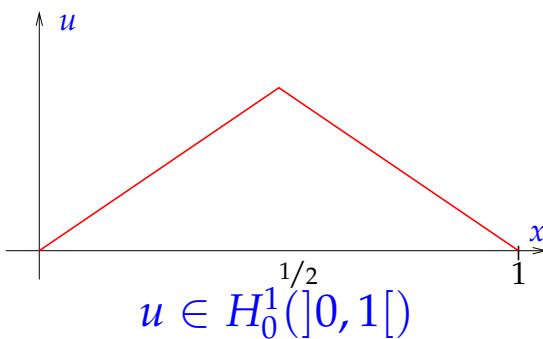
- (*) means existence of continuous derivatives on $\overline{\Omega}_1$ and $\overline{\Omega}_2$, respectively, that is, “up to the boundary of the sub-domains”, cf. Rem. 1.2.1.13.

Fig. 45



The proof of this theorem requires the notion of *weak derivatives* that will not be introduced in this course. \square

EXAMPLE 1.3.4.24 (Piecewise linear functions (not) in $H_0^1([0,1])$) We conclude from Thm. 1.3.4.23 applied in 1D that a “tent function” belongs to H^1 , whereas jumps are not allowed for functions in this space:



§1.3.4.25 (Piecewise smooth functions contained in Sobolev space) From Thm. 1.3.4.23 we conclude that the function spaces we opted for in Section 1.2.1.3 were not far off:

- $C_{\text{pw}}^1([a, b]) \subset H^1([a, b])$ and $C_{\text{pw},0}^1([a, b]) \subset H_0^1([a, b])$,
- but $C_{\text{pw}}^0([a, b]) \not\subset H^1([a, b])$.

On more general domains $\Omega \subset \mathbb{R}^d$ still holds true

- $C_{\text{pw}}^1(\overline{\Omega}) \subset H^1(\Omega)$ and $C_{\text{pw},0}^1(\overline{\Omega}) \subset H_0^1(\Omega)$,
- but $C_{\text{pw}}^0(\overline{\Omega}) \not\subset H^1(\Omega)$.

If you are feeling uneasy when dealing with Sobolev spaces, do not hesitate to resort to the following “mental substitutions”:

$$L^2(\Omega) \rightarrow C_{\text{pw}}^0(\Omega), \quad H_0^1(\Omega) \rightarrow C_{\text{pw},0}^1(\Omega).$$

Thm. 1.3.4.23 also provides a simple recipe for computing the norm $|u|_{H^1(\Omega)}$ of a piecewise C^1 -function that is continuous in all of Ω .

Corollary 1.3.4.26. H^1 -norm of piecewise smooth functions

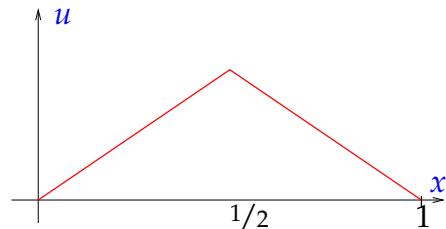
Under the assumptions of Thm. 1.3.4.23 we have for a continuous, piecewise smooth function $u \in C^0(\Omega)$

$$|u|_{H^1(\Omega)}^2 = |u|_{H^1(\Omega_1)}^2 + |u|_{H^1(\Omega_2)}^2 = \int_{\Omega_1} |\mathbf{grad} u(x)|^2 dx + \int_{\Omega_2} |\mathbf{grad} u(x)|^2 dx .$$

Thus, if $u \in C_{pw}^1(\Omega)$, we can evaluate all energy functionals from (1.2.3.1) by ‘piecewise differentiation’ followed by integration of the resulting discontinuous function.

EXAMPLE 1.3.4.27 (Non-differentiable function in $H_0^1([0,1])$) We demonstrate the computation of the norm $|\cdot|_{H^1(\Omega)}$ for the non-smooth function in Ex. 1.3.4.24, which is an example for a $u \in H_0^1(\Omega)[0,1]$, which is not globally differentiable. $d = 1, \Omega =]0, 1[$:

“Tent function” $u(x) = \begin{cases} 2x & \text{for } 0 < x < 1/2, \\ 2(1-x) & \text{for } 1/2 < x < 1 . \end{cases}$



Compute

$$|u|_{H^1(\Omega)}^2 = \int_0^{\frac{1}{2}} |u'(x)|^2 dx + \int_{\frac{1}{2}}^1 |u'(x)|^2 dx = 4 < \infty .$$

Cor. 1.3.4.26 makes it possible to talk about the gradient of a function, which is only piecewise continuously differentiable.

Definition 1.3.4.28. Weak/generalized gradient

The **weak/generalized gradient** of a function $u \in C_{pw}^1(\bar{\Omega})$, still denoted by $\mathbf{grad} u$, is a function in $C_{pw}^0(\bar{\Omega}) \subset L^2(\Omega)$, which agrees with the classical gradient of u , wherever the function is differentiable.

Note the point values of the weak/generalized gradient $\mathbf{grad} u$ are not defined at kinks of $u \in C_{pw}^1(\bar{\Omega})$. This does not matter, because point values are not meaningful for functions in $L^2(\Omega)$ anyway.

Review question(s) 1.3.4.29 (Sobolev spaces)

(Q1.3.4.29.A) Which of the following functions belong to the spaces $L^2([-1,1])$ and $H^1([-1,1])$, respectively?

1. $f_1(x) = |x|$,
2. $f_2(x) = \log|x|$,
3. $f_3(x) = \operatorname{sgn}(x)$,
4. $f_4(x) = \sqrt{|x|+x}$.

(Q1.3.4.29.B) Explain the statement

For bounded domains $\Omega \subset \mathbb{R}^d$ the restriction of functions to the boundary $\partial\Omega$ makes sense in $H^1(\Omega)$

in terms of $H^1(\Omega)$ -norms of functions.

(Q1.3.4.29.C) Show that the point evaluation $v \mapsto v(\tfrac{1}{2})$ is an unbounded linear functional on $L^2(]0,1[)$.

(Q1.3.4.29.D) For a bounded domain $\Omega \subset \mathbb{R}^d$, $d = 1, 2$, which of the spaces $C_{\text{pw}}^0(\overline{\Omega})$, $C_{\text{pw}}^1(\overline{\Omega})$, and $C_{\text{pw}}^2(\overline{\Omega})$ is/is not contained in $L^2(\Omega)$ and $H^1(\Omega)$, respectively?

(Q1.3.4.29.E) Let $\Omega \subset \mathbb{R}^2$ be a bounded domain. Define the Sobolev space fitting the quadratic minimization problem for the functional

$$J(\mathbf{v}) := \int_{\Omega} |\operatorname{div} \mathbf{v}(x)|^2 + \|\mathbf{v}\|^2 dx, \quad \mathbf{v} = (C^1(\overline{\Omega}))^2.$$

(Q1.3.4.29.F) Which Sobolev space, call it W , fits minimization problems for the functional

$$J(v) := \int_{\Omega} |\mathbf{d} \cdot \operatorname{grad} u|^2 + u^2 dx, \quad v \in C^\infty(\overline{\Omega}),$$

where $\mathbf{d} \in \mathbb{R}^2$ is a fixed unit vector, and $\Omega =]0, 1[^2$.

- Give an example of a function belonging to W , but not to $H^1(\Omega)$.
- Show that $H^1(\Omega) \subset W$.

(Q1.3.4.29.G) [Square-root functions] For which $\alpha \in \mathbb{R}$ does the function $u_\alpha(x) := x^\alpha$, $0 < x < 1$, belong to $L^2(]0, 1[)$?

Hint. Since $u_\alpha \in C^\infty(]0, 1[)$ all you need to check is whether $\|u_\alpha\|_{L^2(]0, 1[)} < \infty$.

△

1.4 Linear Variational Problems

In this section we derive an **equivalent** reformulation for quadratic minimization problems (→ Def. 1.2.3.11) with symmetric positive definite bilinear forms.

EXAMPLE 1.4.0.1 (Recasting quadratic minimization problems on \mathbb{R}^N) As in Ex. 1.2.3.5 we consider a quadratic functional (→ Def. 1.2.3.2) on the vector space $V_0 := \mathbb{R}^N$, $N \in \mathbb{N}$:

$$J(\vec{\eta}) := \frac{1}{2} \vec{\eta}^\top \mathbf{A} \vec{\eta} - \vec{\beta}^\top \vec{\eta}, \quad \mathbf{A} = \mathbf{A}^\top \in \mathbb{R}^{N,N}, \quad \vec{\beta}, \vec{\eta} \in \mathbb{R}^N, \quad (1.4.0.2)$$

where the matrix \mathbf{A} is symmetric and positive definite (→ Def. 0.3.1.17).

Clearly, a quadratic functional like J is a smooth function $\mathbb{R}^N \rightarrow \mathbb{R}$. Thus from calculus we know that every minimizer of J must be a zero of its gradient:

$$\vec{\mu} \in \underset{\vec{\eta} \in \mathbb{R}^N}{\operatorname{argmin}} J(\vec{\eta}) \iff \operatorname{grad} J(\vec{\mu}) = \mathbf{0}.$$

Let us compute the gradient of J at $\vec{\mu} \in \mathbb{R}^N$ based on its definition

$$\operatorname{grad} J(\vec{\mu})^\top \vec{\eta} = \lim_{t \rightarrow 0} \frac{J(\vec{\mu} + t\vec{\eta}) - J(\vec{\mu})}{t} = \vec{\mu}^\top \mathbf{A} \vec{\eta} - \vec{\beta}^\top \vec{\eta},$$

where we used the symmetry of \mathbf{A} . We have arrived at

$$\mathbf{grad} J(\vec{\mu}) = \mathbf{A}\vec{\mu} - \vec{\beta} \in \mathbb{R}^N, \quad \vec{\mu} \in \mathbb{R}^N. \quad (1.4.0.3)$$

The zeros of $\mathbf{grad} J$ can be found by solving a linear system of equations (LSE):

$$\mathbf{grad} J(\vec{\mu}) = \mathbf{0} \iff \mathbf{A}\vec{\mu} = \vec{\beta}.$$

For s.p.d. \mathbf{A} the linear system of equations $\mathbf{A}\vec{\mu} = \vec{\beta}$ has a unique solution. In addition, Thm. 1.2.3.44 tells us that this J will possess a unique minimizer. We can finally conclude that for J from (1.4.0.2)

$$\vec{\mu} \in \underset{\vec{\eta} \in \mathbb{R}^N}{\operatorname{argmin}} J(\vec{\eta}) \iff \mathbf{A}\vec{\mu} = \vec{\beta}. \quad (1.4.0.4)$$

□

1.4.1 Quadratic Variational Calculus

Again abstracting the quadratic energy functionals summarized in § 1.2.3.7, we consider the quadratic functional (\rightarrow Def. 1.2.3.2)

$$J(v) := \frac{1}{2}a(v, v) - \ell(v) + c, \quad v \in V, \quad (1.2.3.3)$$

on a real vector space V , with a bilinear form $a : V \times V \rightarrow \mathbb{R}$ and a linear form $\ell : V \rightarrow \mathbb{R}$. In light of the theory developed in Section 1.2.3.2 and Section 1.2.3.4 we take for granted conditions that ensure existence and uniqueness of minimizers for J :

Assumption 1.4.1.1.

We assume that

- a is symmetric positive definite (\rightarrow Def. 0.3.1.16), and
- that (maybe only on a subspace $V_0 \subset V$) ℓ is bounded w.r.t. the energy norm $\|\cdot\|_a$ (\rightarrow Def. 1.2.3.35) induced by $a(\cdot, \cdot)$, cf. Lemma 1.2.3.39.

Inspired by the quadratic minimization problems of (1.2.3.1) we also consider a subspace $V_0 \subset V$ and, for some $g \in V$, the associated affine space $\widehat{V} := g + V_0$. Then we can state the (slightly generalized) quadratic minimization problem (\rightarrow Def. 1.2.3.11)

$$\text{Seek } u_* \in \widehat{V}: \quad u_* = \underset{v \in \widehat{V}}{\operatorname{argmin}} J(v). \quad (1.4.1.2)$$

In passing, we remark that existence and uniqueness of u_* is guaranteed provided that V_0 equipped with the norm $\|\cdot\|_a$ is a Hilbert space, see Rem. 1.3.3.9.

The following formal argument is at the heart of an area of mathematics known as variational calculus, which studies extremal problems on infinite-dimensional function spaces.

Assume that $u_* \in \widehat{V}$ solves (1.4.1.2) and, for arbitrary $v \in V_0$ define the function.

$$\varphi_v : \mathbb{R} \rightarrow \mathbb{R}, \quad \varphi_v(t) = J(u_* + tv), \quad t \in \mathbb{R}. \quad (1.4.1.3)$$

Invoking the bilinearity and symmetry of a and the linearity of ℓ , we easily find

$$\varphi_v(t) = \frac{1}{2}t^2a(v, v) + ta(u_*, v) + \frac{1}{2}a(u_*, u_*) - t\ell(v) - \ell(u_*), \quad t \in \mathbb{R}.$$

Note that φ_v is a quadratic polynomial, its graph a parabola. By definition of u_* the function φ_v has a global minimum at $t = 0$. Since φ_v is continuously differentiable, we conclude that

$$\frac{d\varphi_v}{dt}(0) = a(u_*, v) - \ell(v) = 0 . \quad (1.4.1.4)$$

This will hold for any $v \in V_0$ and we end up with a new form of equation, a **linear variational equation**,

$$u_* \text{ solves (1.4.1.2)} \Rightarrow a(u_*, v) = \ell(v) \quad \forall v \in V_0 . \quad (1.4.1.5)$$

Definition 1.4.1.6. (Generalized) Linear variational problem (LVP)

With V a vector space, $\widehat{V} \subset V$ an affine space (\rightarrow Def. 0.3.1.1), and $V_0 \subset V$ the associated subspace the equation

$$u \in \widehat{V}: a(u, v) = \ell(v) \quad \forall v \in V_0 , \quad (1.4.1.7)$$

is called a (generalized) **linear variational problem**, if

- $a : V \times V_0 \mapsto \mathbb{R}$ is a **bilinear form**, that is, linear in both arguments (\rightarrow Def. 0.3.1.4),
- and $\ell : V_0 \rightarrow \mathbb{R}$ is a linear form (\rightarrow Def. 0.3.1.3).

The attribute “generalized” has been added, because (1.4.1.7) is posed on an affine space. Strictly speaking, a “linear variational problem” should be posed on a vector space.

Terminology related to variational problems: \widehat{V} is called the **trial space**

V_0 is called the **test space**

A key result concerns the **equivalence** of (generalized) quadratic minimization problems and linear variational problems.

Theorem 1.4.1.8. Equivalence of quadratic minimization problem and linear variational problem

For a (generalized) quadratic functional $J(v) = \frac{1}{2}a(v, v) - \ell(v) + c$ on a vector space V and with a **symmetric positive definite** bilinear form $a : V \times V \rightarrow \mathbb{R}$ are equivalent:

- (i) The quadratic minimization problem for J has unique minimizer $u_* \in \widehat{V}$ over the affine subspace $\widehat{V} = g + V_0$, $g \in V$.
- (ii) The linear variational problem

$$u \in \widehat{V}: a(u, v) = \ell(v) \quad \forall v \in V_0 ,$$

has a unique solution $u_* \in \widehat{V}$.

Proof. The arguments for the direction (i) \Rightarrow (ii) have been given above.

To conclude (ii) \Rightarrow (i) first observe, that by the bilinearity of a

$$\begin{aligned} a(u_*, v) = \ell(v) \quad \forall v \in V_0 \Rightarrow J(w) - J(u_*) &= \frac{1}{2}a(w, w) - \ell(w) - \frac{1}{2}a(u_*, u_*) + \ell(u_*) \\ &= \frac{1}{2}a(w, w) - \ell(w - u_*) - \underbrace{\frac{1}{2}a(u_*, u_*)}_{\in V_0} \\ &= \frac{1}{2}a(w, w) - a(u_*, w - u_*) + \frac{1}{2}a(u_*, u_*) \\ &= \frac{1}{2}a(w - u_*, w - u_*) = \|w - u_*\|_a^2 , \end{aligned}$$

for every $w \in \widehat{W}$. Since, a is positive definite, this means that $J(w) - J(u_*) \geq 0$ for all $w \in \widehat{V}$: u_* is a global minimizer of J . \square

Graphical summary:

$$\boxed{\text{Quadratic minimization problem (1.4.1.2)}} \iff \boxed{\text{Linear variational problem (1.4.1.7)}}$$

§1.4.1.9 (Offset function technique for linear variational problems) We use the notations of Def. 1.4.1.6. If $\widehat{V} = g + V_0$ for some $g \in V$, then it is easy to convert the linear variational problem into an equivalent one for which trial space and test space coincide:

$$u_* \in \widehat{V}: \quad a(u_*, v) = \ell(v) \quad \forall v \in V_0 \iff \begin{aligned} u \in V_0: \quad a(u + g, v) &= \ell(v) \quad \forall v \in V_0 \\ u \in V_0: \quad a(u, v) &= \ell(v) - a(g, v) \quad \forall v \in V_0. \end{aligned}$$

We end up with a linear variational problem with modified linear functional $v \mapsto \ell(v) - a(g, v)$ posed on the trial and test space V_0 . Its solution $u \in V_0$ allows to recover u_* by adding the offset function $u_* = u + g$.

Recall that this transformation of a linear variational problem just reflects the transformation of a quadratic minimization problem presented in § 1.2.3.12. \square

1.4.2 Linear Second-Order Elliptic Variational Problems

Based on Thm. 1.4.1.8 and the results of § 1.2.3.7, it is straightforward to state the linear variational problems arising from the (generalized) quadratic minimization problems (1.2.3.1a)–(1.2.3.1c).

- For the elastic string model the minimization problem

$$u = \underset{\substack{v \in H^1([a,b]) \\ v(a)=u_a \ v(b)=u_b}}{\operatorname{argmin}} \int_a^b \frac{1}{2} \sigma(x) \left| \frac{dv}{dx}(x) \right|^2 - f(x)v(x) dx, \quad (1.3.4.9a)$$

gives rise to

$$u \in H^1([a,b]), \quad u(a) = u_a, \quad u(b) = u_b : \quad \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1([a,b]). \quad (1.4.2.1)$$

- For the 2D membrane model we can convert

$$u = \underset{\substack{v \in H^1(\Omega) \\ v=g \text{ on } \partial\Omega}}{\operatorname{argmin}} \int_{\Omega} \frac{1}{2} \sigma(x) \|\mathbf{grad} v(x)\|^2 - f(x)v(x) dx,$$

into the linear variational problem

$$u \in H^1(\Omega), \quad u = g \text{ on } \partial\Omega : \quad \int_{\Omega} \sigma(x) \mathbf{grad} u(x) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x)v(x) \quad \forall v \in H_0^1(\Omega). \quad (1.4.2.2)$$

- The electrostatic field problem

$$u = \underset{\substack{v \in H^1(\Omega) \\ v=U \text{ on } \partial\Omega}}{\operatorname{argmin}} \int_{\Omega} \frac{1}{2} (\epsilon(x) \mathbf{grad} v(x)) \cdot \mathbf{grad} v(x) dx, \quad (1.3.4.9c)$$

yields the linear variational problem

$$\begin{aligned} u &\in H^1(\Omega) , \\ u = U &\text{ on } \partial\Omega : \int_{\Omega} (\epsilon(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx = 0 \quad \forall v \in H_0^1(\Omega) . \end{aligned} \quad (1.4.2.3)$$

All the linear variational problems (1.4.2.1), (1.4.2.2), and (1.4.2.3) are structurally similar and fit the following form:

A generic **second-order linear elliptic Dirichlet problem** (*) in variational form seeks

$$\begin{aligned} u &\in H^1(\Omega) , \\ u = g &\text{ on } \partial\Omega : \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x)v(x) dx \quad \forall v \in H_0^1(\Omega) . \end{aligned} \quad (1.4.2.4)$$

Symmetric uniformly positive definite **material tensor** $\alpha : \Omega \mapsto \mathbb{R}^{d,d}$

(*) The attribute “Dirichlet” refers to a setting, in which the function u is prescribed on the entire boundary. This is a particular type of boundary condition, which will be studied in detail in Section 1.7.

Some more explanations and terminology:

- ◆ $\Omega \subset \mathbb{R}^d, d = 2, 3 \hat{=} \text{(spatial) domain, bounded, piecewise smooth boundary}$
- ◆ $g \in C^0(\partial\Omega) \hat{=} \text{boundary values (Dirichlet data)}$
- ◆ $f \in C_{\text{pw}}^0(\Omega) \hat{=} \text{loading function, source function}$
- ◆ $\alpha : \Omega \mapsto \mathbb{R}^{d,d} \hat{=} \text{material tensor, stiffness function, diffusion coefficient}$
(uniformly positive definite, bounded \rightarrow Def. 1.2.2.9):

$$\exists 0 < \alpha^- \leq \alpha^+ : \alpha^- \|z\|^2 \leq (\alpha(x)z) \cdot z \leq \alpha^+ \|z\|^2 \quad \forall z \in \mathbb{R}^d , \quad (1.4.2.5)$$

for almost all $x \in \Omega$.

Putting (1.4.2.4) into the abstract framework of Def. 1.4.1.6 we find

$$\begin{aligned} V_0 &= H_0^1(\Omega) , \quad \widehat{V} = \{v \in H^1(\Omega) : v|_{\partial\Omega} = g\} , \\ a(w, v) &= \int_{\Omega} (\alpha(x) \mathbf{grad} w(x)) \cdot \mathbf{grad} v(x) dx , \end{aligned} \quad (1.4.2.6)$$

$$\ell(v) = \int_{\Omega} f(x)v(x) dx . \quad (1.4.2.7)$$

1.4.3 Stability of Second-Order Elliptic Linear Variational Problems

“Stability” asks the question whether for a mathematical model small changes in the input data cause only small changes of the solution. This is a central concern, because data will usually have been obtained by measurement or some approximation and are inevitably tainted by “small” errors. If those triggered massive changes of the solution, the latter would not be useful.

We address “stability” for a generic second-order linear elliptic Dirichlet problem with a scalar-valued coefficient

$$\begin{aligned} u \in H^1(\Omega) : \quad & \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x) v(x) dx \quad \forall v \in H_0^1(\Omega) . \\ u = g \text{ on } \partial\Omega : \quad & \end{aligned}$$

The highlighted quantities qualify as **data**, namely

- the continuous function $g \in C^0(\partial\Omega)$ providing the (Dirichlet) boundary conditions,
- the uniformly positive and bounded coefficient function

$$\alpha : \Omega \rightarrow \mathbb{R} , \quad \exists 0 < \alpha^- \leq \alpha^+ : \quad \alpha^- \leq \alpha(x) \leq \alpha^+ \quad \text{for almost all } x \in \Omega , \quad (1.4.3.1)$$

- and the right-hand side source/loading function $f : \Omega \rightarrow \mathbb{R}$.

We use norms to give a meaning to “small”. Concretely, we use

- the **energy norm** (\rightarrow Def. 1.2.3.35) $\|v\|_a^2 := \int_{\Omega} \mathbf{grad} v^\top \alpha \mathbf{grad} v dx$ to measure changes of the solution,
- the $L^2(\Omega)$ -norm for right-hand side source/loading functions, as suggested by Cor. 1.3.4.19,
- and the **supremum norm**

$$\|\alpha\|_{L^\infty(\Omega)} := \sup_{x \in \Omega} \|\alpha(x)\|_2 , \quad (1.4.3.2)$$

for coefficient functions.

(The appropriate norm for boundary data is a delicate issue and will not be discussed here.)

The perturbed linear variational problem is

$$\begin{aligned} \tilde{u} \in H^1(\Omega) : \quad & \int_{\Omega} ((\alpha + \delta\alpha)(x) \mathbf{grad} \tilde{u}(x)) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} (f + \delta f)(x) v(x) dx \quad \forall v \in H_0^1(\Omega) , \\ \tilde{u} = g \text{ on } \partial\Omega : \quad & \end{aligned} \quad (1.4.3.3)$$

with “small” perturbation functions $\delta\alpha$ and $\delta f \in L^2(\Omega)$, satisfying

$$\|\delta\alpha\|_{L^\infty(\Omega)} \leq \frac{1}{2}\alpha^- . \quad (1.4.3.4)$$

This condition makes sure that the bilinear form in (1.4.3.3) is still positive definite. Writing $\|\cdot\|_a$ for the energy norm induced by the bilinear form of (1.4.3), the bound (1.4.3.4) also implies

$$\frac{1}{2}\|v\|_a^2 \leq \int_{\Omega} (\alpha + \delta\alpha)(x) \|\mathbf{grad} v\|^2 dx \leq \frac{3}{2}\|v\|_a^2 \quad \forall v \in H^1(\Omega) . \quad (1.4.3.5)$$

In a first step we subtract both variational problems, using the same test function $v \in H_0^1(\Omega)$ for both:

$$\begin{aligned} \int_{\Omega} (\alpha + \delta\alpha)(x) \mathbf{grad} \tilde{u}(x) \cdot \mathbf{grad} v(x) dx &= \int_{\Omega} (f + \delta f)(x) v(x) dx , \\ - \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx &= \int_{\Omega} f(x) v(x) dx , \\ \hline \int_{\Omega} (\alpha + \delta\alpha) \mathbf{grad} \tilde{u} \cdot \mathbf{grad} v dx + \int_{\Omega} \delta\alpha \mathbf{grad} u \cdot \mathbf{grad} v dx &= \int_{\Omega} \delta f v dx , \end{aligned}$$

where we wrote $\delta u := \tilde{u} - u \in H_0^1(\Omega)$ for the perturbation of the solution.



Idea:

Choose the special test function $v := \delta u$

$$\Rightarrow \int_{\Omega} (\alpha + \delta\alpha) \|\mathbf{grad} \delta u\|^2 dx = \int_{\Omega} \delta f \delta u dx - \int_{\Omega} \delta\alpha \mathbf{grad} u \cdot \mathbf{grad} \delta u dx .$$

Next, apply the estimate (1.4.3.5) to the left-hand side, which yields

$$\frac{1}{2} \|\delta u\|_a^2 \leq \left| \int_{\Omega} \delta f \delta u dx \right| + \left| \int_{\Omega} \delta\alpha \mathbf{grad} u \cdot \mathbf{grad} \delta u dx \right| .$$

Then use the **Cauchy-Schwarz inequality** for integrals

$$\int_{\Omega} u(x)v(x) dx \leq \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2} \left(\int_{\Omega} |v(x)|^2 dx \right)^{1/2} = \|u\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} \quad (1.3.4.15)$$

twice and pull $\delta\alpha$ out from the integral:

$$\frac{1}{2} \|\delta u\|_a^2 \leq \|\delta f\|_{L^2(\Omega)} \|\delta u\|_{L^2(\Omega)} + \|\delta\alpha\|_{L^\infty(\Omega)} |u|_{H^1(\Omega)} |\delta u|_{H^1(\Omega)} .$$

The $L^2(\Omega)$ -norm of δu can be dealt with by the Poincaré-Friedrichs estimate from Thm. 1.3.4.17:

$$\frac{1}{2} \|\delta u\|_a^2 \leq (\text{diam}(\Omega) \|\delta f\|_{L^2(\Omega)} + \|\delta\alpha\|_{L^\infty(\Omega)} |u|_{H^1(\Omega)}) |\delta u|_{H^1(\Omega)} .$$

Finally we use $|v|_{H^1(\Omega)}^2 \leq \frac{1}{\alpha^-} \|v\|_a^2$ from (1.4.3.5) and end up with

$$\|\delta u\|_a \leq \frac{2}{\sqrt{\alpha^-}} (\text{diam}(\Omega) \|\delta f\|_{L^2(\Omega)} + \|\delta\alpha\|_{L^\infty(\Omega)} |u|_{H^1(\Omega)}) . \quad (1.4.3.6)$$

This is stability in the chosen norms: the energy norm of the perturbation of the solution decreases proportional to the size (in suitable norms) of the perturbations in the data. In quantitative terms

$$\|\delta u\|_a \lesssim \max\{\|\delta f\|_{L^2(\Omega)}, \|\delta\alpha\|_{L^\infty(\Omega)}\} , \quad (1.4.3.7)$$

with \lesssim indicating constants independent of the perturbations.

Review question(s) 1.4.3.8 (Linear variational problems)

(Q1.4.3.8.A) State the **linear variational problem** equivalent to the minimization of

$$J : \mathbb{R}^2 \rightarrow \mathbb{R} , \quad J(x) := \|x\|_2^2 + x_1 + x_2 + 1 .$$

(Q1.4.3.8.B) Let V be a vector space, $a : V \times V \rightarrow \mathbb{R}$ a symmetric positive definite bilinear form and $\ell : V \rightarrow \mathbb{R}$ a linear form bounded w.r.t. to the energy norm $\|\cdot\|_a$ induced by $a(\cdot, \cdot)$. Assuming that the quadratic functional

$$J(v) := \frac{1}{2} a(v, v) - \ell(v) + c , \quad v \in V , \quad (1.2.3.3)$$

has a minimizer, what is the minimal value of J expressed

- in terms of $\|u\|_a$, or
- in terms of $\ell(u)$,

where $u \in V$ satisfies

$$a(u, v) = \ell(v) \quad \forall v \in V.$$

(Q1.4.3.8.C) For a bounded domain $\Omega \subset \mathbb{R}^2$ consider the second-order linear elliptic Dirichlet problem

$$u \in H_0^1(\Omega): \quad \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x)v(x) dx \quad \forall v \in H_0^1(\Omega).$$

with uniformly positive definite $\alpha : \Omega \rightarrow \mathbb{R}^{2,2}$.

1. For what source functions f will the right-hand side functional still be continuous on $H_0^1(\Omega)$? (Give a reasonably general sufficient condition.)
2. Let $u \in H_0^1(\Omega)$ be the solution of the above variational problem. For which norms introduced in Section 1.3 ($\square \in \{H^1(\Omega), L^2(\Omega)\}$) does the estimate

$$\|u\|_{\square} \leq C \|f\|_{\square}$$

hold with a constant $C > 0$ independent of f ?

△

1.5 Equilibrium Models: Boundary Value Problems

We have not seen a genuine partial differential equation yet. This is intended and meant to convey the crucial message that energy minimization over a Sobolev space is the essence of second-order elliptic boundary value problems, not the way, in which they can be written as a partial differential equation. Nevertheless, in this section we are going to extract (partial) differential equations from the linear variational problems obtained in Section 1.4.2.

1.5.1 Two-Point Boundary Value Problems

We revisit the pinned elastic string linear variational problem

$$u_* \in H^1([a, b]), \quad u(a) = u_a, \quad u(b) = u_b : \quad \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1([a, b]), \quad (1.4.2.1)$$

with a uniformly positive stiffness coefficient function $\sigma \in C_{pw}^0([a, b])$ and $f \in L^2([a, b])$.

\$1.5.1.1\$ (Extra smoothness requirements) The conditions $\sigma \in C_{pw}^0([a, b])$ and $f \in L^2([a, b])$ were the smoothness requirements sufficient for a well-defined variational formulation (1.4.2.1). However, in order to tease out a differential equation from (1.4.2.1) we have to demand **extra smoothness** of coefficients and solution:

Assumption 1.5.1.2. Smoothness required for two-point boundary value problems

The following smoothness requirements have to be satisfied

$$u \in C^2([a, b]), \quad \sigma \in C^1([a, b]), \quad f \in C^0([a, b]). \quad (1.5.1.3)$$

It is instructive to contrast (1.5.1.3) with the weaker smoothness requirements for the variational approach expressed by classical function spaces, cf. § 1.3.4.25. For (1.4.2.1) we merely demanded

$$u \in C_{pw}^1([a, b]), \quad \sigma \in C_{pw}^0([a, b]), \quad f \in C_{pw}^0([a, b]). \quad (1.5.1.4)$$

§1.5.1.5 (Integration by parts in 1D) The main tool to obtain a differential equation from (1.4.2.1) is **integration by parts** (IBP). Remember that integration by parts is a consequence of the fundamental theorem of calculus [Str09, Satz 6.3.4], which tells us that for $F \in C_{\text{pw}}^1([a, b])$, $a, b \in \mathbb{R}$,

$$\int_a^b F'(x) dx = F(b) - F(a), \quad (1.5.1.6)$$

where ' $'$ stands for differentiation w.r.t x . This formula is combined with the product rule [Str09, Satz 5.2.1 (ii)]

$$F(x) = f(x) \cdot g(x) \Rightarrow F'(x) = f'(x)g(x) + f(x)g'(x); \quad (1.5.1.7)$$

► $\int_a^b f'(x)g(x) + f(x)g'(x) dx = f(b)g(b) - f(a)g(a),$

which amounts to the integration by parts formula in 1D

$$\int_a^b f(\xi)v'(\xi) d\xi = - \int_a^b f'(\xi)v(\xi) d\xi + \underbrace{(f(b)v(b) - f(a)v(a))}_{\text{boundary terms}} \quad \forall f, v \in C_{\text{pw}}^1([a, b]). \quad (1.5.1.8)$$

Why “boundary terms”? Well, $\{a, b\}$ is the boundary of $[a, b]$.

§1.5.1.9 (A two-point boundary value problem by integration by parts in 1D) We first tackle the linear variational equation in one spatial dimension:

$$u \in H_0^1([a, b]): \quad \int_a^b u'(x)v'(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1([a, b]). \quad (1.5.1.10)$$

We invoke Ass. 1.5.1.2: Assuming $u \in C^2([a, b])$ as in Ass. 1.5.1.2, we resort to the one-dimensional integration by parts formula (1.5.1.8) on $[a, b]$ to establish

$$\int_a^b u'(x)v'(x) dx = \underbrace{u'(b)v(b) - u'(a)v(a)}_{=0, \text{ since } v(a)=v(b)=0} - \int_a^b u''(x)v(x) dx. \quad (1.5.1.11)$$

Thus, if $u \in C_{\text{pw}}^2([a, b])$ the linear variational problem (1.5.1.10) can equivalently be stated as

$$u \in C^2([a, b]): \quad - \int_a^b u''(x)v(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1([a, b]),$$

⇓

$$u \in C^2([a, b]): \quad \int_a^b (-u''(x) - f(x))v(x) dx = 0 \quad \forall v \in H_0^1([a, b]). \quad (1.5.1.12)$$

Again Ass. 1.5.1.2 will be important: Assuming $f \in C^0([a, b])$, we conclude that $-u'' - f \in C^0([a, b])$. The final step relies on the following mathematical result.

Lemma 1.5.1.13. fundamental lemma of the calculus of variations

Let $f \in C_{\text{pw}}^0([a, b])$, $-\infty < a < b < \infty$, satisfy

$$\int_a^b f(\xi)v(\xi) d\xi = 0 \quad \forall v \in C^\infty([a, b]), v(a) = v(b) = 0.$$

Then $f \equiv 0$.

As $H_0^1(]a, b[)$ contains all compactly supported smooth functions, (1.5.1.12) implies the **differential equation** $-u'' = f$ on $]a, b[$. The condition $u \in H_0^1(]a, b[)$ gives the boundary conditions $u(a) = u(b) = 0$. Eventually, we have found that a solution $u \in C^2([a, b])$ of (1.5.1.10) (for $f \in C^0([a, b])$) also solves the **two-point boundary value problem**

$$-u'' = f \quad \text{in }]a, b[\quad , \quad u(a) = u(b) = 0 . \quad (1.5.1.14)$$

In this case $d = 1$ the differential equation is an **ordinary differential equation**, which can be regarded as a “PDE in 1D”. \square

§1.5.1.15 (General second-order elliptic two-point Dirichlet problem) Under Ass. 1.5.1.2 the more general variational problem

$$\begin{aligned} & u_* \in H^1(]a, b[) , \\ & u(a) = u_a, \quad u(b) = u_b : \quad \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x) v(x) dx \quad \forall v \in H_0^1(]a, b[) , \end{aligned} \quad (1.4.2.1)$$

is amenable to the very same integration-by-parts trick. We pick $v \in C_0^1([a, b]) \subset H_0^1(]a, b[)$ and apply (1.5.1.8) on the left-hand side, observing that boundary terms can be dropped, because $v(a) = v(b) = 0$. This leads to

$$\int_a^b -\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) v(x) dx = \int_a^b f(x) v(x) dx \quad \Leftrightarrow \quad \int_a^b \underbrace{\left(f(x) + \frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) \right)}_{=0} v(x) dx = 0 .$$

This has to hold true for all $v \in C_0^1([a, b])$ and thanks to Lemma 1.5.1.13 we conclude that $f(x) + \frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = 0$. Taking into account the fixed values of u in the points a, b , we arrive at the **two-point boundary value problem**

$$-\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = f \quad \text{in }]a, b[\quad , \quad u(a) = u_a , \quad u(b) = u_b . \quad (1.5.1.16)$$

In mathematics, fixed values of the solution in the endpoint of the interval are usually referred to as **Dirichlet boundary conditions**. \square

1.5.2 Integration by parts in higher dimensions

The main objective of this section is to extend the extraction from linear variational problems of a differential equations to domains $\Omega \subset \mathbb{R}^d$, $d \geq 1$ (usually $d = 2, 3$). To that end, we employ integration by parts, but now we need integration by parts in higher dimensions! We try to follow the derivation in § 1.5.1.5.

Indeed, there is a **product rule** in higher dimensions and it involves functions and **vector fields**, see [Str09, Sect. 7.2]

Lemma 1.5.2.1. General product rule

For all $\mathbf{j} \in (C^1(\overline{\Omega}))^d$, $v \in C^1(\overline{\Omega})$ holds

$$\operatorname{div}(\mathbf{j}v) = v \operatorname{div} \mathbf{j} + \mathbf{j} \cdot \operatorname{grad} v \quad \text{in every point of } \Omega . \quad (1.5.2.2)$$

Supplement 1.5.2.3 (Divergence operator, see also § 0.3.2.19) From § 0.3.2.19 recall the definition of the following first-order **differential operator**, see also [Str09, Def. 8.8.1]:

The divergence of a C^1 -vector field $\mathbf{j} = [f_1, \dots, f_d]^\top : \Omega \mapsto \mathbb{R}^d$ is

$$\operatorname{div} \mathbf{j}(\mathbf{x}) := \frac{\partial f_1}{\partial x_1}(\mathbf{x}) + \dots + \frac{\partial f_d}{\partial x_d}(\mathbf{x}), \quad \mathbf{x} \in \Omega.$$

A widely used “ ∇ -notation” for the divergence is: $(\nabla \cdot \mathbf{j})(\mathbf{x}) := \operatorname{div} \mathbf{j}(\mathbf{x})$.

The importance of the divergence for the mathematical modelling of flow fields will be explained in Chapter 12. \square

Proof. (of Lemma 1.5.2.1) The identity (1.5.2.2) follows from the definition of div and grad and the standard product rule for partial derivatives. \square

A truly fundamental result from differential geometry provides a multidimensional analogue of the fundamental theorem of calculus:

Theorem 1.5.2.4. Gauss' theorem → [Str09, Sect. 8.8]

With $\mathbf{n} : \partial\Omega \mapsto \mathbb{R}^d$ denoting the exterior unit normal vectorfield on $\partial\Omega$ and dS indicating integration over a surface, we have

$$\int_{\Omega} \operatorname{div} \mathbf{j}(\mathbf{x}) d\mathbf{x} = \int_{\partial\Omega} \mathbf{j}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) dS(\mathbf{x}) \quad \forall \mathbf{j} \in (C_{pw}^1(\bar{\Omega}))^d. \quad (1.5.2.5)$$

Note that in (1.5.2.5) the fact that the vector fields and its divergence are integrated allows to relax smoothness requirements and state the formula for piecewise smooth vectorfields.

Proof. ① By linearity and the possibility to renumber coordinate directions, it is sufficient to show Gauss' theorem for vector fields of the special form

$$\mathbf{j}(\mathbf{x}) = f(\mathbf{x}) \mathbf{e}_d, \quad f \in C^1(\bar{\Omega}), \quad \mathbf{e}_d \hat{=} d\text{-th coordinate vector}.$$

Then, writing $\mathbf{n}(\mathbf{x}) = [n_1(\mathbf{x}), \dots, n_d(\mathbf{x})]^\top$ for the exterior unit normal of Ω , (1.5.2.5) becomes

$$\int_{\Omega} \frac{\partial f}{\partial x_d}(\mathbf{x}) d\mathbf{x} = \int_{\partial\Omega} f(\mathbf{x}) n_d(\mathbf{x}) dS(\mathbf{x}). \quad (1.5.2.6)$$

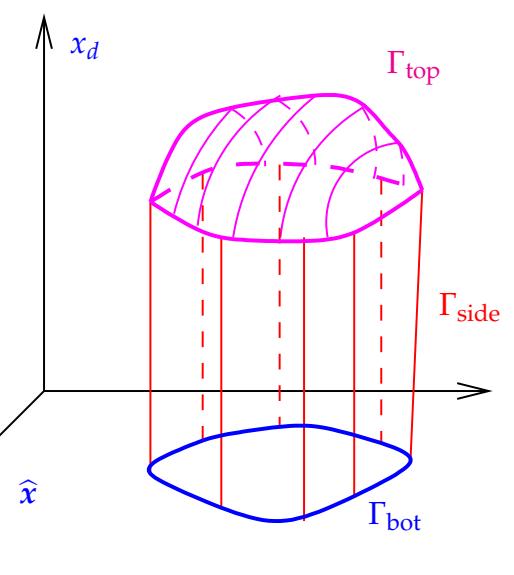
② Next, we assume that Ω is “graph-type”. It is the volume between the graph of a function and a coordinate plane:

$$\Omega = \left\{ \begin{bmatrix} \hat{\mathbf{x}} \\ x_d \end{bmatrix} \in \mathbb{R}^d : \hat{\mathbf{x}} \in \hat{\Omega}, 0 < x_d < g(\hat{\mathbf{x}}) \right\},$$

for a suitable cross-section domain $\hat{\Omega} \subset \mathbb{R}^{d-1}$ and a real-valued function $g \in C^1(\hat{\Omega})$, $g(\hat{\mathbf{x}}) \geq 0$ for all $\hat{\mathbf{x}} \in \hat{\Omega}$. The boundary of that domain is composed of three different parts:

$$\partial\Omega = \Gamma_{\text{top}} \cup \Gamma_{\text{bot}} \cup \Gamma_{\text{side}},$$

Fig. 46



$$\Gamma_{\text{top}} := \left\{ \begin{bmatrix} \hat{x} \\ g(\hat{x}) \end{bmatrix} : \hat{x} \in \hat{\Omega} \right\},$$

with $\Gamma_{\text{bot}} := \left\{ \begin{bmatrix} \hat{x} \\ 0 \end{bmatrix} : \hat{x} \in \hat{\Omega} \right\},$

$$\Gamma_{\text{side}} := \left\{ \begin{bmatrix} \hat{x} \\ x_d \end{bmatrix} : \hat{x} \in \partial\hat{\Omega}, 0 < x_d < g(\hat{x}) \right\}.$$

- (i) For all $x \in \Gamma_{\text{side}}$ we have $n_d(x) = 0$, because Γ_{side} is parallel to the x_d -direction. Thus, this part of the boundary does not contribute to the right-hand side of (1.5.2.6).
- (ii) The exterior unit normal vector at Γ_{bot} is $\begin{bmatrix} 0 \\ -1 \end{bmatrix}$, which means $n_d(x) = -1$ for $x \in \Gamma_{\text{bot}}$, so that

$$\int_{\Gamma_{\text{bot}}} f(x) n_d(x) dS(x) = - \int_{\hat{\Omega}} f\left(\begin{bmatrix} \hat{x} \\ 0 \end{bmatrix}\right) d\hat{x}.$$

- (iii) To elaborate the contribution of Γ_{top} we have to remember the formula for the surface integral of a function $\varphi : \Sigma \rightarrow \mathbb{R}$

$$\int_{\Sigma} \varphi(x) dS(x) := \int_{\hat{\Omega}} \varphi(\Phi(\hat{x})) \sqrt{\det(D\Phi(\hat{x})^T D\Phi(\hat{x}))} d\hat{x}, \quad (0.3.2.34)$$

whence $\Sigma = \Phi(\hat{\Omega})$ for a continuous function $\Phi : \hat{\Omega} \rightarrow \mathbb{R}^d$, the “parameterization” of Σ . In the present particular situation we have

$$\Phi(\hat{x}) := \begin{bmatrix} \hat{x} \\ g(\hat{x}) \end{bmatrix}, \quad \hat{x} \in \hat{\Omega},$$

with $D\Phi(\hat{x}) = \begin{bmatrix} I_{d-1} \\ \text{grad } g(\hat{x})^T \end{bmatrix} \in \mathbb{R}^{d,d-1}, \quad \hat{x} \in \hat{\Omega},$

► $D\Phi(\hat{x})^T D\Phi(\hat{x}) = I_{d-1} + \text{grad } g(\hat{x}) \text{ grad } g(\hat{x})^T,$

► $\det(D\Phi(\hat{x})^T D\Phi(\hat{x})) = 1 + \|\text{grad } g(\hat{x})\|_2^2.$

The exterior normal vector at Γ_{top} can most easily be computed by switching to an implicit representation

$$\Gamma_{\text{top}} := \left\{ x = \begin{bmatrix} \hat{x} \\ x_d \end{bmatrix} \in \mathbb{R}^d : F(x) := g(\hat{x}) - x_d = 0, \hat{x} \in \hat{\Omega} \right\}.$$

Then $-\text{grad } F(x)$ is parallel to $n(x)$ for $x \in \Gamma_{\text{top}}$:

$$\begin{aligned} n(x) \parallel \text{grad } F(x) &= \begin{bmatrix} \text{grad } g(\hat{x}) \\ -1 \end{bmatrix}, \quad x = \begin{bmatrix} \hat{x} \\ g(\hat{x}) \end{bmatrix}, \quad \hat{x} \in \hat{\Omega} \\ \Rightarrow n(x) &= \frac{1}{\sqrt{1 + \|\text{grad } g(\hat{x})\|_2^2}} \begin{bmatrix} -\text{grad } g(\hat{x}) \\ 1 \end{bmatrix}. \end{aligned}$$

Combining all these formulas yields a “lucky cancellation”,

$$\begin{aligned} \int_{\Gamma_{\text{top}}} f(x) n_d(x) dS(x) &= \int_{\hat{\Omega}} f\left(\begin{bmatrix} \hat{x} \\ g(\hat{x}) \end{bmatrix}\right) \frac{1}{\sqrt{1 + \|\text{grad } g(\hat{x})\|_2^2}} \sqrt{1 + \|\text{grad } g(\hat{x})\|_2^2} d\hat{x} \\ &= \int_{\hat{\Omega}} f\left(\begin{bmatrix} \hat{x} \\ g(\hat{x}) \end{bmatrix}\right) d\hat{x}. \end{aligned}$$

Adding all contributions we get

$$\begin{aligned} \int_{\partial\Omega} f(\mathbf{x}) n_d(\mathbf{x}) dS(\mathbf{x}) &= \int_{\Gamma_{\text{top}}} f(\mathbf{x}) n_d(\mathbf{x}) dS(\mathbf{x}) + \int_{\Gamma_{\text{bot}}} f(\mathbf{x}) n_d(\mathbf{x}) dS(\mathbf{x}) + \left[\int_{\Gamma_{\text{side}}} f(\mathbf{x}) n_d(\mathbf{x}) dS(\mathbf{x}) \right] \\ &= \int_{\widehat{\Omega}} f\left(\begin{bmatrix} \widehat{\mathbf{x}} \\ g(\widehat{\mathbf{x}}) \end{bmatrix}\right) - f\left(\begin{bmatrix} \widehat{\mathbf{x}} \\ 0 \end{bmatrix}\right) d\widehat{\mathbf{x}}. \end{aligned}$$

Next, apply the fundamental theorem of calculus (1.5.1.6) in \mathbf{x}_d -direction:

$$\int_{\partial\Omega} f(\mathbf{x}) n_d(\mathbf{x}) dS(\mathbf{x}) = \int_{\widehat{\Omega}} \int_0^{g(\widehat{\mathbf{x}})} \frac{\partial f}{\partial x_d} \left(\begin{bmatrix} \widehat{\mathbf{x}} \\ x_d \end{bmatrix} \right) dx_d d\widehat{\mathbf{x}} = \int_{\Omega} \frac{\partial f}{\partial x_d}(x) dx,$$

and this proves Gauss' theorem for the special “graph-type” domain Ω .

③ Assume that a generic domain $\Omega \subset \mathbb{R}^d$ is partitioned into $N \in \mathbb{N}$ subdomains,

$$\overline{\Omega} = \overline{\Omega}_1 \cup \overline{\Omega}_2 \cup \dots \cup \overline{\Omega}_N , \quad \Omega_\ell \cap \Omega_k = \emptyset \quad \text{for } \ell \neq k.$$

Write $\mathbf{n}^i : \partial\Omega_i \rightarrow \mathbb{R}^d$ for the exterior unit vector field of Ω_i . We point out that on $\partial\Omega_\ell \cap \partial\Omega_k \neq \emptyset$ the two involved exterior unit normals have opposite orientation: $\mathbf{n}^\ell = -\mathbf{n}^k$. Thus we find a cancellation of interior normal fluxes

$$\sum_{\ell=1}^N \int_{\partial\Omega_\ell} \mathbf{j}(x) \cdot \mathbf{n}^\ell(x) dS(x) = \int_{\partial\Omega} \mathbf{j}(x) \cdot \mathbf{n}(x) dS(x) \quad \text{for } \mathbf{j} \in (C^0(\overline{\Omega}))^d.$$

It goes without saying that every “reasonable” domain $\Omega \subset \mathbb{R}^d$ can be partitioned into finitely many small “graph-type” sub-domains $\Omega_1, \dots, \Omega_N$, for which we have already established Gauss’ theorem. Hence,

$$\int_{\Omega} \operatorname{div} \mathbf{j}(x) dx = \sum_{\ell=1}^N \int_{\Omega_\ell} \operatorname{div} \mathbf{j}(x) dx \stackrel{\textcircled{2}}{=} \sum_{\ell=1}^N \int_{\partial\Omega_\ell} \mathbf{j}(x) \cdot \mathbf{n}(x) dS(x) = \int_{\partial\Omega} \mathbf{j}(x) \cdot \mathbf{n}(x) dS(x).$$

This finishes the proof. □

Theorem 1.5.2.7. Green’s first formula

For all vector fields $\mathbf{j} \in (C_{\text{pw}}^1(\overline{\Omega}))^d$ and functions $v \in C_{\text{pw}}^1(\overline{\Omega})$ holds

$$\int_{\Omega} \mathbf{j} \cdot \operatorname{grad} v dx = - \int_{\Omega} \operatorname{div} \mathbf{j} v dx + \int_{\partial\Omega} \mathbf{j} \cdot \mathbf{n} v dS. \quad (1.5.2.8)$$

Note that the dependence on the integration variable \mathbf{x} is suppressed in the formula (1.5.2.8) to achieve a more compact notation. The first Green formula could also have been written as

$$\int_{\Omega} \mathbf{j}(x) \cdot (\operatorname{grad} v)(x) dx = - \int_{\Omega} (\operatorname{div} \mathbf{j})(x) v(x) dx + \int_{\partial\Omega} \mathbf{j}(x) \cdot \mathbf{n}(x) v(x) dS(x). \quad (1.5.2.8)$$

Proof. (of Thm. 1.5.2.7) The assertions follows straightforwardly from Lemma 1.5.2.1 and Thm. 1.5.2.4. □

1.5.3 Linear Scalar Second-order Elliptic Partial Differential Equations

Now we apply Green's first formula to the variational problem

$$\begin{aligned} u \in H^1(\Omega) , \\ u = g \text{ on } \partial\Omega : \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x) v(x) dx \quad \forall v \in H_0^1(\Omega) , \end{aligned} \quad (1.4.2.4)$$

posed on a domain $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, which covers the membrane model and electrostatics. As before, it will be crucial to assume extra smoothness of coefficient, source and solution functions.

Assumption 1.5.3.1. Extra smoothness requirements

We assume that the coefficient, source and solution functions in (1.4.2.4) satisfy

$$\alpha \in (C_{pw}^1(\bar{\Omega}))^{d,d} , \quad f \in C_{pw}^0(\bar{\Omega}) , \quad u \in C_{pw}^2(\bar{\Omega}) . \quad (1.5.3.2)$$

Apply integration by parts through Green's first formula (1.5.2.8) to the left-hand side of (1.4.2.4), where the role of \mathbf{j} is played by the vector field $\alpha \mathbf{grad} u : \Omega \mapsto \mathbb{R}^d$.

$$\begin{aligned} & \int_{\Omega} \underbrace{\alpha(x) \mathbf{grad} u(x)}_{=: \mathbf{j}(x)} \cdot \mathbf{grad} v(x) dx \\ &= - \int_{\Omega} \operatorname{div}(\alpha(x) \mathbf{grad} u(x)) v(x) dx + \int_{\partial\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot n(x) v(x) dS(x) . \end{aligned}$$

Plug this into the variational equation (1.4.2.4):

$$\begin{aligned} (1.4.2.4) \quad \Rightarrow \quad & - \int_{\Omega} \operatorname{div}(\alpha(x) \mathbf{grad} u(x)) v(x) dx \\ &+ \underbrace{\int_{\partial\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot n(x) v(x) dS(x)}_{=0, \text{ since } v|_{\partial\Omega}=0} \\ &= \int_{\Omega} f(x) v(x) dx \quad \forall v \in C_{pw,0}^1(\bar{\Omega}) , \end{aligned} \quad (1.5.3.3)$$

where we appeal to Ass. 1.5.3.1, because for these manipulations we need that u and α are sufficiently smooth so that $\alpha \mathbf{grad} u \in C_{pw}^1(\bar{\Omega})$. Moving everything to one side we get

$$\int_{\Omega} (\operatorname{div}(\alpha(x) \mathbf{grad} u(x)) + f(x)) v(x) dx = 0 \quad \forall v \in C_{pw,0}^1(\bar{\Omega}) .$$

Now we can invoke the multidimensional analogue of the fundamental lemma of the calculus of variations, see Lemma 1.5.1.13.

Lemma 1.5.3.4. Fundamental lemma of calculus of variations in higher dimensions

If $f \in L^2(\Omega)$ satisfies

$$\int_{\Omega} f(x) v(x) dx = 0 \quad \forall v \in C_0^\infty(\Omega) ,$$

then $f \equiv 0$ can be concluded.

This permits us to obtain a partial differential equation from (1.5.3.3):

$$(1.4.2.4) \quad \alpha \mathbf{grad} u \in C^1(\Omega)$$

Partial differential equations (PDE)

$$-\operatorname{div}(\alpha(x) \mathbf{grad} u) = f \quad \text{in } \Omega . \quad (1.5.3.5)$$

Again, for the sake of brevity, the dependence of $\mathbf{grad} u = \mathbf{grad} u(x)$, $f = f(x)$ on x is not made explicit in the PDE in (1.5.3.5).

Remark 1.5.3.6 (Laplace operator) If α agrees with a positive *constant*, by rescaling of (1.6.0.9) we can achieve

$$-\Delta u = f \quad \text{in } \Omega . \quad (1.5.3.7)$$

$$\Delta = \operatorname{div} \circ \mathbf{grad} = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2} = \text{Laplace operator}$$

Terminology: (1.5.3.7) is called **Poisson equation**, $\Delta u = 0$ in Ω is called **Laplace equation**

Finally:

PDE (1.5.3.5) + boundary conditions

$$-\operatorname{div}(\alpha(x) \mathbf{grad} u) = f \quad \text{in } \Omega , \quad u = g \quad \text{on } \partial\Omega . \quad (1.5.3.8)$$

(1.5.3.8) = second-order elliptic BVP with **Dirichlet boundary conditions** Short name for BVPs of the type (1.5.3.8): “Dirichlet problem”

Remark 1.5.3.9 (Extra smoothness requirement for PDE formulation) Same situation as in § 1.5.1.9, cf. Ass. 1.5.3.1.

The transition from a variational equation to PDE requires extra assumptions on smoothness of solution and coefficients.

For instance, in the case of (1.5.3.5) we demand $\operatorname{div}(\alpha(x) \mathbf{grad} u) \in C^0(\overline{\Omega})$, which is an implicit smoothness requirement for u , provided that the smoothness of the coefficient σ is known.

Terminology: A function $u \in C^1(\overline{\Omega})$, for which the partial differential equation (1.5.3.5) holds pointwise in $\overline{\Omega}$ and all derivatives exist in the sense of classical analysis, is called a **classical solution**.

Remark 1.5.3.10 (Equivalent formulations of elliptic BVPs) With (1.5.3.8) we have found a third way to state a second order elliptic boundary value problem, beside the quadratic minimization problem and the linear variational problem (1.4.2.4).

Minimization problem
(1.4.2)

$$u_* = \underset{v \in V}{\operatorname{argmin}} J(v)$$

Variational problem
(1.4.2.4)

$$a(u, v) = \ell(v) \quad \forall v$$

BVP for PDE (1.5.3.8)

$$\begin{aligned} \mathcal{L}u &= f \quad \text{in } \Omega , \\ u|_{\partial\Omega} &= g . \end{aligned}$$

- ❶ Strict equivalence of a minimization problem for a quadratic functional and linear variational problem, see Thm. 1.4.1.8.
- ❷ Statement as a boundary value problem for a (partial) differential equations hinges on extra smoothness, see Rem. 1.5.3.9 above.

Terminology: $\left\{ \begin{array}{l} \text{the minimization problem} \\ \Updownarrow \\ \text{the variational problem} \end{array} \right\}$ are called the **weak form** of the elliptic BVP,
 The statement (1.5.3.8) is called the **strong form** of the elliptic BVP

A solution u of (1.5.3.8), for which all occurring derivatives are continuous and which satisfies the PDE and the boundary conditions pointwise is called a **classical solution** of the elliptic BVP.

Below, Thm. 1.8.0.9 will confirm our expectations: classical solutions of a boundary value problem will also be weak solutions. \square

EXAMPLE 1.5.3.11 (Taut membrane with free boundary values) Above we studied the case of a membrane attached to a frame on all sides. Now let us consider frames with gaps, where the membrane has a “free boundary”. In this setting we explore the entire passage from a minimization problem up to a boundary value problem for a PDE.

Again we use the graph description of a taut membrane’s shape from § 1.2.1.8 through a function $u : \Omega \mapsto \mathbb{R}$, see § 1.2.1.8.

But now the membrane is clamped only on a part $\Gamma_0 \subset \partial\Omega$ of its edge.

- : prescribed boundary values here (Γ_0)
- : “free boundary” (gap of frame)

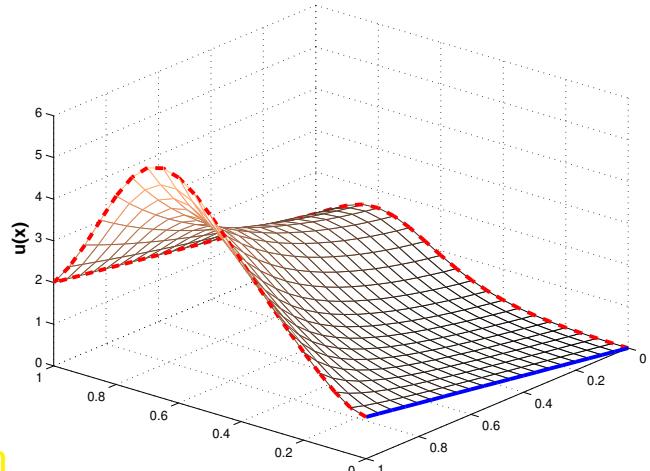


Fig. 47

► Configuration space $\widehat{V} := \{u \in H^1(\Omega) : u|_{\Gamma_0} = g\} \rightarrow$ Def. 1.3.4.8

The expression for the total potential energy remains the same as in (1.2.1.19):

$$J_M(u) := \int_{\Omega} \frac{1}{2} \sigma(x) \|\mathbf{grad} u\|^2 - f(x)u(x) dx . \quad (1.2.1.19)$$

Next we derive the variational formulation for the quadratic minimization problem for J_M . The only change compared to Section 1.4.1 concerns a modified test and trial space. The trial space is given by \widehat{V} , the test space is the subspace associated with this affine space:

► test space in variational formulation

$$V_0 := \{u \in H^1(\Omega) : u|_{\Gamma_0} = 0\}$$

► Variational formulation, c.f. (1.4.2.2)

$$u \in H^1(\Omega) : \int_{\Omega} \sigma(x) \mathbf{grad} u(x) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x)v(x) \quad \forall v \in V_0 . \quad (1.5.3.12)$$

$$u = g \text{ on } \Gamma_0$$

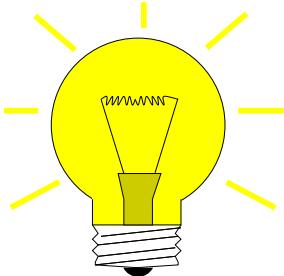
Our goal is to extract a second-order boundary value problem from this variational formulation. To begin with, an application of Green's first formula (1.5.2.8) to (1.5.3.12) leads to

$$\begin{aligned} - \int_{\Omega} (\operatorname{div}(\sigma(x) \operatorname{grad} u(x)) + f(x)) v(x) dx \\ + \int_{\partial\Omega \setminus \Gamma_0} ((\sigma(x) \operatorname{grad} u(x)) \cdot n(x)) v(x) dS(x) = 0 \quad \forall v \in V_0 . \end{aligned} \quad (1.5.3.13)$$

Note that, unlike in (1.5.3.3), the boundary integral term cannot be dropped entirely, because the test function v need not vanish on all of $\partial\Omega$: $v \neq 0$ on $\partial\Omega \setminus \Gamma_0$ is possible!

In the sequel we assume (\rightarrow Rem. 1.5.3.9) extra smoothness $u \in C_{\text{pw}}^2(\Omega)$, $\sigma \in C_{\text{pw}}^1(\Omega)$

How to deal with the boundary term in (1.5.3.13) ?



(Note that test functions in Lemma 1.5.3.4 vanish on $\partial\Omega$.)

Idea:

① First restrict test function v to $C_0^\infty(\Omega) \subset V$

\Rightarrow Boundary term vanishes !

Then, apply Lemma 1.5.3.4.

► PDE: $\operatorname{div}(\sigma(x) \operatorname{grad} u(x)) + f(x) = 0 \quad \text{in } \Omega .$ (1.5.3.14)

② Then test (1.5.3.13) with generic $v \in V_0$ and make use of (1.5.3.14). More precisely, plugging (1.5.3.14) into (1.5.3.13) makes the domain integral $\int_{\Omega} \dots$ disappear and only boundary terms remain.

$$\blacktriangleright \int_{\partial\Omega \setminus \Gamma_0} ((\sigma(x) \operatorname{grad} u(x)) \cdot n(x)) v(x) dS(x) = 0 \quad \forall v \in V_0 . \quad (1.5.3.15)$$

Thanks to the assumed smoothness of u and σ , $u \in C_{\text{pw}}^2(\Omega)$ and $\sigma \in C_{\text{pw}}^1(\Omega)$, we infer that $\{x \mapsto (\sigma(x) \operatorname{grad} u(x)) \cdot n(x)\} \in C^0(\partial\Omega)$. This paves the way for applying the following “boundary-based” variant of Lemma 1.5.3.4.

Lemma 1.5.3.16. Fundamental lemma of calculus of variations on boundaries

Let $\Omega \subset \mathbb{R}^d$ be a bounded domain with piecewise smooth boundary and $\Gamma_0 \subset \partial\Omega$ a part of $\partial\Omega$ with non-zero measure. If $g \in C^0(\partial\Omega)$ satisfies

$$\int_{\Gamma_0} g(x) v(x) dS(x) = 0 \quad \forall v \in C^\infty(\overline{\Omega}) ,$$

then g vanishes on Γ_0 : $g|_{\Gamma_0} \equiv 0$.

In combination with (1.5.3.15) this lemma gives boundary conditions on Γ_0 :

$$(1.5.3.15) \xrightarrow{\text{Lemma 1.5.3.16 on } \partial\Omega \setminus \Gamma_0} (\sigma(x) \operatorname{grad} u(x)) \cdot n(x) = 0 \quad \text{on } \partial\Omega \setminus \Gamma_0 . \quad (1.5.3.17)$$

When removing pinning conditions on $\partial\Omega \setminus \Gamma_0$ the equilibrium conditions imply the (homogeneous) **Neumann boundary conditions** $(\sigma(x) \mathbf{grad} u(x)) \cdot \mathbf{n}(x) = 0$ on $\partial\Omega \setminus \Gamma_0$.

Boundary value problem (strong form) for membrane clamped at $\Gamma_0 \subset \partial\Omega$

$$-\operatorname{div}(\sigma(x) \mathbf{grad} u) = f \quad \text{in } \Omega, \quad \begin{aligned} u &= g && \text{on } \Gamma_0, \\ (\sigma(x) \mathbf{grad} u) \cdot \mathbf{n} &= 0 && \text{on } \partial\Omega \setminus \Gamma_0. \end{aligned} \quad (1.5.3.18)$$

(1.5.3.18) = Second-order elliptic BVP with Neumann boundary conditions on $\partial\Omega \setminus \Gamma_0$ Short name for BVPs of the type (1.5.3.18): “Mixed Neumann–Dirichlet problem”

Review question(s) 1.5.3.19 (Boundary Value Problems for Equilibrium Models)

(Q1.5.3.19.A) State **Gauss' theorem** for a vector field $\mathbf{j} \in (\mathcal{C}^1(\bar{\Omega}))^d$ on a domain $\Omega \subset \mathbb{R}^d$.

(Q1.5.3.19.B) What is meant by the **weak form** and the **strong form** of an elliptic boundary value problem?

(Q1.5.3.19.C) What 2-point boundary value problem corresponds to the variational problem

$$u_* \in H^1([a, b]), \quad u(a) = u_a, \quad u(b) = u_b : \quad \int_a^b \frac{du}{dx}(x) \frac{dv}{dx}(x) + u(x)v(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1([a, b]),$$

(Q1.5.3.19.D) State the 2-point boundary value problem satisfied by the solution of the variational equation

$$u \in H^1([0, 1]) : \quad \int_0^1 (1 + x^2) \frac{du}{dx}(x) \left(\frac{dv}{dx}(x) - v(x) \right) dx = v(0) \quad \forall v \in H^1([0, 1]).$$

(Q1.5.3.19.E) Give an example for a linear variational problem on $H_0^1([0, 1])$ with continuous and s.p.d. bilinear form and continuous right-hand side linear form whose solution will not be the solution of a two-point boundary value problem in the sense of classical calculus.

Hint. “in the sense of classical calculus” implies that the solution u should at least be continuously differentiable: $u \in \mathcal{C}^1([0, 1])$.

(Q1.5.3.19.F) State the boundary value problem satisfied by the solution of the following variational problem

$$u \in H^1(\Omega), \quad u = g \text{ on } \partial\Omega : \quad \int_{\Omega} \frac{\partial u}{\partial x_1}(x) \frac{\partial v}{\partial x_1} + 2 \frac{\partial u}{\partial x_2} \frac{\partial v}{\partial x_2} dx = 0 \quad \forall v \in H_0^1(\Omega).$$

(Q1.5.3.19.G) For a bounded domain $\Omega \subset \mathbb{R}^3$ state the second-order elliptic boundary value problem associated with the linear variational problem

$$u \in H^1(\Omega) : \quad \int_{\Omega} \mathbf{grad} u(x) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} v(x) dx + \int_{\partial\Omega} v(x) dS(x) \quad \forall v \in H^1(\Omega).$$

(Q1.5.3.19.H) We consider the variational problem

$$\mathbf{u} : \Omega \rightarrow \mathbb{R}^2 : \quad \int_{\Omega} \operatorname{div} \mathbf{u}(x) \operatorname{div} \mathbf{v}(x) + \mathbf{u}(x) \cdot \mathbf{v}(x) dx = \int_{\partial\Omega} \mathbf{v}(x) \cdot \mathbf{n}(x) dx \quad \forall \mathbf{v} : \Omega \rightarrow \mathbb{R}^2.$$

What is a suitable Sobolev space and what boundary value problem is satisfied by the vector field \mathbf{u} ?

(Q1.5.3.19.I) Which boundary value problem does the minimizer of the functional

$$J(v) = \int_{\Omega} \left| \frac{\partial u}{\partial x_1}(x) - \frac{\partial u}{\partial x_2}(x) \right|^2 + |u(x)|^2 - \|x\| u(x) \, dx , \quad v \in H_0^1(\Omega) ,$$

solve? Here, $\Omega \subset \mathbb{R}^2$ is a bounded domain.

(Q1.5.3.19.J) For a bounded domain $\Omega \subset \mathbb{R}^2$ and a given vector field $\mathbf{a} \in C^0(\Omega, \mathbb{R}^2)$, we consider the functional

$$J : H_0^1(\Omega) \rightarrow \mathbb{R}, \quad J(v) := \int_{\Omega} [|\mathbf{a}(x) \cdot \nabla v(x)|^2 - v(x)] \, dx . \quad (1.5.3.20)$$

State the variational problem satisfied by a global minimizer of J .

(Q1.5.3.19.K) What is the “physical meaning” of the homogeneous **Neumann boundary conditions** in

$$-\operatorname{div}(\sigma(x) \nabla u) = f \quad \text{in } \Omega , \quad \begin{aligned} u &= g && \text{on } \Gamma_0 , \\ (\sigma(x) \nabla u) \cdot \mathbf{n} &= 0 && \text{on } \partial\Omega \setminus \Gamma_0 , \end{aligned} \quad (1.5.3.18)$$

if this boundary value problem is regarded as a mathematical model for the vertical displacement of a taut membrane?

△

1.6 Diffusion Models (Stationary Heat Conduction)

Now we look at a class of physical phenomena, for which models are based on two building blocks

1. a **conservation principle** (of mass, energy, etc.),
2. a **potential driven flux** of the conserved quantity.

Mathematical modelling for these phenomena naturally involves partial differential equations in the first steps, which are supplemented with boundary conditions. Hence, second-order elliptic boundary value problems arise first, while variational formulations are deduced from them, thus reversing the order of steps followed for equilibrium models in Section 1.2 through Section 1.5.

§1.6.0.1 (Heat flux) In order to keep the presentation concrete, the discussion will target **heat conduction**, about which everybody should have a sound “intuitive grasp”.

☞ notation: $\Omega \subset \mathbb{R}^3$: bounded open region occupied by solid object
 $(\hat{=} \Omega \rightarrow \text{computational domain})$

Fundamental concept:

heat flux, modelled by vector field $\mathbf{j} : \Omega \mapsto \mathbb{R}^3$

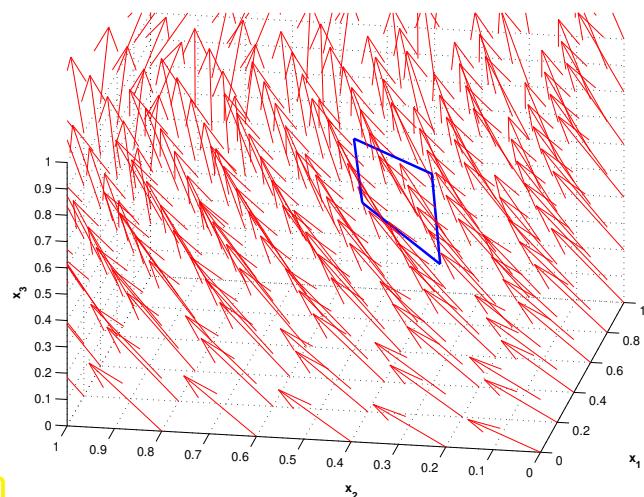
Heat flux = power flux: $[\mathbf{j}] = \frac{W}{m^2}$

Vector field $\mathbf{j} : \Omega := [0, 1]^2 \rightarrow \mathbb{R}^3$

normal vector \mathbf{n}

Total heat flux through oriented surface $\Sigma \subset \mathbb{R}^3$

$$\text{Power} \quad P_\Sigma = \int_{\Sigma} \mathbf{j} \cdot \mathbf{n} dS. \quad (1.6.0.2)$$



P_Σ ($[P_\Sigma] = 1\text{W}$): directed total power flowing through the oriented surface Σ per unit time. Note that the sign of P_Σ will change when flipping the normal of Σ !

For stationary heat conduction it is clear that for every volume the total net power flux through its boundary must be balanced by heat production or cooling inside. This is the effect of a heat source modeled by a power density function f . The following **balance law** casts this into formulas.

Conservation of energy

$$\int_{\partial V} \mathbf{j} \cdot \mathbf{n} dS = \int_V f dx \quad \text{for all "control volumes" } V. \quad (1.6.0.3)$$

↑ ←

power flux through surface of V heat production inside V

$f = \text{heat source/sink}$ ($[f] = \frac{\text{W}}{\text{m}^3}$), $f = f(\mathbf{x})$ and f can be discontinuous ($f \in C_{\text{pw}}^0(\Omega)$)

§1.6.0.4 (Flux law) A flow of heat is triggered by temperature differences. Now we aim to quantify this relationship.

Intuition:

- ◆ heat flows from hot zones to cold zones
- ◆ the larger the temperature difference, the stronger the heat flow

Experimental evidence supports this intuition and, for many materials, yields the following quantitative relationship that connects the heat flux in a point to the local change of temperature.

Fourier's law

$$\mathbf{j}(\mathbf{x}) = -\kappa(\mathbf{x}) \operatorname{grad} u(\mathbf{x}), \quad \mathbf{x} \in \Omega. \quad (1.6.0.5)$$

Meaning of the quantities:

\mathbf{j} = heat flux	$([\mathbf{j}] = 1 \frac{\text{W}}{\text{m}^2})$
u = temperature	$([u] = 1 \text{K})$
κ = heat conductivity	$([\kappa] = 1 \frac{\text{W}}{\text{Km}})$

(1.6.0.5) \Rightarrow Heat flow from hot to cold regions is *linearly proportional* to the gradient of the temperature u

Some facts about the heat conductivity κ :

- ☞ • $\kappa = \kappa(\mathbf{x})$ for **non-homogeneous** materials. (spatially varying heat conductivity)
- κ can even be discontinuous for composite materials.
- κ may be $\mathbb{R}^{3,3}$ -valued (heat conductivity tensor).

The most general form of the heat conductivity (tensor) enjoys the very same properties as the dielectric tensor introduced in Section 1.2.2:

From thermodynamic principles, cf. (1.2.2.8):

$$\exists \kappa^-, \kappa^+ > 0: \quad 0 < \kappa^- \leq \kappa(\mathbf{x}) \leq \kappa^+ < \infty \quad \text{for almost all } \mathbf{x} \in \Omega . \quad (1.6.0.6)$$

Terminology: (1.6.0.6) \leftrightarrow κ is bounded and **uniformly positive**, see Def. 1.2.2.9. \square

§1.6.0.7 (Derivation of 2nd-order linear elliptic PDE) From (1.6.0.3), by Gauss' theorem Thm. 1.5.2.4, we deduce a the conservation law in integral form:

$$\int_V \operatorname{div} \mathbf{j}(\mathbf{x}) d\mathbf{x} = \int_V f(\mathbf{x}) d\mathbf{x} \quad \text{for all “control volumes” } V \subset \Omega .$$

Now appeal to another version of the fundamental lemma of the calculus of variations, see Lemma 1.5.3.4, this time sporting piecewise constant test functions.

► local form of energy conservation:

$$\operatorname{div} \mathbf{j} = f \quad \text{in } \Omega . \quad (1.6.0.8)$$

Combine equations (1.6.0.8) & (1.6.0.5)

$$\mathbf{j} = -\kappa(\mathbf{x}) \operatorname{grad} u \quad (1.6.0.5)$$



$$\operatorname{div} \mathbf{j} = f \quad (1.6.0.8)$$

$$-\operatorname{div}(\kappa(\mathbf{x}) \operatorname{grad} u) = f \quad \text{in } \Omega . \quad (1.6.0.9)$$

This is a *linear* scalar second order elliptic PDE satisfied by the unknown temperature u . It is exactly the same type of equation that we found in Section 1.5.3, (1.5.3.5). \square

Review question(s) 1.6.0.10 (Stationary heat conduction)

(Q1.6.0.10.A) Why is Fourier's law called a *linear* material law?

(Q1.6.0.10.B) What is the physical meaning of the right hand side function f of the stationary heat equation.

(Q1.6.0.10.C) Derive the partial differential equation governing stationary heat conduction by combining the **balance law**

$$\int_{\partial V} \mathbf{j} \cdot \mathbf{n} dS = \int_V f d\mathbf{x} \quad \text{for all “control volumes” } V ,$$

with **Fourier's law**

$$\mathbf{j} = -\kappa(\mathbf{x}) \mathbf{grad} u .$$

What physical quantities and properties are modelled by u , \mathbf{j} , f , and κ ?

△

1.7 Boundary Conditions

In the model problems (1.3.4.9a)–(1.4.2.3) listed in the beginning of Section 1.4.2 we fixed the value of the unknown function $u : \Omega \mapsto \mathbb{R}$ on the boundary $\partial\Omega$, which gave rise to **Dirichlet boundary conditions** in the strong form (1.5.3.8)

$$u = g \quad \text{on } \partial\Omega \quad \text{for given } g \in C^0(\partial\Omega) .$$

The only exception was the gappy frame setting for a membrane, see Ex. 1.5.3.11: There we found (**homogeneous**) **Neumann boundary conditions** in the strong form (1.5.3.18) of the boundary value problem:

$$(\sigma(\mathbf{x}) \mathbf{grad} u) \cdot \mathbf{n} = 0 \quad \text{on } \partial\Omega .$$

In this section we resume the discussion of boundary conditions and examine them for stationary heat conduction, see previous section. This has the advantage that for this everyday physical phenomenon boundary conditions have a very clear intuitive meaning.

Specifying boundary conditions is essential for obtaining a well-posed problem, because the plain (partial) differential equation usually has infinitely many solutions. We need to impose boundary conditions in order to obtain a unique solution. This is why we invariably study boundary value problems,

$$-\operatorname{div}(\kappa(\mathbf{x}) \mathbf{grad} u) = f + \text{boundary conditions} \Rightarrow \text{elliptic boundary value problem (BVP)},$$

instead of aiming to “solve partial differential equations”.

Fundamental boundary conditions for 2nd-order elliptic BVPs

Boundary conditions on surface/boundary $\partial\Omega$ of Ω :

- (i) Temperature u is fixed: with $g : \partial\Omega \mapsto \mathbb{R}$ prescribed

$$u = g \quad \text{on } \partial\Omega . \tag{1.7.0.2}$$



Dirichlet boundary conditions

- (ii) Heat flux \mathbf{j} through $\partial\Omega$ is fixed: with $h : \partial\Omega \mapsto \mathbb{R}$ prescribed ($\mathbf{n} : \partial\Omega \mapsto \mathbb{R}^3$ exterior unit normal vectorfield) on $\partial\Omega$

$$\mathbf{j} \cdot \mathbf{n} = -h \quad \text{on } \partial\Omega . \tag{1.7.0.3}$$



Neumann boundary conditions

- (iii) Heat flux through $\partial\Omega$ depends on (local) temperature: with increasing function $\Psi : \mathbb{R} \mapsto \mathbb{R}$

$$\mathbf{j} \cdot \mathbf{n} = \Psi(u) \quad \text{on } \partial\Omega \tag{1.7.0.4}$$



radiation boundary conditions

EXAMPLE 1.7.0.5 (Convective cooling (simple model)) Heat is carried away from the surface of the body by a fluid at bulk temperature u_0 . A crude model assumes that the heat flux depends *linearly* on the temperature difference between the surface of Ω and the bulk temperature of the fluid.

$$\mathbf{j} \cdot \mathbf{n} = q(u - u_0) \quad \text{on } \partial\Omega, \quad \text{where } 0 < q^- \leq q(x) \leq q^+ < \infty \quad \text{for almost all } \mathbf{x} \in \partial\Omega.$$

When combined with Fourier's law (1.6.0.5), the convective cooling boundary conditions become

$$\kappa(\mathbf{x}) \mathbf{grad} u(\mathbf{x}) + q(u(\mathbf{x}) - u_0) = 0, \quad \mathbf{x} \in \partial\Omega, \quad (1.7.0.6)$$

and in this form they are known as **Robin boundary conditions** or **impedance boundary conditions**. □

EXAMPLE 1.7.0.7 (Radiative cooling (simple model)) A hot body emits electromagnetic radiation (blackbody emission), which drains thermal energy. The radiative energy loss is roughly proportional to the 4th power of the temperature difference between the surface temperature of the body and the ambient temperature. This spawns a boundary condition of the type (1.7.0.4),

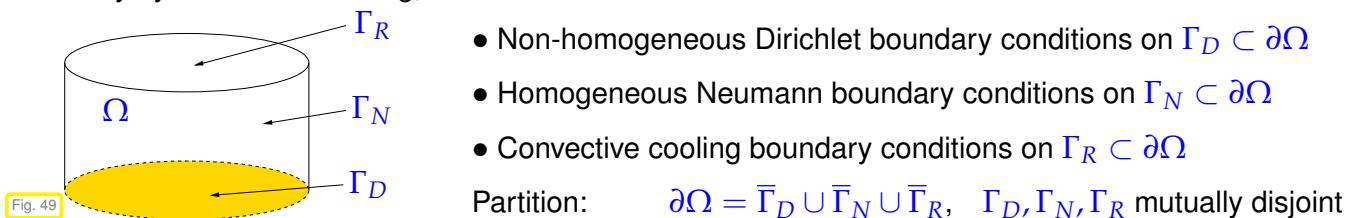
$$\mathbf{j} \cdot \mathbf{n} = \alpha |u - u_0| (u - u_0)^3 \quad \text{on } \partial\Omega, \quad \text{with } \alpha > 0. \quad (1.7.0.8)$$

We point out that this constitutes a **non-linear** boundary condition, because the heat flux \mathbf{j} obviously does not depend affine-linearly on the temperature u as in □

Terminology: If $g = 0$ or $h = 0$, we talk about **homogeneous** Dirichlet or Neumann boundary conditions

Remark 1.7.0.9 (Mixed boundary conditions) Different boundary conditions can be prescribed on different parts of $\partial\Omega$ (→ **mixed boundary conditions**, cf. Ex. 1.5.3.11) □

EXAMPLE 1.7.0.10 (“Wrapped rock on a stove”) We consider a solid cylinder mounted on a heating plate whose temperature can be controlled. The vertical walls of the cylinder are covered with an insulating layer, which is assumed to be perfect. The top face is in contact with air and, thus, heat is transported away by convective cooling, see Ex. 1.7.0.5.



This example demonstrates a setting, in which different boundary conditions are imposed on different parts of the boundary of the computational domain. □

What we have just observed in the previous example reflects a general principle:

For second order elliptic boundary value problems **exactly one** boundary condition is needed on **every** part of $\partial\Omega$.

Remark 1.7.0.11 (Linear BVP) So far we have exclusively studied linear boundary value problems. What does “linearity” mean in this context? What we have in mind is that

the solution mapping $\begin{pmatrix} f \\ g \end{pmatrix} \mapsto u$ for (1.6.0.9), (1.7.0.2) is **linear**.

This means that, if u_i solves the Dirichlet problem with source function f_i and Dirichlet data g_i , $i = 1, 2$, then $u_1 + u_2$ solves (1.6.0.9) & (1.7.0.2) for source $f_1 + f_2$ and boundary values $g_1 + g_2$. □

Review question(s) 1.7.0.12 (Boundary conditions)

(Q1.7.0.12.A) [Exterior unit normal vector field] Give the formula for the exterior unit normal vectorfield $\mathbf{n} : \partial\Omega \rightarrow \mathbb{R}^2$ for $\Omega := \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| < \frac{1}{2}\}$.

(Q1.7.0.12.B) [Different types of boundary conditions] For the second-order elliptic partial differential equation

$$-2 \frac{\partial^2 u}{\partial x_1^2} - \frac{\partial^2 u}{\partial x_2^2} + u = f \quad \text{in } \Omega \subset \mathbb{R}^3.$$

give the formulas for

1. **Dirichlet boundary conditions** on $\partial\Omega$,
2. **Neumann boundary conditions** on $\partial\Omega$,
3. and **impedance boundary conditions** on $\partial\Omega$.

(Q1.7.0.12.C) We learned that

For second order elliptic boundary value problems *exactly one* boundary condition is needed on *every* part of $\partial\Omega$.

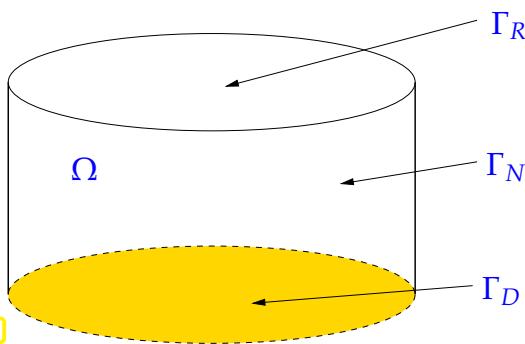


Fig. 50

Describe in your own words what this rule means for the setting discussed in Ex. 1.7.0.10, see figure beside.

(Q1.7.0.12.D) [Stationary current model] In this quizz we consider *stationary electric currents* in a conducting body occupying $\Omega \subset \mathbb{R}^3$. In this model a vector field $\mathbf{j} : \Omega \rightarrow \mathbb{R}^3$ describes the electric current density (units $[\mathbf{j}] = \frac{\text{A}}{\text{m}^2}$) obeying *Ohm's law* $\mathbf{j} = -\sigma \mathbf{grad} u$, which corresponds to Fourier's law (1.6.0.5). Here, u is the electric potential, cf. (1.2.2.2) (units $[u] = \text{V}$), and $\sigma : \Omega \rightarrow \mathbb{R}^+$ stands for the uniformly positive conductivity (units $[\sigma] = \frac{\text{A}}{\text{Vm}}$).

- What is the meaning of $\operatorname{div} \mathbf{j}$?
- Argue, why the *normal component* of \mathbf{j} has to be continuous across any smooth surface.
- What is the physical meaning of Dirichlet and Neumann boundary conditions in the stationary current model?
- What could be the physical interpretation of a *Neumann boundary condition*

$$\mathbf{j} \cdot \mathbf{n} = -h \quad \text{on } \partial\Omega, \tag{1.7.0.3}$$

for the stationary current model?

△

1.8 Second-Order Elliptic Variational Problems

In Section 1.2 through Section 1.5 we pursued the derivation:

Minimization problem (e.g., (1.2.1.24), (1.2.2.16))	>	Variational problem (e.g., (1.4.2.3), (1.4.2.2))	>	BVP for PDE (e.g., (1.5.3.8), (1.5.3.18))
--------------------------------------------------------	---	-----------------------------------------------------	---	----------------------------------------------

Now want to move in the opposite direction:

$$\text{PDE} \quad + \quad \text{boundary conditions} \quad \Rightarrow \quad \text{variational problem}$$

(e.g. (1.6.0.9)) (e.g., (1.7.0.2), (1.7.0.3), (1.7.0.4))

This can be done following a sequence of routine steps:

Formal transition from boundary value problem for PDE to variational problem

STEP 1: *test PDE with smooth functions*

(do not test, where the solution is known, e.g., on the boundary)

STEP 2: *integrate over domain*

STEP 3: *perform integration by parts*

(e.g. by using Green's first formula, Thm. 1.5.2.7)

STEP 4: [optional] *incorporate boundary conditions into boundary terms*

STEP 5: *Choose suitable function spaces (Sobolev spaces)*

(Section 1.3.1: largest function space on which variational problem well posed)

EXAMPLE 1.8.0.2 (Variational formulation for heat conduction with Dirichlet boundary conditions)

Remember that Dirichlet boundary conditions mean prescribed values of the solution on the boundary $\partial\Omega$. In the case of heat conduction this leads to the elliptic BVP

$$-\operatorname{div}(\kappa(x) \operatorname{grad} u) = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega. \quad (1.8.0.3)$$

Here the solution is fixed on $\partial\Omega$. Therefore, we test with functions that vanish on $\partial\Omega$: “do not test where the solution is known”.

STEP 1 & 2: test the PDE with $v \in C_0^\infty(\Omega)$ (smooth functions vanishing on $\partial\Omega$) and integrate over Ω

$$\Rightarrow - \int_{\Omega} \operatorname{div}(\kappa(x) \operatorname{grad} u) v \, dx = \int_{\Omega} f v \, dx. \quad (1.8.0.4)$$

Again note: $v|_{\partial\Omega} = 0$ for test function, because u is known already on $\partial\Omega$.

STEP 3: use **Green's formula** from Thm. 1.5.2.7 on $\Omega \subset \mathbb{R}^d$ (multidimensional integration by parts):
Apply

$$\int_{\Omega} \mathbf{j} \cdot \operatorname{grad} v \, dx = - \int_{\Omega} \operatorname{div} \mathbf{j} v \, dx + \int_{\partial\Omega} \mathbf{j} \cdot \mathbf{n} v \, dS. \quad (1.5.2.8)$$

to (1.8.0.4) choosing the vector field as $\mathbf{j} := \kappa(x) \operatorname{grad} u$:

$$\Rightarrow \int_{\Omega} \kappa(x) \operatorname{grad} u \cdot \operatorname{grad} v \, dx - \underbrace{\int_{\partial\Omega} \kappa(x) \operatorname{grad} u \cdot \mathbf{n} v \, dS}_{=0, \text{because } v|_{\partial\Omega}=0} = \int_{\Omega} f v \, dx \quad \forall v \in C_0^\infty(\Omega).$$

This gives the variational formulation after we switch to “maximal admissible function spaces” (Sobolev spaces, see Section 1.3, as spaces of functions with finite energy).

Variational form of (1.8.0.3): seek

$$\begin{aligned} u \in H^1(\Omega) : \quad & \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in H_0^1(\Omega) . \\ u = g \text{ on } \partial\Omega : \quad & \end{aligned} \quad (1.8.0.5)$$

Note that this weak form is exactly the same as that obtained for the clamped membrane problem (1.4.2.2): Rather different physical phenomena have led to the same mathematical model! \square

EXAMPLE 1.8.0.6 (Variational formulation: heat conduction with general radiation boundary conditions) In this case the appropriate treatment of boundary conditions in STEP 4 can be demonstrated.

$$\text{BVP:} \quad -\operatorname{div}(\kappa(x) \mathbf{grad} u) = f \quad \text{in } \Omega , \quad -\kappa(x) \mathbf{grad} u \cdot \mathbf{n} = \Psi(u) \quad \text{on } \partial\Omega . \quad (1.8.0.7)$$

STEP 1 & 2: $u|_{\partial\Omega}$ not fixed \Rightarrow test with $v \in C^\infty(\overline{\Omega})$

$$\Rightarrow - \int_{\Omega} \operatorname{div}(\kappa(x) \mathbf{grad} u) v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in C^\infty(\overline{\Omega}) .$$

STEP 3 & 4: STEP 3 & 4: apply Green's first formula (1.5.2.8) and incorporate boundary conditions:

$$\Rightarrow \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx - \int_{\partial\Omega} \underbrace{\kappa(x) \mathbf{grad} u \cdot \mathbf{n}}_{=-\Psi(u) \text{ (STEP 4)}} v \, dS = \int_{\Omega} f v \, dx \quad \forall v \in C^\infty(\overline{\Omega}) .$$


Variational formulation of (1.8.0.7): seek

$$u \in H^1(\Omega): \quad \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx + \int_{\partial\Omega} \Psi(u) v \, dS = \int_{\Omega} f v \, dx \quad \forall v \in H^1(\Omega) . \quad (1.8.0.8)$$

In the above manipulations we have tacitly assumed that the coefficients and source functions of the boundary value problems we started from are sufficient smooth and that the solutions u are classical solutions. Then integration by parts showed that u also satisfies the weak (variational) form of the boundary value problem.

Theorem 1.8.0.9. Classical solutions are weak solutions

If $\kappa \in C^1(\Omega)$, classical solutions $u \in C^2(\Omega)$ of the boundary value problems (1.8.0.3) and (1.8.0.7) also solve the associated variational problems.

Proof. Apply Thm. 1.5.2.7 as in the derivation of the weak formulations. \square

EXAMPLE 1.8.0.10 (Variational formulation for Neumann problem) We start from the heat conduction problem with given fixed heat flux h on the boundary, the following 2nd-order elliptic (inhomogeneous) Neumann problem:

$$\begin{aligned} \text{BVP:} \quad & -\operatorname{div}(\kappa(x) \mathbf{grad} u) = f \quad \text{in } \Omega , \\ & \kappa(x) \mathbf{grad} u \cdot \mathbf{n} = h(x) \quad \text{on } \partial\Omega . \end{aligned} \quad (1.8.0.11)$$

In this example we deal with Neumann boundary conditions (1.7.0.3), prescribed heat flux) on the whole boundary.

The corresponding variational formulation can be derived as in Ex. 1.8.0.6. It is

$$u \in H^1(\Omega): \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx + \int_{\partial\Omega} \Psi(u) v \, dS = \int_{\Omega} f v \, dx \quad \forall v \in H^1(\Omega). \quad (1.8.0.8)$$

with the special choice $\Psi(u) = -h$:

$$u \in H^1(\Omega): \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx - \int_{\partial\Omega} h v \, dS = \int_{\Omega} f v \, dx \quad \forall v \in H^1(\Omega). \quad (1.8.0.12)$$

In the “standard form” of a linear variational problem it reads:

$$u \in H^1(\Omega): \underbrace{\int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx}_{=:a(u,v)} = \underbrace{\int_{\Omega} f v \, dx + \int_{\partial\Omega} h v \, dS}_{=:l(v)} \quad \forall v \in H^1(\Omega).$$

For this simple linear variational problem existence and uniqueness of solutions becomes an issue.

Observation: when we test (1.8.0.8) with $v \equiv 1$ $\Rightarrow - \int_{\partial\Omega} h \, dS = \int_{\Omega} f \, dx$ (1.8.0.13)

This is a **compatibility condition** for the existence of (variational) solutions of the Neumann problem!

Interpretation of (1.8.0.13) against the backdrop of the stationary heat conduction model:

conservation of energy \rightarrow (1.6.0.3): Heat generated inside Ω ($\leftrightarrow f$) must be offset by heat flux through $\partial\Omega$ ($\rightarrow h$). ◻

Remark 1.8.0.14 (Uniqueness of solutions of the Neumann problem) We saw that for the Neumann problem in weak form (1.8.0.12), the given data f and h have to fulfill the compatibility condition (1.8.0.13). This is a necessary condition for the existence of a solution.

We also observe that if (1.8.0.13) holds true, then

$$v \in H^1(\Omega) \text{ solves (1.8.0.8)} \iff v + \gamma \text{ solves (1.8.0.8)} \quad \forall \gamma \in \mathbb{R},$$

we say, “the solution is unique only up to constants”.

A complementary observation is that the bilinear form $a(u, v) := \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx$ of (1.8.0.12) is *not* s.p.d (\rightarrow Def. 1.2.3.27) on the trial/test space $H^1(\Omega)$. How can we fix this?



Idea: Restore uniqueness of solutions by

enforcing average temperature to be zero $\int_{\Omega} u(x) \, dx = 0$

This amounts to posing the variational problem (1.8.0.8) over the **constrained** function space

$$H_*^1(\Omega) := \{v \in H^1(\Omega): \int_{\Omega} v(x) \, dx = 0\}. \quad (1.8.0.15)$$

The norm on $H_*^1(\Omega)$ is the same as on $H_0^1(\Omega)$, see Def. 1.3.4.8:

$$\|v\|_{H_*^1(\Omega)}^2 = |v|_{H^1(\Omega)}^2 = \int_{\Omega} \|\mathbf{grad} v\|^2 \, dx, \quad v \in H_*^1(\Omega).$$

The norm property (N1) is satisfied, because

$$|v|_{H^1(\Omega)} = 0 \Rightarrow \mathbf{grad} v \equiv 0 \text{ in } \Omega \Rightarrow v \equiv \text{const} \xrightarrow{\int_{\Omega} v \, dx = 0} v \equiv 0.$$

In this argument a condition on Ω stated in § 1.2.1.14 plays a crucial role, the assumption that Ω is *connected*. Otherwise, a vanishing gradient would just imply that v is merely piecewise constant in different connected components of Ω , and these constant values could be chosen to make the overall mean vanish.

These arguments also show that a is s.p.d (\rightarrow Def. 1.2.3.27) on $H_*^1(\Omega)$, cf. Thm. 1.8.0.20.

► Uniquely solvable variational formulation of Neumann problem:

$$u \in H_*^1(\Omega): \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx + \int_{\partial\Omega} h v \, dS \quad \forall v \in H_*^1(\Omega). \quad (1.8.0.16)$$

↓

Supplement 1.8.0.17 (Stability of variational Neumann problem) For the sake of simplicity we consider the *homogeneous Neumann problem* with constant coefficients, that is (1.8.0.16) with vanishing Neumann data $h = 0$ and $\kappa \equiv 1$:

$$u \in H_*^1(\Omega): \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in H_*^1(\Omega). \quad (1.8.0.18)$$

General Neumann data will be discussed below in § 1.9.0.9.

We ask the same question as in Section 1.4.3: How do perturbations δf in the source function f , measured in $L^2(\Omega)$ -norm, affect the energy norm ($= \|\cdot\|_{H^1(\Omega)}$) of the solution. As in Section 1.4.3, we find that the induced perturbation δu of the solution again solves a linear variational equation:

$$\delta u \in H_*^1(\Omega): \int_{\Omega} \mathbf{grad} \delta u \cdot \mathbf{grad} v \, dx = \int_{\Omega} \delta f v \, dx \quad \forall v \in H_*^1(\Omega). \quad (1.8.0.19)$$

Recall that the estimate (1.4.3.6) was a consequence of the first Poincaré-Friedrichs inequality of Thm. 1.3.4.17, which makes a statement about norms on $H_0^1(\Omega)$. However, now we deal with a variational problem posed on $H_*^1(\Omega)$. Thus, we need a counterpart of Thm. 1.3.4.17 on that space:

Theorem 1.8.0.20. Second Poincaré-Friedrichs inequality

If $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, is *bounded* and *connected*, then

$$\exists C = C(\Omega) > 0: \|u\|_0 \leq C \operatorname{diam}(\Omega) \|\mathbf{grad} u\|_0 \quad \forall u \in H_*^1(\Omega).$$

☞ notation: $C = C(\Omega)$ indicates that the constant C may depend on the shape of the domain Ω .

Proof. We deal with the simple case $d = 1$, $\Omega = [0, 1]$ first. As in the proof of Thm. 1.3.4.17, we employ a density argument and assume that u is sufficiently smooth, $u \in C^1([0, 1])$.

By the fundamental theorem of calculus (1.5.1.6)

$$u(x) = u(y) + \int_y^x \frac{du}{dx}(\tau) \, d\tau, \quad 0 \leq x, y \leq 1.$$

$$\blacktriangleright \quad u(x) = \int_0^1 u(x) dy = \underbrace{\int_0^1 u(y) dy}_{=0} + \int_0^1 \int_y^x \frac{du}{dx}(\tau) d\tau dy .$$

Then use the Cauchy-Schwarz inequality (1.3.4.15):

$$u(x)^2 \leq \int_0^1 \int_y^x 1 d\tau dy \int_0^1 \int_y^x \left| \frac{du}{dx}(\tau) \right|^2 d\tau dy \leq \int_0^1 \left| \frac{du}{dx}(\tau) \right|^2 d\tau .$$

Integrating over Ω yields the estimate

$$\|u\|_0^2 = \int_0^1 u^2(x) dx \leq \int_0^1 \left| \frac{du}{dx}(\tau) \right|^2 d\tau = \|u\|_{H^1(\Omega)}^2 .$$

By (1.3.4.15), Thm. 1.8.0.20 implies the continuity of the first term in ℓ .

In higher dimensions $d > 1$ we restrict ourselves to **convex** domains $\Omega \subset \mathbb{R}^d$, i.e., domains containing all line segments connecting two of their points. We pick any $u \in C^1(\bar{\Omega})$ with *vanishing mean* and we employ the fundamental theorem of calculus along lines, also known as “mean-value theorem”:

$$u(x) = u(y) + \int_0^1 \mathbf{grad} u(y + \tau(x - y)) \cdot (x - y) d\tau \quad \forall x, y \in \Omega . \quad (1.8.0.21)$$

We integrate the right-hand side over Ω in y , which gives an integral representation of u :

$$u(x) = \frac{1}{|\Omega|} \underbrace{\int_{\Omega} u(y) dy}_{=0, \text{ since } u \in H_*^1(\Omega)} + \frac{1}{|\Omega|} \int_{\Omega} \int_0^1 \mathbf{grad} u(y + \tau(x - y)) \cdot (x - y) d\tau dy . \quad (1.8.0.22)$$

$$\begin{aligned} \int_{\Omega} u(x)^2 dx &= \frac{1}{|\Omega|^2} \int_{\Omega} \left| \int_{\Omega} \int_0^1 \mathbf{grad} u(y + \tau(x - y)) \cdot (x - y) d\tau dy \right|^2 dx \\ &\stackrel{\textcircled{1}}{\leq} \frac{1}{|\Omega|^2} \int_{\Omega} \left(\int_{\Omega} 1 dy \right) \int_{\Omega} \left| \int_0^1 \mathbf{grad} u(y + \tau(x - y)) \cdot (x - y) d\tau \right|^2 dy dx \\ &\stackrel{\textcircled{2}}{\leq} \frac{1}{|\Omega|} \int_{\Omega} \int_{\Omega} \int_0^1 |\mathbf{grad} u(y + \tau(x - y)) \cdot (x - y)|^2 d\tau dy dx \\ &= \frac{1}{|\Omega|} \int_{\Omega} \int_{\Omega} \int_{\Omega}^{1/2} |\mathbf{grad} u(y + \tau(x - y)) \cdot (x - y)|^2 d\tau dx dy + \\ &\quad \frac{1}{|\Omega|} \int_{\Omega} \int_{\Omega} \int_0^{1/2} |\mathbf{grad} u(x + (1 - \tau)(y - x)) \cdot (y - x)|^2 d\tau dy dx =: I + II . \end{aligned}$$

In step $\textcircled{1}$ we invoked the Cauchy-Schwarz inequality (1.3.4.15) for $L^2(\Omega)$, whereas $\textcircled{2}$ is a consequence of the Cauchy-Schwarz inequality in $L^2([0,1])$.

Next, we employ the transformation formula for integrals to deal with I and II . For I we change the variables according to

$$\begin{bmatrix} z \\ w \\ \sigma \end{bmatrix} = \begin{bmatrix} y + \tau(x - y) \\ y \\ \tau \end{bmatrix} \quad \Rightarrow \quad dz d\sigma dw = \sigma^d d\tau dx dy .$$

The integration in z -direction will be over the w/σ -dependent domain $\Omega' := y + \sigma(\Omega - y) \subset \Omega$, which is contained in Ω thanks to convexity. This makes it possible to enlarge the domain of integration to the whole of Ω and obtain the following estimate.

$$I = \frac{1}{|\Omega|} \int_{\Omega} \int_{1/2}^1 \int_{\Omega'} \left| \operatorname{grad} u(z) \cdot \frac{1}{\sigma}(z - w) \right|^2 \sigma^{-d} dz d\sigma dw \leq 2^{d+2} \operatorname{diam}(\Omega)^2 \|\operatorname{grad} u\|_{L^2(\Omega)}^2 .$$

For II we use the transformation

$$\begin{bmatrix} z \\ w \\ \sigma \end{bmatrix} = \begin{bmatrix} x + (1 - \tau)(y - x) \\ x \\ 1 - \tau \end{bmatrix} \quad \Rightarrow \quad dz d\sigma dw = -\sigma^d d\tau dy dx ,$$

and the same arguments as before yield

$$II = \frac{1}{|\Omega|} \int_{\Omega} \int_{1/2}^1 \int_{\Omega'} \left| \operatorname{grad} u(z) \cdot \frac{1}{\sigma}(z - w) \right|^2 \sigma^{-d} dz d\sigma dw \leq 2^{d+2} \operatorname{diam}(\Omega)^2 \|\operatorname{grad} u\|_{L^2(\Omega)}^2 .$$

The final estimate is

$$\|u\|_{L^2(\Omega)} \leq \sqrt{2}^{d+3} \operatorname{diam}(\Omega) \|\operatorname{grad} u\|_{L^2(\Omega)} ,$$

which implies the assertion of the theorem by density arguments as in the proof of Thm. 1.3.4.17. \square

An immediate consequence of Thm. 1.8.0.20 and the Cauchy-Schwarz inequality (1.3.4.15) for integrals is

$$|\delta u|_{H^1(\Omega)} \leq \operatorname{diam}(\Omega) C \|\delta f\|_{L^2(\Omega)} , \quad (1.8.0.23)$$

where δu solves (1.8.0.19) and $C > 0$ depends on the shape of Ω only. \dashv

Supplement 1.8.0.24 (Generalizing Poincaré-Friedrichs estimates) The Poincaré-Friedrichs estimates from both Thm. 1.3.4.17 and Thm. 1.8.0.20 are special cases of a more general result. To state it we need to know what are “bounded linear operators”

Definition 1.8.0.25. Bounded linear operator

Given two normed vector spaces X and W we call a *linear* mapping $F : X \rightarrow W$ a **bounded linear operator**, if

$$\exists C > 0: \|F(x)\|_W \leq C\|x\|_X \quad \forall x \in X .$$

Below, read “Lipschitz domain” as spatial computational domain according to § 1.2.1.14.

Theorem 1.8.0.26. Poincaré-Friedrichs-type estimate

Let $\Omega \subset \mathbb{R}^d$ a *bounded* and *connected Lipschitz domain* and $F : H^1(\Omega) \rightarrow W$ a *bounded/continuous linear operator*, where W is some normed vector space.

If $F(\{x \mapsto 1\}) \neq \mathbf{0}$, then

$$\exists C > 0: \|u\|_{L^2(\Omega)} \leq C(\|\mathbf{grad} u\|_{L^2(\Omega)} + \|F(u)\|_W) \quad \forall u \in H^1(\Omega). \quad (1.8.0.27)$$

The proof relies on a deep and, unfortunately, truly non-intuitive result about Sobolev space, see [Eva98, Sect. 5.7].

Theorem 1.8.0.28. Rellich's Theorem: Compact embedding of $H^1(\Omega)$ in $L^2(\Omega)$

Let $\Omega \subset \mathbb{R}^d$ be a *bounded Lipschitz domain*. Then every *bounded* sequence in $H^1(\Omega)$ possesses a **sub-sequence** that **converges in $L^2(\Omega)$** .

Proof. (of Thm. 1.8.0.26, by contradiction) Assume that (1.8.0.27) is false. Then there is a sequence

$$(u_n)_{n \in \mathbb{N}} \subset H^1(\Omega), \quad \|u_n\|_{L^2(\Omega)} = 1: \quad \boxed{\|u_n\|_{L^2(\Omega)} \geq n(\|\mathbf{grad} u_n\|_{L^2(\Omega)} + \|F(u_n)\|_W)}, \quad (1.8.0.29)$$

which means

$$\lim_{n \rightarrow \infty} \|\mathbf{grad} u_n\|_{L^2(\Omega)} = 0 \quad \text{and} \quad \lim_{n \rightarrow \infty} \|F(u_n)\|_W = 0. \quad (1.8.0.30)$$

Obviously, the sequence $(u_n)_{n \in \mathbb{N}}$ is bounded in $H^1(\Omega)$ and, by Thm. 1.8.0.28, possesses a sub-sequence that converges in $L^2(\Omega)$. Abusing notation, we still write $(u_n)_{n \in \mathbb{N}}$ for that sub-sequence. Let $u^* \in L^2(\Omega)$ be its limit. From (1.8.0.30) we conclude that this sub-sequence also converges in $H^1(\Omega)$ and

$$\lim_{n \rightarrow \infty} \|\mathbf{grad} u_n\|_{L^2(\Omega)} = 0 \Rightarrow u^* \in H^1(\Omega) \quad \text{and} \quad \mathbf{grad} u^* = 0.$$

As a consequence, $u^* \equiv \text{const}$. However, from (1.8.0.29) and due to the continuity of F

$$F(u^*) = \lim_{n \rightarrow \infty} F(u_n) = \mathbf{0}.$$

This cannot be reconciled with the assumption $F(\{x \mapsto 1\}) \neq \mathbf{0}$. □

Note that both Thm. 1.3.4.17 and Thm. 1.8.0.20 can immediately be inferred from Thm. 1.8.0.26:

- In the case of Thm. 1.3.4.17 choose the “restriction to the boundary operator” $F : H^1(\Omega) \rightarrow L^2(\partial\Omega)$, $F(u) := u|_{\partial\Omega}$, whose continuity is a consequence of Thm. 1.9.0.10.
- To conclude Thm. 1.8.0.20 pick $F : H^1(\Omega) \rightarrow \mathbb{R}$, $F(u) := \int_{\Omega} u(x) dx$, which is trivially continuous by the Cauchy-Schwarz inequality.

Supplement 1.8.0.31 (Compact embedding) In functional analysis the statement of Rellich's theorem Thm. 1.8.0.28 is often phrased as

$H^1(\Omega)$ is **compactly embedded** in $L^2(\Omega)$.

This paragraph should give you some insight into this notion.

We begin with an explanation of that mysterious “convergent sb-sequence”: Let K be a **compact** subset of \mathbb{R}^d . As you know from real analysis this means that K is closed and bounded. Consider a sequence $(p_j)_{j \in \mathbb{N}}$ of points in K : $p_j \in K$ for all $j \in \mathbb{N}$. Then we know that

There is at least one point $p^* \in K$ such that **every** open neighborhood of p^* contains **infinitely many** points of the sequence:

$$\exists p^* \in K: \forall \epsilon > 0: \exists I_\epsilon \subset \mathbb{N}: \#I_\epsilon = \infty \wedge \|p_j - p^*\| < \epsilon \quad \forall j \in I_\epsilon .$$

This is equivalent to the statement:

The sequence $(p_j)_{j \in \mathbb{N}}$ contains a convergent sub-sequence.

Next, we give an example that bounded closed subsets of infinite-dimensional spaces need not enjoy that special “compactness property” of bounded closed subsets in finite dimensions. For the interval $\Omega =]0, 2\pi[$ we consider

$$K := \{v \in L^2(]0, 2\pi[): \|v\|_{L^2(]0, 2\pi[)} \leq 1\} \quad (\text{bounded and closed in } L^2(]0, 2\pi[)),$$

and the sequence $\left(v_j := \{t \mapsto \frac{1}{\sqrt{2\pi}} \sin(jt)\}\right)_{j \in \mathbb{N}} \subset K$.

Elementary computations yield

$$\int_0^{2\pi} v_j(t) v_k(t) dt = \begin{cases} \frac{1}{2} & \text{for } j = k, \\ 0 & \text{for } j \neq k, \end{cases} \Rightarrow \|v_j - v_k\|_{L^2(\Omega)} = 1 \quad \forall j, k \in \mathbb{N}, j \neq k .$$

All members of the sequence have an $L^2(]0, 2\pi[)$ -distance 1! Thus it cannot have a convergent sub-sequence in $L^2(]0, 2\pi[)$. This does not supply a counterexample for Thm. 1.8.0.28, because $\|v_j\|_{H^1(]0, 2\pi[)} \rightarrow \infty$ for $j \rightarrow \infty$; the sequence is **not bounded** in $H^1(]0, 2\pi[)$.

It is also instructive to understand, why the assumption that Ω is bounded is necessary in Thm. 1.8.0.28. Consider $\Omega = \mathbb{R}$ and the sequence of “tent functions”

$$\varphi_j := \left\{ t \mapsto \begin{cases} 1 + t - 2j & \text{for } 2j - 1 \leq t \leq 2j, \\ 1 + 2j - t & \text{for } 2j \leq t \leq 2j + 1, \\ 0 & \text{elsewhere.} \end{cases} \right\}, \quad j \in \mathbb{N} .$$

These are continuous piecewise smooth functions, and, as we have seen in § 1.3.4.22, they all belong to $H^1(\mathbb{R})$. Their supports do not overlap and, thus, $|\varphi_j - \varphi_k|_{H^1(\Omega)} = 2$ whenever $j \neq k$. As before, this rules out the existence of a convergent sub-sequence. \square

Review question(s) 1.8.0.32 (Elliptic variational problems)

(Q1.8.0.32.A) What is the meaning and relationship of *classical* and *weak* solutions of 2nd-order elliptic boundary value problems?

(Q1.8.0.32.B) State the linear variational equation and the corresponding quadratic minimization problem related to the second-order linear elliptic boundary value problem

$$-\operatorname{div} \left(\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \operatorname{grad} u \right) + u = 0 \quad \text{in } \Omega \subset \mathbb{R}^2, \quad u = 1 \quad \text{on } \partial\Omega .$$

(Q1.8.0.32.C) Which linear variational problem gives the weak form of the boundary value problem

$$-\Delta u = 0 \quad \text{in } \Omega \subset \mathbb{R}^2 , \quad \begin{aligned} -\mathbf{grad} u \cdot \mathbf{n} &= u - 1 && \text{on } \Gamma_0 , \\ u &= 0 && \text{on } \Gamma_1 , \end{aligned}$$

where $\partial\Omega = \bar{\Gamma}_0 \cup \Gamma_1$ is a partition of the boundary $\partial\Omega$?

(Q1.8.0.32.D) Consider the pure Neumann boundary value problem

$$-\operatorname{div}(\mathbf{A}(x) \mathbf{grad} u) = f \quad \text{in } \Omega \subset \mathbb{R}^d , \quad \mathbf{A} \mathbf{grad} u \cdot \mathbf{n} = h \quad \text{on } \partial\Omega .$$

State the compatibility conditions on the data f and h that is necessary for existence of weak solutions. Give a physical interpretation in the context of stationary heat conduction.

(Q1.8.0.32.E) Consider the partial differential equation

$$\mathbf{grad} \operatorname{div} \mathbf{u} + c(x) \mathbf{u} = \mathbf{f} \quad \text{in } \Omega \subset \mathbb{R}^3 ,$$

where $c : \Omega \rightarrow \mathbb{R}$ is a bounded and uniformly positive definite coefficient function. Derive the formal variational formulations for boundary value problems for this PDE when equipped with the boundary conditions

1. $\mathbf{u} \cdot \mathbf{n} = 0$ on $\partial\Omega$, where \mathbf{n} is the exterior unit normal vectorfield on $\partial\Omega$.
2. $\operatorname{div} \mathbf{u} = 0$ on $\partial\Omega$.

△

1.9 Essential and Natural Boundary Conditions

Let us take a closer look at the boundary conditions that we have found so far.

§1.9.0.1 (A synopsis of scalar 2nd-order linear elliptic boundary value problems) BVPs in strong and weak form, see Section 1.7 for a discussion of boundary conditions and both Section 1.8 and Section 1.5 for how to connect weak and strong forms.

☛ 2nd-order elliptic Dirichlet problem:

$$-\operatorname{div}(\alpha(x) \mathbf{grad} u) = f \quad \text{in } \Omega , \quad u = g \quad \text{on } \partial\Omega . \quad (1.5.3.8)$$

with variational formulation

$$\begin{aligned} u &\in H^1(\Omega) , \\ u &= g \quad \text{on } \partial\Omega : \quad \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) \, dx = \int_{\Omega} f(x) v(x) \, dx \quad \forall v \in H_0^1(\Omega) . \end{aligned} \quad (1.4.2.4)$$

☛ 2nd-order elliptic Neumann problem:

$$-\operatorname{div}(\alpha(x) \mathbf{grad} u) = f \quad \text{in } \Omega , \quad (\alpha(x) \mathbf{grad} u) \cdot \mathbf{n} = h \quad \text{on } \partial\Omega . \quad (1.9.0.2)$$

with variational formulation

$$u \in H_*^1(\Omega) : \quad \int_{\Omega} \alpha(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx + \int_{\partial\Omega} h v \, dS \quad \forall v \in H_*^1(\Omega) . \quad (1.8.0.16)$$

☛ 2nd-order elliptic mixed Neumann-Dirichlet problem, see Ex. 1.5.3.11:

$$-\operatorname{div}(\alpha(x) \mathbf{grad} u) = f \quad \text{in } \Omega , \quad \begin{aligned} u &= g && \text{on } \Gamma_0 \subset \partial\Omega , \\ (\alpha(x) \mathbf{grad} u) \cdot \mathbf{n} &= h && \text{on } \partial\Omega \setminus \Gamma_0 . \end{aligned} \quad (1.9.0.3)$$

with variational formulation

$$\begin{aligned} u \in H^1(\Omega), \quad & \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x)v(x) dx + \int_{\partial\Omega \setminus \Gamma_0} h v dS \quad (1.9.0.4) \\ u = g \text{ on } \Gamma_0, \quad & \end{aligned}$$

for all $v \in H^1(\Omega)$ with $v|_{\Gamma_0} = 0$. □

Natural and essential boundary conditions

A pattern has emerged: In the variational formulations of 2nd-order elliptic BVPs of Section 1.8:

Dirichlet boundary conditions are *directly imposed* on trial space and (in homogeneous form) on test space.

Terminology: **essential boundary conditions**

Neumann boundary conditions are enforced *only* through the variational equation.

Terminology: **natural boundary conditions**

The attribute “natural” has been coined, because Neumann boundary conditions “naturally” emerge when removing constraints on the boundary, as we have seen for the partially free membrane of Ex. 1.5.3.11.

§1.9.0.6 (Admissible Dirichlet data) In the framework of variational problems in Sobolev spaces the minimum requirement for “Dirichlet data” $g : \partial\Omega \mapsto \mathbb{R}$ is that (1.5.3.8):

$$\text{there is } u \in H^1(\Omega) \text{ such that } u|_{\partial\Omega} = g$$

There is a criterion analogous to Thm. 1.3.4.23 that permits us to distinguish valid Dirichlet data:

If $g : \partial\Omega \mapsto \mathbb{R}$ is piecewise continuously differentiable (and bounded with bounded piecewise derivatives), then it can be extended to an $u_0 \in H^1(\Omega)$, if and only if it is continuous on $\partial\Omega$.

Important conclusion:

Dirichlet boundary values have to be continuous

This is also stipulated by physical insight, e.g. in the case of the taut membrane model of Section 1.2.1: discontinuous displacement on $\partial\Omega$ would entail ripping apart the membrane. □

Supplement 1.9.0.7 ($H^{\frac{1}{2}}(\partial\Omega)$) The set of admissible Dirichlet data $g : \partial\Omega \rightarrow \mathbb{R}$ can also be characterized as an energy space in the sense of Section 1.3.1, albeit with a strange-looking norm:

$$\|g\|_{H^{\frac{1}{2}}(\partial\Omega)}^2 := \int_{\partial\Omega} |g(x)|^2 dS(x) + \int_{\partial\Omega} \int_{\partial\Omega} \frac{|g(x) - g(y)|^2}{\|x - y\|^d} dS(x)dS(y). \quad (1.9.0.8)$$

The common notation for this energy space in mathematics is $H^{\frac{1}{2}}(\partial\Omega)$. Its properties are rather peculiar and not relevant for understanding the numerical methods presented in this course. □

§1.9.0.9 (Admissible Neumann data) In the variational problem (1.8.0.16) Neumann data $h : \partial\Omega \mapsto \mathbb{R}$ enter through the linear form on the right hand side

$$\ell(v) := \int_{\Omega} f(x)v(x) dx + \int_{\partial\Omega} h(x)v(x) dS(x).$$

Remember the discussion in the beginning of Section 1.3, also Ex. 1.2.3.45: we have to establish that ℓ is continuous on $H_*^1(\Omega)$ defined in (1.8.0.15). This is sufficient, because the coefficient function κ is uniformly positive and bounded, see (1.6.0.6). Thus, the energy $\|\cdot\|_a$ associated with the bilinear form

$$a(u, v) = \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx$$

can be bounded from above and below by $|\cdot|_{H^1(\Omega)}$.

Continuity of the boundary contribution to the right hand side linear functional ℓ is ensured by a **trace theorem**:

Theorem 1.9.0.10. Multiplicative trace inequality

$$\exists C = C(\Omega) > 0: \|u\|_{L^2(\partial\Omega)}^2 \leq C \|u\|_{L^2(\Omega)} \cdot \|u\|_{H^1(\Omega)} \quad \forall u \in H^1(\Omega).$$

Proof. 1D: To elucidate idea we first give the proof for $d = 1, \Omega = [0, 1]$:

As in the proof of Thm. 1.3.4.17 and Thm. 1.8.0.20, we employ a density argument and assume that u is sufficiently smooth, $u \in C^1([0, 1])$.

By the fundamental theorem of calculus (1.5.1.6):

$$\begin{aligned} u(1)^2 &= \int_0^1 \frac{dw}{d\xi}(x) \, dx, \quad \text{with } w(\xi) := \xi u^2(\xi), \\ \Rightarrow u(1)^2 &= \int_0^1 u^2(x) + 2u(x) \frac{du}{dx}(x)x \, dx. \end{aligned}$$

Then use the Cauchy-Schwarz inequality (1.3.4.15):

$$u(1)^2 \leq \int_0^1 u^2(x) \, dx + 2 \int_0^1 |x| |u(x)| \left| \frac{du}{dx}(x) \right| \, dx \leq \|u\|_0^2 + 2\|u\|_0 \left\| \frac{du}{dx} \right\|_0.$$

A similar estimate holds for $u(0)^2$.

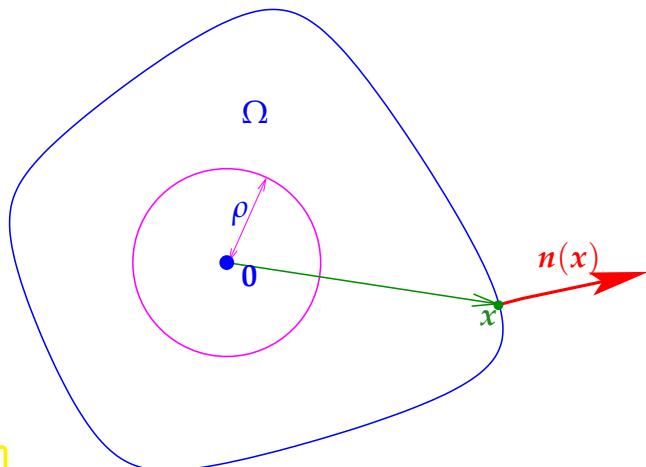


Fig. 51

Next, we treat general $d \in \mathbb{N}$. Temporarily we make the following assumption

There is a $\rho > 0$ so that Ω is **star-shaped** with respect to every point in $B_\rho(\mathbf{0}) := \{x \in \mathbb{R}^d, \|x\| < \rho\}$:

$$\forall x \in \Omega, z \in B_\rho(\mathbf{0}): \{\xi x + (1 - \xi)z, 0 \leq \xi \leq 1\} \subset \Omega. \quad (1.9.0.11)$$

In other words, every point of the boundary $\partial\Omega$ “can be seen” from every point inside the circle with radius ρ around the origin.

Elementary geometric considerations reveal a consequence of (1.9.0.11):

$$\exists \delta > 0: x \cdot n(x) \geq \delta \quad \forall x \in \partial\Omega. \quad (1.9.0.12)$$

This means that the vector \mathbf{x} and the exterior unit normal $\mathbf{n}(\mathbf{x})$ “point in the same direction”. The key idea is to apply Gauss’ theorem to a smartly chosen function.

Theorem 1.5.2.4. Gauss’ theorem → [Str09, Sect. 8.8]

With $\mathbf{n} : \partial\Omega \mapsto \mathbb{R}^d$ denoting the *exterior unit normal vectorfield* on $\partial\Omega$ and dS indicating integration over a surface, we have

$$\int_{\Omega} \operatorname{div} \mathbf{j}(\mathbf{x}) d\mathbf{x} = \int_{\partial\Omega} \mathbf{j}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) dS(\mathbf{x}) \quad \forall \mathbf{j} \in (\mathcal{C}_{pw}^1(\overline{\Omega}))^d. \quad (1.5.2.5)$$

We apply this “fundamental theorem of calculus in higher dimensions” with $\mathbf{j}(\mathbf{x}) := \mathbf{x} u(\mathbf{x})^2$, where $u \in \mathcal{C}^1(\overline{\Omega})$:

$$\begin{aligned} \|u\|_{L^2(\partial\Omega)}^2 &= \int_{\partial\Omega} u^2(\mathbf{x}) dS(\mathbf{x}) \\ &\stackrel{(1.9.0.12)}{\leq} \delta^{-1} \int_{\Omega} u^2(\mathbf{x}) \mathbf{x} \cdot \mathbf{n}(\mathbf{x}) dS(\mathbf{x}) \stackrel{\text{Thm. 1.5.2.4}}{=} \delta^{-1} \int_{\Omega} \operatorname{div}(u^2(\mathbf{x}) \mathbf{x}) d\mathbf{x} \\ &= \delta^{-1} \int_{\Omega} d u^2(\mathbf{x}) + 2u(\mathbf{x}) \operatorname{grad} u(\mathbf{x}) \cdot \mathbf{x} d\mathbf{x} \\ &\leq \frac{d}{\delta} \int_{\Omega} u^2(\mathbf{x}) + |u(\mathbf{x})| \|\operatorname{grad} u(\mathbf{x})\| \|\mathbf{x}\| d\mathbf{x} \\ &\leq \frac{2d}{\delta} \left(\int_{\Omega} |u(\mathbf{x})|^2 d\mathbf{x} \right)^{1/2} \left(\int_{\Omega} |u(\mathbf{x})|^2 + \operatorname{diam}(\Omega)^2 \|\operatorname{grad} u(\mathbf{x})\|^2 d\mathbf{x} \right)^{1/2}. \end{aligned}$$

In the last step we used the Cauchy-Schwarz inequality.

In order to dispense with the assumption of star-shapedness, we split Ω into sub-domains Ω_j , $j = 1, \dots, M$, $M \in \mathbb{N}$, all of which are star-shaped with respect to some balls and satisfy

$$\Omega_j \cap \Omega_\ell = \emptyset \quad \text{if } k \neq \ell, \quad \overline{\Omega} = \overline{\Omega}_1 \cup \dots \cup \overline{\Omega}_M.$$

Thus, appealing to the estimate from above, with $C > 0$ depending on the shapes of the sub-domains,

$$\|u\|_{L^2(\partial\Omega)}^2 \leq \sum_{j=1}^M \|u\|_{L^2(\partial\Omega_j)}^2 \leq C \sum_{j=1}^M \|u\|_{L^2(\Omega_j)} \|u\|_{H^1(\Omega_j)} \leq C \|u\|_{L^2(\Omega)} \|u\|_{H^1(\Omega)}.$$

The last estimate amounts to an application of the Cauchy-Schwarz inequality in \mathbb{R}^M . Then density arguments finish the proof. \square

Now we can combine

- ◆ the Cauchy-Schwarz inequality (1.3.4.15) on $\partial\Omega$,
- ◆ the 2nd Poincaré-Friedrichs inequality of Thm. 1.8.0.20,
- ◆ the multiplicative trace inequality of Thm. 1.9.0.10:

$$\begin{aligned} \int_{\partial\Omega} h v dS &\stackrel{(1.3.4.15)}{\leq} \|h\|_{L^2(\partial\Omega)} \|v\|_{L^2(\partial\Omega)} \stackrel{\text{Thm. 1.9.0.10}}{\leq} C \|h\|_{L^2(\partial\Omega)} \|v\|_{H^1(\Omega)} \\ &\stackrel{\text{Thm. 1.8.0.20}}{\leq} C \|h\|_{L^2(\partial\Omega)} |v|_{H^1(\Omega)} \quad \forall v \in H_*^1(\Omega). \end{aligned}$$



$h \in L^2(\partial\Omega)$ provides valid Neumann data for the 2nd order elliptic BVP (1.9.0.2).

In particular, Neumann data h can be *discontinuous*. □

Review question(s) 1.9.0.13 (Essential and natural boundary conditions)

(Q1.9.0.13.A) For a scalar 2nd-order elliptic boundary value problem for $-\operatorname{div}(\alpha(x) \operatorname{grad} u) = f$ the Robin boundary conditions read, cf. Ex. 1.7.0.5,

$$\alpha(x) \operatorname{grad} u + \gamma(x)u = 0 \quad \text{on } \partial\Omega ,$$

with $\gamma : \partial\Omega \rightarrow \mathbb{R}$ uniformly positive. Are these boundary conditions essential or natural.

(Q1.9.0.13.B) Describe the minimal regularity of Dirichlet and Neumann data for scalar 2nd-order elliptic BVPs on $\Omega \subset \mathbb{R}^d$ in terms of classical smoothness spaces $C_{\text{pw}}^k(\partial\Omega)$.

(Q1.9.0.13.C) Appealing to the following estimate

Theorem 1.9.0.10. Multiplicative trace inequality

$$\exists C = C(\Omega) > 0: \|u\|_{L^2(\partial\Omega)}^2 \leq C \|u\|_{L^2(\Omega)} \cdot \|u\|_{H^1(\Omega)} \quad \forall u \in H^1(\Omega) .$$

prove that for the linear variational problem

$$u \in H^1(\Omega): \int_{\Omega} \operatorname{grad} u \cdot \operatorname{grad} v \, dx + \int_{\partial\Omega} uv \, dS = \int_{\partial\Omega} hv \, dS \quad \forall v \in H^1(\Omega) ,$$

both the bilinear form and the right-hand side linear form are continuous on $H^1(\Omega)$.

(Q1.9.0.13.D) In the case of the PDE $-\operatorname{grad} \operatorname{div} \mathbf{u} + \mathbf{u} = \mathbf{f}$ on $\Omega \subset \mathbb{R}^3$, what are essential, what are natural boundary conditions. To answer this questions apply Thm. 1.5.2.7 and determine and study the boundary terms arising from it.

(Q1.9.0.13.E) Another stability issue: How does a perturbation (measured in $L^2(\partial\Omega)$ -norm) $\delta h \in L^2(\partial\Omega)$ of the Neumann data $h \in L^2(\partial\Omega)$ in the linear variational problem

$$u \in H_*^1(\Omega): \int_{\Omega} \alpha(x) \operatorname{grad} u \cdot \operatorname{grad} v \, dx = \int_{\Omega} fv \, dx + \int_{\partial\Omega} \color{red} h v \, dS \quad \forall v \in H_*^1(\Omega) .$$

impact the energy norm of the solution of that variational problem? The coefficient function $\alpha : \Omega \rightarrow \mathbb{R}^{3,3}$ can be assumed to be uniformly positive definite,

$$\exists 0 < \alpha^- \leq \alpha^+ < \infty: \alpha^- \|\mathbf{z}\|^2 \leq (\alpha(x)\mathbf{z}) \cdot \mathbf{z} \leq \alpha^+ \|\mathbf{z}\|^2 \quad \forall \mathbf{z} \in \mathbb{R}^3, \forall x \in \Omega .$$

(Q1.9.0.13.F) For a bounded polygon $\Omega \subset \mathbb{R}^2$, we consider the linear variational problem

$$u \in H^1(\Omega): \int_{\Omega} e^{\|\mathbf{x}\|} \operatorname{grad} u(\mathbf{x}) \cdot \operatorname{grad} v(\mathbf{x}) \, dx = \int_{\partial\Omega} x_1 v(\mathbf{x}) \, dS, \quad \forall v \in H^1(\Omega) . \quad (1.9.0.14)$$

State the boundary value problem satisfied by sufficiently smooth solutions of Eq. (1.9.0.14). Write down a domain Ω for which Eq. (1.9.0.14) has a solution. △

Learning outcomes

After having studied this chapter you should (be able to)

- be familiar with a few mathematical models whose core is the minimization of quadratic energy functionals over infinite-dimensional function spaces,
- convert a quadratic minimization problem into a linear variational problem,
- understand and verify necessary conditions for the existence of unique solutions of a quadratic minimization problem,
- know the norms of the Sobolev spaces $L^2(\Omega)$, $H^1(\Omega)$, and $H_0^1(\Omega)$ and how to use them in the statement of variational problems,
- state the continuity featured by piecewise smooth functions in a Sobolev space,
- appreciate the importance of the continuity in the energy norm of right hand side functionals of variational problems,
- extract a PDE and boundary conditions from a variational problem using integration by parts,
- recast a boundary value problem for a 2nd-order PDE in variational form (using suitable Sobolev spaces).
- tell which boundary conditions make sense for a given 2nd-order PDE.
- distinguish essential and natural boundary conditions for a PDE in variational form.
- know sufficient conditions for admissible Dirichlet- and Neumann data in the case of scalar 2nd-order elliptic variational problems.
- know the compatibility conditions for the data in the case of a pure Neumann problem.

Bibliography

- [Eva98] L.C. Evans. *Partial differential equations*. Vol. 19. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 1998 (cit. on pp. 55, 92, 127).
- [Hac92] W. Hackbusch. *Elliptic Differential Equations. Theory and Numerical Treatment*. Vol. 18. Springer Series in Computational Mathematics. Berlin: Springer, 1992 (cit. on p. 92).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 57, 61, 67, 69, 73, 74).
- [Rud86] W. Rudin. *Real and Complex Analysis*. 3rd. McGraw–Hill, 1986 (cit. on p. 83).
- [Str09] M. Struwe. *Analysis für Informatiker*. Lecture notes, ETH Zürich. 2009 (cit. on pp. 61, 74, 78, 84, 91, 104, 106, 132).
- [Wer95] D. Werner. *Funktionalanalysis*. Berlin: Springer, 1995 (cit. on p. 85).

Chapter 2

Finite Element Methods (FEM)

Contents

2.1	Introduction	137
2.1.1	Discretization	138
2.1.2	Focus: Variational Formulations of BVPs	138
2.1.3	Preview	140
2.2	Principles of Galerkin Discretization	141
2.2.1	Galerkin Discretization: First Step	142
2.2.2	Galerkin Discretization: Second Step	144
2.2.3	Galerkin Matrices	147
2.3	Case Study: Linear FEM for Two-Point Boundary Value Problems	150
2.3.1	Step I: Choice of Discrete Trial/Test Space	150
2.3.2	Step II: Choice of Basis	152
2.3.3	Formulas for Galerkin Matrices and Right-Hand-Side Vectors	153
2.4	Case Study: Triangular Linear FEM in Two Dimensions	160
2.4.1	Meshes in 2D: Triangulations	161
2.4.2	Linear Finite Element Space	163
2.4.3	Nodal Basis Functions	165
2.4.4	Sparse Galerkin Matrix	168
2.4.5	Computation of Galerkin Matrix	170
2.4.6	Computation of Right-Hand Side Vector	181
2.5	Building Blocks of General Finite Element Methods	188
2.5.1	Meshes	188
2.5.2	Polynomials	190
2.5.3	Basis Functions	192
2.6	Lagrangian Finite Element Methods	197
2.6.1	Simplicial Lagrangian FEM	198
2.6.2	Tensor-Product Lagrangian FEM	201
2.7	Implementation of Finite Element Methods	207
2.7.1	Mesh Generation and Mesh File Format	209
2.7.2	Mesh Information and Mesh Data Structures	216
2.7.3	Vectors and Matrices	228
2.7.4	Assembly Algorithms	230
2.7.5	Local Computations	249
2.7.6	Treatment of Essential Boundary Conditions	262
2.8	Parametric Finite Element Methods	271
2.8.1	Affine Equivalence	271
2.8.2	Example: Quadrilateral Lagrangian Finite Elements	276
2.8.3	Transformation Techniques	279
2.8.4	Application of Parametric FEM: Boundary Approximation	292

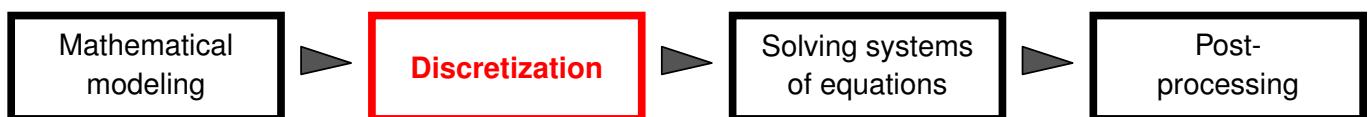
2.1 Introduction

Our goal is to solve linear scalar second-order elliptic boundary value problem as introduced in Chapter 1 in various forms, as a minimization problem (1.2.3.1), a linear variational problem like (1.4.2.4) (“weak form”), or in strong form (1.5.3.5). Only, in exceptional situations we can find an **analytic solution**, that is, a concrete formula for the unknown function $u : \Omega \rightarrow \mathbb{R}$. In all other cases, we have to put up with **approximate** solutions obtained by running a numerical algorithm on a computer:



2.1.1 Discretization

In this chapter we introduce the most important class of methods designed for the numerical solution of second-order elliptic boundary value problems. More precisely, these methods provide **discretizations** of boundary value problems, which is only one, but a crucial step in the numerical treatment of boundary value problems.

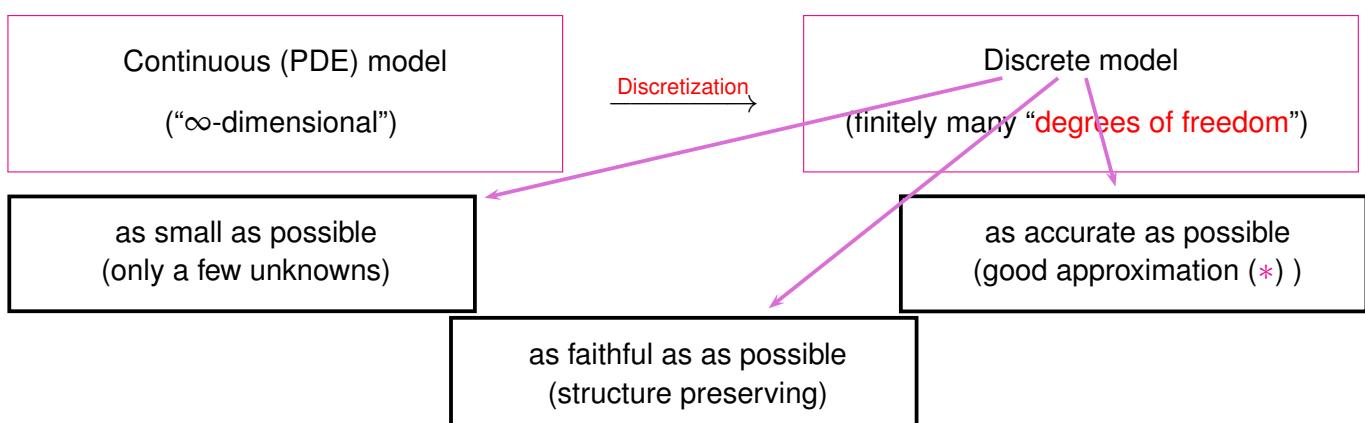


According to our understanding, PDE models inherently rely on infinite-dimensional state spaces. Thus it takes an infinite amount of information to characterize the solution. Since computers are finite automata, (numerical) algorithms can operate only on discrete models.

Definition 2.1.1.1. Discrete model

A **discrete** model for a physical system/phenomenon is a model, for which both data/parameters and states can be described by a **finite number** of real numbers.

The construction of meaningful discrete models from continuous models whose data/unknowns contain an infinite amount of information is the task of **discretization**:



(*): needs a measure for quality of a solution, usually a **norm** of the **error** where the error is regarded as the difference of exact/analytic and approximate solution.

Parlance: number of “degrees of freedom” $\hat{=}$ number of **doubles** required to describe discrete configuration space (usually agrees with number of “unknowns” in the discrete model.)

2.1.2 Focus: Variational Formulations of BVPs

This chapter is devoted to the discretization of *linear* second-order elliptic boundary value problems by means of the so-called finite element method. This methods tackles the BVPs in variational (“weak”) form, abstractly written as linear variational problem (LVP)

$$u \in \widehat{V}: \quad a(u, v) = l(v) \quad \forall v \in V_0, \quad (1.4.1.7)$$

with bi-/linear form a/l . Thus, the present chapter heavily relies on the material covered in Section 1.3, Section 1.8, and Section 1.9. The reader will not be able to grasp the ideas and arguments of this chapter, unless she or he is familiar with these foundation parts of the course.

Condensed into one line, the objective of this chapter is to supply tools for the discretization of (linear) variational problems posed on function spaces:



§2.1.2.1 (Targeted boundary value problems) We will restrict ourselves to *linear* 2nd-order elliptic variational problems on spatial domains $\Omega \in \mathbb{R}^d$, $d = 2, 3$, whose properties have been elaborated in § 1.2.1.14. Concretely, with source functions $f \in L^2(\Omega)$, we tackle the

- ☛ 2nd-order elliptic **Dirichlet problem**:

$$\begin{aligned} u &\in H^1(\Omega), \\ u &= g \text{ on } \partial\Omega: \quad \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x)v(x) dx \quad \forall v \in H_0^1(\Omega), \end{aligned} \quad (1.4.2.4)$$

with *continuous* (\rightarrow § 1.9.0.6) Dirichlet data $g \in C^0(\partial\Omega)$ and uniformly positive definite (\rightarrow Def. 1.2.2.9) diffusion tensor $\alpha \in (C_{pw}^0(\overline{\Omega}))^{d,d}$.

- ☛ 2nd-order elliptic **Neumann problems**:

$$u \in H_*^1(\Omega): \quad \int_{\Omega} (\alpha(x) \mathbf{grad} u) \cdot \mathbf{grad} v dx = \int_{\Omega} fv dx + \int_{\partial\Omega} hv dS \quad \forall v \in H_*^1(\Omega), \quad (1.8.0.16)$$

posed on the constrained Sobolev space of functions with vanishing mean

$$H_*^1(\Omega) := \{v \in H^1(\Omega): \int_{\Omega} v(x) dx = 0\}, \quad (1.8.0.15)$$

and with *piecewise continuous* (\rightarrow § 1.9.0.9) Neumann data $h \in C_{pw}^0(\partial\Omega)$ that satisfy the **compatibility condition**

$$-\int_{\partial\Omega} h(x) dS(x) = \int_{\Omega} f(x) dx. \quad (1.8.0.13)$$

A simpler version with homogeneous Neumann data and reaction term reads

$$u \in H^1(\Omega): \quad \int_{\Omega} \alpha(x) \mathbf{grad} u \cdot \mathbf{grad} v + c(x)uv dx = \int_{\Omega} fv dx \quad \forall v \in H^1(\Omega), \quad (2.1.2.2)$$

with *uniformly positive* reaction coefficient $c : \Omega \mapsto \mathbb{R}^+$, $c \in C_{pw}^0(\Omega)$, cf. (1.2.2.8), Def. 1.2.2.9:

$$\exists 0 < \gamma^- \leq \gamma^+ < \infty: \quad \gamma^- \leq c(x) \leq \gamma^+ \quad \text{for almost all } (\rightarrow \text{Suppl. 2.1.2.4}) \quad x \in \Omega. \quad (2.1.2.3)$$

The variational problem (2.1.2.2) is easier than (1.8.0.13), because no compatibility condition like (1.8.0.13) is needed to ensure existence and uniqueness of solutions.

The considerations in Section 1.8 and Section 1.9 established the following key properties of these variational problems:

The linear variational problems (1.4.2.4), (1.8.0.16), and (2.1.2.2) feature symmetric positive definite bilinear forms (\rightarrow Def. 1.2.3.27) and right hand side linear forms that are continuous (\rightarrow Def. 1.2.3.42) with respect to the energy norm (\rightarrow Def. 1.2.3.35).



existence and uniqueness of solutions (\rightarrow Rem. 1.3.3.9)

Please remember that all the variational problems are connected with quadratic minimization problems, see Section 1.2.3, Def. 1.2.3.11.

Supplement 2.1.2.4 (Almost all/almost everywhere) In (mathematical) articles on function spaces and variational formulations posed on them you will often encounter phrases like “almost all” or “almost everywhere”. The designate statements about point values of functions that remain true, if the function is changed on sets of points that “do not matter for integration”.

For instance, since in (2.1.2.2) the reaction coefficient c occurs only in an integrand, we do not care about it being positive on “sets of measure/volume zero” like lines in 2D or surfaces in 3D. Whether domains of integration are open or closed is immaterial, too.

Remark 2.1.2.5 (Data in procedural form) A developer of software for numerical simulation must not expect that the data (here the coefficient functions a , c , source function f , Dirichlet data g , Neumann data h) are given in closed form (as formulas). Instead they will usually be given in **procedural form** through sub-routines that merely allow point evaluations. This may be the only way to access the coefficient functions, because they may have been obtained, for instance, as results of another computation, or by interpolation from a table of measured values. The sub-routines might even be “black-box” library functions providing little information about how the function is actually evaluated.

In C++ data types providing functions are called **functors**. The two main ways of defining functors are the following, illustrated by sample code snippets, see also [Hip19, ??].

- (I) A functor is represented by a class supplying an evaluation operator: **operator ()**

C++ code 2.1.2.6: Example: a functor class definition

```

1 template<typename ReturnType, typename PointCoordinates>
2 class Function {
3     using value_type = ReturnType;
4     using arg_type = PointCoordinates;
5     Function(void);
6     // evaluation operator
7     value_type operator ()(const arg_type &x) const;
8     ....
9 };

```

- (II) A simple function can be defined through a **lambda function**, e.g.

C++ code 2.1.2.7: A simple lambda function

```

1 auto f = [&param](const PointCoordinates &x) -> double
2 { /* function body */ };

```

Recall that parameters can be passed to a lambda function through the capture list [\[\]](#).

A dedicated functor class will be preferred to a lambda function, if the evaluation is complicated or if it requires substantial pre-computations that can be done in the constructor of the class. [↳](#)

2.1.3 Preview

The next section introduces the simple but powerful idea underlying the **Galerkin discretization** of linear variational problems (\rightarrow Def. 1.4.1.6). This is done on an abstract level, in order to convey the main ideas.

Section 2.3 marks our first encounter with the finite element method in the simplest conceivable setting, in one-dimension for two-point second-order elliptic boundary value problem like the elastic string model discussed in Section 1.5.1. We will present the simplest finite element method making use of approximation by piecewise linear continuous functions.

The following Section 2.4 is devoted to extending the linear finite element method from one to two dimensions. The leap from $d = 1$ to $d = 2$ will encounter additional difficulties and many new aspects will emerge. This will entail a more detailed discussion of algorithms and data structures, demonstrated with a model C++ code based on EIGEN.

In Section 2.5 we return to a more abstract level and learn about fundamental aspects and characteristics of finite element methods (FEM). Everything will be made concrete again in Section 2.5, in which the class of so-called Lagrangian finite element methods for 2nd-order linear variational problems on bounded spatial domains Ω in two and three dimensions will be presented in full generality.

In Section 2.7 we dip into details of algorithms for FEM and their implementation in C++. Simultaneously, we introduce the LEHRFEM++ finite element library as a model code and environment for solving BVPs with finite elements. We will regularly inspect codes to connect mathematical ideas with their algorithmic realizations.

Finally, Section 2.8 is devoted to another key conceptual and algorithmic approach to FEM: constructions based on mappings and pullbacks. This will mesh nicely with the use of local quadrature rules.

2.2 Principles of Galerkin Discretization



Video tutorial for Section 2.2: Principles of Galerkin Discretization: (47 minutes)
[Download link](#), [tablet notes](#)

§2.2.0.1 (Recalled from Section 1.4: linear variational problems, LVPs) In this section we adopt an abstract perspective: Targets of discretization will be **linear variational problems** (1.4.1.7) of the special form

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0, \quad (2.2.0.2)$$

- ◆ $V_0 \hat{=} \text{a vector space (Hilbert space) (usually a Sobolev space} \rightarrow \text{Section 1.3) with norm } \|\cdot\|_V,$
- ◆ $a(\cdot, \cdot) \hat{=} \text{bilinear form, } \textcolor{red}{continuous} (\rightarrow \text{Def. 1.2.3.42}) \text{ on } V_0,$ which means

$$\exists C > 0: \quad |a(u, v)| \leq C \|u\|_V \|v\|_V \quad \forall u, v \in V_0. \quad (2.2.0.3)$$

- ◆ $\ell \hat{=} \text{continuous linear form in the sense of Def. 1.2.3.42, cf. (1.2.3.40),}$

$$\exists C > 0: \quad |\ell(v)| \leq C \|v\|_V \quad \forall v \in V_0. \quad (2.2.0.4)$$

The importance of this *continuity* is discussed in the beginning of Section 1.2.3.4, see also Ex. 1.2.3.45. (The C s in (2.2.0.3) and (2.2.0.4) are so-called “generic constants”, whose values need not agree though they are designated by the same symbol, see Rem. 3.3.5.8 below.)

If \mathbf{a} is symmetric and positive definite (\rightarrow Def. 1.2.3.27), we may choose $\|\cdot\|_V := \|\cdot\|_{\mathbf{a}}$, the “energy norm” of Def. 1.2.3.35. By the Cauchy-Schwarz inequality of Thm. 0.3.1.19 continuity of \mathbf{a} w.r.t. $\|\cdot\|_{\mathbf{a}}$ is clear. \square

Remark 2.2.0.5 (Linear variational problems on affine spaces) Many of the linear variational problems we saw in Chapter 1, in particular the Dirichlet problem (1.4.2.4) with general boundary data, go beyond (2.2.0.2) in that they are posed on an affine space (\rightarrow Def. 1.2.3.13) $\widehat{V} := u_0 + V_0$, $u_0 \in V$, V a “hold-all” vector space, of which V_0 is a subspace, cf. Def. 1.4.1.6:

$$u \in \widehat{V}: \quad \mathbf{a}(u, v) = \ell(v) \quad \forall v \in V_0, \quad (2.2.0.6)$$

In this case we expect (2.2.0.3) to hold on the large space V :

$$\exists C > 0: \quad |\mathbf{a}(u, v)| \leq C \|u\|_V \|v\|_V \quad \forall u, v \in V. \quad (2.2.0.7)$$

In the concrete case of (1.5.3.5) we have

$$V = H^1(\Omega), \quad u_0 \in H^1(\Omega) \quad \text{such that} \quad u_0|_{\partial\Omega} = g, \quad V_0 = H_0^1(\Omega).$$

We have seen in § 1.4.1.9 how the linear variational problem (2.2.0.6) can be converted into the form (2.2.0.2) with a modified right hand side functional through the “offset function trick”. It gives us

$$w \in V_0: \quad \mathbf{a}(w, v) = \ell(v) - \mathbf{a}(u_0, v) \quad \forall v \in V_0. \quad (2.2.0.8)$$

If and only if $w \in V_0$ solves (2.2.0.8), then $w + u_0 \in \widehat{V}$ will solve (2.2.0.6). This equivalence justifies the focus on the simpler case (2.2.0.2). \square

2.2.1 Galerkin Discretization: First Step



The simple idea of the first step of Galerkin discretization:

Replace the infinite-dimensional function space V_0 in the linear variational problem (2.2.0.2) with a finite-dimensional subspace $V_{0,h} \subset V_0$.

Thus we arrive at a restriction of (1.4.1.7) to $V_{0,h}$:

Discrete (linear) variational problem (DVP):

$$u_h \in V_{0,h}: \quad \mathbf{a}(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h}. \quad (2.2.1.1)$$

the Galerkin solution

Remark 2.2.1.2 (Notation for “discrete entities”) Note that a subscript tag h distinguishes “discrete functions/quantities”, that is, functions/operators etc. that are associated with a finite dimensional space. The symbol h has its origin in a widely used notation for the stepsize/meshwidth underlying a discretization, see § 2.3.1.3 below. \square

Remark 2.2.1.3 (Discrete quadratic minimization problems) In this remark assume that \mathbf{a} is symmetric and positive definite (\rightarrow Def. 1.2.3.27), that is it induces an “energy norm” $\|\cdot\|_{\mathbf{a}}$ (\rightarrow Def. 1.2.3.35), and that the vector space V_0 equipped with the norm $\|\cdot\|_{\mathbf{a}}$ is a Hilbert space (\rightarrow Def. 1.3.3.4).

In this setting, according to Thm. 1.4.1.8, we have the equivalence of the quadratic minimization problem (\rightarrow Def. 1.2.3.11)

$$u = \underset{w \in V_0}{\operatorname{argmin}} J(w) \quad , \quad J(w) := \frac{1}{2} \mathbf{a}(w, w) - \ell(w), \quad (2.2.1.4)$$

and of the linear variational problem

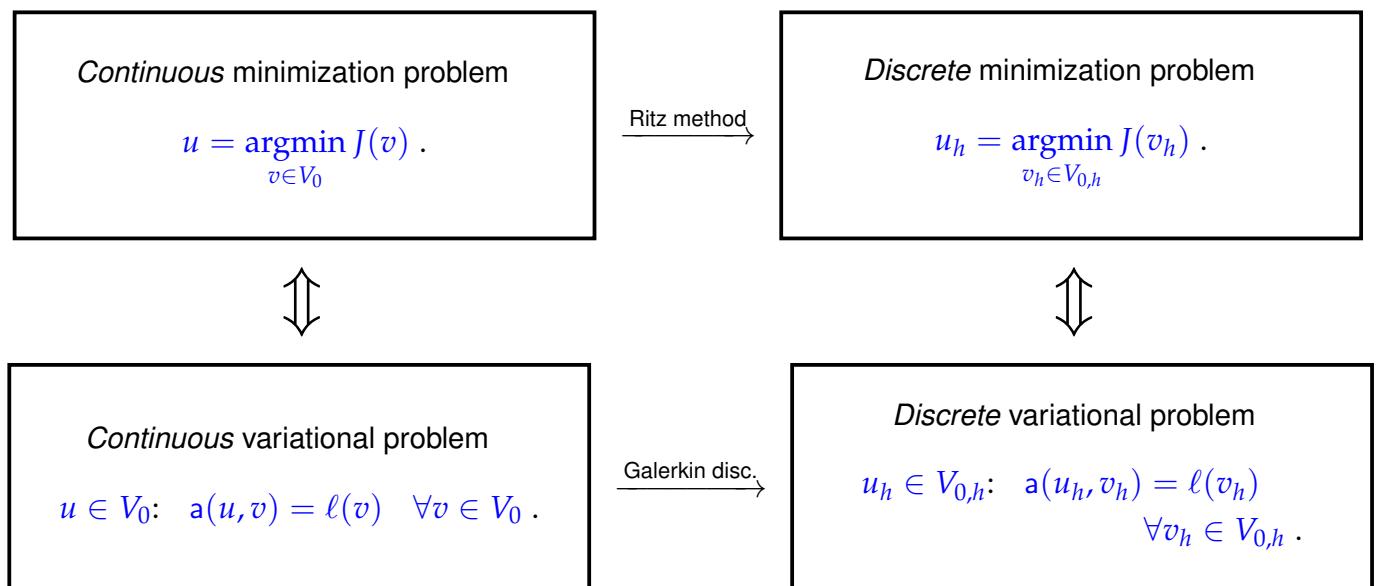
$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0. \quad (2.2.0.2)$$

According to Thm. 1.3.3.6 for both (2.2.1.4) and (2.2.0.2) unique solutions exist and they agree.

Exactly the same proof as that of Thm. 1.4.1.8 shows that this equivalence carries over to the discrete context.

Solving the discrete variational problem (2.2.1.1) amounts to determining a minimizer of J over the finite dimensional subspace $V_{0,h} \subset V_0$.

In other words, the Galerkin discretization of abstract (linear) variational problem (2.2.0.2) corresponds to discretizing a minimization problem over a functions space by considering it on a finite-dimensional subspace. This approach is often called the **Ritz method** [PR05, Sect. 10.2.2].



Terminology: Problems on infinite-dimensional spaces are often called “continuous”, those on finite-dimensional spaces “discrete”, remember Def. 2.1.1.1.

In light of Thm. 1.2.3.44 the equivalence discussed in the previous remark has a straightforward consequence for the case of s.p.d. a inducing an energy norm $\|\cdot\|_a$ via $\|v\|_a^2 := a(v, v)$, Def. 1.2.3.35.

Theorem 2.2.1.5. Existence and uniqueness of solutions of discrete variational problems

If the bilinear form $a : V_0 \times V_0 \mapsto \mathbb{R}$ is symmetric and positive definite (\rightarrow Def. 1.2.3.27) and the linear form $\ell : V_0 \mapsto \mathbb{R}$ is continuous in the sense of

$$\exists C_\ell > 0: \quad |\ell(u)| \leq C_\ell \|u\|_a \quad \forall u \in V_0, \quad (1.2.3.40)$$

then the discrete variational problem (2.2.1.1) has a unique **Galerkin solution** $u_h \in V_{0,h}$ that satisfies the **energy estimate**

$$\|u_h\|_a \leq \sup_{v_h \in V_{0,h}} \frac{|\ell(v_h)|}{\|v_h\|_a} \leq C_\ell. \quad (2.2.1.6)$$

Proof. Uniqueness of \mathbf{u}_h is clear: Similar to the proof of Thm. 1.2.3.31,

$$\begin{aligned} \mathbf{a}(\mathbf{u}_h, \mathbf{v}_h) &= \ell(\mathbf{v}_h) \quad \forall \mathbf{v}_h \in V_{0,h} \\ \mathbf{a}(\mathbf{w}_h, \mathbf{v}_h) &= \ell(\mathbf{v}_h) \quad \forall \mathbf{v}_h \in V_{0,h} \quad \Rightarrow \quad \mathbf{a}(\mathbf{u}_h - \mathbf{w}_h, \mathbf{v}_h) = 0 \quad \forall \mathbf{v}_h \in V_{N,0} \\ \mathbf{v}_h := \mathbf{u}_h - \mathbf{w}_h \in V_{0,h} \quad &\| \mathbf{u}_h - \mathbf{w}_h \|_a = 0 \quad \stackrel{\text{a.s.p.d.}}{\Rightarrow} \quad \mathbf{u}_h - \mathbf{w}_h = 0. \end{aligned}$$

The discrete linear variational problem (2.2.1.1) is set in the *finite-dimensional* space $V_{0,h}$. Thus, uniqueness of solutions is equivalent to existence of solutions (\rightarrow linear algebra).

If you do not like this abstract argument, wait and see the equivalence of (2.2.1.1) with a linear system of equations. It will turn out that under the assumptions of the theorem, the resulting system matrix will be symmetric and positive definite in the sense of [Hip19, ??], Ex. 1.2.3.28.

The estimate (2.2.1.6) is immediate from setting $\mathbf{v}_h := \mathbf{u}_h$ in (2.2.1.1)

$$|\mathbf{a}(\mathbf{u}_h, \mathbf{u}_h)| = |\ell(\mathbf{u}_h)| \leq C_\ell (\mathbf{a}(\mathbf{u}_h, \mathbf{u}_h))^{1/2}.$$

Then cancel a square root of $\mathbf{a}(\mathbf{u}_h, \mathbf{u}_h) = \| \mathbf{u}_h \|_a^2$. This holds for any C_ℓ satisfying (1.2.3.40), which makes it possible to switch to the infimum in (2.2.1.6). \square

Remark 2.2.1.7 (Discrete variational problems in affine spaces) We revisit the linear variational problem examined in Rem. 2.2.0.5:

$$\mathbf{u} \in \widehat{V} := \mathbf{u}_0 + V_0: \quad \mathbf{a}(\mathbf{u}, \mathbf{v}) = \ell(\mathbf{v}) \quad \forall \mathbf{v} \in V_0, \quad (2.2.0.6)$$

If \mathbf{u}_0 is a simple function and allows a closed-form representation, that is, \mathbf{u}_0 is given by a formula that can be implemented easily, we may choose as a trial space the finite-dimensional affine space $\widehat{V}_h := \mathbf{u}_0 + V_{0,h}$. We end up with the slightly more general discrete variational problem

$$\mathbf{u}_h \in \widehat{V}_h: \quad \mathbf{a}(\mathbf{u}_h, \mathbf{v}_h) = \ell(\mathbf{v}_h) \quad \forall \mathbf{v}_h \in V_{0,h}. \quad (2.2.1.8)$$

Any solution \mathbf{u}_h can be written as $\mathbf{u}_h = \mathbf{u}_0 + \mathbf{w}_h$, with

$$\mathbf{w}_h \in V_{0,h}: \quad \mathbf{a}(\mathbf{w}_h, \mathbf{v}_h) = \ell(\mathbf{v}_h) - \mathbf{a}(\mathbf{u}_0, \mathbf{v}_h) \quad \forall \mathbf{v}_h \in V_{0,h}. \quad (2.2.1.9)$$

However, if \mathbf{u}_0 is not available in a “coding-suitable” format, one is forced to use an approximation. We return to this issue in Rem. 2.3.3.15 and § 2.7.6.1. \square

2.2.2 Galerkin Discretization: Second Step

A computer is clueless about a concept like “finite dimensional subspace”. What it can process are arrays of floating point numbers (vectors and matrices). Hence, all discretization methods must yield equations connecting vectors, aka, systems of equations. There is a well-known tool from linear algebra that accomplishes this reduction to matrix-vector algebra for the Galerkin method.

Definition 0.3.1.2. Basis of a finite dimensional vector space

Let V be a real vector space. A finite subset $\{b^1, \dots, b^N\} \subset V$, $N \in \mathbb{N}$, is a **basis** of V , if for **every** $\mathbf{v} \in V$ there are **unique** coefficients $\mu_\ell \in \mathbb{R}$, $\ell \in \{1, \dots, N\}$, such that $\mathbf{v} = \sum_{\ell=1}^N \mu_\ell b^\ell$. Then N agrees with the **dimension** of V .

Idea: ♦ choose (ordered) basis $\mathfrak{B}_h = \{b_h^1, \dots, b_h^N\}$, $N := \dim V_{0,h}$, of $V_{0,h}$:



$$V_{0,h} = \text{Span}\{\mathfrak{B}_h\}$$

♦ insert basis representation into the variational equation (2.2.1.1)

$$v_h \in V_{0,h} \Rightarrow v_h = v_1 b_h^1 + \dots + v_N b_h^N, \quad v_i \in \mathbb{R}, \quad (2.2.2.1)$$

$$u_h \in V_{0,h} \Rightarrow u_h = \mu_1 b_h^1 + \dots + \mu_N b_h^N, \quad \mu_i \in \mathbb{R}. \quad (2.2.2.2)$$

The number $N \in \mathbb{N}$ above is the **dimension** of the discrete trial and test space $V_{0,h}$.

Thus, the task of finding the Galerkin solution $u_h \in V_{0,h}$ is recast as the task of finding the $N < \infty$ basis expansion coefficients in (2.2.2.2). Of course, we have to employ the discrete linear variational problem (2.2.1.1). Key to the following manipulations is the **bi-linearity** of a ,

$$\begin{aligned} a(\alpha_1 v_1 + \beta_1 u_1, \alpha_2 v_2 + \beta_2 u_2) = \\ \alpha_1 \alpha_2 a(v_1, v_2) + \alpha_1 \beta_2 a(v_1, u_2) + \beta_1 \alpha_2 a(u_1, v_2) + \beta_1 \beta_2 a(u_1, u_2), \end{aligned}$$

for all $u_i, v_i \in V_0$, $\alpha_i, \beta_i \in \mathbb{R}$, and the **linearity** of ℓ

$$\ell(\alpha u + \beta v) = \alpha \ell(u) + \beta \ell(v),$$

for all $u, v \in V_0$, $\alpha, \beta \in \mathbb{R}$, see Def. 0.3.1.4. They give the following chain of **equivalent** statements:

$$u_h \in V_{0,h}: \quad a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h}. \quad (2.2.1.1)$$

$$\Updownarrow \quad [\begin{array}{l} u_h = \mu_1 b_h^1 + \dots + \mu_N b_h^N, \mu_i \in \mathbb{R} \\ v_h = v_1 b_h^1 + \dots + v_N b_h^N, v_i \in \mathbb{R} \end{array}]$$

$$\sum_{k=1}^N \sum_{j=1}^N \mu_k v_j a(b_h^k, b_h^j) = \sum_{j=1}^N v_j \ell(b_h^j) \quad \forall v_1, \dots, v_N \in \mathbb{R},$$

↑

$$\sum_{j=1}^N v_j \left(\sum_{k=1}^N \mu_k a(b_h^k, b_h^j) - \ell(b_h^j) \right) = 0 \quad \forall v_1, \dots, v_N \in \mathbb{R},$$

↑ (*)

$$\sum_{k=1}^N \mu_k a(b_h^k, b_h^j) = \ell(b_h^j) \quad \text{for } j = 1, \dots, N.$$

$$\Updownarrow \quad [\vec{\mu} = (\mu_1, \dots, \mu_N)^\top \in \mathbb{R}^N]$$

$$\mathbf{A} = \left[a(b_h^k, b_h^j) \right]_{j,k=1}^N \in \mathbb{R}^{N,N},$$

$$\vec{\phi} = \left[\ell(b_h^j) \right]_{j=1}^N.$$

$\mathbf{A} \vec{\mu} = \vec{\phi}$

with

A **linear system of equations** (LSE)

Note that the equivalence (*) is implied by the following lemma, applied to the linear form $v \mapsto a(u_h, v) - \ell(v)$.

Lemma 2.2.2.3. Testing with basis vectors

For every linear form $\ell : V \mapsto \mathbb{R}$ (\rightarrow Def. 0.3.1.4) on a vector space V holds

$$\ell(v) = 0 \quad \forall v \in V \Leftrightarrow \ell(b) = 0 \quad \forall b \in \mathfrak{B}$$

for any basis \mathfrak{B} (\rightarrow Def. 0.3.1.2) of V .

Summary: notions connected with Galerkin discretization

$\text{Linear discrete variational problem}$ $u_h \in V_{0,h}: \quad a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h}$	 Choosing basis \mathfrak{B}_h	$\text{Linear system of equations}$ $\mathbf{A}\vec{\mu} = \vec{\phi}$
$\text{Galerkin matrix: } \mathbf{A} = [a(b_h^k, b_h^j)]_{j,k=1}^N \in \mathbb{R}^{N,N},$		
$\text{Right hand side vector: } \vec{\phi} = [\ell(b_h^j)]_{j=1}^N \in \mathbb{R}^N,$		
$\text{Coefficient vector: } \vec{\mu} = [\mu_1, \dots, \mu_h]^\top \in \mathbb{R}^N,$		
$\text{Recovery of solution: } u_h = \sum_{k=1}^N \mu_k b_h^k.$		

Remark 2.2.2.5 (Alternative (“legacy”) terminology) In the context of finite element methods the building blocks of the linear systems of equations arising from Galerkin discretization have special names:

(Legacy) terminology for FEM:	Galerkin matrix	= stiffness matrix
	Right hand side vector	= load vector
	Galerkin matrix for $(u, v) \mapsto \int_\Omega uv \, dx$	= mass matrix

This hails from the times (lates 60s and early 70s), when finite element methods were mainly applied to solid mechanics (linear elasticity).

The term **degree of freedom** (d.o.f./DOF) is frequently used in connection with finite element methods. It has a double meaning denoting

- (i) either a single component of the basis expansion vector $\vec{\mu}$ for the Galerkin solution,
- (ii) or a basis function $b_h^k \in \mathfrak{B}_h$.

□

Of course, there are infinitely many ways to choose the basis \mathfrak{B}_h and in terms of the idea of Step I they are all equivalent:

Theorem 2.2.2.6. Independence of Galerkin solution of choice of basis

The choice of the basis \mathfrak{B}_h has no impact on the (set of) Galerkin solutions u_h of (2.2.1.1).

In other words, different bases \mathfrak{B}_h will, of course, yield different solutions for the vector $\vec{\mu}$ of basis expansion coefficients, but the linear combination (2.2.2.2) will always yield the same function $u_h \in V_{0,h}$.

2.2.3 Galerkin Matrices

As explained above the Galerkin matrix is the matrix \mathbf{A} of the linear system of equations arising from the Galerkin discretization of (2.2.0.2):

$$\mathbf{A} = \left[a(b_h^k, b_h^j) \right]_{j,k=1}^N \in \mathbb{R}^{N,N}, \quad \begin{array}{l} j \hat{=} \text{row index,} \\ k \hat{=} \text{column index,} \end{array}$$

where $\{b_h^1, \dots, b_h^N\}$ is the chosen (ordered) basis of the discrete trial/test space $V_{0,h} \subset V_0$, $N := \dim V_{0,h}$.

A consequence of the equivalence of the linear system of equations $\mathbf{A}\vec{\mu} = \vec{\phi}$ and the discrete variational problem (2.2.1.1) is immediate:

Corollary 2.2.3.1.

$$(2.2.1.1) \text{ has unique solution} \Leftrightarrow \mathbf{A} \text{ nonsingular (invertible)}$$

On one hand, we know from Thm. 2.2.2.6 that the Galerkin solution $u_h \in V_{0,h}$ does not depend on the choice of basis \mathfrak{B}_h . On the other hand, it is clear that for different bases \mathfrak{B}_h we get different Galerkin matrices. What do they have in common? The next lemma gives answers.

Lemma 2.2.3.2. Effect of change of basis on Galerkin matrix

Consider (2.2.1.1) and two bases of $V_{0,h}$,

$$\mathfrak{B}_h := \{b_h^1, \dots, b_h^N\}, \quad \tilde{\mathfrak{B}}_h := \{\tilde{b}_h^1, \dots, \tilde{b}_h^N\},$$

related by the basis transformation matrix \mathbf{S} according to

$$\tilde{b}_h^j = \sum_{k=1}^N s_{jk} b_h^k \quad \text{with} \quad \mathbf{S} = [s_{jk}]_{j,k=1}^N \in \mathbb{R}^{N,N} \text{ regular.} \quad (2.2.3.3)$$

Then the Galerkin matrices $\mathbf{A}, \tilde{\mathbf{A}} \in \mathbb{R}^{N,N}$, the right hand side vectors $\vec{\phi}, \tilde{\vec{\phi}} \in \mathbb{R}^N$, and the coefficient vectors $\vec{\mu}, \tilde{\vec{\mu}} \in \mathbb{R}^N$, respectively, satisfy

$$\tilde{\mathbf{A}} = \mathbf{S} \mathbf{A} \mathbf{S}^\top, \quad \tilde{\vec{\phi}} = \mathbf{S} \vec{\phi}, \quad \tilde{\vec{\mu}} = \mathbf{S}^{-\top} \vec{\mu}. \quad (2.2.3.4)$$

☞ notation: $\mathbf{S}^{-\top} := (\mathbf{S}^{-1})^\top = (\mathbf{S}^\top)^{-1}$ for any $\mathbf{S} \in \mathbb{R}^{N,N}$

Proof. Thanks to linearity and the definition of the entries of the right-hand-side vector $\tilde{\vec{\phi}}$, we get for $j \in \{1, \dots, N\}$

$$(\tilde{\vec{\phi}})_j = \ell(\tilde{b}_h^j) = \ell\left(\sum_{k=1}^N s_{jk} b_h^k\right) = \sum_{k=1}^N s_{jk} \ell(b_h^k) = \sum_{k=1}^N s_{jk} (\vec{\phi})_k = (\mathbf{S} \vec{\phi})_j,$$

which implies the assertion for the right-hand-side vector.

Concerning the Galerkin matrix, make use of the bilinearity of a (\rightarrow Def. 0.3.1.4), (2.2.3.3) and the definition of the entries of the Galerkin matrix: for $l, m \in \{1, \dots, N\}$,

$$(\tilde{\mathbf{A}})_{lm} = a(\tilde{b}_h^m, \tilde{b}_h^l) = \sum_{k=1}^N \sum_{j=1}^N s_{mk} a(b_h^k, b_h^j) s_{lj} = \sum_{k=1}^N \underbrace{\left(\sum_{j=1}^N s_{lj} (\mathbf{A})_{jk} \right)}_{(\mathbf{S} \mathbf{A})_{lk}} s_{mk} = (\mathbf{S} \mathbf{A} \mathbf{S}^\top)_{lm},$$

where we used the rules for the product of square matrices. □

The relationship $\tilde{\mathbf{A}} = \mathbf{SAS}^T$ between matrices has a special name in linear algebra:

Definition 2.2.3.5. Congruent matrices

Two matrices $\mathbf{A} \in \mathbb{R}^{N,N}$, $\mathbf{B} \in \mathbb{R}^{N,N}$, $N \in \mathbb{N}$, are called **congruent**, if there is a regular matrix $\mathbf{S} \in \mathbb{R}^{N,N}$ such that $\mathbf{B} = \mathbf{SAS}^T$.

Congruence of matrices is an equivalence relation and partitions the space of $N \times N$ -matrices into disjoint **congruence classes**. Any regular matrix describes a change of basis. Hence, any regular matrix can occur as basis transformation matrix \mathbf{S} in Lemma 2.2.3.2. Thus, given a discrete variational problem (2.2.1.1) all matrices of an entire congruence class can arise as Galerkin matrices.

Lemma 2.2.3.6. Congruent Galerkin matrices

$$\text{Matrix property invariant under congruence} \quad \Leftrightarrow \quad \text{Property of Galerkin matrix invariant under change of basis } \mathfrak{B}_h$$

§2.2.3.7 (Invariant properties of congruent matrices) The reader may wonder what properties of Galerkin matrices can be predicted without knowing the basis \mathfrak{B}_h . They are the following

- | | |
|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| matrix properties invariant under congruence : | <ul style="list-style-type: none"> • regularity → [Hip19, ??] • symmetry ($\mathbf{A}^T = \mathbf{A}$) • positive definiteness → [Hip19, ??] |
|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Proving the invariance of these properties is straightforward from the definition of congruence. We point out that the list above is almost exhaustive, because almost no other property one may associate with a matrix is stable under congruence, in particular

- not invariant are** ♦ any structure and sparsity of the Galerkin matrix, and
 ♦ its eigenvalues and condition number (→ Section 0.3.1.4).

Nevertheless, these latter properties have fundamental consequences for the numerical solution of the linear system of equations (required storage, computational effort, and impact of roundoff errors), as was already remarked above. □

Remark 2.2.3.8 (Properties of the Galerkin matrix inherited from a) The invariant properties of Galerkin matrices found in § 2.2.3.7 are not surprising, because they directly arise from corresponding properties of the underlying bilinear form a :

- a **symmetric** (→ Def. 0.3.1.15) ⇒ Galerkin matrix \mathbf{A} **symmetric**.
 - a **positive definite** (→ Def. 0.3.1.16) ⇒ Galerkin matrix \mathbf{A} **positive definite**.
-

Remark 2.2.3.9 (Summary: Impact of choice of basis) Let us summarize the dichotomy applying to the choice of basis \mathfrak{B}_h for a Galerkin method.

Thm. 2.2.2.6: the choice of \mathfrak{B}_h does **not** affect u_h ⇒ No impact on discretization error !
 (assuming exact arithmetic)

But: Key properties (e.g., conditioning, sparsity) of the Galerkin matrix crucially depend on \mathfrak{B}_h !

- The choice of trial/test space $V_{0,h}$ **alone** determines the quality of the Galerkin solution.
- The choice of basis \mathcal{B}_h determines how well (stably, efficiently) we can compute the Galerkin solution.

Review question(s) 2.2.3.10 (Principles of Galerkin Discretization)

(Q2.2.3.10.A) Described the **first step** of the Galerkin discretization of a linear variational problem. What does it yield?

(Q2.2.3.10.B) The Galerkin discretization of a linear variational problem leads to a linear system of equations. What determines the size of its coefficient matrix?

(Q2.2.3.10.C) Give simple *necessary and sufficient* conditions for existence and uniqueness of solutions of a linear system of equations arising from the Galerkin discretization of a linear variational problem.

(Q2.2.3.10.D) Write $\mathbf{A}\vec{\mu} = \vec{\phi}$ for the square linear system of equations produced by the Galerkin discretization of a linear variational problem posed on the vector space V_0 and using the basis $\mathcal{B} := \{b_h^1, \dots, b_h^N\}$ of the discrete trial/test space $V_{0,h}$, $N := \dim V_{0,h}$. Now we switch to the *rescaled basis* $\tilde{\mathcal{B}} := \{b_h^1, 2b_h^2, 3b_h^3, \dots, Nb_h^N\}$. What linear system will we get?

(Q2.2.3.10.E) Let $\mathbf{A}\vec{\mu} = \vec{\phi}$, $\mathbf{A} \in \mathbb{R}^{N,N}$, $\vec{\phi} \in \mathbb{R}^N$, $N \in \mathbb{N}$, be the linear system of equations obtained by the Galerkin discretization of a linear variational problem (LVP)

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0,$$

using a finite-dimensional trial/test space $V_{0,h} \subset V_0$ equipped with basis $\mathcal{B} := \{b_h^1, \dots, b_h^N\}$. Which linear system do we get when performing the Galerkin discretization of the same LVP but dropping every other basis function, that is, using the discrete trial/test space spanned by $\{b_h^1, b_h^3, \dots, b_h^{N-1}\}$ (for even N).

(Q2.2.3.10.F) Which properties of a Galerkin matrix do not depend on the choice of basis for the discrete trial and test space?

(Q2.2.3.10.G) As regards Galerkin discretization of a linear variational problem, which properties of the right-hand-side vector do not depend on the choice of basis for the discrete trial/test space?

(Q2.2.3.10.H) Let $\mathbf{A} \in \mathbb{R}^{N,N}$ be the Galerkin matrix obtained by the Galerkin discretization of a bilinear form $a(\cdot, \cdot)$. Given $\vec{\mu}, \vec{\eta} \in \mathbb{R}^N$, express the number $\vec{\mu}^\top \mathbf{A} \vec{\eta}$ through $a(\cdot, \cdot)$.

(Q2.2.3.10.I) On a vector space V_0 we consider a linear variational problem

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0,$$

with a *symmetric positive definite* bilinear form a and an $\|\cdot\|_a$ -bounded linear form $\ell: V_0 \rightarrow \mathbb{R}$. Show that for every finite-dimensional subspace $V_{0,h} \subset V_0$ there exists a basis $\mathcal{B} \subset V_{0,h}$ such that the associated Galerkin matrix for $a(\cdot, \cdot)$ is the identity matrix.

Hint. Use the following fundamental result from linear algebra:

Theorem 0.3.1.26. Real diagonalization of symmetric matrices

For *every* symmetric matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, that is, $\mathbf{A}^\top = \mathbf{A}$, we can find a *diagonal* matrix $\mathbf{D} \in \mathbb{R}^{n,n}$ and an *orthogonal* matrix $\mathbf{Q} \in \mathbb{R}^{n,n}$ such that

$$\mathbf{Q}^\top \mathbf{A} \mathbf{Q} = \mathbf{D}. \quad (2.2.3.11)$$

For a symmetric positive definite matrix \mathbf{A} the non-zero entries of the diagonal matrix \mathbf{D} are all positive.

(Q2.2.3.10.J) Give an example of a linear variational problem and of a discrete variational problem arising from its Galerkin discretization such that

- the linear variational problem has a unique solution,
- but the discrete variational problem fails to have a unique solution.

△

2.3 Case Study: Linear FEM for Two-Point Boundary Value Problems



Video tutorial for Section 2.3: Case Study: Linear FEM for Two-Point Boundary Value Problems: (49 minutes) [Download link](#), [tablet notes](#)

Now we make first acquaintance with the simplest conceivable finite element Galerkin discretization for the simplest conceivable class of second-order boundary value problems, the second-order two-point boundary value problem with homogeneous Dirichlet boundary conditions in weak form, cf. Section 1.5.1 for a slightly more general case. On an interval $]a, b[\subset \mathbb{R}$, $a < b$, we consider

$$u \in H_0^1(]a, b[): \underbrace{\int_a^b \frac{du}{dx}(x) \frac{dv}{dx}(x) dx}_{=:a(u, v)} = \underbrace{\int_a^b f(x)v(x) dx}_{=:l(v)} \quad \forall v \in H_0^1(]a, b[). \quad (2.3.0.1)$$

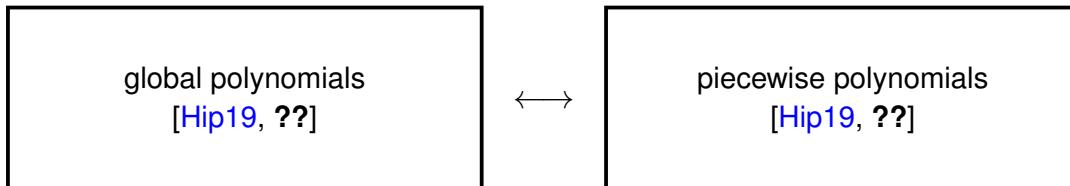
This is a linear variational problem fitting (2.2.0.2). In particular, in this case V_0 is the Sobolev space $H_0^1(]a, b[)$, see Def. 1.3.4.3. The strong form of (2.3.0.1) was given in Section 1.5.1:

$$-u'' = f \quad \text{in }]a, b[, \quad u(a) = u(b) = 0.$$

2.3.1 Step I: Choice of Discrete Trial/Test Space

Functions in the Galerkin trial/test space $V_{0,h}$ must be capable of approximating the solution function u . Thus, the discretization of boundary value problem is intimately connected with the problem of approximating functions, a topic covered in [Hip19, ??]. For the sake of function approximation the finite element method resorts to **polynomials**.

§2.3.1.1 (Approximation by (piecewise) polynomials) From the introduction to numerical methods we know two ways to harness polynomials for the approximation of functions $[a, b] \rightarrow \mathbb{R}$: approximation can rely on



The finite element method is based on approximation by piecewise polynomials, where “piecewise” is to be understood with respect to a partition of the computational domain Ω , $\Omega =]a, b[$ in this section.

Approximation in the FEM



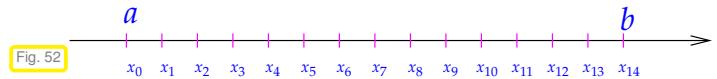
Idea underlying the finite element method:

approximate u by means of continuous, *piecewise polynomial* functions.

§2.3.1.3 (Meshes (grids) in one dimension) When talking about piecewise polynomials one has to fix a partitioning of the domain $[a, b]$ first.

Therefore we equip $\Omega = [a, b]$ with **$M + 1$ nodes**

($M \in \mathbb{N}$) forming the set



$$\mathcal{V}(\mathcal{M}) := \{a = x_0 < x_1 < \dots < x_{M-1} < x_M = b\}.$$

The nodes define small intervals that constitute a **mesh/grid**

$$\mathcal{M} := \{[x_{j-1}, x_j] : 1 \leq j \leq M\}.$$

The intervals $[x_{j-1}, x_j]$, $j = 1, \dots, M$ are the **cells** of the mesh \mathcal{M} , which is often identified with the set of its cells. A special case is an **equidistant** mesh with uniformly spaced nodes:

$$x_j := a + jh, \quad h := \frac{b - a}{M}.$$

The local and global resolution of a mesh/grid is measured through two quantities, the

$$\begin{array}{ll} \text{(local) cell size} & h_j := |x_j - x_{j-1}|, \quad j = 1, \dots, M \\ \text{(global) meshwidth} & h_{\mathcal{M}} := \max_j |x_j - x_{j-1}| \end{array}$$

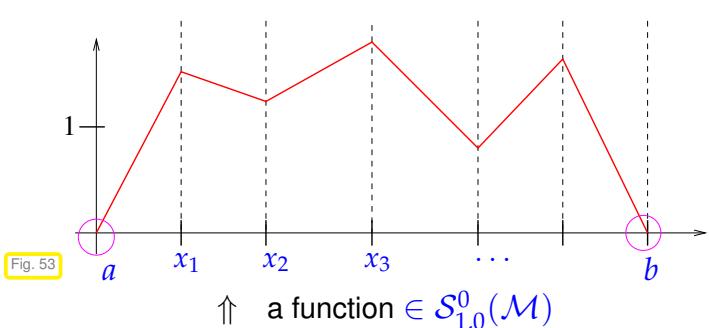
§2.3.1.4 (Piecewise linear finite element trial space) Now we take for granted that the interval $[a, b]$ is equipped with a mesh \mathcal{M} as introduced in § 2.3.1.3.

Recall from Thm. 1.3.4.23 that merely *continuous*, piecewise C^1 functions on $[a, b]$ belong to the Sobolev space $H^1([a, b])$:

Theorem 1.3.4.23. Compatibility conditions for piecewise smooth functions in $H^1(\Omega)$

Let Ω be partitioned into sub-domains Ω_1 and Ω_2 . A function u that is continuously differentiable in both sub-domains and continuous up to their boundary, belongs to $H^1(\Omega)$, if and only if u is continuous on Ω .

Thus, *continuous* piecewise polynomials provide valid trial and test functions for the variational problem (2.3.0.1). We give a first example.



The simplest space of continuous, \mathcal{M} -piecewise polynomial functions in $H_0^1([a, b])$:

$$\begin{aligned} V_{0,h} &= \mathcal{S}_{1,0}^0(\mathcal{M}) \\ &:= \left\{ v \in C^0([a, b]) : v|_{[x_{i-1}, x_i]} \text{ linear, } \begin{array}{l} i = 1, \dots, M, \\ v(a) = v(b) = 0 \end{array} \right\} \end{aligned}$$

$$N := \dim \mathcal{S}_{1,0}^0(\mathcal{M}) = M - 1$$

We infer the dimension of $\mathcal{S}_{1,0}^0(\mathcal{M})$ from the obvious fact that every function $\in \mathcal{S}_{1,0}^0(\mathcal{M})$ is uniquely determined by prescribing its values at the interior nodes x_1, x_2, \dots, x_{M-1} of the mesh \mathcal{M} .

☞ notation: The symbol \mathcal{S} alludes to the fact that $\mathcal{S}_{1,0}^0(\mathcal{M})$ is a space of **splines**, actually a subspace of the degree-1 spline space $\mathcal{S}_{1,\mathcal{M}}$ from [Hip19, ??]. It also indicates that we deal with a space of **scalar-valued functions**.

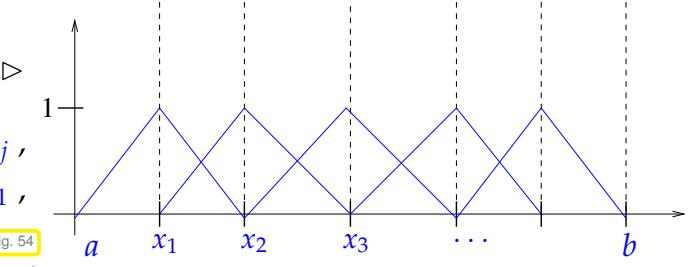
2.3.2 Step II: Choice of Basis

The space $\mathcal{S}_{1,0}^0(\mathcal{M})$ will serve as both trial and test space for the Galerkin discretization (→ Section 2.2.1) of (2.3.0.1). As we learned in Section 2.2.2, for a complete description of both steps we also need to specify an (ordered) basis of $\mathcal{S}_{1,0}^0(\mathcal{M})$.

Our choice of the (ordered) basis $\mathfrak{B}_h\{b_h^1, \dots, b_h^{M-1}\}$ of \mathcal{V}_h is the following:

1D “tent functions” [Hip19, ??]

$$b_h^j(x) := \begin{cases} (x - x_{j-1})/h_j & , \text{ if } x_{j-1} \leq x \leq x_j , \\ (x_{j+1} - x)/h_{j+1} & , \text{ if } x_j \leq x \leq x_{j+1} , \\ 0 & \text{elsewhere.} \end{cases} \quad \text{Fig. 54} \quad (2.3.2.1)$$



$$\mathfrak{B}_h = \{b_h^j\}_{j=1}^{M-1} \quad b_h^j(x_i) = \delta_{ij} := \begin{cases} 1 & , \text{ if } i = j , \\ 0 & , \text{ if } i \neq j . \end{cases} \quad (2.3.2.2)$$

We point out that (2.3.2.2) qualifies the tent functions as a **cardinal basis** of $\mathcal{S}_{1,0}^0(\mathcal{M})$ with respect to the node set $\mathcal{X} := \{x_i\}$. As a consequence the **basis expansion coefficients** for any function in $\mathcal{S}_{1,0}^0(\mathcal{M})$ are given by the function values at the interior nodes of the mesh:

$$u_h \in \mathcal{S}_{1,0}^0(\mathcal{M}) \iff u_h = \sum_{i=1}^{M-1} u_h(x_i) b_h^i . \quad (2.3.2.3)$$

It is clear that \mathfrak{B}_h is a basis of $\mathcal{S}_{1,0}^0(\mathcal{M})$. Moreover a key property of the tent basis functions is that their support just comprises two adjacent cells of the mesh,

$$\text{supp}(b_h^j) = [x_{j-1}, x_{j+1}] , \quad j \in \{1, \dots, M-1\} , \quad (2.3.2.4)$$

where we have relied on the following notion of “support”:

Definition 2.3.2.5. Support of a function

The **support** of a function $f : \Omega \mapsto \mathbb{R}$ is defined as

$$\text{supp}(f) := \overline{\{x \in \Omega : f(x) \neq 0\}} .$$

To differentiate b_h^j despite the presence of a kink, we can apply the rule that the derivative of a function $\in C_{\text{pw}}^1([a, b])$ as a function in $L^2([a, b])$ can be computed *piecewise*, cf. Ex. 1.3.4.24. So from (2.3.2.1) we immediately get

$$(2.3.2.1) \quad \frac{db_h^j}{dx}(x) = \begin{cases} \frac{1}{h_j} & , \text{ if } x_{j-1} \leq x \leq x_j , \\ -\frac{1}{h_{j+1}} & , \text{ if } x_j < x \leq x_{j+1} , \\ 0 & \text{elsewhere.} \end{cases} \quad (2.3.2.6)$$

Obviously, this derivative is \mathcal{M} -piecewise constant and discontinuous. \square

Remark 2.3.2.7 (Benefit of variational formulations of BVPs) The possibility of using simple piecewise linear trial and test functions is a clear benefit of the weak/variational formulations of elliptic two-point boundary value problems, since these still make sense for merely piecewise continuously differentiable functions.

Below, in Section 4.4 we will learn about a method that targets the strong form of the 2-point BVP and, thus, has to impose more regularity on the trial functions. \square

2.3.3 Formulas for Galerkin Matrices and Right-Hand-Side Vectors

We follow the policy of Galerkin discretization elaborated in Section 2.2. We plug in trial functions from the finite element space $\mathcal{S}_{1,0}^0(\mathcal{M})$ into (2.3.0.1), expand them as a linear combination of tent functions, and test with all tent functions as explained in Section 2.2.2. Using $u_h = \mu_1 b_h^1 + \dots + \mu_N b_h^N$ with coefficients μ_j , $j = 1, \dots, N$, and $N = \dim \mathcal{S}_{1,0}^0(\mathcal{M})$, by Lemma 2.2.2.3 we arrive at the “algebraic” variational problem: seek $\mu_l \in \mathbb{R}$, $l = 1, \dots, N$,

$$\underbrace{\int_a^b \sum_{l=1}^N \mu_l \frac{db_h^l}{dx}(x) \frac{db_h^k}{dx}(x) dx}_{=: a(u_h, b_h^k)} = \underbrace{\int_a^b f(x) b_h^k(x) dx}_{=: \ell(b_h^k)}, \quad k = 1, \dots, N.$$

\Updownarrow

$$\sum_{l=1}^N \left(\int_a^b \frac{db_h^l}{dx}(x) \frac{db_h^k}{dx}(x) dx \right) \mu_l = \underbrace{\int_a^b f(x) b_h^k(x) dx}_{=: \varphi_k}, \quad k = 1, \dots, N.$$

\Updownarrow

$$\boxed{\mathbf{A}\vec{\mu} = \vec{\varphi}} \quad \text{with} \quad (\mathbf{A})_{kl} := \int_a^b \frac{db_h^l}{dx}(x) \frac{db_h^k}{dx}(x) dx, \quad k, l = 1, \dots, N,$$

$$\vec{\mu} = [\mu_l]_{l=1}^N \in \mathbb{R}^N, \quad \vec{\varphi} = [\varphi_k]_{k=1}^N \in \mathbb{R}^N.$$

A linear system of equations!

- ▷ Galerkin matrix $\mathbf{A} \in \mathbb{R}^{N,N}$, $(\mathbf{A})_{ij} := \int_a^b \frac{db_h^i}{dx}(x) \frac{db_h^j}{dx}(x) dx$, $1 \leq i, j \leq N$
- piecewise derivatives
- ▷ r.h.s. vector $\vec{\varphi} \in \mathbb{R}^N$, $(\vec{\varphi})_k := \int_a^b f(x) b_h^k(x) dx$, $k = 1, \dots, N$.

§2.3.3.1 (Computation of entries of Galerkin matrix) We rely on the tent functions b_h^j as basis functions on a mesh \mathcal{M} as described in § 2.3.1.3.

$$b_h^j(x) := \begin{cases} (x - x_{j-1})/h_j & , \text{ if } x_{j-1} \leq x \leq x_j , \\ (x_{j+1} - x)/h_{j+1} & , \text{ if } x_j \leq x \leq x_{j+1} , \\ 0 & \text{elsewhere.} \end{cases} \quad (2.3.2.1)$$

The detailed computations start with the evident fact that

$$|i - j| \geq 2 \quad \Rightarrow \quad \frac{db_h^j}{dx}(x) \cdot \frac{db_h^i}{dx}(x) = 0 \quad \forall x \in [a, b] ,$$

because there is *no overlap* of the *supports* (\rightarrow Def. 2.3.2.5) of the two basis functions.

In addition, we use that the gradients of the tent functions are piecewise constant:

$$\frac{db_h^j}{dx}(x) = \begin{cases} \frac{1}{h_j} & , \text{ if } x_{j-1} \leq x \leq x_j , \\ -\frac{1}{h_{j+1}} & , \text{ if } x_j < x \leq x_{j+1} , \\ 0 & \text{elsewhere,} \end{cases} \quad (2.3.2.6)$$

and immediately get

$$\int_a^b \frac{db_h^j}{dx}(x) \frac{db_h^i}{dx}(x) dx = \begin{cases} 0 & , \text{ if } |i - j| \geq 2 \\ -\frac{1}{h_{i+1}} & , \text{ if } j = i + 1 \\ -\frac{1}{h_i} & , \text{ if } j = i - 1 \\ \frac{1}{h_i} + \frac{1}{h_{i+1}} & , \text{ if } 1 \leq i = j \leq M - 1 \end{cases} \rightarrow$$

We obtain the following Galerkin matrix, which is symmetric, **positive definite** (\rightarrow Def. 0.3.1.16), and tridiagonal:

$$\mathbf{A} = \begin{bmatrix} \frac{1}{h_1} + \frac{1}{h_2} & -\frac{1}{h_2} & 0 & & & 0 \\ -\frac{1}{h_2} & \frac{1}{h_2} + \frac{1}{h_3} & -\frac{1}{h_3} & & & \\ 0 & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & 0 & \\ & & \ddots & \ddots & -\frac{1}{h_{M-1}} & \\ 0 & & 0 & -\frac{1}{h_{M-1}} & \frac{1}{h_{M-1}} + \frac{1}{h_M} & \end{bmatrix} \in \mathbb{R}^{N,N}, \quad N := M - 1 . \quad (2.3.3.2)$$

☞ notation: $h_j := |x_j - x_{j-1}|$ $\hat{=}$ local meshwidth, cell size

Remark 2.3.3.3 (Special case: Linear system of equations for linear finite element discretization on equidistant mesh) On an equidistant mesh with uniform meshwidth $h > 0$ we learn from (2.3.3.2) that

the finite element linear system of equations becomes

$$\frac{1}{h} \begin{bmatrix} 2 & -1 & 0 & & & 0 \\ -1 & 2 & -1 & & & \\ 0 & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & 0 \\ & & & -1 & 2 & -1 \\ 0 & & & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_N \end{bmatrix} = h \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix}. \quad (2.3.3.4)$$

This is a symmetric positive definite **Töplitz matrix**, that is, a matrix with constant entries on diagonals, see [Hip19, ??]. Its eigenvectors, indexed by $\ell = 1, \dots, N$, have the components

$$\eta_j^\ell = \sin\left(2\pi \frac{\ell j}{N+1}\right), \quad j = 1, \dots, N. \quad (2.3.3.5)$$

□

§2.3.3.6 (Computation of right hand side vector for linear finite element Galerkin discretization) The right hand side linear form of the linear variational equation (2.3.0.1) involves a general source function $f = f(x)$, which may be available only in procedural form, recall Rem. 2.1.2.5.



If only point evaluations of f are possible, then the only option is the computation of right hand side vector $\vec{\phi} \in \mathbb{R}^N$ by **numerical quadrature** [Hip19, ??]!

The standard approach is to approximate an integral by a weighted sum of function values, a so-called quadrature formula:

Replace the integral with an m -point **quadrature formula/quadrature rule** on $[a, b]$, $m \in \mathbb{N} \rightarrow$ [Hip19, ??]:

$$\int_a^b \psi(t) dt \approx Q_m(\psi) := \sum_{j=1}^m \omega_j^m \psi(\zeta_j^m). \quad (2.3.3.7)$$

ω_j^m : quadrature weights , ζ_j^m : quadrature nodes $\in [a, b]$.

Special considerations apply in the current setting: The natural choice of numerical quadrature for M -piecewise linear trial/test spaces is the use of an M -based **composite quadrature rule**, which applies a simple quadrature formula on all cells of the mesh, e.g., the **composite trapezoidal rule**

$$\int_a^b \psi(t) dt \approx \sum_{l=1}^M \frac{1}{2} h_l (\psi(x_{l-1}) + \psi(x_l)). \quad (2.3.3.8)$$

Now the cardinal basis property (2.3.2.2) of the tent functions comes handy and leads to

► $\varphi_k := (\vec{\phi})_k = \int_a^b f(x) b_h^k(x) dx \approx \frac{1}{2}(h_k + h_{k+1})f(x_k), \quad 1 \leq k \leq N. \quad (2.3.3.9)$

□

§2.3.3.10 (Linear finite element Galerkin discretization for general stiffness coefficient) We generalize the considerations of § 2.3.3.1 and perform a piecewise linear finite element Galerkin discretization of the linear variational problem arising from the pinned elastic string model, cf. (1.4.2.1),

$$u \in H_0^1([a, b]): \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1(\Omega)[a, b].$$

We repeat that the coefficient function $\sigma : [a, b] \rightarrow \mathbb{R}$ is uniformly positive

$$\exists \sigma_0 > 0: \sigma(x) \geq \sigma_0 \quad \forall x \in [a, b]. \quad (1.2.1.20)$$

and is given in procedural form only as discussed in Rem. 2.1.2.5. Our design of the finite element method relies on the following additional properties of σ :

Assumption 2.3.3.11. Smoothness requirement for stiffness coefficient

σ is piecewise continuous, $\sigma \in C_{\text{pw}}^0([a, b])$, with jumps *only* at grid nodes x_j

As above we plug in a basis expansion of the trial function $u_h \in \mathcal{S}_{1,0}^0(\mathcal{M})$, $u_h = \mu_1 b_h^1 + \dots + \mu_N b_h^N$, and test with all tent functions b_h^k :

$$\int_a^b \sigma(x) \sum_{l=1}^N \mu_l \frac{db_h^l}{dx}(x) \frac{db_h^k}{dx}(x) dx = \int_a^b f(x) b_h^k(x) dx, \quad k = 1, \dots, N. \quad (2.3.3.12)$$

In light of Rem. 2.1.2.5 numerical quadrature will be required for the (approximate) evaluation of both integrals. We use different quadrature rules,

- ◆ the **composite midpoint rule** for left-hand-side integral → [Hip19, ??]

$$\int_a^b \psi(x) dx \approx \sum_{j=1}^M h_j \psi(m_j), \quad m_j := \frac{1}{2}(x_j + x_{j-1}). \quad (2.3.3.13)$$

and, as before, the **composite trapezoidal rule**

$$\int_a^b \psi(t) dt \approx \sum_{l=1}^M \frac{1}{2} h_l (\psi(x_{l-1}) + \psi(x_l)), \quad (2.3.3.8)$$

see also [Hip19, ??], for the right-hand side integral.

This numerical quadrature applied to (2.3.3.12) results in the following linear system of equations for the expansion coefficients μ_ℓ :

$$\sum_{\ell=1}^N \underbrace{\left(\sum_{j=1}^M h_j \sigma(m_j) \frac{db_h^\ell}{dx}(m_j) \frac{db_h^k}{dx}(m_j) \right)}_{=(\mathbf{A})_{k,\ell}} \mu_\ell = \underbrace{\frac{1}{2} (h_{k+1} + h_k) f(x_k)}_{=: (\vec{\varphi})_k}, \quad k = 1, \dots, N,$$

↑
 $\mathbf{A} \vec{\mu} = \vec{\varphi}.$

Concretely, on an equidistant mesh with uniform cell size $h > 0$ we end up with the linear system of equations

$$\frac{1}{h} \begin{bmatrix} \sigma_1 + \sigma_2 & -\sigma_2 & 0 & \dots & & \dots & 0 \\ -\sigma_2 & \sigma_2 + \sigma_3 & -\sigma_3 & & & & \mu_1 \\ 0 & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ \vdots & & & & -\sigma_{M-2} & \sigma_{M-2} + \sigma_{M-1} & -\sigma_{M-1} \\ 0 & \dots & & \dots & 0 & -\sigma_{M-1} & \sigma_{M-1} + \sigma_M \end{bmatrix} \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_N \end{bmatrix} = h \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix}, \quad (2.3.3.14)$$

with $\sigma_j := \sigma(m_j)$, $j = 1, \dots, m$. \square

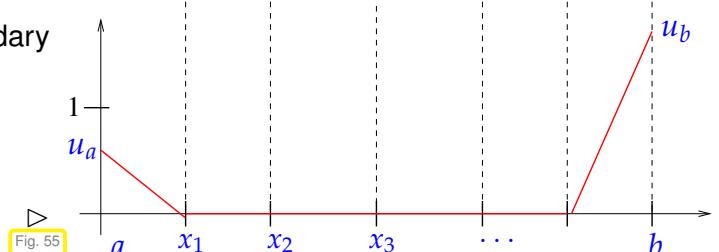
Remark 2.3.3.15 (Offset function for finite element Galerkin discretization) Recall the offset function trick as discussed in § 1.2.3.12, where it enabled us to return to a vector space as trial space, if the original linear variational problem was posed on an affine space. We face this situation for the pinned elastic string variational problem (1.4.2.1) unless the boundary values u_a and u_b are both zero, which we have assumed so far.

Now we return to the general situation of non-homogeneous Dirichlet boundary conditions and wonder what are computationally convenient offset functions in the context of linear finite element Galerkin discretization?

To deal with the case of general Dirichlet boundary conditions

$$u(a) = u_a, \quad u(b) = u_b$$

we use a *piecewise linear* offset function



$$u_{0,h}(x) = \begin{cases} u_a(1 - \frac{x-a}{h_1}) & , \text{ if } a \leq x \leq x_1 , \\ u_b(1 - \frac{b-x}{h_M}) & , \text{ if } x_{M-1} \leq x \leq b , \\ 0 & \text{elsewhere,} \end{cases} \quad u_{0,h} \in H^1([a,b]), \quad u_{0,h}(a) = u_a, \quad u_{0,h}(b) = u_b. \quad (2.3.3.16)$$

Of course, we could have used a linear offset function as in Ex. 1.2.3.17

$$u_0(x) = \frac{b-x}{b-a}u_a + \frac{x-a}{b-a}u_b, \quad x \in [a,b],$$

but the above choice (2.3.3.16) has considerable benefits:

- ◆ $u_{0,h}$ is a *simple* function (since it is M -piecewise linear),
- $u_{0,h}$ is *locally supported*: contributions from $u_{0,h}$ will alter only first and last component of right hand side vector. To understand why, recall the linear variational problem arising from the use of an offset function

$$w \in V_0: \quad a(w, v) = \ell(v) - a(u_{0,h}, v) \quad \forall v \in V_0. \quad (2.2.0.8)$$

and verify that, $a(u_{0,h}, b_h^j) \neq 0$ only for $j = 1, M-1$. Thus, only two entries of the right-hand-side vector will change.

Review question(s) 2.3.3.17.

Linear FEM for two-point boundary value problems

(Q2.3.3.17.A) [Non-differentiable finite-element trial and test functions] The piecewise linear functions used for the finite element Galerkin discretization of a second-order elliptic boundary value problem on $]a, b[$ are not differentiable on $]a, b[$ in general. Explain why they can be used nevertheless.

(Q2.3.3.17.B) [Graded mesh] We consider a **graded mesh** \mathcal{M}_α of the interval $\Omega := [0, 1]$ with node set

$$\mathcal{V}(\mathcal{M}) := \{x_j := (j/n)^\alpha, j = 0, \dots, n\}, \quad n \in \mathbb{N}, \quad \alpha > 0. \quad (2.3.3.18)$$

What is the smallest and largest cell size of \mathcal{M}_α ?

(Q2.3.3.17.C) [Tent functions on graded mesh] Let $\{b_h^1, \dots, b_h^{n-1}\}$ stand for the **tent function basis** of $\mathcal{S}_{1,0}^0(\mathcal{M}_2)$ on the graded mesh \mathcal{M}_2 of $\Omega :=]0, 1[$, whose nodes are given by (2.3.3.18) for $\alpha = 2$. These basis functions satisfy $b_h^j(x_i) = \delta_{ij}$, $i, j = 1, \dots, n - 1$. Compute the norms $\|b_h^i\|_{L^2(\Omega)}$ and $|b_h^i|_{H^1(\Omega)}$.

(Q2.3.3.17.D) [Galerkin discretization of a right-hand side functional on a graded mesh] On the graded mesh \mathcal{M}_2 as given by (2.3.3.18) for $\alpha = 2$ we consider the finite element space $\mathcal{S}_{1,0}^0(\mathcal{M})$ equipped with the tent function basis $\{b_h^1, \dots, b_h^{n-1}\}$, $b_h^j(x_i) = \delta_{ij}$, $i, j = 1, \dots, n - 1$. Compute the vector $\vec{\phi} \in \mathbb{R}^{n-1}$ obtained by the Galerkin discretization of the right-hand side functional $\ell(v) := \int_0^1 v(x) dx$.

(Q2.3.3.17.E) [Non-zero entries of Galerkin matrix] We conduct the Galerkin finite-element discretization of the generic second-order elliptic two-point Dirichlet boundary value problem

$$u \in H_0^1(]0, 1[): \quad \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1(]0, 1[),$$

with uniformly positive coefficient σ , using the trial and test space $\mathcal{S}_{1,0}^0(\mathcal{M}_\alpha)$ on the graded mesh according to (2.3.3.18) and with the standard tent function basis. Give sharp upper and lower bounds for the number of non-zero entries of the Galerkin matrix.

(Q2.3.3.17.F) [Numerical quadrature] What is **numerical quadrature** and how is it applied in the context of the Galerkin discretization of the right-hand side linear functional

$$v \mapsto \int_0^1 f(x)v(x) dx, \quad f \in C^0([0, 1]).$$

(Q2.3.3.17.G) [Composite trapezoidal rule] The use of the **composite trapezoidal quadrature rule** on a mesh \mathcal{M} of $]a, b[$ with nodes $a = x_0 < x_1 < \dots < x_{M-1} < x_M := b$ amounts to the approximation

$$\int_a^b \phi(x) dx \approx \frac{1}{2}h_1\phi(x_0) + \sum_{j=1}^{M-1} \phi(x_j)\frac{1}{2}(h_j + h_{j+1}) + \frac{1}{2}h_M\phi(x_M).$$

Explain the difficulty encountered when applying this quadrature rule for the computation of entries of the Galerkin matrix for the linear variational problem

$$u \in H_0^1(]a, b[): \quad \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1(]a, b[),$$

discretized based on $\mathcal{S}_{1,0}^0(\mathcal{M})$ and its tent function basis.

(Q2.3.3.17.H) [Finite-element Galerkin discretization non-symmetric bilinear form] We consider the bilinear form

$$b(u, v) := \int_0^1 u(x) \frac{dv}{dx}(x) dx, \quad u \in L^2([0, 1]), \quad v \in H^1([0, 1]).$$

We study its Galerkin discretization based on the trial/test space $\mathcal{S}_{1,0}^0(\mathcal{M})$ on an equidistant mesh of $[0, 1]$ with M cells, using the standard tent function basis. Compute the resulting Galerkin matrix.

△

2.4 Case Study: Triangular Linear FEM in Two Dimensions

1.  Video tutorial for Section 2.4: Case Study: Triangular Linear FEM in Two Dimensions (I): (53 minutes) [Download link](#), [tablet notes](#)
2.  Video tutorial for Section 2.4: Case Study: Triangular Linear FEM in Two Dimensions (II): (61 minutes) [Download link](#), [tablet notes](#)

This section elaborates how to extend the linear finite element Galerkin discretization of Section 2.3 to two dimensions. Familiarity with the 1D setting is essential for understanding the current section.

Parts of the presentation are based on a simple C++ finite element code that is available on the course Git repository → [GITHUB](#)

§2.4.0.1 (Polygonal computational domain) In this section we consider boundary value problems posed on a two-dimensional bounded domain $\Omega \subset \mathbb{R}^2$.

For general assumptions on the domain $\Omega \subset \mathbb{R}^2$ refer to § 1.2.1.14:

- Ω is bounded and
- Ω has a piecewise smooth boundary.

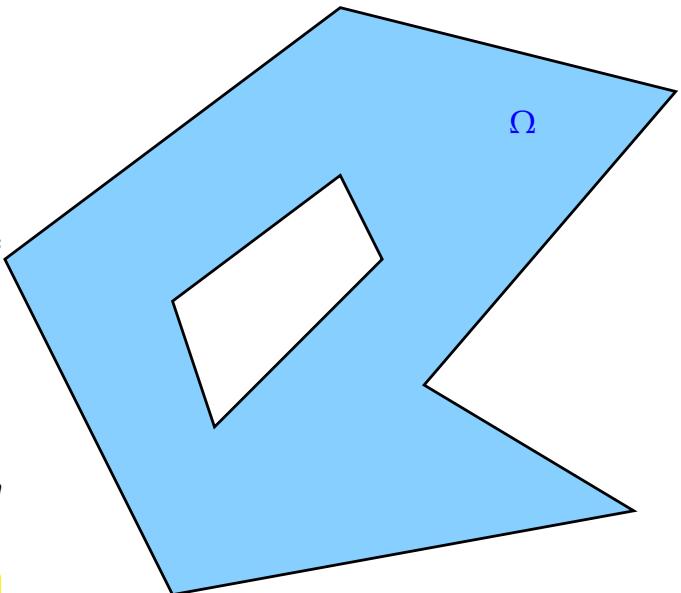
For the sake of simplicity we further restrict the set of admissible domains.

Additional assumption: Ω is a polygon

polygon with 10 corners ▷

(By default, the domain Ω is assumed to be an *open* set, that is, $x \in \Omega$ implies $x \notin \partial\Omega$!)

Fig. 56



↓

§2.4.0.2 (Model problem) We initially focus on the following well-posed 2nd-order linear variational problem posed on the Sobolev space $H^1(\Omega)$ (→ Def. 1.3.4.8). It represents a Neumann problem (→ Ex. 1.8.0.10) with homogeneous Neumann data and reaction term

$$u \in H^1(\Omega): \quad \int_{\Omega} \alpha(x) \mathbf{grad} u \cdot \mathbf{grad} v + c(x) u v dx = \int_{\Omega} f v dx \quad \forall v \in H^1(\Omega), \quad (2.1.2.2)$$

↔ see Section 1.5

strong form, BVP:

$$\begin{aligned} -\operatorname{div}(\alpha(x) \operatorname{grad} u) + c(x)u &= f \quad \text{in } \Omega, \\ \operatorname{grad} u \cdot n &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

We take for granted that the reaction coefficient $c = c(x)$ is supposed to be uniformly positive definite according to (2.1.2.3), a concept explained in (1.2.2.8) and Def. 1.2.2.9. Thus we avoid potential non-uniqueness of solutions, see Rem. 1.8.0.14 for a discussion. \square

2.4.1 Meshes in 2D: Triangulations

Question: What is the 2D counterpart of the 1D mesh/grid \mathcal{M} as introduced in § 2.3.1.3? There the mesh was defined through a **partition** of the computational domain $\Omega =]a, b[$. While in 1D splitting the interval into disjoint sub-intervals is about the only meaningful option to define a partition, we have many more possibilities in higher dimensions. We opt for a very special way to cut $\Omega \subset \mathbb{R}^2$ into pieces, which relies on **triangulations**.

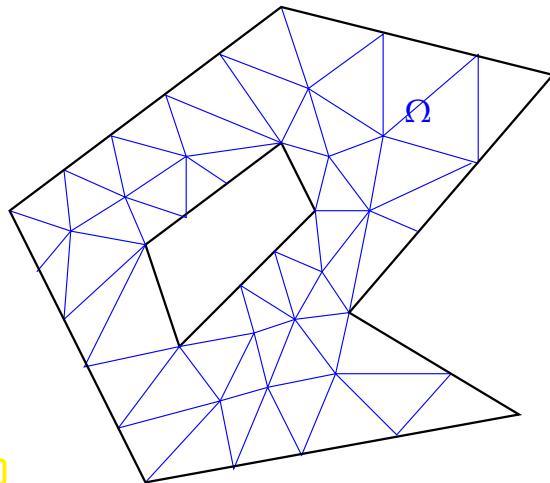


Fig. 57

A **triangulation** \mathcal{M} of Ω satisfies

- (i) $\mathcal{M} = \{K_i\}_{i=1}^M$, $M \in \mathbb{N}$, $K_i \triangleq$ open triangle
- (ii) disjoint interiors: $i \neq j \Rightarrow K_i \cap K_j = \emptyset$
- (iii) tiling/partition property: $\bigcup_{i=1}^M \bar{K}_i = \bar{\Omega}$
- (iv) intersection $\bar{K}_i \cap \bar{K}_j$, $i \neq j$,
is
 - either \emptyset
 - or an edge of both triangles
 - or a vertex of both triangles

notation: overline tag $\bar{-}$ \triangleq a subset of \mathbb{R}^d together with its boundary ("closure")

Common parlance:
vertices of triangles = **nodes** of mesh (= set $\mathcal{V}(\mathcal{M})$)
triangles of the mesh = **cells** or **elements** of mesh (= set \mathcal{M})

For both sets we assume that their elements are numbered: $\mathcal{M} = \{K_1, \dots, K_M\}$, $\mathcal{V}(\mathcal{M}) = \{x^1, \dots, x^N\}$, $N \in \mathbb{N}$.



In most parts of this document we stick to the mathematical indexing convention counting from 1, as opposed to the C++ indexing convention starting with 0. Confusing both is a major source of errors.

Most decompositions of Ω into non-overlapping triangles will not qualify as triangulations.

A mesh that does not comply with the property (iv) from above.

Parlance: $\bullet \triangleq$ "hanging node"

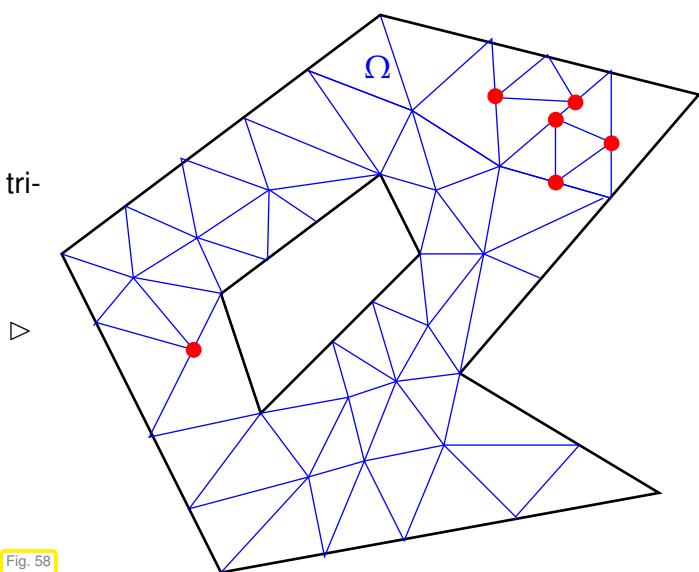


Fig. 58

§2.4.1.1 (Data structure describing a (triangular) triangulation in 2D) Thanks to the constraints imposed on the triangles of a triangulation with $M \in \mathbb{N}$ triangles and N vertices, its full description requires only two matrices, see Code 2.4.1.2:

- (I) $_nodecoords \hat{=} N \times 2$ matrix $\in \mathbb{R}^{N,2}$, i -th row containing the coordinates of the i -th vertex, $i \in \{1, \dots, N\}$
- (II) $_elements \hat{=} M \times 3$ -matrix $\in \mathbb{N}^{M,3}$, j -th row containing the index numbers of the vertices of the j -th triangle, $j \in \{1, \dots, M\}$.

Note: A *local ordering*/numbering of the vertices of every triangle of the triangulation is implicitly provided by this data structure.

(Here we follow the mathematical convention using indices $\in \mathbb{N}$.)

The following C++ class **TriaMesh2D** stores this minimal information of a planar triangular mesh. This is a rudimentary implementation; a proper object oriented design would call for many more access and manipulation methods.

C++ code 2.4.1.2: Class handling planar triangular mesh → GITHUB

```

2 // Matrix containing vertex coordinates of a triangle
3 using TriGeo_t = Eigen::Matrix<double, 2, 3>;
4 struct TriaMesh2D {
5     // Constructor: reads mesh data from file, whose name is passed
6     TriaMesh2D(std::string filename); //
7     virtual ~TriaMesh2D(void) {}
8     // Retrieve coordinates of vertices of a triangles as rows
9     // of a fixed-size 3x2 matrix
10    TriGeo_t getVtCoords(std::size_t) const;
11    // Data members describing geometry and topology
12    Eigen::Matrix<double, Eigen::Dynamic, 2> _nodecoords;
13    Eigen::Matrix<int, Eigen::Dynamic, 3> _elements;
14 };

```

The constructor of **TriaMesh2D**, Line 6, Code 2.4.1.2, reads the mesh from a file with this format displayed on the right ▷

1st line: positive integer N followed by keyword `vertices`, $N \hat{=}$ number of vertices.

line $2 \leftrightarrow N+1$: pairs x y of reals $\hat{=}$ coordinates of vertices

line $N+2$: positive integer M and keyword `triangles`, $M \hat{=}$ number of triangles.

line $N+3 \leftrightarrow N+2+M$: triplets v_1 v_2 v_3 of positive integers $\in \{1, \dots, N\}$, indices of vertices of triangles.

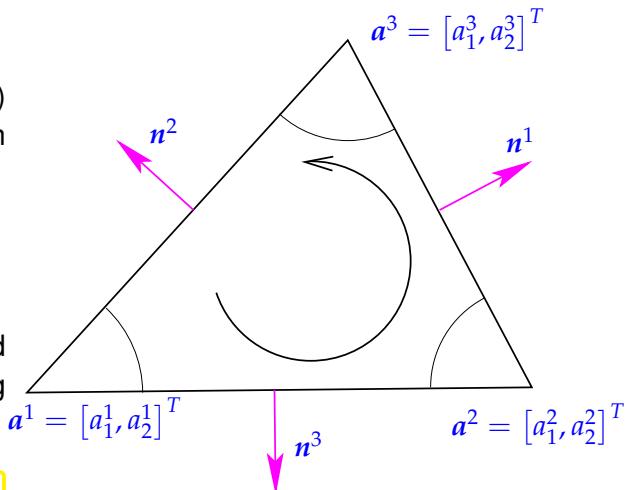
1	N vertices
2	x y
3
4	x y
5	M triangles
6	v_1 v_2 v_3
7
8	v_1 v_2 v_3

Details of the mesh data structure can be inferred from how a mesh is built from the information stored in a file, see the constructor of the class **TriaMesh2D** reading mesh from file → GITLAB.

The member function `getVtCoords` (\rightarrow Code 2.4.1.3) stores the coordinates of the three vertices of a triangle in the rows of a 2×3 -matrix:

$$\begin{bmatrix} a_1^1 & a_1^2 & a_1^3 \\ a_2^1 & a_2^2 & a_2^3 \end{bmatrix}.$$

This format is used in the C++ code below and the fixed size matrix data type `TriGeo_t` is introduced for storing triplets of triangle vertex coordinates.



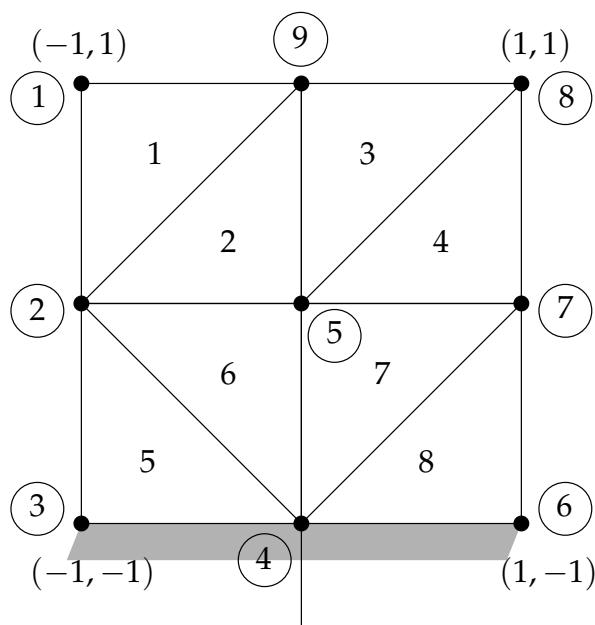
C++ code 2.4.1.3: Retrieve coordinates of vertices of a triangles as rows of a 3×2 -matrix

[→ GITHUB](#)

```

1 TriGeo_t TriMesh2D::getVtCoords(size_t i) const {
2 // Check whether valid cell index (starting from zero!)
3 assert(i < _elements.rows());
4 // Obtain numbers of vertices of triangle i
5 const Eigen::RowVector3i idx = _elements.row(i);
6 // Build matrix of vertex coordinates
7 Eigen::Matrix<double, 3, 2> vtc;
8 vtc << _nodecoords.row(idx[0]),
9 _nodecoords.row(idx[1]),
10 _nodecoords.row(idx[2]);
11 return vtc.transpose();
12 }
```

EXAMPLE 2.4.1.4 (Internal array representation of 2D triangular mesh) We make explicit the contents of the `_nodecoords` and `_elements` matrices, data members of `TriMesh2D`, for the special triangulation of the square $\Omega = [-1, 1]^2$ drawn in Fig. 60.

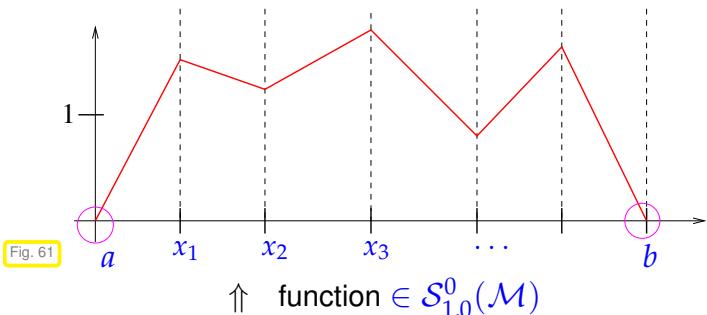


i	$(x_i)_1$	$(x_i)_2$	K_j	Vertex indices
1	-1	1	1	1 2 9
2	-1	0	2	2 5 9
3	-1	-1	3	5 8 9
4	0	-1	4	5 7 8
5	0	0	5	3 4 2
6	1	-1	6	4 5 2
7	1	0	7	4 7 5
8	1	1	8	4 6 7
9	0	1		

Matrix `_nodecoords` Matrix `_elements`

2.4.2 Linear Finite Element Space

§2.4.2.1 (Recalled: Piecewise linear functions in 1D) Recall the spline space $\mathcal{S}_{1,0}^0(\mathcal{M}) \subset H_0^1([a,b])$ of piecewise linear functions on a 1D grid \mathcal{M} with M cells, see § 2.3.1.4, that was used as Galerkin trial/test space in 1D in Section 2.3.



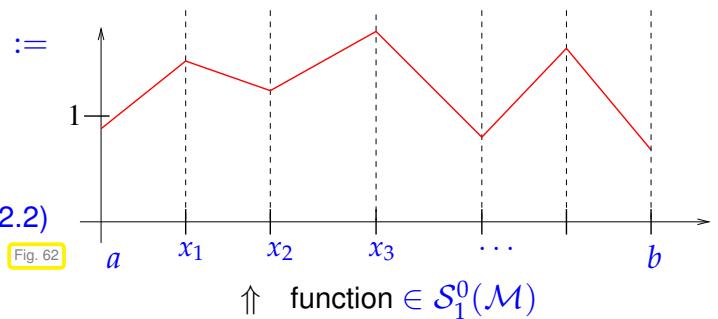
1D linear finite element trial space on mesh $\mathcal{M} := \{x_{j-1}, x_j\} : j = 1, \dots, M\}$ of $\Omega :=]a, b[\subset \mathbb{R}$:

$$\mathcal{S}_{1,0}^0(\mathcal{M}) := \left\{ v \in C^0([a, b]): v|_{[x_{i-1}, x_i]} \text{ linear, } i = 1, \dots, M, v(a) = v(b) = 0 \right\}$$

This was tailored to provide the test space $V_{0,h}$ for a two-point Dirichlet problem. We can easily alter this space to obtain a piecewise linear finite element space suitable for the Galerkin discretization of a two-point Neumann problem.

Full 1D linear finite element space on mesh $\mathcal{M} := \{x_{j-1}, x_j\} : j = 1, \dots, M\}$ of $]a, b[\subset \mathbb{R}$:

$$\mathcal{S}_1^0(\mathcal{M}) := \left\{ v \in C^0([a, b]): v|_{[x_{i-1}, x_i]} \text{ linear } \forall i \right\} \quad (2.4.2.2)$$



The goal is to generalize the space $\mathcal{S}_1^0(\mathcal{M})$ as defined in (2.4.2.2) to 2D and 3D. To do so we first extend the concept of (affine) linear scalar-valued functions. The following table exhibits the natural correspondence of concepts in 1D and 2D. Then we define $\mathcal{S}_{1,0}^0(\mathcal{M})$ over a triangular mesh in 2D in the same fashion as we defined it in 1D over a partition of an interval.

$d = 1$

$d = 2$

Grid/mesh **cells**: intervals $]x_{i-1}, x_i[$, $i = 1, \dots, M$

triangles K_i , $i = 1, \dots, M$

Linear functions: $x \in \mathbb{R} \mapsto \alpha + \beta \cdot x$, $\alpha, \beta \in \mathbb{R}$ $x \in \mathbb{R}^2 \mapsto \alpha + \beta \cdot x$, $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}^2$

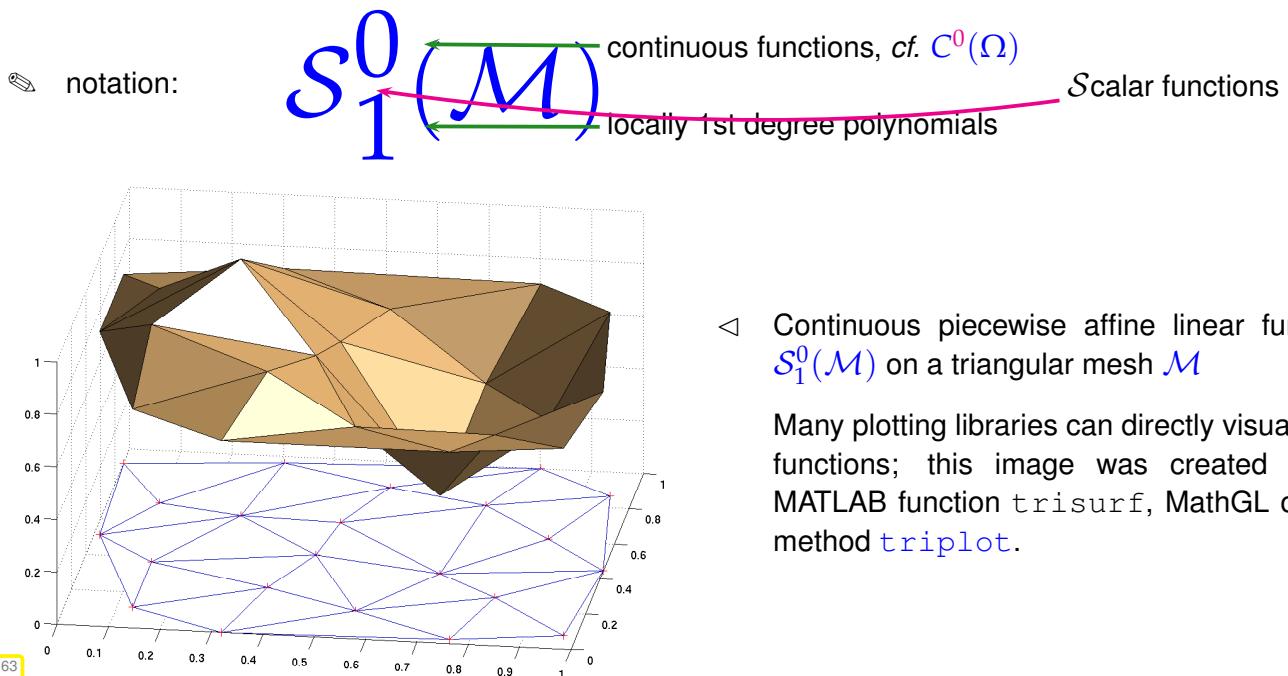
This suggests that we try a definition analogous to the 1D case (2.4.2.2):

$$V_{0,h} = \mathcal{S}_1^0(\mathcal{M}) := \left\{ v \in C^0(\bar{\Omega}): \forall K \in \mathcal{M}: v|_K(x) = \alpha_K + \beta_K \cdot x, \alpha_K \in \mathbb{R}, \beta_K \in \mathbb{R}^2, x \in K \right\} \subset H^1(\Omega)$$

see Thm. 1.3.4.23

Recall that Thm. 1.3.4.23 tells us that a function that is piecewise (w.r.t to a “nice” partition of Ω) smooth and bounded belongs to $H^1(\Omega)$, if and only if it is continuous on the entire domain $\bar{\Omega}$. This accounts for the requirement $v \in C^0(\bar{\Omega})$ in the above definition.

Parlance: Functions of the form $x \mapsto \alpha_K + \beta_K \cdot x$, $\alpha_K \in \mathbb{R}$, $\beta_K \in \mathbb{R}^2$ are called (affine) linear.



Continuous piecewise affine linear function $\in S_1^0(\mathcal{M})$ on a triangular mesh \mathcal{M}

Many plotting libraries can directly visualize such functions; this image was created with the MATLAB function `trisurf`, MathGL offers the method `triplot`.

Remark 2.4.2.3 (Piecewise gradient → Section 1.3) Functions in $S_1^0(\mathcal{M})$ will usually have kinks across intercell interfaces, which rules out global differentiability. However, we can differentiate them nevertheless, if we do not insist on getting a value for the derivative in every point. This is acceptable, if the derivative occurs only under an integral as it is invariably the case in weak formulations. Again, also in two dimensions we reap the benefit of the reduced smoothness requirements inherent in a variational formulation, cf. Rem. 2.3.2.7.

Moreover, we can simply patch together the derivative by separately differentiating the smooth parts of the function, see Ex. 1.3.4.24 and (2.3.2.6) for the same reasoning in 1D. Summing up:

Thm. 1.3.4.23 $\Rightarrow S_1^0(\mathcal{M}) \subset H^1(\Omega)$, because $S_1^0(\mathcal{M}) \subset C^0(\bar{\Omega})$ and piecewise smooth.
 \Rightarrow for $u_h \in S_1^0(\mathcal{M})$ the gradient $\mathbf{grad} u_h$ can be computed on each triangle as piecewise constant function, cf. Ex. 1.3.4.27.
 (Easy: on $K \in \mathcal{M}$: $\mathbf{grad}\{x \mapsto \alpha_K + \beta_K \cdot x\} = \beta_K \in \mathbb{R}^2$)

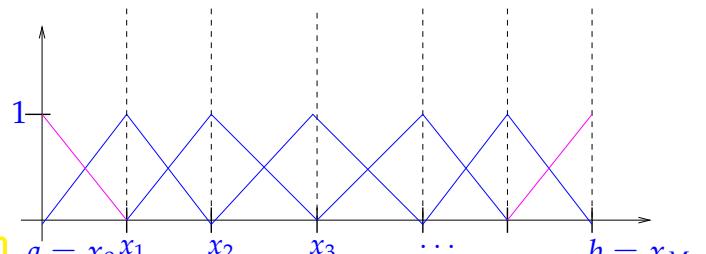
2.4.3 Nodal Basis Functions

Our next goal is the generalization of “tent functions”, see (2.3.2.1).

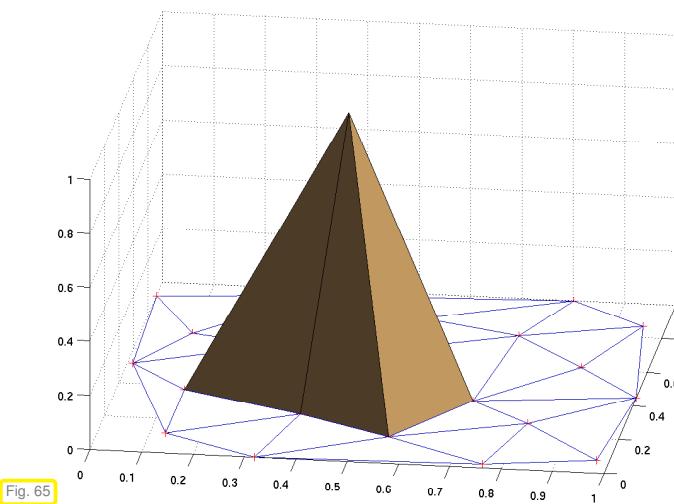
Recall the 1D “tent functions” [Hip19, ??] \triangleright
 In 1D, adding two more “half-tent” functions, cardinal basis functions belonging to the endpoints x_0 and x_M , we obtain a basis of $S_1^0(\mathcal{M})$:

$$\mathfrak{B} = \{b_h^0, \dots, b_h^M\}, \quad (2.4.3.1)$$

$$b_h^j(x_i) = \delta_{ij} := \begin{cases} 1 & , \text{if } i = j, \\ 0 & , \text{if } i \neq j, \end{cases} \quad (2.3.2.2)^{[4]}$$



The “nodal (value) property” condition (2.3.2.2) already defines a tent function in the space $S_1^0(\mathcal{M})$. This approach carries over to 2D.



Idea: define (?) basis function b_h^x , $x \in \mathcal{V}(\mathcal{M})$, by “nodal conditions”

$$b_h^x(y) = \begin{cases} 1 & , \text{ if } y = x , \\ 0 & , \text{ if } y \in \mathcal{V}(\mathcal{M}) \setminus \{x\} . \end{cases} \quad (2.4.3.2)$$

Is this possible ?

§2.4.3.3 (Fixing a piecewise affine linear function) Heuristic reasoning: there is exactly one plane through three non-collinear points in \mathbb{R}^3 . Moreover, the graph of a linear function $\mathbb{R}^2 \mapsto \mathbb{R}$ is a plane.

This can be made rigorous by a little linear algebra. Let $x \rightarrow \alpha + \beta \cdot x$ describe the plane through (a^1, v_1) , (a^2, v_2) , (a^3, v_3) , $v_i \in \mathbb{R}$, $a^i \in \mathbb{R}^2$ not collinear. Then α, β_1, β_2 satisfy the linear system of equations

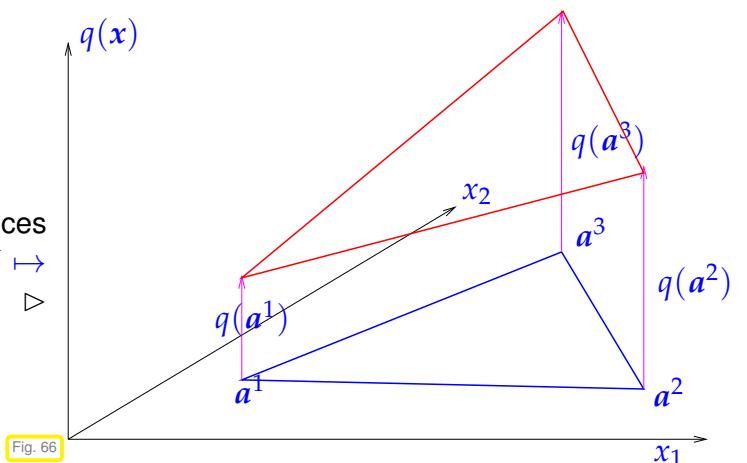
$$\begin{bmatrix} 1 & a_1^1 & a_2^1 \\ 1 & a_1^2 & a_2^2 \\ 1 & a_1^3 & a_2^3 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \quad (2.4.3.4)$$

where $a^i = \begin{bmatrix} a_1^i \\ a_2^i \end{bmatrix}$. Since the points a^i do not lie on a line, the vectors $a^2 - a^1$ and $a^3 - a^1$ are linearly independent, which ensures that (2.4.3.4) always has a unique solution, because

$$\det \begin{bmatrix} 1 & a_1^1 & a_2^1 \\ 1 & a_1^2 & a_2^2 \\ 1 & a_1^3 & a_2^3 \end{bmatrix} = \det [a_1^2 - a_1^1 \ a_2^2 - a_2^1 \ a_3^2 - a_3^1] = 2|K| \neq 0 .$$

“Visual proof”:

On a non-degenerate triangle K with vertices a^1, a^2, a^3 : an (affine) linear scalar function $q : K \mapsto \mathbb{R}$ is uniquely determined by the values $q(a^i)$. \triangleright



Issue: Is a \mathcal{M} -piecewise affine linear function $v : \Omega \rightarrow \mathbb{R}$ continuous, when its vertex values are fixed?

Yes, because on each edge e of \mathcal{M} $v|_e$ is linear and, thus, uniquely determined by its values in the endpoints of e , see Fig. 66 for an illustration. As a consequence, v has the same value on e no matter from which side it is approached.



$v_h \in \mathcal{S}_1^0(\mathcal{M})$ is uniquely determined by $\{v_h(x), x \text{ node of } \mathcal{M}\}$!



$\dim \mathcal{S}_1^0(\mathcal{M}) = \#\mathcal{V}(\mathcal{M})$ $(\mathcal{V}(\mathcal{M}))$ = set of nodes (= vertices of triangles) of \mathcal{M})

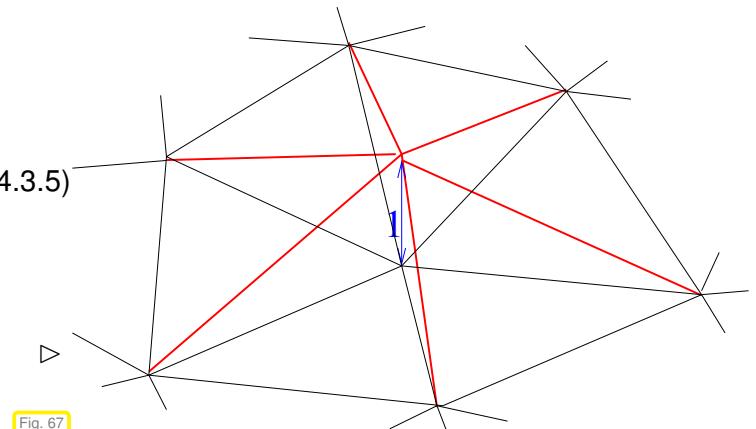
Note that the *condition (iv)* for a valid triangulation makes possible the construction of the basis function b_h^x for each $x \in \mathcal{V}(\mathcal{M})$; no simple basis functions could be associated with the red vertices (“hanging nodes”) in Fig. 58. \square

Now we have found the perfect 2D counterpart of the tent function basis (\rightarrow (2.3.2.2)) of the linear finite element space in 1D:

Writing $\mathcal{V}(\mathcal{M}) = \{x_1, \dots, x_N\}$, an ordering of the nodes is implied, the **nodal basis** $\mathfrak{B}_h := \{b_h^1, \dots, b_h^N\}$ of $\mathcal{S}_1^0(\mathcal{M})$ is defined by the conditions

$$\begin{aligned} b_h^i &\in \mathcal{S}_1^0(\mathcal{M}), \\ b_h^i(x_j) &= \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{else,} \end{cases} \\ i, j &\in \{1, \dots, N\}. \end{aligned} \quad (2.4.3.5)$$

Piecewise linear nodal basis function
("hat function"/ "tent function")



Note that the defining relations (2.4.3.5) amount to the **cardinal basis** property of \mathfrak{B}_h with respect to the node set $\mathcal{V}(\mathcal{M}) = \{x_1, \dots, x_N\}$. As a consequence, the coefficients of the nodal basis expansion of a $u_h \in \mathcal{S}_1^0(\mathcal{M})$ agree with the so-called **nodal values** $u_h(x_i)$:

$$u_h \in \mathcal{S}_1^0(\mathcal{M}): \quad u_h = \sum_{i=1}^N \mu_i b_h^i \Leftrightarrow \mu_i = u_h(x_i) \quad \forall i = 1, \dots, N. \quad (2.4.3.6)$$

§2.4.3.7 (Linear finite element space for homogeneous Dirichlet problem) Recall that the Dirichlet problem with homogeneous boundary conditions $u|_{\partial\Omega} = 0$ is posed on the Sobolev space $H_0^1(\Omega)$ (\rightarrow Def. 1.3.4.3), see (1.4.2.4), Ex. 1.8.0.2.

This leads to a “formal” characterization:

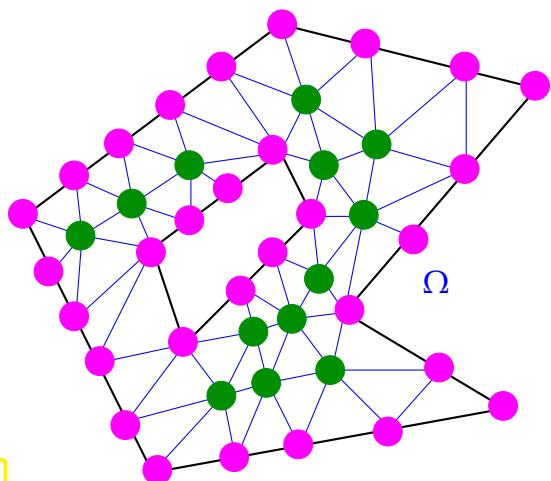
Galerkin space for homogeneous Dirichlet b.c.: $V_{0,h} = \mathcal{S}_{1,0}^0(\mathcal{M}) := \mathcal{S}_1^0(\mathcal{M}) \cap H_0^1(\Omega)$

☞ Notation: $\mathcal{S}_{1,0}^0(\mathcal{M})$ zero on $\partial\Omega$, cf. $H_0^1(\Omega)$

Fortunately, this space can immediately be obtained from $\mathcal{S}_1^0(\mathcal{M})$ by dropping basis functions on the boundary:

► $\mathcal{S}_{1,0}^0(\mathcal{M}) = \text{Span}\{b_h^j : x_j \in \Omega \text{ (interior node !)}\}$

► $\dim \mathcal{S}_{1,0}^0(\mathcal{M}) = \#\{x \in \mathcal{V}(\mathcal{M}) : x \notin \partial\Omega\}$



▷ “Location” of nodal basis functions:
(mesh \mathcal{M} → Fig. 57)

•, ● → nodal basis functions of $\mathcal{S}_1^0(\mathcal{M})$

● → nodal basis functions of $\mathcal{S}_{1,0}^0(\mathcal{M})$

Bottom line: the linear finite element trial/test space contained in $H_0^1(\Omega)$ is obtained by culling from the basis all “tent functions” of $\mathcal{S}_1^0(\mathcal{M})$ that do not vanish on $\partial\Omega$.

2.4.4 Sparse Galerkin Matrix

Already for linear finite element Galerkin discretization in one dimension in Section 2.3 the *tridiagonal* structure of the Galerkin matrices (2.3.3.14) caught our attention. Will Galerkin matrices in 2D also turn out to be banded?

To answer this question, we now study the filling pattern of the Galerkin matrix arising from the discretization of a 2nd-order scalar linear elliptic variational problem with linear finite elements. By filling pattern we mean the number and location of non-zero entries of that matrix. It will turn out that Galerkin matrices are always sparse, that is, most of their entries vanish.

Throughout the discussion we have in mind a second-order elliptic variational problem featuring a bilinear form

$$\mathbf{a}(u, v) := \int_{\Omega} (\alpha(x) \mathbf{grad} u) \cdot \mathbf{grad} v + c(x) u v \, dx = \int_{\partial\Omega} h v \, dS, \quad u, v \in H^1(\Omega). \quad (2.4.4.1)$$

We write b_h^j for the nodal basis function associated with the node x_j of the triangulation \mathcal{M} of Ω , see Section 2.4.3.

Note that \mathbf{a} is symmetric which will spawn a symmetric Galerkin matrix.

Now, for the “tent” basis functions b_h^i of $\mathcal{S}_1^0(\mathcal{M})$ from (2.4.3.5), we study the **sparsity** (→ [Hip19, ??]) of the Galerkin matrix

$$\mathbf{A} := \left(\mathbf{a}(b_h^i, b_h^j) \right)_{i,j=1}^N \in \mathbb{R}^{N,N}, \quad N := \dim \mathcal{S}_1^0(\mathcal{M}) = \#\mathcal{V}(\mathcal{M}),$$

as introduced in Section 2.2.

The consideration are fairly parallel to those that made us understand that the Galerkin matrix for the 1D case was *tridiagonal*, see (2.3.3.2). Again, a key concept is that of the **support** of a function as defined in Def. 2.3.2.5. We first examine the possible relative locations of the supports of two nodal basis functions.

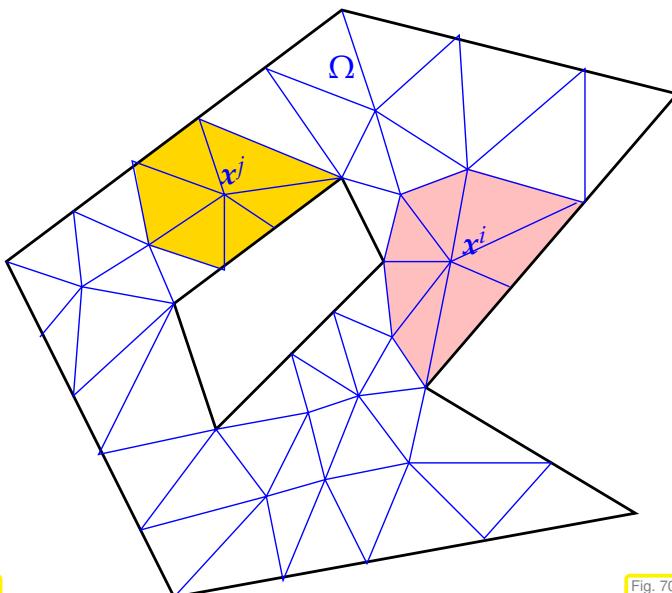


Fig. 69

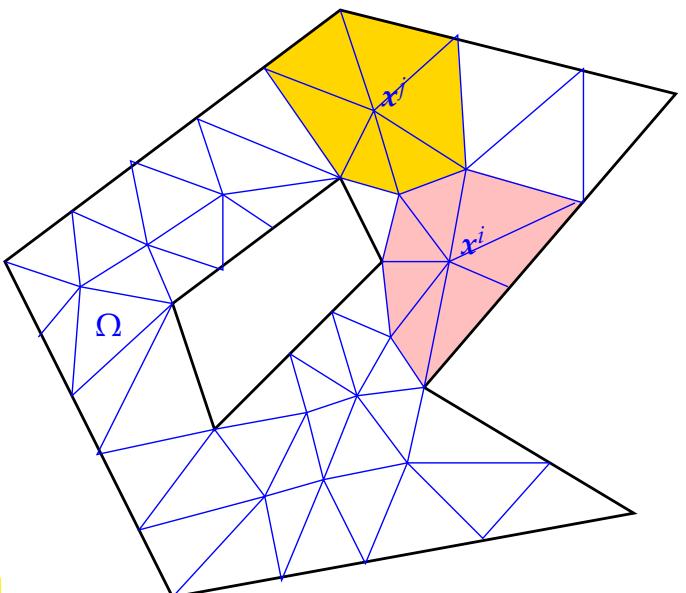


Fig. 70

$$\left\{ \begin{array}{l} \text{Nodes } x_i, x_j \in \mathcal{V}(\mathcal{M}) \\ \text{not connected by an edge} \end{array} \right. \Leftrightarrow \text{Vol}(\text{supp}(b_h^i) \cap \text{supp}(b_h^j)) = 0 \} \Rightarrow (\mathbf{A})_{ij} = 0 .$$

For instance, the number of non-zero entries in row i of the Galerkin matrix tells us, the minimal number of edges abutting the node x_i .

Lemma 2.4.4.2. Sparsity of Galerkin matrix

There is a constant $C > 0$ depending only on the topology of Ω , that is, the number of “holes” in it, such that for any triangular mesh \mathcal{M} of Ω ($N := \#\mathcal{V}(\mathcal{M}) = \text{number of vertices}$)

$$\#\{(i, j) \in \{1, \dots, N\}^2 : (\mathbf{A})_{ij} \neq 0\} \leq 7 \cdot N + C ,$$

where \mathbf{A} is any Galerkin matrix arising from a discretization of a 2nd-order linear scalar elliptic variational problem with linear finite elements.

Proof. We rely on Euler’s formula for triangulations.

$$\#\mathcal{M} - \#\mathcal{E}(\mathcal{M}) + \#\mathcal{V}(\mathcal{M}) = \chi_\Omega , \quad \chi_\Omega = \text{Euler characteristic of } \Omega .$$

Note that χ_Ω is a topological invariant (alternating sum of Betti numbers).

By combinatorial considerations (traverse edges and count triangles):

$$2 \cdot \#\mathcal{E}_I(\mathcal{M}) + \#\mathcal{E}_B(\mathcal{M}) = 3 \cdot \#\mathcal{M} ,$$

where $\mathcal{E}_I(\mathcal{M}), \mathcal{E}_B(\mathcal{M})$ stand for the sets of interior and boundary edges of \mathcal{M} , respectively.

$$\blacktriangleright \quad \#\mathcal{E}_I(\mathcal{M}) + 2\#\mathcal{E}_B(\mathcal{M}) = 3(\#\mathcal{V}(\mathcal{M}) - \chi_\Omega) .$$

Then use

$$N = \#\mathcal{V}(\mathcal{M}) , \quad \text{nnz}(\mathbf{A}) \leq N + 2 \cdot \#\mathcal{E}(\mathcal{M}) \leq 7 \cdot \#\mathcal{V}(\mathcal{M}) - 6\chi_\Omega ,$$

which yields the assertion for any triangulation. \square

Recall from [Hip19, ??] (not a definition in a rigorous mathematical sense):

Notion 2.4.4.3. Sparse matrix

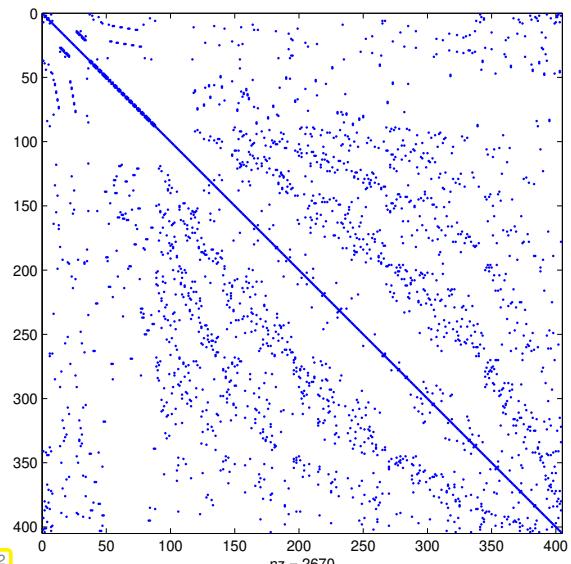
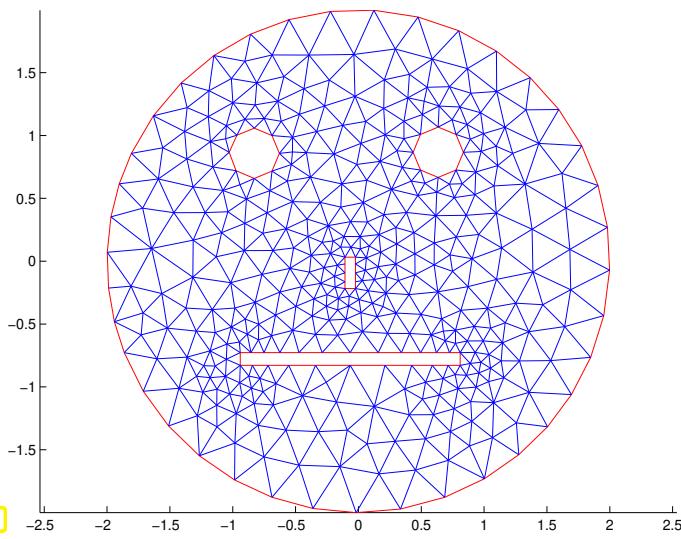
$\mathbf{A} \in \mathbb{K}^{m,n}$, $m, n \in \mathbb{N}$, is **sparse**, if

$$\text{nnz}(\mathbf{A}) := \#\{(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\} : a_{ij} \neq 0\} \ll mn.$$

Sloppy parlance: matrix **sparse** : \Leftrightarrow “almost all” entries = 0 /“only a few percent of” entries $\neq 0$

Galerkin discretization of a 2nd-order linear variational problem
utilizing the *nodal basis* of $\mathcal{S}_1^0(\mathcal{M})/\mathcal{S}_{1,0}^0(\mathcal{M})$
leads to **sparse** linear systems of equations.

EXAMPLE 2.4.4.4 (Sparse Galerkin matrices) We compute on the triangular mesh \mathcal{M} shown in Fig. 71 (left figure). The right figure displays a “spy-plot” of the Galerkin matrix obtained from the finite element Galerkin discretization of a second-order elliptic Dirichlet BVP by means of the trial/test space $V_{0,h} = \mathcal{S}_{1,0}^0(\mathcal{M})$.



A spy-plot indicates the location of the non-zero entries of a matrix by drawing a dot at the position of every non-zero entry. □

2.4.5 Computation of Galerkin Matrix

Now we learn about an efficient algorithm for computing the non-zero entries of the sparse finite element Galerkin matrix. For sake of simplicity we consider the bilinear form for the Poisson equation

$$a(u, v) := \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx, \quad u, v \in H_0^1(\Omega).$$

and a Galerkin discretization based on

- a triangular mesh, see Section 2.4.1, set of vertices $\{\mathbf{x}_i\} = \mathcal{V}(\mathcal{M})$,
- the discrete trial/test space $\mathcal{S}_1^0(\mathcal{M}) \subset H^1(\Omega)$,

- the *nodal basis* $\mathfrak{B}_h = \{b_h^j\}$ of tent functions according to (2.4.3.2).

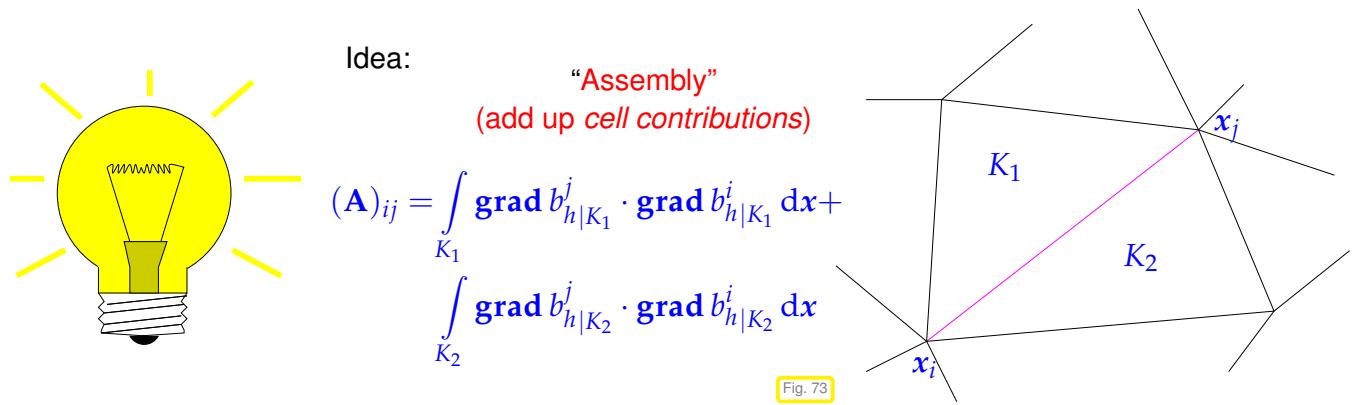
The entries of the Galerkin matrix \mathbf{A} are given by the formula

$$(\mathbf{A})_{i,j} = a(b_h^j, b_h^i) = \int_{\Omega} \mathbf{grad} b_h^j \cdot \mathbf{grad} b_h^i \, dx.$$

In Section 2.4.4 we realized that when computing $(\mathbf{A})_{i,j}$ we need deal only with the situations, where the two nodes $x_i, x_j \in \mathcal{V}(\mathcal{M})$

- are connected by an edge of the triangulation,
- coincide,

because in all other cases the matrix entries are known to vanish a priori. We first elaborate the case (i):



2.4.5.1 Local Computations

Motivated by the above formula we now zero in on a single triangle $K \in \mathcal{M}$, restrict the bilinear form to it, and examine the cell contribution

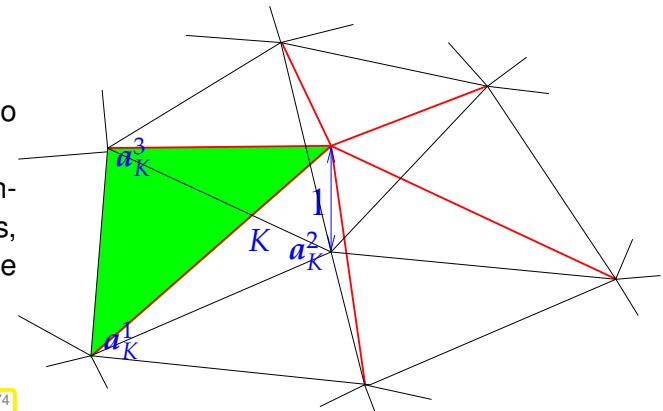
$$a_K(b_h^j, b_h^i) := \int_K \mathbf{grad} b_h^j|_K \cdot \mathbf{grad} b_h^i|_K \, dx , \quad x_i, x_j \text{ nodes } \in \text{vertices of } K. \quad (2.4.5.1)$$

§2.4.5.2 (Barycentric coordinate functions) We aim to find analytic formulas for the restrictions $b_h^i|_K$. If a_K^1, a_K^2, a_K^3 are the vertices of the triangle K with coordinates $a_K^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}$, $a_K^2 = \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix}$, and $a_K^3 = \begin{bmatrix} a_1^3 \\ a_2^3 \end{bmatrix}$, we write

$$\lambda_i := b_h^i|_K \quad \text{with} \quad a_K^i = x_j \quad \left[\begin{array}{l} i \leftrightarrow \text{local vertex number} \\ j \leftrightarrow \text{global node number} \end{array} \right] \quad (2.4.5.3)$$

The functions $\lambda_1, \lambda_2, \lambda_3$ on the triangle K are also known as **barycentric coordinate functions**.

The barycentric coordinate functions are the non-trivial restrictions of 2D tent functions to triangles, see Fig. 74, where the green surface represents the graph of λ_2 .

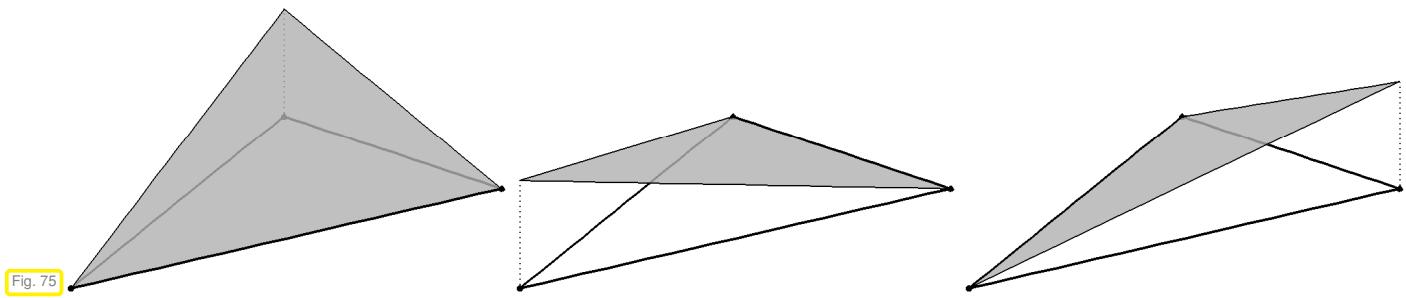


The barycentric coordinate functions owe their name to the fact that they can be regarded as “coordinates of a point with respect to the vertices of a triangle” in the sense that

$$\mathbf{x} = \lambda_1(\mathbf{x})\mathbf{a}_K^1 + \lambda_2(\mathbf{x})\mathbf{a}_K^2 + \lambda_3(\mathbf{x})\mathbf{a}_K^3.$$

The attribute “barycentric” is related to barycenter = center of gravity, which has barycentric coordinates $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

Plots of the graphs of the functions $\lambda_i, i = 1, 2, 3$ look as follows:

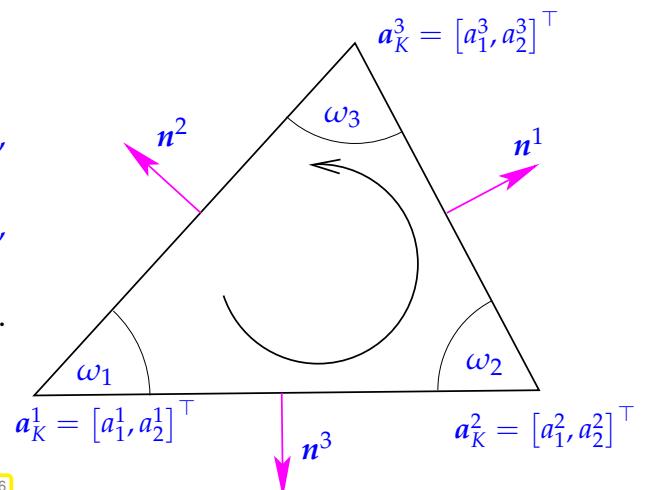


Restrictions $\lambda_1, \lambda_2, \lambda_3$ of p.w. linear nodal basis functions of $\mathcal{S}_1^0(\mathcal{M})$ to a triangle K

Formulas for the λ_i can easily be found from the defining property that they are affine linear and that $\lambda_i(\mathbf{a}_K^j) = \delta_{ij}$.

$$\begin{aligned}\lambda_1(\mathbf{x}) &= \frac{1}{2|K|} (\mathbf{x} - \mathbf{a}_K^2) \cdot \begin{bmatrix} \mathbf{a}_K^2 - \mathbf{a}_K^3 \\ \mathbf{a}_1^3 - \mathbf{a}_1^2 \end{bmatrix} = -\frac{|\mathbf{e}_1|}{2|K|} (\mathbf{x} - \mathbf{a}_K^2) \cdot \mathbf{n}^1, \\ \lambda_2(\mathbf{x}) &= \frac{1}{2|K|} (\mathbf{x} - \mathbf{a}_K^3) \cdot \begin{bmatrix} \mathbf{a}_K^3 - \mathbf{a}_K^1 \\ \mathbf{a}_1^1 - \mathbf{a}_1^3 \end{bmatrix} = -\frac{|\mathbf{e}_2|}{2|K|} (\mathbf{x} - \mathbf{a}_K^3) \cdot \mathbf{n}^2, \\ \lambda_3(\mathbf{x}) &= \frac{1}{2|K|} (\mathbf{x} - \mathbf{a}_K^1) \cdot \begin{bmatrix} \mathbf{a}_K^1 - \mathbf{a}_K^2 \\ \mathbf{a}_1^2 - \mathbf{a}_1^1 \end{bmatrix} = -\frac{|\mathbf{e}_3|}{2|K|} (\mathbf{x} - \mathbf{a}_K^1) \cdot \mathbf{n}^3.\end{aligned}$$

(\mathbf{e}_i = edge opposite vertex \mathbf{a}_K^i , see Figure for numbering scheme \triangleright)



Obviously, these formulas define affine linear functions. Their complete justification of appeals to the distance formula for a point w.r.t. to a line given in **Hesse normal form**:

$$|(\mathbf{a}_K^i - \mathbf{a}_K^j) \cdot \mathbf{n}_i| = \text{dist}(\mathbf{a}_K^i; \mathbf{e}_i) = h_i \quad (\text{height}) \quad \text{and} \quad 2|K| = |\mathbf{e}_i| h_i \Rightarrow \lambda_i(\mathbf{a}_K^i) = 1.$$

This shows that the λ_i really provide the restrictions of p.w. linear nodal basis functions (tent functions) of $\mathcal{S}_1^0(\mathcal{M})$ to triangle K , because they are clearly (affine) linear and comply with (2.4.3.2).

For the constant gradients of the barycentric coordinate functions we find

$$\mathbf{grad} \lambda_1 = -\frac{|\mathbf{e}_1|}{2|K|} \mathbf{n}^1 = \frac{1}{2|K|} (\mathbf{a}_K^2 - \mathbf{a}_K^3)^\perp = \frac{1}{2|K|} \begin{bmatrix} \mathbf{a}_2^2 - \mathbf{a}_2^3 \\ \mathbf{a}_1^3 - \mathbf{a}_1^2 \end{bmatrix}, \quad (2.4.5.4a)$$

$$\mathbf{grad} \lambda_2 = -\frac{|\mathbf{e}_2|}{2|K|} \mathbf{n}^2 = \frac{1}{2|K|} (\mathbf{a}_K^3 - \mathbf{a}_K^1)^\perp = \frac{1}{2|K|} \begin{bmatrix} \mathbf{a}_2^3 - \mathbf{a}_2^1 \\ \mathbf{a}_1^1 - \mathbf{a}_1^3 \end{bmatrix}, \quad (2.4.5.4b)$$

$$\mathbf{grad} \lambda_3 = -\frac{|\mathbf{e}_3|}{2|K|} \mathbf{n}^3 = \frac{1}{2|K|} (\mathbf{a}_K^1 - \mathbf{a}_K^2)^\perp = \frac{1}{2|K|} \begin{bmatrix} \mathbf{a}_2^1 - \mathbf{a}_2^2 \\ \mathbf{a}_1^2 - \mathbf{a}_1^1 \end{bmatrix}. \quad (2.4.5.4c)$$

Here \mathbf{x}^\perp for $\mathbf{x} \in \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ indicates clockwise rotation by $\pi/2$: $\mathbf{x}^\perp := \begin{bmatrix} x_2 \\ -x_1 \end{bmatrix}$. \square

§2.4.5.5 (A formula for the element matrix for $-\Delta$) Armed with the formula (2.4.5.4), $\mathbf{grad} \lambda_i = -\frac{|e_i|}{2|K|} \mathbf{n}^i$, $i = 1, 2, 3$, we can now compute the contributions to entries of the Galerkin matrix as entries of the **element (stiffness) matrix**

$$\mathbf{A}_K = \left[\int_K \mathbf{grad} \lambda_i \cdot \mathbf{grad} \lambda_j \, dx \right]_{i,j=1}^3 \in \mathbb{R}^{3,3}. \quad (2.4.5.6)$$

Concretely, since the gradients are constant we get

$$a(\lambda_i, \lambda_j) = \int_K \mathbf{grad} \lambda_i \cdot \mathbf{grad} \lambda_j \, dx = \frac{1}{4|K|} |e_i| |e_j| \mathbf{n}_i \cdot \mathbf{n}_j.$$

Then use \spadesuit the angle formula $\mathbf{n}_i \cdot \mathbf{n}_j = \cos(\pi - \omega_k) = -\cos \omega_k$, ($i \neq j$)
 \spadesuit the area formula $|K| = \frac{1}{2} |e_i| |e_j| \sin \omega_k$, ($i \neq j$).

The case $i = j$ employs a trick:

$$\sum_{i=1}^3 \lambda_i = 1 \Rightarrow \sum_{i=1}^3 a_K(\lambda_i, \lambda_j) = 0 \Rightarrow a(\lambda_i, \lambda_i) = - \sum_{j \neq i} a(\lambda_i, \lambda_j). \quad (2.4.5.7)$$

Plugging in these formulas we end up with [KA03, Lemma 3.47]

$$\mathbf{A}_K = \frac{1}{2} \begin{bmatrix} \cot \omega_3 + \cot \omega_2 & -\cot \omega_3 & -\cot \omega_2 \\ -\cot \omega_3 & \cot \omega_3 + \cot \omega_1 & -\cot \omega_1 \\ -\cot \omega_2 & -\cot \omega_1 & \cot \omega_2 + \cot \omega_1 \end{bmatrix}. \quad (2.4.5.8)$$

The local numbering and naming conventions are displayed in Fig. 76. \square

Remark 2.4.5.9 (Alternative computation of element matrix for $-\Delta$) From (2.4.3.4) we conclude that the coefficients in the representation $\lambda_i(\mathbf{x}) = \alpha_i + \beta^i \cdot \mathbf{x}$ of the barycentric coordinate functions $\lambda_1, \lambda_2, \lambda_3$ on a triangle with vertices $\mathbf{a}_K^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}$, $\mathbf{a}_K^2 = \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix}$, and $\mathbf{a}_K^3 = \begin{bmatrix} a_1^3 \\ a_2^3 \end{bmatrix}$ satisfy

$$\begin{bmatrix} 1 & a_1^1 & a_2^1 \\ 1 & a_1^2 & a_2^2 \\ 1 & a_1^3 & a_2^3 \end{bmatrix} \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1^1 & \beta_1^2 & \beta_1^3 \\ \beta_2^1 & \beta_2^2 & \beta_2^3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.4.5.10)$$

Observe that $\mathbf{grad} \lambda_i = \beta^i$, which explains, why Code 2.4.5.11 computes the gradients of the barycentric coordinate functions:

C++ code 2.4.5.11: Computation of gradients of barycentric coordinate functions on a triangle → [GITHUB](#)

```

2 Eigen::Matrix<double, 2, 3> gradbarycoordinates(const TriGeo_t& vertices) {
3     Eigen::Matrix<double, 3, 3> X;
4     // Argument vertices passes the vertex positions of the triangle
5     // as the columns of a  $2 \times 3$ -matrix, , see
6     // Code 2.4.1.3. The function returns the components of the
7     // gradients as the columns of a  $2 \times 3$ -matrix
8
9     // Computation based on (2.4.5.10), solving for the
10    // coefficients of the barycentric coordinate functions.
11    X.block<3, 1>(0, 0) = Eigen::Vector3d::Ones();
12    X.block<3, 2>(0, 1) = vertices.transpose();
13    return X.inverse().block<2, 3>(1, 0);
14 }
```

This suggests another way to compute the element matrix \mathbf{A}_K given in (2.4.5.8):

$$\mathbf{A}_K = |K| \begin{bmatrix} \beta_1^1 & \beta_1^2 & \beta_1^3 \\ \beta_2^1 & \beta_2^2 & \beta_2^3 \\ \beta_3^1 & \beta_3^2 & \beta_3^3 \end{bmatrix}^\top \begin{bmatrix} \beta_1^1 & \beta_1^2 & \beta_1^3 \\ \beta_2^1 & \beta_2^2 & \beta_2^3 \\ \beta_3^1 & \beta_3^2 & \beta_3^3 \end{bmatrix} \in \mathbb{R}^{3,3}. \quad (2.4.5.12)$$

This formula should be used in a code, because directly evaluating (2.4.5.8) requires the angles ω_k and involves expensive evaluations of trigonometric functions.

C++ code 2.4.5.13: Computation of element matrix for $-\Delta$ on a triangle and for linear Lagrangian finite elements → [GITHUB](#)

```

2 Eigen::Matrix3d ElementMatrix_LapI_LFE(const TriGeo_t& V) {
3     // Argument V same as vertices in Code 2.4.5.11.
4     // The function returns the  $3 \times 3$  element matrix as a fixed size
5     // EIGEN matrix.
6
7     // Evaluate (2.4.5.1), exploiting that the gradients are constant.
8     // First compute the area of triangle by determinant formula
9     double area = 0.5 * std::abs((V(0, 1) - V(0, 0)) * (V(1, 2) - V(1, 1)) -
10                                (V(0, 2) - V(0, 1)) * (V(1, 1) - V(1, 0)));
11
12     // Compute gradients of barycentric coordinate functions, see
13     // Code 2.4.5.11
14     Eigen::Matrix<double, 2, 3> X = gradbarycoordinates(V);
15     // compute inner products of gradients through matrix multiplication
16     return area * X.transpose() * X;
}
```

Supplement 2.4.5.14 (Scaling of entries of element matrix for $-\Delta$) When we scale a mesh, we subject all cells to a uniform dilation. Let us elaborate, how entries of the Galerkin matrix change in the process.

An observation:

(2.4.5.8) ➤ \mathbf{A}_K does not depend on the “size” of triangle K !
 (more precisely, element matrices are equal for *similar* triangles)

This can be seen by the following reasoning:

- Obviously translation and rotation of K does not change. \mathbf{A}_K
- Scaling of K by a factor $\rho > 0$ has the following effect that
 - the area $|K|$ is scaled by ρ^2 ,
 - the gradients $\mathbf{grad} \lambda_i$ are scaled by ρ^{-1} (the barycentric coordinate functions λ_i become steeper when the triangle shrinks in size.).

Both effects just offset in a_K from (2.4.5.1) such that \mathbf{A}_K remains invariant under scaling.

Note, however that for $d = 3$ the element matrix \mathbf{A}_K behaves differently under scaling. What is the scaling in 3D precisely?

□

2.4.5.2 Assembly of Full Galerkin Matrix

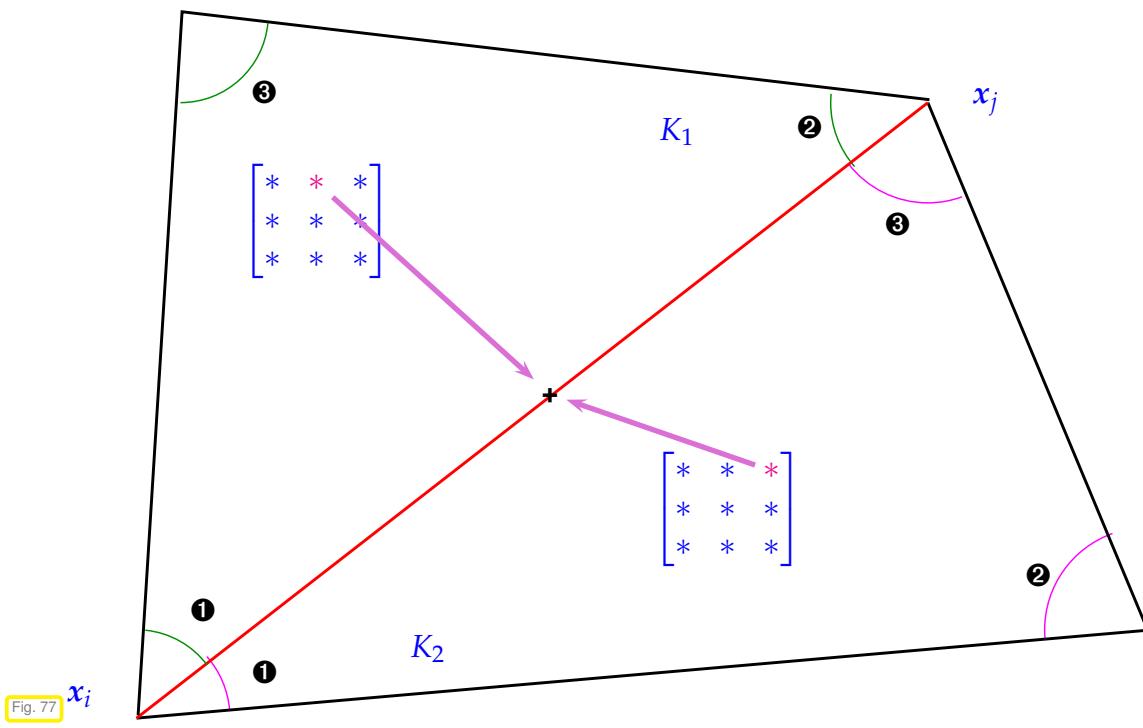
Now we tackle the computation of the full Galerkin matrix $\mathbf{A} \in \mathbb{R}^{N,N}$. This so-called “assembly” of $(\mathbf{A})_{ij}$ for $i \neq j$ starts from summing cell contributions

$$(\mathbf{A})_{ij} = \int_{K_1} \mathbf{grad} b_h^j|_{K_1} \cdot \mathbf{grad} b_h^i|_{K_1} dx + \int_{K_2} \mathbf{grad} b_h^j|_{K_2} \cdot \mathbf{grad} b_h^i|_{K_2} dx.$$

$(\mathbf{A})_{ij}$ can be obtained by summing respective^(*) entries of the element matrices of the elements adjacent to the edge connecting x^i and x^j .

(*): determined by the correspondence of local and global vertex numbers!

When we use (2.4.5.8), the origin of the matrix entry $(\mathbf{A})_{ij}$, $i \neq j$, can be visualized as follows (❶, ❷, and ❸ give the local vertex numbers):



Next we look at the “assembly” of the diagonal entry $(\mathbf{A})_{ii}$ of the Galerkin matrix \mathbf{A} . It can be obtained by summing corresponding diagonal entries of element matrices belonging to triangles adjacent to node x_i .

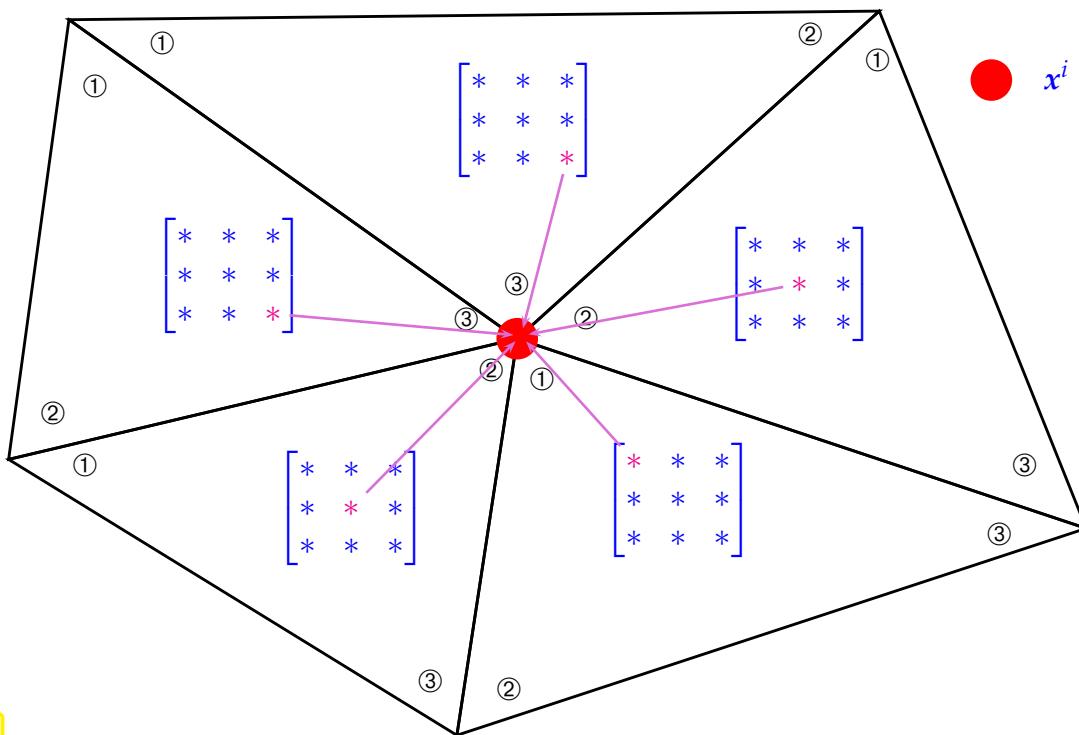


Fig. 78

(A)_{ii} by summing diagonal entries of element matrices of adjacent triangles

§2.4.5.15 (Assembly algorithm for linear Lagrangian finite elements) We make a first attempt to outline an algorithm for building the $\mathcal{S}_1^0(\mathcal{M})$ -Galerkin matrix and

- assume:
- ◆ a numbering of nodal basis functions \leftrightarrow numbering of mesh vertices $\in \mathcal{V}(\mathcal{M})$
 - ◆ a numbering of triangles (cells) of mesh $\mathcal{M} = \{K_1, \dots, K_M\}$, $M := \#\mathcal{M}$
 - ◆ a (local) numbering of the three vertices of every triangle $K \in \mathcal{M}$

The adding up of entries of element matrices illustrated in Fig. 77 and Fig. 78 might suggest the following implementation (pseudo-code) of the “*collect approach*” visualized in Fig. 77 and Fig. 78.

Pseudocode 2.4.5.16: Vertex-centered assembly of Galerkin matrix for linear finite elements

```

foreach  $e \in \mathcal{E}(\mathcal{M})$       ( $\triangleleft$  notation:  $\mathcal{E}(\mathcal{M}) \doteq$  set of edges of  $\mathcal{M}$ )
   $(i, j) \doteq$  vertex numbers of endpoints of  $e$ 
   $(\mathbf{A})_{i,j} \leftarrow 0$ ,  $(\mathbf{A})_{j,i} \leftarrow 0$ ,
  foreach triangle  $K$  adjacent to  $e$ 
    find local numbers  $l, m \in \{1, 2, 3\}$  of endpoints of  $e$ 
     $(\mathbf{A})_{i,j} \leftarrow (\mathbf{A})_{i,j} + (\mathbf{A}_K)_{l,m}$        $\rightarrow$  Fig. 77,  $\mathbf{A}_K$  from (2.4.5.8)
     $(\mathbf{A})_{j,i} \leftarrow (\mathbf{A})_{j,i} + (\mathbf{A}_K)_{m,l}$        $\rightarrow$  Fig. 77,  $\mathbf{A}_K$  from (2.4.5.8)
  endfor
endfor
foreach  $v \in \mathcal{V}(\mathcal{M})$ 
   $j \doteq$  number of vertex  $v$ 
   $(\mathbf{A})_{j,j} \leftarrow 0$ 
  foreach triangle  $K$  adjacent to  $v$ 
     $l \doteq$  local number of  $v$  in  $K$ 
     $(\mathbf{A})_{j,j} \leftarrow (\mathbf{A})_{j,j} + (\mathbf{A}_K)_{l,l}$        $\rightarrow$  Fig. 78,  $\mathbf{A}_K$  from (2.4.5.8)
  endfor
endfor

```

§2.4.5.17 (Cell-oriented assembly) The algorithm implemented in Code 2.4.5.16 will strain the capabilities of the simple data structures available in a mesh object of type **TriaMesh2D**, because it requires information about the edges of the mesh. There is a dual way of organizing assembly, which needs only the basic topology and geometry information stored in **TriaMesh2D**, see Code 2.4.1.2.

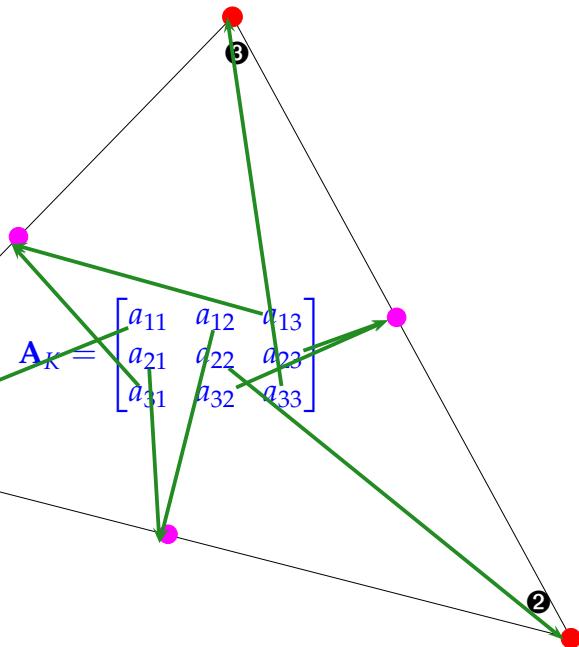
In short, the idea of cell-oriented assembly is

- ◆ to loop over all cells $K \in \mathcal{M}$
- ◆ and to “distribute” all entries of the element matrices \mathbf{A}_K to the corresponding entries of the Galerkin matrix.

This distribute step is illustrated in the figure

- \leftrightarrow edges, to which we can formally associate off-diagonal entries of the Galerkin matrix.
- \leftrightarrow vertices, carrying diagonal entries of the Galerkin matrix, local numbering given.
- $\rightarrow \hat{\triangleq}$ “contributes to”

Fig. 79



Cell oriented assembly

A simpler implementation can be achieved by adopting the perspective of **cell-oriented assembly** (“distribute scheme”): instead of traversing edges and vertices as in the above algorithm and collecting entries of element matrices of adjacent triangles, we loop over all triangles and *distribute* entries of their element matrices to their vertices and edges.

This is how the “distribute scheme” can be implemented:

Pseudocode 2.4.5.19: Cell-oriented assembly of Galerkin matrix for linear finite elements

```

SparseMatrix  $\mathbf{A} \in \mathbb{R}^{N,N}$ ,  $N := \#\mathcal{V}(\mathcal{M})$  (No of vertices)
 $\mathbf{A} := \mathbf{O};$ 
 $M := \#\mathcal{M};$  (no of cells)
for  $i = 1$  to  $M$  do
     $K \leftarrow \text{mesh.getVtCoords}(i);$  (Obtain cell-shape information)
     $\mathbf{A}_K \leftarrow \text{getElementMatrix}(K);$  (Compute element matrix)
    for  $k = 1$  to 3 do
        for  $j = 1$  to 3 do
             $\mathbf{A}(m : x^m = a_K^k, \ell : x^\ell = a_K^j) += \mathbf{A}_K(k,j);$ 
        endfor
    endfor
endfor endfor

```

We see that global vertex indices for the vertices of triangles have to be accessed. The next § will discuss how to organize this.

§2.4.5.20 (Index mapping for linear finite elements on triangular mesh) Invariably, cell oriented assembly entails knowing the global number of the basis functions associated with the vertices of each triangle. This information must be provided in an easily accessible form:

Data structure: $\text{dofh} \in \mathbb{N}^{\#\mathcal{M}, 3}$: local \rightarrow global *index mapping array*: “d.o.f. mapper”

$$\begin{aligned} \text{dofh}(k, l) &= \text{global number of vertex } l \text{ of } k\text{-th cell } \in \{1, \dots, N\} \\ x_{\text{dofh}(k, l)} &= a^l \text{ when } a^1, a^2, a^3 \text{ are the vertices of } K_k, \end{aligned} \quad (2.4.5.21)$$

for $l \in \{1, 2, 3\}$, $k \in \{1, \dots, M\}$, $M := \#\mathcal{M}$, $N := \#\mathcal{V}(\mathcal{M})$ (“mathematical indexing”!).

We assume that the triangulation is encoded in the data members `_nodecoords` $\in \mathbb{R}^{N, 2}$ and `_elements` $\in \mathbb{N}^{M, 3}$ of an object `Mesh` of type `TriaMesh2D` as explained in § 2.4.1.1.

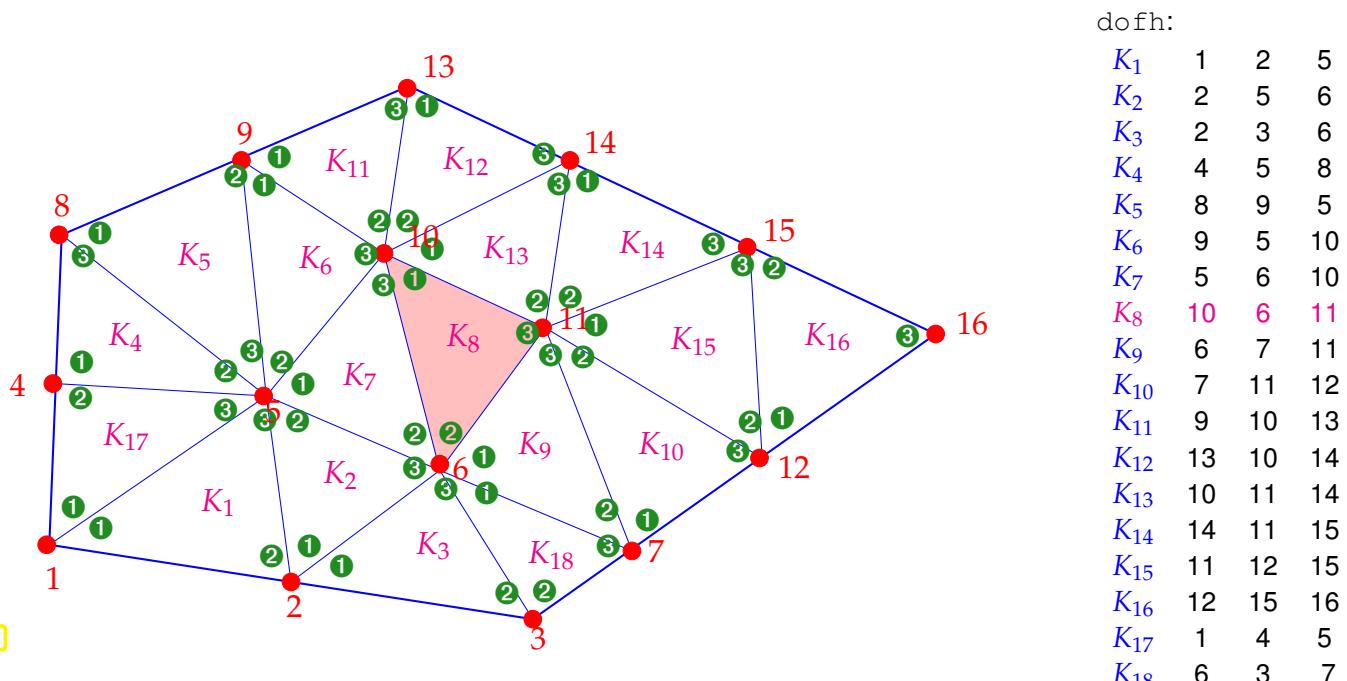
➤ simple realization of index mapping:

$$\text{dofh}(k, l) := \text{Mesh._elements}(k-1, l-1)$$

mathematical indexing C++ indexing

The use of index mapping in the context of assembly of a finite element Galerkin matrix will be discussed in more generality and detail in Section 2.7.4. □

EXAMPLE 2.4.5.22 (Index mapping by d.o.f. mapper) Fig. 80 displays a small planar triangulation, complete with all *local and global* index numbers of the vertices and the index numbers of the triangles. On the right the corresponding `dofh`-array complying with (2.4.5.21) is displayed (“mathematical indexing”).



In Fig. 80, for cell K_8 : element matrix \mathbf{A}_K contributes to $\mathbf{A}([10 \ 6 \ 11], [10 \ 6 \ 11])$ □

The algorithmic details of cell-oriented assembly are remarkably simple and illustrated by the following pseudocode. It takes for granted information about a triangular mesh to be available in an object named `mesh` of a type similar to `TriaMesh2D`, see § 2.4.1.1 and, in particular, Code 2.4.1.2.

Pseudocode 2.4.5.23: Assembly of finite element Galerkin matrix for linear finite elements

SparseMatrix $\mathbf{A} \in \mathbb{R}^{N, N}$, $N := \#\mathcal{V}(\mathcal{M})$ (number of vertices)

```

 $\mathbf{A} := \mathbf{O};$ 
 $\text{for } i = 1 \text{ to } N \text{ do}$ 
     $K \leftarrow \text{mesh.getVtCoords}(i)$ 
     $\mathbf{A}_K \leftarrow \text{getElementMatrix}(K);$ 

```

```

for k = 1 to 3 do
    for j = 1 to 3 do
        A(dofh(i,k),dofh(i,j)) += AK(k,j);
    endfor
endfor endfor

```

Note that homogeneous Dirichlet boundary conditions are not taken into account in Code 2.4.5.23, it builds the Galerkin matrix for the full finite element space $S_1^0(\mathcal{M})$, as needed, for instance, for the Neumann boundary value problem (2.1.2.2).

Code 2.4.5.23 demonstrates a fundamental paradigm in the implementation of finite element Galerkin schemes for variational problems connected with partial differential equations: loops generally run over the mesh cells and, if possible, computations are carried out on the level of the mesh cells, which usually, results in *optimal* (*) computational effort. For Code 2.4.5.23 this means the following.

$$\text{Computational effort} = O(\#\mathcal{M})$$

(*): computational cost for assembly that is linearly proportional to the number of nonzero entries of the Galerkin matrix is considered optimal.

A concrete C++ implementation of Code 2.4.5.23 is given next. The function argument `Mesh` refers to an object of `TriaMesh2D` describing the triangulation in the form of the `_nodecoords` and `_elements` matrices according to § 2.4.1.1. The parameter `getElementMatrix` must contain a function that expects a 2×3 -matrix of vertex coordinates and returns a 3×3 element matrix:

```

typedef function<Eigen::Matrix3d(const TriGeo_t &)>
LocalMatrixHandle_t;

```

An example is the function `ElementMatrix_Lapl_LFE` from Code 2.4.5.13. The function `assembleGalMatLFE()` returns a sparse $N \times N$ -matrix in CRS format, where $N = \#\mathcal{V}(\mathcal{M})$ is the number of vertices of the mesh.

C++ code 2.4.5.24: Cell-oriented assembly of Galerkin matrix for linear finite elements on a triangular mesh → [GITLAB](#)

```

1  Eigen::SparseMatrix<double>
2  assembleGalMatLFE(const TriaMesh2D& Mesh,
3  const LocalMatrixHandle_t getElementMatrix) {
4      // Fetch the number of vertices
5      int N = Mesh._nodecoords.rows();
6      // Fetch the number of elements/cells, see § 2.4.1.1
7      int M = Mesh._elements.rows();
8      // Create empty sparse Galerkin matrix A
9      Eigen::SparseMatrix<double> A(N,N);
10     // Loop over elements and "distribute" local contributions
11     for (int i = 0; i < M; i++) {
12         // Get local→global index mapping for current element, cf. (2.4.5.21)
13         Eigen::Vector3i dofhk = Mesh._elements.row(i);
14         TriGeo_t Vertices;
15         // Extract vertices of current element, see § 2.4.1.1
16         for (int j = 0; j < 3; j++)
17             Vertices.col(j) = Mesh._nodecoords.row(dofhk(j)).transpose();
18         // Compute 3×3 element matrix AK
19         Eigen::Matrix3d Ak = getElementMatrix(Vertices);

```

```

20 // Add local contribution to Galerkin matrix
21 for (int j = 0; j < 3; j++)
22     for (int k = 0; k < 3; k++)
23         A.coeffRef(dofhk(j), dofhk(k)) += Ak(j, k);
24 }
25 // Convert into CRS format, see [Hip19, ??].
26 A.makeCompressed();
27 return A;
28 }
```



Regard Code 2.4.5.24 as “C++ pseudo-code”: in actual implementation `A` must be initialized differently (→ Rem. 2.4.5.25), because random Lvalue access to entries of a sparse matrix in CRS format in Line 23 might be inefficient.

Remark 2.4.5.25 (Efficient assembly of sparse Galerkin matrices) Entry-by-entry initialization of a sparse matrix as in Code 2.4.5.23 involves huge hidden effort for moving data in memory, because sparse matrices are usually stored in CRS/CCS format, which exploits knowledge about vanishing matrix entries. An more detailed presentation is given in [Hip19, ??] and [GCS92].

More efficient initialization can be achieved by using an intermediate triplet/coordinate list (COO) format, see [Hip19, ??]. first store the $N \times N$ matrix as a vector of triplets (i, j, a_{ij}) , $i, j \in \{1, \dots, N\}$, which allows adding entries with little effort, and finally compute the more economical CRS/CCS format. How to do it in EIGEN is explained in [Hip19, ??]. Triplet initialization is used the function `assembleGalMatLFE` from Code 2.4.5.24.

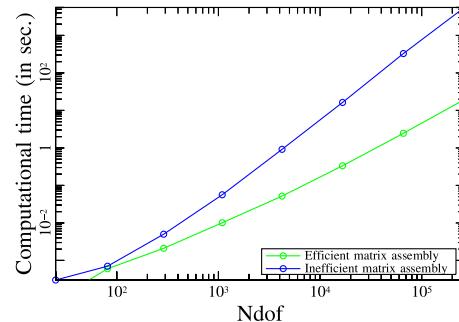
EXAMPLE 2.4.5.26 (Impact of efficient initialization of sparse Galerkin matrix) A code that blindly implements the entry-by-entry initialization of the Galerkin matrix according to Code 2.4.5.23 is correct, but will probably be rather slow due to massive moving of data in memory.

Comparison of runtimes of assembly of Galerkin matrices for $-\Delta$ (bilinear form as in (2.4.5.1)) on triangular meshes with different numbers of elements.

Computation of element matrices by direct vs. triplet initialization, timing by C++ `sys/time.h` routines, minimal time over 10 runs, Implementation → GITLAB, functions ([in](#))`efficientGalerkinAssembly()`

(OS: Linux Fedora 22, CPU: AMD Opteron 6174, Compiler: c++, optimization flag -O3.)

Fig. 81



We observe that for large matrices the triplet based initialization is significantly faster.

2.4.6 Computation of Right-Hand Side Vector

§2.4.6.1 (Model right hand side linear form) We consider the linear form $(,)$, which may represent the right-hand side of a linear variational problem, see (1.4.2.4), (2.1.2.2):

$$\ell(v) := \int_{\Omega} f(x) v(x) dx, \quad v \in H^1(\Omega), \quad f \in L^2(\Omega).$$

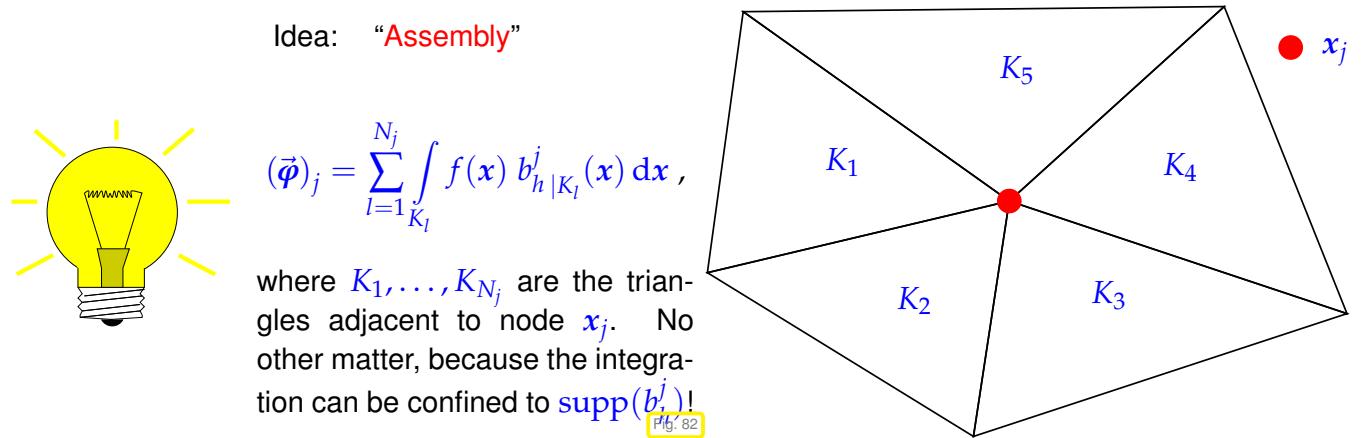
Recall the formula for the entries of the right hand side vector: With $N = \dim V_{N,0}$, $\mathfrak{B} = \{b_h^1, \dots, b_h^N\}$

the tent function basis of $\mathcal{S}_1^0(\mathcal{M})$ according to (2.4.3.5),

$$(\vec{\varphi})_j = \ell(b_h^j) = \int_{\Omega} f(x) b_h^j(x) dx, \quad j = 1, \dots, N. \quad (2.4.6.2)$$

□

Our considerations run parallel to those in Section 2.4.5: we split off right hand side linear form into **cell contributions**, cf. (2.4.5.1), page 171, for similar approach to the bilinear form a .



► Zero in on single triangle $K \in \mathcal{M}$:

$$\ell_K(b_h^j) := \int_K f(x) b_h^j|_K(x) dx = \ell_K(\lambda_i), \quad x_j \text{ vertex of } K, \quad (2.4.6.3)$$

where λ_i is the barycentric coordinate function associated with (local) vertex i of the triangle and $j = \text{dofh}(k, i)$, with k the (global) number of the triangle K and dofh defined in (2.4.5.21) on page 178. Recall that in this case $b_h^j|_K = \lambda_i$.

As above in Fig. 78: Entries of the right hand side vector can be obtained by summing up the values that the localized right hand side functionals ℓ_K return for barycentric coordinate functions:

This can be expressed through the **vertex-oriented** formula (“collect scheme”)

$$(\vec{\varphi})_j = \sum_{K,i:\text{dofh}(k,i)=j} \ell_K(\lambda_i). \quad (2.4.6.4)$$

Here: $k \leftrightarrow$ global index of triangle K

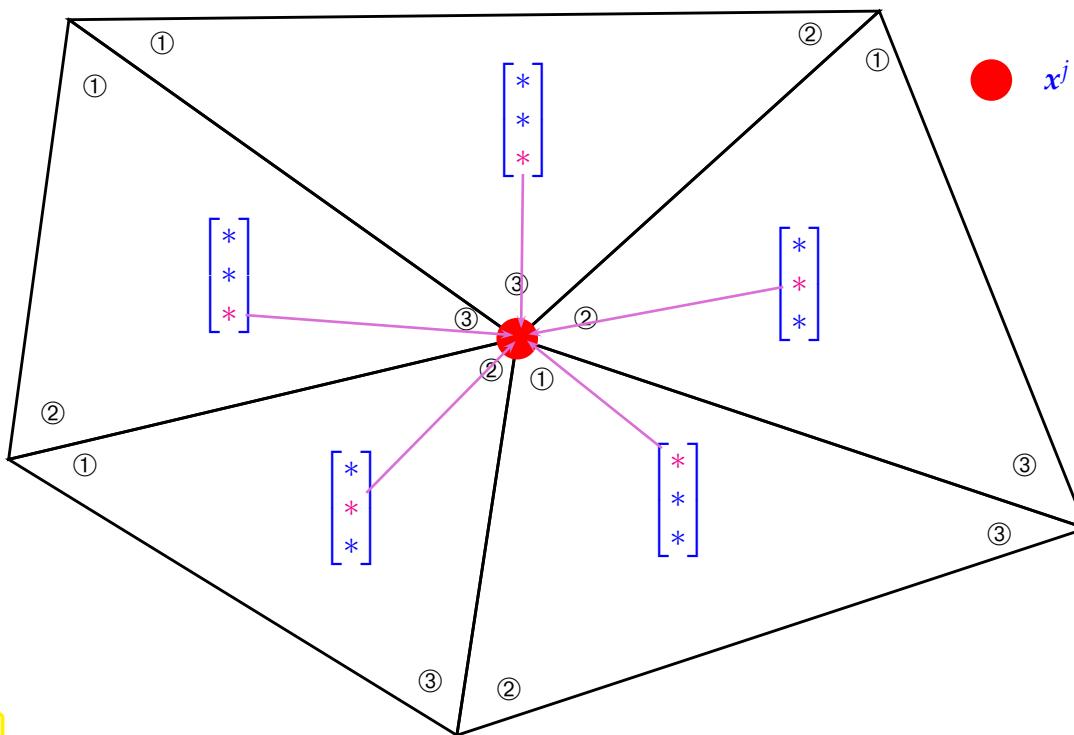


Fig. 83

However, implementation according to this formula would emulate the cumbersome algorithm on page 176 for the computation of the Galerkin matrix.



Idea: As in Section 2.4.5, § 2.4.5.15, and Code 2.4.5.23 we aim to compute $\vec{\phi}$ in a *cell-oriented* fashion (“distribute scheme”).

To that end we need a counterpart of the element (stiffness) matrix from (2.4.5.8), the

$$\text{element (load) vector : } \vec{\phi}_K := [\ell_K(\lambda_i)]_{i=1}^3 \in \mathbb{R}^3, \quad (2.4.6.5)$$

which is obtained by plugging the restrictions of basis functions to an element into that part of the right hand side linear form belonging to the element.

- Cell-oriented “assembly” of $(\vec{\phi})_j$ by summing up contributions from element vectors of triangles adjacent to x_j ($N_j \triangleq$ no. of triangles abutting x_j)

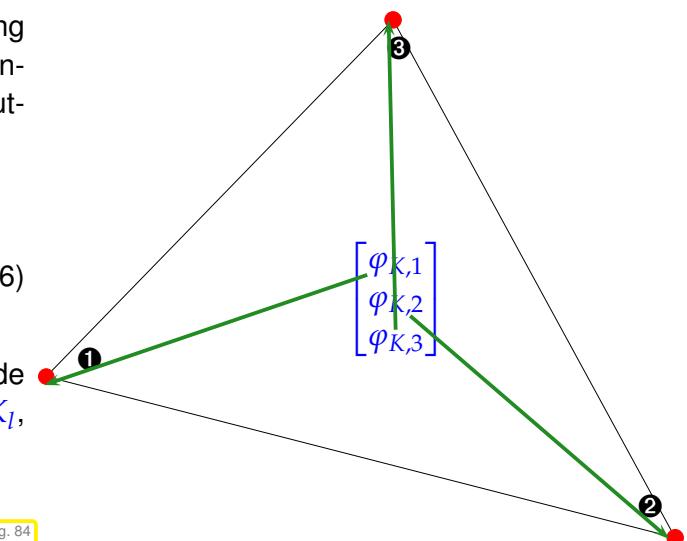
$$(\vec{\phi})_j = \sum_{l=1}^{N_j} \ell_{K_l}(b_h|_{K_l}) = \sum_{l=1}^{N_j} (\vec{\phi}_K)_{i(l,j)}, \quad (2.4.6.6)$$

where $i(l,j)$ is the *local* vertex index of the node x_j (*global* index j , $j = 1, \dots, N$) in the triangle K_l , $l = 1, \dots, \#M$.

Note: with index array `dofh` from § 2.4.5.15:

$$\text{dofh}(l, i(l,j)) = j$$

Fig. 84



Remark 2.4.6.7 (Assembly of right hand side vector for linear finite elements → § 2.4.5.15) Entries of element load vectors from triangles sharing a vertex are summed up, see Fig. 83 for illustration and Code 2.4.6.8 for the implementation of this *cell-oriented* assembly.

The argument `Mesh` passes a reference to an object of type `TriaMesh2D`, see Code 2.4.1.2 for the class definition, the argument `getElementVector` is a functor whose evaluation operator

- (i) takes the geometry of a triangle in the form of a 3×2 coordinate matrix and returns the element load vector as defined in (2.4.6.6),
- (ii) accepts a handle to a function $\mathbb{R}^2 \rightarrow \mathbb{R}$, which provides the source function f .

C++ code 2.4.6.8: Cell-oriented assembly of right hand side vector for linear finite elements, see (2.4.6.6) → GITLAB

```

1  typedef function<double(const Eigen::Vector2d&)> FHandle_t;
2  typedef function<Eigen::Vector3d(const TriGeo_t &, FHandle_t)> LocalVectorHandle_t;
3
4  Eigen::VectorXd assemLoad_LFE(const TriaMesh2D &Mesh,
5  const LocalVectorHandle_t &getElementVector,
6  const FHandle_t &FHandle)
7  {
8      // Obtain the number of vertices and cells (elements)
9      int N = Mesh._nodecoords.rows();
10     int M = Mesh._elements.rows();
11     // Initialize right hand side vector with zero.
12     Eigen::VectorXd phi = Eigen::VectorXd::Zero(N);
13
14     // Loop over elements and "distribute" local contributions
15     for (int i = 0; i < M; i++) {
16         // get local→global index mapping for current element,
17         // cf. (2.4.5.21)
18         Eigen::Vector3i dofhk = Mesh._elements.row(i);
19         TriGeo_t Vertices;
20         // Extract geometry of current element, see § 2.4.1.1
21         for (int j = 0; j < 3; j++)
22             Vertices.col(j) = Mesh._nodecoords.row(dofhk(j)).transpose();
23         //compute element right hand side vector
24         Eigen::Vector3d philoc = getElementVector(Vertices, FHandle);
25         //add contributions to global load vector
26         for (int j = 0; j < 3; j++)
27             phi(dofhk(j)) += philoc(j);
28     }
29     return phi;
30 }
```

Same as Code 2.4.5.23, also Code 2.4.6.8 employs only a loop over all cells of the mesh (*cell oriented assembly*), again resulting in *optimal computational effort* $\mathcal{O}(\#\mathcal{M})$. □

§2.4.6.9 (Numerical quadrature for assembly of right hand side vector) Recall Rem. 2.1.2.5: $f : \Omega \mapsto \mathbb{R}$ is usually given in *procedural form*

```
typedef function<double(const Eigen::Vector2d &)> FHandle_t;
```

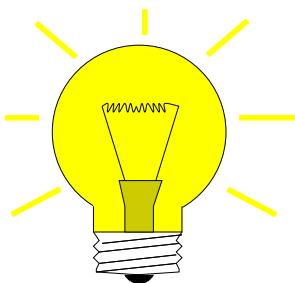
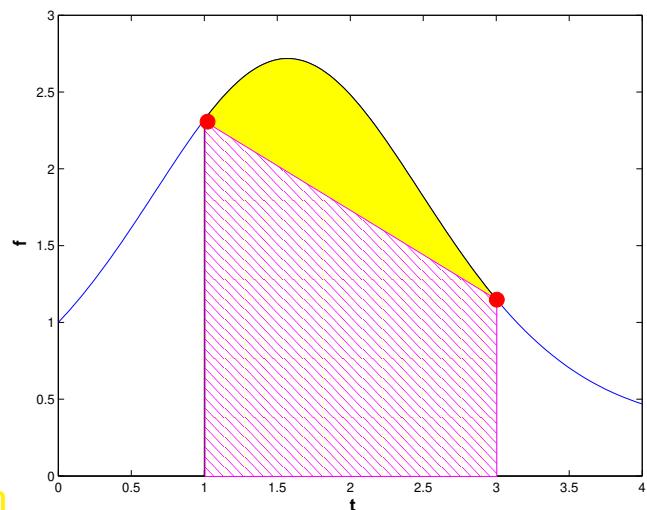
► Mandatory: use of *numerical quadrature* for approximate evaluation of $\ell_K(b_h^j)$, cf. (2.3.3.8).

In the 1D setting of Section 2.3 we used composite quadrature rules based on low order Gauss/Newton-Cotes quadrature formulas on the cells $[x_{j-1}, x_j]$ of the grid, e.g. the composite trapezoidal rule (2.3.3.8).

What is the 2D counterpart of the composite trapezoidal rule ?

Recall:

trapezoidal rule [Hip19, ??] integrates linear interpolant of integrand based on endpoint values



Idea:

2D trapezoidal rule

for triangle K with vertices $\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3$

$$\int_K f(x) dx \approx \frac{|K|}{3} (f(\mathbf{a}^1) + f(\mathbf{a}^2) + f(\mathbf{a}^3)). \quad (2.4.6.10)$$

$\hat{=}$ integration of linear interpolant $\sum_{i=1}^3 f(\mathbf{a}^i) \lambda_i$ of f .

► element (load) vector: $\vec{\varphi}_K := \left[\ell_K(b_h^{j(i)}|_K) \right]_{i=1}^3 = [\ell_K(\lambda_i)]_{i=1}^3 \approx \frac{|K|}{3} \begin{bmatrix} f(\mathbf{a}^1) \\ f(\mathbf{a}^2) \\ f(\mathbf{a}^3) \end{bmatrix}, \quad (2.4.6.11)$

where $\mathbf{x}_{j(i)} = \mathbf{a}^i, i = 1, 2, 3$ (global node number \leftrightarrow local vertex number).

The following code relies on (2.4.6.11) to compute the element load vector for an arbitrary triangle, whose vertex coordinates are passed as rows of a 3×2 -matrix, cf. Code 2.4.5.13. The source function f is made available through a functor object.

C++ code 2.4.6.12: (Approximate) computation of element load vector by means of 2D trapezoidal local quadrature rule (2.4.6.11) \Rightarrow GITLAB

```

1 // Functor type for right hand side source function
2 typedef function<double(const Eigen::Vector2d &)> FHandle_t;
3
4 Eigen::Vector3d localLoadLFE(const t_TriGeo& V, const FHandle_t& FHandle)
5 {
6     // Compute area of triangle, cf. Code 2.4.5.13
7     double area =
8         0.5*((V(1,0)-V(0,0))*(V(2,1)-V(1,1))-(V(2,0)-V(1,0))*(V(1,1)-V(0,1)));
9     // Evaluate source function for vertex location
10    Eigen::Vector3d philoc = Eigen::Vector3d::Zero();
11    // Implements (2.4.6.11)
12    for (int i = 0; i < 3; i++) philoc(i) = FHandle(V.row(i));
13    // Scale with  $\frac{1}{3} \cdot \text{area}$  of triangle
14    philoc *= area/3.0;
15    return philoc;
}
```

Review question(s) 2.4.6.13 (Linear finite elements in 2D)

(Q2.4.6.13.A) [“Closing” a mesh with hanging nodes]

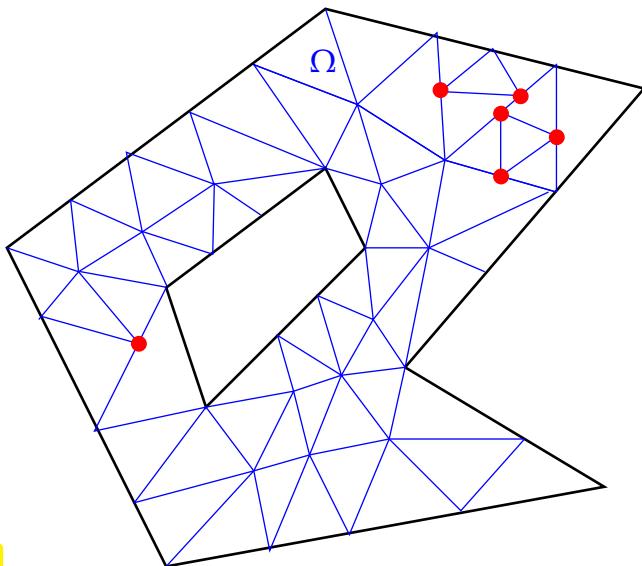


Fig. 86

The mesh $\tilde{\mathcal{M}}$ displayed beside contains a few **hanging nodes** marked with \bullet .

Sketch a triangulation (without hanging nodes) with the same nodes as that mesh $\tilde{\mathcal{M}}$.

(Q2.4.6.13.B) [“Tensor-product” triangular mesh] Chop up a square $\Omega \subset \mathbb{R}^2$ into n^2 congruent small squares and create a triangular mesh \mathcal{M} of Ω by splitting each small square along parallel diagonals. What is $\dim \mathcal{S}_1^0(\mathcal{M})$ and $\dim \mathcal{S}_{1,0}^0(\mathcal{M})$ in terms of n ?

(Q2.4.6.13.C) [Imposing Dirichlet boundary conditions] Let \mathcal{M} be a triangulation of a polygonal domain. Describe a modification of the space $\mathcal{S}_1^0(\mathcal{M})$ of \mathcal{M} -piecewise linear continuous functions that yields a subspace $\mathcal{S}_{1,0}^0(\mathcal{M}) \subset \mathcal{S}_1^0(\mathcal{M})$ of **maximal dimension** such that $\mathcal{S}_{1,0}^0(\mathcal{M}) \subset H_0^1(\Omega)$.

(Q2.4.6.13.D) [Linear independence of barycentric coordinate functions] Prove that the barycentric coordinate functions for any non-degenerate planar triangle $K \subset \mathbb{R}^2$ are linearly independent.

(Q2.4.6.13.E) [Non-zero entries of Galerkin matrix] For the domain and mesh from Question (Q2.4.6.13.B) determine the maximal number of non-zero entries of the Galerkin matrix obtained when discretizing (2.1.2.2) with trial and test space $\mathcal{S}_{1,0}^0(\mathcal{M})$ (sharp bound).

(Q2.4.6.13.F) [Computing gradients of barycentric coordinate functions] Explain the mathematics underlying the following code.

C++ code 2.4.5.11: Computation of gradients of barycentric coordinate functions on a triangle → [GITHUB](#)

```

2 Eigen::Matrix<double, 2, 3> gradbarycoordinates(const TriGeo_t& vertices) {
3     Eigen::Matrix<double, 3, 3> X;
4     // Argument vertices passes the vertex positions of the triangle
5     // as the columns of a  $2 \times 3$ -matrix, , see
6     // Code 2.4.1.3. The function returns the components of the
7     // gradients as the columns of a  $2 \times 3$ -matrix
8
9     // Computation based on (2.4.5.10), solving for the
10    // coefficients of the barycentric coordinate functions.
11    X.block<3, 1>(0, 0) = Eigen::Vector3d::Ones();
12    X.block<3, 2>(0, 1) = vertices.transpose();
13    return X.inverse().block<2, 3>(1, 0);
14 }
```

(Q2.4.6.13.G) [Computing gradients of barycentric coordinate functions] Justify, why the following code computes an element matrix for $-\Delta$ on a triangle.

C++ code 2.4.5.13: Computation of element matrix for $-\Delta$ on a triangle and for linear Lagrangian finite elements → GITHUB

```

2 Eigen::Matrix3d ElementMatrix_Lapl_LFE(const TriGeo_t& V) {
3     // Argument V same as vertices in Code 2.4.5.11.
4     // The function returns the  $3 \times 3$  element matrix as a fixed size
5     // EIGEN matrix.
6
7     // Evaluate (2.4.5.1), exploiting that the gradients are constant.
8     // First compute the area of triangle by determinant formula
9     double area = 0.5 * std::abs((V(0, 1) - V(0, 0)) * (V(1, 2) - V(1, 1)) -
10                                (V(0, 2) - V(0, 1)) * (V(1, 1) - V(1, 0)));
11    // Compute gradients of barycentric coordinate functions, see
12    // Code 2.4.5.11
13    Eigen::Matrix<double, 2, 3> X = gradbarycoordinates(V);
14    // compute inner products of gradients through matrix multiplication
15    return area * X.transpose() * X;
16 }
```

(Q2.4.6.13.H) [Taking in account zero Dirichlet boundary conditions] We are provided with an `TriaMesh2D` object describing a planar triangulation \mathcal{M} of a polygon Ω with N nodes and `std::vector<bool> bdflags`, where `bdflags[k] == true`, if the node with number k is located on $\partial\Omega$. Outline how one has to modify Code 2.4.5.24 (this you may look up in the lecture document) so that it assembles a Galerkin matrix w.r.t. the trial/test space $\mathcal{S}_{1,0}^0(\mathcal{M})$.

C++ code 2.4.1.2: Class handling planar triangular mesh → GITLAB

```

2 // Matrix containing vertex coordinates of a triangle
3 using TriGeo_t = Eigen::Matrix<double, 2, 3>;
4 struct TriaMesh2D {
5     // Constructor: reads mesh data from file, whose name is passed
6     TriaMesh2D(std::string filename); //
7     virtual ~TriaMesh2D(void) {}
8     // Retrieve coordinates of vertices of a triangles as rows
9     // of a fixed-size  $3 \times 2$  matrix
10    TriGeo_t getVtCoords(std::size_t) const;
11    // Data members describing geometry and topology
12    Eigen::Matrix<double, Eigen::Dynamic, 2> _nodecoords;
13    Eigen::Matrix<int, Eigen::Dynamic, 3> _elements;
14 }
```

(Q2.4.6.13.I) [Element matrices] Write \mathbf{A}_K for the element matrix for linear finite elements, the bilinear form $a(u, v) := \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx$, and a triangle K . We use numerical quadrature based on the 2D trapezoidal rule to compute the element matrix \mathbf{B}_K for the bilinear form $\tilde{b}(u, v) := \int_{\Omega} \sigma(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx$, $\sigma \in C^0(\bar{\Omega})$. How can \mathbf{B}_K be computed from \mathbf{A}_K ?

(Q2.4.6.13.J) [Galerkin matrix]

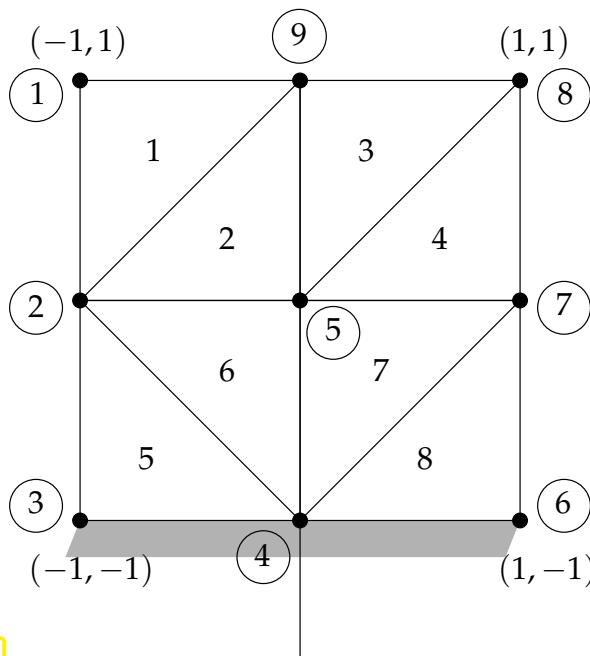


Fig. 87

Compute the Galerkin matrix for the bilinear form

$$a(u, v) = \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx ,$$

$u, v \in H^1(\Omega)$, $\Omega =]-1, 1[^2$, and the trial/test space $S_1^0(\mathcal{M})$, \mathcal{M} shown in Fig. 87, using the tent function bases numbered according to the numbering of the nodes of the mesh as indicated in Fig. 87.

Hint. All triangles of the mesh are congruent. You may also use the formula

$$\mathbf{A}_K = \frac{1}{2} \begin{bmatrix} \cot \omega_3 + \cot \omega_2 & -\cot \omega_3 & -\cot \omega_2 \\ -\cot \omega_3 & \cot \omega_3 + \cot \omega_1 & -\cot \omega_1 \\ -\cot \omega_2 & -\cot \omega_1 & \cot \omega_2 + \cot \omega_1 \end{bmatrix} , \quad (2.4.5.8)$$

giving the $S_1^0(\mathcal{M})$ -element matrix for $-\Delta$ on a triangle with angles ω_i , $i = 1, 2, 3$.

(Q2.4.6.13.K) [Galerkin matrix for n -gon] The cells of a triangular mesh \mathcal{M} are generated by connecting all n corners of a regular polygon with diameter 2 with its center. On this mesh we consider the finite element space $S_1^0(\mathcal{M})$ and the corresponding Galerkin matrix for the bilinear form

$$a(u, v) = \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx \quad u, v \in H^1(\Omega) ,$$

where Ω is the interior of the polygon. We assume that the standard tent function basis of $S_1^0(\mathcal{M})$ is used and that their numbering starts with the central node and then continues counterclockwise through the corners of the polygon.

- (i) Sketch the pattern formed by the non-zero entries of the finite-element Galerkin matrix $\mathbf{A} \in \mathbb{R}^{n+1, n+1}$.
- (ii) Compute the non-zero entries of \mathbf{A} .

Hint. The formula

$$\mathbf{A}_K = \frac{1}{2} \begin{bmatrix} \cot \omega_3 + \cot \omega_2 & -\cot \omega_3 & -\cot \omega_2 \\ -\cot \omega_3 & \cot \omega_3 + \cot \omega_1 & -\cot \omega_1 \\ -\cot \omega_2 & -\cot \omega_1 & \cot \omega_2 + \cot \omega_1 \end{bmatrix} , \quad (2.4.5.8)$$

can be used.

(Q2.4.6.13.L) [Gradients of barycentric coordinate functions] Let $K \subset \mathbb{R}^2$ be a triangle and λ_i , $i = 1, 2, 3$, the associated **barycentric coordinate functions**. Explain, why $\mathbf{grad} \lambda_i$ is orthogonal to the direction of the edge opposite the vertex i .

△

2.5 Building Blocks of General Finite Element Methods



Video tutorial for Section 2.5: Building Blocks of General Finite Element Methods: (39 minutes) [Download link](#), [tablet notes](#)

§2.5.0.1 (Overview) The previous section explored the details of a simple finite element discretization of 2nd-order elliptic variational problems. Yet, it already introduced *key features and components* that distinguish the finite element approach to the discretization of linear boundary value problems for partial differential equations:

- ◆ **variational formulations** of a boundary value problem as starting point → Section 1.8,
- ◆ a partitioning of the computational domain Ω by means of a **mesh** \mathcal{M} (→ Section 2.4.1)
- ◆ the use of Galerkin trial and test spaces based on **piecewise polynomials** w.r.t. \mathcal{M} (→ Section 2.4.2),
- ◆ the use of **locally supported** basis functions for the assembly of the resulting linear system of equations (→ Section 2.4.3).

In this section a more abstract point of view is adopted than in Section 2.4.2, and the components of a finite element method for scalar 2nd-order elliptic boundary value problems will be discussed in greater generality. However, prior perusal of Section 2.4 is strongly recommended. □

2.5.1 Meshes

First main ingredient of FEM: **triangulation/mesh** of Ω , generalizes the concepts of Section 2.4.1.

Definition 2.5.1.1. Finite element mesh/triangulation

A **mesh** (or **triangulation**) of $\Omega \subset \mathbb{R}^d$ is a finite collection $\mathcal{M} := \{K_i\}_{i=1}^M$, $M \in \mathbb{N}$, of *open* non-degenerate (curvilinear) polygons ($d = 2$)/polyhedra ($d = 3$) such that

- (A) $\overline{\Omega} = \bigcup \{\overline{K}_i, i = 1, \dots, M\}$,
- (B) $K_i \cap K_j = \emptyset \Leftrightarrow i \neq j$,
- (C) for all $i, j \in \{1, \dots, M\}$, $i \neq j$, the intersection $\overline{K}_i \cap \overline{K}_j$ is either empty or a vertex, edge, or face of both K_i and K_j .

Requirements (A) & (B) ensure that the cells K_i of the mesh form a partition of Ω (up to a set of measure zero)

Requirement (C) rules out “hanging nodes”, cf. condition (iv) on the triangulation introduced in Section 2.4.1, page 161. Fig. 58 depicts the “hanging node” situation.

§2.5.1.2 (Finite element meshes: customary terminology) To talk about finite element algorithms concisely, we need quite a few terms and notions:

- **Entities** refer to the geometric entities “vertex”, “edge”, “face” of polygon/polyhedron: meaning of these terms corresponds to geometric intuition.

Entities can be classified by their **dimension** or **co-dimension**, which add up to the **world dimension/physical dimension d** :

mesh entity	dimension	codimension
2D, $d = 2$:		
triangles	2	0
edges	1	1
vertices	0	2
3D, $d = 3$:		
tetrahedra	3	0
faces	2	1
edges	1	2
vertices	0	3

- Given a mesh $\mathcal{M} := \{K_i\}_{i=1}^M$ the K_i called **cells** or **elements** $\hat{=}$ entities of co-dimension 0
- Vertices of a mesh are often called **nodes** $\hat{=}$ entities of co-dimension d
(☞ notation for set of nodes: $\mathcal{V}(\mathcal{M})$)
- Given a mesh and one of its mesh entities K , the **sub-entities** of K are all those mesh entities contained in the closure \bar{K} of K , including K itself. For example, a triangle has three edges and three nodes as sub-entities.

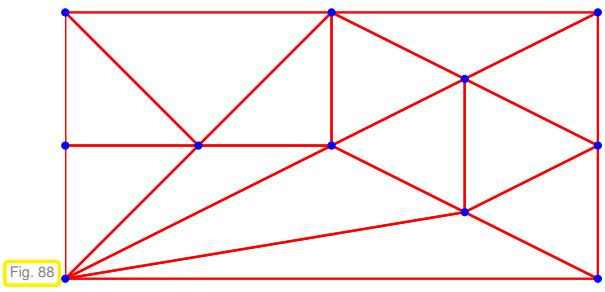
The **relative co-dimension** of a sub-entity K' of K is the difference of the dimensions of K and K' .

Relative co-dimensions for entities and different sub-entities of a 2D mesh. \triangleright

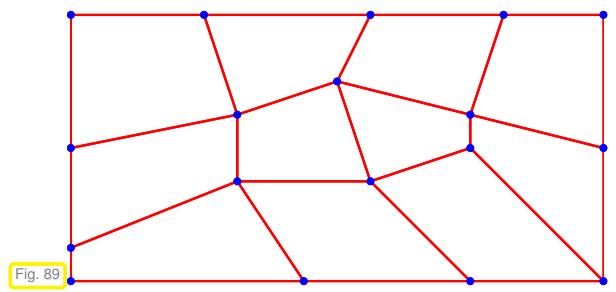
entity	sub-entity		
	cell	edge	node
cell	0	1	2
edge	–	0	1
node	–	–	0

In finite element codes the use of co-dimension is popular, because loops over cells are a core part of most algorithms, recall § 2.4.5.17, and cells will invariably be distinguished by **co-dimension = 0**, independently of the ambient dimension. \downarrow

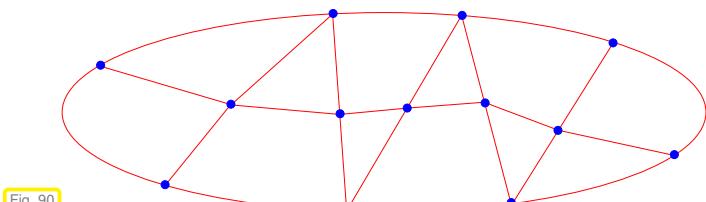
§2.5.1.3 (Types of meshes) Meshes according to Def. 2.5.1.1 can be classified further:



Triangular mesh in 2D



Quadrilateral mesh in 2D



(Curved) tetrahedral meshes in 3D (created with **NETGEN**):

- ◁ 2D **hybrid** mesh comprising
- triangles
 - quadrilaterals
 - curvilinear cells (at $\partial\Omega$)

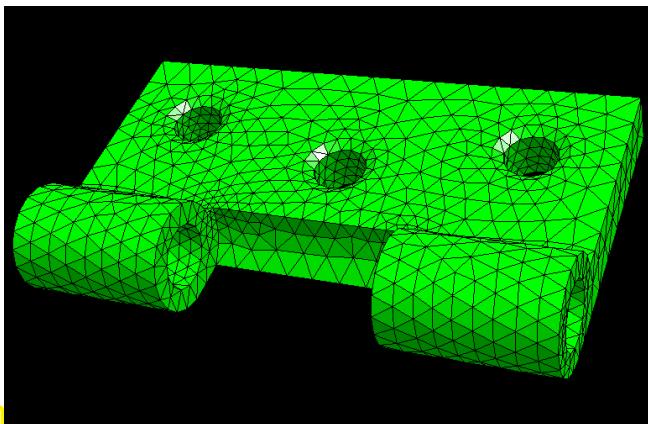


Fig. 91

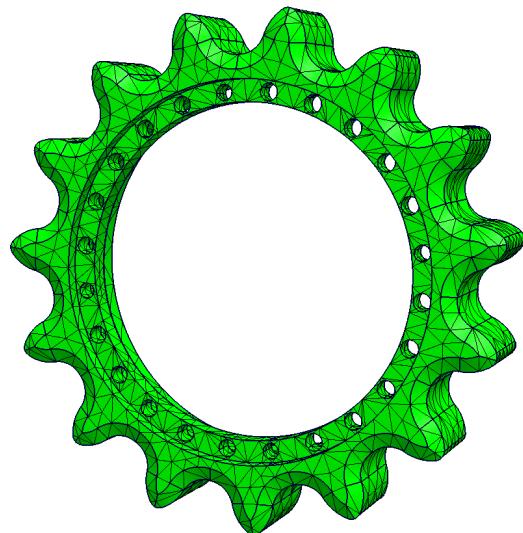


Fig. 92

Another special type are tensor product meshes, also called **grids**

in 2D: $a = x_0 < x_1 < \dots < x_n = b$,
 $c = y_0 < y_1 < \dots < y_m = d$.

► $\mathcal{M} = \{]x_{i-1}, x_i[\times]y_{j-1}, y_j[: \quad (2.5.1.4)$
 $1 \leq i \leq n, 1 \leq j \leq m\}$.

☞ Restricted to tensor product domains

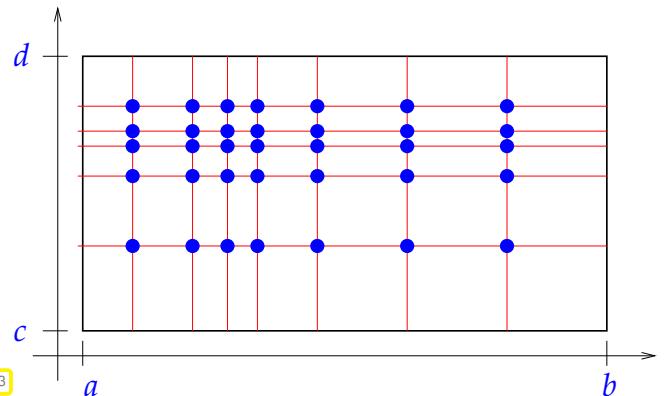


Fig. 93

Terminology:

Simplicial mesh \triangleq

triangular mesh in 2D
tetrahedral mesh in 3D

Remark 2.5.1.5 (Finite element meshes with hanging nodes)

If (C) does not hold

► Triangular **non-conforming** mesh
(with **hanging nodes**)

$K_i \cap K_j$ is only part of an edge/face for at most one of the adjacent cells.

(However, this mesh is conforming, if degenerate quadrilaterals are admitted.)

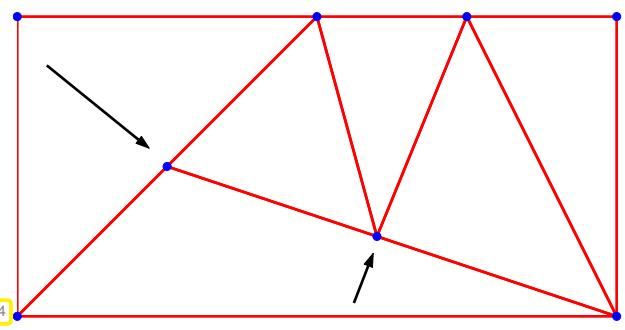


Fig. 94

2.5.2 Polynomials

The gist of FEMs is approximation by piecewise polynomials.

In FEM: Galerkin trial/test spaces comprise \mathcal{M} -locally polynomial functions on Ω

Polynomials are attractive, because

- they allow fast and easy evaluation [Hip19, ??] and straightforward analytic differentiation and integration,
- (smooth) functions can be approximated efficiently by means of polynomials [Hip19, ??].

The reader will certainly know polynomials of $\text{degree} \leq p$, $p \in \mathbb{N}_0$, in 1D (**uni-variate** polynomials)

$$\mathcal{P}_p(\mathbb{R}) := \{x \mapsto c_0 + c_1 x + c_2 x^2 + \dots + c_p x^p\}.$$

In higher dimensions this concept allows various generalizations, one given in the following definition, one given in Def. 2.5.2.7.

§2.5.2.1 (Multivariate Polynomials) This is a first way to generalize uni-variate polynomials so that they are recovered for $d = 1$:

Definition 2.5.2.2. Multivariate polynomials

Space of **multivariate (d -variate) polynomials** of (total) $\text{degree } p \in \mathbb{N}_0$:

$$\mathcal{P}_p(\mathbb{R}^d) := \{x \in \mathbb{R}^d \mapsto \sum_{\alpha \in \mathbb{N}_0^d, |\alpha| \leq p} c_\alpha x^\alpha, c_\alpha \in \mathbb{R}\}.$$

Def. 2.5.2.2 relies on **multi-index notation**:

$$\alpha = (\alpha_1, \dots, \alpha_d): \quad x^\alpha := x_1^{\alpha_1} \cdots x_d^{\alpha_d}, \quad (2.5.2.3)$$

$$|\alpha| = \alpha_1 + \alpha_2 + \cdots + \alpha_d. \quad (2.5.2.4)$$

Special case:

$$d = 2: \quad \mathcal{P}_p(\mathbb{R}^2) = \left\{ \sum_{\substack{\alpha_1, \alpha_2 \geq 0 \\ \alpha_1 + \alpha_2 \leq p}} c_{\alpha_1, \alpha_2} x_1^{\alpha_1} x_2^{\alpha_2}, c_{\alpha_1, \alpha_2} \in \mathbb{R} \right\}.$$

Examples: $\mathcal{P}_2(\mathbb{R}^2) = \text{Span}\{1, x_1, x_2, x_1^2, x_2^2, x_1 x_2\}$,
 $\mathcal{P}_1(\mathbb{R}^2)$ = affine linear functions $\mathbb{R}^2 \mapsto \mathbb{R}$, see Section 2.4.2

Lemma 2.5.2.5. Dimension of spaces of polynomials

$$\dim \mathcal{P}_p(\mathbb{R}^d) = \binom{d+p}{p} \quad \text{for all } p \in \mathbb{N}_0, d \in \mathbb{N}$$

Proof. Distribute p “powers” to the d independent variables or discard them $\triangleright d+1$ bins.

Combinatorial model: number of different linear arrangements of p identical items and d separators $= \binom{d+p}{p}$. □

Leading order for $p \rightarrow \infty$:

$$\dim \mathcal{P}_p(\mathbb{R}^d) = O(p^d)$$

§2.5.2.6 (Tensor product polynomials) This is another way to extend uni-variate polynomials to higher dimensions.

Definition 2.5.2.7. Tensor product polynomials

Space of **tensor product polynomials** of degree $p \in \mathbb{N}$ in each coordinate direction

$$\begin{aligned}\mathcal{Q}_p(\mathbb{R}^d) &:= \left\{ x \mapsto \sum_{\ell_1=0}^p \cdots \sum_{\ell_d=0}^p c_{\ell_1, \dots, \ell_d} x_1^{\ell_1} \cdots x_d^{\ell_d}, c_{\ell_1, \dots, \ell_d} \in \mathbb{R} \right\} \\ &= \text{Span}\{x \mapsto p_1(x_1) \cdots p_d(x_d), p_i \in \mathcal{P}_p(\mathbb{R}), i = 1, \dots, d\}.\end{aligned}$$

Example:

$$\mathcal{Q}_2(\mathbb{R}^2) = \text{Span}\{1, x_1, x_2, x_1 x_2, x_1^2, x_1^2 x_2, x_1^2 x_2^2, x_1 x_2^2, x_2^2\}$$

Lemma 2.5.2.8. Dimension of spaces of tensor product polynomials

$$\dim \mathcal{Q}_p(\mathbb{R}^d) = (p+1)^d \quad \text{for all } p \in \mathbb{N}_0, d \in \mathbb{N}$$

Terminology:

$\mathcal{P}_p(\mathbb{R}^d)/\mathcal{Q}_p(\mathbb{R}^d)$ = complete spaces of polynomials/tensor product polynomials

2.5.3 Basis Functions

Third main ingredient of FEM:

locally supported basis functions

(see Section 2.2 for role of bases in Galerkin discretization)

Basis functions b_h^1, \dots, b_h^N for a finite element trial/test space $V_{0,h}$ built on a mesh \mathcal{M} **must** satisfy:

- (B₁) $\mathcal{B}_h := \{b_h^1, \dots, b_h^N\}$ is basis of $V_{0,h} \geq N = \dim V_{0,h}$,
- (B₂) each b_h^i is **associated** with a single geometric entity (cell/edge/face/vertex) of \mathcal{M} ,
- (B₃) $\text{supp}(b_h^i) = \bigcup \{\bar{K} : K \in \mathcal{M}, p \subset \bar{K}\}$, if b_h^i associated with cell/edge/face/vertex p .

Some finite element terminology connected with the basis functions:

- The basis functions b_h^i are also called **global shape functions** (GSF)/**global basis functions/degrees of freedom** (DOFs)
- We say the a mesh entity K is **covered** by a basis function b_h^i , if $K \subset$ interior of $\text{supp}(b_h^i)$. (Note the distinction: a basis function is associated with exactly one entity, but can cover many.)

Often finite element spaces are directly defined by specifying a set of basis functions:

Mesh \mathcal{M} + global shape functions \rightarrow complete description of finite element space

The specification of the global shape functions is considered an integral part of the description of a finite element method. However, remember from Thm. 2.2.2.6 and Section 2.2 that it is the sheer finite element space, that is, the span of the global shape functions, that determines the Galerkin solution u_h .

EXAMPLE 2.5.3.1 (Supports of global shape functions in 1D \rightarrow Section 2.3) We recall the construction of the finite element space $\mathcal{S}_{1,0}^0(\mathcal{M})$ in 1D and the choice of basis functions made in Section 2.3.2. Now we try to give a concrete meaning to (B₃) in this case.

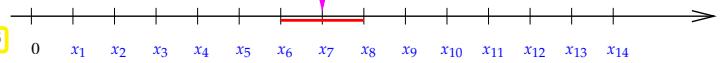
- ◆ $\Omega =]a, b[\hat{=} \text{interval}$
- ◆ Equidistant mesh

Support (\rightarrow Def. 2.3.2.5) of global shape function (tent function) associated with x_7

$$\mathcal{M} := \{]x_{j-1}, x_j[, j = 1, \dots, M\},$$

$$x_j := a + hj, h := (b - a)/M, M \in \mathbb{N}.$$

Fig. 95



Obviously the support of each tent functions is the interval comprised by two adjacent cells. □

EXAMPLE 2.5.3.2 (Supports of global shape functions on triangular mesh) We discuss (\mathfrak{B}_3) for a 2D triangular mesh. For the 2D tent function found in Section 2.4.3 it is clear that their supports cover all the triangles surrounding the vertex to which the basis function is associated. The general case addressed in (\mathfrak{B}_3) is best illustrated by pictures. The symbol \bullet marks the mesh entity to which a basis function is associated.

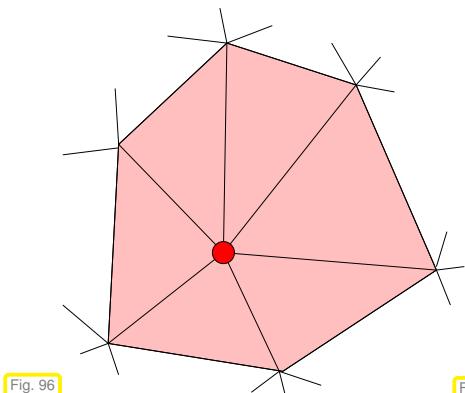


Fig. 96

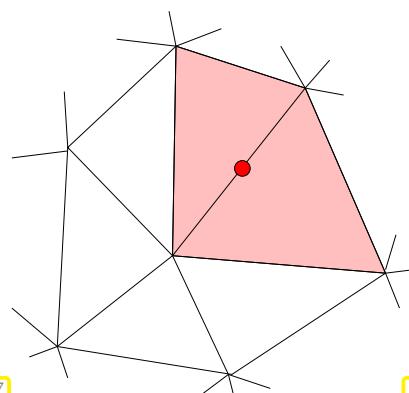


Fig. 97

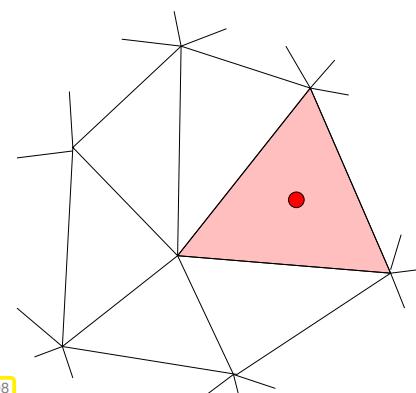


Fig. 98

Support of node-associated basis function, cf. Fig. 99 Support of edge-associated basis function Support of cell-associated basis function

The higher the dimension of the entity, the smaller the support. □

§2.5.3.3 (Importance of local supports) The requirement (\mathfrak{B}_3) implies that

global finite element basis functions are **locally** supported.

What is the rationale for this requirement ?

Consider a generic bilinear form a arising from a linear scalar 2nd-order elliptic BVP, see (2.4.4.1): it involves integration over $\Omega/\partial\Omega$ of products of (derivatives of) basis functions. Thus the integrand for $a(b_h^j, b_h^i)$ vanishes outside the overlap of the supports of b_h^j and b_h^i .

► Galerkin matrix $\mathbf{A} \in \mathbb{R}^{N,N}$ with $(\mathbf{A})_{ij} := a(b_h^j, b_h^i), i, j = 1, \dots, N$ satisfies

$(\mathbf{A})_{ij} \neq 0$ only if b_h^i and b_h^j associated with vertices/faces/edges(cells) adjacent to common cell



Finite element Galerkin matrices are **sparse** (\rightarrow Notion 2.4.4.3)

In turns, sparsity of the coefficient matrix is crucial for

- the ability to store the Galerkin matrix with $O(N)$ memory requirements for large N , where N is the dimension of the finite element space,
- for the fast direct or iterative solution of the linear system of equations arising from finite element Galerkin discretization.

Now we introduce an important notion that will be key to understanding the efficient assembly of finite element Galerkin matrices. Recall that “assembly” means the initialization of finite element Galerkin matrix from element contributions, cf. § 2.4.5.15.

Definition 2.5.3.4. Local shape functions (LSF)

Given a finite element function space on a mesh \mathcal{M} with global shape functions b_h^i , $i = 1, \dots, N$, for every mesh entity K we define

$$\{b_K^j\}_{j=1}^{Q(K)} := \{b_h^j|_K, K \subset \text{interior of } \text{supp}(b_h^j)\} := \text{set of local shape functions (LSF)},$$

that is the local shape functions are the basis functions that *cover* K , restricted to K .

$$\text{Global shape functions} \xrightarrow{\text{Restriction to entity}} \text{local shape functions} \quad (2.5.3.5)$$

Note that some authors use the term “local shape functions” only for entities of co-dimension 0, that is, for cells.

Also note a consequence of property \mathfrak{B}_2 of global shape functions:

$\mathfrak{B}_2 \Rightarrow$ Also local shape functions b_K^1, \dots, b_K^Q , $Q = Q(K) \in \mathbb{N}$, are associated with a unique **sub-entity** (a vertex/edge/face/the interior) of K .

EXAMPLE 2.5.3.6 (Cell-local shape functions for $\mathcal{S}_1^0(\mathcal{M})$ in 2D → Section 2.4.3) We assume a triangular mesh and view the barycentric coordinate functions through the lens of Def. 2.5.3.4: (2.4.5.3) tells us that they are local shape functions for $\mathcal{S}_1^0(\mathcal{M})$ on triangles.

Global basis function for $\mathcal{S}_1^0(\mathcal{M})$

▷

Example: On the “unit triangle” \hat{K} with vertices

$$\mathbf{a}^1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathbf{a}^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{a}^3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}:$$

we find the local shape functions

$$b_{\hat{K}}^1(\mathbf{x}) := 1 - x_1 - x_2,$$

$$b_{\hat{K}}^2(\mathbf{x}) := x_1,$$

$$b_{\hat{K}}^3(\mathbf{x}) := x_2.$$

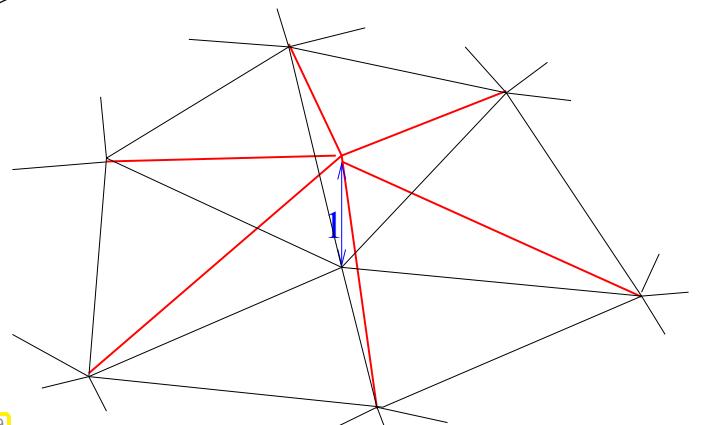


Fig. 99

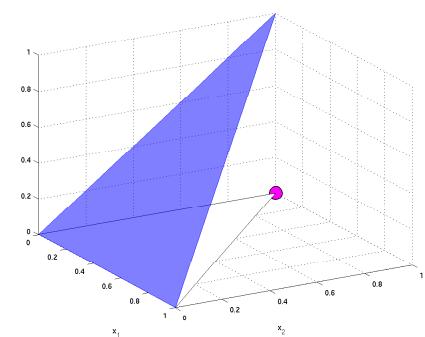
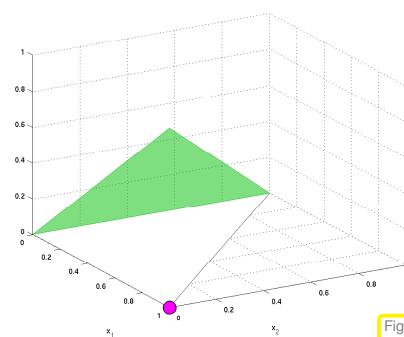
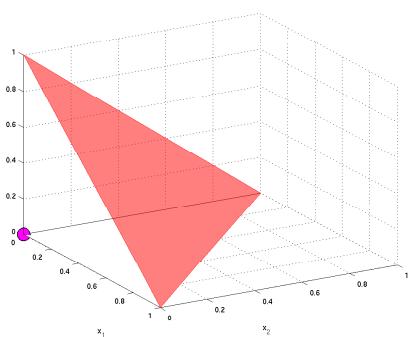


Fig. 100

These are the **barycentric coordinate functions** $\lambda_1, \lambda_2, \lambda_3$ on \widehat{K} as introduced in Section 2.4.5. □

EXAMPLE 2.5.3.7 (LSFs for $S_1^0(\mathcal{M})$ on edges) Restrictions of tent functions to edges emanating from the vertex where they sit are straightforward:

Fig. 101

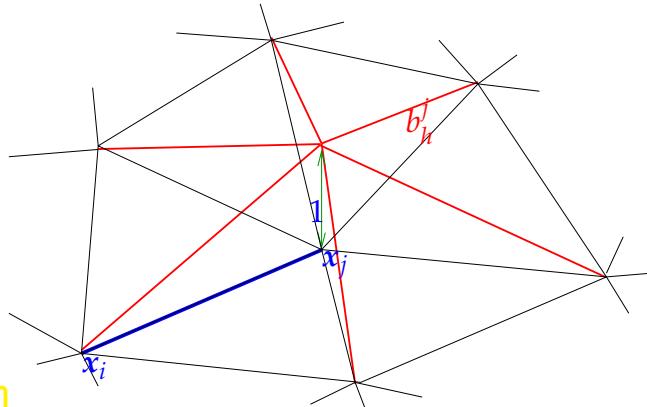
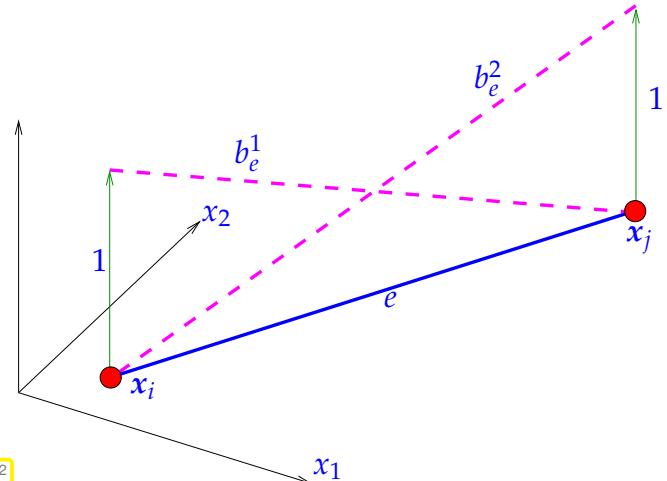


Fig. 102



There are two local shape functions on an edge e , linear along e , attaining values $0, 1$ at its endpoints. If e connects the nodes x_i (first endpoint) and x_j (second endpoint), then

$$x = \xi x_i + (1 - \xi) x_j \in e \Rightarrow b_e^1(x) = \xi, \quad b_e^2(x) = 1 - \xi. \quad (2.5.3.8)$$
□

Review question(s) 2.5.3.9 (Building blocks of FEM)

(Q2.5.3.9.A) [Homogeneous polynomials]

Definition Def. 2.5.2.2. Multivariate homogeneous polynomials

The space of d -variate **homogeneous polynomials** of (total) **degree** $p \in \mathbb{N}_0$ is

$$\tilde{\mathcal{P}}_p(\mathbb{R}^d) := \{x \in \mathbb{R}^d \mapsto \sum_{\alpha \in \mathbb{N}_0^d, |\alpha|=p} c_\alpha x^\alpha, c_\alpha \in \mathbb{R}\}.$$

What is $\dim \tilde{\mathcal{P}}_p(\mathbb{R}^d)$? Show by induction that

$$\sum_{j=0}^p \dim \tilde{\mathcal{P}}_j(\mathbb{R}^d) = \dim \mathcal{P}_p(\mathbb{R}^d) \quad \forall p \in \mathbb{N}_0.$$

(Q2.5.3.9.B) [A basis for $\mathcal{P}_1(\mathbb{R}^2)$] Show that

$$\{x \mapsto x_1, x \mapsto x_2, x \mapsto 1 - x_1 - x_2\}, \quad x := \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2,$$

is a **basis** of $\mathcal{P}_1(\mathbb{R}^2)$.

(Q2.5.3.9.C) [Quadratic polynomials vanishing on a line] What is the dimension of the space of polynomials

$$V := \left\{ p \in \mathcal{P}_2(\mathbb{R}^2) : p\left(\begin{bmatrix} \xi \\ 0 \end{bmatrix}\right) = 0 \quad \forall \xi \in \mathbb{R} \right\}?$$

Give a basis for V .

(Q2.5.3.9.D) [Rational for piecewise polynomial functions] What are the advantages of piecewise polynomial trial and test functions in the context of the Galerkin discretization of 2nd-order elliptic variational problems on 2D triangular meshes.

(Q2.5.3.9.E) [Rationale for locally supported basis functions] Explain why finite element methods employ basis functions with *local supports* for the Galerkin discretization of partial differential equations in weak form.

(Q2.5.3.9.F) [From hanging nodes to hybrid mesh] What 2D triangular meshes with hanging nodes can be regarded as valid hybrid meshes comprising triangular and quadrilateral cells?

(Q2.5.3.9.G) [Data structure for quadrilateral mesh] Devise a class **QuadMesh2D** for general planar quadrilateral meshes in analogy to the class **TriaMesh2D** listed in Code 2.4.1.2.

C++ code 2.4.1.2: Class handling a planar triangular mesh

```

2 // Matrix containing vertex coordinates of a triangle
3 using TriGeo_t = Eigen::Matrix<double, 2, 3>;
4 struct TriaMesh2D {
5     // Constructor: reads mesh data from file, whose name is passed
6     TriGeo_t getVtCoords(std::size_t) const;
7     // Data members describing geometry and topology
8     Eigen::Matrix<double, Eigen::Dynamic, 2> _nodecoords;
9     Eigen::Matrix<int, Eigen::Dynamic, 3> _elements;
10    ~TriaMesh2D(void) {}
11 };
12 
```

(Q2.5.3.9.H) [Converting a quadrilateral into a triangular mesh] How can a quadrilateral mesh be converted into a triangular mesh? Based on the data structures **QuadMesh2D** and **TriaMesh2D** considered in Question (Q2.5.3.9.G) outline an algorithm.

(Q2.5.3.9.I) [Local shape functions and global shape functions] Elaborate on the relationship between local shape functions (LSF) and global shape functions (GSF) in the finite-element method.

(Q2.5.3.9.J) [Supports of basis functions] Consider a finite-element space on a planar triangular mesh. How many nodes/edges/cells are contained in the **support** of a vertex/edge/cell-associated finite-element basis function

Hint. The support of a function is closed set:

Definition 2.3.2.5. Support of a function

The **support** of a function $f : \Omega \mapsto \mathbb{R}$ is defined as

$$\text{supp}(f) := \overline{\{x \in \Omega : f(x) \neq 0\}}.$$

(Q2.5.3.9.K) [Edge-associated basis functions] Consider the Galerkin discretization of

$$u \in H^1(\Omega): \int_{\Omega} \alpha(x) \mathbf{grad} u \cdot \mathbf{grad} v + c(x) u v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in H^1(\Omega), \quad (2.1.2.2)$$

on a planar triangular mesh \mathcal{M} using global shape functions associated with the edges of the mesh. Give a sharp bound for the number of non-zero entries of the Galerkin matrix in terms of the number of vertices $\#\mathcal{V}(\mathcal{M})$, number of edges $\#\mathcal{E}(\mathcal{M})$, and number of cells $\#\mathcal{M}$ of the mesh.

△

2.6 Lagrangian Finite Element Methods



Video tutorial for Section 2.6: Lagrangian Finite Element Methods: (43 minutes)
[Download link](#), [tablet notes](#)

In the previous section we adopted a bird's eye's view of finite element methods and established their essential characteristics. Unfortunately, we had only the simple case of $\mathcal{S}_1^0(\mathcal{M})$, piecewise linear finite element functions on simplicial meshes \mathcal{M} as a concrete realization. In this section we will see an important family of finite element spaces contained in $H^1(\Omega)$, the **Lagrangian finite element spaces** and they will all fit the mold of Section 2.5. Throughout this section we take for granted that the computational domain $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, is equipped with a finite element mesh \mathcal{M} according to Def. 2.5.1.1.

§2.6.0.1 (H^1 -conforming finite element spaces) Our goal is the construction of finite element spaces and global shape functions of higher polynomials degrees, generalizing the space $\mathcal{S}_1^0(\mathcal{M})$ introduced in Section 2.4.3.

Throughout, the Lagrangian finite element spaces introduced in this section provide spaces $V_{0,h}$ of \mathcal{M} -piecewise polynomials that fulfill

$$V_{N,0} \subset C^0(\bar{\Omega}) \xrightarrow{\text{Thm. 1.3.4.23}} V_{N,0} \subset H^1(\Omega) .$$

Theorem 1.3.4.23. Compatibility conditions for piecewise smooth functions in $H^1(\Omega)$

Let Ω be partitioned into sub-domains Ω_1 and Ω_2 . A function u that is continuously differentiable in both sub-domains and continuous up to their boundary, belongs to $H^1(\Omega)$, if and only if u is continuous on Ω .

Parlance: finite element spaces that are contained in $H^1(\Omega)$ are often called " **H^1 -conforming**".

Notation:
 (Lagrangian FE spaces)

$\mathcal{S}_p^0(\mathcal{M})$ continuous functions, cf. $C^0(\Omega)$
 locally polynomials of degree p , e.g. $\mathcal{P}_p(\mathbb{R}^d)$

2.6.1 Simplicial Lagrangian FEM

In this section let \mathcal{M} be a simplicial mesh of a polygonal/polyhedral domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, consisting of triangles in 2D, tetrahedra in 3D.

Now we generalize $\mathcal{S}_1^0(\mathcal{M})/\mathcal{S}_{1,0}^0(\mathcal{M})$ from Section 2.4 to higher polynomial degree $p \in \mathbb{N}_0$.

Definition 2.6.1.1. Simplicial Lagrangian finite element spaces

The space of **p -th degree Lagrangian finite element** functions on simplicial mesh \mathcal{M} is defined as

$$\mathcal{S}_p^0(\mathcal{M}) := \{v \in C^0(\bar{\Omega}): v|_K \in \mathcal{P}_p(K) \quad \forall K \in \mathcal{M}\} .$$

Def. 2.6.1.1 merely describes the space of trial/test functions used in a Lagrangian finite element method on a simplicial mesh. A crucial ingredient is still missing (→ Section 2.5.3): the global shape functions still

need to be specified. This is done by generalizing (2.4.3.2) based on sets of special *interpolation nodes*. We begin the explanations with the case $p = 2$

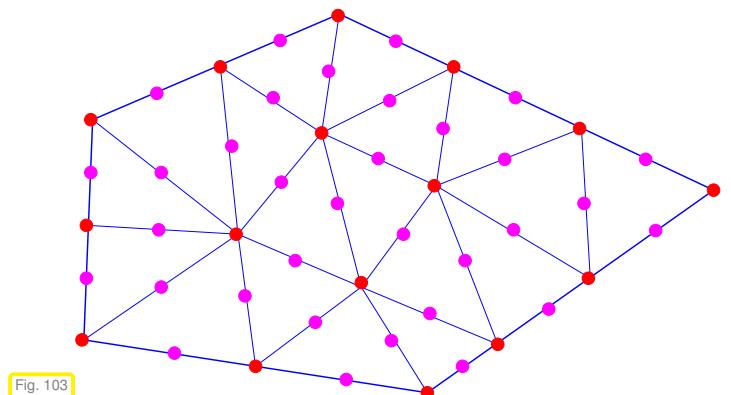
EXAMPLE 2.6.1.2 (Triangular quadratic ($p = 2$) Lagrangian finite elements) Let \mathcal{M} be a planar triangular mesh. Functions in the space $\mathcal{S}_2^0(\mathcal{M})$ coincide with quadratic multi-variate polynomials on every cell of the mesh \mathcal{M} :

$$v_h \in \mathcal{S}_2^0(\mathcal{M}) \Rightarrow v_h|_K \in \text{Span}\{x \mapsto 1, x_1, x_2, x_1^2, x_2^2, x_1 x_2\} \quad \forall K \in \mathcal{M}. \quad (2.6.1.3)$$

Def. 2.6.1.1 and (2.6.1.3) provide scant hints about how to choose suitable locally supported basis functions b_h^i , $i = 1, \dots, N$, nor is it clear what is $N := \dim \mathcal{S}_2^0(\mathcal{M})$. We propose a more useful device for fixing the finite element basis.

We rely on a set \mathcal{N} of *interpolation nodes*

$$\begin{aligned} \mathcal{N} &:= \mathcal{V}(\mathcal{M}) \cup \{\text{midpoints of edges}\}, \\ \mathcal{N} &= \{p_1, \dots, p_N\} \quad (\text{ordered}). \end{aligned}$$



Nodal basis functions b_h^j , $j = 1, \dots, N$, are then defined by the familiar *cardinal basis* property, cf. (2.4.3.2)

$$b_h^j(p_i) = \begin{cases} 1 & , \text{if } i = j, \\ 0 & \text{else,} \end{cases} \quad i, j \in \{1, \dots, N\}. \quad (2.6.1.4)$$

STEP I: A “definition” like (2.6.1.4) is cheap, but it may be pointless, in case no such functions b_h^j exist. To establish their existence, we first study the case of a *single triangle* K .

We have to show that there is a basis of $\mathcal{P}_2(\mathbb{R}^2)$ that satisfies (2.6.1.4) in the case of a mesh consisting of a single triangle $\mathcal{M} = \{K\}$.

The six interpolation nodes on a triangle K with vertices a^1 , a^2 , and a^3 are, see Fig. 104:

$$\begin{aligned} p_1 &= a^1 & p_2 &= a^2 & p_3 &= a^3, \\ p_4 &= \frac{1}{2}(a^1 + a^2) & p_5 &= \frac{1}{2}(a^2 + a^3) & p_6 &= \frac{1}{2}(a^1 + a^3). \end{aligned} \quad (2.6.1.5)$$

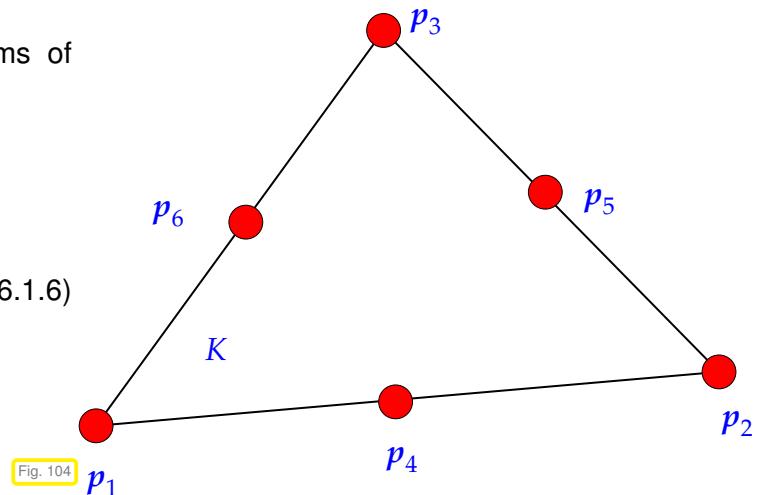
What is the rationale for this numbering? There is absolutely **none**, because the numbering of the local interpolation nodes can be chosen *arbitrarily*. Once it is decided, however, one has to adhere to this choice consistently throughout a finite element code.

A first simple *consistency check*: does the number of interpolation nodes $\#\mathcal{N}$ for $\mathcal{M} = \{K\}$ agree with $\dim \mathcal{P}_2(\mathbb{R}^2) = 6$? Yes, it does!

Next, we give a “Proof of existence by construction”, that is we give precise formulas for local shape functions.

We express the local shape functions in terms of barycentric coordinate functions:

$$\begin{aligned} b_K^1 &= (2\lambda_1 - 1)\lambda_1, \\ b_K^2 &= (2\lambda_2 - 1)\lambda_2, \\ b_K^3 &= (2\lambda_3 - 1)\lambda_3, \\ b_K^4 &= 4\lambda_1\lambda_2, \\ b_K^5 &= 4\lambda_2\lambda_3, \\ b_K^6 &= 4\lambda_1\lambda_3. \end{aligned} \quad (2.6.1.6)$$

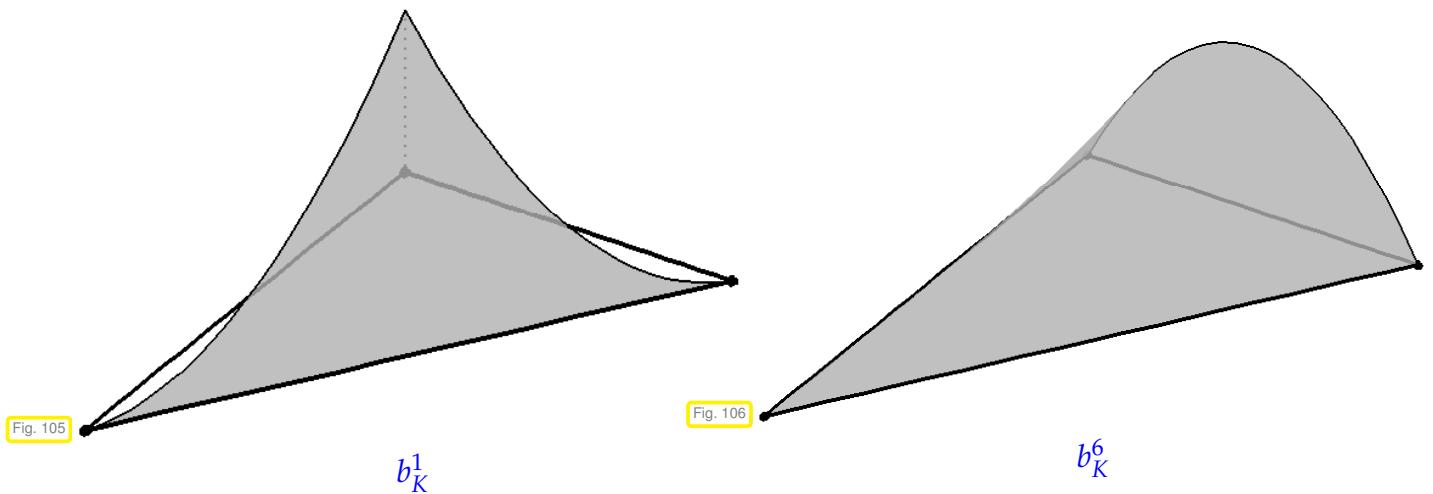


We remark that it is generally true for Lagrangian finite elements that local shape functions are linear combinations of (products of) barycentric coordinate functions, see (2.7.5.1) below.

To confirm the validity of the formulas (2.6.1.6), that is, the compliance with the local cardinal basis property (2.6.1.4), note that

- both $\lambda_i \in \mathcal{P}_1(\mathbb{R}^2)$ and $\lambda_i\lambda_j \in \mathcal{P}_2(\mathbb{R}^2)$ for all $1 \leq i, j \leq 3$,
- $\lambda_i(\mathbf{a}^i) = 1$ and $\lambda_i(\mathbf{a}^j) = 0$, if $i \neq j$, where $\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3$ are the vertices of the triangle K ,
- $\lambda_1(\mathbf{m}^{12}) = \lambda_1(\mathbf{m}^{13}) = \frac{1}{2}$, $\lambda_2(\mathbf{m}^{12}) = \lambda_2(\mathbf{m}^{23}) = \frac{1}{2}$, $\lambda_3(\mathbf{m}^{13}) = \lambda_3(\mathbf{m}^{23}) = \frac{1}{2}$, $\lambda_1(\mathbf{m}^{23}) = \lambda_2(\mathbf{m}^{13}) = \lambda_3(\mathbf{m}^{12}) = 0$, where $\mathbf{m}^{ij} = \frac{1}{2}(\mathbf{a}^i + \mathbf{a}^j)$ denotes the midpoint of the edge connecting \mathbf{a}^i and \mathbf{a}^j ,
- each barycentric coordinate function λ_i is affine linear such that $\lambda_i\lambda_j \in \mathcal{P}_2(\mathbb{R}^2)$.

Let us look at the graphs of selected local shape functions for $\mathcal{S}_2^0(\mathcal{M})$ over a triangle:

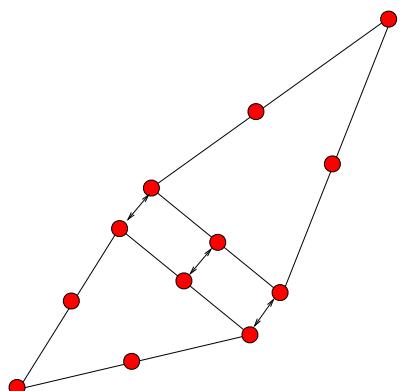


So far we have seen that *local shape functions* can be found that satisfy (2.6.1.4).

STEP II: We have to tackle the following issue:

Can the local shape functions from (2.6.1.6) be “stitched together” across interelement edges such that they yield a *continuous* global basis function?

(Remember that Thm. 1.3.4.23 demands global continuity in order to obtain a subspace of $H^1(\Omega)$.)



Considerations:

1. The restriction of a quadratic polynomial to an edge is an *univariate* quadratic polynomial.
2. Fixing its value in three points, the midpoint of the edge and the endpoints, *uniquely* fixes this polynomial.
3. The local shape functions associated with the same interpolation node “from left and right” agree on the edge.

➢ continuity !

Fig. 107

“Stitching” visualized:

Fig. 108

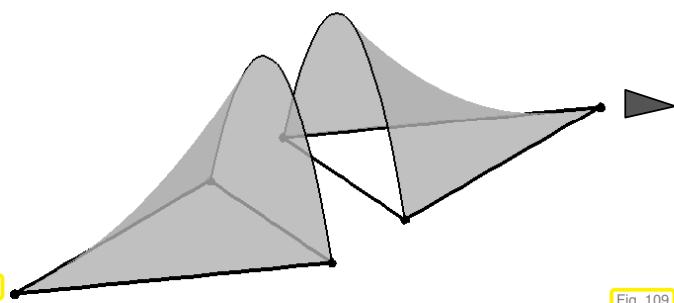


Fig. 109

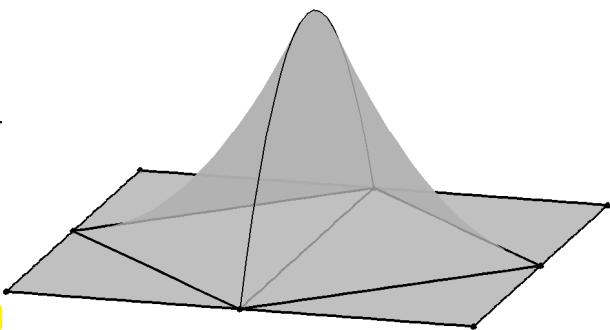
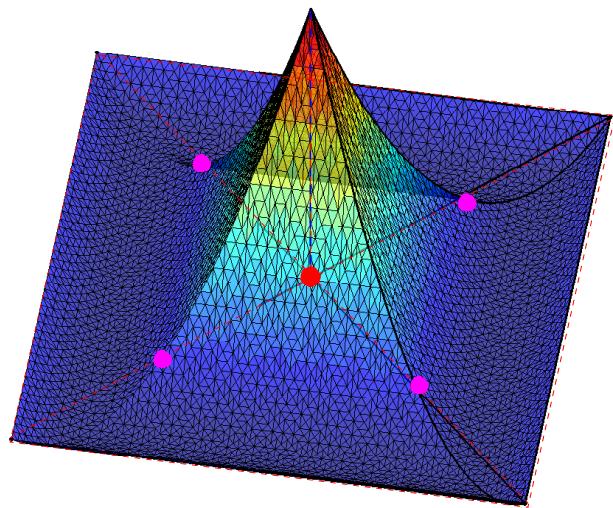


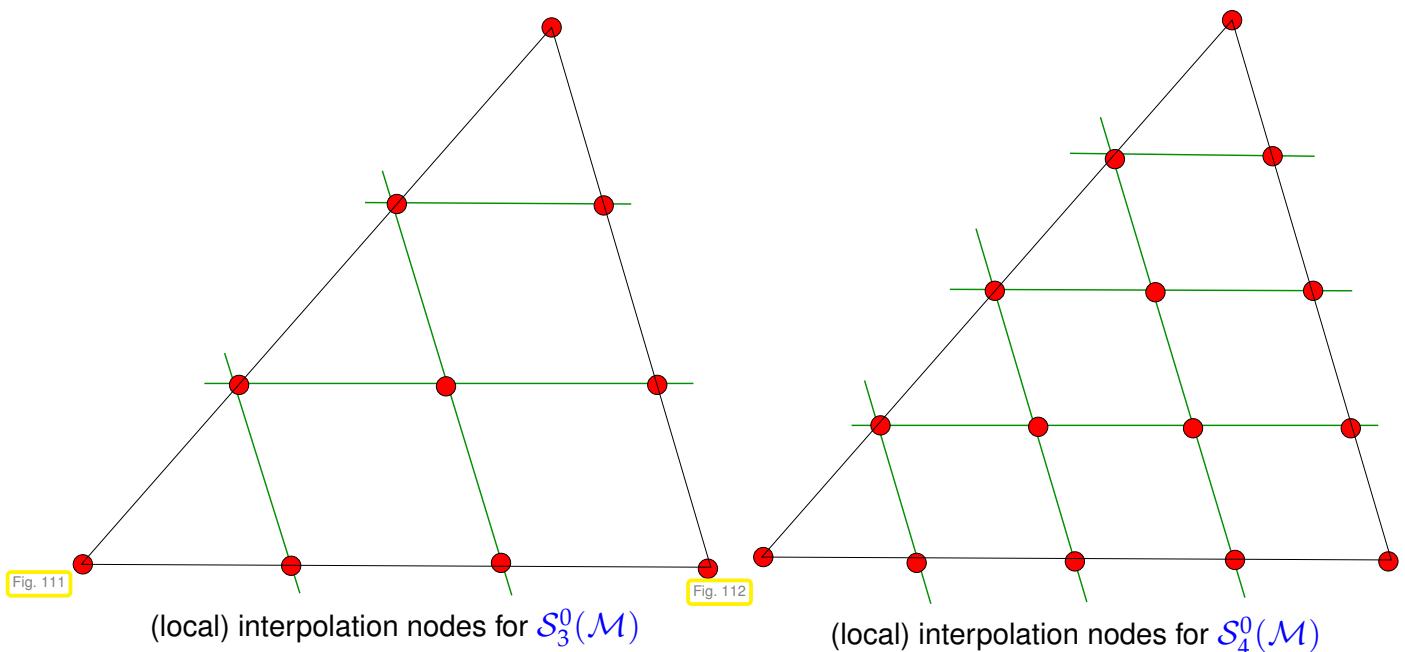
Fig. 110



▷ Global basis function for $\mathcal{S}_2^0(\mathcal{M})$ associated with a vertex

The cardinal basis property (2.6.1.4) is satisfied, because this function attains value = 1 at a vertex (●) and vanishes at the midpoints (●) of the edges of adjacent triangles, as well as at any other vertex.

EXAMPLE 2.6.1.7 (Local interpolation nodes for cubic ($p = 3$) and quartic ($p = 4$) Lagrangian FE in 2D) The construction for $p = 2$ can be generalized to any polynomial degree and justified with similar arguments:



Can you already guess a general pattern underlying the location of local interpolation nodes for degree p Lagrangian finite elements on triangles? They are the points whose barycentric coordinates satisfy $\lambda_i(\mathbf{p}_j) \in \{\frac{0}{p}, \frac{1}{p}, \dots, \frac{p-1}{p}, \frac{p}{p}\}$. \square

2.6.2 Tensor-Product Lagrangian FEM

Now we consider tensor product meshes (grids), see (2.5.1.4), Fig. 93, for a 2D example.

EXAMPLE 2.6.2.1 (Bilinear Lagrangian finite elements) We seek a generalization of 1D piecewise linear finite element functions from Section 2.3, see § 2.3.1.4, to a 2D tensor product grid \mathcal{M} .



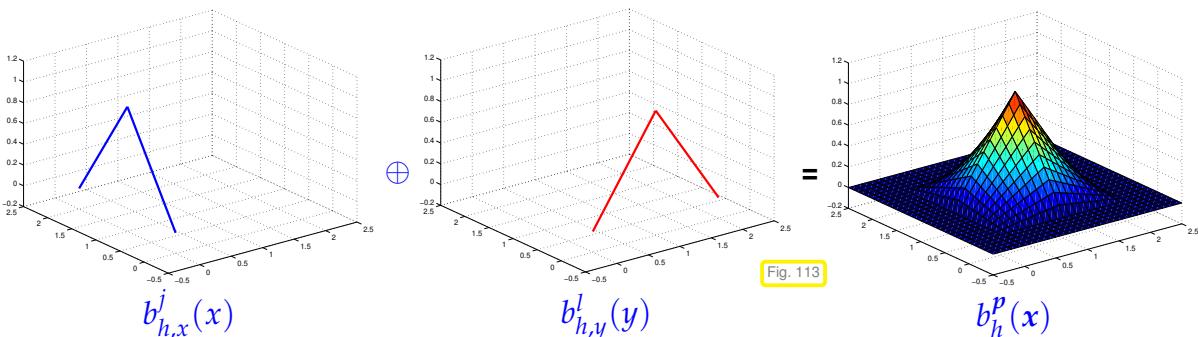
Idea: Tensor product structure of \mathcal{M} \rightarrow *tensor product construction* of FE space

This is best elucidated by a tensor product construction of basis functions:

$$\begin{aligned} b_{N,x}^j(x) &: 1\text{D tent function on } \mathcal{M}_x = \{[x_{j-1}, x_j], j = 1, \dots, n\} \\ b_{N,y}^l(y) &: 1\text{D tent function on } \mathcal{M}_y = \{[y_{l-1}, y_l], l = 1, \dots, n\} \end{aligned}$$

tensor product “tent function” associated with node \mathbf{p} :

$$b_h^p(x) = b_{N,x}^j(x_1) \cdot b_{N,y}^l(x_2), \quad \text{where } \mathbf{p} = \begin{bmatrix} x_j \\ y_l \end{bmatrix}. \quad (2.6.2.2)$$



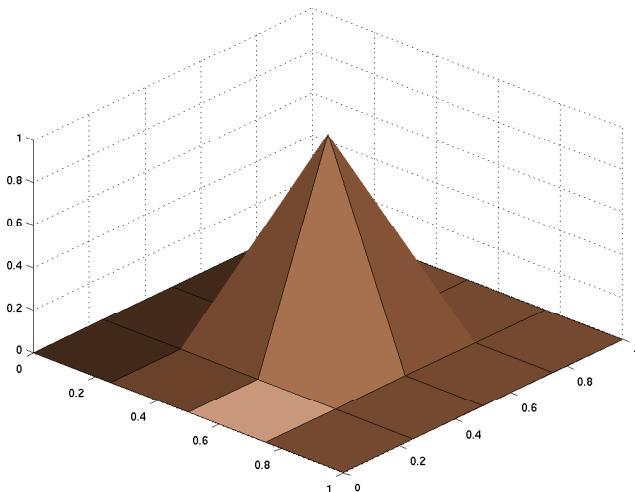


Fig. 114

Tensor product construction ➤ **bilinear** local shape functions, e.g. on $K =]0, 1[^2$

$$\begin{aligned} b_K^1(x) &= (1 - x_1)(1 - x_2), \\ b_K^2(x) &= x_1(1 - x_2), \\ b_K^3(x) &= x_1x_2, \\ b_K^4(x) &= (1 - x_1)x_2. \end{aligned} \quad (2.6.2.3)$$

$$\blacktriangleright b_K^i(a^j) = \delta_{ij}, \quad 1 \leq i, j \leq 4,$$

that is, these basis functions satisfy a local version of the cardinal basis property (2.6.1.4).

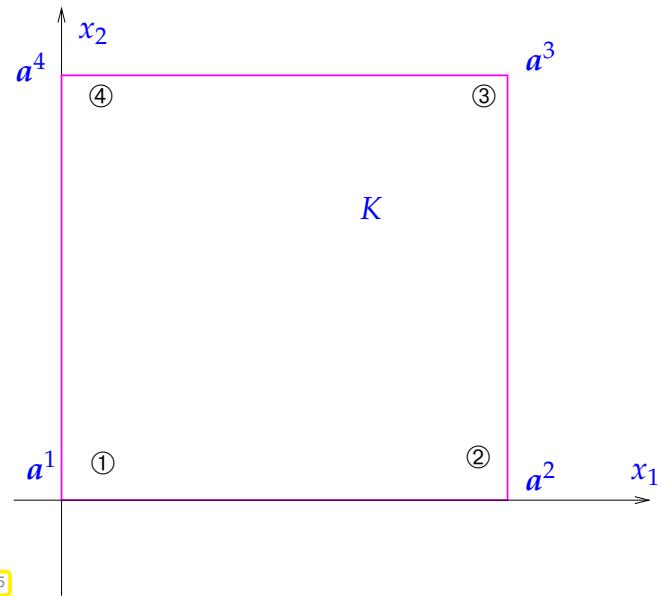


Fig. 115

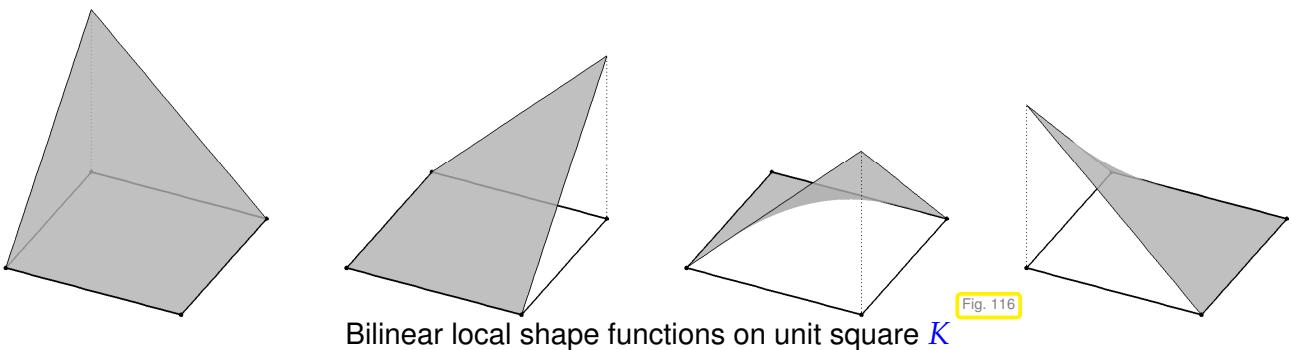


Fig. 116

Bilinear Lagrangian finite element space on 2D tensor product mesh \mathcal{M} :

$$\mathcal{S}_1^0(\mathcal{M}) := \{v \in C^0(\Omega): v|_K \in \mathcal{Q}_1(\mathbb{R}^2) \ \forall K \in \mathcal{M}\}. \quad (2.6.2.4)$$

The following is a natural generalization of (2.6.2.4) to higher degree local tensor product polynomials, see Def. 2.5.2.7. We apply the rule that on tensor-product meshes local spaces consisting of tensor-product polynomials should be used.

Definition 2.6.2.5. Tensor product Lagrangian finite element spaces

Space of p -th degree Lagrangian finite element functions on tensor product mesh \mathcal{M}

$$\mathcal{S}_p^0(\mathcal{M}) := \{v \in C^0(\bar{\Omega}): v|_K \in \mathcal{Q}_p(K) \forall K \in \mathcal{M}\}.$$

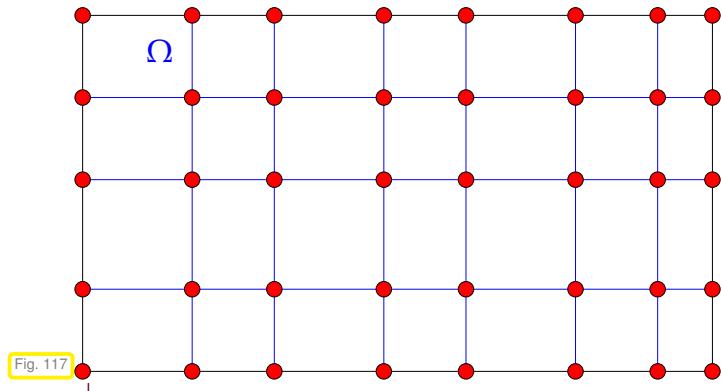
Terminology: $\mathcal{S}_1^0(\mathcal{M})$ = multilinear finite elements ($p = 1, d = 2$ = bilinear finite elements)

Remaining issue: definition of global basis functions (global shape functions)

Policy: Use of interpolation nodes p_1, \dots, p_N , $N := \dim \mathcal{S}_p^0(\mathcal{M})$ as in Section 2.6.1, see Ex. 2.6.1.2.
Define global shape functions b_h^1, \dots, b_h^N through cardinal basis property

$$b_h^j(p_i) = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{else,} \end{cases} \quad i, j \in \{1, \dots, N\}. \quad (2.6.1.4)$$

EXAMPLE 2.6.2.6 (Bi-linear tensor-product Lagrangian finite element functions) In the lowest-order case $p = 1$ everything is clear:



▷ — ≈ grid lines of a 2D tensor product mesh

● ≈ interpolation nodes for $\mathcal{S}_1^0(\mathcal{M})$

For $p = 1$ the interpolation nodes agree with the nodes of the mesh.

Note: A function $\in \mathcal{Q}_1(\mathbb{R}^2)$ is uniquely determined by its values in the corners of a rectangle.

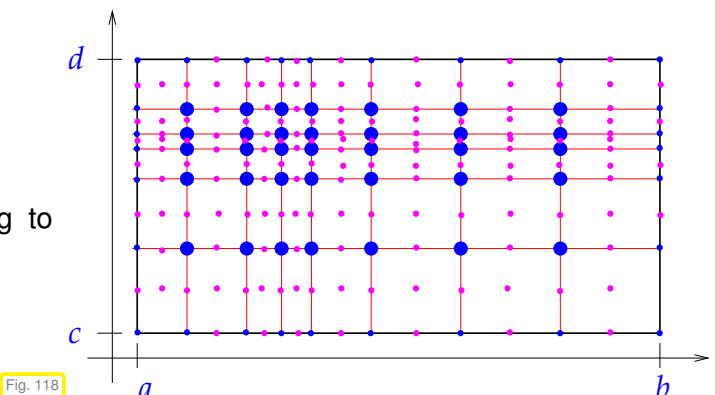
EXAMPLE 2.6.2.7 (Quadratic tensor product Lagrangian finite elements) Consider the case $p = 2, d = 2$ for Def. 2.6.2.5:

Interpolation nodes for $\mathcal{S}_2^0(\mathcal{M})$

$$\mathcal{N} = \mathcal{V}(\mathcal{M}) \cup \{\text{midpoints of edges}\}.$$

Note: number of interpolation nodes belonging to one cell is

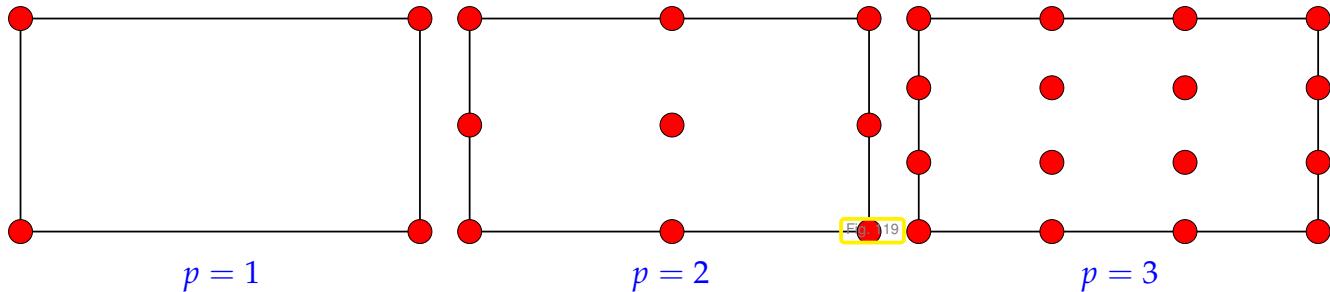
$$9 = \dim \mathcal{Q}_2(\mathbb{R}^2).$$



► Global basis functions defined by cardinal basis property analogously to (2.6.1.4).

$$\mathcal{N} = \{p_1, \dots, p_N\}: \quad b_h^j(p_i) \in \mathcal{S}_2^0(\mathcal{M}), \quad b_h^j(p_i) := \begin{cases} 1 & \text{if } j = i, \\ 0 & \text{else.} \end{cases}$$

Choice of interpolation nodes for tensor product Lagrangian finite elements of higher degree:



§2.6.2.8 (Imposing homogeneous Dirichlet boundary conditions) What is a global basis for $\mathcal{S}_p^0(\mathcal{M}) \cap H_0^1(\Omega)$, where \mathcal{M} is either a simplicial mesh or a tensor product mesh?

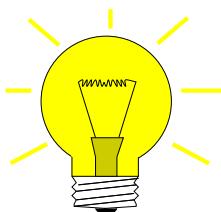
We proceed analogous to § 2.4.3.7: recall that global basis functions are defined via interpolation nodes $\mathbf{p}_j, j = 1, \dots, N$, see (2.6.1.4).

$$\mathcal{S}_{p,0}^0(\mathcal{M}) := \mathcal{S}_p^0(\mathcal{M}) \cap H_0^1(\Omega) = \text{Span}\{\mathbf{b}_h^j : \mathbf{p}_j \in \Omega \text{ (interior node)}\}. \quad (2.6.2.9)$$

In words: the subspace $\mathcal{S}_{p,0}^0(\mathcal{M})$ of functions in $\mathcal{S}_p^0(\mathcal{M})$ that vanish on $\partial\Omega$ can be obtained by dropping all global shape functions associated with interpolation nodes on $\partial\Omega$. \square

Remark 2.6.2.10 ((Bi)-linear Lagrangian finite elements on hybrid meshes) 2D hybrid meshes are meshes containing both triangles and quadrilaterals. At this point we assume that all the quadrilaterals are rectangles in order to remain in the framework of the tensor-product construction of Ex. 2.6.2.7.

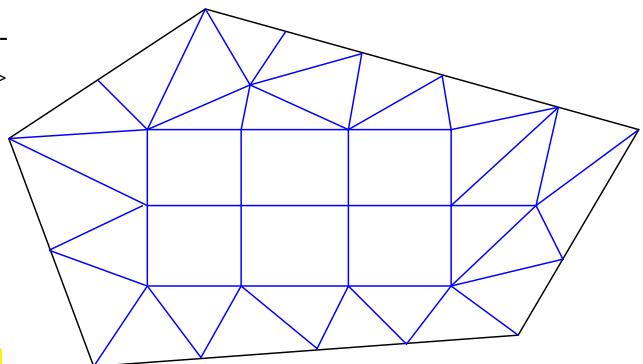
\mathcal{M} : 2D hybrid mesh comprising triangles & rectangles



Idea: use

- ◆ linear functions (\rightarrow Def. 2.5.2.2, $p = 1$) on triangular cells,
- ◆ bi-linear functions (\rightarrow Def. 2.6.2.5, $p = 1$) on rectangles.

▷ Fig. 120



$$\mathcal{S}_1^0(\mathcal{M}) = \left\{ v \in H^1(\Omega) : v|_K \in \begin{cases} \mathcal{P}_1(\mathbb{R}^2) & , \text{if } K \in \mathcal{M} \text{ is triangle,} \\ \mathcal{Q}_1(\mathbb{R}^2) & , \text{if } K \in \mathcal{M} \text{ is rectangle} \end{cases} \right\}. \quad (2.6.2.11)$$

Two issues arise:

1. Does the prescription (2.6.2.11) yield a large enough space? (Note that $v \in H^1(\Omega) \Rightarrow \mathcal{S}_1^0(\mathcal{M}) \subset C^0(\Omega)$, see Thm. 1.3.4.23, but continuity might enforce too many constraints.)
2. Does the space from (2.6.2.11) allow for locally supported basis functions associated with nodes of the mesh?

We will give a positive answer to both questions by constructing the basis functions:

Define global shape functions \mathbf{b}_h^j according to (2.4.3.5)

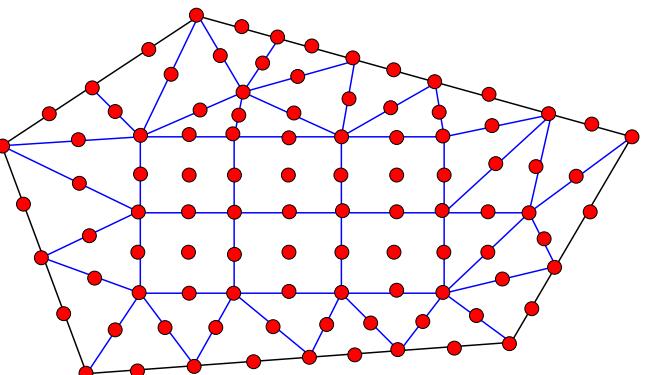
This makes sense, because

- ◆ linear/bi-linear functions on K are uniquely determined by their values in the vertices,
 - ◆ the restrictions to an edge of K of the local linear and bi-linear shape functions are both *linear* univariate functions, see Fig. 75 and Fig. 116.
- Fixing vertex values for $v_h \in \mathcal{S}_1^0(\mathcal{M})$ uniquely determines v on all edges of \mathcal{M} already, thus, *ensuring global continuity*, which is necessary due to Thm. 1.3.4.23. □

Remark 2.6.2.12 (Lagrangian finite elements on hybrid meshes) Does the construction employed in Rem. 2.6.2.10 carry over to polynomial degrees $p > 1$? let us examine the case $p = 2$, drawing on Ex. 2.6.1.2 and Ex. 2.6.2.7.

- \mathcal{M} : 2D hybrid mesh comprising triangles & rectangles, as in Rem. 2.6.2.10
- ☞ Matching interpolation nodes on edges of triangles and rectangles
 - Glueing of local shape functions on triangles and rectangles possible

global interpolation nodes for $p = 2$ □
Fig. 121



Review question(s) 2.6.2.13 (Lagrangian finite elements)

(Q2.6.2.13.A)

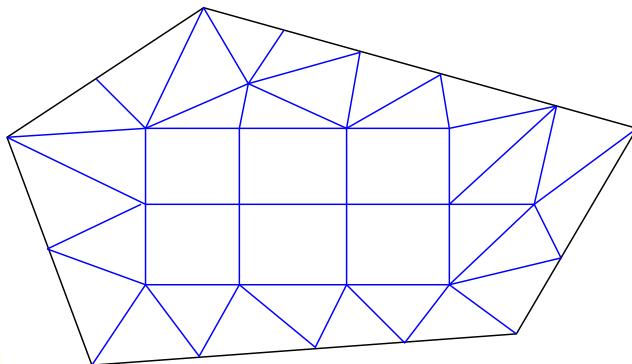


Fig. 122

(Q2.6.2.13.B) Let \mathcal{M} be a triangular mesh with $\#\mathcal{V}(\mathcal{M})$ vertices, $\#\mathcal{E}(\mathcal{M})$ edges, and $\#\mathcal{M}$ cells. What is $\dim \mathcal{S}_p^0(\mathcal{M})$ for $p = 1, 2, 3$?

(Q2.6.2.13.C) For a triangular mesh \mathcal{M} with $\#\mathcal{V}(\mathcal{M})$ vertices, $\#\mathcal{E}(\mathcal{M})$ edges, and $\#\mathcal{M}$ cells give sharp upper bounds for the number of non-zero entries of the Galerkin matrix arising from the finite element discretization of

$$u \in H^1(\Omega): \quad \int_{\Omega} \alpha(x) \operatorname{grad} u \cdot \operatorname{grad} v + c(x) u v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in H^1(\Omega), \quad (2.1.2.2)$$

with trial and test space $\mathcal{S}_p^0(\mathcal{M})$, $p = 1, 2$.

(Q2.6.2.13.D) Consider a tensor product mesh \mathcal{M} of $\Omega := [0, 1]^2$ with $n \in \mathbb{N}$ cells in each direction and the finite-element space

$$V_{0,h} := \left\{ v \in H_0^1(\Omega) : v|_K \in \mathcal{Q}_1(\mathbb{R}^2) \forall K \in \mathcal{M} \right\}.$$

What is the dimension of this space?

Explain why the local shape functions

- for $\mathcal{S}_1^0(\mathcal{M})$ on a triangular mesh (= the barycentric coordinate function of Ex. 2.5.3.6)
- and those for $\mathcal{S}_1^0(\mathcal{M})$ on a tensor-product mesh as defined by (2.6.2.3)

remain valid local shape functions for the lowest degree Lagrangian finite element space on the hybrid mesh displayed beside.

(Q2.6.2.13.E) Let \mathcal{M} be a tensor product mesh of $\Omega := [0, 1]^2$ and $\widetilde{\mathcal{M}}$ a triangular mesh arising from \mathcal{M} by splitting each rectangular cell into two congruent triangles. Show that $\mathcal{S}_1^0(\mathcal{M}) \neq \mathcal{S}_1^0(\widetilde{\mathcal{M}})$.

(Q2.6.2.13.F) Express the local shape functions for linear Lagrangian finite elements on a triangle as linear combinations of the quadratic local shape functions as given in

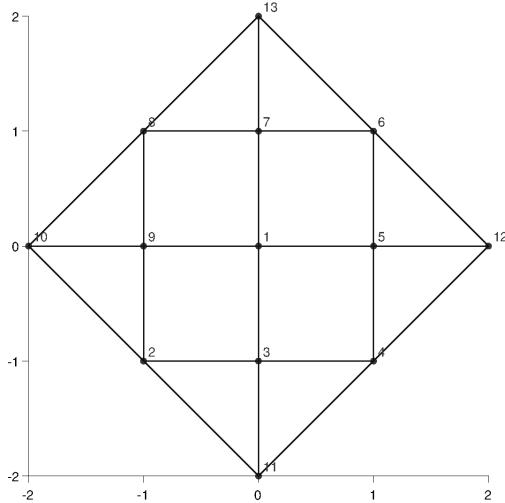
$$\begin{aligned} b_K^1 &= (2\lambda_1 - 1)\lambda_1, & b_K^2 &= (2\lambda_2 - 1)\lambda_2, \\ b_K^3 &= (2\lambda_3 - 1)\lambda_3, & b_K^4 &= 4\lambda_1\lambda_2, \\ b_K^5 &= 4\lambda_2\lambda_3, & b_K^6 &= 4\lambda_1\lambda_3. \end{aligned} \quad (2.6.1.6)$$

(Q2.6.2.13.G) Characterize the space of gradients of $\mathcal{P}_p(\mathbb{R}^2)$ and $\mathcal{Q}_p(\mathbb{R}^2)$ as spaces of componentwise polynomial vectorfields.

(Q2.6.2.13.H) We consider the bilinear form

$$b(u, v) := \int_{\partial\Omega} (1 + \|x\|) u(x)v(x) \, dS, \quad (2.6.2.14)$$

where $\partial\Omega$ is the boundary of the domain sketched in Fig. 123.



Writing \mathcal{M} for the mesh drawn in Fig. 123, what is the maximal number of nonzero entries of the Galerkin matrix arising from the finite element Galerkin discretization of $b(\cdot, \cdot)$ using $\mathcal{S}_1^0(\mathcal{M})$ equipped with the standard nodal basis as trial and test space?

Fig. 123

(Q2.6.2.13.I) For the hybrid mesh \mathcal{M} displayed in Fig. 123 determine the dimensions of the finite element spaces

- (i) $\mathcal{S}_1^0(\mathcal{M}) := \left\{ v \in C^0(\overline{\Omega}) : \begin{array}{l} v|_K \in \mathcal{P}_1(K) \quad \forall \text{triangles } K \in \mathcal{M} \\ v|_K \in \mathcal{Q}_1(K) \quad \forall \text{rectangles } K \in \mathcal{M} \end{array} \right\},$
- (ii) $\mathcal{S}_{1,0}^0(\mathcal{M}) := \mathcal{S}_1^0(\mathcal{M}) \cap H_0^1(\Omega).$

(Q2.6.2.13.J) We perform the Galerkin discretization of

$$a(u, v) := \int_{\partial\Omega} u(x)v(x) \, dS,$$

where $\partial\Omega$ is the boundary of the domain from Fig. 123, by means of the finite element space $\mathcal{S}_1^0(\mathcal{M})$ equipped with the standard nodal basis. Here \mathcal{M} is the hybrid mesh sketched in Fig. 123.

Compute the **element matrices** for the two cells of the mesh of Fig. 123 whose vertices are the nodes with the following numbers:

- (i) nodes 9, 1, 7, 8 (quadrilateral),
- (ii) nodes 3, 11, 4 (triangle)

Any local numbering of the nodes can be used.

△

2.7 Implementation of Finite Element Methods

This section discusses algorithmic details of Galerkin finite element discretization of 2nd-order elliptic variational problems for spatial dimension $d = 2, 3$ on bounded polygonal/polyhedral domains $\Omega \subset \mathbb{R}^d$.

Remember that guiding principle behind the implementation of finite element codes is

to rely on *local* computations as much as possible!

We witnessed this principle in action already in Section 2.4.5 (→ Code 2.4.5.23) and Section 2.4.6 (→ Code 2.4.6.8). Local computations are enough thanks to *local supports* of the global basis functions, see Section 2.5.3, Ex. 2.5.3.2.

Remark 2.7.0.1 (LEHRFEM++ – A simple but flexible C++ finite element library → GITHUB) The implementation of finite element methods in this course will rely on the C++ finite element library **LEHRFEM++**, a software package offering a framework for the implementation of finite element methods on 2D hybrid meshes.

LEHRFEM++ was designed with the following objectives in mind: ease of use, clarity of structure, modularity, consistency of design, ability to accommodate a vast range of 2D FEM, and compliance with best practices of C++ programming.

LEHRFEM++ is an open source software and is being developed at ETH Zurich. Lead developers are Dr. Raffael Casagrande and Prof. Dr. Ralf Hiptmair and several students of ETH Zurich have contributed and are contributing to LEHRFEM++. An extensive [documentation](#) is available created using [DOXYGEN](#). All modules of the library come with thorough unit tests in `test` sub-directories.

LEHRFEM++ can be cloned from → [GITHUB](#) using the command.

```
1 git clone https://github.com/crafael/lehrfempp.git
```

However, LEHRFEM++ and all other required libraries will automatically be installed by the [cmake](#) build system set up for this course.

Remark 2.7.0.2 (Learning LEHRFEM++) Ultimately, proficiency in LEHRFEM++ is a learning outcome of this course and essential for performing well in the examination.

Though individual learning styles differ widely, here are some recommendations on how to become familiar with LEHRFEM++:

- ◆ Follow the order in which aspects of LEHRFEM++ are covered in the course and in this lecture document.
- ◆ Study the [sample code listings](#). The sources are also available on → [GITHUB](#).
- ◆ Consult the [DOXYGEN documentation](#) → [GITHUB](#), which give comprehensive information about all functions and classes and also contains code snippets demonstrating their use.
- ◆ Do not hesitate to inspect the very source code, which usually comes with verbose inline comments.

- ◆ Look at the test codes in `test` subdirectories of every module directory. They demonstrate the use of classes and functions for a particular purpose.
- ◆ Last, but not least: Practice coding finite element computations with **LEHRFEM++** by doing all corresponding homework programming assignments.

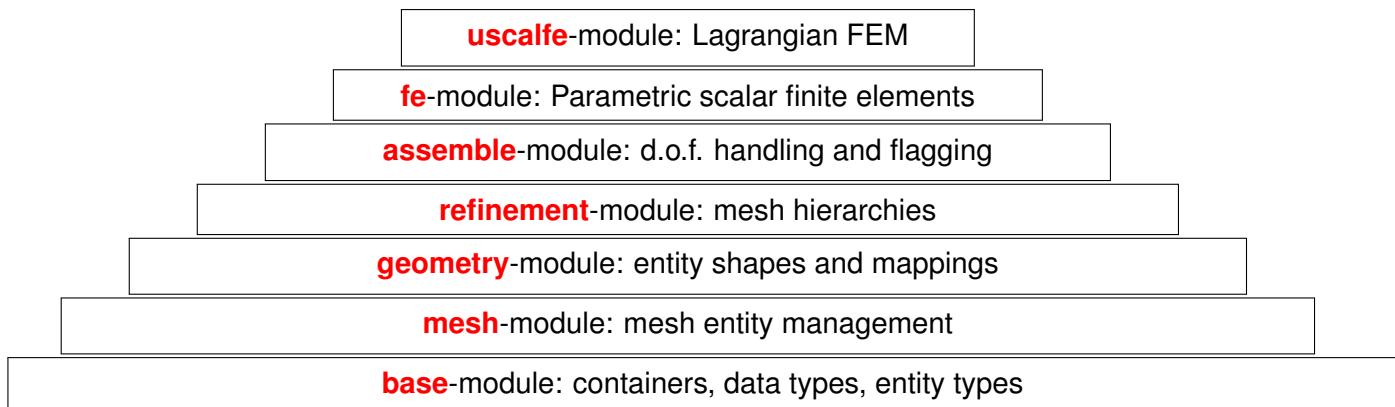
↓

Remark 2.7.0.3 (The rationale behind learning LEHRFEM++) Of course, LEHRFEM++ is a special piece of software and not even a particularly powerful one, though it has turned out to be useful as a software platform for small thesis projects.

However, the main reason for covering LEHRFEM++ in this course and using it for exercises is that it can serve as *prototypical finite-element software library* in terms of modules, data structures, and algorithms. Many other FE software libraries with many more and more sophisticated capabilities are available, for instance **MFEM** [And+21], **XLIFFE++**, or **NETGEN/NGSOLVE**. However, their wealth of functionality makes them hard to grasp for a beginner and they are hardly suitable for teaching. Learning LEHRFEM++ will prepare you well for using such software.

↓

Remark 2.7.0.4 (Modules of LEHRFEM++) Using C++ namespaces LEHRFEM++ is organized in *modules* according to a hierarchy of dependency that can be visualized by the following pyramid diagram:



These are supplemented by the

- **io**-module: reading and writing from/to files meshes and finite element functions,
- **quad**-module: supplying pre-defined quadrature rules.

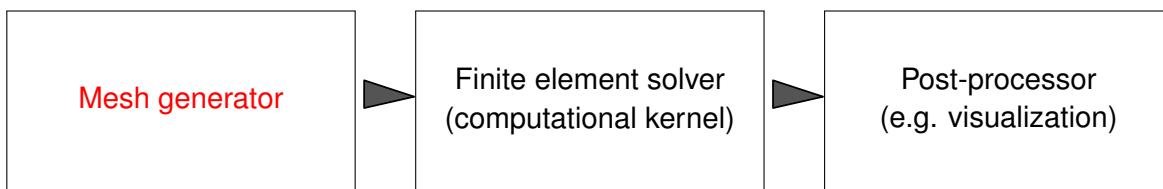
The purpose of some might be obscure at this point, but this chapter and the next one will shed light on everything.

↓

2.7.1 Mesh Generation and Mesh File Format

In Section 2.5.1 we identified triangulations (→ Def. 2.5.1.1) as one of the main building blocks of finite element methods. Their algorithmic generation turns out to be a separate issue, because the data flow in (most) finite element software packages look like this:





Here “ \rightarrow ” designates passing of information, which is usually done by writing and reading **files** to and from hard disk. This requires particular file formats.

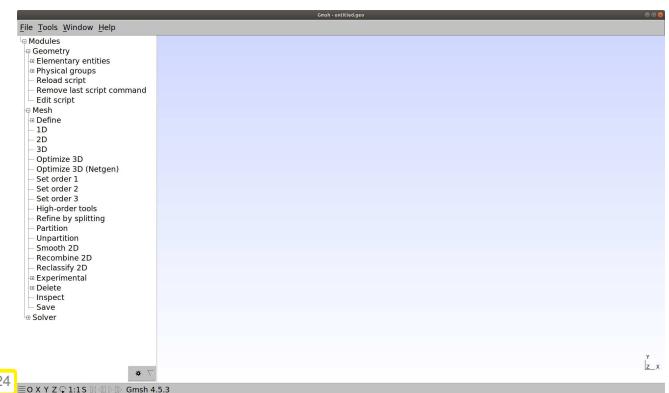
Algorithms for generating a finite element mesh from some description of the geometry of the computational domain are *beyond the scope of this course*. Sophisticated methods have been developed over many years and they are implemented in powerful commercial software packages. The problem of generating “suitable” finite element meshes without user interference is a persistent research topic, because complex geometries (slender domain, multiple length scales, layered media, etc.) entail immense challenges.

Supplement 2.7.1.1 (Gmsh – geometric modeling and mesh generation tool [GR09]) We use the open source geometric modeler and mesh generator **Gmsh** (pronounced “G-mesh”), which is employed in many projects in academic and industrial research.

Gmsh has been and is being developed by [Prof. Ch. Geuzaine](#) and [Prof. J.-F. Remacle](#) at the University of Liège in Belgium. Code and documentation are available through

<http://gmsh.info/>

Gmsh graphical user interface (GUI) \Rightarrow



For most Linux distributions Gmsh is available in repositories of precompiled codes accessible through the standard installers.



GMSH development is still ongoing and new versions with expanded functionality and slightly changed interfaces are being released continuously. Thus, some GMSH-related information in this document may not be current.

The latest stable version of GMSH as of January 13, 2023 is 4.5.3, which saves meshes in a format different from what we discuss below. In order to save in the old format you have to do the following steps:

1. Menu item **File** \rightarrow **Export...**
2. A dialogue pops up; there choose a filename with the extension **.msh**. Then click “OK”.
3. Another dialogue “**MSH Options**” pops up, in which you may choose “**Version 2 ASCII**” as format. Then click “OK”.

In order to make the “**Version 2 ASCII**” format the default, you visit the menu item **File** \rightarrow **Save Options As Default**.

EXAMPLE 2.7.1.2 (Geometric modeling with Gmsh) In this example we define a simple geometry interactively using the **Gmsh** geometric modeling interface. We specify *Points*, *Lines* and *Surfaces* that define our computational domain, the unit square $\Omega =]0,1[^2$.

① Setting points. Select the menu item

Modules → Geometry → Elementary entities → Add → Point.

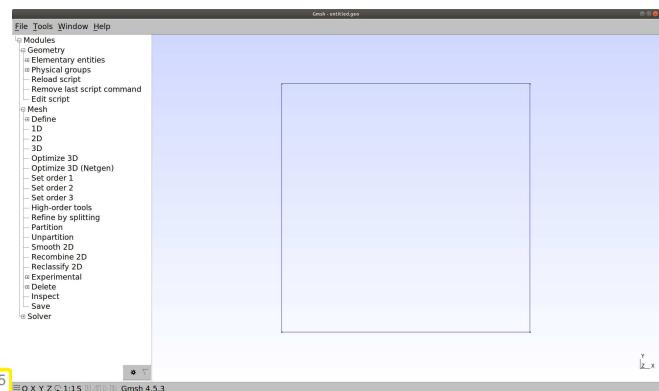
You can start adding points by interactively clicking, holding the position of the mouse, and pressing 'e'. The coordinates of your mouse pointer are reflected on the Contextual Geometry Definitions window that appears. It is, however, advisable to use this window and manually enter the coordinates of the points you want to create in the X, Y and Z coordinate text boxes.

Start by adding your first point with coordinates $(0, 0, 0)$ in the coordinate field. After you press return, one point should appear on your canvas. Similarly, add points $(1, 0, 0)$, $(1, 1, 0)$, and $(0, 1, 0)$. This sets all four corners of the square.

② Defining lines.

To add the lines that form the edges of our square, use the menu

Modules → Geometry → Elementary entities → Add → Line.

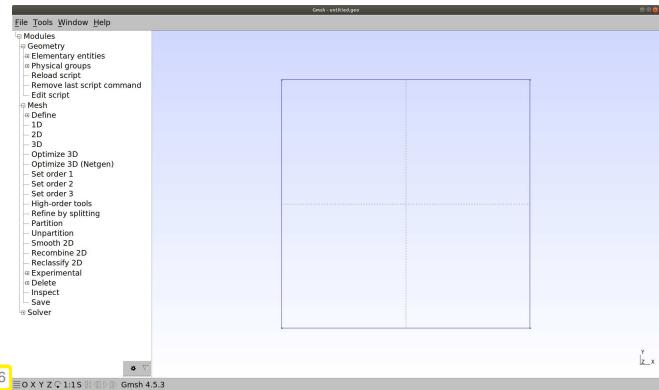


Now, select the point $(0, 0)$ as starting point. The selected point will be shown in red. Then, complete the line by selecting $(1, 0)$ as the end point. Similarly, create three other lines, forming a square. See the figure beside.

③ Creating surfaces (domains).

To finish the definition of a computational domain, we have to tell **Gmsh** which of closed line loops form a surface. This can be easily done by using the menu

Modules → Geometry → Elementary entities → Add → Plane Surface.



Then click on any part of the square. After the boundary has been selected, press 'e', to create a surface. See the figure beside for a screenshot.

Many online video tutorials for GMSH are available. e.g. [here](#).

EXAMPLE 2.7.1.3 (Gmsh geometry description file) When geometric elements are created interactively using the GUI, **gmsh** stores the data in a `.geo` file, with its own scripting language. This file can be opened, and edited by the menu

Modules → Geometry → Edit Script

The `.geo` file for the square mesh reads (without comments):

```

1 Point(1) = {0, 0, 0, 1};
2 Point(2) = {1, 0, 0, 1};
3 Point(3) = {1, 1, 0, 1};
4 Point(4) = {0, 1, 0, 1};

```

```

5 Line(1) = {1, 2};
6 Line(2) = {2, 3};
7 Line(3) = {3, 4};
8 Line(4) = {4, 1};
9 Curve Loop(1) = {1, 2, 3, 4};
10 Plane Surface(1) = {1};

```

The line numbers 1-4 in the above code define Points 1-4, with the following syntax,

`Point(id) = {x, y, z, mesh-size};`

Similarly, line numbers 5-8 specify the four edges of the square, as

`Line(id) = {id-of-start-point, id-of-end-point};`

Defining a closed polygon (line loop) also follows a similar syntax, but you have to make sure that lines are in proper cyclic order. To invert the direction of a line, use a minus sign before line id.

`Line Loop(id) = {id-of-line-1, id-of-line-2, id-of-line-3, ...};`

The final line of the code defines surface that is created with the line loop as the boundary, and is defined as follows.

`Plane Surface(id) = {id-of-line-loop, id-of-holes-loop};`

§2.7.1.4 (Generating a mesh with Gmsh) After the geometry has been specified or a .geo file has been read in, a mesh can be generated for currently active domain (surface). Click the menu

Modules → Mesh → 2D.

Then **Gmsh** should display an unstructured mesh for the square surface, see the figure beside for a mesh covering the square domain created in Ex. 2.7.1.2.

When points are added (→ Ex. 2.7.1.2) in the dialogue there is text box for Prescribed mesh size at point. This can be used to define the size of the meshes around a particular point. In this example this was set to 1. In general, this parameter can be used to control the local resolution of the mesh.

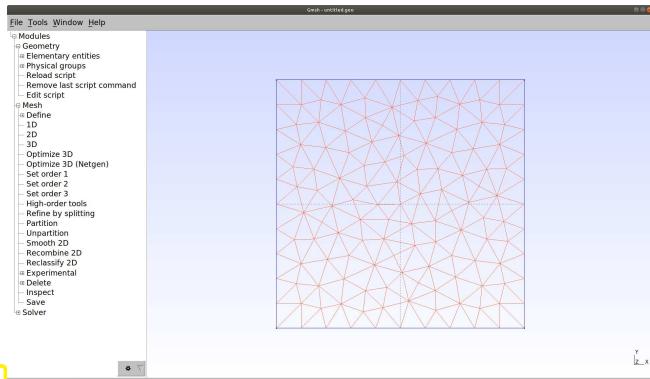
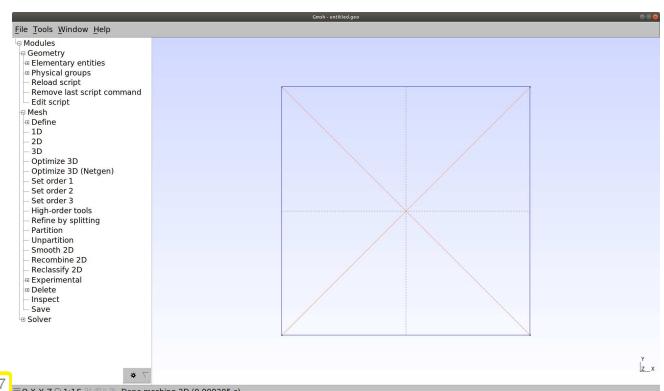


Fig. 128



To create a finer mesh, edit the .geo-file and specify a smaller local mesh size for the points (→ Ex. 2.7.1.3). Do not forget a subsequent click on Modules → Geometry → Reload.

The figure beside displays a mesh for the square generated with local mesh size 0.1. The **Delaunay** algorithm was used for that purpose after it had been selected in Tools → Options → Mesh.

EXAMPLE 2.7.1.5 (Gmsh file format for storing meshes) Now we take a look at the main mesh file format used in the course, the one in which GMSH saves its output.



The below mesh file is the version-2.2 format. By default latest versions of GMSH store mesh data in a different version-4 file format. Thus, it may be advisable to set the default format for saving meshes to “Version 2”.

Gmsh can store mesh data in plain ASCII .msh-files. The file corresponding to the mesh from Fig. 127 is as follows

Lines 1-3: Version number, file format, and floating point format used.

Line 4-11 (between \$Nodes and \$EndNodes): [List of nodes](#)

The first line in the nodes section gives the number of nodes in the mesh, followed by each of the nodes. In our case, the mesh comprises 5 nodes. In each node line, the first integer describes the id(identifier) of the entity in the .msh-file, followed by the x -, y - and z - coordinates of the node (floating point numbers). In the example, the four points we created are part of the mesh, and **Gmsh** has created a new fifth node in the center of the mesh at coordinates $(0.5, 0.5, 0)$.

Line 12-26 (between \$Elements and \$EndElement s): [List of elements and boundary entities](#)

Some entities, such as points, lines, triangles, quadrangles, etc., are coded in this section. The first line (line number 13) gives the number of entities listed. In each entity line, the integer denotes the entity id(identifier), followed by an identifier for a *type* of the entity. The third integer denotes the number of tags for this entity, followed by that many integers (tags). The meaning of the tags will be covered in Rem. 2.7.1.8. The remainder of the line lists the id(identifier)s of nodes which are contained in the boundary of this particular entity.

```

1 $MeshFormat
2 2.2 0 8
3 $EndMeshFormat
4 $Nodes
5
6 1 0 0 0
7 2 1 0 0
8 3 1 1 0
9 4 0 1 0
10 5 0.5 0.5 0
11 $EndNodes
12 $Elements
13 12
14 1 15 2 0 1 1 1
15 2 15 2 0 2 2 2
16 3 15 2 0 3 3 3
17 4 15 2 0 4 4 4
18 5 1 2 0 1 1 2
19 6 1 2 0 2 2 3
20 7 1 2 0 3 3 4
21 8 1 2 0 4 4 1
22 9 2 2 0 1 1 2 5
23 10 2 2 0 1 4 1 5
24 11 2 2 0 1 2 3 5
25 12 2 2 0 1 3 4 5
26 $EndElement s

```

§2.7.1.6 (Gmsh Element Types) GMSH uses numeric type specifiers to convey information about the topological type of mesh entities in .msh files:

A selection of *entity types* used by **Gmsh** for 2D meshes

The meaning of “3-node line” and “6-node triangle” will be explained in 2.8.4.3.

Number	Element Type
15	1-node point
1	2-node line
2	3-node triangle
3	4-node quadrilateral
8	3-node line
9	6-node triangle

For example, in Ex. 2.7.1.5 line number 14 describes an entity with identifier 1, and the element type of 15, which is a *1-node point*. This entity has 2 tags (for now ignore the following 2 integers). The last integer, 1, corresponds to the node identifier which is a part of the entity. The node identifier 1 is the point that is located at $(0, 0, 0)$. Hence the entity 1, corresponds to the node 1. The entities 2, 3, and 4 are also of type “1-node point”, and represent the points $(1, 0, 0)$, $(1, 1, 0)$ and $(0, 1, 0)$, respectively.

The line numbers 18-21 code for entities with identifiers 5-8, and represent entities of type 1, which is a *2-node line*. Ignoring the tags, the last two integers represent the nodes corresponding to the endpoints of the lines. Their order endows the line with a direction. For instance, entity number 7 represents a line

between the nodes 3 and 4. Hence, this is a line between the Points $(1, 1, 0)$ and $(0, 1, 0)$.

The element numbers 9-12 represent, the element of type 2, which are **3-node triangles**. The last three integers give the node numbers of the vertices. For example, the element number 9, is a triangle with vertices $(0, 0, 0)$, $(1, 0, 0)$ and $(0.5, 0.5, 0)$. \square

Gmsh mesh file format: interior edges skipped

Note that only the points and lines (edges) which are a part of the boundary are included as separate entities; interior edges and points are not.

Remark 2.7.1.8 (Gmsh – marking parts of a mesh by tags) Often one wants to distinguish parts of the computational domain (sub-domains), where special coefficient functions or source functions should be used. Moreover, parts of the boundary have to be marked, if they carry different boundary conditions as in Ex. 1.7.0.10. In **Gmsh** this can be achieved by assigning mesh entities to different **physical groups**. Those can be created using the menu item

Modules → Geometry → Physical Groups → Add

```

1 Point(1) = {0, 0, 0, 1};
2 Point(2) = {1, 0, 0, 1};
3 Point(3) = {1, 1, 0, 1};
4 Point(4) = {0, 1, 0, 1};
5 Line(1) = {1, 2};
6 Line(2) = {2, 3};
7 Line(3) = {3, 4};
8 Line(4) = {4, 1};
9 Curve Loop(1) = {1, 2, 3, 4};
10 Plane Surface(1) = {1};
11 Physical Point("bottom-pts") = {1, 2};
12 Physical Curve("top-bottom") = {1, 3};
13 Physical Curve("left-right") = {4, 2};
14 Physical Surface("thesurface") = {1};

```

Physical groups are distinguished by their name and the .geo-file for the square extended by physical groups may look as follows. \triangleright

In this file four physical groups are present, called “bottom-pts”, “top-bottom”, “left-right”, and “thesurface”.

Gmsh manages physical groups using the **tags**, whose discussion we skipped in Ex. 2.7.1.5. For instance, the `.msh`-file generated from the above `.geo`-file is printed beside.

The `.msh` file has changed; a new section `$PhysicalNames` has been created which tabulates the keys for looking up the physical groups the entities belong to. Line 5 gives the number of physical groups. Each line of this section has the following format: the first integer defines the dimension of the entities of the group (0 for points, 1 for lines and 2 for surfaces). The second integer gives the tag number for the physical group, and third its corresponding string identifier.

Comparing with Ex. 2.7.1.5, we can see that the tag numbers of elements have changed. The first tag denotes the physical group an entity belongs to, and the second tag represents the geometric entity it belongs to. The groupings can be read off from the tags of the entities and the corresponding name of the physical group. For example, the elements 1 and 2 belong to the physical group 1, which is “bottom-pts”. The elements 3 and 5 that are lines have the physical tag 2, which corresponds to the group “top-bottom”. Similarly, entities 4 and 6 belong to “left-right”. The remaining entities that are triangles belong to the physical group “thesurface”.

```

1 $MeshFormat
2 2.2 0 8
3 $EndMeshFormat
4 $PhysicalNames
5 4
6 0 1 "bottom-pts"
7 1 2 "top-bottom"
8 1 3 "left-right"
9 2 4 "thesurface"
10 $EndPhysicalNames
11 $Nodes
12 5
13 1 0 0 0
14 2 1 0 0
15 3 1 1 0
16 4 0 1 0
17 5 0.5 0.5 0
18 $EndNodes
19 $Elements
20 10
21 1 15 2 1 1 1
22 2 15 2 1 2 2
23 3 1 2 2 1 1 2
24 4 1 2 3 2 2 3
25 5 1 2 2 3 3 4
26 6 1 2 3 4 4 1
27 7 2 2 4 1 1 2 5
28 8 2 2 4 1 4 1 5
29 9 2 2 4 1 2 3 5
30 10 2 2 4 1 3 4 5
31 $EndElements

```

↓

EXAMPLE 2.7.1.9 (Gmsh – meshing more complex geometries) Curved boundaries can also be modelled in **Gmsh**. Refer to the documentation for details.

```

1 Point (1) = {0, 0, 0, 0.25};
2 Point (2) = {1, 0, 0, 0.25};
3 Point (3) = {0, 2, 0, 0.25};
4 Point (4) = {1, 2, 0, 0.25};
5 Point (5) = {2, 2, 0, 0.25};
6 Point (6) = {2, 3, 0, 0.25};
7 Point (7) = {2, 4, 0, 0.25};
8 Point (8) = {4, 4, 0, 0.25};
9 Point (9) = {4, 3, 0, 0.25};
10 Line(1) = {3, 1};
11 Line(2) = {1, 2};
12 Line(3) = {2, 4};
13 Line(4) = {6, 9};
14 Line(5) = {9, 8};
15 Line(6) = {8, 7};
16 Circle(7) = {4, 5, 6};
17 Circle(8) = {3, 5, 7};
18 Curve Loop(1) = {1, 2, 3, 7, 4, 5, 6, -8};
19 Plane Surface(1) = {1};

```

↓

Remark 2.7.1.10 (Other tools for mesh generation) Freely available mesh generators:

- ◆ [DistMesh](#) (MATLAB)
- ◆ [Triangle](#) (easy to use 2D mesh generator)
- ◆ [TETGEN](#) (Tetrahedral mesh generation)
- ◆ [NETGEN](#) (industrial strength open source mesh generator)

↓

EXAMPLE 2.7.1.11 (LEHRFEM++ building mesh from Gmsh mesh file) LEHRFEM++ offers rather advanced facilities for parsing Gmsh mesh files (suffix .msh) and building mesh data structures from the information contained in them. The following code reads the data for a 2D hybrid mesh from file.

C++ code 2.7.1.12: Use of LEHRFEM++'s `If::io::GmshReader` functionality → [GITHUB](#)

```

2 // Create a 2D mesh data structure from the information contained in
3 // the file mesh_file. A factory object is in charge of
4 // creating mesh entities and has to be initialized first.
5 auto factory = std::make_unique<If::mesh::hybrid2d::MeshFactory>(2); //
6 If::io::GmshReader reader(std::move(factory), mesh_file.string()); //
7 // Obtain pointer to read only mesh from the mesh reader object
8 // Meshes in LEHRFEM++ are managed through shared pointers, see
9 // documentation.
10 std::shared_ptr<const If::mesh::Mesh> mesh_ptr = reader.mesh();
11 const If::mesh::Mesh &mesh{*mesh_ptr};

12
13 // Output general information on mesh; self-explanatory
14 std::cout << "Mesh from file " << mesh_file.string() << ":" [
15     << mesh.DimMesh() << ',' << mesh.DimWorld()
16     << "] dim:" << std::endl;
17 std::cout << mesh.NumEntities(0) << " cells, " << mesh.NumEntities(1)
18     << " edges, " << mesh.NumEntities(2) << " nodes" << std::endl;

```

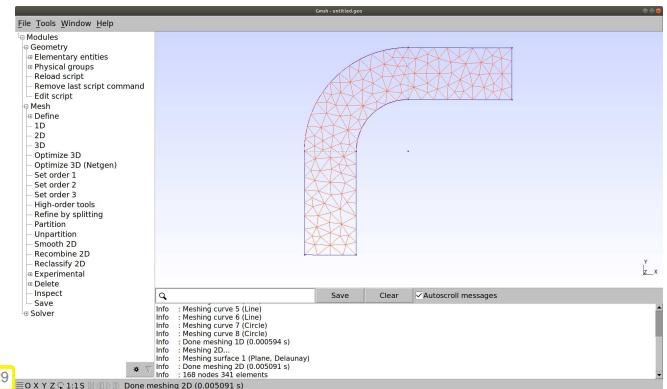


Fig. 129

The **factory object** of type **If::mesh::MeshFactory** built in Line 5 serves the policy to separate the creation of the building blocks of the mesh (nodes, edges, cells) from constructing their relationships describing the mesh. The purpose is to be able to use the algorithms for mesh construction with different types for the building blocks.

In LEHRFEM++ mesh objects are to be derived from the interface class **If::mesh::Mesh** → (documentation). All operations on a mesh are mediated through this class. Objects of type **If::mesh::Mesh** are created dynamically and managed through **shared pointers** > CppRef in order to ensure that they remain valid as long as they are in use in some part of the code. □

EXAMPLE 2.7.1.13 (Processing extra information in Gmsh mesh file with LEHRFEM++)

In Rem. 2.7.1.8 we saw that geometric mesh entities can be endowed with special “physical groups” tags in Gmsh. These are automatically extracted by LEHRFEM++ **If::io::GmshReader**-type helper objects through a method **PhysicalEntities()**. A demonstration code is available → GITHUB. □

Supplement 2.7.1.14 (Postprocessing: Visualization with PARAVIEW)

LEHRFEM++ supplies the helper class **If::io::VtkWriter**, which can be used to write data describing functions defined on a triangulated domain to so-called .vtk-files. Here, **VTK** stands for **Visulaization Toolkit**, an open-source software project targetting basic visualization tasks.

The documentation of **If::io::VtkWriter** gives details about how to created .vtk input files for the data visualization software **PARAVIEW** and about some basic visualizations with **PARAVIEW**.

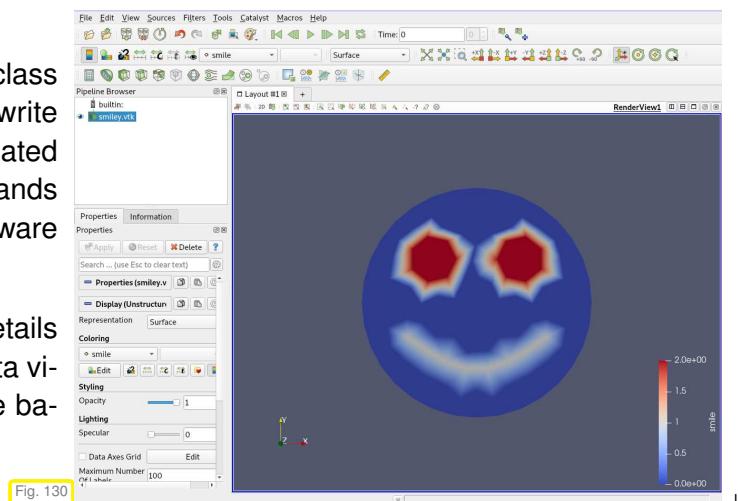


Fig. 130

Review question(s) 2.7.1.15 (Mesh generation and mesh file format)

(Q2.7.1.15.A) [Rationale for shared pointer] Explain, why **If::mesh::Mesh** objects in LEHRFEM++ are managed through shared pointers.

(Q2.7.1.15.B) [Creating a mesh with GMSH] Using GMSH create a .msh file describing a triangular mesh of the domain

$$\Omega := \{x \in \mathbb{R}^2 : \frac{1}{2} < \|x\| < 1\}.$$

What is the problem, if the mesh cells are all of type “3-node triangle”? △

2.7.2 Mesh Information and Mesh Data Structures



Video tutorial for Section 2.7.2: Mesh Information and Mesh Data Structures: (39 minutes)
[Download link](#), [tablet notes](#)

In this section we examine the internal *representation* of a mesh (→ Def. 2.5.1.1) in a computer code and the definition of suitable programming *interfaces*, with focus on LEHRFEM++, of course.

From an algorithmic and software point of view

a mesh is a collection of (geometric) entities (→ § 2.5.1.2) of particular shapes and location, connected by adjacency/incidence relations (*).

(*) An example for an incidence relation on the set of mesh entities is “is sub-entity of”.

To begin with, we have to be aware of the principal purposes to be served by the collection of objects and data that represent meshes in finite element codes.

Purposes of mesh data structures

Mesh data structures must

1. offer unique identification of mesh entities [= cells/(faces)/(edges)/vertices] (for instance, by an integer index)
2. make possible traversal of cells of the mesh (→ **global numbering**)
3. represent **mesh topology** (= incidence relationships of cells/faces/edges/vertices)
4. allow sequential access to edges/faces of a cell
(→ traversal of local shape functions/degrees of freedom)
5. describe **mesh geometry** (= location/shape of cells/faces/edges/vertices)

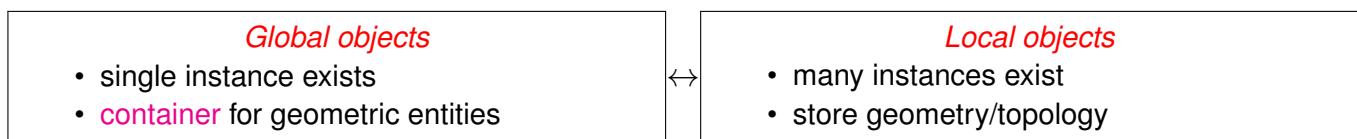
Remark 2.7.2.2 (Importance of global numbering of geometric entities) Remember from Section 2.2: we need on *ordered* basis \mathfrak{B} of the finite element space, that is, we have to establish a consecutive numbering of the finite element basis functions/global shape functions, $\mathfrak{B} = \{b_h^1, \dots, b_h^N\}$.

For assembly as explained in Section 2.4.5 we also assumed that the local shape functions carried numbers, there corresponding to the (local) numbers of the vertices of each triangle of the mesh. Thus, in a code using linear Lagrangian finite elements, we have to number the vertices of a mesh.

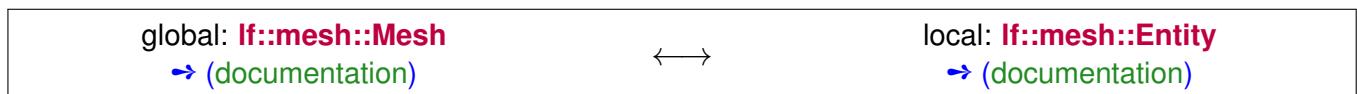
More generally, in Section 2.5.3 we saw that global shape functions are associated with geometric entities.

➤ Numbering geometric entities paves the way for numbering global shape functions.

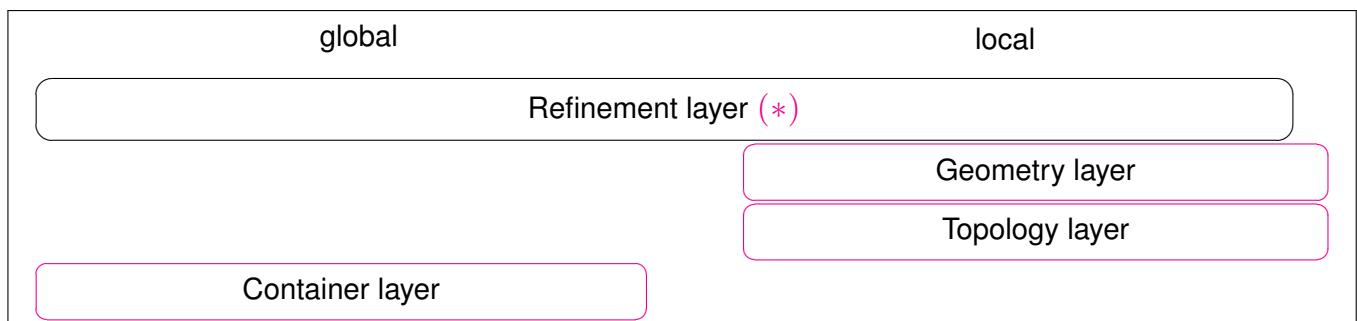
§2.7.2.3 (Classification of mesh functionality and data) The substantial functionality of a mesh representation and the supporting data structure can be categorized according to different aspects. The first approach distinguishes two kinds of data types and associated objects according to “local ↔ global”, “many ↔ few”:



In LEHRFEM++ this classification directly manifests itself in two fundamental interface classes (abstract base classes with purely virtual functions)



Another classification considers different **layers of information** contained in a mesh.



This classification is natural from the above point of view that, in terms of data structures and algorithms,

a mesh is a collection (\triangleright **container**) of (geometric) entities of particular shapes and location (\triangleright **geometry**), connected by adjacency relations (\triangleright **topology**).

- (*) “Refinement” refers to the process of creating a new mesh from an existing one by splitting all or some of its cells. This will briefly be addressed later in this course.

□

2.7.2.1 LEHRFEM++ Mesh: Container Layer

The container aspect for a mesh is covered by the main methods supplied by **If::mesh::Mesh** objects \rightarrow [\(documentation\)](#),

- `size_type lf::mesh::Mesh::NumEntities(dim_t codim) const;`

telling the total number of entities of the co-dimension `codim`,

- `nonstd::span<const Entity* const> lf::mesh::Mesh::Entities(unsigned codim) const;`

which returns a so-called `span` \rightarrow [\(documentation\)](#), an object that allows to run sequentially through all entities of a particular co-dimension (\rightarrow § 2.5.1.2).

- `bool lf::mesh::Mesh::Contains (const lf::mesh::Entity &e) const;`

which tests whether an entity belongs to the mesh.

Moreover, **If::mesh::Mesh** provides **indexing**, numbering of geometric entities through consecutive non-negative integers. For every entity in the mesh its index is returned by the member function

- `size_type lf::mesh::Mesh::Index(const Entity &e) const;`

Its index is *unique* for an entity of a fixed co-dimension, and it can even serve as an array index:

Global indexing of mesh entities in LEHRFEM++

For every mesh the global indices of the entities of any fixed co-dimension run from `0` to `n - 1`, where `n` is the total number of those entities.

This statement can be written as

$$\{ \text{mesh.Index}(e) : e \in \text{mesh.Entities}(\text{codim}) \} = \{0, \dots, \text{mesh.NumEntities}(\text{codim}) - 1\}. \quad (2.7.2.5)$$

Therefore, by the index we can access the entities of a mesh and a fixed co-dimension like the elements of an array:

```
lf::mesh::Entity *lf::mesh::Mesh::EntityByIndex(dim_t codim,
    lf::base::glb_idx_t index) const;
```

EXAMPLE 2.7.2.6 (Using entity iterators in LEHRFEM++) In this example we see a LEHRFEM++ function which implements loops over the entities of a **If::mesh::Mesh** object and prints their indices. These loops are fundamental for almost every task in a finite element code.

C++ code 2.7.2.7: Traversal of entities of a mesh → GITHUB

```

2 int traverseEntities(const lf::mesh::Mesh &mesh, dim_t codim) {
3     LF_ASSERT_MSG((codim <= mesh.DimMesh()), "codim " << codim << " too large");
4     std::cout << "Mesh dimension = " << mesh.DimMesh()
5             << ", iterating over entities of co-dim. " << codim << ":" 
6             << mesh.NumEntities(codim) << " exist" << std::endl;
7     size_type cnt = 0;
8     // Typical loop for running through all entities of a specific
9     // co-dimension
10    for (const lf::mesh::Entity *entity : mesh.Entities(codim)) {
11        // Print entity information including its unique index
12        std::cout << cnt << ": Entity #" << mesh.Index(*entity) << ":" << *entity
13                    << std::endl;
14        cnt++;
15    }
16    return cnt;
}

```

Note that the loop variable `entity` is a *pointer* to `lf::mesh::Entity` objects.

Supplement 2.7.2.8 (Mesh data sets in LEHRFEM++) Connected with the container capabilities of LEHRFEM++’s mesh data structures are auxiliary classes that can be used to attach arbitrary data to the entities of a mesh. The most important is `lf::mesh::utils::CodimMeshDataSet` → GITHUB, whose most important member function is the set/get access operator

```

T& lf::mesh::utils::CodimMeshDataSet::operator()(const Entity& e);
const T lf::mesh::utils::CodimMeshDataSet::operator()(const Entity&
e) const;

```

A (pointer to an) object of type `lf::mesh::utils::CodimMeshDataSet` can be created by calling

```

template <class S, class>
std::shared_ptr<CodimMeshDataSet<S>>
lf::mesh::utils::make_CodimMeshDataSet(
    std::shared_ptr<const lf::mesh::Mesh> mesh_p, base::dim_t codim, S
    init);

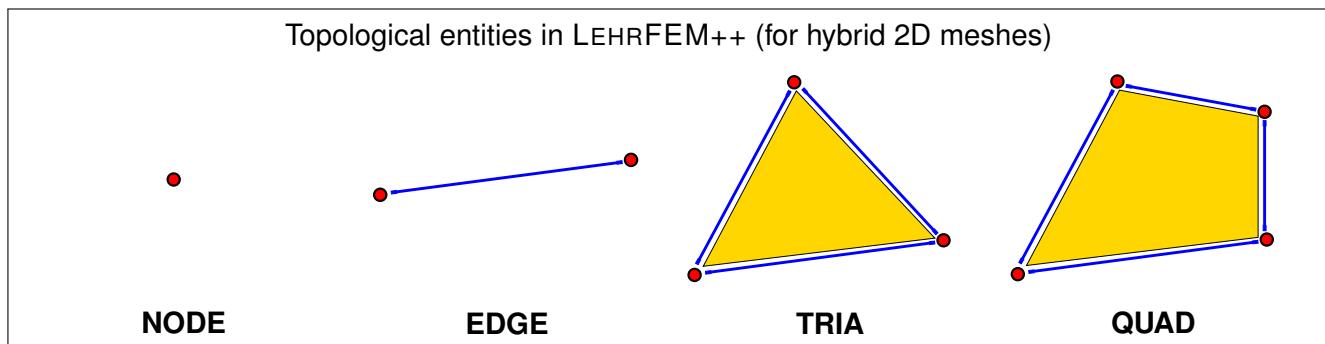
```

where `codim` specified the co-dimension of the entities to which data should be attached, and `init` is an optional default value for the data. The functions implemented in `special_entity_sets.cc` → GITHUB well demonstrate the use of these “arrays indexed by entities”.

2.7.2.2 LEHRFEM++ Mesh: Topology Layer

§2.7.2.9 (Topological types of mesh entities in 2D) We examine the adjacency/incidence relations restricted to a single entity. In LEHRFEM++ the topology of an entity can be queried by the method `RefEl()` of `lf::mesh::Entity`. It returns a *static object* of type `lf::base::RefEl` of which there are four different versions

- (I) **NODE**-type (= `lf::base::RefEl::kPoint()`): 0-dimensional entity without sub-entities,
- (II) **EDGE**-type (= `lf::base::RefEl::kSegment()`): a 1-dimensional entity with two 0-dimensional **NODE**-type subentities
- (III) **TRIA**-type (= `lf::base::RefEl::kTria()`): a 2-dimensional entity with three sub-entities of **EDGE**-type and further three sub-entities of **NODE**-type
- (IV) **QUAD**-type (= `lf::base::RefEl::kQuad()`): another 2-dimensional entity possessing four sub-entities of **EDGE**-type and **NODE**-type, respectively.



EXAMPLE 2.7.2.10 (Testing for topological type) The following C++ function demonstrates how to test for a particular topological type of an entity in LEHRFEM++.

C++ code 2.7.2.11: Counting cells of topological type QUAD and TRIA → GITHUB

```

2 std::pair<size_type, size_type> countCellTypes(const If::mesh::Mesh &mesh) {
3     size_type tria_cnt = 0;
4     size_type quad_cnt = 0; // Counters
5     // Loop over all cells (= co-dimension-0 entities) of the mesh
6     for (const If::mesh::Entity* cell : mesh.Entities(0)) {
7         // Fetch type information
8         If::base::RefEl ref_el{cell->RefEl()};
9         // Test, if current cell is of a particular type
10        switch (ref_el) {
11            case If::base::RefEl::kTri(): { tria_cnt++; break; }
12            case If::base::RefEl::kQuad(): { quad_cnt++; break; }
13            default: { LF_VERIFY_MSG(false, "Unknown cell type"); }
14        }
15    }
16    return {tria_cnt, quad_cnt};
}

```

An important aspect of local topology concerns the **local numbering/local ordering** of sub-entities, which underlies the implementation of the fundamental method

```

nonstd::span<const lf::mesh::Entity* const>
SubEntities(unsigned rel_codim) const

```

It returns a random access container, an array, of sub-entities of a particular *relative co-dimension*, which is the dimension gap between the current entity object and the requested sub-entity.

EXAMPLE 2.7.2.12 (Accessing sub-entities in LEHRFEM++) This code is a snipped from a unit test using the [Google Test](#) framework. It checks the matching of relative and absolute dimensions.

C++ code 2.7.2.13: Verification of consistency of relative dimensions → GITHUB

```

2 void checkRelCodim(const Entity &e) {
3     using dim_t = lf::base::RefEl::dim_t;
4     using RefEl = lf::base::RefEl;
5     using size_type = lf::mesh::Mesh::size_type;
6
7     // Obtain basic information about current Entity
8     const RefEl ref_el = e.RefEl();
9     const dim_t dimension = ref_el.Dimension();
10    // Loop over all possible co-dimensions of sub-entities
11    for (dim_t sub_codim = 1; sub_codim <= dimension; ++sub_codim) {
12        // Obtain array of sub-entities of co-dimension sub_codim

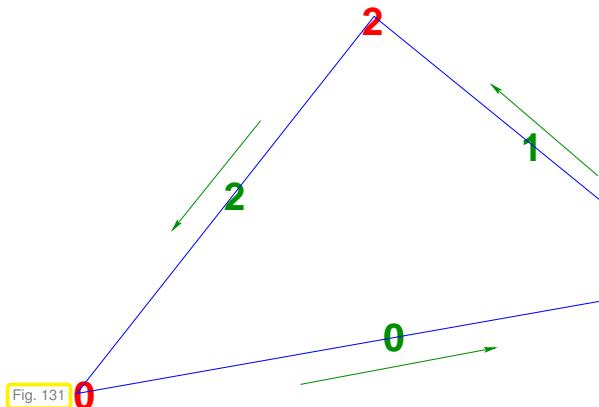
```

```

13     nonstd::span<const If<mesh::Entity *const> sub_ent_array{
14         e.SubEntities(sub_codim)};
15     // Query number of sub-entities
16     const size_type num_subent = ref_el.NumSubEntities(sub_codim);
17     // Index-based loop over sub-entities
18     for (int sub_ent_idx = 0; sub_ent_idx < num_subent; ++sub_ent_idx) {
19         // Test whether relative dimension matches absolute dimensions
20         EXPECT_EQ(sub_codim,
21             dimension - sub_ent_array[sub_ent_idx]->RefEl().Dimension());
22         << "Dimension mismatch: " << e << " <-> subent(" << sub_ent_idx << ")");
23         << std::endl;
24     }
25 }
26 }
```

Note the access to the sub-entity objects via the `[]`-operator.

§2.7.2.14 (Local numbering of cell sub-entities in LEHRFEM++) As demonstrated in Code 2.7.2.13 the sub-entities returned from `SubEntities()` can be accessed through the `[]` operator using their **local index**, thus defining the local numbering of the sub-entities of an entity. This local numbering starts from 0 and must be fixed by *convention*.



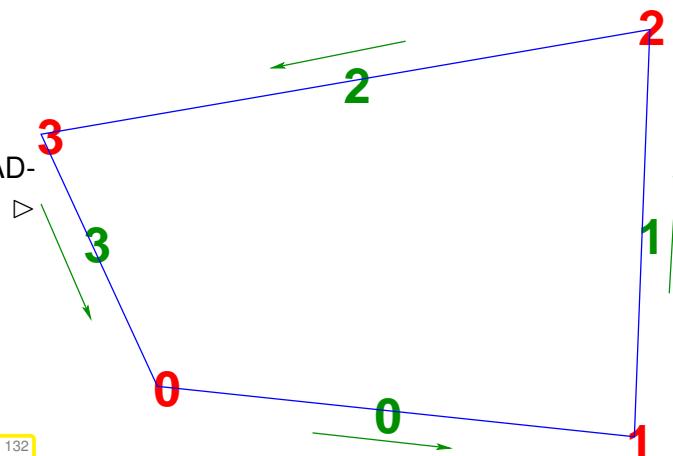
The convention adopted by LEHRFEM++ for setting the local indices of the edges of a TRIA entity are illustrated in Fig. 131

(red \leftrightarrow node numbers, green \leftrightarrow edge numbers).

This means that, for instance, the EDGES with local numbers 0 and 1 are guaranteed to share the NODE with local number 1.

LEHRFEM++'s numbering convention for QUAD-type entities is given beside

(red \leftrightarrow node numbers, green \leftrightarrow edge numbers).



Refer to [→ GITHUB](#) for a unit test verifying the fulfillment of local conventions on topology.

Remark 2.7.2.15 (Orientation of sub-entities) In the figures Fig. 131 and Fig. 132 the green arrows indicate the convention about the **local** orientation of the edges of a cell, which are always supposed to point from the vertex with the smaller index to the vertex with the larger index. Distinguish from **global** orientation, which is intrinsic to an edge and fixed arbitrarily during the construction of a mesh.

§2.7.2.16 (Global mesh topology) When we mention the “(global) topology” of a mesh we have in mind the **adjacency relations** among mesh entities, while completely ignoring their location and shape. We are

interested in the

incidence relations: ♦ “boundary contains”-relation: boundary of an entity of a higher dimension contains entity of a lower dimension, a so-called **sub-entity**.

♦ “is part of”-relation: entity of lower dimension is part of the boundary of an entity of a higher dimension, a **super-entity**.

► The incidence relations yield an abstract (generalized) **graph description** of a finite element mesh

Possible internal realization of incidence relations:

- **Dynamic, distributed objects**: for some $(j, k) \in \{0, \dots, d\}^2$, $0 \leq j < k \leq d$, entities of dimension k hold **ordered lists** (vectors) of references (**pointers**) to those entities of dimension j contained in their boundary. This is the storage scheme in LEHRFEM++.
- **Array-based**: for some $(m, n) \in \{0, \dots, d\}^2$, $0 \leq m < n \leq d$, the code stores matrices of indices, with rows corresponding to entities of co-dimension m , and columns to entities of co-dimension n . The matrix entries supply unique identifiers for sub-entities. This is the storage scheme used in the code of Section 2.4, see Code 2.4.1.2.

EXAMPLE 2.7.2.17 (Storing topology of triangular mesh in 2D) Let \mathcal{M} be a triangular mesh according to Def. 2.5.1.1 as in Section 2.4.1. Various schemes for storing topological information are conceivable. In the figures below: black \leftrightarrow triangles, blue \leftrightarrow edges, red \leftrightarrow vertices.

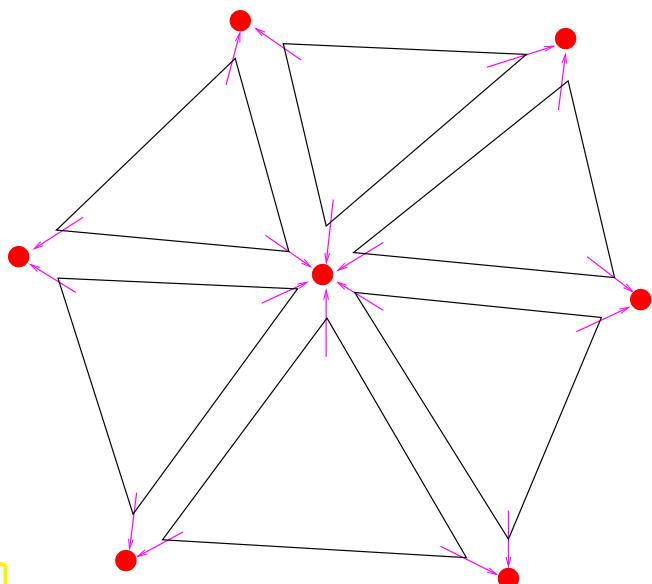


Fig. 133

(A) Minimal scheme: triangles hold lists/vectors of references to their vertices. Edges not stored.

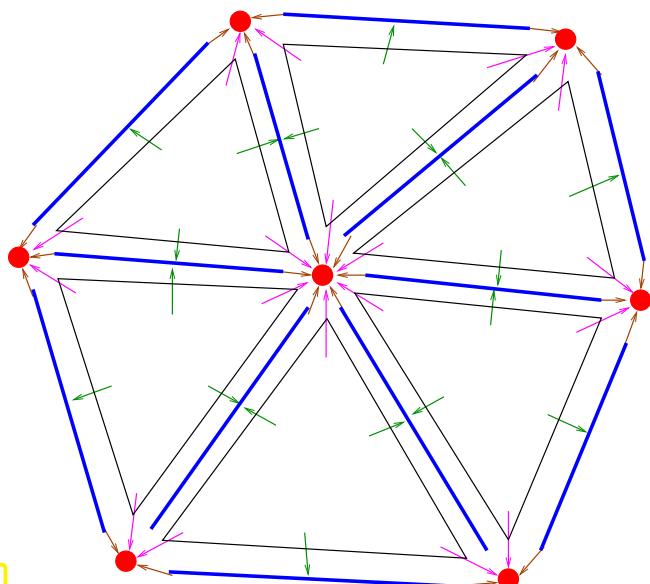
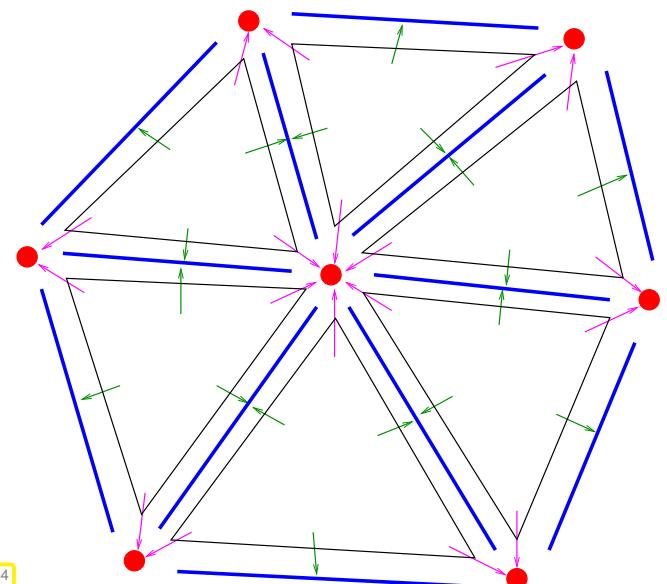
Realized in the mesh data structure discussed in Section 2.4.1, see the class **TriaMesh2D** listed in Code 2.4.1.2. This is sufficient for linear Lagrangian finite elements, if no special boundary conditions have to be dealt with.

Note that this scheme already provides **complete topological information** (edges can be reconstructed)!

This scheme is used by **MFEM**.

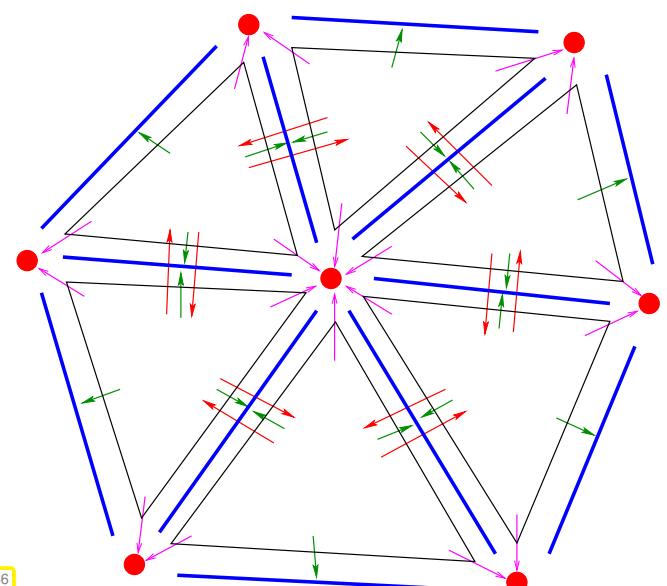
(B) Element centered scheme with edges:

- ◆ Elements, edges and vertices stored as (virtual) “objects”
- ◆ Elements have lists/vectors of references to their vertices and edges.

**(C) Full unidirectional topology representation:**

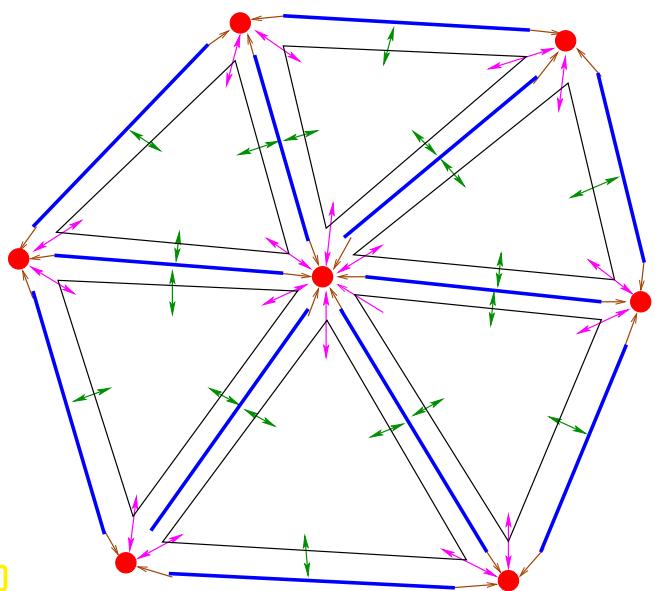
- ◆ All geometric entities are stored as (virtual) “objects”.
- ◆ Elements hold lists/vectors of references to their vertices and edges.
- ◆ Edges have references to their endpoints.

► Topology representation in LEHRFEM++

**(D) Restricted bidirectional topology representation:**

- ◆ All geometric entities are stored as (virtual) “objects”.
- ◆ Elements hold vectors of references to their vertices and edges.
- ◆ Elements also possess a vector of references to their neighbors.

Fig. 137

**(E) Full bidirectional topology representation:**

- ◆ Elements hold vectors of references to their vertices and edges.
- ◆ Edges have references to their endpoints and their adjacent triangles
- ◆ Vertices have references to their adjacent triangles.

Notation: \mathcal{M} = mesh (set of elements = set of geometric entities of co-dimension 0)

$\mathcal{V}(\mathcal{M})$ = set of nodes (vertices) in \mathcal{M} (geometric entities of co-dimension 2)

$\mathcal{E}(\mathcal{M})$ = set of edges in \mathcal{M} (geometric entities of co-dimension 1)

Global topology of the mesh is completely, and even redundantly, encoded by the mesh entities' `SubEntities()` member function that we have already seen above. □

EXAMPLE 2.7.2.18 (Inspecting mesh topology in LEHRFEM++) The following code demonstrates the access to sub-entities and the retrieval of their indices.

C++ code 2.7.2.19: Printing topology information for a mesh in LEHRFEM++ → GITHUB

```

2 void scanTopology(const If::mesh::Mesh &mesh, dim_t codim) {
3     LF_ASSERT_MSG((codim <= mesh.DimMesh()), 
4         "codim " << +codim << " too large");
5     // loop over all entities of the specified codimension
6     for (const If::mesh::Entity* ent : mesh.Entities(codim)) {
7         // Fetch topology type (TRIA or QUAD so far)
8         const If::base::RefEl ref_el{ent->RefEl()};
9         // Print topological type and global index of the ent
10        const glb_idx_t ent_idx = mesh.Index(*ent);
11        std::cout << ref_el << ": idx = " << ent_idx << std::endl;
12        // Inspect sub-entities of any co-dimension
13        for (dim_t sub_codim = 1; sub_codim <= mesh.DimMesh() - codim;
14            ++sub_codim) {
15            // Obtain iterator over sub-entities
16            nonstd::span<const If::mesh::Entity* const>
17            sub_ent_range { ent->SubEntities(sub_codim) };
18            size_type sub_cnt = 0; // Counter for sub-entities
19            // Loop over sub-entities, whose types and indices will be output
20            for (const If::mesh::Entity* subent : sub_ent_range) { // 
21                std::cout << "\t rel. codim " << +sub_codim << " sub-ent "
22                    << sub_cnt << ": " << *subent << ", idx = "
23                    << mesh.Index(*subent) << std::endl;
24                sub_cnt++;
25            }
}

```

The loop of Line 20 could be replaced with an index-based loop as in Code 2.7.2.13. □

EXAMPLE 2.7.2.20 (Indexing and numbering of (sub-)entities in LEHRFEM++) We report the output of

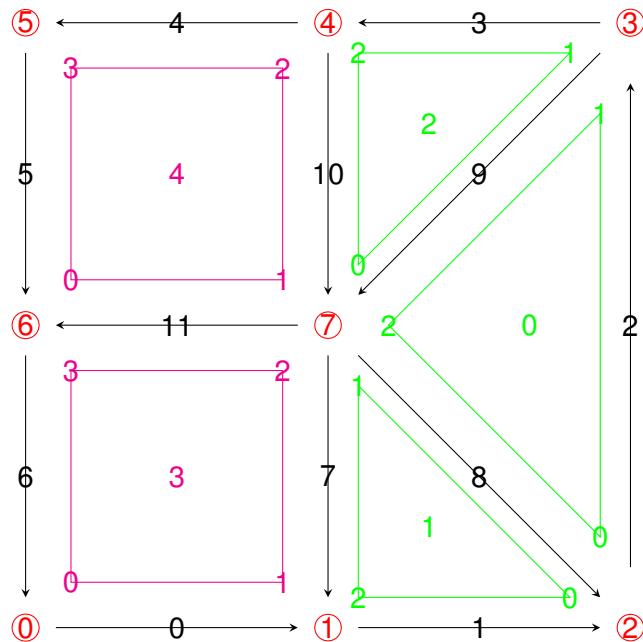
a LEHRFEM++ demo program → [GITHUB](#) (invoked with `-d 1` option) that reads a mesh in GMSH format from a `.msh`-file and prints information calling the `scanTopology()` function from Code 2.7.2.19.

On the left we show the `.msh`-file (→ Ex. 2.7.1.5) created by **Gmsh** describing the simple planar hybrid mesh comprising triangular (in green) and rectangular (in magenta) cells drawn on the right

```

1 $MeshFormat
2 2.2 0 8
3 $EndMeshFormat
4 $Nodes
5 8
6 1 0 0 0
7 2 1 0 0
8 3 2 0 0
9 4 2 2 0
10 5 1 2 0
11 6 0 2 0
12 7 0 1 0
13 8 1 1 0
14 $EndNodes
15 $Elements
16 16
17 1 15 2 5 1 1
18 2 15 2 5 3 3
19 3 15 2 5 4 4
20 4 15 2 5 6 6
21 5 1 2 1 1 1 2
22 6 1 2 1 2 2 3
23 7 1 2 2 3 3 4
24 8 1 2 3 4 4 5
25 9 1 2 3 5 5 6
26 10 1 2 4 6 6 7
27 11 1 2 4 6 7 1
28 12 2 2 6 2 8 4 5
29 13 2 2 6 2 3 8 2
30 14 2 2 6 2 3 4 8
31 15 3 2 7 1 5 6 7 8
32 16 3 2 7 1 8 7 1 2
33 $EndElements

```



Annotations:

- $0, \dots \hat{=} \text{global indices of vertices},$
- $0, \dots \hat{=} \text{global indices of edges},$
- cells and cell indices drawn in **magenta** and **green**, local indices of nodes placed at cell corners.

The rendering of the mesh was produced by the function `lf::io::writeTikZ()` → [GITHUB](#). The following output is produced by `scanTopology()` from Code 2.7.2.19.

<pre> 1 TRIA: idx = 0 2 rel. codim 1 sub-ent 0: EDGE, idx = 2 3 rel. codim 1 sub-ent 1: EDGE, idx = 9 4 rel. codim 1 sub-ent 2: EDGE, idx = 8 5 rel. codim 2 sub-ent 0: NODE, idx = 2 6 rel. codim 2 sub-ent 1: NODE, idx = 3 7 rel. codim 2 sub-ent 2: NODE, idx = 7 8 TRIA: idx = 1 9 rel. codim 1 sub-ent 0: EDGE, idx = 8 10 rel. codim 1 sub-ent 1: EDGE, idx = 7 11 rel. codim 1 sub-ent 2: EDGE, idx = 1 12 rel. codim 2 sub-ent 0: NODE, idx = 2 13 rel. codim 2 sub-ent 1: NODE, idx = 7 14 rel. codim 2 sub-ent 2: NODE, idx = 1 15 TRIA: idx = 2 16 rel. codim 1 sub-ent 0: EDGE, idx = 9 17 rel. codim 1 sub-ent 1: EDGE, idx = 3 18 rel. codim 1 sub-ent 2: EDGE, idx = 10 19 rel. codim 2 sub-ent 0: NODE, idx = 7 20 rel. codim 2 sub-ent 1: NODE, idx = 3 21 rel. codim 2 sub-ent 2: NODE, idx = 4 22 QUAD: idx = 3 23 rel. codim 1 sub-ent 0: EDGE, idx = 0 24 rel. codim 1 sub-ent 1: EDGE, idx = 7 25 rel. codim 1 sub-ent 2: EDGE, idx = 11 26 rel. codim 1 sub-ent 3: EDGE, idx = 6 27 rel. codim 2 sub-ent 0: NODE, idx = 0 28 rel. codim 2 sub-ent 1: NODE, idx = 1 29 rel. codim 2 sub-ent 2: NODE, idx = 7 30 rel. codim 2 sub-ent 3: NODE, idx = 6 31 QUAD: idx = 4 32 rel. codim 1 sub-ent 0: EDGE, idx = 11 33 rel. codim 1 sub-ent 1: EDGE, idx = 10 34 rel. codim 1 sub-ent 2: EDGE, idx = 4 35 rel. codim 1 sub-ent 3: EDGE, idx = 5 36 rel. codim 2 sub-ent 0: NODE, idx = 6 37 rel. codim 2 sub-ent 1: NODE, idx = 7 38 rel. codim 2 sub-ent 2: NODE, idx = 4 39 rel. codim 2 sub-ent 3: NODE, idx = 5 </pre>	<pre> 1 EDGE: idx = 0 2 rel. codim 1 sub-ent 0: NODE, idx = 0 3 rel. codim 1 sub-ent 1: NODE, idx = 1 4 EDGE: idx = 6 5 rel. codim 1 sub-ent 0: NODE, idx = 6 6 rel. codim 1 sub-ent 1: NODE, idx = 0 7 EDGE: idx = 1 8 rel. codim 1 sub-ent 0: NODE, idx = 1 9 rel. codim 1 sub-ent 1: NODE, idx = 2 10 EDGE: idx = 7 11 rel. codim 1 sub-ent 0: NODE, idx = 7 12 rel. codim 1 sub-ent 1: NODE, idx = 1 13 EDGE: idx = 2 14 rel. codim 1 sub-ent 0: NODE, idx = 2 15 rel. codim 1 sub-ent 1: NODE, idx = 3 16 EDGE: idx = 8 17 rel. codim 1 sub-ent 0: NODE, idx = 7 18 rel. codim 1 sub-ent 1: NODE, idx = 2 19 EDGE: idx = 3 20 rel. codim 1 sub-ent 0: NODE, idx = 3 21 rel. codim 1 sub-ent 1: NODE, idx = 4 22 EDGE: idx = 9 23 rel. codim 1 sub-ent 0: NODE, idx = 3 24 rel. codim 1 sub-ent 1: NODE, idx = 7 25 EDGE: idx = 4 26 rel. codim 1 sub-ent 0: NODE, idx = 4 27 rel. codim 1 sub-ent 1: NODE, idx = 5 28 EDGE: idx = 10 29 rel. codim 1 sub-ent 0: NODE, idx = 4 30 rel. codim 1 sub-ent 1: NODE, idx = 7 31 EDGE: idx = 5 32 rel. codim 1 sub-ent 0: NODE, idx = 5 33 rel. codim 1 sub-ent 1: NODE, idx = 6 34 EDGE: idx = 11 35 rel. codim 1 sub-ent 0: NODE, idx = 7 36 rel. codim 1 sub-ent 1: NODE, idx = 6 </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2.7.2.3 LEHRFEM++ Mesh: Geometry Layer

LEHRFEM++ stores the shape of any entity in objects derived from the interface class **If::geometry::Geometry** ([→ \(documentation\)](#)). A pointer to an object of this type can be obtained by calling the following member function of **If::mesh::Entity**:

```
lf::geometry::Geometry *lf::mesh::Entity::Geometry () const;
```

The discussion of **If::geometry::Geometry** will be postponed until we have introduced the technique of parametric finite elements.

§2.7.2.21 (Coordinates in LEHRFEM++) LEHRFEM++ encodes all geometric information by vectors of Cartesian coordinates of points in Euclidean space \mathbb{R}^2 .

In LEHRFEM++ coordinate vectors are small *fixed size EIGEN vector types*, see [Hip19, ??] and Section 2.7.3. Thus, all of EIGEN's linear algebra operations and functions are available for them.

Often, LEHRFEM++ stores the coordinates of several points packed into the *columns* of a matrix: the positions of $n \in \mathbb{N}$ points $\in \mathbb{R}^d$ are held in an $n \times d$ matrix. □

At this point the main method for learning the shape of an entity is the function

```
Eigen::MatrixXd
lf::geometry::Corners (const lf::geometry::Geometry& geo);
```

It returns the Cartesian coordinates of the corner points of a (geometric) entity as the columns of a matrix. For instance, if the shape of the entity is a 2D triangle with vertices $\mathbf{a}^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}$, $\mathbf{a}^2 = \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix}$, and $\mathbf{a}^3 := \begin{bmatrix} a_1^3 \\ a_2^3 \end{bmatrix}$, then this matrix reads

$$\begin{bmatrix} a_1^1 & a_1^2 & a_1^3 \\ a_2^1 & a_2^2 & a_2^3 \end{bmatrix} \in \mathbb{R}^{2,3}.$$

EXAMPLE 2.7.2.22 (Accessing corner coordinates in LEHRFEM++) This sample code shows the actual use of the function `lf::geometry::Corners()` to obtain information about the geometry of mesh entities.

C++ code 2.7.2.23: Output of locations of entity corners → [GITHUB](#)

```
2 void PrintGeometryInfo(const If::mesh::Mesh &mesh, dim_t codim) {
3     LF_ASSERT_MSG((codim <= mesh.DimMesh()), 
4         "codim " << +codim << " too large");
5     // loop over all entities of the specified codimension
6     for (const If::mesh::Entity* ent : mesh.Entities(codim)) {
7         // Number of nodes = number of corner points
8         const size_type num_nodes = ent->RefEl().NumNodes();
9         // Obtain pointer to geometry object associated with entity
10        const If::geometry::Geometry *geo_ptr = ent->Geometry();
11        LF_ASSERT_MSG(geo_ptr != nullptr, "Missing geometry!");
12        // Fetch coordinates of corner points in packed format § 2.7.2.21
13        Eigen::MatrixXd corners = lf::geometry::Corners(*geo_ptr);
14        LF_ASSERT_MSG(corners.rows() == geo_ptr->DimGlobal(),
15            "dimension mismatch for coordinate vectors");
16        LF_ASSERT_MSG(corners.cols() == num_nodes, "#corners mismatch");
17        std::cout << ent->RefEl() << "(" << mesh.Index(*ent) << ")" pts: ";
18        for (int i = 0; i < num_nodes; ++i) {
```

```

19     std::cout << I << " =[ " << corners.col(I).transpose() << "] , ";
20 }
21 std::cout << std::endl;
22 }
```

Review question(s) 2.7.2.24 (Mesh data structures)

(Q2.7.2.24.A) Give concise descriptions of the three fundamental aspects of information contained in a finite element mesh:

- container aspect,
- topological information,
- geometric information

(Q2.7.2.24.B) [The `Index()` method] Explain the meaning of the following set identity:

$$\{ \text{mesh.Index}(e) : e \in \text{mesh.Entities}(\text{codim}) \} = \{0, \dots, \text{mesh.NumEntities}(\text{codim}) - 1\}, \quad (2.7.2.5)$$

where `mesh` is a immutable reference to an object of type `If::mesh::Mesh`.

(Q2.7.2.24.C) [An index invariant] The two methods of `If::mesh::Mesh`

- `lf::mesh::Mesh::Index()`,
- and `lf::mesh::Mesh::EntityByIndex()`

realize inverse operations. Devise three lines of C++ code that test this statement for a given `If::mesh::Mesh` object.

(Q2.7.2.24.D) [Sub-entities] What is the value of the following expression

`e.SubEntities(rel_codim).size()`

depending on the topological type of `e` and the argument `rel_codim`? Here, `e` is a reference to an `If::mesh::Entity` object of a 2D mesh, and `rel_codim` an integer $\in \{0, 1, 2\}$.

(Q2.7.2.24.E) [Cells adjacent to edges] Given a reference `mesh` to an immutable `If::mesh::Mesh` object describing a planar 2D hybrid mesh, design a C++ code snippet that initializes

```
lf::mesh::utils::CodimMeshDataSet<std::array<const
lf::mesh::Entity *, 2>> cell_at_edge;
```

so that it contains, for each edge of the mesh, pointers to all adjacent cells.

(Q2.7.2.24.F) [Orienting cells] For a planar mesh the **orientation** of a cell is defined through the numbering of its vertices as “clockwise” or “counterclockwise” sense of moving around the cell. Using the facilities of LEHRFEM++, in particular the method `lf::mesh::Entity::RelativeOrientation()`, outline an algorithm for initializing an object `rel_cell_ori` of type `If::mesh::utils::CodimMeshDataSet<int>` such that

$$\text{rel_cell_ori}(e) = \begin{cases} 1 & \text{, if the orientation of } e \text{ agrees with that of } e_0 \\ -1 & \text{otherwise,} \end{cases}$$

where `e0` is the cell of the mesh with index = 0.

(Q2.7.2.24.G) [EntityByIndex()] What is the purpose of the following code snippet?

```
((idx < mesh.NumEntities(codim)) &&
mesh.Index(*mesh.EntityByIndex(codim, idx)) == idx)
```

What is an appropriate type of `idx`?

△

2.7.3 Vectors and Matrices

§2.7.3.1 (Functions of vectors and matrices in a FE code) Data structures representing matrices and vectors serve different important purposes in a finite element code:

- ❶ They represent **coordinate vectors** of points and have to support geometric calculations.
- ❷ They store element matrices and element vectors, recall Section 2.4.5, Section 2.4.6, and see Def. 2.7.4.5 below.
- ❸ They are needed for handling the Galerkin matrices and have to be used by the linear (direct or iterative) solver.

For ❶ small **fixed size** vectors and matrices are sufficient, and they may also be used for ❷, if the mesh consists of a single type of cells only. For ❸ we need data structures suitable for large **variable size** vectors and matrices, where the latter are **sparse**, moreover, see Section 2.4.4. □

§2.7.3.2 (EIGEN – A C++ template library for numerical linear algebra) C++ relies on the open source software EIGEN for its numerical linear algebra needs.

EIGEN is a C++ template library designed to enable easy, natural and efficient numerical linear algebra: it provides data structures and a wide range of operations for matrices and vectors, see below. EIGEN also implements (→ [doc](#))

- all important matrix decompositions of dense numerical linear algebra (LU-, QR-, Cholesky-decompositions) and direct solvers based on them,
- “direct” eigensolvers for various types of dense eigenvalue problems,
- the singular value decomposition (SVD) of a matrix,
- ranks, determinants and inverses of matrices.

Eigen relies on **expression templates** to allow the efficient evaluation of complex expressions involving matrices and vectors. Refer to the [example](#) given in the EIGEN documentation for details.

The principal components and capabilities of the EIGEN library have been covered in the course “Numerical Methods for Computational Science and Engineering” [[Hip19](#), ??].

In LEHRFEM++ *all* matrices and vectors are objects of a suitable **Eigen::(Sparse)Matrix** type.

□

§2.7.3.3 (LEHRFEM++ triplet-based sparse matrix format) In Rem. 2.4.5.25 we saw that sparse Galerkin matrices can be initialized efficiently using an intermediate triplet-based format, also called **COO format**.

LEHRFEM++ implements a dedicated type **If::assemble::COOMatrix** → [GITHUB](#) to manage data in COO format. The main member functions are

- Eigen::Index lf::assemble::COOMatrix::rows() const;
Eigen::Index lf::assemble::COOMatrix::cols() const;

telling the size of the matrix,

- **void** lf::assemble::COOMatrix::AddToEntry(gdof_idx_t i,
gdof_idx_t j, SCALAR increment);

adding the value increment to the matrix entry at position (i, j) ,

- **std::vector**<Eigen::Triplet<SCALAR>> &triplets();

gives access to the internal vector of triplets.

↳

§2.7.3.4 (EIGEN: some pointers to information) Comprehensive information about EIGEN is available online:

- ◆ Matrix and vector data types in EIGEN: see [Hip19, ??] and [documentation](#).
- ◆ Initialization of *dense* matrices in EIGEN: see [Hip19, ??].
- ◆ Access to submatrices in EIGEN: see [Hip19, ??] and [documentation](#).
- ◆ Componentwise operations in EIGEN: see [Hip19, ??] and [documentation](#).
- ◆ Sparse matrices in EIGEN (CRS/CCS-format): see [Hip19, ??] and [documentation](#); already used in Code 2.4.5.24.

↳

2.7.4 Assembly Algorithms



Video tutorial for Section 2.7.4: Assembly Algorithms: (46 minutes) [Download link](#), [tablet notes](#)

In English “assemble” means “fit pieces together”, the meaning in FEM is similar:

“Assembly” = term used for computing entries of stiffness matrix/right hand side vector (load vector) in a finite element context, *cf.* § 2.4.5.15.

Aspects of assembly for linear Lagrangian finite elements ($V_{0,h} = \mathcal{S}_{1,0}^0(\mathcal{M})$) were discussed in Section 2.4.5 and Section 2.4.6. (Refresh yourself on these sections in case you cannot remember the main ideas behind building the Galerkin matrix and right hand side vector.)

2.7.4.1 Assembly: Localization

Cell-local concepts and operations play a key role in the efficient initialization of finite element Galerkin matrices and right hand side vectors.

§2.7.4.1 (Localized (bi-)linear forms in variational formulations) We consider a discrete variational problem ($V_{0,h}$ = FE space, $\dim V_{0,h} = N \in \mathbb{N}$, see (2.2.1.1))

$$u_h \in V_{0,h}: \quad a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h}. \quad (2.2.1.1)$$

Let $\mathcal{B} = \{b_h^1, \dots, b_h^N\}$ designate the set of locally supported basis functions of $V_{0,h}$ to be used in the computations (\rightarrow Section 2.5.3). Then we have to compute (see also Section 2.4.5 and Section 2.4.6) the following algebraic information, the

- Galerkin matrix (stiffness matrix): $\mathbf{A} = \left[a(b_h^j, b_h^i) \right]_{i,j=1}^N \in \mathbb{R}^{N,N}$,
- and the r.h.s. vector (load vector): $\vec{\phi} := \left[\ell(b_h^i) \right]_{i=1}^N \in \mathbb{R}^N$.

both can be written in terms of *local cell contributions*, since usually

$$a(u, v) = \sum_{K \in \mathcal{M}} a_K(u|_K, v|_K) , \quad \ell(v) = \sum_{K \in \mathcal{M}} \ell_K(v|_K) , \quad (2.7.4.2)$$

where $\cdot|_K$ designates the restriction of a function to cell K ; $u|_K$ and $v|_K$ are not defined outside K .

Example: bilinear forms/linear forms arising from 2nd-order elliptic BVPs, e.g., (1.9.0.2), (1.9.0.3), (1.9.0.4), can be localized in straightforward fashion by restricting integration to mesh cells (\rightarrow Rem. 2.4.2.3): for $u, v \in H^1(\Omega)$

$$a(u, v) := \int_{\Omega} \alpha(x) \operatorname{grad} u \cdot \operatorname{grad} v \, dx = \sum_{K \in \mathcal{M}} \underbrace{\int_K \alpha(x) \operatorname{grad} u \cdot \operatorname{grad} v \, dx}_{=: a_K(u|_K, v|_K)} , \quad (2.7.4.3)$$

$$\ell(v) := \int_{\Omega} f v \, dx = \sum_{K \in \mathcal{M}} \underbrace{\int_K f v \, dx}_{=: \ell_K(v|_K)} . \quad (2.7.4.4)$$

Recall (2.5.3.5): Restrictions of global shape functions to entities = local shape functions

Definition 2.7.4.5. Element (stiffness) matrix and element (load) vector

Given a mesh entity $K \in \mathcal{M}$ and local shape functions $\{b_K^1, \dots, b_K^Q\}$, $Q = Q(K) \in \mathbb{N}$, we introduce

the **element (stiffness) matrix** $\mathbf{A}_K := \left[a_K(b_K^j, b_K^i) \right]_{i,j=1}^Q \in \mathbb{R}^{Q,Q}$,
and the **element (load) vector** $\vec{\phi}_K := \left[\ell_K(b_K^i) \right]_{i=1}^Q \in \mathbb{R}^Q$.

§2.7.4.6 (Numbers of local shape functions) In Def. 2.7.4.5: Q = the number of local shape functions on element $K \in \mathcal{M}$, may be different for different mesh cells K . For instance, this occurs

- ◆ in the case of hybrid meshes as discussed in Rem. 2.6.2.10, e.g., $Q(K) \in \{3, 4\}$ for linear Lagrangian finite element spaces $S_1^0(\mathcal{M})$.
- ◆ in the case of enforcement of zero essential boundary conditions by dropping basis functions associated with interpolation nodes on $\partial\Omega$, as explained in § 2.6.2.8: according to the formal definition Def. 2.5.3.4 this will lead to a reduced number of local shape functions. However, in implementations zero essential boundary conditions are handled differently, see Section 2.7.6.

For standard Lagrangian finite element spaces $\mathcal{S}_p^0(\mathcal{M})$ the dimensions of the spaces spanned by local shape functions are the same for all mesh cells and given by the following formulas:

Type of FE space	Q
degree p Lagrangian FE on <i>triangular</i> mesh	$\rightarrow \dim \mathcal{P}_p(\mathbb{R}^2) = \frac{1}{2}(p+1)(p+2)$
degree p Lagrangian FE on <i>tetrahedral</i> mesh	$\rightarrow \dim \mathcal{P}_p(\mathbb{R}^3) = \frac{1}{6}(p+1)(p+2)(p+3)$
degree p Lagrangian FE on <i>tensor product</i> mesh in 2D	$\rightarrow \dim \mathcal{Q}_p(\mathbb{R}^2) = (p+1)^2$

We arrive at these formulas, by the following considerations:

- ◆ For Lagrangian finite element spaces the local shape functions span a polynomial space, either $\mathcal{P}_p(\mathbb{R}^d)$ (simplicial mesh) or $\mathcal{Q}_p(\mathbb{R}^d)$ (tensor product mesh).
 - ◆ The dimensions of $\mathcal{P}_p(\mathbb{R}^d)/\mathcal{Q}_p(\mathbb{R}^d)$ are given in Lemma 2.5.2.5 and Lemma 2.5.2.8.

Lemma 2.5.2.5. Dimension of spaces of polynomials

$$\dim \mathcal{P}_p(\mathbb{R}^d) = \binom{d+p}{p} \quad \text{for all } p \in \mathbb{N}_0, d \in \mathbb{N}$$

Lemma 2.5.2.8. Dimension of spaces of tensor product polynomials

$$\dim \mathcal{Q}_p(\mathbb{R}^d) = (p+1)^d \quad \text{for all } p \in \mathbb{N}_0, d \in \mathbb{N}$$

2.7.4.2 Assembly: Index Mappings

What we have discovered in the case of linear finite elements in Section 2.4.5 (conveyed in Fig. 77 and Fig. 78 and the accompanying remarks) and implemented in Code 2.4.5.23 is a general principle.

We find that in the (not so special) setting of this section, characterized by the possibility to localize the bilinear form a and right hand side linear form ℓ in the sense of (2.7.4.2),

- ◆ the entries of the finite element Galerkin matrix can be obtained by summing *corresponding* entries of *some* element matrices,
 - ◆ this corresponding entry of an element matrices is determined by the unique association of a local basis function to a global basis function (through a “d.o.f. mapper”/“d.o.f. handler”).

§2.7.4.7 (Abstract “d.o.f. mapper” facility) The correct assignment of local contributions to entries of the Galerkin matrix and the right hand side vector requires a

☞ Local → global index map (“d.o.f. mapper”/“d.o.f. handler”)

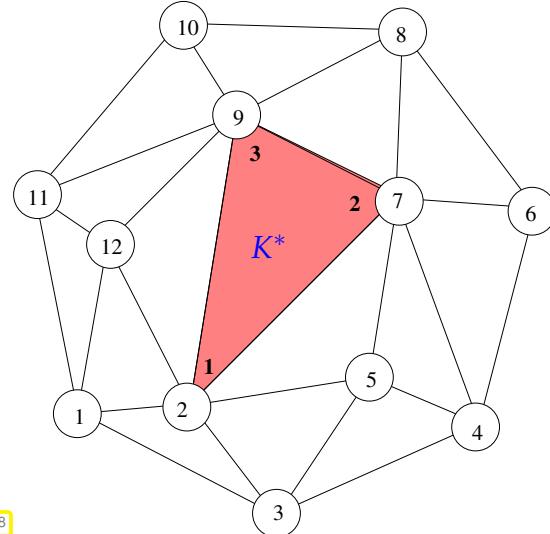
We point out that according to Def. 2.5.3.4 every entity K of a finite element mesh is endowed with a set $\{b_K^1, \dots, b_K^{Q(K)}\}$ of local shape functions, not only cells.

Remark 2.7.4.9 (Local→global index mapping and index array) The mapping `locglobmap` generalizes the device of the index mapping array `dofh` introduced in (2.4.5.21) on Page 178 for linear Lagrangian finite elements on 2D triangular meshes and also used in Code 2.4.5.23: Precisely, they are related by

$$\text{dofh}(k, l) = \text{locglobmap}(K, l), \quad \text{if } K \text{ has index } k, \quad l \in \{1, 2, 3\}.$$

(Mathematical indexing starting from 1 is used!) □

EXAMPLE 2.7.4.10 (Local→global mapping for linear Lagrangian finite elements on triangular mesh) This example refreshes § 2.4.5.20.



Using the local/global numbering indicated in the figure to the right the local→global index map for the marked cells yields

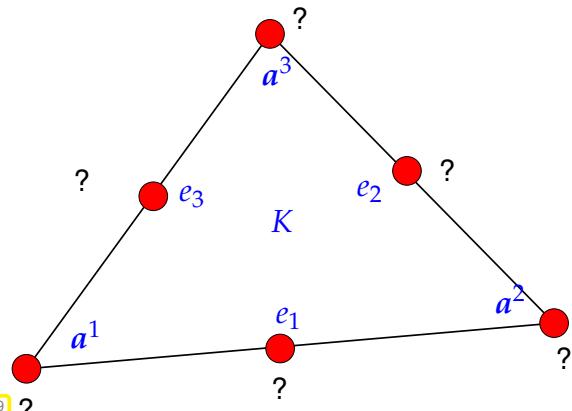
$$\begin{aligned}\text{locglobmap}(K^*, 0) &= 2, \\ \text{locglobmap}(K^*, 1) &= 7, \\ \text{locglobmap}(K^*, 2) &= 9.\end{aligned}$$

See also Fig. 80 for similar considerations.

§2.7.4.11 (Ordering convention for local shape functions in LEHRFEM++) In Ex. 2.7.4.10 it seemed natural that the local shape function (= barycentric coordinate function) associated with NODE # i , $i = 0, 1, 2$, had the number i . Yet, at second glance this turns out to be a mere convention.

To understand this better, consider quadratic Lagrangian finite elements, whose local shape functions on a triangle are associated with both vertices and edges, see (2.6.1.6). In this case it is clear that we have ample freedom to number the local shape functions – vertices first, then edges vs. edges first then vertices – and that any scheme is as good as any other.

However, in a code we have to fix a *convention* for numbering the local shape functions and we have to adhere to it throughout.



The following local numbering convention is universally applied in LEHRFEM++ for the cells of logically two-dimensional meshes:

- (I) The local shape functions of every cell (co-dimension-0 entity) are arranged according to *increasing dimension* of the geometric entities they are associated with:

POINT → **SEGMENT** → **{TRIA,QUAD}**.

- (II) Local shape functions belonging to geometric entities of the same dimension are ordered *according to* the intrinsic *local indexing* of those entities. See § 2.7.2.14 for LEHRFEM++’s conventions.
- (III) No ordering of local shape functions attached to the same geometric entity is implied. Here every user may follow her or his own conventions.

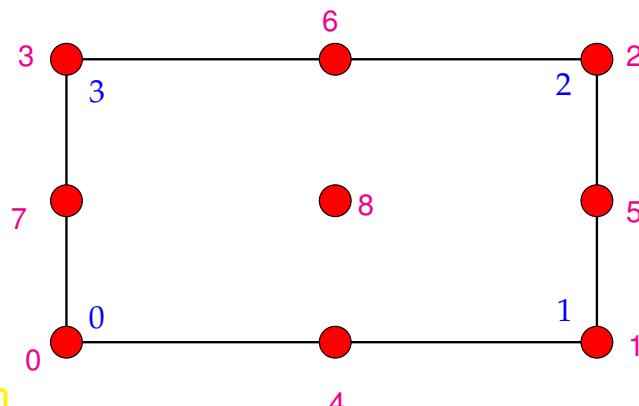
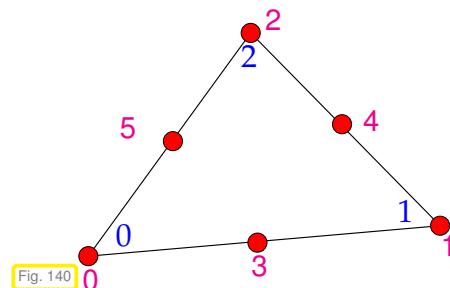
(LN)

EXAMPLE 2.7.4.12 (Numbering of local shape functions for quadratic Lagrangian finite elements)

Following the rules (LN) we find the following local numbering for the local shape functions for quadratic Lagrangian finite elements.

Magenta numbers give LEHRFEM++'s numbering *convention* applied to the **local shape functions** for triangular quadratic Lagrangian finite elements from (2.6.1.6). ▷

($0, \dots, 2 \hat{=} \text{local indices of nodes}$)



◁ Numering convention (LN) for local shape functions of $\mathcal{S}_2^0(\mathcal{M})$ on a quadrilateral.

($0, \dots, 3 \hat{=} \text{local indices of nodes}$)

Note: C++ indexing!

§2.7.4.13 (D.o.f. handler/d.o.f. mapper in LEHRFEM++) In LEHRFEM++ local→global index mappings in the spirit of the d.o.f. mapper function `loclglobmap` from (2.7.4.8) are managed by objects of the interface type **If::assemble::DofHandler** → ([documentation](#)), which supplies the following main methods:

- `size_type lf::assemble::DofHandler::NumDofs () const;`

which returns the total number of global basis functions, that is, the dimension of the finite element space.

- `size_type lf::assemble::DofHandler::NumLocalDofs (
 const lf::mesh::Entity &) const;`

which returns the number of global/local shape function *covering* any geometric entity of the mesh of **any** co-dimension.

- `nonstd::span<const lf::assemble::gdof_idx_t>
 lf::assemble::DofHandler::GlobalDofIndices (
 const lf::mesh::Entity &entity) const;`

which returns an array of index numbers $\in \{0, \dots, \text{NumDofs} () - 1\}$ for the global shape functions *covering* a particular entity of **any** dimension. The convention (LN) for the numbering of local shape functions applies.

- `size_type lf::assemble::DofHandler::NumInteriorDofs (
 const lf::mesh::Entity &) const;`

which tells us the number of global/local shape functions *associated with* a particular geometric entity of **any** co-dimension.

- `nonstd::span<const lf::assemble::gdof_idx_t>`
`lf::assemble::DofHandler::InteriorGlobalDofIndices (`
`const lf::mesh::Entity &entity) const;`

which provides the array of global indices $\in \{0, \dots, \text{NumDofs}() - 1\}$ of the global shape functions *associated with* a specific geometric entity of *any* co-dimension.

- `const lf::mesh::Entity &lf::assemble::DofHandler::Entity (`
`gdof_idx_t dofnum) const;`

which returns a reference to the unique geometric entity to which the global shape function with index `dofnum` is *associated*.

Internally a `lf::assemble::DofHandler` object maintains the information about the indices of global shape functions in array format. ↴

EXAMPLE 2.7.4.14 (Use of methods of `lf::assemble::DofHandler`) We examine a C++ function that prints comprehensive information about the local-to-global index mapping realized by a `lf::assemble::DofHandler` object.

C++ code 2.7.4.15: Listing of d.o.f. indexing → GITHUB

```

2 void printDofInfo(const lf::assemble::DofHandler &dofh) {
3     // Obtain pointer to the underlying mesh
4     auto mesh = dofh.Mesh();
5     // Number of degrees of freedom managed by the DofHandler object
6     const lf::assemble::size_type N_dofs(dofh.NumDofs());
7     std::cout << "DofHandler(" << dofh.NumDofs() << " dofs):" << std::endl;
8     // Output information about dofs for entities of all co-dimensions
9     for (lf::base::dim_t codim = 0; codim <= mesh->DimMesh(); codim++) {
10        // Visit all entities of a codimension codim
11        for (const lf::mesh::Entity* e : mesh->Entities(codim)) {
12            // Fetch unique index of current entity supplied by mesh object
13            const lf::base::glb_idx_t e_idx = mesh->Index(*e);
14            // Number of shape functions covering current entity
15            const lf::assemble::size_type no_dofs(dofh.NumLocalDofs(*e));
16            // Obtain global indices of those shape functions ...
17            nonstd::span<const lf::assemble::gdof_idx_t> dofarray{
18                dofh.GlobalDofIndices(*e)};
19                // and print them
20                std::cout << *e << ' ' << e_idx << ":" << no_dofs << " dofs = [ ";
21                for (int loc_dof_idx = 0; loc_dof_idx < no_dofs; ++loc_dof_idx) {
22                    std::cout << dofarray[loc_dof_idx] << ' ';
23                }
24                std::cout << ']';
25                // Also output indices of interior shape functions
26                nonstd::span<const lf::assemble::gdof_idx_t>
27                intdofarray{dofh.InteriorGlobalDofIndices(*e)};
28                std::cout << " int = [ ";
29                for (lf::assemble::gdof_idx_t int_dof : intdofarray) {
30                    std::cout << int_dof << ' ';
31                }
32                std::cout << ']' << std::endl;
33            }
34        }
35        // List entities associated with the dofs managed by the current
36        // DofHandler object
37        for (lf::assemble::gdof_idx_t dof_idx = 0; dof_idx < N_dofs; dof_idx++) {
38            const lf::mesh::Entity &e(dofh.Entity(dof_idx));
39            std::cout << "dof " << dof_idx << " -> " << e << " " << mesh->Index(e)

```

```

40           << std::endl;
41     }
42 } // end function printDofInfo

```

§2.7.4.16 (Initialization of d.o.f. handlers in L^HR_FE_M++) In order to set up a local-to-global index mapping for shape functions/degrees of freedom, it is necessary to specify the number of shape functions *associated with* every mesh entity. This is the information required for the most general implementation of the **If::assemble::DofHandler** interface in the class **If::assemble::DynamicFEDofHandler**, whose constructor

```

template <typename LOCALDOFINFO>
DynamicFEDofHandler(std::shared_ptr<const lf::mesh::Mesh> mesh_p,
LOCALDOFINFO && locdof)

```

has to be supplied with an object `locdof` featuring an evaluation operator

```
size_type operator()(const lf::mesh::Entity &);
```

which tells the number of shape functions *associated with* every entity of the mesh passed as `mesh_p` argument to the constructor.

A more specialized implementation is **If::assemble::UniformFEDofHandler** with constructor

```

using dof_map_t = std::map<lf::base::RefEl, base::size_type>;
UniformFEDofHandler(std::shared_ptr<const lf::mesh::Mesh> mesh_p,
dof_map_t dofmap);

```

Its `dofmap` argument is an associative array linking entity types to fixed numbers of global shape functions associated with them. This defines local-to-global index mappings for finite element spaces with the *same* arrangement of local shape functions for every cell like the Lagrangian finite element spaces introduced in Section 2.6. The constructor can be called as follows → [GITHUB](#):

```

lf::assemble::UniformFEDofHandler dof_handler(
mesh_p, {{lf::base::RefEl::kPoint(), ndof_node},
{lf::base::RefEl::kSegment(), ndof_edge},
{lf::base::RefEl::kTria(), ndof_tria},
{lf::base::RefEl::kQuad(), ndof_quad}});

```

where the variables `ndof_*` contain the numbers of global/local shape functions associated with the entity type in front of them. For the 2D finite element spaces $S_p^0(\mathcal{M})$ the table beside gives the numbers:

	$p = 1$	$p = 2$	$p = 3$
ndof_node	1	1	1
ndof_edge	0	1	2
ndof_tria	0	0	1
ndof_quad	0	1	4

Remark 2.7.4.17 (L^HR_FE_M++ numbering convention for global shape functions) In principle, as long as the numbering of global shape functions established by **If::assemble::DofHandler** is unique and consecutive from 0 to `NumDofs() - 1`, it implements a valid local-to-global index mapping.

In practice, it may be convenient to impose more structure on the indices and the current implementations of the **If::assemble::DofHandler** interface comply with the following rules:

- (I) D.o.f. associated with lower-dimensional entities are numbered first:

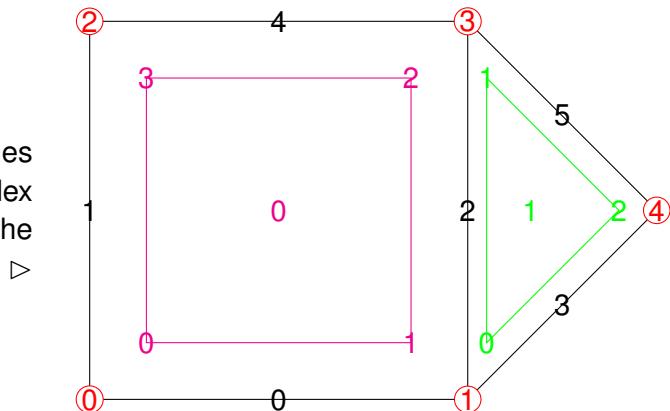
POINT → **SEGMENT** → **{TRIA,QUAD}**.

- (II) The indices of d.o.f. belonging to entities of the same co-dimension increase with increasing entity indices as returned by the `Index()` function.

Note that this reflects a *mere convention*. □

EXAMPLE 2.7.4.18 (Numbering of global shape functions) We give a concrete example of how a `If::assemble::DofHandler` object indexes the global shape functions.

We rely on a simple 2-cell hybrid mesh with 6 edges and 5 nodes displayed beside was used. The index numbers of all mesh entities are given along with the local numbers of corners.



The `If::assemble::DofHandler` studied in this example was initialized as follows:

```
lf::assemble::UniformFEDofHandler dof_handler(
mesh_p, {{lf::base::RefEl::kPoint(), 1},
{lf::base::RefEl::kSegment(), 2},
{lf::base::RefEl::kTria(), 1},
{lf::base::RefEl::kQuad(), 4}});
```

This means that every NODE carries one d.o.f., every EDGE two of them, every TRIA 1, and there are four global shape functions associated to every QUAD. Thus we compute

```
(dof_handler.NumDofs() == 22) == true;
```

Note that this arrangement of shape functions is characteristic for **cubic** Lagrangian finite elements $S_2^0(\mathcal{M})$, see Ex. 2.6.1.7 and Fig. 119.

Next, we list an excerpt of the output of the program in `examples/lecturedemo` → **GITLAB** listing the indices of global shape functions covering and associated with all the mesh entities (“dofs” $\hat{=}$ covering d.o.f.s, “int” $\hat{=}$ associated d.o.f.s):

```

1 QUAD 0: 16 dofs = [0 1 3 2 5 6 9 10 13 14 7 8 17 18 19 20]
2 int = [17 18 19 20]
3 TRIA 1: 10 dofs = [1 3 4 9 10 15 16 11 12 21 ] int = [21]
4 EDGE 0: 4 dofs = [0 1 5 6 ] int = [5 6]
5 EDGE 1: 4 dofs = [2 0 7 8 ] int = [7 8]
6 EDGE 2: 4 dofs = [1 3 9 10 ] int = [9 10]
7 EDGE 3: 4 dofs = [4 1 11 12 ] int = [11 12]
8 EDGE 4: 4 dofs = [3 2 13 14 ] int = [13 14]
9 EDGE 5: 4 dofs = [3 4 15 16 ] int = [15 16]
10 NODE 0: 1 dofs = [0 ] int = [0]
11 NODE 1: 1 dofs = [1 ] int = [1]
12 NODE 2: 1 dofs = [2 ] int = [2]
13 NODE 3: 1 dofs = [3 ] int = [3]
14 NODE 4: 1 dofs = [4 ] int = [4]
```

2.7.4.3 Distribute Assembly Schemes

§2.7.4.19 (Cell-oriented assembly of finite element Galerkin matrix and right hand side vector) Another fundamental design principle for the assembly realized already in Code 2.4.5.23 was to rely on

loops only over mesh cells combined with purely local operations.

The loops *distributed* entries of element matrices/vectors to the Galerkin matrix and right-hand-side vector, respectively.

Notion: local operations $\hat{=}$

- ◆ require data only from fixed “neighbourhood” of cell K
- ◆ computational effort “ $O(1)$ ”: independent of $\#M$

This design principle is honored in the following “pseudo-code” Code 2.7.4.20, which extends Code 2.4.5.23, which was confined to linear Lagrangian finite elements, to general finite element methods. The local→global index mapping is realized through the `locglobmap`-function/matrix. Prior knowledge about the dimension N of the finite element space is assumed and indexing from 1 is employed.

Pseudocode 2.7.4.20: Abstract assembly routine for finite element Galerkin matrices

```

1 Sparse Matrix  $\leftarrow$  assembleGalMat(Mesh  $M$ ) {
2    $A = N \times N$  sparse matrix; // Allocated zero sparse matrix
3   foreach  $K \in M$  { // loop over all cells
4      $Q_k = \text{no\_loc\_shape\_functions}(K);$ 
5     // Local operation: compute  $Q_k \times Q_k$  element matrix  $\rightarrow$  Def. 2.7.4.5,
6     // usually incurs cost of only " $O(1)$ "
7      $A_k = \text{getElementMatrix}(K);$ 
8     // Get vector of global indices (length  $Q_k$ );
9     // Usage of locglobmap as in Ex. 2.7.4.10
10    Vector idx = {locglobmap( $K, 1$ ), ..., locglobmap( $K, Q_k$ )};
11    // Add local contributions to global matrix
12    for i := 1 to  $Q_k$  {
13      for j := 1 to  $Q_k$  {
14        // Update entry of FE Galerkin matrix
15         $A(\text{idx}(i), \text{idx}(j)) += A_k(i, j);$ 
16      }
17    } // end main loop
18    return ( $A$ );
19 }
```

Note that in Code 2.4.5.24 the local→global index mapping could be inferred from the mesh data directly through the `Elements`-vector.

The very same ideas in a somewhat simpler version govern the initialization of the right hand side vector from element (load) vectors. The following MATLAB-style “pseudocode” Code 2.7.4.21 extends Code 2.4.6.8 and supplies a generic finite element assembly algorithm for right hand side vectors:

Pseudocode 2.7.4.21: Generic assembly algorithm for finite element right hand side vectors

```

1 Vector  $\leftarrow$  assembleRhsVector(Mesh  $M$ ) {
2    $\phi_i = \text{Vector}(N);$  // Allocate zero vector of appropriate length
3   foreach  $K \in M$  { // loop over all cells
```

```

4   // Obtain number  $Q(K)$  of local shape functions, see Def. 2.7.4.5
5   Qk = no_loc_shape_functions(K);
6   // Local operation: compute element vector, length  $Q_k \rightarrow$ 
7   // Def. 2.7.4.5,
8   // (usually incurs cost of only " $O(1)$ ")
9   phi_k = getElementVector(K);
10  // Get vector of global indices (length  $Q(K)$ );
11  // Usage of locglobmap as in Ex. 2.7.4.10
12  Vector idx = {locglobmap(K,1)),...,locglobmap(K,Qk)};
13  // Add local contributions to global right-hand-side vector
14  for i:=1 to Qk {
15      phi(idx(i)) += phi(idx(i)) + phi_k(i);
16  }
17  return(phi);
18 }
```

EXAMPLE 2.7.4.22 (Assembly of Galerkin matrices in LEHRFEM++) We list the complete code of the core matrix assembly function of LEHRFEM++:

C++ code 2.7.4.23: Assembly function of LEHRFEM++ → GITHUB

```

1 template <typename TMPMATRIX, class ENTITY_MATRIX_PROVIDER>
2 void AssembleMatrixLocally(dim_t codim, const DofHandler &dof_handler_trial,
3 const DofHandler &dof_handler_test,
4 ENTITY_MATRIX_PROVIDER &entity_matrix_provider,
5 TMPMATRIX &matrix) {
6     // Fetch pointer to underlying mesh
7     auto mesh = dof_handler_trial.Mesh();
8     // Central assembly loop over entities of co-dimension specified by
9     // the function argument codim
10    for (const If::mesh::Entity *entity : mesh->Entities(codim)) {
11        // Some entities may be skipped
12        if (entity_matrix_provider.isActive(*entity)) {
13            // Size, aka number of rows and columns, of element matrix
14            const size_type nrows_loc = dof_handler_test.NumLocalDofs(*entity);
15            const size_type ncols_loc = dof_handler_trial.NumLocalDofs(*entity);
16            // row indices of for contributions of cells
17            nonstd::span<const gdof_idx_t> row_idx(
18                dof_handler_test.GlobalDofIndices(*entity));
19            // Column indices of for contributions of cells
20            nonstd::span<const gdof_idx_t> col_idx(
21                dof_handler_trial.GlobalDofIndices(*entity));
22            // Request local matrix from ENTITY_MATRIX_PROVIDER object. In the
23            // case codim = 0, when entity is a cell,
24            // this is the element matrix
25            const auto elem_mat{entity_matrix_provider.Eval(*entity)}; //
26            // Assembly double loop over element matrix
27            for (int i = 0; i < nrows_loc; i++) {
28                for (int j = 0; j < ncols_loc; j++) {
29                    // Add the element at position (i,j) of the local matrix
30                    // to the entry at (row_idx[i], col_idx[j]) of the global
31                    // matrix
32                    matrix.AddToEntry(row_idx[i], col_idx[j], elem_mat(i, j));
33                }
34            }
35        }
36    }
37 }
```

33 }}}

The template argument parameters and corresponding function arguments must match particular requirements:

- **TMPMATRIX** must be a matrix type supporting the **efficient entry-wise initialization** of large sparse Galerkin matrices. It must provide a member function

```
void AddToEntry(gdof_idx_t i, gdof_idx_t j, SCALAR increment);
```

that adds the `increment` argument to the matrix entry with row index `i` and column index `j`. A suitable type meeting this requirement is **If::assemble::COOMatrix**, see § 2.7.3.3.

The argument `matrix` is a **mutable** reference to a **TMPMATRIX** object, meant for returning the initialized matrix. The size of this matrix must fit the number of d.o.f.s managed by the **If::assemble::DofHandler** arguments. Note that assembly adds information to this matrix. Pass a matrix with all entries set to zero, if you want to build a Galerkin matrix from scratch.

- **ENTITY_MATRIX_PROVIDER** → (documentation) and the corresponding argument `entity_matrix_provider` is a class/object intended to perform the computation of **element matrices** through a call to the member function

```
auto Eval(const lf::mesh::Entity &entity);
```

which has to return a (dense) matrix-type object

- allowing access to entries via an **operator () (int i, int j)**,
- giving information about its size through the member functions `rows ()` and `cols ()` like any matrix of EIGEN.

Note that compile-time type resolution via **auto** in Line 25 of Code 2.7.4.23 offers a lot of flexibility about the return type of `Eval ()`.

The `entity_matrix_provider` must also have a member function

```
bool isActive(const lf::mesh::Entity &entity);
```

which makes the assembly loop skip entities, if it returns **false**.

Remark 2.7.4.24 (Variational problems with different trial and test spaces) Now we give an explanation for the occurrence of two different **If::assemble::DofHandler** arguments `dof_handler_trial` and `dof_handler_test` in the `dof_handler_trial` in the definition of the LEHRFEM++ function `AssembleMatrixLocally()` presented in Ex. 2.7.4.22.

So far we have always considered variational problems where trial and test space coincided both on the continuous and discrete level. This need not be the case, because the natural Sobolev spaces for a bilinear form may differ, as for

$$a(u, v) := \int_{\Omega} c \cdot \mathbf{grad} u(x) v(x) dx, \quad u \in H^1(\Omega), \quad v \in L^2(\Omega). \quad (2.7.4.25)$$

The simplest Galerkin finite element discretization of this bilinear form on a mesh \mathcal{M} would employ $S_1^0(\mathcal{M}) \subset H^1(\Omega)$ for u and merely \mathcal{M} -piecewise constant functions as test space.

Another more exotic case is the deliberate use of different finite element subspaces even in the case of a variational problem for which test and function space are the same. This generalization of the Galerkin approach is called a **Petrov-Galerkin discretization**.

§2.7.4.26 (Cell oriented assembly: “ $O(N)$ ” computational effort) If we assume “constant cost” for the local operations (invocation of `Eval()` in Code 2.7.4.23) then we conclude for the asymptotic computational effort as we use meshes with more and more elements:

$$\text{Computational cost(Assembly of Galerkin matrix } \mathbf{A} \text{)} = O(\#\mathcal{M})$$

EXAMPLE 2.7.4.27 (Global assembly of right-hand-side vector in LEHRFEM++) The following function provides the LEHRFEM++ realization of the algorithm of Code 2.7.4.21.

C++ code 2.7.4.28: Vector assembly function of LEHRFEM++ → GITHUB

```

1  template <typename VECTOR, class ENTITY_VECTOR_PROVIDER>
2  void AssembleVectorLocally(dim_t codim, const DofHandler &dof_handler,
3  ENTITY_VECTOR_PROVIDER &entity_vector_provider,
4  VECTOR &resultvector) {
5      // Pointer to underlying mesh
6      auto mesh = dof_handler.Mesh();
7
8      // Central assembly loop over entities of the co-dimension specified
9      // via
10     // the template argument CODIM
11     for (const If::mesh::Entity &entity : mesh->Entities(codim)) {
12         // Some cells may be skipped
13         if (entity_vector_provider.IsActive(entity)) {
14             // Length of element vector
15             const size_type veclen = dof_handler.NoLocalDofs(entity);
16             // global dof indices for contribution of the entity
17             If::base::RandomAccessRange<const gdof_idx_t> dof_idx(
18                 dof_handler.GlobalDofIndices(entity));
19             // Request local vector from entity_vector_provider object.
20             // In the case codim == 0, when 'entity' is a cell,
21             // this is the element vector
22             const auto elem_vec{entity_vector_provider.Eval(entity)};
23             // Assembly (single) loop: update of vector entries
24             for (int i = 0; i < veclen; i++) {
25                 resultvector[dof_idx[i]] += elem_vec[i];
26             }
27         }
28     }
29 }
```

The template arguments are similar to those discussed in Ex. 2.7.4.22:

- **VECTOR** is a rather generic vector type allowing component access via **operator [] (int)** and telling the vector length by a **size ()** member function.
- Note that the `resultvector` argument is not initialized by zero in the function. It will just be updated, which paves the way for multiple invocations of `AssembleVectorLocally` in order to build a right-hand-side vector in several steps.
- **ENTITY_VECTOR_PROVIDER** → (documentation) and the related argument `entity_vector_provider` are meant for objects that take care of the computation of **element vectors** through a call to an `Eval()` method. An `IsActive()` methods restricts assembly to entities, for which it returns **true**.

EXAMPLE 2.7.4.29 (LEHRFEM++ interface to local computations) We give concrete incarnations of the concepts underlying the parameter types **ENTITY_MATRIX_PROVIDER** and **ENTITY_VECTOR_PROVIDER** required for the assembly functions `AssembleMatrixLocally()` and `AssembleVectorLocally()`.

We consider the following linear variational problem related to a second-order elliptic Neumann boundary value problem for the Laplacian with zero flux boundary conditions, see Ex. 1.8.0.10:

$$u \in H^1(\Omega): \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in H^1(\Omega), \quad (2.7.4.30)$$

with given source function $f \in C^0(\overline{\Omega})$. We opt for a finite element Galerkin discretization with **linear Lagrangian finite elements**, $V_{0,h} = \mathcal{S}_1^0(\mathcal{M})$, where \mathcal{M} is a 2D triangular mesh of a polygonal domain $\Omega \subset \mathbb{R}^2$. The same setting was studied in Section 2.4.

All the details concerning the computation of the 3×3 element matrices for the Laplacian have been presented in Section 2.4.5 and an implementation code is given in Code 2.4.5.13. Refer to Section 2.4.6 for an in-depth discussion of how to compute element vectors for (2.7.4.30) based on the 2D trapezoidal rule and (2.4.6.11) for the final formula. The following types implement these formulas in LEHRFEM++. According to Code 2.7.4.23 and Code 2.7.4.28, they all have to provide the functions `Eval()` and `isActive()` → (documentation).

C++ code 2.7.4.31: Interface to element matrix for $-\Delta$ on a triangle → [GITHUB](#)

```

1 class LinFELaplaceElemMatProvider {
2     public:
3         LinFELaplaceElemMatProvider() = default;
4         virtual bool isActive(const If::mesh::Entity & /*cell*/) { return true; }
5         Eigen::Matrix<double, 3, 3> Eval(const If::mesh::Entity & tria);
6     };

```

C++ code 2.7.4.32: Interface to element vector induced by source function $f \in L^2(\Omega)$ → [GITHUB](#)

```

1 template <typename FUNCTOR>
2 class LinFEElemVecProvider {
3     public:
4         // Constructor: initialization of functor data member
5         explicit LinFEElemVecProvider(FUNCTOR f) : f_(f) {}
6         virtual bool isActive(const If::mesh::Entity & /*cell*/) { return true; }
7         Eigen::Vector3d Eval(const If::mesh::Entity & tria);
8     private:
9         FUNCTOR f_; // Functor object for source function f
10    };

```

The concrete implementations are the LEHRFEM++ equivalents of Code 2.4.5.13 and Code 2.4.6.12. The underlying formulas are discussed there.

C++ code 2.7.4.33: Computation of element matrix for $-\Delta$ on a triangle → [GITHUB](#)

```

2     Eigen::Matrix<double, 3, 3> LinFELaplaceElemMatProvider::Eval( // NOLINT
3         const If::mesh::Entity & tria) {
4             // Throw error in case no triangular cell
5             LF_VERIFY_MSG(tria.RefEl() == If::base::RefEl::kTria(),
6                           "Unsupported cell type " << tria.RefEl());
7             // Obtain vertex coordinates of the triangle in a 2x3 matrix
8             const auto vertices = If::geometry::Corners(*(tria.Geometry()));
9             LF_ASSERT_MSG((vertices.cols() == 3) && (vertices.rows() == 2),
10                           "Invalid vertex coordinate " << vertices.rows() << "x"
11                           << vertices.cols() << "matrix");
12

```

```

13 // Set up an auxiliary 3x3-matrix with a leading column 1 and
14 // the vertex coordinates in its right 3x2 block
15 Eigen::Matrix<double, 3, 3> X; // temporary matrix
16 X.block<3, 1>(0, 0) = Eigen::Vector3d::Ones();
17 X.block<3, 2>(0, 1) = vertices.transpose();
18 // The determinant of the auxiliary matrix also supplies the
19 // determinant
20 const double area = 0.5 * std::abs(X.determinant());
21 // Compute the gradients of the barycentric coordinate functions
22 // and store them in the columns of a 2x3 matrix grad_bary_coords
23 Eigen::Matrix<double, 2, 3>
24     grad_bary_coords{X.inverse().block<2, 3>(1, 0)};
25
26 // Since the gradients are constant, local integration is easy
27 return ((area * grad_bary_coords.transpose()) * grad_bary_coords);
}

```

C++ code 2.7.4.34: Approximate computation of element load vector for $v \mapsto \int_K f(x) dx$ on a triangle $K \rightarrow$ [GITHUB](#)

```

2 template <typename FUNCTOR>
3 Eigen::Vector3d LinFEElemVecProvider<FUNCTOR>::Eval(
4     const If::mesh::Entity &tria) {
5     // Throw error in case no triangular cell
6     LF_VERIFY_MSG(tria.RefEl() == If::base::RefEl::kTria(),
7         "Unsupported cell type " << tria.RefEl());
8     // Obtain vertex coordinates of the triangle in a 2x3 matrix
9     const auto corners{If::geometry::Corners(*(tria.Geometry()))};
10    const double area_third = If::geometry::Volume(*(tria.Geometry())) / 3.0;
11    LF_ASSERT_MSG((corners.cols() == 3) && (corners.rows() == 2),
12        "Invalid vertex coordinate " << corners.rows() << "x"
13        << corners.cols() << " matrix");
14    return Eigen::Vector3d(area_third * f_(corners.col(0)),
15        area_third * f_(corners.col(1)),
16        area_third * f_(corners.col(2)));
17 }

```

EXAMPLE 2.7.4.35 (Assembly of Galerkin linear system for homogeneous Neumann problem) In the previous Ex. 2.7.4.29 we learned about the LEHRFEM++ interfaces to local computations for the finite element space $\mathcal{S}_1^0(\mathcal{M})$ on a triangular 2D planar mesh \mathcal{M} . In the following code we combine them with the assembly functions studied in Ex. 2.7.4.22 to set up the final linear system of equations with a system matrix in compressed storage format amenable to sparse direct elimination techniques.

To begin with, following § 2.7.4.16, the local-to-global d.o.f. index mapper object is set up by the C++ statement:

```
lf::assemble::UniformFEDofHandler dof_handler(
    mesh_p, {{lf::base::RefEl::kPoint(), 1}});
```

that is, a single degree of freedom is assigned to every node of the mesh, no global shape function is associated with any higher-dimensional entities.

C++ code 2.7.4.36: Assembly of Galerkin linear system for (2.7.4.30) and $V_{0,h} = \mathcal{S}_1^0(\mathcal{M})$ [→ GITHUB](#)

```

2 | // Query dimension of the finite element space, equal to the number of |
|   nodes
3 | size_type N_dofs(dof_handler.NumDofs());
4 | // Matrix in triplet format holding temporary Galerkin matrix
5 | If ::assemble::COOMatrix<double> mat(N_dofs, N_dofs);
6 |
7 | // Initialize objects for local computations
8 | LinFELaplaceElemMatProvider loc_mat_laplace{};
9 | LinFEElemVecProvider<decltype(f)> loc_vec_sample(f);
10 |
11 | // Building the Galerkin matrix (trial space = test space)
12 | // for the pure Neumann Laplacian, assembly over cells
13 | mat = If ::assemble::AssembleMatrixLocally<If ::assemble::COOMatrix<double>>(
14 |     0, dof_handler, loc_mat_laplace);
15 | // Filling the right-hand-side vector, assembly over cells
16 | auto rhsvec = If ::assemble::AssembleVectorLocally<Eigen::VectorXd>(
17 |     0, dof_handler, loc_vec_sample);

2 | // Convert the matrix from triplet format to CRS format
3 | const Eigen::SparseMatrix<double> A(mat.makeSparse()); // NOLINT

```

EXAMPLE 2.7.4.37 (Assembly of boundary contributions) In the case of a second-order elliptic boundary value problem with Robin boundary conditions (\rightarrow Ex. 1.7.0.5, Ex. 1.8.0.6), we may face an augmented version of the variational problem (2.7.4.30).

$$u \in H^1(\Omega): \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx + \int_{\partial\Omega} u(x)v(x) \, dS(x) = \int_{\Omega} fv \, dx \quad \forall v \in H^1(\Omega). \quad (2.7.4.38)$$

How can we use the assembly facilities offered by LEHRFEM++ to incorporate the additional contributions from the boundary part of the bilinear form in Eq. (2.7.4.38)? As in Ex. 2.7.4.22 and Ex. 2.7.4.35 we consider the finite element space $S_1^0(\mathcal{M})$ on a triangular planar mesh \mathcal{M} and equipped with the nodal basis of tent functions.



The contribution of the boundary term $\int_{\partial\Omega} \dots dS$ can be computed by “cell-oriented” assembly on $\mathcal{M}|_{\partial\Omega}$.

- Use `AssembleMatrixLocally()` with `codim=1`,
- Use `ENTITY_MATRIX_PROVIDER` for edges, with `isActive() == true` for boundary edges only.

C++ code 2.7.4.39: Interface to element mass matrix for a straight edge \rightarrow [GITHUB](#)

```

1 class LinFEMassEdgeMatProvider {
2 public:
3     LinFEMassEdgeMatProvider(If ::mesh::utils::CodimMeshDataSet<bool> &bd_flags)
4         : bd_flags_(bd_flags) {}
5     // Assumes that bd_flags_ contain true for boundary edges only.
6     virtual bool isActive(const If ::mesh::Entity &edge) {
7         return bd_flags_(edge);
8     }
9     Eigen::Matrix2d Eval(const If ::mesh::Entity &edge);
10    private:
11        If ::mesh::utils::CodimMeshDataSet<bool> &bd_flags_;
12 };

```

Objects of `LinFEMassEdgeMatProvider` hold a reference to a LEHRFEM++ `flag array` of type

If::mesh::utils::CodimMeshDataSet based on **bool**. This data type offers an evaluation operator **bool operator () (const Entity &)** that tells a **true / false** flag value for every mesh entity of co-dimension 1 (edge). This can be used to flag edges on the boundary.

The implementation of the **Eval ()** member function computes the 2×2 “element matrix” **M_e** for the $L^2(e)$ inner product over a straight edge **e** discretized by means of linear Lagrangian finite elements:

$$\mathbf{M}_e = |e| \begin{bmatrix} 1/3 & 1/6 \\ 1/6 & 1/3 \end{bmatrix}, \quad |e| \triangleq \text{length of edge}. \quad (2.7.4.40)$$

C++ code 2.7.4.41: Computation of local mass matrix for straight edge → GITHUB

```

2  Eigen :: Matrix2d LinFEMassEdgeMatProvider :: Eval(const If::mesh::Entity &edge) { // NOLINT
3      LF_VERIFY_MSG(edge.RefEl() == If::base::RefEl::kSegment(),
4                      "Unsupported edge type " << edge.RefEl());
5      // Obtain endpoint coordinates of the triangle in a 2x3 matrix
6      const auto endpoints = If::geometry::Corners(*edge.Geometry());
7      // Compute length of edge
8      const double edge_length = (endpoints.col(1) - endpoints.col(0)).norm();
9      // Diagonal and off-diagonal entries of edge mass matrix
10     const double m1 = edge_length * 1.0 / 3.0;
11     const double m2 = edge_length * 1.0 / 6.0;
12     return ((Eigen::Matrix2d(2, 2) << m1, m2, m2, m1).finished());
13 }
```

We point out that the LEHRFEM++ utility class **If::uscalfe::MassEdgeMatrixProvider** could substitute for **LinFEMassEdgeMatProvider** provided that the right finite element space is used for initialization.

Let us see how to obtain the boundary part of the Galerkin matrix. The main version of **AssembleMatrixLocally** permits us to *update* the Galerkin matrix. This is done by the following code snippet. The variable **mat** is of type **If::assemble::COOMatrix** and contains the Galerkin matrix in triplet format.

C++ code 2.7.4.42: Adding boundary contribution to Galerkin matrix → GITHUB

```

2  // Flag all edge (co-dimension-1 entities) on the boundary
3  If::mesh::utils::CodimMeshDataSet<bool> bd_flags{
4      If::mesh::utils::flagEntitiesOnBoundary(mesh_p, 1);
5      LinFEMassEdgeMatProvider loc_edge_mass(bd_flags);
6      // Add boundary contributions to Galerkin matrix stored in 'mat'
7      // Assembly covers edges (co-dimensions-1 entities) !
8      If::assemble::AssembleMatrixLocally(1, dof_handler, dof_handler,
9                                         loc_edge_mass, mat);
```

This example also demonstrates the use of the LEHRFEM++ utility function

```

lf::mesh::utils::CodimMeshDataSet<bool>
lf::mesh::utils::flagEntitiesOnBoundary(
    const std::shared_ptr<const lf::mesh::Mesh>& mesh_p,
    lf::base::dim_t codim);
```

which supplies a **predicate (*)** on the set of mesh entities of a particular co-dimension. That predicate evaluates to **true**, if an entity is contained in the boundary of meshed domain. The boundary is defined as the union of edges adjacent to only one cell and all of their sub-entities.

(*) a predicate is a type or object with an evaluation operator **bool operator ()**.

2.7.4.4 Assembly: Linear Algebra Perspective

There is a formal “mathematical” way to express assembly in the language of linear algebra in terms of sums of matrix products. This is presented in the next theorem:

Theorem 2.7.4.43. Assembly through index mapping matrices

The stiffness matrix and load vector can be obtained from their cell counterparts, the element (stiffness) matrix \mathbf{A}_K and element (load) vector $\vec{\phi}_K$ (\rightarrow Def. 2.7.4.5), by

$$\mathbf{A} = \sum_K \mathbf{T}_K^\top \mathbf{A}_K \mathbf{T}_K , \quad \vec{\phi} = \sum_K \mathbf{T}_K^\top \vec{\phi}_K , \quad (2.7.4.44)$$

with the **index mapping matrices** (“T-matrices”) $\mathbf{T}_K \in \mathbb{R}^{Q,N}$, defined by

$$(\mathbf{T}_K)_{ij} := \begin{cases} 1 & , \text{if } (b_h^j)|_K = b_K^i , \\ 0 & , \text{otherwise.} \end{cases} \quad 1 \leq i \leq Q, 1 \leq j \leq N . \quad (2.7.4.45)$$

☞ Every index mapping matrix has exactly one non-vanishing entry per row! (why?)

“MATLAB-style pseudo-code” for the initialization of a sparse index mapping matrix based on the local→global index map introduced in (2.7.4.8), Q_k = number of local shape functions, initialization of \mathbf{T}_K from triplet format:

```
TK = sparse(1:Qk, locglobmap(K, 1:Qk), ones(Qk, 1));
```

Proof. (of Thm. 2.7.4.43) Use the definition of the entries of the Galerkin matrix, of the element matrix (\rightarrow Def. 2.7.4.5), and of the local shape functions (\rightarrow Def. 2.5.3.4):

$$\begin{aligned} (\mathbf{A})_{ij} &= \mathbf{a}(b_h^j, b_h^i) = \sum_{K \in \mathcal{M}} \mathbf{a}_K(b_h^j|_K, b_h^i|_K) = \\ &\sum_{\substack{K \in \mathcal{M}, \text{supp}(b_h^j) \cap K \neq \emptyset, \\ \text{supp}(b_h^i) \cap K \neq \emptyset}} \mathbf{a}_K(b_K^{l(j)}, b_K^{l(i)}) = \sum_{\substack{K \in \mathcal{M}, \text{supp}(b_h^j) \cap K \neq \emptyset, \\ \text{supp}(b_h^i) \cap K \neq \emptyset}} (\mathbf{A}_K)_{l(i), l(j)} . \end{aligned}$$

Here, $l(i) \in \{1, \dots, Q\}$, $1 \leq i \leq N \hat{=} \text{index of the local shape function corresponding to the global shape function } b_h^i \text{ on } K$.

➤ By (2.7.4.45), the indices $l(i)$ encode the T-matrix according to

$$(\mathbf{T}_K)_{l(i), i} = 1 , \quad i = 1, \dots, N ,$$

where all other entries of \mathbf{T}_K are understood to vanish.

$$\Rightarrow (\mathbf{A})_{ij} = \sum_{\substack{K \in \mathcal{M}, \text{supp}(b_h^j) \cap K \neq \emptyset, \\ \text{supp}(b_h^i) \cap K \neq \emptyset}} \sum_{l=1}^Q \sum_{n=1}^Q (\mathbf{T}_K)_{li} (\mathbf{A}_K)_{ln} (\mathbf{T}_K)_{nj} .$$

The rules for matrix multiplication give the assertion of the theorem. \square

EXAMPLE 2.7.4.46 (Index mapping matrix for linear Lagrangian finite elements on triangular mesh)

The local → global index mapping for linear finite elements with vertex associated global basis functions and three local basis functions was studied in Section 2.4.5, see also Rem. 2.7.4.9.

This example is connected to Ex. 2.7.4.10.

Using the local/global numbering indicated beside we find

$$\rightarrow \mathbf{T}_{K^*} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

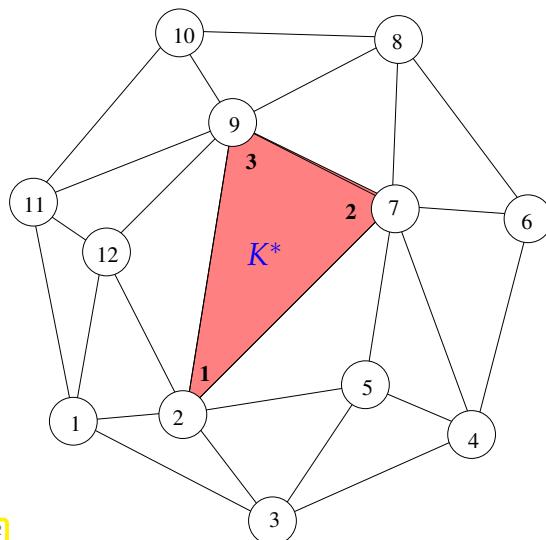


Fig. 142

Now we can rephrase the operation of Code 2.7.4.20 and Code 2.7.4.23 from a linear algebra point of view:

➤ Cell oriented assembly \leftrightarrow (2.7.4.44) $\leftrightarrow \mathbf{A} = \sum_K \mathbf{T}_K^\top \mathbf{A}_K \mathbf{T}_K$

$$\mathbf{A} = \sum_K \mathbf{T}_K^\top \mathbf{A}_K \mathbf{T}_K := \left\{ \begin{array}{l} \text{foreach } K \in \mathcal{M} \text{ do} \\ \quad \text{\color{purple}local operations on } K (\rightarrow \mathbf{A}_K) \text{ and } \mathbf{A} = \mathbf{A} + \mathbf{T}_K^\top \mathbf{A}_K \mathbf{T}_K \\ \text{enddo} \end{array} \right\}$$

Obviously, this is little to do with the actual implementation and may just serve as a convenient notation.

Review question(s) 2.7.4.47 (Assembly Algorithms)

(Q2.7.4.47.A) Explain the following concepts that play an essential role in finite-element methods:

- cell-oriented assembly,
 - local and global shape functions,
 - element matrices.

(Q2.7.4.47.B) What is the number of local shape functions for a cell of a 2D planar hybrid mesh and for the finite-element space $\mathcal{S}_p^0(\mathcal{M})$, $p \in \mathbb{N}$?

Hint. $\mathcal{S}_p^0(\mathcal{M}) \Big|_K$ is a space of polynomials. Which one?

(Q2.7.4.47.C) Consider the local shape functions for $\mathcal{S}_2^0(\mathcal{M})$ for a triangle or a quadrilateral $K \in \mathcal{M}$. Which (sub-)entities of K are they *associated* with?

(Q2.7.4.47.D) We consider the Lagrangian finite element space $\mathcal{S}_3^0(\mathcal{M})$ on a 2D planar hybrid mesh. The variable `dofh` is a reference to a `If::assemble::DofHandler` object for that finite element space. What will be returned by the following function calls

- `dofh.NumDofs()`,
 - `dofh.NumLocalDofs(entity)`,
 - `dofh.NumInteriorDofs(entity)`.

where `entity` is a constant reference to an `If::mesh::Entity` object belonging to the mesh \mathcal{M} . Of course, the return value will depend on `entity`.

(Q2.7.4.47.E) For which Lagrangian finite element space on a 2D hybrid mesh does this code line set up an appropriate **If::assemble::DofHandler** object:

```
lf::assemble::UniformFEDofHandler dof_handler(
mesh_p, {{lf::base::RefEl::kPoint(), 1},
{lf::base::RefEl::kSegment(), 2},
{lf::base::RefEl::kTria(), 1},
{lf::base::RefEl::kQuad(), 4}});
```

(Q2.7.4.47.F) Give an example of a variational problem connected with a second-order elliptic boundary value problem, where assembly will involve loops over entities of co-dimension > 0 .

(Q2.7.4.47.G) In LEHRFEM++ you have to deal with an exotic finite element scheme which assigns two local shape functions to each edge and two to each vertex. What is the type of the geometric entity which the local shape function b_K^i , $i = 1, \dots, Q$, $Q \doteq$ the total number of local shape functions, is associated with?

These are the rules for numbering local shape functions in LEHRFEM++:

- (I) The local shape functions of every cell (co-dimension-0 entity) are arranged according to *increasing dimension* of the geometric entities they are associated with:

POINT → **SEGMENT** → **{TRIA,QUAD}**.

- (II) Local shape functions belonging to geometric entities of the same dimension are ordered *according to* the intrinsic *local indexing* of those entities. See § 2.7.2.14 for LEHRFEM++’s conventions.
- (III) No ordering of local shape functions attached to the same geometric entity is implied. Here every user may follow her or his own conventions.

(Q2.7.4.47.H) For the finite element scheme from Question (Q2.7.4.47.G), what is the dimension of the finite element space on a triangular mesh with $\#M$ cells, $\#\mathcal{E}(M)$ edges, and $\#\mathcal{V}(M)$ vertices?

Give a rule for telling the type of geometric entity associated with a particular component of the vector of basis expansion coefficients.

The convention adopted by LEHRFEM++ when internally numbering the d.o.f.s managed by a **If::assemble::DofHandler** object is as follows:

- (I) D.o.f. associated with lower-dimensional entities are numbered first:

POINT → **SEGMENT** → **{TRIA,QUAD}**.

- (II) The indices of d.o.f. belonging to entities of the same co-dimension increase with increasing entity indices as returned by the **Index()** function.

(Q2.7.4.47.I) Explain, why endowing edges of the mesh with an orientation, which means giving them a well-defined direction, is important for the implementation of cubic Lagrangian finite elements.

△

2.7.5 Local Computations



Video tutorial for Section 2.7.5: Local Computations: (48 minutes) [Download link](#), [tablet notes](#)

We have seen that the (global) Galerkin matrix and right hand side vector are conveniently generated by “assembling” entries of element (stiffness) matrices and element (load) vectors.

Now we study the computation of these local quantities for Lagrangian finite elements on 2nd-order scalar linear boundary value problems in weak form, see also Section 2.4.5 and Section 2.4.6.

2.7.5.1 Analytic Formulas for Entries of Element Matrices

First option: Direct analytic evaluations (➡ “closed form” expressions)

We discuss this for the bilinear form related to $-\Delta$, triangular Lagrangian finite elements of degree p , Section 2.6.1, Def. 2.6.1.1:

$$K \text{ triangle: } a_K(u, v) := \int_K \mathbf{grad} u \cdot \mathbf{grad} v \, dx \quad \blacktriangleright \text{ element stiffness matrix .}$$

Use barycentric coordinate representations of local shape functions, in 2D

$$b_K^i = \sum_{\alpha \in \mathbb{N}_0^3, |\alpha|=p} \kappa_\alpha \lambda_1^{\alpha_1} \lambda_2^{\alpha_2} \lambda_3^{\alpha_3}, \quad \kappa_\alpha \in \mathbb{R}, \quad |\alpha| := \alpha_1 + \alpha_2 + \alpha_3, \quad (2.7.5.1)$$

where λ_i are the affine linear barycentric coordinate functions (linear shape functions), see Fig. 75.

For the barycentric coordinate representation of the quadratic local shape functions see (2.6.1.6), for a justification of (2.7.5.1) consult Suppl. 2.8.1.14.

$$\Rightarrow \mathbf{grad} b_K^i = \sum_{\alpha \in \mathbb{N}_0^3, |\alpha| \leq p} \kappa_\alpha \left(\alpha_1 \lambda_1^{\alpha_1-1} \lambda_2^{\alpha_2} \lambda_3^{\alpha_3} \mathbf{grad} \lambda_1 + \alpha_2 \lambda_1^{\alpha_1} \lambda_2^{\alpha_2-1} \lambda_3^{\alpha_3} \mathbf{grad} \lambda_2 + \alpha_3 \lambda_1^{\alpha_1} \lambda_2^{\alpha_2} \lambda_3^{\alpha_3-1} \mathbf{grad} \lambda_3 \right). \quad (2.7.5.2)$$

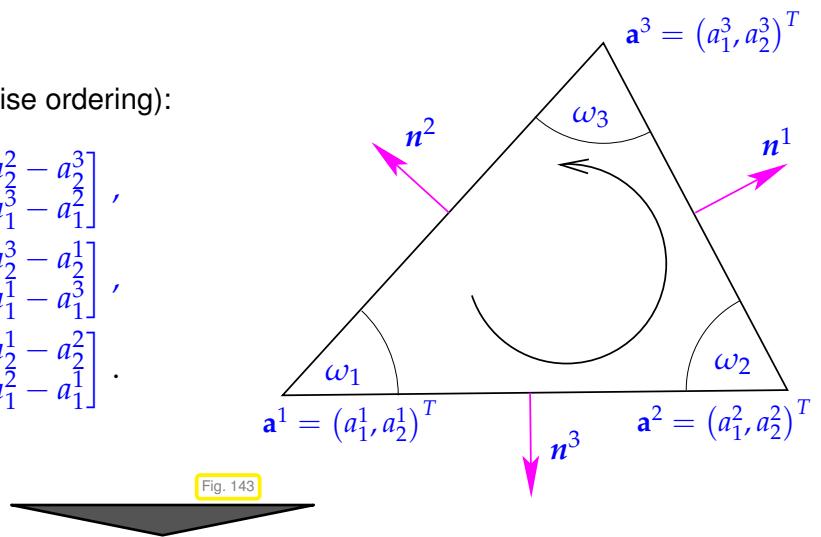


$$\text{To evaluate: } \int_K \lambda_1^{\beta_1} \lambda_2^{\beta_2} \lambda_3^{\beta_3} \mathbf{grad} \lambda_i \cdot \mathbf{grad} \lambda_j \, dx, \quad i, j \in \{1, 2, 3\}, \beta_k \in \mathbb{N}. \quad (2.7.5.3)$$

The (constant!) gradients of barycentric coordinate functions have already been computed in Section 2.4.5 on Page 172, see also Rem. 2.4.5.9.

If $\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3$ vertices of K (counterclockwise ordering):

$$\begin{aligned} \lambda_1(\mathbf{x}) &= \frac{1}{2|K|} \left(\mathbf{x} - \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix} \right) \cdot \begin{bmatrix} a_2^2 - a_2^3 \\ a_1^3 - a_1^2 \end{bmatrix}, \\ \lambda_2(\mathbf{x}) &= \frac{1}{2|K|} \left(\mathbf{x} - \begin{bmatrix} a_1^3 \\ a_2^3 \end{bmatrix} \right) \cdot \begin{bmatrix} a_2^3 - a_2^1 \\ a_1^1 - a_1^3 \end{bmatrix}, \\ \lambda_3(\mathbf{x}) &= \frac{1}{2|K|} \left(\mathbf{x} - \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix} \right) \cdot \begin{bmatrix} a_2^1 - a_2^2 \\ a_1^2 - a_1^1 \end{bmatrix}. \end{aligned}$$



$$\mathbf{grad} \lambda_1 = \frac{1}{2|K|} \begin{bmatrix} a_2^2 - a_2^3 \\ a_1^3 - a_1^2 \end{bmatrix}, \quad \mathbf{grad} \lambda_2 = \frac{1}{2|K|} \begin{bmatrix} a_2^3 - a_2^1 \\ a_1^1 - a_1^3 \end{bmatrix}, \quad \mathbf{grad} \lambda_3 = \frac{1}{2|K|} \begin{bmatrix} a_2^1 - a_2^2 \\ a_1^2 - a_1^1 \end{bmatrix}. \quad (2.7.5.4)$$

By (2.7.5.3), it remains to figure out the integral of products of powers of barycentric coordinate functions over a triangle.

Lemma 2.7.5.5. Integration of powers of barycentric coordinate functions

For any non-degenerate d -simplex K with barycentric coordinate functions $\lambda_1, \dots, \lambda_{d+1}$ and exponents $\alpha_j \in \mathbb{N}$, $j = 1, \dots, d+1$,

$$\int_K \lambda_1^{\alpha_1} \cdots \lambda_{d+1}^{\alpha_{d+1}} dx = d!|K| \frac{\alpha_1! \alpha_2! \cdots \alpha_{d+1}!}{(\alpha_1 + \alpha_2 + \cdots + \alpha_{d+1} + d)!} \quad \forall \alpha \in \mathbb{N}_0^{d+1}. \quad (2.7.5.6)$$

Proof. (for $d = 2$) The idea is to transform K to the “unit triangle” $\widehat{K} := \text{convex}\left\{\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}$:

$$\begin{aligned} \Rightarrow \int_K \lambda_1^{\beta_1} \lambda_2^{\beta_2} \lambda_3^{\beta_3} dx &= 2|K| \int_0^1 \int_0^{1-x_1} x_1^{\beta_1} x_2^{\beta_2} (1-x_1-x_2)^{\beta_3} dx_2 dx_1 \\ &\stackrel{(*)}{=} 2|K| \int_0^1 x_1^{\beta_1} \int_0^1 (1-x_1)^{\beta_2+\beta_3+1} s^{\beta_2} (1-s)^{\beta_3} ds dx_1 \\ &= 2|K| \int_0^1 x_1^{\beta_1} (1-x_1)^{\beta_2+\beta_3+1} dx_1 \cdot B(\beta_2+1, \beta_3+1) \\ &= 2|K| B(\beta_1+1, \beta_2+\beta_3+2) \cdot B(\beta_2+1, \beta_3+1), \end{aligned}$$

At step (*) we performed the substitution $s(1-x_1) = x_2$, $B(\cdot, \cdot) \triangleq$ Euler's beta function, a well known special function defined as

$$B(\alpha, \beta) := \int_0^1 t^{\alpha-1} (1-t)^{\beta-1} dt, \quad 0 < \alpha, \beta < \infty.$$

It satisfies the important relation $\Gamma(\alpha + \beta) B(\alpha, \beta) = \Gamma(\alpha)\Gamma(\beta)$, where Γ denotes the Gamma function, which interpolates the factorials: $\Gamma(n) = (n-1)!$,

$$\Rightarrow \int_K \lambda_1^{\beta_1} \lambda_2^{\beta_2} \lambda_3^{\beta_3} dx = 2|K| \cdot \frac{\Gamma(\beta_1+1)\Gamma(\beta_2+1)\Gamma(\beta_3+1)}{\Gamma(\beta_1+\beta_2+\beta_3+3)}.$$

By the properties of the Gamma function, this amounts to the assertion of the lemma. \square

EXAMPLE 2.7.5.7 (Element matrix for quadratic Lagrangian finite elements) In this example we, again, consider the local bilinear form related to $-\Delta$: $a_K(u, v) = \int_K \mathbf{grad} u \cdot \mathbf{grad} v dx$. We state the element matrix for an arbitrary triangle K for the nodal local shape functions as given in (2.6.1.6):

$$\begin{aligned} [\text{vertex associated}]: \quad b_K^1 &= (2\lambda_1 - 1)\lambda_1, & b_K^2 &= (2\lambda_2 - 1)\lambda_2, & b_K^3 &= (2\lambda_3 - 1)\lambda_3, \\ [\text{edge associated}]: \quad b_K^4 &= 4\lambda_1\lambda_2, & b_K^5 &= 4\lambda_2\lambda_3, & b_K^6 &= 4\lambda_1\lambda_3, \end{aligned}$$

where the $\lambda_i \in \mathcal{P}_1(\mathbb{R}^2)$ are barycentric coordinate functions, see Section 2.4.5, Rem. 2.4.5.9. They are affine-linear functions. The indices of the local shape functions respect LHRFEM++’s local numbering convention from Fig. 140. The entries of the 6×6 element matrix \mathbf{A}_K are given by the formula:

$$\mathbf{A}_K = \left[\int_K \mathbf{grad} b_K^j(x) \cdot \mathbf{grad} b_K^i(x) dx, \quad i, j \in \{1, \dots, 6\} \right]$$

First, by the product rule we compute the gradients of the local shape functions: Writing $\mathbf{g}_\ell := \mathbf{grad} \lambda_\ell$ for the *constant* gradients of the barycentric coordinate functions we find

$$\begin{aligned}\mathbf{grad} b_K^\ell &= (4\lambda_\ell - 1)\mathbf{g}_\ell, \quad \ell = 1, 2, 3, \\ \mathbf{grad} b_K^j &= 4(\lambda_i \mathbf{g}_k + \lambda_k \mathbf{g}_i), \quad j = 4, 5, 6, \quad i = j-3, k = j-2,\end{aligned}$$

where cyclic numbering is used (“ $k = 4 \rightarrow k = 1$ ”). As auxiliary quantities for the computation of \mathbf{A}_K we use the integrals from Lemma 2.7.5.5:

Lemma Lemma 2.7.5.5. Integration of powers of barycentric coordinate functions

For any non-degenerate d -simplex K with barycentric coordinate functions $\lambda_1, \dots, \lambda_{d+1}$ and exponents $\alpha_j \in \mathbb{N}$, $j = 1, \dots, d+1$,

$$\int_K \lambda_1^{\alpha_1} \cdots \lambda_{d+1}^{\alpha_{d+1}} dx = d!|K| \frac{\alpha_1! \alpha_2! \cdots \alpha_{d+1}!}{(\alpha_1 + \alpha_2 + \cdots + \alpha_{d+1} + d)!} \quad \forall \alpha \in \mathbb{N}_0^{d+1}. \quad (2.7.5.6)$$

$$\Rightarrow \int_K \lambda_\ell(x) dx = \frac{|K|}{3}, \quad \int_K \lambda_\ell^2(x) dx = \frac{|K|}{6}, \quad \int_K \lambda_i(x) \lambda_j(x) dx = \frac{|K|}{12},$$

and the $\mathcal{S}_1^0(\mathcal{M})$ -element matrix for $\mathbf{a}(\cdot, \cdot)$:

$$\mathbf{L}_K := [L_{i,j}]_{i,j=1}^3 := [\mathbf{a}_K(\lambda_j, \lambda_i)]_{i,j=1}^3 = |K| [\mathbf{g}_j \cdot \mathbf{g}_i]_{i,j=1}^3 \in \mathbb{R}^{3,3}.$$

Formulas for the entries of \mathbf{L} have been derived in § 2.4.5.5 and a code for its computation is presented in Code 2.4.5.13. The row and column sums of \mathbf{L}_K vanish.

Then the 6×6 element matrix can be written as $(L_{i,j}) := (\mathbf{L}_K)_{i,j}$

$$\mathbf{A}_K = \frac{1}{3} \left[\begin{array}{ccc|ccc} 3L_{1,1} & -L_{1,2} & -L_{1,3} & 4L_{1,2} & 0 & 4L_{1,3} \\ -L_{1,2} & 3L_{2,2} & -L_{2,3} & 4L_{1,2} & 4L_{2,3} & 0 \\ -L_{1,3} & -L_{2,3} & 3L_{3,3} & 0 & 4L_{3,2} & 4L_{3,1} \\ \hline 4L_{1,2} & 4L_{1,2} & 0 & A_{4,4} & 8L_{1,3} & 8L_{2,3} \\ 0 & 4L_{2,3} & 4L_{3,2} & 8L_{1,3} & A_{5,5} & 8L_{1,2} \\ 4L_{1,3} & 0 & 4L_{3,1} & 8L_{2,3} & 8L_{1,2} & A_{6,6} \end{array} \right]$$

with

$$\begin{aligned}A_{4,4} &:= 8(L_{1,1} + L_{1,2} + L_{2,2}), \\ A_{5,5} &:= 8(L_{2,2} + L_{2,3} + L_{3,3}), \\ A_{6,6} &:= 8(L_{1,1} + L_{1,3} + L_{3,3}).\end{aligned}$$

The partitioning of the matrix indicates that different parts of \mathbf{A}_K arise from the interaction of basis functions associated with vertices & vertices (top left block), vertices & edges (off-diagonal blocks), and edges & edges (bottom right block). Also verify that \mathbf{A}_K has zero row and column sums. \square

2.7.5.2 Local Quadrature

At this point turn the pages back to (2.3.3.12) and remember the use of numerical quadrature for computing the Galerkin matrix for the linear finite element method in 1D. Also recall the rationale for using mesh based composite quadrature rules.

Also recall § 2.4.6.9, where a simple local quadrature rule was used for the computation of element vectors. Next we take a look at its implementation in LEHRFEM++.

Since Lagrangian finite element functions are merely \mathcal{M} -piecewise smooth, numerical integration of expressions containing FE functions has to rely on composite quadrature rules on \mathcal{M} (“cell based quadrature”).

§2.7.5.8 (General local quadrature rules) A composite quadrature rule on a mesh \mathcal{M} of a domain $\Omega \subset \mathbb{R}^d$ splits an integral over Ω into cell contributions and approximately evaluates those. This latter step is based on so-called local quadrature rules.

Definition 2.7.5.9. (Local) quadrature rule

A local quadrature rule on the element $K \in \mathcal{M}$ is an approximation

$$\int_K f(x) dx \approx \sum_{l=1}^{P_K} \omega_l^K f(\zeta_l^K), \quad \zeta_l^K \in K, \omega_l^K \in \mathbb{R}, \quad P_K \in \mathbb{N}. \quad (2.7.5.10)$$

Terminology:

$$\begin{aligned} \omega_l^K &\rightarrow \text{weights} , & \zeta_l^K &\rightarrow \text{quadrature nodes} \\ (2.7.5.10) &= P\text{-point local quadrature rule} \end{aligned}$$

Def. 2.7.5.9 generalizes the quadrature rule (2.3.3.7) in 1D. The same terminology still applies. We have already come across a local quadrature rule in 2D, the trapezoidal rule from (2.4.6.10).

Recall from § 2.3.3.6, § 2.3.3.10 that numerical quadrature is inevitable

- for computation of load vector, if f is complicated or only available in procedural form, Rem. 2.1.2.5,
- for computation of stiffness matrix, if the non-constant coefficient $\alpha = \alpha(x)$ in the bilinear form from (1.4.2.4), (1.8.0.16) does not permit analytic integration.

We recall a constraint on the weights of local quadrature rules:

Guideline [Hip19, ??]: only quadrature rules with positive weights are numerically stable.

§2.7.5.11 (Transformation of quadrature rules) Generically, the quadrature rule (2.7.5.10) is specific for the cell K . This begs the questions how local quadrature rules are handled on finite element meshes with millions of cells.

The policy is the same as in 1D in [Hip19, ??]: there the (local) quadrature rule was defined on a *reference interval*, e.g., $[-1, 1]$ for Gaussian quadrature and mapped to a general interval by (affine) transformation, cf. [Hip19, ??].

The local quadrature rules used in finite element methods are obtained by transformation from (a few) local quadrature rules defined on *reference elements*.

Reference elements and the associated transformations will be studied in the sequel with focus on the construction of local quadrature rules, and in a more general context in Section 2.8.

§2.7.5.12 (Affine transformation of triangles) Now we examine the generalization of affine transformations from 1D to two dimensions:

Definition 2.7.5.13. Affine (linear) transformation

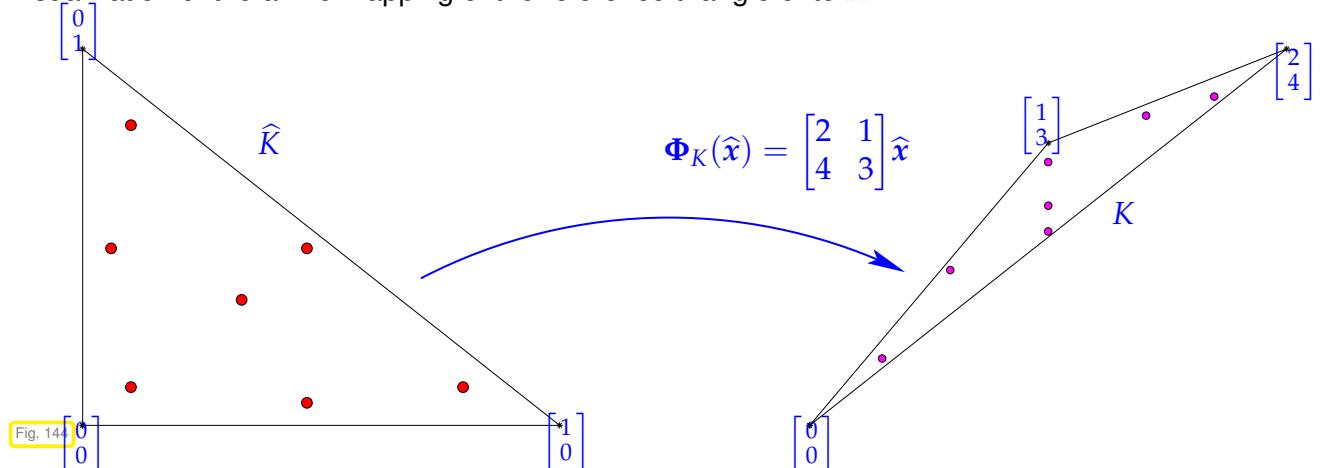
A mapping $\Phi : \mathbb{R}^d \mapsto \mathbb{R}^d$ is **affine (linear)**, if $\Phi(\mathbf{x}) = \mathbf{F}\mathbf{x} + \boldsymbol{\tau}$ with some $\mathbf{F} \in \mathbb{R}^{d,d}$, $\boldsymbol{\tau} \in \mathbb{R}^d$.

☞ **Reference triangle:** ‘unit triangle’ $\hat{K} := \text{convex} \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$ (numbered vertices!)

Lemma 2.7.5.14. Affine transformation of triangles

For any non-degenerate triangle $K \subset \mathbb{R}^2$ ($|K| > 0$) with numbered vertices there is a **unique** affine transformation Φ_K , $\Phi_K(\hat{\mathbf{x}}) = \mathbf{F}_K \hat{\mathbf{x}} + \boldsymbol{\tau}_K$ (\rightarrow Def. 2.7.5.13), with $K = \Phi_K(\hat{K})$ and preserving the numbering of the vertices.

Visualization of the affine mapping of the reference triangle onto K :



The matrix \mathbf{F}_K and translation vector $\boldsymbol{\tau}_K$ can be determined by solving a 6×6 linear system of equations, from which we obtain:

$$K = \text{convex} \left\{ \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}, \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix}, \begin{bmatrix} a_1^3 \\ a_2^3 \end{bmatrix} \right\} \Rightarrow \Phi_K(\hat{\mathbf{x}}) = \begin{bmatrix} a_1^2 - a_1^1 & a_1^3 - a_1^2 \\ a_2^2 - a_2^1 & a_2^3 - a_2^2 \end{bmatrix} \hat{\mathbf{x}} + \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}. \quad (2.7.5.15)$$

Remember that for any affine transformation the resulting relative change of volume is given by the constant determinant of its matrix, which gives in the setting of Lemma 2.7.5.14

$$K := \Phi_K(\hat{K}) \Rightarrow |K| = |\hat{K}| \det \mathbf{F}_K. \quad (2.7.5.16)$$

☞ **§2.7.5.17 (Reference elements and transformations in LEHRFEM++)** Every entity of co-dimension 0 (= cell, element) of the mesh has a **reference element** associated with it, depending on the topological type of the element, which is returned by the member function

```
lf::base::RefEl lf::mesh::Entity::RefEl() const;
```

For the entity types relevant for cells of 2D hybrid meshes we have:

EDGE : reference element $\hat{K} :=]0, 1[$, (2.7.5.18a)

TRIA : reference element $\hat{K} := \text{convex} \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$, (2.7.5.18b)

$$\text{QUAD : reference element } \hat{K} := \text{convex} \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}. \quad (2.7.5.18c)$$

The corner coordinates of the reference element of an entity object `entity` packed into the columns of a matrix can be retrieved as follows:

```
const Eigen::MatrixXd ref_el_corners = entity.RefEl().NodeCoords();
```

Each mesh entity object of any dimension $n \in \{0, \dots, 2\}$ in LEHRFEM++ comes with a mapping Φ_K from the corresponding reference element $\hat{K} \subset \mathbb{R}^n$ to its actual shape $K \subset \mathbb{R}^d$, $d = 2, 3$. This mapping

- is always bijective (one-to-one) and smooth up to boundary of \hat{K} ,
- will take the j -th vertex of \hat{K} to the j -th vertex of K . In § 2.7.2.14 we have seen how the ordering of the nodes of an entity is determined by the array returned by the `SubEntities()` member functions of `If::mesh::Entity`.

The mapping Φ_K is information belonging to the geometry layer and, thus, stored in the `If::geometry::Geometry` object invariably attached to every `If::mesh::Entity` object. Remember that the geometry object of `entity` can be fetched by the following statement:

```
lf::geometry::Geometry &geo_obj *entity.Geometry();
```

The key member function of the interface class `If::geometry::Geometry` encoding the mapping Φ_K is

```
Eigen::MatrixXd lf::geometry::Geometry::Global(
    const Eigen::MatrixXd &local) const;
```

It takes “reference coordinates” of k points $\hat{x} \in \hat{K}$ packed into the columns of the $n \times k$ -matrix `local` and returns the `world coordinates` of their images under Φ_K in the columns of a $d \times k$ -matrix:

$$\text{Global}(\hat{x}^1, \dots, \hat{x}^k) = [\Phi_K(\hat{x}^1), \dots, \Phi_K(\hat{x}^k)], \quad \forall \hat{x}^j \in \hat{K}, \quad j \in \{1, \dots, k\}. \quad (2.7.5.19)$$

To elucidate the use of `lf::geometry::Geometry::Global()`, we list the implementation of the LEHRFEM++ function returning the coordinates of the corners of an entity.

C++ code 2.7.5.20: Implementation of `lf::geometry::Corners()` → [GITHUB](#)

```
1 inline Eigen::MatrixXd Corners(const Geometry& geo) {
2     return geo.Global(geo.RefEl().NodeCoords());
3 }
```

§2.7.5.21 (Transformation of local quadrature rules on triangles) Now we resume the discussion started in § 2.7.5.11: We write $\Phi_K(\hat{x}) := F_K \hat{x} + \tau_K$ for an `affine` transformation (→ Def. 2.7.5.13) of the reference triangle \hat{K} to the general triangle K , see Lemma 2.7.5.14.

By the transformation formula for integrals [Str09, Satz 8.5.2] we can pull back integrals over K to \hat{K} :

$$\int_K f(x) dx = \int_{\hat{K}} f(\Phi_K(\hat{x})) |\det F_K| d\hat{x}. \quad (2.7.5.22)$$

This enables the transition

P -point quadrature formula on \hat{K} ➔ P -point quadrature formula on K ,

and (2.7.5.16) tells us how to adapt the quadrature weights ($|K| = \text{Vol}(K)$):

$$\int_{\hat{K}} f(\hat{x}) d\hat{x} \approx \sum_{\ell=1}^P \hat{\omega}_\ell f(\hat{\zeta}_\ell) \quad \Rightarrow \quad \int_K f(x) dx \approx \frac{|K|}{|\hat{K}|} \sum_{\ell=1}^P \omega_\ell^K f(\zeta_\ell^K) \quad (2.7.5.23)$$

with $\omega_\ell^K = \hat{\omega}_\ell$, $\zeta_\ell^K = \Phi_K(\hat{\zeta}_\ell)$.

- Only the quadrature formula (2.7.5.10) on the reference triangle \hat{K} needs to be specified!
(The same applies to tetrahedra, where affine mappings for $d=3$ are used.)

§2.7.5.24 (Transformation of quadrature rules on general mesh entities) Now let K stand for a mesh entity of any topological type, for instance, a potentially curved EDGE in the plane. Its shape is defined by the bijective transformation $\Phi_K : \hat{K} \rightarrow K$ from the reference element: $K := \Phi_K(\hat{K})$.

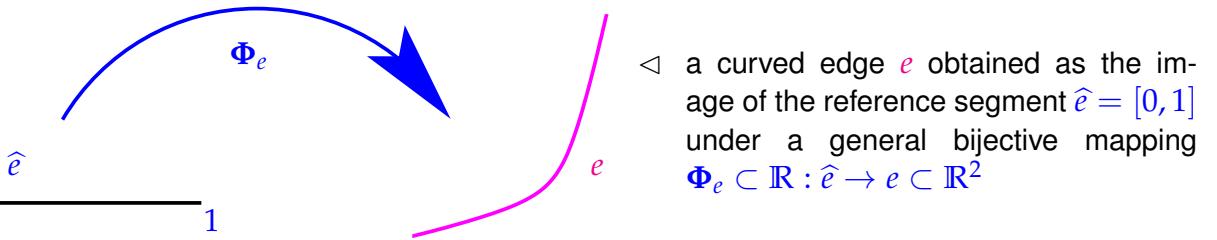


Fig. 145

Note that $\hat{K} \in \mathbb{R}^n$, $K \subset \mathbb{R}^d$, and that $n \neq d$ is possible; in the case of an edge in the plane we have $n=1$ and $d=2$. In terms of LEHRFEM++ functions and K an **If::mesh::Entity** object

$n = K.\text{Geometry}() \rightarrow \text{DimLocal}()$, $d = K.\text{Geometry}() \rightarrow \text{DimGlobal}()$.

This means that $D\Phi_K(\hat{x}) \in \mathbb{R}^{d,n}$ need not be a square matrix. Thus we need a more general transformation formula:

$$\int_K f(x) dS(x) = \int_{\hat{K}} f(\Phi_K(\hat{x})) \underbrace{\sqrt{\det(D\Phi(\hat{x})^\top D\Phi(\hat{x}))}}_{\triangleq \text{Gramian determinant}} d\hat{x}, \quad (2.7.5.25)$$

of which (0.3.2.32) is the special variant for $n=d$.

Now, given the quadrature rule on the reference element \hat{K}

$$\int_{\hat{K}} f(\hat{x}) d\hat{x} \approx \sum_{\ell=1}^P \hat{\omega}_\ell f(\hat{\zeta}_\ell),$$

by means of (2.7.5.25) we can convert it to a quadrature formula on K

$$\int_K f(x) dS(x) \approx \sum_{\ell=1}^P \omega_\ell^K f(\zeta_\ell^K) \quad \text{with} \quad \begin{aligned} \omega_\ell^K &:= \det(D\Phi(\hat{\zeta}_\ell)^\top D\Phi(\hat{\zeta}_\ell))^{1/2} \hat{\omega}_\ell, \\ \zeta_\ell^K &:= \Phi_K(\hat{\zeta}_\ell). \end{aligned} \quad (2.7.5.26)$$

In the case of an **affine** mapping $\Phi_K(\hat{x}) := F_K \hat{x} + \tau_K$, $F_K \in \mathbb{R}^{d,n}$ with full rank, we end up with the Gramian determinant $\det(F_K^\top F_K)^{1/2}$ and the formula $\omega_\ell^K = \det(F_K^\top F_K)^{1/2} \hat{\omega}_\ell$, $\ell = 1, \dots, P$.

In LEHRFEM++ the general Gramian determinant

$$\hat{x} \mapsto \sqrt{\det(D\Phi(\hat{x})^\top D\Phi(\hat{x}))}$$

is available through a dedicated member function of **If::geometry::Geometry**:

```
Eigen::VectorXd lf::geometry::Geometry::IntegrationElement(
```

that takes an array of (reference) coordinates of points $\hat{\mathbf{x}}^j \in \hat{K}$, $j = 1, \dots, k$ stored in the columns of the local $n \times k$ -matrix argument, and returns a k -vector with Gramian determinants:

$$\text{IntegrationElement}\left(\underbrace{\hat{x}^1, \dots, \hat{x}^k}_{\in \mathbb{R}^{n,k}}\right) = \begin{bmatrix} \det(\mathbf{D}\Phi(\hat{x}^1)^\top \mathbf{D}\Phi(\hat{x}^1))^{1/2} \\ \vdots \\ \det(\mathbf{D}\Phi(\hat{x}^k)^\top \mathbf{D}\Phi(\hat{x}^k))^{1/2} \end{bmatrix} \in \mathbb{R}^k, \quad \hat{x}^j \in \hat{K} \subset \mathbb{R}^n.$$

§2.7.5.27 (Order of local quadrature rule) How can we gauge the quality of mapped local quadrature rules ? We briefly review the discussion in [Hip19, ??].

Gauging the quality of a quadrature formula

The quality of a parametric local quadrature rule on K is measured *maximal degree of polynomials* (multivariate → Def. 2.5.2.2, or tensor product → Def. 2.5.2.7) on K integrated exactly by the corresponding quadrature rule on K .

Definition 2.7.5.29. Order of a local quadrature rule

A local quadrature rule according to Def. 2.7.5.9 is said to be of **order** $q \in \mathbb{N}$, if

- for a simplex K (triangle, tetrahedron) it is exact for all *polynomials* $f \in \mathcal{P}_{q-1}(\mathbb{R}^d)$,
 - for a tensor product element K (rectangle, brick) it is exact for all *tensor product polynomials* $f \in \mathcal{Q}_{q-1}(\mathbb{R}^d)$.

Note: Quadrature rule exact for $\mathcal{P}_p(\mathbb{R}^d)$ \Rightarrow quadrature rule of order $p+1$
degree of exactness p

How is the order of a local quadrature rule linked with the number of quadrature points?

Recall 1D: P -point Gaussian quadrature rule achieves maximal order $2P$, see [Hip19, ??]

On triangles/tetrahedra there is no simple general formula has been found linking the order and the minimal number of quadrature nodes, but there is a simple overall relationship for “optimal” quadrature formulas:

The price of higher order quadrature

For “optimal” local quadrature formulas:

the higher the order the more quadrature nodes are required.

§2.7.5.31 (Preservation of order under affine mappings) An important observation is that the space $\mathcal{P}_p(\mathbb{R}^d)$ is *invariant* under *affine mappings*, that is

$$q \in \mathcal{P}_p(\mathbb{R}^d) \Rightarrow \hat{x} \mapsto q(\Phi(\hat{x})) \in \mathcal{P}_p(\mathbb{R}^d) \text{ for any affine transformation } \Phi. \quad (2.7.5.32)$$

This means, if a quadrature rule on the reference element integrates all polynomials up to degree p exactly, the same is achieved by the mapped quadrature rule on K , if the underlying mapping is affine.

- The orders of the quadrature rules on the left and right hand side of (2.7.5.23) agree!
- Its order is an intrinsic property of a quadrature rule on the reference triangle/tetrahedron \hat{K} and will be inherited by all derived quadrature rules on elements that are *affine* images of \hat{K} .

EXAMPLE 2.7.5.33 (Local quadrature rules on triangles) By the transformation policy it is enough to specify the quadrature rule for the **reference triangle** (“unit triangle”) $\hat{K} := \text{convex}\left\{\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}$.

According to Def. 2.7.5.9 quadrature rules on \hat{K} can be described by pairs $(\hat{\omega}_P, \hat{\zeta}_P)$, $P \in \mathbb{N}$, of weights $\hat{\omega}_P$ and nodes $\hat{\zeta}_P \in \hat{K}$.

♦ P3O2: 3-point quadrature rule of order 2 (exact for $\mathcal{P}_1(\hat{K})$)

$$\left\{ \left(\frac{1}{3}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right), \left(\frac{1}{3}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right), \left(\frac{1}{3}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) \right\}. \quad (2.7.5.34)$$

♦ P3O3: 3-point quadrature rule of order 3 (exact for $\mathcal{P}_2(\hat{K})$)

$$\left\{ \left(\frac{1}{3}, \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} \right), \left(\frac{1}{3}, \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} \right), \left(\frac{1}{3}, \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \right) \right\}. \quad (2.7.5.35)$$

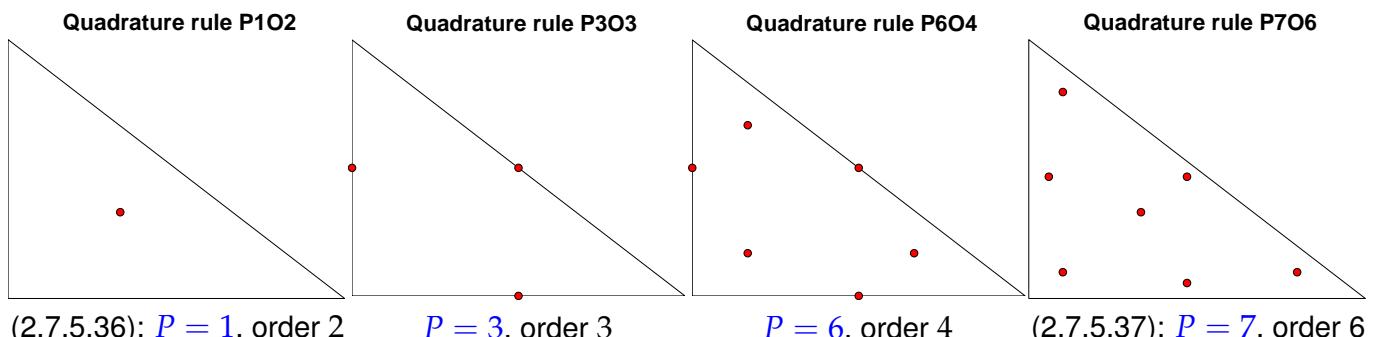
♦ P1O2: One-point quadrature rule of order 2 (exact for $\mathcal{P}_1(\hat{K})$)

$$\left\{ \left(1, \begin{bmatrix} 1/3 \\ 1/3 \end{bmatrix} \right) \right\}. \quad (2.7.5.36)$$

♦ P7O6: 7-point quadrature rule of order 6 (exact for $\mathcal{P}_5(\hat{K})$)

$$\left\{ \left(\frac{9}{40}, \begin{bmatrix} 1/3 \\ 1/3 \end{bmatrix} \right), \left(\frac{155 + \sqrt{15}}{1200}, \begin{bmatrix} 6+\sqrt{15}/21 \\ 6+\sqrt{15}/21 \end{bmatrix} \right), \left(\frac{155 + \sqrt{15}}{1200}, \begin{bmatrix} 9-2\sqrt{15}/21 \\ 6+\sqrt{15}/21 \end{bmatrix} \right), \right. \\ \left. \left(\frac{155 + \sqrt{15}}{1200}, \begin{bmatrix} 6+\sqrt{15}/21 \\ 9-2\sqrt{15}/21 \end{bmatrix} \right), \left(\frac{155 - \sqrt{15}}{1200}, \begin{bmatrix} 6-\sqrt{15}/21 \\ 9+2\sqrt{15}/21 \end{bmatrix} \right), \right. \\ \left. \left(\frac{155 - \sqrt{15}}{1200}, \begin{bmatrix} 9+2\sqrt{15}/21 \\ 6-\sqrt{15}/21 \end{bmatrix} \right), \left(\frac{155 - \sqrt{15}}{1200}, \begin{bmatrix} 6-\sqrt{15}/21 \\ 6-\sqrt{15}/21 \end{bmatrix} \right) \right\} \quad (2.7.5.37)$$

Location of quadrature nodes $\hat{\zeta}_l$ in the unit triangle \hat{K} :



In the article [Dun85] one can find quadrature rules up to order $p = 21$ with $P \leq 1/6p(p+1) + 5$ points.

↓

EXAMPLE 2.7.5.38 (Local quadrature rules on quadrilaterals) If K is a quadrilateral, then the reference element is $\hat{K} := \text{convex}\left\{\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right\}$ (unit square).

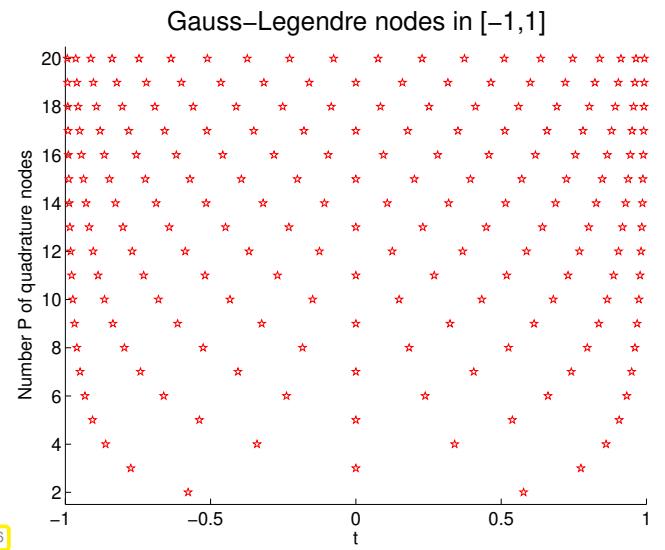
On the unit square \hat{K} use a **tensor product construction**: If $\{(\omega_1, \zeta_1), \dots, (\omega_P, \zeta_P)\}, P \in \mathbb{N}$, quadrature rule on the interval $]0, 1[$, exact for $\mathcal{P}_p)]0, 1[$, then a quadrature rule on the unit square is given by the following sequence of P^2 weight–nodes pairs:

$$\left\{ \begin{array}{ccc} (\omega_1^2, \begin{bmatrix} \zeta_1 \\ \zeta_1 \end{bmatrix}) & \cdots & (\omega_1 \omega_P, \begin{bmatrix} \zeta_1 \\ \zeta_P \end{bmatrix}) \\ \vdots & & \vdots \\ (\omega_1 \omega_P, \begin{bmatrix} \zeta_P \\ \zeta_1 \end{bmatrix}) & \cdots & (\omega_P^2, \begin{bmatrix} \zeta_P \\ \zeta_P \end{bmatrix}) \end{array} \right\}$$

It provides a quadrature rule on the unit square \hat{K} that is exact for $\mathcal{Q}_p(\hat{K})$. → order $p + 1$!

Recall quadrature rules on $]0, 1[$ (→ [Hip19, ??]):

- classical **Newton-Cotes formulas** (equidistant quadrature nodes).
- **Gauss-Legendre quadrature rules**, exact for $\mathcal{P}_{2P}(]0, 1[)$ using only P nodes.
- **Gauss-Lobatto quadrature rules**: P nodes including $\{0, 1\}$, exact for $\mathcal{P}_{2P-1}(]0, 1[)$.



§2.7.5.39 (Quadrature rules in LEHRFEM++) The concept of a quadrature rule according to Def. 2.7.5.9 is incarnated in the LEHRFEM++ class **If::quad::QuadRule** with the following member functions:

- **RefEl()**, returns reference element \hat{K} on which the quadrature rule is defined; one of NODE, EDGE, TRIA, QUAD for 2D hybrid meshes, see (2.7.5.18).
- **Degree()**, **Order()** tell the degree of (polynomial) exactness and the order, respectively, of the quadrature rule as given in Def. 2.7.5.29.
- **NumPoints()** returns the number P of quadrature nodes, see (2.7.5.10).
- **Points()** yields a matrix of size $n \times P$ containing the (reference/local) coordinates of the nodes $\hat{\zeta} \in \hat{K}$ of the P -point quadrature rule on \hat{K} .
- **Weights()** gives a column P -vector (**Eigen::VectorXd**) of quadrature weights $\hat{\omega}_\ell, \ell = 1, \dots, P$, for a P -point quadrature rule on \hat{K} .

LEHRFEM++ comes with a large stock of *predefined quadrature rules*, accessible through the function
→ [GITHUB](#)

```
lf::quad::QuadRule lf::quad::make_QuadRule(
    base::RefEl ref_el, unsigned degree);
```

Here, `ref_el` specifies the type of the reference element, while `degree` passes the desired **minimal** degree of exactness.

Alternatively, a few special quadrature rules can directly be fetched by dedicated functions, for instance, `lf::quad::make_TriaQR_P7O6()` will return a 7-point rule of order 6 on the reference triangle TRIA, see → [GITHUB](#) for other functions available in `make_quad_rule.h`

```
lf::quad::QuadRule lf::quad::make_TriaQR_MidpointRule();
lf::quad::QuadRule lf::quad::make_TriaQR_P1O2();
lf::quad::QuadRule lf::quad::make_TriaQR_EdgeMidpointRule();
lf::quad::QuadRule lf::quad::make_TriaQR_P3O3();
lf::quad::QuadRule lf::quad::make_TriaQR_P7O6();
lf::quad::QuadRule lf::quad::make_TriaQR_P6O4();
lf::quad::QuadRule lf::quad::make_QuadQR_MidpointRule();
lf::quad::QuadRule lf::quad::make_QuadQR_P1O2();
lf::quad::QuadRule lf::quad::make_QuadQR_P4O2();
```

The naming convention for the functions `lf::quad::make_TriaQR_P?O?()` is explained in Ex. 2.7.5.33. □

§2.7.5.40 (Local quadrature on mesh entities in LEHRFEM++) We discuss how to apply numerical quadrature on a mesh entity K of any co-dimension in LEHRFEM++. Since we do not impose restrictions on the mapping Φ_K from the reference element \hat{K} we have to resort to the general formula

$$\int_K f(x) dS(x) \approx \sum_{\ell=1}^P \omega_\ell^K f(\zeta_\ell^K) \quad \text{with} \quad \begin{aligned} \omega_\ell^K &:= \det(D\Phi(\hat{\zeta}_\ell)^T D\Phi(\hat{\zeta}_\ell))^{1/2} \hat{\omega}_\ell, \\ \zeta_\ell^K &:= \Phi_K(\hat{\zeta}_\ell), \end{aligned} \quad (2.7.5.26)$$

from § 2.7.5.24 based on the P -point quadrature rule on \hat{K}

$$\int_{\hat{K}} f(\hat{x}) d\hat{x} \approx \sum_{\ell=1}^P \hat{\omega}_\ell f(\hat{\zeta}_\ell).$$

The following code implements (2.7.5.26) based on methods of the interfaces **If::quad::QuadRule** and **If::geometry::Geometry** to accomplish the composite local quadrature of a function $f \in C^0(\bar{\Omega})$ given through a functor object. Of course, a mesh M of Ω has to be supplied. Note that the argument `quadrules` passes different quadrature rules for different reference elements.

C++ code 2.7.5.41: Entity-based composite numerical quadrature → [GITHUB](#)

```
2 template <typename FUNCTOR>
3 auto localQuadFunction(
4     const If::mesh::Mesh &mesh,
5     std::map<If::base::RefEl, If::quad::QuadRule> quadrules, FUNCTOR &&f,
6     dim_t codim,
7     std::function<bool(const If::mesh::Entity &)> pred =
8         [](&const If::mesh::Entity &/>entity*/) -> bool { return true; }) {
9     LF_ASSERT_MSG(mesh.DimMesh() >= codim, "Illegal codim = " << codim);
10    // Variable for summing the result
11    using value_t = std::invoke_result_t<FUNCTOR, Eigen::VectorXd>;
12    value_t sum_var{};
13    // Loop over entities of co-dimension codim
14    for (&const If::mesh::Entity *entity : mesh.Entities(codim)) {
15        // Obtain geometry information for entity
16        const If::geometry::Geometry &geo{*entity->Geometry()};
17        // obtain quadrature rule suitable for entity type
18        auto tmp = quadrules.find(entity->RefEl());
```

```

19   if (tmp != quadrules.end()) {
20     // A quadrature rule has been found
21     const If::quad::QuadRule &qr{tmp->second};
22     // Number of quadrature points
23     const size_type P = qr.NumPoints();
24     // Quadrature points
25     const Eigen::MatrixXd zeta_ref{qr.Points()};
26     // Map quadrature points to physical/world coordinates
27     const Eigen::MatrixXd zeta{geo.Global(zeta_ref)};
28     // Quadrature weights
29     const Eigen::VectorXd w_ref{qr.Weights()};
30     // Gramian determinants
31     const Eigen::VectorXd gram_dets{geo.IntegrationElement(zeta_ref)};
32     // Iterate over the quadrature points
33     for (int l = 0; l < P; ++l) {
34       sum_var += w_ref[l] * f(zeta.col(l)) * gram_dets[l];
35     }
36   } else {
37     LF_VERIFY_MSG(false, "Missing quadrature rule for " << entity->RefEl());
38   }
39 }
40 return sum_var;
41 }
```

Note that the functor object may return any type that supports cumulative addition `+=` and multiplication with `double`. This function may be called as follows, using an initializer list to generate the associative array of quadrature rules.

C++ code 2.7.5.42: Use of `localQuadFunction()` → [GITHUB](#)

```

2   // Function to be integrated
3   auto f = []([const Eigen::VectorXd& x) -> double {
4     return (x[0] * x[0] + x[1] * x[1]);
5   };
6   // Cell-based composite quadrature covering all cells
7   double integral_val = localQuadFunction(
8     *mesh_p,
9     {{If::base::RefEl::kTria(), If::quad::make_TriaQR_EdgeMidpointRule()},
10      {If::base::RefEl::kQuad(), If::quad::make_QuadQR_P4O4()}},
11      f, 0);
```

Review question(s) 2.7.5.43 (Local Computations in FEM)

(Q2.7.5.43.A) We have the general formula:

Lemma 2.7.5.5. Integration of powers of barycentric coordinate functions

For any non-degenerate d -simplex K with barycentric coordinate functions $\lambda_1, \dots, \lambda_{d+1}$ and exponents $\alpha_j \in \mathbb{N}$, $j = 1, \dots, d+1$,

$$\int_K \lambda_1^{\alpha_1} \cdots \lambda_{d+1}^{\alpha_{d+1}} dx = d!|K| \frac{\alpha_1! \alpha_2! \cdots \alpha_{d+1}!}{(\alpha_1 + \alpha_2 + \cdots + \alpha_{d+1} + d)!} \quad \forall \alpha \in \mathbb{N}_0^{d+1}. \quad (2.7.5.6)$$

What is the value of

$$\int_K (\lambda_1(x))^p (\lambda_2(x))^q \, dx, \quad p, q \in \mathbb{N}_0,$$

for a planar triangle K with area $|K|$.

(Q2.7.5.43.B) Let b_K^1, \dots, b_K^4 be the local shape functions for the finite element space $\mathcal{S}_1^0(\mathcal{M})$ and a rectangle K with vertices

$$\mathbf{a}_K^1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathbf{a}_K^2 = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}, \quad \mathbf{a}_K^3 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}, \quad \mathbf{a}_K^4 = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}.$$

What is the value of

$$\int_K (b_K^1(x))^p (b_K^3(x))^q \, dx, \quad p, q \in \mathbb{N}_0 ?$$

Hint. The basis functions are products of functions of x_1 and function of x_2 . This leads to integrals you have already seen in the proof of Lemma 2.7.5.5.

(Q2.7.5.43.C) Outline a way to create (in LEHRFEM++) a vector of pairs of pointers to **POINT** objects with each pair corresponding to the endpoints of an edge of a 2D hybrid mesh.

(Q2.7.5.43.D) Recall the following definition of the order of a local quadrature rule:

Definition 2.7.5.29. Order of a local quadrature rule

A local quadrature rule according to Def. 2.7.5.9 is said to be of **order** $q \in \mathbb{N}$, if

- for a simplex K (triangle tetrahedron) it is exact for all *polynomials* $f \in \mathcal{P}_{q-1}(\mathbb{R}^d)$,
- for a tensor product element K (rectangle, brick) it is exact for all *tensor product polynomials* $f \in \mathcal{Q}_{q-1}(\mathbb{R}^d)$.

Based on Def. 2.7.5.29 determine the minimal order of a quadrature rule on the *unit square* that is exact for all polynomials in $\mathcal{P}_p(\mathbb{R}^2)$.

(Q2.7.5.43.E) The following three pairs $(\hat{\omega}_\ell, \hat{\zeta}_\ell)$, $\ell = 1, 2, 3$, of weights and nodes define the three-point quadrature of **order 3** on the reference triangle \hat{K} :

$$\left\{ \left(\frac{1}{3}, \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} \right), \left(\frac{1}{3}, \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} \right), \left(\frac{1}{3}, \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \right) \right\}. \quad (2.7.5.35)$$

We construct a quadrature rule on the unit square \square by applying (2.7.5.35) on the two triangles created by splitting \square into two triangles along the diagonal connecting $\mathbf{0}$ and $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

- Describe the pairs of weights and nodes for the resulting quadrature rule on \square .
- What is the order of the new quadrature rule?

△

2.7.6 Treatment of Essential Boundary Conditions



Video tutorial for Section 2.7.6: Treatment of Essential Boundary Conditions: (38 minutes)
[Download link](#), [tablet notes](#)

According to the terminology introduced in Section 1.9, we call those boundary conditions **essential** that are imposed on the functions in the trial space of variational problems. For second order elliptic boundary value problems and the variational formulations discussed in Section 1.8, essential boundary conditions are synonymous to Dirichlet boundary conditions. Now we elaborate how to handle non-zero (non-homogeneous) Dirichlet boundary conditions within finite element Galerkin discretization.

Recall the variational formulation of a *non-homogeneous* Dirichlet boundary value problem from Ex. 1.8.0.2:

$$\begin{aligned} u \in H^1(\Omega) : \quad & \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in H_0^1(\Omega) . \\ u = g \text{ on } \partial\Omega : \quad & -\operatorname{div}(\kappa(x) \mathbf{grad} u) = f \quad \text{in } \Omega , \quad u = g \quad \text{on } \partial\Omega , \end{aligned} \quad (1.8.0.5)$$

with (admissible \rightarrow § 1.9.0.6) Dirichlet data $g \in C^0(\partial\Omega)$. This problem fits the abstract notation from Def. 1.4.1.6 for a linear variational problem posed on an affine space:

$$u \in \widehat{V} : \quad a(u, v) = \ell(v) \quad \forall v \in V_0 , \quad (1.4.1.7)$$

with obvious meanings of a (bilinear form on $H_0^1(\Omega) \times H_0^1(\Omega)$), ℓ (linear form on $H_0^1(\Omega)$), \widehat{V} (affine space $\widehat{g} + H_0^1(\Omega)$), and $V_0 := H_0^1(\Omega)$.

Recall from Section 1.9 that Dirichlet boundary conditions are **essential boundary conditions**, that is, they are built into the trial space \widehat{V}

Now we will learn, how discrete trial spaces and algorithms have to be modified in order to accommodate essential boundary conditions.

§2.7.6.1 (Offset functions for Lagrangian finite element methods) Remember the offset function technique, explained in § 1.4.1.9, that can be used to convert (1.8.0.5) into a variational problem with the same trial and test space:

$$(1.8.0.5) \Leftrightarrow \begin{aligned} w \in H_0^1(\Omega) : \quad & \int_{\Omega} \kappa(x) \mathbf{grad} w \cdot \mathbf{grad} v \, dx \\ & = \int_{\Omega} -\kappa(x) \mathbf{grad} u_0 \cdot \mathbf{grad} v + f v \, dx \quad \forall v \in H_0^1(\Omega) , \end{aligned} \quad (2.7.6.2)$$

with **offset function** $u_0 \in H^1(\Omega)$ satisfying

$$u_0 = g \quad \text{on } \partial\Omega$$

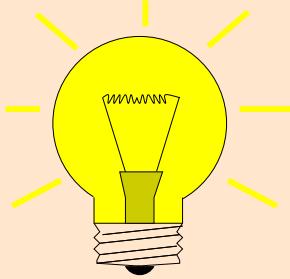
We adapt the offset function policy to finite element Galerkin discretization by generalizing the 1D example from Rem. 2.3.3.15 to $d = 2, 3$:

Remember: we already know finite element subspaces $V_{0,h} := S_{p,0}^0(\mathcal{M}) \subset H_0^1(\Omega)$, see § 2.6.2.8:

$$S_{p,0}^0(\mathcal{M}) := S_p^0(\mathcal{M}) \cap H_0^1(\Omega) = \operatorname{Span}\{b_h^j : p^j \in \Omega \text{ (interior node)}\} , \quad (2.6.2.9)$$

that is, the nodal basis of $S_{p,0}^0(\mathcal{M})$ is obtained by dropping all those nodal basis functions (global shape functions) of $S_p^0(\mathcal{M})$ that belong to geometric entities $\subset \partial\Omega$, see § 2.4.3.7 for an example.

Finite element offset functions



Idea (inspired by choice in 1D), Rem. 2.3.3.15):

use offset function $u_0 \in V_h := \mathcal{S}_p^0(\mathcal{M})$

locally supported near the boundary:

\Updownarrow

use offset function in the span of global basis functions associated with geometric entities on $\partial\Omega$

The rules governing the supports of global shape functions (\rightarrow Section 2.5.3, Section 2.5.3) tell us the maximal support of finite element boundary offset functions:



$$\text{supp}(u_0) \subset \bigcup_{K \in \mathcal{M}} \{K \in \mathcal{M} : \bar{K} \cap \partial\Omega \neq \emptyset\}. \quad (2.7.6.4)$$

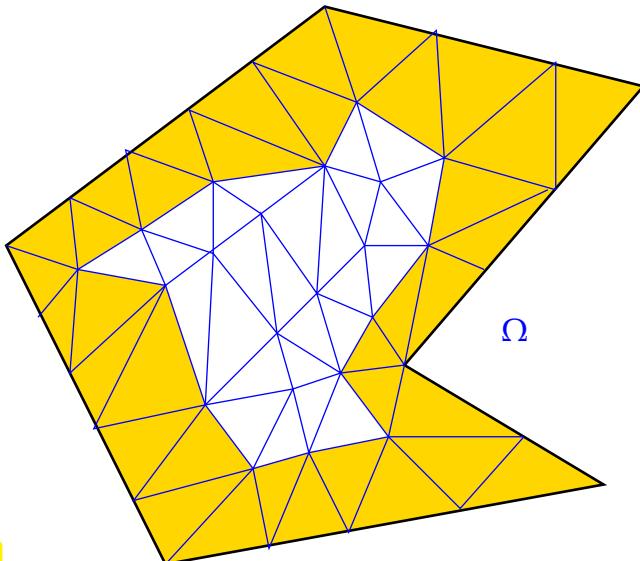


Fig. 147

(2.7.6.4) is a consequence of the local support property of finite element basis functions, see Ex. 2.5.3.2.

◀ Largest possible support of u_0 on a triangular mesh.

As in Rem. 2.3.3.15, a small support of u_0 will ensure that changes to the right-hand-side vector will be confined to only a small percentage of its entries.

EXAMPLE 2.7.6.5 (Offset functions for linear Lagrangian FE) Now we apply this idea to the special case of linear Lagrangian finite elements, which will yield a direct generalization of the choice of an offset function in 1D presented in Rem. 2.3.3.15.

For $V_h = \mathcal{S}_1^0(\mathcal{M})$ and Dirichlet data $g \in C^0(\partial\Omega)$ use

$$u_0 = \sum_{x \in \mathcal{V}(\mathcal{M}) \cap \partial\Omega} g(x) b_h^x \quad (2.7.6.6)$$

b_h^x $\hat{=}$ tent function associated with node $x \in \mathcal{V}(\mathcal{M})$,
cf. Section 2.4.3. (2.7.6.6) generalizes (2.3.3.16) to 2D.

Note that this offset functions vanishes in all interior vertices: $u_0(x) = 0$ for all $x \in \mathcal{V}(\mathcal{M}) \cap \Omega$.

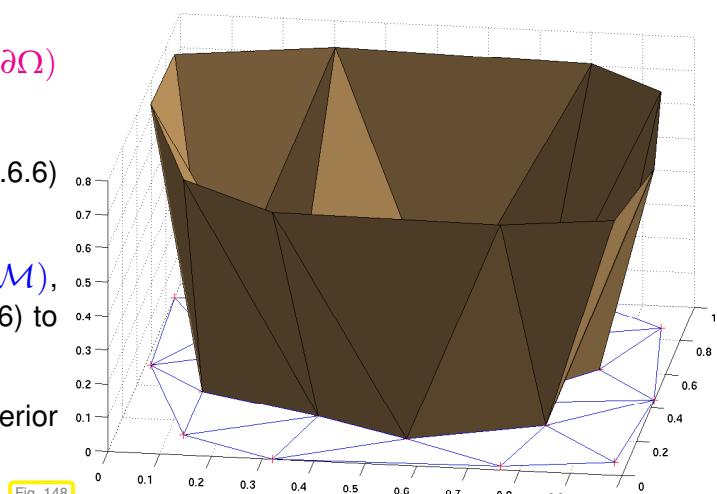


Fig. 148

Remark 2.7.6.7 (Approximate Dirichlet boundary conditions) Be aware that the formula (2.7.6.6) ac-

tually violates the strict trace condition, because in general

$$u_0 \neq g \quad \text{on } \partial\Omega.$$

Rather, u_0 is a *piecewise linear interpolant* of the Dirichlet data $g \in C^0(\partial\Omega)$. Therefore, another *approximation* comes into play when enforcing Dirichlet boundary conditions by means of piecewise polynomial offset functions. \square

§2.7.6.8 (Implementation of non-homogeneous Dirichlet b.c. for linear FE: Elimination) Consider the linear variational problem

$$\begin{aligned} u \in H^1(\Omega) \\ u = g \text{ on } \partial\Omega \end{aligned} : \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in H_0^1(\Omega), \quad (1.8.0.5)$$

its finite-element Galerkin discretization by means of p.w. linear Lagrangian finite elements, and, for the sake of simplicity of presentation, assume the following ordering of the nodal basis functions, see Fig. 68

$$\begin{aligned} \mathcal{B}_0 := \{b_h^1, \dots, b_h^N\} &\triangleq \text{nodal basis of } \mathcal{S}_{1,0}^0(\mathcal{M}), \\ &\quad (\text{tent functions associated with interior nodes}) \\ \mathcal{B} := \mathcal{B}_0 \cup \{b_h^{N+1}, \dots, b_h^M\} &\triangleq \text{nodal basis of } \mathcal{S}_1^0(\mathcal{M}) \\ &\quad (\text{extra basis functions associated with nodes } \in \partial\Omega). \end{aligned}$$

Here: $M := \#\mathcal{V}(\mathcal{M}) = \dim \mathcal{S}_1^0(\mathcal{M})$,

$N := \#\{x \in \mathcal{V}(\mathcal{M}), x \notin \partial\Omega\} = \dim \mathcal{S}_{1,0}^0(\mathcal{M})$ (no. of interior nodes).

$$\begin{aligned} \mathbf{A}_0 \in \mathbb{R}^{N,N} &\triangleq \text{Galerkin matrix for discrete trial/test space } \mathcal{S}_{1,0}^0(\mathcal{M}), \\ \mathbf{A} \in \mathbb{R}^{M,M} &\triangleq \text{Galerkin matrix for discrete trial/test space } \mathcal{S}_1^0(\mathcal{M}). \end{aligned}$$

This gives rise to a **block-partitioning** of the Galerkin matrix \mathbf{A} ,

$$\Rightarrow \mathbf{A} = \begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_{0\partial} \\ \mathbf{A}_{0\partial}^T & \mathbf{A}_{\partial\partial} \end{bmatrix}, \quad \begin{aligned} \mathbf{A}_{0\partial} &:= \left(\mathbf{a}(b_h^j, b_h^i) \right)_{\substack{i=1, \dots, N \\ j=N+1, \dots, M}} \in \mathbb{R}^{N, M-N}, \\ \mathbf{A}_{\partial\partial} &:= \left(\mathbf{a}(b_h^j, b_h^i) \right)_{\substack{i=N+1, \dots, M \\ j=N+1, \dots, M}} \in \mathbb{R}^{M-N, M-N}. \end{aligned} \quad (2.7.6.9)$$

If $u_0 \in \mathcal{S}_1^0(\mathcal{M})$ is chosen according to (2.7.6.6), then

$$u_0 \in \text{Span}\{b_h^{N+1}, \dots, b_h^M\} \Leftrightarrow u_0 = \sum_{j=N+1}^M \gamma_j b_h^j,$$

with suitable coefficients $\gamma_j, j = 1, \dots, M - N$, defined, for instance, by (2.7.6.6). We can now plug this into the discrete variational problem for the “correction” $w_h \in V_{0,h}$, which in abstract form reads

$$w_h \in V_{0,h}: \quad \mathbf{a}(w_h, v_h) = \ell(v_h) - \mathbf{a}(u_0, v_h) \quad \forall v_h \in V_{0,h},$$

where we used the abbreviation \mathbf{a} for the bilinear form in Eq. (1.8.0.5) and ℓ for the right hand side linear form. Thus, we get with $w_h = \sum_{j=1}^N \nu_j b_h^j$

$$\sum_{j=1}^N \nu_j \mathbf{a}(b_h^j, b_h^i) = \ell(b_h^i) - \sum_{k=N+1}^M \gamma_k \mathbf{a}(b_h^k, b_h^i), \quad i = 1, \dots, N,$$

which means that the basis expansion coefficient vector $\vec{\nu} = [\nu_1, \dots, \nu_N]^\top$ of the finite element approximation $w_h \in \mathcal{S}_{1,0}^0(\mathcal{M})$ of $w \in H_0^1(\Omega)$ from (2.7.6.2) solves the linear system of equations ($\vec{\gamma} = [\gamma_1, \dots, \gamma_{M-N}]^\top$)

$$\boxed{\mathbf{A}_0 \vec{\nu} = \vec{\phi} - \mathbf{A}_{0\partial} \vec{\gamma}}. \quad (2.7.6.10)$$

- Summing up, non-homogeneous Dirichlet boundary data are taken into account through a **modified right hand side vector**. □

Supplement 2.7.6.11 (Alternative consideration leading to (2.7.6.10)) We may want to keep the original size $M \times M$ of the linear system and modify it in a way so that (2.7.6.10) is embedded into it. The rationale for doing this will become clear in § 2.7.6.13.

- ❶ First ignore essential boundary conditions and assemble the linear system of equations arising from the discretization of a on the (larger) FE space $\mathcal{S}_1^0(\mathcal{M})$:

$$\begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_{0\partial} \\ \mathbf{A}_{0\partial}^T & \mathbf{A}_{\partial\partial} \end{bmatrix} \begin{bmatrix} \vec{\mu}_0 \\ \vec{\mu}_\partial \end{bmatrix} = \begin{bmatrix} \vec{\phi} \\ \vec{\varphi}_\partial \end{bmatrix}. \quad (2.7.6.12)$$

Here,

$\vec{\mu}_0 \triangleq$ coefficients for *interior* basis functions b_h^1, \dots, b_h^N

$\vec{\mu}_\partial \triangleq$ coefficient for basis functions b_h^{N+1}, \dots, b_h^M associated with nodes located on $\partial\Omega$.

- ❷ We realize that the coefficient vector of (2.7.6.12) is that of a FE approximation of u



$\vec{\mu}_\partial$ known = values of g at boundary nodes: $\vec{\mu}_\partial = \vec{\gamma}$

- ❸ Moving known quantities in (2.7.6.12) to the right hand side yields (2.7.6.10). □

§2.7.6.13 (Imposing essential constraints in LEHRFEM++) LEHRFEM++ provides tools for solving partitioned linear like (2.7.6.12),

$$\mathbf{A}\vec{\mu} := \begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_{0\partial} \\ \mathbf{A}_{0\partial}^T & \mathbf{A}_{\partial\partial} \end{bmatrix} \begin{bmatrix} \vec{\mu}_0 \\ \vec{\gamma} \end{bmatrix} = \begin{bmatrix} \vec{\phi} \\ * \end{bmatrix}, \quad (2.7.6.14)$$

where the matrix $\mathbf{A} \in \mathbb{R}^{M,M}$, and the vectors $\vec{\gamma} \in \mathbb{R}^{M-N}$ and $\vec{\phi} \in \mathbb{R}^N$ are given. We want to find the solution component $\vec{\mu}_0 \in \mathbb{R}^N$ and do not take into account the $*$ -part of the right-hand-side vector.

In practice the basis functions will not be numbered in a way to yield a nice block-partitioning as in (2.7.6.14). Rather the fixed degrees of freedom corresponding to components of $\vec{\gamma}$ may rather erratically be scattered among all d.o.f.s. They will usually be identified by a **predicate**, a mapping $\{1, \dots, M\} \mapsto \{\text{true}, \text{false}\}$, realized by a suitable functor in a C++ code.

In LEHRFEM++ there are two pre-processing functions → [GITHUB](#) to transform linear systems of equations with fixed solution components. Both take a predicate argument of a type **SELECTOR**, which must feature an evaluation operator of the form

```
std::pair<bool, SCALAR> operator() (unsigned int dof_idx) const;
```

accepting the index $j \in \{0, \dots, M-1\}$ of a d.o.f. as an argument and returning the tuple (true, μ_j) , if this d.o.f. has a fixed value μ_j , $(\text{false}, *)$ otherwise. The two functions are

```
(I) template <typename SCALAR, typename SELECTOR,
           typename RHSVECTOR>
void FixFlaggedSolutionComponents (SELECTOR &&selectvals,
                                   ld::assemble::COOMatrix<SCALAR> &A, RHSVECTOR &b);
```

which changes A and b such that, in the case of (2.7.6.14), they correspond to the modified linear system

$$\begin{bmatrix} \mathbf{A}_0 & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \vec{\mu}_0 \\ \vec{\mu}_\partial \end{bmatrix} = \begin{bmatrix} \vec{\phi} - \mathbf{A}_{0\partial} \vec{\gamma} \\ \vec{\gamma} \end{bmatrix}. \quad (2.7.6.15)$$

```
(II) template <typename SCALAR, typename SELECTOR,
           typename RHSVECTOR>
void FixFlaggedSolutionCompAlt(SELECTOR &&selectvals,
                                lf::assemble::COOMatrix<SCALAR> &A, RHSVECTOR &b);
```

which changes the matrix and right hand side of the linear system of equations $\vec{A}\vec{\mu} = \vec{b}$ in a way that would transform (2.7.6.14) into

$$\begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_{0\partial} \\ \mathbf{O} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \vec{\mu}_0 \\ \vec{\mu}_\partial \end{bmatrix} = \begin{bmatrix} \vec{\varphi} \\ \vec{\gamma} \end{bmatrix}. \quad (2.7.6.16)$$

A possible implementation of `FixFlaggedSolutionCompAlt()` is given next to show the manipulation of the triplets representing the sparse matrix **A**:

C++ code 2.7.6.17: Transformation function (2.7.6.14) → (2.7.6.16) → [GITHUB](#)

```
1 template <typename SCALAR, typename SELECTOR, typename RHSVECTOR>
2 void FixFlaggedSolutionCompAlt(SELECTOR &&selectvals, COOMatrix<SCALAR> &A,
3 RHSVECTOR &b) {
4     const lf::assemble::size_type N(A.cols());
5     LF_ASSERT_MSG(A.rows() == N, "Matrix must be square!");
6     LF_ASSERT_MSG(N == b.size(), "Mismatch N = " << N << " <-> b.size() = " <<
7         b.size());
8     // I: Set components of right-hand-side vector to prescribed values
9     for (lf::assemble::gdof_idx_t k = 0; k < N; ++k) {
10         const auto selval{selectvals(k)};
11         if (selval.first) b[k] = selval.second;
12     }
13     // II: Set rows of the sparse matrix corresponding
14     // to the fixed solution components to zero
15     typename lf::assemble::COOMatrix<SCALAR>::TripletVec::iterator new_last =
16     std::remove_if(
17         A.triplets().begin(), A.triplets().end(),
18         [&fixed_comp_flags]);
19     typename lf::assemble::COOMatrix<SCALAR>::Triplet &triplet) {
20         return (fixed_comp_flags[triplet.row()]);
21     });
22     // Adjust size of triplet vector
23     A.triplets().erase(new_last, A.triplets().end());
24     // III: Add Unit diagonal entries corresponding to fixed components
25     for (lf::assemble::gdof_idx_t dofnum = 0; dofnum < N; ++dofnum) {
26         if (selectvals(dofnum).first) A.AddToEntry(dofnum, dofnum, 1.0);
27     }
```

It is worth trying to understand this code! The **SELECTOR** type must provide an evaluation operator

```
std::pair<bool, SCALAR> operator()(lf::assemble::gdof_idx_t);
```

which, for each d.o.f. referenced by its index returns a flag and a value. If the flag is set, that d.o.f. has to be fixed to the associated value.

Remark 2.7.6.18 (Ordering of global shape functions required?) We emphasize that the ordering of the global shape functions underlying the presentation in § 2.7.6.8 and, in particular (2.7.6.9), (2.7.6.12), (2.7.6.14), (2.7.6.15), and (2.7.6.16), was just imposed “for the sake of simplicity of presentation”. In fact, both LEHRFEM++ functions `FixFlaggedSolutionComponents()` and `FixFlaggedSolutionCompAlt()` work regardless of the numbering of global shape functions.

The only information they need is that passed through `selectvals` and involves flags for d.o.f.s to be fixed and the corresponding values.

EXAMPLE 2.7.6.19 (Solution of a Dirichlet BVP with LEHRFEM++) We examine a simple LEHRFEM++ code capable of solving the Dirichlet boundary value problem

$$-\Delta u = 0 \quad \text{in } \Omega \subset \mathbb{R}^2, \quad u = g \quad \text{on } \partial\Omega.$$

The polygonal domain Ω is specified through a triangular planar finite element mesh \mathcal{M} and a finite element Galerkin discretization based on $S_1^0(\mathcal{M})$ is employed. The Dirichlet boundary conditions are treated as discussed in § 2.7.6.8 using the LEHRFEM++ tools introduced in § 2.7.6.13.

C++ code 2.7.6.20: Solving a Dirichlet BVP $-\Delta u = 0$ with LEHRFEM++ → GITHUB

```

2 // Initialization of local-to-global index mapping for linear finite
3 // elements
4 If ::assemble::UniformFEDofHandler dof_handler(
5     mesh_p, {{If ::base::RefEl::kPoint(), 1}});
6 // Query dimension of the finite element space, equal to the number of
7 // nodes
8 size_type N_dofs(dof_handler.NumDofs());
9 // Matrix in triplet format holding temporary Galerkin matrix
10 If ::assemble::COOMatrix<double> mat(N_dofs, N_dofs);
11 // Initialize objects for local computation of element matrices for
12 // -Δ
13 LinFELaplaceElemMatProvider loc_mat_laplace {};
14
15 // Building the Galerkin matrix (trial space = test space)
16 // for the pure Neumann Laplacian, assembly over cells
17 mat = If ::assemble::AssembleMatrixLocally<If ::assemble::COOMatrix<double>>(
18     0, dof_handler, loc_mat_laplace);
19 // Zero right-hand side vector
20 Eigen::VectorXd rhsvec(N_dofs);
21 rhsvec.setZero();
22
23 // Treatment of Dirichlet boundary conditions  $g = u|_{\partial\Omega}$ 
24 // Flag all nodes on the boundary (and only those)
25 auto bd_flags{If ::mesh::utils::flagEntitiesOnBoundary(mesh_p, 2)};
26 // Set up predicate: Run through all global shape functions and check
27 // whether
28 // they are associated with an entity on the boundary, store Dirichlet
29 // data.
30 std::vector<std::pair<bool, double>> ess_dof_select {};
31 for (If ::assemble::gdof_idx_t dofnum = 0; dofnum < N_dofs; ++dofnum) {
32     const If ::mesh::Entity &dof_node{dof_handler.Entity(dofnum)};
33     const Eigen::Vector2d node_pos{
34         If ::geometry::Corners(*dof_node.Geometry()).col(0)};
35     const double g_val = u_sol(node_pos);
36     if (bd_flags(dof_node)) {
37         // Dof associated with a entity on the boundary: "essential dof"
38         // The value of the dof should be set to the value of the function
39         //  $u$  at the location of the node.
40         ess_dof_select.emplace_back(true, g_val);
41     } else {
42         // Interior node, also store value of solution for comparison
43         // purposes
44         ess_dof_select.emplace_back(false, g_val);
45     }
46 }
47
48 // modify linear system of equations
49 If ::assemble::FixFlaggedSolutionCompAlt<double>(
50     [&ess_dof_select](glb_idx_t dof_idx) -> std::pair<bool, double> {
51         return ess_dof_select[dof_idx];
52     }
53 )

```

```

45     } ,
46     mat, rhsvec);
47
48 // Convert the matrix from triplet format to CRS format
49 const Eigen::SparseMatrix<double> A(mat.makeSparse());

```

Review question(s) 2.7.6.21 (Treatment of essential boundary conditions)

(Q2.7.6.21.A) Explain the **offset-function technique** for dealing with essential boundary conditions imposed on a 2nd-order elliptic variational problem.

(Q2.7.6.21.B) What is a “convenient” offset function when performing the Galerkin discretization of a pure Dirichlet boundary value problem based on the quadratic Lagrangian finite-element space $S_2^0(\mathcal{M})$ and a simplicial mesh \mathcal{M} ?

(Q2.7.6.21.C) On a polyognally bounded domain $\Omega \subset \mathbb{R}^2$ we solve the Dirichlet problem

$$-\Delta u = 0 \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega,$$

with $g \in C^0(\partial\Omega)$.

We use a finite element Galerkin discretization based on $S_1^0(\mathcal{M})$, where \mathcal{M} is the triangular mesh drawn beside.

We employ the elimination of essential boundary conditions based on the offset function technique.

How many entries of the right-hand side vector of the finite-element linear system of equations will change, when the data g change?

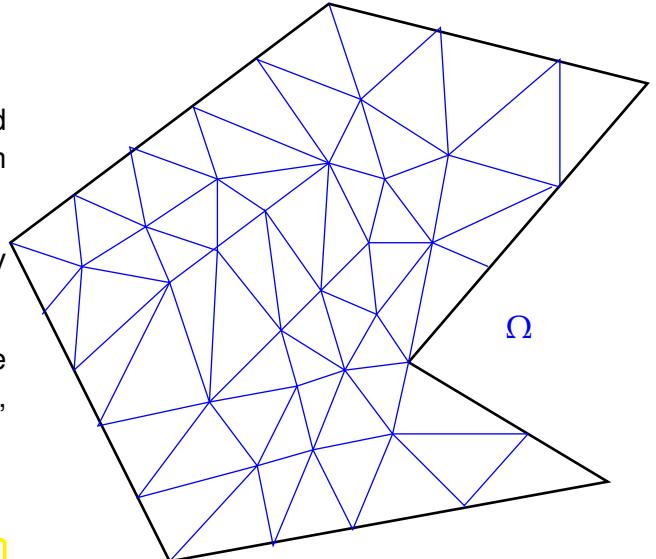


Fig. 149

(Q2.7.6.21.D) Sketch the implementation of a class

```

class SolveLaplaceBVP {
public:
    SolveLaplaceBVP(std::shared_ptr<const If::assemble::DofHandler>
                     dofh_p);
    ~SolveLaplaceBVP() = default;

    template <typename FUNCTOR>
    Eigen::VectorXd solveLaplaceBVP(FUNCTOR &&g) const;
private:
    ...
};

```

The constructor takes a shared pointer to a **If::assemble::DofHandler** object associated belonging to the finite element space $S_1^0(\mathcal{M})$. The method `solveLaplaceBVP()` takes a functor argument of

type `std::function<double (Eigen::Vector2D)>`, which supplies the boundary data g . It is supposed to return the basis expansion coefficient vector (with respect to the standard tent function basis) of the finite element solution $\in \mathcal{S}_1^0(\mathcal{M})$ of the Dirichlet boundary value problem

$$-\Delta u = 0 \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega.$$

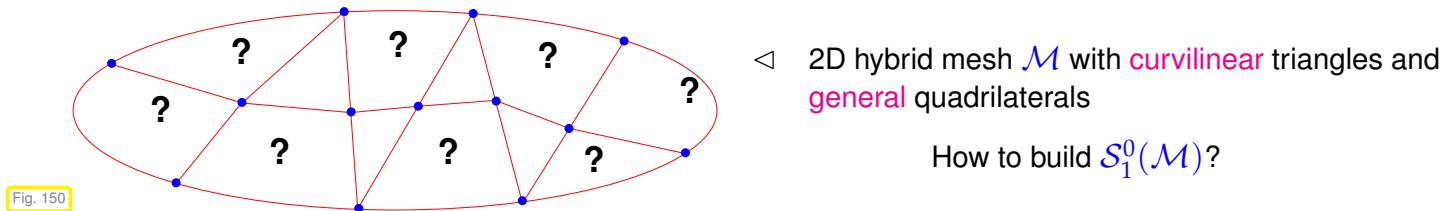
△

2.8 Parametric Finite Element Methods

1.  Video tutorial for Section 2.8: Parametric Finite Element Methods (I): (45 minutes)
[Download link](#), [tablet notes](#)
2.  Video tutorial for Section 2.8: Parametric Finite Element Methods (II): (56 minutes)
[Download link](#), [tablet notes](#)

Already in Section 2.7.5 we exploited (affine) transformation (\rightarrow Def. 2.7.5.13) to a reference cell in order to obtain numerical quadrature formulas (2.7.5.10) for all cells of a mesh in one fell swoop. In this section we will witness the full power of this idea of using **transformations to reference cells**. It will enable us to extend the range of Lagrangian finite element spaces significantly, and will also be a key element in algorithm design (The entire LEHRFEM++ finite element library relies on the construction of finite elements by transformation).

We need to enhance the flexibility of finite element spaces. For instance, the construction of Lagrangian finite element spaces done in Section 2.6 cannot cope with the following situation:



2.8.1 Affine Equivalence

We recall a transformation that we have already seen in Lemma 2.7.5.14, namely the affine transformation of triangles (2.7.5.15).

Lemma 2.7.5.14. Affine transformation of triangles

For any non-degenerate triangle $K \subset \mathbb{R}^2$ ($|K| > 0$) with numbered vertices there is a **unique** affine transformation Φ_K , $\Phi_K(\hat{x}) = F_K \hat{x} + \tau_K$ (\rightarrow Def. 2.7.5.13), with $K = \Phi_K(\hat{K})$ and preserving the numbering of the vertices.

This lemma conveys the following insight:

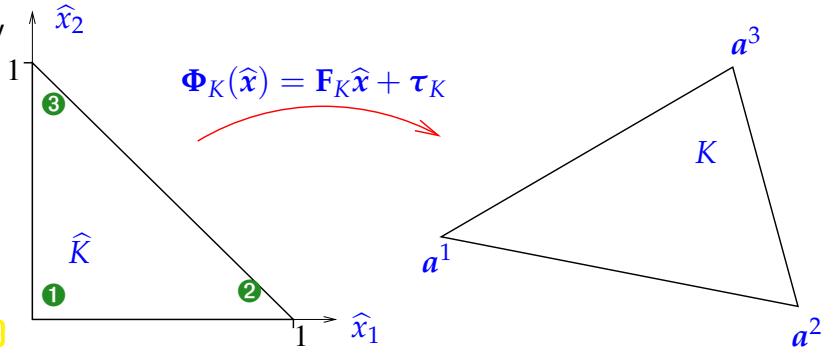
- All cells of a triangular mesh are affine images of the “unit triangle” $\hat{K} := \text{convex}\left\{\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right\}$

The affine mapping Φ_K from the “Unit triangle” \hat{K} to a general triangle K is easily determined:
For $K = \text{convex}\{\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3\}$ it reads

$$\Phi_K(\hat{x}) := \mathbf{F}_K \hat{x} + \boldsymbol{\tau}_K ,$$

with

$$\mathbf{F}_K := \begin{bmatrix} \mathbf{a}_2^2 - \mathbf{a}_1^1 & \mathbf{a}_1^3 - \mathbf{a}_1^1 \\ \mathbf{a}_2^2 - \mathbf{a}_2^1 & \mathbf{a}_2^3 - \mathbf{a}_2^1 \end{bmatrix}, \quad \boldsymbol{\tau}_K = [\mathbf{a}_1^1]$$



§2.8.1.1 (Pullback of functions) In a natural way, a transformation of domains induces a transformation of the functions defined on them:

Definition 2.8.1.2. Pullback

Given domains $\Omega, \hat{\Omega} \subset \mathbb{R}^d$ and a bijective mapping $\Phi : \hat{\Omega} \mapsto \Omega$, the **pullback** $\Phi^* u : \hat{\Omega} \mapsto \mathbb{R}$ of a function $u : \Omega \mapsto \mathbb{R}$ is a function on $\hat{\Omega}$ defined by

$$(\Phi^* u)(\hat{x}) := u(\Phi(\hat{x})), \quad \hat{x} \in \hat{\Omega} .$$

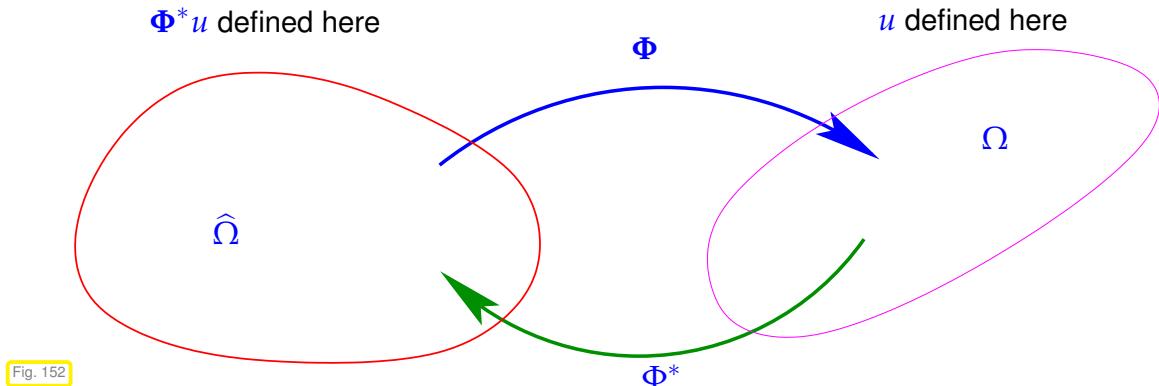
- ♦ Implicitly, we used the pullback of integrands when defining quadrature rules through transformation, see (0.3.2.32) and (2.7.5.22):

$$\int_K f(x) dx = \int_{\hat{K}} f(\Phi(\hat{x})) |\det \mathbf{F}_K| d\hat{x} . \quad (2.7.5.22)$$

- ♦ Obviously, the pullback Φ^* induces a *linear mapping* between spaces of functions on Ω and $\hat{\Omega}$, respectively:

$$\Phi^*(\alpha f + \beta g) = \alpha \Phi^* f + \beta \Phi^* g \quad \forall f, g : \Omega \rightarrow \mathbb{R}, \quad \alpha, \beta \in \mathbb{R} . \quad (2.8.1.3)$$

The following picture can aid understanding: the pullback Φ_K^* maps in the “opposite direction” compared to Φ_K :



In the context of numerical quadrature, when wondering whether transformation preserved the order of a quadrature rule, we made the following observation, cf. (2.7.5.32):

Lemma 2.8.1.4. Preservation of polynomials under affine pullback

If $\Phi : \mathbb{R}^d \mapsto \mathbb{R}^d$ is an **affine** (linear) transformation (\rightarrow Def. 2.7.5.13), then

$$\Phi^*(\mathcal{P}_p(\mathbb{R}^d)) = \mathcal{P}_p(\mathbb{R}^d) \quad \text{and} \quad \Phi^*(\mathcal{Q}_p(\mathbb{R}^d)) = \mathcal{Q}_p(\mathbb{R}^d).$$

In fact, Lemma 2.7.5.14 reveals another reason for the preference for polynomials in building discrete Galerkin spaces.

Proof. (of Lemma 2.7.5.14) Since the pullback is linear, we only need to study its action on the (monomial) basis $x \mapsto x^\alpha$, $\alpha \in \mathbb{N}_0^d$ of $\mathcal{P}_p(\mathbb{R}^d)$, see Def. 2.5.2.2 and the explanations on multi-index notation (2.5.2.3).

Then resort to induction w.r.t. degree p .

$$\Phi_K^*(x^\alpha) = \Phi_K^*(x_1) \cdot \Phi_K^*\left(\underbrace{x^{\alpha'}}_{\in \mathcal{P}_{p-1}(\mathbb{R}^d)}\right) = \underbrace{\left(\sum_{l=1}^d (\mathbf{F})_{1l} \hat{x}_l + \tau_1\right)}_{\in \mathcal{P}_1(\mathbb{R}^d)} \cdot \underbrace{\Phi_K^*(x^{\alpha'})}_{\in \mathcal{P}_{p-1}(\mathbb{R}^d)} \in \mathcal{P}_p(\mathbb{R}^d),$$

with $\alpha' := (\alpha_1 - 1, \alpha_2, \dots, \alpha_d)$, where we assumed $\alpha_1 > 0$. Here, we have used the induction hypothesis to conclude $\Phi_K^*(x^{\alpha'}) \in \mathcal{P}_{p-1}(\mathbb{R}^d)$. \square

§2.8.1.5 (Pullback of local shape functions for Lagrangian finite elements) We start with simple observation: Consider the lowest-order Lagrangian finite element space $\mathcal{S}_1^0(\mathcal{M})$ and a triangle $K \in \mathcal{M}$. Let \hat{K} be the unit triangle and Φ_K the unique affine mapping $\hat{K} \mapsto K$. We write

- b_K^1, b_K^2, b_K^3 for the (standard) local shape functions on K , as defined in Ex. 2.5.3.6.
- $\hat{b}^1, \hat{b}^2, \hat{b}^3$ for the (standard) local shape functions on \hat{K} ,
(In each case we deal with the barycentric coordinate functions introduced in § 2.4.5.2!)

We find a fundamental relationship with respect to the pullback Φ_K^* :

$$\hat{b}^i = \Phi_K^* b_K^i \quad \Leftrightarrow \quad \hat{b}^i(\hat{x}) = b_K^i(x), \quad x = \Phi_K(\hat{x}). \quad (2.8.1.6)$$

Of course, we assume that Φ_K respects the local numbering of the vertices of \hat{K} and K : $\Phi_K(\hat{a}^i) = a^i$, $i = 1, 2, 3$.

The proof of (2.8.1.6) is straightforward: both $\Phi_K^* b_K^i$ (by Lemma 2.8.1.4) and \hat{b}^i are (affine) linear functions that attain the same values at the vertices of \hat{K} . Hence, they have to agree. In fact (2.8.1.6) holds true for **all** simplicial Lagrangian finite element spaces.

Lemma 2.8.1.7. Affine equivalence of Lagrangian finite elements on simplicial meshes

Let the global shape functions for $\mathcal{S}_p^0(\mathcal{M})$, $p \in \mathbb{N}$, \mathcal{M} a simplicial mesh, be defined by the cardinal basis property (2.6.1.4) with respect to the canonical choice of interpolation nodes, see Ex. 2.6.1.2 and Ex. 2.6.1.7. Then for any two simplices $K_1, K_2 \in \mathcal{M}$ there is an **affine mapping** $\Phi : K_1 \rightarrow K_2$, $K_2 = \Phi(K_1)$, such that

$$\Phi^* b_{K_2}^j = b_{K_1}^j, \quad j \in \{1, \dots, Q\}, \quad (2.8.1.8)$$

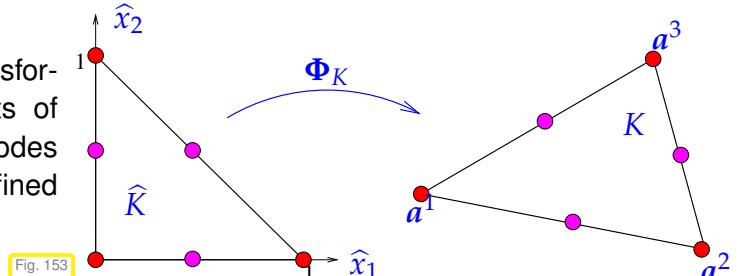
where $b_{K_1}^j, b_{K_2}^j$, $j = 1, \dots, Q$, are the local shape functions on K_1 and K_2 , respectively.

Proof. (of (2.8.1.6) for general Lagrangian finite element spaces) First, recall the definition of global shape functions and also local shape functions for $\mathcal{S}_p^0(\mathcal{M})$, $p \in \mathbb{N}$, by means of the conditions (2.6.1.4) at interpolation nodes, see Ex. 2.6.1.2 for $p = 2$.

Now write $\mathbf{p}_K^i \hat{=} (\text{local}) \text{ interpolation nodes on triangle } K,$
 $\widehat{\mathbf{p}}^i \hat{=} (\text{local}) \text{ interpolation nodes on unit triangle } \widehat{K}.$

Observe: Assuming a matching numbering we have $\mathbf{p}_K^i = \Phi_K(\widehat{\mathbf{p}}^i)$, where $\Phi_K : \widehat{K} \mapsto K$ is the unique affine transformation mapping \widehat{K} onto K , see (2.7.5.15).

This is clear for $p = 2$, because affine transformations take midpoints of edges to midpoints of edges. The same applies to the interpolation nodes for higher degree Lagrangian finite elements defined in Ex. 2.6.1.7.



For Lagrangian finite element spaces the local shape functions $b_K^i \in \mathcal{P}_p(\mathbb{R}^d)$, $\widehat{b}^i \in \mathcal{P}_p(\mathbb{R}^d)$, $i = 1, \dots, Q$, on K and \widehat{K} , respectively, are *uniquely defined* by the interpolation conditions

$$b_K^i(\mathbf{p}_K^j) = \delta_{ij} \quad , \quad \widehat{b}^i(\widehat{\mathbf{p}}^j) = \delta_{ij}. \quad (2.8.1.9)$$

Together with $\mathbf{p}_K^i = \Phi_K(\widehat{\mathbf{p}}^i)$ this shows that $\Phi_K^* b_K^i$ satisfies the interpolation conditions (2.8.1.9) on \widehat{K} and, thus, has to agree with \widehat{b}^i . □

□

□

The property of local shape functions

$$\widehat{b}^i = \Phi_K^* b_K^i \quad \Leftrightarrow \quad \widehat{b}^i(\widehat{\mathbf{x}}) = b_K^i(\mathbf{x}), \quad \mathbf{x} = \Phi_K(\widehat{\mathbf{x}}), \quad (2.8.1.6)$$

paves the way for profound algorithmic simplifications in finite element codes. Thus it is very desirable that global basis functions of finite element spaces comply with (2.8.1.6).

Terminology: Finite element spaces satisfying (2.8.1.6) with a affine mapping (\rightarrow Def. 2.7.5.13) $\Phi_K : \widehat{K} \rightarrow K$ for every $K \in \mathcal{M}$ are called **affine equivalent**.

Remark 2.8.1.10 (Local computations based on affine equivalence) Affine equivalence can be exploited to achieve substantial reduction in computational effort for local computations. Let us consider Lagrangian finite element spaces on a simplicial mesh \mathcal{M} . We denote by $\{b_K^1, \dots, b_K^Q\}$, $Q \in \mathbb{N}$, the set of **local shape functions** for the cell $K \in \mathcal{M}$. For the same cell let $\Phi_K : \widehat{K} \rightarrow K$ stand for the affine mapping satisfying $K = \Phi_K(\widehat{K})$, where \widehat{K} is the “unit simplex”.

Frequently, in finite-element codes the following type of integrals has to be evaluated

$$\int_K F(b_K^1(\mathbf{x}), \dots, b_K^Q(\mathbf{x})) \, d\mathbf{x}, \quad \text{for some } F : \mathbb{R}^Q \rightarrow \mathbb{R}. \quad (2.8.1.11)$$

Such integrals may occur, for instance, for L^2 -type bilinear forms: for $\int_K b_K^j(\mathbf{x}) b_K^\ell(\mathbf{x}) \, d\mathbf{x}$ we have $F(\xi, \eta) := \xi \eta$.

Of course, for general F , we have to rely on **numerical quadrature**. Recall from Section 2.7.5 the definition (2.7.5.23) of local quadrature formulas via transformation from a “unit simplex” (reference cell/element \widehat{K}), where a Q -point quadrature formula with weights $\widehat{\omega}_\ell$ and nodes $\widehat{\zeta}_\ell$, $\ell = 1, \dots, Q$, is given.

$$\int_{\widehat{K}} f(\widehat{\mathbf{x}}) \, d\widehat{\mathbf{x}} \approx \sum_{\ell=1}^Q \widehat{\omega}_\ell f(\widehat{\zeta}_\ell) \quad \Rightarrow \quad \int_K f(\mathbf{x}) \, d\mathbf{x} \approx \frac{|K|}{|\widehat{K}|} \sum_{\ell=1}^Q \omega_\ell^K f(\zeta_\ell^K) \quad (2.7.5.23)$$

with $\omega_\ell^K = \widehat{\omega}_\ell$, $\zeta_\ell^K = \Phi_K(\widehat{\zeta}_\ell)$.

This means that we actually need to compute

$$\begin{aligned} \int_K F(b_K^1(\mathbf{x}), \dots, b_K^Q(\mathbf{x})) \, d\mathbf{x} &\stackrel{(0.3.2.32)}{=} \int_{\hat{K}} F(\Phi_K^* b_K^1(\hat{\mathbf{x}}), \dots, \Phi_K^* b_K^Q(\hat{\mathbf{x}})) |\det D\Phi_K(\hat{\mathbf{x}})| \, d\hat{\mathbf{x}} \\ &\stackrel{(2.8.1.6)}{=} \frac{|K|}{|\hat{K}|} \int_{\hat{K}} F(\hat{b}^1(\hat{\mathbf{x}}), \dots, \hat{b}^Q(\hat{\mathbf{x}})) \, d\hat{\mathbf{x}} \\ &\approx \frac{|K|}{|\hat{K}|} \sum_{\ell=1}^Q \hat{\omega}_\ell F(\hat{b}^1(\hat{\zeta}_\ell), \dots, \hat{b}^Q(\hat{\zeta}_\ell)), \end{aligned}$$

because, thnanks to (2.8.1.6), we can exploit the relationship between local shape functions on K and \hat{K} :

$$\Phi_K^*(b_K^i)(\hat{\zeta}^\ell) = \hat{b}^i(\hat{\zeta}^\ell) \quad \text{independent of } K! . \quad (2.8.1.12)$$

This can be exploited for the fast numerical quadrature of expressions depending on local shape functions only:

$$\int_K F(b_K^1(\mathbf{x}), \dots, b_K^Q(\mathbf{x})) \, d\mathbf{x} \approx \frac{|K|}{|\hat{K}|} \sum_{\ell=1}^Q \hat{\omega}_\ell F(\hat{b}^1(\hat{\zeta}_\ell), \dots, \hat{b}^Q(\hat{\zeta}_\ell)), \quad (2.8.1.13)$$

for any integrable function $F : \mathbb{R}^Q \mapsto \mathbb{R}$. We observe that

we can *precompute* the values $F(\hat{b}^1(\hat{\zeta}_\ell), \dots, \hat{b}^Q(\hat{\zeta}_\ell))$, $\ell = 1, \dots, P$, and store them in a table!

□

Supplement 2.8.1.14 (Barycentric representation of local shape functions) We consider Lagrangian finite element spaces on a simplicial mesh \mathcal{M} in 2D, standard reference triangle used. From Lemma 2.8.1.7 we know that we deal with an affine equivalent finite element method. This paves the way for an “affine equivalent” representation of local shape functions.

Already in (2.6.1.6) the formulas for local shape functions for $\mathcal{S}_2^0(\mathcal{M})$ ($d = 2$) were given in terms of barycentric coordinate functions λ_i , $i = 1, 2, 3$. Is this possibility coincidental? **NO!** Does

$$b_K^i = \sum_{\alpha \in \mathbb{N}_0^3, |\alpha| \leq p} \kappa_\alpha \lambda_1^{\alpha_1} \lambda_2^{\alpha_2} \lambda_3^{\alpha_3}, \quad \kappa_\alpha \in \mathbb{R}, \quad (2.7.5.1)$$

hold for any (simplicial) Lagrangian finite element space?

YES, because

$$\begin{aligned} b_K^i(\mathbf{x}) &\stackrel{(2.8.1.6)}{=} (\Phi_K^{-1})^* \left(\hat{\mathbf{x}} \mapsto \hat{b}^i(\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2) \right) \\ &= \hat{b}^i((\Phi_K^{-1})^*(\hat{\lambda}_2)(\mathbf{x}), (\Phi_K^{-1})^*(\hat{\lambda}_3)(\mathbf{x})) = \hat{b}^i(\lambda_2(\mathbf{x}), \lambda_3(\mathbf{x})), \end{aligned}$$

where $\lambda_2(\hat{\mathbf{x}}) = \hat{\mathbf{x}}_1$, $\lambda_3(\hat{\mathbf{x}}) = \hat{\mathbf{x}}_2$, $\lambda_1(\hat{\mathbf{x}}) = 1 - \hat{\mathbf{x}}_1 - \hat{\mathbf{x}}_2 \hat{=} \text{barycentric coordinate functions on } \hat{K}$, see Ex. 2.5.3.6,

$\lambda_i \hat{=} \text{barycentric coordinate functions on triangle } K$, see Fig. 75,

$\Phi_K \hat{=} \text{affine transformation} (\rightarrow \text{Def. 2.7.5.13})$, $\Phi_K(\hat{K}) = K$, see (2.7.5.15).

The above formula is a consequence of the trivial fact that for an affine transformation $\Phi_K : \hat{K} \rightarrow K$ between simplices (triangles or tetrahedra) the corresponding pullback (\rightarrow Def. 2.8.1.2) maps barycentric coordinate functions onto each other, cf. Lemma 2.8.1.7 for $p = 1$,

$$\Phi_K^*(\lambda_k) = \hat{\lambda}_k, \quad k = 1, \dots, d+1. \quad (2.8.1.15)$$

➤ By the chain rule:

$$\begin{aligned}\mathbf{grad} b_K^i(\mathbf{x}) &= \frac{\partial \widehat{b}^i}{\partial \widehat{x}_1}(\widehat{\mathbf{x}}) \mathbf{grad} \lambda_2 + \frac{\partial \widehat{b}^i}{\partial \widehat{x}_2}(\widehat{\mathbf{x}}) \mathbf{grad} \lambda_3 \\ &= [\mathbf{grad} \lambda_2 \quad \mathbf{grad} \lambda_3] \mathbf{grad}_{\widehat{\mathbf{x}}} \widehat{b}^i(\widehat{\mathbf{x}}), \quad \mathbf{x} = \Phi_K(\widehat{\mathbf{x}}).\end{aligned}\quad (2.8.1.16)$$

This formula is convenient, because $\mathbf{grad} \lambda_i \equiv \text{const}$, see (2.7.5.4).

This facilitates the computation of element (stiffness) matrices for 2nd-order elliptic problems in variational form with scalar valued coefficient $\alpha = \alpha(\mathbf{x})$: when using a quadrature formula according to (2.7.5.23)

$$\begin{aligned}&\int_K (\alpha(\mathbf{x}) \mathbf{grad} b_K^i) \cdot \mathbf{grad} b_K^j \, d\mathbf{x} \\ &\approx \frac{|K|}{|\widehat{K}|} \sum_{l=1}^{P_K} \widehat{\omega}_l \alpha(\zeta_l) \left(\begin{bmatrix} \frac{\partial \widehat{b}^i}{\partial \widehat{x}_1}(\zeta_l) \\ \frac{\partial \widehat{b}^i}{\partial \widehat{x}_2}(\zeta_l) \end{bmatrix}^\top \begin{bmatrix} \mathbf{grad} \lambda_2 \cdot \mathbf{grad} \lambda_2 & \mathbf{grad} \lambda_2 \cdot \mathbf{grad} \lambda_3 \\ \mathbf{grad} \lambda_2 \cdot \mathbf{grad} \lambda_3 & \mathbf{grad} \lambda_3 \cdot \mathbf{grad} \lambda_3 \end{bmatrix} \begin{bmatrix} \frac{\partial \widehat{b}^j}{\partial \widehat{x}_1}(\zeta_l) \\ \frac{\partial \widehat{b}^j}{\partial \widehat{x}_2}(\zeta_l) \end{bmatrix} \right)\end{aligned}$$

This is attractive from an implementation point of view, because

- ◆ the values $\frac{\partial \widehat{b}^i}{\partial \widehat{x}_1}(\zeta_l)$ can be *precomputed*,
- ◆ simple expressions for $\mathbf{grad} \lambda_i \cdot \mathbf{grad} \lambda_j$ are available, see Section 2.4.5.

More on the use of these transformation techniques ➤ Section 2.8.3

2.8.2 Example: Quadrilateral Lagrangian Finite Elements

So far, see Section 2.5.3 and Eq. (2.5.3.5), we have adopted the perspective that we first define global basis functions/global shape functions for a finite element space, from which the local shape functions are deduced according to Def. 2.5.3.4:

global shape functions Restriction to element local shape functions

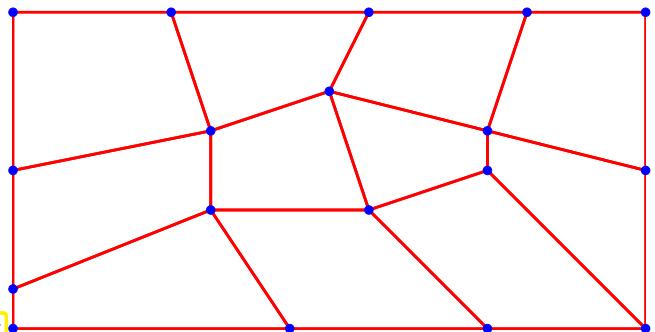
Now we reverse this construction!

local shape functions “glueing” global shape functions (2.8.2.1)

In fact, when building the global basis functions for quadratic Lagrangian finite elements we already proceeded this way, see Ex. 2.6.1.2. Fig. 109 lucidly conveys what is meant by “glueing”.

Be aware that the possibility to achieve a continuous global basis function by glueing local shape function on adjacent cells, entails a smart choice of the local shape functions.

This section will demonstrate how the policy (2.8.2.1) together with the formula (2.8.1.6) will enable us to extend Lagrangian finite element constructions beyond the meshes discussed in Section 2.6.



▷ quadrilateral mesh M in 2D

What is “ $S_1^0(M)$ ”?

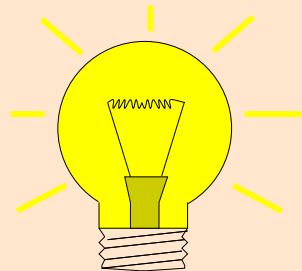
So far we know Lagrangian finite elements only on rectangles, see Section 2.6.2, for which the local spaces are given by $\mathcal{Q}_p(K)$ (→ Def. 2.6.2.5).

What to do on general quadrilaterals?

§2.8.2.2 (Bilinear transformations) It is clear that if K is a rectangle, \hat{K} the unit square, then there is a unique affine transformation Φ_K (\rightarrow Def. 2.7.5.13) with $K = \Phi_K(\hat{K})$. In this case (2.8.1.6) holds for the local shape functions of bilinear Lagrangian finite elements from Ex. 2.6.2.1 (and all tensor product Lagrangian finite elements introduced in Section 2.6.2)!

Now we turn (2.8.1.6) upside down!

Principle of constructing of parametric finite elements



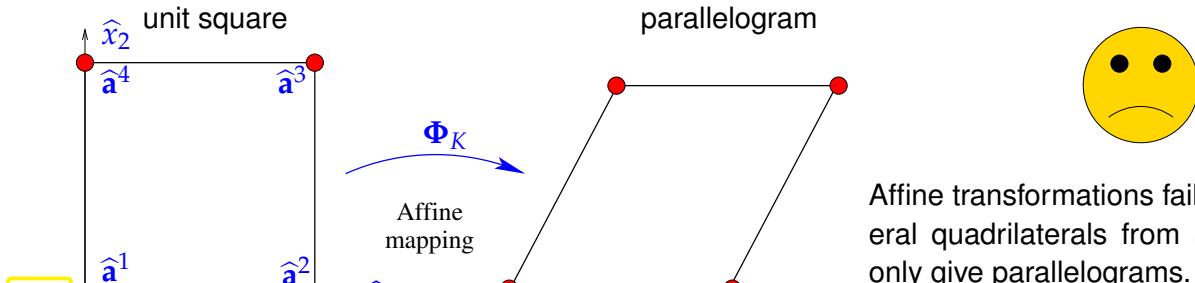
Idea:

- ◆ local shape functions $\xrightarrow{\text{"glueing"}}$ global shape functions
- ◆ Build local shape functions by “inverse pullback”

$$b_K^i = (\Phi_K^{-1})^* \hat{b}^i, \quad (2.8.2.4)$$

where $\{\hat{b}^i\}_{i=1}^Q \doteq \text{set of shape functions on reference element } \hat{K}$.

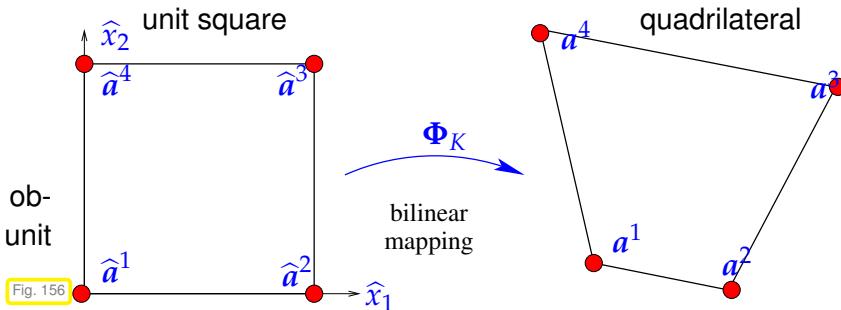
➤ This idea instantly begs the question: What is Φ_K for a general quadrilateral?



Affine transformations fail to produce general quadrilaterals from a square. They only give parallelograms.



It takes **bilinear transformations** to obtain a generic quadrilateral from the unit square!



The following formula describes a **bilinear transformation** of unit square to quadrilateral with vertices a^i , $i = 1, 2, 3, 4$:

$$\Phi_K(\hat{x}) = (1 - \hat{x}_1)(1 - \hat{x}_2) a^1 + \hat{x}_1(1 - \hat{x}_2) a^2 + \hat{x}_1 \hat{x}_2 a^3 + (1 - \hat{x}_1) \hat{x}_2 a^4. \quad (2.8.2.5)$$

↓

$$\Phi_K(\hat{x}) = \begin{bmatrix} \alpha_1 + \beta_1 \hat{x}_1 + \gamma_1 \hat{x}_2 + \delta_1 \hat{x}_1 \hat{x}_2 \\ \alpha_2 + \beta_2 \hat{x}_1 + \gamma_2 \hat{x}_2 + \delta_2 \hat{x}_1 \hat{x}_2 \end{bmatrix}, \quad \text{for some } \alpha_i, \beta_i, \gamma_i, \delta_i \in \mathbb{R}.$$

The mapping property $\Phi_K(\hat{a}^i) = a^i$ is evident. In order to see $\Phi_K(\hat{K}) = K$ ($\hat{K} \doteq$ unit square) for (2.8.2.5), verify that Φ_K maps all parallels to the coordinate axes to straight lines.

Moreover, a simple computation establishes:

If \widehat{K} is the unit square, $\Phi_K : \widehat{K} \mapsto K$ a bilinear transformation, and \widehat{b}^i the bilinear local shape functions (2.6.2.3) on \widehat{K} ,

then $(\Phi_K^{-1})^* \widehat{b}^i$ are linear on the edges of K .

§2.8.2.6 (Glueing of local shape functions on quadrilateral meshes) The last observation in § 2.8.2.2 makes possible the “glueing” of local shape functions obtained by inverse pullback from a nodal basis of $\mathcal{Q}_1(\widehat{K})$ on the unit square \widehat{K} . We give a detailed explanation:

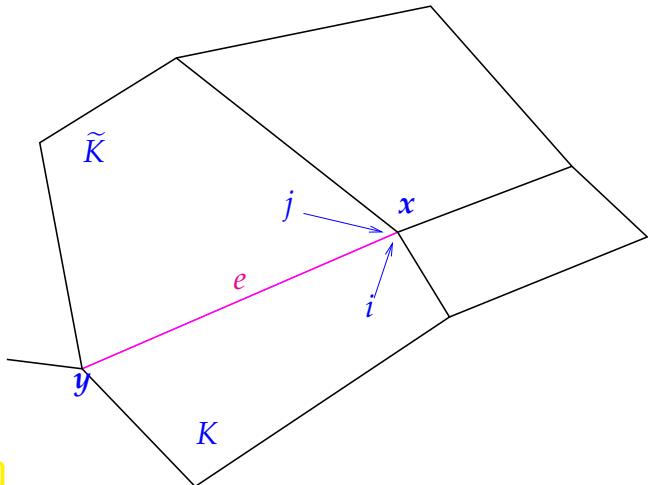


Fig. 157

- ① Pick a vertex $x \in \mathcal{V}(M)$ and consider an adjacent quadrilateral K , on which there is a local shape function b_K^i such that $b_K^i(x) = 1$ and b_K^i vanishes on all other vertices of K . This local shape function is obtained by inverse pullback of the \widehat{b}^i associated with $\Phi_K^{-1}(x)$.
- ② The same construction can be carried out for another quadrilateral \widetilde{K} that shares the vertex x and an edge e with K . On that quadrilateral we find the local shape function $b_{\widetilde{K}}^j$

- ③ Both $b_K^i|_e$ and $b_{\widetilde{K}}^j|_e$ are linear and attain the same values, that is 0 and 1 at the endpoints x and y of e , respectively.



$$b_K^i|_e = b_{\widetilde{K}}^j|_e$$

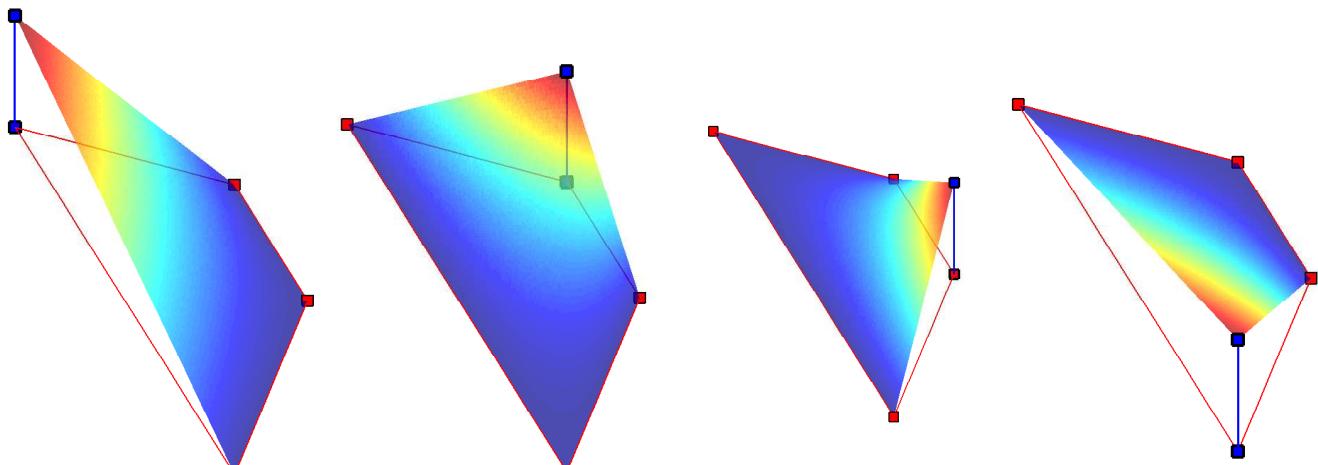


Continuity of global shape function (defined by interpolation conditions at nodes)

Remark 2.8.2.7 (Non-polynomial “bilinear” local shape functions) Note that the components of Φ_K^{-1} are *not polynomial* even if Φ_K is a bilinear transformation (2.8.2.5).

The local shape functions b_K^i defined by (2.8.2.4), where Φ_K is a bilinear transformation and \widehat{b}^i are the bilinear local shape functions on the unit square, are **not polynomial** in general.

Visualization of local shape functions on trapezoidal cell $K := \text{convex}\left\{\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 3 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}$:



Remark 2.8.2.8 (Parametric bilinear finite elements in LHRFEM++) In LHRFEM++ the implementation `If::geometry::QuadO1` of the interface `If::geometry::Geometry` supplies general quadrilaterals with straight edges, whose shape is defined by specifying the location of the four vertices, passed as columns of a matrix to the constructor:

```
1f::geometry::QuadO1 (
  Eigen::Matrix<double, Eigen::Dynamic, 4> coords);
```

The transformation Φ_K is implemented in the `Global()` member function is listed in Code 2.8.2.9. The coordinates of points in the reference element are made available as the column of a matrix in order to support modest vectorization. The member variable `coords_` is a matrix containing the vertex coordinates in its columns.

C++ code 2.8.2.9: `Global()` member function of `If::geometry::QuadO1` → [GITHUB](#)

```
2 Eigen::MatrixXd QuadO1::Global(const Eigen::MatrixXd& local) const {
3   LF_ASSERT_MSG(local.rows() == 2, "reference coords must be 2-vectors");
4   // Componentwise bilinear transformation from unit square in a
5   // vectorized fashion.
6   // Note the use of array and matrix views offered by Eigen.
7   return coords_.col(0) *
8     ((1 - local.array().row(0)) * (1 - local.array().row(1)))
9     .matrix() +
10    coords_.col(1) *
11      (local.array().row(0) * (1 - local.array().row(1))).matrix() +
12    coords_.col(2) *
13      (local.array().row(0) * local.array().row(1)).matrix() +
14    coords_.col(3) *
15      ((1 - local.array().row(0)) * local.array().row(1)).matrix();
16 }
```

2.8.3 Transformation Techniques

In the previous section we already generalized the notion of affine equivalent finite element spaces from Section 2.8.1. Now we make this a universal concept and introduce the parametric construction of finite elements. Lagrangian finite elements can also be obtained this way.

“Bilinear” Lagrangian finite elements = a specimen of **parametric finite elements**

Definition 2.8.3.1. Parametric finite elements

A finite element space on a mesh \mathcal{M} is called **parametric**, if there exist a few **reference elements** $\widehat{K}_1, \dots, \widehat{K}_R$, $R \in \mathbb{N}$, numbers $Q_r \in \mathbb{N}$, and functions $\widehat{b}_r^i \in C^0(\widehat{K})$, $i = 1, \dots, Q_r$, $r = 1, \dots, R$, such that

$$\forall K \in \mathcal{M}: \exists r \in \{1, \dots, R\}, \text{ bijection } \Phi_K : \widehat{K} \mapsto K: \quad \widehat{b}_r^i = \Phi_K^* b_K^i, \quad i = 1, \dots, Q_r ,$$

where $\{b_K^1, \dots, b_K^{Q_r}\}$ is the set of local shape functions on K .

For 2D hybrid meshes as commonly used in LEHRFEM++, Def. 2.8.3.1 applies with $R = 2$, and \widehat{K}_1 the “reference triangle” (topological cell type TRIA), \widehat{K}_2 the unit square (topological cell type QUAD), see also § 2.7.2.9.

Remark 2.8.3.2 (Non-degenerate parametric mappings) The inconspicuous requirement that $\Phi_K : \widehat{K} \rightarrow K$ is bijective may not be easy to meet or even to check for simple constructions. An “almost equivalent” requirement is that $\det D\Phi_K$ does not change sign.

Assumption 2.8.3.3. Non-degeneracy of Φ_K

For a parametric finite element method on a mesh \mathcal{M} we assume that

$$\forall K \in \mathcal{M}: \exists \underline{\delta}_K > 0: \det D\Phi_K(\widehat{x}) \geq \underline{\delta}_K \quad \forall \widehat{x} \in \widehat{K} . \quad (2.8.3.4)$$

Remark 2.8.3.5. This definition takes the possibility of “glueing” for granted: the concept of a local shape function, see (2.5.3.5), implies the existence of a global shape function with the right continuity properties (C^0 -continuity for $H^1(\Omega)$ -conforming finite element spaces). □

It turns out that parametric finite elements offer huge **algorithmic benefits**, which we elaborate now in the case of a generic elliptic 2nd-order variational Dirichlet problem

$$\begin{aligned} u \in H^1(\Omega), \\ u = g \text{ on } \partial\Omega: \quad \int_{\Omega} (\alpha(x) \mathbf{grad} u(x)) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x)v(x) dx \quad \forall v \in H_0^1(\Omega) . \end{aligned} \quad (1.4.2.4)$$

§2.8.3.6 (Local computations for parametric finite elements) We focus on the computation of element (stiffness) matrices and element (load) vectors (→ Def. 2.7.4.5), a key step in the set-up of the Galerkin matrix and right hand side vector.

Both challenges and opportunities arise from the implicit definition of the local local shape functions b_K^i via the pullback (→ Def. 2.8.1.2) of local shape functions \widehat{b}^i on the reference element ($R = 1$ assumed in Def. 2.8.3.1, index r suppressed):

$$b_K^i = (\Phi_K^{-1})^* \widehat{b}^i \Leftrightarrow \widehat{b}^i = \Phi_K^* b_K^i, \quad i = 1, \dots, Q .$$

This formula can result in very complicated or even elusive closed-form expressions for the local shape functions b_K^i . Consequently, their evaluation or that of their gradients in quadrature points might not be possible immediately. This problem is solved by transformation to the reference element.



Known: transformation $\Phi_K : \hat{K} \mapsto K$ from reference element \hat{K} .

Idea: use transformation to \hat{K} to compute element stiffness matrix \mathbf{A}_K , and element load vector $\vec{\varphi}_K$:

We elaborate the details for the variational problem (1.4.2.4). In this case the formulas for entries of element matrices \mathbf{A}_K and element vectors $\vec{\varphi}_K$ are, $i, j = 1, \dots, Q$,

$$\begin{aligned} (\mathbf{A}_K)_{ij} &= \int_K \alpha(x) \mathbf{grad} b_K^j(x) \cdot \mathbf{grad} b_K^i(x) dx \\ &= \int_{\hat{K}} (\Phi_K^* \alpha)(\hat{x}) \underbrace{(\Phi_K^*(\mathbf{grad} b_K^j))(\hat{x})}_{=?} \cdot \underbrace{(\Phi_K^*(\mathbf{grad} b_K^i))(\hat{x})}_{=?} |\det D\Phi_K(\hat{x})| d\hat{x}, \\ (\vec{\varphi}_K)_i &= \int_K f(x) b_K^i(x) dx = \int_{\hat{K}} (\Phi_K^* f)(\hat{x}) \hat{b}^i(\hat{x}) |\det D\Phi_K(\hat{x})| d\hat{x}, \end{aligned}$$

by the **transformation formula** (for multidimensional integrals, see also (0.3.2.32))

$$\int_K \varphi(x) dx = \int_{\hat{K}} (\Phi_K^* \varphi)(\hat{x}) |\det D\Phi_K(\hat{x})| d\hat{x} \quad \text{for integrable } \varphi : K \mapsto \mathbb{R}. \quad (2.8.3.7)$$

Recall the notation $\Phi_K^* f$ for the pullback (\rightarrow Def. 2.8.1.2) of a function to \hat{K} :

$$(\Phi_K^* u)(\hat{x}) := u(\Phi_K(\hat{x})), \quad \hat{x} \in \hat{K}.$$

By now, all integrals have been transformed to the reference element \hat{K} , where we can now apply a quadrature formula:

$$\int_{\hat{K}} \hat{f}(\hat{x}) d\hat{x} \approx \sum_{l=1}^P \hat{\omega}_l \hat{f}(\hat{\zeta}_l), \quad \hat{\zeta}_l \in \hat{K}, \quad \hat{\omega}_l \in \mathbb{R}, \quad (2.7.5.23)$$

which, as in § 2.7.5.21, can be combined with (2.8.3.7):

$$\Rightarrow \int_K f(x) dx \approx \sum_{l=1}^P \hat{\omega}_l f(\Phi_K(\hat{\zeta}_l)) |\det D\Phi_K(\hat{\zeta}_l)|. \quad (2.8.3.8)$$

Thus, we get the following approximation of an entry of the right-hand side vector:

$$(\vec{\varphi}_K)_i \approx \sum_{\ell=1}^P \hat{\omega}_\ell f(\Phi_K(\hat{\zeta}_\ell)) \hat{b}^i(\hat{\zeta}_\ell) |\det D\Phi_K(\hat{\zeta}_\ell)|. \quad (2.8.3.9)$$

Required information and evaluations:

- values $\hat{b}^i(\hat{\zeta}_l)$, $i = 1, \dots, Q$, $l = 1, \dots, P$,
- gradients $\Phi_K^*(\mathbf{grad} b_K^i)$ at quadrature nodes $\hat{\zeta}_l \in \hat{K}$!?
- metric factors at quadrature nodes in \hat{K} : $\det D\Phi_K(\hat{\zeta}_l)$
- values $\alpha(\Phi_K(\hat{\zeta}_l)) \in \mathbb{R}^{d,d}$ and $f(\Phi_K(\hat{\zeta}_l)) \in \mathbb{R}$ from point evaluations of functions $\alpha : \bar{\Omega} \rightarrow \mathbb{R}^{d,d}$, $f : \bar{\Omega} \rightarrow \mathbb{R}$.

The gradients seem to pose a problem (!?) as b_K^i may be elusive, cf. Rem. 2.8.2.7! Fortunately we can compute them from the gradients of the local shape functions \hat{b}^i on the reference element using the formulas given in the next lemma.

Lemma 2.8.3.10. Transformation formula for gradients

For differentiable $u : K \mapsto \mathbb{R}$ and any diffeomorphism $\Phi : \hat{K} \mapsto K$ we have

$$(\mathbf{grad}_{\hat{x}}(\Phi^* u))(\hat{x}) = (\mathbf{D}\Phi(\hat{x}))^\top \underbrace{(\mathbf{grad}_x u)(\Phi(\hat{x}))}_{=\Phi^*(\mathbf{grad} u)(\hat{x})} \quad \forall \hat{x} \in \hat{K}. \quad (2.8.3.11)$$

Proof. Use the **chain rule** to compute components of the gradient vector

$$(\mathbf{grad} \Phi^* u(\hat{x}))_i = \frac{\partial \Phi^* u}{\partial \hat{x}_i}(\hat{x}) = \frac{\partial}{\partial \hat{x}_i} u(\Phi(\hat{x})) = \sum_{j=1}^d \frac{\partial u}{\partial x_j}(\Phi(\hat{x})) \frac{\partial \Phi_j}{\partial \hat{x}_i}(\hat{x}).$$

$$\blacktriangleright \begin{bmatrix} \frac{\partial \Phi^* u}{\partial \hat{x}_1}(\hat{x}) \\ \vdots \\ \frac{\partial \Phi^* u}{\partial \hat{x}_d}(\hat{x}) \end{bmatrix} = (\mathbf{grad}_{\hat{x}} \Phi^* u)(\hat{x}) = \mathbf{D}\Phi(\hat{x})^\top \begin{bmatrix} \frac{\partial u}{\partial x_1}(\Phi(\hat{x})) \\ \vdots \\ \frac{\partial u}{\partial x_d}(\Phi(\hat{x})) \end{bmatrix} = \mathbf{D}\Phi(\hat{x})^\top (\mathbf{grad}_x u)(\Phi(\hat{x})).$$

Here, $\mathbf{D}\Phi(\hat{x}) \in \mathbb{R}^{d,d}$ is the Jacobian of Φ at $\hat{x} \in \hat{K}$, according to the general formula from § 0.3.2.15:

$$\mathbf{Df}(x) \leftrightarrow \mathbf{Df}(x) = \left[\frac{\partial f_i}{\partial x_j}(x) \right]_{i,j=1}^d = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \cdots & \frac{\partial f_1}{\partial x_d}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & & & & \frac{\partial f_2}{\partial x_d}(x) \\ \vdots & & & & \vdots \\ \frac{\partial f_d}{\partial x_1}(x) & \frac{\partial f_d}{\partial x_2}(x) & \cdots & \cdots & \frac{\partial f_d}{\partial x_d}(x) \end{bmatrix} \in \mathbb{R}^{d,d}, \quad (0.3.2.16)$$

see also [Str09, Bem. 7.6.1]. □

Using Lemma 2.8.3.10 we arrive at a tractable expression for the entries of the element matrix:

$$\begin{aligned} (\mathbf{A}_K)_{ij} &= \int_{\hat{K}} (\alpha(\Phi(\hat{x})) (\mathbf{D}\Phi)^{-\top} \mathbf{grad} \hat{b}^i) \cdot ((\mathbf{D}\Phi)^{-\top} \mathbf{grad} \hat{b}^j) |\det \mathbf{D}\Phi| d\hat{x} \\ &= \int_{\hat{K}} ((\mathbf{D}\Phi)^{-1} \alpha(\Phi(\hat{x})) (\mathbf{D}\Phi)^{-\top}) \mathbf{grad} \hat{b}^i \cdot \mathbf{grad} \hat{b}^j |\det \mathbf{D}\Phi| d\hat{x}. \end{aligned} \quad (2.8.3.12)$$

Note that the argument \hat{x} is suppressed for some terms in the integrand.

☞ notation: for matrix \mathbf{S} write $\mathbf{S}^{-\top} := (\mathbf{S}^{-1})^\top = (\mathbf{S}^\top)^{-1}$

The next step is the approximation of (2.8.3.12) by means of a quadrature rule (2.7.5.10) on \hat{K}

$$\int_{\hat{K}} f(\hat{x}) d\hat{x} \approx \sum_{\ell=1}^P \hat{\omega}_\ell f(\hat{\zeta}_\ell), \quad (2.7.5.23)$$

which yields

$$\begin{aligned} (\mathbf{A}_K)_{ij} &\approx \sum_{\ell=1}^P \hat{\omega}_\ell \alpha(\Phi(\hat{\zeta}_\ell)) (\mathbf{D}\Phi(\hat{\zeta}_\ell))^{-\top} \mathbf{grad} \hat{b}^i(\hat{\zeta}_\ell) \cdot \\ &\quad ((\mathbf{D}\Phi(\hat{\zeta}_\ell))^{-\top} \mathbf{grad} \hat{b}^j(\hat{\zeta}_\ell)) |\det \mathbf{D}\Phi(\hat{\zeta}_\ell)| \\ &= \sum_{\ell=1}^P \hat{\omega}_\ell (\mathbf{M}_K(\hat{\zeta}_\ell) \mathbf{grad} \hat{b}^i(\hat{\zeta}_\ell)) \cdot \mathbf{grad} \hat{b}^j(\hat{\zeta}_\ell) |\det \mathbf{D}\Phi(\hat{\zeta}_\ell)|, \end{aligned} \quad (2.8.3.13)$$

with $\mathbf{M}_K(\hat{x}) := (\mathbf{D}\Phi)^{-1}(\hat{x}) \alpha(\Phi(\hat{x})) (\mathbf{D}\Phi)^{-\top}(\hat{x}), \quad \hat{x} \in \hat{K}.$

The vectors $\mathbf{grad} \hat{b}^i(\hat{\zeta}_l)$ are easily computed, since the local shape functions \hat{b}^i will usually be simple polynomials. In addition, they are independent of K , so they can be precomputed and stored in a table. The same holds for the numbers $|\det D\Phi(\hat{\zeta}_l)|$.

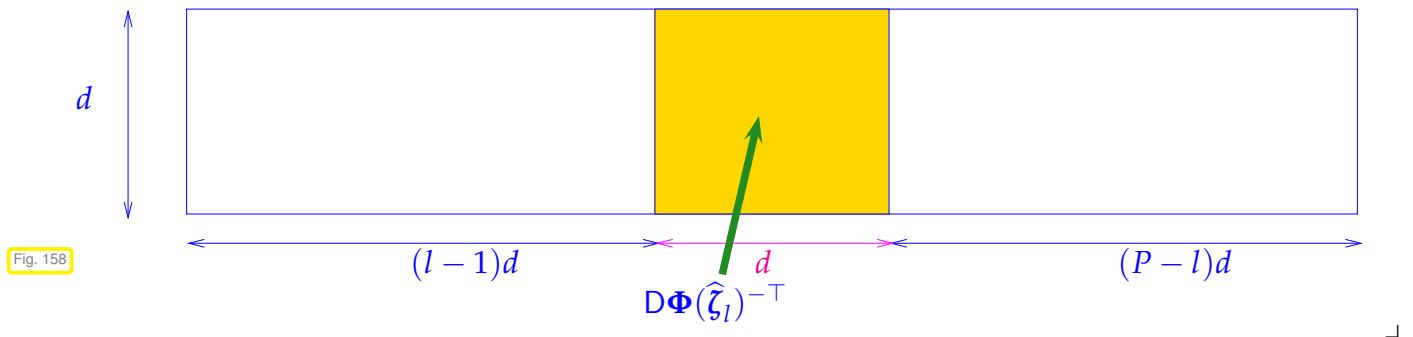
§2.8.3.14 (LEHRFEM++ support for transformation of gradients) From (2.8.3.12) and (2.8.3.13) we see that $D\Phi(\hat{x})^{-\top}$ for a few (quadrature) points $\hat{x} \in \hat{K}$ is required for the computation of entries of element matrices. LEHRFEM++ supplies this matrix through a function of **If::geometry::Geometry**:

```
Eigen::MatrixXd lf::geometry::Geometry::JacobianInverseGramian(const
    Eigen::MatrixXd &local) const;
```

This function takes a $n \times P$ -matrix argument local ($n \in \mathbb{N}$) as returned from `ent.Geometry() -> DimLocal()` for an **If::mesh::Entity** object `ent`, whose columns provide P point coordinates $\hat{\zeta}_l, l = 1, \dots, P$, in the reference element as explained in § 2.7.2.21.

We focus on the case `DimLocal() == DimGlobal`, that is $n = d$. Then the function returns the P inverses of the transposed Jacobian, horizontally concatenated into a big $d \times (P \cdot d)$ matrix. The $d \times d$ matrix $D\Phi(\hat{\zeta}_l)^{-\top}$ can be fetched by the statement

```
Eigen::MatrixXd JTinv =
    ent.Geometry() -> JacobianInverseGramian(local).block(0, 1*d, d, d);
```



§2.8.3.15 (Local computations for parametric finite elements, § 2.8.3.6 cnt'd) In light of the formula

$$(A_K)_{ij} \approx \sum_{l=1}^P \hat{\omega}_l (\mathbf{M}_K(\hat{\zeta}_l) \mathbf{grad} \hat{b}^i(\hat{\zeta}_l)) \cdot \mathbf{grad} \hat{b}^j(\hat{\zeta}_l) |\det D\Phi(\hat{\zeta}_l)|, \quad (2.8.3.13)$$

with $\mathbf{M}_K(\hat{x}) := (D\Phi)^{-1}(\hat{x}) \alpha(\Phi(\hat{x})) (D\Phi)^{-\top}(\hat{x}), \quad \hat{x} \in \hat{K}.$

we find that for a scalar parametric $H^1(\Omega)$ -conforming finite element method for a second-order elliptic boundary value problem using a *fixed mapped quadrature rule* with nodes $\hat{\zeta}_l \in \hat{K}, l = 1, \dots, P$,

(I) the following quantities can be pre-computed for all cells:

- the $P \cdot Q$ values $\hat{b}^i(\hat{\zeta}_l) \in \mathbb{R}, i = 1, \dots, Q, l = 1, \dots, P$,
- and the $P \cdot Q$ vectors $\mathbf{grad} \hat{b}^i(\hat{\zeta}_l) \in \mathbb{R}^d, i = 1, \dots, Q, l = 1, \dots, P$,

In LEHRFEM++ this information is precomputed and provided by objects of type **If::uscalfe::PrecomputedScalarReferenceFiniteElement**.

(II) Conversely, the following evaluations have to be done for every cell K separately:

- the computation of $\Phi_K(\hat{\zeta}_l), l = 1, \dots, P$
(by `lf::mesh::Geometry::Global()` in LEHRFEM++),
- the calculation of $|\det D\Phi_K(\hat{\zeta}_l)|, l = 1, \dots, P$
(by `lf::mesh::Geometry::IntegrationElement()` in LEHRFEM++),

- the evaluations of $D\Phi_K(\hat{\zeta}_l)^{-\top}, l = 1, \dots, P$
(by `lf::mesh::Geometry::JacobianInverseGramian()` in `LEHRFEM++`).

If we assume a parametric construction matching Def. 2.8.3.1, need to know surprising little about the finite element space.

Information defining a $H^1(\Omega)$ -conforming parametric finite element

In order to define a $H^1(\Omega)$ -conforming parametric finite element in a finite element code we have to implement for all reference elements and $i = 1, \dots, Q$

- a function realizing $\hat{x} \mapsto \hat{b}^i(\hat{x}), \hat{x} \in \hat{K}$,
- and the evaluation $\hat{x} \mapsto \text{grad } \hat{b}^i(\hat{x}), \hat{x} \in \hat{K}$.

EXAMPLE 2.8.3.17 (Cell-dependent evaluations for bilinear transformations) In Section 2.8.2 we saw that it takes a general bilinear transformation (2.8.2.5) to map a square onto a general quadrilateral cell, see Fig. 156 on page 277. It turned out that these bilinear mappings are key to defining *parametric* Lagrangian finite elements on general quadrilaterals.

In order to compute the element (stiffness) matrices according to (2.8.3.13), we have to evaluate the Jacobians for bilinear transformations $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ and their determinants. This can be done through the following formulas:

$$\text{For } \Phi(\hat{x}) = \begin{bmatrix} \alpha_1 + \beta_1 \hat{x}_1 + \gamma_1 \hat{x}_2 + \delta_1 \hat{x}_1 \hat{x}_2 \\ \alpha_2 + \beta_2 \hat{x}_1 + \gamma_2 \hat{x}_2 + \delta_2 \hat{x}_1 \hat{x}_2 \end{bmatrix}, \quad \alpha_i, \beta_i, \gamma_i, \delta_i \in \mathbb{R}, \quad (2.8.3.18a)$$

$$\Rightarrow D\Phi(\hat{x}) = \begin{bmatrix} \beta_1 + \delta_1 \hat{x}_2 & \gamma_1 + \delta_1 \hat{x}_1 \\ \beta_2 + \delta_2 \hat{x}_2 & \gamma_2 + \delta_2 \hat{x}_1 \end{bmatrix}, \quad (2.8.3.18b)$$

$$\Rightarrow \det(D\Phi(\hat{x})) = \beta_1 \gamma_2 - \beta_2 \gamma_1 + (\beta_1 \delta_2 - \beta_2 \delta_1) \hat{x}_1 + (\gamma_1 \delta_2 - \gamma_2 \delta_1) \hat{x}_2. \quad (2.8.3.18c)$$

Note that both $D\Phi(\hat{x})$ and $\det(D\Phi(\hat{x}))$ are (componentwise) linear functions in \hat{x} .

If $\Phi = \Phi_K$ for a generic quadrilateral K with corners $\mathbf{a}^\ell, \ell = 1, 2, 3, 4$, as in (2.8.2.5), then the coefficients $\alpha_i, \beta_i, \gamma_i, \delta_i$ depend on the shape of K in a straightforward fashion:

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \mathbf{a}^1, \quad \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} = \mathbf{a}^2 - \mathbf{a}^1, \quad \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} = \mathbf{a}^4 - \mathbf{a}^1, \quad \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \mathbf{a}^3 - \mathbf{a}^2 - \mathbf{a}^4 + \mathbf{a}^1. \quad (2.8.3.19)$$

The formulas (2.8.3.18a)–(2.8.3.18c) and (2.8.3.19) are directly implemented in the member functions of `If::geometry::QuadO1`, see Rem. 2.8.2.8. For details refer to the following listings. In each case, coordinates of points in the unit square \hat{K} are passed as columns of a matrix in order to allow vectorization.

C++ code 2.8.3.20: Member function `Jacobian()` of `If::geometry::QuadO1`, see (2.8.3.18b) → [GITHUB](#)

```

2 Eigen::MatrixXd QuadO1::Jacobian(const Eigen::MatrixXd& local) const {
3   Eigen::MatrixXd result(DimGlobal(), local.cols() * 2);
4
5   // Note that coords_ stores the coordinates of the vertices of
6   // the quadrilateral in its columns.
7   for (Eigen::Index i = 0; i < local.cols(); ++i) {
8     // Partial derivative of componentwise bilinear mapping
9     // w.r.t. to first reference coordinate
10    result.col(2 * i) = (coords_.col(1) - coords_.col(0)) * (1 - local(1, i)) +
11                  (coords_.col(2) - coords_.col(3)) * local(1, i);
12    // Partial derivative of componentwise bilinear mapping

```

```

13     // w.r.t. to second reference coordinate
14     result.col(2 * i + 1) =
15         (coords_.col(3) - coords_.col(0)) * (1 - local(0, i)) +
16         (coords_.col(2) - coords_.col(1)) * local(0, i);
17 }
18 return result;
19 }
```

C++ code 2.8.3.21: Member function `JacobianInverseGramian()` of `If::geometry::QuadO1` → [GITHUB](#)

```

2 Eigen::MatrixXd QuadO1::JacobianInverseGramian(
3     const Eigen::MatrixXd& local) const {
4     Eigen::MatrixXd result(DimGlobal(), local.cols() * 2);
5     Eigen::MatrixXd jacobian(DimGlobal(), 2);
6
7     // Loop over all evaluation points
8     for (Eigen::Index i = 0; i < local.cols(); ++i) {
9         // Compute Jacobian matrix in one evaluation point.
10        jacobian.col(0) = (coords_.col(1) - coords_.col(0)) * (1 - local(1, i)) +
11                         (coords_.col(2) - coords_.col(3)) * local(1, i);
12        jacobian.col(1) = (coords_.col(3) - coords_.col(0)) * (1 - local(0, i)) +
13                         (coords_.col(2) - coords_.col(1)) * local(0, i);
14
15        // Distinguish whether the cell lies in a planar mesh or a surface
16        // mesh
17        if (DimGlobal() == 2) {
18            // Standard case: 2D planar mesh
19            // Eigen has a built-in function for computing the inverse of a
20            // small
21            // matrix
22            result.block(0, 2 * i, DimGlobal(), 2) = jacobian.transpose().inverse();
23        } else {
24            result.block(0, 2 * i, DimGlobal(), 2) = (jacobian.transpose() * jacobian)
25                                         .colPivHouseholderQr()
26                                         .solve(jacobian.transpose())
27                                         .transpose();
28        }
29    }
30
31    return result;
32 }
```

C++ code 2.8.3.22: Member function `IntegrationElement()` of `If::geometry::QuadO1`, see (2.8.3.18c) → [GITHUB](#)

```

2 Eigen::VectorXd QuadO1::IntegrationElement(const Eigen::MatrixXd& local) const {
3     Eigen::VectorXd result(local.cols());
4     Eigen::MatrixXd jacobian(DimGlobal(), 2);
5
6     // Loop over all evaluation points
7     for (Eigen::Index i = 0; i < local.cols(); ++i) {
8         // Compute Jacobian matrix in one evaluation point.
9         jacobian.col(0) = (coords_.col(1) - coords_.col(0)) * (1 - local(1, i)) +
10                         (coords_.col(2) - coords_.col(3)) * local(1, i);
11         jacobian.col(1) = (coords_.col(3) - coords_.col(0)) * (1 - local(0, i)) +
12                         (coords_.col(2) - coords_.col(1)) * local(0, i);
13
14         // For planar cell, simply the determinant, for a cell in 3D space,
15         // the
16         // volume form
```

```

15     if (DimGlobal() == 2) {
16         result(i) = std::abs(jacobian.determinant());
17     } else {
18         result(i) = std::sqrt((jacobian.transpose() * jacobian).determinant());
19     }
20 }
21 return result;
22 }
```

Remark 2.8.3.23 (Finite element methods on surfaces) In Code 2.8.3.21 and Code 2.8.3.22 the “world dimension” of the cell is tested by calling the `DimGlobal()` member function of the entity. The standard case of cells of a 2D planar mesh is `DimGlobal() == 2`.

However, `DimGlobal() == 3` is possible. In this case we have 2D quadrilateral in 3D space. In this case we have to deal with a mapping $\Phi_K : \hat{K} \subset \mathbb{R}^2 \rightarrow K \subset \mathbb{R}^3$ and its Jacobian will be a 3×2 -matrix: $D\Phi_K(\hat{x}) \in \mathbb{R}^{3,2}$.

Then the member function `IntegrationElement()` has to implement the metric factor from the transformation formula for surface integrals, *cf.* (0.3.2.34),

$$\int_K \varphi(x) dS(x) = \int_{\hat{K}} \varphi(\Phi_K(\hat{x})) \sqrt{\det(D\Phi_K^\top(\hat{x}) D\Phi_K(\hat{x}))} d\hat{x}, \quad f \in C^0(K). \quad (2.8.3.24)$$

This accounts for the formula implemented in Code 2.8.3.22.

LEHRFEM++’s capability to handle meshes covering 2D surfaces in three-dimensional space paves the ways for solving PDEs on surfaces. Those frequently occur in biological models (diffusion on a membrane) and meteorological simulations (fluid flow on the surface of the globe).

The following pictures exhibit meshes describing the surface of a torus.

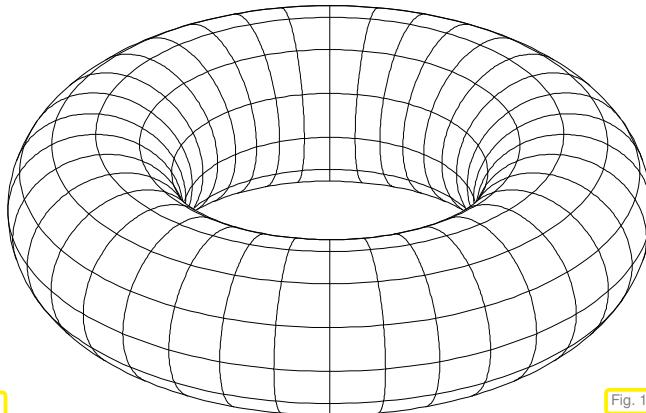


Fig. 159

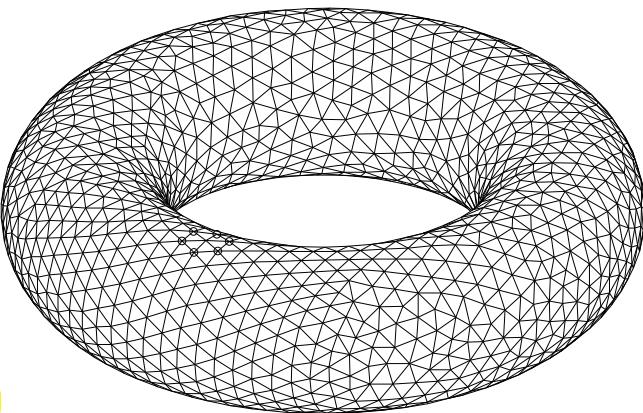


Fig. 160

A LEHRFEM++ demo showing the creation of a mesh describing a toroidal surface in 3D is available.
[→ GITHUB](#)

§2.8.3.25 (Interface for scalar-valued parametric finite in LEHRFEM++) Motivated by the insights gleaned in § 2.8.3.15 the LEHRFEM++ class `If::fe::ScalarReferenceFiniteElement` defines a general interface for defining scalar parametric finite elements. Among others, it provides the member functions

- `base::RefEl ScalarReferenceFiniteElement::RefEl() const;`
- `unsigned int ScalarReferenceFiniteElement::Degree() const;`

which provide the topological type of the associated entity and the polynomial degree of the local finite element space.

- `size_type ScalarReferenceFiniteElement::NumRefShapeFunctions () const;`
- `size_type ScalarReferenceFiniteElement::NumRefShapeFunctions (dim_t codim) const;`
- `size_type ScalarReferenceFiniteElement::NumRefShapeFunctions (dim_t codim, sub_idx_t subidx) const;`

which tell the number Q of local shape functions, either covering the whole entity or some sub-entity.

- `Eigen::Matrix<SCALAR, Eigen::Dynamic, Eigen::Dynamic> ScalarReferenceFiniteElement::EvalReferenceShapeFunctions (const Eigen::MatrixXd& refcoords) const;`

which performs the evaluation $\hat{x} \mapsto \hat{b}^i(\hat{x})$ for all i and all the P points $\in \hat{K}$, whose coordinates are passed through the columns of `refcoords`. It returns a $Q \times P$ -matrix with the values.

- `Eigen::Matrix<SCALAR, Eigen::Dynamic, Eigen::Dynamic> ScalarReferenceFiniteElement::GradientsReferenceShapeFunctions (const Eigen::MatrixXd& refcoords) const;`

meant for computing $\hat{x} \mapsto \text{grad } \hat{b}^i(\hat{x})$, at the P points passed in `refcoords`. The gradients for all i are returned packed into the rows of an $Q \times (Pn)$ -matrix, n the dimension of \hat{K} : If $\text{refcoords} = [\hat{x}_1, \dots, \hat{x}_P]^\top$, and $\hat{b}^1, \dots, \hat{b}^Q$ are the reference shape functions, then the returned matrix is

$$\begin{bmatrix} (\text{grad } \hat{b}^1(\hat{x}_1))^\top & \dots & (\text{grad } \hat{b}^1(\hat{x}_P))^\top \\ \vdots & & \vdots \\ (\text{grad } \hat{b}^Q(\hat{x}_1))^\top & \dots & (\text{grad } \hat{b}^Q(\hat{x}_P))^\top \end{bmatrix} \in \mathbb{R}^{Q,Pn}.$$

Another group of member functions of `If::fe::ScalarReferenceFiniteElement` is meant to support the definition of “nodal” interpolation operators. An in-depth discussion will be postponed to Chapter 3, see Def. 3.3.2.1 for a particular example, and § 3.3.5.2 for the case of Lagrangian finite elements of degree $p \in \mathbb{N}$.

For parametric scalar-valued finite element schemes, these nodal interpolation operators are first defined on the reference elements and they rely on Q , $Q \in \mathbb{N}$ the number of reference shape functions, **reference interpolation nodes** (also known as “evaluation nodes”) $\hat{p}_1, \dots, \hat{p}_Q \in \hat{K}$. Concretely, one defines reference nodal interpolation operators as mappings

$$\hat{\iota}: \mathbb{R}^Q \rightarrow \hat{V} := \text{Span}\{\hat{b}^1, \dots, \hat{b}^Q\}, \quad (2.8.3.26)$$

such that

$$\iota(\vec{\eta})(\hat{p}_k) = (\vec{\eta})_k, \quad \forall \vec{\eta} \in \mathbb{R}^Q, \quad k = 1, \dots, Q. \quad (2.8.3.27)$$

The involved member functions of `If::fe::ScalarReferenceFiniteElement` are

- `Eigen::MatrixXd ScalarReferenceFiniteElement::EvaluationNodes () const;`

which returns the reference coordinates of the interpolation nodes \hat{p}_ℓ , $\ell = 1, \dots, Q$, as the columns of an $d \times Q$ matrix, where d is the dimension of \hat{K} .

- `lf::assemble::size_type NumEvaluationNodes() const;`

which gives the number Q of evaluation nodes.

- `Eigen::Matrix<SCALAR, 1, Eigen::Dynamic> ScalarReferenceFiniteElement::NodalValuesToDofs(const Eigen::Matrix<SCALAR, 1, Eigen::Dynamic>& nodvals) const;`

returns the vector of $\{\hat{b}^k\}$ -basis expansion coefficients for $\hat{\mathbf{l}}(\text{nodvals})$:

$$\begin{aligned} \hat{\mathbf{l}}(\text{nodvals}) &= \sum_{k=1}^Q \text{NodalValuesTodofs}(\text{nodvals})[k] \cdot \hat{b}^k, \\ &\Downarrow \\ \text{nodvals}[i] &= \sum_{k=1}^Q \text{NodalValuesTodofs}(\text{nodvals})[k] \cdot \hat{b}^k(p_i), \quad i = 1, \dots, Q, \end{aligned}$$

with $\hat{\mathbf{l}}$ defined in (2.8.3.26) and (2.8.3.27).

Note that `NodalValuesToDofs()` just returns its argument row-vector, if the reference shape functions \hat{b}^ℓ are a **cardinal basis** with respect to the reference interpolation nodes \hat{p}_ℓ , that is, satisfy $\hat{b}^i(\hat{p}_j) = \delta_{i,j}$, $i, j \in \{1, \dots, Q\}$.

We point out that a `If::fe::ScalarReferenceFiniteElement` object can be created for any type of entity, in particular also for entities of topological type EDGE. The type of entity to which an `If::fe::ScalarReferenceFiniteElement` object belongs can be queried through the member function `lf::fe::ScalarReferenceFiniteElement::RefEl()`.

The LEHRFEM++ library implements the `If::fe::ScalarReferenceFiniteElement` interface for lowest-order (“linear”) Lagrangian finite elements, that is, the spaces $\mathcal{S}_1^0(\mathcal{M})$ on hybrid 2D meshes \mathcal{M} , in the following classes

- `If::uscalfe::FeLagrangeO1Tria` (barycentric coordinate functions, $Q = 3$)
- `If::uscalfe::FeLagrangeO1Quad` (local shape functions (2.6.2.3), $Q = 4$)
- `If::uscalfe::FeLagrangeO1Segment` (local shape functions as in Ex. 2.5.3.7, $Q = 2$)

Here, $Q \in \mathbb{N}$ stands for the number returned by the virtual method `lf::uscalfe::ScalarReferenceFiniteElement::NumRefShapeFunctions()`.

↓

§2.8.3.28 (Simple scalar-valued parametric “finite element spaces” in LEHRFEM++) If the local shape functions for a scalar-valued parametric finite element method are specified, the definition of the finite element space is complete! Thus giving `If::fe::ScalarReferenceFiniteElement` objects for every type of entity of a mesh is sufficient information for the initialization of `If::assemble::DofHandler` objects in charge of local→global index mappings for shape functions.

This is essentially the purpose of the LEHRFEM++ interface `If::uscalfe::UniformScalarFEspace`, whose constructor takes a (pointer to) an `If::mesh::Mesh` object and suitable `If::fe::ScalarReferenceFiniteElement` objects to build the appropriate `If::assemble::DofHandler` objects and stores them in member data variables.

```
template<SCALAR>
UniformScalarFEspace<SCALAR>::
UniformScalarFEspace(
    std::shared_ptr<const lf::mesh::Mesh> mesh_p,
```

```
std::shared_ptr<const ScalarReferenceFiniteElement<SCALAR>>
rfs_tria_p,
std::shared_ptr<const ScalarReferenceFiniteElement<SCALAR>>
rfs_quad_p,
std::shared_ptr<const ScalarReferenceFiniteElement<SCALAR>>
rfs_edge_p);
```

Any **If::fe::ScalarReferenceFiniteElement** argument is optional and a **nullptr** may be given.

The main member functions of **If::uscalfe::UniformScalarFESpace** are:

- **std::shared_ptr<const lf::mesh::Mesh>**
`lf::uscalfe::UniformScalarFESpace<SCALAR>::`
Mesh();

which returns a pointer to the underlying mesh.

- **const lf::assemble::DofHandler &**
`lf::uscalfe::UniformScalarFESpace<SCALAR>::`
LocGlobMap();

which returns a reference to the **If::assemble::DofHandler** object for the finite element space.

- **std::shared_ptr<const ScalarReferenceFiniteElement<SCALAR>>**
`lf::uscalfe::UniformScalarFESpace<SCALAR>::`
ShapeFunctionLayout(lf::base::RefEl rel_el_type);

which gives the **If::fe::ScalarReferenceFiniteElement** object for a particular type of entity.

In general, the interface **If::uscalfe::UniformScalarFESpace** is a convenient way to initialize **If::assemble::DofHandler** objects for parametric scalar-valued finite element methods.

A concrete implementation is given by **If::uscalfe::FeSpaceLagrangeO1** for lowest-order Lagrangian finite element spaces $S_1^0(\mathcal{M})$ on hybrid 2D meshes, see Section 2.6. □

EXAMPLE 2.8.3.29 (Computing element matrices for parametric FEM in LEHRFEM++) LEHRFEM++ comes with a class dedicated to the computation of element matrices (→ Def. 2.7.4.5) for the bilinear form

$$a(u, v) = \int_{\Omega} \alpha(x) \operatorname{grad} u \cdot \operatorname{grad} v + \gamma(x) u v \, dx, \quad u, v \in H^1(\Omega), \quad (2.8.3.30)$$

and a general scalar-valued parametric finite element method according to Def. 2.8.3.1, whose reference shape functions are defined via objects of type **If::fe::ScalarReferenceFiniteElement**, see § 2.8.3.25. Here $\alpha : \Omega \rightarrow \mathbb{R}^{2,2}$ is a matrix-valued diffusion coefficient, and $\gamma : \Omega \rightarrow \mathbb{R}$ a reaction coefficient, both given in procedural form through functor object.

The key type is **If::uscalfe::ReactionDiffusionElementMatrixProvider** which complies with LEHRFEM++’s concept of an **ENTITY_MATRIX_PROVIDER**. Its constructor is specified as follows:

```
template <typename SCALAR, typename DIFF_COEFF, typename
REACTION_COEFF>
ReactionDiffusionElementMatrixProvider<SCALAR, DIFF_COEFF,
REACTION_COEFF>::
ReactionDiffusionElementMatrixProvider(
std::shared_ptr<UniformScalarFESpace<SCALAR>> fe_space,
DIFF_COEFF alpha, REACTION_COEFF gamma, quad_rule_collection_t
qr_collection);
```

The meaning of the `fe_space` argument is explained in § 2.8.3.28, the two functor arguments pass mesh function objects. The last argument is a

```
using quad_rule_collection_t = std::map<lf::base::RefEl,
    lf::quad::QuadRule>;
```

and, optionally, supplies a suitable `lf::quad::QuadRule` object (→ § 2.7.5.39) for every type of entity. Thus the user can prescribe the local quadrature rules to be used for local computations.

C++ code 2.8.3.31: `Eval()` method for LEHRFEM++ class `If::uscalfe::ReactionDiffusionElementMatrixProvider`, → GITHUB

```
1 template <typename SCALAR, typename DIFF_COEFF, typename REACTION_COEFF>
2 typename If::uscalfe::ReactionDiffusionElementMatrixProvider<
3     SCALAR, DIFF_COEFF, REACTION_COEFF>::ElemMat const
4 ReactionDiffusionElementMatrixProvider<
5     SCALAR, DIFF_COEFF, REACTION_COEFF>::Eval(const If::mesh::Entity &cell) {
6     // Topological type of the cell
7     const If::base::RefEl ref_el{cell.RefEl()};
8     // Fetch precomputed quantities
9     PrecomputedScalarReferenceFiniteElement<SCALAR> &pfe = // 
10    fe_recomp_[ref_el.Id()];
11    // Query the shape of the cell
12    const If::geometry::Geometry *geo_ptr = cell.Geometry();
13    // Physical dimension of the cell
14    const dim_t world_dim = geo_ptr->DimGlobal();
15    // Request Gramian determinants at quadrature points
16    const Eigen::VectorXd determinants(
17        geo_ptr->IntegrationElement(pfe.Qr().Points()));
18    // Fetch the transformation matrices for the gradients
19    const Eigen::MatrixXd JinvT(
20        geo_ptr->JacobianInverseGramian(pfe.Qr().Points()));
21    // compute values of coefficients alpha, gamma at quadrature points:
22    auto alphaval = alpha_(cell, pfe.Qr().Points()); //
23    auto gammaval = gamma_(cell, pfe.Qr().Points()); //

24    // For returning the element matrix
25    elem_mat_t mat(pfe.NumRefShapeFunctions(), pfe.NumRefShapeFunctions());
26    mat.setZero();
27

28    // Loop over quadrature points
29    for (int k = 0; k < pfe.Qr().NumPoints(); ++k) {
30        const double w = pfe.Qr().Weights()[k] * determinants[k];
31        // Transformed gradients according to (2.8.3.11)
32        const auto trf_grad(JinvT.block(0, 2 * k, world_dim, 2) *
33            pfe.PrecompGradientsReferenceShapeFunctions()
34            .block(0, 2 * k, mat.rows(), 2)
35            .transpose());
36        // Transformed gradients multiplied with coefficient
37        const auto alpha_trf_grad(alphaval[k] * trf_grad); //
38        // Add low-rank contribution of current quadrature point
39        mat += w * (alpha_trf_grad.transpose() * trf_grad +
40            (gammaval[k] * pfe.PrecompReferenceShapeFunctions().col(k)) *
41            (pfe.PrecompReferenceShapeFunctions().col(k).transpose()));
42
43    }
44    return mat;
45 }
```

- The template parameters `DIFF_COEFF` and `REACTION_COEFFICIENT` must comply with the `MeshFunction` concept, which requires that they supply the evaluation operator

```
T operator () (const lf::mesh::Entity &, const::Eigen::MatrixXd
&refcoords);
```

which takes an entity and coordinates of points *in the corresponding reference element* (packed into the columns of the matrix `refcoords` → § 2.7.2.21) as arguments and returns an object of some type. The evaluation operators are called in Line 22 and Line 23.

For **DIFF_COEFF** the type **T** must be either a scalar or a type compatible with EIGEN's matrix arithmetic, that is, an **Eigen::Matrix** itself, because it has to support the matrix-matrix multiplication of Line 38.

For **REACTION_COEFFICIENT** the return type **T** should be a simple scalar type like **double** or **complex<double>**.

- Line 9: the class **If::uscalfe::PrecomputedScalarReferenceFiniteElement** exploits the possibility to pre-compute values of reference shape functions and their gradients, if a fixed local parametric quadrature rule is used, see § 2.8.3.6.

The use of **If::uscalfe::ReactionDiffusionElementMatrixProvider** to solve a second-order scalar elliptic boundary value problem is demonstrated in an example code → [GITHUB](#). An analogous helper class dedicated to the computation of element load vectors is **If::uscalfe::ScalarLoadElementVectorProvider**.

Supplement 2.8.3.32 (MeshFunction in LEHRFEM++) The LEHRFEM++ library knows the **concept** of a **MeshFunction**, which describes a type

- templated with another type **R** that should be *CopyAssignable* and *CopyConstructible*,
- offers an evaluation operator with signature

```
std::vector<R> operator() (const lf::mesh::Entity& e, const
Eigen::MatrixXd& local) const;
```

The `local` argument of `operator()` passes an $d \times n$ -matrix of reference coordinates, where d agrees with the local dimension of the entity `e` and n the number of points in the reference entity, see also the description of the `lf::geometry::Global()` method in § 2.7.5.17.

The evaluation operator returns an array of length n , representing the value of the **MeshFunction** at those points in `e` whose local coordinates are given by the columns of the argument matrix `local`.

Special incarnations of mesh functions are:

- **If::mesh::utils::MeshFunctionGlobal** wraps a functor object `std::function<R (Eigen::Vector2d)` into a **MeshFunction** object.
- **If::mesh::utils::MeshFunctionConstant** represents a constant as a **MeshFunction** object.
- **If::fe::MeshFunctionFE** is constructed based on an **If::fe::ScalarFESpace** object together with a basis expansion coefficient vector and represents the associated scalar-valued finite element function.
- **If::fe::MeshFunctionGradFE** represents the gradient of a scalar-valued continuous finite element function. The evaluation operator returns an EIGEN column vector.

Note that **MeshFunction** objects support the binary arithmetic operations $+$, $-$, and $*$, including scalar multiplication, provided that such operations are possible for their underlying types. In addition the unary operations $-$, `transpose()`, and `squaredNorm()` are supplied.

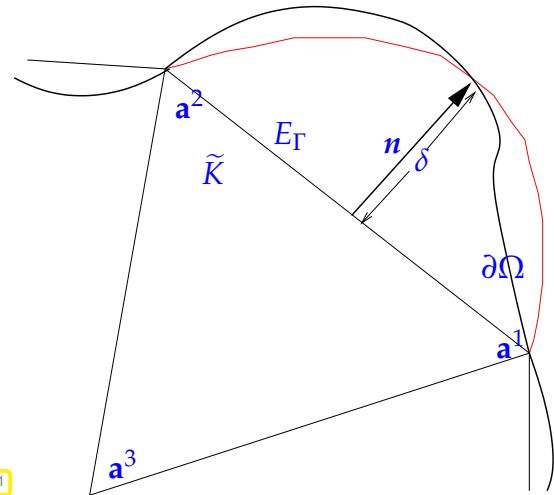
2.8.4 Application of Parametric FEM: Boundary Approximation

This section presents an important application of parametric finite elements, which is already suggested in Fig. 150. There, *curved edges* occurred at the boundary $\partial\Omega$, which was a smooth curve rather than a polygon in this case.

Intuitively it's clear that approximating a (smooth) curved boundary $\partial\Omega$ by a polygon/polyhedron will amount to fudging the model and introduce another kind of error. Hence, it is desirable to represent the boundary either exactly or employ a highly accurate approximation. Parametric finite element constructions allow this, as we will see now.

Parametric finite element constructions provide a tool going beyond polygonal/polyhedral approximation of boundaries (by simple straight lines or flat faces).

Here we discuss the treatment of piecewise smooth curved boundaries for the very simple case of triangular meshes in 2D (more details → [BS02, Sect. 10.2]).



- Idea: Piecewise polynomial approximation of boundary (boundary fitting)
($\partial\Omega$ locally considered as function over straight edge of an element)
- Example: Piecewise quadratic boundary approximation
(Part of $\partial\Omega$ between a^1 and a^2 approximated by parabola)

§2.8.4.1 (Piecewise quadratic polynomial boundary approximation) Mapping $\tilde{K} \rightarrow$ “curved element” K :

$$\tilde{\Phi}_K(\tilde{x}) := \tilde{x} + 4\delta \lambda_1(\tilde{x})\lambda_2(\tilde{x}) \mathbf{n} . \quad (2.8.4.2)$$

(λ_i barycentric coordinate functions on \tilde{K} , \mathbf{n} normal to E_Γ , see Fig. 161)

Note: Essential: δ sufficiently small $\implies \Phi$ bijective

The complete transformation $\Phi_K : \hat{K} \mapsto K$ is obtained by joining an affine transformation (→ Def. 2.7.5.13) $\Phi_K^a : \hat{K} \mapsto \tilde{K}$, $\Phi_K^a(\hat{x}) := \mathbf{F}_K \hat{x} + \boldsymbol{\tau}_K$, and $\tilde{\Phi}_K$:

$$\Phi_K = \tilde{\Phi}_K \circ \Phi_K^a .$$

For parabolic boundary fitting:

$$D\tilde{\Phi}_K = \mathbf{I} + 4\delta \mathbf{n} \cdot \mathbf{grad}(\lambda_1\lambda_2)^\top \in \mathbb{R}^{2,2} , \quad \det(D\tilde{\Phi}_K) = 1 + 4\delta \mathbf{n} \cdot \mathbf{grad}(\lambda_1\lambda_2) .$$

EXAMPLE 2.8.4.3 (Second-order geometry approximation in GMSH) Most mesh generators operator of computer-aided-design (CAD) descriptions of geometries, which usually give spline-based representations of curved boundaries. So all information needed for high-order polynomial approximation of boundaries is available.

Of course, also GMSH can generate meshes containing curved entities like the “3-node line” and “6-node triangle” mentioned in § 2.7.1.6. These two, in particular, are meant for piecewise quadratic polynomial boundary approximation. The menu item Mesh->Set Order 2 makes GMSH insert information into the .msh-file that is necessary for parabolic boundary approximation.

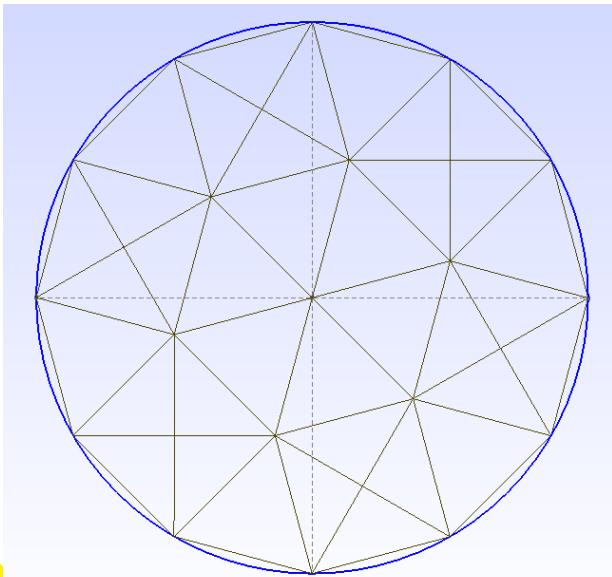


Fig. 162

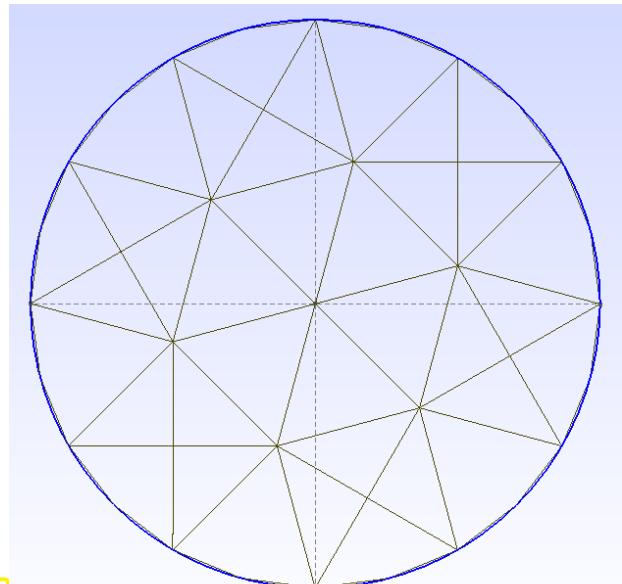


Fig. 163

Polygonal (left) and parabolic (right) approximation of a circular boundary

The .msh-file then contains entities of type 8 and 9, which corresponds to **3-node lines** and **6-node triangles**. The former is described by three point locations, the latter by six, where the extra points designate the (shifted) midpoints of edges. □

EXAMPLE 2.8.4.4 (6-Node triangles in LEHRFEM++) LEHRFEM++ offers an implementation of the `If::geometry::Geometry` interface for cells with quadratic polynomial parameterizations: `If::geometry::TriaO2` for triangular cells of type TRIA.

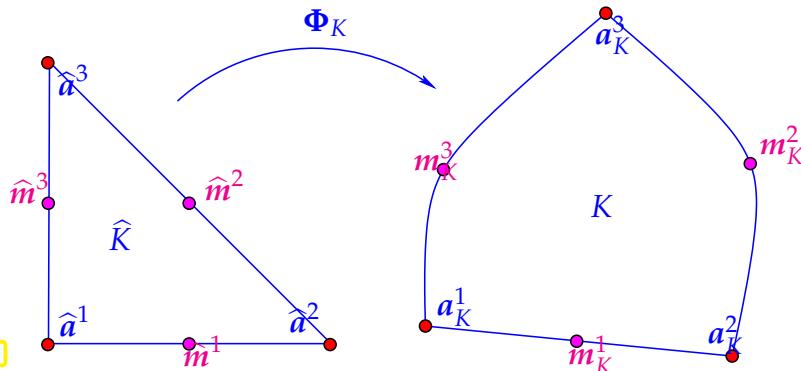


Fig. 164

Their shape is defined by prescribing the three vertex locations \mathbf{a}_K^ℓ , $\ell = 1, 2, 3$, plus the coordinates of three “midpoints” \mathbf{m}_K^ℓ , $\ell = 1, 2, 3$, of the curved edges of the “curved triangle” K .

Then there is a **unique** 2-vector-valued quadratic polynomial $\Phi_K : \hat{K} \rightarrow K$ such that

$$\Phi_K(\hat{\mathbf{a}}^\ell) = \mathbf{a}_K^\ell \quad , \quad \Phi_K(\hat{\mathbf{m}}^\ell) = \mathbf{m}_K^\ell \quad ,$$

where $\hat{\mathbf{a}}^\ell$, $\hat{\mathbf{m}}^\ell$ are the vertices/midpoints of edges of the “unit triangle”. We can give a precise formula based on the reference shape functions for degree-2 triangular Lagrangian finite elements, cf. (2.6.1.6),

$$\begin{aligned} \hat{b}^1(\hat{\mathbf{x}}) &= (1 - 2\hat{x}_1 - 2\hat{x}_2)(1 - \hat{x}_1 - \hat{x}_2) , & \hat{b}^2(\hat{\mathbf{x}}) &= (2\hat{x}_1 - 1)\hat{x}_1 & \hat{b}^3(\hat{\mathbf{x}}) &= (2\hat{x}_2 - 1)\hat{x}_2 , \\ \hat{b}^4(\hat{\mathbf{x}}) &= 4(1 - \hat{x}_1 - \hat{x}_2)\hat{x}_1 , & \hat{b}^5(\hat{\mathbf{x}}) &= 4\hat{x}_1\hat{x}_2 , & \hat{b}^6(\hat{\mathbf{x}}) &= 4(1 - \hat{x}_1 - \hat{x}_2)\hat{x}_2 , \end{aligned} \quad (2.8.4.5)$$

►
$$\Phi_K(\hat{\mathbf{x}}) = \mathbf{a}_K^1 \hat{b}^1(\hat{\mathbf{x}}) + \mathbf{a}_K^2 \hat{b}^2(\hat{\mathbf{x}}) + \mathbf{a}_K^3 \hat{b}^3(\hat{\mathbf{x}}) + \mathbf{m}_K^1 \hat{b}^4(\hat{\mathbf{x}}) + \mathbf{m}_K^2 \hat{b}^5(\hat{\mathbf{x}}) + \mathbf{m}_K^3 \hat{b}^6(\hat{\mathbf{x}}) \quad (2.8.4.6)$$

The L^EH^RFEM++ implementation of the mapping Φ_K is given in Code 2.8.4.7.

C++ code 2.8.4.7: `Global()` member function of `If::geometry::TriaO2` → [GITHUB](#)

```

2 Eigen::MatrixXd TriaO2::Global(const Eigen::MatrixXd& local) const {
3   LF_VERIFY_MSG((0. <= local.array()).all() && (local.array() <= 1.).all(),
4                 "local coordinates out of bounds for reference element");
5   // Direct vectorized evaluation of componentwise quadratic mapping
6   // given through its monomial coefficients.
7   return ((beta_* local) + (gamma_* local.array().square().matrix()) +
8          (delta_* local.row(0).cwiseProduct(local.row(1))) +
9          .colwise() +
10         alpha_);
11 }
```

The member variables α_* , β_* , γ_* , δ_* contain the coefficients of the vector-valued quadratic polynomial Φ_K .

If some of the “midpoints” \mathbf{m}_K^ℓ , $\ell = 1, 2, 3$, are too far off the midpoints of the straight line segment $\frac{1}{2}(\mathbf{a}_K^\ell + \mathbf{a}_K^{\ell+1})$, then the bijectivity of the resulting bi-quadratic mapping $\Phi_K : \hat{K} \rightarrow K$ may break down. There might arise

$$\hat{x} \in \hat{K}: \det D\Phi_K(\hat{x}) = 0.$$

The following example demonstrates this. The green line in Fig. 165 shows the shape of a `If::geometry::TriaO2` mesh entity object with

$$[\mathbf{a}_K^1 \quad \mathbf{a}_K^2 \quad \mathbf{a}_K^3] = \begin{bmatrix} 1 & 6 & 3 \\ 1 & 3 & 8 \end{bmatrix}, \quad [\mathbf{m}_K^1 \quad \mathbf{m}_K^2 \quad \mathbf{m}_K^3] = \begin{bmatrix} 3.7 & 4.2 & 2.3 \\ 1.2 & 5.2 & 4.5 \end{bmatrix}.$$

(* $\hat{\cdot}$ \doteq vertices \mathbf{a}_K^ℓ , * $\hat{\cdot}$ \doteq “midpoints” \mathbf{m}_K^ℓ) This is a “moderately deformed triangle”.

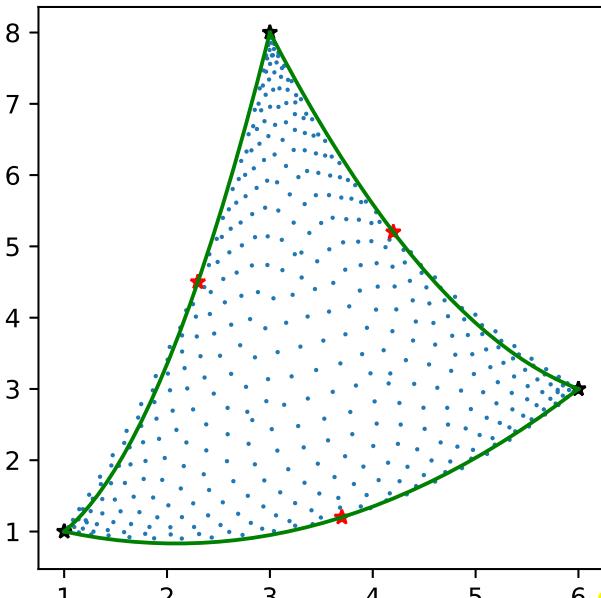


Fig. 165

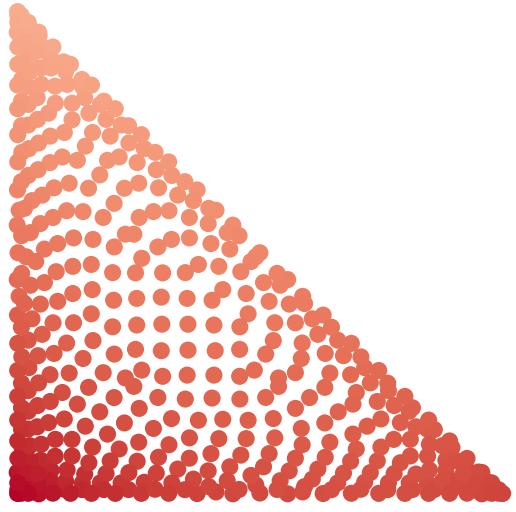


Fig. 166

Fig. 166 shows the sign of $\hat{x} \mapsto \det D\Phi_K(\hat{x})$, $\hat{x} \in \hat{K}$, bluish $\doteq < 0$, reddish $\doteq > 0$: no change of sign.

In Fig. 167 we see a strongly deformed triangle with

$$[\mathbf{a}_K^1 \quad \mathbf{a}_K^2 \quad \mathbf{a}_K^3] = \begin{bmatrix} 1 & 6 & 3 \\ 1 & 3 & 8 \end{bmatrix}, \quad [\mathbf{m}_K^1 \quad \mathbf{m}_K^2 \quad \mathbf{m}_K^3] = \begin{bmatrix} 5 & 4.5 & 1.75 \\ 4 & 9 & 6.5 \end{bmatrix}.$$

(Fig. 167: $\star \hat{\triangleq}$ vertices a_K^ℓ , $\star \hat{\triangleq}$ “midpoints” m_K^ℓ)

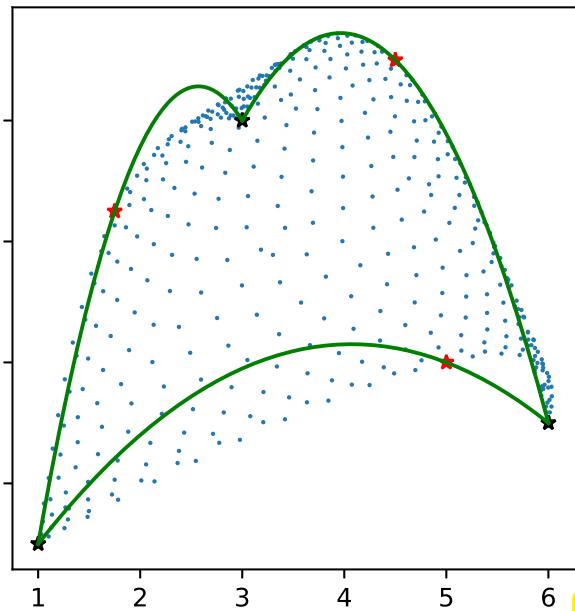


Fig. 167

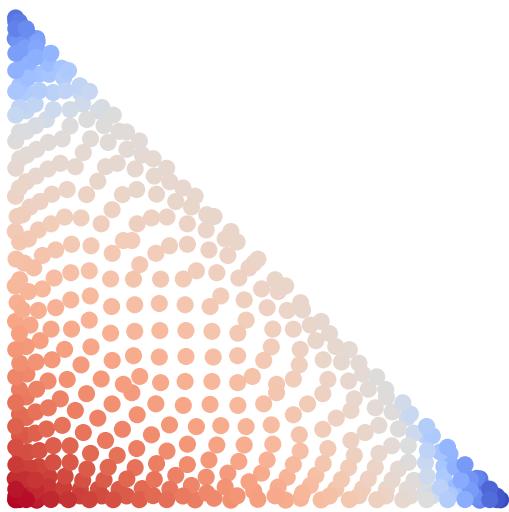


Fig. 168

In this case $\hat{x} \mapsto \det D\Phi_K(\hat{x})$ changes sign on \hat{K} and there will be points $\hat{x} \in \hat{K}$ where $D\Phi_K(\hat{x})$ becomes singular; Ass. 2.8.3.3 is violated. □

Review question(s) 2.8.4.8 (Parametric Finite Elements)

(Q2.8.4.8.A) Which data required for the computation of the element matrices for 2nd-order elliptic variational problems discretized by means of Lagrangian finite elements, depend on the current cell, and which do not?

(Q2.8.4.8.B) [Transformation of an elliptic BVP] The mapping $\hat{x} \mapsto [a\hat{x}_1, b\hat{x}_2]^\top$, $a, b > 0$, takes the unit disc $\hat{D} \subset \mathbb{R}^2$ to a ellipse Ω with axes a and b . Let $u \in H_0^1(\Omega)$ solve $\Delta u = f$, $f \in L^2(\Omega)$. Using Lemma 2.8.3.10, derive the variational problem solved by the pullback of u to \hat{D} .

To answer this question use the variational formulation and the transformation formula for the gradient:

Lemma 2.8.3.10. Transformation formula for gradients

For differentiable $u : K \mapsto \mathbb{R}$ and any diffeomorphism $\Phi : \hat{K} \mapsto K$ we have

$$(\mathbf{grad}_{\hat{x}}(\Phi^* u))(\hat{x}) = (D\Phi(\hat{x}))^\top \underbrace{(\mathbf{grad}_x u)(\Phi(\hat{x}))}_{=\Phi^*(\mathbf{grad} u)(\hat{x})} \quad \forall \hat{x} \in \hat{K}. \quad (2.8.3.11)$$

(Q2.8.4.8.C) [Bilinear transformation onto a triangle] Give the formula for the bilinear transformation that maps the unit square to the “triangular” quadrilateral with vertices $\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

(Q2.8.4.8.D) [Decomposing bilinear transformations] The unit square $[0, 1]^2$ can be mapped onto a general non-degenerate quadrilateral K by means of a mapping Φ_K composed of (i) an affine mapping Φ_K^{aff} (\rightarrow Def. 2.7.5.13) and (ii) a mapping $\mathbb{R}^2 \rightarrow \mathbb{R}^2$, $\hat{x} \mapsto d\hat{x}_1 \hat{x}_2$. Find formulas for both Φ_K^{aff} and $d \in \mathbb{R}^2$ in terms of the vertex coordinates a^1, \dots, a^4 of K .

(Q2.8.4.8.E) [Degenerate bilinear transformation] For which $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$ will the determinant of the bilinear transformation that takes the unit square to a quadrilateral with vertices

$$a^1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad a^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad a^3 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad a^4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

change sign.

Recall the formulas for bilinear transformations:

$$\text{For } \Phi(\hat{x}) = \begin{bmatrix} \alpha_1 + \beta_1\hat{x}_1 + \gamma_1\hat{x}_2 + \delta_1\hat{x}_1\hat{x}_2 \\ \alpha_2 + \beta_2\hat{x}_1 + \gamma_2\hat{x}_2 + \delta_2\hat{x}_1\hat{x}_2 \end{bmatrix}, \quad \alpha_i, \beta_i, \gamma_i, \delta_i \in \mathbb{R}, \quad (2.8.3.18a)$$

$$\Rightarrow D\Phi(\hat{x}) = \begin{bmatrix} \beta_1 + \delta_1\hat{x}_2 & \gamma_1 + \delta_1\hat{x}_1 \\ \beta_2 + \delta_2\hat{x}_2 & \gamma_2 + \delta_2\hat{x}_1 \end{bmatrix}, \quad (2.8.3.18b)$$

$$\Rightarrow \det(D\Phi(\hat{x})) = \beta_1\gamma_2 - \beta_2\gamma_1 + (\beta_1\delta_2 - \beta_2\delta_1)\hat{x}_1 + (\delta_1\gamma_2 - \delta_2\gamma_1)\hat{x}_2. \quad (2.8.3.18c)$$

(Q2.8.4.8.F) [Mapping a square to a circle] The unit disk $\{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| < 1\}$ can be equipped with a mesh consisting of four congruent curvilinear triangles.

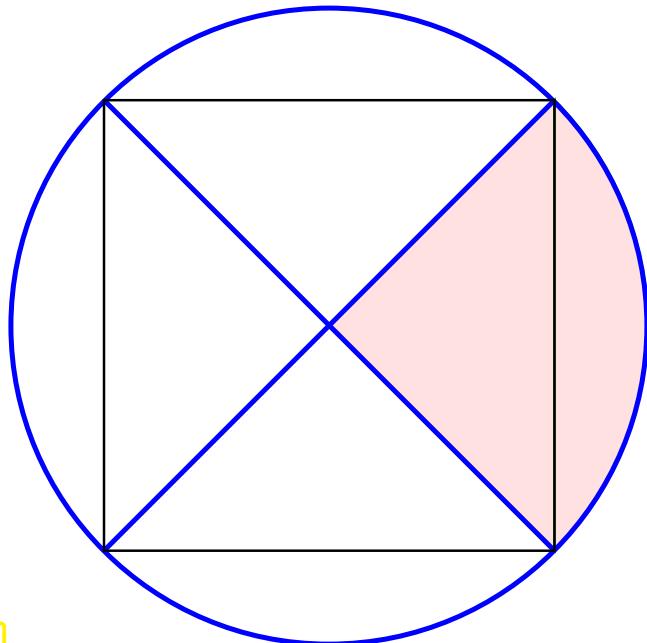


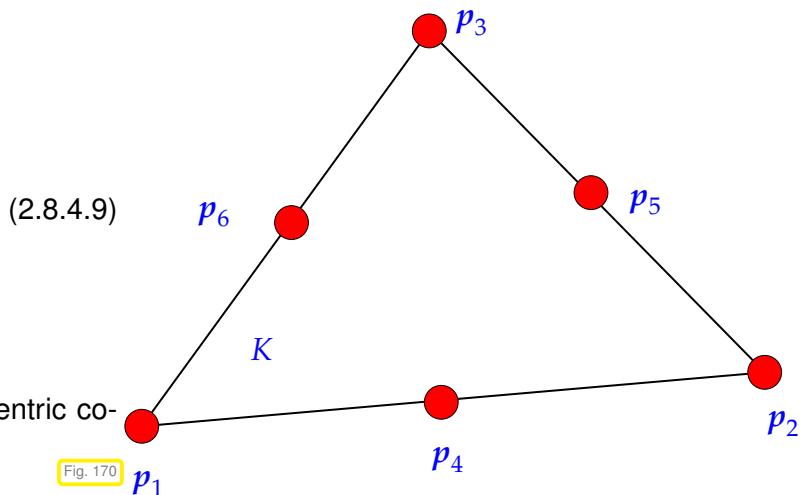
Fig. 169

One of these triangles K^* is shaded in the figure beside.

Which mapping $\Phi_K : K \rightarrow K^*$ takes the triangle K with straight edges and contained in K^* to K^* ? Restricted to the straight edges of K^* it should coincide with the identity mapping.

(Q2.8.4.8.G) [Hierarchical local shape functions for $S_2^0(\mathcal{M})$] For the quadratic Lagrangian finite element space $S_2^0(\mathcal{M})$ the following choice of local shape functions on a triangle describes the **hierarchical local shape functions**.

$$\begin{aligned} b_K^1 &= \lambda_1, \\ b_K^2 &= \lambda_2, \\ b_K^3 &= \lambda_3, \\ b_K^4 &= 4\lambda_1\lambda_2, \\ b_K^5 &= 4\lambda_2\lambda_3, \\ b_K^6 &= 4\lambda_1\lambda_3, \end{aligned} \quad (2.8.4.9)$$



where λ_ℓ , $\ell = 1, 2, 3$, stands for the barycentric coordinate functions of K .

Fig. 170

You have to develop a specialization of **If::fe::ScalarReferenceFiniteElement** for a cell of type `TRIA`, the set (2.8.4.9) of local shape functions, and local evaluation/interpolation nodes as given in Fig. 170. Sketch the implementation of the method `NodalValuesToDofs()`.

(Q2.8.4.8.H) Outline the implementation of a function in `LEHRFEM++` that takes a vector \vec{p} of expansion coefficients of a finite element function $u_h \in S_2^0(\mathcal{M})$ (\mathcal{M} a triangular mesh), a suitable

If::uscalfe::FeSpaceLagrangeO2 argument, and a coordinate vector $\mathbf{p} \in \mathbb{R}^2$ and returns the value $u_h(\mathbf{p})$.

△

Learning Outcomes

This is what you should master after you have thoroughly studied this chapter:

- You must know in detail the idea of the Galerkin discretization of a linear variational problem and how it leads to a linear system of equations.
- You should remember the concept of a finite-element mesh.
- You should be familiar with the notion of global and local finite-element shape functions.
- You should know what it means that global/local finite-element shape functions are associated with a geometric entity and that they cover others.
- You should understand the local supports of global finite-element shape functions and their consequences.
- You should know the definition and canonical global shape functions for Lagrangian finite element spaces of degree one and two on 2D hybrid meshes.
- You grasp the concept of parametric finite-element spaces and how it can be harnessed in the implementation of finite-element methods.
- You can explain how the parametric finite-element constructions can be used to deal with curved boundaries and interfaces.
- Skills concerning LEHRFEM++, access to C++ reference pages and complete EIGEN and LEHRFEM++ DOXYGEN documentation assumed:
 - You can retrieve information fast from the online documentation of EIGEN and LEHRFEM++.
 - You should be able to extract and process various kinds of topological and geometric information from objects encoding a finite-element mesh.
 - You must know the role and use of local-to-global index mappings as provided by **If::assemble::DofHandler**.
 - You should be able to write bespoke classes providing element matrices and vectors and use them build linear systems of equations.
 - You must know how to use LEHRFEM++’s built-in numerical quadrature facilities.
 - You understand the way how LEHRFEM++ implements parametric finite elements.
 - You can implement essential boundary conditions in LEHRFEM++.
 - You know how to work with finite-element functions given through their basis expansion coefficient vectors.

Bibliography

- [And+21] Robert Anderson, Julian Andrej, Andrew Barker, and et al. “MFEM: A modular finite element methods library”. In: *Comput. Math. Appl.* 81 (2021), pp. 42–74. DOI: [10.1016/j.camwa.2020.06.009](https://doi.org/10.1016/j.camwa.2020.06.009) (cit. on p. 208).
- [BS02] S. Brenner and R. Scott. *Mathematical theory of finite element methods*. 2nd. Texts in Applied Mathematics. Springer–Verlag, New York, 2002 (cit. on p. 292).
- [Can+15] C.D. Cantwell et al. “Nektar++: An open-source spectral/hp element framework”. In: *Computer Physics Communications* 192 (2015), pp. 205–219. DOI: <https://doi.org/10.1016/j.cpc.2015.02.008> (cit. on p. 136).
- [Che11] Z.-X. Chen. *The Finite Element Method*. Singapore: World Scientific, 2011 (cit. on p. 136).
- [Dun85] D.A. Dunavant. “High degree efficient symmetrical Gaussian quadrature rules for the triangle”. In: *Int. J. Numer. Meth. Engr.* 21 (1985), pp. 1129–1148 (cit. on p. 258).
- [FPW11] Stefan Funken, Dirk Praetorius, and Philipp Wissgott. “Efficient implementation of adaptive P1-FEM in Matlab”. In: *Comput. Methods Appl. Math.* 11.4 (2011), pp. 460–490. DOI: [10.2478/cmam-2011-0026](https://doi.org/10.2478/cmam-2011-0026) (cit. on p. 160).
- [GR09] Christophe Geuzaine and Jean-François Remacle. “Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities”. In: *Internat. J. Numer. Methods Engrg.* 79.11 (2009), pp. 1309–1331. DOI: [10.1002/nme.2579](https://doi.org/10.1002/nme.2579) (cit. on p. 209).
- [GCS92] J.R. Gilbert, C.Moler, and R. Schreiber. “Sparse Matrices in MATLAB: Design and Implementation”. In: *SIAM Journal on Matrix Analysis and Applications* 13.1 (1992), pp. 333–356 (cit. on p. 180).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 140, 144, 148, 150–152, 155–157, 165, 168, 170, 180, 184, 191, 226, 229, 253, 256, 257, 259).
- [KA03] P. Knabner and L. Angermann. *Numerical Methods for Elliptic and Parabolic Partial Differential Equations*. Vol. 44. Texts in Applied Mathematics. Heidelberg: Springer, 2003 (cit. on p. 173).
- [LC94] J. N. Lyness and Ronald Cools. “A survey of numerical cubature over triangles”. In: *Mathematics of Computation 1943–1993: a half-century of computational mathematics (Vancouver, BC, 1993)*. Vol. 48. Proc. Sympos. Appl. Math. Providence, RI: Amer. Math. Soc., 1994, pp. 127–150 (cit. on p. 249).
- [Mar12] C. Marcati. *Tspeed – A C++ library for spectral discontinuous methods on simplicial elements in elastodynamics*. Online lecture notes, Politecnico di Milano. 2012 (cit. on pp. 136, 271).
- [PR05] Yehuda Pinchover and Jacob Rubinstein. *An introduction to partial differential equations*. Cambridge University Press, Cambridge, 2005, pp. xii+371 (cit. on p. 143).
- [Say08] F. Sayas. *A Gentle Introduction to the Finite Element Method*. Online Lecture notes, www.math.umn.edu/~sayas002/anIntro2FEM.pdf. 2008 (cit. on p. 136).
- [Str09] M. Struwe. *Analysis für Informatiker*. Lecture notes, ETH Zürich. 2009 (cit. on pp. 255, 282).

Chapter 3

FEM: Convergence and Accuracy

It goes without saying that, on the one hand, a finite element Galerkin solution $u_h \in H^1(\Omega)$ of a scalar second-order elliptic boundary value problem will hardly ever be equal to its exact solution u . On the other hand, it also goes without saying that we want it to be “close” to u in some sense. This motivates the study of the accuracy of Galerkin solutions u_h of variational boundary value problems. More precisely, in this chapter we are going to examine the *asymptotic convergence* of relevant norms $\|u - u_h\|$ of the discretization error $u - u_h$ as we let the dimension of the discrete trial space tend to ∞ .

Based on the developments of the previous chapters, the focus will be on **finite element Galerkin discretization** of *linear* scalar 2nd-order elliptic boundary value problems in 2D, 3D.

Contents

3.1 Abstract Galerkin Error Estimates	301
3.1.1 Setting	301
3.1.2 Galerkin Discretization Error	302
3.1.3 Galerkin Orthogonality (GO)	304
3.1.4 Refinement	305
3.2 Empirical (Asymptotic) Convergence of Lagrangian FEM	309
3.2.1 Asymptotic Convergence	310
3.2.2 Algebraic and Exponential Convergence	311
3.2.3 Convergence of FEM: Numerical Experiments	314
3.3 A Priori (Asymptotic) Finite Element Error Estimates	324
3.3.1 Estimates for Linear Interpolation in 1D	325
3.3.2 Error Estimates for Linear Interpolation in 2D	329
3.3.3 The Sobolev Scale of Function Spaces	336
3.3.4 Anisotropic Interpolation Error Estimates	338
3.3.5 General Approximation Error Estimates for Lagrangian FEM: A Survey	341
3.4 Elliptic Regularity Theory	350
3.5 Variational Crimes	355
3.5.1 The Impact of Numerical Quadrature	357
3.5.2 Approximation of the Boundary	358
3.6 FEM: Duality Techniques for Error Estimation	362
3.6.1 Linear Output Functionals	362
3.6.2 Case Study: Computation of Boundary Fluxes with FEM	366
3.6.3 Lagrangian FEM: L^2 -Estimates	372
3.7 Discrete Maximum Principle	377
3.7.1 Maximum Principle for Scalar 2nd-Order Elliptic BVPs	377
3.7.2 Maximum Principle for Piecewise Linear Lagrangian FEM	379
3.8 Validation and Debugging of Finite Element Codes	387

§3.0.0.1 (Prerequisite knowledge) Familiarity with the following concepts is essential for understanding the material in this chapter:

- Second-order elliptic boundary value problems (from equilibrium models, diffusion models): Section 1.5, Section 1.7,
- Variational formulation: Section 1.8, see also (1.4.2.4), (1.8.0.16), (2.1.2.2),
- Some Sobolev spaces and their norms: $L^2(\Omega)$, $H^1(\Omega)$, see Section 1.3
- Abstract Galerkin discretization: Section 2.2,
- Lagrangian finite elements: Section 2.6, Section 2.4.

□

3.1 Abstract Galerkin Error Estimates



Video tutorial for Section 3.1: Abstract Galerkin Error Estimates: (38 minutes) [Download link](#), [tablet notes](#)

3.1.1 Setting

§3.1.1.1 (Linear variational problems revisited → **Section 1.4.1)** In this section we adopt an abstract perspective and study **linear variational problems** (1.4.1.7) in the form

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0, \quad (2.2.0.2)$$

with the following standard ingredients, see also Def. 1.4.1.6:

- ◆ $V_0 \hat{=} (\text{real})$ vector space, a space of functions $\Omega \mapsto \mathbb{R}$ for scalar 2nd-order elliptic variational problems,
- ◆ $a : V_0 \times V_0 \mapsto \mathbb{R} \hat{=} \text{a bilinear form}$, see Def. 0.3.1.4,
- ◆ $\ell : V_0 \mapsto \mathbb{R} \hat{=} \text{a linear form}$, see Def. 0.3.1.4,

In addition we want (2.2.0.2) to be related to a **quadratic minimization problem** (→ Def. 1.2.3.11), which suggests the following assumption.

Assumption 3.1.1.2. S.p.d. bilinear form

The bilinear form $a : V_0 \times V_0 \mapsto \mathbb{R}$ in (2.2.0.2) is symmetric and **positive definite** (→ Def. 1.2.3.27).



a supplies an inner product on V_0



a induces **energy norm** $\|\cdot\|_a$ on V_0 (→ Def. 1.2.3.35)

We also want (2.2.0.2) to be stable, remember the discussion in Section 1.4.3; In Section 1.2.3.4 elaborates the motivation for the next assumption.

Assumption 3.1.1.3. Continuity of r.h.s. functional

The right hand side functional $\ell : V_0 \mapsto \mathbb{R}$ from (2.2.0.2) is *continuous* (\rightarrow Def. 1.2.3.42) w.r.t. to the energy norm (\rightarrow Def. 1.2.3.35) induced by a :

$$\exists C > 0: |\ell(u)| \leq C \|u\|_a \quad \forall u \in V_0 . \quad (1.2.3.40)$$

The next assumption is meant to appease fastidious mathematicians, see Rem. 1.3.3.9 for further discussion:

Assumption 3.1.1.4.

V_0 equipped with the energy norm $\|\cdot\|_a$ is a *Hilbert space*, that is, *complete* (\rightarrow Def. 1.3.3.4).

Theorem 3.1.1.5. Existence and uniqueness of solution of linear variational problem

Under Ass. 3.1.1.2–Ass. 3.1.1.4 the linear variational problem has a unique solution $u \in V_0$.

This repeats Thm. 1.3.3.6 from Rem. 1.3.3.9 and is also known as *Riesz representation theorem* for continuous linear functionals. \square

Remark 3.1.1.6 (Repetition: Well-posed 2nd-order linear elliptic variational problems) For instance, thanks to the Poincaré-Friedrichs inequality from Thm. 1.3.4.17, Ass. 3.1.1.2 is satisfied for the bilinear form of a second-order linear elliptic (pure) Dirichlet problem, see (1.5.3.8), (1.4.2.4), with

$$a(u, v) := \int_{\Omega} (\alpha(x) \mathbf{grad} u) \cdot \mathbf{grad} v \, dx , \quad u, v \in H_0^1(\Omega) , \quad (3.1.1.7)$$

and uniformly positive definite (\rightarrow Def. 1.2.2.9) coefficient tensor $\alpha : \Omega \mapsto \mathbb{R}^{d,d}$, see Section 1.2.3.

A second-order linear elliptic Neumann problem involves the same bilinear form (3.1.1.7), but Ass. 3.1.1.2 holds only on the smaller space

$$H_*^1(\Omega) := \{v \in H^1(\Omega): \int_{\Omega} v(x) \, dx = 0\} , \quad (1.8.0.15)$$

thanks to second Poincaré-Friedrichs inequality from Thm. 1.8.0.20.

For the right hand side functional of a 2nd-order Neumann problem, see (1.9.0.2), (1.8.0.16),

$$\ell(v) := \int_{\Omega} f(x)v(x) \, dx + \int_{\partial\Omega} h(x)v(x) \, dS , \quad v \in H^1(\Omega) ,$$

we found in Section 1.3, see (1.3.4.15), and § 1.9.0.9, that $f \in L^2(\Omega)$ and $h \in L^2(\partial\Omega)$ ensures Ass. 3.1.1.3.

Ass. 3.1.1.4 for a from (3.1.1.7) is a deep result in the theory of Sobolev spaces [Eva98, Sect. 5.2.3, Thm. 2]. It has been stated earlier as Thm. 1.3.4.11. \square

3.1.2 Galerkin Discretization Error

§3.1.2.1 (Galerkin discretization error) Now consider a Galerkin discretization of (2.2.0.2) (\rightarrow Section 2.2) based on the Galerkin trial/test space

$$V_{0,h} \subset V_0 \quad , \quad N := \dim V_{0,h} < \infty ,$$

which leads to the discrete variational problem

$$u_h \in V_{0,h}: \quad a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h} . \quad (2.2.1.1)$$

Since (2.2.1.1) is a linear variational problem with s.p.d. bilinear form posed on a finite-dimensional space, Thm. 2.2.1.5 guarantees existence and uniqueness of the Galerkin solution $u_h \in V_{0,h}$.

Our goal:	Bound <i>relevant norms</i> of discretization error $u - u_h$
-----------	----------------------------------------------------------------------

Note that the discretization error is an element of the vector space V_0 . For instance, in the particular case of the second-order elliptic Dirichlet problem (3.1.1.7), the discretization error is a function in $H_0^1(\Omega)$.

§3.1.2.2 (“Relevant” norms) For the abstract linear variational problem (2.2.0.2) the energy norm immediately comes to mind as a relevant norm. Recall from Section 1.4.1 that (2.2.0.2) is an equivalent statement of a minimization problem for a quadratic energy functional $J: V_0 \rightarrow \mathbb{R}$:

$$\begin{aligned} u \in V_0: \quad & a(u, v) = \ell(v) \quad v \in V_0 \\ & \Updownarrow \\ u = \underset{v \in V_0}{\operatorname{argmin}} J(v) \quad & \text{for} \quad J(v) := \frac{1}{2}a(v, v) - \ell(v) . \end{aligned}$$

Since minimization of an energy represents the fundamental equilibrium condition of the modeled “physical/real-world” phenomenon, energy will certainly be a “relevant” quantity. Accepting this, the **energy norm** $\|\cdot\|_a$ ($\|v\|_a^2 := a(v, v)$, Def. 1.2.3.35) is promoted to a relevant norm for measuring the size of discretization error, because it immediately tells us how far we are off the minimal energy:

Lemma 3.1.2.3. Energy norm and energy deviation

Let $u \in V_0$ solve the linear variational problem (2.2.0.2),

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0 , \quad (2.2.0.2)$$

and let Ass. 3.1.1.2–Ass. 3.1.1.4 be satisfied. Then, with $J(v) := \frac{1}{2}a(v, v) - \ell(v)$,

$$J(w) - J(u) = \frac{1}{2}\|w - u\|_a^2 \quad \forall w \in V_0 . \quad (3.1.2.4)$$

Proof. We copy the proof of Thm. 1.4.1.8, which was even slightly more general, because it addressed variational problems on affine spaces:

$$\begin{aligned} a(u, v) = \ell(v) \quad \forall v \in V_0 \Rightarrow J(w) - J(u) &= \frac{1}{2}a(w, w) - \ell(w) - \frac{1}{2}a(u, u) + \ell(u) \\ &= \frac{1}{2}a(w, w) - \ell(w - u) - \frac{1}{2}a(u, u) \\ &= \frac{1}{2}a(w, w) - a(u, w - u) + \frac{1}{2}a(u, u) \\ &= \frac{1}{2}a(w - u, w - u) = \frac{1}{2}\|w - u\|_a^2 , \end{aligned}$$

for every $w \in V_0$. \square

No doubt, energy is a key quantity for the solution of an equilibrium problem. However, other relevant norms can be suggested by the application context and objectives of numerical simulations. For second-order elliptic boundary value problems, such other relevant norms can be the following, see Section 0.3.2.4 and Section 1.3,

- the mean square norm or $L^2(\Omega)$ -norm, see Def. 1.3.2.3,
- the supremum norm or $L^\infty(\Omega)$ -norm, see Def. 0.3.2.25.

□

3.1.3 Galerkin Orthogonality (GO)

The Galerkin approach allows a remarkably simple bound of the energy norm of the **discretization error** $u - u_h$. For the latter we immediately derive the following relationship:

$$\begin{aligned} \mathbf{a}(u, v) &= \ell(v) \quad \forall v \in V_0, \\ \mathbf{a}(u_h, v_h) &= \ell(v_h) \quad \forall v_h \in V_{0,h} \end{aligned} \implies \mathbf{a}(u - u_h, v_h) = 0 \quad \forall v_h \in V_{0,h}. \quad (3.1.3.1)$$

This is known under a special name:

Galerkin orthogonality

$\mathbf{a}(u - u_h, v_h) = 0 \quad \forall v_h \in V_{0,h}.$ (3.1.3.2)

[Geometric meaning for inner product $\mathbf{a}(\cdot, \cdot)$ →]

Remark 3.1.3.3 (“Energy-geometry”) In linear algebra we learned that an symmetric positive definite bilinear form (\rightarrow Def. 1.2.3.27) on a (finite-dimensional) vector spaces induces a Euclidean geometry with meaningful notions of length (\rightarrow Def. 1.2.3.35) and angle. This carries over to Hilbert spaces and makes it possible for us to draw “geometric” pictures like Fig. 171.

□

Parlance: We say that the discretization error $e_h := u - u_h$ is “ $\mathbf{a}(\cdot, \cdot)$ -orthogonal” to the discrete trial/test space V_h .

Remark 3.1.3.4 (Pythagoras’ theorem)

Fig. 172

If $\mathbf{a}(\cdot, \cdot)$ is inner product on V “Pythagoras’ theorem” tells us, see Fig. 172:

$$\|u - v_h\|_a^2 = \|u - u_h\|_a^2 + \|u_h - v_h\|_a^2. \quad (3.1.3.5)$$

This is immediate from (3.1.3.2) and the bilinearity of $\mathbf{a}(\cdot, \cdot)$.

Notice that (3.1.3.5) with $v_h = 0$ gives a simple formula for computation of energy norm of Galerkin discretization error in numerical experiments with known solution u of (2.2.0.2) and u_h of (2.2.1.1):

$$\|u - u_h\|_a^2 = \|u\|_a^2 - \|u_h\|_a^2. \quad (3.1.3.6)$$

□

In Euclidean geometry: the point in a hyperplane nearest to a given point is its orthogonal projection onto the hyperplane. The next theorem states this for the inner product $\mathbf{a}(\cdot, \cdot)$ and $V_{0,h}$ instead of a hyperplane, see Fig. 172 for an illustration.

Theorem 3.1.3.7. Cea's lemma

Under Ass. 3.1.1.2–Ass. 3.1.1.4 the energy norm of the Galerkin discretization error for (2.2.0.2) satisfies

$$\|u - u_h\|_a = \inf_{v_h \in V_{0,h}} \|u - v_h\|_a .$$

Proof. Use bilinearity of a and Galerkin orthogonality (3.1.3.2): for *any* $v_h \in V_{0,h}$

$$\|u - u_h\|_a^2 = a(u - u_h, u - u_h) = a(u - v_h, u - u_h) + \underbrace{a(v_h - u_h, u - u_h)}_{=0} .$$

Next, use the Cauchy-Schwartz inequality (CSI) for the inner product a :

$$\begin{aligned} \text{CSI: } \quad a(u, v) &\leq \|u\|_a \|v\|_a \quad \forall u, v \in V_0 . \\ \Rightarrow \|u - u_h\|_a^2 &\leq \|u - v_h\|_a \cdot \|u - u_h\|_a , \end{aligned}$$

and cancel one factor $\|u - u_h\|_a$. □

An alternative proof can invoke Pythagoras' theorem (3.1.3.5):

$$(3.1.3.5) \Rightarrow \|u - u_h\|_a^2 = \|u - v_h\|_a^2 - \|u_h - v_h\|_a^2 \leq \|u - v_h\|_a^2 \quad \forall v_h \in V_{0,h} .$$

We highlight an obvious, but fundamental consequence of Thm. 3.1.3.7:

Optimality of Galerkin solutions:

$$\|u - u_h\|_a = \inf_{v_h \in V_{0,h}} \|u - v_h\|_a , \quad (3.1.3.9)$$

((energy) norm of) **discretization error** ((energy) norm of) **best approximation error**

- ☞ As regards the energy norm, the Galerkin solution is the best possible solution we can obtain in a given trial space.

Thus, Cea's lemma Thm. 3.1.3.7 permits us to predict the accuracy of Galerkin solution w.r.t. the energy norm $\|\cdot\|_a$ by just studying the capability of functions in $V_{0,h}$ to approximate u !

3.1.4 Refinement

3.1.4.1 (More accurate solutions by refinement) From Cea's lemma Thm. 3.1.3.7 we infer a certain “monotonicity” of the energy norm of Galerkin discretization errors inherited from best approximation errors: consider different trial/test spaces

$$V_{0,h}, V'_{0,h} \subset V_0 , \quad V_{0,h} \subset V'_{0,h} \Rightarrow \inf_{v_h \in V'_h} \|u - v_h\|_a \leq \inf_{v_h \in V_{0,h}} \|u - v_h\|_a .$$

Thus, when we measure the Galerkin discretization error in the energy norm we can

improve accuracy by simply enlarging (“refining”) the trial space.

§3.1.4.2 (Refinement of finite element spaces) Now we return to Lagrangian finite element (\rightarrow Section 2.6) used for the Galerkin discretization of linear 2nd-order elliptic variational problems.

How can we accomplish the refinement of a (Lagrangian) FE space ?

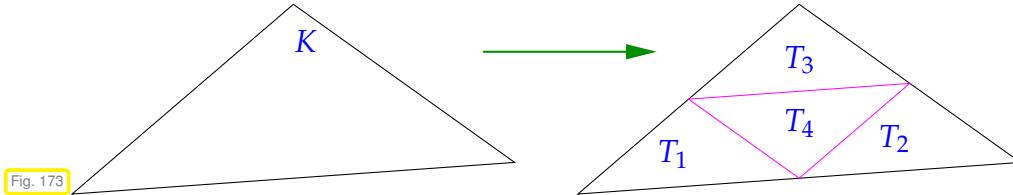
- **h-refinement:** Replace the mesh \mathcal{M} (underlying $V_{0,h}$) with a finer mesh \mathcal{M}' (underlying larger discrete trial space $V'_{0,N}$).

- **p-refinement:** Replace $V_{0,h} := \mathcal{S}_p^0(\mathcal{M})$, $p \in \mathbb{N}$, with $V'_{0,h} := \mathcal{S}_{p+1}^0(\mathcal{M}) \Rightarrow V_{0,h} \subset V'_{0,h}$

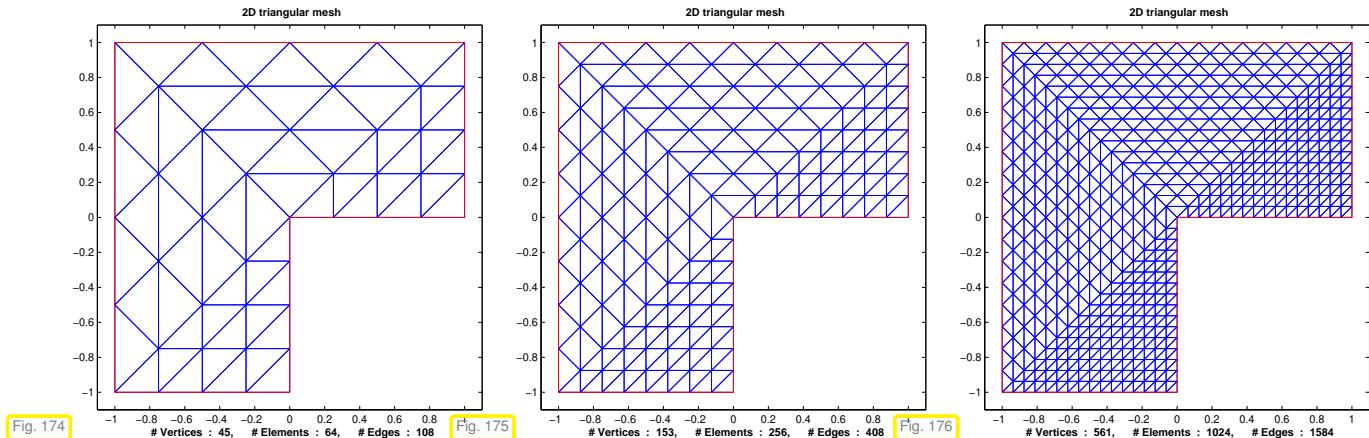
The extreme case of p-refinement amounts to the use of *global* polynomials on Ω as trial and test functions. This leads to the class of (polynomial) **spectral Galerkin methods**.

Combination of h-refinement and p-refinement ? OF COURSE (**hp-refinement**, [Sch98])

EXAMPLE 3.1.4.3 (Regular/uniform refinement of triangular mesh in 2D) “Regular refinement” of a triangle K means that it is split into four congruent “child triangles” T_1, T_2, T_3, T_4 :



Logically, the (global) regular/uniform refinement of a triangular mesh is accomplished by executing a regular refinement of every triangle:



Obviously, this process creates another valid finite-element mesh as no hanging nodes are introduced. Global regular refinement creates so-called **nested meshes**, where each (closed) cell of the coarse mesh \mathcal{M} is a union of (closed) cells of the fine mesh \mathcal{M}' . As a consequence Lagrangian finite element spaces of fixed degree $p \in \mathbb{N}$ (\rightarrow Def. 2.6.1.1) will also be nested:

$$\mathcal{M} \xrightarrow[\text{regular refinement}]{} \mathcal{M}' \quad \Rightarrow \quad \mathcal{S}_p^0(\mathcal{M}) \subset \mathcal{S}_p^0(\mathcal{M}') ,$$

that is, h -refinement through global regular refinement is a true refinement in the sense that it creates a larger finite element space, which contains the original finite element space.

EXAMPLE 3.1.4.4 (Global regular refinement in LEHRFEM++) LEHRFEM++ supports the refinement of 2D hybrid meshes in the module `lf:refinement`. At this point we only discuss high-level routines

that perform a fixed number of steps of regular refinement of meshes. A special data structure manages a sequence of meshes generate by successive refinement: **If::refinement::MeshHierarchy**. Beside keeping an array of meshes, this type also store copious information about the connections between the meshes. A member function

```
std::shared_ptr<lf::mesh::Mesh> getMesh(size_type level);
```

permits us to obtain a pointer to the mesh on a particular refinement level, `level == 0` corresponding to the coarsest mesh.

The following code demonstrates the creation of a **If::refinement::MeshHierarchy** object by the convenience function `lf::refinement::GenerateMeshHierarchyByUniformRefinement()`.

C++11 code 3.1.4.5: Demo code for usage of **If::refinement::MeshHierarchy** → GITHUB

```
2 void regrefMeshSequence(const std::shared_ptr<If::mesh::Mesh>& mesh_p,
3   int refsteps) {
4   std::shared_ptr<If::refinement::MeshHierarchy> multi_mesh_p =
5     If::refinement::GenerateMeshHierarchyByUniformRefinement(mesh_p, refsteps);
6   If::refinement::MeshHierarchy& multi_mesh{*multi_mesh_p};
7
8   // Output summary information about hierarchy of nested meshes
9   std::cout << "\t Sequence of nested meshes created\n";
10  multi_mesh.PrintInfo(std::cout);
11  // Number of levels
12  size_type L = multi_mesh.NumLevels();
13
14  // Retrieve meshes on all levels
15  for (size_type level = 0; level < L; ++level) {
16    std::shared_ptr<const If::mesh::Mesh> lev_mesh_p =
17      multi_mesh.getMesh(level);
18    // Reference to current mesh
19    const If::mesh::Mesh& mesh{*lev_mesh_p};
20    // Output of mesh information
21    std::cout << "==== Mesh on level " << level << ": "
22    << mesh.NumEntities(If::base::RefEl::kPoint()) << " NODEs, "
23    << mesh.NumEntities(If::base::RefEl::kSegment()) << " EDGEs, "
24    << mesh.NumEntities(If::base::RefEl::kTria()) << " TRIAs, "
25    << mesh.NumEntities(If::base::RefEl::kQuad()) << " QUADs."
26    << std::endl;
27  }
28 }
```

It may be interesting to see the code actually constructing the sequence of meshes using the `RefineRegular()` method of **If::refinement::MeshHierarchy**.

C++11 code 3.1.4.6: Implementation of **GenerateMeshHierarchyByUniformRefinement** → GITHUB

```
2 std::shared_ptr<MeshHierarchy> GenerateMeshHierarchyByUniformRefinement(
3   const std::shared_ptr<If::mesh::Mesh> &mesh_p, If::base::size_type ref_lev,
4   RefPat ref_pat) {
5   LF_ASSERT_MSG(mesh_p != nullptr, "No valid mesh supplied!");
6   // Set up the builder object for mesh entities, here suitable for a 2D
7   // hybrid mesh comprising triangles and quadrilaterals
8   std::unique_ptr<If::mesh::hybrid2d::MeshFactory> mesh_factory_ptr =
9     std::make_unique<If::mesh::hybrid2d::MeshFactory>(2);
10  // Create a mesh hierarchy with a single level
11  std::shared_ptr<MeshHierarchy> multi_mesh_p =
```

```

12     std::make_shared<MeshHierarchy>(mesh_p, std::move(mesh_factory_ptr));
13 // Perform the desired number of steps of uniform refinement
14 for (unsigned refstep = 0; refstep < ref_lev; ++refstep) {
15     // Conduct regular refinement of all cells of the currently finest
16     // mesh.
17     // This adds another mesh to the sequence of meshes.
18     multi_mesh_p->RefineRegular(ref_pat);
19 }
20 }
```

Review question(s) 3.1.4.7 (Estimates for Galerkin discretization error)

(Q3.1.4.7.A) [Galerkin orthogonality] Explain the notion of “**Galerkin orthogonality**” and prove it for the Galerkin discretization of a linear variational problem

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0,$$

with a symmetric positive definite (s.p.d.) bilinear form $a: V_0 \times V_0 \rightarrow \mathbb{R}$.

(Q3.1.4.7.B) [Pythagoras’ theorem] We consider the Galerkin discretization of the linear variational problem

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0,$$

with a symmetric positive definite (s.p.d.) bilinear form $a: V_0 \times V_0 \rightarrow \mathbb{R}$ on two *nested* finite dimensional subspaces $V_{0,H}, V_{0,h}$ of V_0 :

$$V_{0,H} \subset V_{0,h} \subset V_0, \quad \dim V_{0,H} \leq \dim V_{0,h} < \infty.$$

Designate by $u_H \in V_{0,H}$ and $u_h \in V_{0,h}$ the two Galerkin solutions. Show that

$$\|u_{0,h} - v_H\|_a^2 = \|u_{0,h} - u_{0,H}\|_a^2 + \|u_{0,H} - v_H\|_a^2 \quad \forall v_H \in V_{0,H}.$$

(Q3.1.4.7.C) [Generalized Cea’s lemma] Let the bilinear form $a: V \times V \rightarrow \mathbb{R}$ on a normed real vector space satisfy

$$|a(u, v)| \leq C\|u\|\|v\| \quad \forall u, v \in V, \quad |a(v, v)| \geq \gamma\|v\|^2 \quad \forall v \in V,$$

for some constants $C > 0, \gamma > 0$. Derive a bound for the norm $\|u - u_h\|$ of the Galerkin discretization error for a linear variational problem with bilinear form a in terms of the best approximation error for its exact solution $u \in V$.

(Q3.1.4.7.D) [Sum or energies of errors] Let $u_\ell \in V_{0,\ell}$, $\ell = 0, \dots, L$, denote the Galerkin solution of the linear variational problem

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0,$$

with a symmetric positive definite (s.p.d.) bilinear form $a: V_0 \times V_0 \rightarrow \mathbb{R}$ and on a nested sequence of trial/test spaces

$$V_{0,\ell} \subset V_0, \quad V_{0,\ell-1} \subset V_{0,\ell}, \quad \ell \in \{1, \dots, L\}, \quad L \in \mathbb{N}.$$

Show that

$$\|u_L - u_0\|_a^2 = \sum_{\ell=1}^L \|u_\ell - u_{\ell-1}\|_a^2.$$

(Q3.1.4.7.E) [Representation of error norm] Let $u \in V_0$ be the solution and $u_h \in V_{0,h}$, $V_{0,h} \subset V_0$, be a Galerkin solution of the linear variational problem

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0,$$

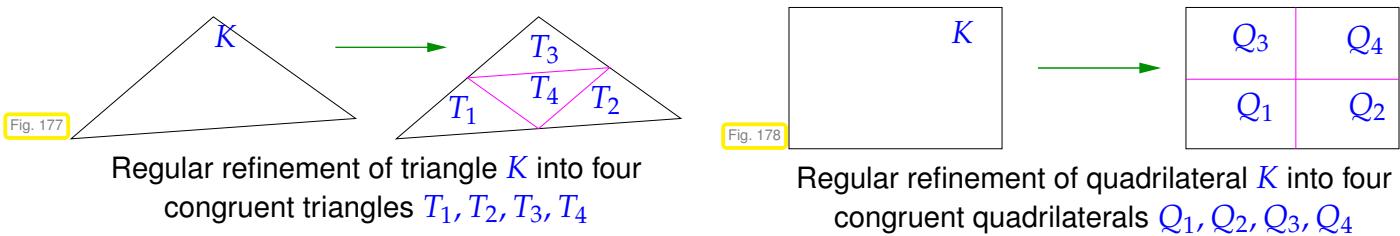
with a symmetric positive definite (s.p.d.) bilinear form $a: V_0 \times V_0 \rightarrow \mathbb{R}$. Show that

$$\|u - u_h\|_a^2 = \|u\|_a^2 - \|u_h\|_a^2.$$

(Q3.1.4.7.F) [Exact finite-element Galerkin solution] Give an example for a 2nd-order linear elliptic boundary value problem for $-\Delta$ on a 2D domain for which finite element discretization by means of $S_1^0(\mathcal{M})$, \mathcal{M} a triangular mesh, will always produce the exact solution.

(Q3.1.4.7.G) [Number of geometric entities generated by refinement] Start from a hybrid mesh with n_c cells, n_e edges, and n_p vertices. Develop a formula that gives the numbers of cells, edges, and vertices of the mesh created by k steps of **regular refinement**.

Hint: Regular refinement of a quadrilateral cell is done by connecting the midpoints of opposite edges.



△

3.2 Empirical (Asymptotic) Convergence of Lagrangian FEM



Video tutorial for Section 3.2: Empirical (Asymptotic) Convergence of FEM: (53 minutes)
[Download link](#), [tablet notes](#)

§3.2.0.1 (Disambiguation: Convergence) Unfortunately, in computational mathematics the term **convergence** is used for different unrelated phenomena:

- (I) Convergence of an iterative method, like Newton's method [Hip19, ??]: we study how fast the iterates $x^{(k)}$, $k \in \mathbb{N}_0$, approach a limit x^* and watch the iteration error norm $\|x^{(k)} - x^*\|$.
- (II) **Convergence of approximations**: we study the dependence of the approximation error on the applied approximations scheme. This notion of convergence was central in [Hip19, ??] and will also be adopted in the analysis of discretization methods for boundary value problems.

□

3.2.1 Asymptotic Convergence

§3.2.1.1 (Convergence of approximations) The reader has probably seen several different situations where convergence of approximations is major issue:

- (i) In the context of **numerical quadrature** for the numerical computation of $\int_a^b f(x) dx$ [Hip19, ??] one is interested in the dependence of the quadrature error on the number N of evaluations of the integrand f for families of quadrature formulas.

- (ii) In the course “Numerical Methods for CSE” [Hip19] a whole chapter was devoted to the approximation of functions in 1D [Hip19, ??]. There, given a function $f : I \subset \mathbb{R} \rightarrow \mathbb{R}$, I a finite interval, we built a “simple” function $\tilde{f} : I \rightarrow \mathbb{R}$ that could be described by a finite number N of degrees of freedom. For instance, \tilde{f} could be a global polynomial encoded through its expansion coefficients with respect to a basis of the space of polynomials [Hip19, ??].

In this context the counterpart of the discretization error is the **approximation error** $f - \tilde{f}$, also a function on I .

The size of $f - \tilde{f}$ was naturally measured by computing a suitable **norm** [Hip19, ??]. The supremum norm, the L^2 -norm, and L^1 -norm were introduced as most important specimens of relevant norms.

- (iii) In numerical integration, where the goal is the approximate solution of initial value problems $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$ [Hip19, ??] examined the dependence of (a norm of) the error at final time on the timestep τ for single-step methods. The generic behavior of the error turned out to be $O(\tau^q)$ for timestep $\tau \rightarrow 0$, and we called $q \in \mathbb{N}$ the order of the method.

]

In all these cases we regard the approximation error as a function of continuous or discrete **discretization parameters** (number of f -evaluations, polynomial degree p timestep τ), and investigate the behavior of the error as a **discretization parameter** (DP) approaches a suitable limit; this is the gist of **asymptotic convergence** analysis.

Convergence: asymptotic perspective

Crucial: our notion of convergence is *asymptotic* !

sequence of discrete models \Rightarrow sequence of approximate solutions $(u_h^{(i)})_{i \in \mathbb{N}}$
 \Rightarrow study sequence $(\|u_h^{(i)} - u\|)_{i \in \mathbb{N}}$ as $i \rightarrow \infty$

↑
created by *variation* of a **discretization parameter**.

§3.2.1.3 (Discretization parameters in the finite element method) Talking about (asymptotic) convergence implies agreement on meaningful discretization parameters and their limits. Necessarily, the discretization parameter must be linked to the **resolution** (“capability to approximate a generic solution”) of the Galerkin trial/test space $V_{0,h}$.

Meaningful discretization parameters for finite element Galerkin methods are suggested by the enhancement of the resolution of finite element spaces through refinement:

- ♦ In the case of ***h-refinement*** the maximal size of cells of the mesh can serve as discretization parameter. The following generalizes the concept of “mesh width” introduced in one dimension in Section 2.3.

Definition 3.2.1.4. Mesh width

Given a mesh $\mathcal{M} = \{K\}$, its **mesh width** $h_{\mathcal{M}}$ is defined as

$$h_{\mathcal{M}} := \max\{\operatorname{diam} K : K \in \mathcal{M}\} , \quad \operatorname{diam} K := \max\{|\mathbf{p} - \mathbf{q}| : \mathbf{p}, \mathbf{q} \in K\} .$$

The natural limit to consider for the meshwidth of a sequence of meshes is $h_{\mathcal{M}} \rightarrow 0$.

- ♦ For ***p-refinement*** the polynomial degree $p \in \mathbb{N}$ is a natural discretization parameter. The pertinent limit for asymptotic convergence is $p \rightarrow \infty$.

A universal discretization parameter for families of Galerkin discretizations is $N := \dim V_{0,h}$, often called the “number of degrees of freedom” or “number of unknowns”. It gives the size of the systems of equations arising from Galerkin discretization and, thus, is often advertised as a good measure for the *cost* of a Galerkin scheme.

However, for linear problem a more appropriate measure for the cost may be $\text{nnz}(\mathbf{A})$, the number of non-zero entries of the Galerkin matrix, which can also be used as a discretization parameter. Note that for fixed-degree Lagrangian finite element spaces $\dim V_{0,h} \sim \text{nnz}(\mathbf{A})$. Hence, both discretization parameters give the same results concerning the asymptotic behavior of norms of the discretization error in the case of h -refinement. \square

3.2.2 Algebraic and Exponential Convergence

In Section 3.1 we pointed out parallels between studying approximation errors and discretization errors. In the case of approximation errors we discovered rather regular behavior when adopting an *asymptotic perspective*, which was introduced, e.g., in [Hip19, ??]. It regards suitable norms of approximation errors as functions of the number N of parameters for families of approximation schemes and examines their decay as $N \rightarrow \infty$. We do the same for Galerkin discretization errors, where N will stand for the dimension of the Galerkin trial/test space.

For many different approximation and discretization methods we observe that the asymptotic behavior of error norms fits a few typical patterns. We recall them writing (in this section) u_N for the approximation/discrete solutions belonging to the parameter value N .

Definition 3.2.2.1. Types of convergence → [Hip19, ??], [Hip19, ??]

$$\begin{aligned} \|u - u_N\| = O(N^{-\alpha}), \alpha > 0 &\iff \text{algebraic convergence with rate } \alpha \\ \|u - u_N\| = O(\exp(-\gamma N^\delta)), \text{ with } \gamma, \delta > 0 &\iff \text{exponential convergence} \\ (\text{asymptotically for } N \rightarrow \infty) \end{aligned}$$

Here we have relied on the Landau “O-notation” according to (0.3.2):

$$f(N) = O(g(N)) \iff \begin{array}{l} \exists N_0 > 0, \exists C > 0 \text{ independent of } N \\ \text{such that } |f(N)| \leq Cg(N) \text{ for } N > N_0. \end{array} \quad (3.2.2.2)$$

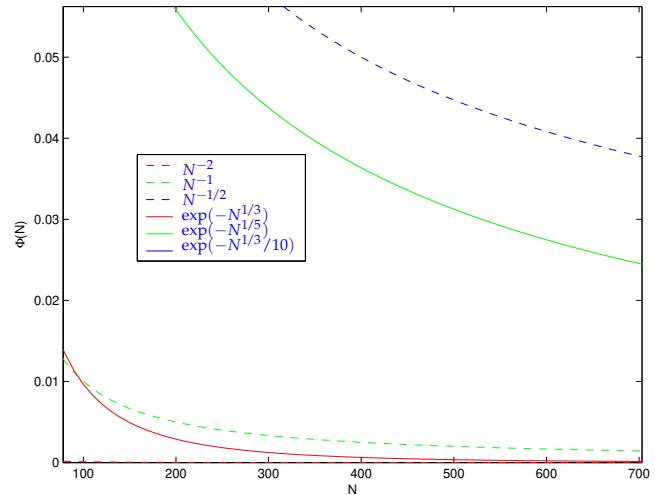
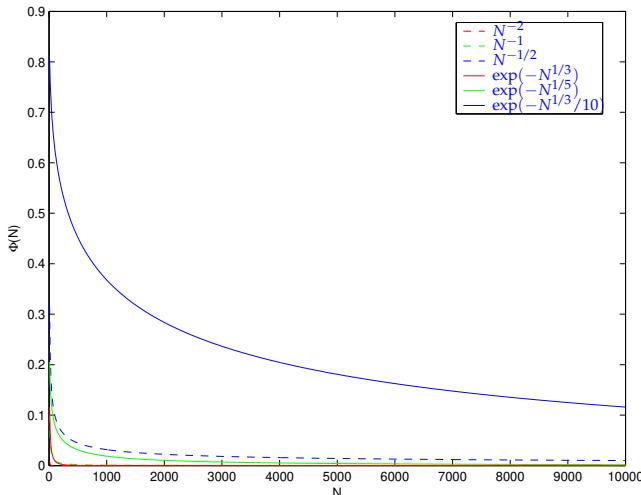
Definition 3.2.2.3. Rate of convergence

In the case of algebraic convergence the exponent α in Def. 3.2.2.1 is called the *rate* of (algebraic) convergence.

Remark 3.2.2.4 (Implicit sharpness of asymptotic convergence) Often, also in this course, the assertion of a particular qualitative and quantitative kind of asymptotic convergence will imply “*sharpness*”:

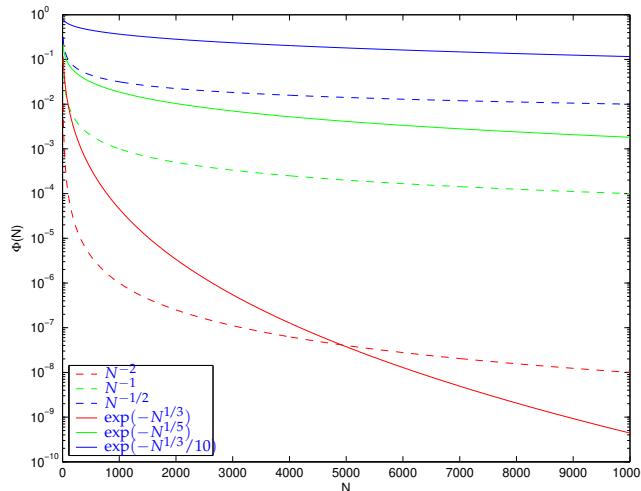
- in the case of algebraic convergence, the claim that $\|u - u_N\|$ converges algebraically with rate $\alpha > 0$, will also mean that
 - there is no $\beta > \alpha$ such that $\|u - u_N\| = O(N^{-\beta})$ for $N \rightarrow \infty$.
- in the case of exponential convergence, saying that $\|u - u_N\|$ converges exponentially like $\exp(-\gamma N^\delta)$ also implies that
 - there are no $\nu > \delta$ and $\mu > \delta$ such that $\|u - u_N\| = O(\exp(-\nu N^\mu))$ for $N \rightarrow \infty$.

The following plots illustrate the qualitative behavior of error norms implied by the two different types of convergence for various parameters.

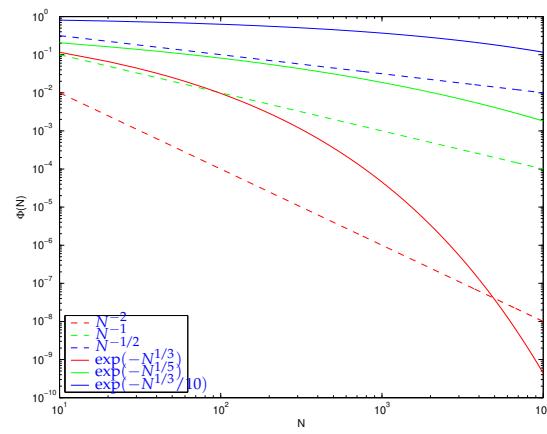


Linear plot of qualitative convergence behavior: algebraic/exponential convergence rates

► Exponential convergence will always win (asymptotically)



Log-linear plot of decrease of discretization error for algebraic/exponential convergence rates



Log-log plot of decrease of discretization error for algebraic/exponential convergence rates

§3.2.2.5 (Exploring convergence empirically) → [Hip19, ??] When (in homework problems) you are asked to “investigate the (asymptotic) convergence of a method” in a numerical experiment, you are expected to make a

qualitative and quantitative statement

about the asymptotic behavior of a suitable norm (*) of the discretization error in the sense of Definition 3.2.2.1:

- ☛ “qualitative”: does the error display algebraic or exponential convergence according to Def. 3.2.2.1, or none of these?
- ☛ “quantitative”: determine the rate α in the case of algebraic convergence, γ, δ in the case of exponential convergence.

(*): the norm of interest and how it is evaluated has to be specified as part of the question!

How to tease qualitative/quantitative information about asymptotic convergence out of raw norms of discretization error?

Given: data tuples (N_i, ϵ_i) , $i = 1, 2, 3, \dots$, $N_i \hat{=} \text{no. of d.o.f.s}$, $\epsilon_i \hat{=} \text{error norms}$

1. Conjecture: algebraic convergence with rate α : $\epsilon_i \approx C N_i^{-\alpha}$

$$\log(\epsilon_i) \approx \log(C) - \alpha \log N_i \quad (\text{affine linear in log-log scale}).$$

- visual evidence: (almost) linear error plot in *doubly logarithmic* scale, slope α ,
- linear regression on data $(\log N_i, \log \epsilon_i)$, $i = 1, 2, 3, \dots$ to determine rate α , see Code 3.2.2.8.

2. Conjecture: exponential convergence: $\epsilon_i \approx C \exp(-\gamma N_i^\delta)$

$$\log \epsilon_i \approx \log(C) - \gamma N_i^\delta. \quad (3.2.2.6)$$

$$\log \epsilon_i - \log \epsilon_{i-1} \approx -\gamma(N_i^\delta - N_{i-1}^\delta), \quad (3.2.2.7)$$

$$\frac{\log \epsilon_{i+1} - \log \epsilon_i}{\log \epsilon_i - \log \epsilon_{i-1}} \approx \frac{N_{i+1}^\delta - N_i^\delta}{N_i^\delta - N_{i-1}^\delta}.$$

Special case: geometric increase/decrease of problem size parameters: $N_i = Q N_{i-1}$ for some known $Q > 0$.

$$\blacktriangleright \frac{\log \epsilon_{i+1} - \log \epsilon_i}{\log \epsilon_i - \log \epsilon_{i-1}} \approx \frac{Q^\delta - 1}{1 - Q^{-\delta}} = \frac{(Q^\delta)^2 - Q^\delta}{Q^\delta - 1}.$$

From this you can determine δ by solving a quadratic equation. Then you get γ from (3.2.2.7) or, as above, by linear regression from (3.2.2.6).

Alternative: non-linear least squares fit (\rightarrow [Hip19, ??]) to determine δ :

$$(c, \gamma, \delta) = \operatorname{argmin} \left\{ \sum_i |\log \epsilon_i - c + \gamma N_i^\delta|^2 \right\},$$

residual \leftrightarrow validity of conjecture. This can be done by a short code, which his left as an exercise.

C++ code 3.2.2.8: Estimating the rate of algebraic convergence

```

1 double eoc(const Eigen::VectorXd &N, const Eigen::VectorXd &err,
2           unsigned fromindex = 0, std::string filename = "conv.eps") {
3 // The argument N has to pass a sorted vector of length L > 1 of
4 // problem size parameter values, whereas the L-vector err
5 // contains the corresponding error norms. The argument
6 // fromindex ∈ {1, …, L − 1} restricts the relevant data to
7 // fromindex, …, L} in order to suppress the impact of
8 // possible pre-asymptotic behavior.
9 // Returns the estimated rate of convergence.
10 const unsigned dim = N.size();
11
12 //truncate preasymptotic behavior if desired:
13 const unsigned int newdim = dim - fromindex;
14 //compute log(N) and log(err) componentwise
15 auto logfun = [] (double d) {return std::log(d);};
16 Eigen::VectorXd Nlog(newdim), errlog(newdim);
17 std::transform(N.data() + fromindex, N.data() + dim, Nlog.data(), logfun);
18 std::transform(err.data() + fromindex, err.data() + dim, errlog.data(), logfun);
19
20 // perform linear regression, aka least squares fitting to a line.
21 linearFit
22 // returns the coefficients of the linear polynomial, the second of

```

```

    which is its slope
22 Eigen::Vector2d polyfit = linearFit(Nlog, errlog);
23 double alpha = -polyfit[1];
24
25 return alpha;
26 }
```

Linear fitting of a data vector (x_i, y_i) , $i = 1, \dots, n$, means to solve the least squares problem

$$(\beta_*, \alpha_*) := \underset{\alpha, \beta \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^n (y_i - \alpha x_i - \beta)^2. \quad (3.2.2.9)$$

The task of finding (α_*, β_*) is called **linear regression**. Numerical methods for solving (3.2.2.9) are covered in [Hip19, ??].

C++11 code 3.2.2.10: Linear regression of data

```

1 // Linear fitting of data passed in vectors x and y of equal length.
2 // Returns the 2-vector  $[\beta_*, \alpha_*]^T$ , cf. (3.2.2.9).
3 Eigen::Vector2d
4     linearFit(const Eigen::VectorXd& x, const Eigen::VectorXd& y) {
5     assert(x.rows() == y.rows());
6     Eigen::Matrix<double, Eigen::Dynamic, 2> X(x.rows(), 2);
7     // Set up matrix of overdetermined system of equations
8     X.col(0) = Eigen::VectorXd::Constant(x.rows(), 1);
9     X.col(1) = x;
10    // Solve least squares problem by QR-decomposition
11    return X.fullPivHouseholderQr().solve(y);
12 }
```

3.2.3 Convergence of FEM: Numerical Experiments

In this section we study the convergence of Galerkin solutions obtained from Lagrangian finite element discretization of linear scalar 2nd-order elliptic variational problems (\rightarrow Section 1.8) **empirically**. This means that we conduct **numerical experiments**, in which we measure norms of the discretization errors. Of course, this can be done only for finite sequences of discrete models. However, if these cover a sufficiently wide range of discretization parameters, they will provide evidence of general laws governing convergence.

§3.2.3.1 (Our model problem) Throughout we consider the Dirichlet problem for Poisson equation on an interval (1D)/polygonal domain (2D) $\Omega \subset \mathbb{R}^d$, $d = 1, 2$:

$$-\Delta u = f \in L^2(\Omega) \quad \text{in } \Omega, \quad u = g \in C^0(\partial\Omega) \quad \text{on } \partial\Omega, \quad (3.2.3.2)$$

To this problem we apply Lagrangian finite element discretization on equidistant partitions ($d = 1$)/triangular meshes ($d = 2$). For details refer to Section 2.3 ($d = 1$) and both Section 2.4 and Section 2.6.1 ($d = 2$). \square

Remark 3.2.3.3 (Approximate computation of error norms) Even if the exact solution u of a boundary value problem is known and a finite element solution u_h has been computed, it will usually be all but impossible to determine the exact value of $\|u - u_h\|$ for all interesting norms like $\|\cdot\|_{L^2(\Omega)}$, $\|\cdot\|_{H^1(\Omega)}$, $\|\cdot\|_{L^\infty(\Omega)}$, etc.

In the case of norms involving an integral, $\|u - u_h\|$ has to be computed by means of numerical quadrature, which will boil down to the cell-wise application of a local quadrature rule (2.7.5.10), as discussed in Section 2.7.5.

Danger: The inevitable **quadrature error** may dominate the discretization error!

Safeguard: (Bolstered by theory) Choose local quadrature of “sufficiently high order”!

Guideline: Choose order $\geq 2p + 1$, when using finite element methods based on local polynomial degree p . In this case and for h -refinement the quadrature error will shrink faster than the finite element discretization error and it will not “pollute” the observed asymptotic behavior of $\|u - u_h\|_{L^2(\Omega)}$, $|u - u_h|_{H^1(\Omega)}$.

All examples in this section rely on “overkill quadrature”: the order of the local quadrature rule is much higher than even demanded by the above guideline. Hence, the impact of quadrature errors can be ignored. \square

EXAMPLE 3.2.3.4 (Computation of norms of finite element discretization errors in LEHRFEM++)

Let us assume that the exact solution $u \in C^0(\bar{\Omega})$ of a second-order elliptic boundary value problem is available in procedural form through a factor object or even a lambda-function, if one has an analytic expression u as in the “method of manufactured solutions”.

Then the $L^2(\Omega)$ -norm of the difference of a finite element function $u_h \in S_p^0(\mathcal{M})$ available through its vector \vec{u} of expansion coefficients with respect to global shape functions, and of u can be computed approximately as follows:

```
const lf::uscalfe::MeshFunctionGlobal mf_u{u_functor};
const lf::uscalfe::MeshFunctionFE mf_sol(fe_space_p, mu_vec);
double L2err = std::sqrt(
    lf::uscalfe::IntegrateMeshFunction(mesh,
        lf::mesh::utils::squaredNorm(mf_sol - mf_u), quad_degree));
```

Here `fe_space_p` is a pointer to an `lf::uscalfe::UniformScalarFESpace` object, `u_functor` a functor object providing `u` and `mu_vec` the coefficient vector, compatible with `Eigen::VectorXd`. The argument `quad_degree` tells the function the degree of exactness of the quadrature rule to be used.

A full example for the computation of discretization errors in LEHRFEM++ based on manufactured solutions is given in `ellbvp_linfede_demo.cc` → [GITHUB](#). Some snippets from that code are listed next:

C++11 code 3.2.3.5: Computation of error norms → [GITHUB](#)

```
2 // Exact solution u
3 auto u = [](Eigen::Vector2d x) -> double {
4     return std::log(x[0] * x[0] + x[1] + 1.0);
5 };
6 // Has to be wrapped into a mesh function for error computation
7 lf::mesh::utils::MeshFunctionGlobal mf_u{u};
8
9 // Gradient of exact solution
10 auto grad_u = [](Eigen::Vector2d x) -> Eigen::Vector2d {
11     double den = x[0] * x[0] + x[1] + 1.0;
12     return ((Eigen::Vector2d() << 2.0 * x[0], 1.0).finished()) / den;
13 };
14 // Convert into mesh function to use for error computation
15 lf::mesh::utils::MeshFunctionGlobal mf_grad_u{grad_u};

2 // Assembly completed: Convert COO matrix A into CRS format using
   Eigen's
```

```

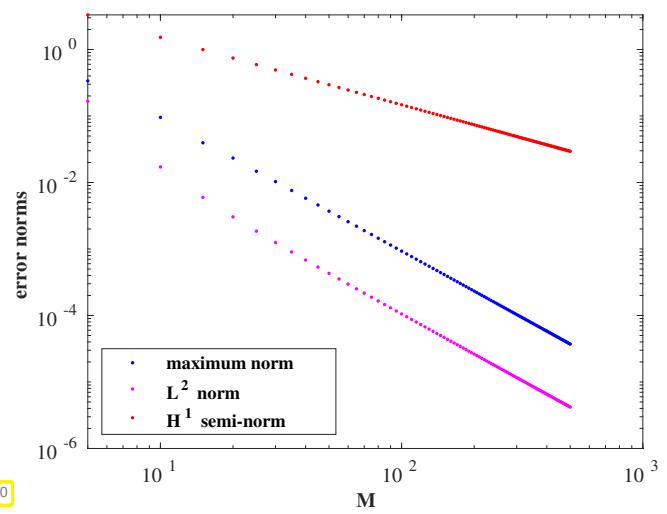
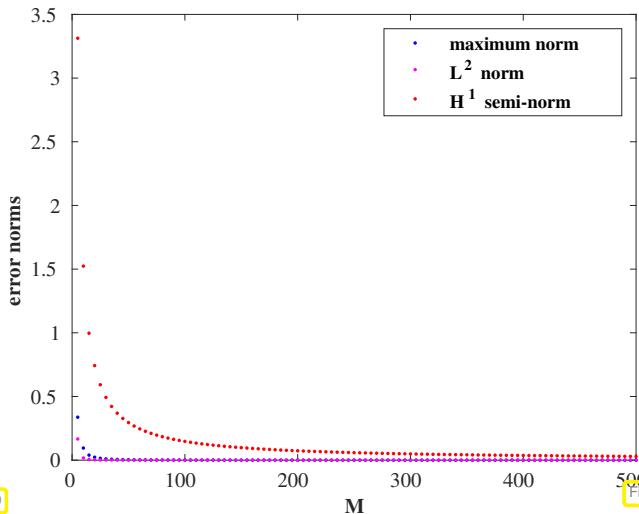
3 // internal conversion routines.
4 Eigen::SparseMatrix<double> A_crs = A.makeSparse();
5
6 // Solve linear system using Eigen's sparse direct elimination
7 // Examine return status of solver in case the matrix is singular
8 Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
9 solver.compute(A_crs);
10 LF_VERIFY_MSG(solver.info() == Eigen::Success, "LU decomposition failed");
11 Eigen::VectorXd sol_vec = solver.solve(phi);
12 LF_VERIFY_MSG(solver.info() == Eigen::Success, "Solving LSE failed");
13
14 // Postprocessing: Compute error norms
15 // create mesh functions representing solution / gradient of
16 // solution
16 const If::fe::MeshFunctionFE mf_sol(fe_space, sol_vec);
17 const If::fe::MeshFunctionGradFE mf_grad_sol(fe_space, sol_vec);
18 // compute errors with 3rd order quadrature rules, which is
19 // sufficient for
20 // piecewise linear finite elements
21 double L2err = // NOLINT
22     std::sqrt(If::fe::IntegrateMeshFunction(
23         mesh, If::mesh::utils::squaredNorm(mf_sol - mf_u), 2));
24 double H1err = std::sqrt(If::fe::IntegrateMeshFunction( // NOLINT
25     mesh, If::mesh::utils::squaredNorm(mf_grad_sol - mf_grad_u), 2));

```

EXPERIMENT 3.2.3.6 (Convergence of linear finite element method for two-point BVP) We use piecewise linear Lagrangian finite elements on equidistant meshes with M cells ($\rightarrow \S\ 2.3.1.3$) to tackle the following two-point boundary value problem:

- ◆ domain $\Omega =]0, 1[$, ▶ unique solution
- ◆ ODE: $-\frac{d^2u}{dx^2} = f$ in Ω , $u(x) = \sin(2\pi x^2)$. $0 < x < 1$.
- ◆ load $f(x) = -4\pi(\cos(2\pi x^2) - 4\pi x^2 \sin(2\pi x^2))$,
- ◆ boundary values $u_a = u_b = 0$. ("manufactured solution")

Computation of integral error norms $\|u - u_h\|_{L^2(\Omega)}$ and $|u - u_h|_{H^1(\Omega)}$ by local (fourth order) 2-point Gauss quadrature rule. Maximum norm $\|u - u_h\|_{L^\infty(\Omega)}$ by sampling in nodes of the mesh.



We observe conspicuous *algebraic convergence* (error curves are almost straight lines in doubly logarithmic plot) with the following rates:

- $L^\infty(\Omega)$ -/ $L^2(\Omega)$ -norm: $\|u - u_h\|_* = O(M^{-2})$, rate 2,
- $H^1(\Omega)$ -semi-norm: $|u - u_h|_{H^1(\Omega)} = O(M^{-1})$, rate 1.

□

EXPERIMENT 3.2.3.7 (Convergence for linear and quadratic Lagrangian finite elements in energy norm)

Setting: $\Omega = [0, 1]^2$, $f(x_1, x_2) = 2\pi^2 \sin(\pi x_1) \sin(\pi x_2)$, $x \in \Omega$, $g = 0$

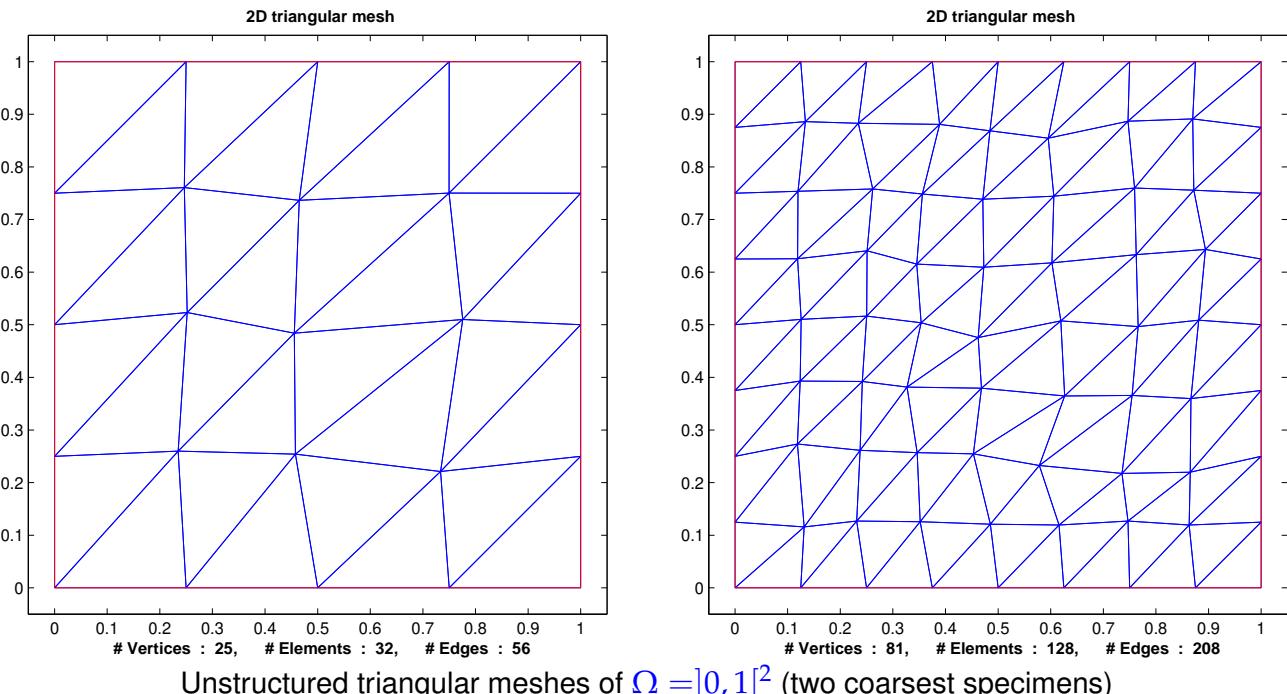
➤ Smooth solution $u(x, y) = \sin(\pi x) \sin(\pi y)$.

- Galerkin finite element discretization based on triangular meshes and
 - linear Lagrangian finite elements, $V_{0,N} = \mathcal{S}_{1,0}^0(\mathcal{M}) \subset H_0^1(\Omega)$ (\rightarrow Section 2.4),
 - quadratic Lagrangian finite elements, $V_{0,N} = \mathcal{S}_{2,0}^0(\mathcal{M}) \subset H_0^1(\Omega)$ (\rightarrow Ex. 2.6.1.2),
- quadrature rule (2.7.5.37) for assembly of local load vectors (\rightarrow Section 2.7.5),

Monitored: $H^1(\Omega)$ -semi-norm (\rightarrow Def. 1.3.4.3) of the Galerkin discretization error $u - u_h$

➤ Approximate (*) computation of $|u - u_h|_{H^1(\Omega)}$ on a sequence of meshes (created by successive regular refinement (\rightarrow Ex. 3.1.4.3) of coarse initial mesh)

(*): use of local quadrature rule (2.7.5.37) (on current FE mesh)



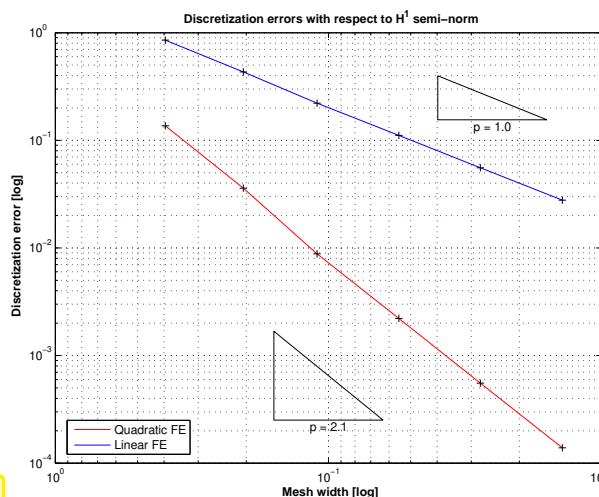


Fig. 181

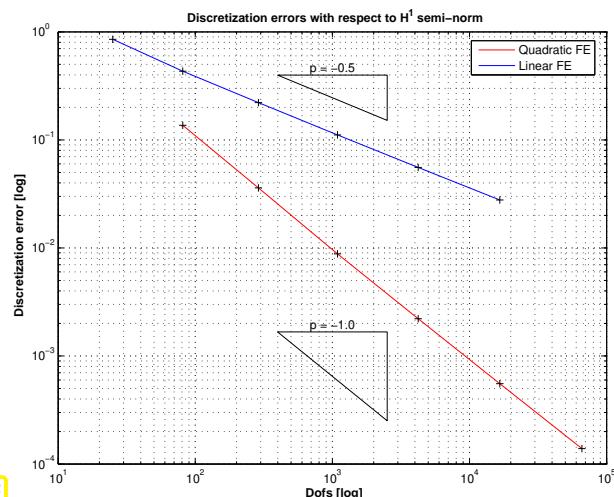


Fig. 182

$H^1(\Omega)$ -semi-norm of discretization error on unit square ($\text{—} \leftrightarrow p = 1$, $\text{—} \leftrightarrow p = 2$)

Again, recall the two types of convergence (algebraic convergence vs. exponential convergence) from Def. 3.2.2.1 and how to detect them in a numerical experiment by inspecting appropriate graphs, see § 3.2.2.5.

- Observations:
- Algebraic rates of convergence in terms of N and h
 - Quadratic Lagrangian FE converge with double the rate of linear Lagrangian FE

Recall: Rates of algebraic convergence can be estimated by linear least squares fitting → Code 3.2.2.8. In Fig. 181 and Fig. 182 these estimated rates are indicated by the slopes of hypotenuses of triangles (German “Steigungsdreieck”). We find

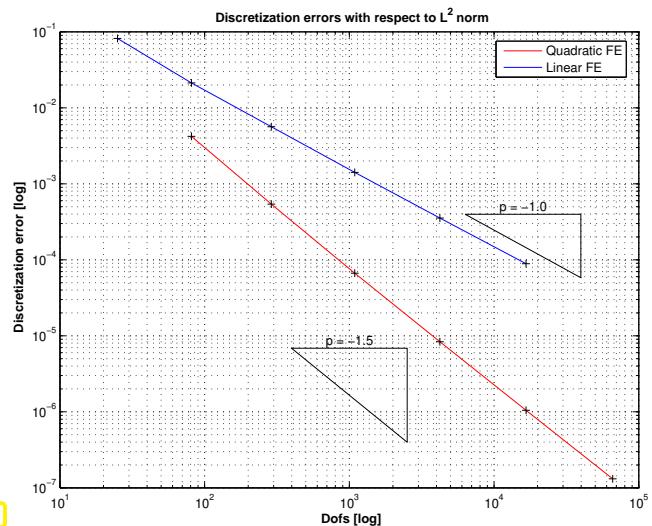
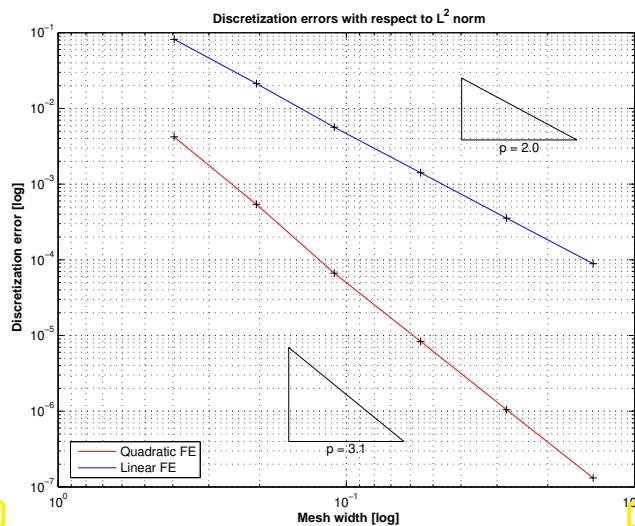
linear Lagrangian finite elements:	$ u - u_h _{H^1(\Omega)} = O(h_M) = O(N^{-\frac{1}{2}})$
quadratic Lagrangian finite elements:	$ u - u_h _{H^1(\Omega)} = O(h_M^2) = O(N^{-1})$

]

EXPERIMENT 3.2.3.8 (Convergence of linear and quadratic Lagrangian finite elements in L^2 -norm)

Setting as above in Exp. 3.2.3.7, $\Omega =]0, 1[^2$.

Monitored: asymptotics of the $L^2(\Omega)$ -semi-norm of the Galerkin discretization error (approximate computation of $\|u - u_h\|_{L^2(\Omega)}$ by means of local quadrature rule (2.7.5.37) on a sequence of meshes created by successive regular refinement (→ Ex. 3.1.4.3) of coarse initial mesh).



$L^2(\Omega)$ -norm of discretization error on unit square ($\text{---} \leftrightarrow p = 1$, $\text{—} \leftrightarrow p = 2$)

- Observations:
- Linear Lagrangian FE ($p = 1$) $\Rightarrow \|u - u_h\|_0 = O(h_M^2) = O(N^{-1})$
 - Quadratic Lagrangian FE ($p = 2$) $\Rightarrow \|u - u_h\|_0 = O(h_M^3) = O(N^{-1.5})$

For the “conversion” of convergence rates with respect to the mesh width h_M and $N := \dim \mathcal{S}_p^0(\mathcal{M})$, note that in 2D for Lagrangian finite element spaces with fixed polynomial degree (\rightarrow Section 2.6) and meshes created by global (that is, carried out everywhere) regular refinement

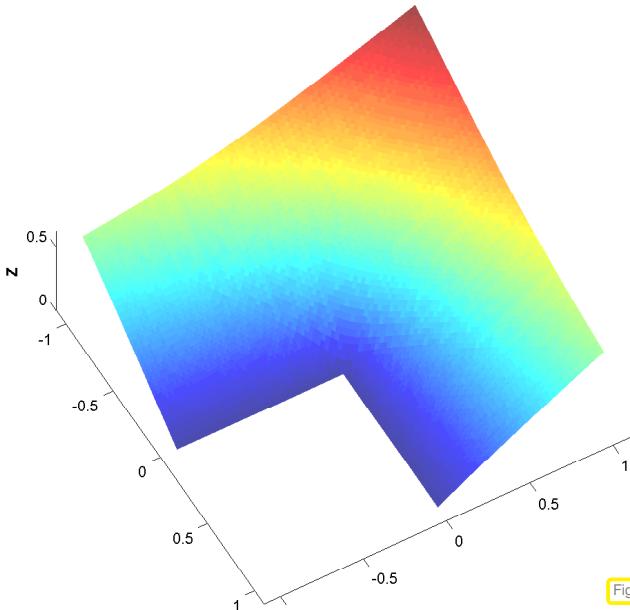
$$N = O(h_M^{-2}). \quad (3.2.3.9)$$

See Section 3.3.5, page 345 for further discussion, (3.3.5.16) for a more general relationship.

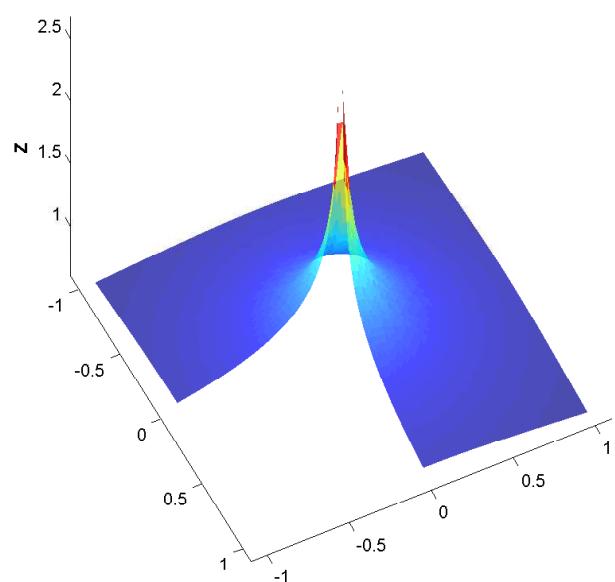
EXPERIMENT 3.2.3.10 (h -convergence of Lagrangian FEM on L-shaped domain)

Setting: model problem (3.2.3.2) on $\Omega = [-1, 1]^2 \setminus ([0, 1] \times [-1, 0])$, exact solution (in polar coordinates, see (1.2.3.47))

$$u(r, \varphi) = r^{2/3} \sin(2/3\varphi) \quad \Rightarrow \quad f = 0, g = u_{|\partial\Omega}.$$



Exact solution u

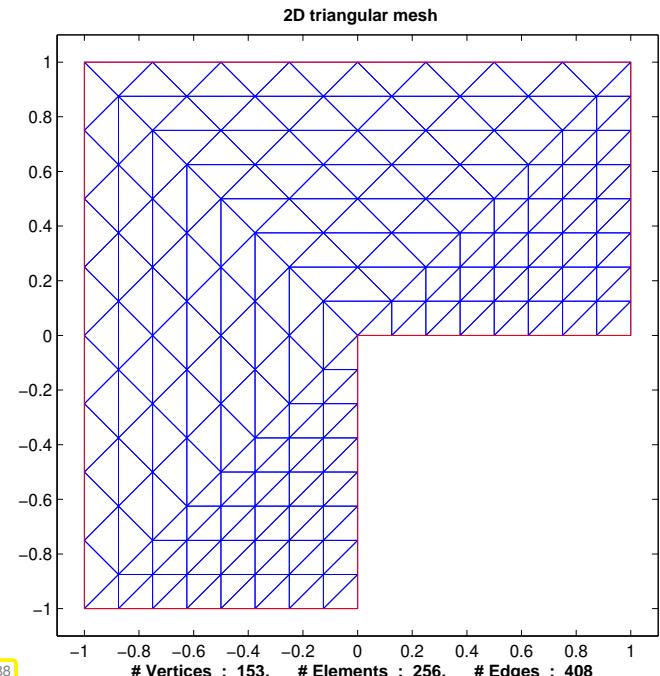
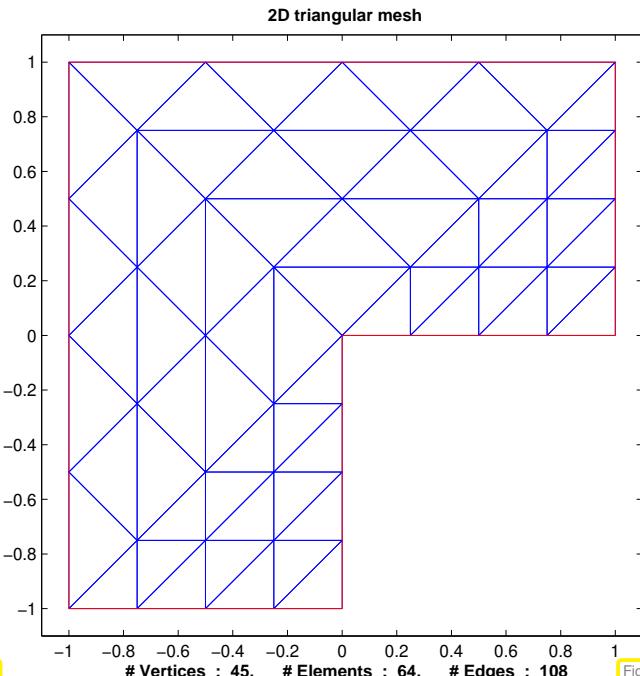


Norm of gradient: $\|\mathbf{grad} u\|$

Note: $\mathbf{grad} u$ has a so-called **singularity** at 0 , that is, “ $\|\mathbf{grad} u(0)\| = \infty$ ”.

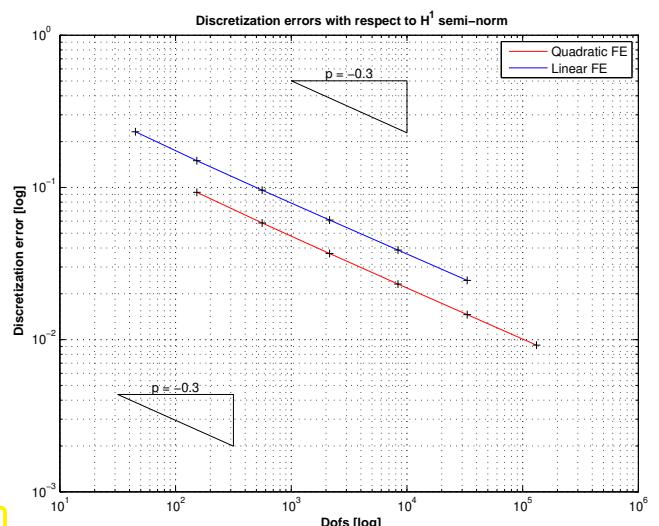
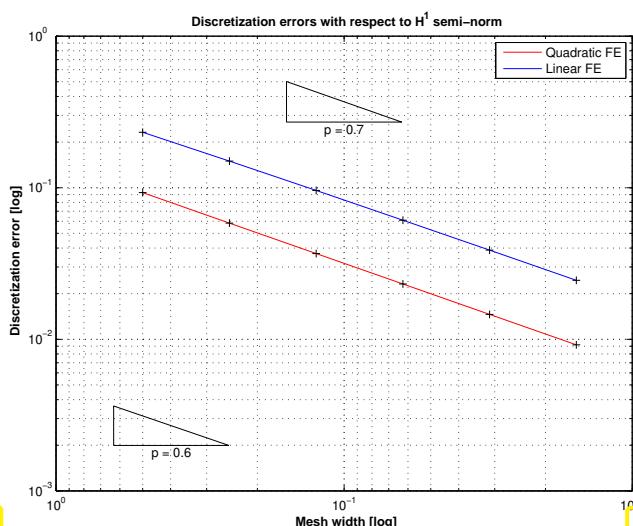
- Galerkin finite element discretization based on triangular meshes and
 - linear Lagrangian finite elements, $V_{0,N} = \mathcal{S}_{1,0}^0(\mathcal{M}) \subset H_0^1(\Omega)$ (\rightarrow Section 2.4),
 - quadratic Lagrangian finite elements, $V_{0,N} = \mathcal{S}_{2,0}^0(\mathcal{M}) \subset H_0^1(\Omega)$ (\rightarrow Ex. 2.6.1.2),
- linear/quadratic interpolation of Dirichlet data to obtain offset function $u_0 \in \mathcal{S}_{p,0}^0(\mathcal{M})$, $p = 1, 2$, see Section 2.7.6, Ex. 2.7.6.5.

Sequence of meshes created by successive regular refinement (\rightarrow Ex. 3.1.4.3) of coarse initial mesh, see Fig. 187 and Fig. 188.



Unstructured triangular meshes of $\Omega =]-1, 1[^2 \setminus]0, 1[\times]-1, 0[$ (two coarsest specimens)

Approximate computation of $\|u - u_h\|_{H^1(\Omega)}$ by using local quadrature formula (2.7.5.37) on FE meshes.



$H^1(\Omega)$ -semi-norm of discretization error on "L-shaped" domain ($- \leftrightarrow p = 1$, $- \leftrightarrow p = 2$)

- Observations:
- For both $p = 1, 2$: $\|u - u_h\|_1 = O(N^{-1/3})$
 - No gain from higher polynomial degree

Conjecture: singularity of $\mathbf{grad} u$ at $\mathbf{x} = \mathbf{0}$ seems to foil faster algebraic convergence of quadratic Lagrangian finite element solutions! □

EXPERIMENT 3.2.3.11 (Convergence of Lagrangian FEM for p -refinement)

- ◆ Model BVP as in Exp. 3.2.3.7 ➤ unit square domain $\Omega = [0, 1]^2$,
- Exp. 3.2.3.10 ➤ L-shaped domain $\Omega = [-1, 1]^2 \setminus ([0, 1] \times [-1, 0])$.
- ◆ Galerkin finite element discretization based on $\mathcal{S}_p^0(\mathcal{M})$, $p = 1, 2, 3, 5, 6, 7, 8, 9, 10$, built on a *fixed* coarse triangular mesh of Ω .



p -refinement

Monitored: $H^1(\Omega)$ -semi-norm (energy norm) and $L^2(\Omega)$ -norm of discretization error as functions of polynomial degree p and $N := \dim \mathcal{S}_p^0(\mathcal{M})$.

(Computation of norms by means of local quadrature rule of order 19!. This renders the error in norm computations introduced by numerical quadrature negligible.)

Meaningful discretization parameters for asymptotic study of error norms:

- ◆ polynomial degree p for Lagrangian finite element space,
- ◆ $N := \dim V_{0,N}$ as a measure of the “cost” of a discretization, see Section 3.2.2.

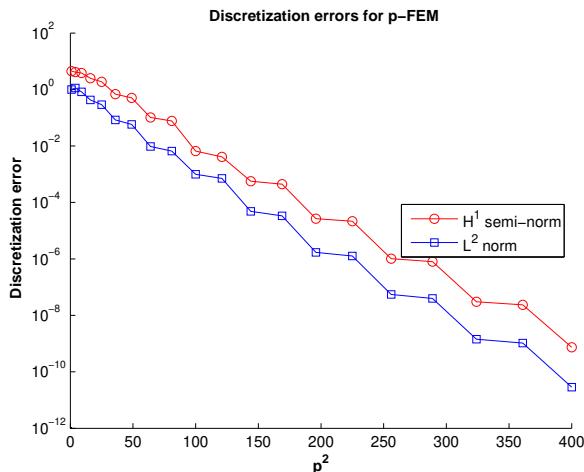


Fig. 191

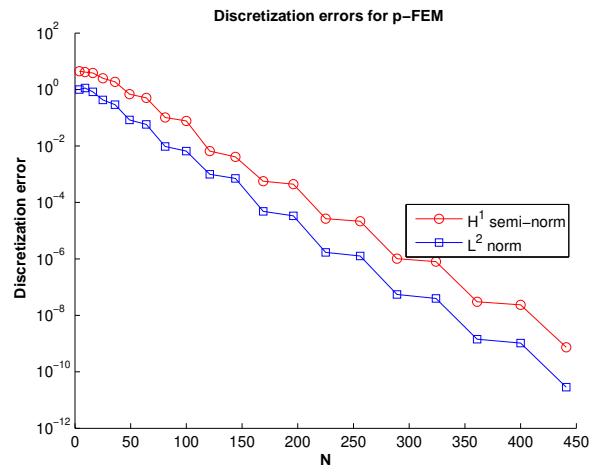


Fig. 192

$\Omega = [0, 1]^2$: behavior of $|u - u_h|_{H^1(\Omega)}$ for different polynomial degrees.
Lagrangian FEM: p -convergence for smooth (analytic) solution

Observation: We witness **exponential convergence** of the FE discretization error!

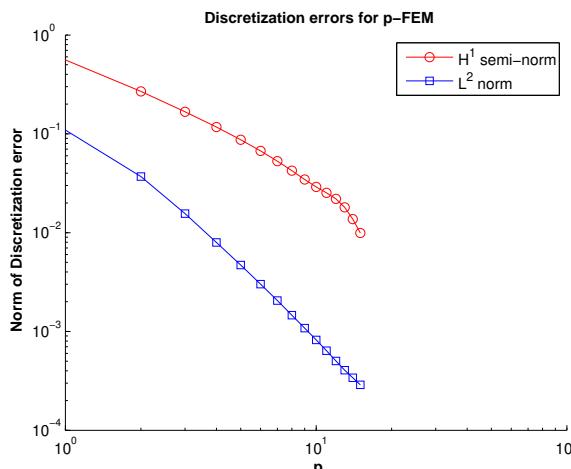


Fig. 193

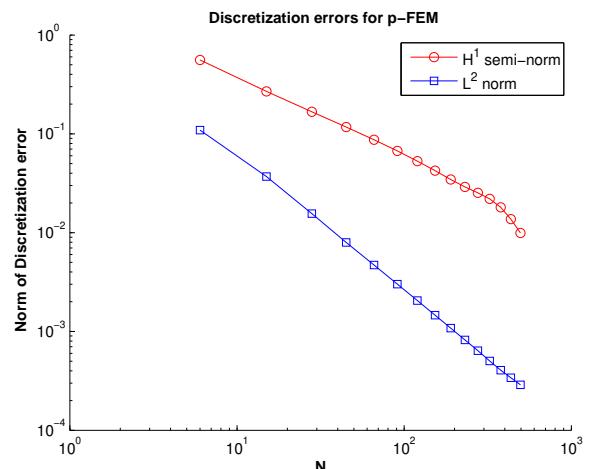


Fig. 194

Lagrangian FEM: p -convergence for solution with singular gradient (L-shaped domain)

Observation: Now “only” **algebraic convergence** of the FE discretization error!

We suspect that the “singular behavior” of $\mathbf{grad} \mathbf{u}$ at $x = 0$ thwarts exponential convergence and allows only algebraic convergence whose rate will be limited by the (lack of) smoothness of the solution. □

EXPERIMENT 3.2.3.12 (Asymptotic nature of convergence) We consider the Galerkin finite element discretization of the two-point boundary value problem

$$-\frac{d^2u}{dx^2} = g(x), \quad u(0) = u(1) = 0, \quad \Omega =]0, 1[, \quad \Leftrightarrow \quad u(x) = \sin(50\pi x^2),$$

by means of piecewise linear Lagrangian finite element on an equidistant meshes with $M \in \mathbb{N}$ cells. We perform the same evaluations as in Exp. 3.2.3.6.

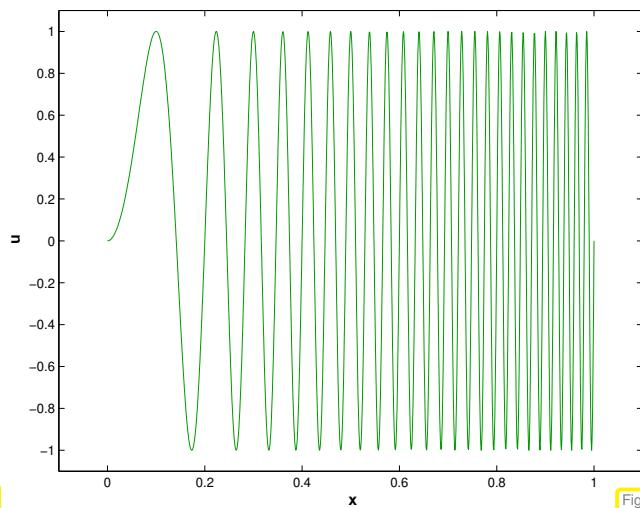


Fig. 195

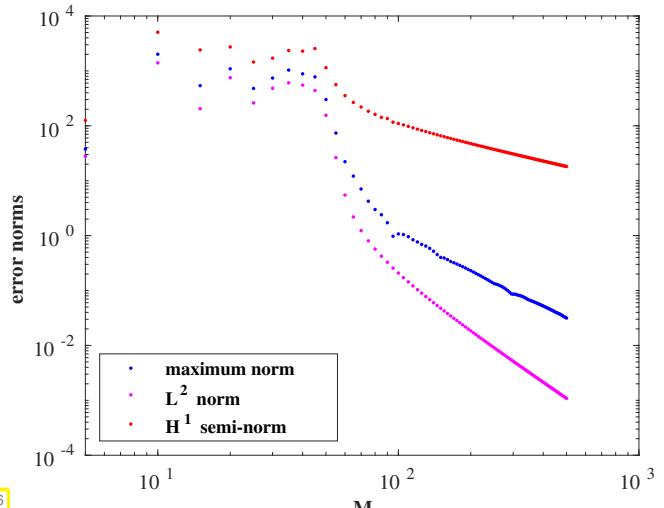


Fig. 196

For large M we still observe asymptotic algebraic convergence with roughly the same rates as in Exp. 3.2.3.6. This is the “typical” **asymptotic** behavior of the discretization error norms for lowest-order Lagrangian finite element methods in one dimension.



However, the onset of asymptotic convergence occurs only for rather small meshwidth, respectively, beyond thresholds that may never be reached in a computation. During a long pre-asymptotic phase the error is hardly reduced when increasing the resolution of the discretization. □

§3.2.3.13 (Summary of observations) Observations on convergence of Galerkin finite element solutions of 2nd-order elliptic BVPs obtained by means of Lagrangian finite elements:

- ◆ For h -refinement we generally observe **algebraic convergence** of $H^1(\Omega)$ -/ $L^2(\Omega)$ -norms of the discretization errors in meshwidth/problem size.
- ◆ The rate of convergence seems to depend on
 - the kind of error norm considered,
 - properties of the exact solution u of the boundary value problem,
 - the (uniform) polynomial degree of the Lagrangian finite element space.
- ◆ In general $\|u - u_h\|_{L^2(\Omega)}$ seems to converge faster than $|u - u_h|_{H^1(\Omega)}$.
- ◆ Asymptotic convergence behavior may not emerge for “practical” meshwidths.

The following sections will be devoted to providing some mathematical underpinning for these observations, which will yield deeper insights into the asymptotic behavior of finite element discretization errors. □

Review question(s) 3.2.3.14 (Empirical Convergence of FEM)

(Q3.2.3.14.A) [Forgetting to take the square root] You forget the call to `std::sqrt()` when computing approximations of $\|u - u_h\|_{L^2(\Omega)}$ and $|u - u_h|_{H^1(\Omega)}$, $u - u_h$ a finite-element Galerkin discretization error for a second-order elliptic BVP. What will be the impact on observed rates of algebraic convergence?

(Q3.2.3.14.B) [Estimates for boundary traces] In a numerical experiment we observe the following asymptotic convergence for the finite-element solutions of a second-order elliptic BVP on $\Omega \subset \mathbb{R}^2$ and h -refinement ($h_M \rightarrow 0$)

$$|u - u_h|_{H^1(\Omega)} = O(h_M) , \quad \|u - u_h\|_{L^2(\Omega)} = O(h_M^2) .$$

What asymptotic convergence for $h_M \rightarrow 0$ do you predict for the error norm $\|u - u_h\|_{L^2(\partial\Omega)}$?

Hint. You can refer the the following result

Theorem 1.9.0.10. Multiplicative trace inequality

$$\exists C = C(\Omega) > 0: \|u\|_{L^2(\partial\Omega)}^2 \leq C \|u\|_{L^2(\Omega)} \cdot \|u\|_{H^1(\Omega)} \quad \forall u \in H^1(\Omega) .$$

(Q3.2.3.14.C) [Determining convergence rates from data] The following table list error norms recorded for a sequence of finite element solutions $u_\ell \in H^1(\Omega)$ belonging to finite element spaces with dimensions N_ℓ . From the data predict qualitatively and quantitatively the asymptotic convergence for $N_\ell \rightarrow \infty$.

N_ℓ	8	16	32	64	128	256	512	1024
$ u - u_\ell _{H^1(\Omega)}$	4.98e-01	4.08e-01	3.23e-01	2.57e-01	2.01e-01	1.53e-01	1.26e-01	1.02e-01
$\ u - u_\ell\ _{L^2(\Omega)}$	3.57e-01	2.55e-01	1.79e-01	1.24e-01	8.89e-02	6.15e-02	4.49e-02	3.05e-02

(Q3.2.3.14.D) [Impact of singularities] We solve a second-order elliptic BVP based on the Lagrangian finite-element spaces $S_1^0(\mathcal{M}_\ell)$ and $S_2^0(\mathcal{M}_\ell)$, and on a sequence of triangular meshes $(\mathcal{M}_\ell)_{\ell=0}^L$ obtained by uniform regular refinement. We get the finite element Galerkin solutions $u_\ell^1 \in S_1^0(\mathcal{M}_\ell)$ and $u_\ell^2 \in S_2^0(\mathcal{M}_\ell)$, $\ell = 0, \dots, L$.

How will the presence of a singularity of $\mathbf{grad} u$, $u \in H^1(\Omega)$ the exact solution, manifest itself in the asymptotic behavior of the $L^2(\Omega)$ - and $H^1(\Omega)$ -norms of the finite element discretization errors?

(Q3.2.3.14.E) [Exponential versus algebraic convergence] How should you read the following statement?

“Exponentially convergent Galerkin schemes are better than algebraically convergent methods”

△

3.3 A Priori (Asymptotic) Finite Element Error Estimates

1.  Video tutorial for Section 3.3: A Priori (Asymptotic) Finite Element Error Estimates (I): (21 minutes) [Download link](#), [tablet notes](#)
2.  Video tutorial for Section 3.3: A Priori (Asymptotic) Finite Element Error Estimates (II): (35 minutes) [Download link](#), [tablet notes](#)
3.  Video tutorial for Section 3.3: A Priori (Asymptotic) Finite Element Error Estimates (III): (35 minutes) [Download link](#), [tablet notes](#)

§3.3.0.1 (A priori versus a posteriori error estimates) We are interested in **a priori estimates** of norms of the discretization error $\mathbf{u} - \mathbf{u}_h$, where \mathbf{u} is the exact solution of a linear 2nd-order elliptic boundary value problem and \mathbf{u}_h its finite element Galerkin approximation.

A priori estimate: bounds for error norms available **before** computing approximate solutions.



A posteriori estimate: bounds for error norms based on an approximate solution **already computed**.

We repeat our assumptions: The variational formulation of the elliptic boundary value problem leads to a linear variational problem (2.2.0.2) with symmetric and positive definite bilinear form \mathbf{a} (\rightarrow Ass. 3.1.1.2) and \mathbf{a} -continuous right hand side functional (\rightarrow Ass. 3.1.1.3).

§3.3.0.2 (General policy for obtaining a-priori error estimates in energy norm) The results of Section 3.1 provide us with the tools to address a-priori error estimates for Galerkin discretization error:

Optimality (3.1.3.9) of Galerkin solution \blacktriangleright a priori error estimates

Thm. 3.1.3.7
(Cea's lemma)



Estimate energy norm of Galerkin discretization error $\mathbf{u} - \mathbf{u}_h$ by bounding the best approximation error for exact solution \mathbf{u} in the finite element space:

$$\| \mathbf{u} - \mathbf{u}_h \|_{\mathbf{a}} \leq \underbrace{\inf_{v_h \in V_{0,h}} \| \mathbf{u} - v_h \|_{\mathbf{a}}}_{\text{(norm of) discretization error}} , \quad \underbrace{\uparrow}_{\text{best approximation error}} \quad (3.1.3.9)$$

How to estimate **best approximation error** $\inf_{v_h \in V_{0,h}} \| \mathbf{u} - v_h \|_V$?

\blacktriangleright Well, given solution \mathbf{u} seek a **candidate function** $w_h \in V_{0,h}$ with

$$\| \mathbf{u} - w_h \|_V \approx \inf_{v_h \in V_h} \| \mathbf{u} - v_h \|_V .$$

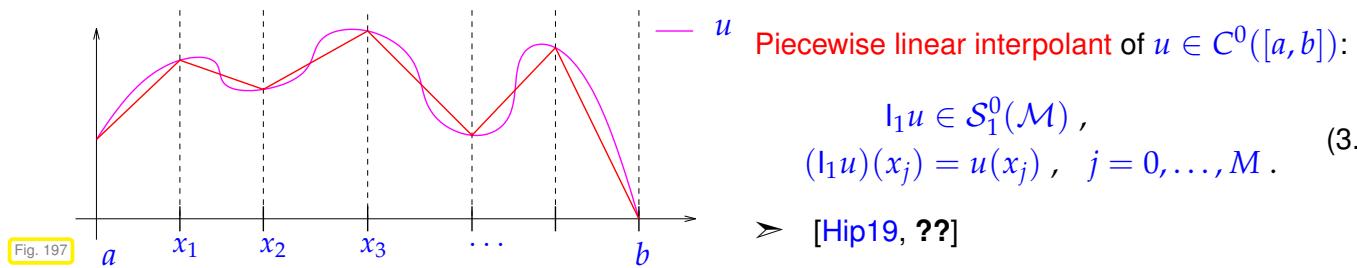
Natural choice: w_h by interpolation/averaging of (*unknown, but existing*) \mathbf{u}

Thus, the task of bounding the Galerkin discretization error can be reduced to **interpolation error estimates**.

3.3.1 Estimates for Linear Interpolation in 1D

In this section we first study interpolation error estimates in one spatial dimension, in order to elucidate the general approach and the structure of the estimates.

The 1D computational domain (\rightarrow Section 1.5.1) is just an interval $\Omega = [a, b]$. It is equipped with a 1D mesh (\rightarrow Section 2.3): $\mathcal{M} := \{x_{j-1}, x_j : j = 1, \dots, M\}$, $M \in \mathbb{N}$, on which we define **piecewise linear** interpolation:



Goal: Bound suitable norm of the **interpolation error** $u - l_1 u$ in terms of geometric quantities (*) characterizing $\mathcal{M} \doteq$ discretization parameters \rightarrow § 3.2.1.3.

(*): A typical such quantity is the **mesh width** $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}|$, cf. Def. 3.2.1.4.

Now we investigate **different norms** of the interpolation error. Beforehand, recall the various norms on spaces of (bounded/integrable) functions, the supremum norm $\|\cdot\|_{L^\infty([a,b])}$ (\rightarrow Def. 0.3.2.25), the L^2 -norm $\|\cdot\|_{L^2([a,b])}$ (\rightarrow Def. 0.3.2.27), and the H^1 -(semi)norm $\|\cdot\|_{H^1([a,b])}$ (\rightarrow Def. 1.3.4.3).

§3.3.1.2 (Estimating the error norm $\|u - l_1 u\|_{L^\infty([a,b])}$) We rely on results from basic numerical analysis, [Hip19, ??] and [Hip19, ??]. In particular, we can appeal to the following elementary result about polynomial interpolation in 1D.

Theorem [Hip19, ??]. Error representation for Lagrangian polynomial interpolation

We consider $f \in C^{n+1}(I)$ and the Lagrangian polynomial interpolation operator l_T for a node set $T := \{t_0, \dots, t_n\} \subset I \rightarrow$ [Hip19, ??].

Then, for every $t \in I$ there exists a $\tau_t \in]\min\{t, t_0, \dots, t_n\}, \max\{t, t_0, \dots, t_n\}[$ such that

$$f(t) - l_T(f)(t) = \frac{f^{(n+1)}(\tau_t)}{(n+1)!} \cdot \prod_{j=0}^n (t - t_j) .$$

From this theorem for $n = 1$ we learn that for $u \in C^2([a, b])$ we have

$$\forall x \in [x_{j-1}, x_j]: \quad u(x) - (l_1 u)(x) = \frac{1}{4} u''(\xi_x)(x_j - x_{j-1})^2, \quad \text{for some } \xi_x \in]x_{j-1}, x_j[, \quad (3.3.1.3)$$

$$\text{with local linear interpolant} \quad (l_1 u)(x) = \frac{x - x_{j-1}}{x_j - x_{j-1}} u(x_j) + \frac{x_j - x}{x_j - x_{j-1}} u(x_{j-1}) . \quad (3.3.1.4)$$

(3.3.1.3) implies the following interpolation error estimate in $L^\infty([a, b])$

$$\|u - l_1 u\|_{L^\infty([a,b])} \leq \frac{1}{4} h_{\mathcal{M}}^2 \|u''\|_{L^\infty([a,b])} . \quad (3.3.1.5)$$

This is obtained by simply taking the maximum over all *local* norms of the interpolation error.

However, we should actually target the energy norm. Hence, we also have to study other norms of the interpolation error:

§3.3.1.6 (Estimating the error norm $\|u - l_1 u\|_{L^2([a,b])}$) We start from the observation that all mesh cells contribute to this error norm: with $l_1 u$ from (3.3.1.4)

$$\|u - l_1 u\|_{L^2([a,b])}^2 = \sum_{j=1}^M \|u - l_1 u\|_{L^2([x_{j-1}, x_j])}^2 = \sum_{j=1}^M \int_{x_{j-1}}^{x_j} |(u - l_1 u)(x)|^2 dx . \quad (3.3.1.7)$$



This suggests **localization**: Estimate error norms on individual mesh cells and sum local bounds.

This idea is very natural for piecewise linear interpolation, because it is local in the sense that $\|_1 u$ on a cell K depends only on the values of u on \bar{K} !

The local estimates rely on integration by parts in one dimension:

$$\int_0^1 u(\xi) v'(\xi) d\xi = - \int_0^1 u'(\xi) v(\xi) d\xi + \underbrace{(u(1)v(1) - u(0)v(0))}_{\text{boundary terms}} \quad \forall u, v \in C_{\text{pw}}^1([0, 1]). \quad (1.5.1.8)$$

Apply this formula *twice*, for $u \in C^2([x_{j-1}, x_j])$, $x \in [x_{j-1}, x_j]$, thus removing derivatives from u :

$$\begin{aligned} & \int_{x_{j-1}}^x \frac{(x_j - x)(\xi - x_{j-1})}{x_j - x_{j-1}} u''(\xi) d\xi + \int_x^{x_j} \frac{(x - x_{j-1})(x_j - \xi)}{x_j - x_{j-1}} u''(\xi) d\xi \\ &= - \int_{x_{j-1}}^x \frac{x_j - x}{x_j - x_{j-1}} u'(\xi) d\xi + \frac{(x_j - x)(x - x_{j-1})}{x_j - x_{j-1}} u'(x) \\ &+ \int_x^{x_j} \frac{x - x_{j-1}}{x_j - x_{j-1}} u'(\xi) d\xi - \frac{(x_j - x)(x - x_{j-1})}{x_j - x_{j-1}} u'(x) \\ &= \frac{x_j - x}{x_j - x_{j-1}} (u(x_{j-1}) - u(x)) + \frac{x - x_{j-1}}{x_j - x_{j-1}} (u(x_j) - u(x)) \\ &= \underbrace{\frac{x_j - x}{x_j - x_{j-1}} u(x_{j-1}) + \frac{x - x_{j-1}}{x_j - x_{j-1}} u(x_j)}_{= \|_1 u(x) !} - u(x). \quad (3.3.1.8) \end{aligned}$$

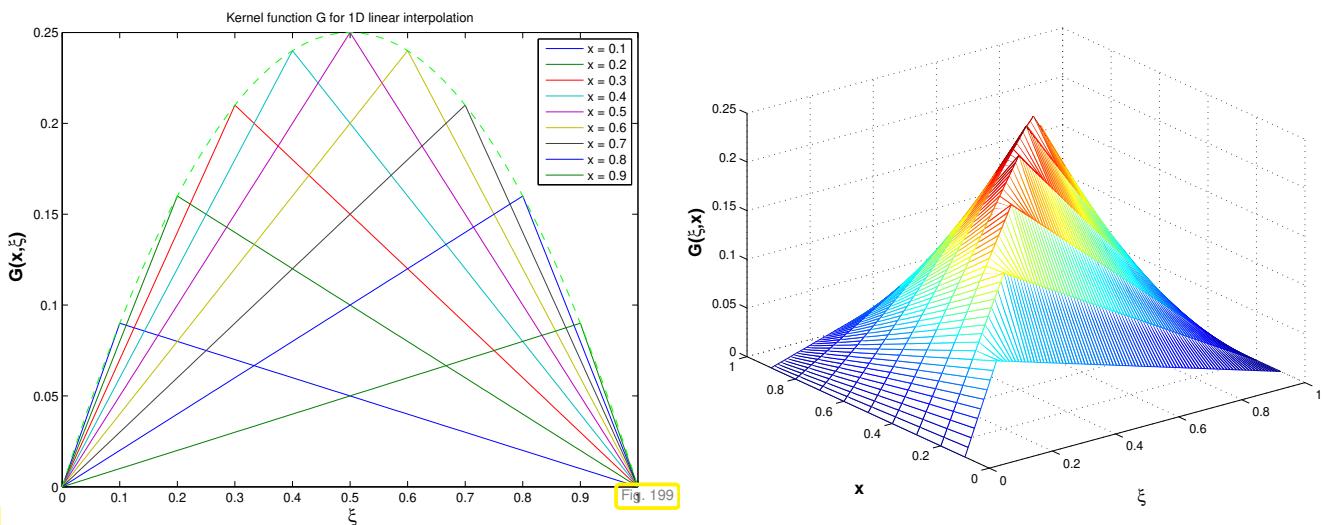
We also appealed to the fundamental theorem of calculus, which is (1.5.1.8) for $v \equiv 1$. What we have obtained is a (kernel) **representation formula** for the local interpolation error $\|_1 u - u$ of the form

$$(\|_1 u - u)(x) = \int_{x_{j-1}}^{x_j} G(x, \xi) u''(\xi) d\xi. \quad (3.3.1.9)$$

with $G(x, \xi) = \begin{cases} \frac{(x_j - x)(\xi - x_{j-1})}{x_j - x_{j-1}} & \text{for } x_{j-1} \leq \xi < x, \\ \frac{(x - x_{j-1})(x_j - \xi)}{x_j - x_{j-1}} & \text{for } x \leq \xi \leq x_j. \end{cases}$, which satisfies

$$|G(x, \xi)| \leq |x_j - x_{j-1}| \Rightarrow \int_{x_{j-1}}^{x_j} G(x, \xi)^2 d\xi \leq |x_j - x_{j-1}|^3. \quad (3.3.1.10)$$

The following figures display the **kernel function G** for 1D linear interpolation and for $x_{j-1} = 0$, $x_j = 1$.



The next step relies on the Cauchy-Schwarz inequality

$$\int_{\Omega} f(x)g(x) dx \leq \|f\|_{L^2(\Omega)} \|g\|_{L^2(\Omega)} \quad \forall f, g \in L^2(\Omega), \quad (1.3.4.15)$$

for $\Omega = [x_{j-1}, x_j]$, which is applied to the representation (3.3.1.9):

$$\begin{aligned} \blacktriangleright \int_{x_{j-1}}^{x_j} |u(x) - l_1 u(x)|^2 dx &= \int_{x_{j-1}}^{x_j} \left| \int_{x_{j-1}}^{x_j} G(x, \xi) u''(\xi) d\xi \right|^2 dx \\ &\stackrel{(1.3.4.15)}{\leq} \int_{x_{j-1}}^{x_j} \left\{ \int_{x_{j-1}}^{x_j} G(x, \xi)^2 d\xi \cdot \int_{x_{j-1}}^{x_j} |u''(\xi)|^2 d\xi \right\} dx. \end{aligned} \quad (3.3.1.11)$$

As a consequence of (3.3.1.10) we can drop the kernel function G from right hand side of (3.3.1.11)

$$\begin{aligned} \stackrel{(3.3.1.10)}{\Rightarrow} \|u - l_1 u\|_{L^2([x_{j-1}, x_j])}^2 &= \int_{x_{j-1}}^{x_j} |u(x) - l_1 u(x)|^2 dx \leq |x_j - x_{j-1}|^4 \int_{x_{j-1}}^{x_j} |u''(\xi)|^2 d\xi. \end{aligned} \quad (3.3.1.12)$$

Use this estimate on $[x_{j-1}, x_j]$, note that $|x_j - x_{j-1}| \leq h_M$ for any j , sum over all cells of the mesh \mathcal{M} , and take the square root.

$$(3.3.1.12) \Rightarrow \|u - l_1 u\|_{L^2([a,b])} \leq h_M^2 \|u''\|_{L^2([a,b])}. \quad (3.3.1.13)$$

§3.3.1.14 (Estimating the error norm $\|u - l_1 u\|_{H^1([a,b])}$) In light of the definition of the $H^1([a,b])$ -seminorm in Def. 1.3.4.3, we first differentiate the representation formula (3.3.1.9): for $x_{j-1} < x < x_j$, using the explicit piecewise linear representation of G ,

$$\begin{aligned} \frac{d}{dx} (l_1 u - u)(x) &= \int_{x_{j-1}}^{x_j} \frac{\partial G}{\partial x}(x, \xi) u''(\xi) d\xi \\ &= \int_{x_{j-1}}^{x_j} -\frac{\xi - x_{j-1}}{x_j - x_{j-1}} u''(\xi) d\xi + \int_{x_{j-1}}^{x_j} \frac{x_j - \xi}{x_j - x_{j-1}} u''(\xi) d\xi. \end{aligned}$$

Again, the Cauchy-Schwarz inequality (1.3.4.15) on $\Omega :=]x_{j-1}, x_j[$ is useful and yields

$$\begin{aligned} \int_{x_{j-1}}^{x_j} \left| \frac{d}{dx} (\mathbf{l}_1 u - u)(x) \right|^2 dx &= \int_{x_{j-1}}^{x_j} \left| \int_{x_{j-1}}^{x_j} \frac{\partial G}{\partial x}(x, \xi) u''(\xi) d\xi \right|^2 dx \\ &\leq \int_{x_{j-1}}^{x_j} \left\{ \int_{x_{j-1}}^{x_j} \underbrace{\left| \frac{\partial G}{\partial x}(x, \xi) \right|^2}_{\leq 1} d\xi \cdot \int_{x_{j-1}}^{x_j} |u''(\xi)|^2 d\xi \right\} dx. \end{aligned} \quad (3.3.1.15)$$

► $|u - \mathbf{l}_1 u|_{H^1([x_{j-1}, x_j])}^2 \leq (x_j - x_{j-1})^2 \int_{x_{j-1}}^{x_j} |u''(\xi)|^2 d\xi.$ (3.3.1.16)

As above, apply this estimate on $[x_{j-1}, x_j]$, use $|x_j - x_{j-1}| \leq h_{\mathcal{M}}$, sum over all cells of the mesh \mathcal{M} and take square root.

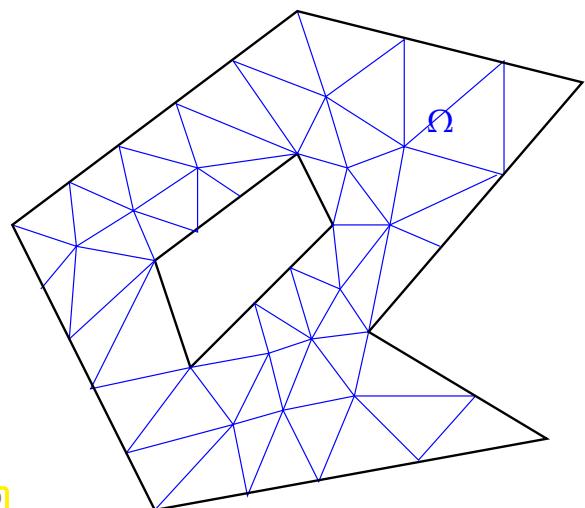
$$(3.3.1.16) \Rightarrow |u - \mathbf{l}_1 u|_{H^1([a,b])} \leq h_{\mathcal{M}} \|u''\|_{L^2([a,b])}. \quad (3.3.1.17)$$

□

Lessons from the last three §s:

1. We have to rely on **smoothness** of the interpolant u to obtain bounds for norms of the interpolation error. In the above estimates, we have to take for granted boundedness/square integrability of the second derivative.
2. The bounds for the norms of the interpolation error involve norms of derivatives of the interpolant.
3. For sufficiently smooth u we find **algebraic convergence** (\rightarrow Def. 3.2.2.1) of norms of the interpolation error *in terms of mesh width* $h_{\mathcal{M}} \rightarrow 0$.

3.3.2 Error Estimates for Linear Interpolation in 2D



The setting for this section is as follows.

We are given

- ◆ a polygonal domain $\Omega \subset \mathbb{R}^2$
- ◆ a triangular mesh \mathcal{M} of Ω
 $(\rightarrow$ Def. 2.5.1.1)

Section 3.3.1 introduced piecewise linear interpolation on a mesh/grid in 1D. The next definition gives the natural 2D counterpart on a triangular mesh and for piecewise linear Lagrangian finite elements.

Definition 3.3.2.1. Linear interpolation in 2D

The linear interpolation operator $\mathbf{l}_1 : C^0(\bar{\Omega}) \mapsto \mathcal{S}_1^0(\mathcal{M})$ is defined by

$$\mathbf{l}_1 u \in \mathcal{S}_1^0(\mathcal{M}) \quad , \quad \mathbf{l}_1 u(\mathbf{p}) = u(\mathbf{p}) \quad \forall \mathbf{p} \in \mathcal{V}(\mathcal{M}) .$$



This is a valid definition, because a function $v_h \in \mathcal{S}_1^0(\mathcal{M})$ is *uniquely determined* by its values in the vertices of the mesh (\rightarrow 2.4.3.3), which are the interpolation nodes for the linear Lagrangian finite element space.

Recalling the definition of the nodal basis $\mathfrak{B} = \{b_h^\mathbf{p} : \mathbf{p} \in \mathcal{V}(\mathcal{M})\}$ of $\mathcal{S}_1^0(\mathcal{M})$ from (2.4.3.5), where $b_h^\mathbf{p}$ is the “tent function” associated with node \mathbf{p} , an equivalent definition is, cf. (2.7.6.6),

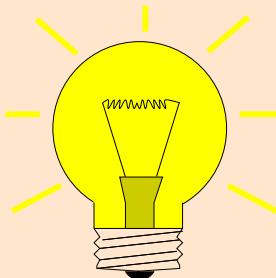
$$\mathbf{l}_1 u = \sum_{\mathbf{p} \in \mathcal{V}(\mathcal{M})} u(\mathbf{p}) b_h^\mathbf{p} , \quad u \in C^0(\bar{\Omega}) . \quad (3.3.2.2)$$

Task: For “sufficiently smooth” $u : \Omega \mapsto \mathbb{R}$, $u \in C^\infty(\bar{\Omega})$ to begin with, establish a bound for the $H^1(\Omega)$ -norm of the **interpolation error** $\|u - \mathbf{l}_1 u\|_{H^1(\Omega)}$ in terms of geometric discretization parameters related to the mesh \mathcal{M} .

As in § 3.3.1.6 we note that the error norm $\|u - \mathbf{l}_1 u\|_{H^1(\Omega)}$ can be computed by summing local contribution. This suggests **localization** in the same vein as in 1D.

Interpolation error estimation: localization trick

Again, linear interpolation in 2D according to Def. 3.3.2.1 is **local** in the sense that $\mathbf{l}_1 u$ inside a triangle K depends only on the values of u in \bar{K} .



Idea:

\mathbf{l}_1 local \Rightarrow first, estimate $\|u - \mathbf{l}_1 u\|_{H^1(K)}^2$, $K \in \mathcal{M}$,
then, global estimate via summation as in § 3.3.1.6.
 \Rightarrow Focus on single triangle $K \in \mathcal{M}$

Localization

Crucial for localization to work is the fact that the linear interpolation operator $\mathbf{l}_1 : C^0(\bar{\Omega}) \mapsto \mathcal{S}_1^0(\mathcal{M})$ can be defined **purely locally** by the concrete barycentric interpolation formula

$$\mathbf{l}_1 u|_K = u(\mathbf{a}^1)\lambda_1 + u(\mathbf{a}^2)\lambda_2 + u(\mathbf{a}^3)\lambda_3 , \quad (3.3.2.4)$$

for each triangle $K \in \mathcal{M}$ with vertices $\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3$ ($\lambda_k \hat{=} \text{barycentric coordinate functions} = \text{local shape functions for } \mathcal{S}_1^0(\mathcal{M})$, see Fig. 75).

§3.3.2.5 (Representation formula for interpolation error) The main steps parallel those for the 1D case in § 3.3.1.6 and § 3.3.1.14, though the technicalities are much more intricate. We start with a **representation formula** for local interpolation errors, cf. (3.3.1.8). Its derivation solely relies on elementary formulas from calculus.

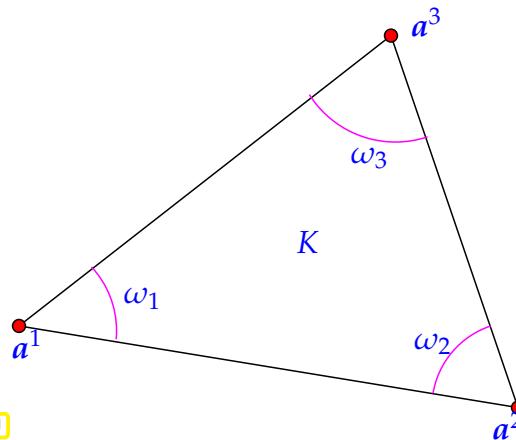


Fig. 201

$u \in C^2(\bar{K})$: by mean value formula $\forall x \in K$,

$$u(a^j) = u(x) + \mathbf{grad} u(x) \cdot (a^j - x) + \int_0^1 (a^j - x)^\top D^2 u(x + \xi(a^j - x))(a^j - x)(1 - \xi) d\xi , \quad (3.3.2.6)$$

$$D^2 u(x) = \begin{bmatrix} \frac{\partial^2 u}{\partial x_1^2}(x) & \frac{\partial^2 u}{\partial x_1 \partial x_2}(x) \\ \frac{\partial^2 u}{\partial x_1 \partial x_2}(x) & \frac{\partial^2 u}{\partial x_2^2}(x) \end{bmatrix} \triangleq \text{Hessian.}$$

The formula (3.3.2.6) is easily verified by applying integration by parts (1.5.1.8) in the form

$$f(b) - f(a) = [\xi f'(\xi)]_a^b - \int_a^b \xi f''(\xi) d\xi = f'(a)(b - a) + \int_a^b (b - \xi) f''(\xi) d\xi .$$

to the function $f(t) = u(ta^j + (1-t)x)$ with $a = 0, b = 1$. Use the multi-dimensional chain rule to express the derivatives of f through derivatives of u :

$$\begin{aligned} f'(t) &= \mathbf{grad} u(ta^j + (1-t)x)^\top (a^j - x) , \\ f''(t) &= (a^j - x)^\top D^2 u(ta^j + (1-t)x)(a^j - x) . \end{aligned}$$

Next, use (3.3.2.6) to replace $u(a^j)$ in the formula (3.3.2.4) for local linear interpolation. Also use the identities for the barycentric coordinate functions

$$\sum_{j=1}^3 \lambda_j(x) = 1 \quad , \quad x = \sum_{j=1}^3 a^j \lambda_j(x) . \quad (3.3.2.7)$$

$$l_1 u(x) = \sum_{j=1}^3 u(a^j) \lambda_j(x) = u(x) \cdot \underbrace{\sum_{j=1}^3 \lambda_j(x)}_{=1} + \mathbf{grad} u(x) \cdot \underbrace{\sum_{j=1}^3 (a^j - x) \lambda_j(x)}_{=0} + R(x) ,$$

$$\text{with } R(x) := \sum_{j=1}^3 \left(\int_0^1 (a^j - x)^\top D^2 u(x + \xi(a^j - x))(a^j - x)(1 - \xi) d\xi \right) \lambda_j(x) . \quad (3.3.2.8)$$

Again, as in the case of (3.3.1.8) for 1D linear interpolation we have arrived at an **integral representation formula** for the local interpolation error:

$$(u - l_1 u)(x) = \sum_{j=1}^3 \left(\int_0^1 (a^j - x)^\top D^2 u(x + \xi(a^j - x))(a^j - x)(1 - \xi) d\xi \right) \lambda_j(x) . \quad (3.3.2.9)$$

§3.3.2.10 (Estimate for L^2 -norm of interpolation error) Together with the triangle inequality, the trivial bound $|\lambda_j| \leq 1$ yields

$$\|u - l_1 u\|_{L^2(K)} \leq \sum_{j=1}^3 \left(\int_K \left(\int_0^1 (a^j - x)^\top D^2 u(x + \xi(a^j - x))(a^j - x)(1 - \xi) d\xi \right)^2 dx \right)^{\frac{1}{2}} .$$

To estimate an expression of the form

$$\int_K \left(\int_0^1 (\mathbf{a}^j - \mathbf{x})^T D^2 u(\mathbf{x} + \xi(\mathbf{a}^j - \mathbf{x})) (\mathbf{a}^j - \mathbf{x})(1 - \xi) d\xi \right)^2 d\mathbf{x}, \quad (3.3.2.11)$$

we may assume, without loss of generality, that $\mathbf{a}^j = \mathbf{0}$.

➤ Task: estimate terms (where 0 is a vertex of K !)

$$\int_K \left(\int_0^1 \mathbf{x}^\top D^2 u((1 - \xi)\mathbf{x}) \mathbf{x}(1 - \xi) d\xi \right)^2 d\mathbf{x} = \int_K \left(\int_0^1 \mathbf{x}^\top D^2 u(\xi\mathbf{x}) \mathbf{x}\xi d\xi \right)^2 d\mathbf{x}.$$

Denote $\gamma \hat{=} \text{angle of } K \text{ at vertex } 0$,
 $h \hat{=} \text{length of longest edge of } K$.



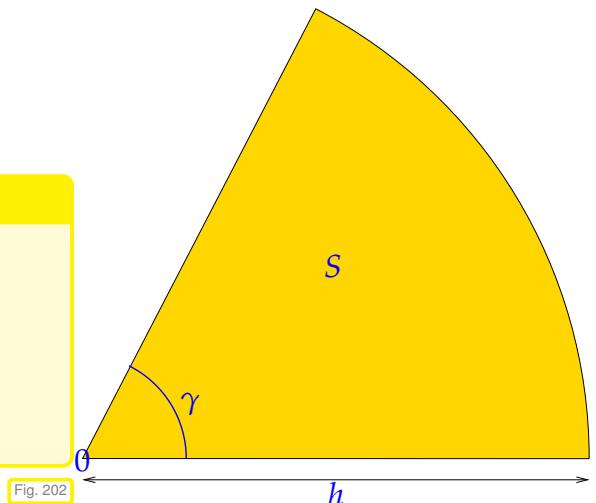
K is contained in the sector

$$S := \{ \mathbf{x} = (r \cos \varphi, r \sin \varphi) : 0 \leq r < h, 0 \leq \varphi \leq \gamma \}$$

Lemma 3.3.2.12. Auxiliary estimate on sector

For any $\psi \in L^2(S)$ holds

$$\int_S \left(\int_0^1 |\mathbf{y}|^2 \psi(\tau \mathbf{y}) \tau d\tau \right)^2 d\mathbf{y} \leq \frac{h^4}{8} \|\psi\|_{L^2(S)}^2.$$



Using polar coordinates (r, φ) , $\hat{\mathbf{s}}_\varphi = (\cos \varphi, \sin \varphi)$, see [Str09, Bsp. 8.5.3], and Cauchy-Schwarz inequality (1.3.4.15):

$$\begin{aligned} \int_S \left(\int_0^1 |\mathbf{y}|^2 \psi(\tau \mathbf{y}) \tau d\tau \right)^2 d\mathbf{y} &= \int_0^\gamma \int_0^h \left(\int_0^1 r^2 \psi(\tau r \hat{\mathbf{s}}_\varphi) \tau d\tau \right)^2 r dr d\varphi \\ &= \int_0^\gamma \int_0^h \left(\int_0^r \psi(\sigma \hat{\mathbf{s}}_\varphi) \sigma d\sigma \right)^2 r dr d\varphi \leq \int_0^\gamma \int_0^h \int_0^r \psi^2(\sigma \hat{\mathbf{s}}_\varphi) \sigma d\sigma \cdot \int_0^r \sigma d\sigma r dr d\varphi \\ &\leq \frac{1}{2} \int_0^\gamma \int_0^h \psi^2(\sigma \hat{\mathbf{s}}_\varphi) \sigma d\sigma d\varphi \cdot \int_0^h r^3 dr. \end{aligned}$$

Use $|\mathbf{z}^\top \mathbf{A} \mathbf{y}| \leq \|\mathbf{A}\|_F \|\mathbf{z}\| \|\mathbf{y}\|$, $\mathbf{A} \in \mathbb{R}^{n,n}$, $\mathbf{z}, \mathbf{y} \in \mathbb{R}^n$, and then apply § 3.3.2.10 with $\mathbf{y} := \mathbf{x} - \mathbf{a}^j$, $\tau = 1 - \xi$

$$\Rightarrow \|u - I_1 u\|_{L^2(K)}^2 \leq \frac{3}{8} h_K^4 \left\| \left\| D^2 u \right\|_F \right\|_{L^2(K)}^2, \quad (3.3.2.13)$$

with **Frobenius matrix norm** $\left\| D^2 u(\mathbf{x}) \right\|_F^2 := \sum_{i,j=1}^2 \left| \frac{\partial^2 u}{\partial x_i \partial x_j}(\mathbf{x}) \right|^2$ (\rightarrow [Hip19, ??]),
size of triangle $h_K := \text{diam } K := \max\{|\mathbf{p} - \mathbf{q}| : \mathbf{p}, \mathbf{q} \in K\}$

§3.3.2.14 (Estimate of local H^1 -seminorm of the interpolation error) Estimate for gradient: from (3.3.2.6) we infer the local integral representation formula, which can also be obtained by taking the gradient of (3.3.2.9).

$$\begin{aligned}\mathbf{grad} \mathbf{l}_1 u(\mathbf{x}) &= \sum_{j=1}^3 u(\mathbf{a}^j) \mathbf{grad} \lambda_j(\mathbf{x}) \\ &= \sum_{j=1}^3 \left(u(\mathbf{x}) + \mathbf{grad} u(\mathbf{x}) \cdot (\mathbf{a}^j - \mathbf{x}) + \int_0^1 \dots d\xi \right) \mathbf{grad} \lambda_j(\mathbf{x}) \\ &= u(\mathbf{x}) \underbrace{\sum_{j=1}^3 \mathbf{grad} \lambda_j(\mathbf{x})}_{=0} + \underbrace{\left(\sum_{j=1}^3 (\mathbf{a}^j - \mathbf{x})^\top \mathbf{grad} \lambda_j(\mathbf{x}) \right) \cdot \mathbf{grad} u(\mathbf{x})}_{=1} + G(\mathbf{x}), \\ \text{with } G(\mathbf{x}) &:= \sum_{j=1}^3 \underbrace{\left(\int_0^1 (\mathbf{a}^j - \mathbf{x})^\top D^2 u(\mathbf{x} + \xi(\mathbf{a}^j - \mathbf{x})) (\mathbf{a}^j - \mathbf{x})(1 - \xi) d\xi \right)}_{\text{cf. (3.3.2.11)}} \mathbf{grad} \lambda_j(\mathbf{x}).\end{aligned}$$

Note that $\mathbf{grad} \sum_{j=1}^3 \lambda_j(\mathbf{x}) = \mathbf{grad} \mathbf{1} = 0$ and

$$\sum_{j=1}^3 \mathbf{grad} \lambda_j(\mathbf{x}) (\mathbf{a}^j - \mathbf{x})^\top = \sum_{j=1}^3 \mathbf{grad} \lambda_j(\mathbf{x}) (\mathbf{a}^j)^\top = \mathbf{grad} \left(\sum_{j=1}^3 \lambda_j(\mathbf{x}) \mathbf{a}^j \right) = \mathbf{grad} \mathbf{x} = \mathbf{I}. \quad (3.3.2.15)$$

As an immediate consequence of the formulas from Section 2.4.5

$$\begin{aligned}\mathbf{grad} \lambda_1 &= -\frac{|e_1|}{2|K|} \mathbf{n}^1 = \frac{1}{2|K|} (\mathbf{a}^2 - \mathbf{a}^3)^\perp = \frac{1}{2|K|} \begin{pmatrix} a_2^2 - a_2^3 \\ a_1^3 - a_1^2 \end{pmatrix}, \\ \mathbf{grad} \lambda_2 &= -\frac{|e_2|}{2|K|} \mathbf{n}^2 = \frac{1}{2|K|} (\mathbf{a}^3 - \mathbf{a}^1)^\perp = \frac{1}{2|K|} \begin{pmatrix} a_2^3 - a_2^1 \\ a_1^1 - a_1^3 \end{pmatrix}, \\ \mathbf{grad} \lambda_3 &= -\frac{|e_3|}{2|K|} \mathbf{n}^3 = \frac{1}{2|K|} (\mathbf{a}^1 - \mathbf{a}^2)^\perp = \frac{1}{2|K|} \begin{pmatrix} a_1^1 - a_1^2 \\ a_2^2 - a_2^1 \end{pmatrix},\end{aligned}$$

we conclude

$$(2.7.5.4) \quad \mathbf{grad} \lambda_j(\mathbf{x}) \leq \frac{h_K}{2|K|}, \quad \mathbf{x} \in K. \quad (3.3.2.16)$$

$$\Rightarrow \| \mathbf{grad}(u - \mathbf{l}_1 u) \|_{L^2(K)}^2 \leq \frac{h_K^2}{4|K|^2} \| R \|_{L^2(K)}^2 \stackrel{(3.3.2.13)}{\leq} \frac{3}{8} \frac{h_K^6}{4|K|^2} \| \|D^2 u\|_F\|_{L^2(K)}^2. \quad (3.3.2.17)$$

Summary of *local* interpolation error estimates for linear interpolation according to Def. 3.3.2.1:

Lemma 3.3.2.18. Local interpolation error estimates for 2D linear interpolation

For any triangle K and $u \in C^2(\bar{K})$ the following holds

$$\|u - \mathbf{l}_1 u\|_{L^2(K)}^2 \leq \frac{3}{8} h_K^4 \| \|D^2 u\|_F\|_{L^2(K)}^2, \quad (3.3.2.13)$$

$$\| \mathbf{grad}(u - \mathbf{l}_1 u) \|_{L^2(K)}^2 \leq \frac{3}{32} \frac{h_K^6}{|K|^2} \| \|D^2 u\|_F\|_{L^2(K)}^2. \quad (3.3.2.17)$$

§3.3.2.19 (Shape regularity) Note: the estimates of Lemma 3.3.2.18 are structurally similar to the 1D estimates (3.3.1.12) and (3.3.1.16) in the sense that the bounds involve L^2 -norms of second derivatives and factors depending on the cell size.

New aspect compared to Section 3.3.1: *shape* of K enters error bounds of Lemma 3.3.2.18. This dependence on shape can be reduced to a single number:

Definition 3.3.2.20. Shape regularity measure

For a simplex $K \in \mathbb{R}^d$ we define its **shape regularity measure** as the ratio

$$\rho_K := h_K^d : |K| , \quad h_K := \text{diam}(K) ,$$

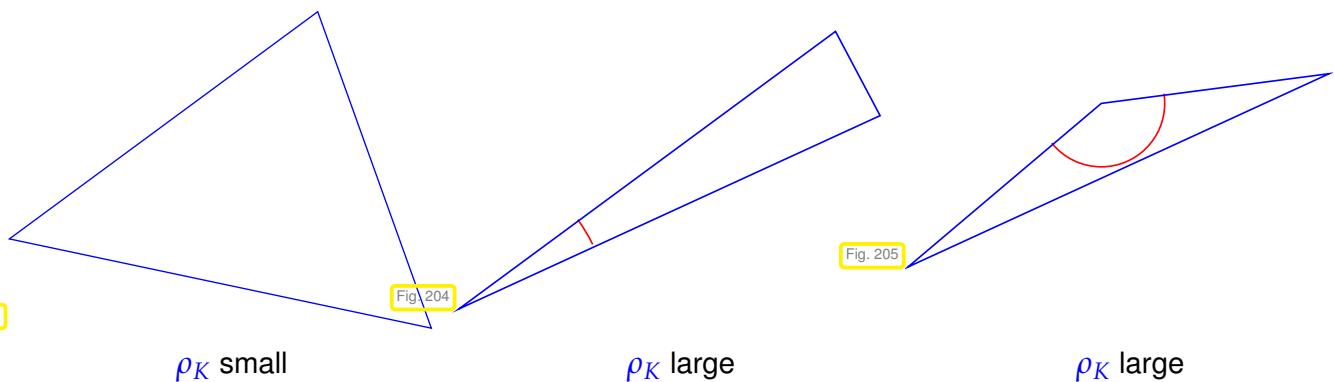
and the shape regularity measure of a simplicial mesh $\mathcal{M} = \{K\}$ as

$$\rho_{\mathcal{M}} := \max_{K \in \mathcal{M}} \rho_K .$$

Important: shape regularity measure ρ_K is an invariant of a similarity class of triangles.

This means that the shape regularity measure does not change when a triangle is transformed by scaling, rotation, and translation. Sloppily speaking, ρ_K depends only on the shape, not the size of K .

For triangle K : ρ_K large $\Leftrightarrow K$ “distorted” $\Leftrightarrow K$ has small angles



The shape regularity measure $\rho_{\mathcal{M}}$ is often used to gauge the *quality* of meshes produced by mesh generators. \square

Now we return to estimates for norms of the interpolation error for piecewise linear interpolation I_1 . The final step is to add up the local estimates from Lemma 3.3.2.18 over all triangles of the mesh and take the square root.

Theorem 3.3.2.21. Error estimate for piecewise linear interpolation

For any $u \in C^2(\bar{\Omega})$ and 2D piecewise linear interpolation $I_1 : C^0(\bar{\Omega}) \rightarrow \mathcal{S}_1^0(\mathcal{M})$, \mathcal{M} a triangular mesh, holds

$$\begin{aligned} \|u - I_1 u\|_{L^2(\Omega)} &\leq \sqrt{\frac{3}{8}} h_{\mathcal{M}}^2 \left\| \left\| D^2 u \right\|_F \right\|_{L^2(\Omega)}, \\ \|\mathbf{grad}(u - I_1 u)\|_{L^2(\Omega)} &\leq \sqrt{\frac{3}{24}} \rho_{\mathcal{M}} h_{\mathcal{M}} \left\| \left\| D^2 u \right\|_F \right\|_{L^2(\Omega)}. \end{aligned}$$

where $h_{\mathcal{M}}$ denotes the mesh width (\rightarrow Def. 3.2.1.4) and $\rho_{\mathcal{M}}$ the shape regularity measure (\rightarrow Def. 3.3.2.20) of \mathcal{M} .

Remark 3.3.2.22 (Energy norm and $H^1(\Omega)$ -norm) One might object that Cea's lemma Thm. 3.1.3.7 refers to the energy norm, but Thm. 3.3.2.21 provides estimates in $H^1(\Omega)$ -norm only!

- ☞ For uniformly positive definite (\rightarrow Def. 1.2.2.9) and bounded coefficient tensor $\alpha : \Omega \mapsto \mathbb{R}^{d,d}$, cf. (1.2.2.8),

$$\exists 0 < \alpha^- < \alpha^+ : \alpha^- \|z\|^2 \leq z^T \alpha(x) z \leq \alpha^+ \|z\|^2 \quad \forall z \in \mathbb{R}^d, x \in \Omega ,$$

and the energy norm (\rightarrow Def. 1.2.3.35) induced by

$$a(u, v) := \int_{\Omega} (\alpha(x) \mathbf{grad} u) \cdot \mathbf{grad} v \, dx , \quad u, v \in H_0^1(\Omega) , \quad (3.1.1.7)$$

we immediately find the **equivalence** (= two-sided uniform estimate)

$$\sqrt{\alpha^-} |v|_{H^1(\Omega)} \leq \|v\|_a \leq \sqrt{\alpha^+} |v|_{H^1(\Omega)} \quad \forall v \in H^1(\Omega) . \quad (3.3.2.23)$$

Thus, interpolation error estimates in $|\cdot|_{H^1(\Omega)}$ immediately translate into estimates in terms of the energy norm (with bounds for the coefficient entering the constants). \square

3.3.3 The Sobolev Scale of Function Spaces

Interpolation error estimates like in Thm. 3.3.2.21 hinge on smoothness of the interpolant u : the bounds in Thm. 3.3.2.21 the term $\|\|D^2 u\|_F\|_{L^2(\Omega)}$. This norm conveys two messages:

- ◆ $\|\|D^2 u\|_F\|_{L^2(\Omega)} < \infty$ is a **smoothness requirement**.
- ◆ The size of $\|\|D^2 u\|_F\|_{L^2(\Omega)}$ is a measure for the smoothness of u .

In this section we take a closer look at norms involving derivatives and their capability to indicate the smoothness of a function. In fact, in the guise of the $H^1(\Omega)$ -seminorm from Def. 1.3.4.3 we have already come across an example for such a norm. Thus, what we pursue in this section can also be regarded as a generalization of $H^1(\Omega)$.

Definition 3.3.3.1. Higher order Sobolev spaces/norms

The **m -th order Sobolev norm**, $m \in \mathbb{N}_0$, for $u : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$ (sufficiently smooth) is defined by

$$\|u\|_{H^m(\Omega)}^2 := \sum_{k=0}^m \sum_{\alpha \in \mathbb{N}^d, |\alpha|=k} \int_{\Omega} |D^\alpha u|^2 \, dx , \quad \text{where} \quad D^\alpha u := \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}} .$$

$$\text{Sobolev space} \quad H^m(\Omega) := \{v : \Omega \mapsto \mathbb{R} : \|v\|_{H^m(\Omega)} < \infty\} .$$

- ☞ To understand this definition recall the multi-index notation from (2.5.2.3) and (2.5.2.4):

$$\begin{aligned} \alpha = (\alpha_1, \dots, \alpha_d) : \quad x^\alpha &:= x_1^{\alpha_1} \cdots x_d^{\alpha_d} , \\ |\alpha| &= \alpha_1 + \alpha_2 + \cdots + \alpha_d . \end{aligned}$$

§3.3.3.2 (Purposes of Sobolev spaces) Gripe (as in Section 1.3): Don't bother me with these Sobolev spaces !

Response: Well, these concepts are pervasive in the numerical analysis literature and you have to be familiar with them, in particular, with the notations.

Reassuring:

Again, it is only the norms $\|u\|_{H^m(\Omega)}$ that matter for us !

Now, we have come across an additional purpose of Sobolev spaces and their norms:



It goes without saying that the Sobolev spaces $H^m(\Omega)$ yield a sequence of *nested* function spaces on Ω , the

Sobolev scale: $\dots \subset H^3(\Omega) \subset H^2(\Omega) \subset H^1(\Omega) \subset L^2(\Omega)$

Observation: The bounds in Thm. 3.3.2.21 are determined by the “principal parts” of the Sobolev norms, that is, the parts containing the highest partial derivatives. The next concept isolates these highest-order derivative terms in the norms, and generalizes the **semi-norm** $|\cdot|_{H^1(\Omega)}$, refer to Rem. 1.3.4.12.

Definition 3.3.3.3. Higher-order Sobolev semi-norms

The m -th order Sobolev semi-norm, $m \in \mathbb{N}$, for sufficiently smooth $u : \Omega \mapsto \mathbb{R}$ is defined by

$$|u|_{H^m(\Omega)}^2 := \sum_{\alpha \in \mathbb{N}^d, |\alpha|=m} \int_{\Omega} |D^\alpha u|^2 \, dx .$$

Elementary observation: $|p|_{H^m(\Omega)} = 0 \Leftrightarrow p \in \mathcal{P}_{m-1}(\mathbb{R}^d)$

► By density arguments we can rewrite the interpolation error estimates of Thm. 3.3.2.21 in terms of Sobolev semi-norms:

Corollary 3.3.3.4. Error estimate for piecewise linear interpolation in 2D

Under the assumptions/with notations of Thm. 3.3.2.21

$$\begin{aligned} \|u - I_1 u\|_{L^2(\Omega)} &\leq \sqrt{\frac{3}{8}} h_M^2 |u|_{H^2(\Omega)}, & \forall u \in H^2(\Omega) . \\ |u - I_1 u|_{H^1(\Omega)} &\leq \sqrt{\frac{3}{24}} \rho_M h_M |u|_{H^2(\Omega)}, \end{aligned}$$

Remark 3.3.3.5 (Continuity of interpolation operators) An interpolation operator like I_1 maps functions to functions; it represents a *linear* mapping (= **operator**) between two function spaces. If we specify norms on these spaces, we may ask whether this mapping is continuous in the sense of the following definition,

which generalizes Def. 1.2.3.42.

Definition 3.3.3.6. Continuous linear operator

A linear mapping $T : X \rightarrow Y$ between two normed vector spaces X and Y is **continuous** or **bounded**, if and only if

$$\exists C > 0: \quad \|Tv\|_Y \leq C\|v\|_X \quad \forall v \in X.$$

In order to investigate the continuity of I_1 apply the Δ -inequality to the estimates of Cor. 3.3.3.4:

$$\|I_1 u\|_{L^2(\Omega)} \leq \|u\|_{L^2(\Omega)} + \sqrt{\frac{3}{8}} h_M^2 |u|_{H^2(\Omega)} \leq 2\|u\|_{H^2(\Omega)}, \quad (3.3.3.7)$$

if lengths are scaled such that $h_M \leq 1$. In light of Def. 3.3.3.6 estimate (3.3.3.7) means that $I_1 : H^2(\Omega) \mapsto L^2(\Omega)$ is a **continuous linear** mapping.

The same conclusion could have been drawn from the following fundamental result:

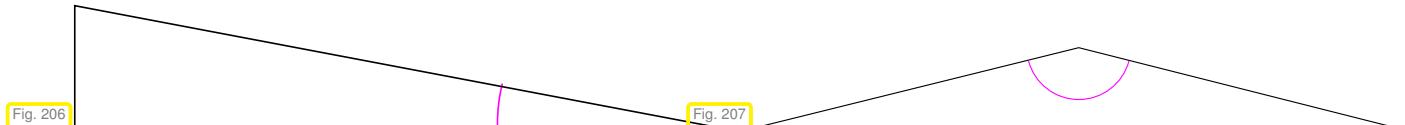
Theorem 3.3.3.8. Sobolev embedding theorem

$$m > \frac{d}{2} \Rightarrow H^m(\Omega) \subset C^0(\bar{\Omega}) \quad \wedge \quad \exists C = C(\Omega) > 0: \quad \|u\|_\infty \leq C\|u\|_{H^m(\Omega)} \quad \forall u \in H^m(\Omega).$$

Yet, for $d > 1$ the nodal interpolation operator $I_1 : H^1(\Omega) \mapsto L^2(\Omega)$ is **not** continuous, as we learn from Ex. 1.2.3.45 (“needle load” on taut membrane). \square

3.3.4 Anisotropic Interpolation Error Estimates

Look at the following triangular cells with “bad shape regularity” (ρ_K “large”): very small/large angles:



The estimates of Lemma 3.3.2.18 might suggest that we face huge local interpolation errors, once ρ_K becomes large.

Issue: are the estimates of Lemma 3.3.2.18 **sharp**?

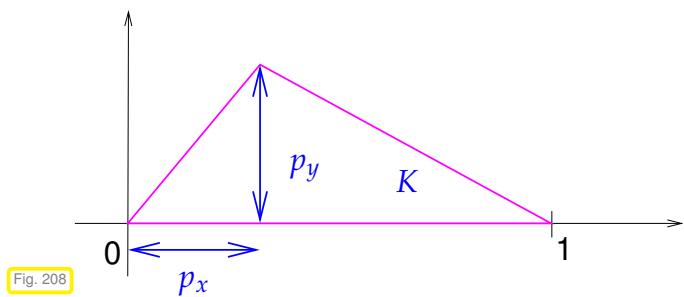
We will try to find this out experimentally by computing the best possible constants in the estimates

$$\|u - I_1 u\|_{L^2(K)} \leq C_{K,2} h_k^2 \|u\|_{H^2(K)}, \quad \|u - I_1 u\|_{H^1(K)} \leq C_K h_K \|u\|_{H^2(K)}.$$

Note: Merely translating, rotating, or scaling K does not affect the constants $C_{K,2}$ and C_K . Therefore, we can restrict ourselves to “canonical triangles”. Every general triangle can be mapped to one of these by translating, rotating, and scaling.

$$C_{K,2} := \sup_{u \in H^2(K) \setminus \{0\}} \frac{\|u - I_1 u\|_{L^2(K)}}{\|u\|_{H^2(K)}}, \quad C_K := \sup_{u \in H^2(k) \setminus \{0\}} \frac{\|u - I_1 u\|_{H^1(K)}}{\|u\|_{H^2(K)}},$$

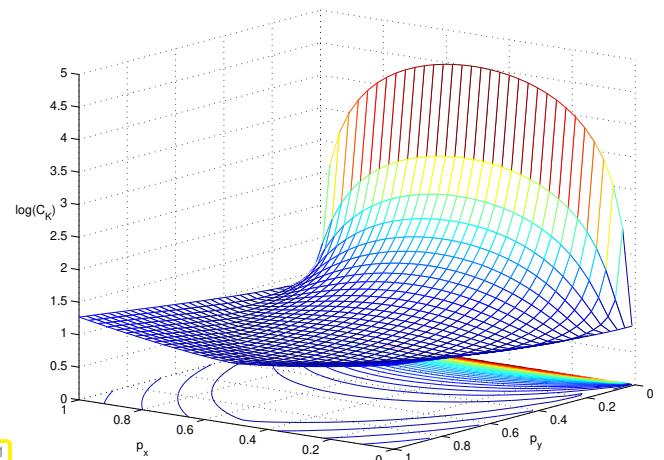
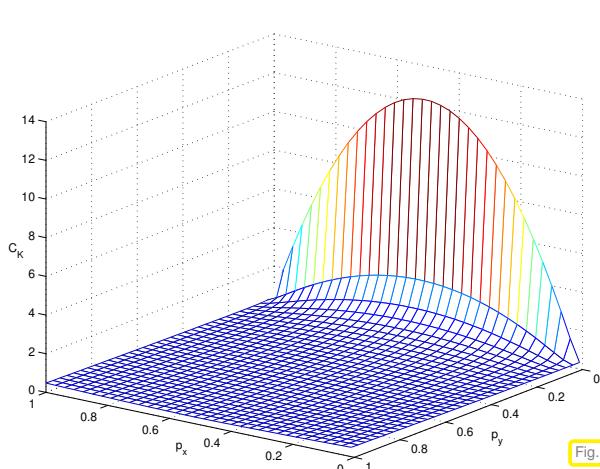
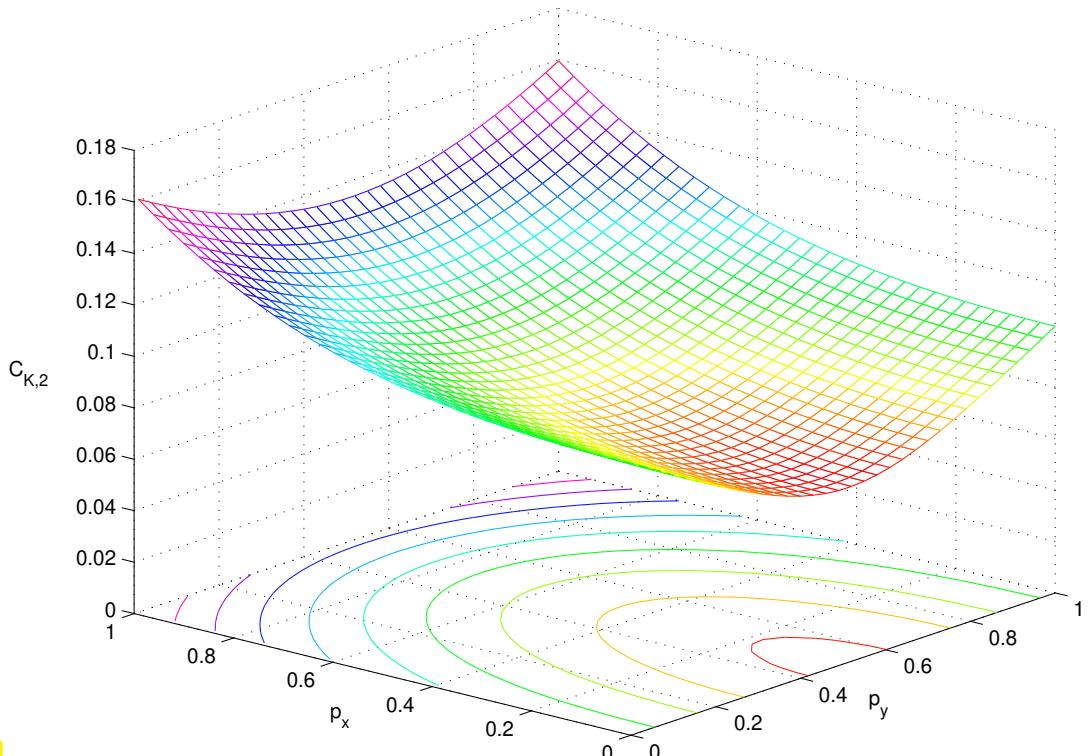
on triangle $K := \text{convex}\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \end{pmatrix} \right\}$.



Sampling the space of “canonical” triangles
(modulo similarity)

$$0 \leq p_x, p_y \leq 1.$$

+ Numerical computation of $C_K, C_{K,2}$
implementation by A. Inci (spectral polynomial
Galerkin method)



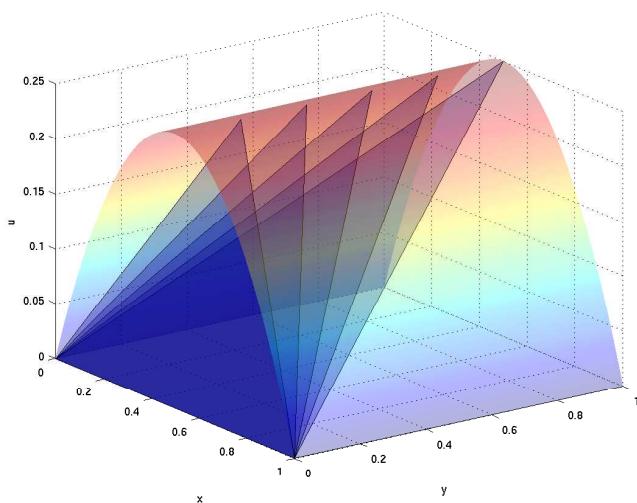
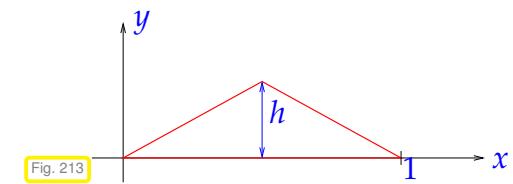


Fig. 212



triangle $K := \text{convex}\left\{\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right\}$, $h > 0$,
 $u(x, y) = x(1 - x)$, $0 < x < 1$.

◁ linear interpolant of u on K as $h \rightarrow 0$

The interpolant becomes steeper and steeper as $h \rightarrow 0$:

► $\|u\|_{H^2(K)}^2 = \frac{3031}{1440}h$, $\|u - I_1 u\|_{H^1(K)}^2 = \frac{29}{2880}h + \frac{1}{12}h + \frac{1}{32}h^{-1}$, $\|u - I_1 u\|_{L^2(K)}^2 = \frac{29}{2889}h$



$$\frac{\|u - I_1 u\|_{H^1(K)}^2}{\|u\|_{H^2(K)}^2} \geq \frac{269}{6062} + \frac{45}{3031}h^{-2}, \quad \frac{\|u - I_1 u\|_{L^2(K)}^2}{\|u\|_{H^2(K)}^2} = \frac{29}{6062}.$$

EXPERIMENT 3.3.4.1 (Good accuracy on “bad” meshes) $\Omega =]0, 1[^2$, $u(x_1, x_2) = \sin(\pi x_1) \sin(\pi x_2)$,
BVP $-\Delta u = f$, $u|_{\partial\Omega} = 0$, finite element Galerkin discretization on triangular meshes, $V_h = S_{1,0}^0(\mathcal{M})$.

☒ meshes created by random distortion of tensor product grids

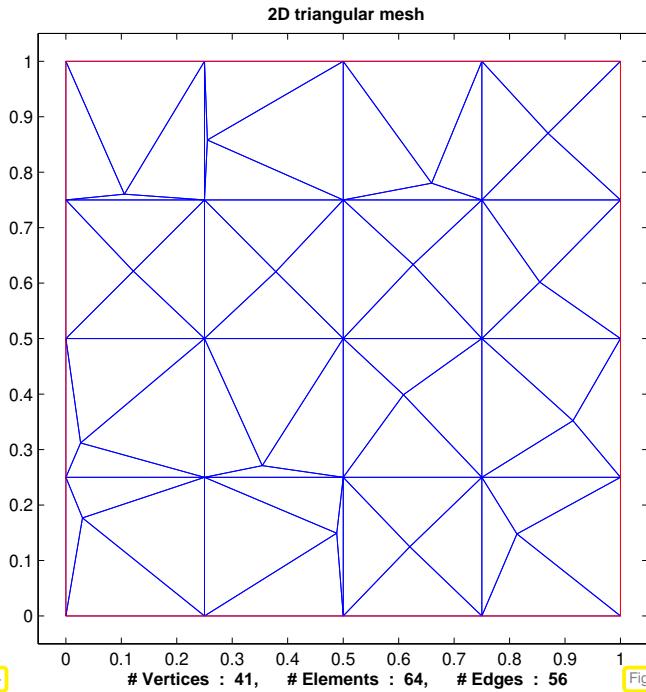


Fig. 214

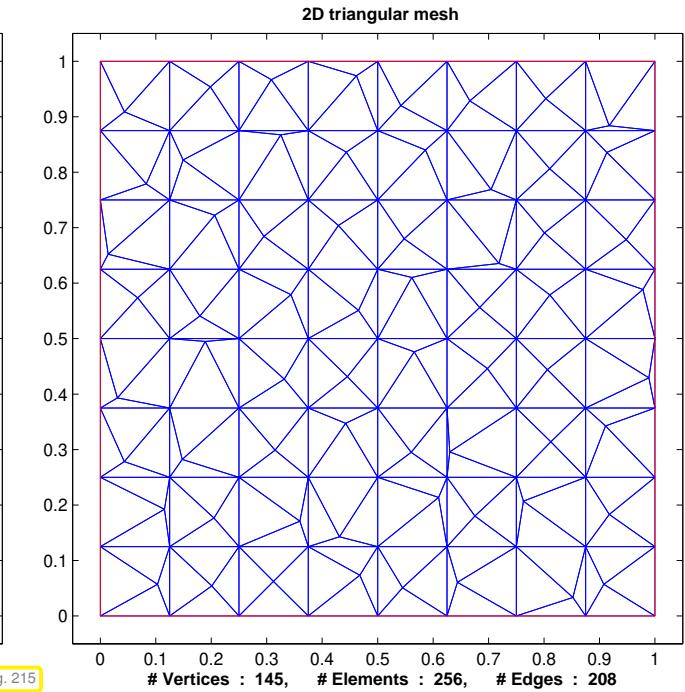
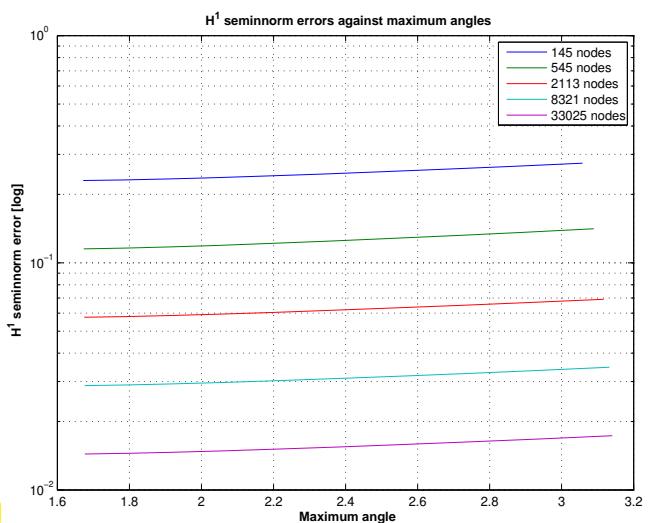
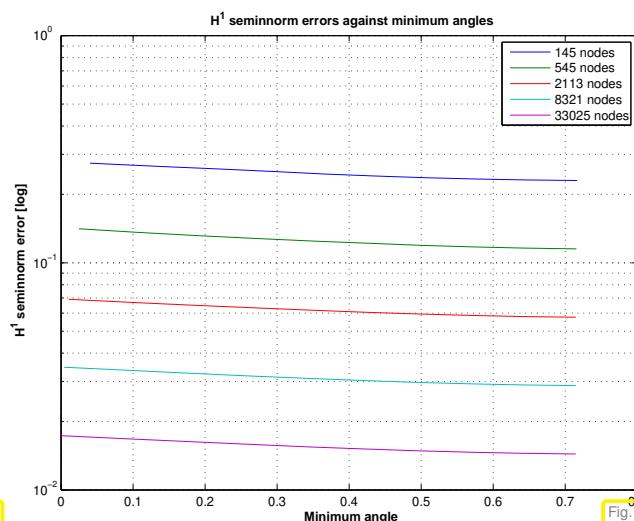


Fig. 215



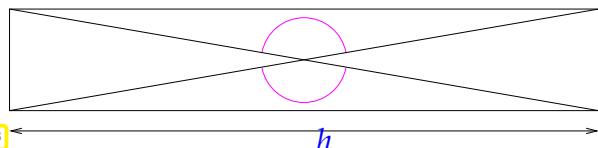
Monitored: for different mesh resolutions, $H^1(\Omega)$ -seminorm of discretization error as function of smallest/largest angle in the mesh.

Observation: Accuracy does *not* suffer much from distorted elements !

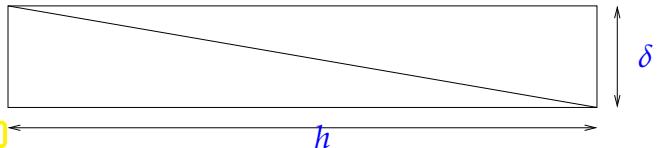
Remark 3.3.4.2 (Gap between interpolation error and best approximation error) Exp. 3.3.4.1 raises doubts whether the interpolation error can be trusted to provide good, that is, reasonably sharp bounds for the best approximation error.

In this example we will see that

$$\inf_{v_h \in \mathcal{S}_p^0(\mathcal{M})} \|u - v_h\|_{H^1(\Omega)} \ll \|u - I_p u\|_{H^1(\Omega)} \text{ is possible !}$$



Elementary cell of “bad mesh” \mathcal{M}_{bad}



Elementary cell of “good mesh” $\mathcal{M}_{\text{good}}$

On “bad” mesh : $\sup_{u \in H^2(\Omega)} \frac{\|u - I_1 u\|_{H^1(\Omega)}}{\|u\|_{H^2(\Omega)}} \rightarrow \infty \text{ as } h/\delta \rightarrow \infty,$

On “good” mesh : $\sup_{u \in H^2(\Omega)} \frac{\|u - I_1 u\|_{H^1(\Omega)}}{\|u\|_{H^2(\Omega)}} \text{ uniformly bounded in } h/\delta.$

Yet, $\inf_{v_h \in \mathcal{S}_1^0(\mathcal{M}_{\text{bad}})} \|u - v_h\|_{H^1(\Omega)} \leq \inf_{v_h \in \mathcal{S}_1^0(\mathcal{M}_{\text{good}})} \|u - v_h\|_{H^1(\Omega)} \quad \forall u \in H^2(\Omega).$

3.3.5 General Approximation Error Estimates for Lagrangian FEM: A Survey

In Section 3.3.2 we only examined the behavior of norms of the interpolation error for piecewise linear interpolation into $\mathcal{S}_1^0(\mathcal{M})$, that is, the case of Lagrangian finite elements of degree $p = 1$.

However, Exp. 3.2.3.7 sent the clear message that quadratic Lagrangian finite elements achieve faster convergence of the energy norm of the Galerkin discretization error, see Fig. 181, Fig. 182.

⇓

On the other hand quadratic finite elements could not deliver faster convergence in Exp. 3.2.3.10.

In this section we learn about theoretical results that shed light on these observations and extend the results of Section 3.3.2.

§3.3.5.1 (L^∞ interpolation error estimate in 1D) The faster convergence of quadratic Lagrangian FE in Exp. 3.2.3.7 does not come as a surprise: recall the estimate from [Hip19, ??]:

$$\|u - I_p u\|_{L^\infty([a,b])} \leq \frac{h_M^{p+1}}{(p+1)!} \|u^{(p+1)}\|_{L^\infty([a,b])} \quad \forall u \in C^{p+1}([a,b]),$$

where $I_p u$ is the \mathcal{M} -piecewise polynomial interpolant of u of local degree p . It generalizes (3.3.1.5), where this result was stated for $p = 1$.

➤ $|DS\|u - I_p u\|_{L^\infty([a,b])} = O(h_M^{p+1})$, provided that $u \in C^{p+1}([a,b])$. □

§3.3.5.2 (Local interpolation onto higher degree Lagrangian finite element spaces) Let \mathcal{M} be a triangular/tetrahedral/quadrilateral/hybrid mesh of domain Ω (→ Section 2.5.1)

Recall (→ Section 2.6) that the nodal basis functions of p -th degree Lagrangian finite element space $S_p^0(\mathcal{M})$ are defined as cardinal basis functions with respect to a set of **interpolation nodes**, cf. (2.6.1.4).

☞ Set of interpolation nodes: $\mathcal{N} = \{\mathbf{p}_1, \dots, \mathbf{p}_h\} \subset \overline{\Omega}$, $N = \dim S_p^0(\mathcal{M})$.

Based on these interpolation nodes we define a general **nodal Lagrangian interpolation operator** (which agrees with I_1 from Def. 3.3.2.1 for $p = 1$):

$$I_p : \begin{cases} C^0(\overline{\Omega}) & \mapsto S_p^0(\mathcal{M}) \\ u & \mapsto I_p(u) := \sum_{l=1}^N u(\mathbf{p}_l) b_h^l \end{cases}, \quad (3.3.5.3)$$

where b_h^l are the nodal basis functions. Thanks to their cardinal basis property with respect to \mathcal{N} we easily infer

$$(2.6.1.4) \Rightarrow I_p(u)(\mathbf{p}_l) = u(\mathbf{p}_l), \quad l = 1, \dots, N \quad (\text{Interpolation property!}).$$

By virtue of the location of the interpolation nodes, see Ex. 2.6.1.2, Ex. 2.6.1.7, and Fig. 119, the nodal interpolation operators are **purely local**:

$$\forall K \in \mathcal{M}: \quad |I_p u|_K = \sum_{i=1}^Q u(\mathbf{q}_i^K) b_K^i, \quad (3.3.5.4)$$

$\mathbf{q}_i^K, i = 1, \dots, Q \hat{=} \text{local interpolation nodes in cell } K \in \mathcal{M}$, see Ex. 2.6.1.2, Ex. 2.6.1.7, and Fig. 119,
 $b_i^K, i = 1, \dots, Q \hat{=} \text{local shape functions: } b_i^K(\mathbf{q}_j^K) = \delta_{ij}$.

In LEHRFEM++ the nodal interpolation operators I_p can be realised based on the function

```

template <typename SCALAR, typename MF,
        typename SELECTOR = base::PredicateTrue>
Eigen::Matrix<SCALAR, Eigen::Dynamic, 1>
NodalProjection(const UniformScalarFESpace<SCALAR> &fe_space,
                  MF &&u,
                  SELECTOR &&pred = base::PredicateTrue{});

```

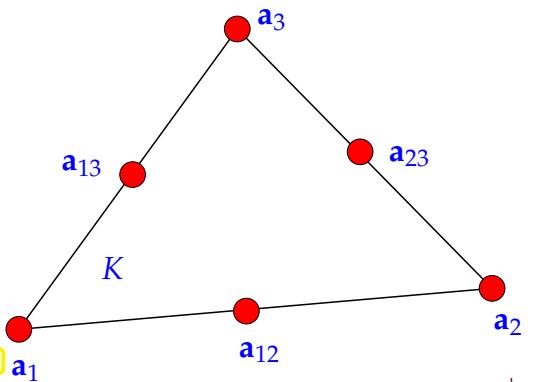
The argument u has to be given as an object compatible with **If::mesh::utils::MeshFunctionGlobal** and the resulting finite function is returned in the form of its basis expansion coefficient vector. \square

EXAMPLE 3.3.5.5 (Piecewise quadratic interpolation \rightarrow Ex. 2.6.1.2)

For a triangle $K = \text{convex}\{\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3\}$ and $p = 2$ the piecewise quadratic interpolation operator on K is given by

$$\begin{aligned} l_2 u|_K = & - \sum_{i=1}^3 \lambda_i (1 - 2\lambda_i) u(\mathbf{a}^i) \\ & + \sum_{1 \leq i < j \leq 3} 4\lambda_i \lambda_j u\left(\frac{1}{2}(\mathbf{a}^i + \mathbf{a}^j)\right). \end{aligned}$$

local shape functions, see (2.6.1.6)
Fig. 220



The following theorem summarizes best approximation results for **affine equivalent** Lagrangian FE spaces $S_p^0(\mathcal{M})$ (\rightarrow Section 2.6) on mesh \mathcal{M} of a bounded polygonal/polyhedral domain $\Omega \subset \mathbb{R}^d$. It is the result of many years of research in approximation theory, see [Sch98, Sect. 3.3], [BS87].

Theorem 3.3.5.6. Best approximation error estimates for Lagrangian finite elements

Let $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, be a bounded polygonal/polyhedral domain equipped with a mesh \mathcal{M} consisting of simplices or parallelepipeds. Then, for each $k \in \mathbb{N}$, there is a constant $C > 0$ depending only on k and the shape regularity measure $\rho_{\mathcal{M}}$ such that

$$\inf_{v_h \in S_p^0(\mathcal{M})} \|u - v_h\|_{H^1(\Omega)} \leq C \left(\frac{h_{\mathcal{M}}}{p} \right)^{\min\{p+1,k\}-1} \|u\|_{H^k(\Omega)} \quad \forall u \in H^k(\Omega). \quad (3.3.5.7)$$

This theorem is a typical example of finite element analysis results that you can find in literature. It is important to know what kind of information can be gleaned from statements like that of Thm. 3.3.5.6.

Remark 3.3.5.8 (“Generic constants”) A statement like (3.3.5.7) is typical of a priori error estimates in the numerical analysis literature, which often come in the form

$$\|u - u_h\|_X \leq C \cdot \text{“discretization parameter”} \cdot \|u\|_Y,$$

where

- ◆ $C > 0$ is not specified precisely or only claimed to exist (“there is”, though, in principle, they could be computed),
- ◆ C must neither depend on the exact solution u nor the discrete solution u_h ,
- ◆ the possible dependence of C on problem parameters or discretization parameters has to be stated unequivocally.

Such constants $C > 0$ are known as **generic constants**. Customarily, different generic constants are even denoted by the same symbol (“ C ” is most common). The use of generic constants is an alternative to the Landau- \mathcal{O} notation (3.2.2.2). \square

§3.3.5.9 (Nature of a priori estimates) → Section 3.2.2 In combination with Cea’s lemma (Thm. 3.1.3.7) Thm. 3.3.5.6 implies a priori estimates of the energy norm of the finite element Galerkin discretization error (see also Rem. 3.3.2.22) of the form

$$\|u - u_h\|_a \leq C \left(\frac{h_M}{p} \right)^{\min\{p+1,k\}-1} \|u\|_{H^k(\Omega)}, \quad (3.3.5.10)$$

where u is the exact solution of the discretized 2nd-order elliptic boundary value problem.



(3.3.5.10) does not give concrete information about $\|u - u_h\|_a$, because

- ◆ we do not know the value of the “generic constant” $C > 0$, see Rem. 3.3.5.8,
- ◆ as u is unknown, a bound for $\|u\|_{H^k(\Omega)}$ may not be available.

A priori error estimates like (3.3.5.10) exhibit only the **trend** of the (norm of) the discretization error as discretization parameters h_M (mesh width), p (polynomial degree) are varied.

Supplement 3.3.5.11. The estimate of Thm. 3.3.5.6 is **sharp**: the powers of h_M and p cannot be increased. \square

§3.3.5.12 (The message of asymptotic a priori convergence estimates) What questions can Thm. 3.3.5.6 and (3.3.5.10) answer? What do they tell us about the accuracy and **efficiency** of a Lagrangian finite element Galerkin discretization of a 2nd-order elliptic BVP? Closely related discussions have been presented for numerical quadrature, see [Hip19, ??], and higher order single step methods for initial value problems from ODEs, see [Hip19, ??]. You are advised to review these passages in order to understand the parallels.

Question 3.3.5.13. *What computational effort buys us what error (measured in energy norm)?*

Bad luck (\rightarrow § 3.3.5.9): actual error norm remains elusive! Therefore, rephrase the question so that it fits the available information about the effect of changing discretization parameters on the error:

Question 3.3.5.14. *What increase in computational effort buys us a prescribed decrease of the (energy norm of the) error?*

The answer to this question offers an a priori gauge of the **asymptotic efficiency** of a discretization method.

Convention: computational effort \approx number of unknowns $N = \dim \mathcal{S}_p^0(\mathcal{M})$ (**problem size**)

§3.3.5.15 (The price of a finite element space) Let us consider a family \mathbf{M} of simplicial meshes of the computational domain $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, created by *global* regular refinement of a single initial mesh, that is, we focus on ***h*-refinement**.

Important specimens of such families of meshes are provided by sequences of simplicial meshes created by global regular refinement (\rightarrow Ex. 3.1.4.3). This refinement rule has distinct benefits:

- ◆ it avoids greater distortion of “child cells” w.r.t. their parents,
- ◆ it spawns meshes with fairly uniform size h_K of cells.

A mathematical way to express these insights:

$$\begin{aligned} \text{uniform shape-regularity: } & \exists C > 0: \quad \rho_{\mathcal{M}} \leq C, \\ \text{local quasi-uniformity } & \exists C > 0: \quad \max\{h_K/h_{K'}, K, K' \in \mathcal{M}\} \leq C, \end{aligned}$$

Now, for meshes $\in \mathbb{M}$, we investigate “ N -dependence”, $N = \dim \mathcal{S}_p^0(\mathcal{M})$, of energy norm of finite element discretization error:

$$\begin{array}{l} \text{Counting argument} \quad N = \dim \mathcal{S}_p^0(\mathcal{M}) \approx p^d h_{\mathcal{M}}^{-d} \Rightarrow \frac{h_{\mathcal{M}}}{p} \approx N^{-1/d} \\ \text{dimensions of local spaces, Lemma 2.5.2.5} \quad \sim \#\mathcal{M} \sim \#\mathcal{V}(\mathcal{M}), \mathcal{E}(\mathcal{M}) \text{ etc.} \end{array} \quad (3.3.5.16)$$

Notation: $\approx \hat{=}$ uniform equivalence on the set \mathbb{M} , that is, each side can be bounded by a constant times the other, and the constants can be chosen independently of the mesh $\mathcal{M} \in \mathbb{M}$

§3.3.5.17 (Dimensions of Lagrangian finite element spaces on triangular meshes) We consider planar meshes, $d = 2$. For triangular meshes \mathcal{M} , by Lemma 2.5.2.5 and simply counting global shape functions we find for Lagrangian finite element spaces:

$$\dim \mathcal{S}_p^0(\mathcal{M}) = \#\{\text{nodes}(\mathcal{M})\} + \#\{\text{edges}(\mathcal{M})\} (p-1) + \#\mathcal{M} \frac{1}{2}(p-1)(p-2).$$

↑
1 basis function per vertex ↑
 $p-1$ basis functions per edge ↑
 $\frac{1}{2}(p-1)(p-2)$ “interior” basis functions

We continue with geometric considerations based on the fact that for a triangle K its shape regularity measure ρ_K (\rightarrow Def. 3.3.2.20) directly controls the smallest angle. We find that the number of triangles sharing a vertex can be bounded in terms of $\rho_{\mathcal{M}}$, because $\rho_{\mathcal{M}}$ implies a lower bound for the smallest angles of the triangular cells.

$$\exists C = C(\rho_{\mathcal{M}}): \quad \#\{K_i \in \mathcal{M}: \bar{K}_i \cap \bar{K}_j \neq \emptyset\} \leq C \quad (i = 1, 2, \dots, \#\mathcal{M}). \quad (3.3.5.18)$$

If every vertex belongs only to a small number of triangles, the number $\#\{\text{nodes}(\mathcal{M})\}$ can be bounded by $C \cdot \#\mathcal{M}$, where $C > 0$ will depend on $\rho_{\mathcal{M}}$ only. The same applies to the edges.

► $\#\{\text{nodes}(\mathcal{M})\}, \#\{\text{edges}(\mathcal{M})\} \approx \#\mathcal{M}. \quad (3.3.5.19)$

► ($d = 2$) $\dim \mathcal{S}_p^0(\mathcal{M}) \approx (\#\mathcal{M})p^2, \quad (3.3.5.20)$

with constants hidden in \approx depending on $\rho_{\mathcal{M}}$ only.

Now, we merge (3.3.5.7) and (3.3.5.16):

$$u \in H^k(\Omega) \xrightarrow{\text{Thm. 3.3.5.6}} \inf_{v_h \in \mathcal{S}_p^0(\mathcal{M})} \|u - v_h\|_{H^1(\Omega)} \leq CN^{-\frac{\min\{p,k-1\}}{d}} \|u\|_{H^k(\Omega)}, \quad (3.3.5.21)$$

with $C > 0$ depending *only* on d, k , and $\rho_{\mathcal{M}}$.

Convergence of best approximation error for Lagrangian finite elements

(3.3.5.21) \Rightarrow Energy norm of the discretization error features **algebraic convergence**
 $(\rightarrow \text{Def. 3.2.2.1})$ in the problem size (= number of unknowns) with a
rate $= \frac{\min\{p, k-1\}}{d}$.

We observe that

- ◆ the rate of convergence is limited by the polynomial degree p of the Lagrangian FEM,
- ◆ the rate of convergence is limited by the smoothness of the exact solution u , measured by means of the Sobolev index k , see Section 3.3.3,
- ◆ the rate of convergence will be worse for $d = 3$ than for $d = 2$, the effect being more pronounced for small k or p .

§3.3.5.23 (Asymptotic efficiency of Lagrangian finite elements) Now we answer Question 3.3.5.14 (“What increase in computational effort buys us a prescribed decrease of the (energy norm of the) error?”):

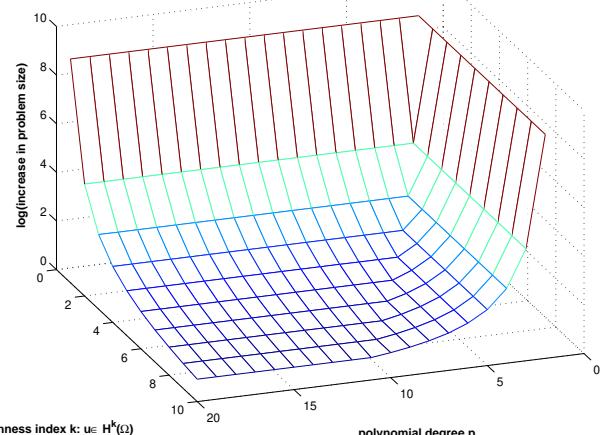
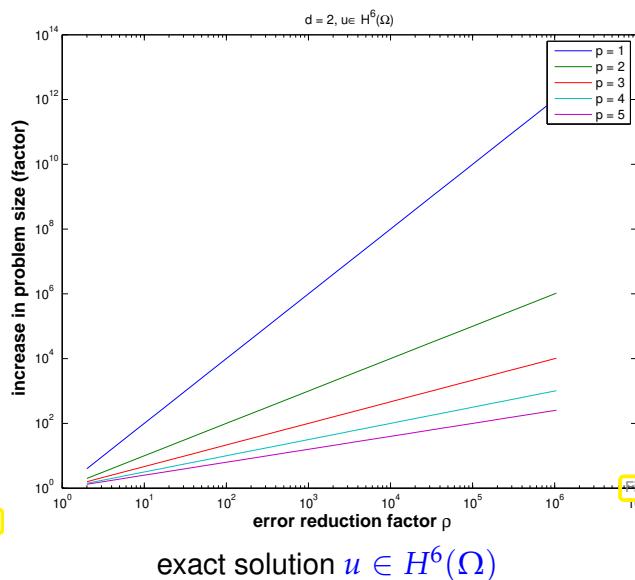
Assumption: a priori error estimate (3.3.5.21) is *sharp*

$$\exists C = C(u, \dots) > 0: \text{error norm}(N) \approx CN^{-\frac{\min\{p, k-1\}}{d}} \quad \forall M \in \mathbb{M}.$$

$$\blacktriangleright \frac{\text{error norm}(N_1)}{\text{error norm}(N_2)} \approx \left(\frac{N_1}{N_2}\right)^{-\frac{\min\{p, k-1\}}{d}}.$$

reduction of (the energy norm of)
the error by a factor $\rho > 1$ requires

increase of the problem size
by factor $\rho^{\frac{d}{\min\{p, k-1\}}}$



Discussion:

Solution $u \in H^k(\Omega)$ \Rightarrow optimal asymptotic efficiency for $p = k - 1$

(Here, $u \in H^k(\Omega)$ is supposed to be sharp in the sense that we cannot take for granted $u \in H^{k+1}(\Omega)$.)

Remark 3.3.5.24 (General asymptotic estimates) Recall from Section 3.2.2 that convergence is an asymptotic notion that describes the behavior of error norms as a discretization parameter, here we use the number N of “degrees of freedoms/unknowns”, tends to a limit value.

Now we deduce **asymptotic** estimates for the best approximation errors from Thm. 3.3.5.6, and (3.3.5.21), in particular, for the limit $N \rightarrow \infty$, where N is the dimension of the finite element space:

For the exact solution u we assume a certain smoothness expressed by its Sobolev regularity:

$$u \in H^k(\Omega) \text{ for some } k \in \mathbb{N}.$$

We discuss the meaning of (3.3.5.21) for different refinement settings with emphasis on asymptotic rates of algebraic convergence:

- h-refinement: p fixed, $h_M \rightarrow 0$ for $M \in \mathbb{M}$:

$$(3.3.5.21) \Rightarrow \text{algebraic convergence w.r.t. } N$$

☞ $p \leq k - 1$ ➔

$$\inf_{v_h \in S_p^0(M)} \|u - v_h\|_1 = O(N^{-p/d}) \quad (3.3.5.25)$$

Here the polynomial degree of the Lagrangian finite elements limits the rate of algebraic convergence.

☞ $k \leq p + 1$ ➔

$$\inf_{v_h \in S_p^0(M)} \|u - v_h\|_1 = O(N^{-(k-1)/d}) \quad (3.3.5.26)$$

Here, the smoothness (measured in the Sobolev scale) is the limiting factor for the rate of algebraic convergence.

Note: for very smooth solution u , i.e. $k \gg 1$, polynomial degree p limits speed of convergence

- p-refinement: $M \in \mathbb{M}$ fixed, $p \rightarrow \infty$:

☞ p large ➔

$$\inf_{v_h \in S_p^0(M)} \|u - v_h\|_1 = O(N^{-(k-1)/d}) \quad (3.3.5.27)$$

Note: arbitrarily fast (**super**-)algebraic convergence for very smooth solutions $u \in C^\infty(\bar{\Omega})$.
 (However, the exponential convergence observed in Exp. 3.2.3.11 is not captured by the approximation error estimates of Thm. 3.3.5.6.)

Remark 3.3.5.28 (Curse of dimension) Note that the dimension d of the domain affects the asymptotic rates of algebraic convergence in (3.3.5.25), (3.3.5.26), and (3.3.5.27) through a factor d^{-1} ! Thus, for large d convergence in terms of N becomes very slow. This is then bane of the finite element discretization of boundary value problems in high dimensions and known as the **curse of dimension**. Such high-dimensional boundary value problems are common in areas like quantum physics, probability, and in computational finance. Special Galerkin methods have been developed for them.

Review question(s) 3.3.5.29 (Convergence of finite element solutions)

(Q3.3.5.29.A) Let I_p denote nodal interpolation onto $\mathcal{S}_p^0(\mathcal{M})$, $p \in \mathbb{N}$, where \mathcal{M} is a mesh of the 1D domain $\Omega =]0, 1[$. Which smoothness is required of the function $u : [0, 1] \rightarrow \mathbb{R}$ such that we can expect the asymptotic interpolation error estimate

$$\|u - I_p u\|_{L^\infty(]0, 1[)} = O(h_\mathcal{M}^{p+1}) \quad \text{for } h_\mathcal{M} \rightarrow 0 ?$$

(Q3.3.5.29.B) What is meant by the following statement?

The nodal interpolation operator onto $\mathcal{S}_2^0(\mathcal{M})$, \mathcal{M} a triangular mesh of some polygonal domain Ω , is *purely local*.

(Q3.3.5.29.C) Characterize the set of functions

$$Z := \{v \in H^p(\Omega) : |v|_{H^p(\Omega)} = 0\} .$$

Discuss the implications of your insights for the following theorem.

Theorem 3.3.3.4. Error estimate for piecewise linear interpolation in 2D

Under the assumptions/with notations of Thm. 3.3.2.21

$$\begin{aligned} \|u - I_1 u\|_{L^2(\Omega)} &\leq \sqrt{\frac{3}{8}} h_\mathcal{M}^2 |u|_{H^2(\Omega)}, \\ |u - I_1 u|_{H^1(\Omega)} &\leq \sqrt{\frac{3}{24}} \rho_\mathcal{M} h_\mathcal{M} |u|_{H^2(\Omega)}, \end{aligned} \quad \forall u \in H^2(\Omega) .$$

(Q3.3.5.29.D) If \mathcal{M}' has been created by regular refinement of a triangular mesh \mathcal{M} , how are shape-regularity measures $\rho_\mathcal{M}$ and $\rho_{\mathcal{M}'}$ related?

Definition 3.3.2.20. Shape regularity measure

For a simplex $K \in \mathbb{R}^d$ we define its **shape regularity measure** as the ratio

$$\rho_K := h_K^d : |K| , \quad h_K := \text{diam}(K) ,$$

and the shape regularity measure of a simplicial mesh $\mathcal{M} = \{K\}$ as

$$\rho_\mathcal{M} := \max_{K \in \mathcal{M}} \rho_K .$$

(Q3.3.5.29.E) For which exponents $\alpha > 0$ does the function $x \mapsto x^\alpha$ belong to the Sobolev space $H^m(]0, 1[)$, $m \in \mathbb{N}$?

Hint. Since the function belongs to $C^\infty(]0, 1[)$, it is sufficient to show that the improper integral defining its $H^m(]0, 1[)$ -norm has a finite value.

(Q3.3.5.29.F) Suppose it is known that the solution u of a scalar 2nd-order elliptic boundary value problem belongs to $H^2(\Omega)$, but fails to be contained in $H^3(\Omega)$.

1. Describe the convergence of the $H^1(\Omega)$ -norm of the discretization error one can expect from a finite element Galerkin discretization by means of degree p , $p \in \mathbb{N}$, Lagrangian finite elements on a sequence of meshes obtained by uniform regular refinement.
2. Which convergence of $\|u - u_h\|_{H^1(\Omega)}$ in terms of polynomial degree p will probably be observed for finite element solutions $u_h \in \mathcal{S}_p^0(\mathcal{M})$, \mathcal{M} fixed, and increasing p ?

(Q3.3.5.29.G) Appealing to Ex. 1.2.3.45 explain why for a bounded polygonal domain $\Omega \subset \mathbb{R}^2$ and $I_1 : C^0(\overline{\Omega}) \rightarrow \mathcal{S}_1^0(\mathcal{M})$ denoting the nodal interpolation operator according to Def. 3.3.2.1 the *interpolation* error estimate

$$\|u - I_1 u\|_{L^2(\Omega)} \leq Ch_{\mathcal{M}} \|u\|_{H^1(\Omega)} \quad \forall u \in H^1(\Omega) ,$$

with $C > 0$ depending only on the shape regularity measure $\rho_{\mathcal{M}}$ of a triangular mesh \mathcal{M} **cannot** be true.

(Q3.3.5.29.H) Explain the following statement concerning the asymptotic convergence of finite-element Galerkin solutions $u_{\ell} \in \mathcal{S}_p^0(\mathcal{M}_{\ell})$ of a second-order elliptic boundary value problem on $\Omega \subset \mathbb{R}^2$ obtained by means of degree- p Lagrangian finite elements on sequences $(\mathcal{M}_{\ell})_{\ell \in \mathbb{N}_0}$ of triangular meshes generated by uniform regular refinement.

The rate of convergence of $\|u - u_{\ell}\|_{H^1(\Omega)}$ for $\ell \rightarrow \infty$ is limited by both the smoothness of u and the degree p .

Theorem 3.3.5.6. Best approximation error estimates for Lagrangian finite elements

Let $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, be a bounded polygonal/polyhedral domain equipped with a mesh \mathcal{M} consisting of simplices or parallelepipeds. Then, for each $k \in \mathbb{N}$, there is a constant $C > 0$ depending **only** on k and the shape regularity measure $\rho_{\mathcal{M}}$ such that

$$\inf_{v_h \in \mathcal{S}_p^0(\mathcal{M})} \|u - v_h\|_{H^1(\Omega)} \leq C \left(\frac{h_{\mathcal{M}}}{p} \right)^{\min\{p+1,k\}-1} \|u\|_{H^k(\Omega)} \quad \forall u \in H^k(\Omega) . \quad (3.3.5.7)$$

(Q3.3.5.29.I) Let $u_p \in \mathcal{S}_p^0(\mathcal{M})$, $p \in \mathbb{N}$, be finite-element Galerkin solutions of a second-order elliptic BVP posed on a domain $\Omega \subset \mathbb{R}^2$. Assume that

$$|u - u_p|_{H^1(\Omega)} = O(\exp(-cN_p^{1/2})) , \quad N_p := \dim \mathcal{S}_p^0(\mathcal{M}) , \quad \text{for } p \rightarrow \infty . \quad (3.3.5.30)$$

- Which type of convergence is enjoyed by the finite-element method in this case?
- By what factor do you have to increase the computational effort, assumed to be proportional to N_p , in order to achieve a reduction of the $H^1(\Omega)$ -semi-norm of the discretization error by a factor of 2?

(Q3.3.5.29.J) A student wants to test his finite element code through a numerical experiment on the L-shaped domain $\Omega :=]-1, 1[^2 \setminus [-1, 0]^2$. He solves the 2nd-order scalar elliptic BVP

$$-\Delta u = f \quad \text{in } \Omega , \quad u = g \quad \text{on } \partial\Omega ,$$

with $f \equiv 4$ and $g(x) = x_1^2 + x_2^2$ so that the exact solution will be $u(x) = x_1^2 + x_2^2$.

All experiments rely on a sequence of nested triangular meshes $(\mathcal{M}_k)_{k=1}^5$, obtained by successive global regular refinement. We write $u_{k,p} \in \mathcal{S}_p^0(\mathcal{M}_k)$, $p = 1, 2, 3$, for the finite-element Galerkin solutions.

Sketch a doubly logarithmic plot that gives shows the dependence of $\|u - u_{k,p}\|_{H^1(\Omega)}$ on $\dim \mathcal{S}_p^0(\mathcal{M}_k)$, $k = 1, \dots, 5$, in a qualitatively correct way.

△

3.4 Elliptic Regularity Theory



Video tutorial for Section 3.4: Elliptic Regularity Theory: (21 minutes) [Download link](#), [tablet notes](#)

Crudely speaking, in Section 3.3.5 we saw that the asymptotic behavior of the Lagrangian finite element Galerkin discretization error (for 2nd-order elliptic BVPs) can be predicted provided that

- we use families of meshes, whose cells have rather uniform size and whose shape regularity measure is uniformly bounded,
- we have an *idea about the smoothness of the exact solution u* , that is, we know $u \in H^k(\Omega)$ for a (maximal) k , see Thm. 3.3.5.6.

Knowledge about the mesh can be taken for granted, but

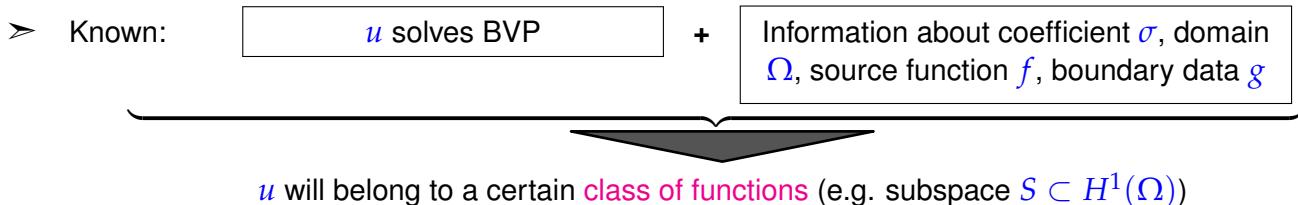
how can we guess the smoothness of the (unknown !) exact solution u ?

A (partial) answer is given in this section.

Focus: Scalar 2nd-order elliptic BVP with homogeneous Dirichlet boundary conditions

$$-\operatorname{div}(\sigma(x) \operatorname{grad} u) = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega.$$

To begin with, we summarize the available information:



EXAMPLE 3.4.0.1 (Elliptic lifting result in 1D) We look at the simplest case $d = 1$, $\Omega =]0, 1[$, coefficient $\sigma \equiv 1$, homogeneous (zero) Dirichlet boundary conditions:

$$u'' = f, \quad u(0) = u(1) = 0.$$

Obvious from Def. 3.3.3.1: $f \in H^k(\Omega) \Rightarrow u \in H^{k+2}(\Omega)$ (a **lifting theorem**)

]

Can this be generalized to higher dimensions $d > 1$?

Partly so:

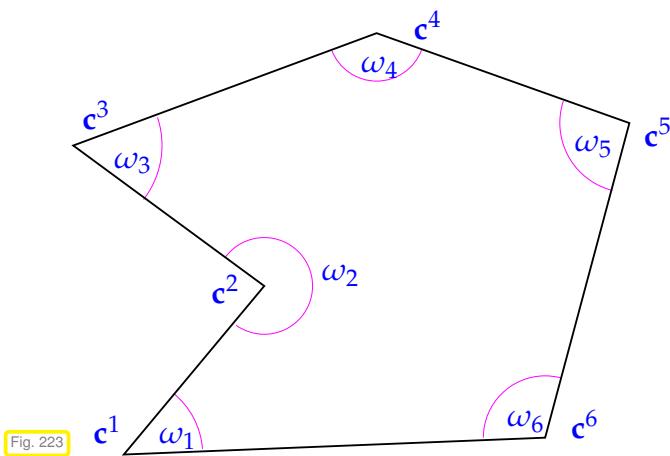
Theorem 3.4.0.2. Smooth elliptic lifting theorem

If $\partial\Omega$ is C^∞ -smooth, ie. possesses a local parameterization by C^∞ -functions, and $\sigma \in C^\infty(\overline{\Omega})$, then, for any $k \in \mathbb{N}$,

$$\begin{aligned} u \in H_0^1(\Omega) \quad \text{and} \quad -\operatorname{div}(\sigma \operatorname{grad} u) \in H^k(\Omega) \\ u \in H^1(\Omega), \quad -\operatorname{div}(\sigma \operatorname{grad} u) \in H^k(\Omega) \quad \text{and} \quad \operatorname{grad} u \cdot \mathbf{n} = 0 \quad \text{on } \partial\Omega \end{aligned} \Rightarrow u \in H^{k+2}(\Omega).$$

In addition, for such u there is $C = C(k, \Omega, \sigma)$ such that

$$\|u\|_{H^{k+2}(\Omega)} \leq C \|\operatorname{div}(\sigma \operatorname{grad} u)\|_{H^k(\Omega)}.$$



What about non-smooth $\partial\Omega$?

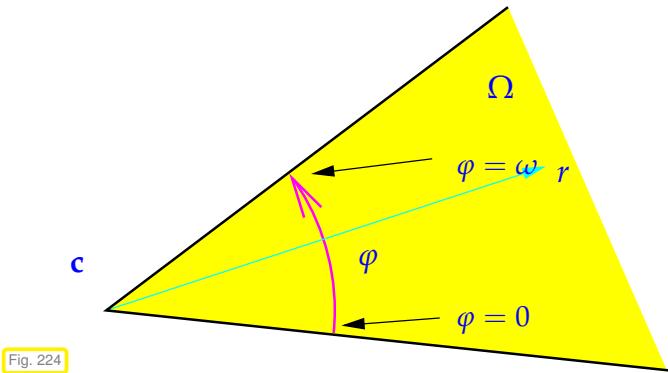
These are very common in engineering applications (“CAD-geometries”, polygonal/polyhedral domains). Most of the numerical examples have been computed on such domains so far.

▷ polygonal domain with corners c^i

How will the corners affect the smoothness of solutions of

$$u \in H_0^1(\Omega) : \Delta u = f \in C^\infty(\overline{\Omega}) ?$$

EXAMPLE 3.4.0.3 (Corner singular functions) This example answers some of the questions asked above by exhibiting locally harmonic functions satisfying homogeneous Dirichlet boundary conditions that, nevertheless, feature a **singularity** at a corner of the domain:



corner singular function

$$u_s(r, \varphi) = r^{\frac{\pi}{\omega}} \sin\left(\frac{\pi}{\omega}\varphi\right), \quad (3.4.0.4)$$

$$r \geq 0, \quad 0 \leq \varphi \leq \omega.$$

(in local polar coordinates)

► $u_s = 0$ on $\partial\Omega$ locally at c !

Straightforward computation (in polar coordinates):

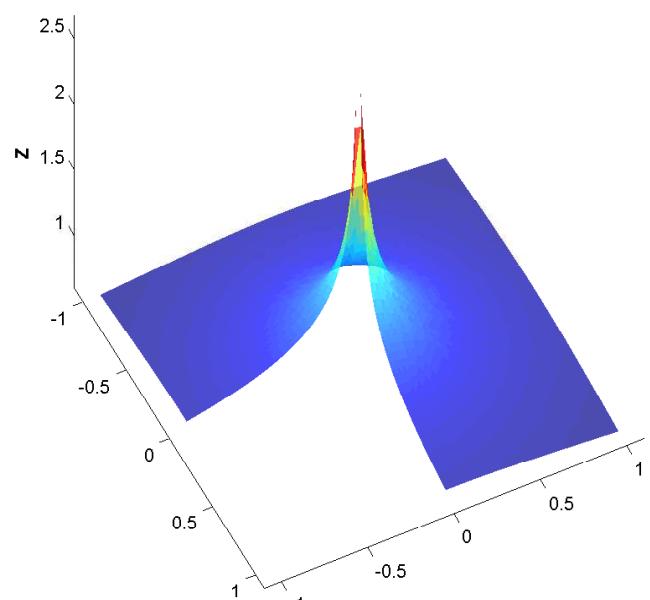
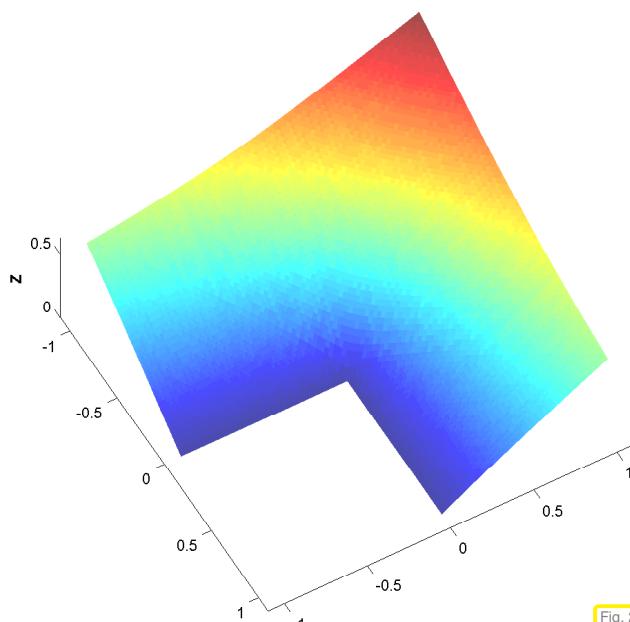
$$\Delta u_s = 0 \quad \text{in } \Omega !$$

To see this recall: Δ in polar coordinates:

$$\Delta u = \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 u}{\partial \varphi^2}. \quad (3.4.0.5)$$

$$\begin{aligned} \stackrel{(3.4.0.4)}{\Rightarrow} \Delta u_s(r, \varphi) &= \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\pi}{\omega} r^{\frac{\pi}{\omega}-1} \sin\left(\frac{\pi}{\omega}\varphi\right) \right) + \frac{1}{r^2} r^{\frac{\pi}{\omega}} \frac{\partial}{\partial \varphi} \cos\left(\frac{\pi}{\omega}\varphi\right) \frac{\pi}{\omega} \\ &= \left(\frac{\pi}{\omega}\right)^2 r^{\frac{\pi}{\omega}-2} \sin\left(\frac{\pi}{\omega}\varphi\right) - \left(\frac{\pi}{\omega}\right)^2 r^{\frac{\pi}{\omega}-2} \sin\left(\frac{\pi}{\omega}\varphi\right) = 0. \end{aligned}$$

What is “singular” about these functions? Plot them for $\omega = \frac{3\pi}{2}$, cf. Exp. 3.2.3.10



Recall gradient (1.2.3.48) in polar coordinates

$$\begin{aligned} \mathbf{grad} u &= \frac{\partial u}{\partial r} \mathbf{e}_r + \frac{1}{r} \frac{\partial u}{\partial \varphi} \mathbf{e}_\varphi . \\ \xrightarrow{(3.4.0.4)} \quad \mathbf{grad} u_s(r, \varphi) &= \frac{\pi}{\omega} r^{\frac{\pi}{\omega}-1} (\sin(\frac{\pi}{\omega}\varphi) \mathbf{e}_r + \cos(\frac{\pi}{\omega}\varphi)) \mathbf{e}_\varphi . \end{aligned} \quad (1.2.3.48)$$

$$\omega > \pi \text{ ("re-entrant corner")} \implies \text{"}\mathbf{grad} u_s(0) = \infty\text{"}$$

How does this “blow-up” of the gradient affect the **Sobolev regularity** (that is, the smoothness as expressed through “ $u_s \in H^k(\Omega)$ ”) of the corner singular function u_s ?

We try to compute $|u|_{H^2(D)}$, with (in polar coordinates, see Fig. 224)

$$D := \{(r, \varphi) : 0 < r < 1, 0 < \varphi < \omega\} .$$

By tedious computations we find

$$\begin{aligned} \omega > \pi &\Rightarrow \int_D \|D^2 u_s(r, \varphi)\|_F^2 r d(r, \varphi) = \infty . \\ \xrightarrow{\text{Def. 3.3.3.1}} \quad \left\{ \begin{array}{l} \omega > \pi \Rightarrow u_s \notin H^2(D) \end{array} \right\} . \end{aligned}$$

Bad news: With the exception of “concocted/manufactured” examples,



corner singular functions like (3.4.0.4) will be present in the solution of linear scalar 2nd-order elliptic BVP on polygonal domains!

The meaning of “being present” is elucidated in the following theorem:

Theorem 3.4.0.6. Corner singular function decomposition

Let $\Omega \subset \mathbb{R}^2$ be a polygon with J corners \mathbf{c}^i . Denote the polar coordinates in the corner \mathbf{c}^i by (r_i, φ_i) and the inner angle at the corner \mathbf{c}^i by ω_i . Additionally, let $f \in H^l(\Omega)$ with $l \in \mathbb{N}_0$ and $l \neq \lambda_{ik} - 1$, where the λ_{ik} are given by the *singular exponents*

$$\lambda_{ik} = \frac{k\pi}{\omega_i} \quad \text{for } k \in \mathbb{N}. \quad (3.4.0.7)$$

Then $u \in H_0^1(\Omega)$ with $-\Delta u = f$ in Ω can be decomposed

$$u = u^0 + \sum_{i=1}^J \psi(r_i) \sum_{\lambda_{ik} < l+1} \kappa_{ik} s_{ik}(r_i, \varphi_i), \quad \kappa_{ik} \in \mathbb{R}, \quad (3.4.0.8)$$

with *regular part* $u^0 \in H^{l+2}(\Omega)$, *cut-off functions* $\psi \in C^\infty(\mathbb{R}^+)$ ($\psi \equiv 1$ in a neighborhood of 0), and *corner singular functions*

$$\begin{aligned} \lambda_{ik} \notin \mathbb{N}: \quad & s_{ik}(r, \varphi) = r^{\lambda_{ik}} \sin(\lambda_{ik}\varphi), \\ \lambda_{ik} \in \mathbb{N}: \quad & s_{ik}(r, \varphi) = r^{\lambda_{ik}} (\ln r) \sin(\lambda_{ik}\varphi). \end{aligned} \quad (3.4.0.9)$$

$\Omega \subset \mathbb{R}^2$ has *re-entrant corners* \Rightarrow if u solves $\Delta u = f$ in Ω , $u = 0$ on $\partial\Omega$, then $u \notin H^2(\Omega)$ in general.

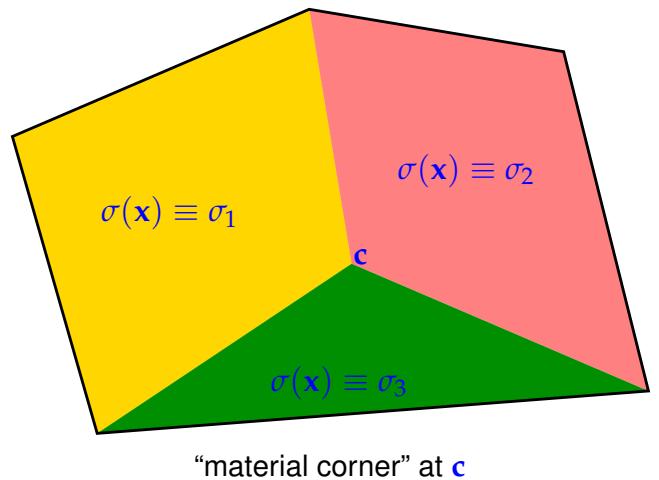
Theorem 3.4.0.10. Elliptic lifting theorem on convex domains [Gri85, Thm. 3.2.1.2]

If $\Omega \subset \mathbb{R}^d$ convex, $u \in H_0^1(\Omega)$, $\Delta u \in L^2(\Omega)$ \Rightarrow $u \in H^2(\Omega)$.

Terminology: if conclusion of Thm. 3.4.0.10 true \rightarrow Dirichlet problem *2-regular*.

Similar lifting theorems also hold for Neumann BVPs, BVPs with *smooth* coefficients.

§3.4.0.11 (Causes for non-smoothness of solutions of elliptic BVPs) Causes for poor Sobolev regularity of solution u of BVPs for $-\operatorname{div}(\sigma(x) \operatorname{grad} u) = f$:



- Corners of $\partial\Omega$, see above
- Discontinuities of σ
→ singular functions at “material corners”
- Mixed boundary conditions
- Non-smooth source function f

Review question(s) 3.4.0.12 (Elliptic regularity)

(Q3.4.0.12.A) [Elliptic regularity in 1D] Consider the one-dimensional boundary value problem $u'' = f$ in $]0, 1[$, $u(0) = u(1) = 0$. Which Sobolev regularity will the weak solution $u \in H_0^1(]0, 1[)$ possess, if f is piecewise smooth with a single discontinuity?

(Q3.4.0.12.B) [Corner singular function (I)] How does increasing $\alpha > 0$ in the polar-coordinate formula

$$u_s(r, \varphi) = r^\alpha \sin(\alpha\varphi), \quad r \geq 0, \quad 0 \leq \varphi \leq 2\pi, \quad (3.4.0.4)$$

affect the regularity (measured in the Sobolev scale) of the function u_s ? For which values of α do we get $u_s \in C^\infty(\mathbb{R}^2)$?

(Q3.4.0.12.C) [Corner singular function (II)] Compute the gradient of the corner singular function ($0 < \omega < 2\pi$)

$$u_s(r, \varphi) = r^{\frac{\pi}{\omega}} \sin\left(\frac{\pi}{\omega}\varphi\right), \quad r \geq 0, \quad 0 \leq \varphi \leq \omega, \quad (3.4.0.4)$$

in Cartesian coordinates.

Hint. The gradient in polar coordinates is

$$\mathbf{grad} u = \frac{\partial u}{\partial r} \mathbf{e}_r + \frac{1}{r} \frac{\partial u}{\partial \varphi} \mathbf{e}_\varphi, \quad \mathbf{e}_r := \begin{bmatrix} \cos \varphi \\ \sin \varphi \end{bmatrix}, \quad \mathbf{e}_\varphi := \begin{bmatrix} -\sin \varphi \\ \cos \varphi \end{bmatrix}. \quad (1.2.3.48)$$

(Q3.4.0.12.D) [Neumann corner singular functions] On the wedge-shaped domain

$$\Omega := \left\{ \begin{bmatrix} r \cos \varphi \\ r \sin \varphi \end{bmatrix} \in \mathbb{R}^2 : 0 < \varphi < \omega \right\}, \quad 0 < \omega < 2\pi,$$

exhibit a corner singular function u_n that satisfies $\Delta u_n = 0$ and $\mathbf{grad} u_n \cdot \mathbf{n} = 0$ on $\partial\Omega$.

Hint. In the formula for the Dirichlet corner singular functions replace \sin with \cos . Also use the formula for the Laplacian on polar coordinates:

$$\Delta u = \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 u}{\partial \varphi^2}. \quad (3.4.0.5)$$

(Q3.4.0.12.E) [Smooth solutions despite re-entrant corners] Give an example for Dirichlet data $g : \partial\Omega \rightarrow \mathbb{R}$ such that the solution of the Dirichlet BVP

$$\begin{aligned} \Delta u &= 0 \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega, \\ \Omega &:= \{x \in \mathbb{R}^2 : \|x\| < 1, x_1 > 0 \text{ or } x_2 > 0\}, \end{aligned}$$

is non-zero, but contained in $C^\infty(\overline{\Omega})$.



3.5 Variational Crimes



Video tutorial for Section 3.5: Variational Crimes: (30 minutes) [Download link](#), [tablet notes](#)

We retain the abstract setting outlined in Section 3.1 and consider the linear variational problem

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0, \quad (2.2.0.2)$$

and its Galerkin discretization based on the finite-dimensional trial and test space $V_{0,h} \subset V_0$.

The term “**variational crime**” means that instead of solving (exact) discrete (linear) variational problem

$$u_h \in V_{0,h}: \quad a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h}, \quad (2.2.1.1)$$

we solve the **perturbed variational problem**

$$\tilde{u}_h \in V_{0,h}: \quad a_h(\tilde{u}_h, v_h) = \ell_h(v_h) \quad \forall v_h \in V_{0,h}, \quad (3.5.0.1)$$

with a modified bilinear form $a_h : V_{0,h} \times V_{0,h} \rightarrow \mathbb{R}$ and linear form $\ell_h : V_{0,h} \rightarrow \mathbb{R}$.

This causes a perturbation of Galerkin solution u_h and we end up with a perturbed solution $\tilde{u}_h \in V_{0,h}$.

In practice, the approximate building blocks $a_h(\cdot, \cdot) \approx a(\cdot, \cdot)$, $f_h(\cdot) \approx f(\cdot)$ are usually due to

- the use of numerical quadrature → Section 2.7.5,
- an approximation of the boundary $\partial\Omega$ → Section 2.8.4.

We are all sinners! Variational crimes are *inevitable* in practical FEM, recall Rem. 2.1.2.5!

Which “variational petty crimes” can be tolerated?

Guideline for acceptable variational crimes

Variational crimes must not affect (type and rate) of asymptotic convergence!

This requirement must be met for *all* boundary value problems the finite element methods has been designed to solve, in particular, for problems with smooth solutions, for which maximal rates of algebraic convergence can be achieved (→ Rem. 3.3.5.24).

Hence, when probing the impact of variational crimes in a numerical experiment, always choose test problems with smooth solutions.

Supplement 3.5.0.3 (Strang's lemmas) Let's get a glimpse of finite-element numerical analysis techniques tackling variational crimes and of the results often called Strang's lemma, named after the eminent applied mathematician Gilbert Strang, author of one of the first textbooks on FEM [[SF73](#)].

The setting is as follows:

- ◆ V_0 is a Hilbert space with norm $\|\cdot\|$,

- ◆ $a : V_0 \times V_0 \rightarrow \mathbb{R}$ is a continuous bilinear form satisfying

$$\exists \gamma > 0: |a(v, v)| \geq \gamma \|v\|^2 \quad \forall v \in V_0, \quad (3.5.0.4a)$$

$$\exists C > 0: |a(u, v)| \leq C \|u\| \|v\| \quad \forall u, v \in V_0, \quad (3.5.0.4b)$$

- ◆ $\ell : V_0 \rightarrow \mathbb{R}$ is a continuous linear form.

Then the linear variational problem

$$u \in V_0: a(u, v) = \ell(v) \quad \forall v \in V_0, \quad (2.2.0.2)$$

has a unique solution $u \in V_0$.

Galerkin discretization is based on the finite-dimensional trial and test space $V_{0,h} \subset V_0$ and the perturbed variational problem

$$\tilde{u}_h \in V_{0,h}: a_h(\tilde{u}_h, v_h) = \ell_h(v_h) \quad \forall v_h \in V_{0,h}. \quad (3.5.0.1)$$

We take for granted

- ◆ that a_h satisfies estimates analogous to (3.5.0.4a):

$$\exists \gamma_h > 0: |a(v_h, v_h)| \geq \gamma_h \|v_h\|^2 \quad \forall v_h \in V_{0,h}, \quad (3.5.0.5a)$$

$$\exists C_h > 0: |a(u_h, v_h)| \leq C_h \|u_h\| \|v_h\| \quad \forall u_h, v_h \in V_{0,h}, \quad (3.5.0.5b)$$

- ◆ $\ell : V_{0,0} \rightarrow \mathbb{R}$ is a continuous linear form.

Theorem 3.5.0.6. Strang's second lemma

Under assumptions (3.5.0.4) and (3.5.0.5) there exists a constant $K > 0$ depending only on the constants in (3.5.0.4) and (3.5.0.5) such that

$$\|u - \tilde{u}_h\| \leq K \left(\inf_{v_h \in V_{0,h}} \|u - v_h\| + \sup_{w_h \in V_{0,h}} \frac{|a_h(u, w_h) - \ell_h(w_h)|}{\|w_h\|} \right), \quad (3.5.0.7)$$

where u/u_h are the solutions of the continuous/discrete variational problems (2.2.0.2)/(3.5.0.1).

Proof. Pick any $v_h \in V_{0,h}$. From (3.5.0.5a) we conclude

$$\begin{aligned} \gamma_h \|\tilde{u}_h - v_h\|^2 &\leq a_h(\tilde{u}_h - v_h, \tilde{u}_h - v_h) \\ &= a_h(u - v_h, \tilde{u}_h - v_h) + (\ell_h(\tilde{u}_h - v_h) - a_h(u, \tilde{u}_h - v_h)). \end{aligned}$$

Using (3.5.0.5b), dividing by $\|u_h - v_h\|$ and replacing $u_h - v_h$ by w_h , we get

$$\gamma_h \|u_h - v_h\| \leq C_h \|u - v_h\| + \frac{|a_h(u, w_h) - \ell_h(w_h)|}{\|w_h\|}.$$

Then the assertion follows by the triangle inequality. \square

We may use $a(u, w_h) - \ell(w_h) = 0$ to rewrite the estimate (3.5.0.7) as

$$\|u - u_h\| \leq K \left(\inf_{v_h \in V_{0,h}} \|u - v_h\| + \sup_{w_h \in V_{0,h}} \frac{|(a - a_h)(u, w_h)|}{\|w_h\|} + \sup_{w_h \in V_{0,h}} \frac{|(\ell - \ell_h)(w_h)|}{\|w_h\|} \right). \quad (3.5.0.8)$$

The two highlighted terms are called **consistency errors**. They can be estimated separately. \square

3.5.1 The Impact of Numerical Quadrature

Model problem: on polygonal/polyhedral $\Omega \subset \mathbb{R}^d$:

$$u \in H_0^1(\Omega): \quad a(u, v) := \int_{\Omega} \sigma(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \ell(v) := \int_{\Omega} f v \, dx. \quad (3.5.1.1)$$

Assumptions:

$$\sigma \text{ satisfies (1.6.0.6), } \sigma \in C^0(\bar{\Omega}), f \in C^0(\bar{\Omega})$$

- Galerkin finite element discretization, $V_h := \mathcal{S}_p^0(\mathcal{M})$ on simplicial mesh \mathcal{M}
- Approximate evaluation of $a(u_h, v_h)$, $\ell(v_h)$ by a fixed stable local numerical quadrature rule (\rightarrow Section 2.7.5)
 - perturbed bilinear form a_h , right hand side f_h (see (3.5.0.1))

Focus: h -refinement (key discretization parameter is the mesh width $h_{\mathcal{M}}$)

EXPERIMENT 3.5.1.2 (Impact of numerical quadrature on finite element discretization error) We consider the 2nd-order elliptic boundary value problem (3.5.1.1) on $\Omega = [0, 1]^2$, with $\sigma \equiv 1$, $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$, $[x, y] \in \Omega$.

- This BVP has the smooth solution $u(x, y) = \sin(\pi x) \sin(\pi y)$.

Details of numerical experiment:

- **Quadratic** Lagrangian FE ($V_h = \mathcal{S}_2^0(\mathcal{M})$) on triangular meshes \mathcal{M} , obtained by regular refinement
- “Exact” evaluation of bilinear form by very high order quadrature
- ℓ_h based on one point quadrature rule (2.7.5.36) **of order 2**

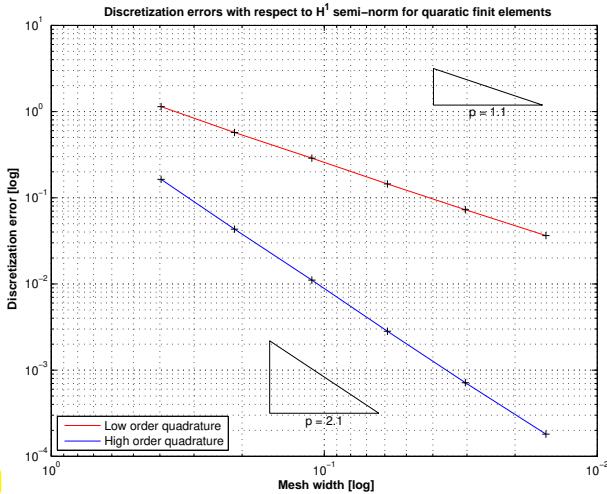
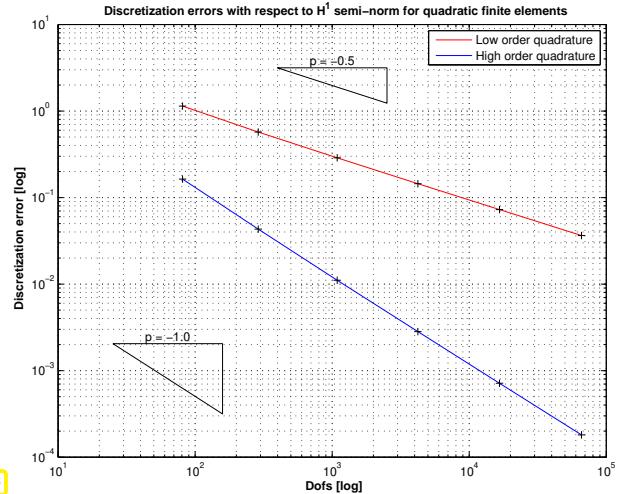


Fig. 227

 $H^1(\Omega)$ -norm of discretization error on unit square (— \leftrightarrow rule (2.7.5.36), — \leftrightarrow rule (2.7.5.37))

Observation: Use of quadrature rule of order 2 \Rightarrow Algebraic rate of convergence (w.r.t. N) drops from $\alpha = 1$ to $\alpha = 1/2$!

Hence, by using the one-point local quadrature rule for the evaluation of the right-hand side linear functional we have committed a **non-admissible variational crime** in this case. \square

Finite element theory [Cia78, Ch. 4, §4.1] tells us that the above guideline can be met, if the local numerical quadrature rule has sufficiently high order. The quantitative results can be condensed into the following rules of thumb:

$\|u - u_h\|_1 = O(h_{\mathcal{M}}^p)$ at best \Rightarrow Quadrature rule of order $2p - 1$ sufficient for right hand side functional ℓ_h .

$\|u - u_h\|_1 = O(h_{\mathcal{M}}^p)$ at best \Rightarrow Quadrature rule of order $2p - 1$ sufficient for bilinear form a_h .

3.5.2 Approximation of the Boundary

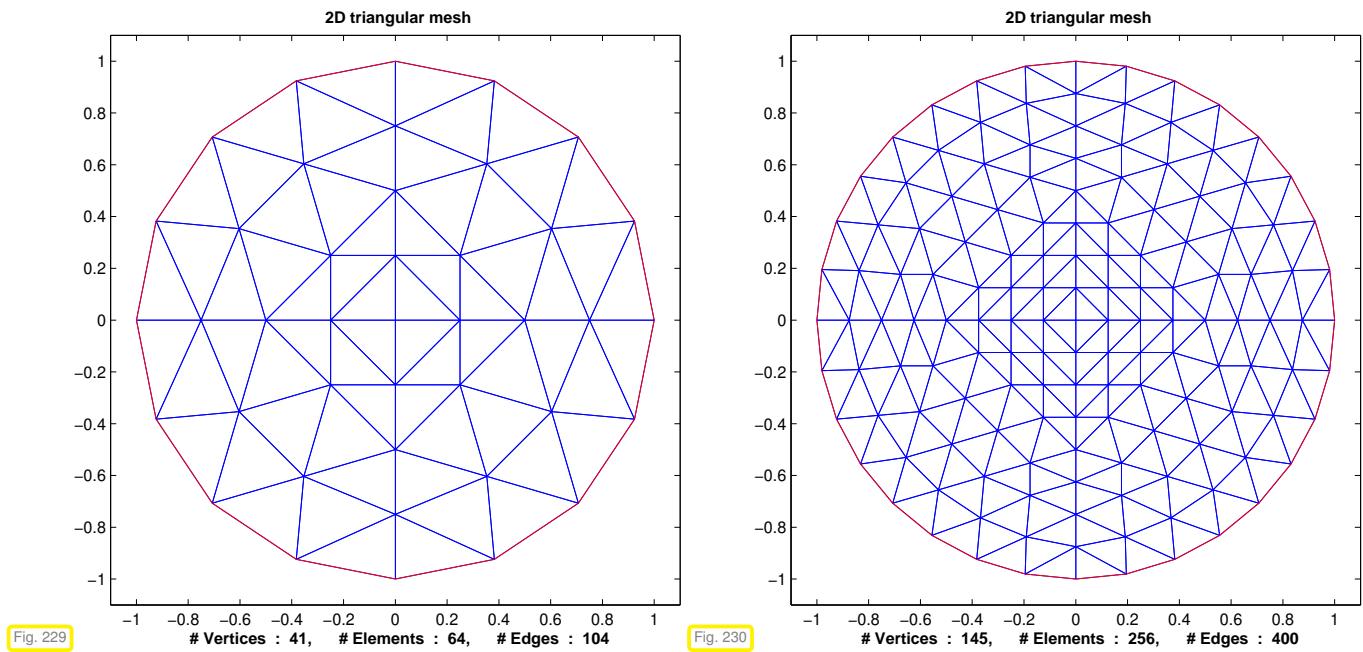
We focus on 2nd-order scalar linear variational problems as in the previous section. In Section 2.8.4 we saw how curved boundaries can be approximated by means of local polynomial interpolants and how this method meshes well with parametric finite element schemes.

It goes without saying that an approximation of the boundary amounts to solving the variational boundary value problem on a perturbed domain Ω_h instead of Ω , which constitutes a variational crime. The next experiment studies the consequences of such a domain perturbation. It also highlights the conceptual difficulties of measuring discretization errors in this case.

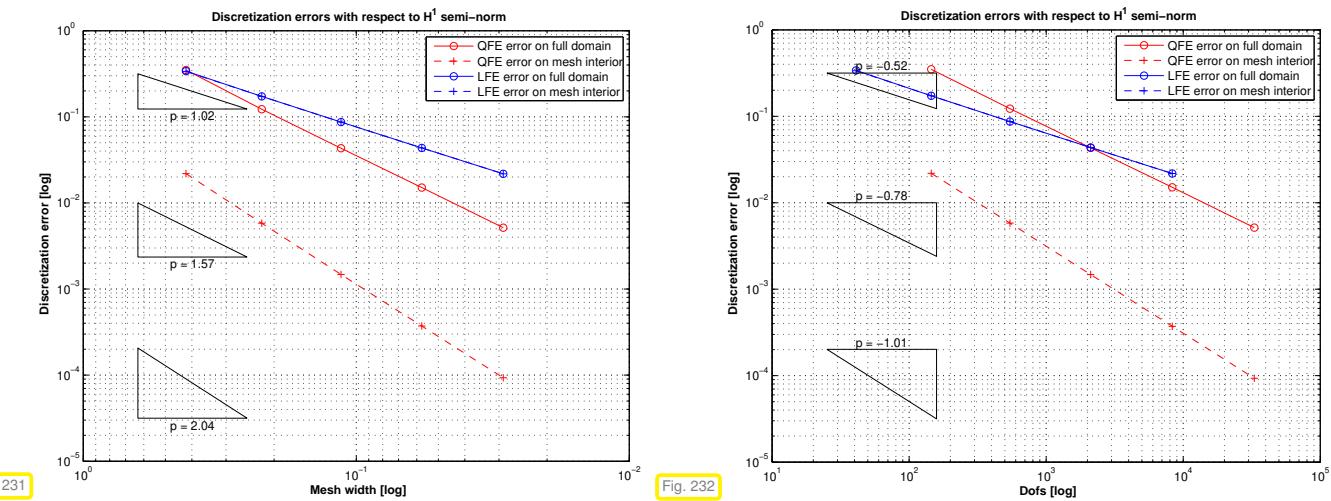
EXPERIMENT 3.5.2.1 (Impact of linear boundary approximation on FE convergence) We consider a pure Dirichlet boundary value problem for $-\Delta$ with zero Dirichlet boundary conditions.

Setting: $\Omega := B_1(0) := \{\mathbf{x} \in \mathbb{R}^2 : |\mathbf{x}| < 1\}$, $u(r, \varphi) = \cos(r\pi/2)$ (polar coordinates)
 $\Rightarrow f = \frac{\pi}{2r} \sin(r\pi/2) + \frac{\pi}{2} \cos(r\pi/2)$

- Sequences of unstructured triangular meshes \mathcal{M} obtained by regular refinement (of coarse mesh with 4 triangles) + linear boundary fitting.
- Galerkin FE discretization based on $V_h := \mathcal{S}_{1,0}^0(\mathcal{M})$ or $V_h := \mathcal{S}_{2,0}^0(\mathcal{M})$.
- Recorded: approximate norm $|u - u_h|_{1,\Omega_h}$, evaluated using numerical quadrature rule (2.7.5.37).
 (FE solution extended beyond the domain covered by \mathcal{M} (“mesh interior”) to Ω (“full domain”) by means of polynomial extrapolation.)



Linearly boundary fitted unstructured triangular meshes of $\Omega = B_1(0)$.



$H^1(\Omega)$ -norm of discretization error on unit ball ($- \leftrightarrow p = 1$, $- + \leftrightarrow p = 2$)

Dashed lines in Fig. 231, Fig. 232: error norms computed on polygonal domain covered by the mesh $\neq \Omega$; this spurious “error norm” suggests no deterioration of the convergence! \square

We may wonder whether guidelines for “safe” boundary approximation can be given, similar to the recipes formulated for numerical quadrature in Section 3.5.1. Fortunately, there are such guidelines:

Rule of thumb deduced from sophisticated finite element theory:

If $V_{0,h} = \mathcal{S}_p^0(\mathcal{M})$, use boundary fitting with polynomials of degree p .

Here, in two dimensions, $d = 2$, boundary fitting with polynomials of degree p means that

- we first employ a polygonal approximation Γ_{poly} of $\partial\Omega$ with nodes located on $\partial\Omega$,
- and then, over each edge of Γ_{poly} , employ local interpolation of the corresponding piece of $\partial\Omega$ by means of polynomials of degree p . The interpolants combined give a piecewise polynomial approximation Γ_h of $\partial\Omega$. This is illustrated in Fig. 161 for $p = 2$.

The corresponding algorithms in the context of parametric finite-element Galerkin approximation have

been discussed in Section 2.8.4 including implementation in L^EHRFEM++.

Review question(s) 3.5.2.2 (Variational Crimes)

(Q3.5.2.2.A) The local 2D trapezoidal rule on a triangular mesh \mathcal{M} of a domain $\Omega \subset \mathbb{R}^2$

$$\int_{\Omega} \varphi(x) dx \approx \sum_{K \in \mathcal{M}} \frac{1}{3} |K| \sum_{\ell=1}^3 \varphi(a_K^\ell) \quad a_K^\ell \text{ the vertices of } K ,$$

is used to approximate the right-hand-side functional $\ell(v) := \int_{\Omega} f(x) v(x) dx$, $f \in C^0(\overline{\Omega})$, for a second-order elliptic boundary value problem, that is we use the perturbed functional

$$\ell_h(v) := \sum_{p \in \mathcal{N}(\mathcal{M})} \omega_p f(p) v(p) , \quad \omega_p \in \mathbb{R} , \quad (3.5.2.3)$$

where $\mathcal{N}(\mathcal{M})$ is the set of nodes of \mathcal{M} .

- What are the weights ω_p ?
- Is the functional ℓ_h from (3.5.2.3) bounded on $H^1(\Omega)$?
- Is the functional ℓ_h from (3.5.2.3) continuous on $\mathcal{S}_1^0(\mathcal{M})$?

(Q3.5.2.2.B) We approximate the right-hand-side functional $\ell(v) := \int_{\Omega} f(x) v(x) dx$, $f \in C^\infty(\overline{\Omega})$, for a linear variational problem on $H_0^1(\Omega)$ by

$$\ell_h(v) := \int_{\Omega} (I_p f)(x) v(x) dx , \quad v \in H_0^1(\Omega) , \quad (3.5.2.4)$$

where $I_p : C^0(\overline{\Omega}) \rightarrow \mathcal{S}_p^0(\mathcal{M})$, $p \in \mathbb{N}$, is the standard nodal interpolation operator, \mathcal{M} a triangular mesh.

- Is ℓ_h from (3.5.2.4) bounded on $H_0^1(\Omega)$?
- Predict the asymptotic dependence of the quantities

$$\delta(\mathcal{M}) := \sup_{v \in H_0^1(\Omega)} \frac{|(\ell - \ell_h)(v)|}{\|v\|_{H^1(\Omega)}}$$

on the meshwidth $h_{\mathcal{M}}$ on sequences of meshes obtained by uniform regular refinement.

One of the following results can help you answer the second question:

Theorem 3.3.5.6. Best approximation error estimates for Lagrangian finite elements

Let $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, be a bounded polygonal/polyhedral domain equipped with a mesh \mathcal{M} consisting of simplices or parallelepipeds. Then, for each $k \in \mathbb{N}$, there is a constant $C > 0$ depending only on k and the shape regularity measure $\rho_{\mathcal{M}}$ such that

$$\inf_{v_h \in \mathcal{S}_p^0(\mathcal{M})} \|u - v_h\|_{H^1(\Omega)} \leq C \left(\frac{h_{\mathcal{M}}}{p} \right)^{\min\{p+1,k\}-1} \|u\|_{H^k(\Omega)} \quad \forall u \in H^k(\Omega) . \quad (3.3.5.7)$$

Theorem 3.5.2.5. $H^1(\Omega)$ -Norm interpolation error estimates for Lagrangian finite elements

Let $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, be a bounded polygonal/polyhedral domain equipped with a mesh \mathcal{M} consisting of simplices or parallelepipeds. Then, for each $k \in \mathbb{N}$, $k \geq 2$, $p \in \mathbb{N}$ there is a constant $C > 0$ depending only on k , the polynomial degree p , and the shape regularity measure $\rho_{\mathcal{M}}$ such that

$$\|u - I_p u\|_{H^1(\Omega)} \leq C h_{\mathcal{M}}^{\min\{p+1,k\}-1} \|u\|_{H^k(\Omega)}, \quad \forall u \in H^k(\Omega),$$

where $I_p : C^0(\bar{\Omega}) \rightarrow \mathcal{S}_p^0(\mathcal{M})$ is a nodal interpolation operator.

Theorem 3.5.2.6. $L^2(\Omega)$ -Norm interpolation error estimates for Lagrangian finite elements

Let $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, be a bounded polygonal/polyhedral domain equipped with a mesh \mathcal{M} consisting of simplices or parallelepipeds. Then, for each $k \in \mathbb{N}$, $k \geq 2$, $p \in \mathbb{N}$ there is a constant $C > 0$ depending only on k , the polynomial degree p , and the shape regularity measure $\rho_{\mathcal{M}}$ such that

$$\|u - I_p u\|_{L^2(\Omega)} \leq C h_{\mathcal{M}}^{\min\{p+1,k\}} \|u\|_{H^k(\Omega)}, \quad \forall u \in H^k(\Omega),$$

where $I_p : C^0(\bar{\Omega}) \rightarrow \mathcal{S}_p^0(\mathcal{M})$ is a nodal interpolation operator.

(Q3.5.2.2.C) Which of the following practices is a **variational crime** (V.C.)?

1. The use of a Lagrangian finite elements of insufficient polynomial degree for a second-order elliptic boundary value problem



Definitely a V.C.



Not necessarily a V.C.

2. Computing the finite element solution of a BVP posed on $\Omega \subset \mathbb{R}^2$ on a mesh \mathcal{M} for which

$$\bigcup \{\bar{K} : K \in \mathcal{M}\} \neq \bar{\Omega} ?$$



Definitely a V.C.



Not necessarily a V.C.

3. The use of the 2D trapezoidal rule for computing the element vectors for the right-hand-side functional $v \mapsto \int_{\Omega} f(x)v(x) dx$, $f \in C^0(\bar{\Omega})$.



Definitely a V.C.



Not necessarily a V.C.

4. The use of global shape functions that fail to satisfy the cardinal basis property with respect to a set of interpolation nodes



Definitely a V.C.



Not necessarily a V.C.

△

3.6 FEM: Duality Techniques for Error Estimation

The focus on convergence of the energy norm of the finite element discretization error, and, implicitly, on the convergence of underlying quadratic energy functionals, is often too narrow. In this section we study convergence of other quantities of interest, among them, the $L^2(\Omega)$ -norm of the discretization error.

3.6.1 Linear Output Functionals



Video tutorial for Section 3.6.1: Linear Output Functionals: (28 minutes) [Download link](#), [tablet notes](#)

§3.6.1.1 (Setting) We adopt abstract setting of Section 3.1 and consider a **linear variational problem** (1.4.1.7) in the form

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0, \quad (2.2.0.2)$$

- ◆ $V_0 \doteq$ (real) vector space, a space of functions $\Omega \mapsto \mathbb{R}$ for scalar 2nd-order elliptic variational problems, usually “energy space” $H^1(\Omega)/H_0^1(\Omega)$, see Section 1.3
- ◆ $a : V_0 \times V_0 \mapsto \mathbb{R} \doteq$ a bilinear form, see Def. 0.3.1.4,
- ◆ $\ell : V_0 \mapsto \mathbb{R} \doteq$ a linear form, see Def. 0.3.1.4,
- ◆ Ass. 3.1.1.2, Ass. 3.1.1.3, Ass. 3.1.1.4 are supposed to hold \Rightarrow existence, uniqueness, and stability of solution u by Thm. 3.1.1.5.

(Examples of variational forms for 2nd-order linear BVPs are discussed in Rem. 3.1.1.6, Section 1.8)

The Galerkin discretization of (2.2.0.2) using the trial/test space $V_{0,h} \subset V_0$ leads to the discrete variational problem

$$u_h \in V_{0,h}: \quad a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h}. \quad (2.2.1.1)$$

□

New twist: Now we are interested mainly/only in the *number* $F(u)$, where

$F : V_0 \mapsto \mathbb{R}$ is a given **output functional**.

Mathematical terminology: **functional** \doteq mapping from a function space into \mathbb{R}

EXAMPLE 3.6.1.2 (Output functionals) There are some output functionals for solutions of PDEs commonly encountered in applications:

- mean values, see Exp. 3.6.1.6 below,
- total heat flux through a surface (for heat conduction model \rightarrow Section 1.6), see Exp. 3.6.2.3 below,
- total surface charge of a conducting body (for electrostatics \rightarrow Section 1.2.2),
- total heat production (Ohmic losses) by stationary currents,
- total force on a charged conductor (for electrostatics \rightarrow Section 1.2.2),
- lift and drag in computational fluid dynamics (aircraft simulation),
- monostatic radar cross section for wave scattering problems in frequency domain,
- and many more . . .

□

We consider output functionals with special properties, which are rather common in practice:

Assumption 3.6.1.3. Linearity of output functional

The output functional F is a **linear** form (\rightarrow Def. 0.3.1.4) on V_0

To put the next assumption into context, please recall Ass. 3.1.1.3 and Section 1.4.3.

Assumption 3.6.1.4. Continuity of output functional \rightarrow **Def. 1.2.3.42**

The output functional is **continuous** w.r.t. the energy norm in the sense that

$$\exists C_f > 0: |F(v)| \leq C_f \|v\|_a \quad \forall v \in V_0 .$$

Now consider the Galerkin discretization of (2.2.0.2) based on the Galerkin trial/test space $V_{0,h} \subset V_0$, $N := \dim V_{0,h} < \infty$, and the associated discrete variational problem

$$u_h \in V_{0,h}: a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h} . \quad (2.2.1.1)$$

What would you dare to sell as an approximation of $F(u)$? Of course, ...

Galerkin solution $u_h \in V_{0,h}$ \Rightarrow approximate output value $F(u_h)$

§3.6.1.5 (A simple estimate) How accurate is $F(u_h)$, that is, how big is the **output error** $|F(u) - F(u_h)|$?

Linearity (\rightarrow Ass. 3.6.1.3) and continuity according to Ass. 3.6.1.4 conspire to furnish a very simple estimate

$$|F(u) - F(u_h)| \leq C_f \|u - u_h\|_a .$$

► A priori estimates for $\|u - u_h\|_a \Rightarrow$ estimates for $|F(u) - F(u_h)|$

Hence, Thm. 3.3.5.6 immediately tells us the asymptotic convergence of linear and continuous output functionals defined for solutions of 2nd-order scalar elliptic BVPs and Lagrangian finite element discretization. \square

EXPERIMENT 3.6.1.6 (Approximation of mean temperature) We consider a simple non-dimensional heat conduction model (\rightarrow Section 1.6), scaled heat conductivity $\kappa \equiv 1$, on the domain $\Omega =]0, 1[^2$, with fixed temperature $u = 0$ on $\partial\Omega$:

$$-\Delta u = f \quad \text{in } \Omega , \quad u = 0 \quad \text{on } \partial\Omega .$$

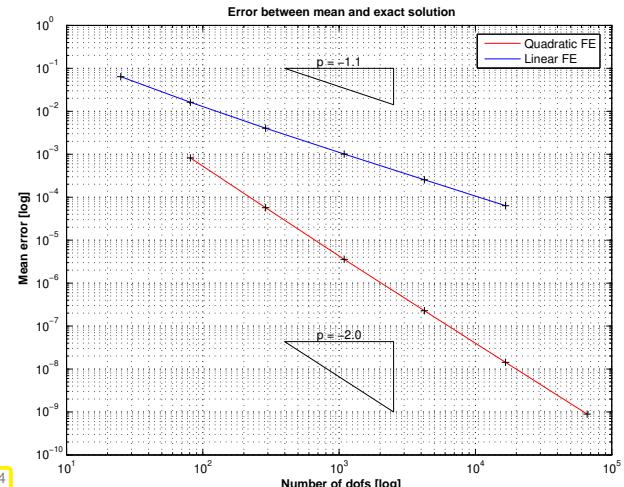
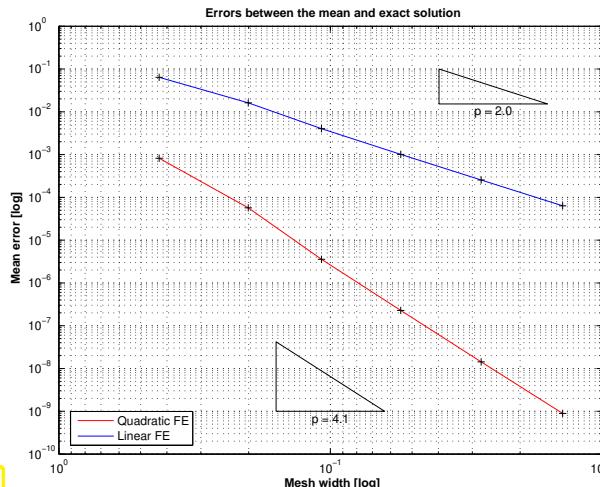
We choose the heat source function $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$, $\begin{bmatrix} x \\ y \end{bmatrix} \in \Omega$, such that we obtain the exact solution $u(x, y) = \sin(\pi x) \sin(\pi y)$. We are interested in the

$$\text{mean temperature: } F(u) := \frac{1}{|\Omega|} \int_{\Omega} u \, dx .$$

Details of finite element Galerkin discretization:

- Sequence of triangular meshes \mathcal{M} created by regular refinement.
- Galerkin discretization: (I) $V_{0,h} := S_{1,0}^0(\mathcal{M})$ (linear Lagrangian finite elements \rightarrow Section 2.4),
 (II) $V_{0,h} := S_{2,0}^0(\mathcal{M})$ (quadratic Lagrangian finite elements \rightarrow Ex. 2.6.1.2).
- Quadrature rule (2.7.5.37) of order 6 for assembly of right hand side vector
 (more than sufficiently accurate according to the guidelines of Section 3.5.1)

Expected: algebraic convergence in h_M with rate 1 of approximate mean temperature



Error in mean value on unit square ($\text{---} \leftrightarrow p = 1$, $\text{---} \leftrightarrow p = 2$)

Observation: Mean value converges twice as fast as expected: algebraic convergence $O(h_M^2)$ for $h_M \rightarrow 0!$

Theorem 3.6.1.7. Duality estimate for linear functional output

Define the **dual solution** $g_F \in V_0$ belonging to the linear functional $F : V_0 \rightarrow \mathbb{R}$ as solution of the **dual variational problem**

$$g_F \in V_0: \quad a(v, g_F) = F(v) \quad \forall v \in V_0.$$

Then

$$|F(u) - F(u_h)| \leq \|u - u_h\|_a \inf_{v_h \in V_{0,h}} \|g_F - v_h\|_a. \quad (3.6.1.8)$$

Proof. For any $v_h \in V_{0,h}$:

$$F(u) - F(u_h) = a(u - u_h, g_F) \stackrel{(*)}{=} a(u - u_h, g_F - v_h) \leq \|u - u_h\|_a \|g_F - v_h\|_a.$$

(*) \leftarrow by Galerkin orthogonality (3.1.3.2).

□

If g_F can be approximated well in $V_{0,h}$, then the **output error** can converge $\rightarrow 0$ (much) faster than $\|u - u_h\|_a$.

EXAMPLE 3.6.1.9 (Approximation of mean temperature cont'd \rightarrow Exp. 3.6.1.6)

- ◆ The mean temperature functional (3.6.1.14) is obviously linear \rightarrow Ass. 3.6.1.3
- ◆ By the Cauchy-Schwarz inequality (1.3.4.15) it clearly satisfies Ass. 3.6.1.4 even with $\|\cdot\|_a = \|\cdot\|_{L^2(\Omega)}$, let alone for $\|\cdot\|_a = |\cdot|_{H^1(\Omega)}$ on $H_0^1(\Omega)$.

What is $g_F \in H_0^1(\Omega)$ in this case? By Thm. 3.6.1.7 it is the solution of the variational problem

$$\int_{\Omega} \mathbf{grad} g_F \cdot \mathbf{grad} v \, dx = F(v) = \frac{1}{|\Omega|} \int_{\Omega} v \, dx \quad \forall v \in H_0^1(\Omega).$$

The associated 2nd-order BVP reads in strong form

$$-\Delta g_F = \frac{1}{|\Omega|} \quad \text{in } \Omega, \quad g_F = 0 \quad \text{on } \partial\Omega. \quad (3.6.1.10)$$

Now recall the elliptic lifting theory Thm. 3.4.0.10 for convex domains: since $\Omega =]0,1[^2$ is convex, we conclude $g_F \in H^2(\Omega)$.

- By interpolation estimate of Thm. 3.3.2.21 ($I_1 \doteq$ linear interpolation onto $\mathcal{S}_1^0(\mathcal{M})$) or Thm. 3.3.5.6 with $p = 1, k = 2$:

$$\inf_{v_h \in \mathcal{S}_1^0(\mathcal{M})} |g_F - v_h|_{H^1(\Omega)} \leq |g_F - I_1 g_F|_{H^1(\Omega)} \leq C h_{\mathcal{M}} |g_F|_{H^2(\Omega)},$$

where $C > 0$ may depend on Ω and the shape regularity measure (\rightarrow Def. 3.3.2.20) of \mathcal{M} .

Plug this into the **duality estimate** (3.6.1.14) of Thm. 3.6.1.7 and note that $u \in H^2(\Omega)$ by virtue of Thm. 3.4.0.10 and $f \in L^2(\Omega)$:

$$\blacktriangleright |F(u) - F(u_h)| \leq C h_{\mathcal{M}} \cdot \underbrace{|u - u_h|_{H^1(\Omega)}}_{\leq C h_{\mathcal{M}} \text{ if } u \in H^2(\Omega)} \leq C h_{\mathcal{M}}^2,$$

where the “generic constant” $C > 0$ depends only on $\Omega, u, \rho_{\mathcal{M}}$.

Again, by the elliptic lifting theory Thm. 3.4.0.10 we infer that $u \in H^2(\Omega)$ holds for this example since $f \in L^2(\Omega)$. □

Remark 3.6.1.11 (Elliptic lifting result for rectangular domains) Owing to their tensor-product geometry, rectangular domains allow more powerful elliptic lifting results than general polygonally bounded 2D domains. In particular we can conclude

$$\Omega =]0,1[^2, \quad u \in H_0^1(\Omega), \quad \Delta u \in H^1(\Omega) \Rightarrow \|u\|_{H^3(\Omega)} \leq \|\Delta u\|_{H^1(\Omega)}. \quad (3.6.1.12)$$

In Exp. 3.6.1.6, the constant right-hand-side source function for the dual Dirichlet boundary value problem (3.6.1.10) is certainly in $H^1(\Omega)$. Hence, for the dual solution g_F we conclude $g_F \in H^3(\Omega)$, which implies

$$\exists C = C(\Omega, g_F): \int_{v_h \in \mathcal{S}_2^0(\mathcal{M})} |g_F - v_h|_{H^1(\Omega)} \leq C h_{\mathcal{M}}^2,$$

when we invoke Thm. 3.3.5.6 for $p = 2, k = 3$. As a consequence, the rate of convergence $F(u_h) \rightarrow F(u)$ for $h_{\mathcal{M}} \rightarrow 0$ and quadratic Lagrangian finite elements is larger by 2 compared to the rate of $\|u - u_h\|_{H^1(\Omega)}$. This was observed in Exp. 3.6.1.6. □

Review question(s) 3.6.1.13 (Linear output functionals and duality techniques)

(Q3.6.1.13.A) Prove that the “mean temperature functional”

$$F : H^1(\Omega) \rightarrow \mathbb{R}, \quad v \mapsto F(v) := \frac{1}{|\Omega|} \int_{\Omega} v(x) dx$$

is continuous/bounded.

Definition 1.2.3.42. Continuity of a linear form and bilinear form

Consider a normed vector space V_0 with norm $\|\cdot\|$. A linear form $\ell : V_0 \rightarrow \mathbb{R}$ (\rightarrow Def. 0.3.1.4) is **continuous** or **bounded** on V_0 , if

$$\exists C > 0: |\ell(v)| \leq C\|v\| \quad \forall v \in V_0.$$

A bilinear form $a : V_0 \times V_0 \rightarrow \mathbb{R}$ (\rightarrow Def. 0.3.1.4) on V_0 is **continuous**, if

$$\exists K > 0: |a(u, v)| \leq K\|u\|\|v\| \quad \forall u, v \in V_0.$$

(Q3.6.1.13.B) [Proof of duality estimate] Let $u \in V_0$ denote that solution of the linear variational problem

$$u \in V_0: a(u, v) = \ell(v) \quad \forall v \in V_0, \quad (2.2.0.2)$$

posed on a vector space V_0 , and with an s.p.d. bilinear form $a : V_0 \times V_0 \rightarrow \mathbb{R}$ inducing the energy norm $\|\cdot\|_a$, $\ell : V_0 \rightarrow \mathbb{R}$ a linear functional, continuous/bounded with respect to $\|\cdot\|_a$.

Write $F : V_0 \rightarrow \mathbb{R}$ for a linear functional that is continuous/bounded with respect to $\|\cdot\|_a$, and $u_h \in V_{0,h}$ for the Galerkin solution of (2.2.0.2), $V_{0,h} \subset V_0$ some subspace

Give a three-line proof of the following **duality estimate**.

Theorem 3.6.1.7. Duality estimate for linear functional output

Define the **dual solution** $g_F \in V_0$ to F as solution of the **dual variational problem**

$$g_F \in V_0: a(v, g_F) = F(v) \quad \forall v \in V_0.$$

Then

$$|F(u) - F(u_h)| \leq \|u - u_h\|_a \inf_{v_h \in V_{0,h}} \|g_F - v_h\|_a. \quad (3.6.1.14)$$

△

3.6.2 Case Study: Computation of Boundary Fluxes with FEM

Model problem (process engineering):

Long pipe carrying turbulent flow of coolant (water)

$\Omega \subset \mathbb{R}^2$: cross-section of pipe

κ : (scaled) heat conductivity of pipe material
(assumed homogeneous, $\kappa = \text{const}$)

Assumption: Constant temperatures u_o , u_i at outer/inner wall Γ_o , Γ_i of pipe

Task: Compute heat flow pipe → water

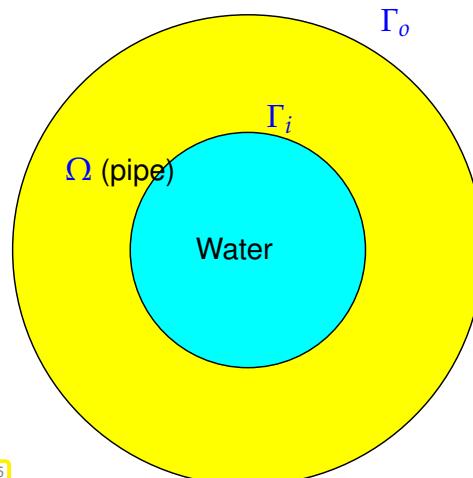


Fig. 235

Mathematical model: elliptic boundary value for stationary heat conduction (\rightarrow Section 1.6)

$$-\operatorname{div}(\kappa \operatorname{grad} u) = 0 \quad \text{in } \Omega, \quad u = u_x \quad \text{on } \Gamma_x, x \in \{i, o\}. \quad (3.6.2.1)$$

$$\text{Heat flux through } \Gamma_i: \quad J(u) := \int_{\Gamma_i} \kappa \mathbf{grad} u \cdot \mathbf{n} \, dS . \quad (3.6.2.2)$$

Relate to abstract framework: $(3.6.2.1) \cong (2.2.0.2)$, $V_0 \cong H_0^1(\Omega)$ (\rightarrow Section 1.8)

(Actually, $u \in H^1(\Omega)$, but by means of offset functions we can switch to the variational space $H_0^1(\Omega)$, see Section 1.2.3, Section 2.7.6.)

Numerical method: finite element computation of heat conduction in pipe
(e.g. linear Lagrangian finite element Galerkin discretization, Section 2.4)

Expectation: Algebraic convergence $|J(u) - J(u_h)| = O(h_M^2)$ for regular h -refinement

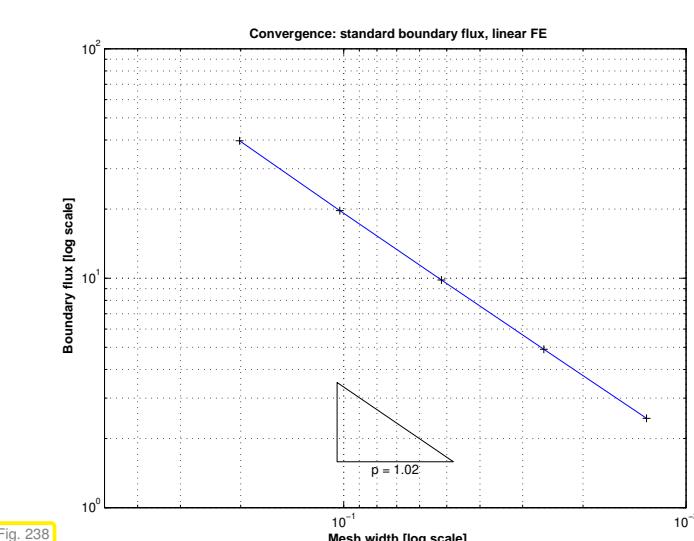
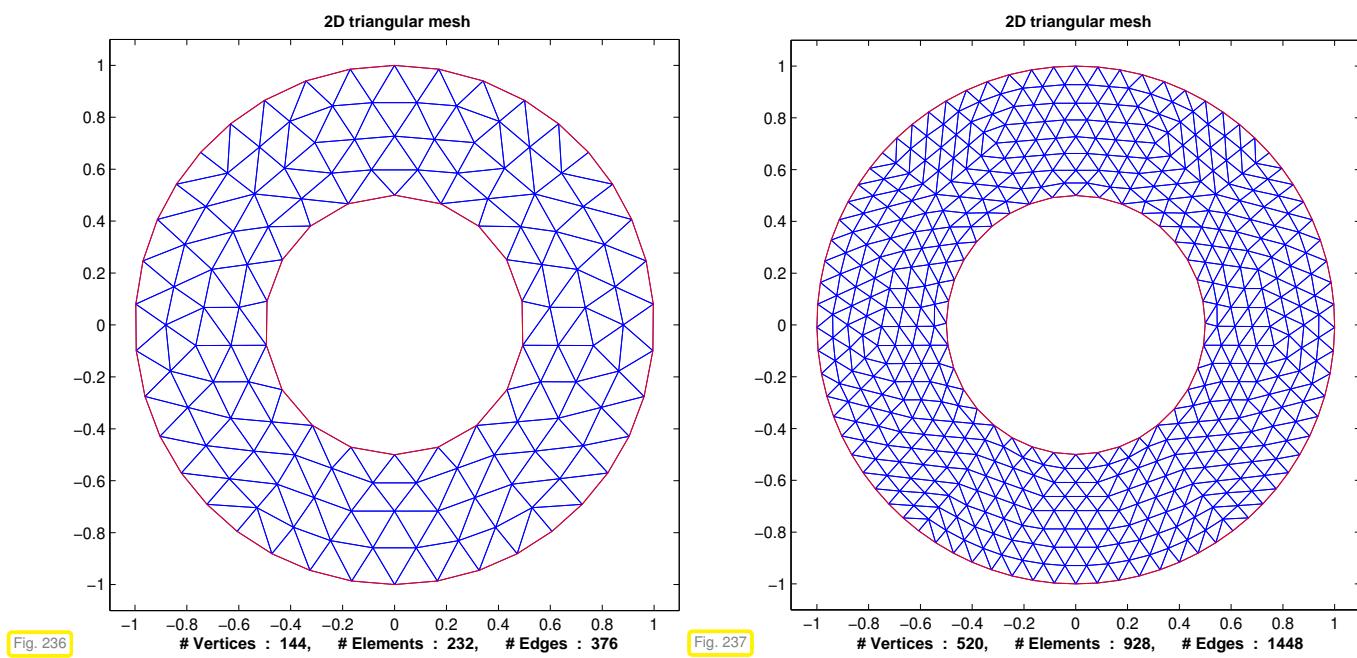
This expectation is based on the analogy to Exp. 3.6.1.6 (Approximation of mean temperature), where duality estimates yielded $O(h_M^2)$ convergence of the mean temperature error in the case of Galerkin discretization by means of linear Lagrangian finite elements on a sequence of meshes obtained by regular refinement. Now, it seems, we can follow the same reasoning.

EXPERIMENT 3.6.2.3 (Computation of heat flux)

- ◆ Setting: model problem “heat flux pipe \rightarrow water”, see (3.6.2.1) and Fig. 235.
- ◆ Linear output functional from (3.6.2.2)
- ◆ Domain $\Omega = B_{R_o}(0) \setminus B_{R_i}(0) := \{\mathbf{x} \in \mathbb{R}^2 : R_i < |\mathbf{x}| < R_o\}$ with $R_o = 1$ and $R_i = 1/2$
- ◆ Dirichlet boundary data $u_i = 60^\circ\text{C}$ on Γ_i , $u_o = 10^\circ\text{C}$ on Γ_o , heat source $f \equiv 0$, heat conductivity $\kappa \equiv 1$.
- Exact solution: $u(r, \varphi) = C_1 \ln(r) + C_2$, with $C_1 := (u_o - u_i) / (\ln R_i - \ln R_o)$,
- Exact heat flux: $J = 2\pi\kappa C_1$, $C_2 := (\ln R_o u_i - \ln R_i u_o) / (\ln R_i - \ln R_o)$.

Details of linear Lagrangian finite element Galerkin discretization:

- Sequences of unstructured triangular meshes \mathcal{M} obtained by regular refinement of coarse mesh (from grid generator).
- Galerkin FE discretization based on $V_{0,h} := S_{1,0}^0(\mathcal{M})$.
- Approximate evaluation of $a(u_h, v_h)$, $f(v_h)$ by six point quadrature rule (2.7.5.37) (“overkill quadrature”, see Section 3.5.1)
- Approximate evaluation of $J(u_h)$ by 4 point Gauss-Legendre quadrature rule on boundary edges of \mathcal{M} .
- Linear boundary approximation (circle replaced by polygon).
- Recorded: errors $|J(u) - J(u_h)|$ on sequence of meshes.



Observation:

Algebraic convergence of output error for J from (3.6.2.2) *only with rate 1* (in mesh width h_M)!
 (This is not the fault of the piecewise linear boundary approximation, which is sufficient when using piecewise linear Lagrangian finite elements, see Section 3.5.2.)

Why was our expectation mistaken ?

Suspicion: the output functional J fails to meet requirements of duality estimates of Thm. 3.6.1.7:



boundary flux functional J from (3.6.2.2) is **not** continuous on $H^1(\Omega)$!

§3.6.2.4 (Non-continuity of boundary flux functional) How can we corroborate our suspicion that J from (3.6.2.2) fails to be continuous? First remember Def. 1.2.3.42.

Idea:

find $u \in H^1(\Omega)$, for which " $J(u) = \infty$ ",

cf. investigation of non-continuity of point evaluation functional on $H^1(\Omega) \rightarrow$ Ex. 1.2.3.45.

On $\Omega = \{x \in \mathbb{R}^2: \|x\| < 1\}$ (unit disk) consider

$$u(x) = (1 - \|x\|)^\alpha =: g(\|x\|), \quad \frac{1}{2} < \alpha < 1,$$

and the boundary flux functional (3.6.2.2) on $\partial\Omega$.

☞ On the one hand, using the expression (1.2.3.48) for the gradient in polar coordinates,

$$J_0(v) = \int_{\partial\Omega} \frac{\partial u}{\partial r}(x) dS(x) = 2\pi\alpha(1-r)^{\alpha-1}|_{r=1}'' = \infty''.$$

☞ On the other hand, straightforward computation of improper integral using (1.2.3.49):

$$\begin{aligned} |u|_{H^1(\Omega)}^2 &= \int_{\Omega} \|\mathbf{grad} u(x)\|^2 dx = 2\pi \int_0^1 |g'(r)|^2 r dr = 2\pi\alpha^2 \int_0^1 (1-r)^{2\alpha-2} r dr \\ &= 2\pi\alpha^2 \int_0^1 s^{2\alpha-2} (1-s) ds = 2\pi\alpha \left[\frac{s^{2\alpha-1}}{2\alpha-1} - \frac{s^{2\alpha}}{2\alpha} \right]_{s=0}^{s=1} = 2\pi \frac{1}{2\alpha-1} < \infty. \\ \text{Def. 1.3.4.8} \implies u &\in H^1(\Omega) \quad (u \in C^0(\bar{\Omega}) \text{ and } u \in C^\infty(\Omega \setminus \{0\})!). \end{aligned}$$

→

The considerations of § 3.6.2.4 show that the duality estimates of Thm. 3.6.1.7 cannot be applied



No guarantees for good convergence of flux obtained from straightforward evaluation of $J(u_h)$ for FE solution $u_h \in \mathcal{S}_{1,0}^0(\mathcal{M})$!

Apparently there is no remedy for this, because the boundary flux functional (3.6.2.2) seems to be enforced on us by the problem: we are not allowed to tinker with it, are we?

“Recovering” continuity of boundary flux functional

Trick:

use fixed **cut-off function** $\psi \in C^0(\bar{\Omega}) \cap H^1(\Omega)$, $\psi \equiv 1$ on Γ_i , $\psi|_{\Gamma_o} = 0$

$$\int_{\Gamma_i} \kappa \mathbf{grad} u \cdot \mathbf{n} dS = \int_{\Gamma_i} (\kappa \mathbf{grad} u \cdot \mathbf{n}) \psi dS = \int_{\Omega} \underbrace{\operatorname{div}(\kappa \mathbf{grad} u) \psi + \kappa \mathbf{grad} u \cdot \mathbf{grad} \psi}_{=0} dx$$

► use $J^*(u) := \int_{\Omega} \kappa \mathbf{grad} u \cdot \mathbf{grad} \psi dx$. (3.6.2.6)

Obviously (*): $J^* : H^1(\Omega) \mapsto \mathbb{R}$ continuous & $J^*(u) = J(u)$ for solution of (3.6.2.1)

(*): By the Cauchy-Schwarz inequality (1.3.4.15), since $\kappa = \text{const}$,

$$|J^*(u)| \leq \kappa \|\mathbf{grad} u\|_{L^2(\Omega)} \|\mathbf{grad} \psi\|_{L^2(\Omega)} \leq C|u|_{H^1(\Omega)},$$

with $C := \kappa \|\mathbf{grad} \psi\|_{L^2(\Omega)}$, which is a constant independent of u , as ψ is a fixed function.

Objection: You cannot just tamper with the output functional of a problem just because you do not like it!

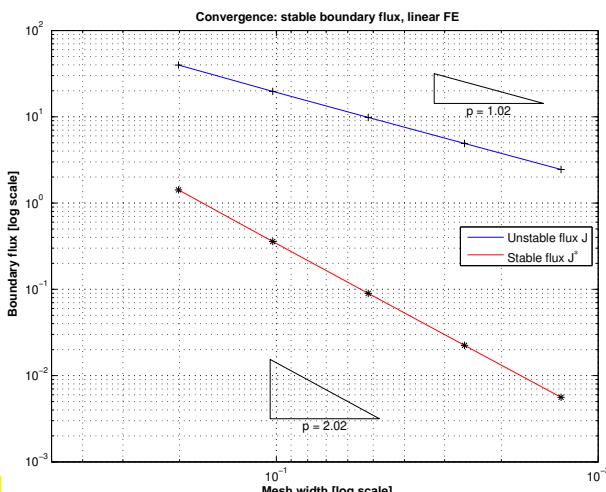
Rebuttal: Of course, one can replace the output function J with another one J^* as long as

$$J(u) = J^*(u) \quad \text{for the exact solution } u \text{ of the BVP,}$$

because the objective is not to “evaluate J ”, but to obtain an approximation for $J(u)$!

EXPERIMENT 3.6.2.7 (Computation of heat flux cont'd → Exp. 3.6.2.3) We empirically study the use of a modified heat flux functional J^* of the form (3.6.2.6). Additional details of the numerical tests are:

- Galerkin FE discretization based on $V_{0,h} := \mathcal{S}_{1,0}^0(\mathcal{M})$ or $V_{0,h} := \mathcal{S}_{2,0}^0(\mathcal{M})$.
- Approximate evaluation of $J^*(u_h)$ by six point quadrature rule (2.7.5.37) (“overkill quadrature”, see Section 3.5.1)
- Cut-off function with linear decay in radial direction: $\psi(x) = 2\|x\| - 1$, $\psi \in C^\infty(\bar{\Omega})$.
- Recorded: output errors $|J(u) - J(u_h)|$ and $|J(u) - J^*(u_h)|$.



▷ Convergence of $|J(u) - J(u_h)|$ and $|J(u) - J^*(u_h)|$ for linear Lagrangian finite element discretization.

Fig. 239

Additional observations:

- Algebraic convergence $|J(u) - J^*(u_h)| = O(h_M^2)$ (rate 2 !) for alternative output functional J^* from (3.6.2.6).
- Dramatically reduced output error!

Remark 3.6.2.8 (Duality estimate for modified heat flux functional) Let us try to understand the reason for the improved convergence of the modified heat flux functional based on the duality estimate of Thm. 3.6.1.7.

We consider the boundary value problem

$$-\Delta u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega,$$

whose weak formulation is posed on $H_0^1(\Omega)$, and the stabilized heat flux functional (3.6.2.6)

$$J^*(v) := \int_{\Omega} \mathbf{grad} v(x) \cdot \mathbf{grad} \psi(x) dx, \quad v \in H_0^1(\Omega).$$

The **dual variational problem** is

$$g_F \in H_0^1(\Omega): \int_{\Omega} \mathbf{grad} v \cdot \mathbf{grad} g_F dx = J^*(v) = \int_{\Omega} \mathbf{grad} v(x) \cdot \mathbf{grad} \psi(x) dx \quad \forall v \in H_0^1(\Omega). \quad (3.6.2.9)$$

To infer extra smoothness of g_F we rely on Green's formula Thm. 1.5.2.7 to recast the right-hand-side functional and convert (3.6.2.9) into

$$g_F \in H_0^1(\Omega): \int_{\Omega} \mathbf{grad} v \cdot \mathbf{grad} g_F \, dx = - \int_{\Omega} \Delta \psi v \, dx \quad \forall v \in H_0^1(\Omega). \quad (3.6.2.10)$$

The corresponding BVP in strong form is

$$\Delta g_F = \Delta \psi \quad \text{in } \Omega, \quad g_F = 0 \quad \text{on } \partial\Omega. \quad (3.6.2.11)$$

Assume that Ω is *convex*. Nevertheless, **only if** $\Delta \psi \in L^2(\Omega)$ we can invoke to Thm. 3.4.0.10 to conclude $g_F \in H^2(\Omega)$!

Thus, in actual computations, as we did in Exp. 3.6.2.7, we had better choose $\psi \in C_{pw}^2(\bar{\Omega})$ in addition to its other required properties. We cannot simply use a finite element function, e.g., $\psi \in S_1^0(\mathcal{M})$. \square

Remark 3.6.2.12 (Finding continuous replacement functionals) Now you will ask: How can we find good (continuous) replacement functionals, if we are confronted with an unbounded output functional on the energy space?

Unfortunately, there is *no recipe*, and sometimes it does not seem to be possible to find a suitable J^* at all, for instance in the case of point evaluation, cf. Ex. 1.2.3.45.

Good news: another opportunity to show off how smart you are! \square

Review question(s) 3.6.2.13 (Duality techniques (II))

(Q3.6.2.13.A) We consider the Galerkin discretization of the Dirichlet BVP

$$-\Delta u = f \in H^1(\Omega) \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega,$$

$\Omega =]0, 1[^2$, by means of degree-2 Lagrangian finite elements, $V_{0,h} = S_{2,0}^0(\mathcal{M}) \subset H_0^1(\Omega)$, on sequences of triangular meshes created by uniform regular refinement.

For the “mean temperature functional”

$$F : H^1(\Omega) \rightarrow \mathbb{R}, \quad v \mapsto \frac{1}{|\Omega|} \int_{\Omega} v(x) \, dx$$

determine the rate of algebraic convergence $F(u_h) \rightarrow F(u)$ in terms of meshwidth $h_{\mathcal{M}} \rightarrow 0$, when $u_h \in S_{2,0}^0(\mathcal{M})$ is the finite-element solution.

Hint (\rightarrow Rem. 3.6.1.11): If $\Omega =]0, 1[^2$, $u \in H_0^1(\Omega)$, $-\Delta u = f$, $f \in H^1(\Omega)$, then $u \in H^3(\Omega)$ and there is a constant $C > 0$ such that $\|u\|_{H^3(\Omega)} \leq C \|f\|_{H^1(\Omega)}$.

(Q3.6.2.13.B) We consider the Dirichlet boundary value problem

$$-\Delta u = 0 \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega,$$

for given boundary data $g \in C^0(\partial\Omega)$ and on a domain $\Omega \subset \mathbb{R}^2$. In Exp. 3.6.2.7 we studied the regularized boundary flux functional

$$J^*(u) := \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} \psi \, dx \quad \psi \in H^1(\Omega).$$

State the boundary value problem that is solved by the dual solution g_F induced by this regularized output functional J^* in strong (PDE) form.

(Q3.6.2.13.C) If in Exp. 3.6.2.7 we change the computational domain from an annulus to $\Omega :=]-1, 1[^2 \setminus [-\frac{1}{2}, \frac{1}{2}]^2$. Speculate how this would affect the convergence rates observed in Fig. 240. \triangle

3.6.3 Lagrangian FEM: L^2 -Estimates

So far we have only studied the energy norm ($\leftrightarrow H^1(\Omega)$ -norm, see Rem. 3.3.2.22) of the finite element discretization error for 2nd-order elliptic BVP.

The reason was the handy tool of Cea's lemma Thm. 3.1.3.7.

What about error estimates in other “relevant norms”, e.g.,

- in the mean square norm or $L^2(\Omega)$ -norm, see Def. 1.3.2.3,
- in the supremum norm or $L^\infty(\Omega)$ -norm, see Def. 0.3.2.25?

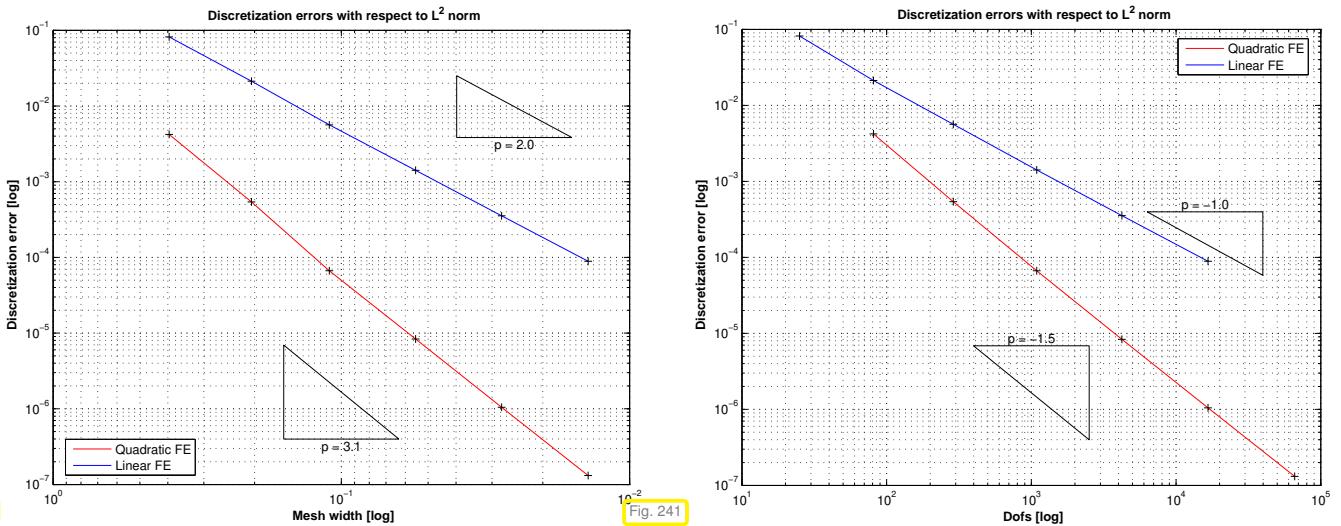
In this section we tackle $\|u - u_h\|_{L^2(\Omega)}$. We largely reuse the abstract framework of Section 3.6.1: linear variational problem (2.2.0.2) with exact solution $u \in V_0$, Galerkin finite element solution $u_h \in V_{0,h}$, see p. 362, and the special framework of linear 2nd-order elliptic BVPs, see Rem. 3.1.1.6: concretely,

$$a(u, v) := \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx, \quad u, v \in H_0^1(\Omega).$$

EXPERIMENT 3.6.3.1 (L^2 -convergence of FE solutions, Exp. 3.2.3.8 cnt'd)

Setting: $\Omega = [0, 1]^2$, $D \equiv 1$, $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$, $(x, y)^\top \in \Omega$
 ➤ exact solution $u(x, y) = \sin(\pi x) \sin(\pi y)$, $u|_{\partial\Omega} = 0$.

- Sequence of triangular meshes \mathcal{M} , created by regular refinement.
- FE Galerkin discretization based on $S_{1,0}^0(\mathcal{M})$ or $S_{2,0}^0(\mathcal{M})$.
- Quadrature rule (2.7.5.37) for assembly of local load vectors (\rightarrow Section 2.7.5).
- Approximate $L^2(\Omega)$ -norm by means of quadrature rule (2.7.5.37).



- Observations:
- Linear Lagrangian FE ($p = 1$) $\Rightarrow \|u - u_h\|_0 = O(N^{-1})$
 - Quadratic Lagrangian FE ($p = 2$) $\Rightarrow \|u - u_h\|_0 = O(N^{-1.5})$

The $L^2(\Omega)$ -norm of the Galerkin discretization error seems to converge algebraically with a higher rate for $h_{\mathcal{M}} \rightarrow N \rightarrow \infty$. \square

§3.6.3.2 (L^2 interpolation error) Recall the interpolation error estimate of Thm. 3.3.2.21

$$\|u - I_1 u\|_{L^2(\Omega)} = O(h_M^2) \quad \text{vs.} \quad |u - I_1 u|_{H^1(\Omega)} = O(h_M),$$

on a family of meshes with uniformly bounded shape regularity measure.

- ☞ Higher rate of algebraic convergence of the interpolation error when measured in the **weaker** $L^2(\Omega)$ -norm compared to the **stronger** $H^1(\Omega)$ -norm.

Therefore a similar observation in the case of the finite element approximation error is not so surprising.

↓

§3.6.3.3 (Duality techniques for L^2 -estimates) Now we supply a rigorous underpinning and explanation of the behavior of $\|u - u_h\|_{L^2(\Omega)}$ that we have observed and expect.



Idea: Consider special **continuous linear “output functional”**

$$F(v) := \int_{\Omega} v \cdot (u - u_h) \, dx !$$

This is not a practical output functional, because its evaluation will not be possible even if the finite element solution u_h is available. Nevertheless, this F is well defined, because existence and uniqueness of both u and u_h are guaranteed.

This functional is highly relevant for L^2 -estimates, because

$$F(u) - F(u_h) = \|u - u_h\|_{L^2(\Omega)}^2 !$$

- estimates for the output error will provide bounds for $\|u - u_h\|_{L^2(\Omega)}$!

Note: Both u and u_h are *fixed* functions $\in H^1(\Omega)$!

- Linearity of F (\rightarrow Ass. 3.6.1.3) is obvious.
- Continuity $F : H_0^1(\Omega) \mapsto \mathbb{R}$ (\rightarrow Ass. 3.6.1.4) is clear, use Cauchy-Schwarz inequality (1.3.4.15).

Duality estimate of Thm. 3.6.1.7 can be applied:

Thm. 3.6.1.7

$$\blacktriangleright F(u) - F(u_h) = \|u - u_h\|_{L^2(\Omega)}^2 \leq C |u - u_h|_{H^1(\Omega)} \inf_{v_h \in V_{0,h}} |g_F - v_h|_{H^1(\Omega)}, \quad (3.6.3.4)$$

where $C > 0$ may depend only on κ , and the **dual solution** $g_F \in H_0^1(\Omega)$ satisfies

$$\begin{aligned} a(g_F, v) = F(v) \quad \forall v \in V_0 \quad \Leftrightarrow \quad & \int_{\Omega} \kappa(x) \mathbf{grad} g_F \cdot \mathbf{grad} v \, dx = \int_{\Omega} v(u - u_h) \, dx \quad \forall v \in H_0^1(\Omega) \\ & \downarrow \\ & - \operatorname{div}(\kappa(x) \mathbf{grad} g_F) = u - u_h \quad \text{in } \Omega \quad , \quad g_F = 0 \quad \text{on } \partial\Omega . \end{aligned} \quad (3.6.3.5)$$

Assumption 3.6.3.6. 2-regularity of homogeneous Dirichlet problem

We assume that the homogeneous Dirichlet problem with coefficient κ is **2-regular** on Ω : There is $C > 0$, which depends on Ω only such that

$$\begin{aligned} u \in H_0^1(\Omega) \\ \operatorname{div}(\kappa(x) \operatorname{grad} u) \in L^2(\Omega) \end{aligned} \quad \Rightarrow \quad u \in H^2(\Omega) \quad \text{and} \quad |u|_{H^2(\Omega)} \leq C \|\operatorname{div}(\kappa(x) \operatorname{grad} u)\|_{L^2(\Omega)} .$$

By the elliptic lifting theorem for convex domains Thm. 3.4.0.10 we know

$$\kappa \text{ is } C^1\text{-smooth} \quad \& \quad \Omega \text{ convex} \implies \text{Ass. 3.6.3.6 is satisfied.}$$

§3.6.3.7 (Estimates under assumption of 2-regularity) Ass. 3.6.3.6 in conjunction with (3.6.3.5) yields

$$|g_F|_{H^2(\Omega)} \leq C \|u - u_h\|_{L^2(\Omega)}, \quad (3.6.3.8)$$

where $C > 0$ depends only on Ω .

Now we can appeal to the general best approximation theorem for Lagrangian finite element spaces Thm. 3.3.5.6 In the present setting it is applied in the form

$$\inf_{v_h \in \mathcal{S}_p^0(\mathcal{M})} |g_F - v_h|_{H^1(\Omega)} \leq C \frac{h_{\mathcal{M}}}{p} |g_F|_{H^2(\Omega)} \stackrel{(3.6.3.8)}{\leq} C \frac{h_{\mathcal{M}}}{p} \|u - u_h\|_{L^2(\Omega)}, \quad (3.6.3.9)$$

where the “generic constants” $C > 0$ depend only on Ω and the shape regularity measure $\rho_{\mathcal{M}}$ (\rightarrow Def. 3.3.2.20).

Combine (3.6.3.4) and (3.6.3.9) and cancel one power of $\|u - u_h\|_{L^2(\Omega)}$: With $C > 0$ depending only on Ω, κ , and the shape regularity measure $\rho_{\mathcal{M}}$ we conclude

$$\boxed{\text{Ass. 3.6.3.6} \quad \Rightarrow \quad \|u - u_h\|_{L^2(\Omega)} \leq C \frac{h_{\mathcal{M}}}{p} \|u - u_h\|_{H^1(\Omega)}}. \quad (3.6.3.10)$$

► for h -refinement: gain of one factor $O(h_{\mathcal{M}})$ (vs. $H^1(\Omega)$ -estimates) □

Is it important to assume 2-regularity, Ass. 3.6.3.6 or merely a technical requirement of the theoretical approach?

EXPERIMENT 3.6.3.11 (L^2 -estimates on non-convex domain cf. Exp. 3.2.3.10)

Setting: $\Omega = [-1, 1]^2 \setminus ([0, 1] \times [-1, 0]), D \equiv 1,$
 $u(r, \varphi) = r^{2/3} \sin(2/3\varphi)$ (exact solution in polar coordinates),
 ➤ we use $f = 0$, Dirichlet data $g = u|_{\partial\Omega}$.

Finite element Galerkin discretization and evaluations as in Exp. 3.6.3.1.

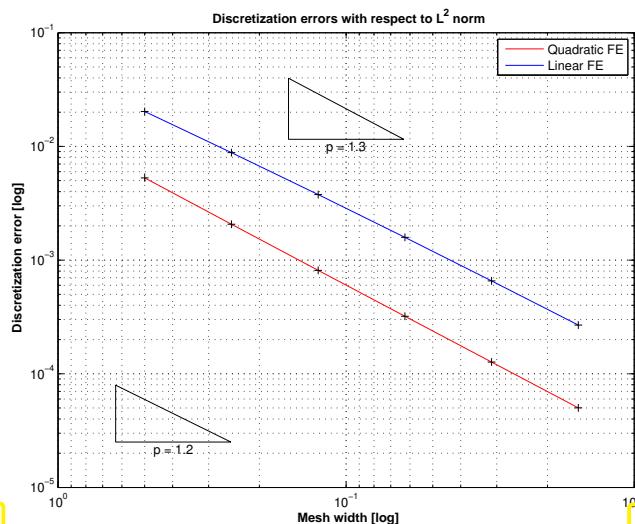
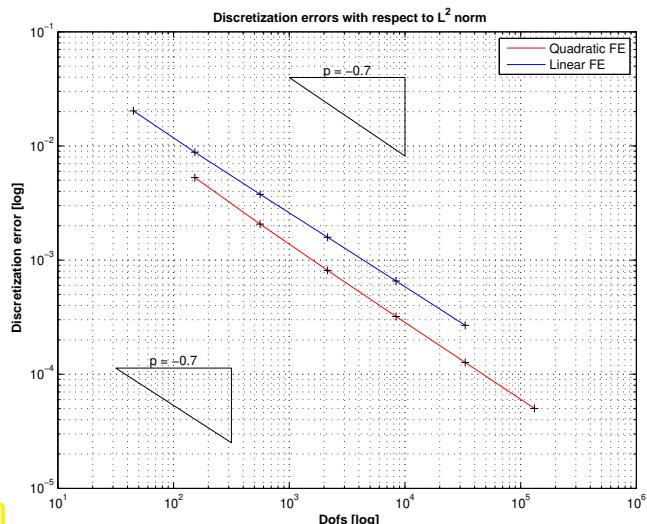


Fig. 242

 $L^2(\Omega)$ -norm of discretization error on "L-shaped" domain ($- \leftrightarrow p = 1$, $- \leftrightarrow p = 2$)

Observation: For both ($p = 1, 2$) \rightarrow algebraic convergence $\|u - u_h\|_0 = O(N^{-2/3})$

Comparison with Exp. 3.2.3.10: for both linear and quadratic Lagrangian FEM

$$\|u - u_h\|_{L^2(\Omega)} = O(N^{-2/3}) \longleftrightarrow \|u - u_h\|_{H^1(\Omega)} = O(N^{-1/3}),$$

that is, we again observe a doubling of the rate of convergence for the weaker norm.

No gain through the use of quadratic FEM, because of limited smoothness of both u and dual solution g_F . For both the solution and the dual solution the gradient will have a singularity at 0. \square

Remark 3.6.3.12 (Usefulness of L^2 -estimates) To begin with, the L^2 -estimates derived in this section are mainly motivated by curiosity: can we expect the higher rates of convergence that we are accustomed to for weaker norms of interpolation errors also from weaker norm of Galerkin discretization errors.

However, comparing observed convergence in L^2 -norm with what is predicted by theory, should be used for testing the correctness of finite element codes, following the procedure of § 3.8.0.9. \square

Review question(s) 3.6.3.13 (Duality techniques (III))

(Q3.6.3.13.A) In the setting of Exp. 3.6.3.1 give an estimate (as sharp as possible) for

1. $\|u - u_h\|_{L^2(\Omega)}$
2. and $\|u - u_h\|_{L^2(\partial\Omega)}$,

where u_h stands for finite element solutions obtained with either piecewise linear or piecewise quadratic Lagrangian finite elements.

Theorem 1.9.0.10. Multiplicative trace inequality

$$\exists C = C(\Omega) > 0: \|u\|_{L^2(\partial\Omega)}^2 \leq C \|u\|_{L^2(\Omega)} \cdot \|u\|_{H^1(\Omega)} \quad \forall u \in H^1(\Omega).$$

(Q3.6.3.13.B) The components of the linear variational problem

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0, \quad (2.2.0.2)$$

on the vector space V_0 , are supposed to satisfy the “usual assumptions”. Thus, it possesses a unique solution $u \in V_0$ for every ($\|\cdot\|_a$ -continuous) right-hand-side linear functional ℓ . We perform its Galerkin discretization based on a subspace $V_{0,h} \subset V_0$.

For a continuous output functional $F : V_0 \rightarrow \mathbb{R}$ let $g_{F,h}$ be the Galerkin solution of (2.2.0.2) with ℓ replaced with F . What does $\ell(g_{F,h})$ give you?

(Q3.6.3.13.C) [Multiple right-hand-side functionals] Let

$$u \in H^1(\Omega): \quad a(u, v) = \int_{\Omega} f_k(x) v(x) dx \quad \forall v \in H^1(\Omega), \quad (3.6.3.14)$$

be the variational formulation of a second-order elliptic boundary value problem, $a : H^1(\Omega) \times H^1(\Omega) \rightarrow \mathbb{R}$ a symmetric positive definite bilinear form, $f_k \in L^2(\Omega)$.

We are interested in approximately evaluating $F(u_k)$, u_k the solution of (3.6.3.14), for a bounded *linear functional* $F : H^1(\Omega) \rightarrow \mathbb{R}$ and for *many* different and unrelated source functions $f_k \in L^2(\Omega)$, $k = 1, \dots, m$, $m \in \mathbb{N}$. We rely on finite element Galerkin approximation based on piecewise linear Lagrangian finite elements, $V_{0,h} = \mathcal{S}_1^0(\mathcal{M})$, producing the Galerkin solutions $u_{k,h} \in \mathcal{S}_1^0(\mathcal{M})$.

Outline an algorithm for computing $F(u_{k,h})$, $k = 1, \dots, m$, which involves a minimal number of solves of sparse linear systems of equations.

(Q3.6.3.13.D) [Quadratic output functionals] We consider the linear variational problem

$$u \in V_0: \quad a(u, v) = \ell(v) \quad \forall v \in V_0,$$

on a vector space V_0 , and with an s.p.d. bilinear form $a : V_0 \times V_0 \rightarrow \mathbb{R}$ inducing the energy norm $\|\cdot\|_a$, $\ell : V_0 \rightarrow \mathbb{R}$ a linear functional, bounded with respect to $\|\cdot\|_a$.

We are interested in a *quadratic output functional* (\rightarrow Def. 1.2.3.2)

$$F(v) = \frac{1}{2}b(v, v) - f(v), \quad (3.6.3.15)$$

$b : V_0 \times V_0 \rightarrow \mathbb{R}$ a symmetric bilinear form, $f : V_0 \rightarrow \mathbb{R}$ a linear form satisfying

$$\exists C > 0: \quad |b(v, w)| \leq C\|v\|_a\|w\|_a, \quad |f(v)| \leq C\|v\|_a \quad \forall v, w \in V_0.$$

For the Galerkin solution $u_h \in V_{0,h}$, $V_{0,h} \subset V$ a subspace, establish the duality estimate

$$|F(u_h) - F(u)| \leq \|u - u_h\|_a \inf_{v_h \in V_{0,h}} \|g - v_h\|_a$$

and characterize $g \in V_0$.

△

3.7 Discrete Maximum Principle



Video tutorial for Section 3.7: Discrete Maximum Principle: (39 minutes) [Download link](#), [tablet notes](#)

So far we have investigated the *accuracy* of finite element Galerkin solutions: we studied relevant norms $\|u - u_h\|$ of the discretization error.

Now we adopt a new perspective:

structure preservation by FEM

To what extent does the finite element solution u_h inherit key structural properties of the solution u of a 2nd-order scalar elliptic BVP?

3.7.1 Maximum Principle for Scalar 2nd-Order Elliptic BVPs

The aspect of structure preservation will be discussed for a special structural property of the solution of the following linear 2nd-order elliptic BVP (inhomogeneous Dirichlet problem) in variational form (\rightarrow Section 1.8)

$$u \in \tilde{g} + H_0^1(\Omega): \quad a(u, v) := \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx \quad \forall v \in H_0^1(\Omega). \quad (3.7.1.1)$$

where $\tilde{g} \hat{=} \text{offset function, extension of Dirichlet data } g \in C^0(\partial\Omega)$, see Ex. 1.8.0.2, (1.9.0.6),
 $\kappa \hat{=} \text{bounded and uniformly positive definite diffusion coefficient, see (1.6.0.6)}$.

As we know, under some assumptions on the regularity of u , κ , and f (3.7.1.1) gives rise to a BVP in strong (PDE) form

$$-\operatorname{div}(\kappa(x) \mathbf{grad} u) = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega.$$

Recall (\rightarrow Section 1.6): (3.7.1.1) models *stationary* temperature distribution in body, when temperature on its surface is prescribed by g .

Our intuition predicts:

- ◆ In the absence of heat sources maximal and minimal temperature attained on surface.

In the presence of a heat source ($f \geq 0$) the temperature minimum will be attained on surface $\partial\Omega$.

- ◆ If $f \leq 0$ (heat sink), then the maximal temperature will be attained on the surface.

In fact this is a theorem:

Theorem 3.7.1.2. Maximum principle for 2nd-order elliptic BVP

For $u \in C^0(\bar{\Omega}) \cap H^1(\Omega)$ holds the **maximum principle**

$$\begin{aligned} -\operatorname{div}(\kappa(x) \mathbf{grad} u) \geq 0 &\implies \min_{x \in \partial\Omega} u(x) = \min_{x \in \Omega} u(x), \\ -\operatorname{div}(\kappa(x) \mathbf{grad} u) \leq 0 &\implies \max_{x \in \partial\Omega} u(x) = \max_{x \in \Omega} u(x). \end{aligned}$$

The message of Thm. 3.7.1.2 for the special case $\kappa \equiv 1$, $f \equiv 0$ is the following maximum principle for harmonic functions:

If $\Delta u = 0$ in Ω , then u has its maximum and minimum on the boundary $\partial\Omega$

a harmonic function over the disk \triangleright

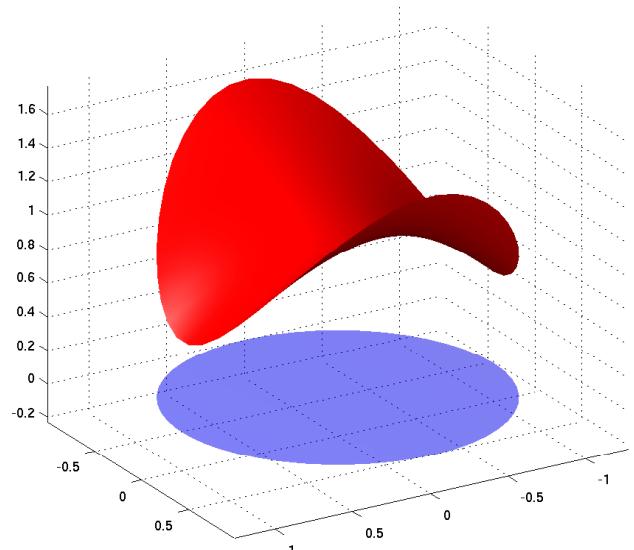


Fig. 244

Proof. ①: case $-\operatorname{div}(\kappa(x) \operatorname{grad} u) = 0$

Section 1.2.3 ➤ u solves quadratic minimization problem

$$u = J(v) := \operatorname{argmin}_{\substack{v \in H^1(\Omega) \\ v=g \text{ on } \partial\Omega}} \int_{\Omega} \kappa(x) \|\operatorname{grad} v(x)\|^2 dx .$$

If u had a global maximum at x^* in the interior of Ω , that is

$$\exists \delta > 0: u(x^*) \geq \max_{x \in \partial\Omega} u(x) + \delta .$$

Now “chop off” the maximum and define

$$w(x) := \min\{u(x), u(x^*) - \delta\}, \quad x \in \Omega . \quad (3.7.1.3)$$

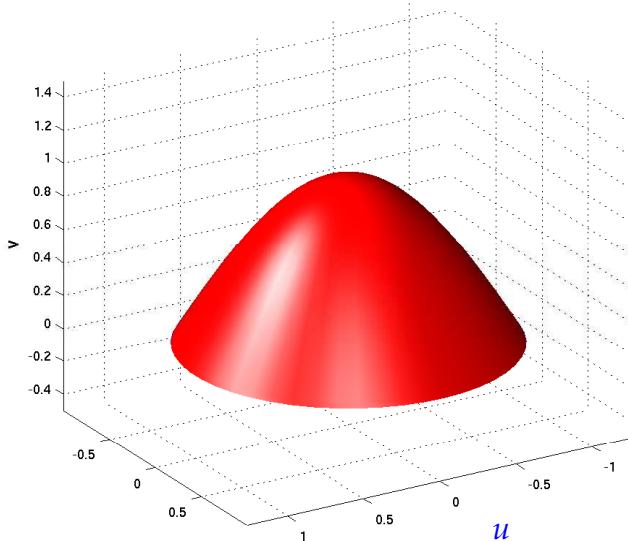


Fig. 245

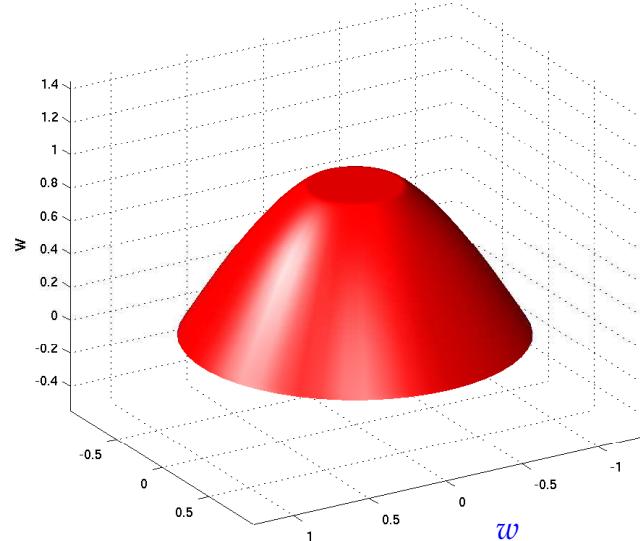


Fig. 246

Since the “flat top” of w does not contribute to the energy functional J , whereas u makes a positive contribution in that zone, we conclude

$$\int_{\Omega} \kappa(x) \|\operatorname{grad} u(x)\|^2 dx \geq \int_{\Omega} \kappa(x) \|\operatorname{grad} w(x)\|^2 dx .$$

Obviously, $w \in C^0(\overline{\Omega})$, and as a continuous function which is piecewise in H^1 the function w will also belong to $H^1(\Omega)$ (→ Thm. 1.3.4.23). However

$$\int_{\Omega} \kappa(x) \|\operatorname{grad} w(x)\|^2 dx < \int_{\Omega} \kappa(x) \|\operatorname{grad} u(x)\|^2 dx ,$$

which contradicts the definition of u as the global minimizer of the quadratic energy functional.

②: case $f := -\operatorname{div}(\kappa(x) \operatorname{grad} u) < 0$

Section 1.2.3 ➤ u solves quadratic minimization problem

$$u = \operatorname{argmin}_{\substack{v \in H^1(\Omega) \\ v=g \text{ on } \partial\Omega}} \int_{\Omega} \frac{1}{2} \kappa(x) \|\operatorname{grad} v(x)\|^2 - f(x) u(x) dx .$$

The function w from (3.7.1.3) satisfies $w \leq u$. Thus

$$\int_{\Omega} \underbrace{-f(x)}_{\geq 0} u(x) dx \geq \int_{\Omega} \underbrace{-f(x)}_{\geq 0} w(x) dx .$$

Hence, again w realizes a smaller value of the energy functional than u . \square

3.7.2 Maximum Principle for Piecewise Linear Lagrangian FEM

Now we consider a finite element Galerkin discretization of

$$-\operatorname{div}(\kappa(x) \operatorname{grad} u) = f \quad \text{in } \Omega , \quad u = g \quad \text{on } \partial\Omega ,$$

(with weak form (3.7.1.1)) by means of linear Lagrangian finite elements (\rightarrow Section 2.6). To enforce the non-homogeneous Dirichlet boundary conditions we use offset functions supported near $\partial\Omega$ as explained in Section 2.7.6. We obtain the finite element Galerkin solution $u_h \in \mathcal{S}_1^0(\mathcal{M}) \subset C^0(\overline{\Omega})$.

The key question is: Does u_h satisfy a **maximum principle**, that is, can we conclude

$$\begin{aligned} f \geq 0 &\implies \min_{x \in \partial\Omega} u_h(x) = \min_{x \in \Omega} u_h(x) , \\ f \leq 0 &\implies \max_{x \in \partial\Omega} u_h(x) = \max_{x \in \Omega} u_h(x) ? \end{aligned} \tag{3.7.2.1}$$

Note, how this mirrors the assertions of Thm. 3.7.1.2. If the answer to the questions is “yes”, we say that the FEM based on $\mathcal{S}_1^0(\mathcal{M})$ satisfies a **discrete maximum principle**.

§3.7.2.2 (Discrete maximum principle on tensor-product mesh)

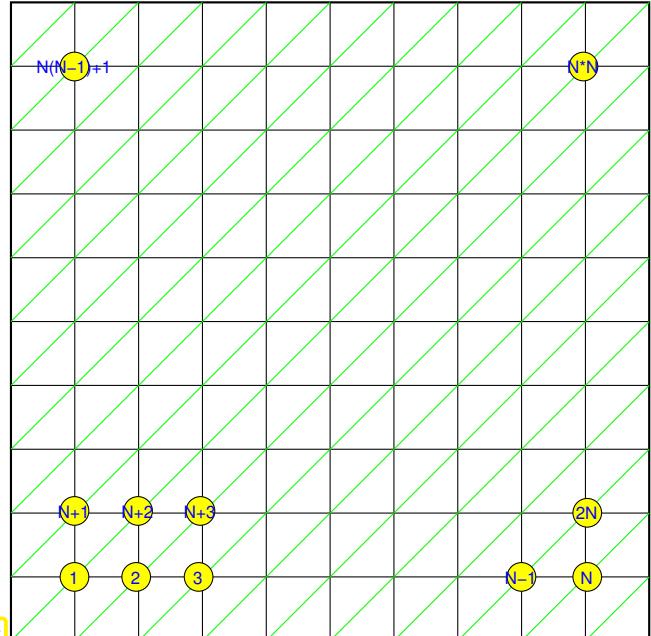
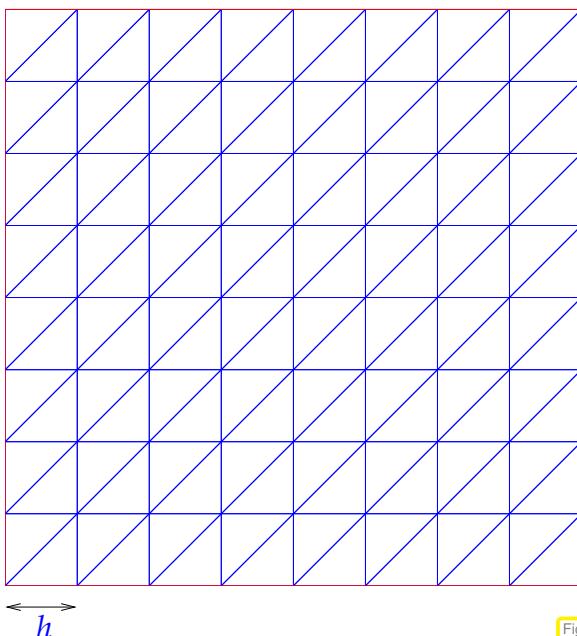
We consider the Galerkin finite element discretization of the non-homogeneous Dirichlet boundary value problem

$$\begin{aligned} -\Delta u &= 0 \quad \text{in } \Omega := [0,1]^2 , \\ u &= g \quad \text{on } \partial\Omega , \end{aligned}$$

on an $M \times M$ “tensor-product triangular mesh” with equidistant cell spacing h in both directions, see Fig. 247,

$$\mathcal{M} := \left\{ \begin{array}{l} \operatorname{convex}\left\{ \begin{bmatrix} (i-1)h \\ (j-1)h \end{bmatrix}, \begin{bmatrix} ih \\ (j-1)h \end{bmatrix}, \begin{bmatrix} ih \\ jh \end{bmatrix} \right\}, \\ \operatorname{convex}\left\{ \begin{bmatrix} (i-1)h \\ (j-1)h \end{bmatrix}, \begin{bmatrix} (i-1)h \\ jh \end{bmatrix}, \begin{bmatrix} ih \\ jh \end{bmatrix} \right\}, \end{array} \quad 1 \leq i, j \leq M \right\} , \quad M \in \mathbb{N} .$$

This is a standard tensor-product mesh with each square $[(i-1)h, ih] \times [(j-1)h, jh]$ split into two congruent triangles.



Thanks to the cardinal basis property of the tent-function basis of $\mathcal{S}_1^0(\mathcal{M})$ with respect to the node set $\mathcal{V}(\mathcal{M}) = \{(x^i)\}_i$ of the mesh, the basis expansion coefficients of $u_h \in \mathcal{S}_1^0(\mathcal{M})$ agree with the values of u_h in the nodes:

$$u_h = \sum_{\ell=1}^N \mu_\ell b_h^\ell \Rightarrow \mu_\ell = u_h(x^\ell), \quad \ell = 1, \dots, N := \dim \mathcal{S}_1^0(\mathcal{M}).$$

Hence, the linear system of equations arising from the finite element discretization imposes relations to be satisfied by the values of u_h in neighboring nodes.

For the tensor-product mesh the nodes are conveniently indexed by tuples:

$$\mathcal{V}(\mathcal{M}) = \left\{ x^{i,j} := \begin{bmatrix} ih \\ jh \end{bmatrix}, 0 \leq i, j \leq M \right\}.$$

Since the global shape functions (tent functions) of $\mathcal{S}_1^0(\mathcal{M})$ are associated with the nodes of \mathcal{M} , we can index them in the same way. This can also be done for the basis expansion coefficients

$$u_h = \sum_{i=0}^M \sum_{j=0}^M \mu_{i,j} b_h^{i,j} \Rightarrow \mu_{i,j} = u_h(x^{i,j}) = u_h\left(\begin{bmatrix} ih \\ jh \end{bmatrix}\right), \quad 0 \leq i, j \leq M.$$

Following the approach of Section 2.4.5 we can compute the linear system of equations satisfied by the vector $\bar{\mu}$ of basis expansion coefficients. To begin with, we notice that all cells of \mathcal{M} are congruent and, thus, all element matrices agree modulo index permutations. For the triangle

$$K := \text{convex} \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} h \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ h \end{bmatrix} \right\}, \quad h > 0,$$

the formula (2.4.5.8) gives the element matrix for $-\Delta$:

$$\mathbf{A}_K = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}.$$

Then perform pen-and-paper cell-oriented assembly around a node of the mesh at $[ih, jh]^\top$, visiting the adjacent triangles, which form the support of the associated tent function. This yields a single row of the Galerkin linear system of equations:

$$4\mu_{i,j} - \mu_{i-1,j} - \mu_{i+1,j} - \mu_{i,j-1} - \mu_{i,j+1} = 0, \quad 1 \leq i, j \leq M-1, \quad (3.7.2.3)$$

where values corresponding to points on the boundary are gleaned from g :

$$\mu_{0,j} := g(0, h j) , \quad \mu_{M,j} := g(1, h j) , \quad \mu_{i,0} := g(h i, 0) , \quad \mu_{i,M} := g(h i, 1) , \quad 1 \leq i, j < M .$$

The linear relations (3.7.2.3) can be expressed graphically by means of a **stencil**, as depicted in Fig. 249. (A cluster of nodes connected by the “stencil” is marked in green in Fig. 250.)

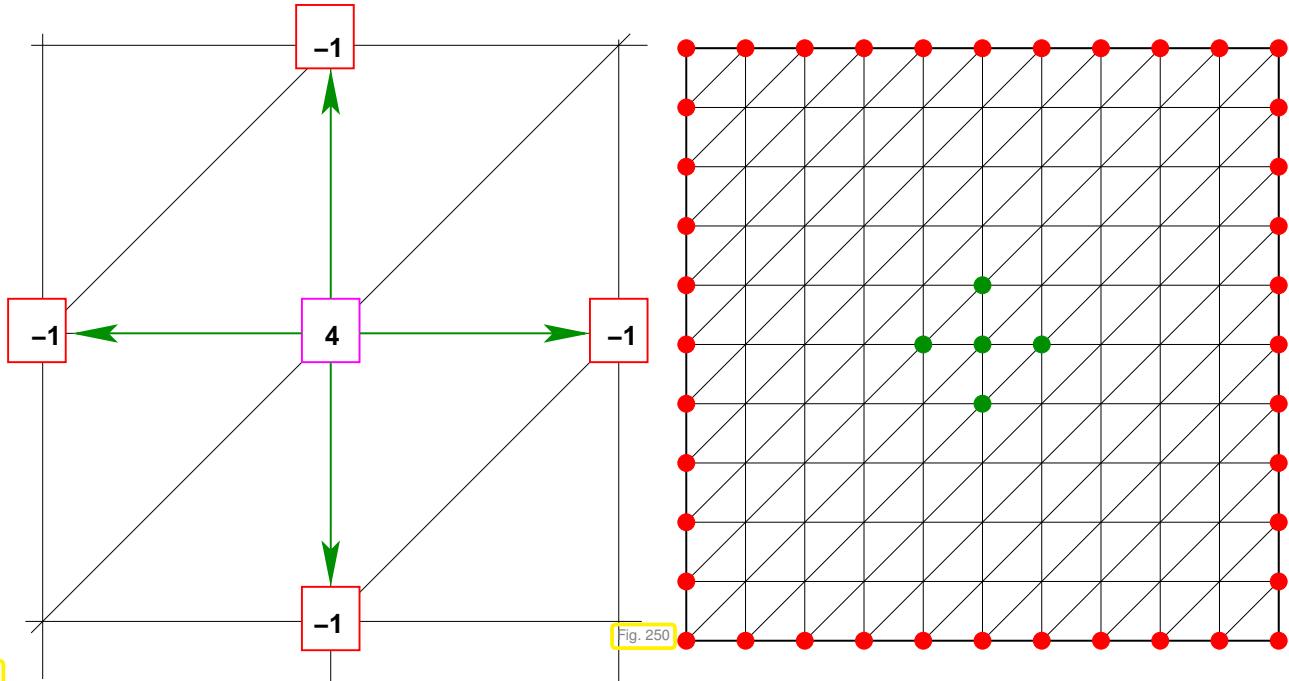


Fig. 250

The finite element solution expressed through its “nodal values” $(\mu_{i,j})_{1 \leq i,j < M}$ will attain its maximal value somewhere:

$$\exists n, m \in \{1, \dots, M-1\}: \quad \mu_{n,m} = \mu_{\max} := \max_{0 \leq i, j \leq M} \mu_{i,j} . \quad (3.7.2.4)$$

Assume: $[nh, mh]^T$ lies int the interior of $\Omega \Leftrightarrow 1 \leq n, m < M$.

Be aware of the following two facts:

$$(3.7.2.4) \Rightarrow \mu_{n-1,m}, \mu_{n+1,m}, \mu_{n,m-1}, \mu_{n,m+1} \leq \mu_{n,m} , \quad (3.7.2.5)$$

$$(3.7.2.3) \Rightarrow \mu_{n,m} = \frac{1}{4}(\mu_{n-1,m} + \mu_{n+1,m} + \mu_{n,m-1} + \mu_{n,m+1}) \quad (\text{average!}) .$$

↓← “averaging argument”

$$\mu_{n-1,m} = \mu_{n+1,m} = \mu_{n,m-1} = \mu_{n,m+1} = \mu_{n,m} !$$

The same argument can now target the neighboring grid points $((n-1)h, mh)^T, ((n+1)h, mh)^T, (nh, (m-1)h)^T, (nh, (m+1)h)^T$. By induction we find:

$$\blacktriangleright \quad \mu_{i,j} = \mu_{\max} \quad \forall 0 \leq i, j \leq M ,$$

that is, the finite difference solution has to be *constant*!

- The finite difference solution can attain its maximum in the interior only in the case of constant boundary data g !
- The discrete maximum principle is satisfied for $f = 0$. □

Remark 3.7.2.6 (Importance of discrete maximum principle) Discretizations that satisfy the maximum principles will be *positivity preserving*: they yield non-negative solutions for non-negative sources and

boundary values (Why?). This can be essential, when we want to compute a quantity that must never drop below zero, like a density or absolute temperatures. \square

§3.7.2.7 (Maximum principle for $\mathcal{S}_1^0(\mathcal{M})$ -FEM on triangular meshes) Now we try to generalize the considerations of the previous paragraph to the discretization by means of *linear Lagrangian finite elements* (space $\mathcal{S}_1^0(\mathcal{M}) \subset H^1(\Omega)$) on a triangular mesh (of a polygonal domain $\Omega \subset \mathbb{R}^2$) see Section 2.4.

Here: $\tilde{\mathbf{A}} \in \mathbb{R}^{M,M} \doteq \mathcal{S}_1^0(\mathcal{M})$ -Galerkin matrix for \mathbf{a} from (3.7.1.1) ($M := \#\mathcal{V}(\mathcal{M})$)

A row of this matrix connects all nodal values $\mu_j = u_h(\mathbf{x}^j)$ of the finite element Galerkin solution $u_h \in \mathcal{S}_1^0(\mathcal{M})$ according to

$$(\tilde{\mathbf{A}})_{ii}\mu_i + \sum_{j \neq i} (\tilde{\mathbf{A}})_{ij}\mu_j = (\vec{\varphi})_i, \quad \mathbf{x}^i \text{ interior node},$$

where $\mu_j := g(\mathbf{x}^j)$ for $\mathbf{x}^j \in \partial\Omega$.

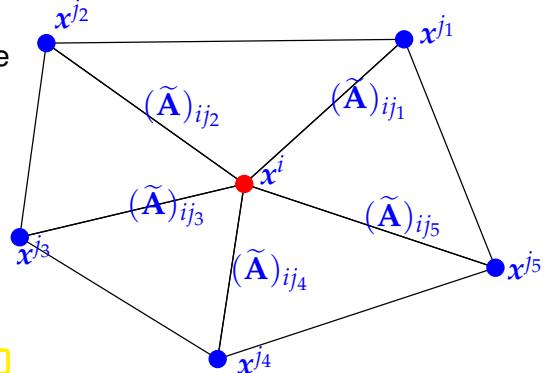


Fig. 251

This formula holds even in the case of Dirichlet boundary conditions, as can be seen from the first row of (2.7.6.12) or (2.7.6.14).

Next we note that the components of the load vector $\vec{\varphi}$ inherit the sign of f , because the nodal basis functions for $\mathcal{S}_1^0(\mathcal{M})$ (\rightarrow Section 2.4.3) are non-negative.

$$(\vec{\varphi})_i = \int_{\Omega} f(\mathbf{x}) b_h^i(\mathbf{x}) \, d\mathbf{x} \Rightarrow \begin{cases} f \geq 0 & \Rightarrow (\vec{\varphi})_i \geq 0 \ \forall i, \\ f = 0 & \Rightarrow (\vec{\varphi})_i = 0 \ \forall i, \\ f \leq 0 & \Rightarrow (\vec{\varphi})_i \leq 0 \ \forall i. \end{cases}$$

The above averaging argument from § 3.7.2.2 carries over, if the entries of $\tilde{\mathbf{A}}$ satisfy the following conditions:

- $(\tilde{\mathbf{A}})_{ii} > 0$ (positive diagonal), (3.7.2.8)
- $(\tilde{\mathbf{A}})_{ij} \leq 0$ for $j \neq i$ (non-positive off-diagonal entries), (3.7.2.9)
- $\sum_j (\tilde{\mathbf{A}})_{ij} = 0$, if \mathbf{x}^i is interior node . (3.7.2.10)

(Recall [Hip19, ??]: A matrix $\tilde{\mathbf{A}}$ satisfying (3.7.2.8)–(3.7.2.10) is called **diagonally dominant**.)

Averaging argument: For an interior vertex \mathbf{x}^i is μ_i a **convex combination** of the nodal values in adjacent vertices,

$$\mu_i = \sum_{j \neq i} \omega_j \mu_j, \quad \omega_j > 0, \quad \sum_{j \neq i} \omega_j = 1, \text{ since } \omega_j := -\frac{(\tilde{\mathbf{A}})_{ij}}{(\tilde{\mathbf{A}})_{ii}}.$$

$$\blacktriangleright \quad \min_{j \neq i} \mu_j \leq \mu_i \leq \max_{j \neq i} \mu_j,$$

where the index j always runs through all the vertices for which $(\tilde{\mathbf{A}})_{ij} \neq 0$.

averaging argument \blacktriangleright $u_h(\mathbf{x}^i) = \max_{y \in \mathcal{V}(\mathcal{M})} u_h(y)$ can only hold for an interior node \mathbf{x}^i , if $u_h \equiv \text{const.}$

- Since $u_h \in \mathcal{S}_1^0(\mathcal{M})$ attains its extremal values at nodes of the mesh, the maximum principle holds for it in the case $f = 0$ provided that (3.7.2.8)–(3.7.2.10) are satisfied.

More general case $f \leq 0 \Rightarrow (\vec{\varphi})_i \leq 0$:

Then the averaging argument again rules out the existence of an interior maximum for a non-constant solution. The case $f \geq 0$ follows similarly. □

Remark 3.7.2.11 (M-matrices) The sign condition (3.7.2.8)–(3.7.2.10) for the extended Galerkin matrix $\tilde{\mathbf{A}}$ mean that the Galerkin matrix $\mathbf{A} \in \mathbb{R}^{N,N}$, $N := \dim \mathcal{S}_{1,0}^0(\mathcal{M})$, with respect to the finite element space $\mathcal{S}_{1,0}^0(\mathcal{M})$ will satisfy

- $(\mathbf{A})_{ii} > 0$ (positive diagonal), (3.7.2.12)

- $(\mathbf{A})_{ij} \leq 0$ for $j \neq i$ (non-positive off-diagonal entries), (3.7.2.13)

- $\sum_j (\mathbf{A})_{ij} \geq 0$ (non-negative row sums). (3.7.2.14)

These conditions are easy to verify and already supply a simple criterion for the invertibility of a matrix.

Lemma 3.7.2.15. Invertibility of “averaging matrices”

If a matrix $\mathbf{A} \in \mathbb{R}^{N,N}$ satisfies the sign conditions (3.7.2.12)–(3.7.2.14), and

(i) $\sum_j (\mathbf{A})_{ij} > 0$ for a single $i \in \{1, \dots, N\}$,

(ii) the graph of \mathbf{A} is connected,

then \mathbf{A} is regular.

Proof. Pick $\vec{\xi} = [\xi_1, \dots, \xi_h]^\top \in \mathbb{R}^N$ with $\mathbf{A}\vec{\xi} = \mathbf{0}$. Let

$$\ell \in \{j \in \{1, \dots, N\} : |\xi_j| = \max_{i=1, \dots, N} |\xi_i|\}$$

be the index of the largest (in modulus) component of $\vec{\xi}$. Without loss of generality assume $\xi_\ell \geq 0$. Since

$$(\mathbf{A})_{\ell,\ell} \xi_\ell + \sum_{j \in \mathcal{U}_\ell} (\mathbf{A})_{\ell,j} \xi_j = 0, \quad \mathcal{U}_\ell := \{j \neq \ell : (\mathbf{A})_{\ell,j} \neq 0\},$$

we conclude from (3.7.2.12), (3.7.2.13) that

$$\xi_\ell = \sum_{j \in \mathcal{U}_\ell} \frac{|(\mathbf{A})_{\ell,j}|}{|(\mathbf{A})_{\ell,\ell}|} \xi_j.$$

As, owing to (3.7.2.14),

$$\sum_{j \in \mathcal{U}_\ell} \frac{|(\mathbf{A})_{\ell,j}|}{|(\mathbf{A})_{\ell,\ell}|} \leq 1, \quad (3.7.2.16)$$

the maximality of ξ_ℓ can hold only if

$$\xi_j = \xi_\ell \quad \text{for all } j \in \mathcal{U}_\ell.$$

Since the graph of \mathbf{A} is connected, this implies that $\vec{\xi}$ has constant components. Recall that for at least one row the sum of the matrix entries is strictly positive, which implies $\vec{\xi} = \mathbf{0}$. □

Theorem 3.7.2.17. Inverse positivity

The inverse of a matrix $\mathbf{A} \in \mathbb{R}^{N,N}$ satisfying the assumptions of Lemma 3.7.2.15 has **no negative entries**.

Proof. Pick a vector $\vec{\rho} \in \mathbb{R}^N$ with $\rho_k \geq 0$ for all $k \in \{1, \dots, N\}$ and let $\vec{\xi} \in \mathbb{R}^N$ satisfy $\mathbf{A}\vec{\xi} = \vec{\rho}$. Let $\ell \in \{1, \dots, N\}$ be the index of the smallest (with sign) component of $\vec{\xi}$ that is

$$\xi_\ell \leq \xi_j \quad \forall j \in \{1, \dots, N\} .$$

From (3.7.2.12)–(3.7.2.13) and (3.7.2.16) we infer

$$\xi_\ell = \rho_\ell + \sum_{j \in \mathcal{U}_\ell} \frac{|(\mathbf{A})_{\ell,j}|}{(\mathbf{A}_{\ell,\ell})} \xi_j \geq \sum_{j \in \mathcal{U}_\ell} \frac{|(\mathbf{A})_{\ell,j}|}{(\mathbf{A}_{\ell,\ell})} \xi_j \geq \xi_\ell \cdot \underbrace{\sum_{j \in \mathcal{U}_\ell} \frac{|(\mathbf{A})_{\ell,j}|}{(\mathbf{A}_{\ell,\ell})}}_{\leq 1} \geq 0 .$$

If $\sum_j (\mathbf{A})_{\ell,j} > 0$, we immediately conclude $\xi_\ell \geq 0$. Otherwise, repeating the reasoning in the proof of Lemma 3.7.2.15, we conclude $\xi_j = \xi_\ell$ for all $j \in \mathcal{U}_\ell$. Marching through the components, we finally reach the index for which the matrix row sum is strictly positive and then get $\xi_\ell \geq 0$. \square

Matrices with positive inverse, are not only relevant for discrete maximum principles, but also in many other modeling contexts, in particular in statistics and optimization.

Definition 3.7.2.18. M-matrix, [Hac94, Sect. 6.4]

An invertible matrix satisfying (3.7.2.12)–(3.7.2.14) is called an **M-matrix**.

§3.7.2.19 (Angle condition for (3.7.2.8)–(3.7.2.10)) When will (3.7.2.8)–(3.7.2.10) hold for $\mathcal{S}_1^0(\mathcal{M})$ -Galerkin matrix?

First consider the simple case

$$\kappa \equiv 1, \quad \Leftrightarrow \quad -\Delta u = f$$

The linear finite element discretization of this BVP was scrutinized in Section 2.4. There we also derived the following formula for the $\mathcal{S}_1^0(\mathcal{M})$ -element matrix for a general triangle

$$\mathbf{A}_K = \frac{1}{2} \begin{bmatrix} \cot \omega_3 + \cot \omega_2 & -\cot \omega_3 & -\cot \omega_2 \\ -\cot \omega_3 & \cot \omega_3 + \cot \omega_1 & -\cot \omega_1 \\ -\cot \omega_2 & -\cot \omega_1 & \cot \omega_2 + \cot \omega_1 \end{bmatrix}, \quad (2.4.5.8)$$

where ω_ℓ is the interior angle at vertex ℓ .

From formula (2.4.5.8) & “assembly” as in Fig. 77 we obtain

$$(\tilde{\mathbf{A}})_{ij} = -\cot \alpha - \cot \beta = -\frac{\sin(\alpha + \beta)}{\sin \alpha \sin \beta} .$$

\Downarrow

$$(\tilde{\mathbf{A}})_{ij} \leq 0 \Leftrightarrow \alpha + \beta < \pi .$$

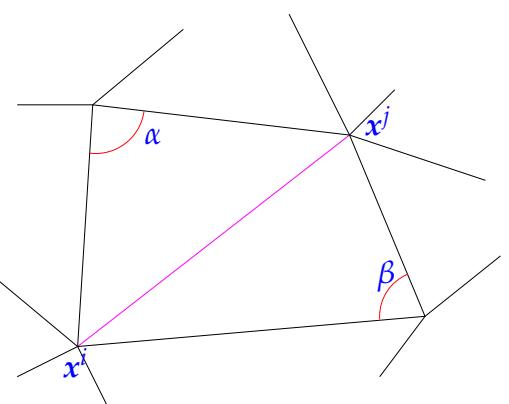


Fig. 252

Moreover

$$\sum_{x \in \mathcal{V}(\mathcal{M})} b_h^x \equiv 1 \Rightarrow \sum_j (\tilde{\mathbf{A}})_{ij} = 0 \quad (\Leftrightarrow (3.7.2.10)).$$

The condition (3.7.2.8) $\Leftrightarrow (\tilde{\mathbf{A}})_{ii} > 0$ is straightforward.

Theorem 3.7.2.20. Maximum principle for linear FE solution of Poisson equation

The linear finite element solution of

$$-\Delta u = 0 \quad \text{in } \Omega \subset \mathbb{R}^2, \quad u = g \quad \text{on } \partial\Omega,$$

on a triangular mesh \mathcal{M} satisfies the maximum principle (3.7.2.1), if \mathcal{M} is a Delaunay triangulation.

Remark 3.7.2.21 (Maximum principle for linear FE for 2nd-order elliptic BVPs) For the $\mathcal{S}_1^0(\mathcal{M})$ -Galerkin discretization of a general second-order elliptic scalar Dirichlet boundary value problem (3.7.1.1) on a triangular mesh, the conditions (3.7.2.8)–(3.7.2.10) are fulfilled,

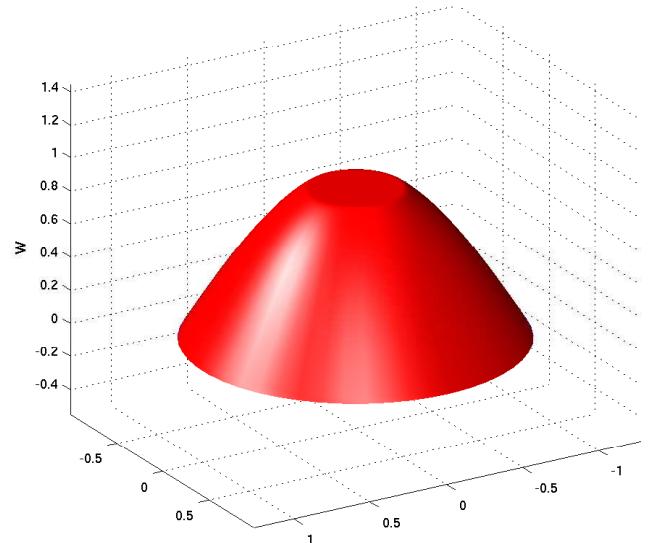
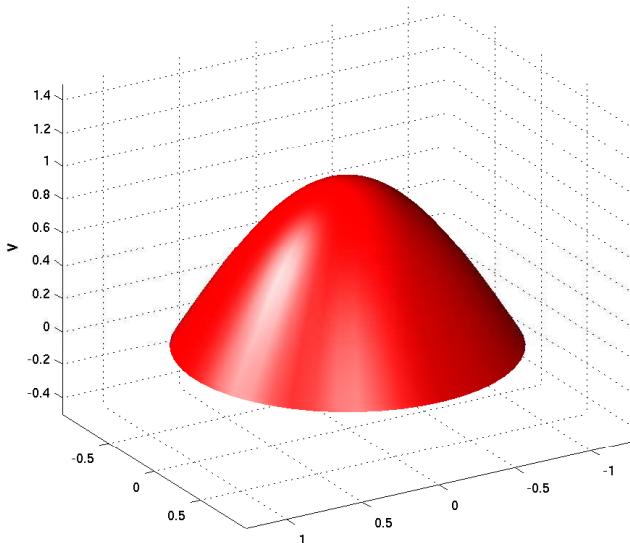
$$\text{if all angles of triangles of } \mathcal{M} \leq \frac{\pi}{2}.$$

Remark 3.7.2.22 (Maximum principle for higher order Lagrangian FEM) Even when using p -degree Lagrangian finite elements with nodal basis functions associated with interpolation nodes, see Section 2.6.1, the discrete maximum principle will fail to hold on *any mesh* for $p > 1$.

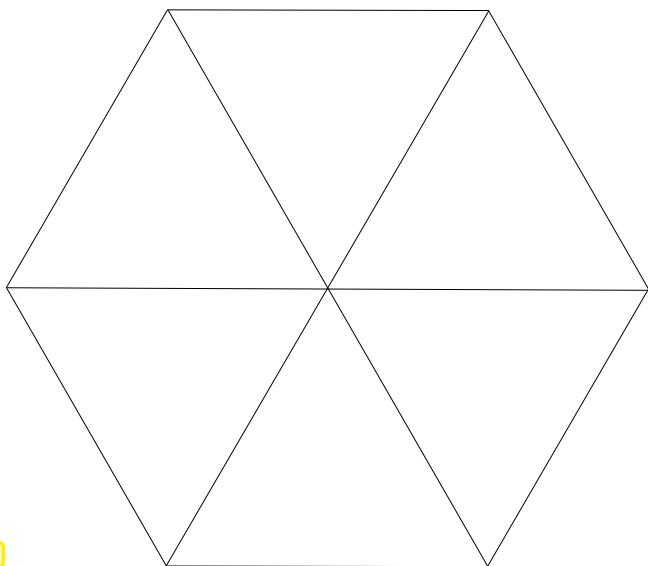
Review question(s) 3.7.2.23 (Discrete maximum principle)

(Q3.7.2.23.A) [Maximum principle in electrostatics] Why will a point charge moving in an electrostatic field never come to rest?

(Q3.7.2.23.B) [Visual proof of maximum principle] Explain the “visual proof” of the maximum principle for harmonic functions given in the following plots.



(Q3.7.2.23.C) [Triangular lattic mesh]



We consider the Galerkin discretization of a Dirichlet boundary value problem by means of linear Lagrangian finite elements on a connected lattice-type mesh with equilateral triangular cells.

The variational problem is translation- and rotation-invariant so that all element matrices are the same and of the form

$$\mathbf{A}_K = \begin{bmatrix} \alpha & \beta & \beta \\ \beta & \alpha & \beta \\ \beta & \beta & \alpha \end{bmatrix}.$$

Formula sufficient and necessary conditions on the matrix entries $\alpha, \beta \in \mathbb{R}$ so that the resulting Galerkin matrix is an **M-matrix**.

Definition 3.7.2.18. M-matrix

An invertible matrix satisfying

- $(\mathbf{A})_{ii} > 0$ (positive diagonal) , (3.7.2.12)

- $(\mathbf{A})_{ij} \leq 0$ for $j \neq i$ (non-positive off-diagonal entries) , (3.7.2.13)

- $\sum_j (\mathbf{A})_{ij} \geq 0$ (non-negative row sums) . (3.7.2.14)

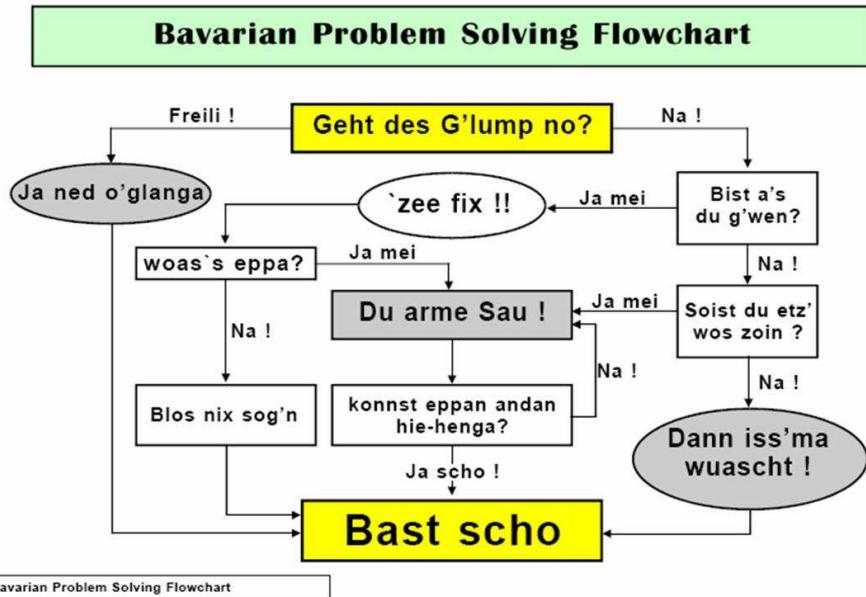
is called an **M-matrix**.

△

3.8 Validation and Debugging of Finite Element Codes

In this section you will learn about an important application of a priori finite element convergence results which you will never find mentioned in any textbook: the detection of programming errors (“debugging”) in finite element codes. On one hand, whenever, for a well-defined numerical experiment the observed convergence rates are worse than those predicted by theory, the code must be faulty. On the other hand, convergence matching theory is *circumstantial evidence* (no proof, however) for the correctness of the implementation.

Also applies to debugging
a code



§3.8.0.1 (The code under scrutiny (“model problem”)) At our disposal is a code that implements a Lagrangian finite element discretization (→ Section 2.6) of general scalar linear second-order elliptic variational problems (→ Section 1.8, Section 1.9) on domains $\Omega \subset \mathbb{R}^d$, $d = 2, 3$:

$$\begin{aligned} u \in H^1(\Omega) : \quad & \int_{\Omega} \alpha(x) \operatorname{grad} u \cdot \operatorname{grad} v + \gamma(x) u v \, dx + \int_{\Gamma_R} \lambda(x) u v \, dS(x) \\ &= \int_{\Omega} f v \, dx + \int_{\Gamma_N} h v \, dS(x) \quad \forall v \in H_{\Gamma_D}^1(\Omega), \quad (3.8.0.2) \end{aligned}$$

where, based on a partition $\partial\Omega = \bar{\Gamma}_D \cup \bar{\Gamma}_N \cup \Gamma_R$, the test space is the Sobolev space (→ Def. 1.3.4.3)

$$H_{\Gamma_D}^1(\Omega) := \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D \right\}. \quad (3.8.0.3)$$

Data are the coefficients $\alpha : \Omega \rightarrow \mathbb{R}^{d,d}$, $\gamma : \Omega \rightarrow \mathbb{R}$, $\lambda : \Gamma_R \rightarrow \mathbb{R}$, an the source functions $f \in L^2(\Omega)$, $h \in L^2(\Gamma_N)$, all or some of them usually given in procedural form as discussed in Rem. 2.1.2.5.

- (3.8.0.2) is the variational formulation for a boundary value problem with mixed Dirichlet, Neumann, and Robin boundary conditions as in Ex. 1.7.0.10.

The following possibilities are available for the sake of testing:

- The source function $f \in L^2(\Omega)$, Dirichlet data $g \in C^0(\Gamma_D)$, Neumann data $h \in L^2(\Gamma_N)$, coefficient functions $\alpha : \Omega \rightarrow \mathbb{R}^{d,d}$ (uniformly positive definite → Def. 1.2.2.9), $\gamma : \Omega \rightarrow \mathbb{R}_0^+$, $\lambda : \partial\Omega \rightarrow \mathbb{R}_0^+$ can be set within the code by defining suitable function classes.
- The code can handle general simplicial meshes (which may be read from file, see Section 2.7.1). The mesh implicitly defines the domain Ω .
- The code can compute the Galerkin finite element solution of (3.8.0.2) based on the Lagrangian finite element trial and test space $V_{0,h} := \mathcal{S}_p^0(\mathcal{M}) \cap H_{\Gamma_D}^1(\Omega)$ (→ Def. 2.6.1.1) for *fixed uniform* local polynomial degree $p \in \mathbb{N}$.

Note that the techniques presented in this section are applicable to finite element discretization of variational problems way beyond this model setting.

Task:

- ◆ Code **validation**: gather evidence for the correctness of the code.
- ◆ Code **debugging**: detect and locate errors in the code.

It will turn out that *asymptotic estimates* for error norms as provided by (3.3.5.10), Thm. 3.3.5.6 and in Section 3.6.3 are *key tools* for tackling this task. (This is another reason why finite element convergence theory is relevant for anyone programming finite element methods.)

For testing we will take for granted the availability of sequences of meshes $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \dots$, which satisfy (see § 3.3.5.15 for related requirements)

- 1) that the meshwidth decreases geometrically: $h_k = q h_{k-1}$ for some $0 < q < 1$, where h_k is the meshwidth of \mathcal{M}_k .
- 2) that all cells of \mathcal{M}_k have about the same size h_k . This feature is called **quasi-uniformity**.
- 3) that the shape regularity measure (\rightarrow Def. 3.3.2.20) all meshes stays below a common bound, a property called **uniform shape regularity**.

Note that Item 1 & Item 3 imply that the number of cells increases in geometric progression: $\#\mathcal{M}_k = \sigma \#\mathcal{M}_{k-1}$ for some $\sigma > 1$ (usually $\sigma = 4$ in 2D),

Sequences of meshes complying with the above requirements can, for instance, be generated by successive **(global) regular refinement** of a coarse initial mesh, see Ex. 3.1.4.3. Refer to Ex. 3.1.4.4 for how to conduct regular refinement in LEHRFEM++. Also **Gmsh** provides a menu item which triggers global regular refinement of the current mesh. Thus suitable sequences of input meshes can be generated.



Simple global regular refinement may sometimes create meshes endowed with “too much structure” to observe “generic convergence behavior”.

In this case small random perturbations of vertex positions (mesh jiggling) can restore “truly unstructured meshes”.

Sequences of meshes with the above properties were used in the numerical experiments of Section 3.2 and in Exp. 3.5.1.2, Exp. 3.5.2.1, Exp. 3.6.1.6, Exp. 3.6.2.3, Exp. 3.6.3.1.

§3.8.0.4 (Observing asymptotic convergence) As explained in Rem. 3.3.5.24 we expect **algebraic convergence** of the energy norm (and of the $L^2(\Omega)$ -norm as well) of the discretization error in terms of the dimension of the finite element space.

We **assume**: asymptotic convergence estimates are known and *sharp*, cf. § 3.3.5.23: with a possibly unknown convergence rate $\alpha > 0$ we have for a targeted norm $\|\cdot\|$

$$\exists C = C(u, \dots) > 0: \|u - u_h\| \approx CN^{-\alpha} \quad \forall \mathcal{M}_k. \quad (3.8.0.5)$$

(u $\hat{=}$ exact solution, u_h $\hat{=}$ finite element Galerkin solution, \approx $\hat{=}$ “approximate equality”; lower and upper bound with two (slightly) different constants ≈ 1)

According to our assumptions on the sequence of meshes, by § 3.3.5.17, the dimensions $N_k := \dim \mathcal{S}_p^0(\mathcal{M}_k)$ will also grow in geometric progression ($\kappa = 4$ for 2D triangular mesh)

$$N_k \approx \kappa N_{k-1} \quad \text{for some } \kappa > 1 \Rightarrow N_k \approx \kappa^k N_0. \quad (3.8.0.6)$$

Write u_k for the finite element Galerkin solution on \mathcal{M}_k , combine (3.8.0.5) and (3.8.0.6) and use the \triangle -inequality

$$\|u_k - u_{k-1}\| \leq \|u_k - u\| + \|u - u_{k-1}\| \approx CN_0 \left(\kappa^{-k\alpha} + \kappa^{-(k-1)\alpha} \right) \approx C' N_k^{-\alpha}, \quad (3.8.0.7)$$

with a constant $C' > 0$ independent of N_k .

Measured norms of differences of Galerkin solutions of consecutive meshes in the sequence should display **algebraic convergence** for $N_k \rightarrow \infty$.

Consult § 3.2.2.5 for instructions on how to recognize algebraic convergence with sequences of empirical error norms.

Caveat: Computing $\|u_k - u_{k-1}\|$ entails forming the difference of finite element functions on different meshes.

For validation purposes we can circumvent the need to compute $u_k - u_{k-1}$ exploiting the \triangle -inequality, because for every norm on a function space

$$\|\|u_k\| - \|u_{k-1}\|\| \leq \|u_k - u_{k-1}\| \stackrel{(3.8.0.7)}{\approx} CN_k^{-\alpha}. \quad (3.8.0.8)$$

Hence, also the differences of norms on different levels of refinement should display algebraic convergence with the same rate as $\|u - u_k\|$ for $N_k \rightarrow \infty$. \square

§3.8.0.9 (Method of manufactured solutions \rightarrow [SK00]) This technique has widely been used in numerical experiments exploring the asymptotic behavior of norms of discretization errors, as in Section 3.2, Exp. 3.5.1.2, and many more. The method usually involves the following steps:

- ① Pick a simple domain Ω (polygon in 2D) that allows exact triangulation with straight edges.
- ② Choose *smooth* exact solution $u \in C^\infty(\bar{\Omega})$ with a simple analytic expression (*) and compute corresponding source function f , boundary data g , and coefficient λ (analytically from the strong form of the BVP). Symbolic computation (Mathematica, MAPLE) should be used.
- ③ Choose coefficient functions α , γ , and λ given by simple analytic expressions; start with constants.
- ④ Solve the resulting “manufactured BVP” on a sequence $(\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_L)$ of meshes as introduced above.
- ⑤ Compute the finite element Galerkin solutions $u_k \in \mathcal{S}_p^0(\mathcal{M}_k)$ on mesh \mathcal{M}_k , $k = 0, \dots, m$, and the norms $\|u - u_k\|$ of the discretization errors. Use “overkill quadrature” for computation of local error norms, see Rem. 3.2.3.3.
- ⑥ Estimate the rate of algebraic convergence (\rightarrow Def. 3.2.2.1) following the recipe in § 3.2.2.5, Code 3.2.2.8, and plot the errors versus meshwidths in doubly logarithmic scale. Ignore coarse meshes if they give rise to “outliers” due to pre-asymptotic effects as in Exp. 3.2.3.12.
- ⑦ If the measured rate well matches the predicted rate from (3.3.5.25)

\Rightarrow code has passed test



Beware of polynomial exact solutions $u \in \mathcal{P}_p$! (Why?) On the other hand, if the above test fails for non-polynomial u , the next step should be to probe $u \in \mathcal{P}_p$ (Why?). \square

§3.8.0.10 (Direct testing of (bi-)linear forms) This approach can be used to examine specific parts of the variational formulation. We abbreviate with $\mathbf{b}(\cdot, \cdot)$ a **continuous** and **symmetric** bilinear form on $H^1(\Omega)$, with ℓ a **continuous** (\rightarrow Def. 1.2.3.42) linear form on $H^1(\Omega)$. They satisfy

$$\exists C_r > 0: \quad \ell(v) \leq C_r \|v\|_{H^1(\Omega)} \quad \forall v \in H^1(\Omega), \quad (3.8.0.11)$$

$$\exists C_c > 0: \quad \mathbf{b}(v, w) \leq C_c \|v\|_{H^1(\Omega)} \|w\|_{H^1(\Omega)} \quad \forall v, w \in H^1(\Omega). \quad (3.8.0.12)$$

The Galerkin matrix for \mathbf{b} and the right hand side vector associated with ℓ can be tested through the following steps:

- ① Pick a simple domain Ω (polygon in 2D) that allows exact triangulation with straight edges.
- ② Choose a **smooth** function $w \in C^\infty(\bar{\Omega})$ that is **not** a polynomial.
- ③ Compute $\mathbf{b}(w, w)$ and $\ell(w)$ **exactly**, that is analytically, which is often feasible, if Ω is a square or a circle. Symbolic computation (Mathematica, MAPLE) is advisable.
- ④ With the finite element code evaluate $\mathbf{l}_k w$, where $\mathbf{l}_k : C^0(\bar{\Omega}) \rightarrow \mathcal{S}_p^0(\mathcal{M}_k)$ is the local nodal interpolation operator as introduced in § 3.3.5.2, (3.3.5.3). Write $\vec{v}_k \in \mathbb{R}^{N_k}$ for the vector of basis expansion coefficients of $\mathbf{l}_k w$.
- ⑤ Use the code to compute the $\mathcal{S}_p^0(\mathcal{M}_k)$ -Galerkin matrix $\mathbf{B}_k \in \mathbb{R}^{N_k \times N_k}$ for \mathbf{b} . Also compute the vector $\vec{\rho}_k \in \mathbb{R}^{N_k}$ arising from the $\mathcal{S}_p^0(\mathcal{M}_k)$ -Galerkin discretization of ℓ .
- ⑥ Using the asymptotic interpolation error estimates of Thm. 3.3.5.6 and the continuity of \mathbf{b} we conclude for $h_k \rightarrow 0$

$$\begin{aligned} \mathbf{b}(w, w) - \vec{v}_k^\top \mathbf{B}_k \vec{v}_k &= \mathbf{b}(w, w) - \mathbf{b}(\mathbf{l}_k w, \mathbf{l}_k w) = \mathbf{b}(w + \mathbf{l}_k w, w - \mathbf{l}_k w) \\ &\stackrel{(3.8.0.12)}{\leq} C_c \|w + \mathbf{l}_k w\|_{H^1(\Omega)} \|w - \mathbf{l}_k w\|_{H^1(\Omega)} \\ &\leq C_c \|w\|_{H^1(\Omega)} \left(1 + Ch_k^p \|w\|_{H^{p+1}(\Omega)}\right) \left(Ch_k^p \|w\|_{H^{p+1}(\Omega)}\right) = O(h_k^p). \end{aligned}$$

Again, we invoke Thm. 3.3.5.6 and continuity:

$$\begin{aligned} \ell(w) - \vec{v}_k^\top \vec{\rho}_k &= \ell(w) - \ell(\mathbf{l}_k w) = \ell(w - \mathbf{l}_k w) \stackrel{(3.8.0.11)}{\leq} C_r \|w - \mathbf{l}_k w\|_{H^1(\Omega)} \\ &\leq C_r Ch_k^p \|w\|_{H^{p+1}(\Omega)} = O(h_k^p). \end{aligned}$$

- ⑦ In both estimates the **values** on left hand side are readily available ($\mathbf{b}(w, w)$ and $\ell(w)$ are supposed to be known!) and theory predicts a rather precise **rate p** of **algebraic convergence** for them. If this rate materializes in empiric data

➤ code has passed test



If your code fails a test, repeat it with “simpler” w , for instance with $w \in \mathcal{P}_p(\mathbb{R}^d)$, which implies $\mathbf{b}(w, w) - \vec{v}_k^\top \mathbf{B}_k \vec{v}_k = 0$. because in this case $w \in \mathcal{S}_p^0(\mathcal{M}_k)$ for all ℓ . □

Review question(s) 3.8.0.13 (Debugging of finite-element codes)

(Q3.8.0.13.A) Let $\Omega \subset \mathbb{R}^2$ be a bounded domain, and $\alpha : \Omega \rightarrow \mathbb{R}^{2,2}$, $\gamma : \Omega \rightarrow \mathbb{R}$, and $\lambda : \partial\Omega \rightarrow \mathbb{R}$ continuous, bounded, and uniformly positive coefficient functions. What is the strong (PDE) form of the boundary value problem, whose weak form reads

$$\begin{aligned} u \in H^1(\Omega) : \quad & \int_{\Omega} \alpha(x) \mathbf{grad} u \cdot \mathbf{grad} v + \gamma(x) u v \, dx + \int_{\Gamma_R} \lambda(x) u v \, dS(x) \\ u = g \text{ on } \Gamma_D : \quad & = \int_{\Omega} f v \, dx + \int_{\Gamma_N} h v \, dS(x) \quad \forall v \in H_{\Gamma_D}^1(\Omega), \quad (3.8.0.2) \end{aligned}$$

with $f \in L^2(\Omega)$ and the Sobolev space

$$H_{\Gamma_D}^1(\Omega) := \left\{ v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D \right\}, \quad (3.8.0.3)$$

based on a partition $\partial\Omega = \bar{\Gamma}_D \cup \bar{\Gamma}_N \cup \Gamma_R$.

(Q3.8.0.13.B) We consider (3.8.0.2) for $\alpha \equiv \mathbf{I}$, $\gamma \equiv 1$, $\lambda \equiv 1$, and on the unit disk domain $\Omega := \{x \in \mathbb{R}^2 : \|x\| < 1\}$ and with $\Gamma_R := \partial\Omega$.

- (i) Is it possible to choose the data f and h such that $u(x) = \cos(\pi/2\|x\|)$ will be the exact solution of the variational problem. If not, suggests a modification that makes it possible.
- (ii) Possibly under the modification found in [(i)], determine those functions f and h that will yield that exact solution $u(x) = \cos(\pi/2\|x\|)$.

(Q3.8.0.13.C) In connection with the method of manufactured solutions you have seen the warning



Beware of polynomial exact solutions $u \in \mathcal{P}_p$! (Why?) On the other hand, if the above test fails for non-polynomial u , the next step should be to probe $u \in \mathcal{P}_p$ (Why?).

Try to answer the “Whys”.

(Q3.8.0.13.D) You have just finished the implementation of the LEHRFEM++-based C++ function

```
template <typename FUNC_ALPHA, typename FUNC_GAMMA, typename
         FUNC_BETA>
Eigen::SparseMatrix<double> compGalerkinMatrix(
    const lf::assemble::DofHandler &lfe_dofh,
    FUNC_ALPHA&& alpha, FUNC_GAMMA&& gamma, FUNC_BETA&& beta);
```

that, given a mesh \mathcal{M} , (approximately, using “safe” local numerical quadrature) computes the $\mathcal{S}_1^0(\mathcal{M})$ -based finite element Galerkin matrix for the left-hand side of the variational boundary value problem

$$\begin{aligned} u \in H^1(\Omega) : \quad & \int_{\Omega} \alpha(x) \mathbf{grad} u(x) \cdot \mathbf{grad} v(x) + \gamma(x) u(x) v(x) \, dx + \int_{\partial\Omega} \beta(x) u(x) v(x) \, dS(x) = \\ & = \int_{\Omega} f(x) v(x) \, dx \quad \forall v \in H^1(\Omega), \end{aligned}$$

where $\alpha, \gamma : \Omega \rightarrow \mathbb{R}$, $\beta : \partial\Omega \rightarrow \mathbb{R}$ are bounded coefficient functions, and $f \in L^2(\Omega)$. The use of the standard nodal basis consisting of “tent functions” is assumed.

The argument `lfe_dofh` passes the loca-to-global index mapping for $\mathcal{S}_1^0(\mathcal{M})$ and the mesh \mathcal{M} , while `alpha`, `gamma`, and `beta` are functors for the coefficient functions $x \mapsto \alpha(x)$, $x \mapsto \gamma(x)$, and $x \mapsto \beta(x)$.

Unfortunately, your implementation does not work properly. Sketch a debugging strategy based on the policy of direct testing of bilinear forms.

△

Learning Outcomes

Essential knowledge and skills acquired in this chapter:

- State, prove and understand Cea's Lemma and its relevance for the finite element Galerkin discretization of elliptic BVP.
- Known the meaning of h -refinement and p -refinement.
- Ability to determine empirical (algebraic) convergence rates of various norms of a finite element discretization error.
- Ability to predict the asymptotic algebraic convergence of the energy norm and L^2 -norm the finite element discretization error for scalar 2nd-order elliptic BVP.
- Familiarity with features of an elliptic BVP (corners, discontinuous coefficients) that can thwart the fastest possible convergence of a Lagrangian finite element discretization for h -refinement.
- Knowledge of how to choose the appropriate order of quadrature and boundary approximation so as to preserve the optimal rate of convergence (for h -refinement).
- Use duality techniques to obtain improved error estimates for the evaluation of linear and continuous output functionals. Understanding of the importance of continuity of output functionals.
- Knowledge of the (discrete) maximum principle for scalar 2nd-order elliptic boundary value problems.
- An idea of common strategies for the debugging and validation of a general finite element code.

Bibliography

- [BS87] I. Babuška and M. Suri. “The optimal convergence rate of the p-version of the finite element method”. In: *SIAM J. Numer. Anal.* 24.4 (1987), pp. 750–769 (cit. on p. 343).
- [Cia78] P.G. Ciarlet. *The Finite Element Method for Elliptic Problems*. Vol. 4. Studies in Mathematics and its Applications. Amsterdam: North-Holland, 1978 (cit. on p. 358).
- [Eva98] L.C. Evans. *Partial differential equations*. Vol. 19. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 1998 (cit. on p. 302).
- [Gri85] P. Grisvard. *Elliptic Problems in Nonsmooth Domains*. Boston: Pitman, 1985 (cit. on p. 353).
- [Hac94] Wolfgang Hackbusch. *Iterative solution of large sparse systems of equations*. Vol. 95. Applied Mathematical Sciences. New York: Springer-Verlag, 1994, pp. xxii+429 (cit. on p. 385).
- [Har10] Helmut Harbrecht. “On output functionals of boundary value problems on stochastic domains”. In: *Math. Methods Appl. Sci.* 33.1 (2010), pp. 91–102 (cit. on p. 371).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 309–314, 326, 333, 341, 344, 383).
- [SK00] K. Salari and P. Knupp. *Code Verification by the Method of Manufactured Solutions*. Report SAND2000-1444. Albuquerque, NM: Sandia National Lab, 2000 (cit. on p. 389).
- [Sch98] Ch. Schwab. *p- and hp-Finite Element Methods. Theory and Applications in Solid and Fluid Mechanics*. Numerical Mathematics and Scientific Computation. Oxford: Clarendon Press, 1998 (cit. on pp. 306, 343).
- [SF73] G. Strang and G.J. Fix. *An Analysis of the Finite Element Method*. Prentice Hall, Eaglewood Cliffs, 1973 (cit. on p. 355).
- [Str09] M. Struwe. *Analysis für Informatiker*. Lecture notes, ETH Zürich. 2009 (cit. on p. 333).

Chapter 4

Beyond FEM: Alternative Discretizations

Now we examine approaches to the discretization of scalar linear 2nd-order elliptic BVPs that offer an alternative to finite element Galerkin methods discussed in Chapter 2. Some are closely related to the finite-element method and just adopt a different perspective on the discretization of a BVP. Others are confined to particular settings and may offer distinct advantages when applicable.

The objective of this chapter is not to introduce non-FEM approaches to the discretization of second-order scalar elliptic BVPs in depth, but to convey their main design principles and relationships with the FEM, and to introduce terminology commonly used in computational science.

Contents

4.1	Finite-Difference Methods (FDMs)	394
4.1.1	Finite-Difference Method for Two-Point Boundary-Value Problems	395
4.1.2	Finite-Difference Method in Two Dimensions	400
4.1.3	Finite Differences and Finite Elements	404
4.2	Finite-Volume Methods (FVMs)	408
4.2.1	Discrete Balance Laws	409
4.2.2	Dual Meshes	410
4.2.3	Relationship of FEM and FVM	414
4.3	Spectral Galerkin Methods	418
4.4	Collocation Methods	427
4.4.1	Spectral Collocation Method	429
4.4.2	Spline Collocation Methods	431

4.1 Finite-Difference Methods (FDMs)



Video tutorial for Section 4.1: Finite-Difference Methods (FDMs): (22 minutes)
[Download link](#), [tablet notes](#)

Finite-difference methods represent the approach to the discretization of 2nd-order elliptic boundary equations with the longest tradition, and, arguably, based on the simplest and most straightforward considerations.

Construction of finite-difference methods (FDMs): Policy



The idea underlying finite-difference methods is to

- (I) replace derivatives with **difference quotients**,
- (II) which are anchored at the nodes/grid points of a (*structured*) mesh/grid,
- (III) and access solution values only at other nodes/grid points.

Note a few implications of this design principle:

- Since they approximate derivatives, finite-difference methods target the “ODE/PDE-formulation” (**strong form**, Rem. 1.5.3.10) of boundary value problems.
- The *unknowns* in a finite-difference method are the *point values* of an approximation solution at the nodes/grid points of the mesh.

Now we present concrete finite-difference methods for scalar elliptic 2nd-order BVPs in one and two space dimensions. This will make clear all the details of the construction and the aspects (I)–(III) of the FDM policy.

4.1.1 Finite-Difference Method for Two-Point Boundary-Value Problems

For given Dirichlet data $u_a, u_b \in \mathbb{R}$ we consider the two-point elliptic 2nd-order Dirichlet boundary-value problem on the interval $]a, b[$, $a < b$, derived from the elastic string model in Section 1.5.1. In line with the FDM policy we focus on its strong form

$$-\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = f(x) \quad \text{in }]a, b[, \quad u(a) = u_a, \quad u(b) = u_b . \quad (1.5.1.16)$$

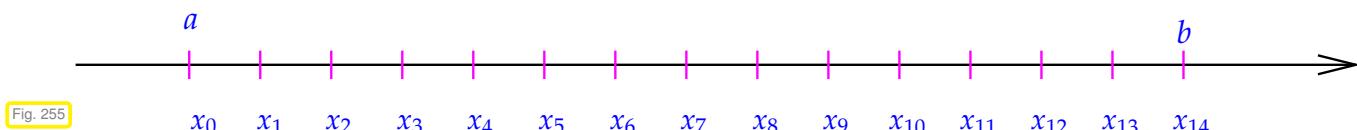
We take for granted the following smoothness properties of the uniformly positive coefficient function $\sigma :]a, b[\rightarrow \mathbb{R}^+$, and of the source function $f :]a, b[\rightarrow \mathbb{R}$:

$$\sigma \in C^0([a, b]) , \quad f \in C^0([a, b]) . \quad (4.1.1.1)$$

In words, both functions have to be continuous, which makes possible point evaluation. We point out that these smoothness requirements are less strict than those of Ass. 1.5.1.2, but stricter than (1.5.1.4), which is required for a meaningful weak formulation.

§4.1.1.2 (1D mesh/grid, recalling § 2.3.1.3) We equip the interval $\Omega :=]a, b[$, $a < b$, with an ordered sequence of $M + 1$, $M \in \mathbb{N}$, **nodes/grid points**:

$$\mathcal{V}(M) := \{a = x_0 < x_1 < \dots < x_{M-1} < x_M = b\} .$$



This defines the **mesh/grid** as a collection of open cells

$$\mathcal{M} := \{]x_{j-1}, x_j[: 1 \leq j \leq M\} .$$

with cell sizes $h_j := |x_j - x_{j-1}|$, $j = 1, \dots, M$. If all those are equal, $h_j = h > 0$ for all $j = 1, \dots, M$, the grid is said to be **equidistant** with mesh width $h_M = h$. \square

§4.1.1.3 (Difference quotients (DQs)) The derivative $\frac{du}{dx}$ of a differentiable function $u : [a, b] \rightarrow \mathbb{R}$ in $x_0 \in]a, b[$ is defined as the limit $h \rightarrow 0$ of some difference quotients with span $h > 0$. For finite span those provide an approximation of $\frac{du}{dx}(x_0)$. The “some” hints that different kinds of difference quotients can be used:

- ◆ Symmetric difference quotient at anchor point x_0 :

$$\frac{du}{dx}(x_0) \approx \frac{u(x_0 + h) - u(x_0 - h)}{2h}, \quad \text{with span } h > 0. \quad (4.1.1.4)$$

- ◆ One-sided difference quotients at anchor point x_0 :

$$\frac{du}{dx}(x_0) \approx \frac{u(x_0 + h) - u(x_0)}{h} \approx \frac{u(x_0) - u(x_0 - h)}{h}, \quad \text{with span } h > 0. \quad (4.1.1.5)$$

We remark that difference quotients provide approximations of $\frac{du}{dx}(x_0)$ of different orders, that is, with different algebraic rates of convergence as $h \rightarrow 0$, provided that $u \in C^2([a, b])$: for any $x_0 \in]a, b[$

$$\text{symmetric DQ: } \left| \frac{du}{dx}(x_0) - \frac{u(x_0 + h) - u(x_0 - h)}{2h} \right| = O(h^2) \quad \text{for } h \rightarrow 0, \quad (4.1.1.6)$$

$$\text{one-sided DQ: } \left| \frac{du}{dx}(x_0) - \frac{u(x_0 + h) - u(x_0)}{h} \right| = O(h) \quad \text{for } h \rightarrow 0. \quad (4.1.1.7)$$

This can be proved by Taylor expansion of u around x_0 , use (0.3.2.3a) together with (0.3.2.4). \square

§4.1.1.8 (Difference-quotient approximation of 2nd-order differential operator) We consider the linear 2nd-order differential operator $\mathcal{L}u := -\frac{d}{dx}\left(\sigma \frac{du}{dx}\right)$ from (1.5.1.16) with $\sigma \in C^0([a, b])$. We implement the finite-difference policy and successively replace the derivative operators with (two-sided, symmetric) difference quotients at a mesh node x_j :

- Replace outer derivative with a two-sided difference quotient ($x_{j-1/2} := \frac{1}{2}(x_j + x_{j-1})$):

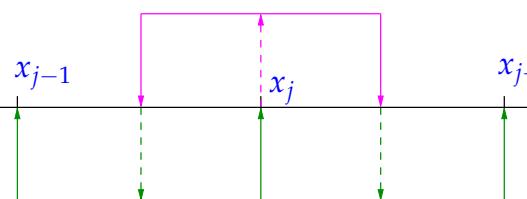
$$\frac{d}{dx}\left(\sigma(x)\frac{du}{dx}(x)\right)_{|x=x_j} \approx \frac{2}{h_j + h_{j+1}} \left(\sigma(x_{j+1/2})\frac{du}{dx}(x_{j+1/2}) - \sigma(x_{j-1/2})\frac{du}{dx}(x_{j-1/2}) \right).$$

Note that the possibility for point evaluation of the coefficient function σ is essential: $\sigma \in C^0(]a, b[)$ is required.

- Replace inner derivative with symmetric difference quotients:

$$\frac{du}{dx}(x_{j+1/2}) \approx \frac{u(x_{j+1}) - u(x_j)}{h_{j+1}}.$$

$$-\frac{d}{dx}\left(\sigma(x)\frac{du}{dx}(x)\right)_{|x=x_j} \approx \frac{\sigma(x_{j-1/2})\frac{u(x_j) - u(x_{j-1})}{h_j} - \sigma(x_{j+1/2})\frac{u(x_{j+1}) - u(x_j)}{h_{j+1}}}{\frac{1}{2}(h_j + h_{j+1})}. \quad (4.1.1.9)$$



Construction of (4.1.1.9):
 magenta: outer difference quotient
 green: inner difference quotients

Fig. 256

We can apply the approximation (4.1.1.9) to the PDE

$$-\frac{d}{dx}\left(\sigma(x)\frac{du}{dx}(x)\right) = f(x), \quad x \in]a, b[,$$

in every interior node $x_j, j = 1, \dots, M - 1$ of the mesh \mathcal{M} and get the $M - 1$ linear equations

$$\frac{\sigma(x_{j-1/2}) \frac{u(x_j) - u(x_{j-1})}{h_j} - \sigma(x_{j+1/2}) \frac{u(x_{j+1}) - u(x_j)}{h_{j+1}}}{\frac{1}{2}(h_j + h_{j+1})} = f(x_j), \quad j = 1, \dots, M - 1. \quad (4.1.1.10)$$

Note that the span of the difference quotients have judiciously been chosen so that they only involve values of u at the nodes of the mesh. \square

Remark 4.1.1.11 (Symmetric difference quotient for second derivative) In the case $h := h_j = h_{j+1}$ and for constant coefficient $\sigma \equiv 1$ (4.1.1.9) boils down to the well-known symmetric difference quotient approximation of the second derivative $-\frac{d^2u}{dx^2}(x_j)$:

$$-\frac{d^2u}{dx^2}(x_j) \approx \frac{-u(x_{j+1}) + 2u(x_j) - u(x_{j-1})}{h^2}. \quad (4.1.1.12)$$

\square

§4.1.1.13 (Finite-difference linear system of equations) The point values of the unknown solution $u(x_j)$, $j = 0, \dots, M$, occurring in (4.1.1.10) will play the role of the unknowns in the finite-difference method, denote them by $\mu_j \approx u(x_j)$, $j = 0, \dots, M$. Using this notation we can rewrite (4.1.1.10)

$$-\frac{\sigma(x_{j+1/2})}{\frac{1}{2}h_{j+1}(h_j + h_{j+1})}\mu_{j+1} + \frac{h_{j+1}^{-1}\sigma(x_{j+1/2}) + h_j^{-1}\sigma(x_{j-1/2})}{\frac{1}{2}(h_j + h_{j+1})}\mu_j - \frac{\sigma(x_{j-1/2})}{\frac{1}{2}h_j(h_j + h_{j+1})}\mu_{j-1} = f(x_j) \quad (4.1.1.14)$$

for $j = 1, \dots, M - 1$. This represents a linear system of $M - 1$ equations for $M + 1$ unknowns, which can be written in matrix-vector form:

$$\vec{A}\vec{\mu} = \vec{\varphi}, \quad \text{with } (\mathbf{A})_{j,\ell} := \begin{cases} 0 & , \text{ if } |j - \ell| > 1 , \\ -\frac{\sigma(x_{j+1/2})}{\frac{1}{2}h_{j+1}(h_j + h_{j+1})} & , \text{ if } \ell = j + 1 , \\ \frac{h_{j+1}^{-1}\sigma(x_{j+1/2}) + h_j^{-1}\sigma(x_{j-1/2})}{\frac{1}{2}(h_j + h_{j+1})} & , \text{ if } j = \ell , \\ -\frac{\sigma(x_{j-1/2})}{\frac{1}{2}h_j(h_j + h_{j+1})} & , \text{ if } \ell = j - 1 , \end{cases} \quad (4.1.1.15)$$

$$\varphi_j := f(x_j),$$

with $\vec{\mu} := [\mu_0, \dots, \mu_M]^\top \in \mathbb{R}^{M+1}$ and $j \in \{1, \dots, M - 1\}$, $\ell \in \{0, \dots, M\}$. Obviously, the matrix \mathbf{A} is a *tridiagonal* matrix in the sense of [Hip19, ??]. Also note that the row sums of the matrix \mathbf{A} all vanish, reflecting the fact that constant functions $x \mapsto c \in \mathbb{R}$ are in the nullspace of \mathcal{L} .

Significant simplifications result for an *equidistant* mesh, $h_j = h$ for all $j = 1, \dots, M$. Then the matrix $\mathbf{A} \in \mathbb{R}^{M-1, M+1}$ as defined in (4.1.1.15) becomes

$$\mathbf{A} = \frac{1}{h^2} \begin{bmatrix} -\sigma_{\frac{1}{2}} & \sigma_{\frac{1}{2}} + \sigma_{\frac{3}{2}} & -\sigma_{\frac{3}{2}} & & & & \\ & -\sigma_{\frac{3}{2}} & \sigma_{\frac{3}{2}} + \sigma_{\frac{5}{2}} & -\sigma_{\frac{5}{2}} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \ddots & \ddots & -\sigma_{M-\frac{3}{2}} \\ & & & & & -\sigma_{M-\frac{3}{2}} & \sigma_{M-\frac{1}{2}} + \sigma_{M-\frac{3}{2}} & -\sigma_{M-\frac{1}{2}} \end{bmatrix}, \quad (4.1.1.16)$$

with the abbreviation $\sigma_* := \sigma(x_*)$. □

§4.1.1.17 (Finite-difference methods for two-point Dirichlet BVP) Then linear system $\mathbf{A}\vec{\mu} = \vec{f}$ from (4.1.1.15) features two equations more than the number of unknowns. These numbers can easily balanced by using the Dirichlet boundary conditions $u(a) = u_a$ and $u(b) = u_b$ in (1.5.1.16). Motivated by $\mu_0 \approx u(a)$ and $\mu_M \approx u(b)$ we can simply set $\mu_0 = u_a$ and $\mu_M = u_b$ and remove these two unknowns. In the case of an *equidistant* mesh with meshwidth $h > 0$ this yields the square $(M - 1) \times (M - 1)$ *tridiagonal* linear system of equations

$$\frac{1}{h^2} \begin{bmatrix} \sigma_{\frac{1}{2}} + \sigma_{\frac{3}{2}} & -\sigma_{\frac{3}{2}} & & \\ -\sigma_{\frac{3}{2}} & \sigma_{\frac{3}{2}} + \sigma_{\frac{5}{2}} & -\sigma_{\frac{5}{2}} & \\ & \ddots & \ddots & \ddots \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & \ddots & -\sigma_{M-\frac{3}{2}} \\ & & & & -\sigma_{M-\frac{3}{2}} & \sigma_{M-\frac{1}{2}} + \sigma_{M-\frac{3}{2}} \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \vdots \\ \mu_{M-2} \\ \mu_{M-1} \end{bmatrix} = \begin{bmatrix} f(x_1) + \sigma_{\frac{1}{2}} u_a \\ f(x_2) \\ \vdots \\ \vdots \\ f(x_{M-2}) \\ f(x_{M-1}) + \sigma_{M-\frac{1}{2}} u_b \end{bmatrix}. \quad (4.1.1.18)$$

The coefficient matrix is *diagonally dominant* in the sense of [Hip19, ??]. □

§4.1.1.19 (Finite-difference method for two-point Neumann BVP) Instead of Dirichlet boundary conditions we may also impose inhomogeneous Neumann boundary conditions, which leads to the two-point BVP

$$-\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = f(x) \quad \text{in }]a, b[, \quad -\sigma(a) \frac{du}{dx}(a) = v_a , \quad \sigma(b) \frac{du}{dx}(b) = v_b , \quad (4.1.1.20)$$

with given Neumann data $v_a, v_b \in \mathbb{R}$.

We follow the finite-difference policy and replace the derivatives in the definition of the Neumann boundary conditions with *one-sided* difference quotients anchored in $x_0 = a$ and $x_M = b$, respectively:

$$-\sigma(a) \frac{du}{dx}(a) = v_a \quad \blacktriangleright \quad \sigma(x_{1/2}) \frac{u(a) - u(a + h_1)}{h_1} = v_a , \quad (4.1.1.21)$$

$$\sigma(b) \frac{du}{dx}(b) = v_b \quad \blacktriangleright \quad \sigma(x_{M-1/2}) \frac{u(b) - u(b - h_M)}{h_M} = v_b . \quad (4.1.1.22)$$

Using the μ -notation, $\mu_0 \approx u(a)$, $\mu_M \approx u(b)$, $\mu_j \approx u(x_j)$, we can rewrite these equations as

$$(4.1.1.21) \Rightarrow \mu_0 = \mu_1 + \frac{h_1}{\sigma(x_{1/2})} v_a , \quad (4.1.1.23)$$

$$(4.1.1.22) \Rightarrow \mu_M = \mu_{M-1} + \frac{h_M}{\sigma(x_{M-1/2})} v_b . \quad (4.1.1.24)$$

We plug this into the equations (4.1.1.14) for $j = 1$ and $j = M - 1$, using the abbreviation $\sigma_* := \sigma(x_*)$,

$$\begin{aligned} -\frac{\sigma_{3/2}}{\frac{1}{2}h_2(h_1 + h_2)} \mu_2 + \frac{h_2^{-1}\sigma_{3/2} + h_1^{-1}\sigma_{1/2}}{\frac{1}{2}(h_1 + h_2)} \mu_1 - \frac{\sigma_{1/2}}{\frac{1}{2}h_1(h_1 + h_2)} \mu_0 &= f(x_1) , \\ -\frac{\sigma_{M-1/2}}{\frac{1}{2}h_M(h_{M-1} + h_M)} \mu_M + \frac{h_M^{-1}\sigma_{M-1/2} + h_{M-1}^{-1}\sigma_{M-3/2}}{\frac{1}{2}(h_{M-1} + h_M)} \mu_{M-1} - \frac{\sigma_{M-3/2}}{\frac{1}{2}h_{M-1}(h_{M-1} + h_M)} \mu_{M-2} &= f(x_{M-1}) . \end{aligned}$$

This yields the modified equations

$$\begin{aligned} -\frac{\sigma_{3/2}}{\frac{1}{2}h_2(h_1+h_2)}\mu_2 + \frac{h_2^{-1}\sigma_{3/2}}{\frac{1}{2}(h_1+h_2)}\mu_1 &= f(x_1) + \frac{v_a}{\frac{1}{2}(h_1+h_2)}, \\ \frac{h_{M-1}^{-1}\sigma_{M-3/2}}{\frac{1}{2}(h_{M-1}+h_M)}\mu_{M-1} - \frac{\sigma_{M-3/2}}{\frac{1}{2}h_{M-1}(h_{M-1}+h_M)}\mu_{M-2} &= f(x_{M-1}) + \frac{v_b}{\frac{1}{2}(h_{M-1}+h_M)}. \end{aligned}$$

For an *equidistant* mesh with meshwidth $h > 0$ the final linear system of equations is

$$\frac{1}{h^2} \begin{bmatrix} \sigma_{\frac{3}{2}} & -\sigma_{\frac{3}{2}} & & & \\ -\sigma_{\frac{3}{2}} & \sigma_{\frac{3}{2}} + \sigma_{\frac{5}{2}} & -\sigma_{\frac{5}{2}} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & -\sigma_{\frac{M-3}{2}} \\ & & & & -\sigma_{\frac{M-3}{2}} + \sigma_{\frac{M-3}{2}} \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \vdots \\ \mu_{M-2} \\ \mu_{M-1} \end{bmatrix} = \begin{bmatrix} f(x_1) + v_a/h \\ f(x_2) \\ \vdots \\ \vdots \\ f(x_{M-2}) \\ f(x_{M-1}) + v_b/h \end{bmatrix}. \quad (4.1.1.25)$$

□

Remark 4.1.1.26 (Compatibility conditions for 1D FVM for Neumann two-point BVP) Write $\mathbf{A}\vec{\mu} = \vec{\phi}$ for the linear system of equations (4.1.1.25). Since all row sums of \mathbf{A} evaluate to zero, $\mathbf{A}\mathbf{1} = \mathbf{0}$, where $\mathbf{1} = [1, \dots, 1]^\top$. In fact, an easy computation shows $\mathcal{N}(\mathbf{A}) = \text{Span}\{\mathbf{1}\}$. Thus, solutions of $\mathbf{A}\vec{\mu} = \vec{\phi}$ exist, if and only if $\vec{\phi}^\top \mathbf{1} = 0$, which means

$$h \sum_{\ell=1}^{M-1} f(x_\ell) = -(v_a + v_b). \quad (4.1.1.27)$$

This is the finite-difference discrete counterpart of the compatibility condition for the inhomogeneous Neumann problem (1.8.0.11) with right hand side source function $f : \Omega \rightarrow \mathbb{R}$ and Neumann data $H : \partial\Omega \rightarrow \mathbb{R}$,

$$-\int_{\partial\Omega} h \, dS = \int_{\Omega} f \, dx. \quad (1.8.0.13)$$

The parallels between (4.1.1.27) and (1.8.0.13) are evident: the latter can be converted into the former by formally applying a quadrature formula. □

Review question(s) 4.1.1.28 (Finite-difference method for two-point boundary-value problems)

(Q4.1.1.28.A) For given $f \in C^0([0, 1])$ write down the linear system of equations arising from the finite-difference discretization of the two-point homogeneous Dirichlet boundary value problem

$$-\frac{d^2u}{dx^2}(x) = f(x) \quad \text{in }]0, 1[, \quad u(0) = u(1) = 0,$$

on an equidistant mesh with grid points $x_j = hj$, $j = 0, \dots, M$, $h := \frac{1}{M}$, $M \in \mathbb{N}$.

(Q4.1.1.28.B) We consider the elliptic two-point boundary value problem with **impedance boundary conditions**

$$-\frac{d^2u}{dx^2}(x) = f(x) \quad \text{in }]0, 1[, \quad -\frac{du}{dx}(0) + u(0) = \frac{du}{dx}(1) + u(1) = 0. \quad (4.1.1.29)$$

Develop a finite-difference discretization of (4.1.1.29) on an equidistant mesh with grid points $x_j = hj$, $j = 0, \dots, M$, $h := \frac{1}{M}$, $M \in \mathbb{N}$.

△

4.1.2 Finite-Difference Method in Two Dimensions

The finite-difference methods in one space dimension introduced in the previous section is rather general. The limitations of the finite-difference approach, its dependence on “structured” meshes, become apparent only in higher dimension. Now we discuss the case of scalar elliptic boundary value problems in 2D.

§4.1.2.1 (Model problem and structured mesh) For the sake of simplicity we restrict ourselves to the model homogeneous Dirichlet boundary value problem for the Laplacian:

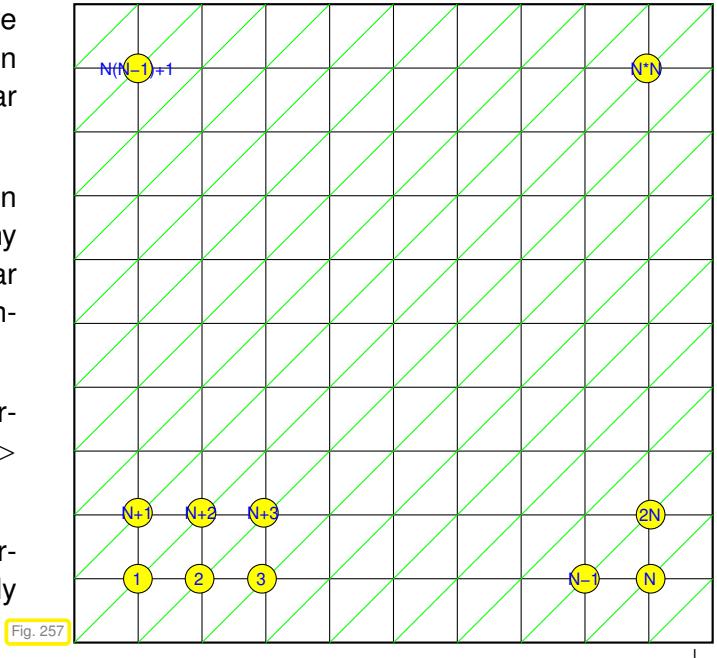
$$-\Delta u = -\frac{\partial^2 u}{\partial x_1^2} - \frac{\partial^2 u}{\partial x_2^2} = f \quad \text{in } \Omega := [0,1]^2, \quad u = 0 \quad \text{on } \partial\Omega. \quad (4.1.2.2)$$

Why that boring unit-square domain Ω ? Well, we will see that the finite-difference policy as outlined in the beginning of Section 4.1 hinges on a very regular structure of the underlying mesh.

Such a regular structure is only present in **tensor-product meshes**, also called grids. This is why we will limit the discussion the equispaced triangular tensor-product mesh \mathcal{M} of $\Omega := [0,1]^2$ with mesh-width $h = \frac{1}{N+1}$, $N \in \mathbb{N}$ displayed beside.

We impose a lexicographic (line-by-line) ordering/numbering of the nodes/grid points of \mathcal{M} . \triangleright

The following elaborations will have a partial overlap with § 3.7.2.2, which relied on the same family of meshes.



§4.1.2.3 (Grid-Based difference quotients on equispaced tensor-product mesh) The first step of finite difference approach to $-\Delta$ is the approximation of the of the *partial* derivatives by means of *directional* symmetric difference quotients. This is nothing new: we did this in (4.1.1.9). The following formulas generalize the symmetric second difference quotient (4.1.1.12) to partial derivatives.

$$\begin{aligned} \frac{\partial^2 u}{\partial x_1^2} \Big|_{\mathbf{x}=(\xi,\eta)} &\approx \frac{u(\xi-h, \eta) - 2u(\xi, \eta) + u(\xi+h, \eta)}{h^2}, \\ \frac{\partial^2 u}{\partial x_2^2} \Big|_{\mathbf{x}=(\xi,\eta)} &\approx \frac{u(\xi, \eta-h) - 2u(\xi, \eta) + u(\xi, \eta+h)}{h^2}. \end{aligned} \quad (4.1.2.4)$$

$\blacktriangleright \quad -\Delta u|_{\mathbf{x}=(\xi,\eta)} \approx \frac{1}{h^2} (4u(\xi, \eta) - u(\xi-h, \eta) - u(\xi+h, \eta) - u(\xi, \eta-h) - u(\xi, \eta+h)).$

In a second step we use this approximation at a grid point $\mathbf{p} = (ih, jh)$, $1 \leq i, j \leq N$. This will connect the five point values of the unknown solution $u(ih, jh)$, $u((i-1)h, jh)$, $u((i+1)h, jh)$, $u(ih, (j-1)h)$, $u(ih, (j+1)h)$. We recall that approximations $\mu_{i,j}$ to the *point values* $u(ih, jh)$ will be the **unknowns** of the finite difference method. \downarrow

§4.1.2.5 (Finite-difference linear system of equations) Centering the above difference quotients at all nodes/grid points of \mathcal{M} and taking into account the PDE $-\Delta u = f$ yields linear relationships between the unknowns $\mu_{i,j} \approx u(ih, jh)$, $i, j \in \{1, \dots, N\}$.

$$\frac{1}{h^2} (4u(ih, jh) - u(ih-h, jh) - u(ih+h, jh) - u(ih, jh-h) - u(ih, jh+h)) = f(ih, jh),$$

$$\frac{1}{h^2} (4\mu_{i,j} - \mu_{i-1,j} - \mu_{i+1,j} - \mu_{i,j-1} - \mu_{i,j+1}) = f(ih, jh). \quad (4.1.2.6)$$

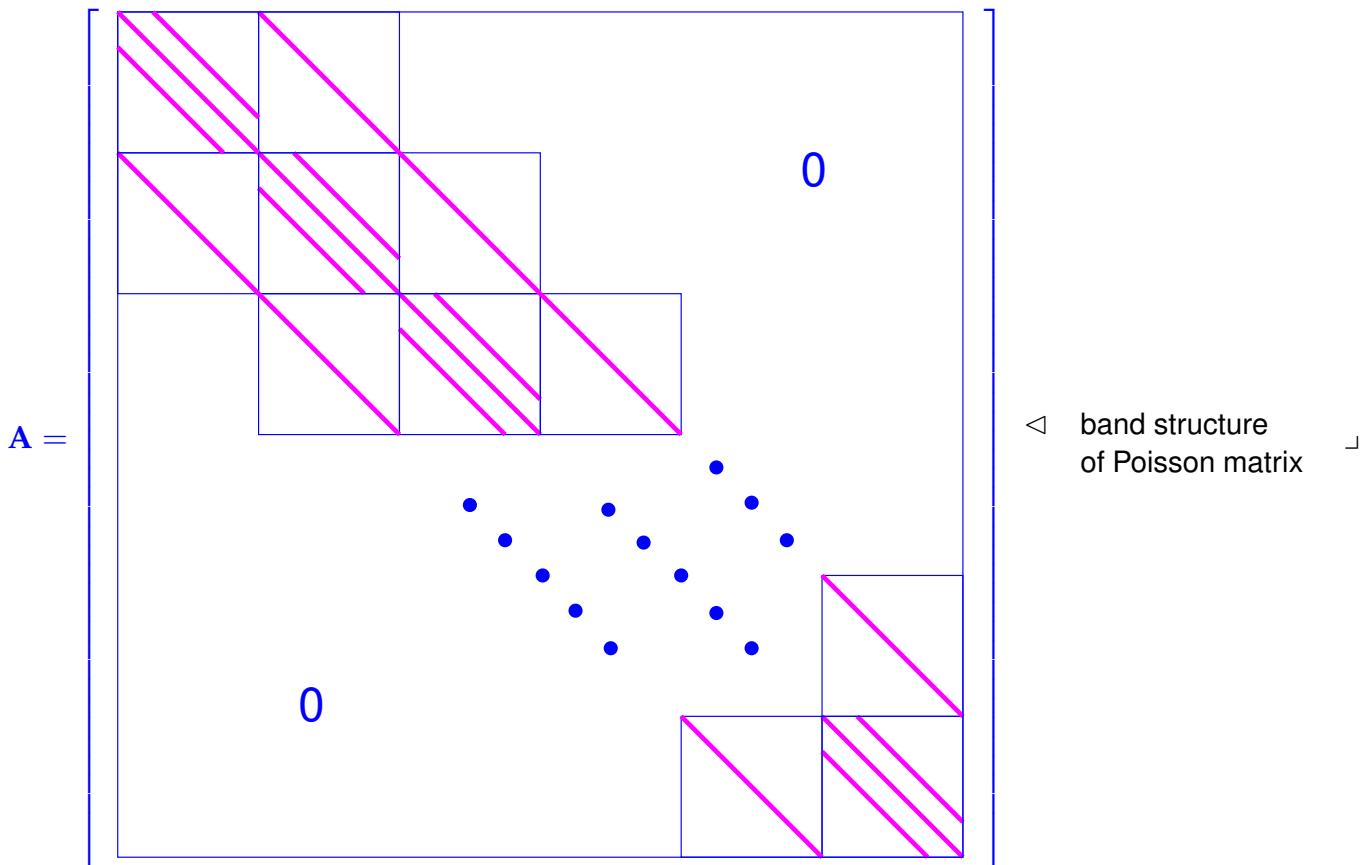
Also this is familiar from the discussion in 1D, see (4.1.1.14). Yet, in 1D the association of the point values and of components of the vector $\vec{\mu}$ of unknowns was straightforward and suggested by the linear ordering of the nodes of the grid. In 2D we have much more freedom. One option on tensor-product grids is the line-by-line ordering (lexicographic ordering) depicted in Fig. 257. This allows a simple indexing scheme:

$$u(\mathbf{p}) \leftrightarrow \mu_{i,j} \leftrightarrow \mu_{(j-1)N+i} \quad i, j = 1, \dots, N.$$

$$(4.1.2.5) \Rightarrow \frac{-\mu_{(j-2)N+i} - \mu_{(j-1)N+i-1} + 4\mu_{(j-1)N+i} - \mu_{(j-1)N+i+1} - \mu_{jN+i}}{h^2} = \underbrace{f(ih, jh)}_{=: \varphi_{(j-1)N+i}}. \quad (4.1.2.7)$$

We end up with a linear system of N^2 equations $\mathbf{A}\vec{\mu} = \vec{\varphi}$ with the $N^2 \times N^2$ block-tridiagonal **Poisson matrix**.

$$\mathbf{A} := \frac{1}{h^2} \begin{bmatrix} \mathbf{T} & -\mathbf{I} & 0 & \cdots & \cdots & \cdots & 0 \\ -\mathbf{I} & \mathbf{T} & -\mathbf{I} & & & & \vdots \\ 0 & -\mathbf{I} & \mathbf{T} & -\mathbf{I} & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & 0 \\ & & & -\mathbf{I} & \mathbf{T} & -\mathbf{I} & \\ 0 & \cdots & \cdots & 0 & -\mathbf{I} & \mathbf{T} & \end{bmatrix}, \quad \mathbf{T} := \begin{bmatrix} 4 & -1 & 0 & & & & 0 \\ -1 & 4 & -1 & & & & \vdots \\ 0 & -1 & 4 & -1 & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ & & & -1 & 4 & -1 & \\ 0 & \cdots & \cdots & 0 & -1 & 4 & \end{bmatrix} \in \mathbb{R}^{N,N} \quad (4.1.2.8)$$



Remark 4.1.2.9 (Extra smoothness of source function in finite difference approach) Obviously, we have to require $f \in C^0(\bar{\Omega})$ in order to render (4.1.2.5) meaningful.

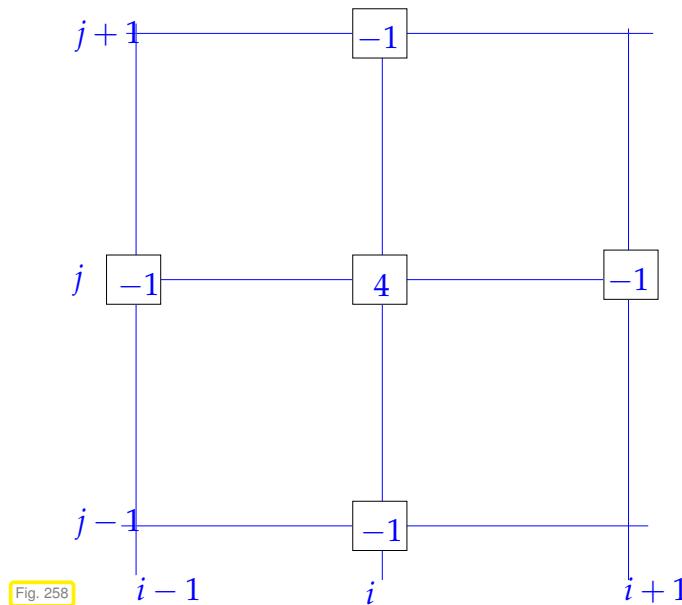
► The FD approach entails using regular source functions compared to finite element methods, for which $f \in L^2(\Omega)$ is enough.

(However, when numerical quadrature (\rightarrow Section 2.7.5) is used for the computation of the right hand side vector \vec{f} as in Section 2.4.6, then point evaluation of f has to be possible and $f \in C^0(\bar{\Omega})$ will also be required.)

§4.1.2.10 (Stencil notation) One row of the linear system of equations arising from the finite difference discretization of the 2D model problem on the tensor product mesh depicted in Fig. 257:

$$\frac{1}{h^2} (4\mu_{i,j} - \mu_{i-1,j} - \mu_{i+1,j} - \mu_{i,j-1} - \mu_{i,j+1}) = f(ih, jh) . \quad (4.1.2.5)$$

Note that the unknowns $\mu_{i,j}$ are indexed by the position of the grid point they are associated with. This suggests the following visualization of (4.1.2.5):



← stencil anchored at $(ih, jh) \in \Omega$.

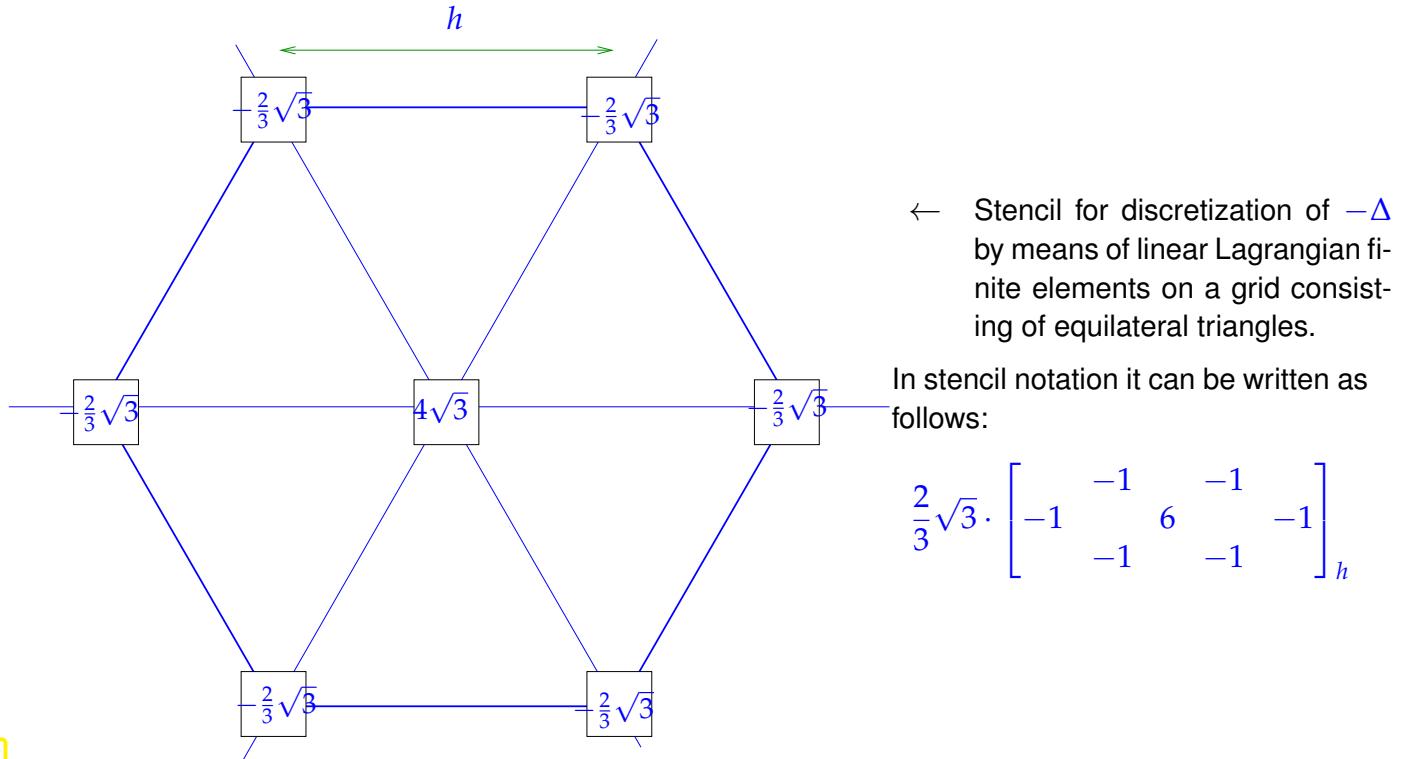
stencil notation:

$$\frac{1}{h^2} \begin{bmatrix} 1 & -1 & -1 \\ -1 & 4 & -1 \\ -1 & -1 & 1 \end{bmatrix}_h$$

A stencil as given above is a way to describe a row of a linear system of equations operating on unknowns associated with the nodes of a grid *without imposing a global numbering*.

The stencil description is particularly convenient in the case of *translation invariance*, where the stencils are the same for (almost) all nodes of the grid. Then, instead of specifying the matrix of the big linear system of equations, it suffices to write down a single stencil. \square

Remark 4.1.2.11 (Stencils on more general meshes) Stencils are not confined to tensor product grids. Here is an example of a stencil describing a finite-difference linear system of equations on a regular triangular mesh comprising equal equilateral triangles.



4.1.3 Finite Differences and Finite Elements

§4.1.3.1 (FDM and FEM matrices to two-points BVPs) For the homogeneous Dirichlet two-point boundary-value problem

$$-\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = f(x) \quad \text{in }]a, b[, \quad u(a) = 0, \quad u(b) = 0 . \quad (4.1.3.2)$$

an *equidistant* mesh with $M+1$ nodes $x_j = h j$ and meshwidth $h := \frac{1}{M}$, $M \in \mathbb{N}$, in Section 4.1.1 we derived the following linear system of equations by the finite-difference approach:

$$\frac{1}{h^2} \begin{bmatrix} \sigma_{\frac{1}{2}} + \sigma_{\frac{3}{2}} & -\sigma_{\frac{3}{2}} & & & \\ -\sigma_{\frac{3}{2}} & \sigma_{\frac{3}{2}} + \sigma_{\frac{5}{2}} & -\sigma_{\frac{5}{2}} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & -\sigma_{M-\frac{3}{2}} \\ & & & & -\sigma_{M-\frac{3}{2}} & \sigma_{M-\frac{1}{2}} + \sigma_{M-\frac{3}{2}} \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \vdots \\ \mu_{M-2} \\ \mu_{M-1} \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ \vdots \\ f(x_{M-2}) \\ f(x_{M-1}) \end{bmatrix},$$

where $\sigma_{\frac{2j-1}{s}} := \sigma(\frac{1}{2}(x_{j-1} + x_j))$.

In § 2.3.3.10 we used linear Lagrangian finite elements for the Galerkin finite-element discretization of (4.1.3.2), employing the standard tent basis functions (→ Section 2.3.2) and numerical quadrature based on the composite midpoint rule and composite trapezoidal rule. This led to the linear system of equations

$$\frac{1}{h} \begin{bmatrix} \sigma_1 + \sigma_2 & -\sigma_2 & 0 & \dots & & 0 \\ -\sigma_2 & \sigma_2 + \sigma_3 & -\sigma_3 & & \vdots & \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & -\sigma_{M-2} & \sigma_{M-2} + \sigma_{M-1} & -\sigma_{M-1} \\ 0 & \dots & \dots & 0 & -\sigma_{M-1} & \sigma_{M-1} + \sigma_M \end{bmatrix} \begin{bmatrix} \mu_1 \\ \vdots \\ \vdots \\ \mu_N \end{bmatrix} = h \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix}, \quad (2.3.3.14)$$

with $\sigma_j := \sigma(m_j)$, $m_j := \frac{1}{2}(x_j + x_{j-1})$, $j = 1, \dots, M-1$. Up to trivial scaling, the linear systems are *exactly the same!* The finite-difference approach to a scalar two-point boundary value problem produces the same equations as the linear Lagrangian finite-element method using numerical quadrature and the cardinal basis of tent functions. Perhaps this should not come as a surprise, because the basis expansion coefficients in the FEM agree with the values of the approximate solution at grid points. \square

§4.1.3.3 (FDM and FEM linear systems for 2D scalar elliptic BVPs) In two dimensions we will also come to the conclusion of § 4.1.3.1. So, let us derive the Galerkin matrix and right hand side vector for the 2D model problem on the tensor product mesh depicted in Fig. 257. To begin with we convert it into a triangular mesh \mathcal{M} by splitting each square into two equal triangles by inserting a diagonal (green lines in Fig. 257). On this mesh we use **linear Lagrangian finite elements** as in Section 2.4.

Now we repeat the considerations of Section 2.4. We consider the linear Lagrangian finite-element Galerkin discretization of 2D model problem

$$-\Delta u = -\frac{\partial^2 u}{\partial x_1^2} - \frac{\partial^2 u}{\partial x_2^2} = f \quad \text{in } \Omega := [0, 1]^2, \quad u = 0 \quad \text{on } \partial\Omega. \quad (4.1.2.2)$$

We rely on the uniform triangular tensor-product mesh (grid) \mathcal{M} described in § 4.1.2.1 and drawn in Fig. 257. On \mathcal{M} we use the finite element space $V_{0,h} = \mathcal{S}_{1,0}^0(\mathcal{M})$ and the “tent functions” as global shape functions, see Fig. 99.

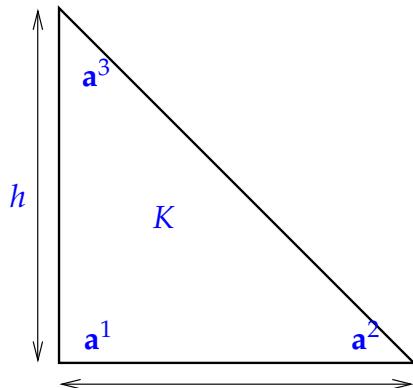


Fig. 260

To compute the element load vectors we use the *three-point quadrature formula* (2.7.5.34) (local trapezoidal rule) and obtain

$$\vec{\varphi}_K = \frac{1}{6}h^2 \begin{bmatrix} f(a^1) \\ f(a^2) \\ f(a^3) \end{bmatrix}.$$

For all the congruent triangles of \mathcal{M} we obtain the following **element stiffness matrix** from (2.4.5.8):

$$\mathbf{A}_K = \frac{1}{2} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}.$$

The numbering of local shape functions is induced by the numbering of vertices as given in the figure.

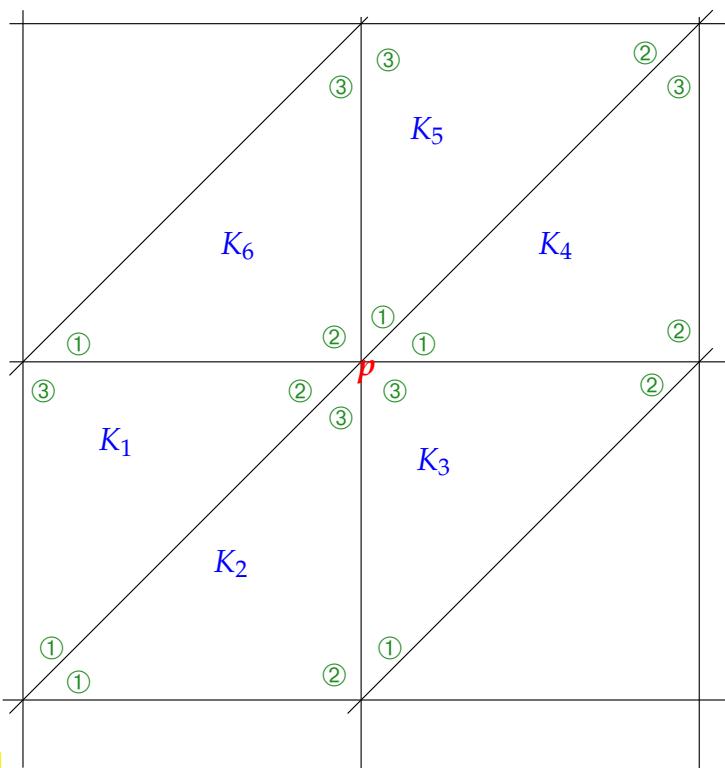


Fig. 261

To compute one entry of the finite-element right-hand-side/load vector $\vec{\varphi}$ we perform local assembly:

← green: local vertex numbers

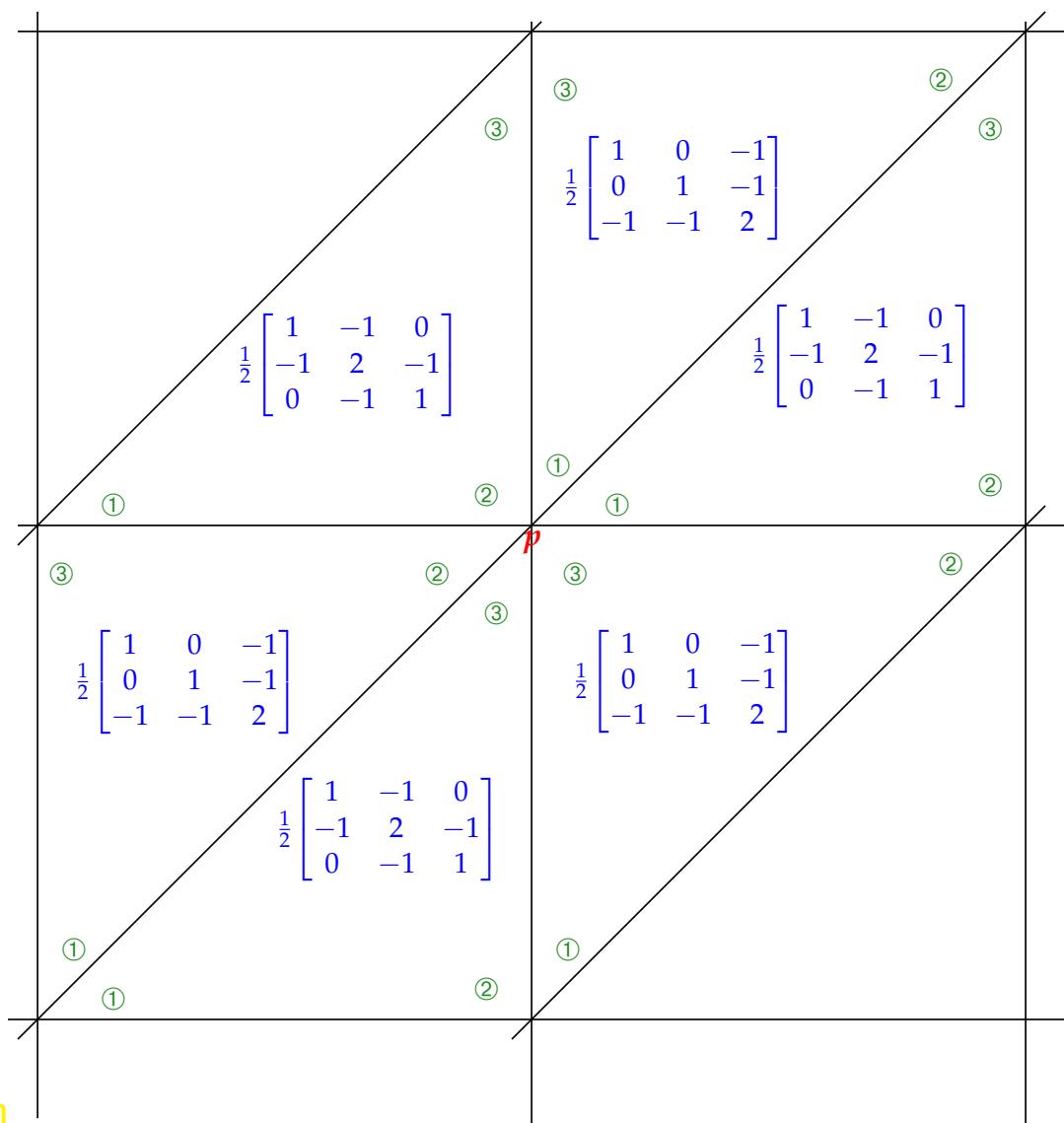
Contributions to load vector component associated with node p :

- From K_1 : $(\vec{\varphi}_{K_1})_2$
- From K_2 : $(\vec{\varphi}_{K_2})_3$
- From K_3 : $(\vec{\varphi}_{K_3})_3$
- From K_4 : $(\vec{\varphi}_{K_4})_1$
- From K_5 : $(\vec{\varphi}_{K_5})_1$
- From K_6 : $(\vec{\varphi}_{K_6})_2$



$$\vec{\varphi}_p = h^2 f(p).$$

To compute a single line of the finite-element linear system of equations associated with a single node of the mesh \mathcal{M} , we carry out another local assembly “on paper”, see Section 2.7.4 for explanations. We start with writing down all the element matrices surrounding a node p :



Then we do local assembly by adding up suitable entries of element matrices. We place the non-zero entries of the resulting row of the final Galerkin matrix on edges of the mesh to indicate which two unknowns/nodes they connect. The diagonal entry is written on top of a node itself. Thus, we immediately obtain a **stencil** notation describing the row of the Galerkin matrix.

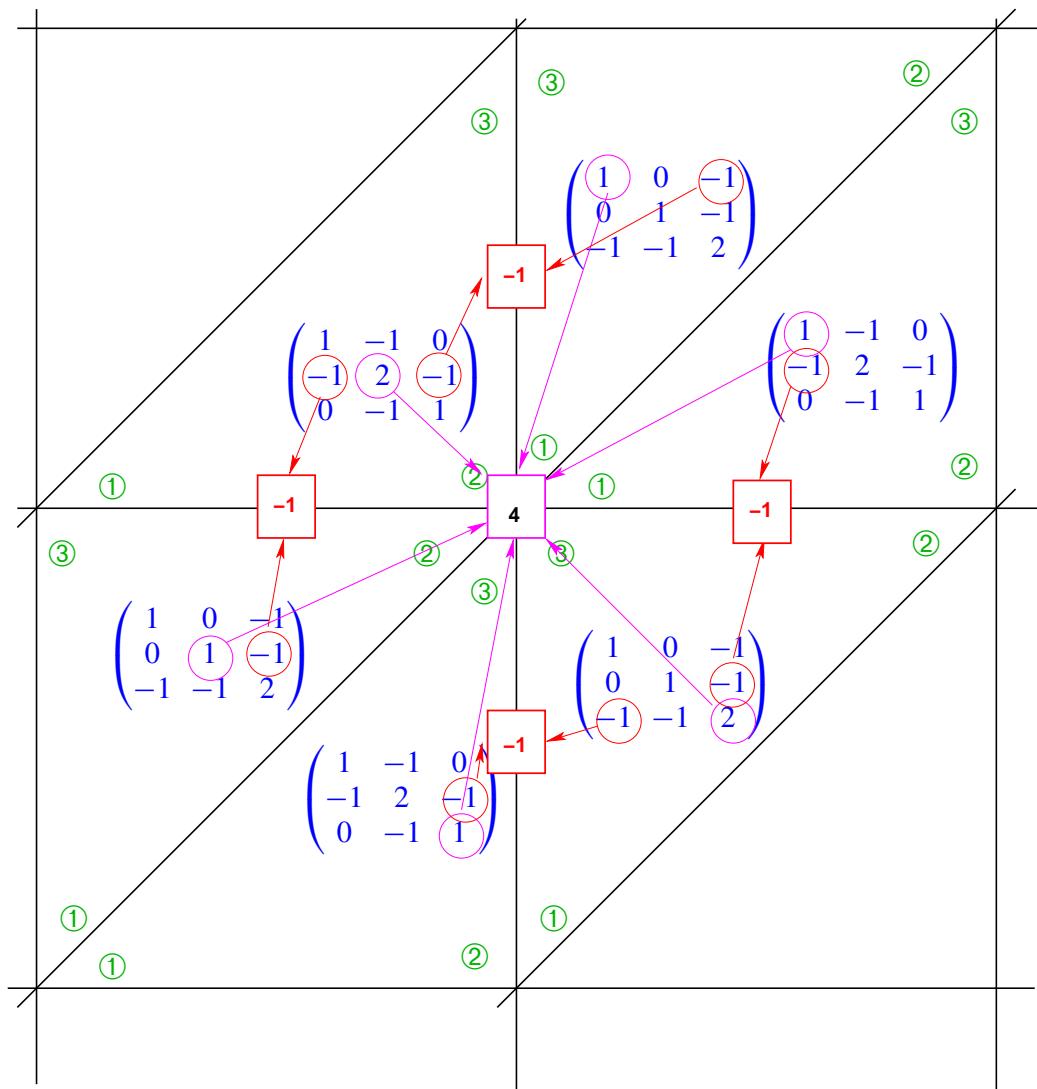


Fig. 263

The stencil we obtain is the very same as that obtained in § 4.1.2.10 by means of the finite-difference approach: Also the Galerkin finite-element discretization of (4.1.2.2) on the triangular tensor-product mesh of Fig. 257 using standard linear Lagrangian finite-element spaces and bases yields the same $N^2 \times N^2$ linear system of equations with the Poisson matrix (4.1.2.8) as coefficient matrix as the FDM. This insight remains valid beyond the model problem in 2D:

(Most) finite difference schemes

finite element Galerkin schemes
with numerical quadrature
on **structured** meshes

§4.1.3.4 (Pros and cons of “finite difference approach”) Let us briefly compare the finite-difference method and the finite-element Galerkin method, of course, with focus on 2nd-order linear scalar problems:

- ◆ Finite element methods can be used on general triangulations and structured (tensor-product) meshes alike, which delivers superior flexibility in terms of geometry resolution (advantage FEM).
- ◆ The correct treatment of all kinds of boundary conditions (\rightarrow Section 1.7). naturally emerges from the variational formulations in the finite element method (advantage FEM).
- ◆ Finite difference approach cannot deal with second-order elliptic boundary value problems with *discontinuous* diffusion coefficient (α in (1.4.2.4), (1.8.0.16)), which does not cause difficulties for finite element methods.

- ◆ Finite element methods have built-in “safety rails” because there are clear criteria for choosing viable finite element spaces and once this is done, there is no freedom left to go astray (advantage FEM).
- ◆ Finite element methods are harder to understand (advantage FD, but only with students who have not attended this course!)

Then, why are “finite difference methods” ubiquitous in scientific and engineering simulations ?

When people talk “finite differences” they have in mind **structured meshes** (translation invariant, tensor product structure) and use the term as synonym for “discretization on structured meshes”. The popularity of structured meshes is justified:

- structured meshes allow regular data layout and **vectorization**, which boost the performance of algorithms on high performance computing hardware. This is outside the scope of the this lecture and treated in courses on “High Performance Computing”.
- translation invariant PDE operators give rise to simple Galerkin matrices that need not be assembled and stored (recall the 5-point-stencil for $-\Delta$) and support very efficient matrix \times vector operations.

Use structured meshes whenever possible!

Review question(s) 4.1.3.5 (Finite-difference method in two dimensions)

(Q4.1.3.5.A) A boundary value problem for the Poisson equation $-\Delta u = f$ on $\Omega :=]0,1[^2$ is discretized by means of **bi-linear** Lagrangian finite elements on a uniform tensor-product mesh (meshwidth $h > 0$) comprising equal squares. What is the “shape” of the resulting stencil for an interior node of the mesh?

(Q4.1.3.5.B) The classical finite-difference discretization of the 2D Poisson equation $-\Delta u = f$ in $\Omega :=]0,1[^2$ and on a uniform tensor-product mesh with meshwidth $h > 0$ can be described by the customary 5-point stencil

$$\begin{bmatrix} -1 & -1 \\ -1 & 4 & -1 \\ -1 & \end{bmatrix}_h .$$

Outline how non-homogeneous Dirichlet boundary conditions, $u = g$ on $\partial\Omega$ with $g \in C^0(\partial\Omega)$, can be dealt with in the context of this FDM.

△

4.2 Finite-Volume Methods (FVMs)



Video tutorial for Section 4.2: Finite-Volume Methods (FVMs): (23 minutes) [Download link](#), [tablet notes](#)

In Section 1.6 we have derived second-order elliptic boundary value problems by combining

- a **conservation/balance law**, expressed by a balance equation

$$\int_{\partial V} \mathbf{j} \cdot \mathbf{n} dS = \int_V f dx \quad \text{for all “control volumes” } V , \quad (1.6.0.3)$$

- with **flux law** like Fourier’s law

$$\mathbf{j}(x) = -\kappa(x) \mathbf{grad} u(x) , \quad x \in \Omega . \quad (1.6.0.5)$$

Finite-volume methods (FVM) for the discretization of second-order elliptic boundary value problems are inspired by this very derivation.

The focus in this section is on linear scalar 2nd-order elliptic boundary value problem in 2D (\rightarrow Section 1.6) with homogeneous Dirichlet boundary conditions (\rightarrow Section 1.7), uniformly positive scalar heat conductivity $\kappa = \kappa(x)$

$$-\operatorname{div}(\kappa(x) \operatorname{grad} u) = f \quad \text{in } \Omega \subset \mathbb{R}^2 , \quad u = g \quad \text{on } \partial\Omega .$$

The data $f \in C^0(\overline{\Omega})$ and $g \in C^0(\partial\Omega)$ are supposed to be given.

4.2.1 Discrete Balance Laws

§4.2.1.1 (Control volumes) As announced above, finite volume methods for 2nd-order elliptic BVP are inspired by the *conservation principle* (1.6.0.3). e

$$\int_{\partial V} \mathbf{j} \cdot \mathbf{n} dS = \int_V f dx \quad \text{for all "control volumes" } V , \quad (1.6.0.3)$$

where $\mathbf{j} := -\kappa \operatorname{grad} u : \Omega \rightarrow \mathbb{R}^2$ is the flux as introduced in § 1.6.0.1. Physics requires that (1.6.0.3) holds for all (infinitely many) “control volumes” $V \subset \Omega$. Since discretization has to lead to a finite number of equations, the idea is to demand that (1.6.0.3) holds for only a *finite number of special control volumes*.



Idea of FVM:

Enforce the conservation/balance law (1.6.0.3) only for finitely many **control volumes**.

► First ingredient of FVM: Finitely many *non-overlapping control volumes* $C_i, i = 1, \dots, \tilde{M}$, covering the computational domain: $\overline{\Omega} = \bigcup_{i=1}^{\tilde{M}} \overline{C}_i$

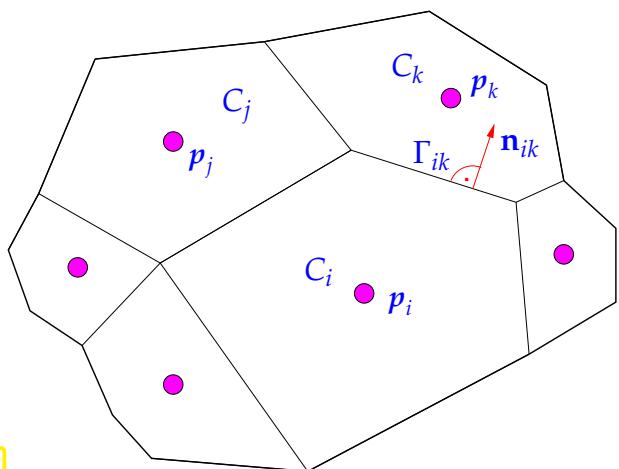


Fig. 264

A concrete collection of 2D control volumes:

- ◀ In 2D the control volumes may be chosen as the (polygonal) cells of a general mesh $\widetilde{\mathcal{M}} = \{C_i\}_i$ of Ω according to Def. 2.5.1.1.
- ☞ notations:
 - $\Gamma_{ik} \triangleq$ interface/edge between C_i and C_k
 - $\mathbf{n}_{ik} \triangleq$ unit normal at Γ_{ik} pointing from C_i into C_k .

§4.2.1.2 (The unknowns of the FVM) Then associate with each control volume $C_i, i = 1, \dots, \tilde{M}$, an *unknown* value $\mu_i \in \mathbb{R}$, which is related to the solution u inside C_i . Different meanings can be assigned to μ_i :

- μ_i can be read as an approximation of the value of the solution u at a “center” p_i of the cell C_i ,

$$\mu_i \approx u(p_i) , \quad p_i = \text{"center" of } C_i . \quad (4.2.1.3)$$

- μ_i may be regarded as an approximation of the average of u over C_i ,

$$\mu_i \approx \frac{1}{|C_i|} \int_{C_i} u(\mathbf{x}) \, d\mathbf{x} . \quad (4.2.1.4)$$

↓

§4.2.1.5 (From discrete flux laws to systems of equations) We have seen that the conservation law (1.6.0.3) had to be linked to the flux law (1.6.0.5) in order to give rise to a 2nd-order scalar PDE see (1.6.0.8)–(1.6.0.9). Correspondingly, “heat conservation in control volumes” has to be supplemented by a rule that furnishes the heat flux between two adjacent control volumes.

► Second ingredient of FVM local **numerical fluxes**

For two adjacent cells C_k, C_i with common edge $\Gamma_{ik} := \bar{C}_i \cap \bar{C}_k$.

$$\text{Numerical flux} \quad J_{ik} = \Psi(\mu_i, \mu_k) \approx \int_{\Gamma_{ik}} \mathbf{j} \cdot \mathbf{n}_{ik} \, dS$$

$\Psi \doteq$ numerical flux function, \mathbf{j} = (heat) flux, see (1.6.0.2), $\mathbf{n}_{ik} \doteq$ edge normal, see Fig. 264.

To obtain a system of equations from combining (1.6.0.3) with a numerical flux we follow the main idea of FVM and consider the balance law on the finitely many control volumes C_i :

$$\int_{\partial C_i} \mathbf{j} \cdot \mathbf{n}_i \, dS = \int_{C_i} f \, d\mathbf{x} \quad \Rightarrow \quad \sum_{k \in \mathcal{U}_i} J_{ik} = \int_{C_i} f \, d\mathbf{x} . \quad (4.2.1.6)$$

☞ notation: $\mathcal{U}_i := \{j : C_i \text{ and } C_j \text{ share edge, } C_j \in \tilde{\mathcal{M}}\}$

Using $J_{ik} = \Psi(\mu_i, \mu_k)$, this leads to a system of $\tilde{M} := \#\tilde{\mathcal{M}}$ equations for the unknowns μ_i :

$$\sum_{k \in \mathcal{U}_i} \Psi(\mu_i, \mu_k) = \int_{C_i} f \, d\mathbf{x} \quad \forall i = 1, \dots, \tilde{M} . \quad (4.2.1.7)$$

Another approximation for the right hand side: Writing \mathbf{p}_i for the “center” of C_i we may use a 1-point quadrature rule for the approximate evaluation of integral over C_i . The resulting system of equations is

$$\sum_{k \in \mathcal{U}_i} \Psi(\mu_i, \mu_k) = |C_i| f(\mathbf{p}_i) \quad \forall i = 1, \dots, \tilde{M} . \quad (4.2.1.8)$$

You may wonder what is done in case a control volume abuts the boundary $\partial\Omega$. This discussion is postponed to the next section, Rem. 4.2.2.8, ???. Formulas for the numerical flux function Ψ will be presented in § 4.2.3.2. ↓

4.2.2 Dual Meshes

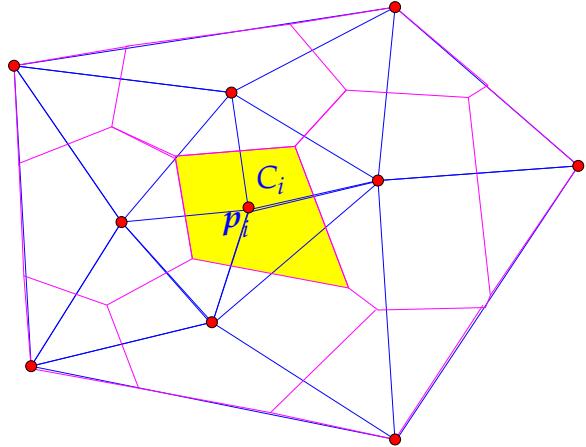
Dual meshes are a commonly used technique for the construction of control volumes for FVM, based on a conventional finite-element mesh \mathcal{M} of Ω , as introduced in Section 2.5.1. The discussion here is confined to dual meshes for a triangular mesh \mathcal{M} in 2D, Ω a polygon. This triangular mesh is often called the **primal mesh**.

§4.2.2.1 (Voronoi dual meshes) A popular choice of control volumes for FVMs is provided by the Voronoi dual mesh associated with the primal triangular mesh \mathcal{M} .

Write $\mathcal{V}(\mathcal{M}) = \{\mathbf{p}_1, \dots, \mathbf{p}_M\}$ for the set of nodes of the primal mesh \mathcal{M} . For every node \mathbf{p}_i defined its **Voronoi cell** as the polygon

$$C_i := \{\mathbf{x} \in \Omega: \|\mathbf{x} - \mathbf{p}_i\| < \|\mathbf{x} - \mathbf{p}_j\| \forall j \neq i\}. \quad (4.2.2.2)$$

In words, C_i is the points of the plane closer to \mathbf{p}_i than any other node. The collection of these Voronoi cells forms the **Voronoi dual mesh** $\tilde{\mathcal{M}} := \{C_i\}_{i=1}^M$.



▷ Voronoi dual mesh:

- : nodes of primal mesh
- : edges of primal mesh
- - -: edges Voronoi dual mesh
- : Voronoi dual cell for \mathbf{p}_i

Fig. 265

Obviously there is a one-to-one relationship between the nodes of the primal mesh \mathcal{M} and the cells of the Voronoi dual mesh $\tilde{\mathcal{M}}$:

$$\text{nodes of } \mathcal{M} \longleftrightarrow \text{cells of } \tilde{\mathcal{M}}.$$

Construction of a polygonal Voronoi dual cell associated with node \mathbf{p}_i :

- Its edges are the perpendicular bisectors of primal edges adjacent to \mathbf{p}_i
- Its vertices are the circumcenters of triangles of \mathcal{M} , which have \mathbf{p}_i as a vertex.

This construction even has a straightforward generalization to 3D.

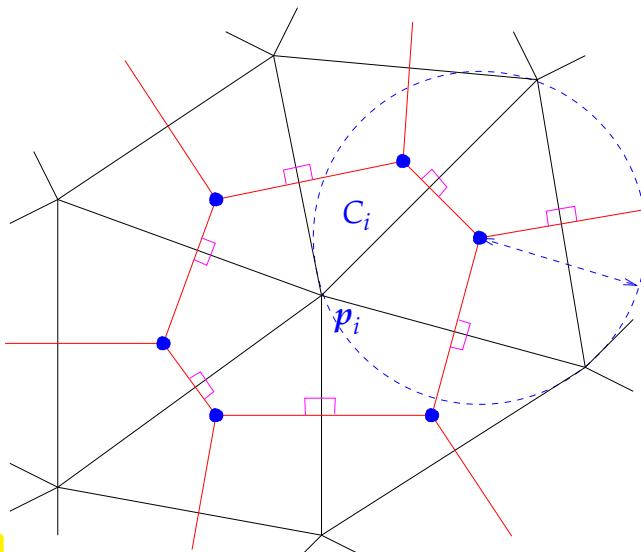


Fig. 266

Remark 4.2.2.3 (Circumcenter-based construction: Geometric obstruction) The construction of a Voronoi dual mesh based on circumcenters of triangles encounters difficulties in case some triangles of \mathcal{M} have obtuse angles.

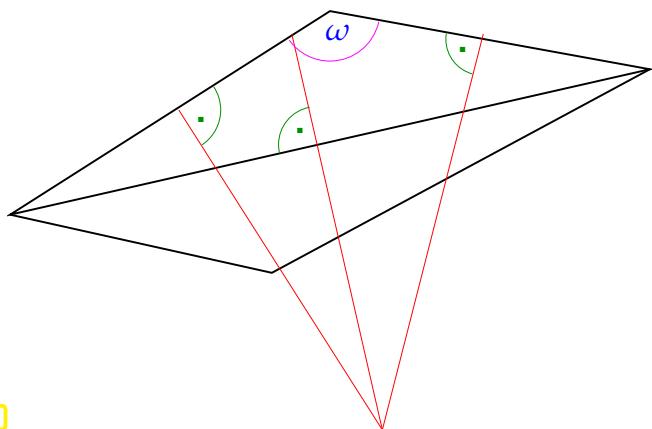


Fig. 267

\Leftrightarrow Obtuse angle ω :

- circumcenter \notin triangle
- $\bar{C}_i \cap \bar{C}_j \neq \emptyset \not\Leftrightarrow$ nodes i, j connected by edge
- geometric construction breaks down
- connectivity of unknowns unclear

What we want is that the two Voronoi cells belonging to two nodes of \mathcal{M} that are connected by an edge also have a common edge. \square

Theorem 4.2.2.4. Angle condition for Voronoi dual meshes

The following **angle condition** ensures that the Voronoi cells belonging to adjacent nodes of a triangular mesh have a common edge ($\bar{C}_i \cap \bar{C}_j \neq \emptyset \Leftrightarrow$ nodes i, j connected by edge of \mathcal{M}):

- (i) sum of angles facing interior edge $\leq \pi$,
- (ii) angles facing boundary edges $\leq \pi/2$.

Note: Condition (ii) important only for FV methods with unknowns attached to boundary vertices, that is, in the case of non-Dirichlet boundary conditions, cf. Rem. 4.2.2.8 below.

Definition 4.2.2.5. Delaunay triangulation

Triangular meshes satisfying the angle conditions (i) and (ii) from Thm. 4.2.2.4 are called **Delaunay triangulations**. \square

§4.2.2.6 (Barycentric dual meshes) Another popular choice for dual meshes associated with triangular primal meshes are **Barycentric dual meshes**. To begin with remember that the **barycenter** of a triangle with vertices $a^1, a^2, a^3 \in \mathbb{R}^2$ is the point $p := \frac{1}{3}(a^1 + a^2 + a^3)$.

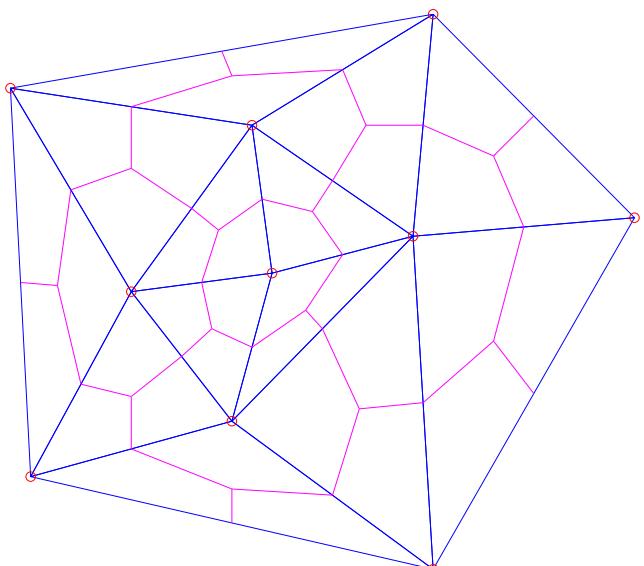


Fig. 268

Construction of barycentric dual mesh:

- The nodes are the barycenters of the triangles of \mathcal{M}
- The edges of dual cells are the lines connecting barycenters and midpoints of edges of \mathcal{M} .

This construction can be carried out for any triangular mesh: There are no geometric obstructions \square

FVM with dual meshes: Summary

The cells $C_i, i = 1, \dots, \tilde{M} = \#\tilde{\mathcal{M}}$ of the dual mesh $\tilde{\mathcal{M}}$ will be used as control volumes in the finite-volume method. This gives us a one-to-one correspondence of nodes of \mathcal{M} and control volumes. Moreover, for both type of dual meshes, Voronoi dual meshes and barycentric dual meshes, the primal node p_i plays the role of the “center” of the associated (dual) control volume C_i .

Remark 4.2.2.8 (FVM: Incorporation of Dirichlet boundary conditions) Assume that the control volumes are obtained either from a Voronoi dual mesh for a Delaunay triangulation or a barycentric dual mesh. In this case, the dual cell belonging to primal nodes on $\partial\Omega$ will completely cover $\partial\Omega$ and no part of the boundary of any dual cell associated with an interior primal node will be contained in $\partial\Omega$.



- Consider only control volumes (dual cells) located in the interior of Ω in (4.2.1.7),
- Fix $\mu_k = g(p_k)$ for control volumes (dual cells) that abut $\partial\Omega$. This makes sense, because the centers p_k of those control volumes will be located on $\partial\Omega$, where u is known.

This leads to a number of equations = number of unknowns equal to the number of interior nodes of the primal mesh. \square

Remark 4.2.2.9 (Treatment of Neumann boundary conditions in finite volume schemes) Consider a finite volume method based on dual meshes for 2nd-order elliptic Neumann problem:

$$-\operatorname{div}(\kappa(x) \operatorname{grad} u) = f \quad \text{in } \Omega , \quad (\kappa(x) \operatorname{grad} u) \cdot n = h \quad \text{on } \partial\Omega . \quad (1.9.0.2)$$

We make the same assumptions on the control volumes as in Rem. 4.2.2.8. Now u on $\partial\Omega$ not known.



- Keep balance equations for control volumes associated with (primal) vertices on the boundary $\partial\Omega$.
- Remember (1.7.0.3): $h = -j \cdot n$, that is, the Neumann data $h \in L^2(\partial\Omega)$ already provide the flux!

Thus, taking for granted a numerical flux function Ψ , we find the following modified instance of (4.2.1.7) for a control volume C_i adjacent to $\partial\Omega$:

$$\sum_{k \in \mathcal{U}_i} \Psi(\mu_i, \mu_k) - \int_{\partial C_i \cap \partial\Omega} h \, dS = |C_i|f(p_i) , \quad i = 1, \dots, \tilde{M} .$$

The number of equations = number of unknowns agrees with the total number of nodes of \mathcal{M} . \square

4.2.3 Relationship of FEM and FVM

Hardly surprising, finite volume methods and finite element Galerkin discretizations are closely related. This will be explored in this section for a model problem.

Setting:

- ◆ We consider the homogeneous Dirichlet problem for the Laplacian Δ

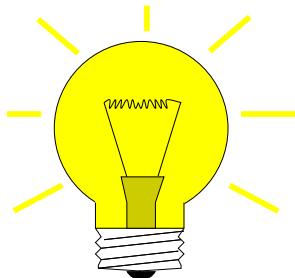
$$-\Delta u = f \quad \text{in } \Omega , \quad u = 0 \quad \text{on } \partial\Omega . \quad (4.2.3.1)$$

- ◆ Discretization by finite volume method based on a triangular mesh \mathcal{M} and on Voronoi dual cells → Fig. 265.

We assume that \mathcal{M} is a Delaunay triangulation of Ω , that is, the angle condition of Thm. 4.2.2.4 is to be fulfilled.

As explained in Rem. 4.2.2.8 that number of *active* control volumes in (4.2.1.7) is the number of interior nodes of \mathcal{M} .

§4.2.3.2 (Numerical flux by interpolation) What is still missing is the specification of the numerical flux function $\Psi : \mathbb{R}^2 \mapsto \mathbb{R}$ for each *dual edge*



Idea: Obtain the numerical flux from Fourier's law

$$\mathbf{j}(x) = -\kappa(x) \mathbf{grad} u(x), \quad x \in \Omega, \quad (1.6.0.5)$$

applied to a (sufficiently smooth) $u_h : \Omega \mapsto \mathbb{R}$ *reconstructed* from dual cell values μ_i .

This is a natural approach, since μ_i can be viewed as an approximation of $u(p_i)$, where the “center” p^i of the dual cell C_i coincides with an *interior node* $x^i \in \mathcal{V}(\mathcal{M})$ of the triangular mesh \mathcal{M} : Recover $u_h \in \mathcal{S}_{1,0}^0(\mathcal{M})$ as an element of the finite element space,

$$u_h = \mathbf{l}_1 \vec{\mu} := \sum_{i=1}^N \mu_i b_h^i, \quad (4.2.3.3)$$

where $N = \#\mathcal{V}(\mathcal{M})$ = number of dual cells, size of vector $\vec{\mu}$,

$b_h^i \hat{=} \text{nodal basis function (“tent function”, Section 2.4.3) of } \mathcal{S}_{1,0}^0(\mathcal{M}) \text{ belonging to the node inside } C_i.$

$u_h \hat{=} \text{piecewise linear interpolant of vertex values } \mu_i$

Note that u_h is not smooth across inner edges of \mathcal{M} . However, we do not care when computing $\mathbf{j} := \kappa(x) \mathbf{grad} u_h$, because this flux is *only needed at edges of the dual mesh*, which lie inside triangles of \mathcal{M} (with the exception of single points that are irrelevant for the flux integrals).

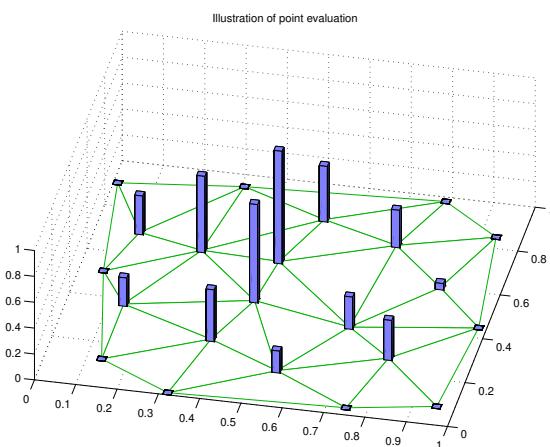


Fig. 269

vertex values μ_i on $\mathcal{V}(\mathcal{M})$

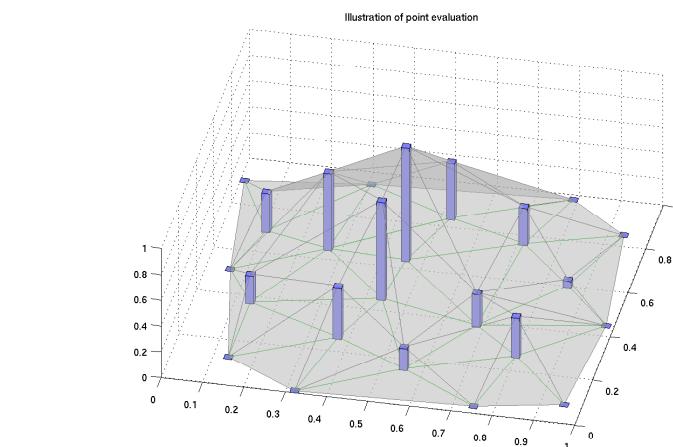
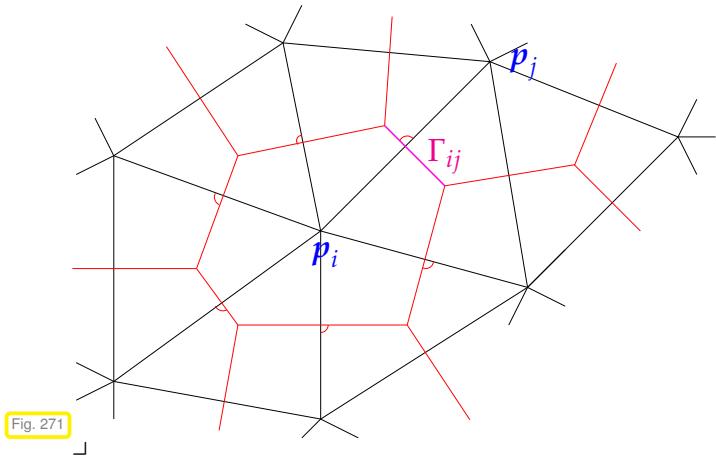


Fig. 270

p.w. linear interpolant $u_h := \mathbf{l}_1 \vec{\mu} \in \mathcal{S}_{1,0}^0(\mathcal{M})$



Choice of **numerical flux**:

$$J_{ik} := - \int_{\Gamma_{ik}} \kappa \operatorname{grad} l_1 \vec{\mu} \cdot \mathbf{n}_{ik} dS , \quad (4.2.3.4)$$

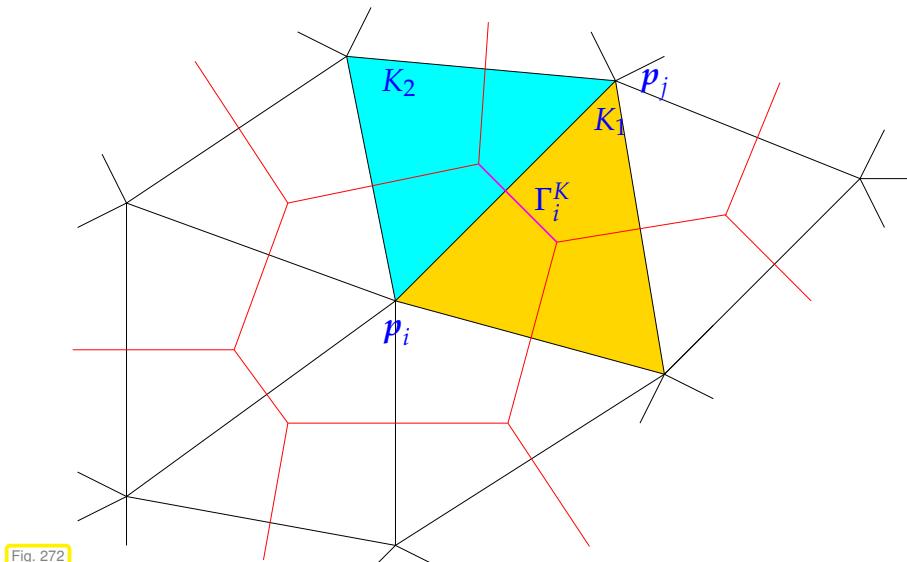
where Γ_{ij} is the interface separating the control volumes C_i and C_j .

Note the orthogonality $\Gamma_{ij} \perp \mathbf{p}_i - \mathbf{p}_j$.

From now we restrict ourselves to the case $\kappa \equiv 1$. Using the numerical flux (4.2.3.4) in (4.2.1.7) we obtain the following formula for the entries of one row of finite volume discretization matrix from the equations

$$\begin{aligned} \sum_{k \in \mathcal{U}_i} \int_{\Gamma_{ik}} \operatorname{grad} l_1 \vec{\mu} \cdot \mathbf{n}_{ik} dS &= \mu_i \underbrace{\int_{\partial C_i} \operatorname{grad} b_h^i dS}_{\text{matrix entry } -(A)_{ii}} + \sum_{j \in \mathcal{U}_i} \mu_j \left(\sum_{k \in \mathcal{U}_i} \int_{\Gamma_{ik}} \operatorname{grad} b_h^j \cdot \mathbf{n}_{ik} dS \right) \\ &\quad \underbrace{\qquad\qquad\qquad}_{\text{matrix entry } -(A)_{ij}} \\ &= - \int_{C_i} f(x) dx . \\ \Rightarrow \quad (A)_{ij} &= - \int_{\partial C_i} \operatorname{grad} b_h^j \cdot \mathbf{n}_i dS , \quad i, j \in \{1, \dots, N\} . \end{aligned} \quad (4.2.3.5)$$

where $\mathbf{n}_i \hat{=} \text{exterior unit normal vector to } \partial C_i$.



Notations used in the formulas below:

$\mathbf{p}_i, \mathbf{p}_j \hat{=} \text{vertices of primal mesh ("location of unknowns } \mu_i, \mu_j)$

$K_1, K_2 \hat{=} \text{triangles adjacent to edge connecting } \mathbf{p}_i \text{ and } \mathbf{p}_j$

Part of the boundary of the control volume C_i :

$$\Gamma_i^K := \partial C_i \cap K .$$

Now, consider $i \neq j \leftrightarrow \text{off-diagonal entries of } \mathbf{A}$:

First, we recall that the intersection of the support of the "tent function" b_h^j with ∂C_i is located inside $K_1 \cup K_2$, see Fig. 272.

$$\blacktriangleright \quad (A)_{ij} \stackrel{(4.2.3.5)}{=} - \int_{\partial C_i} \operatorname{grad} b_h^j \cdot \mathbf{n}_i dS = - \int_{\Gamma_i^{K_1}} \operatorname{grad} b_h^j \cdot \mathbf{n}_i dS - \int_{\Gamma_i^{K_2}} \operatorname{grad} b_h^j \cdot \mathbf{n}_i dS .$$

\leftrightarrow assembly of $(A)_{ij}$ from contributions of the two cells K_1 and K_2 , cf. Section 2.4.5, page 2.4.5.

Next observe that $\operatorname{grad} b_h^j$ is piecewise constant, which implies

$$\operatorname{div} \operatorname{grad} b_h^j = 0 \quad \text{in } K_1 , \quad \operatorname{div} \operatorname{grad} b_h^j = 0 \quad \text{in } K_2 . \quad (4.2.3.6)$$

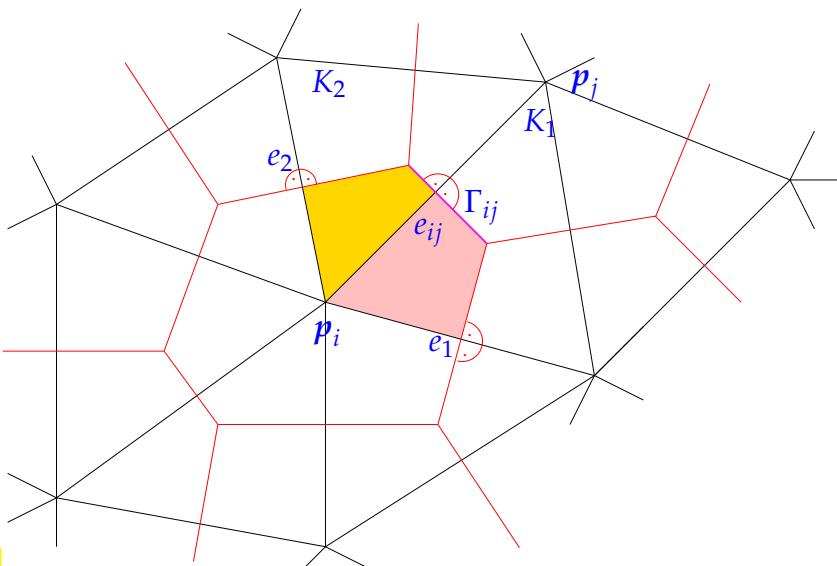


Fig. 273

Now apply Gauss' theorem Thm. 1.5.2.4 to the domains $C_i \cap K_1$ and $C_i \cap K_2$ (shaded in figure).

Also use again that $\mathbf{grad} b_h^j \equiv \text{const}$ on K_1 and K_2 .

Another important observation; conclusion from $\mathbf{grad} \lambda_i$ -formula from Section 2.4.5:

$$\begin{aligned}\mathbf{grad} b_h^j &\perp e_1 \quad \text{in } K_1, \\ \mathbf{grad} b_h^j &\perp e_2 \quad \text{in } K_2.\end{aligned}$$

$$(4.2.3.6) \Rightarrow \int_{\partial(K_\ell \cap C_i)} \mathbf{grad} b_h^j \cdot \mathbf{n} \, dS = 0, \quad \ell = 1, 2,$$

$$\begin{aligned}\blacktriangleright (\mathbf{A})_{ij} &= \frac{1}{2} \int_{e_1} \mathbf{grad} b_h^j|_{K_1} \cdot \mathbf{n}_{e_1} \, dS + \frac{1}{2} \int_{e_{ij}} \mathbf{grad} b_h^j|_{K_1} \cdot \mathbf{n}_{e_{ij}}^1 \, dS \\ &\quad + \frac{1}{2} \int_{e_{ij}} \mathbf{grad} b_h^j|_{K_2} \cdot \mathbf{n}_{e_{ij}}^2 \, dS + \frac{1}{2} \int_{e_2} \mathbf{grad} b_h^j|_{K_1} \cdot \mathbf{n}_{e_2} \, dS. \quad (4.2.3.7)\end{aligned}$$

On the other hand, an entry of finite element Galerkin matrix $\tilde{\mathbf{A}}$ based on linear Lagrangian finite element space $\mathcal{S}_1^0(\mathcal{M})$ can be computed as, see Section 2.4.5:

$$(\tilde{\mathbf{A}})_{ij} = \int_{K_1} \mathbf{grad} b_h^j \cdot \mathbf{grad} b_h^i \, dx + \int_{K_2} \mathbf{grad} b_h^j \cdot \mathbf{grad} b_h^i \, dx.$$

Conduct local integration by parts using Green's first formula from Thm. 1.5.2.7 and taking into account (4.2.3.6) and the linearity of the local shape functions

$$\begin{aligned}\blacktriangleright (\tilde{\mathbf{A}})_{ij} &= \int_{\partial K_1} (\mathbf{grad} b_h^j|_{K_1} \cdot \mathbf{n}_1) b_h^i \, dS + \int_{\partial K_2} (\mathbf{grad} b_h^j|_{K_2} \cdot \mathbf{n}_2) b_h^i \, dS \\ &= \frac{1}{2} |e_1| \mathbf{grad} b_h^j|_{K_1} \cdot \mathbf{n}_{e_1} + \frac{1}{2} |e_{ij}| \mathbf{grad} b_h^j|_{K_1} \cdot \mathbf{n}_{e_{ij}}^1 + \\ &\quad \frac{1}{2} |e_2| \mathbf{grad} b_h^j|_{K_2} \cdot \mathbf{n}_{e_2} + \frac{1}{2} |e_{ij}| \mathbf{grad} b_h^j|_{K_2} \cdot \mathbf{n}_{e_{ij}}^2.\end{aligned}$$

This is the same value as for $(\mathbf{A})_{ij}$ from (4.2.3.7)! Similar considerations apply to the diagonal entries $(\mathbf{A})_{ii}$ and $(\tilde{\mathbf{A}})_{ii}$.



The finite volume discretization and the finite element Galerkin discretization spawn the **same system matrix** for the model problem (4.2.3.1).

Remark 4.2.3.8 (More general finite-volume methods) It must be emphasized that the construction principle of finite volume methods as elaborated in Section 4.2.1 is very versatile and can be applied

to many PDE-based models beyond second-order elliptic boundary value problems, because local flux balance is a key feature of many (physical) systems. Finite-volume methods preserve such structure very well.

An example is non-linear elliptic boundary value problems of the form

$$-\operatorname{div}(\alpha(x) \operatorname{grad} u + \mathbf{f}(x, u)) = f \quad \text{in } \Omega \subset \mathbb{R}^d, \quad u = 0 \quad \text{on } \partial\Omega, \quad (4.2.3.9)$$

where $\mathbf{f} : \Omega \times \mathbb{R} \rightarrow \mathbb{R}^d$ is continuous. In fact, Chapter 11 will be devoted to the construction and analysis of finite-volume methods for a more general class of non-linear problems, known as conservation laws. \square

Review question(s) 4.2.3.10 (Finite-volume methods)

(Q4.2.3.10.A) The equations arising from applying a finite-volume method for the discretization of a second-order elliptic boundary value problem

$$-\operatorname{div}(\kappa(x) \operatorname{grad} u) = f \quad \text{in } \Omega \subset \mathbb{R}^2, \quad u = g \quad \text{on } \partial\Omega.$$

can be stated as

$$\sum_{k \in \mathcal{U}_i} \Psi(\mu_i, \mu_k) = \int_{C_i} f \, dx \quad \forall i = 1, \dots, \tilde{M}. \quad (4.2.1.7)$$

What are the C_i s and Ψ ?

For the “tensor-product” triangular mesh shown in Fig. 274 sketch

(Q4.2.3.10.B)

- the corresponding Voronoi dual mesh,
- and the associated barycentric dual mesh.

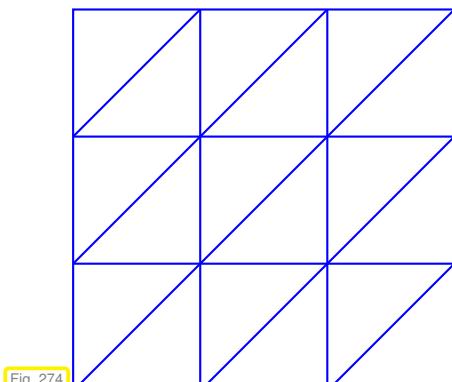


Fig. 274

\triangle

4.3 Spectral Galerkin Methods



Video tutorial for Section 4.3: Spectral Galerkin Methods: (18 minutes) [Download link](#), [tablet notes](#)

Spectral Galerkin methods employ Galerkin discretization as introduced in Section 2.2 using finite-dimensional trial and test spaces of *globally C^∞ -smooth* functions, equipped with *globally supported basis functions*.



Ideas:

Spectral Galerkin approach

- ◆ Trial/test spaces of *globally C^∞ -smooth* functions
- ◆ Globally supported basis functions

This policy contrasts with the use of only piecewise smooth and locally supported basis functions in the finite-element method of Section 2.5. As a consequence, whereas the FEM applied to second-order elliptic BVPs will lead to sparse Galerkin matrices as discussed in § 2.5.3.3, those arising from spectral Galerkin methods will usually be *densely populated*.

§4.3.0.1 (Polynomial spectral Galerkin method) The most natural and widely used trial and test spaces for spectral Galerkin methods are global polynomials spaces.

For second-order elliptic boundary-value problems on the computational domain $\Omega \subset \mathbb{R}^d$ with variational formulation posed over the *unconstrained* Sobolev space $H^1(\Omega)$ (“natural boundary conditions”, § 1.9.0.1), for instance Neumann problems (\rightarrow Ex. 1.8.0.10),

$$u \in H^1(\Omega): \int_{\Omega} \kappa(x) \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \int_{\Omega} f v \, dx + \int_{\partial\Omega} h v \, dS \quad \forall v \in H^1(\Omega),$$

with data $f \in L^2(\Omega)$, $h \in L^2(\partial\Omega)$, there are easy choices for spectral polynomial trial and test spaces of degree $p \in \mathbb{N}$:

$$V_{0,h} = \mathcal{P}_p(\mathbb{R}^2) \Big|_{\Omega} \quad \text{or} \quad V_{0,h} = \mathcal{Q}_p(\mathbb{R}^2) \Big|_{\Omega}, \quad \text{both } \subset H^1(\Omega), \quad (4.3.0.2)$$

where $\mathcal{P}_p(\mathbb{R}^2)$ and $\mathcal{Q}_p(\mathbb{R}^2)$ are the spaces of multi-variate degree- p polynomials introduced in Def. 2.5.2.2 and Def. 2.5.2.7, respectively.

What about *essential boundary conditions* (\rightarrow § 1.9.0.1) as, for instance, we face them for second-order elliptic Dirichlet BVPs with variational problems posed on $H_0^1(\Omega)$?

1 $d = 1$: In one spatial dimension a polynomial spectral Galerkin discretization for Dirichlet BVPs is easy to define and implement. For two-point second-order Dirichlet boundary-value problems on the interval $]a, b[$, $a < b$, with variational formulation

$$u \in H_0^1(]a, b[): \int_a^b \frac{du}{dx}(x) \frac{dv}{dx}(x) \, dx = \int_a^b f(x)v(x) \, dx \quad \forall v \in H_0^1(]a, b[), \quad (2.3.0.1)$$

the following choice of globally polynomial discrete trial and test spaces $V_{0,h} \subset H_0^1(]a, b[)$ is straightforward:

$$V_{0,h} = \mathcal{P}_p(\mathbb{R}) \Big|_{[a,b]} \cap C_0^0([a,b]) = \{v \in \mathcal{P}_p(\mathbb{R}): v(a) = v(b) = 0\} \subset H_0^1(]a, b[). \quad (4.3.0.3)$$

Here, the discretization parameter is the polynomial degree $p \in \mathbb{N}$ and $\dim V_{0,h} = p - 1$.

2 $d > 1$: In higher dimensions $d > 1$ we encounter much more difficulties when trying to enforce Dirichlet (essential) boundary conditions on spectral polynomial trial/test spaces: there is no meaningful way to define polynomial subspaces of $H_0^1(\Omega)$ for a general computational domains $\Omega \subset \mathbb{R}^d$.

An exception are *tensor-product* domains like $\Omega =]a_1, b_1[\times]a_2, b_2[$ for $d = 2$. In this case we can choose a *tensor-product polynomial space* of polynomial degree $p+2$, $p \in \mathbb{N}$:

$$V_{0,h} = \left\{ v(x_1, x_2) = (x_1 - a_1)(b_1 - x_1)q_1(x_1) \cdot (x_2 - a_2)(b_2 - x_2)q_2(x_2), \right. \\ \left. q_1, q_2 \in \mathcal{P}_p(\mathbb{R}), \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \Omega \right\}, \quad (4.3.0.4)$$

which clearly satisfies $V_{0,h} \subset H_0^1(\Omega)$, because the product of the green factors vanishes on $\partial\Omega$. For the special case $\Omega =]0, 1[^2$ and $p = 1$ we have

$$V_{0,h} = \left\{ v(x_1, x_2) = x_1(1 - x_1)(\alpha_1 + \beta_1 x_1) \cdot \right. \\ \left. x_2(1 - x_2)(\alpha_2 + \beta_2 x_2), \alpha_i, \beta_i \in \mathbb{R} \right\}.$$

Remark 4.3.0.5 (Choice of bases for polynomial spectral Galerkin methods) Remember from Section 2.2.2 that a practical Galerkin method entails choosing a concrete basis \mathfrak{B}_h of the finite-dimensional Galerkin trial/test space $V_{0,h}$. We discuss this for the two-point Dirichlet boundary-value problem (2.3.0.1) on $[-1, 1]$ and its polynomial spectral Galerkin discretization based on (4.3.0.3):

$$V_{0,h} := \{v(x) = (1 - x^2)q(x) : q \in \mathcal{P}_{p-2}(\mathbb{R})\}, \quad p \geq 2. \quad (4.3.0.6)$$

- ① A tempting simple choice for \mathfrak{B}_h is the monomial-type basis

$$\mathfrak{B}_h = \{1 - x^2, x(1 - x^2), x^2(1 - x^2), \dots, x^{p-2}(1 - x^2)\}. \quad (4.3.0.7)$$

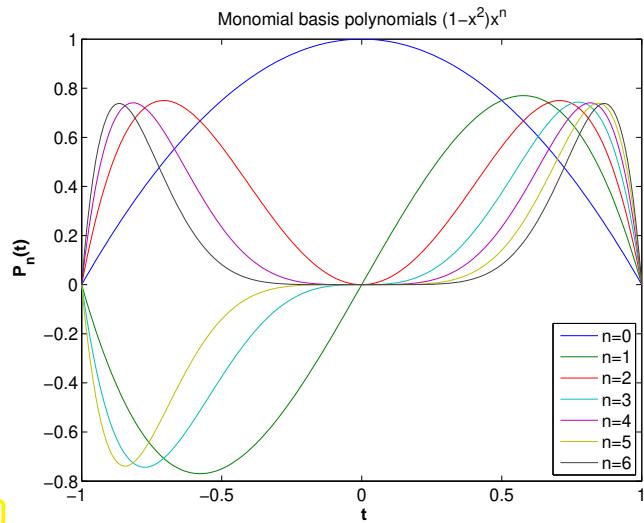


Fig. 275

◀ Monomial basis polynomials

“Visual instability”: for large degree the basis functions look very much alike \leftrightarrow “almost linearly dependent”.

► Monomial basis is **ill-conditioned**, see Exp. 4.3.0.13 below.

Note: in the extreme case of linear dependence of “basis” functions, we certainly lose uniqueness of solutions of the linear system of equations arising from Galerkin discretization.

- ② Alternative: **Integrated Legendre polynomials**

$$\mathfrak{B}_h := \left\{ x \mapsto M_n(x) := \int_{-1}^x P_n(\tau) d\tau, n = 1, \dots, p-1 \right\}, \quad (4.3.0.8)$$

where P_n is the n -th Legendre polynomial according to the following definition.

Definition 4.3.0.9. Legendre polynomials cf. [Hip19, ??]

The n -th Legendre polynomial P_n , $n \in \mathbb{N}_0$, is defined by the Rodriguez formula

$$P_n(x) := \frac{1}{n!2^n} \frac{d^n}{dx^n} [(x^2 - 1)^n]. \quad (4.3.0.10)$$

Since differentiation decreases the degree of a polynomial by one, from (4.3.0.10) we conclude that $P_n \in \mathcal{P}_n$.

The crucial property of the Legendre polynomials is the *orthogonality* relation,

$$\int_{-1}^1 P_n(x) P_m(x) dx = \begin{cases} \frac{2}{2n+1} & , \text{ if } m = n, \\ 0 & \text{else,} \end{cases} \quad (4.3.0.11)$$

that can be shown by integration by parts. For further information about Legendre polynomials refer to [Hip19, ??] and [Hip19, ??].

A first consequence of (4.3.0.11) (, choose $m = 0$,) is that the integrated Legendre polynomials satisfy homogeneous Dirichlet boundary conditions for $[-1, 1]$: $M_n(-1) = M_n(1) = 0$. We also infer that they

are linearly independent. In combination, these properties render \mathfrak{B}_h as in (4.3.0.8) as basis of $V_{0,h}$ from (4.3.0.6).

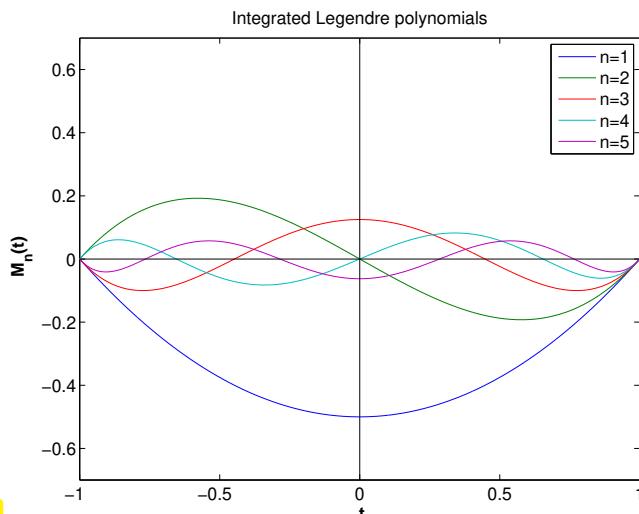


Fig. 276

▷ integrated Legendre polynomials M_1, \dots, M_5

The integrated Legendre polynomials enjoy “visual stability”: the basis functions are very much distinct, that is, “not nearly linearly dependent”.

► We expect $\{M_n\}_{n=1}^{p-1}$ to provide a **well-conditioned** basis of $V_{0,h}$.

For $d > 1$ and on tensor-product computational domains we can simply use products of integrated Legendre polynomials as basis functions. For instance, in the case $\Omega =]a_1, b_1[\times]a_2, b_2[$, to obtain a basis of

$$V_{0,h} = \{v(x_1, x_2) = (x_1 - a_1)(b_1 - x_1)q_1(x_1) \cdot (x_2 - a_2)(b_2 - x_2)q_2(x_2), q_1, q_2 \in \mathcal{P}_p(\mathbb{R})\}, \quad (4.3.0.4)$$

for $p \in \mathbb{N}_0$ we can choose

$$\mathfrak{B}_h = \left\{ \mathbf{x} \mapsto M_n \left(2 \frac{x_1 - a_1}{b_1 - a_1} - 1 \right) M_m \left(2 \frac{x_2 - a_2}{b_2 - a_2} - 1 \right), 1 \leq n, m \leq p+1 \right\}. \quad (4.3.0.12)$$

Also note the affine pullback of the integrated Legendre polynomials to a general interval. □

EXPERIMENT 4.3.0.13 (Conditioning of polynomial spectral Galerkin matrices) We empirically investigate the Euclidean condition number of spectral polynomial Galerkin matrices spawned by the bilinear form

$$\mathbf{a}(u, v) = \int_{-1}^1 \frac{du}{dx}(x) \frac{dv}{dx}(x) dx, \quad u, v \in H_0^1([-1, 1]).$$

Definition [Hip19, ??]. Condition (number) of a matrix

Condition (number) of a matrix $\mathbf{A} \in \mathbb{R}^{n,n}$:

$$\text{cond}(\mathbf{A}) := \|\mathbf{A}^{-1}\| \|\mathbf{A}\|$$

We employ the following bases of the Galerkin trial/test space $V_{0,h} = \mathcal{P}_p \cap H_0^1([-1, 1])$: Either the monomial basis (4.3.0.7) or the integrated Legendre polynomials (4.3.0.8).

Condition number (w.r.t. Euclidean matrix norm) of the Galerkin matrices \triangleright

We observe

- an *exponential increase* in the polynomial degree p of the condition numbers for the *monomial basis*,
- whereas only a moderate increase is seen for the basis comprising integrated Legendre polynomials.

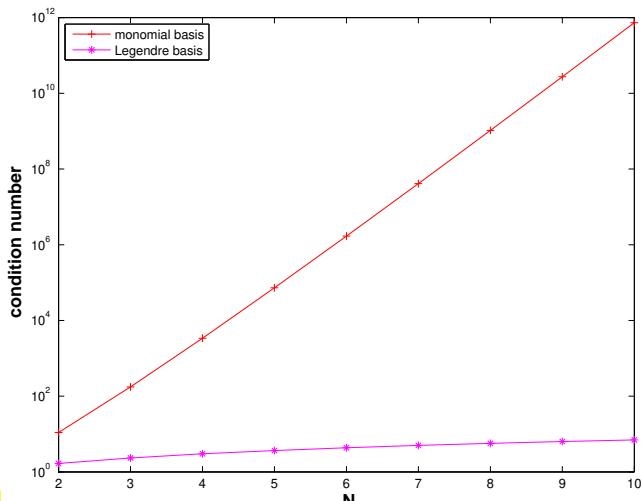


Fig. 277

Finally, recall from [Hip19, ??] that for a linear system of equations $\mathbf{A}\vec{u} = \vec{\phi}$ a huge Euclidean condition number of the coefficient matrix \mathbf{A} means that

- small perturbations of the matrix \mathbf{A} and the right-hand-side vector $\vec{\phi}$ can massively be amplified in the solution,
- and that round-off errors can also have a big impact on \vec{u} .

► Using a monomial-type basis of \mathcal{P}_p for large p is perilous!

§4.3.0.14 (Convergence of polynomial spectral Galerkin solutions) The abstract Galerkin error estimates of Section 3.1 for variational problems with s.p.d. bilinear form, in particular Cea's lemma of Thm. 3.1.3.7 providing the “optimality of Galerkin solutions”, also apply to spectral Galerkin discretizations and let us bound the energy norms of Galerkin discretization errors by those of the best approximation errors.

The maximum norms of the errors incurred by the polynomial approximation of a function $f : [a, b] \rightarrow \mathbb{R}$ were studied in detail in [Hip19, ??] and are given in [Hip19, ??] for functions of *finite smoothness* ($f^{(r)} \in L^\infty([a, b])$, Case **FS**), and in [Hip19, ??] for functions that possess an *analytic extension* to an ellipse $\subset \mathbb{C}$ enclosing $[a, b]$ (Case **A**, [Hip19, ??]). The asymptotic results can be summarized as follows

$$\inf_{q \in \mathcal{P}_p} \|f - q\|_{L^\infty([a, b])} = \begin{cases} O(p^{-r}) & \text{in Case FS ,} \\ O(\rho^{-p}) & \text{for some } \rho > 1 \text{ in Case A .} \end{cases} \quad (4.3.0.15)$$

Thus, in analogy to Def. 3.2.2.1, in Case **FS** we have found asymptotic *algebraic convergence* in the polynomial degree, while in Case **A** the convergence is *exponential*.

Qualitatively, these results remain true for Sobolev norms, in particular the $H^1([a, b])$ -norm, and spaces of uni-variate polynomials like (4.3.0.3) vanishing at the endpoints of an interval. The estimates can also be lifted to the tensor product setting and then apply to trial/test spaces like those specified in (4.3.0.4).

A qualitative summary of the expected asymptotic convergence of polynomial spectral Galerkin approximation schemes for linear scalar second-order elliptic BVPs in one spatial dimension ($d = 1$) or on tensor-product domains ($d > 1$) could read as follows.

Asymptotic convergence of polynomial spectral Galerkin schemes

With $u \in H^1(\Omega)$ the exact solution of a linear scalar second-order elliptic BVP and u_p denoting the polynomial spectral Galerkin solution produced by using global polynomials of degree $p \in \mathbb{N}$, we

can expect for $p \rightarrow \infty$

- algebraic convergence $O(p^{1-m})$, if $u \in H^m(\Omega)$,
- exponential convergence $O(\rho^{-p})$ for some $\rho > 1$, if u has an *analytic extension* (in each variable) to a C^d neighborhood of Ω .

EXPERIMENT 4.3.0.17 (Convergence of polynomial spectral Galerkin method in 1D) We consider the linear 2-point Dirichlet boundary value problem $-\frac{d^2u}{dx^2} = g(x)$ on $]0, 1[$, $u(0) = u(1) = 0$, with exact solution $u(x) = \sin(2\pi x^2)$. We employ Galerkin discretization with degree p polynomial trial and test space $V_{0,h}$ according to (4.3.0.3).

We monitor the $L^\infty(]0, 1[)$ -norm and $L^2(]0, 1[)$ of the discretization error. Both norms are approximated by “overkill” sampling/Gauss quadrature with 10^4 points.

We observe clear asymptotic *exponential convergence* of both error norm for $N := p - 1 \rightarrow \infty$, as seen by almost affine linear error curves in a linear-logarithmic plot, cf. § 3.2.2.5.

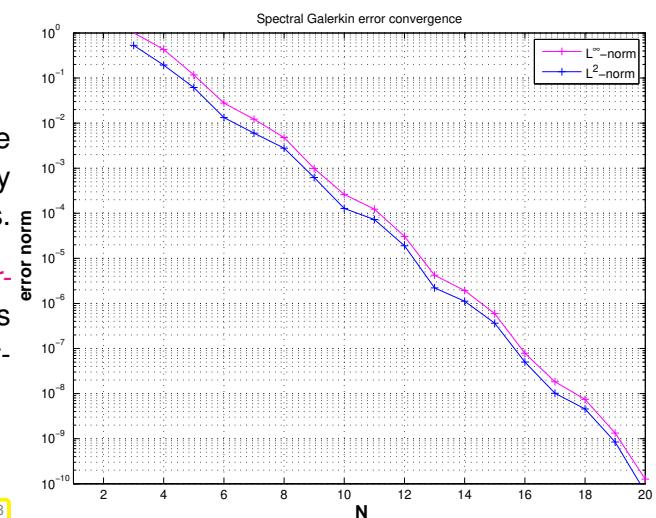


Fig. 278

In this example the exact solution $u(x) = \sin(2\pi x^2)$ has an analytic extension to whole complex plane \mathbb{C} , it is an *entire* function. Thus, in light of the discussion of § 4.3.0.14, exponential convergence of the polynomial spectral Galerkin solution is no surprise,

§4.3.0.18 (Analytic solutions of elliptic BVPs) For $d = 1$, that is, two-point boundary value problems for the differential equation

$$-\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = f(x) \quad \text{in }]a, b[,$$

endowed with any kind of boundary conditions (Dirichlet, Neumann, impedance, Section 1.7) the solution u will possess an analytic extension to a C -neighborhood of $[a, b]$, if this property is satisfied for the uniformly positive function $\sigma : [a, b] \rightarrow \mathbb{R}$ and $f : [a, b] \rightarrow \mathbb{R}$.

For elliptic boundary value problems in higher dimensions $d > 1$, analyticity of the data (right-hand-side source function, boundary data, coefficients) is not enough to ensure analyticity of the solution: It is also required that $\partial\Omega$ is an analytic curve or surface. For instance, even if $f \in L^2(\Omega)$, $\Omega \subset \mathbb{R}^d$, has an analytic extension beyond $\bar{\Omega}$, the solution of

$$-\Delta u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega,$$

may not be analytic in a neighborhood of Ω , if $\partial\Omega$ is not C^∞ -smooth. In particular, on polygonal computational domains, we must not count on analyticity of solutions of BVPs for the Poisson equation $-\Delta u = f$.

§4.3.0.19 (Numerical quadrature for polynomial spectral Galerkin methods) When we apply a polynomial spectral Galerkin method for the solution of the variational two-point boundary value problem

$$u \in H_0^1([a, b]): \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_0^1(\Omega)[a, b],$$

we have to deal with the fact that the functions $\sigma, f \in C^0([a, b])$ may be given only in procedural form, accessible only through point evaluations, cf. Rem. 2.1.2.5. As discussed in § 2.3.3.10 this entails the use of quadrature formulas for the approximate evaluation of the integrals $\int_a^b \dots$ in order to compute the entries of the Galerkin matrix and the right-hand side vector.

This amounts to committing a variational crime. Recalling the discussion of Section 3.5 this is acceptable as long as it does not interfere with the overall asymptotic convergence for polynomial degree $p \rightarrow \infty$: Inevitably and quite naturally, the order (\rightarrow [Hip19, ??]) of the quadrature formula has to be linked to p . We already saw this for the FEM in Section 3.5.1 and a deeper analysis reveals that similar rules of thumb also apply for polynomial spectral Galerkin methods:

The degree- p polynomial spectral Galerkin discretization of a general linear scalar elliptic two-point BVP requires numerical quadrature of order $2p - 1$.

From [Hip19, ??] remember that an order of $2p - 1$ can be achieved by using a **Gauss-Legendre quadrature formula** with $p - 1$ points/nodes. This is the standard choice for polynomial spectral Galerkin methods in one dimension. In higher dimensions $d > 1$ on tensor-product domains **tensor-product Gauss-Legendre quadrature rules** as explained in Ex. 2.7.5.38 can be used. \square

Remark 4.3.0.20 (The quadrature challenge on general domains) In § 4.3.0.1 variational problems posed on the unconstrained space $H^1(\Omega)$, that is, without essential boundary conditions, were said to be amenable to spectral Galerkin discretization for general $\Omega \subset \mathbb{R}^d$ even for $d > 1$, because the full polynomial space $\mathcal{Q}_p(\mathbb{R}^d)$ could always be used as trial/test space.

This is an illusion, because implementation requires high-order quadrature rules on Ω and those are not available apart from tensor-product domain and a few other special shapes like triangles or tetrahedra. Thus, for $d > 1$ the spectral Galerkin method is confined to simple domains also for pure Neumann problems. \square

Remark 4.3.0.21 (Fourier spectral Galerkin methods) Scalar second-order linear elliptic boundary value problems for with **periodic boundary conditions** occur rather frequently in mathematical models. In **one dimension** they read

$$-\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = f(x) \quad \text{in } [a, b], \quad u(a) = u(b), \quad \sigma(a) \frac{du}{dx}(a) = \sigma(b) \frac{du}{dx}(b), \quad (4.3.0.22)$$

with uniformly positive $\sigma C^0([a, b])$ and $f \in L^2([a, b])$. As in Section 1.8 integration by parts in one dimension leads to the variational problem

$$u \in H_{\text{per}}^1([a, b]): \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x)v(x) dx \quad \forall v \in H_{\text{per}}^1([a, b]), \quad (4.3.0.23)$$

where we rely on the space of periodic H^1 -functions on $[a, b]$:

$$H_{\text{per}}^1([a, b]) := \left\{ v|_{[a, b]} : v \in H^1(\mathbb{R}), v(x + (b - a)) = v(x) \quad \forall x \in \mathbb{R} \right\}. \quad (4.3.0.24)$$

A very natural finite-dimensional subspace of $H_{\text{per}}^1([a, b])$ is the $2n + 1$ -dimensional space of $b - a$ -periodic trigonometric polynomials

$$V_{0,h} := \text{Span} \left\{ x \mapsto \cos\left(\frac{2\pi j}{b-a}x\right), x \mapsto \sin\left(\frac{2\pi j}{b-a}x\right) \right\}_{j=0}^n, \quad n \in \mathbb{N}_0, \quad (4.3.0.25)$$

for $[a, b] = [0, 1]$ introduced in [Hip19, ??], [Hip19, ??]. The definition (4.3.0.25) already indicates a viable choice of basis functions. For $d > 1$ and tensor-product domains a tensor-product construction based on (4.3.0.25) can be employed, using a basis analogous to (4.3.0.12).

The general statements about the asymptotic convergence of polynomial spectral Galerkin methods made in § 4.3.0.14 carry over to Fourier spectral Galerkin methods, with the parameter n in (4.3.0.25) playing the role of the polynomial degree.

Parallel to the discussion in § 4.3.0.19 it is clear that usually the implementation of Fourier spectral Galerkin methods cannot avoid using numerical quadrature. This time one prefers the $n - 1$ -point equidistant trapezoidal rule

$$\int_a^b f(t) dt \approx h \sum_{k=0}^{n-1} f(kh), \quad h := \frac{b-a}{n}, \quad n \in \mathbb{N}, \quad [\text{Hip19, ??}]$$

which in [Hip19, ??] was identified as the “magic” numerical quadrature rule for periodic integrands, the counterpart of Gauss-Legendre quadrature in that special case. \square

§4.3.0.26 (Assessment of spectral Galerkin methods) Let us summarize the advantages and drawbacks of the class of spectral Galerkin methods for elliptic boundary value problems:



Possibility to achieve *exponential convergence*, if the exact solution possesses an analytic extension



- ◆ Mere algebraic convergence in case of limited smoothness of the solution
- ◆ Implementations confined to *simple domains*
- ◆ Densely populated Galerkin matrices

Review question(s) 4.3.0.27 (Spectral Galerkin Methods)

(Q4.3.0.27.A) Let $\Omega \subset \mathbb{R}^2$ a bounded domain with a polygonal boundary. Explain, why, *in general*, it is **not** possible to define a polynomial subspace of $H_0^1(\Omega)$ as

$$V_{0,h} := \mathcal{P}_p(\mathbb{R}^2) \Big|_{\Omega} \cap H_0^1(\Omega).$$

(Q4.3.0.27.B) What are the dimensions of the following spaces

$$\begin{aligned} V_1 &:= \mathcal{P}_p(\mathbb{R}) \cap H_0^1([0, 1]), \\ V_2 &:= \left\{ v \in \mathcal{P}_p(\mathbb{R}) : v^{(\ell)}(0) = v^{(\ell)}(1), \ell \in \{0, \dots, p\} \right\} \quad ? \end{aligned}$$

(Q4.3.0.27.C) Consider the bilinear form

$$a(u, v) = \int_0^1 \frac{du}{dx}(x) \frac{dv}{dx}(x) dx, \quad u, v \in H^1([-1, 1]).$$

In the framework of a polynomial spectral Galerkin discretization we use the basis

$$\mathcal{B}_h := \left\{ x \mapsto M_n(x) := \int_{-1}^x P_n(\tau) d\tau, n = 1, \dots, p-1 \right\}, \quad p \in \mathbb{N}, \quad (4.3.0.8)$$

of **integrated Legendre polynomials**. Compute the resulting Galerkin matrix.

Hint. You can use the **orthogonality relation**

$$\int_{-1}^1 P_n(x) P_m(x) dx = \begin{cases} \frac{2}{2n+1} & , \text{ if } m = n \\ 0 & \text{else} . \end{cases} \quad (4.3.0.11)$$

(Q4.3.0.27.D) We consider the Fourier spectral Galerkin discretization of the two-point BVP with periodic boundary conditions

$$u \in H_{\text{per}}^1([a, b]): \quad \int_a^b \sigma(x) \frac{du}{dx}(x) \frac{dv}{dx}(x) dx = \int_a^b f(x) v(x) dx \quad \forall v \in H_{\text{per}}^1([a, b]), \quad (4.3.0.23)$$

using the trial and test space of **trigonometric polynomials**

$$V_{0,h} := \text{Span} \left\{ x \mapsto \cos\left(\frac{2\pi j}{b-a}x\right), x \mapsto \sin\left(\frac{2\pi j}{b-a}x\right) \right\}_{j=0}^n, \quad n \in \mathbb{N}_0. \quad (4.3.0.25)$$

Which formula for the entries of the right-hand-side vector (load vector) has to be implemented, if $f \in C^0([a, b])$ is given only on procedural form as an object of type `std::function<double (double)>`?

(Q4.3.0.27.E) For the Dirichlet problem

$$-\Delta u = f \quad \text{in } \Omega :=]0, 1[^2, \quad u = 0 \quad \text{on } \partial\Omega$$

the right-hand side function f is chosen such that we obtain the exact solution $u(x_1, x_2) = \sin(\pi x_1) \sin(\pi x_2)$.

What kind of asymptotic convergence do you expect from a degree- p spectral polynomial discretization as $p \rightarrow \infty$? Justify your answer.

(Q4.3.0.27.F) Let N denote the number of unknowns in a finite-element method (FEM) and a spectral Galerkin method (SGM) for a second-order elliptic BVP.

- The **FEM**
 - enjoys asymptotic **algebraic convergence** of the energy norm of the discretization error according to $\mathcal{O}(N^{-r})$ for $r \in \mathbb{N}$ and $N \rightarrow \infty$,
 - and involves a computational effort of $\mathcal{O}(N)$ for computing the finite-element solution.
- The **SGM**
 - displays **exponential convergence** $\mathcal{O}(\rho^{-N})$ of the energy norm of the discretization error for some $\rho > 1$ and $N \rightarrow \infty$,
 - and incurs $\mathcal{O}(N^3)$ cost for computing u_h .

Compare the efficiency of both methods for large N .

△

4.4 Collocation Methods



Video tutorial for Section 4.4: Collocation Methods: (16 minutes) [Download link](#), tablet notes

§4.4.0.1 (Starting point: classical solution concept) At the outset of the derivation of the class of collocation-type discretization methods for boundary value problems for second-order scalar boundary value problem is the **classical** concept of solutions already mentioned in Rem. 1.5.3.10. For the Dirichlet boundary value problem

$$-\operatorname{div}(\alpha(x) \operatorname{grad} u) = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega \quad (4.4.0.2)$$

this means that

- one assumes $f \in C^0(\Omega)$, $g \in C^0(\partial\Omega)$, and $\alpha \in C^1(\bar{\Omega})^{3,3}$,
- and seeks a solution $u \in C^2(\Omega)$ satisfying the PDE and the boundary conditions *in pointwise sense*.

↓

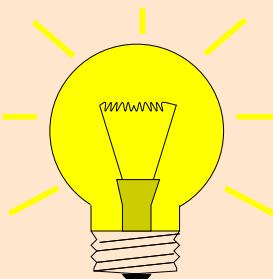
§4.4.0.3 (Collocation principle) We can recast (4.4.0.2) in abstract form,

$$\mathcal{L}u = f \quad \text{in } C^0(\Omega), \quad \mathcal{B}u = g \quad \text{in } C^0(\partial\Omega), \quad (4.4.0.4)$$

where $\mathcal{L} : C^2(\Omega) \rightarrow C^0(\Omega)$ is a *linear* differential operator, and $\mathcal{B} : C^2(\Omega) \rightarrow C^0(\partial\Omega)$ a linear boundary (differential) operator, e.g., the restriction to $\partial\Omega$ as in (4.4.0.2).

Idea of collocation discretization

- ① Seek an approximate solution u_h of (4.4.0.4) in a **finite-dimensional trial space**



$$V_{0,h} \subset C^2(\Omega), \quad N := \dim V_{0,h} < \infty. \quad (4.4.0.6)$$

- ② Pick finitely many **collocation nodes/points**

$$x_1, \dots, x_n \in \Omega, \quad y_1, \dots, y_m \in \partial\Omega, \quad m, n \in \mathbb{N}.$$



Impose **collocation conditions**:

$$u_h \in V_{0,h}: \quad \begin{aligned} (\mathcal{L}u_h)(x_j) &= f(x_j), & j &= 1, \dots, n, \\ (\mathcal{B}u_h)(y_\ell) &= g(y_\ell), & j &= 1, \dots, m. \end{aligned} \quad (4.4.0.7)$$

Sometimes boundary conditions can already be built into the trial space $V_{0,h}$, which can also be an affine space as for the “offset function trick” § 1.4.1.9. In this case no collocation points $y_\ell \in \partial\Omega$ and collocation conditions involving the boundary operator \mathcal{B} are needed.

↓

§4.4.0.8 (Choice of collocation points) The collocation nodes have to fit the trial space in the following sense.

The trial space $V_{0,h}$ and the collocation nodes $\mathbf{x}_j, j = 1, \dots, n$, and $\mathbf{y}_\ell, \ell = 1, \dots, m$ have to satisfy:
For all $f_1, \dots, f_n \in \mathbb{R}, g_1, \dots, g_m \in \mathbb{R}$ the **interpolation problem**

$$u_h \in V_{0,h}: \quad u(\mathbf{x}_j) = f_j, \quad j = 1, \dots, n \quad u(\mathbf{y}_\ell) = g_\ell, \quad \ell = 1, \dots, m, \quad (4.4.0.9)$$

has a unique solution.

By elementary linear algebra a **necessary condition** for the above requirement is

$$N := \dim V_{0,h} = m + n. \quad (4.4.0.10)$$

We remark, that another criterion for a good choice of collocation nodes is that the induced interpolation operator $\mathbb{R}^{m+n} \rightarrow V_{0,h}$ defined by (4.4.0.9) has a small ∞ -norm, also known as Lebesgue constant, see [Hip19, ??]. This is motivated by the convergence theory of collocation methods [Hac95], which is beyond the scope of this section. \square

§4.4.0.11 (From collocation conditions to systems of equations) As explained for the Galerkin method in Section 2.2.2, also collocation methods involve a crucial second step after the collocation conditions (4.4.0.7) have been established.

Collocation method: second step

Choose an ordered basis $\mathfrak{B}_h = \{b_h^1, \dots, b_h^N\}$ of $V_{0,h}$ and plug the basis representation

$$u_h = \mu_1 b_h^1 + \dots + \mu_N b_h^N, \quad \mu_1, \dots, \mu_N \in \mathbb{R}, \quad (4.4.0.13)$$

into the collocation conditions (4.4.0.7), yielding the **collocation equations**

$$\mu_1(\mathbf{L}b_h^1)(\mathbf{x}_j) + \dots + \mu_N(\mathbf{L}b_h^N)(\mathbf{x}_j) = f(\mathbf{x}_j), \quad j = 1, \dots, n, \quad (4.4.0.14a)$$

$$\mu_1(\mathbf{B}b_h^1)(\mathbf{y}_\ell) + \dots + \mu_N(\mathbf{B}b_h^N)(\mathbf{y}_\ell) = g(\mathbf{y}_\ell), \quad \ell = 1, \dots, m. \quad (4.4.0.14b)$$

The transition from (4.4.0.7) and (4.4.0.13) to (4.4.0.14) exploits the linearity of the operators \mathbf{L} and \mathbf{B} .

As in the case of the Galerkin method, if the trial space is an affine space contained in a larger vector space \mathcal{V} , one usually supplements (4.4.0.13) with an **offset function** $u_0 \in \mathcal{V}$, cf. § 1.2.3.12:

$$u_h = u_0 + \mu_1 b_h^1 + \dots + \mu_N b_h^N, \quad \mu_1, \dots, \mu_N \in \mathbb{R}. \quad (4.4.0.15)$$

It is clear that the condition $N = m + n$ from (4.4.0.10) ensures matching numbers of unknowns μ_k and equations. If (4.4.0.10) is satisfied, then (4.4.0.14) is a square linear system of equations. \square

4.4.1 Spectral Collocation Method

We have seen that the trial spaces for a collocation method for second-order elliptic BVPs have to fulfill the rather stringent smoothness requirement $V_{0,h} \subset C^2(\Omega)$. Those are naturally satisfied for the globally smooth trial/test spaces presented in Section 4.3 for the spectral Galerkin method, see, in particular, § 4.3.0.1 and Rem. 4.3.0.21. The considerations of Rem. 4.3.0.5 still apply.

EXAMPLE 4.4.1.1 (Spectral collocation on a square) We discuss the spectral collocation method for the Dirichlet boundary value problem for the Poisson equation on a square

$$-\Delta u = f \quad \text{in } \Omega := [-1, 1]^2, \quad u = g \quad \text{on } \partial\Omega. \quad (4.4.1.2)$$

We require that $f \in C^0(\Omega)$, $g \in C^0(\partial\Omega)$ are available in procedural form.

As **trial space** we choose the space of tensor product polynomials $V_{0,h} = \mathcal{Q}_p$ of degree $p \in \mathbb{N}$, see Def. 2.5.2.7, and have $N := \dim V_{0,h} = (p+1)^2$ by Lemma 2.5.2.8. A basis for $V_{0,h}$ is provided by the $(p+1)^2$ **tensor-product Legendre polynomials**

$$\mathfrak{B}_h := \{\mathbf{x} = [x_1, x_2] \mapsto P_r(x_1)P_s(x_2) : r, s \in \{0, \dots, p\}\}, \quad (4.4.1.3)$$

with the Legendre polynomials P_n as defined in Def. 4.3.0.9. The rationale behind (4.4.1.3) is the good conditioning of this basis, remember Rem. 4.3.0.5. For convenience we label the basis expansion coefficients also with two indices and rewrite (4.4.0.13) as

$$u_h(\mathbf{x}) = \sum_{r,s=0}^p \mu_{r,s} P_r(x_1)P_s(x_2), \quad \mathbf{x} := \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \overline{\Omega}. \quad (4.4.1.4)$$

As **collocation nodes** we start from the stretched Chebychev nodes

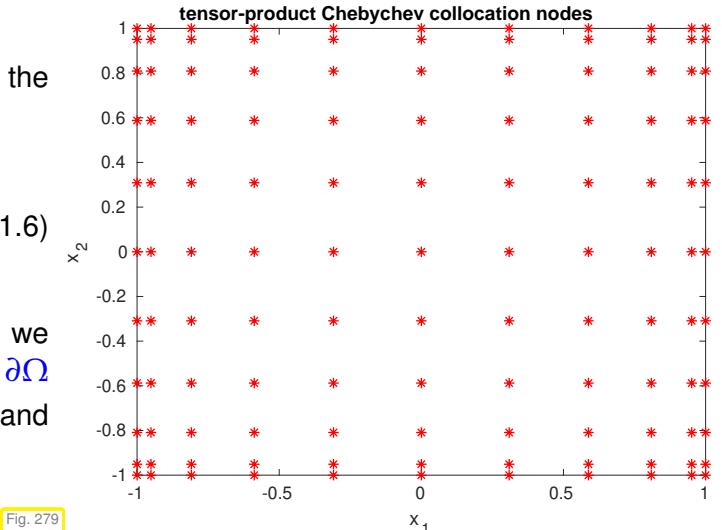
$$\xi_j := \cos\left(\frac{\pi}{p} j\right) \in [-1, 1], \quad j = 0, \dots, p. \quad (4.4.1.5)$$

They arise from the regular Chebychev nodes by a slight dilation and inherit their small Lebesgue constants for the induced polynomial Lagrangian interpolation scheme as discussed in [Hip19, ??]. Note that $\xi_0 = 1$ and $\xi_p = -1$.

A **tensor-product construction** yields the $(p+1)^2 = N$ collocation nodes

$$\begin{aligned} & \{\mathbf{x}_j\}_{j=1}^n \cup \{\mathbf{y}_\ell\}_{\ell=1}^m \\ &= \left\{ [\xi_r, \xi_s]^\top \in \overline{\Omega} : r, s \in \{0, \dots, p\} \right\}. \end{aligned} \quad (4.4.1.6)$$

We have lumped together both types of nodes: we call those in the interior of Ω the \mathbf{x}_j s, those on $\partial\Omega$ the \mathbf{y}_ℓ s. Consequently, we have $n = (p-1)^2$ and $m = 4p$.



With these specifications and owing to

$$\Delta\{\mathbf{x} \mapsto P_r(x_1)P_s(x_2)\} = P_r''(x_1)P_s(x_2) + P_r(x_1)P_s''(x_2),$$

the final (square) linear system of equations (4.4.0.14) becomes

$$\begin{aligned} & - \sum_{r,s=0}^p \mu_{r,s} (P_r''(\xi_k)P_s(\xi_j) + P_r(\xi_k)P_s''(\xi_j)) = f\left(\begin{bmatrix} \xi_k \\ \xi_j \end{bmatrix}\right), \quad k, j \in \{1, \dots, p-1\}, \\ & \sum_{r,s=0}^p \mu_{r,s} P_r(y_{\ell,1})P_s(y_{\ell,2}) = g(\mathbf{y}_\ell), \quad \mathbf{y}_\ell = \begin{bmatrix} y_{\ell,1} \\ y_{\ell,2} \end{bmatrix}, \quad \ell = 1, \dots, m. \end{aligned} \quad (4.4.1.7)$$

This is $(p+1)^2$ equations for the $(p+1)^2$ unknowns $\mu_{r,s}$, $r, s \in \{0, \dots, p\}$. □

EXPERIMENT 4.4.1.8 (Convergence of spectral collocation in 1D) As in Exp. 4.3.0.17 we focus on the linear 2-point boundary value problem $-\frac{d^2 u}{dx^2} = f(x)$, $u(0) = u(1) = 0$ on $\Omega =]0, 1[$, and impose the exact solution $u(x) = \sin(2\pi x^2)$ by a suitable function f .

We employ the one-dimensional counterpart of the spectral collocation method presented in Ex. 4.4.1.1.

Again we track the $L^\infty([0,1])$ -norm and $L^2([0,1])$ of the discretization error $u - u_h$. We approximate those norms as in Exp. 4.3.0.17. Besides, we also recorded a “discrete energy norm” of the error:

$$\text{energy norm(error)}^2 := \sum_{j=1}^p h_j \left| \frac{u_h(\xi_j) - u_h(\xi_{j-1})}{\xi_j - \xi_{j-1}} - \frac{du}{dx}(\tfrac{1}{2}(\xi_j + \xi_{j-1})) \right|^2.$$

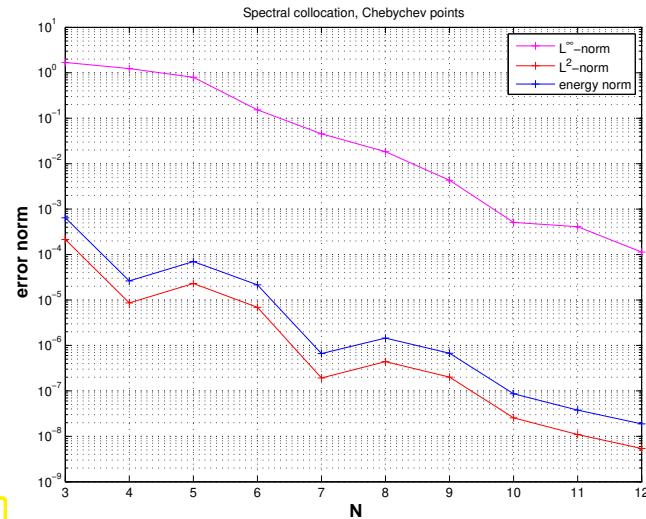


Fig. 280

We conjecture asymptotic **exponential convergence** of all error norms as $p \rightarrow \infty$.

▷ This is suggested by the approximate alignment of the points in a linear-logarithmic plot of the error versus $N := p + 1$.

Also in this case analyticity of the exact solution beyond $[0,1]$ seems to lead to exponential convergence of the spectral collocation solution.

4.4.2 Spline Collocation Methods

First we consider $d = 1$, two-point linear scalar elliptic BVPs on an interval $[a, b]$. We have seen that even in this case the trial space $V_{0,h}$ for a collocation method has to satisfy $V_{0,h} \subset C^2([a, b])$.

The piecewise polynomial trial space $\mathcal{S}_p^0(\mathcal{M})$ used in the finite-element method fall way short of offering this smoothness. However, in one dimension we already know piecewise polynomial functions that are twice continuously differentiable. Recall [Hip19, ??], cf. [Hip19, ??]:

Definition 4.4.2.1. Cubic spline

A function $s : [a, b] \mapsto \mathbb{R}$ is a **cubic spline** function w.r.t. the ordered knot set $\mathcal{T} := \{a = x_0 < x_1 < x_2 < \dots < x_{M-1} < x_M = b\}$, if

- (i) $s \in C^2([a, b])$ (twice continuously differentiable),
- (ii) $s|_{[x_{j-1}, x_j]} \in \mathcal{P}_3(\mathbb{R})$ (a **piecewise cubic polynomial**)

☞ notation: $\mathcal{S}_{3,\mathcal{T}} \hat{=} \text{vector space of cubic splines on knot set } \mathcal{T}$

We known from [Hip19, ??] that

$$\dim \mathcal{S}_{3,\mathcal{T}} = \#\mathcal{T} + 2 = M + 3,$$

and in [Hip19, ??] we learned that the knots x_j are valid interpolation nodes provided that two extra conditions are imposed on the spline $s \in \mathcal{S}_{3,\mathcal{T}}$, like the conditions for **natural cubic spline interpolation** $s''(a) = s''(b) = 0$, see [Hip19, ??]. This suggests the following **trial space** for **cubic spline collocation** based on a given knot set $\mathcal{T} := \{a = x_0 < x_1 < x_2 < \dots < x_{M-1} < x_M = b\}$:

$$V_{0,h} := \{s \in \mathcal{S}_{3,\mathcal{T}} : s''(a) = s''(b) = 0\}. \quad (4.4.2.2)$$

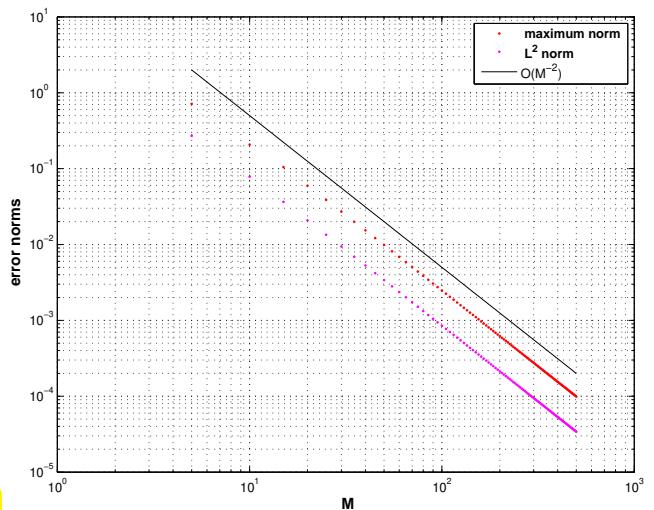
We have $\dim V_{0,h} = M + 1$, which agrees with $\#\mathcal{T}$. Since the knots in \mathcal{T} also serve as **collocation points**, the dimension of the trial space matches the number of collocation points.

In higher dimensions $d > 1$, if Ω is a tensor-product domain, the tensor product construction elaborated above for global polynomial spaces can straightforwardly be adapted to spline spaces. Products of knot sets will supply suitable collocation points in this case.

EXPERIMENT 4.4.2.3 () We revisit the two-point BVP from Exp. 4.3.0.17, Blue $-\frac{d^2u}{dx^2} = g(x)$ on $]0,1[$, $u(0) = u(1) = 0$, f such that $u(x) = \sin(2\pi x^2)$. We employ spline collocation on **equidistant** knot sets using the trial space as in (4.4.2.2).

As in Exp. 4.3.0.17 we monitor approximate error norms $\|u - u_h\|_{L^\infty(\Omega)}[0,1]$ and $\|u - u_h\|_{L^2(\Omega)}[0,1]$ as functions of the number M of knots = collocation points. \triangleright

We observe a clear asymptotic **algebraic convergence** $O(M^{-2})$ for $N \rightarrow \infty$, because the error points lie on neat lines with slope 2 in a doubly logarithmic plot, remember § 3.2.2.5.



Of course, since spline collocation relies on piecewise polynomials of fixed degree, we cannot expect exponential convergence. However, in light of [Hip19, ??], a convergence rate of 2 is a bit disappointing and might be caused by the “nonphysical” boundary conditions $u''_h(0) = u''_h(1) = 0$ enforced in (4.4.2.2). \square

§4.4.2.4 (Assessment: Collocation methods) Let us try to summarize the main advantages and drawbacks of collocation methods also in comparison with the FEM:



- ◆ Numerical quadrature not required
- ◆ Implementations confined to *simple domains*
- ◆ Danger of “stupid choice” of collocation nodes
- ◆ Weaker theoretical guarantees



Review question(s) 4.4.2.5 (Collocation Methods)

(Q4.4.2.5.A) The collocation approach to the abstract boundary value problem

$$\mathcal{L}u = f \quad \text{in} \quad C^0(\Omega) , \quad \mathcal{B}u = g \quad \text{in} \quad C^0(\partial\Omega) , \quad (4.4.0.4)$$

converts it into the system of equations

$$\begin{aligned} \mu_1(\mathcal{L}b_h^1)(x_j) + \dots + \mu_N(\mathcal{L}b_h^N)(x_j) &= f(x_j) , \quad j = 1, \dots, m , \\ \mu_1(\mathcal{B}b_h^1)(y_\ell) + \dots + \mu_N(\mathcal{B}b_h^N)(y_\ell) &= g(y_\ell) , \quad \ell = 1, \dots, m . \end{aligned}$$

- What are the b_h^k , x_j , y_ℓ , and μ_k s?
- Restate the above system of equations in matrix-vector form.

(Q4.4.2.5.B) Sketch a polynomial spectral collocation method for the Neumann boundary value problem on a square domain

$$-\Delta u = f \quad \text{in} \quad \Omega :=]-1, 1[^2, \quad \mathbf{grad} u \cdot \mathbf{n} = 0 \quad \text{on} \quad \partial\Omega.$$

△

Learning outcomes

The chapter aims to impart

- the gist of the “finite difference approach”: starting from strong form of a partial differential equation replace derivatives by difference quotients anchored on a regular grid (finite lattice).
- awareness that finite difference schemes can usually be recovered as finite element discretization (plus quadrature) on special (regular) meshes.
- the principles of the finite volume discretization of 2nd-order elliptic boundary value problems.
- the idea of using dual meshes as a tool to construct control volumes for a finite volume discretization.
- knowledge about spectral Galerkin discretizations based on globally smooth trial and test spaces
- awareness of the strengths and weaknesses and weaknesses of spectral Galerkin methods.
- the idea underlying collocation methods.

Bibliography

- [Hac95] W. Hackbusch. *Integral equations. Theory and numerical treatment.* Vol. 120. International Series of Numerical Mathematics. Basel: Birkhäuser, 1995 (cit. on p. 428).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering.* 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 397, 398, 420–422, 424, 425, 428, 429, 431, 432).
- [Mul+11] Patrick Mullen, Pooran Memari, Fernando de Goes, and Mathieu Desbrun. “HOT: Hodge-optimized Triangulations”. In: *ACM Trans. Graph.* 30.4 (July 2011), 103:1–103:12. DOI: [10.1145/2010324.1964998](https://doi.org/10.1145/2010324.1964998) (cit. on p. 411).

Chapter 5

Non-Linear Elliptic Boundary Value Problems

5.1 Non-linear Elastic Membrane Models

5.1.1 Thin Elastic String Model

5.1.1.1 Configuration Space (Recalled)

5.1.1.2 Mass-Spring Model

5.1.1.3 Continuum Limit

5.1.2 Thin Membrane Model

5.1.3 Taut Membrane Limit

5.2 Variational Approach

5.2.1 Virtual Work Equation

5.2.2 Non-Linear Variational Equations

5.2.3 Elastic Membrane Boundary Value Problem

5.3 Ritz-Galerkin Discretization

5.3.1 Abstract Galerkin Discretization on Non-Linear Variational Problems

5.3.2 Newton's Method in Function Space

5.3.3 Galerkin Discretization of Linearized Variational Problems

Chapter 6

Numerical Integration – Single Step Methods

In this chapter we are concerned with **ordinary differential equations (ODEs)**, the special case of a differential equation whose unknown is a function,

- (i) which depends on a *single independent variable* often representing time in models,
- (ii) and which takes values in a fixed *finite-dimensional* space.

Remember that second-order elliptic PDEs that we studied in Chapter 1 had to be supplemented with boundary conditions in order to obtain a well-posed boundary-value problem. Similarly, ODEs have to be supplied with so-called **initial values** in order to arrive at meaningful (well-posed) **initial-value problem (IVP)**, for which we can expect existence and uniqueness of solutions. Hence, this chapter is devoted to the derivation and analysis of numerical methods for the approximate solution of initial-value problems for ordinary differential equations.

Remark 6.0.0.1 (Why “numerical integration”?) For historical reasons the approximate solution of initial value problems for ordinary differential equations is called “Numerical Integration”, because solving an ODE was often referred to as “integrating it”. □

Contents

6.1	Initial-Value Problems (IVPs) for Ordinary Differential Equations (ODEs)	437
6.1.1	Ordinary Differential Equations (ODEs)	437
6.1.2	Mathematical Modeling with Ordinary Differential Equations: Examples	439
6.1.3	Theory of Initial-Value-Problems (IVPs)	444
6.1.4	Evolution Operators	448
6.2	Introduction: Polygonal Approximation Methods	452
6.2.1	Explicit Euler method	453
6.2.2	Implicit Euler method	455
6.2.3	Implicit midpoint method	457
6.3	General Single-Step Methods	458
6.3.1	Definition	459
6.3.2	(Asymptotic) Convergence of Single-Step Methods	463
6.4	Explicit Runge-Kutta Single-Step Methods (RKSSMs)	472
6.5	Adaptive Stepsize Control	479
6.5.1	The Need for Timestep Adaptation	479
6.5.2	Local-in-Time Stepsize Control	481
6.5.3	Embedded Runge-Kutta Methods	489



Supplementary literature. Some grasp of the meaning and theory of ordinary differential

equations (ODEs) is indispensable for understanding the construction and properties of numerical methods. Relevant information can be found in [Str09, Sect. 5.6, 5.7, 6.5].

Books dedicated to numerical methods for ordinary differential equations:

- [DB02] excellent textbook, but geared to the needs of students of mathematics.
- [HNW93] and [HW11] : *the* standard reference.
- [HLW06]: wonderful book conveying deep insight, with emphasis on mathematical concepts.

6.1 Initial-Value Problems (IVPs) for Ordinary Differential Equations (ODEs)



Video tutorial for Section 6.1: Initial-Value Problems (IVPs) for Ordinary Differential Equations (ODEs): (63 minutes) [Download link](#), [tablet notes](#)

In this section we introduce notations and fundamental concepts, present a few examples of models involving ODEs and briefly review the relevant mathematical theory.

6.1.1 Ordinary Differential Equations (ODEs)

§6.1.1.1 (Terminology and notations related to ordinary differential equations) In our parlance, a **(first-order) ordinary differential equation** (ODE) is an equation of the form

$$\dot{\mathbf{y}} := \frac{d\mathbf{y}}{dt}(t) = \mathbf{f}(t, \mathbf{y}(t)), \quad (\text{ODE})$$

with

- ☞ a (continuous) **right hand side function** (r.h.s) $\mathbf{f} : I \times D \rightarrow \mathbb{R}^N$ of **time** $t \in \mathbb{R}$ and **state** $\mathbf{y} \in \mathbb{R}^N$,
- ☞ defined on a (finite) time interval $I \subset \mathbb{R}$, and **state space** D , which is some sub-set of \mathbb{R}^N : $D \subset \mathbb{R}^N$, $N \in \mathbb{N}$.
- ☞ Notation (due to Newton): $\dot{\cdot} \triangleq$ (total) derivative with respect to time t

An ODE is called **autonomous**, if the right-hand-side function \mathbf{f} does not depend on time: $\mathbf{f} = \mathbf{F}(\mathbf{y})$.

In the context of mathematical modeling the state vector $\mathbf{y} \in \mathbb{R}^N$ is supposed to provide a complete (in the sense of the model) description of a system. In a sense, “state space” is a synonym for “configuration space” introduced in Notion 1.2.1.1. Then (ODE) models a **finite-dimensional dynamical system**. Examples will be provided below, see Ex. 6.1.2.1, Ex. 6.1.2.5, and Ex. 6.1.2.7.

For $N > 1$ $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ can be viewed as a **system of ordinary differential equations**:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \iff \begin{bmatrix} \dot{y}_1 \\ \vdots \\ \dot{y}_N \end{bmatrix} = \begin{bmatrix} f_1(t, y_1, \dots, y_N) \\ \vdots \\ f_N(t, y_1, \dots, y_N) \end{bmatrix}.$$

Definition 6.1.1.2. Solution of an ordinary differential equation

A **solution** of the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ with continuous right hand side function \mathbf{f} is a continuously differentiable **function** “of time t ” $\mathbf{y} : J \subset I \rightarrow D$, defined on an **open** interval J , for which $\dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t))$ holds for all $t \in J$ (\cong “**pointwise**”).

A solution describes a continuous **trajectory** in state space, a one-parameter family of states, parameterized by time.

It goes without saying that smoothness of the right hand side function \mathbf{f} is inherited by solutions of the ODE:

Lemma 6.1.1.3. Smoothness of solutions of ODEs

Let $\mathbf{y} : I \subset \mathbb{R} \rightarrow D$ be a solution of the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ on the time interval I .

If $\mathbf{f} : D \rightarrow \mathbb{R}^N$ is r -times continuously differentiable with respect to both arguments, $r \in \mathbb{N}_0$, then the trajectory $t \mapsto \mathbf{y}(t)$ is $r + 1$ -times continuously differentiable in the interior of I .

§6.1.1.4 (Scalar autonomous ODE: Solution by principals) We consider **scalar ODEs**, namely (ODE) in the case $N = 1$, and, in particular $\dot{y} = f(y)$ with $f : D \subset \mathbb{R} \rightarrow \mathbb{R}$, D an interval.

We embark on **formal calculations**. Assume that f is continuous and $f(t) \neq 0$ for all $t \in D$. Further, suppose that we know a **principal** $F : D \rightarrow \mathbb{R}$ of $\frac{1}{f}$, that is, a function $y \mapsto F(y)$ satisfying $\frac{dF}{dy} = \frac{1}{f}$ on D . Then, by the chain rule, every solution $y : I \subset \mathbb{R} \rightarrow \mathbb{R}$ of $\dot{y} = f(y)$ also solves

$$\frac{d}{dt} F(y(t)) = \frac{1}{f(y(t))} \dot{y}(t) = 1, \quad t \in D. \Leftrightarrow F(y(t)) = t - t_0 \quad \text{for some } t_0 \in \mathbb{R}. \quad (6.1.1.5)$$

We also know that F is monotonic and, thus, possesses an inverse function F^{-1} . Integrating (6.1.1.5) and applying the fundamental theorem of calculus, we find

$$y(t) = F^{-1}(t - t_0) \quad \text{for some } t_0 \in I. \quad (6.1.1.6)$$

This formula describes a one-parameter family of functions (t_0 is the parameter), all of which provide a solution of $\dot{y} = f(y)$ on a suitable interval.

A particularly simple case is $f(y) = \lambda y + c$, $\lambda, c \in \mathbb{R}$, the scalar ODE $\dot{y} = \lambda y + c$. Following the steps outlined above, we calculate the solution

$$\left[F(y) = \frac{1}{\lambda} \log(\lambda y + x) \quad \Rightarrow \right] \quad y(t) = \frac{1}{\lambda} \left(e^{\lambda(t-t_0)} - c \right), \quad t \in \mathbb{R}. \quad (6.1.1.7)$$

§6.1.1.8 (Linear ordinary differential equations) Now we take a look at the simplest class of ODEs, which is also the most important.

Definition 6.1.1.9. Linear first-order ODE

A first-order ordinary differential equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, as introduced in § 6.1.1.1 is **linear**, if

$$\mathbf{f}(t, \mathbf{y}) = \mathbf{A}(t)\mathbf{y}, \quad \mathbf{A} : I \rightarrow \mathbb{R}^{N,N} \quad \text{a continuous function}. \quad (6.1.1.10)$$

Lemma 6.1.11. Space of solutions of linear ODEs

The set of solutions $\mathbf{y} : I \rightarrow \mathbb{R}^N$ of (6.1.1.10) is a vector space.

Proof. We have to show that, if $\mathbf{y}, \mathbf{z} : I \rightarrow \mathbb{R}^N$ are two solutions of (6.1.1.10), then so are $\mathbf{y} + \mathbf{z}$ and $\alpha \mathbf{y}$ for all $\alpha \in \mathbb{R}$. This is an immediate consequence of the linearity of the operations of differentiation and matrix \times vector multiplication. \square

For the scalar case $N = 1$ (6.1.1.10) can be written as $\dot{y} = a(t)y$ with a continuous function $a : I \rightarrow \mathbb{R}$. In this case, the chain rule immediately verifies that for fixed $t_0 \in I$ every function

$$y(t) = C \exp\left(\int_{t_0}^t a(\tau) d\tau\right), \quad C \in \mathbb{R}, \quad (6.1.1.12)$$

is a solution.

If the matrix $\mathbf{A} \in \mathbb{R}^{N,N}$ does not depend on time, (6.1.1.10) is known as a linear ODE with constant coefficients: $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}$. In this case we can choose $I = \mathbb{R}$, and the ODE can be solved by a **diagonalization technique** [Str09, Bemerkung 5.6.1], [NS02, Sect. 8.1]: If

$$\mathbf{A} = \mathbf{S}\mathbf{D}\mathbf{S}^{-1}, \quad \mathbf{S} \in \mathbb{R}^{N,N} \text{ regular}, \quad \mathbf{D} = \text{diag}(\lambda_1, \dots, \lambda_N) \in \mathbb{R}^{N,N}, \quad (6.1.1.13)$$

we can rewrite

$$\dot{\mathbf{y}} = \mathbf{S}\mathbf{D}\mathbf{S}^{-1}\mathbf{y} \Rightarrow \dot{\mathbf{z}} = \mathbf{D}\mathbf{z} \quad \text{with} \quad \mathbf{z}(t) := \mathbf{S}^{-1}\mathbf{y}(t).$$

We get N **decoupled** scalar linear equations $\dot{z}_\ell = \lambda_\ell z_\ell$, $\ell = 1, \dots, N$. Returning to \mathbf{y} we find that every solution $\mathbf{y} : \mathbb{R} \rightarrow \mathbb{R}^N$ of $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}$ can be written as

$$\mathbf{y}(t) = \mathbf{S} \begin{bmatrix} e^{\lambda_1 t} & & \\ & \ddots & \\ & & e^{\lambda_N t} \end{bmatrix} \mathbf{S}^{-1} \mathbf{w} \quad \text{for some } \mathbf{w} \in \mathbb{R}^N. \quad (6.1.1.14)$$

□

6.1.2 Mathematical Modeling with Ordinary Differential Equations: Examples

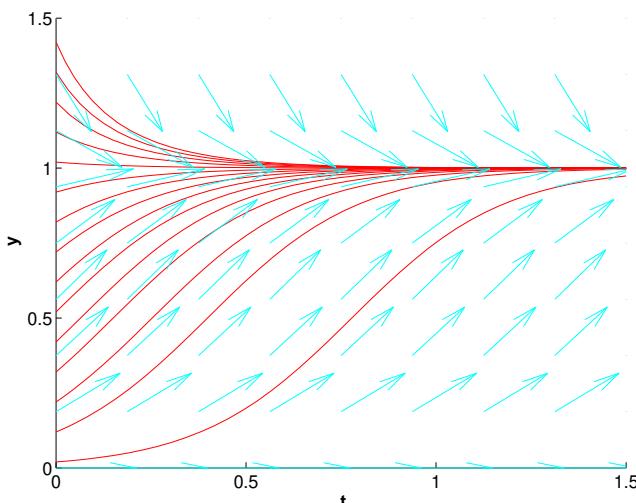
Most models of physical systems and phenomena that are *continuously changing with time* involve ordinary differential equations.

EXAMPLE 6.1.2.1 (Growth with limited resources) [Ama83, Sect. 1.1], [Han02, Ch. 60] This is an example from **population dynamics** with a one-dimensional state space $D = \mathbb{R}_0^+$, $N = 1$. The interpretation of $y : [0, T] \mapsto \mathbb{R}$ is that of the population density of bacteria as a function of time. A scaled, non-dimensional model is assumed, cf. Rem. 1.2.1.25.

ODE-based model: autonomous **logistic differential equations** [Str09, Ex. 5.7.2]

$$\dot{y} = f(y) := (\alpha - \beta y) y \quad (6.1.2.2)$$

- ◆ $y \hat{=} \text{population density}$, $[y] = \frac{1}{\text{m}^2}$
- $\dot{y} \hat{=} \text{instantaneous change (growth/decay) of population density}$
- ◆ growth rate $\alpha - \beta y$ with growth coefficients $\alpha, \beta > 0$, $[\alpha] = \frac{1}{\text{s}}$, $[\beta] = \frac{\text{m}^2}{\text{s}}$: decreases due to fiercer competition as population density increases.

Solution for different $y(0)$ ($\alpha, \beta = 5$)

Note that by fixing the initial value $y(0)$ we can single out a *unique* representative from the family of solutions. This will turn out to be a general principle, see Section 6.1.3. \square

Definition 6.1.2.4. Autonomous ODE

An ODE of the form $\dot{y} = f(y)$, that is, with a right hand side function that does not depend on time, but only on state, is called **autonomous**.

For an autonomous ODE the right hand side function defines a **vector field** (“velocity field”) $y \mapsto f(y)$ on state space.

EXAMPLE 6.1.2.5 (Predator-prey model) [Ama83, Sect. 1.1], [HLW06, Sect. 1.1.1], [Han02, Ch. 60], [DR08, Ex. 11.3]) We consider the following model from population dynamics:

Predators and prey coexist in an ecosystem. Without predators the population of prey would be governed by a simple exponential growth law. However, the growth rate of prey will decrease with increasing numbers of predators and, eventually, become negative. Similar considerations apply to the predator population and lead to an ODE model.

ODE-based model: autonomous **Lotka-Volterra ODE**:

$$\begin{aligned} \dot{u} &= (\alpha - \beta v)u \\ \dot{v} &= (\delta u - \gamma)v \end{aligned} \leftrightarrow \dot{y} = f(y) \quad \text{with} \quad y = \begin{bmatrix} u \\ v \end{bmatrix}, \quad f(y) = \begin{bmatrix} (\alpha - \beta v)u \\ (\delta u - \gamma)v \end{bmatrix}, \quad (6.1.2.6)$$

with positive model parameters $\alpha, \beta, \gamma, \delta > 0$.

population densities:

$u(t)$ → density of prey at time t ,

$v(t)$ → density of predators at time t

Right hand side vector field \mathbf{f} for Lotka-Volterra ODE

▷

Solution curves are trajectories of particles carried along by velocity field \mathbf{f} .

(Parameter values for Fig. 283: $\alpha = 2, \beta = 1, \delta = 1, \gamma = 1$.)

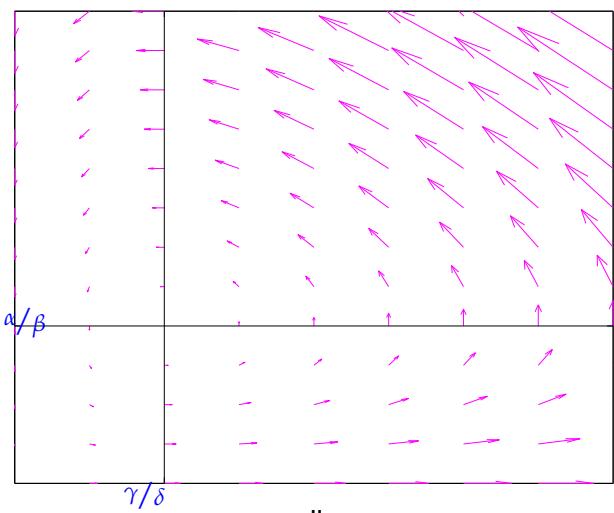


Fig. 283

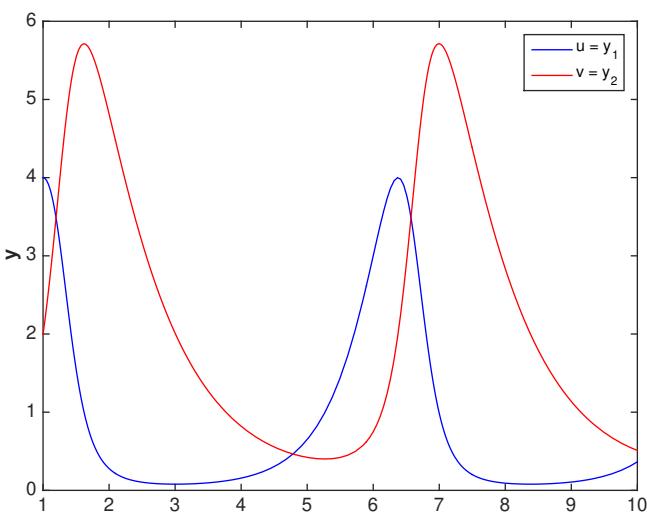


Fig. 284

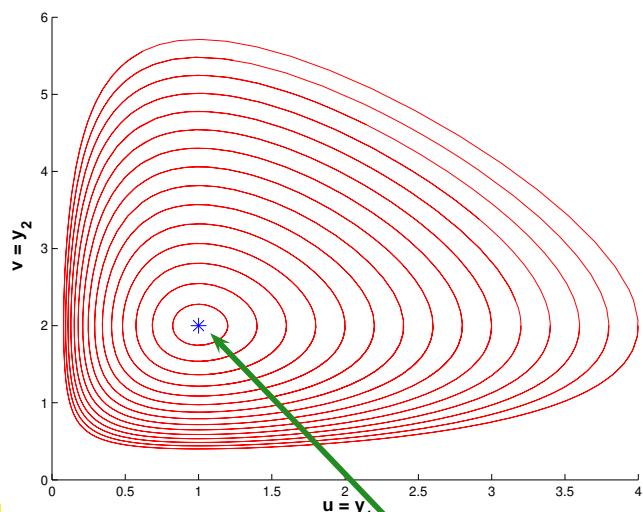


Fig. 285

$$\text{Solution } \begin{bmatrix} u(t) \\ v(t) \end{bmatrix} \text{ for } \mathbf{y}_0 := \begin{bmatrix} u(0) \\ v(0) \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

(Parameter values for Fig. 285, 284: $\alpha = 2, \beta = 1, \delta = 1, \gamma = 1$)

Solution curves for (6.1.2.6)

stationary point

EXAMPLE 6.1.2.7 (Heartbeat model → [Dea80, p. 655]) This example deals with a *phenomenological model* from physiology. A model is called phenomenological, if it is entirely motivated by observations without appealing to underlying mechanisms or first principles.

State of heart described by quantities: $\begin{aligned} l &= l(t) &\hat{=} & \text{length of muscle fiber} \\ p &= p(t) &\hat{=} & \text{electro-chemical potential} \end{aligned}$

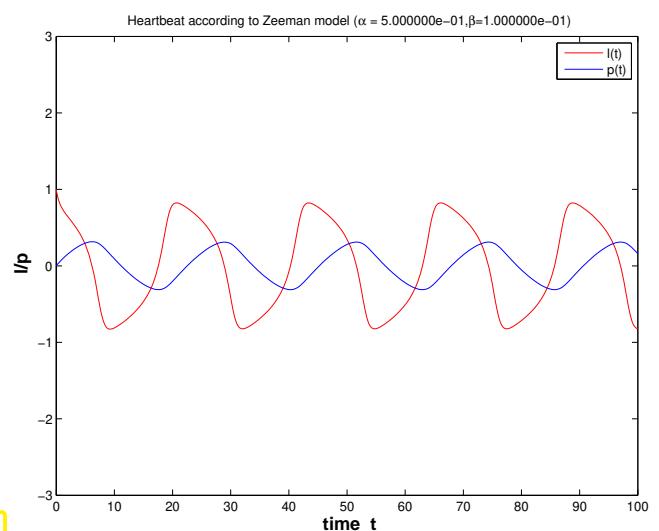
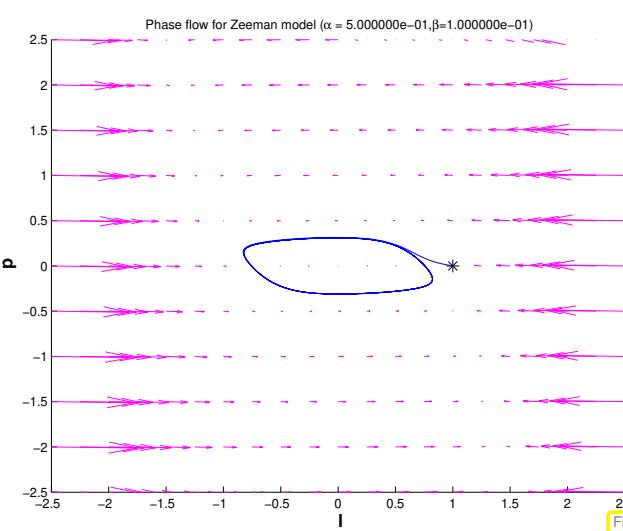
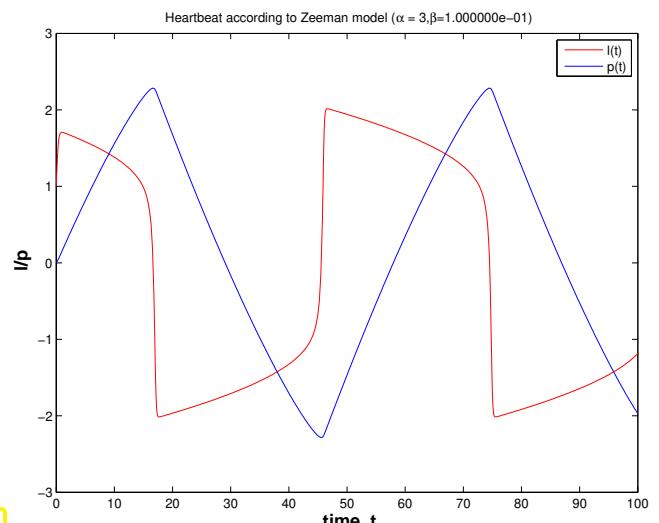
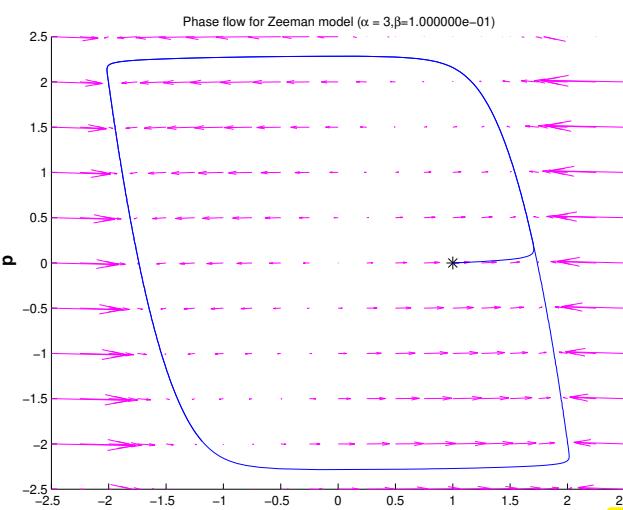
$$\text{Phenomenological model: } \begin{aligned} \dot{l} &= -(l^3 - \alpha l + p), \\ \dot{p} &= \beta l, \end{aligned} \quad (6.1.2.8)$$

with parameters: $\alpha \hat{=} \text{pre-tension of muscle fiber}$

$\beta \hat{=} \text{(phenomenological) feedback parameter}$

This is the so-called Zeeman model: it is a phenomenological model entirely based on macroscopic observations without relying on knowledge about the underlying molecular mechanisms.

Plots of vector fields for (6.1.2.8) and solutions for different choices of parameters are given next:



Observation: $\alpha \ll 1$ (bottom plots) \rightarrow ventricular fibrillation, a life-threatening condition.

EXAMPLE 6.1.2.9 (SIR model for spread of local epidemic [Het00]) The field of epidemiology tries to understand the spread of contagious diseases in populations. It heavily relies on ODEs in its mathematical modeling. This example presents a particularly simple model for an epidemic in a large, stable, isolated, and vigorously mixing homogeneous population.

With respect to the disease we partition the population into three different groups and introduce time-dependent variables for their fractions $\in [0, 1]$:

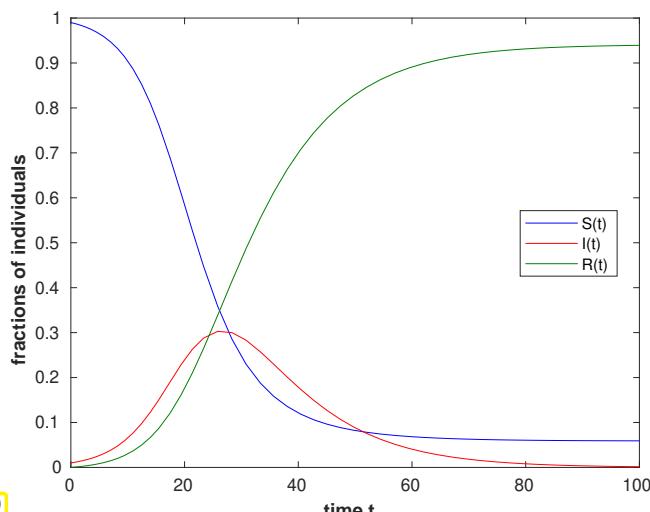
- (I) $S = S(t)$ $\hat{=}$ fraction of **susceptible** persons, who can still contract the disease,
- (II) $I = I(t)$ $\hat{=}$ fraction of **infected/infectious** persons, who can pass on the disease,
- (III) $R = R(t)$ $\hat{=}$ fraction of **recovered/removed** persons, who are immune or have died.

These three quantities enter the SIR model named after the group it considers. Besides, the model involves two crucial **model parameters**, which have to be determined from data:

1. A parameter $\beta > 0$, whose value expresses the probability of transmission, and
2. a parameter $r > 0$, taking into account how quickly sick people recover or die.

With these notation the ODE underlying the SIR model can be stated as

$$\dot{S}(t) = -\beta S(t)I(t) , \quad \dot{I}(t) = \beta S(t)I(t) - rI(t) , \quad \dot{R}(t) = rI(t) . \quad (6.1.2.10)$$



▷ Evolution of an epidemic according to the SIR model (6.1.2.10) for
• $\beta = 0.3, r = 0.1$,
• $S(0) = 0.99, I(0) = 0.01, R(0) = 0.0$
(non-dimensionalized time)

Note that in this case not all people end up infected,
 $\lim_{t \rightarrow \infty} S(t) > 0!$

EXAMPLE 6.1.2.11 (Transient circuit simulation [Han02, Ch. 64]) [Hip19, ??] and [Hip19, ??] discuss circuit analysis as a source of linear and non-linear systems of equations, see [Hip19, ??] and [Hip19, ??]. The former example admitted time-dependent currents and potentials, but dependence on time was confined to be “sinusoidal”. This enabled us to switch to frequency domain, see [Hip19, ??], which gave us a complex linear system of equations for the complex nodal potentials. Yet, this trick is only possible for *linear* circuits. In the general case, circuits have to be modelled by ODEs connecting time-dependent potentials and currents. This will be briefly explained now.

The approach is transient **nodal analysis**, cf. [Hip19, ??], based on the **Kirchhoff current law** [Hip19, ??], which reads for the node • of the simple circuit drawn in Fig. 291

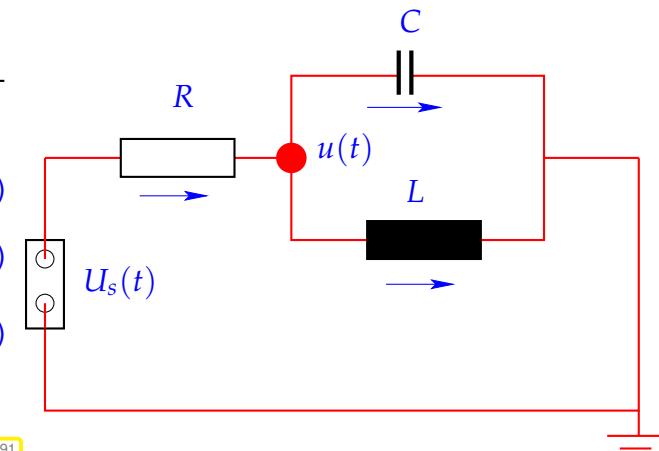
$$i_R(t) - i_L(t) - i_C(t) = 0. \quad (6.1.2.12)$$

In addition we rely on known transient constitutive relations for basic linear circuit elements:

$$\text{resistor: } i_R(t) = R^{-1}u_R(t), \quad (6.1.2.13)$$

$$\text{capacitor: } i_C(t) = C \frac{du_C}{dt}(t), \quad (6.1.2.14)$$

$$\text{coil: } u_L(t) = L \frac{di_L}{dt}(t). \quad (6.1.2.15)$$



We assume that the source voltage $U_s(t)$ is given. To apply nodal analysis to the circuit of Fig. 291 we differentiate (6.1.2.12) w.r.t. t

$$\frac{di_R}{dt}(t) - \frac{di_L}{dt}(t) - \frac{di_C}{dt}(t) = 0,$$

and plug in the above constitutive relations for circuit elements:

$$\Rightarrow R^{-1} \frac{du_R}{dt}(t) - L^{-1}u_L(t) - C \frac{d^2u_C}{dt^2}(t) = 0.$$

We continue following the policy of nodal analysis and express all voltages by potential differences between nodes of the circuit.

$$u_R(t) = U_s(t) - u(t), \quad u_C(t) = u(t) - 0, \quad u_L(t) = u(t) - 0.$$

For this simple circuit there is only one node with unknown potential, see Fig. 291. Its time-dependent potential will be denoted by $u(t)$ and this is the unknown of the model, a function of time satisfying the ordinary differential equation

$$R^{-1}(\dot{U}_s(t) - \dot{u}(t)) - L^{-1}u(t) - C\frac{d^2u}{dt^2}(t) = 0.$$

- This is an autonomous **2nd-order** ordinary differential equation:

$$C\ddot{u} + R^{-1}\dot{u} + L^{-1}u = R^{-1}\dot{U}_s. \quad (6.1.2.16)$$

The attribute “2nd-order” refers to the occurrence of a second derivative with respect to time. □

6.1.3 Theory of Initial-Value-Problems (IVPs)

§6.1.3.1 (Initial value problems) We start with an abstract mathematical description that also introduces key terminology:

A generic **Initial value problem** (IVP) for a **first-order ordinary differential equation** (ODE) (→ [Str09, Sect. 5.6], [DR08, Sect. 11.1]) can be stated as: find a function $\mathbf{y} : I \rightarrow D$ that satisfies, cf. Def. 6.1.1.2,

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0. \quad (6.1.3.2)$$

- $\mathbf{f} : I \times D \mapsto \mathbb{R}^N \hat{=} \text{right hand side}$ (r.h.s.) ($N \in \mathbb{N}$),
- $I \subset \mathbb{R} \hat{=} \text{(time)interval} \leftrightarrow \text{"time variable" } t$,
- $D \subset \mathbb{R}^N \hat{=} \text{state space/phase space} \leftrightarrow \text{"state variable" } \mathbf{y}$,
- $\Omega := I \times D \hat{=} \text{extended state space}$ (of tuples (t, \mathbf{y})),
- $t_0 \in I \hat{=} \text{initial time}, \quad \mathbf{y}_0 \in D \hat{=} \text{initial state} \succ \text{initial conditions}$.

The time interval I may be finite or infinite. Frequently, the extended state space is not specified, but assumed to coincide with the maximal domain of definition of \mathbf{f} . Sometimes, the model suggests constraints on D , for instance, positivity of certain components that represent a density. □

§6.1.3.3 (IVPs for autonomous ODEs) Recall Def. 6.1.2.4: For an autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, that is the right hand side \mathbf{f} does not depend on time t .

Hence, for autonomous ODEs we have $I = \mathbb{R}$ and the right hand side function $\mathbf{y} \mapsto \mathbf{f}(\mathbf{y})$ can be regarded as a stationary vector field (velocity field), see Fig. 283 or Fig. 286.

An important observation: If $t \mapsto \mathbf{y}(t)$ is a solution of an autonomous ODE, then, for any $\tau \in \mathbb{R}$, also the shifted function $t \mapsto \mathbf{y}(t - \tau)$ is a solution.

- For initial value problems for autonomous ODEs the initial time is irrelevant and therefore we can always make the “canonical choice $t_0 = 0$ ”.

Autonomous ODEs naturally arise when modeling **time-invariant** systems or phenomena. All examples for Section 6.1.2 belong to this class. □

§6.1.3.4 (Autonomization: Conversion into autonomous ODE) In fact, autonomous ODEs already represent the general case, because every ODE can be converted into an autonomous one:

The idea is to include time as an extra $N+1$ -st component of an extended state vector $\mathbf{z}(t)$. This solution component has to grow linearly \Leftrightarrow its temporal derivative must be $= 1$

$$\mathbf{z}(t) := \begin{bmatrix} \mathbf{y}(t) \\ t \end{bmatrix} = \begin{bmatrix} \mathbf{z}' \\ z_{N+1} \end{bmatrix}: \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \Leftrightarrow \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}), \mathbf{g}(\mathbf{z}) := \begin{bmatrix} \mathbf{f}(z_{N+1}, \mathbf{z}') \\ 1 \end{bmatrix}.$$

This means $\dot{z}_{N+1} = 1$ and implies $z_{N+1}(t) = t + t_0$, if t_0 stands for the initial time in the original non-autonomous IVP.

➤ We restrict ourselves to autonomous ODEs in the remainder of this chapter. □

Remark 6.1.3.5 (From higher order ODEs to first order systems [DR08, Sect. 11.2])

An ordinary differential equation of **order** $n \in \mathbb{N}$ has the form

$$\boxed{\mathbf{y}^{(n)} = \mathbf{f}(t, \mathbf{y}, \dot{\mathbf{y}}, \dots, \mathbf{y}^{(n-1)})}. \quad (6.1.3.6)$$

where, with notations from § 6.1.3.1, $\mathbf{f} : I \times D \times \dots \times D \rightarrow \mathbb{R}^N$ is a function of time t and n state arguments.

☞ Notation: superscript $(n) \doteq n$ -th temporal derivative $t: \frac{d^n}{dt^n}$

No special treatment of higher order ODEs is necessary, because (6.1.3.6) can be turned into a 1st-order ODE (a system of size nN) by adding all derivatives up to order $n-1$ as additional components to the state vector. This extended state vector $\mathbf{z}(t) \in \mathbb{R}^{nd}$ is defined as

$$\mathbf{z}(t) := \begin{bmatrix} \mathbf{y}(t) \\ \mathbf{y}^{(1)}(t) \\ \vdots \\ \mathbf{y}^{(n-1)}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{bmatrix} \in \mathbb{R}^{Nn}: (6.1.3.6) \Leftrightarrow \dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}), \mathbf{g}(\mathbf{z}) := \begin{bmatrix} \mathbf{z}_2 \\ \mathbf{z}_3 \\ \vdots \\ \mathbf{z}_n \\ \mathbf{f}(t, \mathbf{z}_1, \dots, \mathbf{z}_n) \end{bmatrix}. \quad (6.1.3.7)$$

Note that the extended system requires initial values $\mathbf{y}(t_0), \dot{\mathbf{y}}(t_0), \dots, \mathbf{y}^{(n-1)}(t_0)$:

For ODEs of order $n \in \mathbb{N}$ well-posed initial value problems need to specify initial values for the first $n-1$ derivatives.

§6.1.3.9 (Smoothness classes for right-hand side functions) Now we review results about existence and uniqueness of solutions of initial value problems for first-order ODEs. These are surprisingly general and do not impose severe constraints on right hand side functions. Some kind of smoothness of the right-hand side function \mathbf{f} is required, nevertheless and the following definitions describe it in detail.

Definition 6.1.3.10. Lipschitz continuous function (\rightarrow [Str09, Def. 4.1.4])

Let $\Theta := I \times D$, $I \subset \mathbb{R}$ an interval, $D \subset \mathbb{R}^N$, $N \in \mathbb{N}$, an open domain. A function $\mathbf{f} : \Theta \mapsto \mathbb{R}^N$ is **Lipschitz continuous** (in the second argument) on Θ , if

$$\exists L > 0: \|\mathbf{f}(t, \mathbf{w}) - \mathbf{f}(t, \mathbf{z})\| \leq L \|\mathbf{w} - \mathbf{z}\| \quad \forall (t, \mathbf{w}), (t, \mathbf{z}) \in \Theta. \quad (6.1.3.11)$$

Definition 6.1.3.12. Local Lipschitz continuity (\rightarrow [Str09, Def. 4.1.5])

Let $\Omega := I \times D$, $I \subset \mathbb{R}$ an interval, $D \subset \mathbb{R}^N$, $N \in \mathbb{N}$, an open domain. A function $\mathbf{f} : \Omega \mapsto \mathbb{R}^N$ is **locally Lipschitz continuous**, if for every $(t, \mathbf{y}) \in \Omega$ there is a closed box B with $(t, \mathbf{y}) \in B$ such that \mathbf{f} is Lipschitz continuous on B :

$$\begin{aligned} \forall (t, \mathbf{y}) \in \Omega: \quad & \exists \delta > 0, L > 0: \\ & \|\mathbf{f}(\tau, \mathbf{z}) - \mathbf{f}(\tau, \mathbf{w})\| \leq L \|\mathbf{z} - \mathbf{w}\| \\ & \forall \mathbf{z}, \mathbf{w} \in D: \|\mathbf{z} - \mathbf{y}\| \leq \delta, \|\mathbf{w} - \mathbf{y}\| \leq \delta, \forall \tau \in I: |t - \tau| \leq \delta. \end{aligned} \quad (6.1.3.13)$$

The property of local Lipschitz continuity means that the function $(t, \mathbf{y}) \mapsto \mathbf{f}(t, \mathbf{y})$ has “locally finite slope” in \mathbf{y} . \square

EXAMPLE 6.1.3.14 (A function that is not locally Lipschitz continuous [Str09, Bsp. 6.5.3]) The meaning of local Lipschitz continuity is best explained by giving an example of a function that fails to possess this property.

Consider the square root function $t \mapsto \sqrt{t}$ on the *closed* interval $[0, 1]$. Its slope in $t = 0$ is infinite and so it is not locally Lipschitz continuous on $[0, 1]$.

However, if we consider the square root on the *open* interval $]0, 1[$, then it is locally Lipschitz continuous there. \square

The next lemma gives a simple criterion for local Lipschitz continuity, which can be proved by the mean value theorem, *cf.* the proof of [Hip19, ??].

Lemma 6.1.3.15. Criterion for local Lipschitz continuity

If \mathbf{f} and $D_{\mathbf{y}}\mathbf{f}$ are continuous on the extended state space Ω , then \mathbf{f} is locally Lipschitz continuous (\rightarrow Def. 6.1.3.12).

☞ Notation: $D_{\mathbf{y}}\mathbf{f} \triangleq$ the derivative of \mathbf{f} w.r.t. the state variable \mathbf{y} , a Jacobian matrix $\in \mathbb{R}^{N,N}$ as defined in (0.3.2.16).

The following is the the most important mathematical result in the theory of initial-value problems for ODEs:

Theorem 6.1.3.16. Theorem of Peano & Picard-Lindelöf [Ama83, Satz II(7.6)], [Str09, Satz 6.5.1], [DR08, Thm. 11.10], [Han02, Thm. 73.1]

If the right hand side function $\mathbf{f} : \Omega \mapsto \mathbb{R}^N$ is locally Lipschitz continuous (\rightarrow Def. 6.1.3.12) then for all initial conditions $(t_0, \mathbf{y}_0) \in \Omega$ the IVP

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0. \quad (6.1.3.2)$$

has a solution $\mathbf{y} \in C^1(J(t_0, \mathbf{y}_0), \mathbb{R}^N)$ with **maximal** (temporal) domain of definition $J(t_0, \mathbf{y}_0) \subset \mathbb{R}$.

In light of § 6.1.3.4 and Thm. 6.1.3.16 henceforth we mainly consider

$$\text{autonomous IVPs: } \boxed{\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad , \quad \mathbf{y}(0) = \mathbf{y}_0}, \quad (6.1.3.17)$$

with locally Lipschitz continuous (\rightarrow Def. 6.1.3.12) right hand side \mathbf{f} .

§6.1.3.18 (Domain of definition of solutions of IVPs) We emphasize a subtle message of Thm. 6.1.3.16.

Solutions of an IVP have an *intrinsic* maximal domain of definition

! Also note that the domain of definition/domain of existence $J(t_0, \mathbf{y}_0)$ of the solution usually depends on the initial values (t_0, \mathbf{y}_0) !

Terminology: if $J(t_0, \mathbf{y}_0) = I$, I the maximal temporal domain of definition of \mathbf{f} , we say that the solution $\mathbf{y} : I \mapsto \mathbb{R}^N$ is **global**.

Notation: for autonomous ODE we always have $t_0 = 0$, and therefore we write $J(\mathbf{y}_0) := J(0, \mathbf{y}_0)$. \square

EXAMPLE 6.1.3.19 (“Explosion equation”: finite-time blow-up) Let us explain the still mysterious “maximal domain of definition” in statement of Thm. 6.1.3.16. It is related to the fact that every solution of an initial value problem (6.1.3.17) has its own largest possible time interval $J(\mathbf{y}_0) \subset \mathbb{R}$ on which it is defined naturally.

As an example we consider the autonomous scalar ($d = 1$) initial value problem, modeling “explosive growth” with a growth rate increasing linearly with the density:

$$\dot{y} = y^2 , \quad y(0) = y_0 \in \mathbb{R}. \quad (6.1.3.20)$$

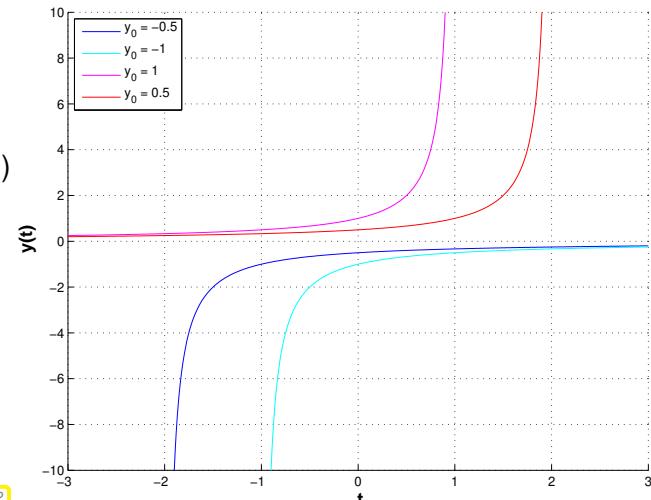
We choose $I = D = \mathbb{R}$. Clearly, $y \mapsto y^2$ is locally Lipschitz-continuous, but only locally! Why not globally?

We find the solutions

$$y(t) = \begin{cases} \frac{1}{y_0^{-1}-t} & , \text{ if } y_0 \neq 0 , \\ 0 & , \text{ if } y_0 = 0 , \end{cases} \quad (6.1.3.21)$$

with domains of definition

$$J(y_0) = \begin{cases}]-\infty, y_0^{-1}[& , \text{ if } y_0 > 0 , \\ \mathbb{R} & , \text{ if } y_0 = 0 , \\]y_0^{-1}, \infty[& , \text{ if } y_0 < 0 . \end{cases}$$



In this example, for $y_0 > 0$ the solution experiences a **blow-up** in finite time and ceases to exist afterwards. \square

§6.1.3.22 (IVPs and BVPs for ODEs) In this course, we have seen ordinary differential equations already in Section 1.5.1. There we derived the 2-point boundary value problem

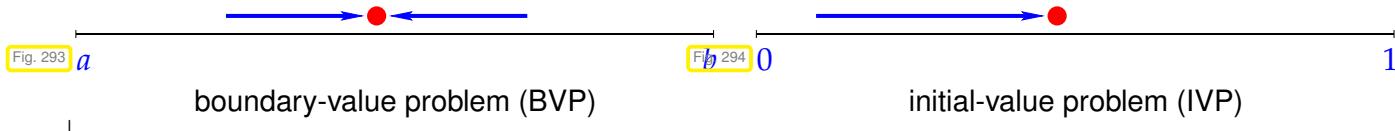
$$-\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = f \quad \text{in }]a, b[, \quad u(a) = u_a , \quad u(b) = u_b . \quad (1.5.1.16)$$

Obviously, this involves a second-order ordinary differential equation $-\frac{d}{dx} \left(\sigma(x) \frac{du}{dx}(x) \right) = f$. Replacing $x \rightarrow t$ and assuming some smoothness of the coefficient $x \mapsto \sigma(x)$, we can rewrite it in the form (6.1.3.6).

Nevertheless, the considerations in this chapter do not include (1.5.1.16), and this does not have to do with properties of the ODE, but with the context, in which it occurs.

In this chapter we are concerned with **initial-value problems** on intervals $[t_0, T]$, which imposes a clear **direction** on time. If $t_0 < T$, then the state solution at time t will depend only on the solution at earlier times and the state at time t will influence the solution only at later times. This reflects causality.

Conversely, for a **boundary value problem** on $[a, b] \subset \mathbb{R}$, the solution at any $x \in [a, b]$ will both depend on and influence the solution at any other point.



 **Supplementary literature.** For other concise summaries of the theory of IVPs for ODEs refer to [QSS00, Sect. 11.1], [DR08, Sect. 11.3].

6.1.4 Evolution Operators

Now we examine a difficult but fundamental concept for time-dependent models stated by means of ODEs. For the sake of simplicity we restrict the discussion to autonomous initial-value problems (IVPs)

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) , \quad \mathbf{y}(0) = \mathbf{y}_0 , \quad (6.1.3.17)$$

with locally Lipschitz continuous (\rightarrow Def. 6.1.3.12) right hand side $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$, $N \in \mathbb{N}$, and make the following assumption. A more general treatment is given in [DB02].

Assumption 6.1.4.1. Global solutions

Solutions of (6.1.3.17) are global: $J(\mathbf{y}_0) = \mathbb{R}$ for **all** $\mathbf{y}_0 \in D$.

Now we return to the study of a generic ODE (ODE) instead of an IVP (6.1.3.2). We do this by temporarily changing the perspective: we fix a “time of interest” $t \in \mathbb{R} \setminus \{0\}$ and follow all trajectories for the duration t . This induces a mapping of points in state space:

$$\geq \text{mapping} \quad \Phi^t : \begin{cases} D & \mapsto D \\ \mathbf{y}_0 & \mapsto \mathbf{y}(t) \end{cases} , \quad t \mapsto \mathbf{y}(t) \text{ solution of IVP (6.1.3.17)} , \quad (6.1.4.2)$$

This is a well-defined mapping of the state space into itself, by Thm. 6.1.3.16 and Ass. 6.1.4.1.

Now, we may also let t vary, which spawns a *family* of mappings $\{\Phi^t\}_{t \in \mathbb{R}}$ of the state space D into itself. However, it can also be viewed as a mapping with two arguments, a duration t and an initial state value \mathbf{y}_0 !

Definition 6.1.4.3. Evolution operator/mapping

Under Ass. 6.1.4.1 the mapping

$$\Phi : \begin{cases} \mathbb{R} \times D & \mapsto D \\ (t, \mathbf{y}_0) & \mapsto \Phi^t \mathbf{y}_0 := \mathbf{y}(t) \end{cases},$$

where $t \mapsto \mathbf{y}(t) \in C^1(\mathbb{R}, \mathbb{R}^N)$ is the unique (global) solution of the IVP $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$, is the **evolution operator**/mapping for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

Note that $t \mapsto \Phi^t \mathbf{y}_0$ describes the solution of $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ for $\mathbf{y}(0) = \mathbf{y}_0$ (a trajectory). Therefore, by virtue of definition, we have

$$\frac{\partial \Phi}{\partial t}(t, \mathbf{y}) = \mathbf{f}(\Phi^t \mathbf{y}). \quad (6.1.4.4)$$

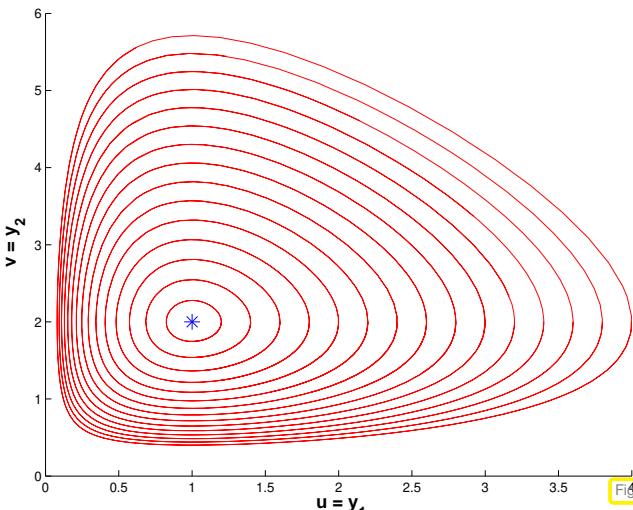
Let us repeat the different kinds of information contained in an evolution operator when viewed from different angles:

$$\begin{array}{lll} t \mapsto \Phi^t \mathbf{y}_0, \quad \mathbf{y}_0 \in D \text{ fixed} & \hat{=} & \text{a trajectory} = \text{solution of an IVP}, \\ \mathbf{y} \mapsto \Phi^t \mathbf{y}, \quad t \in \mathbb{R} \text{ fixed} & \hat{=} & \text{a mapping of the state space onto itself}. \end{array}$$

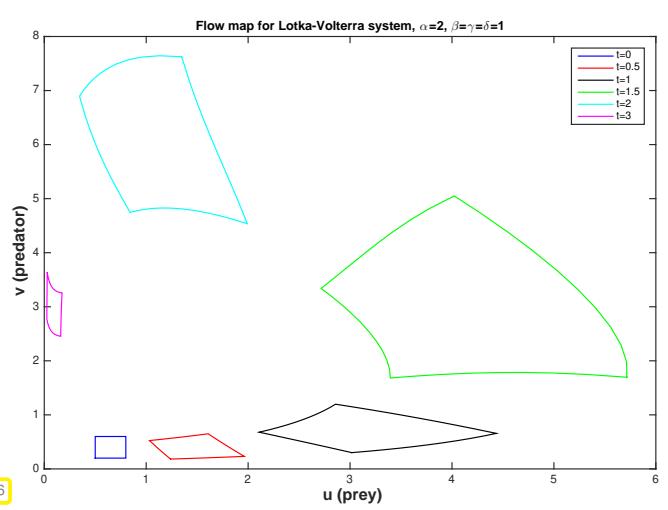
EXAMPLE 6.1.4.5 (Evolution operator for Lotka-Volterra ODE (6.1.2.6)) For $N = 2$ the action of an evolution operator can be visualized by tracking the movement of point sets in state space. Here this is done for the Lotka-Volterra ODE

$$\begin{aligned} \dot{u} &= (\alpha - \beta v)u \\ \dot{v} &= (\delta u - \gamma)v \end{aligned} \leftrightarrow \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad \text{with} \quad \mathbf{y} = \begin{bmatrix} u \\ v \end{bmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{bmatrix} (\alpha - \beta v)u \\ (\delta u - \gamma)v \end{bmatrix}, \quad (6.1.2.6)$$

with positive model parameters $\alpha, \beta, \gamma, \delta > 0$.



trajectories $t \mapsto \Phi^t \mathbf{y}_0$



state mapping $\mathbf{y} \mapsto \Phi^t \mathbf{y}$

Think of $\mathbf{y} \in \mathbb{R}^2 \mapsto \mathbf{f}(\mathbf{y}) \in \mathbb{R}^2$ as the velocity of the surface of a fluid. Specks of floating dust will be carried along by the fluid, patches of dust covering parts of the surface will move and deform over time. This can serve as a “mental image” of Φ . □

Given an evolution operator, we can recover the right-hand side function \mathbf{f} of the underlying autonomous ODE as $\mathbf{f}(\mathbf{y}) = \frac{\partial \Phi}{\partial t}(0, \mathbf{y})$: There is a one-to-one relationship between ODEs and their evolution operators, and those are the key objects behind an ODE.

An ODE “encodes” an evolution operator.

Understanding the concept of evolution operators is indispensable for numerical integration, that is the construction of numerical methods for the solution of IVPs for ODEs:

Numerical integration is concerned with the approximation of evolution operators.

Remark 6.1.4.6 (Group property of autonomous evolutions) Under Ass. 6.1.4.1 the evolution operator gives rise to a **group** of mappings $D \mapsto D$:

$$\Phi^s \circ \Phi^t = \Phi^{s+t} , \quad \Phi^{-t} \circ \Phi^t = Id \quad \forall t \in \mathbb{R} . \quad (6.1.4.7)$$

This is a consequence of the uniqueness theorem Thm. 6.1.3.16. It is also intuitive: following an evolution up to time t and then for some more time s leads us to the same final state as observing it for the whole time $s + t$. \square

Review question(s) 6.1.4.8 (IVPs for ODEs)

(Q6.1.4.8.A) A simple model for the spread of a viral epidemic like SARS-CoV2 is the **SIR model**:

$$\dot{S}(t) = -\beta I(t)S(t) , \quad \dot{I}(t) = \beta I(t)S(t) - \gamma I(t) , \quad \dot{R}(t) = \gamma I(t) , \quad (6.1.4.9)$$

with parameters $\beta, \gamma > 0$. Here $t \mapsto S(t)$ is the fraction of susceptible individuals, $t \mapsto I(t)$ that of infected (and infectious) individuals, and $t \mapsto R(t)$ stands for the removed (immune or dead) individuals.

- Write (6.1.4.9) in the form $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.
- What is a meaningful state space for (6.1.4.9).
- Show that $S(t) + I(t) + R(t) \equiv \text{const}$.
- Show that $t \mapsto R(t)$ is non-decreasing.

(Q6.1.4.8.B) Determine the one-parameter family of solutions of the scalar autonomous ODE $\dot{y} = 1 + y^2$.

Can you expect global solutions defined for all times $t \in \mathbb{R}$?

Hint. $\frac{d}{dy}\{y \mapsto \arctan(y)\} = \frac{1}{1+y^2}$

(Q6.1.4.8.C) Consider the autonomous scalar ODE $\dot{y} = \cos^2 y$.

- What are the stationary states, that is the states y^* that are zeros of the right-hand-side function?
- Compute the (analytic) solution of a related initial-value problem with $y(0) = y_0 \in \mathbb{R}$.
- The evolution operator Φ belonging to $\dot{y} = \cos^2 y$ will satisfy $\Phi^t \circ \Phi^s = \Phi^{t+s}$. Verify this formula based on what you found as analytic solution.

Hint. Remember that $\tan' = \cos^{-2}$.

(Q6.1.4.8.D) Show that the scalar autonomous initial-value problem

$$\dot{y} = \sqrt{y} , \quad y(0) = 0 ,$$

has at least two solutions in the state space \mathbb{R}_0^+ according to the following definition.

Definition Def. 6.1.1.2. Solution of an ordinary differential equation

A **solution** of the ODE $\dot{y} = \mathbf{f}(t, \mathbf{y})$ with continuous right hand side function \mathbf{f} is a continuously differentiable **function** “of time t ” $\mathbf{y} : J \subset I \rightarrow D$, defined on an **open** interval J , for which $\dot{y}(t) = \mathbf{f}(t, \mathbf{y}(t))$ holds for all $t \in J$ ($\hat{=}$ “**pointwise**”).

How can this be reconciled with the assertion of the main theorem?

Theorem Thm. 6.1.3.16. Theorem of Peano & Picard-Lindelöf

If the right hand side function $\mathbf{f} : \hat{\Omega} \mapsto \mathbb{R}^N$ is locally Lipschitz continuous (\rightarrow Def. 6.1.3.12) then for all initial conditions $(t_0, \mathbf{y}_0) \in \hat{\Omega}$ the IVP

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 . \quad (6.1.3.2)$$

has a solution $\mathbf{y} \in C^1(J(t_0, \mathbf{y}_0), \mathbb{R}^N)$ with maximal (temporal) domain of definition $J(t_0, \mathbf{y}_0) \subset \mathbb{R}$.

Hint. Consider the function $y(t) = \left(\frac{1}{2}t\right)^2$.

(Q6.1.4.8.E) For the autonomous scalar ODE $\dot{y} = \sin \frac{1}{y} - 2$ answer the following questions

- What is the maximal state space?
- Which initial values for $t_0 = 0$ will allow a solution on $[0, \infty[$,
- and for which will the solution be defined for a finite time interval only?

Hint. Make use of the geometrically intuitive statement: If a differentiable function $f : [t_0, T] \rightarrow \mathbb{R}$ satisfies $\dot{f}(t) \leq C$ for all $t_0 \leq t \leq T$, then $f(t) \leq f(t_0) + Ct$.

(Q6.1.4.8.F) Rewrite the matrix differential equation $\dot{\mathbf{Y}}(t) = \mathbf{A}\mathbf{Y}(t)$ for $\mathbf{Y} : \mathbb{R} \rightarrow \mathbb{R}^{n,n}$, $n \in \mathbb{N}$, in the standard form $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ with right-hand-side function $\mathbf{f} : \mathbb{R}^N \rightarrow \mathbb{R}^N$ and suitable $N \in \mathbb{N}$.

△

6.2 Introduction: Polygonal Approximation Methods



Video tutorial for Section 6.2: Introduction: Polygonal Approximation Methods: (32 minutes)
[Download link](#), [tablet notes](#)

In this section we will see the first simple methods for the numerical integration (= solution) of initial-value problems (IVPs). We target an initial value problem (6.1.3.2) for a first-order ordinary differential equation

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 . \quad (6.1.3.2)$$

As usual, the right hand side function $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$, $N \in \mathbb{N}$, may be given only in *procedural form*, for instance, in a C++ code as an functor object providing an evaluation operator

```
Eigen::VectorXd operator () (double t, const Eigen::VectorXd &y)
    const;
```

cf. Rem. 2.1.2.5. Occasionally the evaluation of \mathbf{f} may involve costly computations.

§6.2.0.1 (Objectives of numerical integration) Two basic tasks can be identified in the field of **numerical integration** = approximate solution of initial value problems for ODEs (Please distinguish from “numerical quadrature”, see [Hip19, ??].):

- (I) Given initial time t_0 , final time T , and initial state \mathbf{y}_0 compute an approximation of $\mathbf{y}(T)$, where $t \mapsto \mathbf{y}(t)$ is the solution of (6.1.3.2). A corresponding function in C++ could look like

```
State solveivp(double t0, double T, State y0);
```

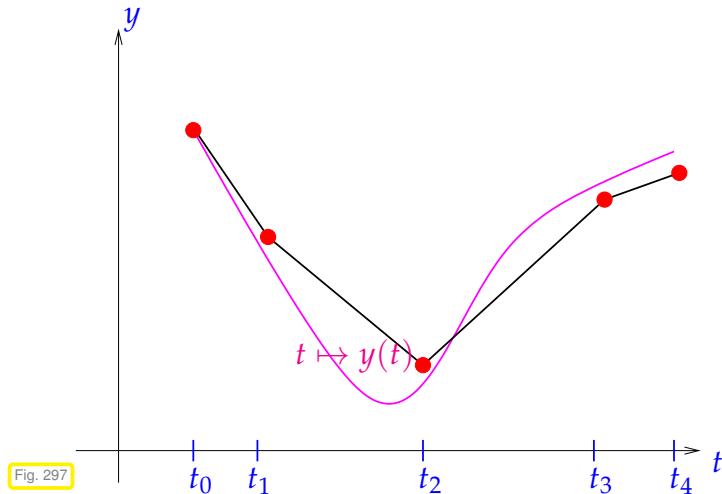
Here **State** is a type providing a fixed size or variable size vector $\in \mathbb{R}^N$, e.g.,

```
using State = Eigen::Matrix<double, N, 1>;
```

- (II) Output an *approximate* solution $t \rightarrow \mathbf{y}_h(t)$ of (6.1.3.2) on $[t_0, T]$ up to **final time** $T \neq t_0$ for “all times” $t \in [t_0, T]$ (in practice, of course, only for finitely many times $t_0 < t_1 < t_2 < \dots < t_{M-1} < t_M = T$, $M \in \mathbb{N}$, consecutively)

```
std::vector<State>  
solveivp(State y0, const std::vector<double> &tvec);
```

This is the “plot solution” task, because we need to know $\mathbf{y}(t)$ for many times, if we want to create a faithful plot of $t \mapsto \mathbf{y}(t)$. □



This section presents three methods that provide a **piecewise linear**, that is, “**polygonal**” approximation of solution trajectories $t \mapsto \mathbf{y}(t)$.

▷ A piecewise linear function, aka a **polygonal** curve, approximating a function $t \mapsto y(t) \in \mathbb{R}$ in grid points $t_0 < t_1 < \dots < t_4$.

§6.2.0.2 (Temporal mesh) As in Section 2.3 the polygonal approximation in this section will be based on a **(temporal) mesh** with $M+1$ mesh points (\rightarrow § 2.3.1.3)

$$\mathcal{M} := \{t_0 < t_1 < t_2 < \dots < t_{M-1} < t_M := T\} \subset [t_0, T], \quad (6.2.0.3)$$

covering the time interval of interest between **initial time** t_0 and **final time** $T > t_0$. We assume that the interval of interest is contained in the domain of definition of the solution of the IVP: $[t_0, T] \subset J(t_0, \mathbf{y}_0)$. □

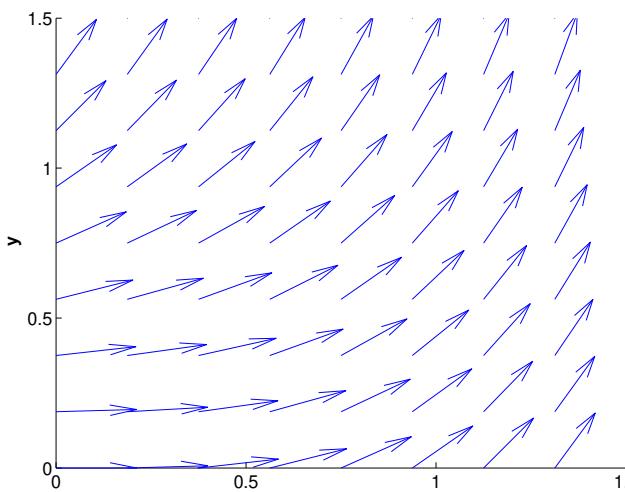
The next three sections will derive three simple mesh-based numerical integration methods, each in two ways,

- (i) based on geometric reasoning interpreting $\dot{\mathbf{y}}$ as the slope/direction of a tangent line,
- (ii) in the spirit of finite difference methods introduced in § 4.1.2.3, we can replace the derivative $\dot{\mathbf{y}}$ with a mesh-based difference quotient.

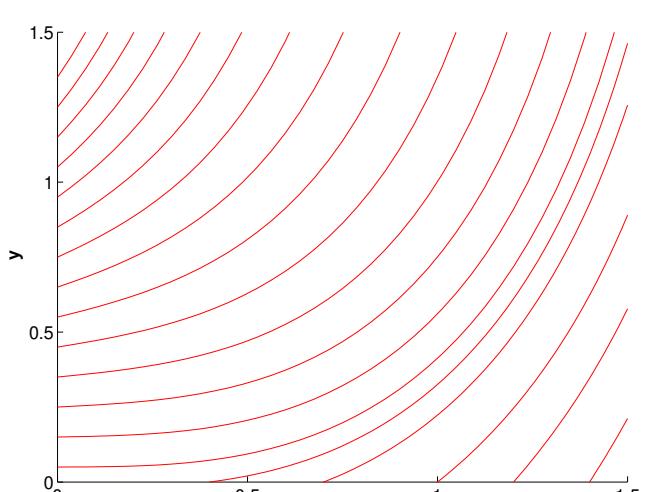
6.2.1 Explicit Euler method

EXAMPLE 6.2.1.1 (Tangent field and solution curves) For $N = 1$ polygonal methods can be constructed by geometric considerations in the $t - y$ plane, a model for the extended state space. We explain this for the **Riccati differential equation**, a scalar ODE:

$$\dot{y} = f(t, y) := y^2 + t^2 \quad \blacktriangleright \quad N = 1, \quad I, D = \mathbb{R}^+. \quad (6.2.1.2)$$

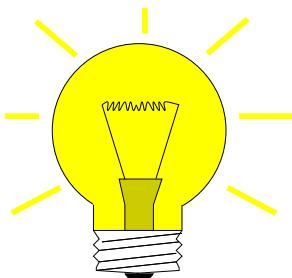


$$\text{tangent field } (t, y) \mapsto \frac{1}{\sqrt{f^2(t, y) + 1}} \begin{bmatrix} 1 \\ f(t, y) \end{bmatrix}$$



solution curves

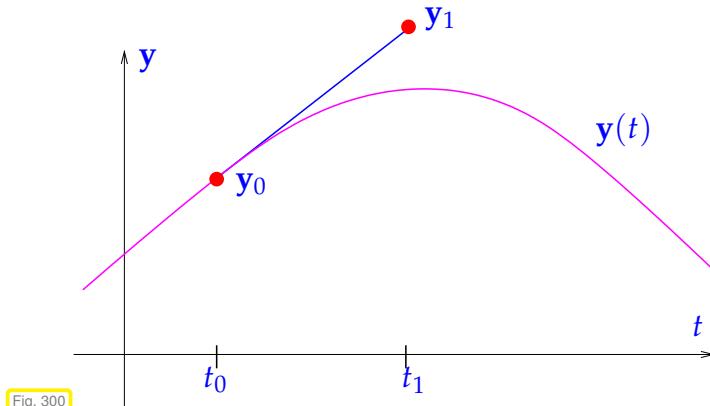
The solution curves run tangentially to the tangent field in each point of the extended state space. □



Idea:

“follow the tangents over short periods of time”

- ① **timestepping**: successive approximation of evolution on *mesh intervals* $[t_{k-1}, t_k]$, $k = 1, \dots, M$, $t_M := T$,
- ② approximation of solution on $[t_{k-1}, t_k]$ by **tangent** line to solution trajectory through (t_{k-1}, y_{k-1}) .



explicit Euler method (Euler 1768)

▷ First step of explicit Euler method ($N = 1$):

Slope of tangent $= f(t_0, y_0)$

y_1 serves as initial value for next step!

See also [Han02, Ch. 74], [DR08, Alg. 11.4]

EXAMPLE 6.2.1.3 (Visualization of explicit Euler method)

We use the temporal mesh

$$\mathcal{M} := \{t_j := j/5 : j = 0, \dots, 5\},$$

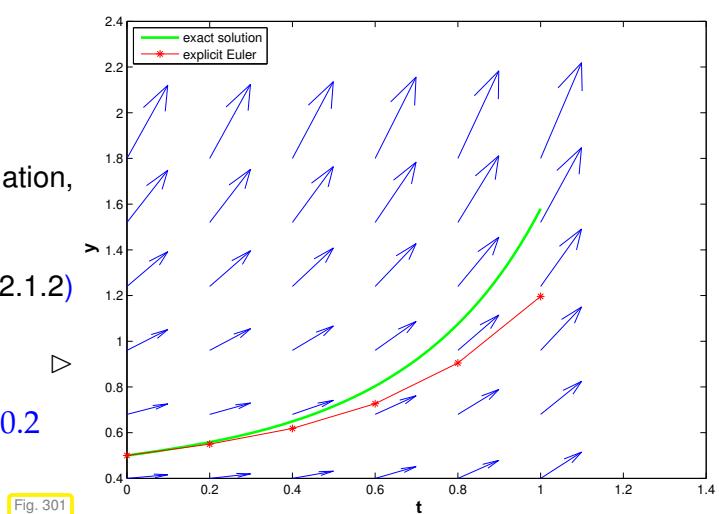
and solve an IVP for the Riccati differential equation,
see Ex. 6.2.1.1

$$\dot{y} = y^2 + t^2. \quad (6.2.1.2)$$

Here: $y_0 = \frac{1}{2}$, $t_0 = 0$, $T = 1$,

— ≈ “Euler polygon” for uniform timestep $h = 0.2$

→ ≈ tangent field of Riccati ODE



↓

Formula: When applied to a general IVP of the form (6.1.3.2) the explicit Euler method generates a sequence $(\mathbf{y}_k)_{k=0}^N$ by the recursion

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_k, \mathbf{y}_k), \quad k = 0, \dots, M-1, \quad (6.2.1.4)$$

with local (size of) timestep (stepsize) $h_k := t_{k+1} - t_k$.

Remark 6.2.1.5 (Explicit Euler method as a difference scheme)

One can obtain (6.2.1.4) by approximating the derivative $\frac{d}{dt}$ by a forward difference quotient on the (temporal) mesh $\mathcal{M} := \{t_0, t_1, \dots, t_M\}$:

$$\begin{aligned} \dot{\mathbf{y}}(t_k) &\approx \frac{\mathbf{y}(t_k + h_k) - \mathbf{y}(t_k)}{h_k} \\ \blacktriangleright \quad \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad \longleftrightarrow \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} &= \mathbf{f}(t_k, \mathbf{y}_h(t_k)), \quad k = 0, \dots, M-1. \end{aligned} \quad (6.2.1.6)$$

Difference schemes follow a simple policy for the *discretization* of differential equations: replace all derivatives by difference quotients connecting solution values on a set of discrete points (the mesh). ↓

Remark 6.2.1.7 (Output of explicit Euler method) To begin with, the explicit Euler recursion (6.2.1.4) produces a sequence $\mathbf{y}_0, \dots, \mathbf{y}_M$ of states. How does it deliver on the task (I) and (II) stated in § 6.2.0.1? By “geometric insight” we expect

$$\mathbf{y}_k \approx \mathbf{y}(t_k).$$

(As usual, we use the notation $t \mapsto \mathbf{y}(t)$ for the exact solution of an IVP.)

Task (I): Easy, because \mathbf{y}_M already provides an approximation of $\mathbf{y}(T)$.

Task (II): The trajectory $t \mapsto \mathbf{y}(t)$ is approximated by the piecewise linear function (“Euler polygon”)

$$\mathbf{y}_h : [t_0, t_N] \rightarrow \mathbb{R}^N, \quad \mathbf{y}_h(t) := \mathbf{y}_k \frac{t_{k+1} - t}{t_{k+1} - t_k} + \mathbf{y}_{k+1} \frac{t - t_k}{t_{k+1} - t_k} \quad \text{for } t \in [t_k, t_{k+1}], \quad (6.2.1.8)$$

see Fig. 301. This function can easily be sampled on any grid of $[t_0, t_M]$. In fact, it is the \mathcal{M} -piecewise linear interpolant of the data points (t_k, \mathbf{y}_k) , $k = 0, \dots, N$, see [Hip19, ??]).

The same considerations apply to the methods discussed in the next two sections and will not be repeated there. ↓

6.2.2 Implicit Euler method

Why forward difference quotient and not backward difference quotient? Let's try!

On (temporal) mesh $\mathcal{M} := \{t_0, t_1, \dots, t_M\}$ we obtain

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \iff \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f(t_{k+1}, \mathbf{y}_h(t_{k+1})) , \quad k = 0, \dots, M-1 . \quad (6.2.2.1)$$

backward difference quotient

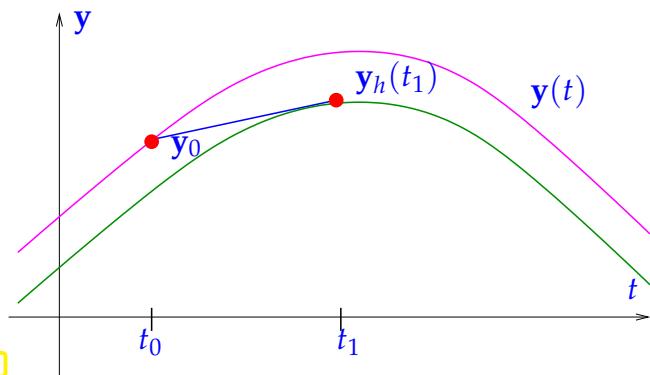
This leads to another simple timestepping scheme analogous to (6.2.1.4):

$$\boxed{\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) , \quad k = 0, \dots, M-1} , \quad (6.2.2.2)$$

with local timestep (stepsize) $h_k := t_{k+1} - t_k$.

(6.2.2.2) = implicit Euler method

Note: (6.2.2.2) requires solving a (possibly non-linear) system of equations to obtain \mathbf{y}_{k+1} !
 (► Terminology “implicit”)



Geometry of implicit Euler method:

Approximate solution through (t_0, y_0) on $[t_0, t_1]$ by

- straight line through (t_0, y_0)
 - with slope $f(t_1, y_1)$
- ▷ $\color{magenta}\hat{=}$ trajectory through (t_0, y_0) ,
 $\color{green}\hat{=}$ trajectory through (t_1, y_1) ,
 $\color{blue}\hat{=}$ tangent at $\color{green}\hat{=}$ in (t_1, y_1) .

Remark 6.2.2.3 (Feasibility of implicit Euler timestepping) The issue is whether (6.2.2.2) well defined, that is, whether we can solve it for \mathbf{y}_{k+1} and whether this solution unique.

Intuition: For small timesteps $h > 0$ the right hand side of (6.2.2.2) is a “small perturbation of the identity”.

Formally: Consider an autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, assume a continuously differentiable right hand side function $\mathbf{f}, \mathbf{f} \in C^1(D, \mathbb{R}^N)$, and regard (6.2.2.2) as an h -dependent non-linear system of equations:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) \iff G(h, \mathbf{y}_{k+1}) = 0 \quad \text{with} \quad G(h, \mathbf{z}) := \mathbf{z} - h \mathbf{f}(t_{k+1}, \mathbf{z}) - \mathbf{y}_k .$$

To investigate the solvability of this non-linear equation we start with an observation about a partial derivative of G :

$$\frac{dG}{d\mathbf{z}}(h, \mathbf{z}) = \mathbf{I} - h D_{\mathbf{y}} \mathbf{f}(t_{k+1}, \mathbf{z}) \Rightarrow \frac{dG}{d\mathbf{z}}(0, \mathbf{z}) = \mathbf{I} .$$

In addition, $G(0, \mathbf{y}_k) = \mathbf{0}$. Next, recall the implicit function theorem [Str09, Thm. 7.8.1]:

Theorem 6.2.2.4. Implicit function theorem

Let $G = G(\mathbf{x}, \mathbf{y})$ a continuously differentiable function of $\mathbf{x} \in \mathbb{R}^k$ and $\mathbf{y} \in \mathbb{R}^\ell$, defined on the open set $\Omega \subset \mathbb{R}^k \times \mathbb{R}^\ell$ with values in \mathbb{R}^ℓ : $G : \Omega \subset \mathbb{R}^k \times \mathbb{R}^\ell \rightarrow \mathbb{R}^\ell$.

Assume that G has a zero in $\mathbf{z}_0 := \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{y}_0 \end{bmatrix} \in \Omega$, $\mathbf{x}_0 \in \mathbb{R}^k$, $\mathbf{y}_0 \in \mathbb{R}^\ell$: $G(\mathbf{z}_0) = 0$.

If the Jacobian $\frac{\partial G}{\partial \mathbf{y}}(\mathbf{p}_0) \in \mathbb{R}^{\ell, \ell}$ is invertible, then there is an open neighborhood U of $\mathbf{x}_0 \in \mathbb{R}^k$ and a continuously differentiable function $\mathbf{g} : U \rightarrow \mathbb{R}^\ell$ such that

$$\mathbf{g}(\mathbf{x}_0) = \mathbf{y}_0 \quad \text{and} \quad G(\mathbf{x}, \mathbf{g}(\mathbf{x})) = 0 \quad \forall \mathbf{x} \in U.$$

For sufficiently small $|h|$ it permits us to conclude that the equation $G(h, \mathbf{z}) = 0$ defines a continuous function $\mathbf{g} = \mathbf{g}(h)$ with $\mathbf{g}(0) = \mathbf{y}_k$.

➤ for sufficiently small $h > 0$ the equation (6.2.2.2) has a unique solution \mathbf{y}_{k+1} . \square

6.2.3 Implicit midpoint method

Beside using forward or backward difference quotients, the derivative $\dot{\mathbf{y}}$ can also be approximated by the symmetric difference quotient, see also [Hip19, ??],

$$\dot{\mathbf{y}}(t) \approx \frac{\mathbf{y}(t+h) - \mathbf{y}(t-h)}{2h}, \quad h > 0. \quad (6.2.3.1)$$

The idea is to apply this formula in $t = \frac{1}{2}(t_k + t_{k+1})$ with $h = h_k/2$, which transforms the ODE into

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \iff \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_k} = f\left(\frac{1}{2}(t_k + t_{k+1}), \mathbf{y}_h\left(\frac{1}{2}(t_k + t_{k+1})\right)\right), \quad k = 0, \dots, M-1. \quad (6.2.3.2)$$

The trouble is that the value $\mathbf{y}_h\left(\frac{1}{2}(t_k + t_{k+1})\right)$ does not seem to be available, unless we recall that the approximate trajectory $t \mapsto \mathbf{y}_h(t)$ is supposed to be piecewise linear, which implies $\mathbf{y}_h\left(\frac{1}{2}(t_k + t_{k+1})\right) = \frac{1}{2}(\mathbf{y}_h(t_k) + \mathbf{y}_h(t_{k+1}))$. This gives the recursion formula for the implicit midpoint method in analogy to (6.2.1.4) and (6.2.2.2):

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}\left(\frac{1}{2}(t_k + t_{k+1}), \frac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1})\right), \quad k = 0, \dots, N-1, \quad (6.2.3.3)$$

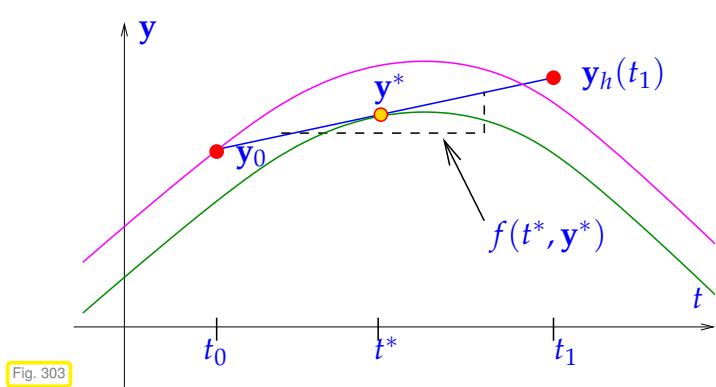
with local timestep (stepsize) $h_k := t_{k+1} - t_k$.

Implicit midpoint method, a geometric view:

Approximate trajectory through (t_0, \mathbf{y}_0) on $[t_0, t_1]$ by

- straight line through (t_0, \mathbf{y}_0)
- with slope $f(t^*, \mathbf{y}^*)$, where $t^* := \frac{1}{2}(t_0 + t_1)$, $\mathbf{y}^* = \frac{1}{2}(\mathbf{y}_0 + \mathbf{y}_1)$

▷ — $\hat{=}$ trajectory through (t_0, \mathbf{y}_0) ,
— $\hat{=}$ trajectory through (t^*, \mathbf{y}^*) ,
— $\hat{=}$ tangent at — in (t^*, \mathbf{y}^*) .



As in the case of (6.2.2.2), also (6.2.3.3) entails solving a (non-linear) system of equations in order to obtain \mathbf{y}_{k+1} . Rem. 6.2.2.3 also holds true in this case: for sufficiently small h (6.2.3.3) will have a unique solution \mathbf{y}_{k+1} , which renders the recursion well defined.

Review question(s) 6.2.3.4 (Polygonal approximation methods)

(Q6.2.3.4.A) We consider the scalar linear IVP

$$\dot{y} = \lambda y , \quad y(0) = 1$$

on the interval $[0, 1]$. We use $M \in \mathbb{N}$ equidistant steps of the explicit Euler method to compute an approximation y_M for $y(1)$.

- Derive a formula for y_M .
- Which known result from calculus is equivalent to the convergence $y_M \rightarrow y(1)$ for $M \rightarrow \infty$?

(Q6.2.3.4.B) For an ODE

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) , \quad \mathbf{f} := \begin{bmatrix} f_1(\mathbf{y}) \\ \vdots \\ f_N(\mathbf{y}) \end{bmatrix} : D \subset \mathbb{R}^N \mapsto \mathbb{R}^N , \quad (*)$$

we know that

$$\sum_{\ell=1}^N f_\ell(\mathbf{y}) = 0 \quad \forall \mathbf{y} \in D .$$

- Show that the sum of the components of every solution $t \mapsto \mathbf{y}(t)$ is constant in time.
- Show that the sums of the components of the vectors $\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \dots$ generated by either the explicit Euler method, the implicit Euler method, or the implicit midpoint method, all applied to solve some IVP for (*), are the same for all vectors \mathbf{y}_k .

(Q6.2.3.4.C) We consider the **implicit Euler method** for the scalar autonomous “explosion ODE” $\dot{y} = y^2$. Given an **explicit** formula for y_{k+1} in terms of y_k and the timestep size $h_k > 0$. Specify potentially necessary constraints on the size of h_k .

The defining equation for recursion of the implicit Euler method (on some temporal mesh) applied to the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ is

$$\mathbf{y}_{k+1} : \quad \mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) . \quad (6.2.2.2)$$

(Q6.2.3.4.D) The recursion of the **implicit midpoint rule** for the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ is

$$\mathbf{y}_{k+1} : \quad \mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}\left(\frac{1}{2}(t_k + t_{k+1}), \frac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1})\right) .$$

Give an **explicit** form of this recursion for the linear ODE $\dot{\mathbf{y}} = \mathbf{A}(t)\mathbf{y}$, where $\mathbf{A} : \mathbb{R} \rightarrow \mathbb{R}^{N,N}$ is a matrix-valued function. When will this recursion break down?

(Q6.2.3.4.E) For a twice continuously differentiable function $f : I \subset \mathbb{R} \rightarrow \mathbb{R}^N$ we can use the **second symmetric difference quotient** as an approximation of the second derivative $f''(x)$, $x \in I$:

$$\frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \approx f''(x) \quad \text{for } |h| \ll 1 .$$

Based on this approximation propose an explicit finite-difference timstepping scheme on a uniform temporal mesh for the second-order ODE $\ddot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

△

6.3 General Single-Step Methods



Video tutorial for Section 6.3: General Single-Step Methods: (31 minutes) [Download link](#), [tablet notes](#)

Now we fit the numerical schemes introduced in the previous section into a more general class of methods for the solution of (autonomous) initial value problems (6.1.3.17) for ODEs. Throughout we assume that all times considered belong to the domain of definition of the unique solution $t \rightarrow \mathbf{y}(t)$ of (6.1.3.17), that is, for $T > 0$ we take for granted $[0, T] \subset J(\mathbf{y}_0)$ (temporal domain of definition of the solution of an IVP is explained in § 6.1.3.18).

6.3.1 Definition

§6.3.1.1 (Discrete evolution operators) From Section 6.2.1 and Section 6.2.2 recall the two Euler methods for an autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$:

$$\begin{aligned} \text{explicit Euler: } \mathbf{y}_{k+1} &= \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k), \\ \text{implicit Euler: } \mathbf{y}_{k+1} &:= \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_{k+1}), \quad h_k := t_{k+1} - t_k. \end{aligned}$$

Both formulas, for sufficiently small h_k (→ Rem. 6.2.2.3), provide a mapping

$$(\mathbf{y}_k, h_k) \mapsto \Psi(h, \mathbf{y}_k) := \mathbf{y}_{k+1}. \quad (6.3.1.2)$$

If \mathbf{y}_0 is the initial value, then $\mathbf{y}_1 := \Psi(h, \mathbf{y}_0)$ can be regarded as an approximation of $\mathbf{y}(h)$, the value returned by the evolution operator Φ (→ Def. 6.1.4.3) for $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ applied to \mathbf{y}_0 over the period h . $\mathbf{y}(t_k)$:

$$\mathbf{y}_1 = \Psi(h, \mathbf{y}_0) \iff \mathbf{y}(h) = \Phi^h \mathbf{y}_0 \Rightarrow \boxed{\Psi(h, \mathbf{y}) \approx \Phi^h \mathbf{y}}, \quad (6.3.1.3)$$

In a sense the polygonal approximation methods are based on approximations for the evolution operator associated with the ODE.

This is what every single step method does: it tries to approximate the evolution operator Φ for an ODE by a mapping Ψ of the kind as described in (6.3.1.2).

→ A mapping Ψ as in (6.3.1.2) is called (a) **discrete evolution** operator.

☞ Notation: In analogy to Φ^h for discrete evolutions we often write $\Psi^h \mathbf{y} := \Psi(h, \mathbf{y})$ ↴

Above we identified the discrete evolutions underlying the polygonal approximation methods. Vice versa, a mapping Ψ as given in (6.3.1.2) defines a single step method.

Definition 6.3.1.4. Single step method (for autonomous ODE) → [QSS00, Def. 11.2]

Given a **discrete evolution** $\Psi : \Omega \subset \mathbb{R} \times D \mapsto \mathbb{R}^N$, an initial state \mathbf{y}_0 , and a temporal mesh $\mathcal{M} := \{0 =: t_0 < t_1 < \dots < t_M := T\}$, $M \in \mathbb{N}$, the recursion

$$\mathbf{y}_{k+1} := \Psi(t_{k+1} - t_k, \mathbf{y}_k), \quad k = 0, \dots, M-1, \quad (6.3.1.5)$$

defines a **single-step method** (SSM) for the autonomous IVP $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$ on the interval $[0, T]$.

- In a sense, a single step method defined through its associated discrete evolution does not approximate a concrete initial value problem, but tries to approximate an ODE in the form of its evolution operator.

In C++ a discrete evolution operator can be incarnated by a functor type offering an evaluation operator

```
State operator () (double h, const State &y) const;
```

see § 6.2.0.1 for the **State** data type.

Remark 6.3.1.6 (Discrete evolutions for non-autonomous ODEs) The concept of single step method according to Def. 6.3.1.4 can be generalized to non-autonomous ODEs, which leads to recursions of the form:

$$\mathbf{y}_{k+1} := \Psi(t_k, t_{k+1}, \mathbf{y}_k), \quad k = 0, \dots, M-1,$$

for a discrete evolution operator Ψ defined on $I \times I \times D$. □

§6.3.1.7 (Consistent single step methods) Now we state a first quantification of the goal that the “discrete evolution should be an approximation of the evolution operator”: $\Psi \approx \Phi$, cf. (6.3.1.3). We want the discrete evolution Ψ to inherit key properties of the evolution operator Φ . One such property is

$$\frac{d}{dt} \Phi^t \mathbf{y} \Big|_{t=0} = \mathbf{f}(\mathbf{y}) \quad \forall \mathbf{y} \in D. \quad (6.3.1.8)$$

Compliance of Ψ with (6.3.1.8) is expressed through the property of **consistency**, which, roughly speaking, demands that a viable discrete evolution operator methods is structurally similar to that for the explicit Euler method (6.2.1.4).

Consistent discrete evolution

The discrete evolution Ψ defining a single step method according to Def. 6.3.1.4 and (6.3.1.5) for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ must be of the form

$$\Psi^h \mathbf{y} = \mathbf{y} + h \psi(h, \mathbf{y}) \quad \text{with} \quad \begin{aligned} \psi : I \times D &\rightarrow \mathbb{R}^N \text{ continuous,} \\ \psi(0, \mathbf{y}) &= \mathbf{f}(\mathbf{y}). \end{aligned} \quad (6.3.1.10)$$

Differentiating $(h \mapsto \Psi^h \mathbf{y})$ relying on the product rule confirms that (6.3.1.8) remains true for Ψ instead of Φ .

Definition 6.3.1.11. Consistent single step methods

A single step method according to Def. 6.3.1.4 based on a discrete evolution of the form (6.3.1.10) is called **consistent** with the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

EXAMPLE 6.3.1.12 (Consistency of implicit midpoint method) The discrete evolution Ψ and, hence, the function $\psi = \psi(h, \mathbf{y})$ for the implicit midpoint method are defined only implicitly, of course. Thus,

consistency cannot immediately be seen from a formula for ψ .

We examine consistency of the implicit midpoint method for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$. A single step is defined by

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}\left(\frac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1})\right), \quad k = 0, \dots, M-1. \quad (6.3.1.13)$$

Assume that

- the right hand side function $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$ is smooth, at least $\mathbf{f} \in C^1(D)$,
- and that $|h|$ is sufficiently small to guarantee the existence of a solution \mathbf{y}_{k+1} of (6.3.1.13), see Rem. 6.2.2.3.

Then we infer from the **implicit function theorem** Thm. 6.2.2.4 that the solution \mathbf{y}_{k+1} of (6.3.1.13) will depend on h in a continuously differentiable way: $h \mapsto \mathbf{y}_{k+1}(h) \in C^1([-h, h], \mathbb{R}^N)$ for small $\delta > 0$. Knowing this, we plug (6.3.1.13) into itself and obtain

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}\left(\frac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1})\right) \stackrel{(6.3.1.13)}{=} \mathbf{y}_k + h\underbrace{\mathbf{f}\left(\mathbf{y}_k + \frac{1}{2}h\mathbf{f}\left(\frac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1})\right)\right)}_{=\psi(h, \mathbf{y}_k)}. \quad \text{---}$$

We repeat that by the implicit function theorem Thm. 6.2.2.4 \mathbf{y}_{k+1} depends continuously on h and \mathbf{y}_k . This means that $\psi(h, \mathbf{y}_k)$ has the desired properties, in particular $\psi(0, \mathbf{y}) = \mathbf{f}(\mathbf{y})$ is clear. ---

Remark 6.3.1.14 (Notation for single step methods) Many authors specify a single step method by writing down the first step for a general stepsize h

$$\mathbf{y}_1 = (\text{implicit}) \text{ expression in } \mathbf{y}_0, h \text{ and } \mathbf{f},$$

for instance, for the implicit midpoint rule

$$\mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{f}\left(\frac{1}{2}(\mathbf{y}_0 + \mathbf{y}_1)\right).$$

Actually, this fixes the underlying discrete evolution. Also this course will sometimes adopt this practice. ---

§6.3.1.15 (Output of single step methods) Here we resume and continue the discussion of Rem. 6.2.1.7 for general single step methods according to Def. 6.3.1.4. Assuming unique solvability of the systems of equations faced in each step of an **implicit** method, every single step method based on a mesh $\mathcal{M} = \{0 = t_0 < t_1 < \dots < t_M := T\}$ produces a finite sequence $(\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_M)$ of states, where the first agrees with the initial state \mathbf{y}_0 .

We expect that the states provide a pointwise approximation of the solution trajectory $t \rightarrow \mathbf{y}(t)$:

$$\mathbf{y}_k \approx \mathbf{y}(t_k), \quad k = 1, \dots, M.$$

Thus task (I) from § 6.2.0.1, computing an approximation for $\mathbf{y}(T)$, is again easy: output \mathbf{y}_M as an approximation of $\mathbf{y}(T)$.

Task (II) from § 6.2.0.1, computing the solution trajectory, requires **interpolation** of the data points (t_k, \mathbf{y}_k) using some of the techniques presented in [Hip19, ??]. The natural option is \mathcal{M} -piecewise polynomial interpolation, generalizing the polygonal approximation [Hip19, ??] used in Section 6.2.

Note that from the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ the derivatives $\dot{\mathbf{y}}_h(t_k) = \mathbf{f}(\mathbf{y}_k)$ are available without any further approximation. This facilitates **cubic Hermite interpolation** (\rightarrow [Hip19, ??]), which yields

$$\mathbf{y}_h \in C^1([0, T]): \quad \mathbf{y}_h|_{[x_{k-1}, x_k]} \in \mathcal{P}_3, \quad \mathbf{y}_h(t_k) = \mathbf{y}_k, \quad \frac{d\mathbf{y}_h}{dt}(t_k) = \mathbf{f}(\mathbf{y}_k).$$

Summing up, an approximate trajectory $t \mapsto \mathbf{y}_h(t)$ is built in two stages:

- (i) Compute sequence $(\mathbf{y}_k)_k$ by running the single step method.
(ii) Post-process the obtained sequence, usually by applying interpolation, to get \mathbf{y}_h .

]

Review question(s) 6.3.1.16 (General single-step methods)

(Q6.3.1.16.A) Explain the concepts

- evolution operator and
- discrete evolution operator

in connection with the numerical integration of initial-value problems for the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{f} : D \subset \mathbb{R}^N \mapsto \mathbb{R}^N$.

(Q6.3.1.16.B) [Single-step methods and numerical quadrature] There is a connection between numerical integration (the design and analysis of numerical methods for the solution of initial-value problems for ODEs) and numerical quadrature (study of numerical methods for the evaluation of integrals).

- Explain, how a class of single-step methods for the solution of scalar initial-value problems

$$\dot{y} = f(t, y) , \quad y(t_0) = y_0 \in \mathbb{R} ,$$

can be used for the approximate evaluation of integrals $\int_a^b \varphi(\tau) d\tau$, $\varphi : [a, b] \rightarrow \mathbb{R}$.

- If the considered single-step methods are of order p , what does this mean for the induced quadrature method.
- Which quadrature formula does the implicit midpoint method yield?

(Q6.3.1.16.C) [Adjoint single-step method] Let a single-step method for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$ be defined by its discrete evolution operator $\Psi : I \times D \mapsto D$. Then the **adjoint single-step method** is spawned by the discrete evolution operator $\tilde{\Psi} : I \times D \mapsto D$ defined according to

$$\tilde{\Psi}^h \mathbf{y} := (\Psi^{-h})^{-1} , \quad \mathbf{y} \in D, h \in \mathbb{R} \text{ sufficiently small .}$$

What is the adjoint of the explicit Euler method?

(Q6.3.1.16.D) We have seen three simple single-step methods for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$, here defined by describing a the first step $\mathbf{y}_0 \rightarrow \mathbf{y}_1$ with stepsize $h \in \mathbb{R}$ ("sufficiently small"):

- The explicit Euler method:

$$\mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{f}(\mathbf{y}_0) .$$

- The implicit Euler method:

$$\mathbf{y}_1 : \quad \mathbf{y}_1 = \mathbf{y}_0 = h\mathbf{f}(\mathbf{y}_1) .$$

- The implicit midpoint method:

$$\mathbf{y}_1 : \quad \mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{f}\left(\frac{1}{2}(\mathbf{y}_0 + \mathbf{y}_1)\right) .$$

For which methods does the associated discrete evolution operator $\Psi : [-\delta, \delta] \times D \rightarrow D$, $\delta > 0$ sufficiently small, satisfy

$$\Psi^h \circ \Psi^{-h} = \text{Id} \quad \forall h \in [0, \delta] ? \quad (6.3.1.17)$$

Try to find a simple (scalar) counterexample, if you think that a method does not have property (6.3.1.17).

△

6.3.2 (Asymptotic) Convergence of Single-Step Methods



Video tutorial for Section 6.3.2:(Asymptotic) Convergence of Single-Step Methods: (39 minutes) [Download link](#), [tablet notes](#)

Of course, the accuracy of the solution sequence $(\mathbf{y}_k)_k$ obtained by a particular single-step method (→ Def. 6.3.1.4) is a central concern. This motivates studying the dependence of suitable norms of the so-called discretization error on the choice of temporal mesh \mathcal{M} .

§6.3.2.1 (Discretization error of single step methods) Approximation errors in numerical integration are also called **discretization errors**, cf. Section 3.1.2.

Depending on the objective of numerical integration as stated in § 6.2.0.1 different (norms of) discretization errors are of interest:

- (I) If only the solution at final time T is sought, the relevant norm of the discretization error is

$$\epsilon_M := \|\mathbf{y}(T) - \mathbf{y}_M\|,$$

where $\|\cdot\|$ is some vector norm on \mathbb{R}^N .

- (II) If we want to approximate the solution trajectory for (6.1.3.17) the discretization error is the function

$$t \mapsto \mathbf{e}(t) \quad , \quad \mathbf{e}(t) := \mathbf{y}(t) - \mathbf{y}_h(t) \quad ,$$

where $t \mapsto \mathbf{y}_h(t)$ is the approximate trajectory obtained by post-processing, see § 6.3.1.15. In this case accuracy of the method is gauged by looking at norms of the function \mathbf{e} , see [Hip19, ??] for examples.

- (III) Between (I) and (II) is the pointwise discretization error, which is the sequence (**grid function**)

$$\mathbf{e} : \mathcal{M} \rightarrow D \quad , \quad \mathbf{e}_k := \mathbf{y}(t_k) - \mathbf{y}_k \quad , \quad k = 0, \dots, M \quad . \quad (6.3.2.2)$$

In this case one usually examines the maximum error in the mesh points

$$\|(\mathbf{e})\|_\infty := \max_{k \in \{1, \dots, N\}} \|\mathbf{e}_k\| \quad ,$$

where $\|\cdot\|$ is a suitable vector norm on \mathbb{R}^N , customarily the Euclidean vector norm.

□

§6.3.2.3 (Asymptotic convergence of single step methods) Once the discrete evolution Ψ associated with the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ is specified, the single step method according to Def. 6.3.1.4 is fixed:

$$\mathbf{y}_{k+1} := \Psi(t_{k+1} - t_k, \mathbf{y}_k) \quad , \quad k = 0, \dots, M - 1 \quad , \quad (6.3.1.5)$$

The only way to control the accuracy of the solution \mathbf{y}_N or $t \mapsto \mathbf{y}_h(t)$ is through the selection of the mesh $\mathcal{M} = \{0 = t_0 < t_1 < \dots < t_N = T\}$.

Hence we study convergence of single step methods for **families of meshes** $\{\mathcal{M}_\ell\}$ and track the decay of (a norm) of the discretization error ($\rightarrow \S\ 6.3.2.1$) as a function of the number $N := \#\mathcal{M}$ of mesh points. In other words, we examine **h -convergence**. Convergence through mesh refinement is discussed for piecewise polynomial interpolation in [Hip19, ??], for composite numerical quadrature in [Hip19, ??], and was studied for finite-element methods in Section 3.2.

When investigating asymptotic convergence of single step methods we often resort to families of **equidistant** meshes of $[0, T]$:

$$\mathcal{M}_N := \{t_k := \frac{k}{M}T : k = 0, \dots, M\}. \quad (6.3.2.4)$$

We also call this the use of **uniform** timesteps of size $h := \frac{T}{N}$. □

EXPERIMENT 6.3.2.5 (Speed of convergence of polygonal methods)

The setting for this experiment is as follows:

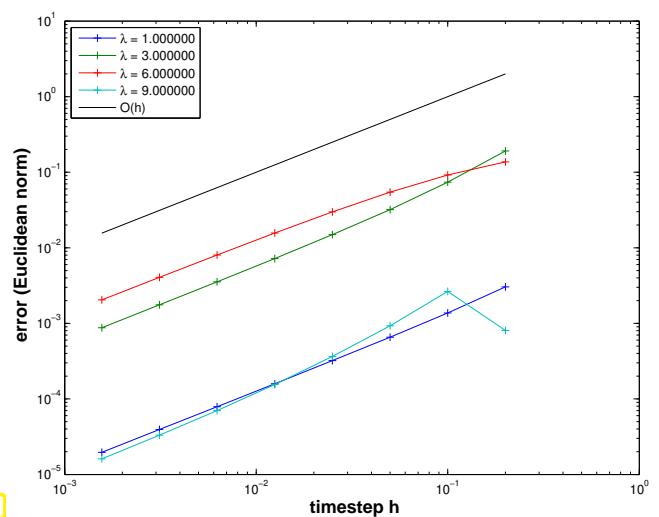
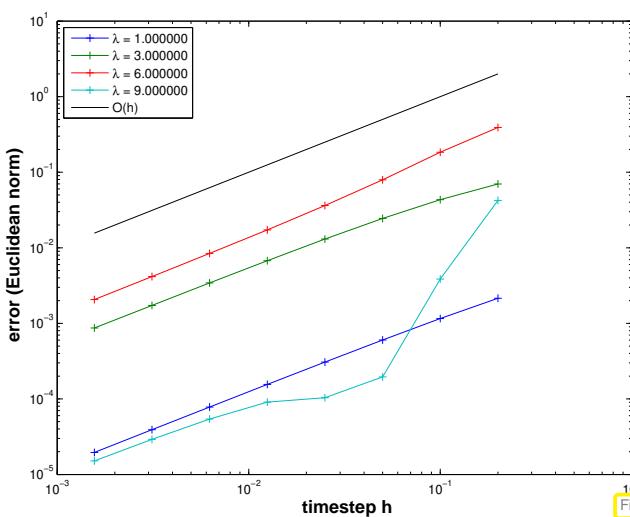
- ◆ We consider the following IVP for the logistic ODE, see Ex. 6.1.2.1

$$\dot{y} = \lambda y(1 - y), \quad y(0) = 0.01.$$

- ◆ We apply explicit and implicit Euler methods (6.2.1.4)/(6.2.2.2) with uniform timestep $h = 1/M$, $M \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.
- ◆ Monitored: Error at final time $E(h) := |y(1) - y_M|$

We are mainly interested in the qualitative nature of the asymptotic convergence as $h \rightarrow 0$ in the sense of the types of convergence introduced in Def. 3.2.2.1 with N replaced with h^{-1} . Abbreviating some error norm with $T = T(h)$, recall the classification of asymptotic convergence from Def. 3.2.2.1:

$$\begin{aligned} \exists p > 0: \quad T(h) \leq h^p &: \text{algebraic convergence, with order/rate } p > 0, \quad \forall h > 0; \\ \exists 0 < q < 1: \quad T(h) \leq q^{1/h} &: \text{exponential convergence,} \end{aligned}$$



$\blacktriangleright O(M^{-1}) = O(h)$ algebraic convergence with order/rate 1 in both cases for $h \rightarrow 0$

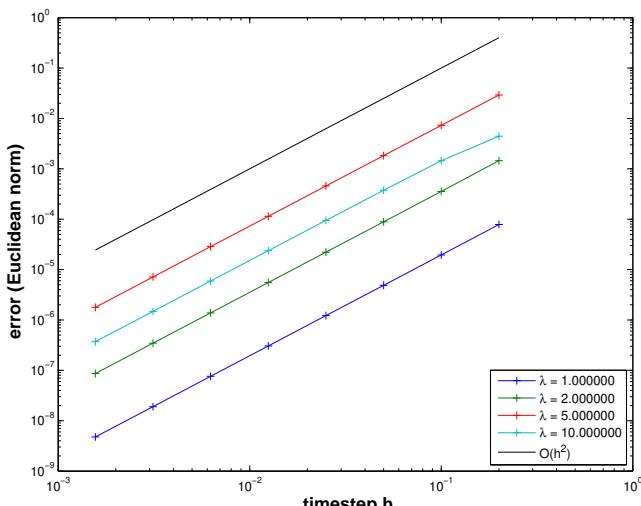


Fig. 306

implicit midpoint method

Parlance: based on the observed rate of algebraic convergence, the two Euler methods are said to “converge with first order”, whereas the implicit midpoint method is called “second-order convergent”. □

The observations made for polygonal timestepping methods reflect a general pattern:

Algebraic convergence of single step methods

Consider the numerical integration of an initial value problem

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) , \quad \mathbf{y}(t_0) = \mathbf{y}_0 , \quad (6.1.3.2)$$

with sufficiently smooth right hand side function $\mathbf{f} : I \times D \rightarrow \mathbb{R}^N$.

Then conventional single step methods (→ Def. 6.3.1.4) will enjoy *asymptotic algebraic convergence in the meshwidth*, more precisely, see [DR08, Thm. 11.25],

there is a $p \in \mathbb{N}$ such that the sequence $(\mathbf{y}_k)_k$ generated by the single step method for $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ on a mesh $\mathcal{M} := \{t_0 < t_1 < \dots < t_M = T\}$ satisfies

$$\max_k \|\mathbf{y}_k - \mathbf{y}(t_k)\| \leq Ch^p \quad \text{for } h := \max_{k=1,\dots,M} |t_k - t_{k-1}| \rightarrow 0 , \quad (6.3.2.7)$$

with $C > 0$ independent of \mathcal{M}

Definition 6.3.2.8. Order of a single step method

The maximal integer $p \in \mathbb{N}$ for which (6.3.2.7) holds for a single step method when applied to an ODE with (sufficiently) smooth right hand side, is called the **order** of the method.

As in the case of quadrature rules (→ [Hip19, ??]) their order is the principal intrinsic indicator for the “quality” of a single step method.

§6.3.2.9 (Convergence analysis for the explicit Euler method [Han02, Ch. 74]) We consider the simplest single-step method, namely the explicit Euler method (6.2.1.4) on a mesh $\mathcal{M} := \{0 = t_0 < t_1 < \dots < t_M = T\}$ for a generic autonomous IVP

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) , \quad \mathbf{y}(0) = \mathbf{y}_0 \in D ,$$

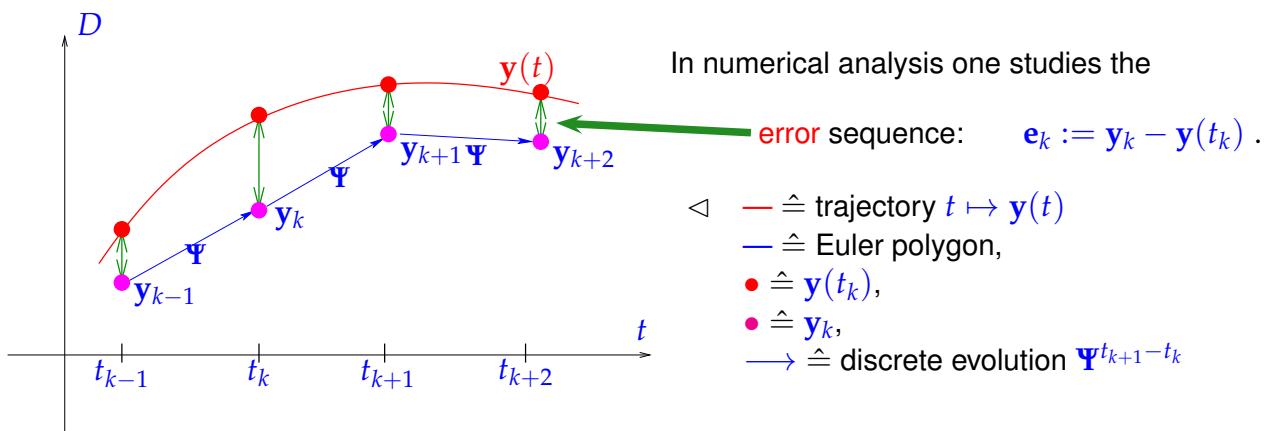
with sufficiently smooth and (*globally*) Lipschitz continuous $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$, that is,

$$\exists L > 0: \quad \|\mathbf{f}(\mathbf{y}) - \mathbf{f}(\mathbf{z})\| \leq L \|\mathbf{y} - \mathbf{z}\| \quad \forall \mathbf{y}, \mathbf{z} \in D , \quad (6.3.2.10)$$

cf. Def. 6.1.3.12, and \mathcal{C}^1 exact solution $t \mapsto \mathbf{y}(t)$. Throughout we assume that solutions of $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ are defined on $[0, T]$ for all initial states $\mathbf{y}_0 \in D$.

Recall the recursion defining the explicit Euler method

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k) , \quad k = 1, \dots, M-1 . \quad (6.2.1.4)$$



The approach to estimate $\|\mathbf{e}_k\|$ follows a fundamental policy that comprises three key steps. To explain them we rely on the abstract concepts of the

- evolution operator Φ associated with the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ (→ Def. 6.1.4.3) and
- discrete evolution operator Ψ defining the explicit Euler single step method, see Def. 6.3.1.4:

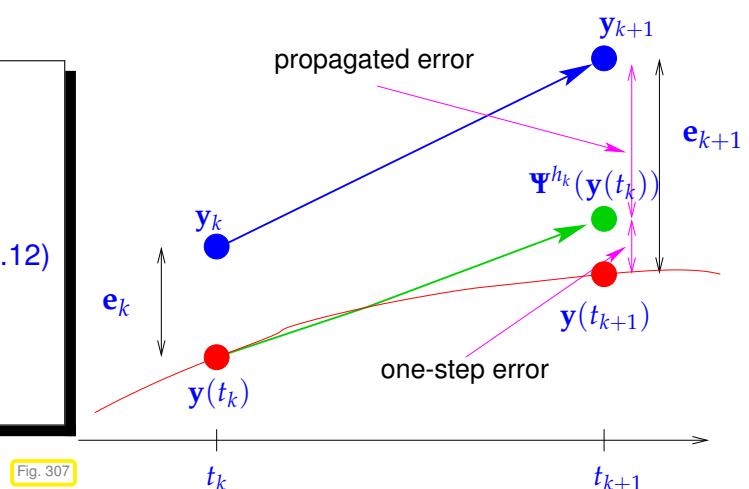
$$(6.2.1.4) \quad \Rightarrow \quad \Psi^h \mathbf{y} = \mathbf{y} + h \mathbf{f}(\mathbf{y}) . \quad (6.3.2.11)$$

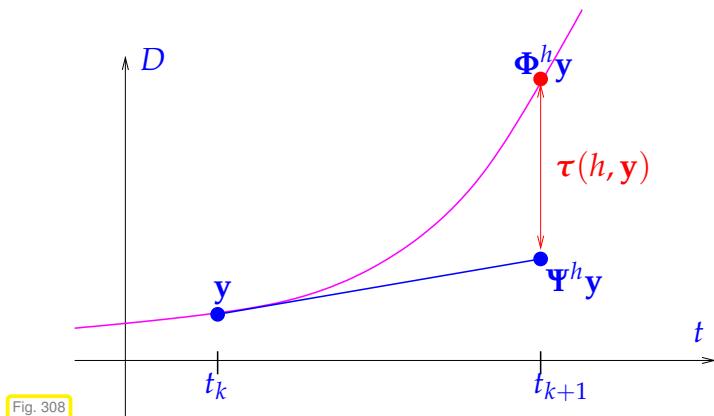
We argue that in this context abstraction pays off, because it helps elucidate a general technique for the convergence analysis of single step methods.

① Abstract splitting of error:

Fundamental error splitting:

$$\begin{aligned} \mathbf{e}_{k+1} &= \Psi^{h_k} \mathbf{y}_k - \Phi^{h_k} \mathbf{y}(t_k) \\ &= \underbrace{\Psi^{h_k} \mathbf{y}_k - \Psi^{h_k} \mathbf{y}(t_k)}_{\text{propagated error}} + \underbrace{\Psi^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k)}_{\text{one-step error}} . \end{aligned} \quad (6.3.2.12)$$





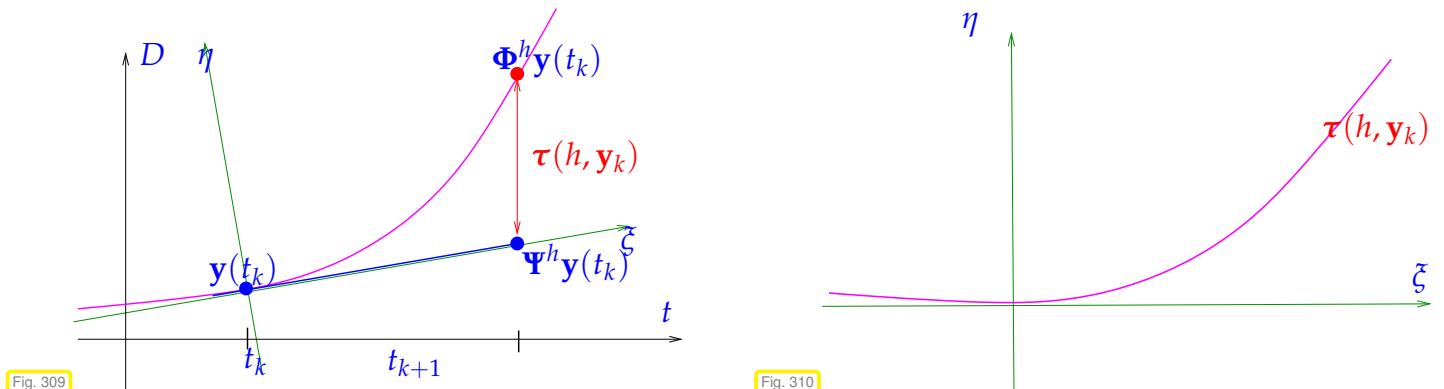
A generic **one-step error** expressed through continuous and discrete evolutions reads:

$$\tau(h, \mathbf{y}) := \Psi^h \mathbf{y} - \Phi^h \mathbf{y} . \quad (6.3.2.13)$$

▷ geometric visualisation of one-step error for explicit Euler method (6.2.1.4), cf. Fig. 300, $h := t_{k+1} - t_k$
—: solution trajectory through (t_k, \mathbf{y})

② Estimate for one-step error $\tau(h_k, \mathbf{y}(t_k))$:

Geometric considerations: distance of a smooth curve and its tangent shrinks as the square of the distance to the intersection point (curve locally looks like a parabola in the $\xi - \eta$ coordinate system, see Fig. 311).



The geometric considerations can be made rigorous by analysis: recall Taylor's formula for the function $\mathbf{y} \in C^{K+1}$ [Str09, Satz 5.5.1]:

$$\begin{aligned} \mathbf{y}(t+h) - \mathbf{y}(t) &= \sum_{j=1}^K \mathbf{y}^{(j)}(t) \frac{h^j}{j!} + \underbrace{\int_t^{t+h} \mathbf{y}^{(K+1)}(\tau) \frac{(t+h-\tau)^K}{K!} d\tau}_{= \frac{\mathbf{y}^{(K+1)}(\xi) h^{K+1}}{K!}}, \end{aligned} \quad (6.3.2.14)$$

for some $\xi \in [t, t+h]$. We conclude that, if $\mathbf{y} \in C^2([0, T])$, which is ensured for smooth \mathbf{f} , see Lemma 6.1.1.3, then

$$\mathbf{y}(t_{k+1}) - \mathbf{y}(t_k) = \dot{\mathbf{y}}(t_k) h_k + \frac{1}{2} \ddot{\mathbf{y}}(\xi_k) h_k^2 = \mathbf{f}(\mathbf{y}(t_k)) h_k + \frac{1}{2} \ddot{\mathbf{y}}(\xi_k) h_k^2 ,$$

for some $t_k \leq \xi_k \leq t_{k+1}$. This leads to an expression for the one-step error from (6.3.2.13)

$$\begin{aligned} \tau(h_k, \mathbf{y}(t_k)) &= \Psi^h \mathbf{y}(t_k) - \mathbf{y}(t_{k+1}) \\ &\stackrel{(6.3.2.11)}{=} \mathbf{y}(t_k) + h_k \mathbf{f}(\mathbf{y}(t_k)) - \mathbf{y}(t_k) - \mathbf{f}(\mathbf{y}(t_k)) h_k + \frac{1}{2} \ddot{\mathbf{y}}(\xi_k) h_k^2 \\ &= \frac{1}{2} \ddot{\mathbf{y}}(\xi_k) h_k^2 . \end{aligned} \quad (6.3.2.15)$$

Sloppily speaking, we observe

$$\boxed{\tau(h_k, \mathbf{y}(t_k)) = O(h_k^2)} \quad \text{uniformly for } h_k \rightarrow 0.$$

③ **Estimate for the propagated error** from (6.3.2.12)

$$\begin{aligned} \|\Psi^{h_k} \mathbf{y}_k - \Psi^{h_k} \mathbf{y}(t_k)\| &= \|\mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k) - \mathbf{y}(t_k) - h_k \mathbf{f}(\mathbf{y}(t_k))\| \\ &\stackrel{(6.3.2.10)}{\leq} (1 + Lh_k) \|\mathbf{y}_k - \mathbf{y}(t_k)\|. \end{aligned} \quad (6.3.2.16)$$


Thus we obtain *recursion* for error norms $\epsilon_k := \|\mathbf{e}_k\|$ by simply applying the \triangle -inequality:

$$\epsilon_{k+1} \leq (1 + h_k L) \epsilon_k + \rho_k, \quad \rho_k := \frac{1}{2} h_k^2 \max_{t_k \leq \tau \leq t_{k+1}} \|\ddot{\mathbf{y}}(\tau)\|. \quad (6.3.2.17)$$

Taking into account $\epsilon_0 = 0$, this leads to

$$\epsilon_k \leq \sum_{l=1}^k \prod_{j=1}^{l-1} (1 + Lh_j) \rho_l, \quad k = 1, \dots, N. \quad (6.3.2.18)$$

Use the elementary estimate $(1 + Lh_j) \leq \exp(Lh_j)$ (by convexity of exponential function):

$$(6.3.2.18) \Rightarrow \epsilon_k \leq \sum_{l=1}^k \prod_{j=1}^{l-1} \exp(Lh_j) \cdot \rho_l = \sum_{l=1}^k \exp(L \sum_{j=1}^{l-1} h_j) \rho_l.$$

Note: $\sum_{j=1}^{l-1} h_j \leq T$ for final time T and conclude

$$\begin{aligned} \epsilon_k &\leq \exp(LT) \sum_{l=1}^k \rho_l \leq \exp(LT) \max_k \frac{\rho_k}{h_k} \sum_{l=1}^k h_l \leq T \exp(LT) \max_{l=1, \dots, k} h_l \cdot \max_{t_0 \leq \tau \leq t_k} \|\ddot{\mathbf{y}}(\tau)\|. \\ \blacktriangleright \quad \|\mathbf{y}_k - \mathbf{y}(t_k)\| &\leq T \exp(LT) \max_{l=1, \dots, k} h_l \cdot \max_{t_0 \leq \tau \leq t_k} \|\ddot{\mathbf{y}}(\tau)\|. \end{aligned} \quad (6.3.2.19)$$

We can summarize the insight gleaned through this theoretical analysis as follows:

Total error arises from accumulation of propagated one-step errors!

From (6.3.2.19) we can conclude

- ◆ an error bound $= O(h)$, $h := \max_l h_l$ (\blacktriangleright 1st-order **algebraic convergence**)
- ◆ and that the error bound grows *exponentially* with the length T of the integration interval.

§6.3.2.20 (One-step error and order of a single step method) In the analysis of the global discretization error of the explicit Euler method in § 6.3.2.9 a one-step error of size $O(h_k^2)$ led to a total error of $O(h)$ through the effect of error accumulation over $M \approx h^{-1}$ steps. This relationship remains valid for almost all single step methods [DB02, Theorem 4.10]:

Order of algebraic convergence of single-step methods

Consider an IVP (6.1.3.2) with solution $t \mapsto \mathbf{y}(t)$ and a single step method defined by the discrete evolution Ψ (\rightarrow Def. 6.3.1.4). If the *one-step error along the solution trajectory* satisfies

(Φ is the evolution map associated with the ODE, see Def. 6.1.4.3)

$$\|\Psi^h \mathbf{y}(t) - \Phi^h \mathbf{y}(t)\| \leq Ch^{p+1} \quad \forall h \text{ sufficiently small}, t \in [0, T], \quad (6.3.2.22)$$

for some $p \in \mathbb{N}$ and $C > 0$, then, usually,

$$\max_k \|\mathbf{y}_k - \mathbf{y}(t_k)\| \leq \bar{C} h_M^p,$$

with $\bar{C} > 0$ independent of the temporal mesh M : The (pointwise) discretization error *converges algebraically* with order/rate p .

A rigorous statement as a theorem would involve some particular assumptions on Ψ , which we do not want to give here. These assumptions are satisfied, for instance, for all the methods presented in the sequel. You may refer to [DB02, Sect. 4.1] for further information.

In fact, it is remarkable that a local condition like (6.3.2.22) permits us to make a quantitative prediction of global convergence. This close relationship has made researchers introduce “order” also as a property of discrete evolutions.

Definition 6.3.2.23. Order of a discrete evolution operator

Let $\Psi : I \times D \mapsto \mathbb{R}^N$ be a discrete evolution for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ (with associated evolution operator $\Phi : I \times D \mapsto \mathbb{R}^N \rightarrow$ Def. 6.1.4.3). The largest integer $q \in \mathbb{N}_0$ such that

$$\forall \mathbf{y} \in D \quad \exists \tau_0 > 0: \quad \|\Psi^\tau \mathbf{y} - \Phi^\tau \mathbf{y}\| \leq C(\mathbf{y})\tau^{q+1} \quad \forall |\tau| \leq \tau_0 \quad (6.3.2.24)$$

is called the **order** of the discrete evolution. .

This notion of “order of a discrete evolution” allows a concise summary:

A single-step method (SSM, Def. 6.3.1.4) based on the discrete evolution Ψ satisfies
 Ψ of order $q \in \mathbb{N}$  **SSM converges algebraically with order q .**

EXAMPLE 6.3.2.25 (Orders of finite-difference single-step methods) Let us determine orders of the discrete evolutions for the three simple single-step methods introduced in Section 6.2, here listed with their corresponding discrete evolution operators Ψ (\rightarrow § 6.3.1.1) when applied to an autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$: for $\mathbf{y}_0 \in D \subset \mathbb{R}^N$,

$$\text{explicit (forward) Euler method (6.2.1.4):} \quad \Psi^\tau \mathbf{y}_0 := \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{y}_0), \quad (6.3.2.26)$$

$$\text{implicit (backward) Euler method (6.2.2.2):} \quad \Psi^\tau \mathbf{y}_0 := \mathbf{w}: \quad \mathbf{w} = \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{w}), \quad (6.3.2.27)$$

$$\text{implicit midpoint method (6.2.3.3):} \quad \Psi^\tau \mathbf{y}_0 := \mathbf{w}: \quad \mathbf{w} = \mathbf{y}_0 + \tau \mathbf{f}(\frac{1}{2}(\mathbf{y}_0 + \mathbf{w})). \quad (6.3.2.28)$$

The computation of their orders will rely on a fundamental technique for establishing (6.3.2.24) based on **Taylor expansion**. It hinges on smoothness of the vectorfield $\mathbf{f} = \mathbf{f}(\mathbf{y})$, which will ensure smoothness of solutions of the associated ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$. Thus, we make the following simplifying assumption:

Assumption 6.3.2.29. Smoothness of right-hand side vectorfield

The vectorfield $\mathbf{y} \mapsto \mathbf{f}(\mathbf{y})$ is C^∞ on \mathbb{R}^N

Let $\Phi = \Phi(t, \mathbf{y})$ denote the evolution operator (\rightarrow Def. 6.1.4.3) induced by $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, which, by definition, satisfies

$$\frac{\partial \Phi}{\partial t}(t, \mathbf{y}_0) = \mathbf{f}(\Phi^t \mathbf{y}_0) \quad \forall \mathbf{y}_0 \in D, t \in J(\mathbf{y}_0). \quad (6.1.4.4)$$

Setting $\mathbf{v}(\tau) := \Phi^\tau \mathbf{y}_0$, which is a solution of the initial-value problem $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$, we find for small τ , appealing to the one-dimensional chain rule and (6.1.4.4),

$$\frac{d\mathbf{v}}{d\tau}(\tau) = \mathbf{f}(\mathbf{v}(\tau)) \quad , \quad \frac{d^2\mathbf{v}}{d\tau^2}(\tau) = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(\mathbf{v}(\tau)) \frac{d\mathbf{v}}{d\tau}(\tau). \quad (6.3.2.30)$$

This yields the following truncated Taylor expansion

$$\begin{aligned} \Phi^\tau \mathbf{y}_0 &= \mathbf{v}(\tau) = \mathbf{v}(0) + \tau \frac{d\mathbf{v}}{d\tau}(0) + \frac{1}{2}\tau^2 \frac{d^2\mathbf{v}}{d\tau^2}(0) + O(\tau^3) \\ &= \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{y}_0) + \frac{1}{2}\tau^2 \mathbf{D}\mathbf{f}(\mathbf{y}_0)\mathbf{f}(t, \mathbf{y}_0) + O(\tau^3) \end{aligned} \quad (6.3.2.31)$$

for $\tau \rightarrow 0$. Note that the derivative $\mathbf{D}\mathbf{f}(\mathbf{y}_0)$ is an $N \times N$ Jacobi matrix. Explicit expressions for the remainder term involve second derivatives of \mathbf{f} .

- ① For the **explicit Euler method** (6.3.2.26) we immediately have from (6.3.2.31)

$$\Psi^\tau \mathbf{y}_0 - \Phi^\tau \mathbf{y}_0 = \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{y}_0) - \mathbf{y}_0 - \tau \mathbf{f}(\mathbf{y}_0) + O(\tau^2) = O(\tau^2) \quad \text{for } \tau \rightarrow 0.$$



The explicit Euler method is of **order 1**.

- ② It is not as straightforward for the **implicit Euler method**

$$\Psi^\tau \mathbf{y}_0 := \mathbf{w}(\tau): \quad \mathbf{w}(\tau) = \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{w}(\tau)). \quad (6.3.2.27)$$

First, we plug (6.3.2.27) into itself

$$\mathbf{w}(\tau) = \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{w}(\tau)) = \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{y}_0 + \tau \mathbf{f}(\mathbf{w}(\tau))),$$

and then use the truncated Taylor expansion of \mathbf{f} around \mathbf{y}_0

$$\mathbf{f}(\mathbf{y}_0 + \mathbf{v}) = \mathbf{f}(\mathbf{y}_0) + \mathbf{D}\mathbf{f}(\mathbf{y}_0)\mathbf{v} + O(\|\mathbf{v}\|^2) \quad \text{for } \mathbf{v} \rightarrow 0. \quad (6.3.2.32)$$

This gives

$$\mathbf{w}(\tau) = \mathbf{y}_0 + \tau(\mathbf{f}(\mathbf{y}_0) + \tau \mathbf{D}\mathbf{f}(\mathbf{y}_0)\mathbf{f}(\mathbf{w}(\tau))) + O(\tau^3) \quad \text{for } \tau \rightarrow 0.$$

Since $\Psi^\tau \mathbf{y}_0 = \mathbf{w}(\tau)$, matching terms with (6.3.2.31) we obtain

$$\Psi^\tau \mathbf{y}_0 - \Phi^\tau \mathbf{y}_0 = \tau^2 \mathbf{D}\mathbf{f}(\mathbf{y}_0)\mathbf{f}(\mathbf{w}(\tau)) + O(\tau^3) = O(\tau^2) \quad \text{for } \tau \rightarrow 0.$$

Thanks to the smoothness of \mathbf{f} the remainder terms will depend continuously on \mathbf{y}_0 .



The implicit Euler method has **order 1**.

- ③ For the **implicit midpoint rule**

$$\Psi^\tau \mathbf{y}_0 := \mathbf{w}(\tau): \quad \mathbf{w}(\tau) = \mathbf{y}_0 + \tau \mathbf{f}(\frac{1}{2}(\mathbf{y}_0 + \mathbf{w}(\tau))), \quad (6.3.2.28)$$

we follow the same idea and consider

$$\begin{aligned} \mathbf{w}(\tau) &= \mathbf{y}_0 + \tau \mathbf{f}(\frac{1}{2}(\mathbf{y}_0 + \mathbf{w}(\tau))) = \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{y}_0 + \frac{1}{2}\tau \mathbf{f}(\frac{1}{2}(\mathbf{y}_0 + \mathbf{w}(\tau)))) \\ &= \mathbf{y}_0 + \tau \mathbf{f}(\mathbf{y}_0 + \frac{1}{2}\tau \mathbf{f}(\mathbf{y}_0 + O(\tau))) \quad \text{for } \tau \rightarrow 0. \end{aligned}$$

Then we resort to the truncated Taylor expansion (6.3.2.32) and get for $\tau \rightarrow 0$

$$\begin{aligned}\mathbf{w}(\tau) &= \mathbf{y}_0 + \tau \left(\mathbf{f}(\mathbf{y}_0) + \mathbf{D}\mathbf{f}(\mathbf{y}_0) \frac{1}{2}\tau \mathbf{f}(\mathbf{y}_0) + O(\tau) \right) + O(\tau^3) \\ &= \mathbf{y}_0 + \tau \left(\mathbf{f}(\mathbf{y}_0) + \mathbf{D}\mathbf{f}(\mathbf{y}_0) \frac{1}{2}\tau (\mathbf{f}(\mathbf{y}_0) + O(\tau)) \right) + O(\tau^3).\end{aligned}$$

Matching with (9.2.6.10) shows $\mathbf{w}(\tau) - \Phi^\tau \mathbf{y}_0 = O(\tau^3)$ where the “O” just comprises *continuous* higher order derivatives of \mathbf{f} .

► The implicit midpoint method is an **order-2** method.

Hardly surprising, these analytic results match the orders of algebraic convergence observed in Exp. 6.3.2.5. □

Review question(s) 6.3.2.33 (Asymptotic convergence of single-step methods)

(Q6.3.2.33.A) We consider an autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ with smooth $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$. Explain, why the **one-step error**

$$\tau(h, \mathbf{y}) = \Psi^h \mathbf{y} - \Phi^h \mathbf{y}, \quad \mathbf{y} \in , \quad h \text{ "sufficiently small"},$$

for a **consistent** single-step method defined by the discrete evolution operator Ψ satisfies

$$\forall \mathbf{y} \in D: \quad \tau(h, \mathbf{y}) = O(h) \quad \text{for } h \rightarrow 0.$$

Definition 6.3.1.11. Consistent single step methods

A single step method according to Def. 6.3.1.4 based on a discrete evolution of the form

$$\Psi^h \mathbf{y} = \mathbf{y} + h \psi(h, \mathbf{y}) \quad \text{with} \quad \begin{aligned} \psi : I \times D &\rightarrow \mathbb{R}^N \text{ continuous,} \\ \psi(0, \mathbf{y}) &= \mathbf{f}(\mathbf{y}). \end{aligned} \quad (6.3.1.10)$$

is called **consistent** with the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

(Q6.3.2.33.B) Let $t \in I \mapsto \mathbf{y}(t)$, $I \subset \mathbb{R}$ an interval containing 0, denote the solution of the autonomous IVP

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0.$$

Assume that \mathbf{f} is continuously differentiable.

Use the chain rule to express $\dot{\mathbf{y}}(t^*)$ and $\ddot{\mathbf{y}}(t^*)$ by means of \mathbf{f} and its Jacobian.

(Q6.3.2.33.C) Based on the answer to Question (Q6.3.2.33.B), determine the order of a single-step method for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$ smooth, whose discrete evolution operator is given by

$$\Psi^h \mathbf{y} := \mathbf{y} + h \mathbf{f}(\mathbf{y}) + \frac{1}{2} h^2 \mathbf{D}\mathbf{f}(\mathbf{y}) \mathbf{f}(\mathbf{y}),$$

where $\mathbf{D}\mathbf{f}(\mathbf{y}) \in \mathbb{R}^{N,N}$ is the Jacobian of \mathbf{f} in $\mathbf{y} \in D$.

△

6.4 Explicit Runge-Kutta Single-Step Methods (RKSSMs)



Video tutorial for Section 6.4: Explicit Runge-Kutta Single-Step Methods (RKSSMs): (46 minutes) [Download link](#), [tablet notes](#)

So far we only know first and second order methods from Section 6.2: the explicit and implicit Euler method (6.2.1.4) and (6.2.2.2), respectively, are of first order, the implicit midpoint rule of second order. We observed this in Exp. 6.3.2.5 and it can be proved rigorously for all three methods adapting the arguments of § 6.3.2.9.

Thus, barring the impact of roundoff, the low-order polygonal approximation methods are guaranteed to achieve any prescribed accuracy provided that the mesh is fine enough. Why should we need any other timestepping schemes?

Remark 6.4.0.1 (Rationale for high-order single step methods cf. [DR08, Sect. 11.5.3])

We argue that the use of higher-order timestepping methods is highly *advisable for the sake of efficiency*. The reasoning is very similar to that of § 3.3.5.23 and Rem. 3.3.5.24, where we considered degree- p Lagrangian finite-element methods. The reader is advised to study those paragraphs again.

As we saw in § 6.3.2.3 error bounds for single step methods for the solution of IVPs will inevitably feature unknown constants “ $C > 0$ ”. Thus they do **not** give useful information about the discretization error for a concrete IVP and mesh. Hence, it is too ambitious to ask how many timesteps are needed so that $\|\mathbf{y}(T) - \mathbf{y}_N\|$ stays below a prescribed bound, cf. the discussion in the context of asymptotic error estimates for finite-element Galerkin methods in Section 3.3.5.

However, as already discussed in § 3.3.5.9 and § 3.3.5.12, an easier question can be answered by *asymptotic estimates* like (6.3.2.7):

What extra computational effort buys a prescribed *reduction of the error*?

The usual concept of “computational effort” for single step methods (→ Def. 6.3.1.4) is as follows

- Computational effort \sim total number of \mathbf{f} -evaluations for approximately solving the IVP,
- \sim number of timesteps, if evaluation of discrete evolution Ψ^h (→ Def. 6.3.1.4) requires fixed number of \mathbf{f} -evaluations,
- $\sim h^{-1}$, in the case of uniform timestep size $h > 0$ (equidistant mesh (6.3.2.4)).

Now, let us consider a single step method of order $p \in \mathbb{N}$, employed with a uniform timestep h_{old} . We focus on the maximal discretization error in the mesh points, see § 6.3.2.1. As in (3.3.5.23) we assume that the asymptotic error bounds are *sharp*:

$$\text{err}(h) \approx Ch^p \quad \text{for small meshwidth } h > 0 ,$$

with a “generic constant” $C > 0$ independent of the mesh.

$$\begin{aligned} \text{Goal: } \frac{\text{err}(h_{\text{new}})}{\text{err}(h_{\text{old}})} &\stackrel{!}{=} \frac{1}{\rho} \quad \text{for reduction factor } \rho > 1 . \\ (6.3.2.7) \Rightarrow \frac{h_{\text{new}}^p}{h_{\text{old}}^p} &\stackrel{!}{=} \frac{1}{\rho} \iff h_{\text{new}} = \rho^{-1/p} h_{\text{old}} . \end{aligned}$$

For single step method of order $p \in \mathbb{N}$

increase effort by factor $\rho^{1/p}$



reduce error by factor $\rho > 1$

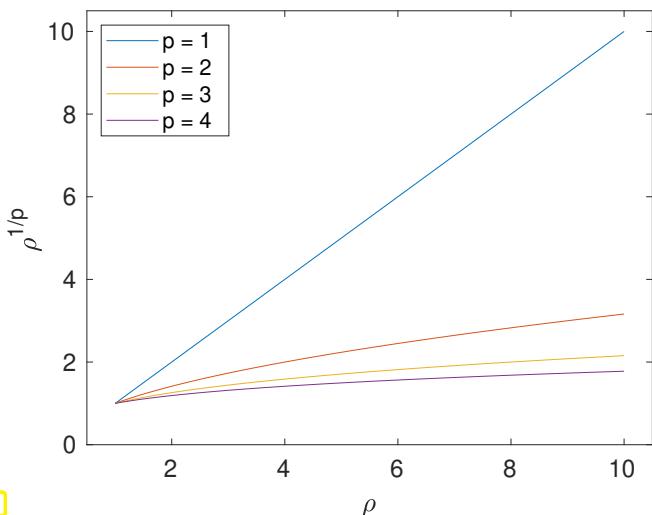


Fig. 311

▷ Plots of $\rho^{1/p}$ vs. ρ

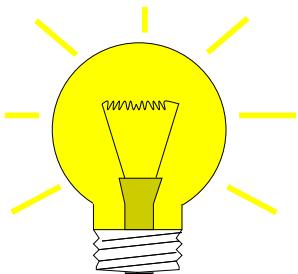
☞ the larger the order p , the less effort for a prescribed reduction of the error!

We remark that another (minor) rationale for using higher-order methods is to curb impact of roundoff errors (→ [Hip19, ??]) accumulating during timestepping [DR08, Sect. 11.5.3].

§6.4.0.2 (Bootstrap construction of explicit single step methods) Now we will build a class of methods that are explicit and achieve orders $p > 2$. The starting point is a simple *integral equation* satisfied by any solution $t \mapsto \mathbf{y}(t)$ of an initial value problems for the general ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$:

$$\text{IVP: } \begin{aligned} \dot{\mathbf{y}}(t) &= \mathbf{f}(t, \mathbf{y}(t)), \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \Rightarrow \mathbf{y}(t_1) = \mathbf{y}_0 + \int_{t_0}^{t_1} \mathbf{f}(\tau, \mathbf{y}(\tau)) d\tau$$

Idea: approximate the integral by means of s -point quadrature formula → [Hip19, ??], defined on the reference interval $[0, 1]$ with nodes c_1, \dots, c_s , weights b_1, \dots, b_s .



$$\mathbf{y}(t_1) \approx \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{f}(t_0 + c_i h, \boxed{\mathbf{y}(t_0 + c_i h)}) , \quad h := t_1 - t_0 . \quad (6.4.0.3)$$

Obtain these values by bootstrapping

“Bootstrapping” = use the same idea in a simpler version to get $\mathbf{y}(t_0 + c_i h)$, noting that these values can be replaced by other approximations obtained by methods already constructed (this approach will be elucidated in the next example).

What error can we afford in the approximation of $\mathbf{y}(t_0 + c_i h)$ (under the assumption that \mathbf{f} is Lipschitz continuous)? We take the cue from the considerations in § 6.3.2.9.

Goal: aim for one-step error bound $\mathbf{y}(t_1) - \mathbf{y}_1 = O(h^{p+1})$

Note that there is a factor h in front of the quadrature sum in (6.4.0.3). Thus, our goal can already be achieved, if only

$\mathbf{y}(t_0 + c_i h)$ is approximated up to an error $O(h^p)$,

again, because in (6.4.0.3) a factor of size h multiplies $\mathbf{f}(t_0 + c_i, \mathbf{y}(t_0 + c_i h))$.

This is accomplished by a less accurate discrete evolution than the one we are about to build. Thus, we can construct discrete evolutions of higher and higher order, in turns, starting with the explicit Euler method. All these methods will be **explicit**, that is, \mathbf{y}_1 can be computed directly from point values of \mathbf{f} . \square

EXAMPLE 6.4.0.4 (Simple Runge-Kutta methods by quadrature & bootstrapping) Now we apply the bootstrapping idea outlined above. We write $\mathbf{k}_\ell \in \mathbb{R}^N$ for the approximations of $\mathbf{y}(t_0 + c_i h)$.

- Quadrature formula = trapezoidal rule [Hip19, ??]:

$$Q(f) = \frac{1}{2}(f(0) + f(1)) \Leftrightarrow s = 2: c_1 = 0, c_2 = 1, b_1 = b_2 = \frac{1}{2}, \quad (6.4.0.5)$$

and $\mathbf{y}(t_1)$ approximated by explicit Euler step (6.2.1.4)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0), \quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1), \quad \mathbf{y}_1 = \mathbf{y}_0 + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2). \quad (6.4.0.6)$$

(6.4.0.6) = **explicit trapezoidal method** (for numerical integration of ODEs).

- Quadrature formula \rightarrow simplest Gauss quadrature formula = midpoint rule \rightarrow [Hip19, ??] & $\mathbf{y}(\frac{1}{2}(t_1 + t_0))$ approximated by explicit Euler step (6.2.1.4)

$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0), \quad \mathbf{k}_2 = \mathbf{f}(t_0 + \frac{h}{2}, \mathbf{y}_0 + \frac{h}{2}\mathbf{k}_1), \quad \mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{k}_2. \quad (6.4.0.7)$$

(6.4.0.7) = **explicit midpoint method** (for numerical integration of ODEs) [DR08, Alg. 11.18].

EXAMPLE 6.4.0.8 (Convergence of simple Runge-Kutta methods) We perform an empiric study of the order of the explicit single step methods constructed in Ex. 6.4.0.4.

- ◆ IVP: $\dot{y} = 10y(1 - y)$ (scalar logistic ODE (6.1.2.2)), initial value $y(0) = 0.01$, final time $T = 1$,
- ◆ Explicit single step methods, uniform timestep h .

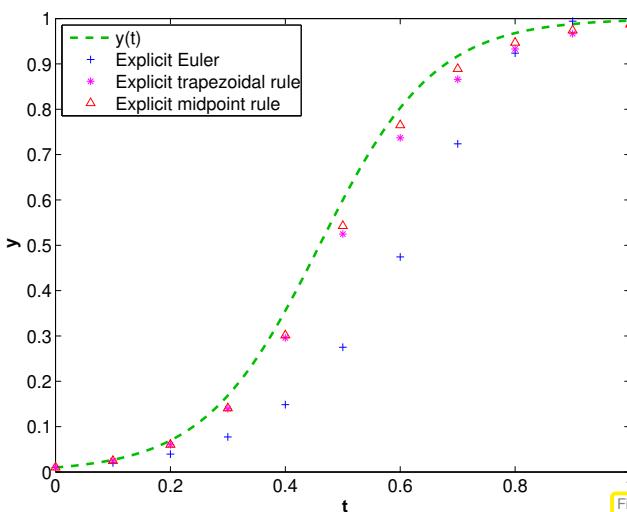
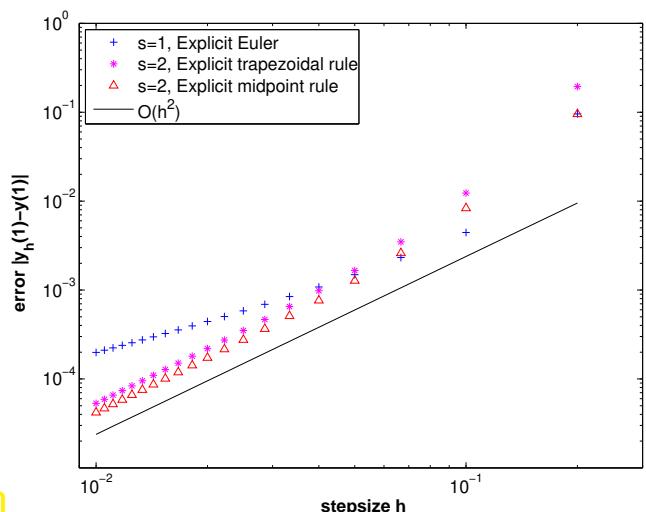


Fig. 312

Errors at final time $y_h(1) - y(1)$

$y_h(j/10), j = 1, \dots, 10$ for explicit RK-methods

Observation: obvious *algebraic convergence* in meshwidth h with integer rates/orders:

explicit trapezoidal method (6.4.0.6) \rightarrow order 2

explicit midpoint method (6.4.0.7) \rightarrow order 2

This is what one expects from the considerations in Ex. 6.4.0.4. □

The formulas that we have obtained follow a general pattern:

Definition 6.4.0.9. Explicit Runge-Kutta single-step method

For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{i-1} a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, an **s -stage explicit Runge-Kutta single step method** (RK-SSM) for the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, $\mathbf{f} : \Omega \rightarrow \mathbb{R}^N$, is defined by ($\mathbf{y}_0 \in D$)

$$\mathbf{k}_i := \mathbf{f}\left(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j\right), \quad i = 1, \dots, s, \quad \mathbf{y}_1 := \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i.$$

The vectors $\mathbf{k}_i \in \mathbb{R}^N$, $i = 1, \dots, s$, are called **increments**, $h > 0$ is the size of the timestep.

Recall Rem. 6.3.1.14 to understand how the discrete evolution for an explicit Runge-Kutta method is specified in this definition by giving the formulas for the first step. This is a convention widely adopted in the literature about numerical methods for ODEs. Of course, the increments \mathbf{k}_i have to be computed anew in each timestep.

The implementation of an s -stage explicit Runge-Kutta single step method according to Def. 6.4.0.9 is straightforward: The increments $\mathbf{k}_i \in \mathbb{R}^N$ are computed successively, starting from $\mathbf{k}_1 = \mathbf{f}(t_0 + c_1 h, \mathbf{y}_0)$.

- Only s \mathbf{f} -evaluations and AXPY operations (→ [Hip19, ??]) are required to compute the next state vector from the current.

In books and research articles a particular way to write down the coefficients characterizing RK-SSMs is widely used:

Butcher scheme notation for explicit RK-SSM

Shorthand notation for (explicit) Runge-Kutta methods [DR08, (11.75)]

Butcher scheme

(Note that \mathfrak{A} is a strictly lower triangular $s \times s$ -matrix)

$$\triangleright \quad \begin{array}{c|cc} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} \quad := \quad \begin{array}{c|cccc} c_1 & 0 & \cdots & & 0 \\ c_2 & a_{21} & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{s,s-1} & 0 \\ \hline b_1 & & \cdots & b_{s-1} & b_s \end{array} \quad (6.4.0.11)$$

Now we restrict ourselves to the case of an autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$. Matching Def. 6.4.0.9 and Def. 6.3.1.4, we see that the discrete evolution induced by an explicit Runge-Kutta single-step method is

$$\Psi^h \mathbf{y} = \mathbf{y} + h \sum_{i=1}^s b_i \mathbf{k}_i, \quad h \in \mathbb{R}, \quad \mathbf{y} \in D, \quad (6.4.0.12)$$

where the increments \mathbf{k}_i are defined by the increment equations

$$\mathbf{k}_i := \mathbf{f}\left(\mathbf{y} + h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j\right).$$

In line with (6.3.1.10), this discrete evolution can be written as

$$\Psi^h \mathbf{y} = \mathbf{y} + h \psi(h, \mathbf{y}) \quad , \quad \psi(h, \mathbf{y}) = \sum_{i=1}^s b_i \mathbf{k}_i.$$

Is this discrete evolution consistent in the sense of § 6.3.1.7, that is, does $\psi(0, \mathbf{y}) = \mathbf{f}(\mathbf{y})$ hold? If $h = 0$, the increment equations yield

$$h = 0 \Rightarrow \mathbf{k}_1 = \dots = \mathbf{k}_s = \mathbf{f}(\mathbf{y}) \quad \Rightarrow \quad \psi(0, \mathbf{y}) = \left(\sum_{i=1}^s b_i\right) \mathbf{f}(\mathbf{y}).$$

Corollary 6.4.0.13. Consistent Runge-Kutta single step methods

A Runge-Kutta single step method according to Def. 6.4.0.9 is *consistent* (\rightarrow Def. 6.3.1.11) with the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, if and only if

$$\sum_{i=1}^s b_i = 1.$$

Remark 6.4.0.14 (RK-SSM and quadrature rules) Note that in Def. 6.4.0.9 the coefficients C_i and b_i , $i \in \{1, \dots, s\}$, can be regarded as nodes and weights of a quadrature formula (\rightarrow [Hip19, ??]) on $[0, 1]$: apply the explicit Runge-Kutta single step method to the “ODE” $\dot{\mathbf{y}} = f(t)$, $f \in C^0([0, 1])$, on $[0, 1]$ with timestep $h = 1$ and initial value $\mathbf{y}(0)$, with exact solution

$$\dot{\mathbf{y}}(t) = f(t), \quad \mathbf{y}(0) = 0 \Rightarrow \mathbf{y}(t) = \int_0^t f(\tau) d\tau.$$

Then the formulas of Def. 6.4.0.9 reduce to

$$y_1 = 0 + \sum_{i=1}^s b_i f(c_i) \approx \int_0^1 f(\tau) d\tau.$$

Recall that the quadrature rule with these weights and nodes c_j will have order ≥ 1 (\rightarrow [Hip19, ??]), if the weights add up to 1! \square

EXAMPLE 6.4.0.15 (Butcher schemes for some explicit RK-SSM [DR08, Sect. 11.6.1]) The following explicit Runge-Kutta single step methods are often mentioned in literature.

- Explicit Euler method (6.2.1.4):

$$\begin{array}{c|cc} 0 & 0 \\ \hline 1 & \end{array} \quad \Rightarrow \quad \text{order} = 1$$

- explicit trapezoidal method (6.4.0.6):

$$\begin{array}{c|ccc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{array} \quad \Rightarrow \quad \text{order} = 2$$

- explicit midpoint method (6.4.0.7):

$$\begin{array}{c|ccc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline 0 & 1 \end{array} \quad \Rightarrow \quad \text{order} = 2$$

- Classical 4th-order RK-SSM:

$$\begin{array}{c|ccccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array} \quad \Rightarrow \quad \text{order} = 4$$

- Kutta's 3/8-method:

$$\begin{array}{c|ccccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{2}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ \hline 1 & 1 & -1 & 1 & 0 \\ \hline & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{array} \quad \Rightarrow \quad \text{order} = 4$$

Hosts of (explicit) Runge-Kutta methods can be found in the literature, see for example the [Wikipedia page](#). They are stated in the form of Butcher schemes (6.4.0.11) most of the time. \square

Remark 6.4.0.16 (Construction of higher order Runge-Kutta single step methods) Runge-Kutta single step methods of order $p > 2$ are not found by bootstrapping as in Ex. 6.4.0.4, because the resulting methods would have quite a lot of stages compared to their order.

Rather one derives **order conditions** yielding large non-linear systems of equations for the coefficients a_{ij} and b_i in Def. 6.4.0.9, see [DB02, Sect. 4.2.3] and [HLW06, Ch. III]. This approach is similar to the construction of a Gauss quadrature rule in [Hip19, ??]. Unfortunately, the systems of equations are very difficult to solve and no universal recipe is available. Nevertheless, through massive use of symbolic computation, explicit Runge-Kutta methods of order up to 19 have been constructed in this way. \square

Remark 6.4.0.17 (“Butcher barriers” for explicit RK-SSM) The following table gives lower bounds for the number of stages needed to achieve order p for an explicit Runge-Kutta method.

order p	1	2	3	4	5	6	7	8	≥ 9
minimal no. s of stages	1	2	3	4	6	7	9	11	$\geq p+3$

No general formula is has been discovered. What is known is that for explicit Runge-Kutta single step methods according to Def. 6.4.0.9

$$\text{order } p \leq \text{number } s \text{ of stages of RK-SSM}$$

\square



Supplementary literature. Runge-Kutta methods are presented in every textbook covering numerical integration: [DR08, Sect. 11.6], [Han02, Ch. 76], [QSS00, Sect. 11.8].

Review question(s) 6.4.0.18 (Explicit Runge-Kutta single-step methods)

(Q6.4.0.18.A) How many parameters describe a *consistent* 2-stage explicit Runge-Kutta method for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$?

Definition 6.4.0.9. Explicit Runge-Kutta method

For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{i-1} a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, an ***s*-stage explicit Runge-Kutta single step method** (RK-SSM) for the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, $\mathbf{f} : \Omega \rightarrow \mathbb{R}^N$, is defined by ($\mathbf{y}_0 \in D$)

$$\mathbf{k}_i := \mathbf{f}\left(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j\right), \quad i = 1, \dots, s, \quad \mathbf{y}_1 := \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i.$$

The vectors $\mathbf{k}_i \in \mathbb{R}^N$, $i = 1, \dots, s$, are called **increments**, $h > 0$ is the size of the timestep.

(Q6.4.0.18.B) Recall that by “autonomization” the initial value problem

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(t_0) = \mathbf{y}_0, \quad (6.4.0.19)$$

with $\mathbf{f} : I \times D \rightarrow \mathbb{R}^N$ can be converted into the equivalent IVP for the extended state $\mathbf{z} = [z_1, \dots, z_N, z_{N+1}]^\top := [\mathbf{y} \ t]^\top \in \mathbb{R}^{N+1}$:

$$\dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}), \quad \mathbf{g}(\mathbf{z}) := \begin{bmatrix} \mathbf{f}(z_1, \dots, z_N) \\ 1 \end{bmatrix}, \quad \mathbf{z}(0) = \begin{bmatrix} \mathbf{y}_0 \\ t_0 \end{bmatrix}. \quad (6.4.0.20)$$

Let us apply the same 2-stage explicit Runge-Kutta method to (6.4.0.19) and (6.4.0.20). When will both approaches produce the same sequence of states $\mathbf{y}_k \in D$?

(Q6.4.0.18.C) Formulate a generic 2-stage explicit Runge-Kutta method for the autonomous *second-order* ODE $\ddot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$.

Hint. Apply a standard 2-stage explicit Runge-Kutta method after transformation to an equivalent first-order ODE.

△

6.5 Adaptive Stepsize Control



Video tutorial for Section 6.5: Adaptive Stepsize Control: (56 minutes) [Download link](#), [tablet notes](#)

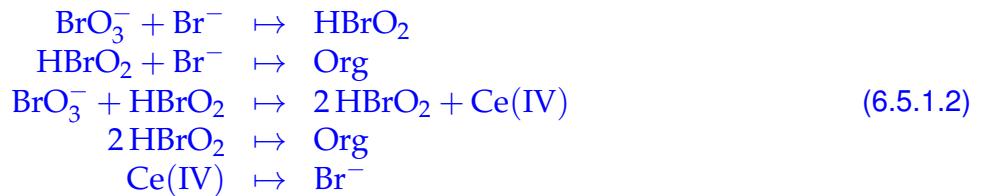
[Hip19, ??], in the context of numerical quadrature, teaches an **a-posteriori** way to adjust the mesh underlying a composite quadrature rule to the integrand: During the computation we estimate the local quadrature error by comparing the approximations obtained by using quadrature formulas of different order. The same policy for adapting the integration mesh is very popular in the context of numerical integration, too. Since the size $h_k := t_{k+1} - t_k$ of the cells of the temporal mesh is also called the **timestep size**, this kind of a-posteriori mesh adaptation is also known as stepsize control.

6.5.1 The Need for Timestep Adaptation

EXAMPLE 6.5.1.1 (Oregonator reaction) Chemical reaction kinetics is a field where ODE based models are very common. This example presents a famous reaction with extremely abrupt dynamics. Refer to

[Han02, Ch. 62] for more information about the ODE-based modelling of kinetics of chemical reactions.

This is a special case of an “oscillating” Zhabotinski-Belousov reaction [Gra02]:



By the laws of reaction kinetics of physical chemistry from (6.5.1.2) we can extract the following (system of) ordinary differential equation(s) for the concentrations of the different compounds:

$$\begin{aligned}
 y_1 := c(\text{BrO}_3^-): \quad \dot{y}_1 &= -k_1 y_1 y_2 - k_3 y_1 y_3, \\
 y_2 := c(\text{Br}^-): \quad \dot{y}_2 &= -k_1 y_1 y_2 - k_2 y_2 y_3 + k_5 y_5, \\
 y_3 := c(\text{HBrO}_2): \quad \dot{y}_3 &= k_1 y_1 y_2 - k_2 y_2 y_3 + k_3 y_1 y_3 - 2k_4 y_3^2, \\
 y_4 := c(\text{Org}): \quad \dot{y}_4 &= k_2 y_2 y_3 + k_4 y_3^2, \\
 y_5 := c(\text{Ce(IV)}): \quad \dot{y}_5 &= k_3 y_1 y_3 - k_5 y_5,
 \end{aligned} \tag{6.5.1.3}$$

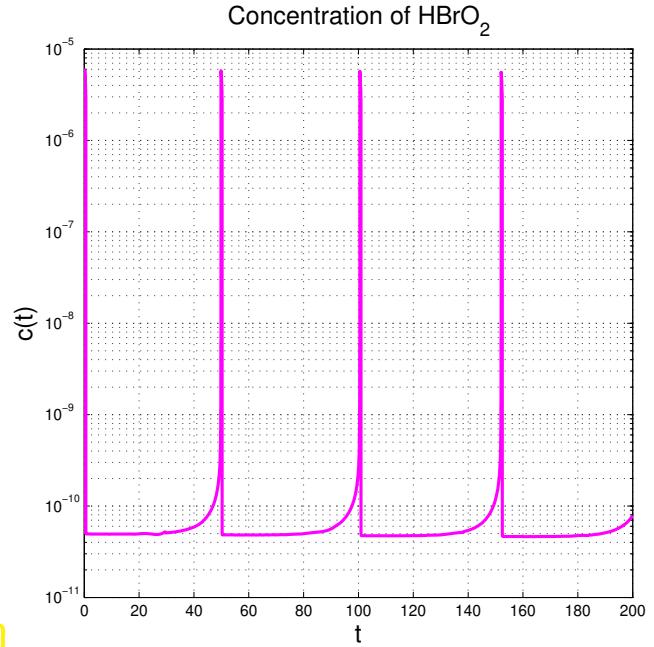
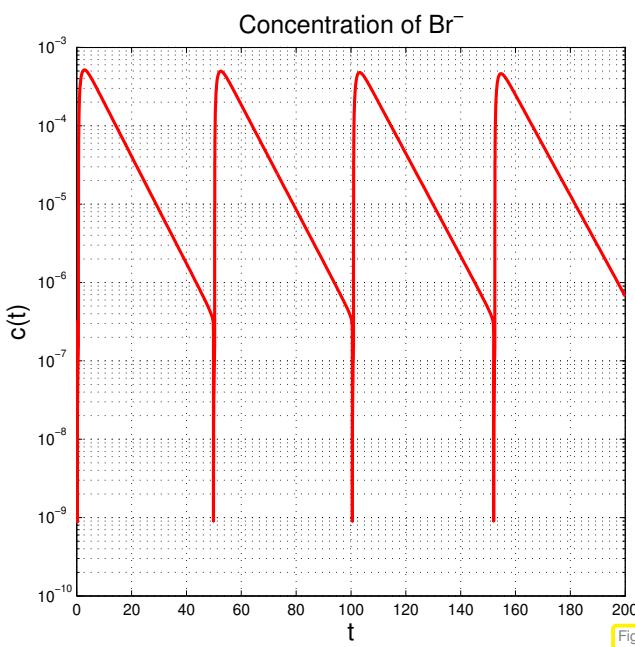
with (non-dimensionalized) reaction constants:

$$k_1 = 1.34, \quad k_2 = 1.6 \cdot 10^9, \quad k_3 = 8.0 \cdot 10^3, \quad k_4 = 4.0 \cdot 10^7, \quad k_5 = 1.0.$$



periodic chemical reaction ➡ Video 1, Video 2

These are results from highly accurate simulations with initial state $y_1(0) = 0.06$, $y_2(0) = 0.33 \cdot 10^{-6}$, $y_3(0) = 0.501 \cdot 10^{-10}$, $y_4(0) = 0.03$, $y_5(0) = 0.24 \cdot 10^{-7}$:



We observe a strongly non-uniform behavior of the solution in time.

This is very common with evolutions arising from practical models (circuit models, chemical reaction models, mechanical systems)

EXAMPLE 6.5.1.4 (Blow-up)

We return to the “explosion ODE” of Ex. 6.1.3.19 and consider the scalar autonomous IVP:

$$\begin{aligned} \dot{y} &= y^2, \quad y(0) = y_0 > 0. \\ \Rightarrow y(t) &= \frac{y_0}{1 - y_0 t}, \quad t < 1/y_0. \end{aligned}$$

As we have seen a solution exists only for finite time and then suffers a **Blow-up**, that is, $\lim_{t \rightarrow 1/y_0} y(t) = \infty$

$$\therefore J(y_0) =]-\infty, 1/y_0]!$$

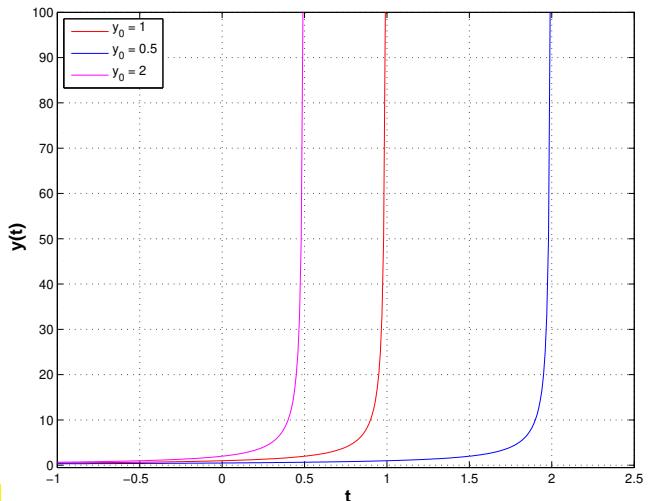


Fig. 316

How to choose temporal mesh $\{t_0 < t_1 < \dots < t_{N-1} < t_N\}$ for single step method in case $J(y_0)$ is not known, even worse, if it is not clear a priori that a blow up will happen?

Just imagine: what will result from equidistant explicit Euler integration (6.2.1.4) applied to the above IVP?

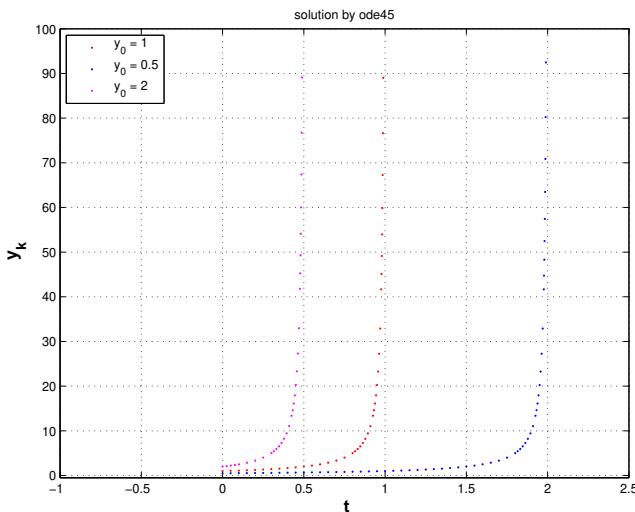


Fig. 317

A preview: There are single-step methods that can detect and resolve blow-ups!

- ▷ Simulations conducted with the numerical integrator **Ode45** introduced in § 6.5.3.3.

We observe that **Ode45** manages to reduce stepsize more and more as it approaches the singularity of the solution! How can it accomplish this feat!

6.5.2 Local-in-Time Stepsize Control

We identify as key challenge (discussed for autonomous ODEs below):

How to choose a *good temporal mesh* $\{0 = t_0 < t_1 < \dots < t_{M-1} < t_M\}$
for a given single step method applied to a concrete IVP?

What does “good” mean ?

Be efficient!

Be reliable!

Stepsize adaptation for single step methods

Objective: M as small as possible & $\max_{k=1,\dots,N} \|\mathbf{y}(t_k) - \mathbf{y}_k\| < \text{TOL}$, $\text{TOL} = \text{tolerance}$
 or $\|\mathbf{y}(T) - \mathbf{y}_M\| < \text{TOL}$

Policy: Try to curb/balance one-step error by

- ◆ adjusting current stepsize h_k ,
- ◆ predicting suitable next timestep h_{k+1}

}

**local-in-time
stepsize control**

Tool: Local-in-time one-step error estimator (*a posteriori*, based on \mathbf{y}_k, h_{k-1})

Why do we embrace local-in-time timestep control (based on estimating only the one-step error)? One could raise a serious objection: If a small time-local error in a single timestep leads to large error $\|\mathbf{y}_k - \mathbf{y}(t_k)\|$ at later times, then local-in-time timestep control is powerless about it and will not even notice!

Nevertheless, local-in-time timestep control is used almost exclusively,

- ☞ because we do not want to discard past timesteps, which could amount to tremendous waste of computational resources,
- ☞ because it is inexpensive and it works for many practical problems,
- ☞ because there is no reliable method that can deliver guaranteed accuracy for general IVP.

§6.5.2.2 (Local-in-time error estimation) We “recycle” heuristics already employed for adaptive quadrature, see [Hip19, ??], [Hip19, ??]. There we tried to get an idea of the local quadrature error by comparing two approximations of different order. Now we pursue a similar idea over a single timestep.

Idea:

Estimation of one-step error



Compare results for two discrete evolutions $\Psi^h, \tilde{\Psi}^h$ of different order over current timestep h :

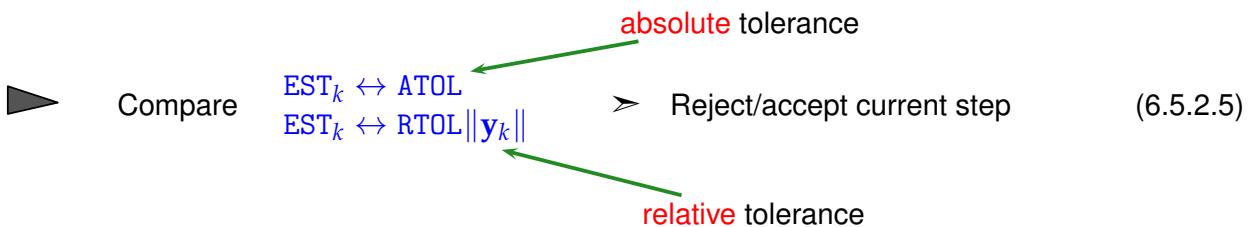
If $\text{Order}(\tilde{\Psi}) > \text{Order}(\Psi)$, then we expect

If $\text{Order}(\tilde{\Psi}) > \text{Order}(\Psi)$, then we expect

$$\underbrace{\Phi^h \mathbf{y}_k - \Psi^h \mathbf{y}_k}_{\text{one-step error}} \approx \text{EST}_k := \tilde{\Psi}^h \mathbf{y}_k - \Psi^h \mathbf{y}_k. \quad (6.5.2.3)$$

Heuristics for concrete $h > 0$ A computable quantity!

§6.5.2.4 ((Crude) local timestep control) We take for granted the availability of a local error estimate EST_k that we have computed for a current stepsize h . We specify target values $\text{ATOL} > 0, \text{RTOL} > 0$ of absolute and relative tolerances to be met by the local error and implement the following policy:



Both tolerances $RTOL > 0$ and $ATOL > 0$ have to be supplied by the user of the adaptive algorithm. The absolute tolerance is usually chosen significantly smaller than the relative tolerance and merely serves as a safeguard against non-termination in case $y_k \approx 0$. For a similar use of absolute and relative tolerances see [Hip19, ??], which deals with termination criteria for iterations, in particular [Hip19, ??].

☞ Simple accept/reject algorithm:

- | | |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $EST_k < \max\{ATOL, \ y_k\ RTOL\}$: | Accept current step: |
| | <ul style="list-style-type: none"> • Advance by one timestep (stepsize h), • use larger stepsize (αh with some $\alpha > 1$) for next step (*) |
| $EST_k > \max\{ATOL, \ y_k\ RTOL\}$: | Reject current step: |
| | <ul style="list-style-type: none"> • Repeat current timestep • with smaller stepsize $< h$, e.g., $\frac{1}{2}h$ |

The rationale behind the adjustment of the timestep size in (*) is the following: if the current stepsize guarantees sufficiently small one-step error, then it might be possible to obtain a still acceptable one-step error with a larger timestep, which would enhance efficiency (fewer timesteps for total numerical integration). This should be tried, since timestep control will usually provide a safeguard against undue loss of accuracy.

The following C++ code implements a wrapper function `odeintadapt()` for a general adaptive single-step method according to the policy outlined above. The arguments are meant to pass the following information:

- `Psilow, Psihigh`: functors passing discrete evolution operators for autonomous ODE of different order, type `@(y, h)`, expecting a state (usually a column vector) as first argument, and a stepsize as second,
- `T`: final time $T > 0$,
- `y0`: initial state y_0 ,
- `h0`: stepsize h_0 for the first timestep
- `reltol, abstol`: relative and absolute tolerances, see (6.5.2.5),
- `hmin`: minimal stepsize, timestepping terminates when stepsize control $h_k < h_{\min}$, which is relevant for detecting blow-ups or collapse of the solution.

C++11 code 6.5.2.6: Simple local stepsize control for single step methods → GITHUB

```

2 // Auxiliary function: default norm for an EIGEN vector type
3 template <class State>
4 double _norm(const State &y) {
5     return y.norm();
6 }
7
8 // Adaptive numerical integrator based on local-in-time stepsize control
9 template <class DiscEvolOp, class State,
10         class NormFunc = decltype(_norm<State>) >

```

```

11 std::vector<std::pair<double, State>> odeintadapt(  

12     DiscEvolOp &&Psiow, DiscEvolOp &&Psihigh, const State &y0, double T,  

13     double h0, double reltol, double abstol, double hmin,  

14     NormFunc &norm = _norm<State>) {  

15     double t = 0; // initial time  $t_0 = 0$   

16     State y = y0; // current state  

17     double h = h0; // timestep to start with  

18     std::vector<std::pair<double, State>>  

19         states; // vector of times/computed states:  

20         //  $(t_k, y_k)_k$   

21     states.push_back({t, y}); // initial time and state  

22  

23     while ((states.back().first < T) &&  

24         (h >= hmin)) { //  

25         State yh = Psihigh(  

26             h, y); // high order discrete evolution  $\tilde{\Psi}^h$   

27             //  

28         State yH = Psiow(h, y); // low order discrete evolution  

29             //  $\Psi^h$   

30         double est =  

31             norm(yH - yh); // local error estimate  

32             //  $EST_k$   

33  

34         if (est <  

35             std::max(  

36                 reltol * norm(y),  

37                 abstol)) { // step accepted  

38             y = yh; // use high order approximation  

39             t = t + std::min(T - t, h); // next time  $t_k$   

40             states.push_back({t, y}); //  

41             h = 1.1 * h; // try with increased stepsize  

42         } else { // step rejected  

43             h = h / 2; // try with half the stepsize  

44         }  

45         // Numerical integration has ground to a halt !  

46         if (h < hmin) {  

47             std::cerr << "Warning: Failure at t=" << states.back().first  

48             << ". Unable to meet integration tolerances without reducing "  

49             "the step "  

50             << "size below the smallest value allowed (" << hmin  

51             << ") at time t." << std::endl;  

52         }  

53     }  

54     return states;  

55 }

```

Comments on Code 6.5.2.6:

- line 24: check whether final time is reached or timestepping has ground to a halt ($h_k < h_{\min}$).
- line 27, 29: advance state by low and high order integrator.
- line 32: compute norm of estimated error, see (6.5.2.3).
- line 37: make comparison (6.5.2.5) to decide whether to accept or reject local step.
- line 40, 41: step accepted, update state and current time and suggest 1.1 times the current stepsize for next step.
- line 43 step rejected, try again with half the stepsize.

- Return value is a vector of pairs consisting of
 - times $t \leftrightarrow$ temporal mesh $t_0 < t_1 < t_2 < \dots < t_N < T$, where $t_N < T$ indicated premature termination (collapse, blow-up),
 - states $y \leftrightarrow$ sequence $(y_k)_{k=0}^N$.

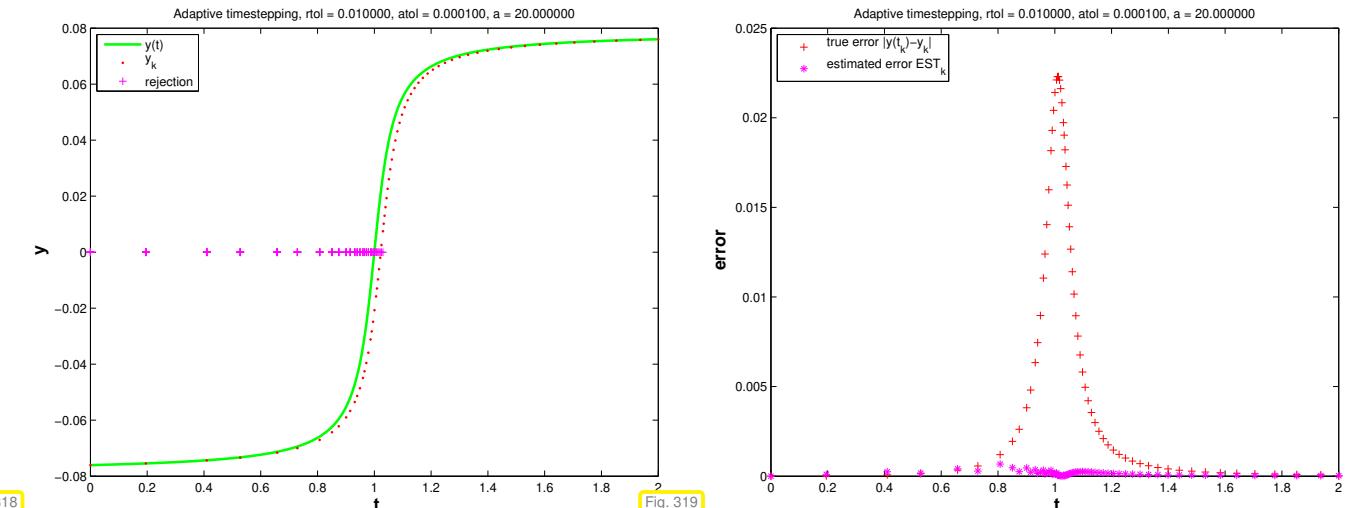
Remark 6.5.2.7 (Estimation of “wrong” error?) We face the same conundrum as in the case of adaptive numerical quadrature, see [Hip19, ??]:

! By the heuristic considerations, see (6.5.2.3) it seems that EST_k measures the one-step error for the low-order method Ψ and that we should use $y_{k+1} = \tilde{\Psi}^{h_k} y_k$, if the timestep is accepted.

However, it would be foolish not to use the better value $y_{k+1} = \tilde{\Psi}^{h_k} y_k$, since it is available for free. This is what is done in every implementation of adaptive methods, also in Code 6.5.2.6, and this choice can be justified by control theoretic arguments [DB02, Sect. 5.2].

EXPERIMENT 6.5.2.8 (Simple adaptive stepsize control) We test adaptive timestepping routine from Code 6.5.2.6 for a scalar IVP and compare the estimated local error and true local error.

- ◆ IVP for ODE $\dot{y} = \cos(\alpha y)^2$, $\alpha > 0$, solution $y(t) = \arctan(\alpha(t - c)) / \alpha$ for $y(0) \in]-\pi/2, \pi/2[$
- ◆ Simple adaptive timestepping based on explicit Euler (6.2.1.4) and explicit trapezoidal rule (6.4.0.6)



Statistics:

66 timesteps, 131 rejected timesteps

Observations:

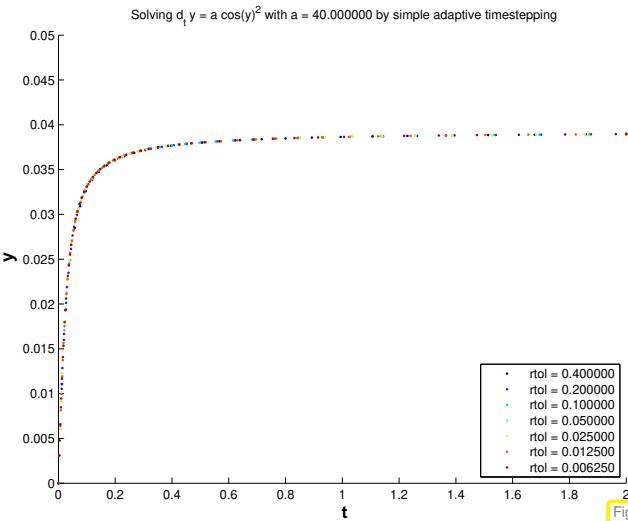
- ☞ Adaptive timestepping well resolves local features of solution $y(t)$ at $t = 1$
- ☞ Estimated error (an estimate for the one-step error) and true error are **not** related! To understand this recall Rem. 6.5.2.7.

EXPERIMENT 6.5.2.9 (Gain through adaptivity → Exp. 6.5.2.8) In this experiment we want to explore whether adaptive timestepping is worth while, as regards reduction of computational effort without sacrificing accuracy.

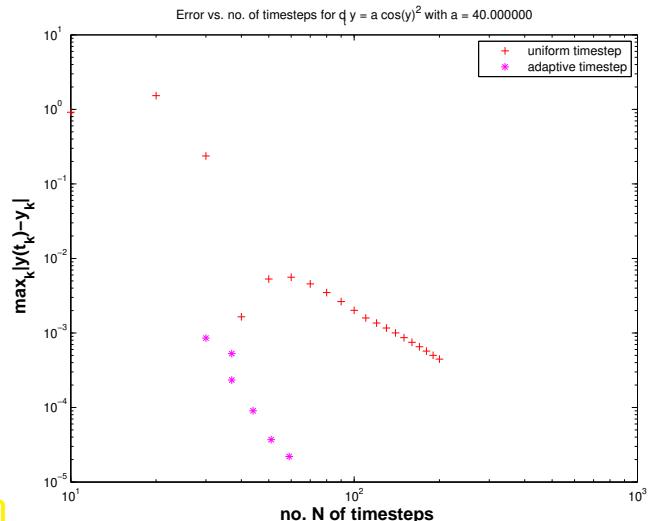
We retain the simple adaptive timestepping from previous experiment Exp. 6.5.2.8 and also study the same IVP.

New: initial state $y(0) = 0$!

Now we examine the dependence of the maximal discretization error in mesh points on the computational effort. The latter is proportional to the number of timesteps.



Solutions $(y_k)_k$ for different values of rtol



Error vs. computational effort

Observations:

- Adaptive timestepping achieves much better accuracy for a fixed computational effort.

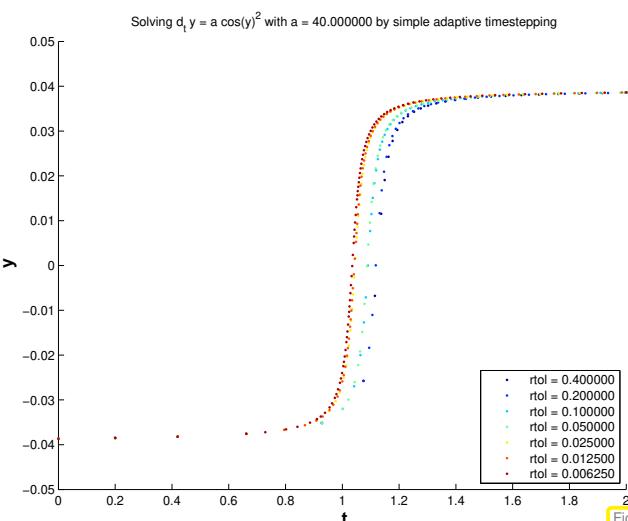
EXPERIMENT 6.5.2.10 (“Failure” of adaptive timestepping → Exp. 6.5.2.9)

Same ODE and simple adaptive timestepping as in previous experiment Exp. 6.5.2.9.

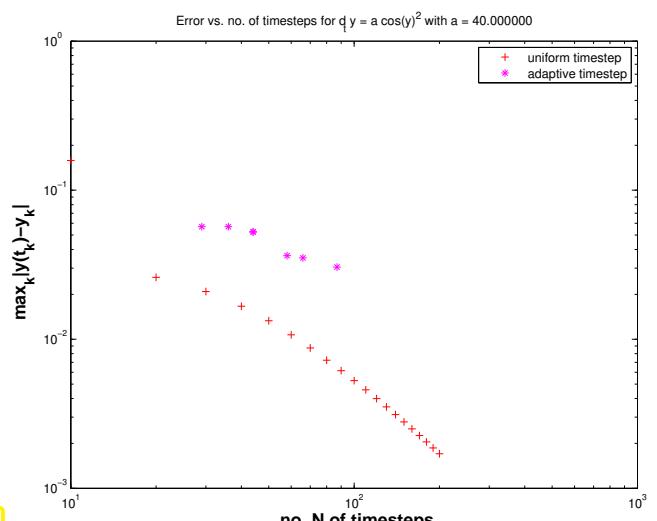
$$\dot{y} = \cos^2(\alpha y) \Rightarrow y(t) = \arctan(\alpha(t - c)) / \alpha, y(0) \in \left[-\frac{\pi}{2\alpha}, \frac{\pi}{2\alpha} \right],$$

for $\alpha = 40$.

Now: initial state $y(0) = -0.0386 \approx \frac{\pi}{2\alpha}$ as in Exp. 6.5.2.8



Solutions $(y_k)_k$ for different values of rtol



Error vs. computational effort

Observations:

- ☞ Adaptive timestepping leads to larger errors at the same computational cost as uniform timestepping !

Explanation: the position of the steep step of the solution has a sensitive dependence on an initial value, if $y(0) \approx \frac{\pi}{2\alpha}$:

$$y(t) = \frac{1}{\alpha} \arctan(\alpha(t + \tan(y_0/\alpha))) , \quad \text{step at } \approx -\tan(y_0/\alpha) .$$

Hence, small local errors in the initial timesteps will lead to large errors at around time $t \approx 1$. The stepsize control is mistaken in condoning these small one-step errors in the first few steps and, therefore, incurs huge errors later.

However, the perspective of **backward error analysis** (\rightarrow [Hip19, ??]) rehabilitates adaptive stepsize control in this case: it gives us a numerical solution that is very close to the exact solution of the ODE with slightly perturbed initial state y_0 . \square

§6.5.2.11 (Refined local stepsize control \rightarrow [DR08, Sect. 11.7]) The above algorithm (Code 6.5.2.6) is simple, but the rule for increasing/shrinking of timestep “squanders” the information contained in $\text{EST}_k : \text{TOL}$:

More ambitious goal !	When $\text{EST}_k > \text{TOL}$: stepsize adjustment better $h_k = ?$
	When $\text{EST}_k < \text{TOL}$: stepsize prediction good $h_{k+1} = ?$

Assumption: At our disposal are two discrete evolutions:

- ◆ Ψ with $\text{order}(\Psi) = p$ (\rightarrow “low order” single step method)
- ◆ $\tilde{\Psi}$ with $\text{order}(\tilde{\Psi}) > p$ (\rightarrow “higher order” single step method)

These are the same building blocks as for the simple adaptive strategy employed in Code 6.5.2.6 (passed as arguments Psilow , Psihigh there).

According to (6.3.2.22) we can expect the following asymptotic decay of the one-step errors for $h \rightarrow 0$:

$$\begin{aligned} \Psi^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k) &= ch^{p+1} + O(h_k^{p+2}), \\ \tilde{\Psi}^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k) &= O(h_k^{p+2}), \end{aligned} \tag{6.5.2.12}$$

with some (unknown) constant $c > 0$. Why h^{p+1} ? Remember the estimate (6.3.2.15) from the error analysis of the explicit Euler method: we also found $O(h_k^2)$ there for the one-step error of a single step method of order 1.

Heuristic reasoning: The timestep h_k is small \Rightarrow “higher order terms” $O(h_k^{p+2})$ can be ignored.

► $\Psi^{h_k} \mathbf{y}(t_k) - \Phi^{h_k} \mathbf{y}(t_k) \doteq ch_k^{p+1} + O(h_k^{p+2}), \quad \Rightarrow \quad \text{EST}_k \doteq ch_k^{p+1}$. $\tag{6.5.2.13}$

☞ notation: \doteq equality up to higher order terms in h_k

$$\text{EST}_k \doteq ch_k^{p+1} \quad \Rightarrow \quad c \doteq \frac{\text{EST}_k}{h_k^{p+1}} . \tag{6.5.2.14}$$

Available in algorithm, see (6.5.2.3)

For the sake of *accuracy* (demands “ $\text{EST}_k < \text{TOL}$ ”) & *efficiency* (favors “ $>$ ”) we aim for

$$\text{EST}_k \stackrel{!}{=} \text{TOL} := \max\{\text{ATOL}, \|\mathbf{y}_k\| \text{RTOL}\}. \quad (6.5.2.15)$$

What timestep h_* can actually achieve (6.5.2.15), if we “believe” (heuristics!) in (6.5.2.13) (and, therefore, in (6.5.2.14))?

$$(6.5.2.14) \& (6.5.2.15) \Rightarrow \text{TOL} = \frac{\text{EST}_k}{h_k^{p+1}} h_*^{p+1}. \quad (6.5.2.16)$$



“Optimal timestep”:
$$h_* = h^{p+1} \sqrt{\frac{\text{TOL}}{\text{EST}_k}}. \quad (6.5.2.17)$$

(stepsize prediction)

The proposed timestep size h_* will be used in both cases:

(Rejection of current timestep): In case $\text{EST}_k > \text{TOL}$ ➤ repeat step with stepsize h_* .

(Acceptance of current timestep): If $\text{EST}_k \leq \text{TOL}$ ➤ use h_* as stepsize for next step.

C++11 code 6.5.2.18: Refined local stepsize control for single step methods

→ GITHUB

```

2 // Auxiliary function: default norm for an EIGEN vector type
3 template <class State>
4 double _norm(const State &y) {
5     return y.norm();
6 }
7 // Adaptive single-step integrator
8 template <class DiscEvolOp, class State,
9         class NormFunc = decltype(_norm<State>) >
10 std::vector<std::pair<double, State>> odeintssctrl(
11     DiscEvolOp &&Psilow, unsigned int p, DiscEvolOp &&Psihigh, const State &y0,
12     double T, double h0, double reltol, double abstol, double hmin,
13     NormFunc &norm = _norm<State> ) {
14     double t = 0; // initial time  $t_0 = 0$ 
15     State y = y0; // current state, initialized here
16     double h = h0; // timestep to start with
17     std::vector<std::pair<double, State>>
18         states; // vector  $(t_k, \mathbf{y}_k)_k$ 
19     states.push_back({t, y});
20
21     // Main timestepping loop
22     while ((states.back().first < T) && (h >= hmin)) { //
23         State yh =
24             Psihigh(h, y); // high order discrete evolution
25             //  $\tilde{\Psi}^h$ 
26         State yH = Psilow(h, y); // low order discrete evolution
27             //  $\Psi^h$ 
28         double est = norm(
29             yH -
30             yh); //  $\leftrightarrow \text{EST}_k$ 
31         double tol =
32             std::max(reltol * norm(y),
33                     abstol); // effective tolerance
34
35         // Optimal stepsize according to (6.5.2.17)
36         if (est < tol) { // step accepted

```

```

37         // 
38     states.push_back({t = t + std::min(T - t, h),
39                         y = yh}); // store next approximate state
40 }
41 h *= std::max(
42     0.5,
43     std::min(2., 0.9 * std::pow(tol / est,
44                               1. / (p + 1)))); // 
45 if (h < hmin) {
46     std::cerr
47     << "Warning: Failure at t=" << states.back().first
48     << ". Unable to meet integration tolerances without reducing the step"
49     << " size below the smallest value allowed (" << hmin
50     << ") at time t." << std::endl;
51 }
52 }
53 return states;
54 }
```

Comments on Code 6.5.2.18 (see comments on Code 6.5.2.6 for more explanations):

- Input arguments as for Code 6.5.2.6, except for $p \hat{=} \text{order of lower order discrete evolution}$.
- line 44: compute presumably better local stepsize according to (6.5.2.17),
- line 37: decide whether to repeat the step or advance,
- line 37: extend output arrays if current step has not been rejected.

6.5.3 Embedded Runge-Kutta Methods

For higher order RK-SSM with a considerable number of stages computing different sets of increments (\rightarrow Def. 6.4.0.9) for two methods of different order just for the sake of local-in-time stepsize control would mean undue effort. This makes the following embedding idea attractive:

Embedding idea for RK-SSM

Use two RK-SSMs based on the **same increments**, that is, built with the same coefficients a_{ij} , but different weights b_i , see Def. 6.4.0.9 for the formulas, and **different orders** p and $p+1$.

Augmented Butcher scheme for **embedded explicit Runge-Kutta methods**

(The lower-order scheme has weights \widehat{b}_i .)

$$\begin{array}{c|ccccc} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T & & & & \\ \hline & \widehat{\mathbf{b}}^T & & & & \end{array} := \begin{array}{c|ccccc} c_1 & a_{11} & \cdots & & a_{1s} \\ \vdots & \vdots & & & \vdots \\ c_s & a_{s1} & \cdots & & a_{ss} \\ \hline b_1 & \cdots & & & b_s \\ \hline \widehat{b}_1 & \cdots & & & \widehat{b}_s \end{array} .$$

EXAMPLE 6.5.3.2 (Commonly used embedded explicit Runge-Kutta methods) The following two embedded RK-SSM, presented in the form of their extended Butcher schemes, provided single step methods of orders 4 & 5.

0	
$\frac{1}{3}$	$\frac{1}{3}$
$\frac{1}{3}$	$\frac{1}{6}$ $\frac{1}{6}$
$\frac{1}{2}$	$\frac{1}{8}$ 0 $\frac{3}{8}$
1	$\frac{1}{2}$ 0 $-\frac{3}{2}$ 2
y_1	$\frac{1}{6}$ 0 0 $\frac{2}{3}$ $\frac{1}{6}$
\hat{y}_1	$\frac{1}{10}$ 0 $\frac{3}{10}$ $\frac{2}{5}$ $\frac{1}{5}$

Merson's embedded RK-SSM

0	
$\frac{1}{2}$	$\frac{1}{2}$
$\frac{1}{2}$	0 $\frac{1}{2}$
1	0 0 1
$\frac{3}{4}$	$\frac{5}{32}$ $\frac{7}{32}$ $\frac{13}{32}$ $-\frac{1}{32}$
y_1	$\frac{1}{6}$ $\frac{1}{3}$ $\frac{1}{3}$ $\frac{1}{6}$
\hat{y}_1	$-\frac{1}{2}$ $\frac{7}{3}$ $\frac{7}{3}$ $\frac{13}{6}$ $-\frac{16}{3}$

Fehlberg's embedded RK-SSM

§6.5.3.3 (EIGEN-compatible adaptive explicit embedded Runge-Kutta integrator) An implementation of an explicit embedded Runge-Kutta single-step method with adaptive stepsize control for solving an autonomous IVP is provided by the utility class **Ode45** → **GITLAB**: The class name already indicates the orders of the pair of single step methods used for adaptive stepsize control:

$$\Psi \triangleq \text{RK-method of order } 4 \quad \tilde{\Psi} \triangleq \text{RK-method of order } 5$$

Ode45

Refer to the class implementation → **GITLAB** and the data members `Ode45::mA`, `Ode45::vb4`, `Ode45::vb5` for the underlying coefficients in \mathbf{a} , \mathbf{b} , and $\widehat{\mathbf{b}}$, as introduced in Section 6.5.3.

The class is templated with two type parameters:

```
template <class StateType,
class RhsType = std::function<StateType (const StateType &) >>
class Ode45 { ... };
```

- (i) **StateType**: type for vectors in state space V , e.g. a fixed size vector type of EIGEN: `Eigen::Matrix<double, N, 1>`, where N is an integer constant § 6.2.0.1.
- (ii) **RhsType**: a functor type, see [Hip19, ??], for the right hand side function f ; must match **StateType**, default type provided.

The functor for the right hand side $f : D \subset V \rightarrow V$ of the ODE $\dot{\mathbf{y}} = f(\mathbf{y})$ is specified as an argument of the constructor. The single-step numerical integrator is invoked by the templated method

```
template <class NormFunc = decltype (_norm<StateType>) >
std::vector< std::pair<StateType, double>>
solve(const StateType & y0, double T, const NormFunc & norm =
      _norm<StateType>);
```

The following arguments have to be supplied:

1. y_0 : the initial value \mathbf{y}_0
2. T : the final time T , initial time $t_0 = 0$ is assumed, because the class can deal with autonomous ODEs only, recall § 6.1.3.3.
3. $norm$: a functor returning a suitable norm for a state vector. Defaults to EIGEN's maximum vector norm.

The method returns a vector of 2-tuples (\mathbf{y}_k, t_k) (note the order!), $k = 0, \dots, N$, of temporal mesh points t_k , $t_0 = 0$, $t_N = T$, see § 6.2.0.2, and approximate states $\mathbf{y}_k \approx \mathbf{y}(t_k)$, where $t \mapsto \mathbf{y}(t)$ stands for the exact solution of the initial value problem.

The arguments of `solve()` are not sufficient to control the behavior of the adaptive integrator. In addition, one can set data members of the data structure **Ode45.options** to configure an instance `ode45obj` of **Ode45**:

```
ode45obj.options.<option_you_want_to_set> = <value>;
```

In particular, the key data fields for adaptive timestepping are

- `rtol`: relative tolerance for error control (default is 10^{-6})
- `atol`: absolute tolerance for error control (default is 10^{-8})

For complete information about all control parameters and their default values see the **Ode45** class definition → [GITLAB](#).

The following self-explanatory code snippet uses the numerical integrator class **Ode45** for solving a scalar autonomous ODE.

C++11 code 6.5.3.4: Invocation of adaptive embedded Runge-Kutta-Fehlberg integrator
[→ GITLAB](#)

```
2 int main(int /*argc*/, char** /*argv*/) {
3     // Types to be used for a scalar ODE with state space ℝ
4     using StateType = double;
5     using RhsType = std::function<StateType(StateType)>;
6     // Logistic differential equation (6.1.2.2)
7     RhsType f = [] (StateType y) { return 5 * y * (1 - y); };
8     StateType y0 = 0.2; // Initial value
9     // Exact solution of IVP, see (6.1.2.3)
10    auto y = [y0](double t) { return y0 / (y0 + (1 - y0) * std::exp(-5 * t)); };
11    // State space ℝ, simple modulus supplies norm
12    auto normFunc = [] (StateType x) { return std::fabs(x); };
13
14    // Invoke explicit Runge-Kutta method with stepsize control
15    Ode45<StateType, RhsType> integrator(f);
16    // Do the timestepping with adaptive strategy of Section 6.5
17    std::vector<std::pair<StateType, double>> states =
18        integrator.solve(y0, 1.0, normFunc);
19    // Output information accumulation during numerical integration
20    integrator.options.do_statistics = true;
21    integrator.print();
22    // Output norm of discretization error in nodes of adaptively
23    // generated
24    // temporal mesh
25    for (auto state : states) {
26        std::cout << "t = " << state.second << ", y = " << state.first
27            << ", |err| = " << fabs(state.first - y(state.second))
28            << std::endl;
29    }
30    return 0;
}
```

Remark 6.5.3.5 (Tolerances and accuracy) As we have learned in § 6.5.3.3 for objects of the class **Ode45** tolerances for the refined local stepsize control of § 6.5.2.11 can be specified by setting the member

variables `options.rtol` and `options.atol`.

The possibility to pass tolerances to numerical integrators based on adaptive timestepping may tempt the user into believing that they allow to control the accuracy of the solutions. However, as is clear from § 6.5.2.11, these tolerances are solely applied to local error estimates and, inherently, have nothing to do with global discretization errors, see Exp. 6.5.2.8.

No global error control through local-in-time adaptive timestepping

The absolute/relative tolerances imposed for local-in-time adaptive timestepping do *not* allow to predict accuracy of solution!

EXAMPLE 6.5.3.7 (Adaptive timestepping for mechanical problem) We test the effect of adaptive stepsize control in MATLAB for the equations of motion describing the planar movement of a point mass in a conservative force field $\mathbf{x} \in \mathbb{R}^2 \mapsto \mathbf{F}(\mathbf{x}) \in \mathbb{R}^2$: Let $t \mapsto \mathbf{y}(t) \in \mathbb{R}^2$ be the trajectory of point mass (in the plane).

From Newton's law: $\ddot{\mathbf{y}} = F(\mathbf{y}) := -\frac{2\mathbf{y}}{\|\mathbf{y}\|_2^2}$. (6.5.3.8)

As in Rem. 6.1.3.5 we can convert the second-order ODE (6.5.3.8) into an equivalent 1st-order ODE by introducing the **velocity** $v := \dot{y}$ as an extra solution component:

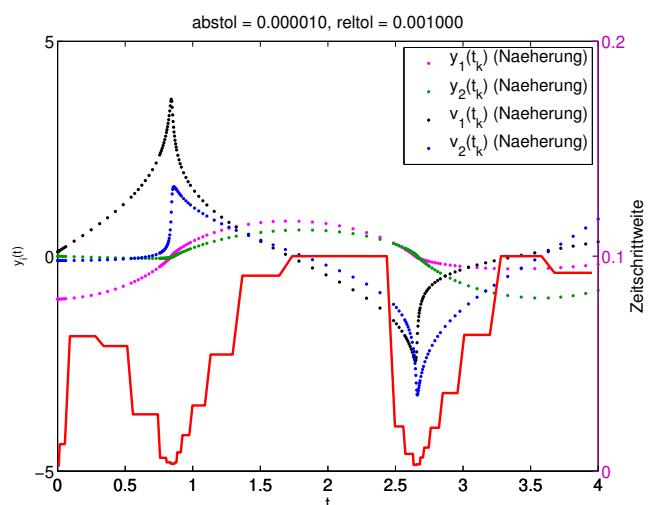
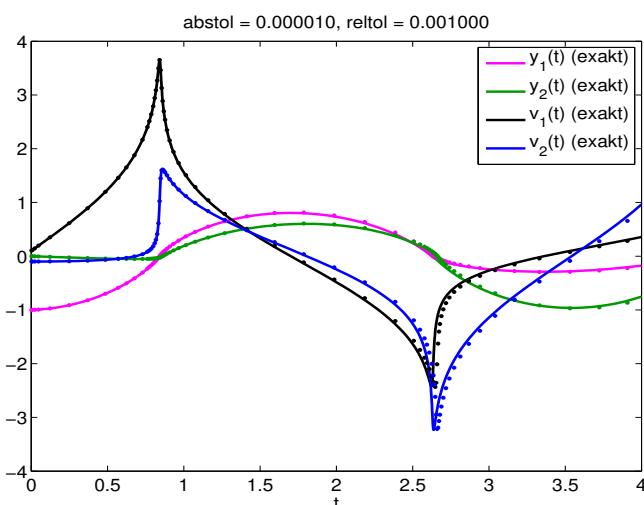
$$(6.5.3.8) \quad \Rightarrow \quad \begin{bmatrix} \dot{\mathbf{y}} \\ \dot{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ -\frac{2\mathbf{y}}{\|\mathbf{v}\|_2^2} \end{bmatrix}. \quad (6.5.3.9)$$

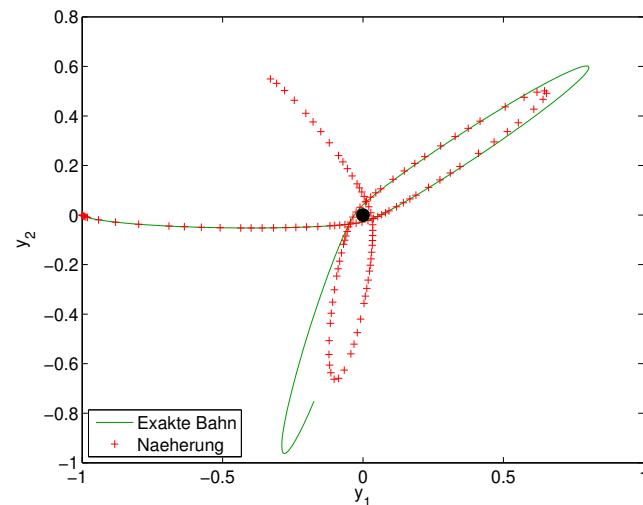
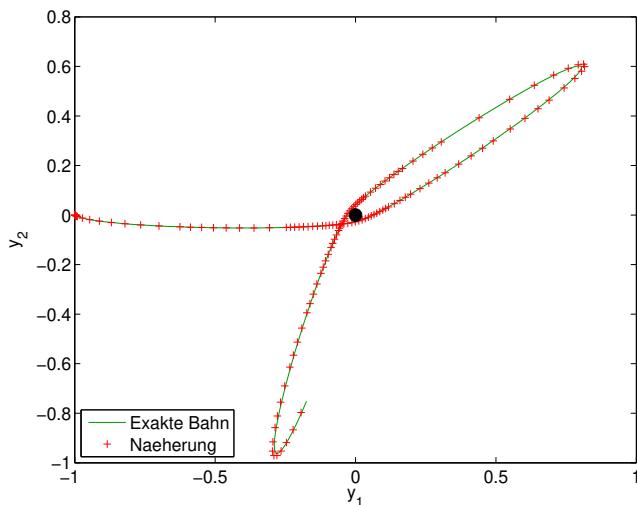
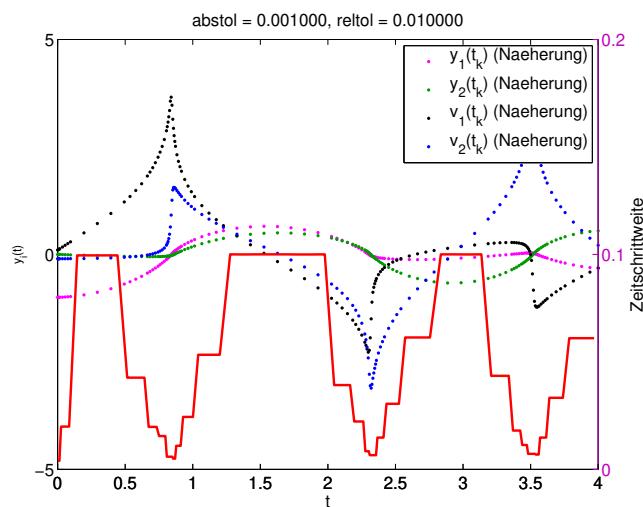
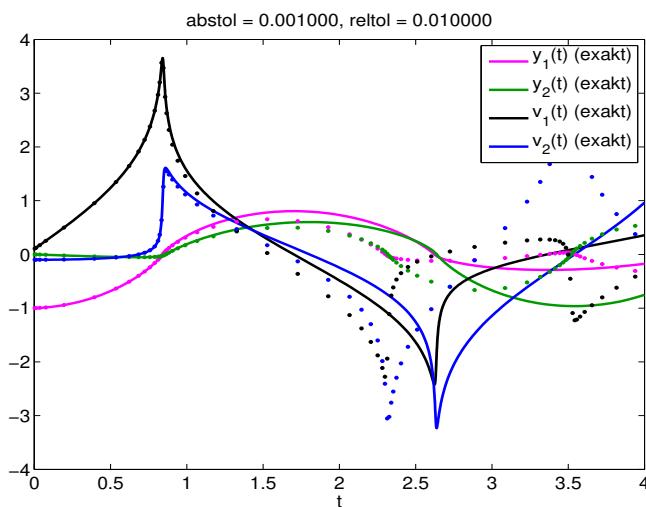
The following initial values used in the experiment:

$$\mathbf{y}(0) := \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad \mathbf{v}(0) := \begin{bmatrix} 0.1 \\ -0.1 \end{bmatrix}$$

Adaptive numerical integration with adaptive numerical integrator **Ode45** (to § 6.5.3.3) with

- ① options.rtol = 0.001,options.atol = 1.0E-5,
 - ② options.rtol = 0.01,options.atol = 1.0E-3,





Observations:

- Fast changes in solution components captured by adaptive approach through very small timesteps.
- Completely wrong solution, if tolerance reduced slightly.

In this example we face a rather **sensitive dependence** of the trajectories on initial states or intermediate states. Small perturbations at one instance in time can have a massive impact on the solution at later times. Local stepsize control is powerless about preventing this. □

Review question(s) 6.5.3.10 (Adaptive timestep control)

(Q6.5.3.10.A) Explain how the blow-up of solutions of an initial-value problem can be captured by a single-step numerical integrator with adaptive stepsize control.

(Q6.5.3.10.B) Code 6.5.2.18 contains the line

```
h = h * std::max(0.5, std::min(2.0, std::pow(tol/est, 1.0/(p+1))));
```

Explain all parts and variables occurring in this expression.

△

Bibliography

- [Ama83] H. Amann. *Gewöhnliche Differentialgleichungen*. 1st. Berlin: Walter de Gruyter, 1983 (cit. on pp. 440, 447).
- [DR08] W. Dahmen and A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. Heidelberg: Springer, 2008 (cit. on pp. 440, 444, 445, 447, 448, 454, 465, 472–474, 476–478, 486).
- [Dea80] M.A.B. Deakin. “Applied catastrophe theory in the social and biological sciences”. In: *Bulletin of Mathematical Biology* 42.5 (1980), pp. 647–679 (cit. on p. 441).
- [DB02] P. Deuflhard and F. Bornemann. *Scientific Computing with Ordinary Differential Equations*. 2nd ed. Vol. 42. Texts in Applied Mathematics. New York: Springer, 2002 (cit. on pp. 437, 448, 468, 469, 477, 484).
- [Gra02] C.R. Gray. “An analysis of the Belousov-Zhabotinski reaction”. In: *Rose-Hulman Undergraduate Math Journal* 3.1 (2002) (cit. on p. 479).
- [HLW06] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration*. 2nd ed. Vol. 31. Springer Series in Computational Mathematics. Heidelberg: Springer, 2006 (cit. on pp. 437, 440, 477).
- [HNW93] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems*. 2nd ed. Berlin, Heidelberg, New York: Springer-Verlag, 1993 (cit. on p. 437).
- [HW11] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*. Vol. 14. Springer Series in Computational Mathematics. Berlin: Springer-Verlag, 2011 (cit. on p. 437).
- [Han02] M. Hanke-Bourgeois. *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*. Mathematische Leitfäden. Stuttgart: B.G. Teubner, 2002 (cit. on pp. 440, 443, 447, 454, 466, 478, 479).
- [Het00] Herbert W. Hethcote. “The mathematics of infectious diseases”. In: *SIAM Rev.* 42.4 (2000), pp. 599–653. DOI: [10.1137/S0036144500371907](https://doi.org/10.1137/S0036144500371907) (cit. on p. 442).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 443, 446, 452, 455, 457, 461, 463, 465, 473–477, 479, 482, 484, 486, 490).
- [NS02] K. Nipp and D. Stoffer. *Lineare Algebra*. 5th ed. Zürich: vdf Hochschulverlag, 2002 (cit. on p. 439).
- [QSS00] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*. Vol. 37. Texts in Applied Mathematics. New York: Springer, 2000 (cit. on pp. 448, 459, 478).
- [Str09] M. Struwe. *Analysis für Informatiker*. Lecture notes, ETH Zürich. 2009 (cit. on pp. 437, 439, 440, 444, 446, 447, 456, 467).

Chapter 7

Single-Step Methods for Stiff Initial-Value Problems

Explicit Runge-Kutta methods with stepsize control (\rightarrow Section 6.5) seem to be able to provide approximate solutions for any IVP with good accuracy provided that tolerances are set appropriately. Does this mean that everything is settled about numerical integration?

EXAMPLE 7.0.0.1 (Explicit adaptive RK-SSM for stiff IVP) In this example we will witness the near failure of a high-order adaptive explicit Runge-Kutta method for a simple scalar autonomous ODE.

$$\text{IVP considered: } \dot{y} = \lambda y^2(1 - y), \quad \lambda := 500, \quad y(0) = \frac{1}{100}. \quad (7.0.0.2)$$

This is a logistic ODE as introduced in Ex. 6.1.2.1. We try to solve it by means of an explicit adaptive embedded Runge-Kutta-Fehlberg method (\rightarrow Section 6.5.3) using the embedded Runge-Kutta single-step method offered by **Ode45** as explained in § 6.5.3.3 (Preprocessor switch MATLABCOEFF activated).

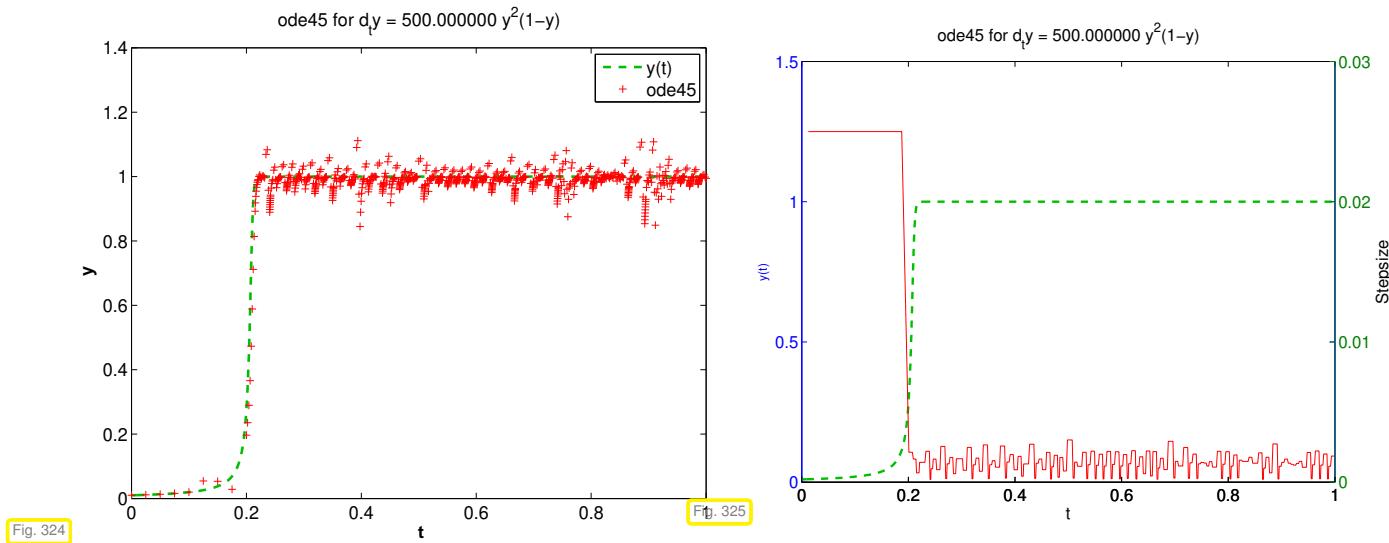
C++11 code 7.0.0.3: Solving (7.0.0.2) with **Ode45** numerical integrator [→ GITHUB](#)

```
2 // Types to be used for a scalar ODE with state space ℝ
3 using StateType = double;
4 using RhsType = std::function<StateType(StateType)>;
5 // Logistic differential equation (6.1.2.2)
6 double lambda = 500.0;
7 RhsType f = [lambda](StateType y) { return lambda * y * y * (1 - y); };
8 StateType y0 = 0.01; // Initial value, will create a STIFF IVP
9 // State space ℝ, simple modulus supplies norm
10 auto normFunc = [] (StateType x) { return fabs(x); };

11
12 // Invoke explicit Runge-Kutta method with stepsize control
13 Ode45<StateType, RhsType> integrator(f);
14 // Set rather loose tolerances
15 integrator.options.rtol = 0.1;
16 integrator.options.atol = 0.001;
17 integrator.options.min_dt = 1E-18;
18 std::vector<std::pair<StateType, double>> states =
19     integrator.solve(y0, 1.0, normFunc);
20 // Output information accumulation during numerical integration
21 integrator.options.do_statistics = true;
22 integrator.print();
```

Statistics of the integrator run▷

¹	- number of steps:	183
²	- number of rejected steps:	185
³	- function calls:	1302

Fig. 324
Fig. 325
Stepsize control of **Ode45** running amok!

❓ The solution is virtually constant from $t > 0.2$ and, nevertheless, the integrator uses tiny timesteps until the end of the integration interval. Why this crazy behavior? ↴

Contents

7.1	Model Problem Analysis	496
7.2	Stiff Initial-Value Problems	509
7.3	Implicit Runge-Kutta Single-Step Methods	514
7.3.1	The Implicit Euler Method for Stiff IVPs	515
7.3.2	Collocation Single-Step Methods	516
7.3.3	General Implicit Runge-Kutta Single-Step Methods (RK-SSMs)	520
7.3.4	Model Problem Analysis for Implicit Runge-Kutta Single-Step Methods (IRK-SSMs)	522
7.4	Semi-Implicit Runge-Kutta Methods	528
7.5	Splitting Methods	532



Supplementary literature. [DR08, Sect. 11.9]

7.1 Model Problem Analysis



Video tutorial for Section 7.1: Model Problem Analysis: (68 minutes) [Download link](#), [tablet notes](#)

Fortunately, full insight into the observations made in Ex. 7.0.0.1 can already be gleaned from a scalar linear model problem that is extremely easy to analyze.

EXPERIMENT 7.1.0.1 (Adaptive explicit RK-SSM for scalar linear decay ODE) To rule out that what we observed in Ex. 7.0.0.1 might have been a quirk of the IVP (7.0.0.2) we conduct the same investigations for the simple **linear, scalar ($N = 1$), autonomous** IVP

$$\dot{y} = \lambda y \quad , \quad \lambda := -80 \quad , \quad y(0) = 1 . \quad (7.1.0.2)$$

We use the adaptive integrator of **Ode45** (\rightarrow § 6.5.3.3) to solve (7.1.0.2) with the same parameters as in Code 6.5.3.4. \triangleright

1	- number of steps:	33
2	- number of rejected steps:	32
3	- function calls:	231

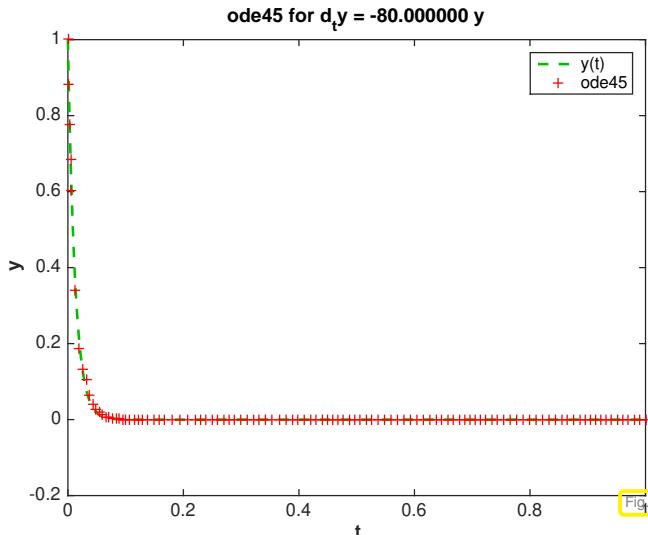
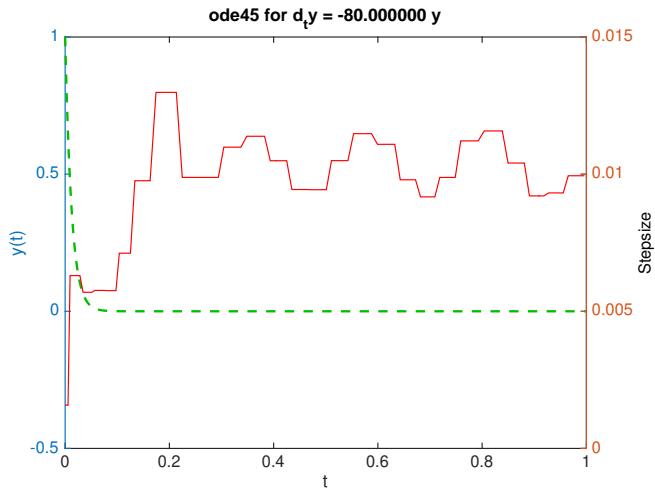


Fig.



Observation: Though $y(t) \approx 0$ for $t > 0.1$, the integrator keeps on using “unreasonably small” timesteps even then. \square

In this section we will discover a simple explanation for the startling behavior of the adaptive timestepping **Ode45** in Ex. 7.0.0.1.

EXAMPLE 7.1.0.3 (Blow-up of explicit Euler method) The simplest explicit RK-SSM is the explicit Euler method, see Section 6.2.1. We know that it should converge like $\mathcal{O}(h)$ for meshwidth $h \rightarrow 0$. In this example we will see that this may be true only for sufficiently small h , which may be extremely small.

- ◆ We consider the IVP for the scalar linear decay ODE:

$$\dot{y} = f(y) := \lambda y, \quad \lambda \ll 0, \quad y(0) = 1.$$

- ◆ We apply the explicit Euler method (6.2.1.4) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.

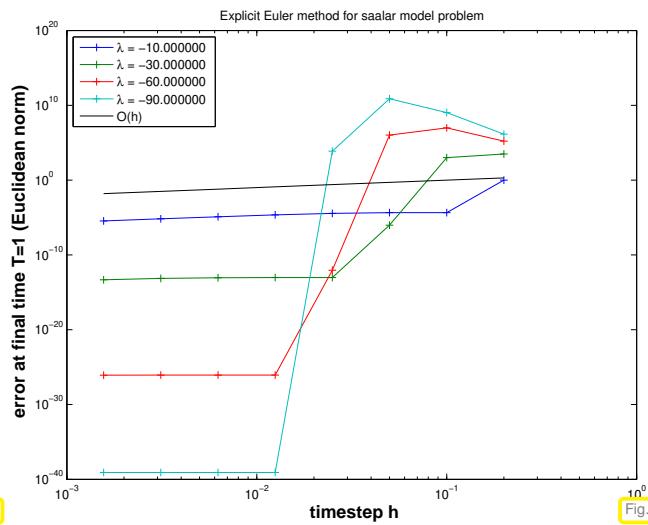
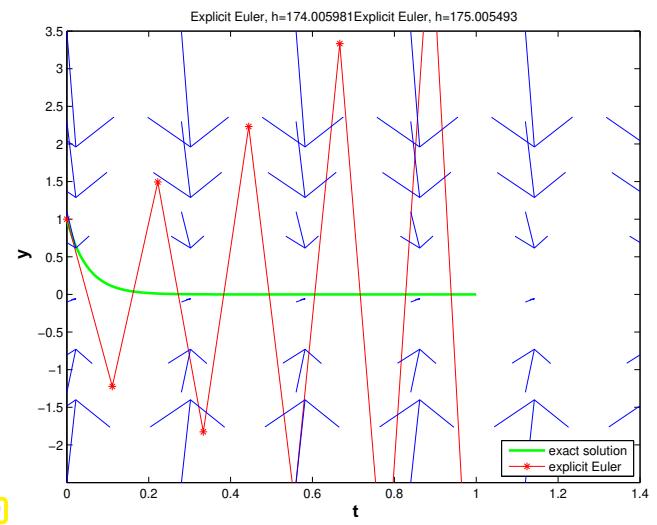


Fig. 328



$\lambda \ll 0$: blow-up of y_k for large timestep h

$\lambda = 20$: $\textcolor{green}{—} \hat{=} y(t)$, $\textcolor{red}{—} \hat{=} \text{Euler polygon}$

Explanation: From Fig. 330 we draw the geometric conclusion that, if h is “large in comparison with λ^{-1} ”, then the approximations y_k may miss the stationary point $y = 0$ due to overshooting.

This leads to a sequence $(y_k)_k$ with exponentially increasing oscillations.

- Now we look at an IVP for the logistic ODE, see Ex. 6.1.2.1:

$$\dot{y} = f(y) := \lambda y(1 - y) , \quad y(0) = 0.01 .$$

- As before, we apply the explicit Euler method (6.2.1.4) with uniform timestep $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.

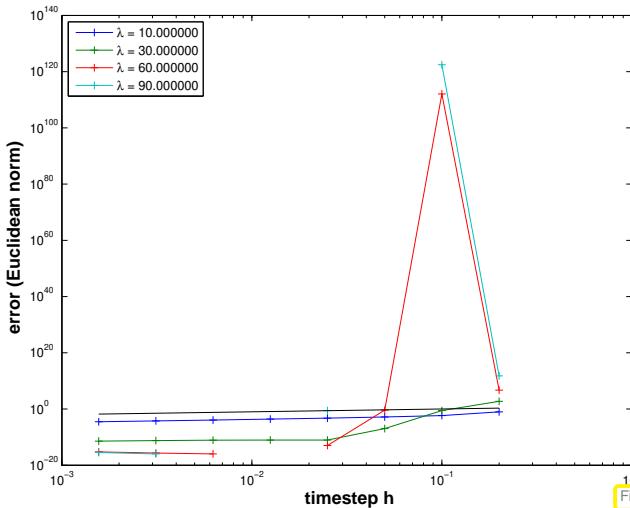


Fig. 330

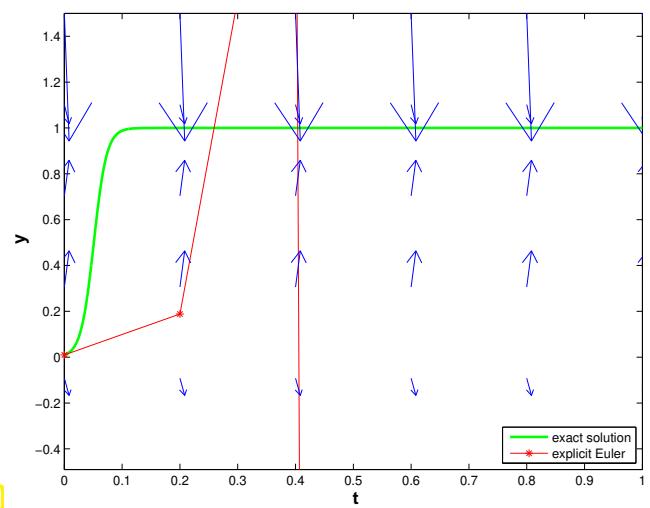


Fig. 331

λ large: blow-up of y_k for large timestep h

$\lambda = 90$: $\text{---} \hat{=} y(t)$, $\text{—} \hat{=} \text{Euler polygon}$

For large timesteps h we also observe oscillatory blow-up of the sequence $(y_k)_k$.

Deeper analysis:

For $y \approx 1$: $f(y) \approx \lambda(1 - y) \Rightarrow$ If $y(t_0) \approx 1$, then the solution of the IVP will behave like the solution of $\dot{y} = \lambda(1 - y)$, which is a linear ODE. Similarly, $z(t) := 1 - y(t)$ will behave like the solution of the “decay equation” $\dot{z} = -\lambda z$. Thus, around the stationary point $y = 1$ the explicit Euler method behaves like it did for $\dot{y} = \lambda y$ in the vicinity of the stationary point $y = 0$; it grossly overshoots. \square

§7.1.0.4 (Linear model problem analysis: explicit Euler method) The phenomenon observed in the two previous examples is accessible to a remarkably simple rigorous analysis: Motivated by the considerations in Ex. 7.1.0.3 we study the explicit Euler method (6.2.1.4) for the

$$\text{linear model problem: } \dot{y} = \lambda y , \quad y(0) = y_0 , \quad \text{with } \lambda \ll 0 , \quad (7.1.0.5)$$

which has an *exponentially decaying* exact solution

$$y(t) = y_0 \exp(\lambda t) \rightarrow 0 \quad \text{for } t \rightarrow \infty .$$

Recall the recursion for the explicit Euler with uniform timestep $h > 0$ method for (7.1.0.5):

$$(6.2.1.4) \text{ for } f(y) = \lambda y: \quad y_{k+1} = y_k(1 + \lambda h) . \quad (7.1.0.6)$$

We easily get a closed form expression for the approximations y_k :

$$\blacktriangleright \quad y_k = y_0(1 + \lambda h)^k \Rightarrow |y_k| \rightarrow \begin{cases} 0 & , \text{if } \lambda h > -2 \quad (\text{qualitatively correct}) , \\ \infty & , \text{if } \lambda h < -2 \quad (\text{qualitatively wrong}) . \end{cases}$$

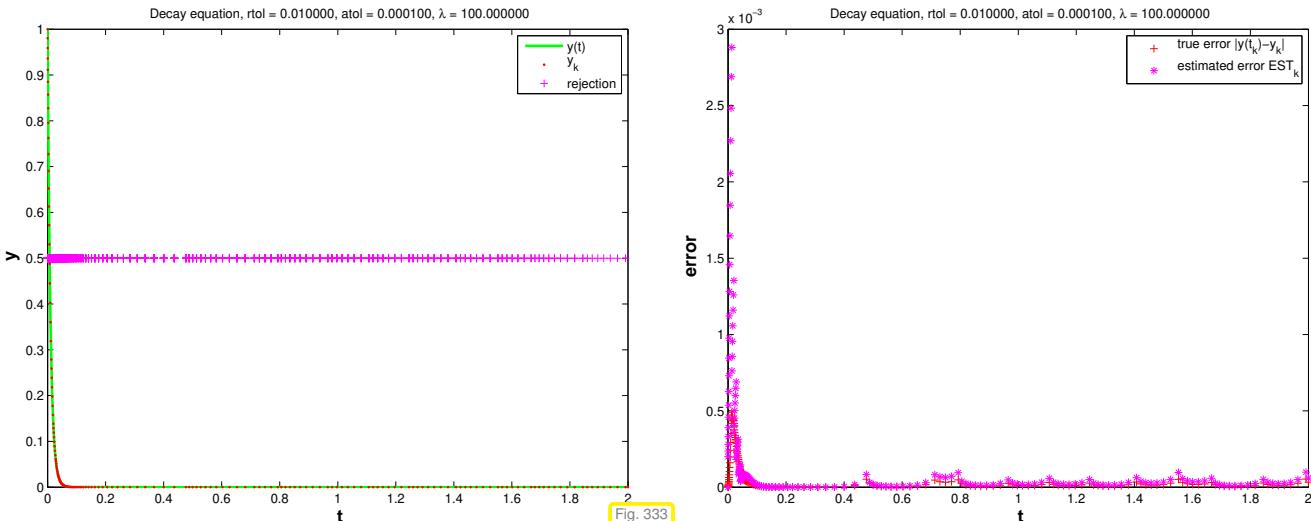
Observed: stability-induced timestep constraint

Only if $|\lambda|h < 2$ we obtain a decaying solution by the explicit Euler method!

Could it be that the timestep control is desperately trying to enforce the qualitatively correct behavior of the numerical solution in Ex. 7.1.0.3? Let us examine how the simple stepsize control of Code 6.5.2.6 fares for model problem (7.1.0.5):

EXPERIMENT 7.1.0.8 (Simple adaptive timestepping for fast decay) In this example we let a transparent adaptive timestep struggle with “overshooting”:

- ◆ “Linear model problem IVP”: $\dot{y} = \lambda y$, $y(0) = 1$, $\lambda = -100$
- ◆ Simple adaptive timestepping method as in Exp. 6.5.2.8, see Code 6.5.2.6. Timestep control based on the pair of 1st-order explicit Euler method and 2nd-order explicit trapezoidal method.



Observation: in fact, stepsize control enforces small timesteps even if $y(t) \approx 0$ and persistently triggers rejections of timesteps. This is necessary to prevent overshooting in the Euler method, which contributes to the estimate of the one-step error.

We see the purpose of *stepsize control thwarted*, because after only a very short time the solution is almost zero and then, in fact, large timesteps should be chosen.

Are these observations a particular “flaw” of the explicit Euler method? Let us study the behavior of another simple explicit Runge-Kutta method applied to the linear model problem.

EXAMPLE 7.1.0.9 (Explicit trapzoidal method for decay equation → [DR08, Ex. 11.29])

The explicit trapezoidal method is a 2-stage explicit Runge-Kutta method, whose Butcher scheme is given in Ex. 6.4.0.15 and which was derived in Ex. 6.4.0.4. We state its recursion for the ODE $\dot{y} = f(t, y)$ in terms of the first step $y_0 \rightarrow y_1$:

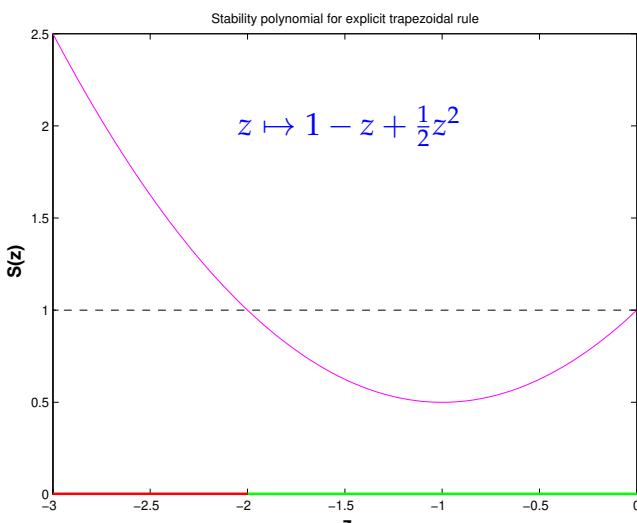
$$\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{y}_0), \quad \mathbf{k}_2 = \mathbf{f}(t_0 + h, \mathbf{y}_0 + h\mathbf{k}_1), \quad \mathbf{y}_1 = \mathbf{y}_0 + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2). \quad (6.4.0.6)$$

Apply it to the model problem (7.1.0.5), that is, the scalar autonomous ODE with right hand side function $f(y) = f(y) = \lambda y$, $\lambda < 0$:

$$\mathbf{k}_1 = \lambda \mathbf{y}_0, \quad \mathbf{k}_2 = \lambda (\mathbf{y}_0 + h\mathbf{k}_1) \Rightarrow \mathbf{y}_1 = \underbrace{(1 + \lambda h + \frac{1}{2}(\lambda h)^2)}_{=:S(h\lambda)} \mathbf{y}_0. \quad (7.1.0.10)$$

- The sequence of approximations generated by the explicit trapezoidal rule can be expressed in closed form as

$$y_k = S(h\lambda)^k y_0, \quad k = 0, \dots, N. \quad (7.1.0.11)$$



Clearly, blow-up can be avoided only if $|S(h\lambda)| \leq 1$:

$$|S(h\lambda)| < 1 \Leftrightarrow -2 < h\lambda < 0.$$

Qualitatively correct decay behavior of $(y_k)_k$ only under **timestep constraint**

$$h \leq |2/\lambda|. \quad (7.1.0.12)$$

◀ the **stability function** for the explicit trapezoidal method

Fig. 334

§7.1.0.13 (Model problem analysis for general explicit Runge-Kutta single step methods) We generalize the approach taken in Ex. 7.1.0.9 and apply an explicit s -stage Runge-Kutta method (\rightarrow Def. 6.4.0.9) encoded by the Butcher scheme $\begin{array}{c|cc} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$, $\mathfrak{A} \in \mathbb{R}^{s,s}$ strictly lower-triangular, to the autonomous scalar linear ODE (7.1.0.5) ($\dot{y} = \lambda y$). We write down the equations for the increments and y_1 from Def. 6.4.0.9 for $f(y) := \lambda y$ and then convert the resulting system of equations into matrix form:

$$\begin{aligned} \blacktriangleright \quad k_i &= \lambda(y_0 + h \sum_{j=1}^{i-1} a_{ij} k_j), \quad \Rightarrow \quad \begin{bmatrix} \mathbf{I} - z\mathfrak{A} & \mathbf{0} \\ -z\mathbf{b}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{k} \\ y_1 \end{bmatrix} = y_0 \begin{bmatrix} \mathbf{1} \\ 1 \end{bmatrix}, \\ y_1 &= y_0 + h \sum_{i=1}^s b_i k_i \end{aligned} \quad (7.1.0.14)$$

where $\mathbf{k} \in \mathbb{R}^s \doteq$ denotes the vector $[k_1, \dots, k_s]^\top / \lambda$ of increments, and $z := \lambda h$. Next we apply block Gaussian elimination (\rightarrow [Hip19, ??]) to solve for y_1 and obtain

$$y_1 = S(z)y_0 \quad \text{with} \quad S(z) := 1 + z\mathbf{b}^T(\mathbf{I} - z\mathfrak{A})^{-1}\mathbf{1}. \quad (7.1.0.15)$$

Alternatively we can express y_1 through determinants appealing to Cramer's rule,

$$y_1 = y_0 \frac{\det \begin{bmatrix} \mathbf{I} - z\mathfrak{A} & \mathbf{1} \\ -z\mathbf{b}^T & 1 \end{bmatrix}}{\det \begin{bmatrix} \mathbf{I} - z\mathfrak{A} & \mathbf{0} \\ -z\mathbf{b}^T & 1 \end{bmatrix}} \quad \Rightarrow \quad S(z) = \det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T), \quad (7.1.0.16)$$

and note that \mathfrak{A} is a strictly lower triangular matrix, which means that $\det(\mathbf{I} - z\mathfrak{A}) = 1$. Thus we have proved the following theorem.

Theorem 7.1.0.17. Stability function of some explicit Runge-Kutta methods → [Han02, Thm. 77.2], [QSS00, Sect. 11.8.4]

The discrete evolution Ψ_λ^h of an explicit s -stage Runge-Kutta single step method (→ Def. 6.4.0.9) with Butcher scheme $\begin{array}{c|cc} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array}$ (see (6.4.0.11)) for the ODE $\dot{\mathbf{y}} = \lambda \mathbf{y}$ amounts to a multiplication with the number

$$\Psi_\lambda^h = S(\lambda h) \Leftrightarrow y_1 = S(\lambda h)y_0,$$

where S is the **stability function** (SF)

$$S(z) := 1 + z \mathbf{b}^T (\mathbf{I} - z \mathbf{A})^{-1} \mathbf{1} = \det(\mathbf{I} - z \mathbf{A} + z \mathbf{1} \mathbf{b}^T), \quad \mathbf{1} := [1, \dots, 1]^\top \in \mathbb{R}^s. \quad (7.1.0.18)$$

EXAMPLE 7.1.0.19 (Stability functions of explicit Runge-Kutta single step methods) From Thm. 7.1.0.17 and their Butcher schemes we can instantly compute the stability functions of explicit RK-SSM. We do this for a few methods whose Butcher schemes were listed in Ex. 6.4.0.15

- Explicit Euler method (6.2.1.4):

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \Rightarrow S(z) = 1 + z.$$

- Expl. trapezoidal method (6.4.0.6):

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \Rightarrow S(z) = 1 + z + \frac{1}{2}z^2.$$

- Classical RK4 method:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array} \Rightarrow S(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4.$$

These examples confirm an immediate consequence of the determinant formula for the stability function $S(z)$.

Corollary 7.1.0.20. Polynomial stability function of explicit RK-SSM

For a consistent (→ Def. 6.3.1.11) s -stage explicit Runge-Kutta single step method according to Def. 6.4.0.9 the stability function S defined by (7.1.0.18) is a non-constant **polynomial** of degree $\leq s$, that is, $S \in \mathcal{P}_s$.

Remark 7.1.0.21 (Stability function and exponential function) Let us compare the two evolution operators:

- $\Phi \doteq$ evolution operator (→ Def. 6.1.4.3) for $\dot{\mathbf{y}} = \lambda \mathbf{y}$,

- $\Psi \hat{=} \text{discrete evolution operator} (\rightarrow \S\ 6.3.1.1)$ for an s -stage Runge-Kutta single step method.

$$\Phi^h y = e^{\lambda h} y \longleftrightarrow \Psi^h y = S(\lambda h) y .$$

In light of that Ψ is supposed to be an approximation for Φ , $\Psi \approx \Phi$, see (6.3.1.3), we expect that

$$S(z) \approx \exp(z) \quad \text{for small } |z| . \quad (7.1.0.22)$$

A more precise statement is made by the following lemma:

Lemma 7.1.0.23. Stability function as approximation of exp for small arguments

Let S denote the stability function of an s -stage explicit Runge-Kutta single step method of order $q \in \mathbb{N}$. Then

$$|S(z) - \exp(z)| = O(|z|^{q+1}) \quad \text{for } |z| \rightarrow 0 . \quad (7.1.0.24)$$

This means that the lowest $q + 1$ coefficients of $S(z)$ must be equal to the first coefficients of the exponential series:

$$S(z) = \sum_{j=0}^q \frac{1}{j!} z^j + z^{q+1} p(z) \quad \text{with some } p \in \mathcal{P}_{s-q-1} .$$

In order to match the first q terms of the exponential series, we need at least $S \in \mathcal{P}_q$, which entails a minimum of q stages.

Corollary 7.1.0.25. Stages limit order of explicit RK-SSM

An explicit s -stage RK-SSM has maximal order $q \leq s$.

§7.1.0.26 (Stability induced timestep constraint) In § 7.1.0.13 we established that for the sequence $(y_k)_{k=0}^\infty$ produced by an explicit Runge-Kutta single step method applied to the linear scalar model ODE $\dot{y} = \lambda y$, $\lambda \in \mathbb{R}$, with uniform timestep $h > 0$ holds

$$y_{k+1} = S(\lambda h) y_k \Rightarrow y_k = S(\lambda h^k) y_0 .$$



$$\begin{array}{lll} (y_k)_{k=0}^\infty \text{ non-increasing} & \Leftrightarrow & |S(\lambda h)| \leq 1 , \\ (y_k)_{k=0}^\infty \text{ exponentially increasing} & \Leftrightarrow & |S(\lambda h)| > 1 . \end{array} \quad (7.1.0.27)$$

where $S = S(z)$ is the stability function of the RK-SSM as defined in (7.1.0.18).

Invariably polynomials tend to $\pm\infty$ for large (in modulus) arguments:

$$\forall S \in \mathcal{P}_s, S \neq \text{const} : \lim_{|z| \rightarrow \infty} S(z) = \infty \quad \text{uniformly} . \quad (7.1.0.28)$$

So, for any $\lambda \neq 0$ there will be a threshold $h_{\max} > 0$ so that $|y_k| \rightarrow \infty$ as $|h| > h_{\max}$.

Reversing the argument we arrive at a **timestep constraint**, as already observed for the explicit Euler methods in § 7.1.0.4.

Only if one ensures that $|\lambda h|$ is sufficiently small, one can avoid exponentially increasing approximations \mathbf{y}_k (qualitatively wrong for $\lambda < 0$) when applying an explicit RK-SSM to the model problem (7.1.0.5) with uniform timestep $h > 0$,

For $\lambda \ll 0$ this stability induced timestep constraint may force h to be much *smaller than required by demands on accuracy*: in this case timestepping becomes **inefficient**. \square

Remark 7.1.0.29 (Stepsize control detects instability) Ex. 7.0.0.1, Exp. 7.1.0.8 send the message that local-in-time stepsize control as discussed in Section 6.5 selects timesteps that avoid blow-up, with a hefty price tag however in terms of computational cost and poor accuracy. \square

Objection: simple linear scalar IVP (7.1.0.5) may be an oddity rather than a model problem: the weakness of explicit Runge-Kutta methods discussed above may be just a peculiar response to an unusual situation. Let us extend our investigations to **systems of linear ODEs**, $N > 1$.

§7.1.0.30 (Systems of linear ordinary differential equations, § 6.1.1.8 revisited) A generic linear ordinary differential equation with constant coefficients on the state space \mathbb{R}^N has the form

$$\dot{\mathbf{y}} = \mathbf{My} \quad \text{with a matrix } \mathbf{M} \in \mathbb{R}^{N,N}. \quad (7.1.0.31)$$

As explained in [NS02, Sect. 8.1], (7.1.0.31) can be solved by **diagonalization**: If we can find a *regular* matrix $\mathbf{V} \in \mathbb{C}^{N,N}$ such that

$$\mathbf{MV} = \mathbf{VD} \quad \text{with diagonal matrix } \mathbf{D} = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{bmatrix} \in \mathbb{C}^{N,N}, \quad (7.1.0.32)$$

then the 1-parameter family of global solutions of (7.1.0.31) is given by

$$\mathbf{y}(t) = \mathbf{V} \begin{bmatrix} \exp(\lambda_1 t) & & 0 \\ & \ddots & \\ 0 & & \exp(\lambda_N t) \end{bmatrix} \mathbf{V}^{-1} \mathbf{y}_0, \quad \mathbf{y}_0 \in \mathbb{R}^N. \quad (7.1.0.33)$$

The columns of \mathbf{V} are a basis of **eigenvectors** of \mathbf{M} , the $\lambda_j \in \mathbb{C}$, $j = 1, \dots, N$ are the associated **eigenvalues** of \mathbf{M} , see [Hip19, ??].

The idea behind diagonalization is the transformation of (7.1.0.31) into N *decoupled* scalar linear ODEs:

$$\dot{\mathbf{y}} = \mathbf{My} \xrightarrow{\mathbf{z}(t) := \mathbf{V}^{-1}\mathbf{y}(t)} \dot{\mathbf{z}} = \mathbf{Dz} \leftrightarrow \begin{array}{l} \dot{z}_1 = \lambda_1 z_1 \\ \vdots \\ \dot{z}_N = \lambda_N z_N \end{array}, \quad \text{since } \mathbf{M} = \mathbf{VDV}^{-1}.$$

The formula (7.1.0.33) can be generalized to

$$\mathbf{y}(t) = \exp(\mathbf{Mt})\mathbf{y}_0 \quad \text{with the matrix exponential} \quad \exp(\mathbf{B}) := \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{B}^k, \quad \mathbf{B} \in \mathbb{C}^{N,N}. \quad (7.1.0.34)$$

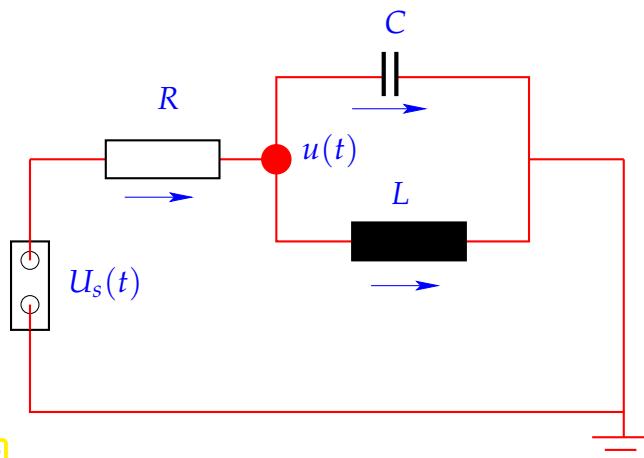
EXAMPLE 7.1.0.35 (Transient simulation of RLC-circuit) This example demonstrates the diagonalization of a linear system of ODEs.

Consider circuit from Ex. 6.1.2.11

Transient nodal analysis leads to the second-order linear ODE

$$\ddot{u} + \alpha \dot{u} + \beta u = g(t),$$

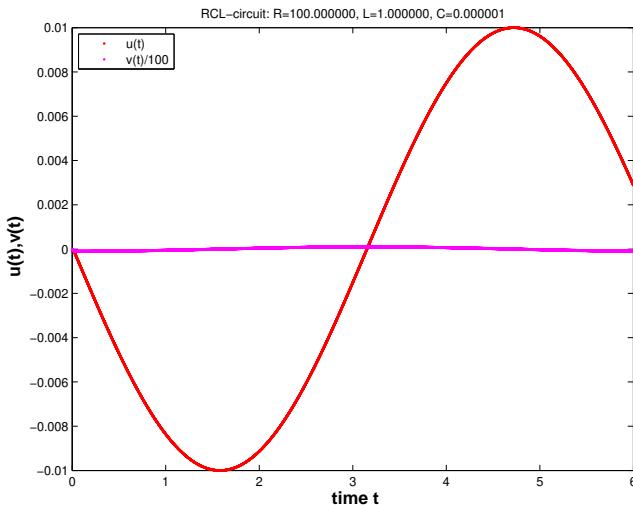
with coefficients $\alpha := (RC)^{-1}$, $\beta = (LC)^{-1}$, $g(t) = \alpha \dot{U}_s(t)$.



We transform it to a linear 1st-order ODE as in Rem. 6.1.3.5 by introducing $v := \dot{u}$ as additional solution component:

$$\underbrace{\begin{bmatrix} \dot{u} \\ v \end{bmatrix}}_{=: \dot{y}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\beta & -\alpha \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} 0 \\ g(t) \end{bmatrix}}_{=: f(t,y)}, \quad \text{with } \beta \gg \alpha \gg 1 \text{ in usual settings.}$$

We integrate IVPs for this ODE by means of the adaptive integrator **Ode45** from § 6.5.3.3.



$R = 100\Omega$, $L = 1\text{H}$, $C = 1\mu\text{F}$, $U_s(t) = 1\text{V} \sin(t)$, $u(0) = v(0) = 0$ ("switch on")

Ode45 statistics:

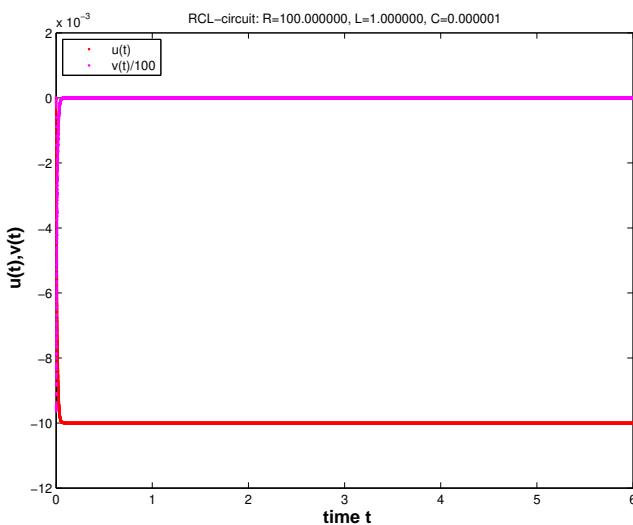
17897 successful steps

1090 failed attempts

113923 function evaluations

Inefficient: way more timesteps than required for resolving smooth solution, cf. remark in the end of § 9.3.5.6.

Maybe the time-dependent right hand side due to the time-harmonic excitation severely affects **ode45**? Let us try a constant exciting voltage:



$R = 100\Omega$, $L = 1\text{H}$, $C = 1\mu\text{F}$, $U_s(t) = 1\text{V}$, $u(0) = v(0) = 0$ ("switch on")

Ode45 statistics:

17901 successful steps

1210 failed attempts

114667 function evaluations

Tiny timesteps despite virtually constant solution!

We make the same observation as in Ex. 7.0.0.1, Exp. 7.1.0.8: the local-in-time stepsize control of **ode45**

(→ Section 6.5) enforces extremely small timesteps though the solution almost constant except at $t = 0$.

To understand the structure of the solutions for this transient circuit example, let us apply the diagonalization technique from § 7.1.0.30 to the linear ODE

$$\dot{\mathbf{y}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\beta & -\alpha \end{bmatrix}}_{=: \mathbf{M}} \mathbf{y} , \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{R}^2 . \quad (7.1.0.36)$$

Above we face the situation $\beta \gg \frac{1}{4}\alpha^2 \gg 1$.

We can obtain the general solution of $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, $\mathbf{M} \in \mathbb{R}^{2,2}$, by diagonalization of \mathbf{M} (if possible):

$$\mathbf{MV} = \mathbf{M}(\mathbf{v}_1, \mathbf{v}_2) = (\mathbf{v}_1, \mathbf{v}_2) \begin{bmatrix} \lambda_1 & \\ & \lambda_2 \end{bmatrix} . \quad (7.1.0.37)$$

where $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^2 \setminus \{0\}$ are the eigenvectors of \mathbf{M} , λ_1, λ_2 are the eigenvalues of \mathbf{M} , see [Hip19, ??]. The latter are the roots of the characteristic polynomial $t \mapsto \chi(t) := t^2 + \alpha t + \beta$ in \mathbb{C} , and we find

$$\lambda_{1/2} = \frac{1}{2}(-\alpha \pm D), \quad D := \begin{cases} \sqrt{\alpha^2 - 4\beta} & , \text{ if } \alpha^2 \geq 4\beta , \\ i\sqrt{4\beta - \alpha^2} & , \text{ if } \alpha^2 < 4\beta . \end{cases}$$

Note that the eigenvalues have a large (in modulus) negative real part and a non-vanishing imaginary part in the setting of the experiment.

Then we transform $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$ into *decoupled* scalar linear ODEs:

$$\dot{\mathbf{y}} = \mathbf{M}\mathbf{y} \Leftrightarrow \mathbf{V}^{-1}\dot{\mathbf{y}} = \mathbf{V}^{-1}\mathbf{M}\mathbf{V}(\mathbf{V}^{-1}\mathbf{y}) \stackrel{\mathbf{z}(t) := \mathbf{V}^{-1}\mathbf{y}(t)}{\Leftrightarrow} \dot{\mathbf{z}} = \begin{bmatrix} \lambda_1 & \\ & \lambda_2 \end{bmatrix} \mathbf{z} . \quad (7.1.0.38)$$

This yields the general solution of the ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, see also [Str09, Sect. 5.6]:

$$\mathbf{y}(t) = A\mathbf{v}_1 \exp(\lambda_1 t) + B\mathbf{v}_2 \exp(\lambda_2 t) , \quad A, B \in \mathbb{R} . \quad (7.1.0.39)$$

Note: $t \mapsto \exp(\lambda_i t)$ is general solution of the ODE $\dot{z}_i = \lambda_i z_i$. □

§7.1.0.40 (“Diagonalization” of explicit Euler method) Recall the discrete evolution of the explicit Euler method (6.2.1.4) for the linear ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, $\mathbf{M} \in \mathbb{R}^{N,N}$:

$$\Psi^h \mathbf{y} = \mathbf{y} + h\mathbf{M}\mathbf{y} \Leftrightarrow \mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{M}\mathbf{y}_k .$$

As in § 7.1.0.30 we assume that \mathbf{M} can be diagonalized, that is (7.1.0.32) holds: $\mathbf{V}^{-1}\mathbf{M}\mathbf{V} = \mathbf{D}$ with a diagonal matrix $\mathbf{D} \in \mathbb{C}^{N,N}$ containing the eigenvalues of \mathbf{M} on its diagonal. Next, apply the *decoupling by diagonalization* idea to the recursion of the explicit Euler method.

$$\mathbf{V}^{-1}\mathbf{y}_{k+1} = \mathbf{V}^{-1}\mathbf{y}_k + h\mathbf{V}^{-1}\mathbf{M}\mathbf{V}(\mathbf{V}^{-1}\mathbf{y}_k) \stackrel{\mathbf{z}_k := \mathbf{V}^{-1}\mathbf{y}_k}{\Leftrightarrow} \underbrace{(\mathbf{z}_{k+1})_i}_{\triangleq \text{explicit Euler step for } \dot{z}_i = \lambda_i z_i} = (\mathbf{z}_k)_i + h\lambda_i(\mathbf{z}_k)_i , \quad (7.1.0.41)$$

with $i \in \{1, \dots, N\}$. This gives us a crucial insight:

The explicit Euler method generates uniformly bounded solution sequences $(\mathbf{y}_k)_{k=0}^\infty$ for $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$ with diagonalizable matrix $\mathbf{M} \in \mathbb{R}^{N,N}$ with eigenvalues $\lambda_1, \dots, \lambda_N$, if and only if it generates uniformly bounded sequences for all the scalar ODEs $\dot{z} = \lambda_i z$, $i = 1, \dots, N$.

So far we conducted the model problem analysis under the premises $\lambda < 0$.

However, in Ex. 7.1.0.35 we face $\lambda_{1/2} = -\frac{1}{2}(\alpha \pm i\sqrt{4\beta - \alpha^2})$ (complex eigenvalues!). Let us now examine how the explicit Euler method and even general explicit RK-methods respond to them.

Remark 7.1.0.42 (Explicit Euler method for damped oscillations) Consider linear model IVP (7.1.0.5) for $\lambda \in \mathbb{C}$:

$$\operatorname{Re} \lambda < 0 \Rightarrow \text{exponentially decaying solution } y(t) = y_0 \exp(\lambda t),$$

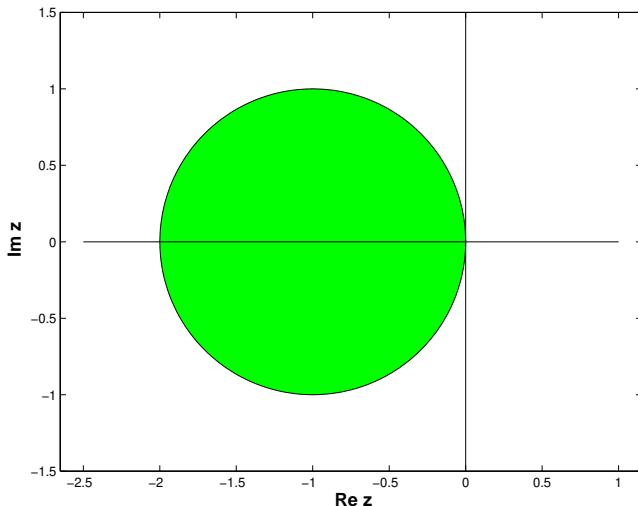
because $|\exp(\lambda t)| = \exp(\operatorname{Re} \lambda \cdot t)$.

The **model problem analysis** from Ex. 7.1.0.3, Ex. 7.1.0.9 can be extended verbatim to the case of $\lambda \in \mathbb{C}$. It yields the following insight for the explicit Euler method and $\lambda \in \mathbb{C}$:

The sequence generated by the explicit Euler method (6.2.1.4) for the model problem (7.1.0.5) satisfies

$$y_{k+1} = y_k(1 + h\lambda) \quad \blacktriangleright \quad \lim_{k \rightarrow \infty} y_k = 0 \Leftrightarrow |1 + h\lambda| < 1. \quad (7.1.0.6)$$

↑
timestep constraint to get decaying (discrete) solution !



$$\Lhd \quad \{z \in \mathbb{C}: |1 + z| < 1\}$$

The green region of the complex plane marks values for λh , for which the explicit Euler method will produce exponentially decaying solutions.

Fig. 338

Now we can conjecture what happens in Ex. 7.1.0.35: the eigenvalues $\lambda_{1/2} = -\frac{1}{2}\alpha \pm i\sqrt{\beta - \frac{1}{4}\alpha^2}$ of \mathbf{M} have a very large (in modulus) negative real part. Since the integrator of **Ode45** can be expected to behave as if it integrates $\dot{z} = \lambda_2 z$, it faces a severe timestep constraint, if exponential blow-up is to be avoided, see Ex. 7.1.0.3. Thus stepsize control must resort to tiny timesteps. \dashv

§7.1.0.43 (Extended model problem analysis for explicit Runge-Kutta single step methods) Recall the definition of a generic explicit RK-SSM for the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$:

Definition 6.4.0.9. Explicit Runge-Kutta method

For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{i-1} a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, an **s-stage explicit Runge-Kutta single step method** (RK-SSM) for the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, $\mathbf{f} : \Omega \rightarrow \mathbb{R}^N$, is defined by ($\mathbf{y}_0 \in D$)

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j), \quad i = 1, \dots, s, \quad \mathbf{y}_1 := \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i.$$

The vectors $\mathbf{k}_i \in \mathbb{R}^N$, $i = 1, \dots, s$, are called **increments**, $h > 0$ is the size of the timestep.

We apply such an explicit s -stage RK-SSM described by the Butcher scheme $\begin{array}{c|cc} \mathbf{c} & \mathbf{a} \\ \hline & \mathbf{b}^T \end{array}$ to the autonomous linear ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, $\mathbf{M} \in \mathbb{C}^{N,N}$, and obtain (for the first step with timestep size $h > 0$)

$$\mathbf{k}_\ell = \mathbf{M}(\mathbf{y}_0 + h \sum_{j=1}^{\ell-1} a_{\ell j} \mathbf{k}_j), \quad \ell = 1, \dots, s, \quad \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{\ell=1}^s b_\ell \mathbf{k}_\ell. \quad (7.1.0.44)$$

Now assume that \mathbf{M} can be diagonalized, that is (7.1.0.32) holds: $\mathbf{V}^{-1}\mathbf{M}\mathbf{V} = \mathbf{D}$ with a diagonal matrix $\mathbf{D} \in \mathbb{C}^{N,N}$ containing the eigenvalues $\lambda_1, \dots, \lambda_N \in \mathbb{C}$ of \mathbf{M} on its diagonal. Then apply the substitutions

$$\hat{\mathbf{k}}_\ell := \mathbf{V}^{-1}\mathbf{k}_\ell, \quad \ell = 1, \dots, s, \quad \hat{\mathbf{y}}_k := \mathbf{V}^{-1}\mathbf{y}_k, \quad k = 0, 1,$$

to (7.1.0.44), which yield

$$\hat{\mathbf{k}}_\ell = \mathbf{D}(\hat{\mathbf{y}}_0 + h \sum_{j=1}^{s-1} a_{\ell j} \hat{\mathbf{k}}_j), \quad \ell = 1, \dots, s, \quad \hat{\mathbf{y}}_1 = \hat{\mathbf{y}}_0 + h \sum_{\ell=1}^s b_\ell \hat{\mathbf{k}}_\ell. \quad (7.1.0.45)$$

\Updownarrow

$$(\hat{\mathbf{k}}_\ell)_i = \lambda_i((\mathbf{y}_0)_i + h \sum_{j=1}^{s-1} a_{\ell j} (\hat{\mathbf{k}}_j)_i), \quad (\hat{\mathbf{y}}_1)_i = (\hat{\mathbf{y}}_0)_i + h \sum_{\ell=1}^s b_\ell (\hat{\mathbf{k}}_\ell)_i, \quad i = 1, \dots, N. \quad (7.1.0.46)$$

We infer that, if $(\mathbf{y}_k)_k$ is the sequence produced by an explicit RK-SSM applied to $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, then

$$\mathbf{y}_k = \mathbf{V} \begin{bmatrix} y_k^{[1]} & & 0 \\ & \ddots & \\ 0 & & y_k^{[d]} \end{bmatrix} \mathbf{V}^{-1},$$

where $(y_k^{[i]})_k$ is the sequence generated by the same RK-SSM with the same sequence of timesteps for the IVP $\dot{y} = \lambda_i y$, $y(0) = (\mathbf{V}^{-1}\mathbf{y}_0)_i$.

The RK-SSM generates uniformly bounded solution sequences $(\mathbf{y}_k)_{k=0}^\infty$ for the ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$ with diagonalizable matrix $\mathbf{M} \in \mathbb{R}^{N,N}$ with eigenvalues $\lambda_1, \dots, \lambda_N$, if and only if it generates uniformly bounded sequences for all the scalar ODEs $\dot{z} = \lambda_i z$, $i = 1, \dots, N$.

Stability analysis: reduction to scalar case

Understanding the behavior of RK-SSM for autonomous scalar linear ODEs $\dot{y} = \lambda y$ with $\lambda \in \mathbb{C}$ is enough to predict their behavior for general autonomous linear systems of ODEs.

From the considerations of § 9.3.5.6 we deduce the following fundamental result.

Theorem 7.1.0.48. (Absolute) stability of explicit RK-SSM for linear systems of ODEs

The sequence $(\mathbf{y}_k)_k$ of approximations generated by an explicit RK-SSM (\rightarrow Def. 6.4.0.9) with stability function S (defined in (7.1.0.18)) applied to the linear autonomous ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, $\mathbf{M} \in \mathbb{C}^{N,N}$, with uniform timestep $h > 0$ decays exponentially for every initial state $\mathbf{y}_0 \in \mathbb{C}^N$, if and only if $|S(\lambda_i h)| < 1$ for all eigenvalues λ_i of \mathbf{M} .

Please note that

$$\operatorname{Re} \lambda_i < 0 \quad \forall i \in \{1, \dots, N\} \implies \|\mathbf{y}(t)\| \rightarrow 0 \quad \text{for } t \rightarrow \infty,$$

for any solution of $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$. This is obvious from the representation formula (7.1.0.33). \square

§7.1.0.49 (Region of (absolute) stability of explicit RK-SSM) We consider an explicit Runge-Kutta single step method with stability function S for the model linear scalar IVP $\dot{y} = \lambda y, y(0) = y_0, \lambda \in \mathbb{C}$. From Thm. 7.1.0.17 we learn that for uniform stepsize $h > 0$ we have $y_k = S(\lambda h)^k y_0$ and conclude that

$$y_k \rightarrow 0 \quad \text{for } k \rightarrow \infty \Leftrightarrow |S(\lambda h)| < 1. \quad (7.1.0.50)$$

Hence, the modulus $|S(\lambda h)|$ tells us for which combinations of λ and stepsize h we achieve exponential decay $y_k \rightarrow \infty$ for $k \rightarrow \infty$, which is the desirable behavior of the approximations for $\operatorname{Re} \lambda < 0$.

Definition 7.1.0.51. Region of (absolute) stability

Let the discrete evolution Ψ for a single step method applied to the scalar linear ODE $\dot{y} = \lambda y, \lambda \in \mathbb{C}$, be of the form

$$\Psi^h y = S(z)y, \quad y \in \mathbb{C}, h > 0 \quad \text{with } z := h\lambda \quad (7.1.0.52)$$

and a function $S : \mathbb{C} \rightarrow \mathbb{C}$. Then the **region of (absolute) stability** of the single step method is given by

$$\mathcal{S}_\Psi := \{z \in \mathbb{C} : |S(z)| < 1\} \subset \mathbb{C}.$$

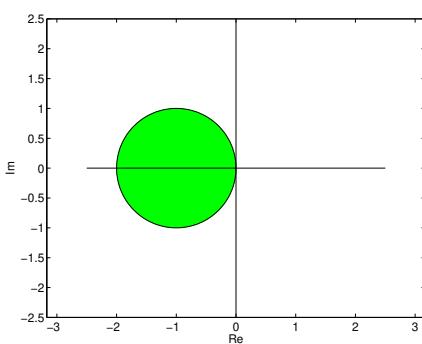
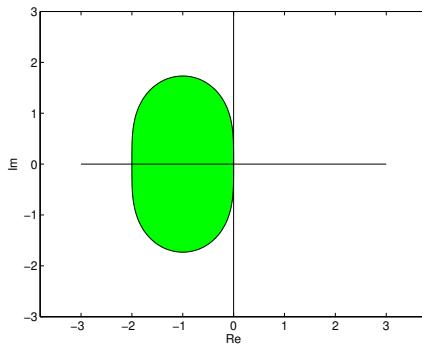
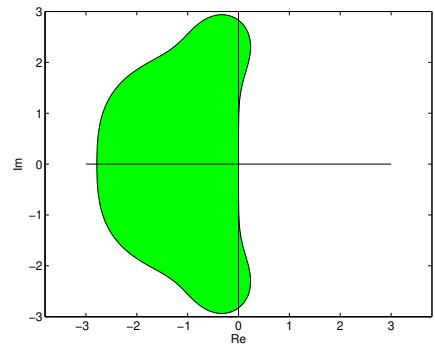
Of course, by Thm. 7.1.0.17, in the case of explicit RK-SSM the function S will coincide with their **stability function** from (7.1.0.18).

We can easily combine the statement of Thm. 7.1.0.48 with the concept of a region of stability and conclude that an explicit RK-SSM will generate exponentially decaying solutions for the linear ODE $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}, \mathbf{M} \in \mathbb{C}^{N,N}$, for every initial state $\mathbf{y}_0 \in \mathbb{C}^N$, if and only if $\lambda_i h \in \mathcal{S}_\Psi$ for all eigenvalues λ_i of \mathbf{M} .

Adopting the arguments of § 9.3.5.6 we conclude from Cor. 7.1.0.20 that

- ◆ the regions of (absolute) stability of explicit RK-SSM are **bounded**,
- ◆ a **timestep constraint** depending on the eigenvalues of \mathbf{M} is necessary to have a guaranteed exponential decay RK-solutions for $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$.

EXAMPLE 7.1.0.53 (Regions of stability of some explicit RK-SSM) The green domains $\subset \mathbb{C}$ depict the bounded regions of stability for some RK-SSM from Ex. 6.4.0.15.

 S_Ψ : explicit Euler (6.2.1.4) S_Ψ : explicit trapezoidal method S_Ψ : classical RK4 method

In general we have for a consistent RK-SSM (\rightarrow Def. 6.3.1.11) that their stability functions satisfy $S(z) = 1 + z + O(z^2)$ for $z \rightarrow 0$. Therefore, $S_\Psi \neq \emptyset$ and the imaginary axis will be tangent to S_Ψ in $z = 0$. \square



Supplementary literature.

Related to this section are [Han02, Ch. 77] and [QSS00,

Sect. 11.3.3].

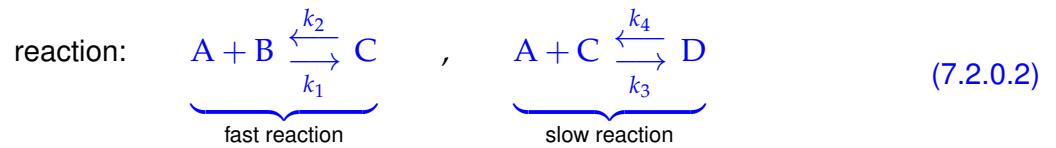
7.2 Stiff Initial-Value Problems



Video tutorial for Section 7.2: Stiff Initial-Value Problems: (39 minutes) [Download link](#), [tablet notes](#)

This section will reveal that the behavior observed in Ex. 7.0.0.1 and Ex. 7.1.0.3 is typical for a large class of problems and that the model problem (7.1.0.5) really represents a “generic case”. This justifies the attention paid to linear model problem analysis in Section 7.1.

EXAMPLE 7.2.0.1 (Kinetics of chemical reactions \rightarrow [Han02, Ch. 62]) In Ex. 6.5.1.1 we already saw an ODE model for the dynamics of a chemical reaction. Now we study an abstract reaction.



Vastly different reaction constants:

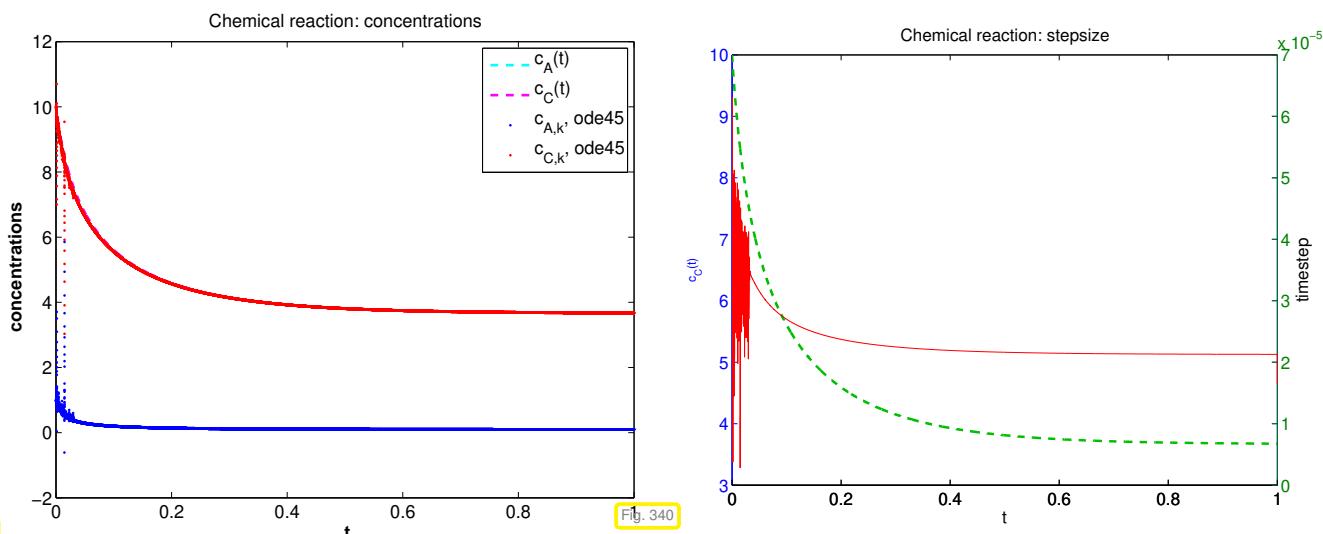
$$k_1, k_2 \gg k_3, k_4$$

► If $c_A(0) > c_B(0)$ \Rightarrow 2nd reaction determines overall long-term reaction dynamics

Mathematical model: non-linear ODE involving concentrations $\mathbf{y}(t) = [c_A(t), c_B(t), c_C(t), c_D(t)]^\top$

$$\dot{\mathbf{y}} := \frac{d}{dt} \begin{bmatrix} c_A \\ c_B \\ c_C \\ c_D \end{bmatrix} = \mathbf{f}(\mathbf{y}) := \begin{bmatrix} -k_1 c_A c_B + k_2 c_C - k_3 c_A c_C + k_4 c_D \\ -k_1 c_A c_B + k_2 c_C \\ k_1 c_A c_B - k_2 c_C - k_3 c_A c_C + k_4 c_D \\ k_3 c_A c_C - k_4 c_D \end{bmatrix}. \quad (7.2.0.3)$$

Concrete choice of parameters: $t_0 = 0$, $T = 1$, $k_1 = 10^4$, $k_2 = 10^3$, $k_3 = 10$, $k_4 = 1$, initial value $\mathbf{y}_0 = [1, 1, 10, 0]^\top$.



Observations: After a **fast initial transient** phase, the solution shows only slow dynamics. Nevertheless, the explicit adaptive integrator used for this simulation insists on using a tiny timestep. It behaves very much like **Ode45** in Ex. 7.0.0.1. ↴

EXAMPLE 7.2.0.4 (Strongly attractive limit cycle) We consider the non-linear Autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ with

$$\mathbf{f}(\mathbf{y}) := \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{y} + \lambda(1 - \|\mathbf{y}\|^2) \mathbf{y}, \quad (7.2.0.5)$$

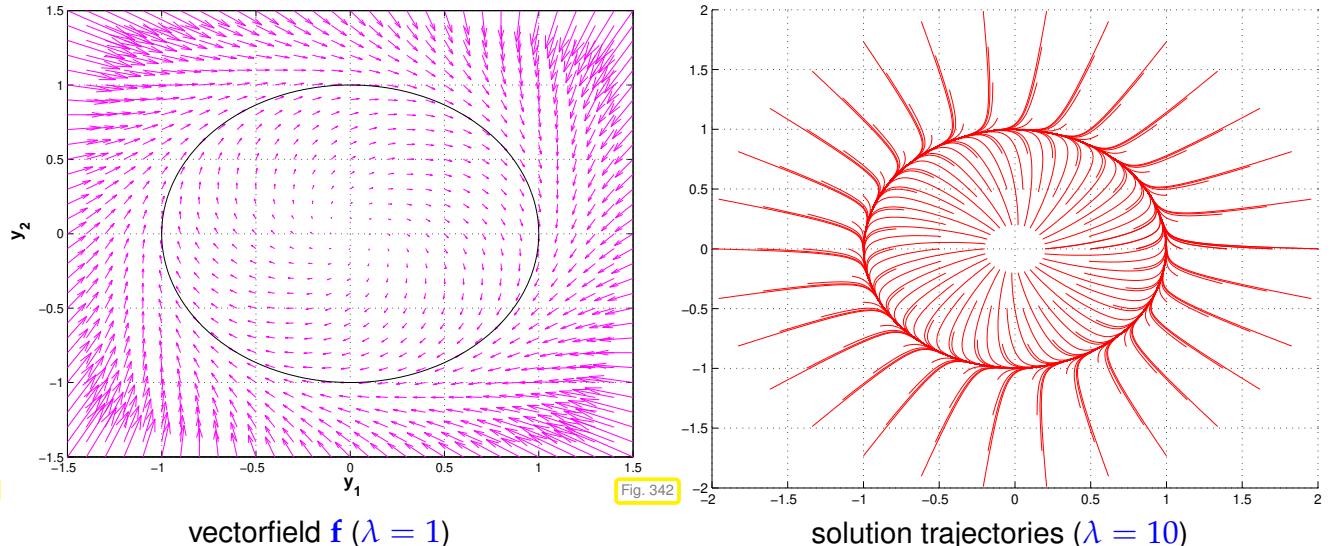
on the state space $D = \mathbb{R}^2 \setminus \{0\}$.

For $\lambda = 0$, the initial value problem $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \begin{bmatrix} \cos \varphi \\ \sin \varphi \end{bmatrix}$, $\varphi \in \mathbb{R}$ has the solution

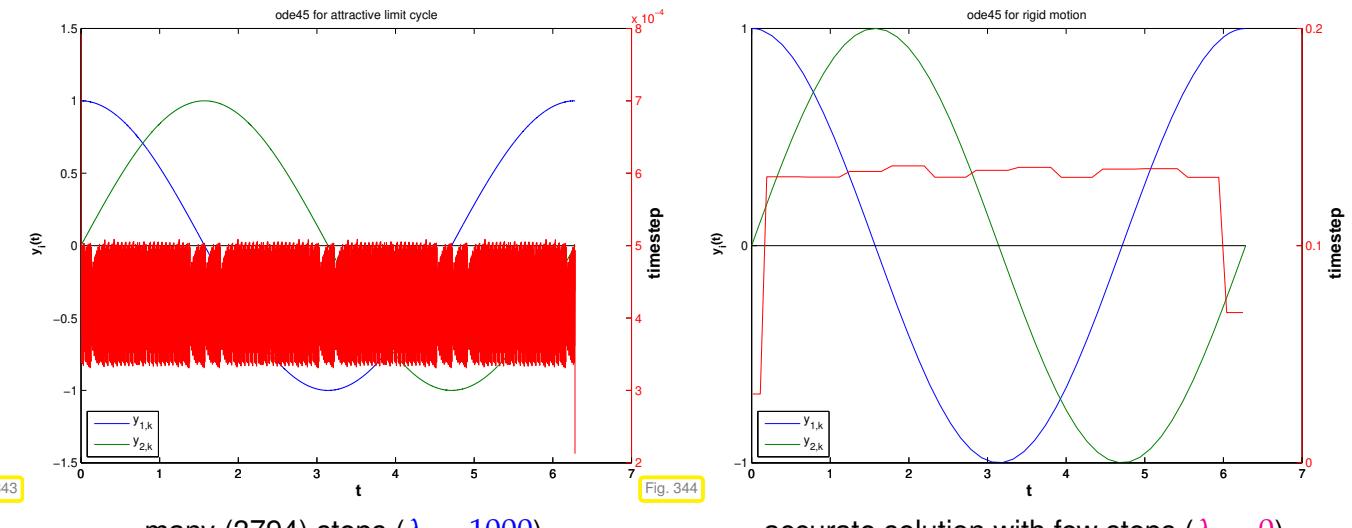
$$\mathbf{y}(t) = \begin{bmatrix} \cos(t - \varphi) \\ \sin(t - \varphi) \end{bmatrix}, \quad t \in \mathbb{R}. \quad (7.2.0.6)$$

For this solution we have $\|\mathbf{y}(t)\|_2 = 1$ for all times.

► (7.2.0.6) provides a solution even for $\lambda \neq 0$, if $\|\mathbf{y}(0)\|_2 = 1$, because in this case the term $\lambda(1 - \|\mathbf{y}\|^2) \mathbf{y}$ will never become non-zero on the solution trajectory.



We study the response of **Ode45** introduced in § 6.5.3.3 to different choice of λ with initial state $\mathbf{y}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. According to the above considerations this initial state should completely “hide the impact of λ from our view”.



Confusing observation: we have $\|\mathbf{y}_0\| = 1$, which implies $\|\mathbf{y}(t)\| = 1 \quad \forall t$!

Thus, the term of the right hand side, which is multiplied by λ will always vanish on the exact solution trajectory, which stays on the unit circle.

Nevertheless, **Ode45** is forced to use tiny timesteps by the *mere presence* of this term! □

We want to find criteria that allow to predict the massive problems haunting explicit single step methods in the case of the *non-linear* IVP of Ex. 7.0.0.1, Ex. 7.2.0.1, and Ex. 7.2.0.4. Recall that for *linear* IVPs of the form $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, $\mathbf{y}(0) = \mathbf{y}_0$, the model problem analysis of Section 7.1 tells us that, given knowledge of the region of stability of the timestepping scheme, the eigenvalues of the matrix $\mathbf{M} \in \mathbb{C}^{N,N}$ provide full information about timestep constraint we are going to face. Refer to Thm. 7.1.0.48 and § 7.1.0.49.

The ODEs we saw in Ex. 7.2.0.1 and Ex. 7.2.0.4 are *non-linear*. Yet, the entire stability analysis of Section 7.1 was based on linear ODEs. Thus, we need to extend the stability analysis to non-linear ODEs.

We start with a “phenomenological notion”, just a keyword to refer to the kind of difficulties presented by the IVPs of Ex. 7.0.0.1, Ex. 7.2.0.1, Ex. 7.1.0.8, and Ex. 7.2.0.4.

Notion 7.2.0.7. Stiff IVP

An initial value problem is called **stiff**, if stability imposes much tighter timestep constraints on *explicit single step methods* than the accuracy requirements.

§7.2.0.8 (Linearization of ODEs) Linear ODEs, though very special, are highly relevant as “local model” for general ODEs:

We consider a general autonomous ODE

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad , \quad \mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N .$$

As usual, we assume \mathbf{f} to be C^2 -smooth and that it enjoys local Lipschitz continuity (\rightarrow Def. 6.1.3.12) on D so that unique solvability of IVPs is guaranteed by Thm. 6.1.3.16.

We fix a state $\mathbf{y}^* \in D$, D the state space, write $t \mapsto \mathbf{y}(t)$ for the solution with $\mathbf{y}(0) = \mathbf{y}^*$. We set $\mathbf{z}(t) = \mathbf{y}(t) - \mathbf{y}^*$, which satisfies

$$\mathbf{z}(0) = 0 \quad , \quad \dot{\mathbf{z}} = \mathbf{f}(\mathbf{y}^* + \mathbf{z}) = \mathbf{f}(\mathbf{y}^*) + D\mathbf{f}(\mathbf{y}^*)\mathbf{z} + R(\mathbf{y}^*, \mathbf{z}) \quad , \quad \text{with} \quad \|R(\mathbf{y}^*, \mathbf{z})\| = O(\|\mathbf{z}\|^2) .$$

This is obtained by Taylor expansion of \mathbf{f} at \mathbf{y}^* , see [Str09, Satz 7.5.2]. Hence, in a neighborhood of a state \mathbf{y}^* on a solution trajectory $t \mapsto \mathbf{y}(t)$, the deviation $\mathbf{z}(t) = \mathbf{y}(t) - \mathbf{y}^*$ satisfies

$$\dot{\mathbf{z}} \approx \mathbf{f}(\mathbf{y}^*) + D\mathbf{f}(\mathbf{y}^*)\mathbf{z} . \quad (7.2.0.9)$$

► The short-time evolution of \mathbf{y} with $\mathbf{y}(0) = \mathbf{y}^*$ is approximately governed by the **affine-linear ODE**

$$\dot{\mathbf{y}} = \mathbf{M}(\mathbf{y} - \mathbf{y}^*) + \mathbf{b} , \quad \mathbf{M} := D\mathbf{f}(\mathbf{y}^*) \in \mathbb{R}^{N,N} , \quad \mathbf{b} := \mathbf{f}(\mathbf{y}^*) \in \mathbb{R}^N . \quad (7.2.0.10)$$

In the scalar case we have come across this linearization already in Ex. 7.1.0.3. □

§7.2.0.11 (Linearization of explicit Runge-Kutta single step methods) We consider one step a general s -stage RK-SSM according to Def. 6.4.0.9 for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, with smooth right hand side function $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$:

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0 + h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j) , \quad i = 1, \dots, s \quad , \quad \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i .$$

We perform linearization at $\mathbf{y}^* := \mathbf{y}_0$ and ignore all terms at least quadratic in the timestep size h (this is indicated by the \approx symbol):

$$\mathbf{k}_i \approx \mathbf{f}(\mathbf{y}^*) + D\mathbf{f}(\mathbf{y}^*)h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j , \quad i = 1, \dots, s \quad , \quad \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i .$$

The defining equations for the same RK-SSM applied to

$$\dot{\mathbf{z}} = \mathbf{M}\mathbf{z} + \mathbf{b} , \quad \mathbf{M} := D\mathbf{f}(\mathbf{y}^*) \in \mathbb{R}^{N,N} , \quad \mathbf{b} := \mathbf{f}(\mathbf{y}^*) ,$$

which agrees with (7.2.0.10) after substitution $\mathbf{z}(t) = \mathbf{y}(t) - \mathbf{y}^*$, are

$$\mathbf{k}_i \approx \mathbf{b} + \mathbf{M}h \sum_{j=1}^{i-1} a_{ij} \mathbf{k}_j , \quad i = 1, \dots, s \quad , \quad \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i .$$

We find that for small timesteps

the discrete evolution of the RK-SSM for $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ in the state \mathbf{y}^* is close to the discrete evolution of the same RK-SSM applied to the linearization (7.2.0.10) of the ODE in \mathbf{y}^* .

By straightforward manipulations of the defining equations of an explicit RK-SSM we find that, if

- $(\mathbf{y}_k)_k$ is the sequence of states generated by the RK-SSM applied to the affine-linear ODE $\dot{\mathbf{y}} = \mathbf{M}(\mathbf{y} - \mathbf{y}_0) + \mathbf{b}$, $\mathbf{M} \in \mathbb{C}^{N,N}$ regular,
- $(\mathbf{w}_k)_k$ is the sequence of states generated by the same RK-SSM applied to the linear ODE $\dot{\mathbf{w}} = \mathbf{M}\mathbf{w}$ and $\mathbf{w}_0 := \mathbf{M}^{-1}\mathbf{b}$, then

$$\mathbf{w}_k = \mathbf{y}_k - \mathbf{y}_0 + \mathbf{M}^{-1}\mathbf{b} .$$

- The analysis of the behavior of an RK-SSM for an affine-linear ODE can be reduced to understanding its behavior for a linear ODE with the same matrix.

Combined with the insights from § 7.1.0.43 this means that

the behavior of an explicit Runge-Kutta single-step method applied to $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ close to the state \mathbf{y}^* is determined by the eigenvalues of the Jacobian $D\mathbf{f}(\mathbf{y}^*)$.

In particular, if $D\mathbf{f}(\mathbf{y}^*)$ has at least one eigenvalue whose modulus is large, then an exponential drift-off of the approximate states \mathbf{y}_k away from \mathbf{y}^* can only be avoided for sufficiently small timestep, again a **timestep constraint**.

How to distinguish stiff initial value problems

An initial value problem for an autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ will probably be stiff, if, for substantial periods of time,

$$\min\{\operatorname{Re} \lambda : \lambda \in \sigma(D\mathbf{f}(\mathbf{y}(t)))\} \ll 0, \quad (7.2.0.13)$$

$$\text{and} \quad \max\{\operatorname{Re} \lambda : \lambda \in \sigma(D\mathbf{f}(\mathbf{y}(t)))\} \lesssim 0, \quad (7.2.0.14)$$

where $t \mapsto \mathbf{y}(t)$ is the solution trajectory and $\sigma(\mathbf{M})$ is the spectrum of the matrix \mathbf{M} , see [Hip19, ??].

The condition (7.2.0.14) has to be read as “the real parts of all eigenvalues are below a bound with small modulus”. If this is not the case, then the exact solution will experience blow-up. It will change drastically over very short periods of time and small timesteps will be required anyway in order to resolve this. □

EXAMPLE 7.2.0.15 (Predicting stiffness of non-linear IVPs)

- ① We consider the IVP from Ex. 7.0.0.1:

$$\text{IVP considered: } \dot{y} = f(y) := \lambda y^2(1 - y), \quad \lambda := 500, \quad y(0) = \frac{1}{100}.$$

We find

$$f'(y) = \lambda(2y - 3y^2) \Rightarrow f'(1) = -\lambda.$$

Hence, in case $\lambda \gg 1$ as in Fig. 326, we face a stiff problem close to the stationary state $y = 1$. The observations made in Fig. 326 exactly match this prediction.

- ② The solution of the IVP from Ex. 7.2.0.4

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) := \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{y} + \lambda(1 - \|\mathbf{y}\|^2) \mathbf{y}, \quad \|\mathbf{y}_0\|_2 = 1. \quad (7.2.0.5)$$

satisfies $\|\mathbf{y}(t)\|_2 = 1$ for all times. Using the product rule [Hip19, ??] of multi-dimensional differential calculus, we find

$$D\mathbf{f}(\mathbf{y}) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} + \lambda \left(-2\mathbf{y}\mathbf{y}^\top + (1 - \|\mathbf{y}\|^2) \mathbf{I} \right).$$

$$\Rightarrow \sigma(D\mathbf{f}(\mathbf{y})) = \left\{ -\lambda - \sqrt{\lambda^2 - 1}, -\lambda + \sqrt{\lambda^2 - 1} \right\}, \quad \text{if } \|\mathbf{y}\|_2 = 1.$$

Thus, for $\lambda \gg 1$, $D\mathbf{f}(\mathbf{y}(t))$ will always have an eigenvalue with large negative real part, whereas the other eigenvalue is close to zero: the IVP is stiff.

Remark 7.2.0.16 (Characteristics of stiff IVPs) Often one can already tell from the expected behavior of the solution of an IVP, which is often clear from the modeling context, that one has to brace for stiffness.

Typical features of stiff IVPs:

- ◆ Presence of **fast transients** in the solution, see Ex. 7.1.0.3, Ex. 7.1.0.35,
- ◆ Occurrence of **strongly attractive** fixed points/limit cycles, see Ex. 7.2.0.4

7.3 Implicit Runge-Kutta Single-Step Methods



Video tutorial for Section 7.3: Implicit Runge-Kutta Single-Step Methods: (78 minutes)
[Download link](#), [tablet notes](#)

Explicit Runge-Kutta single step method cannot escape tight timestep constraints for stiff IVPs that may render them inefficient, see § 7.1.0.49. In this section we are going to augment the class of Runge-Kutta methods by timestepping schemes that can cope well with stiff IVPs.

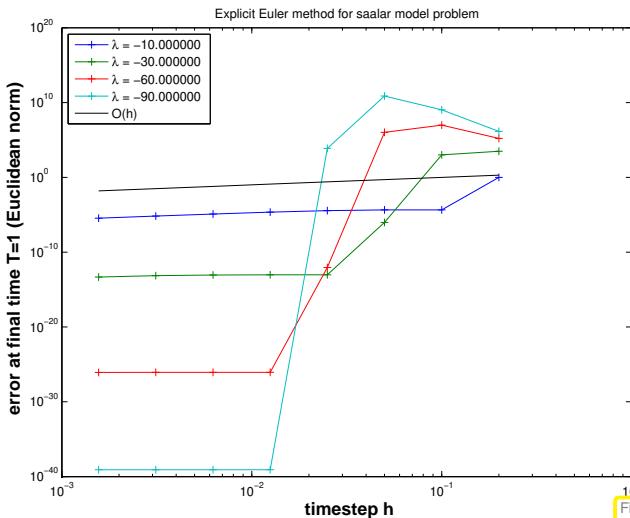
7.3.1 The Implicit Euler Method for Stiff IVPs

EXPERIMENT 7.3.1.1 (Euler methods for stiff decay IVP) We revisit the setting of Ex. 7.1.0.3 and again consider Euler methods for the decay IVP

$$\dot{y} = \lambda y \quad , \quad y(0) = 1 \quad , \quad \lambda < 0 \quad .$$

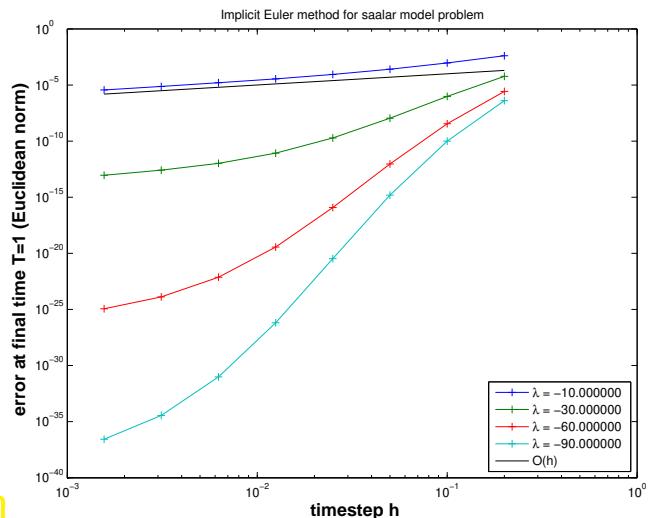
We apply both the explicit Euler method (6.2.1.4) and the implicit Euler method (6.2.2.2) with uniform timesteps $h = 1/N$, $N \in \{5, 10, 20, 40, 80, 160, 320, 640\}$ and monitor the error at final time $T = 1$ for different values of λ .

Explicit Euler method (6.2.1.4)



λ large: blow-up of y_k for large timestep h

Implicit Euler method (6.2.2.2)



λ large: stable for all timesteps $h > 0$!

We observe onset of convergence of the implicit Euler method already for large timesteps h .

§7.3.1.2 (Linear model problem analysis: implicit Euler method) We follow the considerations of § 7.1.0.4 and consider the *implicit* Euler method (6.2.2.2) for the

$$\text{linear model problem: } \dot{y} = \lambda y, \quad y(0) = y_0, \quad \text{with } \operatorname{Re} \lambda \ll 0, \quad (7.1.0.5)$$

with *exponentially decaying* (maybe oscillatory for $\operatorname{Im} \lambda \neq 0$) exact solution

$$y(t) = y_0 \exp(\lambda t) \rightarrow 0 \quad \text{for } t \rightarrow \infty.$$

The recursion of the implicit Euler method for (7.1.0.5) is defined by

$$(6.2.2.2) \text{ for } f(y) = \lambda y \Rightarrow y_{k+1} = y_k + \lambda h y_{k+1}, \quad k \in \mathbb{N}_0. \quad (7.3.1.3)$$

$$\blacktriangleright \text{ generated sequence } y_k := \left(\frac{1}{1 - \lambda h} \right)^k y_0. \quad (7.3.1.4)$$

$$\blacktriangleright \boxed{\operatorname{Re} \lambda < 0 \Rightarrow \lim_{k \rightarrow \infty} y_k = 0 \quad \forall h > 0!} \quad (7.3.1.5)$$

Without any timestep constraint we obtain the qualitatively correct behavior of $(y_k)_k$ for $\operatorname{Re} \lambda < 0$ and **any** $h > 0$!

As in § 7.1.0.40 this analysis can be extended to linear systems of ODEs $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$, $\mathbf{M} \in \mathbb{C}^{N,N}$, by means of **diagonalization**.

As in § 7.1.0.30 and § 7.1.0.40 we assume that \mathbf{M} can be diagonalized, that is (7.1.0.32) holds: $\mathbf{V}^{-1}\mathbf{M}\mathbf{V} = \mathbf{D}$ with a regular matrix $\mathbf{V} \in \mathbb{C}^{N,N}$ and a diagonal matrix $\mathbf{D} \in \mathbb{C}^{N,N}$ containing the eigenvalues $\lambda_1, \dots, \lambda_N$ of \mathbf{M} on its diagonal. Next, apply the **decoupling by diagonalization** idea to the recursion of the implicit Euler method.

$$\mathbf{V}^{-1}\mathbf{y}_{k+1} = \mathbf{V}^{-1}\mathbf{y}_k + h \underbrace{\mathbf{V}^{-1}\mathbf{M}\mathbf{V}}_{=\mathbf{D}} (\mathbf{V}^{-1}\mathbf{y}_{k+1}) \stackrel{\mathbf{z}_k := \mathbf{V}^{-1}\mathbf{y}_k}{\Leftrightarrow} (\mathbf{z}_{k+1})_i = \underbrace{\frac{1}{1 - \lambda_i h} (\mathbf{z}_k)_i}_{\triangleq \text{implicit Euler step for } z_i = \lambda_i z_i}. \quad (7.3.1.6)$$

Crucial insight:

For any timestep, the implicit Euler method generates exponentially decaying solution sequences $(y_k)_{k=0}^\infty$ for $\dot{\mathbf{y}} = \mathbf{M}\mathbf{y}$ with diagonalizable matrix $\mathbf{M} \in \mathbb{R}^{N,N}$ with eigenvalues $\lambda_1, \dots, \lambda_N$, if $\operatorname{Re} \lambda_i < 0$ for all $i = 1, \dots, N$.

Thus we expect that the implicit Euler method will not face stability induced timestep constraints for stiff problems (\rightarrow Notion 7.2.0.7). \square

7.3.2 Collocation Single-Step Methods

Unfortunately the implicit Euler method is of first order only, see Exp. 6.3.2.5. This section presents an algorithm for designing higher order single step methods generalizing the implicit Euler method.

Setting: We consider the general ordinary differential equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, $\mathbf{f} : I \times D \rightarrow \mathbb{R}^N$ locally Lipschitz continuous, which guarantees the local existence of unique solutions of initial value problems, see Thm. 6.1.3.16.

We define the single step method through specifying the first step $\mathbf{y}_0 = \mathbf{y}(t_0) \rightarrow \mathbf{y}_1 \approx \mathbf{y}(t_1)$, where $\mathbf{y}_0 \in D$ is the initial step at initial time $t_0 \in I$. We assume that the exact solution trajectory $t \mapsto \mathbf{y}(t)$

exists on $[t_0, t_1]$. Use as a timestepping scheme on a temporal mesh (\rightarrow § 6.2.0.2) in the sense of Def. 6.3.1.4 is straightforward.

§7.3.2.1 (Collocation approach)

Abstract collocation idea

Collocation is a paradigm for the **discretization** (\rightarrow Section 2.1.1) of *differential equations*:

- (I) Write the discrete solution \mathbf{u}_h , a function, as linear combination of $N \in \mathbb{N}$ sufficiently smooth (basis) functions $\geq N$ unknown coefficients.
- (II) Demand that \mathbf{u}_h satisfies the differential equation at N points/times $\geq N$ equations.

We apply this policy to the differential equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ on $[t_0, t_1]$:

Idea: ① Approximate $t \mapsto \mathbf{y}(t)$, $t \in [t_0, t_1]$, by a function $t \mapsto \mathbf{y}_h(t) \in V$, V an $N \cdot (s+1)$ -dimensional **trial space** V comprising functions $[t_0, t_1] \mapsto \mathbb{R}^N$, cf. Item (I).

② Fix $\mathbf{y}_h \in V$ by imposing **collocation conditions**

$$\begin{aligned} \mathbf{y}_h(t_0) &= \mathbf{y}_0, \\ \dot{\mathbf{y}}_h(\tau_j) &= \mathbf{f}(\tau_j, \mathbf{y}_h(\tau_j)), \quad j = 1, \dots, s, \end{aligned} \tag{7.3.2.3}$$

for **collocation points** $t_0 \leq \tau_1 < \dots < \tau_s \leq t_1 \rightarrow$ Item (II).

③ Choose $\mathbf{y}_1 := \mathbf{y}_h(t_1)$.



§7.3.2.4 (Polynomial collocation) Existence of the function $\mathbf{y}_h : [t_0, t_1] \rightarrow \mathbb{R}^N$ satisfying (7.3.2.3) and the possibility to compute it efficiently will crucially depend on the choice of the trial space V .

Our choice (the “standard option”):

(Componentwise) polynomial trial space $V = (\mathcal{P}_s)^N$

Recalling $\dim \mathcal{P}_s = s+1$ from [Hip19, ??], Lemma 2.5.2.5, we see that our choice makes the number $= N(s+1)$ of collocation conditions matches the dimension of the trial space V .

Now we want to derive a concrete representation for the polynomial \mathbf{y}_h . We draw on concepts introduced in [Hip19, ??]. We define the collocation points as

$$\tau_j := t_0 + c_j h, \quad j = 1, \dots, s, \quad \text{for } 0 \leq c_1 < c_2 < \dots < c_s \leq 1, \quad h := t_1 - t_0.$$

Let $\{L_j\}_{j=1}^s \subset \mathcal{P}_{s-1}$ denote the set of Lagrange polynomials of degree $s-1$ associated with the node set $\{c_j\}_{j=1}^s$, see [Hip19, ??]. They satisfy $L_j(c_i) = \delta_{ij}$, $i, j = 1, \dots, s$ and form a basis of \mathcal{P}_{s-1} .

In each of its N components, the derivative $\dot{\mathbf{y}}_h$ is a polynomial of degree $s-1$: $\dot{\mathbf{y}} \in (\mathcal{P}_{s-1})^N$. Hence, it has the following representation, compare [Hip19, ??].

$$\dot{\mathbf{y}}_h(t_0 + \xi h) = \sum_{j=1}^s \dot{\mathbf{y}}_h(t_0 + c_j h) L_j(\xi), \quad 0 \leq \xi \leq 1. \tag{7.3.2.5}$$

As $\tau_j = t_0 + c_j h$, the **collocation conditions** (7.3.2.3) make it possible to replace $\dot{\mathbf{y}}_h(c_j h)$ with an expression

in the right hand side function \mathbf{f} :

$$(7.3.2.3) \quad \blacktriangleright \quad \dot{\mathbf{y}}_h(t_0 + \xi h) = \sum_{j=1}^s \mathbf{k}_j L_j(\xi) \quad \text{with "coefficients" } \mathbf{k}_j := f(t_0 + c_j h, \mathbf{y}_h(t_0 + c_j h)) .$$

Next we integrate and use $\mathbf{y}_h(t_0) = \mathbf{y}_0$

$$\blacktriangleright \quad \mathbf{y}_h(t_0 + \xi h) = \mathbf{y}_0 + h \sum_{j=1}^s \mathbf{k}_j \int_0^\xi L_j(\zeta) d\zeta .$$

This yields the following formulas for the computation of \mathbf{y}_1 , which characterize the s -stage collocation **single step method** induced by the (normalized) collocation points $c_j \in [0, 1]$, $j = 1, \dots, s$.

$$\begin{aligned} \mathbf{k}_i &= f(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^s a_{ij} \mathbf{k}_j) , & \text{where } a_{ij} &:= \int_0^{c_i} L_j(\tau) d\tau , \\ \mathbf{y}_1 := \mathbf{y}_h(t_1) &= \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i . & b_i &:= \int_0^1 L_i(\tau) d\tau . \end{aligned} \tag{7.3.2.6}$$

Note that, since arbitrary $\mathbf{y}_0 \in D$, $t_0, t_1 \in I$ were admitted, this defines a discrete evolution $\Psi : I \times I \times D \rightarrow \mathbb{R}^N$ by $\Psi^{t_0, t_1} \mathbf{y}_0 := \mathbf{y}_h(t_1)$. \square

Remark 7.3.2.7 (Implicit nature of collocation single step methods) Note that (7.3.2.6) represents a generically non-linear system of $s \cdot N$ equations for the $s \cdot N$ components of the vectors \mathbf{k}_i , $i = 1, \dots, s$. Usually, it will not be possible to obtain the increments $\mathbf{k}_i \in \mathbb{R}^N$ by a fixed number of evaluations of \mathbf{f} . For this reason the single step methods defined by (7.3.2.6) are called **implicit**.

With similar arguments as in Rem. 6.2.2.3 one can prove that for sufficiently small $|t_1 - t_0|$ a unique set of solution vectors $\mathbf{k}_1, \dots, \mathbf{k}_s$ can be found. \square

§7.3.2.8 (Collocation single step methods and quadrature) Clearly, in the case $N = 1$, $f(t, \mathbf{y}) = f(t)$, $\mathbf{y}_0 = 0$ the computation of \mathbf{y}_1 boils down to the evaluation of a quadrature formula on $[t_0, t_1]$, because from (7.3.2.6) we get

$$y_1 = h \sum_{i=1}^s b_i f(t_0 + c_i h) , \quad b_i := \int_0^1 L_i(\tau) d\tau , \tag{7.3.2.9}$$

which is a polynomial quadrature formula [Hip19, ??] on $[0, 1]$ with nodes c_j transformed to $[t_0, t_1]$ according to [Hip19, ??]. \square

EXPERIMENT 7.3.2.10 (Empiric Convergence of collocation single step methods) We consider the initial value problem for the scalar logistic ODE

$$\dot{y} = \lambda y(1 - y) , \quad y(0) = 0.01 , \quad \lambda = 100 ,$$

which is mildly stiff, over the time interval $[0, 1]$

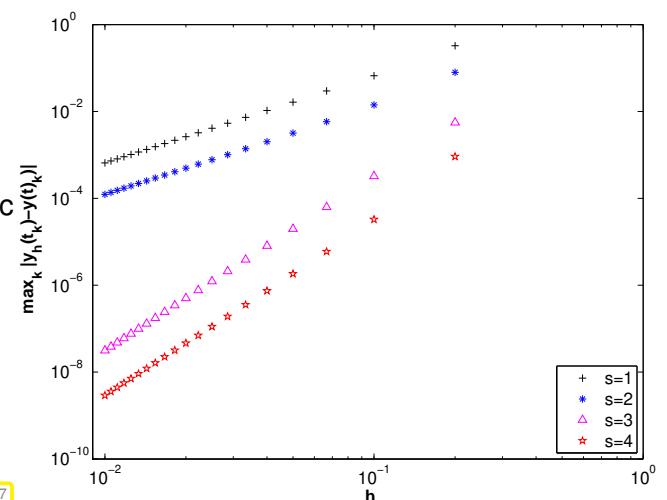
We perform numerical integration by timestepping with uniform timestep h based on a collocation single

step method (7.3.2.6).

- Equidistant collocation points, $c_j = \frac{j}{s+1}$, $j = 1, \dots, s$.

We observe **algebraic convergence** with the empiric rates

$$\begin{aligned} s = 1 &: p = 1.96 \\ s = 2 &: p = 2.03 \\ s = 3 &: p = 4.00 \\ s = 4 &: p = 4.04 \end{aligned}$$



In this case we conclude the following (empiric) order (\rightarrow Def. 6.3.2.8) of the collocation single step method:

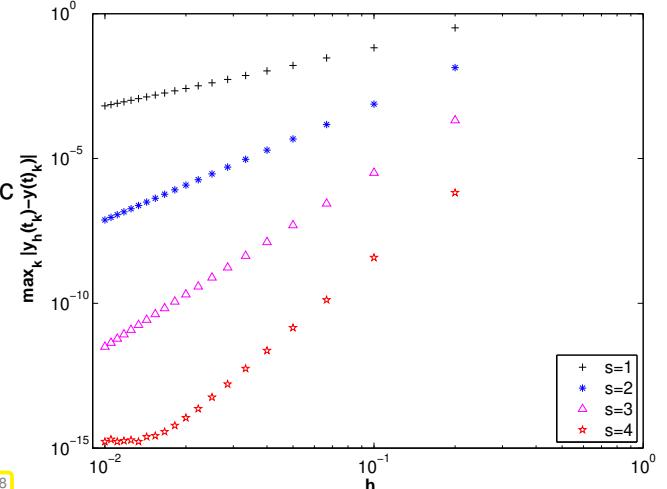
$$(\text{empiric order}) = \begin{cases} s & \text{for even } s, \\ s + 1 & \text{for odd } s. \end{cases}$$

Next, we recall from [Hip19, ??] an exceptional set of quadrature points, the **Gauss points**, provided by the zeros of the $L^2([-1, 1])$ -orthogonal Legendre polynomials, see Fig. 146.

- Gauss points in $[0, 1]$
as normalized collocation points c_j , $j = 1, \dots, s$.

We observe **algebraic convergence** with the empiric rates

$$\begin{aligned} s = 1 &: p = 1.96 \\ s = 2 &: p = 4.01 \\ s = 3 &: p = 6.00 \\ s = 4 &: p = 8.02 \end{aligned}$$



Obviously, for the (empiric) order (\rightarrow Def. 6.3.2.8) of the **Gauss collocation single step method** holds

$$(\text{empiric order}) = 2s.$$

Note that the 1-stage Gauss collocation single step method is the implicit midpoint method from Section 6.2.3. □

§7.3.2.11 (Order of collocation single step method) What we have observed in Exp. 9.2.6.27 reflects

a fundamental result on collocation single step methods as defined in (7.3.2.6).

Theorem 7.3.2.12. Order of collocation single step method [DB02, Satz .6.40]

Provided that $\mathbf{f} \in C^p(I \times D)$, the order (\rightarrow Def. 6.3.2.8) of an s -stage collocation single step method according to (7.3.2.6) agrees with the order (\rightarrow Def. 2.7.5.29) of the quadrature formula on $[0, 1]$ with nodes c_j and weights b_j , $j = 1, \dots, s$.

This also explains the surprisingly high order of the Gauss collocation single-step method, because for s -point Gauss-Legendre numerical quadrature, the family of quadrature rules based on Gauss points as nodes, [Hip19, ??] derived the order $2s$.

- By [Hip19, ??] the s -stage Gauss collocation single step method whose nodes c_j are chosen as the s Gauss points on $[0, 1]$ is of order $2s$. □

7.3.3 General Implicit Runge-Kutta Single-Step Methods (RK-SSMs)

The notations in (7.3.2.6) have deliberately been chosen to allude to Def. 6.4.0.9. In that definition it takes only letting the sum in the formula for the increments run up to s to capture (7.3.2.6).

Definition 7.3.3.1. General Runge-Kutta single step method (cf. Def. 6.4.0.9)

For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, a single step of size $h > 0$ of an s -stage **Runge-Kutta single step method** (RK-SSM) for the IVP (6.1.3.2) is defined by

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^s a_{ij} \mathbf{k}_j), \quad i = 1, \dots, s, \quad \mathbf{y}_1 := \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i.$$

As before, the vectors $\mathbf{k}_i \in \mathbb{R}^N$ are called **increments**.

Note that the computation of the increments \mathbf{k}_i may now require the solution of *(non-linear) systems of equations* of size $s \cdot N$. In this case we speak about an “implicit” method, cf. Rem. 7.3.2.7.

The Butcher scheme notation introduced in (6.4.0.11) can easily be adapted to the the case of general RK-SSMs by dropping the requirement that the Butcher matrix be strictly lower triangular.

General Butcher scheme notation for RK-SSM

Shorthand notation for Runge-Kutta methods

Butcher scheme

$$\triangleright \quad \begin{array}{c|ccccc} \mathbf{c} & \mathfrak{A} \\ \hline \mathbf{b}^T & & & & & \end{array} := \begin{array}{c|cccc} c_1 & a_{11} & \cdots & a_{1s} & \\ \vdots & \vdots & & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} & \\ \hline b_1 & b_1 & \cdots & b_s & \end{array} \quad (7.3.3.3)$$

Note that now, in contrast to (6.4.0.11), \mathfrak{A} can be a general $s \times s$ -matrix.

Summary: terminology for Runge-Kutta single step methods:

- | | |
|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| \mathfrak{A} strict lower triangular matrix
\mathfrak{A} lower triangular matrix | ► explicit Runge-Kutta method, Def. 6.4.0.9
► diagonally-implicit Runge-Kutta method (DIRK) |
|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|

Many of the techniques and much of the theory discussed for explicit RK-SSMs carry over to general (implicit) Runge-Kutta single step methods:

- Sufficient condition for consistence from Cor. 6.4.0.13
- Algebraic convergence for meshwidth $h \rightarrow 0$ and the related concept of order (\rightarrow Def. 6.3.2.8)
- Embedded methods and algorithms for adaptive stepsize control from Section 6.5

§7.3.3.4 (Butcher schemes for Gauss collocation RK-SSMs) As in (6.4.0.11) we can arrange the coefficients of Gauss collocation single-step methods in the form of a **Butcher scheme** and get

$$\text{for } s = 1: \quad \begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}, \quad (7.3.3.5a)$$

$$\text{for } s = 2: \quad \begin{array}{c|cc} \frac{1}{2} - \frac{1}{6}\sqrt{3} & \frac{1}{4} & \frac{1}{4} - \frac{1}{6}\sqrt{3} \\ \frac{1}{2} + \frac{1}{6}\sqrt{3} & \frac{1}{4} + \frac{1}{6}\sqrt{3} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}, \quad (7.3.3.5b)$$

$$\text{for } s = 3: \quad \begin{array}{c|ccc} \frac{1}{2} - \frac{1}{10}\sqrt{15} & \frac{5}{36} & \frac{2}{9} - \frac{1}{15}\sqrt{15} & \frac{5}{36} - \frac{1}{30}\sqrt{15} \\ \frac{1}{2} & \frac{5}{36} + \frac{1}{24}\sqrt{15} & \frac{2}{9} & \frac{5}{36} - \frac{1}{24}\sqrt{15} \\ \frac{1}{2} + \frac{1}{10}\sqrt{15} & \frac{5}{36} + \frac{1}{30}\sqrt{15} & \frac{2}{9} + \frac{1}{15}\sqrt{15} & \frac{5}{36} \\ \hline & \frac{5}{18} & \frac{4}{9} & \frac{5}{18} \end{array}. \quad (7.3.3.5c)$$

Remark 7.3.3.6 (Stage form equations for increments) In Def. 7.3.3.1 instead of the increments we can consider as unknowns the so-called **stages**

$$\mathbf{g}_i := h \sum_{j=1}^s a_{ij} \mathbf{k}_j, \quad i = 1, \dots, s, \quad \Leftrightarrow \quad \mathbf{k}_i = \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + \mathbf{g}_i). \quad (7.3.3.7)$$

This leads to the equivalent defining equations in “stage form” for an implicit RK-SSM

$$\begin{aligned} \mathbf{k}_i &:= \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^s a_{ij} \mathbf{k}_j), \quad i = 1, \dots, s, \\ &\Downarrow \\ \mathbf{g}_i &= h \sum_{j=1}^s a_{ij} \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + \mathbf{g}_j), \quad \mathbf{y}_1 = \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + \mathbf{g}_i). \end{aligned} \quad (7.3.3.8)$$

In terms of implementation there is no difference: Also the **stage equations** (7.3.3.8) are usually solved by means of Newton’s method, see next remark.

Remark 7.3.3.9 (Solving the stage equations for implicit RK-SSMs) We reformulate the increment equations in stage form (7.3.3.8) as a non-linear system of equations in standard form $\mathbf{F}(\mathbf{x}) = \mathbf{0}$.

Unknowns are the total $s \cdot N$ components of the stage vectors $\mathbf{g}_i \in \mathbb{R}^N$, $i = 1, \dots, s$ as defined in (7.3.3.7).

$$\begin{aligned} \mathbf{g} &= [\mathbf{g}_1, \dots, \mathbf{g}_s]^\top \in \mathbb{R}^{s \cdot N}, \\ \mathbf{g}_i &:= h \sum_{j=1}^s a_{ij} \mathbf{f}(t_0 + c_j h, \mathbf{y}_0 + \mathbf{g}_j) \end{aligned} \quad \Rightarrow \quad F(\mathbf{g}) = \mathbf{g} - h(\mathfrak{A} \otimes \mathbf{I}_N) \begin{bmatrix} \mathbf{f}(t_0 + c_1 h, \mathbf{y}_0 + \mathbf{g}_1) \\ \vdots \\ \mathbf{f}(t_0 + c_s h, \mathbf{y}_0 + \mathbf{g}_s) \end{bmatrix} \stackrel{!}{=} \mathbf{0},$$

where \mathbf{I}_N is the $N \times N$ identity matrix and \otimes designates the Kronecker product introduced in [Hip19, ??].

We compute an approximate solution of $F(\mathbf{g}) = \mathbf{0}$ iteratively by means of the simplified Newton method presented in [Hip19, ??]. This is a Newton method with “frozen Jacobian”. As $\mathbf{g} \rightarrow \mathbf{0}$ for $h \rightarrow 0$, we choose zero as initial guess:

$$\mathbf{g}^{(k+1)} = \mathbf{g}^{(k)} - DF(\mathbf{0})^{-1} F(\mathbf{g}^{(k)}) \quad k = 0, 1, 2, \dots, \quad \mathbf{g}^{(0)} = \mathbf{0}. \quad (7.3.3.10)$$

with the Jacobian

$$DF(\mathbf{0}) = \begin{bmatrix} \mathbf{I} - ha_{11} \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_0, \mathbf{y}_0) & \cdots & -ha_{1s} \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_0, \mathbf{y}_0) \\ \vdots & \ddots & \vdots \\ -ha_{s1} \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_0, \mathbf{y}_0) & \cdots & \mathbf{I} - ha_{ss} \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_0, \mathbf{y}_0) \end{bmatrix} \in \mathbb{R}^{sN, sN}. \quad (7.3.3.11)$$

Obviously, $DF(\mathbf{0}) \rightarrow \mathbf{I}$ for $h \rightarrow 0$. Thus, $DF(\mathbf{0})$ will be regular for sufficiently small h .

In each step of the simplified Newton method we have to solve a linear system of equations with coefficient matrix $DF(\mathbf{0})$. If $s \cdot N$ is large, an efficient implementation has to reuse the LU-decomposition of $DF(\mathbf{0})$, see [Hip19, ??] and [Hip19, ??]. \square

7.3.4 Model Problem Analysis for Implicit Runge-Kutta Single-Step Methods (IRK-SSMs)

Model problem analysis for general Runge-Kutta single step methods (\rightarrow Def. 7.3.3.1) runs parallel to that for explicit RK-methods as elaborated in Section 7.1, § 7.1.0.13. Familiarity with the techniques and results of this section is assumed. The reader is asked to recall the concept of **stability function** from Thm. 7.1.0.17, the **diagonalization technique** from § 7.1.0.43, and the definition of **region of (absolute) stability** from Def. 7.1.0.51.

We apply the implicit RK-SSM according to Def. 7.3.3.1 to the autonomous linear scalar ODE $\dot{\mathbf{y}} = \lambda \mathbf{y}$, $\lambda \in \mathbb{C}$, and utterly parallel to the considerations in § 7.1.0.13, (7.1.0.14) we obtain

$$\begin{aligned} \mathbf{k}_i &= \lambda(y_0 + h \sum_{j=1}^s a_{ij} \mathbf{k}_j), \\ \mathbf{y}_1 &= y_0 + h \sum_{i=1}^s b_i \mathbf{k}_i \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} \mathbf{I} - z\mathfrak{A} & \mathbf{0} \\ -z\mathbf{b}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{k} \\ \mathbf{y}_1 \end{bmatrix} = y_0 \begin{bmatrix} \mathbf{1} \\ 1 \end{bmatrix}, \quad (7.3.4.1)$$

where $\mathbf{k} \in \mathbb{R}^s \hat{=} \text{denotes the vector } [\mathbf{k}_1, \dots, \mathbf{k}_s]^\top / \lambda$ of increments, and $z := \lambda h$. As in § 7.1.0.13 we can eliminate the increments and obtain an expression for \mathbf{y}_1 :

$$\mathbf{y}_1 = S(z)y_0 \quad \text{with} \quad S(z) := 1 + z\mathbf{b}^\top (\mathbf{I} - z\mathfrak{A})^{-1} \mathbf{1}. \quad (7.3.4.2)$$

Alternatively, Cramer's rule supplies a formula for y_1 in terms of determinants:

$$y_1 = y_0 \frac{\det \begin{bmatrix} \mathbf{I} - z\mathfrak{A} & \mathbf{1} \\ -z\mathbf{b}^T & 1 \end{bmatrix}}{\det \begin{bmatrix} \mathbf{I} - z\mathfrak{A} & 0 \\ -z\mathbf{b}^T & 1 \end{bmatrix}} \Rightarrow S(z) = \frac{\det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)}{\det(\mathbf{I} - z\mathfrak{A})}. \quad (7.3.4.3)$$

The next theorem summarizes these findings:

Theorem 7.3.4.4. Stability function of Runge-Kutta methods, cf. Thm. 7.1.0.17

The discrete evolution Ψ_λ^h of an s -stage Runge-Kutta single step method (\rightarrow Def. 7.3.3.1) with

Butcher scheme $\begin{array}{c|cc} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array}$ (see (7.3.3.3)) for the ODE $\dot{\mathbf{y}} = \lambda\mathbf{y}$ is given by a multiplication with

$$S(z) := \underbrace{1 + z\mathbf{b}^T(\mathbf{I} - z\mathfrak{A})^{-1}\mathbf{1}}_{\text{stability function}} = \frac{\det(\mathbf{I} - z\mathfrak{A} + z\mathbf{1}\mathbf{b}^T)}{\det(\mathbf{I} - z\mathfrak{A})}, \quad z := \lambda h, \quad \mathbf{1} = [1, \dots, 1]^T \in \mathbb{R}^s.$$

EXAMPLE 7.3.4.5 (Regions of stability for simple implicit RK-SSM) We determine the Butcher schemes (7.3.3.3) for simple implicit RK-SSM and apply the formula from Thm. 7.3.4.4 to compute their stability functions.

- Implicit Euler method:

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \quad \Rightarrow \quad S(z) = \frac{1}{1-z}.$$

- Implicit midpoint method:

$$\begin{array}{c|cc} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array} \quad \Rightarrow \quad S(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}.$$

Their regions of stability \mathcal{S}_Ψ as defined in Def. 7.1.0.51,

$$\mathcal{S}_\Psi := \{z \in \mathbb{C}: |S(z)| < 1\} \subset \mathbb{C},$$

can easily found from the respective stability functions:

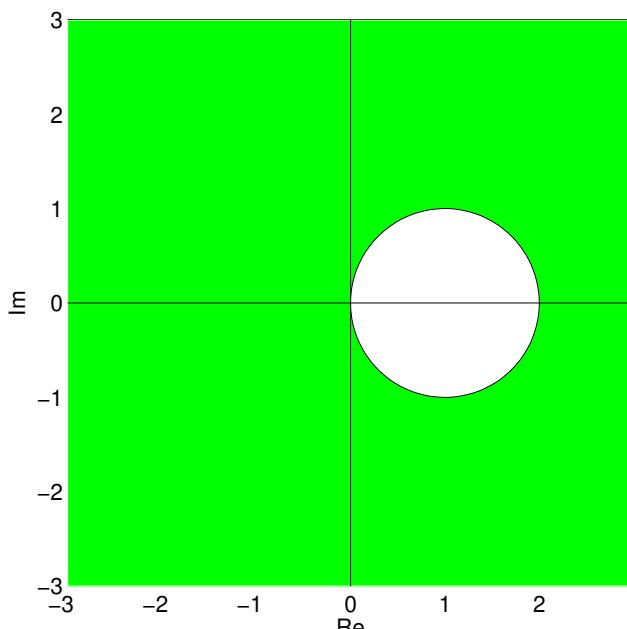


Fig. 349

S_Ψ : implicit Euler method (6.2.2.2)

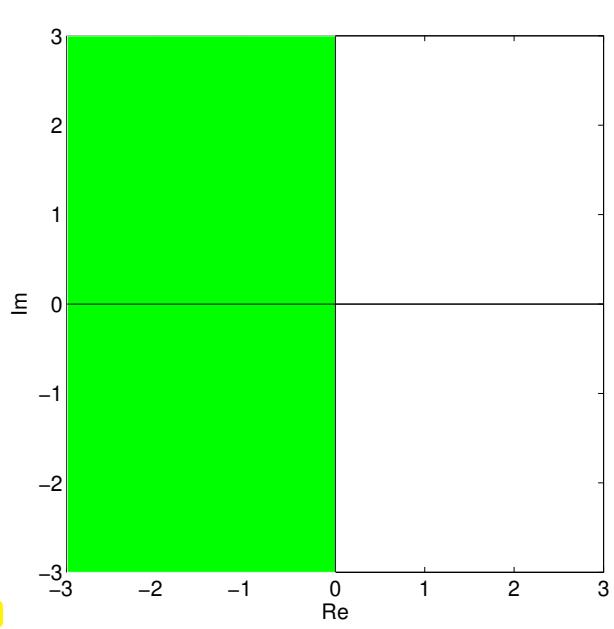


Fig. 350

S_Ψ : implicit midpoint method (6.2.3.3)

We see that in both cases $|S(z)| < 1$, if $\text{Re } z < 0$. □

From the determinant formula for the stability function $S(z)$ we can conclude a generalization of Cor. 7.1.0.20.

Corollary 7.3.4.6. Rational stability function of general RK-SSM

For a consistent (\rightarrow Def. 6.3.1.11) s -stage general Runge-Kutta single step method according to Def. 7.3.3.1 the stability function S is a non-constant **rational function** of the form $S(z) = \frac{P(z)}{Q(z)}$ with polynomials $P \in \mathcal{P}_s$, $Q \in \mathcal{P}_s$.

Of course, a rational function $z \mapsto S(z)$ can satisfy $\lim_{|z| \rightarrow \infty} |S(z)| < 1$ as we have seen in Ex. 7.3.4.5. As a consequence, the region of stability for implicit RK-SSM need not be bounded.

§7.3.4.7 (A-stability) A general RK-SSM with stability function S applied to the scalar linear IVP $\dot{y} = \lambda y$, $y(0) = y_0 \in \mathbb{C}$, $\lambda \in \mathbb{C}$, with uniform timestep $h > 0$ will yield the sequence $(y_k)_{k=0}^\infty$ defined by

$$y_k = S(z)^k y_0 \quad , \quad z = \lambda h . \quad (7.3.4.8)$$

Hence, the next property of a RK-SSM guarantees that the sequence of approximations decays exponentially whenever the exact solution of the model problem IVP (7.1.0.5) does so.

Definition 7.3.4.9. A-stability of a Runge-Kutta single step method

A Runge-Kutta single step method with stability function S is **A-stable**, if

$$\mathbb{C}^- := \{z \in \mathbb{C}: \text{Re } z < 0\} \subset \mathcal{S}_\Psi . \quad (\mathcal{S}_\Psi \hat{=} \text{region of stability Def. 7.1.0.51})$$

From Ex. 7.3.4.5 we conclude that both the implicit Euler method and the implicit midpoint method are

A-stable.

A-stable Runge-Kutta single step methods will not be affected by stability induced timestep constraints when applied to **stiff** IVP (\rightarrow Notion 7.2.0.7).

§7.3.4.10 (“Ideal” region of stability) In order to reproduce the qualitative behavior of the exact solution, a single step method when applied to the scalar linear IVP $\dot{y} = \lambda y$, $y(0) = y_0 \in \mathbb{C}$, $\lambda \in \mathbb{C}$, with uniform timestep $h > 0$,

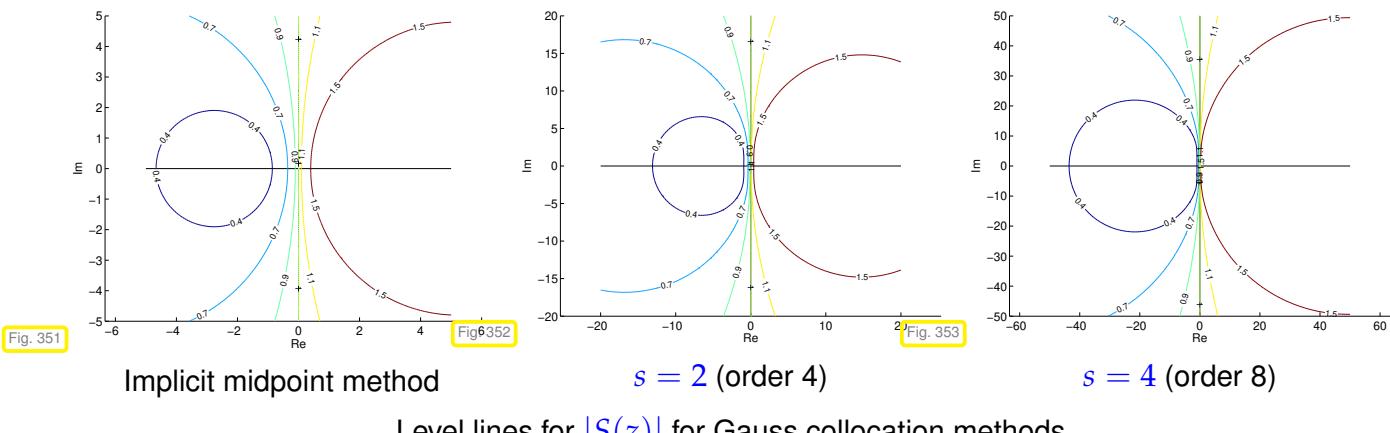
- should yield an *exponentially decaying* sequence $(y_k)_{k=0}^{\infty}$, whenever $\operatorname{Re} \lambda < 0$,
- should produce an *exponentially increasing* sequence $(y_k)_{k=0}^{\infty}$, whenever $\operatorname{Re} \lambda > 0$.

Thus, in light of (7.3.4.8), we agree that the stability if

$$\text{“ideal” region of stability is } \mathcal{S}_{\Psi} = \mathbb{C}^- . \quad (7.3.4.11)$$

Are there RK-SSMs that can boast of an ideal region of stability?

Regions of stability of Gauss collocation single step methods, see Exp. 9.2.6.27:



Theorem 7.3.4.12. Region of stability of Gauss collocation single step methods [DB02, Satz 6.44]

s -stage **Gauss collocation single step methods** defined by (7.3.2.6) with the nodes c_s given by the s Gauss points on $[0, 1]$, feature the “ideal” stability domain:

$$\mathcal{S}_{\Psi} = \mathbb{C}^- . \quad (7.3.4.11)$$

In particular, all Gauss collocation single step methods are A-stable.

EXPERIMENT 7.3.4.13 (Implicit RK-SSMs for stiff IVP) We consider the stiff IVP

$$\dot{y} = -\lambda y + \beta \sin(2\pi t), \quad \lambda = 10^6, \beta = 10^6, \quad y(0) = 1,$$

whose solution essentially is the smooth function $t \mapsto \sin(2\pi t)$. Applying the criteria (7.2.0.13) and (7.2.0.14) we immediately see that this IVP is extremely stiff.

We solve it with different implicit RK-SSM on $[0, 1]$ with large uniform timestep $h = \frac{1}{20}$.

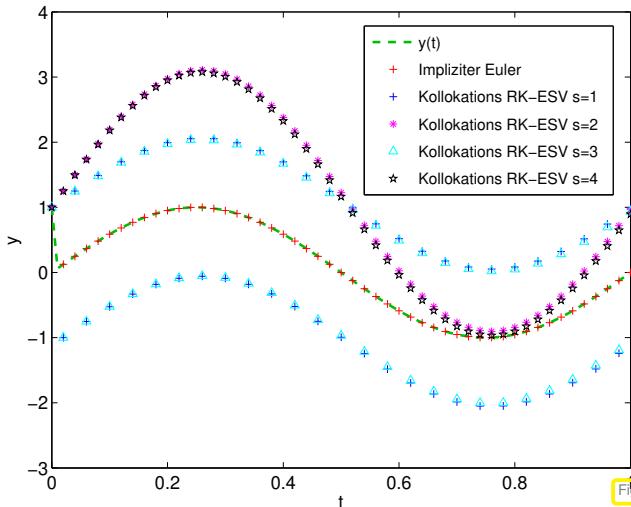
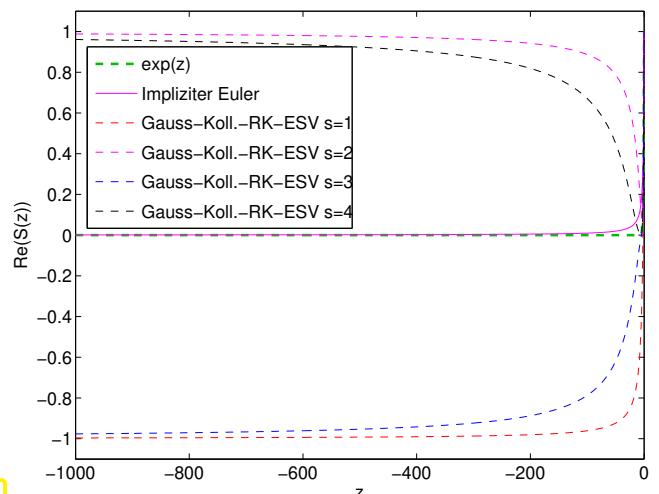


Fig. 354

Fig. 355

Solutions by RK-SSMs

Stability functions on \mathbb{R}^+

We observe that Gauss collocation RK-SSMs incur a huge discretization error, whereas the simple implicit Euler method provides a perfect approximation!

Explanation: The stability functions for Gauss collocation RK-SSMs satisfy

$$\lim_{|z| \rightarrow \infty} |S(z)| = 1 .$$

Hence, when they are applied to $\dot{y} = \lambda y$ with extremely large (in modulus) $\lambda < 0$, they will produce sequences that decay only very slowly or even oscillate, which misses the very rapid decay of the exact solution. The stability function for the implicit Euler method is $S(z) = (1 - z)^{-1}$ and satisfies $\lim_{|z| \rightarrow \infty} S(z) = 0$, which will mean a fast exponential decay of the y_k . \square

§7.3.4.14 (L-stability) In light of what we learned in the previous experiment we can now state what we expect from the stability function of a Runge-Kutta method that is suitable for stiff IVP (\rightarrow Notion 7.2.0.7):

Definition 7.3.4.15. L-stable Runge-Kutta method \rightarrow [Han02, Ch. 77]

A Runge-Kutta method (\rightarrow Def. 7.3.3.1) is **L-stable/asymptotically stable**, if its stability function (\rightarrow Thm. 7.3.4.4) satisfies

$$(i) \quad \operatorname{Re} z < 0 \Rightarrow |S(z)| < 1 , \quad (7.3.4.16)$$

$$(ii) \quad \lim_{\operatorname{Re} z \rightarrow -\infty} S(z) = 0 . \quad (7.3.4.17)$$

Remember:

L-stable \Leftrightarrow A-stable & “ $S(-\infty) = 0$ ” \square

Remark 7.3.4.18 (Necessary condition for L-stability of Runge-Kutta methods)

Consider a Runge-Kutta single step method (\rightarrow Def. 7.3.3.1) described by the Butcher scheme $\begin{array}{c|cc} c & \alpha \\ \hline b^T & \end{array}$.

Assume that $\alpha \in \mathbb{R}^{s,s}$ is regular, which can be fulfilled only for an implicit RK-SSM.

For a rational function $S(z) = \frac{P(z)}{Q(z)}$ the limit for $|z| \rightarrow \infty$ exists and can easily be expressed by the leading coefficients of the polynomials P and Q :

$$\text{Thm. 7.3.4.4} \Rightarrow S(-\infty) = 1 - \mathbf{b}^T \mathfrak{A}^{-1} \mathbf{1}. \quad (7.3.4.19)$$

► If $\mathbf{b}^T = (\mathfrak{A})_{:,j}^T$ (row of \mathfrak{A}) $\Rightarrow S(-\infty) = 0$. (7.3.4.20)

Butcher scheme (7.3.3.3) for L-stable RK-methods, see Def. 9.2.7.46

$$\triangleright \begin{array}{c|c} \mathbf{c} & \mathfrak{A} \\ \hline \mathbf{b}^T & \end{array} := \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_{s-1} & a_{s-1,1} & \cdots & a_{s-1,s} \\ \hline 1 & b_1 & \cdots & b_s \\ \hline & b_1 & \cdots & b_s \end{array}.$$

A closer look at the coefficient formulas of (7.3.2.6) reveals that the algebraic condition (9.2.7.48) will automatically be satisfied for a collocation single step method with $c_s = 1$! □

EXAMPLE 7.3.4.21 (L-stable implicit Runge-Kutta methods) There is a family of s -point quadrature formulas on $[0, 1]$ with a node located in 1 and (maximal) order $2s - 1$: **Gauss-Radau formulas**. They induce the **L-stable** Gauss-Radau collocation single step methods of order $2s - 1$ according to Thm. 9.2.6.29.

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

$$\begin{array}{c|ccc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ \hline 1 & \frac{3}{4} & \frac{1}{4} \\ \hline \frac{3}{4} & \frac{1}{4} \end{array}$$

Implicit Euler method

Radau RK-SSM, order 3

$$\begin{array}{c|cccc} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{4+\sqrt{6}}{10} & \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ \hline 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array}$$

Radau RK-SSM, order 5

The stability functions of s -stage Gauss-Radau collocation SSMs are rational functions of the form

$$S(z) = \frac{P(z)}{Q(z)}, \quad P \in \mathcal{P}_{s-1}, Q \in \mathcal{P}_s.$$

Beware that also " $S(\infty) = 0$ ", which means that Gauss-Radau methods when applied to problems with fast exponential blow-up may produce a spurious decaying solution.

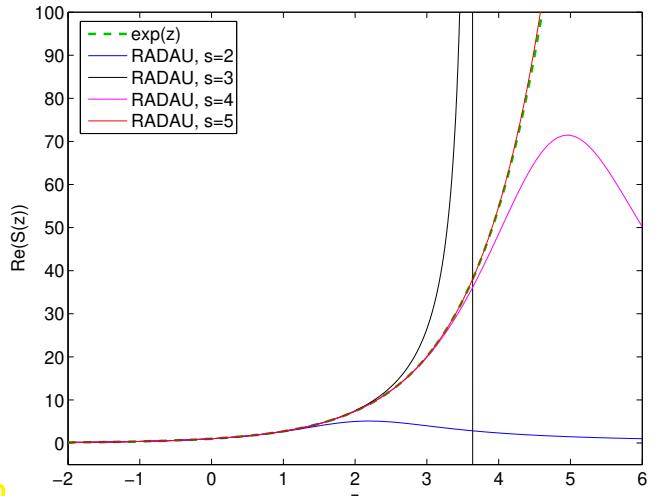
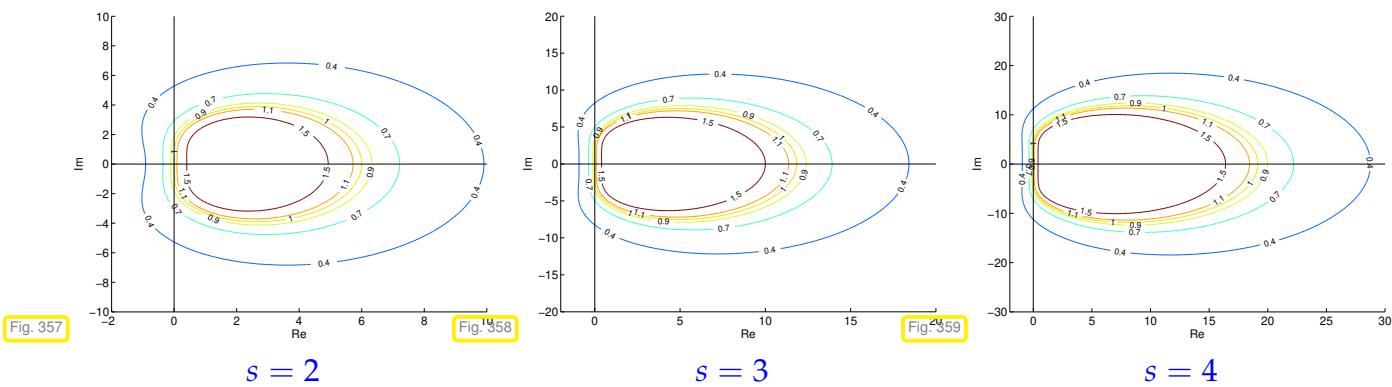


Fig. 356

Level lines of stability functions of s -stage Gauss-Radau collocation SSMs:

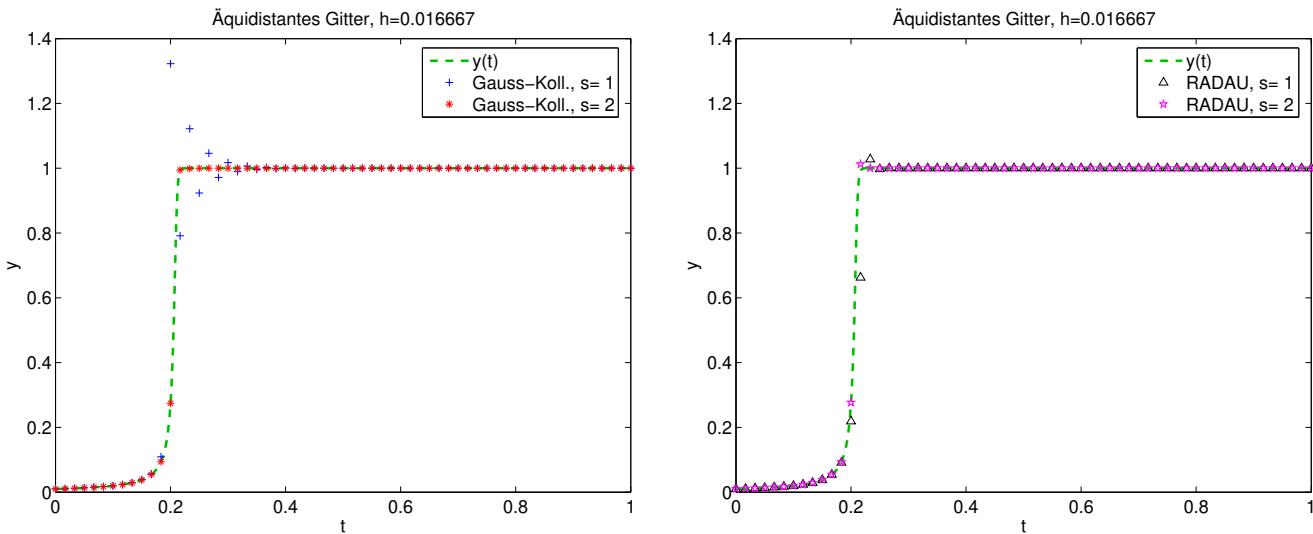


Further information about Radau-Runge-Kutta single step methods can be found in [Han02, Ch. 79]. \square

EXPERIMENT 7.3.4.22 (Gauss-Radau collocation SSM for stiff IVP) We revisit the stiff IVP from Ex. 7.0.0.1

$$\dot{y}(t) = \lambda y^2(1 - y), \quad \lambda = 500, \quad y(0) = \frac{1}{100}.$$

We compare the sequences generated by 1-stage and 2-stage Gauss collocation and Gauss-Radau collocation SSMs, respectively (uniform timestep).



The 2nd-order Gauss collocation SSM (implicit midpoint method) suffers from spurious oscillations when homing in on the stable stationary state $y = 1$. The explanation from Exp. 7.3.4.13 also applies to this example.

The fourth-order Gauss method is already so accurate that potential overshoots when approaching $y = 1$ are damped fast enough. \square

Review question(s) 7.3.4.23 (Implicit Runge-Kutta single-step methods)



7.4 Semi-Implicit Runge-Kutta Methods



Video tutorial for Section 7.5: Splitting Methods: (34 minutes) [Download link, tablet notes](#)

From Section 7.3.3 recall the formulas for general/implicit Runge-Kutta single-step methods for the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$:

Definition 7.3.3.1. General Runge-Kutta single-step method

For $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, an s -stage Runge-Kutta single step method (RK-SSM) for the IVP (6.1.3.2) is defined by

$$\mathbf{k}_i := \mathbf{f}(t_0 + c_i h, \mathbf{y}_0 + h \sum_{j=1}^s a_{ij} \mathbf{k}_j), \quad i = 1, \dots, s, \quad \mathbf{y}_1 := \mathbf{y}_0 + h \sum_{i=1}^s b_i \mathbf{k}_i.$$

As before, the vectors $\mathbf{k}_i \in \mathbb{R}^N$ are called **increments**.



The equations fixing the increments $\mathbf{k}_i \in \mathbb{R}^N$, $i = 1, \dots, s$, for an s -stage implicit RK-method constitute a (Non-)linear system of equations with $s \cdot N$ unknowns.



Several expensive (Newton) iterations needed to find \mathbf{k}_i ?

Remember that we compute approximate solutions anyway, and the increments *are weighted with the stepsize $h \ll 1$* , see Def. 7.3.3.1. So there is no point in determining them with high accuracy!



Idea: Use only a fixed *small* number of Newton steps to solve for the \mathbf{k}_i , $i = 1, \dots, s$.

Extreme case: use only a single Newton step! Let's try.

EXAMPLE 7.4.0.1 (Semi-implicit Euler single-step method) We apply the above idea to the implicit Euler method introduced in Section 6.2.2. For the sake of simplicity we consider the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$.

The recursion for the implicit Euler method with (local) stepsize $h > 0$ is

$$\mathbf{y}_{k+1} := \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_{k+1}). \quad (6.2.2.2)$$

We recast it as a non-linear system of N equations in “standard form $F(\mathbf{x}) = \mathbf{0}$ ”:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_{k+1}) \Leftrightarrow F(\mathbf{y}_{k+1}) := \mathbf{y}_{k+1} - h\mathbf{f}(\mathbf{y}_{k+1}) - \mathbf{y}_k = \mathbf{0}.$$

A single Newton step [Hip19, ??] applied to $F(\mathbf{y}) = \mathbf{0}$ with the natural initial guess \mathbf{y}_k yields

$$\mathbf{y}_{k+1} = \mathbf{y}_k - D\mathbf{f}(\mathbf{y}_k)^{-1}F(\mathbf{y}_k) = \mathbf{y}_k + (\mathbf{I} - hD\mathbf{f}(\mathbf{y}_k))^{-1}h\mathbf{f}(\mathbf{y}_k). \quad (7.4.0.2)$$

This defines the recursion for the **semi-implicit Euler method**.

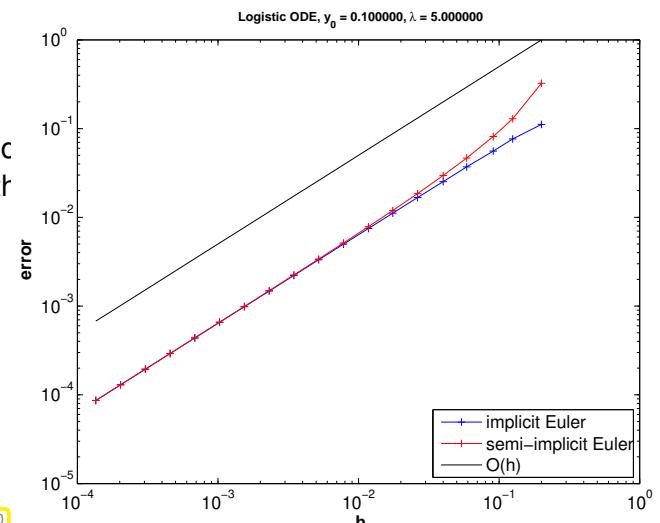
Note that for a linear ODE with $\mathbf{f}(\mathbf{y}) = \mathbf{M}\mathbf{y}$, $\mathbf{M} \in \mathbb{R}^{N,N}$, we recover the original implicit Euler method! ↴

EXPERIMENT 7.4.0.3 (Empiric convergence of semi-implicit Euler single-step method)

- ◆ We consider an Initial value problem for logistic ODE, see Ex. 6.1.2.1

$$\dot{y} = \lambda y(1 - y), \quad y(0) = 0.1, \quad \lambda = 5.$$

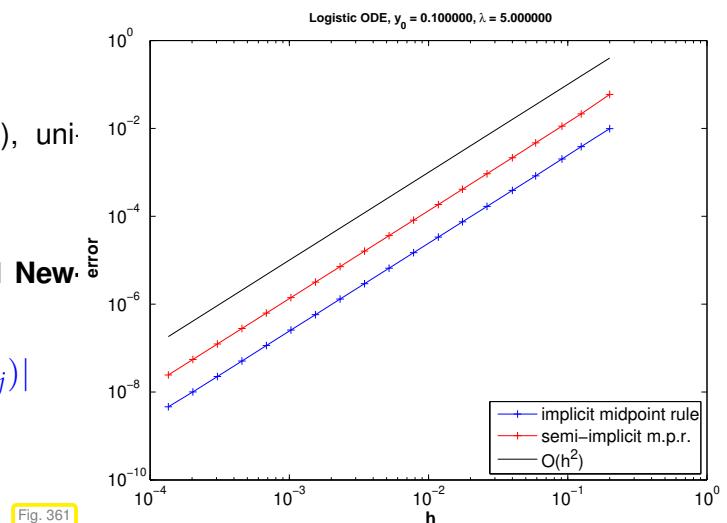
- ◆ We run the implicit Euler method (6.2.2.2) and the semi-implicit Euler method (7.4.0.2) with uniform timestep $h = 1/n$,
 $n \in \{5, 8, 11, 17, 25, 38, 57, 85, 128, 192, 288, 432, 649, 973, 1460, 2189, 3284, 4926, 7389\}$.
- ◆ Measured error $\text{err} = \max_{j=1,\dots,n} |y_j - y(t_j)|$



We observe that the approximate solution of the defining equation for y_{k+1} by a single Newton step preserves the 1st-order convergence of the implicit Newton method. Also the semi-implicit Euler methods seems to be of first order. □

EXPERIMENT 7.4.0.4 (Convergence of semi-implicit midpoint method) Again, we tackle the IVP from Exp. 7.4.0.3.

- ◆ Now, implicit midpoint method (6.2.3.3), uniform timesteps $h = 1/n$ as above
& approximate computation of y_{k+1} by **1 Newton step**, initial guess y_k
- ◆ Measured error $\text{err} = \max_{j=1,\dots,n} |y_j - y(t_j)|$



We still observe second-order convergence! □

Try: Use **linearized increment equations** for implicit RK-SSM



$$\mathbf{k}_i := \mathbf{f}(\mathbf{y}_0 + h \sum_{j=1}^s a_{ij} \mathbf{k}_j), \quad i = 1, \dots, s$$

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}_0) + h D\mathbf{f}(\mathbf{y}_0) \left(\sum_{j=1}^s a_{ij} \mathbf{k}_j \right), \quad i = 1, \dots, s. \quad (7.4.0.5)$$

The good news is that all results about stability derived from model problem analysis (→ Section 7.1)

remain valid despite linearization of the increment equations:

Linearization does nothing for linear ODEs \Rightarrow stability function (\rightarrow Thm. 7.3.4.4) not affected!

The bad news is that the preservation of the order observed in Exp. 7.4.0.3 will no longer hold in the general case.

EXPERIMENT 7.4.0.6 (Convergence of naive semi-implicit Radau method)

- ◆ We consider an IVP for the logistic ODE from Ex. 6.1.2.1:

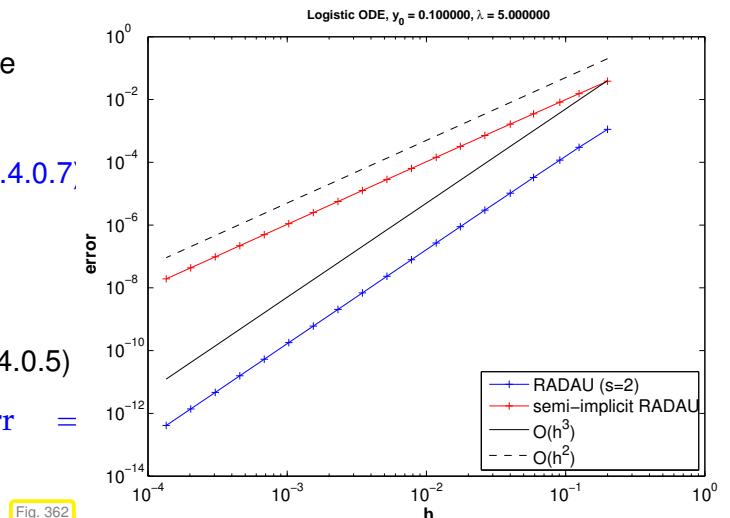
$$\dot{y} = \lambda y(1 - y), \quad y(0) = 0.1, \quad \lambda = 5.$$

- ◆ 2-stage Radau RK-SSM, Butcher scheme

$$\begin{array}{c|cc} & \frac{5}{12} & -\frac{1}{12} \\ \frac{1}{3} & \frac{3}{4} & \frac{1}{4} \\ \hline 1 & \frac{3}{4} & \frac{1}{4} \end{array}, \quad (7.4.0.7)$$

order = 3, see Ex. 7.3.4.21.

- ◆ Increments from linearized equations (7.4.0.5)
- ◆ We monitor the error through $\text{err} = \max_{j=1,\dots,n} |y_j - y(t_j)|$



Loss of order due to linearization !

§7.4.0.8 (Rosenbrock-Wanner methods) We have just seen that the simple linearization according to (7.4.0.5) will degrade the order of implicit RK-SSMs and leads to a substantial loss of accuracy. This is not an option.

Yet, the idea behind (7.4.0.5) has been refined. One does not start from a known RK-SSM, but introduces general coefficients for structurally linear increment equations.

► Class of s -stage **semi-implicit (linearly implicit) Runge-Kutta methods** (Rosenbrock-Wanner (ROW) methods):

$$\begin{aligned} (\mathbf{I} - ha_{ii}\mathbf{J})\mathbf{k}_i &= \mathbf{f}(\mathbf{y}_0 + h \sum_{j=1}^{i-1} (a_{ij} + d_{ij})\mathbf{k}_j) - h\mathbf{J} \sum_{j=1}^{i-1} d_{ij}\mathbf{k}_j, \quad \mathbf{J} = \mathbf{Df}(\mathbf{y}_0), \\ \mathbf{y}_1 &:= \mathbf{y}_0 + h \sum_{j=1}^s b_j \mathbf{k}_j. \end{aligned} \quad (7.4.0.9)$$

Then the coefficients a_{ij} , d_{ij} , and b_i are determined from order conditions by solving large non-linear systems of equations.

In each step s linear systems with coefficient matrices $\mathbf{I} - ha_{ii}\mathbf{J}$ have to be solved. For methods used in practice one often demands that $a_{ii} = \gamma$ for all $i = 1, \dots, s$. As a consequence, we have to solve s linear systems with the same coefficient matrix $\mathbf{I} - h\gamma\mathbf{J} \in \mathbb{R}^{N,N}$, which permits us to reuse LU-factorizations, see [Hip19, ??].



Supplementary literature. A related discussion can be found in [Han02, Ch. 80].

Review question(s) 7.4.0.10 (Semi-implicit Runge-Kutta single-step methods)

(Q7.4.0.10.A) [Semi-implicit midpoint method] The implicit midpoint single-step method applied to the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ and with timestep h leads to the recursion

$$\mathbf{y}_{k+1}: \quad \mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}\left(\frac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1})\right).$$

Derive the defining equation of the semi-implicit variant, which arises from solving the defining equation for \mathbf{y}_{k+1} by a single Newton step with initial guess \mathbf{y}_k .

(Q7.4.0.10.B) [Stability function of ROW-SSM] A Rosenbrock-Wanner (ROW) single-step method for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ can be defined by

$$\begin{aligned} (\mathbf{I} - ha_{ii}\mathbf{J})\mathbf{k}_i &= \mathbf{f}(\mathbf{y}_0 + h \sum_{j=1}^{i-1} (a_{ij} + d_{ij})\mathbf{k}_j) - h\mathbf{J} \sum_{j=1}^{i-1} d_{ij}\mathbf{k}_j, \quad \mathbf{J} = \mathbf{Df}(\mathbf{y}_0), \\ \mathbf{y}_1 &:= \mathbf{y}_0 + h \sum_{j=1}^s b_j \mathbf{k}_j. \end{aligned} \tag{7.4.0.9}$$

Derive its **stability functions** for $s = 2$.

△

7.5 Splitting Methods



Video tutorial for Section 7.5: Splitting Methods: (34 minutes) [Download link](#), [tablet notes](#)

§7.5.0.1 (Splitting idea: composition of partial evolutions) Many relevant ordinary differential equations feature a right hand side function that is the sum to two (or more) terms. Consider an autonomous IVP with a right hand side function that can be split in an additive fashion:

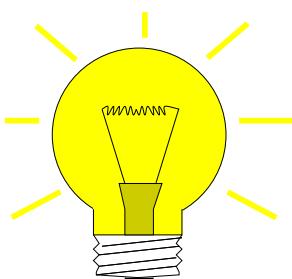
$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) + \mathbf{g}(\mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0, \tag{7.5.0.2}$$

with $\mathbf{f} : D \subset \mathbb{R}^N \mapsto \mathbb{R}^N$, $\mathbf{g} : D \subset \mathbb{R}^N \mapsto \mathbb{R}^N$ “sufficiently smooth”, locally Lipschitz continuous (→ Def. 6.1.3.12).

Let us introduce the evolution operators (→ Def. 6.1.4.3) for both summands:

$$\begin{array}{ll} \text{(Continuous) evolution maps:} & \Phi_f^t \leftrightarrow \text{ODE } \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \\ & \Phi_g^t \leftrightarrow \text{ODE } \dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}). \end{array}$$

Temporarily we assume that both Φ_f^t , Φ_g^t are available in the form of analytic formulas or highly accurate approximations.

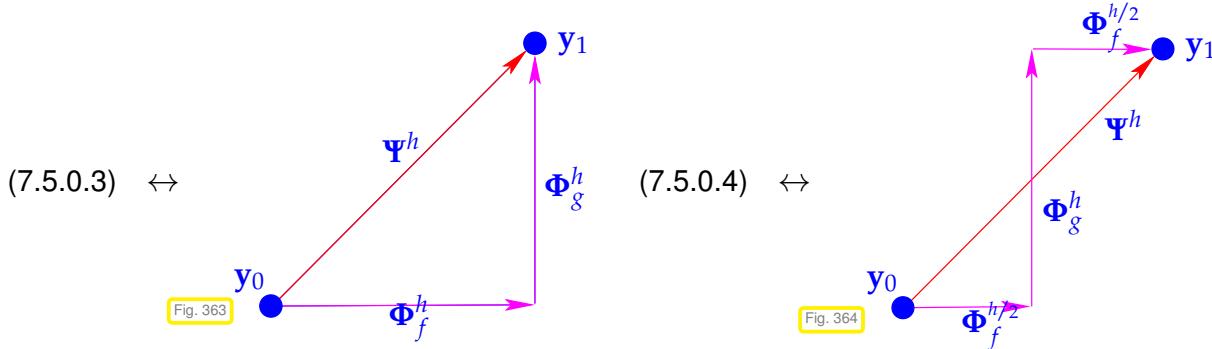


Idea: Build single step methods (→ Def. 6.3.1.4) based on the following discrete evolutions

$$\text{Lie-Trotter splitting: } \Psi^h = \Phi_g^h \circ \Phi_f^h, \tag{7.5.0.3}$$

$$\text{Strang splitting: } \Psi^h = \Phi_f^{h/2} \circ \Phi_g^h \circ \Phi_f^{h/2}. \tag{7.5.0.4}$$

These splittings are easily remembered in graphical form:



Note that over many timesteps the Strang splitting approach is not more expensive than Lie-Trotter splitting, because the actual implementation of (7.5.0.4) should be done as follows:

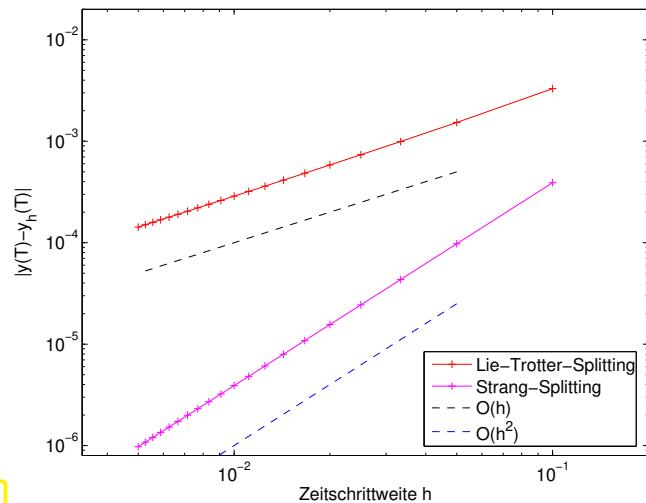
$$\begin{aligned} \mathbf{y}_{1/2} &:= \Phi_f^{h/2}, & \mathbf{y}_1 &:= \Phi_g^h \mathbf{y}_{1/2}, \\ \mathbf{y}_{3/2} &:= \Phi_f^h \mathbf{y}_1, & \mathbf{y}_2 &:= \Phi_g^h \mathbf{y}_{3/2}, \\ \mathbf{y}_{5/2} &:= \Phi_f^h \mathbf{y}_2, & \mathbf{y}_3 &:= \Phi_g^h \mathbf{y}_{5/2}, \\ &\vdots & &\vdots, \end{aligned}$$

because $\Phi_f^{h/2} \circ \Phi_f^{h/2} = \Phi_f^h$. This means that a Strang splitting SSM differs from a Lie-Trotter splitting SSM in the first and the last step only. \square

EXPERIMENT 7.5.0.5 (Convergence of simple splitting methods) We consider the following IVP whose right hand side function is the sum of two functions for which the ODEs can be solved analytically:

$$\dot{y} = \underbrace{\lambda y(1-y)}_{=:f(y)} + \underbrace{\sqrt{1-y^2}}_{=:g(y)}, \quad y(0) = 0.$$

- $\Phi_f^t y = \frac{1}{1 + (y^{-1} - 1)e^{-\lambda t}}, t > 0, y \in [0, 1]$ (logistic ODE (6.1.2.2))
- $\Phi_g^t y = \begin{cases} \sin(t + \arcsin(y)) & , \text{if } t + \arcsin(y) < \frac{\pi}{2}, \\ 1 & , \text{else,} \end{cases} t > 0, y \in [0, 1].$



Numerical experiment:

For $T = 1$, $\lambda = 1$, we compare the two splitting methods for uniform timesteps with a very accurate reference solution obtained in MATLAB by

```
f=@(t,x) lambda*x*(1-x)+sqrt(1-x^2);
options=odeset('reltol',1.0e-10,... 
    'abstol',1.0e-12);
[t,yex]=ode45(f,[0,1],y0,options);
```

\square Error at final time $T = 1$

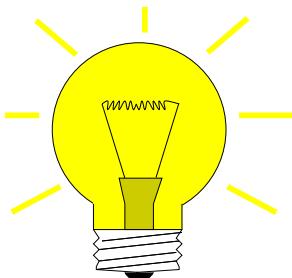
We observe algebraic convergence of the two splitting methods, order 1 for (7.5.0.3), oder 2 for (7.5.0.4). \square

The observation made in Exp. 7.5.0.5 reflects a general truth:

Theorem 7.5.0.6. Order of simple splitting methods

Die single step methods defined by (7.5.0.3) or (7.5.0.4) are of order (\rightarrow Def. 6.3.2.8) 1 and 2, respectively.

§7.5.0.7 (Inexact splitting methods) Of course, the assumption that $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ and $\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y})$ can be solved exactly will hardly ever be met. However, it should be clear that a “sufficiently accurate” approximation of the evolution maps Φ_g^h and Φ_f^h is all we need



Idea: In (7.5.0.3)/(7.5.0.4) replace

$$\begin{array}{ccc} \text{exact evolutions} & \longrightarrow & \text{discrete evolutions} \\ \Phi_g^h, \Phi_f^h & \longrightarrow & \Psi_g^h, \Psi_f^h \end{array}$$

EXPERIMENT 7.5.0.8 (Convergence of inexact simple splitting methods) Again we consider the IVP of Exp. 7.5.0.5 and inexact splitting methods based on different single step methods for the two ODE corresponding to the summands.

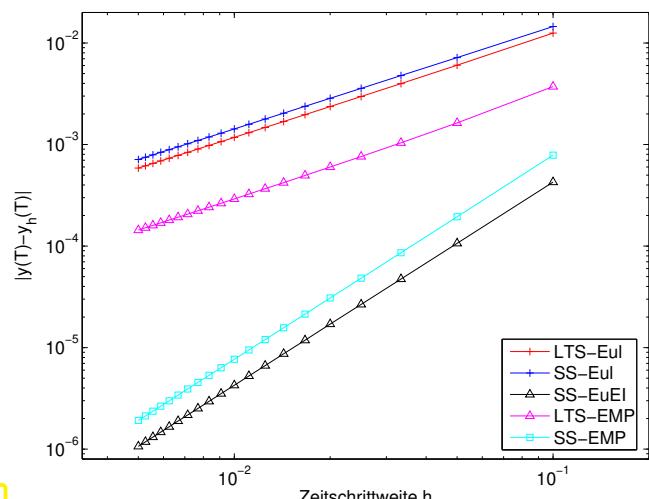


Fig. 366

LTS-Eul	explicit Euler method (6.2.1.4) $\rightarrow \Psi_{h,g}^h$,
SS-Eul	$\Psi_{h,f}^h$ + Lie-Trotter splitting (7.5.0.3)
SS-EuEI	explicit Euler method (6.2.1.4) $\rightarrow \Psi_{h,g}^h$, $\Psi_{h,f}^h$ + Strang splitting (7.5.0.4)
LTS-EMP	Strang splitting (7.5.0.4): explicit Euler method (6.2.1.4) \circ exact evolution Φ_g^h \circ implicit Euler method (6.2.2.2)
SS-EMP	explicit midpoint method (6.2.3.3) $\rightarrow \Psi_{h,g}^h$, $\Psi_{h,f}^h$ + Lie-Trotter splitting (7.5.0.3)
	explicit midpoint method (6.4.0.7) $\rightarrow \Psi_{h,g}^h$, $\Psi_{h,f}^h$ + Strang splitting (7.5.0.4)

- ☞ The order of splitting methods may be (but need not be) limited by the order of the SSMs used for Φ_f^h, Φ_g^h .

§7.5.0.9 (Application of splitting methods) In the following situation the use splitting methods seems advisable:

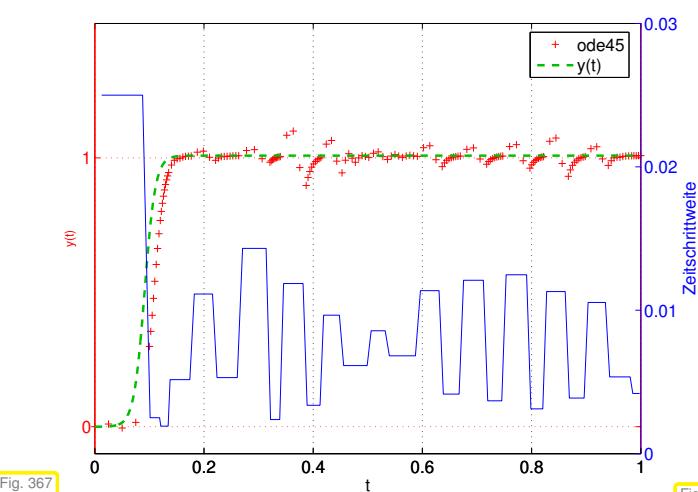
“Splittable” ODEs

$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) + \mathbf{g}(\mathbf{y})$ “difficult” : $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \rightarrow$ stiff, but with an analytic solution
(e.g., stiff \rightarrow Section 7.2) $\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y})$ “easy”, amenable to explicit integration.

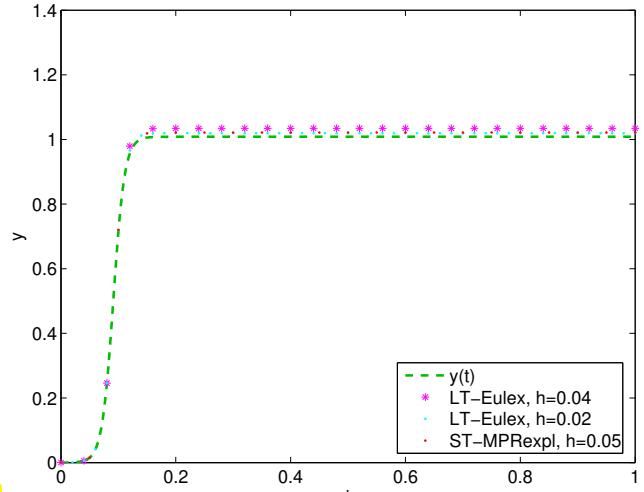
EXPERIMENT 7.5.0.11 (Splitting off stiff components) Recall Ex. 7.0.0.1 and the IVP studied there:

$$\text{IVP} \quad \dot{y} = \lambda y(1 - y) + \alpha \sin(y), \quad \lambda = 100, \quad \alpha = 1, \quad y(0) = 10^{-4}.$$

small perturbation



Solution by **ode45**, see Ex. 7.0.0.1



inexact splitting method: solution (y_k)

	ode45:	152
Total number of timesteps	LT-Eulex, $h = 0.04$:	25
	LT-Eulex, $h = 0.02$:	50
	ST-MPRExpl, $h = 0.05$:	20

Details of the methods:

LT-Eulex: $\dot{y} = \lambda y(1 - y) \rightarrow$ exact evolution, $\dot{y} = \alpha \sin y \rightarrow$ expl. Euler (6.2.1.4) & Lie-Trotter splitting (7.5.0.3)

ST-MPRExpl: $\dot{y} = \lambda y(1 - y) \rightarrow$ exact evolution, $\dot{y} = \alpha \sin y \rightarrow$ expl. midpoint rule (6.4.0.7) & Strang splitting (7.5.0.4)

We observe that this splitting scheme can cope well with the stiffness of the problem, because the stiff term on the right hand side is integrated exactly. \square

EXAMPLE 7.5.0.12 (Splitting linear and decoupling terms) In the numerical treatment of partial differential equation one commonly encounters ODEs of the form

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) := -\mathbf{A}\mathbf{y} + \begin{bmatrix} g(y_1) \\ \vdots \\ g(y_N) \end{bmatrix}, \quad \mathbf{A} = \mathbf{A}^\top \in \mathbb{R}^{N,N} \quad \text{positive definite} (\rightarrow [\text{Hip19}, ??]), \quad (7.5.0.13)$$

with state space $D = \mathbb{R}^N$, where $\lambda_{\min}(\mathbf{A}) \approx 1$, $\lambda_{\max}(\mathbf{A}) \approx N^2$, and the derivative of $g : \mathbb{R} \rightarrow \mathbb{R}$ is bounded. Then IVPs for (7.5.0.13) will be stiff, since the Jacobian

$$\mathbf{Df}(\mathbf{y}) = -\mathbf{A} + \begin{bmatrix} g'(y_1) & & \\ & \ddots & \\ & & g'(y_N) \end{bmatrix} \in \mathbb{R}^{N,N}$$

will have eigenvalues “close to zero” and others that are large (in modulus) and negative. Hence, $\mathbf{Df}(\mathbf{y})$

will satisfy the criteria (7.2.0.13) and (7.2.0.14) for any state $\mathbf{y} \in \mathbb{R}^N$.

The natural splitting is

$$\mathbf{f}(\mathbf{y}) = \mathbf{g}(\mathbf{y}) + \mathbf{q}(\mathbf{y}) \quad \text{with} \quad \mathbf{g}(\mathbf{y}) := -\mathbf{A}\mathbf{y}, \quad \mathbf{q}(\mathbf{y}) := \begin{bmatrix} g(y_1) \\ \vdots \\ g(y_N) \end{bmatrix}.$$

- For the linear ODE $\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y})$ we have to use and L-stable (\rightarrow Def. 9.2.7.46) single step method, for instance a second-order *implicit* Runge-Kutta method. Its increments can be obtained by solving a *linear system of equations*, whose coefficient matrix will be the same for every step, if uniform timesteps are used.
- The ODE $\dot{\mathbf{y}} = \mathbf{q}(\mathbf{y})$ boils down to *decoupled* scalar ODEs $\dot{y}_j = g(y_j)$, $j = 1, \dots, N$. For them we can use an inexpensive *explicit RK-SSM* like the explicit trapezoidal method (6.4.0.6). According to our assumptions on g these ODEs are not haunted by stiffness.

□

Review question(s) 7.5.0.14 (Splitting single-step methods)

△

Bibliography

- [DR08] W. Dahmen and A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. Heidelberg: Springer, 2008 (cit. on pp. 496, 499, 514).
- [DB02] P. Deuflhard and F. Bornemann. *Scientific Computing with Ordinary Differential Equations*. 2nd ed. Vol. 42. Texts in Applied Mathematics. New York: Springer, 2002 (cit. on pp. 519, 525).
- [Han02] M. Hanke-Bourgeois. *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*. Mathematische Leitfäden. Stuttgart: B.G. Teubner, 2002 (cit. on pp. 501, 509, 510, 526, 527, 531).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 500, 503, 505, 513, 514, 517–520, 522, 529, 531, 535).
- [NS02] K. Nipp and D. Stoffer. *Lineare Algebra*. 5th ed. Zürich: vdf Hochschulverlag, 2002 (cit. on p. 503).
- [QSS00] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*. Vol. 37. Texts in Applied Mathematics. New York: Springer, 2000 (cit. on pp. 501, 509, 514).
- [Ran15] Joachim Rang. “Improved traditional Rosenbrock-Wanner methods for stiff ODEs and DAEs”. In: *J. Comput. Appl. Math.* 286 (2015), pp. 128–144. DOI: [10.1016/j.cam.2015.03.010](https://doi.org/10.1016/j.cam.2015.03.010) (cit. on p. 528).
- [Str09] M. Struwe. *Analysis für Informatiker*. Lecture notes, ETH Zürich. 2009 (cit. on pp. 505, 512).

Chapter 8

Structure-Preserving Numerical Integration

Chapter 9

Second-Order Linear Evolution Problems

9.1 Time-Dependent Boundary Value Problems

§9.1.0.1 (Introduction) This chapter is devoted to the discretization of boundary value problems for a class of time-dependent partial differential equations. They are specimens of **evolution problems**.

Prerequisite knowledge is

- the theory and variational formulation of 2nd-order elliptic BVP from Chapter 1,
- basic concepts and algorithms for finite elements, Section 2.2, Section 2.5, Section 2.6,
- knowledge about single step methods for ODEs from § 10.2.2.7 and Chapter 7, briefly recapitulated in Section 9.2.6, in case those chapters were not covered in the course.

In particular, we study scalar *linear* partial differential equations for which *one* coordinate direction is special and identified with **time** and denoted by the independent variable t . The other coordinates are regarded as **spatial coordinates** and designated by $\mathbf{x} = (x_1, \dots, x_d)^T$.

Why is time special? It seems to be just another dimension.

! In contrast to space, time has a *direction* from past to future and this makes the temporal direction special. There is **causality**: the state at a paricular time may depend only on the past. ↴

§9.1.0.2 (Outline)

Contents

9.1	Time-Dependent Boundary Value Problems	539
9.2	Parabolic Initial-Boundary Value Problems	541
9.2.1	Heat Equation	541
9.2.2	Heat Equation: Spatial Variational Formulation	545
9.2.3	Stability of Parabolic Evolution Problems	548
9.2.4	Spatial Semi-Discretization: Method of Lines	552
9.2.5	Recalled from Section 6.1: Ordinary Differential Equations	554
9.2.6	Recalled from § 10.2.2.7: Single-Step Methods for Numerical Integration	557
9.2.7	Timestepping for Method-of-Lines ODE	568
9.2.8	Fully Discrete Method of Lines: Convergence	582
9.3	Linear Wave Equations	587

9.3.1	Models for Vibrating Membrane	588
9.3.2	Wave Propagation	593
9.3.3	Method of Lines for Wave Propagation	598
9.3.4	Timestepping for Semi-Discrete Wave Equations	600
9.3.5	The Courant-Friedrichs-Levy (CFL) Condition	607

This chapter exclusively deals with *linear* evolution problems. We can distinguish two fundamental classes, dissipative and conservative evolutions. This is reflected by the structure of the chapter, which comprises two sections, Section 9.2 devoted to dissipative (**parabolic**) evolutions, Section 9.3 addressing conservative (**hyperbolic**) evolutions. Each section first develops variational formulations, then discretization in space, and, finally, discretization in time. Results on convergence are reviewed and discussed. \square

§9.1.0.3 (Space-time domains) For time-dependent PDEs ($x \leftrightarrow$ spatial variable, $t \leftrightarrow$ time variable)

➤ a/the solution will be a “function of space and time”: $u = u(x, t)$

The domain for such PDEs will have *tensor product structure* (tensor product of spatial domain and a bounded time interval):

Computational domain:

$$\tilde{\Omega} := \Omega \times]0, T[\subset \mathbb{R}^{d+1}.$$

➤ $\tilde{\Omega}$ is a **space-time cylinder**

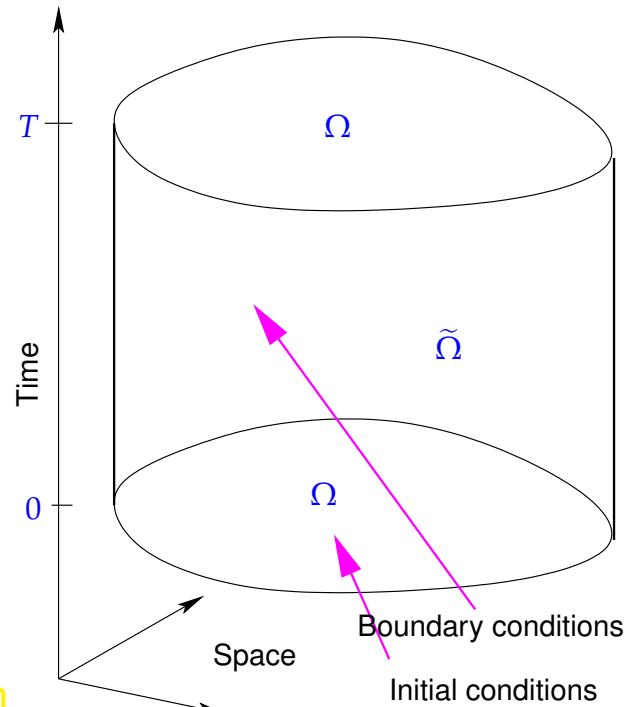
$\Omega \subset \mathbb{R}^d \hat{=} \text{ spatial domain }$ (satisfying assumptions of § 1.2.1.14)

$T > 0 \hat{=} \text{ final time}$

(Setting the initial time to zero is a convention adopted throughout this chapter.)

Data prescribed for u on $\partial\tilde{\Omega}$ have different names:

On $\Omega \times \{0\} \rightarrow$ **initial conditions**,
on $\partial\Omega \times]0, T[\rightarrow$ (spatial) **boundary conditions**.



Remark 9.1.0.4 (Temporally varying spatial domains) The spatial sections of a space-time domain $\tilde{\Omega}$ need not be constant, that is, $\tilde{\Omega}$ can also be of arbitrary shape. However, the mathematical and numerical treatment of this situation is a challenge; even the distinction between initial conditions and boundary conditions becomes blurred. Thus we confine ourselves to space-time cylinders. \square

§9.1.0.5 (Terminology for time-dependent problems) Extending the notion of a “boundary value problem”:

PDE for $u(x, t)$ + initial conditions + boundary conditions

= **evolution problem** for $u : \tilde{\Omega} \rightarrow \mathbb{R}$

Note: No boundary conditions are prescribed on $\Omega \times \{T\}$, that is there are no “final conditions”. This is an immediate consequence of causality: time is supposed to have a “direction” that governs the flow of information in the evolution problem, cf. § 9.1.0.1.



evolution problems (on bounded spatial domains) are also known as
initial-boundary value problems (IBVP).

Remark 9.1.0.6 (Initial time) Why do we always pick initial time $t = 0$ in this chapter?

The modelled physical systems will usually be time-invariant, so that we are free to shift time. Remember the analogous situation with **autonomous** ODE discussed in § 6.1.3.3.

Review question(s) 9.1.0.7 (Time-dependent boundary value problems)

(Q9.1.0.7.A) [Space-time cylinder] Draw a space-time cylinder and mark where initial conditions and (spatial) boundary conditions are imposed.

(Q9.1.0.7.B) [Time-dependent spatial domain] Evolution problems can also be posed on time-dependent spatial domains. For $d = 2$ sketch a generalization of the space-time cylinder for that case.

Outline a “**mapping approach**” that can convert an evolution problem posed on a time-dependent spatial domain into one living on a space-time cylinder.

(Q9.1.0.7.C) [Reversible physical system] You can think of a reversible physical systems as those for which you could not tell whether a video recording of them is played forward or backward.

Give examples of reversible and irreversible physical systems in our everyday world.



9.2 Parabolic Initial-Boundary Value Problems

9.2.1 Heat Equation



Video tutorial for Section 9.2.1: Heat Equation: (18 minutes) [Download link](#), [tablet notes](#)

Section 1.6 treated **stationary** heat conduction: no change of temperature with time (temporal equilibrium). For this situation we derived a mathematical model that boils down to a second order scalar linear elliptic boundary value problem for the temperature $u = u(\mathbf{x})$ as a function of the spatial variable $\mathbf{x} \in \Omega$, see § 1.6.0.7 and Section 1.7 for a discussion of boundary conditions.

Now we consider the evolution (change in time) of a temperature distribution $u = u(\mathbf{x}, t)$ in a solid body occupying a bounded region of space $\Omega \subset \mathbb{R}^d$ over a finite time period $[0, T]$.

§9.2.1.1 (Notations for heat conduction modelling) We use the following symbols in connection with mathematical modelling of transient heat conduction:

$\Omega \subset \mathbb{R}^d$: space occupied by solid body (bounded spatial computational domain),
$x \in \Omega$: spatial independent variable
	: (differential operators acting in space are sometimes tagged with subscript x)
t	: time variable, $\frac{\partial}{\partial t}/\frac{d}{dt} \hat{=} \text{partial/total derivative w.r.t. time}$,
$\kappa = \kappa(x)$: (spatially varying) heat conductivity ($[\kappa] = \frac{W}{Km}$),
$T > 0$: final time for “observation period” $[0, T]$,
$u_0 : \Omega \mapsto \mathbb{R}$: initial temperature distribution in Ω ,
$g : \partial\Omega \times [0, T] \mapsto \mathbb{R}$: surface temperature , varying in space and time: $g = g(x, t)$,
$f : \Omega \times [0, T] \mapsto \mathbb{R}$: time-dependent heat source/sink ($[f] = \frac{W}{m^d}$): $f = f(x, t)$.

Our goal is to derive a PDE governing *transient* heat conduction, a PDE satisfied by the time-dependent temperatur distribution $u = u(x, t)$, on the space-time cylinder $\tilde{\Omega} := \Omega \times [0, T]$. Of course, the tools and concepts from Section 1.6 will be used again: Heat flux (\rightarrow § 1.6.0.1), energy conservation, and a flux law (\rightarrow § 1.6.0.4).

§9.2.1.2 (Derivation of heat equation) For transient heat conduction the energy balance law (1.6.0.3) has to be supplemented by a *storage term* reflecting the fact that heat can accumulate:

Conservation of energy:

$$\frac{d}{dt} \int_V \rho u \, dx + \int_{\partial V} \mathbf{j} \cdot \mathbf{n} \, dS = \int_V f \, dx \quad \text{for all “control volumes” } V. \quad (9.2.1.3)$$

↑ ↑ ↑

energy stored in V power flux through ∂V heat generation in V

$\rho = \rho(x)$: (spatially varying) **heat capacity** ($[\rho] = JK^{-1}$), uniformly positive, cf. (1.6.0.6).

As in § 1.6.0.7, now apply Gauss’ Theorem (\rightarrow Thm. 1.5.2.4)

$$\int_V \operatorname{div} \mathbf{j}(x) \, dx = \int_{\partial V} \mathbf{j}(x) \cdot \mathbf{n}(x) \, dS(x), \quad \mathbf{j} : \Omega \rightarrow \mathbb{R}^d,$$

to the power flux integral in (9.2.1.3). This converts the surface integral to a volume integral over $\operatorname{div} \mathbf{j}$ and we get

$$\frac{d}{dt} \int_V \rho u \, dx + \int_V \operatorname{div} \mathbf{j} \, dx = \int_V f \, dx \quad \text{for all “control volumes” } V$$

We make the casual assumption that all functions are “sufficiently smooth”. Then we appeal to another version of the fundamental lemma of the calculus of variations, see Lemma 1.5.3.4, this time involving piecewise constant test functions:

$$\varphi \in C^0(\bar{\Omega}) \quad \text{and} \quad \int_V \varphi(x) \, dx = 0 \quad \forall V \subset \Omega \quad \Rightarrow \quad \varphi \equiv 0. \quad (9.2.1.4)$$

This permits us to drop integration over control volumes altogether.

► Local form of energy balance law (\Rightarrow **heat equation**)

$$\frac{\partial}{\partial t} (\rho u)(x, t) + (\operatorname{div}_x \mathbf{j})(x, t) = f(x, t) \quad \text{in } \tilde{\Omega}. \quad (9.2.1.5)$$

For standard materials the heat flux is linked to temperature variations by Fourier’s law (\rightarrow § 1.6.0.4):

$$\mathbf{j}(x) = -\kappa(x) \operatorname{grad} u(x), \quad x \in \Omega. \quad (1.6.0.5)$$

From here we let all differential operators like **grad** and **div** act on the spatial independent variable \mathbf{x} . As earlier, the independent variables \mathbf{x} and t will be omitted frequently. Watch out!

Now, plug Fourier's law

$$\mathbf{j}(\mathbf{x}) = -\kappa(\mathbf{x}) \mathbf{grad} u(\mathbf{x}), \quad \mathbf{x} \in \Omega, \quad (1.6.0.5)$$

into the local form of the energy balance law (9.2.1.5).

►
$$\frac{\partial}{\partial t}(\rho u) - \operatorname{div}(\kappa(\mathbf{x}) \mathbf{grad} u) = f \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[. \quad (9.2.1.6)$$

§9.2.1.7 (Evolution problem for heat conduction) As pointed out in § 9.1.0.5 the PDE (9.2.1.6) has to be supplemented with initial conditions for $(\mathbf{x}, t) \in \Omega \times \{0\}$ and boundary conditions for $(\mathbf{x}, t) \in \partial\Omega \times]0, T[$. A simple and intuitive choice is

Dirichlet boundary conditions (fixed surface temperatur) on $\partial\Omega \times]0, T[$:

$$u(\mathbf{x}, t) = g(\mathbf{x}, t) \quad \text{for } (\mathbf{x}, t) \in \partial\Omega \times]0, T[. \quad (9.2.1.8)$$

+ initial conditions for $t = 0$:

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \Omega. \quad (9.2.1.9)$$

Terminology: (9.2.1.6) & (9.2.1.8) & (9.2.1.9) is a specimen of a

2nd-order **parabolic** initial-boundary value problem (IBVP)

§9.2.1.10 ((Spatial) boundary conditions for 2nd-order parabolic IBVPs) As in Section 1.7 we appeal to physical intuition about heat conduction to justify that all of the following spatial boundary conditions make sense for the heat equation (9.2.1.6).

On $\partial\Omega \times]0, T[$ we can impose any of the boundary conditions discussed in Section 1.7:

- Dirichlet boundary conditions $u(\mathbf{x}, t) = g(\mathbf{x}, t)$, see (9.2.1.8) (fixed surface temperature),
- Neumann boundary conditions $\mathbf{j}(\mathbf{x}, t) \cdot \mathbf{n} = -h(\mathbf{x}, t)$ (fixed heat flux through surface),
- radiation boundary conditions $\mathbf{j}(\mathbf{x}, t) \cdot \mathbf{n} = \Psi(u(\mathbf{x}, t))$,

and any combination of these as discussed in Ex. 1.7.0.10, yet, *only one* of them at any part of $\partial\Omega \times]0, T[$, see Rem. 1.7.0.9.

For second order parabolic evolutions we can/must use the *same* spatial boundary conditions as for stationary second order elliptic boundary value problems.

Remark 9.2.1.11 (Compatible boundary and initial data) We consider spatial Dirichlet boundary conditions (9.2.1.8) for the heat equation (9.2.1.6). A physically motivated regularity requirements for the temperature u and the Dirichlet data g we postulate that

u and g are **continuous** in time and space

➤ Natural compatibility requirement at initial time for $u_0 \in C^0(\bar{\Omega})$

$$g(x, 0) = u_0(x) \quad \forall x \in \partial\Omega . \quad (9.2.1.12)$$

↓

Remark 9.2.1.13 (Evolution problems for PDEs and ODEs) Compare the parabolic evolution PDE

$$\frac{\partial u}{\partial t}(x, t) = \operatorname{div}_x(\kappa(x) \operatorname{grad}_x u(x, t)) + f(x, t) \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[,$$

with a “standard” autonomous ordinary differential equations as introduced in Section 6.1.1

$$\frac{d}{dt}y(t) = f(y) \quad , \quad f : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^N .$$

Actually, we can regard the evolution PDE as an ODE, whose state space D is no longer a subset of \mathbb{R}^N , but an infinite-dimensional space of functions $\Omega \rightarrow \mathbb{R}$.

Evolution PDEs can be viewed as ODEs in infinite-dimensional spaces of functions.

↓

Review question(s) 9.2.1.14 (Heat equation)

(Q9.2.1.14.A) Cast the statement

“The change in thermal energy stored in a body is balanced by the heat flow through its surface”

into a mathematical formula and show that it holds for the linear heat equation with pure Neumann spatial boundary conditions.

(Q9.2.1.14.B) In a model for linear transient heat conduction in a homogeneous body let $u = u(x, t) : \Omega \times [0, T] \rightarrow \mathbb{R}$ designated the temperature distribution. What is the physical meaning of the following **spatial boundary conditions**, here written in non-dimensional form:

1. $u(x, t) = g(x, t)$, $(x, y) \in \partial\Omega \times [0, T]$ (Dirichlet b.c.),
2. $\operatorname{grad} u(x, t) \cdot n(x) = 0$, $(x, y) \in \partial\Omega \times [0, T]$, (homogeneous Neumann b.c.),
3. $\operatorname{grad} u(x, t) \cdot n(x) = u(x, t)$, $(x, y) \in \partial\Omega \times [0, T]$, (impedance b.c.)?

△

9.2.2 Heat Equation: Spatial Variational Formulation



Video tutorial for Section 9.2.2: Heat Equation: Spatial Variational Formulation: (15 minutes)
[Download link](#), [tablet notes](#)

§9.2.2.1 (Model second-order linear parabolic evolution problem) Now we study the linear 2nd-order parabolic initial-boundary value problem with pure Dirichlet boundary conditions, introduced in the preceding section:

$$\frac{\partial}{\partial t}(\rho(x)u) - \operatorname{div}(\kappa(x) \operatorname{grad} u) = f \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \quad (9.2.1.6)$$

$$u(x, t) = g(x, t) \quad \text{for } (x, t) \in \partial\Omega \times]0, T[, \quad (9.2.1.8)$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \Omega . \quad (9.2.1.9)$$

Here ρ and κ are uniformly positive (\rightarrow Def. 1.2.2.9) and bounded integrable functions on Ω . The source function $f = f(\mathbf{x}, t)$ may depend on space and time and fulfills $f(\cdot, t) \in L^2(\Omega)$.

A special case is that of **homogeneous Dirichlet boundary conditions** $g \equiv 0$. The general case can be reduced to this by using the offset function trick, see Section 2.7.6, and solve the parabolic initial-boundary value problem for $w(\mathbf{x}, t) := u(\mathbf{x}, t) - \tilde{g}(\mathbf{x}, t)$, where $\tilde{g}(\cdot, t)$ is an extension of the Dirichlet data $g \in C^0([0, T] \times \partial\Omega)$ to $\tilde{\Omega}$. Then w will satisfy homogeneous Dirichlet boundary conditions and solve an evolution equation with a modified source function $\tilde{f}(\mathbf{x}, t)$. \square

§9.2.2.2 (Derivation of spatial variational formulation) Now we pursue the formal derivation of the **spatial** variational formulation of (9.2.1.6)–(9.2.1.8).

The steps completely mirror those discussed in Section 1.8, § 1.8.0.1. This paragraph should be reviewed again.

STEP 1: *test PDE with functions $v \in H_0^1(\Omega)$*

(Rule: do not test, where the solution is known, that is, on the boundary $\partial\Omega$)

Note: test function does *not depend on time*: $v = v(\mathbf{x})!$

STEP 2: *integrate over domain Ω*

►
$$\int_{\Omega} \left(\frac{d}{dt}(\rho u) - \operatorname{div}(\kappa(\mathbf{x}) \operatorname{grad} u) \right) v(\mathbf{x}) \, d\mathbf{x} = \int_{\Omega} f(\mathbf{x}) v(\mathbf{x}) \, d\mathbf{x} \quad \forall v : \Omega \rightarrow \mathbb{R} , \quad v|_{\partial\Omega} = 0 .$$

STEP 3: *perform integration by parts in space*

(by using Green's first formula, Thm. 1.5.2.7)

►
$$\int_{\Omega} \frac{d}{dt}(\rho u)(\mathbf{x}) v(\mathbf{x}) + \kappa(\mathbf{x}) \operatorname{grad} u(\mathbf{x}) \cdot \operatorname{grad} v(\mathbf{x}) \, d\mathbf{x} - \int_{\partial\Omega} \kappa(\mathbf{x}) \operatorname{grad} u(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) \underbrace{v(\mathbf{x})}_{=0} \, dS(\mathbf{x}) = \int_{\Omega} f(\mathbf{x}) v(\mathbf{x}) \, d\mathbf{x} \quad \forall v : \Omega \rightarrow \mathbb{R} , \quad v|_{\partial\Omega} = 0 .$$

Remark 9.2.2.3. For the concrete PDE (9.2.1.6) and boundary conditions (9.2.1.8) refer to Ex. 1.8.0.2 for a discussion of these steps in the stationary context. For more general boundary conditions study Ex. 1.8.0.6 to refresh yourself on how to obtain variational formulations. The derivation will include another STEP 4, which recasts boundary terms using the spatial boundary conditions. \square

The final step is the selection of an appropriate Sobolev space with respect to the dependence on the spatial variable. Following the guideline from Section 1.3.1 we pick the largest space, for which both

left and right hand side of the formal variational problem are still well defined for every time t . With the arguments from Section 1.3.4 we find the space $H_0^1(\Omega)$. \square

Since the coefficient ρ must not depend on time, we arrive at the following variational problem:

Spatial variational form of (9.2.1.6)–(9.2.1.8): seek $t \in]0, T[\mapsto u(t) \in H_0^1(\Omega)$

$$\int_{\Omega} \rho(x) \dot{u}(t) v \, dx + \int_{\Omega} \kappa(x) \mathbf{grad} u(t) \cdot \mathbf{grad} v \, dx = \int_{\Omega} f(x, t) v(x) \, dx \quad \forall v \in H_0^1(\Omega), \quad (9.2.2.4)$$

$$u(0) = u_0 \in H_0^1(\Omega). \quad (9.2.2.5)$$

Remark 9.2.2.6 (Function space valued functions) What does it mean, when we write $u(t)$? Be aware that $t \mapsto u(t)$ describes a **function space valued function** on $]0, T[$, here assigning to every instance of time a function in $H_0^1(\Omega)$:

$$u :]0, T[\rightarrow H_0^1(\Omega).$$

Also note that $\mathbf{grad} = \mathbf{grad}_x$ acts on the spatial independent variables that are suppressed in the notation $u(t)$. Hence $t \mapsto \mathbf{grad}_x u(t)$ is a function space valued function, too, with values in $(L^2(\Omega))^d$.

☞ Notation: $\dot{u}(t) = \frac{\partial u}{\partial t}(t) \hat{=} \text{(partial) derivative w.r.t. time: } \frac{\partial u}{\partial t}(t) \in H_0^1(\Omega)$ \square

§9.2.2.7 (Abstract linear parabolic evolution problems) We use the following shorthand abstract notation for (9.2.2.4) (with obvious correspondences):

$$t \in]0, T[\mapsto u(t) \in V_0 : \quad \begin{cases} m(\dot{u}(t), v) + a(u(t), v) = \ell(t)(v) & \forall v \in V_0, \\ u(0) = u_0 \in V_0. \end{cases} \quad (9.2.2.8)$$

Again, here $\ell(t) \hat{=} \text{linear form valued function on }]0, T[$.

Concretely for evolution problem (9.2.1.6), (9.2.1.8), (9.2.1.9):

$$\begin{aligned} m(\dot{u}(t), v) &:= \int_{\Omega} \rho(x) \dot{u}(t) v \, dx, \quad u, v \in H_0^1(\Omega), \\ a(u(t), v) &:= \int_{\Omega} \kappa(x) \mathbf{grad} u(t) \cdot \mathbf{grad} v \, dx, \quad u, v \in H_0^1(\Omega), \\ \ell(t)(v) &:= \int_{\Omega} f(x, t) v(x) \, dx, \quad v \in H_0^1(\Omega). \end{aligned}$$

Note that both m and a are **symmetric, positive definite** bilinear forms (\rightarrow Def. 1.2.3.27).

➤ Both m and a induce related energy norms $\|\cdot\|_a$ and $\|\cdot\|_m$ (\rightarrow Def. 1.2.3.35)

Since the bilinear form m does not depend on time, we conclude

$$m(\dot{u}, v) = \int_{\Omega} \rho(x) \dot{u}(t) v \, dx = \frac{d}{dt} \int_{\Omega} \rho(x) u(t) v \, dx = \frac{d}{dt} m(u, v),$$

and we can rewrite (9.2.2.8) equivalently as follows:

$$t \in]0, T[\mapsto u(t) \in V_0 : \quad \begin{cases} \frac{d}{dt} m(u(t), v) + a(u(t), v) = \ell(t)(v) & \forall v \in V_0, \\ u(0) = u_0 \in V_0. \end{cases} \quad (9.2.2.9)$$

This is a **linear** evolution problem in the sense that the mapping that associates the solution $u = u(x, t)$ to the **data** (ℓ, u_0) is linear. \square

Review question(s) 9.2.2.10 (Heat equation: spatial variational formulations)

(Q9.2.2.10.A) Give the spatial variational formulation of the following evolution problem with Neumann boundary conditions:

$$\begin{aligned} \frac{\partial}{\partial t}(\rho(x)u) - \Delta u &= 0 \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \\ \mathbf{grad} u(x, t) \cdot \mathbf{n}(x) &= h(x, t) \quad \text{for } (x, t) \in \partial\Omega \times]0, T[, \\ u(x, 0) &= 1 \quad \text{for all } x \in \Omega, \end{aligned}$$

where the Neumann data $h : \partial\Omega \times [0, t] \rightarrow \mathbb{R}$ and the (uniformly positive) coefficient function $\rho : \Omega \rightarrow \mathbb{R}$ are given. Does the solution linearly depend on h ?

At one step use Green's first formula:

Theorem 1.5.2.7. Green's first formula

For all vector fields $\mathbf{j} \in (C_{pw}^1(\bar{\Omega}))^d$ and functions $v \in C_{pw}^1(\bar{\Omega})$ holds

$$\int_{\Omega} \mathbf{j} \cdot \mathbf{grad} v \, dx = - \int_{\Omega} \operatorname{div} \mathbf{j} v \, dx + \int_{\partial\Omega} \mathbf{j} \cdot \mathbf{n} v \, ds. \quad (1.5.2.8)$$

(Q9.2.2.10.B) Let $\Omega \subset \mathbb{R}^2$ be a bounded domain and $\rho, \kappa : \Omega \rightarrow \mathbb{R}$ uniformly positive coefficient functions. Derive the spatial variational formulation for the parabolic initial value problem

$$\begin{aligned} \frac{\partial}{\partial t}(\rho(x)u) - \operatorname{div}(\kappa(x) \mathbf{grad} u) &= 0 \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \\ \mathbf{grad} u(x, t) \cdot \mathbf{n}(x) + \frac{\partial u}{\partial t}(x, t) &= 0 \quad \text{for } (x, t) \in \partial\Omega \times]0, T[, \\ u(x, 0) &= u_0(x) \quad \text{for all } x \in \Omega, \end{aligned}$$

for given initial data $u \in H^1(\Omega)$.

(Q9.2.2.10.C) Let $\Omega \subset \mathbb{R}^2$ be a bounded domain and $\rho, \kappa : \Omega \times]0, T[\rightarrow \mathbb{R}$ be uniformly positive **time-dependent** coefficient functions. Derive the correct spatial variational formulation for the parabolic initial value problem

$$\begin{aligned} \frac{\partial}{\partial t}(\rho(x, t)u) - \operatorname{div}(\kappa(x, t) \mathbf{grad} u) &= 0 \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \\ u(x, t) &= 0 \quad \text{for } (x, t) \in \partial\Omega \times]0, T[, \\ u(x, 0) &= u_0(x) \quad \text{for all } x \in \Omega, \end{aligned}$$

for given initial data $u \in H^1(\Omega)$.

\triangle

9.2.3 Stability of Parabolic Evolution Problems



Video tutorial for Section 9.2.3: Stability of Parabolic Evolution Problems: (20 minutes)
[Download link](#), [tablet notes](#)

Now we are concerned with the **stability** of linear parabolic evolution problems, the question to what extent the size of their solutions, measured in “relevant norms” can be bounded by suitable norms of the **data** u_0 , f (and g for Dirichlet boundary conditions (9.2.1.8)). Similar considerations for (stationary) abstract variational problems can be found in Section 1.4.3.

We investigate only whether $\|u(t)\|_{H^1(\Omega)}$ depends continuously on u_0 for all times t in the case $f \equiv 0$. This will also reveal fundamental **structural properties** of solutions of linear parabolic evolution problems.

For the sake of simplicity we restrict ourselves to constant coefficients $\rho \equiv 1$ and $\kappa \equiv 1$. (The general case of spatially varying coefficients is not more difficult, if both ρ and κ are bounded and uniformly positive, see (1.6.0.6).)

§9.2.3.1 (Solution of heat equation in one dimension) In 1D the homogeneous Dirichlet problem for the constant-coefficient heat equation on the spatial domain $\Omega =]0, 1[$,

$$\begin{aligned} \frac{\partial u}{\partial t}(x, t) - \frac{\partial^2 u}{\partial x^2} &= 0 \quad \text{in }]0, 1[\times [0, T] , \\ u(0, t) = u(1, t) &= 0 \quad \text{for all } 0 < t < T , \\ u(x, 0) &= u_0(x) \quad \text{for all } 0 < x < 1 . \end{aligned}$$

can be solved by **separation of variables** starting from the trial expression

$$u(x, t) = \varphi(x)\psi(t) , \quad 0 < x < 1 , \quad 0 < t < T . \quad (9.2.3.2)$$

Plugging this into the PDE $\frac{\partial u}{\partial t}(x, t) - \frac{\partial^2 u}{\partial x^2} = 0$ we end up with (' $\hat{}$ derivative w.r.t x , \cdot' $\hat{}$ derivative w.r.t. t)

$$\varphi(x)\dot{\psi}(t) - \varphi''(x)\psi(t) = 0 \Rightarrow \frac{\varphi''(x)}{\varphi(x)} = \frac{\dot{\psi}(t)}{\psi(t)} \equiv C \in \mathbb{R} ,$$

which gives us the ordinary differential equations

$$\varphi'' = C\varphi \quad \text{and} \quad \dot{\psi} = C\psi \quad (9.2.3.3)$$

The Dirichlet boundary conditions imply $\varphi(0) = \varphi(1) = 0$ and, together with (9.2.3.3), this fixes the constant C to a discrete set of values with associated solutions for φ :

$$\varphi_k(x) = \sin(\pi kx) , \quad C = -\pi^2 k^2 \Rightarrow \psi_k(t) = \alpha \exp(-\pi^2 k^2 t) , \quad k \in \mathbb{N} , \quad \alpha \in \mathbb{R} . \quad (9.2.3.4)$$

Hence, we obtain a general series formula for $u = u(x, t)$ as a superposition

$$u(x, t) = \sum_{k=1}^{\infty} \alpha_k \sin(\pi kx) \exp(-\pi^2 k^2 t) . \quad (9.2.3.5)$$

The coefficients $\alpha_k \in \mathbb{R}$ are given as **Fourier coefficients** of the initial data

$$u_0(x) = \sum_{k=1}^{\infty} \alpha_k \sin(\pi kx) \Rightarrow \alpha_k = 2 \int_0^1 u_0(x) \sin(\pi kx) dx .$$

By the orthogonality of the Fourier modes $x \mapsto \sin(k\pi x)$ in $L^2(]0, 1[)$ we have Bessel's formula

$$\left\| \left\{ x \mapsto \sum_{k=1}^{\infty} \gamma_k \sin(\pi kx) \right\} \right\|_{L^2(]0, 1[)}^2 = \frac{1}{2} \sum_{k=1}^{\infty} \gamma_k^2 .$$

As a consequence, from (9.2.3.5) we get

$$\|\{x \mapsto u(x, t)\}\|_{L^2([0, 1])}^2 = \sum_{k=1}^{\infty} \alpha_k \exp(-2\pi^2 k^2 t) .$$

This expresses an *exponential decay* of the $L^2([0, 1])$ -norm of the solution as a function of time t . \square

To tackle more general parabolic evolution problems we use that by the first Poincaré-Friedrichs inequality from

Theorem 1.3.4.17. Poincaré-Friedrichs inequality

If $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, is bounded, then

$$\|u\|_0 \leq \text{diam}(\Omega) \|\mathbf{grad} u\|_0 \quad \forall u \in H_0^1(\Omega) .$$

we can conclude that

$$\exists \gamma > 0: |v|_{H^1(\Omega)}^2 \geq \gamma \|v\|_{L^2(\Omega)}^2 \quad \forall v \in H_0^1(\Omega) . \quad (9.2.3.6)$$

In fact, Thm. 1.3.4.17 reveals $\gamma = \text{diam}(\Omega)^{-2}$, but the numerical value of γ is not important for our considerations.

Remark 9.2.3.7 (Differentiating bilinear forms with time-dependent arguments) Consider (temporally) smooth $u : [0, T] \mapsto V_0$, $v : [0, T] \mapsto V_0$ and a *symmetric* bilinear form $b : V_0 \times V_0 \mapsto \mathbb{R}$. We are concerned with computing the temporal derivative $\frac{d}{dt} b(u(t), v(t))$, because this will be a key step in the proof of stability estimates.

Perform formal Taylor expansion:

$$\begin{aligned} b(u(t + \tau), v(t + \tau)) &= b(u(t) + \dot{u}(t)\tau + O(\tau^2), v(t) + \dot{v}(t)\tau + O(\tau^2)) \\ &= b(u(t), v(t)) + \tau(b(\dot{u}(t), v(t)) + b(u(t), \dot{v}(t))) + O(\tau^2) . \\ \Rightarrow \frac{d}{dt} b(u(t), v(t)) &= \lim_{\tau \rightarrow 0} \frac{b(u(t + \tau), v(t + \tau)) - b(u(t), v(t))}{\tau} \\ &= b(\dot{u}(t), v(t)) + b(u(t), \dot{v}(t)) . \end{aligned}$$

This is a general *product rule*, see [Hip19, ??]. \square

The next result connects with the observation made in § 9.2.3.1 in the 1D case.

Lemma 9.2.3.8. Decay of solutions of parabolic evolutions

For $f \equiv 0$ the solution $u(t)$ of (9.2.2.4) satisfies

$$\|u(t)\|_m \leq e^{-\gamma t} \|u_0\|_m , \quad \|u(t)\|_a \leq e^{-\gamma t} \|u_0\|_a \quad \forall t \in]0, T[,$$

where $\gamma > 0$ is the constant from (9.2.3.6), and $\|\cdot\|_a$, $\|\cdot\|_m$ stand for the energy norms induced by $a(\cdot, \cdot)$ and $m(\cdot, \cdot)$, respectively.

Proof. Multiply the solution of the parabolic IBVP with an exponential weight function:

$$w(t) := \exp(\gamma t)u(t) \in H_0^1(\Omega) \Rightarrow \dot{w} := \frac{dw}{dt}(t) = \gamma w(t) + \exp(\gamma t) \frac{du}{dt}(t) , \quad (9.2.3.9)$$

solves the parabolic IBVP

$$\begin{aligned} \mathbf{m}(\dot{w}, v) + \tilde{\mathbf{a}}(w, v) &= 0 \quad \forall v \in V, \\ w(0) &= u_0, \end{aligned} \tag{9.2.3.10}$$

with $\tilde{\mathbf{a}}(w, v) = \mathbf{a}(w, v) - \gamma \mathbf{m}(w, v)$, γ from (9.2.3.6). To see this, use that $u(t)$ solves (9.2.2.9) with $f \equiv 0$ (elementary calculation).

Note:

$$(9.2.3.6) \Rightarrow \tilde{\mathbf{a}}(v, v) \geq 0 \quad \forall v \in V$$

We show the exponential decay of $\|\cdot\|_m$ -norm of solution:

$$\frac{d}{dt} \frac{1}{2} \|w\|_m^2 = \frac{d}{dt} \frac{1}{2} \mathbf{m}(w, w) \stackrel{\text{Rem. 9.2.3.7}}{=} \mathbf{m}(\dot{w}, w) = -\tilde{\mathbf{a}}(w, w) \leq 0 \tag{9.2.3.11}$$

This confirms that $t \mapsto \|w(t)\|_m$ is a decreasing function, which involves

$$(9.2.3.11) \Rightarrow \|w(t)\|_m \leq \|w(0)\|_m,$$

and the first assertion of the Lemma is evident. Next, we verify the exponential decay of $\|\cdot\|_{H^1(\Omega)}$ -norm of solution by a similar trick:

$$\begin{aligned} \frac{1}{2} \frac{d}{dt} \|w\|_{\tilde{\mathbf{a}}}^2 &\stackrel{\text{Rem. 9.2.3.7}}{=} \tilde{\mathbf{a}}\left(\frac{d}{dt} w, w\right) = -\mathbf{m}\left(\frac{d}{dt} w, \frac{d}{dt} w\right) \leq 0 \Rightarrow \|w(t)\|_{\tilde{\mathbf{a}}} \leq \|w(0)\|_{\tilde{\mathbf{a}}}, \\ \blacktriangleright \quad \|w(t)\|_a^2 &\leq \|w(0)\|_a^2 - \underbrace{\gamma (\|w(0)\|_m^2 - \|w(t)\|_m^2)}_{\geq 0 \text{ by (9.2.3.11)}}. \end{aligned}$$

Reverting the transformation (9.2.3.9) gives the estimates for $|u|_{H^1(\Omega)}$. □

Dissipation of energy in parabolic evolutions

 Exponential decay of energy during parabolic evolution without excitation
("Parabolic evolutions dissipate energy")

Note that if the source term f does not depend on time, then the lemma asserts exponential convergence (in time) of $u = u(t)$ solution of (9.2.2.4) to the solution $u^* = u^*(x) \in$ of the stationary boundary value problem

$$\int_{\Omega} \kappa(x) \mathbf{grad} u^*(x) \cdot \mathbf{grad} v(x) dx = \int_{\Omega} f(x) v(x) dx \quad \forall v \in H_0^1(\Omega).$$

 Exponential convergence (in time) to "equilibrium solution" in the case of time-independent excitation

Review question(s) 9.2.3.13 (Stability of parabolic evolution problems)

(Q9.2.3.13.A) Compute the solution of the 1D parabolic evolution problem

$$\begin{aligned} \frac{\partial u}{\partial t}(x, t) - \frac{\partial^2 u}{\partial x^2} &= 0 \quad \text{in }]0, 1[\times [0, T], \\ u(0, t) = u(1, t) &= 0 \quad \text{for all } 0 < t < T, \\ u(x, 0) &= \{x \mapsto \sin(\pi x)\} \quad \text{for all } 0 < x < 1. \end{aligned}$$

Hint. Use the separation of variables approach with $u(x, t) = \sin(\pi x)\psi(t)$ and determine the function ψ .

(Q9.2.3.13.B) For a sufficiently smooth function $u = u(\mathbf{x}, t)$ compute the partial derivative $\frac{\partial w}{\partial t}(\mathbf{x}, t)$ for $w(\mathbf{x}, t) := \exp(\gamma t)u(\mathbf{x}, t)$ in terms of partial derivatives of u .

(Q9.2.3.13.C) For a domain $\Omega \subset \mathbb{R}^2$ and a vector $\mathbf{c} \in \mathbb{R}^2$ we consider the bilinear form

$$b(u, v) := \int_{\Omega} \mathbf{grad} u(\mathbf{x}) \cdot \mathbf{c} v(\mathbf{x}) \, d\mathbf{x}, \quad u, v \in H^1(\Omega).$$

Express $\frac{d}{dt}b(u, u)$ for a sufficiently smooth function $u : \Omega \times [0, T] \rightarrow \mathbb{R}$ using partial derivatives of u .

(Q9.2.3.13.D) For a bounded domain $\Omega \subset \mathbb{R}^2$ consider the parabolic evolution problem

$$\begin{aligned} \frac{\partial}{\partial t}(\rho(\mathbf{x})u) - \Delta u &= 0 \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \\ \mathbf{grad} u(\mathbf{x}, t) \cdot \mathbf{n}(\mathbf{x}) &= 0 \quad \text{for } (\mathbf{x}, t) \in \partial\Omega \times]0, T[, \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \Omega, \end{aligned}$$

with (uniformly positive) coefficient function $\rho : \Omega \rightarrow \mathbb{R}$ and given initial data $u_0 \in H^1(\Omega)$.

Can we expect $u(\mathbf{x}, T) \rightarrow 0$ for $T \rightarrow \infty$? What kind of exponential decay can we observe in this case?

△

9.2.4 Spatial Semi-Discretization: Method of Lines



Video tutorial for Section 9.2.4: Spatial Semi-Discretization: Method of Lines: (13 minutes)
[Download link](#), [tablet notes](#)

We embark on a half-way discretization of a parabolic evolution problem in variational form (9.2.2.9). It will result in an initial value problem for an **ordinary differential equation** (→ § 6.1.3.1) on a finite dimensional state space.



Idea: Apply **Galerkin discretization** (→ Section 2.2) to abstract linear parabolic variational problem (9.2.2.9).

Recall from Section 2.2 that the fundamental ideas behind Galerkin discretization are

- (I) the use of finite dimensional subspaces of the function spaces as trial and test spaces
 ➤ discrete variational problem,
- (II) the choice of ordered bases in order to convert the discrete variational problem into a system of equations for unknown expansion coefficients.

We pursue this steps for the following abstract linear parabolic evolution problem posed over a vector space V_0 :

$$t \in]0, T[\mapsto u(t) \in V_0 : \quad \begin{cases} m(\dot{u}(t), v) + a(u(t), v) = \ell(t)(v) & \forall v \in V_0, \\ u(0) = u_0 \in V_0. \end{cases} \quad (9.2.2.9)$$

1st step: replace V_0 with a finite dimensional subspace $V_{0,h}$, $N := \dim V_{0,h} < \infty$

► (Spatially) discrete parabolic evolution problem

$$t \in]0, T[\mapsto u_h(t) \in V_{0,h} : \quad \begin{cases} m(\dot{u}_h(t), v_h) + a(u_h(t), v_h) = \ell(t)(v_h) & \forall v_h \in V_{0,h}, \\ u_h(0) = \text{projection/interpolant of } u_0 \text{ in } V_{0,h}. \end{cases} \quad (9.2.4.1)$$

2nd step: introduce (ordered) basis $\mathfrak{B}_h := \{b_h^1, \dots, b_h^N\}$ of $V_{0,h}$

Next plug in basis expansion of $u_h(t)$ with *time-dependent* coefficients μ_i :

$$u_h(t) = \sum_{i=1}^N \mu_i(t) b_h^i. \quad (9.2.4.2)$$

Note that the basis functions themselves do not depend on time, of course.

Method-of-lines ordinary differential equation

Combining (9.2.4.1) and (9.2.4.2) we obtain

$$(9.2.4.1) \Rightarrow \begin{cases} \mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = \vec{\phi}(t) & \text{for } 0 < t < T, \\ \vec{\mu}(0) = \vec{\mu}_0. \end{cases} \quad (9.2.4.4)$$

with

- ▷ s.p.d. stiffness matrix $\mathbf{A} \in \mathbb{R}^{N,N}$, $(\mathbf{A})_{ij} := a(b_h^j, b_h^i)$ (independent of time),
- ▷ s.p.d. mass matrix $\mathbf{M} \in \mathbb{R}^{N,N}$, $(\mathbf{M})_{ij} := m(b_h^j, b_h^i)$ (independent of time),
- ▷ source (load) vector $\vec{\phi}(t) \in \mathbb{R}^N$, $(\vec{\phi}(t))_i := \ell(t)(b_h^i)$ (time-dependent),
- ▷ $\vec{\mu}_0 \hat{=} \text{coefficient vector of a projection of } u_0 \text{ onto } V_{0,h}$.

Note:

(9.2.4.4) is an ordinary differential equation (ODE) for $t \mapsto \vec{\mu}(t) \in \mathbb{R}^N$

Conversion (9.2.2.9) \rightarrow (9.2.4.4) through Galerkin discretization *in space only* is known as **method of lines**.

(9.2.4.4) $\hat{=}$ a **semi-discrete** evolution problem

Discretized in space \longleftrightarrow but still continuous in time

§9.2.4.5 (Galerkin matrices in the method of lines ODE) For the concrete linear parabolic evolution problem (9.2.2.4)–(9.2.2.5) and spatial finite element discretization based on a finite element trial/test space $V_{0,h} \subset H^1(\Omega)$ we can compute

- the mass matrix \mathbf{M} as the Galerkin matrix for the bilinear form $(u, v) \mapsto \int_{\Omega} \rho(x) u(x) v(x) dx$, $u, v \in L^2(\Omega)$,
- the stiffness matrix \mathbf{A} as Galerkin matrix arising from the bilinear form $(u, v) \mapsto \int_{\Omega} \kappa(x) \mathbf{grad} u(x) \cdot \mathbf{grad} v(x) dx$, $u, v \in H^1(\Omega)$.

The calculations are explained in Section 2.7.4 and Section 2.7.5 and may involve numerical quadrature.

Remark 9.2.4.6 (Spatial discretization options) Beside the Galerkin approach any other method for spatial discretization of 2nd-order elliptic BVPs can be used in the context of the method of lines: the matrices \mathbf{A}, \mathbf{M} may also be generated by finite differences (\rightarrow Section 4.1), finite volume methods (\rightarrow Section 4.2), or collocation methods (\rightarrow Section 4.4).

Review question(s) 9.2.4.7 (Method of lines)

(Q9.2.4.7.A) The method-of-lines spatial Galerkin discretization of the linear evolution problem in variational form

$$t \in]0, T[\mapsto u(t) \in V_0 : \quad m(\dot{u}(t), v) + a(u(t), v) = \ell(t)(v) \quad \forall v \in V_0 ,$$

leads to the ordinary differential equation

$$\mathbf{M} \frac{d\vec{\mu}}{dt}(t) + \mathbf{A}\vec{\mu}(t) = \vec{\varphi}(t)$$

for the basis expansion coefficient vector $\vec{\mu} \in \mathbb{R}^N$ of the time-dependent semi-discrete Galerkin solution $u_h \in V_{0,h}$.

Give formulas for the entries of the matrices \mathbf{M} , \mathbf{A} and of the vector $\vec{\varphi}$.

(Q9.2.4.7.B) Verify the following result by direct computation.

Theorem 9.2.4.8. Variation-of-constants formula

For a fixed matrix $\mathbf{A} \in \mathbb{R}^{N,N}$, $N \in \mathbb{N}$, and a Lipschitz continuous function $\mathbf{g} : [0, \infty[\rightarrow \mathbb{R}^N$, the initial-value problem

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} + \mathbf{g}(t) , \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{R}^N ,$$

has the unique solution

$$\mathbf{y}(t) = \exp(\mathbf{A}t)(\mathbf{y}_0 + \int_0^t \exp(-\mathbf{A}\tau) \mathbf{g}(\tau) d\tau) \quad t \geq 0 .$$

In this theorem $\exp(\cdot)$ denotes the matrix exponential defined by the exponential series in the very same way as e^z for any $z \in \mathbb{C}$.

(Q9.2.4.7.C) Use Thm. 9.2.4.8 to write down the solution of the method-of-lines initial-value problem

$$\begin{cases} \mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A}\vec{\mu}(t) = \vec{\varphi}(t) & \text{for } 0 < t < T , \\ \vec{\mu}(0) = \vec{\mu}_0 . \end{cases}$$

(Q9.2.4.7.D) Consider

$$\begin{cases} \mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A}\vec{\mu}(t) = \vec{\varphi}(t) & \text{for } 0 < t < T , \\ \vec{\mu}(0) = \vec{\mu}_0 . \end{cases} \quad (9.2.4.4)$$

We replace the initial value $\vec{\mu}_0 \in \mathbb{R}^N$ with $\tilde{\vec{\mu}}_0 \in \mathbb{R}^N$ and, thus, obtain the perturbed solution $t \mapsto \tilde{\vec{\mu}}(t)$. Estimate $(\vec{\mu}(t) - \tilde{\vec{\mu}}(t))^T \mathbf{A} (\vec{\mu}(t) - \tilde{\vec{\mu}}(t))$.

(Q9.2.4.7.E) The spatial Galerkin semi-discretization of the evolution problem

$$t \in]0, T[\mapsto u(t) \in V_0 : \quad \begin{cases} \frac{d}{dt} m(u(t), v) + a(u(t), v) = \ell(t)(v) & \forall v \in V_0 , \\ u(0) = u_0 \in V_0 . \end{cases}$$

leads to an ordinary differential equation, which can be written in the form $\frac{d}{dt} \vec{\mu} = F(\vec{\mu})$. Give an expression for F with detailed formulas for all terms.

△

9.2.5 Recalled from Section 6.1: Ordinary Differential Equations

[Supplementary material. Skip, if § 10.2.2.7 has been treated in the course or if you were given an introduction to numerical methods for ordinary differential equations earlier. More details in Section 6.1.]



Video tutorial for Section 9.2.5: Ordinary Differential Equations: (20 minutes) [Download link](#), [tablet notes](#)

The method of lines from Section 9.2.4 yields an initial value problem for an ordinary differential equation (ODE). These should have been covered in any reasonable analysis course. For the sake of completeness some central concepts and results are refreshed in this section.

§9.2.5.1 (Notion of ordinary differential equation (ODE)) To define an ordinary differential equation we need the following ingredients:

- ◆ A **state space** $D \subset V_0$, a subset of the finite-dimensional “coordinate/coefficientspace” \mathbb{R}^N , $N \in \mathbb{N}$,
- ◆ a finite time interval $I \subset \mathbb{R}$,
- ◆ and a **continuous** right-hand side **vectorfield**, $\mathbf{f} : I \times D \rightarrow \mathbb{R}^N$, a function of time t and state \mathbf{u} : $\mathbf{f} = \mathbf{f}(t, \mathbf{u})$.



ordinary differential equations (ODE):

$$\dot{\mathbf{u}}(t) = \mathbf{f}(t, \mathbf{u}(t))$$

☞ Notation: $\cdot \hat{=}$ differentiation with respect to time t

In terms of vector components of $\mathbf{u} = [u_1, \dots, u_N]^\top$ and $\mathbf{f} = [f_1, \dots, f_N]^\top$ the ODE reads

$$\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u}) \Leftrightarrow \begin{bmatrix} \dot{u}_1(t) \\ \vdots \\ \dot{u}_N(t) \end{bmatrix} = \begin{bmatrix} f_1(t, u_1(t), \dots, u_N(t)) \\ \vdots \\ f_N(t, u_1(t), \dots, u_N(t)) \end{bmatrix}. \quad (9.2.5.2)$$

For the ODE (9.2.4.4) arising from the method of lines applied to (9.2.2.9) we have

- $D = \mathbb{R}^N$, $\mathbf{u}(t) \leftrightarrow \vec{\mu}(t)$, the time-dependent vector of basis expansion coefficients,
- right-hand side vectorfield $(t, \mathbf{u}) \mapsto \mathbf{f}(t, \mathbf{u})$ given by $(t, \vec{\mu}) \mapsto \vec{\varphi}(t) - \mathbf{A}\vec{\mu}(t)$.
(This right-hand side vectorfield is smooth in the state argument.)

For other examples of ODEs refer to Section 6.1.2. ↓

§9.2.5.3 (Solutions of ODEs → Def. 6.1.1.2)

Given an ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$ as introduced in § 9.2.5.1, we call $\mathbf{u} : J \subset I \rightarrow D$ a (local) **solution** on the (open) interval J , if

- \mathbf{u} is differentiable in every $t \in J$, and
- if $\dot{\mathbf{u}}(t) = \mathbf{f}(t, \mathbf{u}(t))$ holds in every point $t \in J$.

Obviously, smoothness of \mathbf{f} will imply smoothness of any solution $t \mapsto \mathbf{u}$ of the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$: If $\mathbf{f} \in C^q$, then $\mathbf{u} \in C^{q+1}$, $q \in \mathbb{N}_0$, see Lemma 6.1.1.3. ↓

Most ODEs have many solutions, even if we consider these on the maximal intervals on which they are defined. In order to single out a unique one, we have to impose initial conditions, and, thus, we arrive at initial-value problems for an ODE, cf. Eq. (6.1.3.2).

Initial-value problem (IVP)

Given an ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$ as introduced in § 9.2.5.1 and $(t_0, \mathbf{y}_0) \in \tilde{\Omega} := I \times D$ find an open interval $J \subset I$ and a solution $\mathbf{u} : J \subset I \rightarrow D$ of the ODE such that $t_0 \in J$ and $\mathbf{u}(t_0) = \mathbf{y}_0$.

In order to skirt technical difficulties we make the following assumption. Less restrictive is the setting of Section 6.1.3.

Assumption 9.2.5.4. Global Lipschitz continuity of vectorfield

Using the notations of § 9.2.5.1, in the sequel we take for granted that $D = \mathbb{R}^N$, $N \in \mathbb{N}$, and that \mathbf{f} is Lipschitz continuous (\rightarrow Def. 6.1.3.10)

$$\exists L > 0: \|\mathbf{f}(t, \mathbf{u}) - \mathbf{f}(s, \mathbf{v})\| \leq L\{|t-s| + \|\mathbf{u} - \mathbf{v}\|\} \quad \forall s, t \in I, \mathbf{u}, \mathbf{v} \in V_0. \quad (9.2.5.5)$$

Then all initial value problems have unique solutions defined on the whole time interval, cf. Thm. 6.1.3.16.

Theorem 9.2.5.6. Existence and uniqueness of solutions of IVPs \rightarrow Thm. 6.1.3.16

Under Ass. 9.2.5.4 the initial-value problem

$$\text{seek } \mathbf{u} : I \rightarrow V_0: \dot{\mathbf{u}}(t) = \mathbf{f}(t, \mathbf{u}(t)) \quad \forall t \in I, \quad \mathbf{u}(t_0) = \mathbf{u}_0, \quad (9.2.5.7)$$

has a solution $\mathbf{u} : I \rightarrow \mathbb{R}^N$ for every $t_0 \in I$ and $\mathbf{u}_0 \in \mathbb{R}^N$.

§9.2.5.8 (Evolution operator Section 6.1.4) Ass. 9.2.5.4 for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$ renders the associated evolution operator well-defined. It is the mapping

$$\Phi : \begin{cases} I \times I \times \mathbb{R}^N & \rightarrow \mathbb{R}^N \\ (t_0, t, \mathbf{u}_0) & \mapsto \Phi^{t_0, t} \mathbf{u}_0 := \Phi(t_0, t, \mathbf{u}_0) := \mathbf{u}(t), \end{cases} \quad (9.2.5.9)$$

where $t \mapsto \mathbf{u}(t)$ is the (unique) solution of the non-autonomous initial-value problem

$$\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u}), \quad \mathbf{u}(t_0) = \mathbf{u}_0.$$

Note that the underlying ODE can be recovered from the evolution operator:

$$\frac{\partial \Phi}{\partial t}(t_0, t, \mathbf{u}) = \mathbf{f}(t, \mathbf{u}) \quad \forall t_0, t \in I, \quad \mathbf{u} \in V_0. \quad (9.2.5.10)$$

The following facts are a consequence of Thm. 9.2.5.6:

1. If Φ is an evolution operator, for fixed $t_0 \in I$ and $\mathbf{u}_0 \in \mathbb{R}^N$ the so-called trajectory $t \mapsto \Phi$ supplies the solution for the initial value problem (9.2.5.7).
2. If Φ is an evolution operator, for fixed $s, t \in I$ the mapping $\mathbf{u} \mapsto \Phi^{s, t} \mathbf{u}$ is bijective and in terms of self-mappings of \mathbb{R}^N we have

$$\Phi^{t, t} = \text{Id}|_{\mathbb{R}^N} \quad \forall t \in I, \quad \Phi^{r, t} \circ \Phi^{s, r} = \Phi^{s, t} \quad \forall s, r, t \in I. \quad (9.2.5.11)$$

§9.2.5.12 (Linear ordinary differential equations \rightarrow § 6.1.1.8)

Definition 6.1.1.9. Linear first-order ODE

A first-order ordinary differential equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, as introduced in § 6.1.1.1 is **linear**, if

$$\mathbf{f}(t, \mathbf{y}) = \mathbf{A}(t)\mathbf{y} \quad , \quad \mathbf{A} : I \rightarrow \mathbb{R}^{N,N} \quad \text{a continuous function .} \quad (6.1.1.10)$$

If $t \mapsto \mathbf{u}(t)$ solves $\dot{\mathbf{u}} = \mathbf{A}(t)\mathbf{u}$, $\mathbf{u}(t_0) = \mathbf{u}_0$,
 $t \mapsto \mathbf{w}(t)$ solves $\dot{\mathbf{w}} = \mathbf{A}(t)\mathbf{w}$, $\mathbf{w}(t_0) = \mathbf{w}_0$,

then $t \mapsto \mathbf{v}(t) := \mathbf{u}(t) + \mathbf{w}(t)$ solves $\dot{\mathbf{v}} = \mathbf{A}(t)\mathbf{v}$, $\mathbf{v}(t_0) = \mathbf{u}_0 + \mathbf{w}_0$.

Hence, if Φ is the evolution operator belonging to a linear ODE, then

$$\mathbf{u} \mapsto \Phi^{t_0, t}\mathbf{u} \quad \text{is a linear bijective mapping } \mathbb{R}^N \rightarrow \mathbb{R}^N \quad \text{for all } t_0, t \in I .$$

A special case is the linear ODE $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u}$ for a given fixed matrix $\mathbf{A} \in \mathbb{R}^{N,N}$. This linear ODE with *constant coefficients* can be solved by a **diagonalization technique**: If

$$\mathbf{A} = \mathbf{S}\mathbf{D}\mathbf{S}^{-1}, \quad \mathbf{S} \in \mathbb{R}^{N,N} \text{ regular , } \quad \mathbf{D} = \text{diag}(\lambda_1, \dots, \lambda_N) \in \mathbb{R}^{N,N}, \quad (9.2.5.13)$$

then then solution of the IVP $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u}$, $\mathbf{u}(t_0) = \mathbf{u}_0 \in \mathbb{R}^N$, is given by

$$\mathbf{u}(t) = \mathbf{S} \begin{bmatrix} e^{\lambda_1(t-t_0)} & & \\ & \ddots & \\ & & e^{\lambda_N(t-t_0)} \end{bmatrix} \mathbf{S}^{-1} \mathbf{u}_0 . \quad (9.2.5.14)$$

]

Review question(s) 9.2.5.15 (Ordinary Differential Equations)

(Q9.2.5.15.A) [Fluid flow] Let $\Omega \subset \mathbb{R}^3$ be the volume of space occupied by a container filled with a fluid. Let $\mathbf{v} : \bar{\Omega} \rightarrow \mathbb{R}^3$ be a stationary **velocity field** describing the movement of the fluid.

- Which property of \mathbf{v} ensures that the fluid stays in the container?
- What is the physical interpretation of the solutions of the ODE $\dot{\mathbf{y}} = \mathbf{v}(\mathbf{y})$?

(Q9.2.5.15.B) [Evolution operator] Determine the **evolution operator** $\Phi : \mathbb{R} \times \mathbb{R} \times D$ for the scalar ODE $\dot{y} = \cos^2 y$ on the state space $D :=]-\pi/2, \pi/2[$.

Hint. $\tan' = \cos^{-2}$.

(Q9.2.5.15.C) [From evolution operator to ODE] The evolution of operator for an ODE on the state space \mathbb{R}^2 is given by

$$\Phi(s, t, \mathbf{y}) = \begin{bmatrix} \cos(t-s) & \sin(t-s) \\ -\sin(t-s) & \cos(t-s) \end{bmatrix} \mathbf{y} , \quad s, t \in \mathbb{R} , \quad \mathbf{y} \in \mathbb{R}^2 .$$

- Verify the so-called **flow properties** $\Phi(t, t, \mathbf{y}) = \mathbf{y}$ and $\Phi(s, t, \Phi(r, s, \mathbf{y})) = \Phi(r, t, \mathbf{y})$ for all $s, r, t \in \mathbb{R}$, $\mathbf{y} \in \mathbb{R}^2$.
- Find the associated ODE.

△

9.2.6 Recalled from § 10.2.2.7: Single-Step Methods for Numerical Integration

[This is supplementary material to be read only if § 10.2.2.7 and Chapter 7 were not covered in the course or you never had an introduction to numerical methods for ordinary differential equations.]



Video tutorial for Section 9.2.6: Single-Step Methods for Numerical Integration: (48 minutes)
[Download link](#), [tablet notes](#)

§9.2.6.1 (Numerical integration of ordinary differential equations: Key concepts) In this section we refresh central concepts from numerical integration of initial value problems for ODEs, see § 10.2.2.7, Chapter 7 for more details:

- single step methods of order p , see Def. 6.3.1.4 and Section 6.3.2, defined as recursions in state space based on **discrete evolution operators**.
- explicit and implicit Runge-Kutta single step methods, see Section 6.4, Section 7.3, encoded by Butcher schemes (6.4.0.11), (7.3.3.3).
- the notion of a **stiff** initial value problem (\rightarrow Notion 7.2.0.7),
- the definition of the **stability function** of a single step method, see Thm. 7.3.4.4,
- the concept of **L-stability** Def. 9.2.7.46 and how to verify it for Runge-Kutta methods.

]

9.2.6.1 Abstract Single-Step Methods for ODEs

Throughout this section we consider an ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$ on $I \times \mathbb{R}^N$ according to § 9.2.5.1. We will assume that $\mathbf{f} = \mathbf{f}(t, \mathbf{u})$ depends smoothly on both the time variable t and the state variable \mathbf{u} . Also Ass. 9.2.5.4 will be made throughout.

Recall from § 9.2.5.8 the concept of the **evolution operator** $\Phi : I \times I \times V_0 \rightarrow V_0$ associated with the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$:

$$\Phi = \Phi(s, t, \mathbf{u}) \Rightarrow \frac{\partial \Phi}{\partial t}(t_0, t, \mathbf{u}) = \mathbf{f}(t, \mathbf{u}) \quad \forall t_0, t \in I, \quad \mathbf{u} \in \mathbb{R}^N. \quad (9.2.5.10)$$

Basically, every single step method employs an approximation of the evolution operator in the form of a discrete evolution operator:

Definition 9.2.6.2. Discrete evolution operator \rightarrow [Section 6.3.1, § 6.3.1.1](#)

A **discrete evolution operator** belonging to an ODE on $I \times \mathbb{R}^N$ is a mapping $\Psi : I \times I \times \mathbb{R}^N \rightarrow \mathbb{R}^N$ such that for all $t \in I$ and $\mathbf{u} \in \mathbb{R}^N$

$$\exists q \in \mathbb{N}_0, \tau_0 > 0: \quad \|\Psi^{t,t+\tau} \mathbf{u} - \Phi^{t,t+\tau} \mathbf{u}\| \leq C(t, \mathbf{u}) \tau^{q+1} \quad \forall |\tau| < \tau_0, \quad (9.2.6.3)$$

with $C = C(t, \mathbf{u})$ depending on t, \mathbf{u} continuously, and Φ the evolution operator (\rightarrow § 9.2.5.8) associated with the ODE.

Terminology: The largest possible q in (9.2.6.3) is called the **order** of the discrete evolution.

The function $\tau \mapsto \Psi^{t,t+\tau} \mathbf{u} - \Phi^{t,t+\tau} \mathbf{u}$ is called the **consistency error** or **one-step error** of the discrete evolution.

9.2.6.2 Simple Single-Step Methods

EXAMPLE 9.2.6.4 (Finite difference single-step methods) → [Section 6.2](#)) Simple single-step methods arise from difference quotient approximation of the derivative in the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$ (posed on $I \times \mathbb{R}^N$).

- ① We may use a **forward difference quotient** with width $\tau > 0$,

$$\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u}) \quad \Rightarrow \quad \frac{\mathbf{u}(t + \tau) - \mathbf{u}(t)}{\tau} \approx \mathbf{f}(t, \mathbf{u}(t)) .$$

Reading \approx as $=$, this induces the discrete evolution operator of the

explicit (forward) Euler method: $\Psi^{t,t+\tau} \mathbf{u} := \mathbf{u} + \tau \mathbf{f}(t, \mathbf{u})$

(9.2.6.5)

- ② We can also rely on a **backward difference quotient** with width $\tau > 0$,

$$\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u}) \quad \Rightarrow \quad \frac{\mathbf{u}(t + \tau) - \mathbf{u}(t)}{\tau} \approx \mathbf{f}(t + \tau, \mathbf{u}(t + \tau)) .$$

This induces the discrete evolution operator of the

implicit (backward) Euler method: $\Psi^{t,t+\tau} \mathbf{u} := \mathbf{w}: \mathbf{w} = \mathbf{u} + \tau \mathbf{f}(t + \tau, \mathbf{w})$

(9.2.6.6)

This method is called **implicit**, because the action of the discrete evolution operator is defined implicitly as solution of an equation.

- ③ Finally, we can use a **symmetric difference quotient** of width $\tau > 0$ anchored at the midpoint between t and $t + \tau$,

$$\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u}) \quad \Rightarrow \quad \frac{\mathbf{u}(t + \tau) - \mathbf{u}(t)}{\tau} \approx \mathbf{f}(t + \frac{1}{2}\tau, \frac{1}{2}(\mathbf{u}(t) + \mathbf{u}(t + \tau))) .$$

This induces the discrete evolution operator of the

implicit midpoint method: $\Psi^{t,t+\tau} \mathbf{u} := \mathbf{w}: \mathbf{w} = \mathbf{u} + \tau \mathbf{f}(t + \frac{1}{2}\tau, \frac{1}{2}(\mathbf{u} + \mathbf{u}))$

(9.2.6.7)

↓

EXAMPLE 9.2.6.8 (Orders of finite-difference single-step methods) Def. 9.2.6.2 also introduced the notion of the **order** of a discrete evolution. Now we compute the order of the three finite-difference single-step methods derived in Ex. 9.2.6.4. We will demonstrate a fundamental technique for establishing (9.2.6.3) based on **Taylor expansion**. It hinges on smoothness of the vectorfield $\mathbf{f} = \mathbf{f}(t, \mathbf{u})$ in both arguments, which will ensure smoothness of solutions of the associated ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$.

Assumption 9.2.6.9. Smoothness of right-hand side vectorfield

The vectorfield $(t, \mathbf{u}) \mapsto \mathbf{f}(t, \mathbf{u})$ is C^∞ on $I \times \mathbb{R}^N$.

The Taylor expansion of $\tau \mapsto \Phi^{t,t+\tau} \mathbf{u}_0$, $t \in I$, $\mathbf{u}_0 \in V_0$, can be deduced from (9.2.5.10). Setting $\mathbf{v}(\tau) := \Phi^{t,t+\tau} \mathbf{u}_0$, which is a solution of the ODE, we find, using the one-dimensional chain rule,

$$\begin{aligned} \frac{d\mathbf{v}}{d\tau}(\tau) &= \mathbf{f}(t + \tau, \mathbf{v}(\tau)) , \\ \frac{d^2\mathbf{v}}{d\tau^2}(\tau) &= \frac{\partial \mathbf{f}}{\partial t}(t + \tau, \mathbf{v}(\tau)) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(t + \tau, \mathbf{v}(\tau)) \frac{d\mathbf{v}}{d\tau}(\tau) . \end{aligned}$$

This yields the following truncated Taylor expansion

$$\begin{aligned}\Phi^{t,t+\tau} \mathbf{u}_0 &= \mathbf{v}(\tau) = \mathbf{v}(0) + \tau \frac{d\mathbf{v}}{d\tau}(0) + \frac{1}{2} \tau^2 \frac{d^2\mathbf{v}}{d\tau^2}(0) + O(\tau^3) \\ &= \mathbf{u}_0 + \tau \mathbf{f}(t, \mathbf{u}_0) + \frac{1}{2} \tau^2 \left(\frac{\partial \mathbf{f}}{\partial t}(t, \mathbf{u}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(t, \mathbf{u}_0) \mathbf{f}(t, \mathbf{u}_0) \right) + O(\tau^3)\end{aligned}\quad (9.2.6.10)$$

for $\tau \rightarrow 0$. Note that the partial derivative $\frac{\partial \mathbf{f}}{\partial \mathbf{u}}(t, \mathbf{u}_0)$ is a Jacobi matrix. Explicit expressions for the remainder term involve second derivatives of \mathbf{f} .

- ① For the **explicit Euler method** (9.2.6.5) we immediately have from (9.2.6.10)

$$\Psi^{t,t+\tau} \mathbf{u}_0 - \Phi^{t,t+\tau} \mathbf{u}_0 = \mathbf{u}_0 + \tau \mathbf{f}(t, \mathbf{u}_0) - \mathbf{u}_0 - \tau \mathbf{f}(t, \mathbf{u}_0) + O(\tau^2) = O(\tau^2) \quad \text{for } \tau \rightarrow 0.$$

► The explicit Euler method is of **order 1**.

- ② It is not as straightforward for the **implicit Euler method** (9.2.6.6). First, we plug (9.2.6.6) into itself

$$\mathbf{w}(\tau) = \mathbf{u}_0 + \tau \mathbf{f}(t + \tau, \mathbf{w}(\tau)) = \mathbf{u}_0 + \tau \mathbf{f}(t + \tau, \mathbf{u}_0 + \tau \mathbf{f}(t + \tau, \mathbf{w}(\tau))),$$

and then use the multi-dimensional truncated Taylor expansion of \mathbf{f} around (t, \mathbf{u}_0)

$$\mathbf{f}(t + \delta, \mathbf{u}_0 + \mathbf{v}) = \mathbf{f}(t, \mathbf{u}_0) + \delta \frac{\partial \mathbf{f}}{\partial t}(t, \mathbf{u}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(t, \mathbf{u}_0) \mathbf{v} + O(\delta^2 + \|\mathbf{v}\|^2) \quad (9.2.6.11)$$

for $\delta \rightarrow 0, \mathbf{v} \rightarrow 0$. This gives

$$\mathbf{w}(\tau) = \mathbf{u}_0 + \tau \left(\mathbf{f}(t, \mathbf{u}_0) + \tau \frac{\partial \mathbf{f}}{\partial t}(t, \mathbf{u}_0) + \tau \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(t, \mathbf{u}_0) \mathbf{f}(t + \tau, \mathbf{w}(\tau)) \right) + O(\tau^3)$$

for $\tau \rightarrow 0$. Since $\Psi^{t,t+\tau} \mathbf{u}_0 = \mathbf{w}(\tau)$, matching terms with (9.2.6.10) we obtain

$$\Psi^{t,t+\tau} \mathbf{u}_0 - \Phi^{t,t+\tau} \mathbf{u}_0 = \tau^2 \frac{\partial \mathbf{f}}{\partial t}(t, \mathbf{u}_0) + \tau^2 \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(t, \mathbf{u}_0) \mathbf{f}(t + \tau, \mathbf{w}(\tau)) + O(\tau^3) = O(\tau^2)$$

for $\tau \rightarrow 0$. Thanks to the smoothness of \mathbf{f} the remainder terms will depend continuously on t and \mathbf{u}_0 .

► The implicit Euler method has **order 1**.

- ③ For the **implicit midpoint rule** (9.2.6.7) we follow the same idea and consider

$$\begin{aligned}\mathbf{w}(\tau) &= \mathbf{u}_0 + \tau \mathbf{f}(t + \frac{1}{2}\tau, \frac{1}{2}(\mathbf{u}_0 + \mathbf{w}(\tau))) \\ &= \mathbf{u}_0 + \tau \mathbf{f}(t + \frac{1}{2}\tau, \mathbf{u}_0 + \tau \mathbf{f}(t + \frac{1}{2}\tau, \frac{1}{2}(\mathbf{u}_0 + \mathbf{w}(\tau)))) \\ &= \mathbf{u}_0 + \tau \mathbf{f}(t + \frac{1}{2}\tau, \mathbf{u}_0 + \frac{1}{2}\tau \mathbf{f}(t + \frac{1}{2}\tau, \mathbf{u}_0 + O(\tau))) \quad \text{for } \tau \rightarrow 0.\end{aligned}$$

Then we resort to the truncated Taylor expansion (9.2.6.11) and get for $\tau \rightarrow 0$

$$\begin{aligned}\mathbf{w}(\tau) &= \mathbf{u}_0 + \tau \left(\mathbf{f}(t, \mathbf{u}_0) + \frac{1}{2} \tau \frac{\partial \mathbf{f}}{\partial t}(t, \mathbf{u}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(t, \mathbf{u}_0) \frac{1}{2} \tau \mathbf{f}(t + \frac{1}{2}\tau, \mathbf{u}_0 + O(\tau)) \right) + O(\tau^3) \\ &= \mathbf{u}_0 + \tau \left(\mathbf{f}(t, \mathbf{u}_0) + \frac{1}{2} \tau \frac{\partial \mathbf{f}}{\partial t}(t, \mathbf{u}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(t, \mathbf{u}_0) \frac{1}{2} \tau (\mathbf{f}(t, \mathbf{u}_0) + O(\tau)) \right) + O(\tau^3).\end{aligned}$$

Matching with (9.2.6.10) shows $\mathbf{w}(\tau) - \Phi^{t,t+\tau} \mathbf{u}_0 = O(\tau^3)$ where the “O” just comprises *continuous* higher order derivatives of \mathbf{f} .

► The implicit midpoint rules features **order 2**.

□

9.2.6.3 Single-Step Method for Initial-Value Problems

Now we want to use a single-step method based on the discrete evolution $\Psi = \Psi^{t_0, t}$, $s, t \in I$, $\mathbf{u} \in \mathbb{R}^N$ (\rightarrow Def. 9.2.6.2), to solve an initial-value problem

$$\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u}) \quad , \quad \mathbf{u}(t_0) = \mathbf{u}_0 \quad , \quad t_0 \in I \quad , \quad \mathbf{u}_0 \in \mathbb{R}^N .$$

on the interval $[t_0, T]$. We write $t \mapsto \mathbf{u}(t)$ for its unique solution.

As for 2-point boundary value problems treated in Section 2.3 we rely on a **(temporal) mesh** with nodes

$$t_0 < t_1 < \dots < t_{M-1} < t_M = T \quad , \quad M \in \mathbb{N} .$$

We call $\tau_j := t_j - t_{j-1}$ the local timestep size. An equidistant temporal mesh has a uniform timestep $\tau_j = \frac{T-t_0}{M}$.

Then we compute a sequence $(\mathbf{u}^{(j)})_{j=0}^M$ of approximations $\mathbf{u}^{(j)} \approx \mathbf{u}(t_j)$ to the solution of (9.2.4.4) at the nodes of the temporal mesh according to, cf. Def. 6.3.1.4,

$$\mathbf{u}^{(0)} := \mathbf{u}_0 \quad , \quad \mathbf{u}^{(j)} := \Psi^{t_{j-1}, t_j} \mathbf{u}^{(j-1)} \quad , \quad j = 1, \dots, M . \quad (9.2.6.12)$$

A key issue is the convergence of a single-step method on sequences of temporal meshes with $M \rightarrow \infty$ and $\max_j \tau_j \rightarrow 0$. To quantify it, one usually considers

- the error at final time $\epsilon_M := \|\mathbf{u}^{(M)} - \mathbf{u}(T)\|$, or
- the maximal error in the nodes of the temporal mesh

$$\epsilon_\infty := \max_{j=1, \dots, M} \|\mathbf{u}^j - \mathbf{u}(t_j)\| .$$

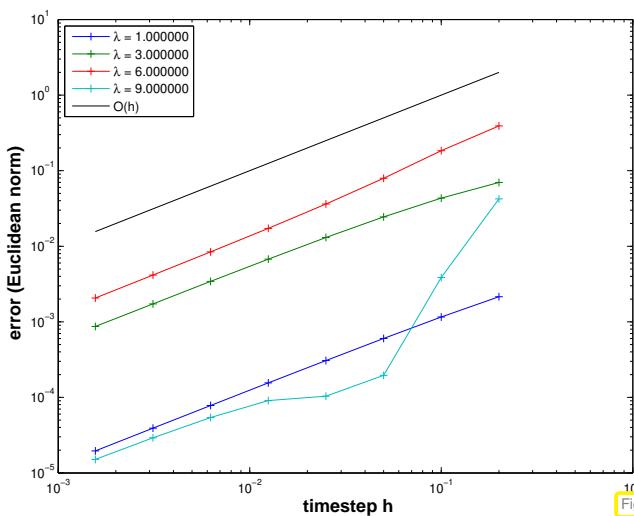
The next numerical experiments demonstrate the rather typical behavior of these errors for single-step methods applied to solve initial value problems with smooth solutions.

EXAMPLE 9.2.6.13 (Convergence of finite-difference single-step methods \rightarrow Exp. 6.3.2.5) We empirically investigate the h-convergence of our simple single-step methods, that is, we study the dependence of errors/error norms on the meshwidth of uniform/equidistant temporal meshes.

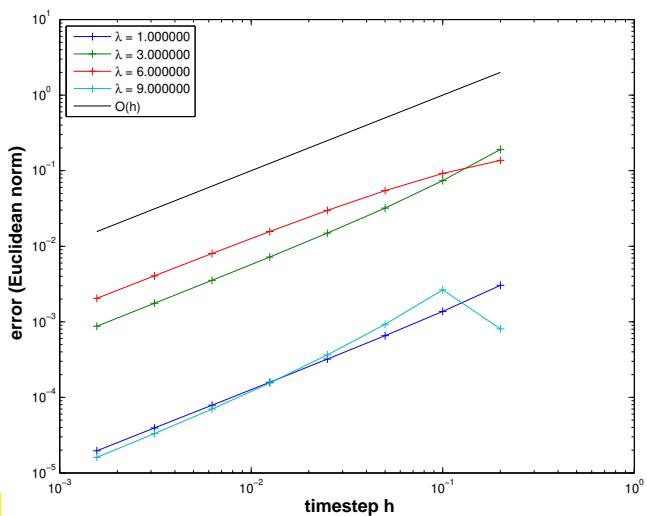
- ◆ We consider the following IVP for the so-called logistic ODE, a scalar ODE with analytically known smooth solution, see Ex. 6.1.2.1

$$\dot{u} = \lambda u(1-u) \quad , \quad u(0) = 0.01 .$$

- ◆ We apply explicit and implicit Euler methods (9.2.6.5)/(9.2.6.6), and the implicit midpoint rule (9.2.6.7) on $[0, 1]$ with uniform timestep $\tau = 1/M$, $M \in \{5, 10, 20, 40, 80, 160, 320, 640\}$.
- ◆ We monitor te error at final time $\epsilon_M(\tau) := |u(1) - u^{(M)}|$

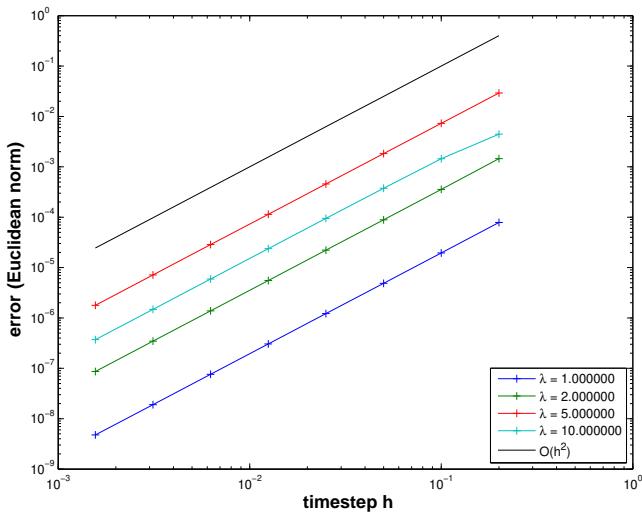


explicit Euler method



implicit Euler method

$O(M^{-1}) = O(\tau)$ algebraic convergence in both cases for $\tau \rightarrow 0$



Moreover, the observed rates of algebraic convergence match the orders of the single-step methods according to Def. 9.2.6.2.

However, finite-difference single-step methods can do better:

- ▷ Convergence of the implicit midpoint method in the above setting.
We observe algebraic convergence $O(\tau^2)$ for $\tau \rightarrow 0$.

What we have seen in Ex. 9.2.6.13 is the manifestation of a general truth, which is discussed in more detail in Section 6.3.2.

Theorem 9.2.6.14. Convergence of single-step methods

Let $\mathbf{u}^{(j)} \in \mathbb{R}^N$, $j = 1, \dots, M$, be the sequence of pointwise approximations of the solution of the initial value problem

$$\text{seek } \mathbf{u} : I \rightarrow V_0: \quad \dot{\mathbf{u}}(t) = \mathbf{f}(t, \mathbf{u}(t)) \quad \forall t \in I, \quad \mathbf{u}(t_0) = \mathbf{u}_0, \quad (9.2.5.7)$$

on a time interval $[t_0, T]$ with fixed final time $T > t_0$ generated by a single step method of order $q \in \mathbb{N}$ on a temporal mesh $t_0 < t_1 < t_2 < \dots < t_M = T$.

If $\mathbf{f} \in C^{q+1}(I \times \mathbb{R}^N)$, then

$$\max_{j=1, \dots, M} \left\| \mathbf{u}^{(j)} - \mathbf{u}(t_j) \right\| \leq C \tau^q \quad \text{for } \tau := \max_{j=1, \dots, M} |t_j - t_{j-1}|, \quad (9.2.6.15)$$

with $C > 0$ independent of the t_j .

Algebraic convergence of single-step methods

For initial value problems with smooth (in time) solutions, a single-step method converges algebraically in the meshwidth with a rate equal to its order.

9.2.6.4 Collocation Single-Step Methods

Repeating the developments of Section 7.3.2, this section elucidates a general approach for the construction of discrete evolution operators Ψ (\rightarrow Def. 9.2.6.2) for an ordinary differential equation $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$. It is based on interpolation onto spaces of polynomials and the resulting order in the sense of Def. 9.2.6.2 is easy to predict.

Notation: Following Def. 2.5.2.2 we write $\mathcal{P}_s(\mathbb{R})$, $s \in \mathbb{N}_0$, for the $N(s+1)$ -dimensional space of vector-valued uni-variate polynomials of degree $\leq s$ on \mathbb{R} with values in \mathbb{R}^N .

We fix $p \in \mathbb{N}$ and arbitrary $t \in I$, $\tau > 0$ such that $t + \tau \in I$ and aim to define the action of the discrete evolution operator $\Psi^{t,t+\tau} \mathbf{u}_0$ for any $\mathbf{u}_0 \in \mathbb{R}^N$.

§9.2.6.17 (Collocation for ODEs)



Idea: **Polynomial approximation** of the solution of $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$, $\mathbf{u}(t_0) = \mathbf{u}_0$, on $[t, t + \tau]$.
 \leftrightarrow piecewise polynomial approximation underlying the finite element method, cf. § 2.3.1.4.

Write $\xi \mapsto \mathbf{u}(\xi)$, $t \leq \xi \leq t + \tau$, for the unique solution of the initial-value problem $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$, $\mathbf{u}(t_0) = \mathbf{u}_0$. Thus we

$$\text{seek } \tilde{\mathbf{u}}_s \in \mathcal{P}_s(\mathbb{R}): \quad \tilde{\mathbf{u}}_s \approx \mathbf{u} \quad \text{on } [t, t + \tau].$$

It is natural to demand, $\tilde{\mathbf{u}}_s(t) = \mathbf{u}(t) = \mathbf{u}_0$. However, we cannot expect $\xi \mapsto \mathbf{f}(\xi, \tilde{\mathbf{u}}_s(\xi))$ to be polynomial. Consequently, in general $\tilde{\mathbf{u}}_s$ cannot be a solution of the ODE.



Idea: Demand that $\tilde{\mathbf{u}}_s$ solves ODE with modified right-hand side vectorfield $\mathbf{l}_s \circ \mathbf{f}$, where \mathbf{l}_{s-1} is a Lagrangian interpolation operator onto $\mathcal{P}_{s-1}(\mathbb{R})$ based on nodes $t + c_j \tau$, $c_j \in [0, 1]$, $j = 1, \dots, s$:

$$\tilde{\mathbf{u}}_s(\sigma) = \mathbf{l}_{s-1}(\{\xi \mapsto \mathbf{f}(\xi, \tilde{\mathbf{u}}_s(\xi))\})(\sigma) \quad \text{on } [t, t + \tau]. \quad (9.2.6.18)$$

Note that interpolation into the space of polynomials of degree $\leq s-1$ is required, because differentiation reduces the degree by one.

Equivalently, invoking the uniqueness of interpolating polynomials, we can (9.2.6.18) rewrite as

$$\tilde{\mathbf{u}}_s(t + c_j \tau) = \mathbf{f}(t + c_j \tau, \tilde{\mathbf{u}}_s(t + c_j \tau)), \quad j = 1, \dots, s. \quad (9.2.6.19)$$

Thus, the polynomial trial function $\tilde{\mathbf{u}}_s$ satisfies the ODE only at s points.

In general, the approach of seeking a solution in an s -dimensional trial space and then obtaining s equations for s basis expansion coefficients by imposing exact satisfaction of the equation in s isolated **collocation points**, is known as **collocation**. In (9.2.6.19) the collocation points are $t + c_j \tau$.

Discrete evolution operator by polynomial collocation

Given $t \in I$, $\tau \in \mathbb{R}$ with $t + \tau \in I$, $\mathbf{u}_0 \in \mathbb{R}^N$, $s \in \mathbb{N}$, and interpolation nodes $0 \leq c_1 < c_2 < \dots < c_s \leq 1$, we set

$$\Psi^{t,t+\tau} \mathbf{u}_0 = \tilde{\mathbf{u}}_s(t + \tau),$$

where $\tilde{\mathbf{u}}_s \in \mathcal{P}_s(\mathbb{R})$ satisfies

$$\tilde{\mathbf{u}}_s(t) = \mathbf{u}_0, \quad (9.2.6.21)$$

$$\tilde{\mathbf{u}}_s(t + c_j \tau) = \mathbf{f}(t + c_j \tau, \tilde{\mathbf{u}}_s(t + c_j \tau)), \quad j = 1, \dots, s. \quad (9.2.6.22)$$

The equations (9.2.6.22) are called **collocation conditions**.

The $N(s+1)$ conditions (9.2.6.21) and (9.2.6.22) match $\dim \mathcal{P}_s(\mathbb{R}) = N(s+1)$. However, note that this is a tentative definition, because there might not exist any polynomial satisfying the collocation conditions. Fortunately for small timestep this cannot happen.

Lemma 9.2.6.23. Existence of solutions of collocation equations [DB02, Sect. 6.3]

There is a threshold $\tau_0 > 0$ depending only on \mathbf{f} and the interpolation nodes $c_j \in [0, 1], j = 1, \dots, s$, such that for any $t \in I$, $\mathbf{u}_0 \in \mathbb{R}^N$, and $0 \leq \tau \leq \tau_0$ ($t + \tau \in I$) the conditions (9.2.6.21) and (9.2.6.22) determine a unique $\tilde{\mathbf{u}}_s \in \mathcal{P}(\mathbb{R})$.

§9.2.6.24 (Explicit collocation equations) As in Section 7.3.2 we derive a concrete representation for the polynomial $\tilde{\mathbf{u}}_s$. We draw on concepts related to Lagrange polynomial interpolation introduced in [Hip19, ??]. Recall that in (9.2.6.22) we had used the collocation points

$$\tau_j := t + c_j \tau, \quad j = 1, \dots, s, \quad \text{for } 0 \leq c_1 < c_2 < \dots < c_s \leq 1.$$

Let $\{L_j\}_{j=1}^s \subset \mathcal{P}_{s-1}$ denote the set of Lagrange polynomials of degree $s-1$ associated with the node set $\{c_j\}_{j=1}^s$, see [Hip19, ??]:

$$L_j \in \mathcal{P}_{s-1}(\mathbb{R}): \quad L_j(c_k) = \delta_{kj} = \begin{cases} 1 & \text{if } k = j, \\ 0 & \text{else,} \end{cases} \quad 1 \leq k, j \leq s.$$

In each of its N components, the derivative $\dot{\tilde{\mathbf{u}}}_s$ is a polynomial of degree $s-1$: $\dot{\tilde{\mathbf{u}}}_s \in \mathcal{P}_{s-1}$. Since the L_j , $j = 1, \dots, s$, form a basis of $\mathcal{P}_{s-1}(\mathbb{R})$, we have following representation:

$$\dot{\tilde{\mathbf{u}}}_s(t + \xi \tau) = \sum_{j=1}^s \dot{\tilde{\mathbf{u}}}_s(t + c_j \tau) L_j(\xi), \quad 0 \leq \xi \leq 1. \quad (9.2.6.25)$$

Thanks to the collocation conditions (9.2.6.22) we can replace $\tilde{\mathbf{u}}_s(t + c_j\tau)$ with an expression involving the right-hand side vectorfield \mathbf{f} :

$$(9.2.6.22) \quad \blacktriangleright \quad \tilde{\mathbf{u}}_s(t + \xi\tau) = \sum_{j=1}^s \mathbf{k}_j L_j(\xi) \quad \text{with increments} \quad \mathbf{k}_j := f(t + c_j\tau, \tilde{\mathbf{u}}_s(t + c_j\tau)) .$$

Next we integrate and use $\tilde{\mathbf{u}}_s(t) = \mathbf{u}_0$:

$$\blacktriangleright \quad \tilde{\mathbf{u}}_s(t + \xi\tau) = \mathbf{u}_0 + \tau \sum_{j=1}^s \mathbf{k}_j \int_0^\xi L_j(\zeta) d\zeta , \quad 0 \leq \xi \leq 1 .$$

This yields the following formulas for the computation of $\Psi^{t,t+\tau}\mathbf{u}_0$, which characterize the s -stage **collocation single step method** induced by the (normalized) interpolation nodes $c_j \in [0, 1]$, $j = 1, \dots, s$.

$$\begin{aligned} \mathbf{k}_i &= f(t + c_i\tau, \mathbf{u}_0 + \tau \sum_{j=1}^s a_{ij} \mathbf{k}_j) , & \text{where } a_{ij} &:= \int_0^{c_i} L_j(\xi) d\xi , \\ \Psi^{t,t+\tau}\mathbf{u}_0 &:= \tilde{\mathbf{u}}_s(t + \tau) = \mathbf{u}_0 + \tau \sum_{i=1}^s b_i \mathbf{k}_i . & b_i &:= \int_0^1 L_i(\xi) d\xi . \end{aligned} \quad (7.3.2.6)$$

Note that, generically, (7.3.2.6) represents a non-linear system of $s \cdot N$ equations for the $s \cdot N$ components of the vectors \mathbf{k}_i , $i = 1, \dots, s$. Usually, it will not be possible to obtain \mathbf{k}_i by a fixed number of evaluations of \mathbf{f} . For this reason the single step methods defined by (7.3.2.6) are called **implicit**.

By Lemma 9.2.6.23 a unique increments \mathbf{k}_i , $i = 1, \dots, s$, can be found provided that the timestep τ is sufficiently small. \square

Remark 9.2.6.26 (Simple collocation single-step methods) It turns out that we recover all finite-difference single-step methods found in Ex. 9.2.6.4 as particular collocation single step methods:

- ① For the **explicit Euler method** (9.2.6.5) we have

$$s = 1 , \quad c_1 = 0 .$$

- ② The **implicit Euler method** (9.2.6.6) is obtained for

$$s = 1 , \quad c_1 = 1 .$$

- ③ The **implicit midpoint rule** (9.2.6.7) arises from choosing

$$s = 1 , \quad c_1 = \frac{1}{2} .$$

EXPERIMENT 9.2.6.27 (Empiric Convergence of collocation single step methods Exp. 9.2.6.27)
We consider the scalar logistic ODE $\dot{u} = \lambda(1 - u)u$ with parameter $\lambda = 10$, initial state $y_0 = 0.01$, $T = 1$.

Numerical integration by timestepping with uniform timestep τ based on collocation single step method

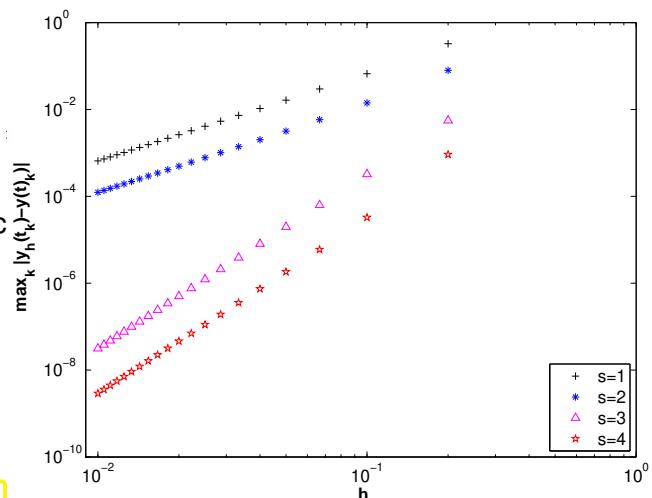
(7.3.2.6).

①

Equidistant collocation points, $c_j = \frac{j}{s+1}$, $j = 1, \dots, s$.

We observe **algebraic convergence** with the empirical rates

- $s = 1$: rate = 1.96
- $s = 2$: rate = 2.03
- $s = 3$: rate = 4.00
- $s = 4$: rate = 4.04



In this case, in light of Thm. 9.2.6.14, we conjecture the following order (\rightarrow Def. 9.2.6.2) of the collocation single step method:

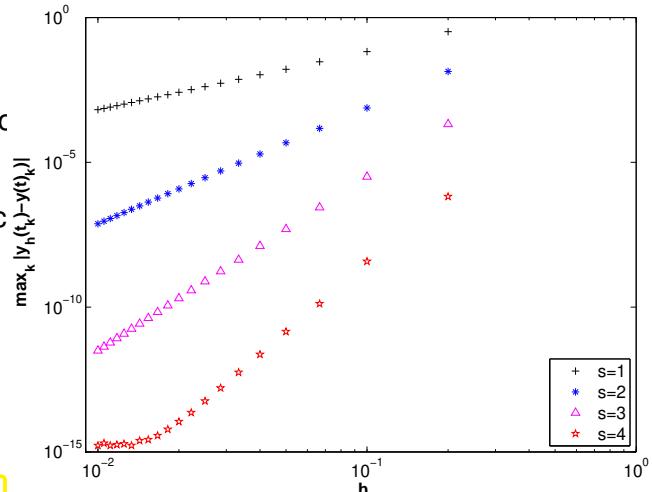
$$\text{(empiric) order} = \begin{cases} s & \text{for even } s, \\ s + 1 & \text{for odd } s. \end{cases}$$

①

Gauss points in $[0, 1]$ as normalized collocation points c_j , $j = 1, \dots, s$.

We observe **algebraic convergence** with the empirical rates

- $s = 1$: rate = 1.96
- $s = 2$: rate = 4.01
- $s = 3$: rate = 6.00
- $s = 4$: rate = 8.02



Obviously, for the (empiric) order (\rightarrow Def. 9.2.6.2) of the **Gauss collocation single step method** holds

$$\text{(empiric) order} = 2s.$$

Note that the 1-stage Gauss collocation single step method is the implicit midpoint method from Eq. (9.2.6.7). □

§9.2.6.28 (Order of collocation single-step methods) What we have observed in Exp. 9.2.6.27 reflects a fundamental result on collocation single step methods as defined in (7.3.2.6).

Theorem 9.2.6.29. Order of collocation single step method [DB02, Satz .6.40]

Provided that $f \in C^p(I \times \mathbb{R}^N)$, the order (\rightarrow Def. 9.2.6.2) of an s -stage collocation single step method according to (7.3.2.6) agrees with the order (\rightarrow [Hip19, ??]) of the quadrature formula on $[0, 1]$ with nodes c_j and weights b_j , $j = 1, \dots, s$.

- By the known fact that s -point Gauss quadrature has the (maximal) order $2s$ (\rightarrow [Hip19, ??]) the s -stage **Gauss collocation single step method** whose nodes c_j are chosen as the s Gauss points on $[0, 1]$ is of order $2s$.

In light of Rem. 9.2.6.26, Thm. 9.2.6.29 immediately gives the results of Ex. 9.2.6.13 without any effort:

- ① The one-point quadrature rule with node $c_1 = 0$, weight $b_1 = 1$ is exact only for constants and, hence, of order 1 (like the explicit Euler method).
- ② The same applies to the rule with $c_1 = 1$ and weight 1 (and gives the order of the implicit Euler method).
- ③ The 1-point Gauss rule with $c_1 = 1, b_1 = 1$ has order 2 (= order of implicit midpoint rule).

□

9.2.6.5 Runge-Kutta Single-Step Methods

The formulas of (7.3.2.6) reveal that collocation single-step methods belong to a famous class of single step methods, the **Runge-Kutta single step methods**, see also Section 6.4, Section 7.3.

Definition 7.3.3.1. General Runge-Kutta single-step method

For coefficients $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, the discrete evolution $\Psi^{s,t}$ of an s -stage **Runge-Kutta single step method** (RK-SSM) for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$, is defined by

$$\mathbf{k}_i := \mathbf{f}\left(t + c_i \tau, \mathbf{u} + \tau \sum_{j=1}^s a_{ij} \mathbf{k}_j\right), \quad i = 1, \dots, s, \quad \Psi^{t,t+\tau} \mathbf{u} := \mathbf{u} + \tau \sum_{i=1}^s b_i \mathbf{k}_i.$$

The $\mathbf{k}_i \in V_0$ are called **increments**.

§9.2.6.30 (Butcher-scheme notation) A compact way to write down s -stage Runge-Kutta methods is **Butcher schemes**:

$$\begin{array}{c|ccccc} & c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ \hline \mathbf{c} & \mathfrak{A} & \hat{=} & c_2 & a_{21} & \ddots & a_{2s} \\ & \vdots & & \vdots & \ddots & \ddots & \vdots \\ & c_s & a_{s1} & \dots & & a_{ss} \\ \hline & b_1 & b_2 & \dots & \dots & b_s \end{array}, \quad \mathbf{c}, \mathbf{b} \in \mathbb{R}^s, \quad \mathfrak{A} \in \mathbb{R}^{s,s}. \quad (7.3.3.3)$$

The Butcher schemes for the finite-difference single-step methods from Ex. 9.2.6.4 are

$$\begin{array}{c|c} 0 & 0 \\ \hline 1 \end{array}$$

Explicit Euler (9.2.6.5)

$$\begin{array}{c|c} 1 & 1 \\ \hline 1 \end{array}$$

Implicit Euler (9.2.6.6)

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline 1 \end{array}$$

Implicit midpoint rule (9.2.6.7)

The following Butcher schemes characterize the Gauss collocation methods for $s = 2, 3$ [DB02, Tables 6.4 & 6.5]:

$\frac{1}{2} - \sqrt{3}/6$	$\frac{1}{4}$	$\frac{1}{4} - \sqrt{3}/6$
$\frac{1}{2} + \sqrt{3}/6$	$\frac{1}{4} + \sqrt{3}/6$	$\frac{1}{4}$
	$\frac{1}{2}$	$\frac{1}{2}$

2-stage Gauss SSM, order 4

$\frac{1}{2} - \sqrt{15}/10$	$\frac{5}{36}$	$\frac{2}{9} - \sqrt{15}/15$	$\frac{5}{36} - \sqrt{15}/30$
$\frac{1}{2}$	$\frac{5}{36} + \sqrt{15}/24$	$\frac{2}{9}$	$\frac{5}{36} - \sqrt{15}/24$
$\frac{1}{2} + \sqrt{15}/10$	$\frac{5}{36} + \sqrt{15}/30$	$\frac{2}{9} + \sqrt{15}/15$	$\frac{5}{36}$
	$\frac{5}{18}$	$\frac{4}{9}$	$\frac{5}{18}$

3-stage Gauss SSM, order 6

Review question(s) 9.2.6.31 (Single-step methods)

(Q9.2.6.31.A) Explain, why the condition $\sum_{i=1}^s b_i = 1$ has to be satisfied for the coefficient for a general Runge-Kutta method according to Def. 7.3.3.1.

Definition 7.3.3.1. General Runge-Kutta method

For coefficients $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, the discrete evolution $\Psi^{s,t}$ of an s -stage Runge-Kutta single step method (RK-SSM) for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$, is defined by

$$\mathbf{k}_i := \mathbf{f}\left(t + c_i \tau, \mathbf{u} + \tau \sum_{j=1}^s a_{ij} \mathbf{k}_j\right), \quad i = 1, \dots, s, \quad \Psi^{t,t+\tau} \mathbf{u} := \mathbf{u} + \tau \sum_{i=1}^s b_i \mathbf{k}_i.$$

The $\mathbf{k}_i \in V_0$ are called increments.

△

9.2.7 Timestepping for Method-of-Lines ODE



Video tutorial for Section 9.2.7: Timestepping for Method-of-Lines ODE: (57 minutes)
[Download link](#), [tablet notes](#)

From the method of lines (MOL) presented in Section 9.2.4 we get only a semi-discrete problem in the form of an ODE. However, for implementation we need a **fully discrete** evolution problem. This requires additional discretization in time:

semi-discrete evolution problem (9.2.4.4) + timestepping ➔ fully discrete evolution problem

An attractive feature of the method of lines is that we can apply already known integrators for initial value problems for ODEs to (9.2.4.4). Chapters 10.2.2.7 and 7 provide a rich selection of single-step timestepping methods for the approximate solution of initial-value problems for ordinary differential equations. In this section we will look at their concrete application in the method-of-lines context and establish criteria for the selection of suitable timestepping schemes.

9.2.7.1 Single-Step Methods Applied to MOL ODE

EXAMPLE 9.2.7.1 (Euler timestepping → Ex. 9.2.6.4, Section 6.2) The two Euler single-step methods that were introduced in Section 6.2.1 and Section 6.2.2 are the simplest conceivable timestepping schemes. Now we apply them to the ODE arising from method-of-lines Galerkin spatial semi-discretization

$$\begin{aligned} \mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) &= \vec{\varphi}(t) \quad \text{for } 0 < t < T, \\ \vec{\mu}(0) &= \vec{\mu}_0. \end{aligned} \tag{9.2.4.4}$$

The gist of the **explicit Euler method** (6.2.1.4) is to replace $\frac{d}{dt}$ in (9.2.4.4) with a forward difference quotient, see Ex. 9.2.6.4 or Rem. 6.2.1.5:

$$(9.2.4.4) \quad \Rightarrow \quad \mathbf{M}\vec{\mu}^{(j)} = \mathbf{M}\vec{\mu}^{(j-1)} - \tau_j(\mathbf{A}\vec{\mu}^{(j-1)} - \vec{\varphi}(t_{j-1})) , \quad j = 1, \dots, M-1 . \quad (9.2.7.2)$$

Conversely, the **implicit Euler method** (6.2.2.2) replaces $\frac{d}{dt}$ in (9.2.4.4) with a backward difference quotient:

$$(9.2.4.4) \quad \Rightarrow \quad \mathbf{M}\vec{\mu}^{(j)} = \mathbf{M}\vec{\mu}^{(j-1)} - \tau_j(\mathbf{A}\vec{\mu}^{(j)} - \vec{\varphi}(t_j)) , \quad j = 1, \dots, M-1 . \quad (9.2.7.3)$$

Note that *both* (9.2.7.2) and (9.2.7.3) require the solution of a linear system of equations in each step

$$(9.2.7.2): \quad \vec{\mu}^{(j)} = \vec{\mu}^{(j-1)} + \tau_j \mathbf{M}^{-1}(\vec{\varphi}(t_{j-1}) - \mathbf{A}\vec{\mu}^{(j-1)}) ,$$

$$(9.2.7.3): \quad \vec{\mu}^{(j)} = (\tau_j \mathbf{A} + \mathbf{M})^{-1}(\mathbf{M}\vec{\mu}^{(j-1)} + \tau_j \vec{\varphi}(t_j)) .$$

Recall from Section 6.3.2 or Ex. 9.2.6.8 that both Euler methods are merely of first order. \square

EXAMPLE 9.2.7.4 (Crank-Nicolson timestepping) In Section 6.2.3/Rem. 9.2.6.26 we have learned about the implicit midpoint method (6.2.3.2)/(9.2.6.7). When applied to the method-of-lines ODE (9.2.4.4) this method is known as the **Crank-Nicolson method**. It can be derived by replacing $\frac{d}{dt}$ in (9.2.4.4) with a symmetric difference quotient and average right hand side:

$$\begin{aligned} \mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) &= \vec{\varphi}(t) \\ \Downarrow \\ \mathbf{M} \frac{\vec{\mu}^{(j)} - \vec{\mu}^{(j-1)}}{\tau} &= -\frac{1}{2} \mathbf{A} (\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}) + \frac{1}{2} (\vec{\varphi}(t_j) + \vec{\varphi}(t_{j-1})) . \end{aligned} \quad (9.2.7.5)$$

This yields a method that is of **order 2**, as we have seen in Exp. 6.3.2.5. This order can also be inferred from Thm. 9.2.6.29, since the implicit midpoint method is a collocation single-step method induced by the midpoint quadrature rule of second order. \square

§9.2.7.6 (Application of general Runge-Kutta timestepping to method of lines ODE) Concretely, for linear parabolic evolution after spatial semi-discretization: Application of s -stage Runge-Kutta method to the method of lines ODE

$$\mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = \vec{\varphi}(t) \Leftrightarrow \dot{\vec{\mu}} = \underbrace{\mathbf{M}^{-1}(\vec{\varphi}(t) - \mathbf{A}\vec{\mu}(t))}_{=\mathbf{f}(t, \vec{\mu})} . \quad (9.2.4.4)$$

Then simply plug this into the formulas of Def. 7.3.3.1.

Definition 7.3.3.1. General Runge-Kutta single-step method

For coefficients $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, the discrete evolution $\Psi^{s,t}$ of an **s -stage Runge-Kutta single step method** (RK-SSM) for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$, is defined by

$$\mathbf{k}_i := \mathbf{f}(t + c_i \tau, \mathbf{u} + \tau \sum_{j=1}^s a_{ij} \mathbf{k}_j) , \quad i = 1, \dots, s , \quad \Psi^{t,t+\tau} \mathbf{u} := \mathbf{u} + \tau \sum_{i=1}^s b_i \mathbf{k}_i .$$

The $\mathbf{k}_i \in V_0$ are called **increments**.

What applied to (9.2.4.4) this results in the following timestepping scheme: compute $\vec{\mu}^{(j+1)}$ from $\vec{\mu}^{(j)}$ through

$$\vec{\kappa}_i \in \mathbb{R}^N: \quad \mathbf{M}\vec{\kappa}_i + \sum_{m=1}^s \tau a_{im} \mathbf{A}\vec{\kappa}_m = \vec{\varphi}(t_j + c_i \tau) - \mathbf{A}\vec{\mu}^{(j)}, \quad i = 1, \dots, s, \quad (9.2.7.7)$$

$$\vec{\mu}^{(j+1)} = \vec{\mu}^{(j)} + \tau \sum_{m=1}^s \vec{\kappa}_m b_m. \quad (9.2.7.8)$$

Note that in the case of a general (implicit) Runge-Kutta single-step method the equations (9.2.7.7) represent a linear system of equations of size $N \cdot s$. Using the **Kronecker product** of matrices for $\mathbf{A} \in \mathbb{K}^{m,n}$ and $\mathbf{B} \in \mathbb{K}^{l,k}$, $m, n, l, k \in \mathbb{N}$, defined as (\rightarrow [Hip19, ??])

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} (\mathbf{A})_{11}\mathbf{B} & (\mathbf{A})_{1,2}\mathbf{B} & \dots & \dots & (\mathbf{A})_{1,n}\mathbf{B} \\ (\mathbf{A})_{2,1}\mathbf{B} & (\mathbf{A})_{2,2}\mathbf{B} & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ (\mathbf{A})_{m,1}\mathbf{B} & (\mathbf{A})_{m,2}\mathbf{B} & \dots & \dots & (\mathbf{A})_{m,n}\mathbf{B} \end{bmatrix} \in \mathbb{K}^{nl,nk},$$

this linear system of equations can be recast into the following form:

$$(9.2.7.7) \quad \Leftrightarrow \quad (\mathbf{I}_s \otimes \mathbf{M} + \tau \mathfrak{A} \otimes \mathbf{A}) \begin{bmatrix} \vec{\kappa}_1 \\ \vdots \\ \vec{\kappa}_s \end{bmatrix} = \begin{bmatrix} \vec{\varphi}(t_j + c_1 \tau) - \mathbf{A}\vec{\mu}^{(j)} \\ \vdots \\ \vec{\varphi}(t_j + c_s \tau) - \mathbf{A}\vec{\mu}^{(j)} \end{bmatrix}. \quad (9.2.7.9)$$

□

9.2.7.2 Stability

In Section 9.2.3 we have seen that the energy norm and L^2 -norm of solutions of linear parabolic evolution problems remain bounded for all times. The same arguments confirm that this remains true for the solution $\vec{\mu}(t)$ of the semi-discrete evolution (9.2.4.4). However, some well-established single step methods applied to (9.2.4.4) may not preserve this stability property.

EXPERIMENT 9.2.7.10 (Convergence of Euler timestepping for MOL ODE) We consider a parabolic evolution problem in one spatial dimension (IBVP):

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2} \quad \text{in } [0, 1] \times]0, 1[, \\ u(t, 0) &= u(t, 1) = 0 \quad \text{for } 0 \leq t \leq 1, \quad \blacktriangleright \text{ exact solution} \quad u(t, x) = \exp(-\pi^2 t) \sin(\pi x). \\ u(0, x) &= \sin(\pi x) \quad \text{for } 0 < x < 1. \end{aligned} \quad (9.2.7.11)$$

This means that as exact solution we obtain

$$u(t, x) = \exp(-\pi^2 t) \sin(\pi x), \quad 0 < x < 1, \quad 0 < t < 1.$$

Details of the numerical experiment:

- ◆ Spatial finite element Galerkin discretization by means of linear finite elements ($V_{0,h} = \mathcal{S}_{1,0}^0(\mathcal{M})$) on equidistant mesh \mathcal{M} with meshwidth $h := \frac{1}{N+1} \rightarrow$ Section 2.3.
- ◆ $u_{N,0} := I_1 u_0$ by linear interpolation on \mathcal{M} , see Section 3.3.1.
- ◆ Timestepping by explicit and implicit Euler method (9.2.7.2), (9.2.7.3) with uniform timestep $\tau := \frac{1}{M}$.

We obtain tridiagonal $N \times N$ Galerkin matrices, see (2.3.3.2):

$$\mathbf{A} = \frac{1}{h} \begin{bmatrix} 2 & -1 & 0 & & & & & \\ -1 & 2 & -1 & & & & & \\ 0 & \ddots & \ddots & \ddots & & & & \\ & & & & 0 & & & \\ & & & & & 2 & -1 & \\ & & & & & 0 & -1 & 2 \\ 0 & & & & & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{M} = \frac{h}{6} \begin{bmatrix} 4 & 1 & 0 & & & & & & & \\ 1 & 4 & 1 & & & & & & & \\ 0 & \ddots & \ddots & \ddots & & & & & & \\ & & & & 0 & & & & & \\ & & & & & 1 & 4 & 1 & & \\ & & & & & 0 & 1 & 4 & & \end{bmatrix}.$$

We monitor the approximate space-time L^2 -norm of the discretization error

$$\text{err}^2 := h\tau \cdot \sum_{j=1}^M \sum_{i=1}^N |u(t_j, x_i) - \mu_i^{(j)}|^2. \quad (9.2.7.12)$$

Here N is the number of equidistant mesh nodes x_i in space, M the number of timesteps. In fact, this formula can be viewed as a 2D trapezoidal rule on the space-time cylinder $[0,1] \times [0,1]$.

Space-time (discrete) L^2 -norm of error for **explicit Euler** timestepping:

$N \setminus M$	50	100	200	400	800	1600	3200
5	Inf	0.009479	0.006523	0.005080	0.004366	0.004011	0.003834
10	Inf	Inf	Inf	Inf	0.001623	0.001272	0.001097
20	Inf	Inf	Inf	Inf	Inf	Inf	0.000405
40	Inf	Inf	Inf	Inf	Inf	Inf	Inf
80	Inf	Inf	Inf	Inf	Inf	Inf	Inf
160	Inf	Inf	Inf	Inf	Inf	Inf	Inf
320	Inf	Inf	Inf	Inf	Inf	Inf	Inf

Here Inf indicates that the method suffered an exponential blow-up.

Space-time (discrete) L^2 -norm of error for **implicit Euler** timestepping:

$N \setminus M$	50	100	200	400	800	1600	3200
5	0.007025	0.001828	0.000876	0.002257	0.002955	0.003306	0.003482
10	0.009641	0.004500	0.001826	0.000461	0.000228	0.000575	0.000749
20	0.010303	0.005175	0.002509	0.001149	0.000461	0.000116	0.000058
40	0.010469	0.005345	0.002681	0.001321	0.000634	0.000289	0.000116
80	0.010511	0.005387	0.002724	0.001364	0.000677	0.000332	0.000159
160	0.010521	0.005398	0.002734	0.001375	0.000688	0.000343	0.000170
320	0.010524	0.005400	0.002737	0.001378	0.000691	0.000346	0.000172

Summary:

For **explicit** Euler timestepping we observe a glaring **instability** (exponential blow-up) in case of *large timestep combined with fine mesh*.

Implicit Euler timestepping incurs **no blow-up** for any combination of spatial and temporal mesh width.

Please compare the observations made in this experiment with what we saw in Exp. 7.3.1.1 and notice the striking similarity. □

The instability observed in the previous experiment could be a quirk of the explicit Euler single-step method investigated there. In Section 6.4 many more so-called **explicit Runge-Kutta methods** have been presented, whose increments can be computed one after the other, which entails inverting only the mass

matrix \mathbf{M} in the case of the method-of-lines ODE (9.2.4.4). By means of suitable numerical quadrature it is often possible to obtain **diagonal mass matrices**, a trick called **mass lumping**. Then the increments can be computed with asymptotic effort $O(Ns)$ for $N \rightarrow \infty$!

Conversely, general Runge-Kutta methods applied to (9.2.4.4) invariably lead to big sparse linear systems for the Ns components of the increments, recall § 9.2.7.6. Thus, explicit RK-SSM present an apparently attractive option, more so, as high-order adaptive variants are readily available in libraries Section 6.5. However, the next experiments dashes all hopes, because stability problems also haunt adaptive explicit RK-SSMs.

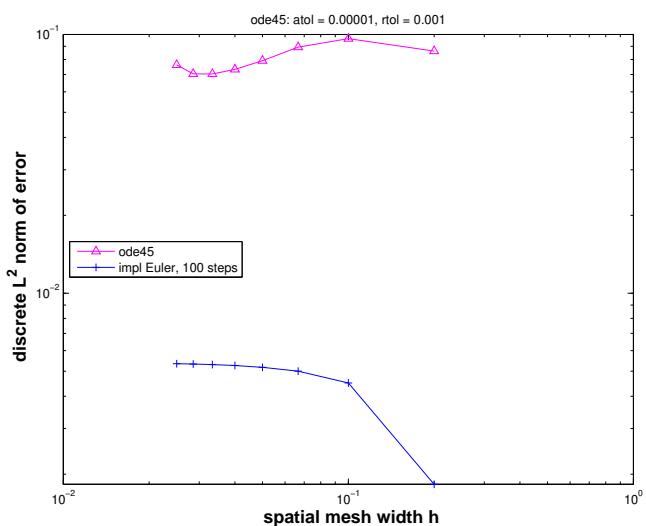
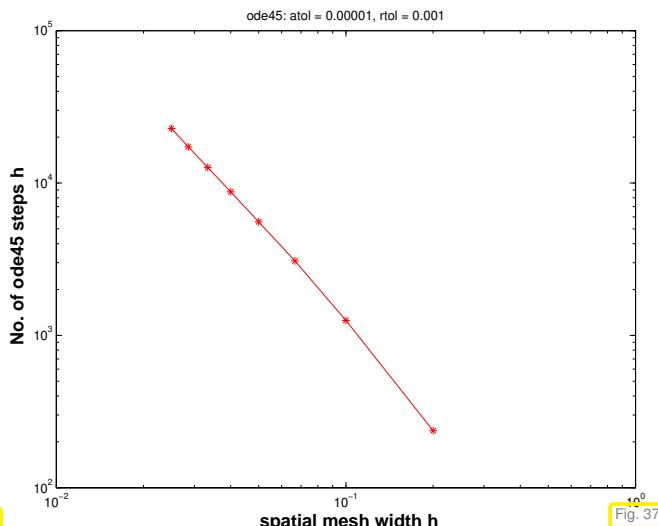
EXPERIMENT 9.2.7.13 (Adaptive explicit high-order Runge-Kutta method for discrete parabolic evolution)

We consider the same IBVP and spatial discretization as in Exp. 9.2.7.10

This time we perform timestepping by means of an adaptive **explicit** Runge-Kutta single-step method using the **Ode45** auxiliary class presented in § 6.5.3.3. It relies on an embedded RK-SSM of order 5 and 4.

We watch

- ◆ the number of timesteps as a function of spatial meshwidth h ,
- ◆ the discrete L^2 -error (9.2.7.12).



Observations:

- ◆ **Ode45** suffers a dramatic increase of no. of timesteps for $h_M \rightarrow 0$ without gain in accuracy.
- ◆ The implicit Euler achieves better accuracy with only 100 equidistant timesteps!

This is not a new observation. A similar failure of **Ode45** was manifest in Ex. 7.0.0.1, Exp. 7.1.0.1, Ex. 7.1.0.35. ↓

This reminds us of the **stiff initial value problems** studied in Section 7.2:

Notion 7.2.0.7. Stiff IVP

An initial value problem is called **stiff**, if stability imposes much tighter timestep constraints on **explicit single step methods** than the accuracy requirements.

The previous Exp. 9.2.7.13 suggests the following conjecture:

Stiffness of parabolic evolution problems

The spatially semi-discrete parabolic evolution problem (9.2.4.4) arising from the method-of-lines spatial Galerkin finite-element discretization of a second-order parabolic IBVP (9.2.2.9) is **stiff** in the sense of Notion 7.2.0.7.

Admittedly, “stiffness” remains a fuzzy notion. It cannot be made more precise on the abstract level, but has to be discussed for concrete evolution problem, which is done next.

Let us try to understand, why semi-discrete parabolic evolutions (9.2.4.4) arising from the method of lines lead to stiff initial value problems.

§9.2.7.15 (Diagonalization of method-of-lines ODE) We employ the analysis technique of **Diagonalization**, cf. (6.1.1.14) and Eq. (7.1.0.38). Diagonalization (also called spectral decomposition) is a very versatile technique for decomposing a big problem into *decoupled* small, even one-dimensional, problems. Here we discuss it for the method-of-lines ODE (9.2.4.4):

$$\mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = \vec{\varphi}(t) . \quad (9.2.4.4)$$

Let $\vec{\psi}_1, \dots, \vec{\psi}_N \in \mathbb{R}^N$ denote the N linearly independent **generalized eigenvectors** satisfying

$$\mathbf{A} \vec{\psi}_i = \lambda_i \mathbf{M} \vec{\psi}_i , \quad \vec{\psi}_j^\top \mathbf{M} \vec{\psi}_i = \delta_{ij} , \quad 1 \leq i, j \leq N . \quad (9.2.7.16)$$

with positive **eigenvalues** $\lambda_i > 0$. Introducing the regular square matrices

$$\mathbf{T} = [\vec{\psi}_1, \dots, \vec{\psi}_N] \in \mathbb{R}^{N,N} , \quad (9.2.7.17)$$

$$\mathbf{D} := \text{diag}(\lambda_1, \dots, \lambda_N) \in \mathbb{R}^{N,N} , \quad (9.2.7.18)$$

we can rewrite (9.2.7.16) as

$$\mathbf{AT} = \mathbf{MTD} , \quad \mathbf{T}^\top \mathbf{MT} = \mathbf{I} . \quad (9.2.7.19)$$

Supplement 9.2.7.20. The existence of eigenvectors $\vec{\psi}_i$ with positive associated eigenvalues is guaranteed, since both \mathbf{A} and \mathbf{M} are positive definite: Thus, the **generalized eigenvalue problem** (9.2.7.16) can be transformed to a standard eigenvalue problem for a symmetric matrix by multiplying from left and right with the inverse of the “square root” $\mathbf{M}^{1/2}$ of \mathbf{M} , see [Hip19, ??]:

$$\mathbf{A} \vec{\psi}_i = \lambda_i \mathbf{M} \vec{\psi}_i \implies \underbrace{\mathbf{M}^{-1/2} \mathbf{A} \mathbf{M}^{-1/2}}_{\text{s.p.d. matrix}} \vec{\rho}_i = \lambda_i \vec{\rho}_i \quad \text{with} \quad \vec{\rho}_i := \mathbf{M}^{1/2} \vec{\psi}_i .$$

Then apply the result that every symmetric matrix can be diagonalized by means of an orthogonal transformation [Hip19, ??]. \square

We can use the generalized eigenvalue problems in different ways to achieve the diagonalization of the method-of-lines ODE. Of course, the resulting small evolution problems will be the same.

Diagonalization approach ①: Expand $\vec{\mu}(t)$ in the eigenvectors $\vec{\psi}_i$ (with time-dependent expansion coefficients)

$$\vec{\mu}(t) = \sum_{k=1}^N \eta_k(t) \vec{\psi}_k , \quad (9.2.7.21)$$

and plug this expansion into

$$\mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = \vec{\varphi}(t) . \quad (9.2.4.4)$$

Using (9.2.7.16) this yields

$$\sum_{k=1}^N \frac{d}{dt} \eta_k(t) \mathbf{M} \vec{\psi}_k + \eta_k(t) \lambda_k \mathbf{M} \vec{\psi}_k = \vec{\varphi}(t) .$$

Multiply from left with $\vec{\psi}_i^\top, i = 1, \dots, N$, and use (9.2.7.16) again:

$$\Rightarrow \frac{d}{dt} \eta_i(t) + \lambda_i \eta_i(t) = \vec{\psi}_i^\top \vec{\varphi}(t) .$$

We have ended up with N decoupled scalar linear ODEs.

Diagonalization approach ②: Using compact matrix notations, set

$$\vec{\mu}(t) = \mathbf{T} \vec{\eta}(t) \Leftrightarrow \mathbf{T}^\top \mathbf{M} \vec{\mu}(t) = \vec{\eta}(t) .$$

Substitute this in (9.2.4.4) and invoke (9.2.7.19):

$$\Rightarrow \mathbf{M} \mathbf{T} \frac{d}{dt} \vec{\eta}(t) + \mathbf{M} \mathbf{T} \mathbf{D} \vec{\eta}(t) = \vec{\varphi}(t) .$$

Then multiply this equation from left with \mathbf{T}^\top and use (9.2.7.19) again:

$$\Rightarrow \frac{d}{dt} \vec{\eta}(t) + \mathbf{D} \vec{\eta}(t) = \mathbf{T}^\top \vec{\varphi}(t) .$$

Through both approaches, setting $\vec{\eta} = (\eta_1, \dots, \eta_N)^\top \in \mathbb{R}^N$, we have thus arrived at the *transformed ODE*

$$(9.2.4.4) \quad \vec{\eta} := \mathbf{T}^\top \mathbf{M} \vec{\mu} \quad \frac{d}{dt} \vec{\eta}(t) + \mathbf{D} \vec{\eta} = \mathbf{T}^\top \vec{\varphi}(t) . \quad (9.2.7.22)$$

(Note that, thanks to the \mathbf{M} -orthogonality of the $\vec{\psi}_i$ stated in (9.2.7.16), (9.2.7.21) is equivalent to $\vec{\eta} = \mathbf{T}^\top \mathbf{M} \vec{\mu}$.)

- Since \mathbf{D} is diagonal, (9.2.7.22) amounts to N decoupled scalar ODEs (for eigencomponents η_i of $\vec{\mu}$).

Note:

$$\text{for } \vec{\varphi} \equiv 0, \lambda > 0 : \quad \eta_i(t) = \exp(-\lambda_i t) \eta_i(0) \rightarrow 0 \quad \text{for } t \rightarrow \infty$$

□

§9.2.7.23 (Diagonalization applied to explicit Euler timestepping) As in § 7.1.0.40 the above diagonalizing transformation can be applied to the explicit Euler timestepping (9.2.7.2) (for $\vec{\varphi} \equiv 0$, uniform timestep $\tau > 0$)

$$\vec{\mu}^{(j)} = \vec{\mu}^{(j-1)} - \tau \mathbf{M}^{-1} \mathbf{A} \vec{\mu}^{(j-1)} \quad \vec{\eta} := \mathbf{T}^\top \mathbf{M} \vec{\mu} \quad \vec{\eta}^{(j)} = \vec{\eta}^{(j-1)} - \tau \mathbf{D} \vec{\eta}^{(j-1)} ,$$

that is, the decoupling of eigencomponents carries over to the explicit Euler method: for $i = 1, \dots, N$

$$\eta_i^{(j)} = \eta_i^{(j-1)} - \tau \lambda_i \eta_i^{(j-1)} \Rightarrow \boxed{\eta_i^{(j)} = (1 - \tau \lambda_i)^j \eta_i^{(0)}} . \quad (9.2.7.24)$$



$$|1 - \tau \lambda_i| < 1 \Leftrightarrow \lim_{j \rightarrow \infty} \eta_i^{(j)} = 0 .$$

The condition $|1 - \tau \lambda_i| < 1$ enforces a

$$\text{timestep size constraint: } \tau < \frac{2}{\lambda_i}$$

in order to achieve the qualitatively correct behavior $\lim_{j \rightarrow \infty} \eta_i^{(j)} = 0$ and to avoid blow-up $\lim_{j \rightarrow \infty} |\eta_i^{(j)}| = \infty$: the timestep size constraint (9.2.7.23) is necessary *only* for the sake of stability (not in order to guarantee a prescribed accuracy).

This accounts to the observed blow-ups in Exp. 9.2.7.10. From Exp. 9.2.7.13 we conclude that adaptive stepsize control as introduced in Section 6.5 manages to enforce the timestep constraint, but at the expense of prohibitively small timesteps that render the method *grossly inefficient, if some of the λ_i are large.* \square

Remark 9.2.7.25 (von Neumann stability analysis) The diagonalization approach to the stability analysis of timestepping methods for fully discrete linear evolution problems is a generalization of the classical **von Neumann stability analysis**, which applies to cases, where the eigenfunctions of the generalized eigenvalue problem (9.2.7.16) are Fourier harmonics (sines/cosines). This special version of stability analysis will be covered in Section 11.4.3. \square

The next numerical demonstrations and Lemma show that $\lambda_{\max} := \max_i \lambda_i$ will inevitably become huge for finite element discretization on fine meshes.

EXPERIMENT 9.2.7.26 (Behavior of generalized eigenvalues of $\mathbf{A}\vec{\mu} = \lambda\mathbf{M}\vec{\mu}$) Bilinear forms associated with parabolic IBVP and homogeneous Dirichlet boundary conditions

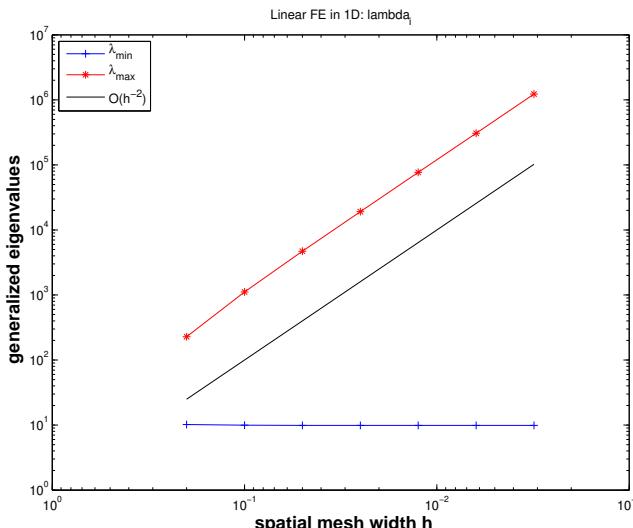
$$a(u, v) = \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx , \quad m(u, v) = \int_{\Omega} u(x)v(x) \, dx , \quad u, v \in H_0^1(\Omega) .$$

Linear finite element Galerkin discretization, see Section 2.3 for 1D, and Section 2.4 for 2D.

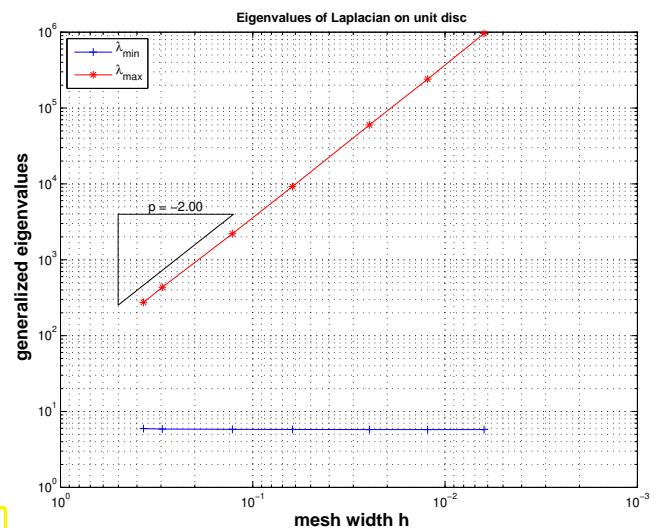
Numerical experiments in 1D & 2D:

- $\Omega =]0, 1[$, equidistant meshes \rightarrow Exp. 9.2.7.10
- “disk domain” $\Omega = \{x \in \mathbb{R}^2 : \|x\| < 1\}$, sequence of regularly refined meshes.

Monitored: largest and smallest generalized eigenvalue



$$\Omega =]0, 1[$$



$$\Omega = \{x \in \mathbb{R}^2 : \|x\| < 1\}$$

Observation:

- ◆ $\lambda_{\min} := \min_i \lambda_i$ does hardly depend on the mesh width.
- ◆ $\lambda_{\max} := \max_i \lambda_i$ displays a $O(h_M^{-2})$ growth as $h_M \rightarrow 0$

Remark 9.2.7.27 (Spectrum of elliptic operators) The observation made in Exp. 9.2.7.26 is not surprising! Now we establish them as general property of finite element Galerkin matrices for second-order linear scalar variational problems.

To do so, let us translate the generalized eigenproblem “back to the ODE/PDE level”:

$$\begin{aligned}
 \mathbf{A}\vec{\mu} = \lambda \mathbf{M}\vec{\mu} & \quad (9.2.7.28) \\
 \Updownarrow & \\
 u_h \in V_{0,h}: \quad \mathbf{a}(u_h, v_h) = \lambda \mathbf{m}(u_h, v_h) \quad \forall v_h \in V_{0,h} . & \\
 \text{---} & \leftarrow \text{“undo Galerkin discretization”} \\
 u \in H_0^1(\Omega): \quad \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx = \lambda \int_{\Omega} u \cdot v \, dx \quad \forall v \in H_0^1(\Omega) . & \\
 \Downarrow & \\
 -\Delta u = \lambda u \quad \text{in } \Omega , \quad u = 0 \quad \text{on } \partial\Omega , & \quad (9.2.7.29)
 \end{aligned}$$

which is a so-called **elliptic eigenvalue problem**.

It is easily solved in 1D on $\Omega =]0, 1[$:

$$\begin{aligned}
 (9.2.7.29) \hat{=} \quad \frac{d^2u}{dx^2}(x) &= \lambda u(x) , \quad 0 < x < 1 , \quad u(0) = u(1) = 0 . \\
 \Rightarrow \quad u_k(x) &= \sin(k\pi x) \quad \leftrightarrow \quad \lambda_k = (\pi k)^2 , \quad k \in \mathbb{N} .
 \end{aligned}$$

Note that we find an infinite number of eigenfunctions and eigenvalues, parameterized by $k \in \mathbb{N}$. Assuming that the λ_k are sorted, the eigenvalues tend to ∞ for $k \rightarrow \infty$:

$$\lambda_k = O(k^2) \quad \text{for } k \rightarrow \infty .$$

Of course, the matrix eigenvalue problem (9.2.7.28) can have a finite number of eigenvectors only. Crudely speaking, they correspond to those eigenfunctions $u_k(x) = \sin(k\pi x)$ that can be resolved by the mesh (if u_k “oscillates too much”, then it cannot be represented on a grid). These are the first N so that we find in 1D for an equidistant mesh

$$\lambda_{\max} = O(N^2) = O(h_{\mathcal{M}}^{-2}) .$$

This is heuristics, but the following Lemma will a precise statement.

Lemma 9.2.7.30. Behavior of of generalized eigenvalues

Let \mathcal{M} be a simplicial mesh and \mathbf{A}, \mathbf{M} denote the Galerkin matrices for the bilinear forms $\mathbf{a}(u, v) = \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx$ and $\mathbf{m}(u, v) = \int_{\Omega} u(x)v(x) \, dx$, respectively, and $V_{0,h} := \mathcal{S}_{p,0}^0(\mathcal{M})$. Then the smallest and largest generalized eigenvalues of $\mathbf{A}\vec{\mu} = \lambda \mathbf{M}\vec{\mu}$, denoted by λ_{\min} and λ_{\max} , satisfy

$$\frac{1}{\text{diam}(\Omega)^2} \leq \lambda_{\min} \leq C , \quad \lambda_{\max} \geq Ch_{\mathcal{M}}^{-2} ,$$

where the “generic constants” (\rightarrow Rem. 3.3.5.8) depend only on the polynomial degree p , the domain Ω , and the shape regularity measure $\rho_{\mathcal{M}}$.

Proof. (partial) We rely on the **Courant-Fischer min-max theorem** [Hip19, ??] that, among other consequences, expresses the boundaries of the spectrum of a symmetric matrix through the extrema of its Rayleigh quotient

$$\mathbf{T} = \mathbf{T}^T \in \mathbb{R}^{N,N} \Rightarrow \lambda_{\min}(\mathbf{T}) = \min_{\vec{\xi} \in \mathbb{R}^N \setminus \{0\}} \frac{\vec{\xi}^T \mathbf{T} \vec{\xi}}{\vec{\xi}^T \vec{\xi}}, \quad \lambda_{\max}(\mathbf{T}) = \max_{\vec{\xi} \in \mathbb{R}^N \setminus \{0\}} \frac{\vec{\xi}^T \mathbf{T} \vec{\xi}}{\vec{\xi}^T \vec{\xi}}.$$

Apply this to the generalized eigenvalue problem (Recall the concept of a “square root” $\mathbf{M}^{1/2}$ of an s.p.d. matrix \mathbf{M} , see [Hip19, ??])

$$\mathbf{A}\vec{\mu} = \lambda \mathbf{M}\vec{\mu} \quad \vec{\zeta} := \mathbf{M}^{1/2}\vec{\mu} \Leftrightarrow \underbrace{\mathbf{M}^{-1/2} \mathbf{A} \mathbf{M}^{-1/2}}_{=: \mathbf{T}} \vec{\zeta} = \lambda \vec{\zeta}. \\ \lambda_{\min} = \min_{\vec{\mu} \neq 0} \frac{\vec{\mu}^T \mathbf{A} \vec{\mu}}{\vec{\mu}^T \mathbf{M} \vec{\mu}}, \quad \lambda_{\max} = \max_{\vec{\mu} \neq 0} \frac{\vec{\mu}^T \mathbf{A} \vec{\mu}}{\vec{\mu}^T \mathbf{M} \vec{\mu}}. \quad (9.2.7.31)$$

As a consequence we only have to find bounds for the extrema of a **generalized Rayleigh quotient**, cf. [Hip19, ??]. This generalized Rayleigh quotient can be expressed as

$$\frac{\vec{\mu}^T \mathbf{A} \vec{\mu}}{\vec{\mu}^T \mathbf{M} \vec{\mu}} = \frac{a(u_h, u_h)}{m(u_h, u_h)}, \quad \vec{\mu} \hat{=} \text{coefficient vector for } u_h. \quad (9.2.7.32)$$

Now we discuss a lower bound for λ_{\max} , which can be obtained by inserting a suitable *candidate function* into (9.2.7.32).

Discussion for special setting: $V_{0,h} = \mathcal{S}_1^0(\mathcal{M})$ on triangular mesh \mathcal{M}

Candidate function: “tent function” $u_h = b_h^i$ (\rightarrow Section 2.4.3) for some node $x^i \in \mathcal{V}(\mathcal{M})$ of the mesh!

By elementary computations as in Section 2.4.5 we find

$$a(b_h^i, b_h^i) \approx C, \quad m(b_h^i, b_h^i) \leq C \max_{K \in \mathcal{U}(x^i)} h_K^2, \quad (9.2.7.33)$$

where the generic constants $C > 0$ depend on the shape regularity measure $\rho_{\mathcal{M}}$ only.

$$(9.2.7.31) \& (9.2.7.33) \Rightarrow \lambda_{\max} \geq Ch_{\mathcal{M}}^{-2}.$$

This provides the estimate (from below) for the largest eigenvalue. \square

Lemma 9.2.7.30 & (9.2.7.23) imply concrete timestep constraint for explicit Euler method in the case of spatial Galerkin discretization by means of Lagrangian finite elements

$$\tau < Ch_{\mathcal{M}}^2, \quad (9.2.7.34)$$

with $C > 0$ depending only on the polynomial degree and the shape regularity measure $\rho_{\mathcal{M}}$.

From Section 7.3 and also Exp. 9.2.7.10 we already know that some *implicit* single step methods are not affected by stability induced timestep constraints. This can be confirmed by rigorous analysis.

§9.2.7.35 (Diagonlization applied to implicit Euler timestepping) Recall § 7.3.1.2 and apply the diagonalization technique, see (9.2.7.22), to implicit Euler timestepping with uniform timestep $\tau > 0$

$$\vec{\mu}^{(j)} = \vec{\mu}^{(j-1)} - \tau \mathbf{M}^{-1} \mathbf{A} \vec{\mu}^{(j)} \quad \vec{\eta} := \mathbf{T}^T \mathbf{M} \vec{\mu} \quad \vec{\eta}^{(j)} = \vec{\eta}^{(j-1)} - \tau \mathbf{D} \vec{\eta}^{(j)},$$

that is, the decoupling of eigencomponents carries over to the implicit Euler method: for $i = 1, \dots, N$

$$\eta_i^{(j)} = \eta_i^{(j-1)} - \tau \lambda_i \eta_i^{(j)} \Rightarrow \boxed{\eta_i^{(j)} = \left(\frac{1}{1+\tau\lambda_i}\right)^j \eta_i^{(0)}}. \quad (9.2.7.36)$$

$$\left[\left| \frac{1}{1+\tau\lambda_i} \right| < 1 \quad \text{and} \quad \lambda_i > 0 \quad \Rightarrow \quad \right] \quad \lim_{j \rightarrow \infty} \eta_i^{(j)} = 0 \quad \forall \tau > 0. \quad (9.2.7.37)$$

- ☞ The implicit Euler method for (9.2.4.4) will never suffer blow-up regardless of timestep size; it is **unconditionally stable**. ↓

§9.2.7.38 (Diagonalization applied to general Runge-Kutta timestepping) As in § 7.1.0.43 the diagonalization trick from § 9.2.7.15 can be applied to general Runge-Kutta single step methods (RKSSM, → Def. 7.3.3.1). We can start from the increment equations

$$\vec{\kappa}_i \in \mathbb{R}^N: \quad \mathbf{M} \vec{\kappa}_i + \sum_{m=1}^s \tau a_{im} \mathbf{A} \vec{\kappa}_m = \vec{\phi}(t_j + c_i \tau) - \mathbf{A} \vec{\mu}^{(j)}, \quad i = 1, \dots, s, \quad (9.2.7.7)$$

$$\vec{\mu}^{(j+1)} = \vec{\mu}^{(j)} + \tau \sum_{m=1}^s \vec{\kappa}_m b_m. \quad (9.2.7.8)$$

and apply diagonalization using

$$\mathbf{A} \mathbf{T} = \mathbf{M} \mathbf{T} \mathbf{D}, \quad \mathbf{D} = \begin{bmatrix} \lambda_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \lambda_N \end{bmatrix}, \quad \mathbf{T}^\top \mathbf{M} \mathbf{T} = \mathbf{I}, \quad (9.2.7.19)$$

and the transformed coefficient and increment vectors:

$$\begin{aligned} \vec{\mu}^{(j)} &= \mathbf{T} \vec{\eta}^{(j)}, & \Leftrightarrow & \mathbf{T}^\top \mathbf{M} \vec{\mu}^{(j)} = \vec{\eta}^{(j)}, \\ \vec{\kappa}_i &= \mathbf{T} \vec{\zeta}_i. & & \mathbf{T}^\top \mathbf{M} \vec{\kappa}_i = \vec{\zeta}_i. \end{aligned}$$

We multiply the increment equations (9.2.7.7) with \mathbf{T}^\top from the left and rewrite them in terms of $\vec{\zeta}_i$ and $\vec{\eta}^{(j)}$:

$$\underbrace{\mathbf{T}^\top \mathbf{M} \mathbf{T}}_{=\mathbf{I}} \vec{\zeta}_i + \sum_{m=1}^s \tau a_{im} \underbrace{\mathbf{T}^\top \mathbf{A} \mathbf{T}}_{=\mathbf{D}} \vec{\zeta}_m = \mathbf{T}^\top \vec{\phi}(t_j + c_i \tau) - \underbrace{\mathbf{T}^\top \mathbf{A} \mathbf{T}}_{=\mathbf{D}} \vec{\eta}^{(j)}, \quad i = 1, \dots, s,$$

$$\vec{\eta}^{(j+1)} = \vec{\eta}^{(j)} + \tau \sum_{m=1}^s \vec{\zeta}_m b_m.$$

We can write these equations in components taking into account that \mathbf{D} is diagonal with diagonal entries $\lambda_j, j = 1, \dots, N$.

$$\begin{aligned} \left(\vec{\zeta}_i\right)_k + \sum_{m=1}^s \tau a_{im} \lambda_k \left(\vec{\zeta}_m\right)_k &= \left(\mathbf{T}^\top \vec{\phi}(t_j + c_i \tau)\right)_k - \lambda_k \left(\vec{\eta}^{(j)}\right)_k, \quad i = 1, \dots, s, \\ \left(\vec{\eta}^{(j+1)}\right)_k &= \left(\vec{\eta}^{(j)}\right)_k + \tau \sum_{m=1}^s \left(\vec{\zeta}_m\right)_k b_m. \end{aligned} \quad (9.2.7.39)$$

Compare this with the formulas arising when applying the same Runge-Kutta single step method to the scalar ODE $\dot{\eta} = -\lambda\eta + \psi(t)$:

$$\begin{aligned}\kappa_i &= -\lambda(\eta^{(j)} + \tau \sum_{m=1}^s a_{im}\kappa_m) + \psi(t_j + c_i\tau) \quad i = 1, \dots, s, \\ \eta^{(j+1)} &= \eta^{(j)} + \tau \sum_{m=1}^s b_m \kappa_m.\end{aligned}\tag{9.2.7.40}$$

Obviously, (9.2.7.39) for fixed k and $\lambda_k = \lambda$ and (9.2.7.40) describe the same recursion. Summing up, we have found that the following diagram commutes

$$\begin{array}{ccc} \mathbf{M} \frac{d}{dt} \vec{\mu} + \mathbf{A} \mu = 0 & \xrightarrow{\text{transformation } \vec{\eta} = \mathbf{T}^T \mathbf{M} \vec{\mu}} & \frac{d}{dt} \eta_i = -\lambda_i \eta_i, \quad i = 1, \dots, N \\ \text{RK-SSM} \downarrow & & \downarrow \text{RK-SSM} \\ \vec{\mu}^{(j)} = \Psi^{t,t+\tau} \vec{\mu}^{(j-1)} & \xrightarrow{\text{transformation } \vec{\eta}^{(j)} = \mathbf{T}^T \mathbf{M} \vec{\mu}^{(j)}} & \eta_i^{(j)} = \tilde{\Psi}_i^{t,t+\tau} \eta_i^{(j-1)}, \quad i = 1, \dots, N. \end{array}\tag{9.2.7.41}$$

The bottom line is

that we have to study the behavior of the RK-SSM *only* for linear scalar ODEs $\dot{y} = -\lambda y$, $\lambda > 0$.

This is the gist of the **model problem analysis** discussed in Section 7.3.

There we saw that everything boils down to inspecting the modulus of a rational **stability function** on \mathbb{C} , see Thm. 7.3.4.4. This gave rise to the concept of **L-stability**, see Def. 9.2.7.46. Here, we will not delve into a study of stability functions. \square

Unconditional stability of single step methods

Necessary condition for ***universal unconditional stability*** of a single step method for semi-discrete parabolic evolution problem (9.2.4.4)(“method of lines”):

The discrete evolution $\Psi_\lambda^\tau : \mathbb{R} \mapsto \mathbb{R}$ of the single step method applied to the scalar ODE $\dot{u} = -\lambda u$ satisfies

$$\lambda > 0 \Rightarrow \lim_{j \rightarrow \infty} (\Psi_\lambda^\tau)^j u_0 = 0 \quad \forall u_0 \in \mathbb{R}, \quad \forall \tau > 0.\tag{9.2.7.43}$$

§9.2.7.44 (Stability functions →Section 7.3.4) It is straightforward to elaborate the discrete evolution Ψ_λ for a general Runge-Kutta single-step method according to Def. 7.3.3.1 applied to the scalar linear autonomous ODE $\dot{u} = -\lambda u$. For a method characterized by the Butcher scheme $\begin{array}{c|cc} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array}$ (see (7.3.3.3)) we obtain [Hip19, ??]

$$\Psi_\lambda^{t,t+\tau} u = S(-\lambda\tau)u,\tag{9.2.7.45a}$$

with rational **stability function**

$$S(z) = 1 + z\mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{1} = \frac{\det(\mathbf{I} - z\mathbf{A} + z\mathbf{1}\mathbf{b}^T)}{\det(\mathbf{I} - z\mathbf{A})}, \quad \mathbf{1} = [1, \dots, 1]^\top \in \mathbb{R}^s.\tag{9.2.7.45b}$$

The next definition connects (9.2.7.43) with properties of the stability function.

Definition 9.2.7.46. L(π)-stability

A Runge-Kutta single-step method satisfying (9.2.7.43) is called **L(π)-stable**, if its stability function $S(z)$ according to (9.2.7.45b) satisfies

- (i) $|S(z)| < 1$ for all $z < 0$, and
- (ii) “ $S(-\infty)$ ” := $\lim_{z \in \mathbb{R} \rightarrow -\infty} S(z) = 0$.

L(π)-stability can be checked by inspecting the stability function, which always is a rational function, $S(z) = \frac{P(z)}{Q(z)}$ with polynomials $P, Q \in \mathcal{P}_s(\mathbb{R})$ Cor. 7.3.4.6. We can directly compute the limit of S for $z \rightarrow \infty$, which features in Item (ii):

$$(9.2.7.45b) \Rightarrow S(-\infty) = 1 - \mathbf{b}^T \mathfrak{A}^{-1} \mathbf{1}. \quad (9.2.7.47)$$

This yields a sufficient condition for Item (ii) Rem. 7.3.4.18:

$$\text{If } \mathbf{b}^T = (\mathfrak{A})_{:,j}^T \text{ (row of } \mathfrak{A}) \Rightarrow S(-\infty) = 0. \quad (9.2.7.48)$$

This condition can be read off the Butcher scheme of a RK-SSM

$$\begin{array}{c|ccccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_{s-1} & a_{s-1,1} & \cdots & a_{s-1,s} \\ \hline 1 & b_1 & \cdots & b_s \\ \hline & b_1 & \cdots & b_s \end{array} \Rightarrow S(-\infty) = 0.$$

□

EXAMPLE 9.2.7.49 (L(π)-stable Runge-Kutta single step methods) Simplest L(π)-stable Runge-Kutta single step method = implicit Euler timestepping (9.2.7.3).

Next we list two commonly used higher order L(π)-stable Runge-Kutta methods, specified through their Butcher schemes, see (7.3.3.3):

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array} \quad (9.2.7.50)$$

RADAU-3 scheme (order 3)

$$\begin{array}{c|cc} \lambda & \lambda & 0 \\ 1 & 1-\lambda & \lambda \\ \hline 1-\lambda & \lambda \end{array}, \quad \lambda := 1 - \frac{1}{2}\sqrt{2}, \quad (9.2.7.51)$$

SDIRK-2 scheme (order 2)

More examples are given in Ex. 7.3.4.21. This introduces the class of RADAU RK-SSMs, which provides L(π)-stable Runge-Kutta methods up to arbitrary order.

□

Review question(s) 9.2.7.52 (Timestepping for method-of-lines ODE)

(Q9.2.7.52.A) The implicit 2-stage Gauss-Radau method is described by the Butcher scheme

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}.$$

Describe the implementation of a single step of this method for the method-of-lines ODE

$$\frac{d}{dt} \vec{\mu}(t) + \mathbf{A} \vec{\mu}(t) = \vec{\varphi}(t). \quad (9.2.4.4)$$

Definition 7.3.3.1. General Runge-Kutta method

For coefficients $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, the discrete evolution $\Psi^{s,t}$ of an s -stage Runge-Kutta single step method (RK-SSM) for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$, is defined by

$$\mathbf{k}_i := \mathbf{f}\left(t + c_i \tau, \mathbf{u} + \tau \sum_{j=1}^s a_{ij} \mathbf{k}_j\right), \quad i = 1, \dots, s, \quad \Psi^{t,t+\tau} \mathbf{u} := \mathbf{u} + \tau \sum_{i=1}^s b_i \mathbf{k}_i.$$

The $\mathbf{k}_i \in V_0$ are called **increments**.

Widely used is the **Butcher-scheme** notation introduced in Section 7.3.3:

$$\begin{array}{c|ccccc} \mathbf{c} & \mathfrak{A} \\ \hline & \mathbf{b}^T \end{array} \hat{=} \begin{array}{c|ccccc} c_1 & a_{11} & a_{12} & \dots & \dots & a_{1s} \\ c_2 & a_{21} & \ddots & & & a_{2s} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ c_s & a_{s1} & \dots & & & a_{ss} \\ \hline & b_1 & b_2 & \dots & \dots & b_s \end{array}, \quad \mathbf{c}, \mathbf{b} \in \mathbb{R}^s, \quad \mathfrak{A} \in \mathbb{R}^{s,s}. \quad (7.3.3.3)$$

(Q9.2.7.52.B) [Reduction to scalar model problem] How does **blow-up** of a timestepping scheme for the method-of-lines ODE (9.2.4.4) manifest itself.

(Q9.2.7.52.C) Explain, why it is enough to understand the behavior of a Runge-Kutta single-step method for the scalar IVP

$$\dot{y} = \lambda y, \quad y(0) = 1, \quad \lambda \in \mathbb{R},$$

in order to predict its stability properties when applied to the method-of-lines ODE (9.2.4.4).

(Q9.2.7.52.D) [Timestep constraint] We discretize the parabolic initial-value problem

$$\begin{aligned} \frac{\partial u}{\partial t} - \operatorname{div}(\alpha(x) \operatorname{grad} u) &= 0 \quad \text{in } \Omega \times [0, T], \\ u &= 0 \quad \text{on } \partial\Omega \times [0, T], \\ u(x, 0) &= u_0(x) \quad \text{on } \Omega, \end{aligned}$$

in space by means of quadratic Lagrangian finite elements on sequences of triangular meshes created by uniform regular refinement. Method-of-lines timestepping relies on an **explicit** Runge-Kutta single-step method with uniform timestep $\tau > 0$.

What relationship between timestep τ and meshwidth h_M has to be imposed in order to ensure stability of the resulting fully discrete evolution?

(Q9.2.7.52.E) Consider the evolution problem

$$t \in]0, T[\mapsto u(t) \in H^1(\Omega) : \quad \frac{d}{dt} \int_{\partial\Omega} u(t) v \, dS + \int_{\Omega} \operatorname{grad} u(t) \cdot \operatorname{grad} v \, dx = 0 \quad \forall v \in H^1(\Omega).$$

We perform spatial finite element Galerkin semi-discretization based on $S_1^0(M)$ in the spirit of the method of lines.

1. Which problem does the application of explicit Runge-Kutta timestepping face?
2. Show that implicit Euler timestepping is feasible.

(Q9.2.7.52.F) Show that Crank-Nicolson timestepping

$$\begin{aligned} \mathbf{M} \left\{ \frac{d}{dt} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) &= \vec{\varphi}(t) \\ \Downarrow \\ \mathbf{M} \frac{\vec{\mu}^{(j)} - \vec{\mu}^{(j-1)}}{\tau} &= -\frac{1}{2} \mathbf{A} (\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}) + \frac{1}{2} (\vec{\varphi}(t_j) + \vec{\varphi}(t_{j-1})), \end{aligned} \quad (9.2.7.5)$$

for the method-of-lines ODE for a standard parabolic evolution problem with s.p.d. bilinear forms $\mathbf{m}(\cdot, \cdot)$ and $\mathbf{a}(\cdot, \cdot)$ is *unconditionally stable*.

△

9.2.8 Fully Discrete Method of Lines: Convergence



Video tutorial for Section 9.2.8: Fully Discrete Method of Lines: Convergence: (35 minutes)
[Download link](#), [tablet notes](#)

Now we investigate the asymptotic *algebraic convergence* for fully discretized second-order linear parabolic evolution problems, when Lagrangian finite elements in space are used together with some Runge-Kutta single step method. Here we have two natural discretization parameters, namely the mesh width (\rightarrow Def. 3.2.1.4) of the finite element mesh, and the size τ of the (uniform) timestep.

For general considerations about asymptotic convergence and its meaning refer to § 3.3.5.9 and § 3.3.5.12.

We start with a question: Why should one prefer complicated implicit $L(\pi)$ -stable Runge-Kutta single step methods (\rightarrow Ex. 9.2.7.49) to the simple implicit Euler method?

Silly question! Because these “higher-order” methods deliver “better accuracy”!

However, we need some clearer idea of what is meant by this. To this end, we now study the dependence of (a norm of) the discretization error for a parabolic IBVP on the parameters of the spatial and temporal discretization.

EXPERIMENT 9.2.8.1 (Convergence of fully discrete timestepping in one spatial dimension)

- ◆ 1D parabolic evolution problem: $\frac{d}{dt}u - u'' = f(t, x)$ on $]0, 1[\times]0, 1[$
- ◆ exact solution $u(x, t) = (1 + t^2)e^{-\pi^2 t} \sin(\pi x)$, source term accordingly
- ◆ Linear finite element Galerkin discretization equidistant mesh, see Section 2.3, $V_{0,h} = S_{1,0}^0(\mathcal{M})$, ◆ piecewise linear spatial approximation of source term $f(x, t)$
- ◆ implicit Euler timestepping (\rightarrow Ex. 9.2.7.1) with uniform timestep $\tau > 0$

Monitored: error norm $\left(\tau \sum_{j=1}^M |u - u_h(\tau j)|_{H^1(\Omega)}^2 \right)^{\frac{1}{2}}$.

The norms $|u - u_h(\tau j)|_{H^1(\Omega)}$ were approximated by high order local quadrature rules, whose impact can be neglected.

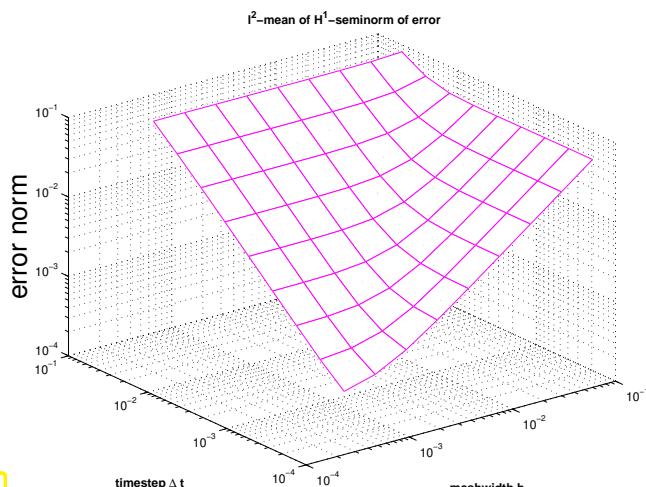


Fig. 379

Recall from Section 3.3.5, Thm. 3.1.3.7, Thm. 3.3.5.6:

energy norm of spatial finite element discretization error $\mathcal{O}(h_M)$ for $h_M \rightarrow 0$

Since the implicit Euler method is *first order consistent* we expect

temporal timestepping error $\mathcal{O}(\tau)$

(9.2.8.2) \geq conjecture: total error is **sum** of spatial and temporal discretization error.

From Fig. 380 we draw the compelling conclusion:

- for big mesh width h_M (spatial error dominates) further reduction of timestep size τ is useless,
- if timestep τ is large (temporal error dominates), refinement of the finite element space does not yield a reduction of the total error.

EXPERIMENT 9.2.8.3 (Higher order timestepping for 1D heat equation)

- ◆ same IBVP as in Exp. 9.2.8.1,
- ◆ spatial discretization on equidistant grid, *very small* meshwidth $h = 0.5 \cdot 10^{-4}$, $V_{0,h} = \mathcal{S}_{1,0}^0(M)$.

Various timestepping methods

(\geq different **orders of consistency**)

- implicit Euler timestepping (9.2.7.3), first order
- Crank-Nicolson-method (9.2.7.5), order 2
- SDIRK-2 timestepping (\rightarrow Ex. 9.2.7.49), order 2
- Gauss-Radau-Runge-Kutta collocation methods with s stages, order $2s - 1$

Note: all methods $L(\pi)$ -stable (\rightarrow Def. 9.2.7.46)
except for Crank-Nicolson-method.

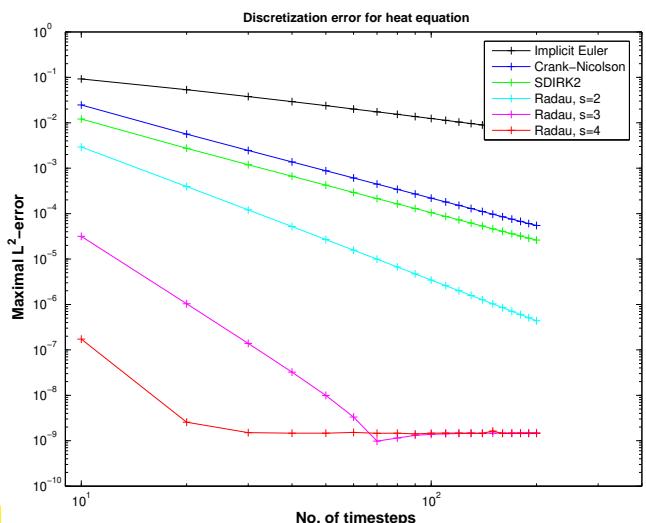


Fig. 380

Monitored: $\max_j \|u(t_j) - u_h^{(j)}\|_{L^2([0,1])}$ (evaluated by high order quadrature)

We observe that higher-order $L(\pi)$ -stable Runge-Kutta timestepping leads to a faster algebraic decay of the temporal discretization error, the rate matching the theoretical order of the methods. This can be

observed until we reach the spatial discretization error which is $\approx 10^{-9}$ in Fig. 381. □

§9.2.8.4 (Spatial and temporal error contributions) Theoretical results confirm the conjecture suggested from observation (9.2.8.2) in Exp. 9.2.8.1:

“Meta-theorem” 9.2.8.5. Convergence of solutions of fully discrete parabolic evolution problems

Assume that

- ◆ the solution of the parabolic IBVP (9.2.1.6)–(9.2.1.9) is “sufficiently smooth” (both in space and time),
- ◆ its spatial Galerkin finite element discretization relies on degree p Lagrangian finite elements (\rightarrow Section 2.6) on uniformly shape-regular families of meshes,
- ◆ timestepping is based on an $L(\pi)$ -stable single step method of order q with uniform timestep $\tau > 0$.

Then we can expect an asymptotic behavior of the total discretization error according to

$$\left(\tau \sum_{j=1}^M \left| u(\tau j) - u_h^{(j)} \right|_{H^1(\Omega)}^2 \right)^{\frac{1}{2}} \leq C(h_M^p + \tau^q) , \quad (9.2.8.6)$$

where $C > 0$ must not depend on h_M , τ .

This has been dubbed a “meta-theorem”, because quite a few technical assumptions on the exact solution and the methods have been omitted in its statement. Therefore it is not a mathematically rigorous statement of facts. More details in [KA03].

A message contained in (9.2.8.6):

total discretization error = spatial error + temporal error

§ 3.3.5.9 still applies: (9.2.8.6) does not give information about actual error, but only about the trend of the error, when discretization parameters h_M and τ are varied.

► Nevertheless, as in the case of the a priori error estimates of Section 3.3.5, we can draw conclusions about optimal refinement strategies in order to achieve prescribed *error reduction*.

As in Section 3.3.5 we make the **assumption** that the estimates (9.2.8.6) are sharp for all contributions to the total error and that the constants are the same (!)

$$\begin{aligned} \text{contribution of spatial error} &\approx Ch_M^p , h_M \doteq \text{mesh width } (\rightarrow \text{Def. 3.2.1.4}) , \\ \text{contribution of temporal error} &\approx C\tau^q , \tau \doteq \text{timestep size} . \end{aligned} \quad (9.2.8.7)$$

This suggests the following change of h_M , τ in order to achieve *error reduction* by a factor of $\rho > 1$:

$$\begin{aligned} \text{reduce mesh width by factor } &\rho^{1/p} && \xrightarrow{(9.2.8.7)} & \text{error reduction by } \rho > 1 . \\ \text{reduce timestep by factor } &\rho^{1/q} \end{aligned} \quad (9.2.8.8)$$



Refinement for fully discrete parabolic evolution problems

Guideline: spatial and temporal resolution have to be adjusted in tandem

Remark 9.2.8.10 (Potential inefficiency of conditionally stable single step methods)

Terminology: A timestepping scheme is labelled **conditionally stable**, if blow-up can be avoided by using sufficient small timesteps (timestep constraint). Examples: all explicit Runge-Kutta methods

Now we can answer the question, why a stability induced timestep constraint like (9.2.7.34), that is,

$$\tau \leq O(h_{\mathcal{M}}^{-2}) \quad (9.2.8.11)$$

can render a single step method grossly inefficient for integrating semi-discrete parabolic IBVPs.

(9.2.8.8) \Rightarrow in order to reduce the error by a fixed factor ρ one has to reduce both timestep and mesh-width by some other fixed factors (asymptotically). More concretely, for the timestep τ :

(9.2.8.8) \Rightarrow **accuracy** requires reduction of τ by a factor $\rho^{1/q}$

(9.2.8.11) \Rightarrow **stability** entails reduction of τ by a factor $(\rho^{1/p})^2 = \rho^{2/p}$.

$$\frac{1}{q} < \frac{2}{p} \Rightarrow \begin{aligned} &\text{stability enforces smaller timestep than required by accuracy} \\ &\Rightarrow \text{timestepping is } \textcolor{red}{inefficient!} \end{aligned}$$



When faced with conditional stability (9.2.8.11), for the sake of efficiency
use *high-order spatial discretization* combined with *low order timestepping*.

However, this may not be easy to achieve

- ◆ because high-order timestepping is much simpler than high-order spatial discretization,
- ◆ because limited spatial smoothness of exact solution (\rightarrow results of Section 3.4 apply!) may impose a limit on q in (9.2.8.6).

Concretely: 5th-order timestepping ($q = 5$) with **Ode45** $\xrightarrow{\frac{1}{q} = \frac{2}{p}}$ use degree-10 Lagrangian FEM!

Moreover, high-order convergence of spatial discretization error is conditional on sufficient smoothness of the solution $u(t)$ for all times, remember (3.3.5.21). \downarrow

Remark 9.2.8.12 (Guessing timestep constraint) Even if the timestep constraint $\tau < O(h_{\mathcal{M}}^{-2})$ does not thwart the efficiency of the full discretization (finite elements in space & Runge-Kutta timestepping), the actual stability threshold for τ may not be easy to guess, because precise estimates for the spectrum of the generalized eigenvalue problem (9.2.7.28) are difficult to obtain for general domains and meshes. \downarrow

EXPERIMENT 9.2.8.13 (Convergence for conditionally stable Runge-Kutta timestepping)

Parabolic IBVP of Exp. 9.2.8.1:

- ◆ $\frac{d}{dt}u - u'' = f(t, x)$ on $]0, 1[\times]0, 1[$
- ◆ exact solution $u(x, t) = (1 + t^2)e^{-\pi^2 t} \sin(\pi x)$, source term accordingly
- ◆ Linear finite element Galerkin discretization equidistant mesh, see Section 2.3, $V_{0,h} = S_{1,0}^0(\mathcal{M})$,
- ◆ piecewise linear spatial approximation of source term $f(x, t)$

- ◆ explicit Euler timestepping (9.2.7.2) with uniform timestep $\tau \sim h^2$ close to the stability limit.

Monitored: error norms $\left(\tau \sum_{j=1}^M \|u - u_h(\tau j)\|_{H^1([0,1])}^2 \right)^{\frac{1}{2}}$, $\left(\tau \sum_{j=1}^M \|u - u_h(\tau j)\|_{L^2([0,1])}^2 \right)^{\frac{1}{2}}$.

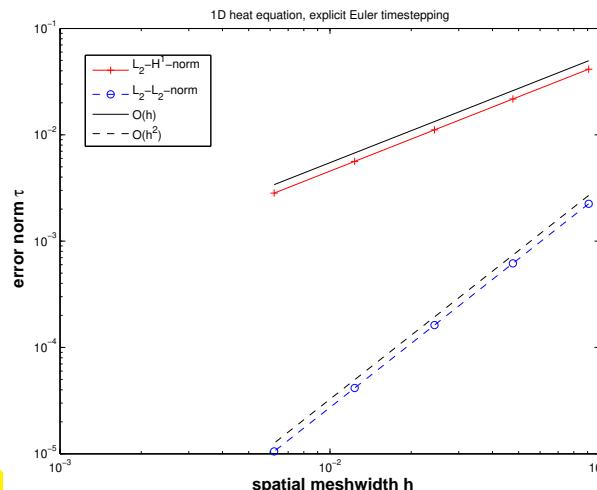


Fig. 381

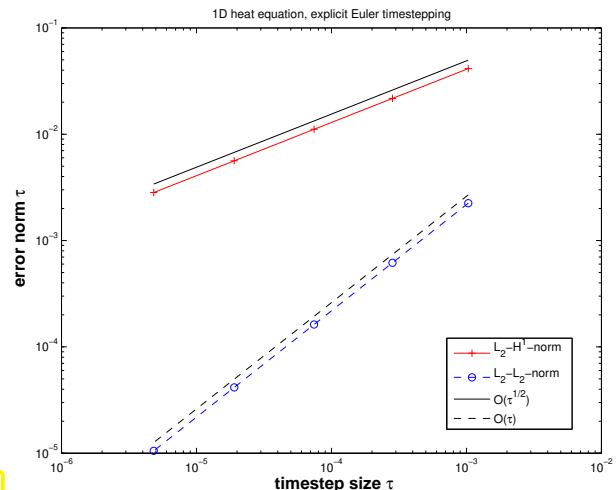


Fig. 382

In comparison with Exp. 9.2.8.1: degraded rate of convergence $O(\sqrt{\tau})$ for $L^2\text{-}H^1$ space-time norm, because conditional stability prevents us from employing sufficient refinement in space. □

Review question(s) 9.2.8.14 (Parabolic evolution problems)

(Q9.2.8.14.A) How will the assertion of Thm. 9.2.8.5 will probably have to be altered in case we face $u(t) \in H^m(\Omega)$, $m \geq 2$, but $u(t) \notin H^{m+1}(\Omega)$ for all times t .

Theorem 9.2.8.5. Convergence of solutions of fully discrete parabolic evolution problems

Assume that

- ◆ the solution of the parabolic IVP (9.2.1.6)–(9.2.1.9) is “sufficiently smooth” (both in space and time),
- ◆ its spatial Galerkin finite element discretization relies on degree p Lagrangian finite elements (→ Section 2.6) on uniformly shape-regular families of meshes,
- ◆ timestepping is based on an $L(\pi)$ -stable single step method of order q with uniform timestep $\tau > 0$.

Then we can expect an asymptotic behavior of the total discretization error according to

$$\left(\tau \sum_{j=1}^M \|u - u_h(\tau j)\|_{H^1(\Omega)}^2 \right)^{\frac{1}{2}} \leq C(h_M^p + \tau^q), \quad (9.2.8.6)$$

where $C > 0$ must not depend on h_M , τ .

(Q9.2.8.14.B) Comment on the statement

For the temporal discretization of the MOL-ODE arising from the spatial finite-element discretization of an IVP for the heat equation one *must* use implicit timestepping schemes.

△

9.3 Linear Wave Equations

This section is dedicated to a class of initial-boundary value problems (IBVP) that have the same structure as (abstract) parabolic IVP (→ § 9.2.2.7) except for the occurrence of *second derivatives in time*. This

will have profound consequences as regards properties of solutions and choice of timestepping schemes.

§9.3.0.1 (A conservative evolution) In the case of transient heat conduction Lemma 9.2.3.8 teaches that in the absence of time-dependent sources the rate of change of temperature will decay exponentially

Now we will encounter a class of evolution problems where temporal and spatial fluctuations will not be damped and will persist for good:

This will be the class of linear conservative **wave propagation** problems

As before these initial-boundary value problems (IBVP) will be posed on a space time cylinder $\tilde{\Omega} := \Omega \times]0, T[\subset \mathbb{R}^{d+1}$ (\rightarrow Fig. 370), where $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, is a bounded spatial domain as introduced in the context of elliptic boundary value problems, see § 1.2.1.14.

The unknown will be a function depending on space and time: $u = (x, t) : \tilde{\Omega} \mapsto \mathbb{R}$. □

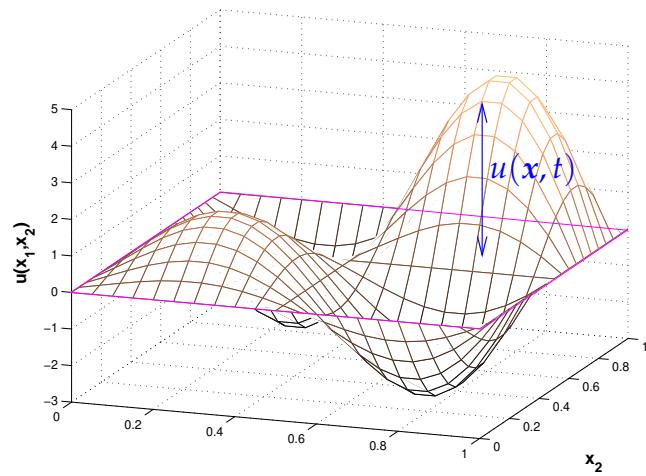
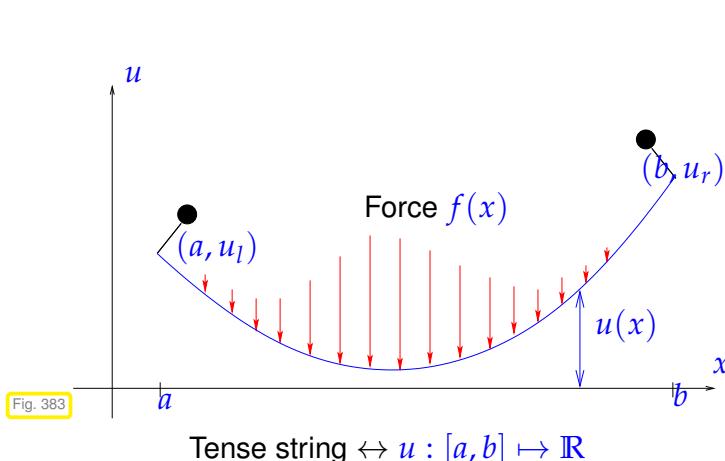
9.3.1 Models for Vibrating Membrane



Video tutorial for Section 9.3.1: Models for Vibrating Membranes: (24 minutes)
[Download link](#), [tablet notes](#)

§9.3.1.1 (Repetition: linear elastic string and membrane models) Recall the stationary simplified (linearized) models for taut string (1D) and membrane (2D):

- ◆ For the tense elastic string model (\rightarrow § 1.2.1.3) the shape of the string is described by a continuous displacement function $u : \Omega := [a, b] \mapsto \mathbb{R}$, $u \in H^1([a, b])$.
- ◆ For the taut membrane model (\rightarrow § 1.2.1.8), for which the shape of membrane is given by a continuous displacement function $u : \Omega \mapsto \mathbb{R}$, $u \in H^1(\Omega)$, defined on the bounded base domain $\Omega \subset \mathbb{R}^2$.



In Section 1.2.3 we introduced the general variational (weak) formulation of membrane models based on an energy minimization principle: with Dirichlet data (elevation of frame/pinning conditions) given by a continuous function $g \in C^0(\partial\Omega)$,

$$V := \{v \in H^1(\Omega) : v|_{\partial\Omega} = g\}$$

we seek

$$u \in V: \int_{\Omega} \sigma(x) \operatorname{grad} u \cdot \operatorname{grad} v \, dx = \int_{\Omega} f(x) v(x) \, dx, \quad \forall v \in H_0^1(\Omega), \quad (9.3.1.2)$$

where $f: \Omega \mapsto \mathbb{R} \hat{=} \text{density of vertical force}$,

$\sigma: \Omega \mapsto \mathbb{R} \hat{=} \text{uniformly positive stiffness coefficient (characteristic of material of the membrane).}$

↓

§9.3.1.3 (Transient membrane model with inertial forces) Now we switch to a *dynamic setting*: we allow variation of displacement with time, $u = u(x, t)$, the membrane is allowed to vibrate.

Recall (secondary school): **Newton's second law of motion** (law of inertia)

$$\begin{array}{c} F = m a \\ \text{force} = \text{mass} \cdot \text{acceleration} \end{array} \quad (9.3.1.4)$$

$$(9.3.1.5)$$

Apply this in a local version (stated for densities) to the membrane model and get the

$$\text{force density } f(x, t) = \rho(x) \cdot \frac{\partial^2 u}{\partial t^2}(x, t), \quad (9.3.1.6)$$

where

♦ $\rho: \Omega \mapsto \mathbb{R}^+ \hat{=} \text{uniformly positive mass density of membrane, } [\rho] = \text{kg m}^{-2}$,

♦ $\ddot{u} := \frac{\partial^2 u}{\partial t^2} \hat{=} \text{vertical acceleration (second temporal derivative of position).}$

Now, we assume that the force f in (1.4.2.2) is due to inertia forces only and express these using (9.3.1.6):

$$(1.4.2.2) \quad \blacktriangleright \int_{\Omega} \sigma(x) \operatorname{grad} u(x, t) \cdot \operatorname{grad} v(x) \, dx = - \int_{\Omega} \rho(x) \frac{\partial^2 u}{\partial t^2}(x, t) v(x) \, dx \quad \forall v \in H_0^1(\Omega).$$

Why the “−”-sign? Because, here the inertia force enters as a *reaction* force.

What we have arrived at is a homogeneous **linear wave equation** in (spatial) variational form and with (possibly inhomogeneous) Dirichlet boundary conditions:

$$u(t) \in V(t): \int_{\Omega} \rho(x) \cdot \frac{\partial^2 u}{\partial t^2}(x, t) v(x) \, dx + \int_{\Omega} \sigma(x) \operatorname{grad} u(x, t) \cdot \operatorname{grad} v(x) \, dx = 0 \quad \forall v \in H_0^1(\Omega) \quad (9.3.1.7)$$

$$u(t) \in V(t): m(\ddot{u}, v) + a(u, v) = 0 \quad \forall v \in V_0. \quad (9.3.1.8)$$

where

$$V(t) := \{v :]0, T[\mapsto H^1(\Omega) : v(x, t) = g(x, t) \text{ for } x \in \partial\Omega, 0 < t < T\}$$

(with continuous time-dependent Dirichlet data $g: \partial\Omega \times]0, T[\mapsto \mathbb{R}$.)

The bilinear forms a and m (→ Def. 1.2.3.27) in (9.3.1.8) are the same as those in (9.2.2.8), § 9.2.2.7

(except for the notation for the coefficient σ):

$$\begin{aligned} \mathbf{m}(u, v) &= \int_{\Omega} \rho(\mathbf{x}) u(\mathbf{x}) v(\mathbf{x}) \, d\mathbf{x}, \quad u, v \in L^2(\Omega), \\ \mathbf{a}(u, v) &= \int_{\Omega} \sigma(\mathbf{x}) \mathbf{grad} u(\mathbf{x}) \cdot \mathbf{grad} v(\mathbf{x}) \, d\mathbf{x}, \quad u, v \in H^1(\Omega). \end{aligned}$$

In particular, both \mathbf{a} and \mathbf{m} are *symmetric and positive definite* (\rightarrow Def. 1.2.3.27). Thus they induce energy norms $\|\cdot\|_a$ and $\|\cdot\|_m$ (\rightarrow Def. 1.2.3.35):

$$\|v\|_a^2 := \mathbf{a}(v, v), \quad \|v\|_m^2 := \mathbf{m}(v, v).$$

↓

§9.3.1.9 (Wave equation) Undo integration by parts by reverse application of Green's first formula

Theorem 1.5.2.7. Green's first formula

For all vector fields $\mathbf{j} \in (C_{pw}^1(\bar{\Omega}))^d$ and functions $v \in C_{pw}^1(\bar{\Omega})$ holds

$$\int_{\Omega} \mathbf{j} \cdot \mathbf{grad} v \, d\mathbf{x} = - \int_{\Omega} \operatorname{div} \mathbf{j} v \, d\mathbf{x} + \int_{\partial\Omega} \mathbf{j} \cdot \mathbf{n} v \, dS. \quad (1.5.2.8)$$

we obtain from (9.3.1.7)

$$\int_{\Omega} \left\{ \rho(\mathbf{x}) \frac{\partial^2 u}{\partial t^2}(\mathbf{x}, t) - \operatorname{div}_{\mathbf{x}}(\sigma(\mathbf{x})(\mathbf{grad}_{\mathbf{x}} u)(\mathbf{x}, t)) \right\} v(\mathbf{x}) \, d\mathbf{x} = 0 \quad \forall v \in H_0^1(\Omega). \quad (9.3.1.10)$$

Here it is indicated that the differential operators \mathbf{grad} and div act on the spatial independent variable \mathbf{x} only. As in the case of the heat equation (\rightarrow § 9.2.1.2) this will tacitly be assumed below.

Now appeal to the fundamental lemma of calculus of variations in higher dimensions.

Lemma 1.5.3.4. Fundamental lemma of calculus of variations in higher dimensions

If $f \in L^2(\Omega)$ satisfies

$$\int_{\Omega} f(\mathbf{x}) v(\mathbf{x}) \, d\mathbf{x} = 0 \quad \forall v \in C_0^\infty(\Omega),$$

then $f \equiv 0$ can be concluded.

This gives a PDE on the space-time cylinder $\tilde{\Omega} := \Omega \times]0, T[$, $T > 0 \hat{=} \text{"final time"}$, see § 9.1.0.3.

$$(9.3.1.10) \quad \xrightarrow{\text{Lemma 1.5.3.4}} \quad \rho(\mathbf{x}) \frac{\partial^2 u}{\partial t^2} - \operatorname{div}(\sigma(\mathbf{x}) \mathbf{grad} u) = 0 \quad \text{in } \tilde{\Omega}. \quad (9.3.1.11)$$

(9.3.1.11) is called a (homogeneous, linear) **wave equation**. A general wave equation is obtained, when an additional exciting vertical force density $f = f(\mathbf{x}, t)$ comes into play:

$$\rho(\mathbf{x}) \frac{\partial^2 u}{\partial t^2} - \operatorname{div}(\sigma(\mathbf{x}) \mathbf{grad} u) = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega}. \quad (9.3.1.12)$$

↓

§9.3.1.13 (Initial and boundary conditions) The wave equations (9.3.1.11), (9.3.1.12) have to be supplemented by

- **spatial** (Dirichlet) **boundary conditions**: $v(x, t) = g(x, t)$ for $x \in \partial\Omega, 0 < t < T$,
- **two initial conditions**

$$u(x, 0) = u_0(x) , \quad \frac{\partial u}{\partial t}(x, 0) = v_0 \quad \text{for } x \in \Omega ,$$

with initial data $u_0, v_0 \in H^1(\Omega)$, satisfying the compatibility conditions $u_0(x) = g(x, 0)$ for $x \in \partial\Omega$.

(9.3.1.11) & boundary conditions & initial conditions = (linear) **hyperbolic evolution problem**

Excuse me, why do we need **two** initial conditions in contrast to the heat equation?

Remember that

- (9.3.1.11) is a *second-order equation* also in time (whereas the heat equation is merely first-order),
- and that for second order ODEs $\ddot{\mathbf{u}} = \mathbf{f}(\mathbf{u})$, $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$, we need **two** initial conditions

$$\mathbf{u}(0) = \mathbf{u}_0 \quad \text{and} \quad \dot{\mathbf{u}}(0) = \mathbf{v}_0 , \quad (9.3.1.14)$$

in order to get a well-posed initial value problem, see Rem. 6.1.3.5.

The physical meaning of the initial conditions (9.3.1.14) in the case of the membrane model is

- $u_0 \hat{=} \text{initial displacement of membrane}$, $u_0 \in H^1(\Omega)$ “continuous”,
- $v_0 \hat{=} \text{initial vertical velocity of membrane}$.

Remark 9.3.1.15 (Boundary conditions for wave equation) The message of § 9.2.1.10 also applies to the wave equation (9.3.1.11):

On $\partial\Omega \times]0, T[$ we can impose any of the **spatial boundary conditions** discussed in Section 1.7:

- Dirichlet boundary conditions $u(x, t) = g(x, t)$ (membrane attached to frame),
- Neumann boundary conditions $\mathbf{j}(x, t) \cdot \mathbf{n} = 0$ (free boundary, Ex. 1.5.3.11)
- radiation boundary conditions $\mathbf{j}(x, t) \cdot \mathbf{n} = \Psi(u(x, t))$,

and any combination of these as discussed in Ex. 1.7.0.10, yet, *only one* of them at any part of $\partial\Omega \times]0, T[$, see Rem. 1.7.0.9.

In the above formulas recall from § 1.6.0.1 the definition of the flux $\mathbf{j} = -\sigma(\mathbf{x}) \mathbf{grad} u$ for linear diffusion models.

§9.3.1.16 (Wave equation as first order system in time) We recall a fundamental transformation in the theory of ordinary differential equations from Rem. 6.1.3.5: any higher-order ODE can be converted into first-order ODEs by introducing derivatives as additional solution components. In the concrete case of a second-order ODE on state space \mathcal{V} we obtain

$$\ddot{\mathbf{u}}(t) = \mathbf{g}(t, \mathbf{u}) \quad \xrightarrow{\mathbf{v} := \dot{\mathbf{u}}} \quad \begin{bmatrix} \dot{\mathbf{u}}(t) \\ \dot{\mathbf{v}}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{g}(t, \mathbf{u}(t)) \end{bmatrix} \Leftrightarrow \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} . \quad (9.3.1.17)$$

This approach also works for the second-order (in time) wave equation (9.3.1.11):

We introduce an additional unknown:

$$\text{velocity} \quad v(x, t) = \frac{\partial u}{\partial t}(x, t)$$

$$\rho(\mathbf{x}) \frac{\partial^2 u}{\partial t^2} - \operatorname{div}(\sigma(\mathbf{x}) \mathbf{grad} u) = 0 \quad \Rightarrow$$

$$\begin{cases} \dot{u} = v, \\ \rho(\mathbf{x}) \dot{v} = \operatorname{div}(\sigma(\mathbf{x}) \mathbf{grad} u) \end{cases} \quad \text{in } \tilde{\Omega} \quad (9.3.1.18)$$

with initial conditions

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad , \quad v(\mathbf{x}, 0) = v_0(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega . \quad (9.3.1.19)$$

□

Review question(s) 9.3.1.20 (Models for vibrating membranes)

(Q9.3.1.20.A) The movement of a taut membrane also subject to friction can be modeled by an additional vertical force density proportional to the local velocity:

$$f(\mathbf{x}) = -\rho(\mathbf{x}) \frac{\partial^2 u}{\partial t^2} - \eta(\mathbf{x}) \frac{\partial u}{\partial t} ,$$

with uniformly positive friction coefficient $\eta = \eta(\mathbf{x})$. Give the spatial variational formulation for a membrane model with friction (membrane attached to a flat frame everywhere on its edge).

Hint. The dynamic membrane model with a generic force density $f = f(\mathbf{x})$ is:

$$u \in V: \int_{\Omega} \sigma(\mathbf{x}) \operatorname{grad} u \cdot \operatorname{grad} v \, d\mathbf{x} = \int_{\Omega} f(\mathbf{x}) v(\mathbf{x}) \, d\mathbf{x} , \quad \forall v \in H_0^1(\Omega) . \quad (9.3.1.2)$$

(Q9.3.1.20.B) Which initial-boundary value problem describes the frictionless movement of a membrane clamped to a flat and level frame on three sides of a square, but free on the fourth side under the influence of gravity?

(Q9.3.1.20.C) The propagation of sound in space can be described by the following first-order system of linear partial differential equations:

$$\frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\rho_0} \operatorname{grad} p = \mathbf{0} , \quad \frac{\partial \rho}{\partial t} + \rho_0 \operatorname{div} \mathbf{v} = 0 , \quad \frac{\partial \rho}{\partial t} - \frac{1}{c^2} \frac{\partial p}{\partial t} = 0 .$$

Here $\mathbf{v} = \mathbf{v}(\mathbf{x}, t)$ is the velocity field ($[\mathbf{v}] = \text{ms}^{-1}$), $p = p(\mathbf{x}, t)$ the pressure field ($[p] = \text{Nm}^{-2}$), $\rho_0 = \rho_0(\mathbf{x})$ a uniformly positive density ($[\rho_0] = \text{krm}^{-3}$), and $c = c(\mathbf{x})$ the local speed of sound ($[c] = \text{ms}^{-1}$).

1. Derive a second-order PDE governing the evolution of the pressure field.
2. At hard walls we have $\mathbf{v} \cdot \mathbf{n} = 0$, where \mathbf{n} is a unit vector normal to the wall. Which spatial boundary conditions does this entail for the second-order PDE?

(Q9.3.1.20.D) For symmetric positive definite matrices $\mathbf{M}, \mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,n}$ convert the second-order ODE for $\vec{\mu} : [0, T] \rightarrow \mathbb{R}^n$

$$\mathbf{M} \frac{\partial^2 \vec{\mu}}{\partial t^2} + \mathbf{B} \frac{\partial \vec{\mu}}{\partial t} + \mathbf{A} \vec{\mu} = \mathbf{0}$$

into an equivalent first-order ODE in the standard form $\dot{\mathbf{u}} = \mathbf{f}(\mathbf{u})$.

△

9.3.2 Wave Propagation



Video tutorial for Section 9.3.2: Wave Propagation: (33 minutes) [Download link](#), [tablet notes](#)

Now we study properties of solutions of initial-boundary value problems (IBVPs) for the wave equation (9.3.1.11):

$$\rho(\mathbf{x}) \frac{\partial^2 u}{\partial t^2} - \operatorname{div}(\sigma(\mathbf{x}) \operatorname{grad} u) = 0 \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[\quad (9.3.1.11)$$

+ spatial boundary conditions & in initial conditions $\begin{cases} u(\mathbf{x}, 0) = u_0(\mathbf{x}), \\ \frac{\partial u}{\partial t}(\mathbf{x}, 0) = v_0(\mathbf{x}), \end{cases} \quad \mathbf{x} \in \Omega.$

§9.3.2.1 (Cauchy problem) We first look at a particularly simple setting: We consider the constant-coefficient wave equation ($\rho \equiv 1$) for $d = 1$ on the whole real line, $\Omega = \mathbb{R}$, the so-called **Cauchy problem** for the wave equation:

$$c > 0: \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0, \quad u(x, 0) = u_0(x), \quad \frac{\partial u}{\partial t}(x, 0) = v_0(x), \quad x \in \mathbb{R}. \quad (9.3.2.2)$$

For this we perform a change of variables into **characteristic coordinates**:

$$\xi = x + ct, \quad \tau = x - ct: \quad \tilde{u}(\xi, \tau) := u\left(\frac{\xi + \tau}{2}, \frac{\xi - \tau}{2c}\right). \quad (9.3.2.3)$$

Applying the chain rule we immediately see

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \quad \Rightarrow \quad \frac{\partial^2 \tilde{u}}{\partial \xi \partial \tau} = 0 \quad \Rightarrow \quad \tilde{u}(\xi, \tau) = F(\xi) + G(\tau),$$

for **any** $F, G \in C^2(\mathbb{R})$! This means that we can write the solution of the Cauchy problem (9.3.2.2) as

$$u(x, t) = F(x + ct) + G(x - ct),$$

that is, as a sum of a function F “traveling to the left” and a function G “traveling to the right”. Obviously, the initial conditions from (9.3.2.2)

$$u(x, 0) = u_0(x), \quad \frac{\partial u}{\partial t}(x, 0) = v_0(x), \quad x \in \mathbb{R},$$

already fix those functions F and G , because for all $x \in \mathbb{R}$

$$\begin{aligned} u(x, 0) &= F(x) + G(x) = u_0(x), \\ \frac{\partial u}{\partial t}(x, 0) &= c \frac{\partial \tilde{u}}{\partial \xi}(x, 0) - c \frac{\partial \tilde{u}}{\partial \tau}(x, 0) = cF'(x) - cG'(x) = v_0(x). \end{aligned}$$

We assume that v_0 has compact support and integrate the second equation from $-\infty$ to x , which yields

$$F(x) + G(x) = u_0(x), \quad F(x) - G(x) = \frac{1}{c} \int_{-\infty}^x v_0(s) ds.$$

Solving for F and G is straightforward now and we end up with

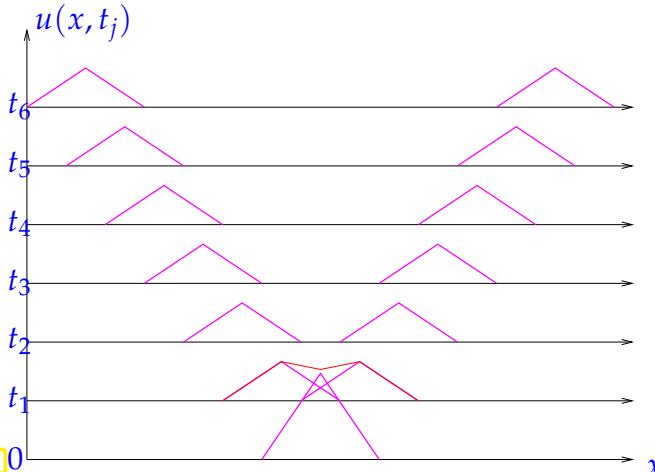
$$u(x, t) = \frac{1}{2}(u_0(x + ct) + u_0(x - ct)) + \frac{1}{2c} \int_{x-ct}^{x+ct} v_0(s) ds. \quad (9.3.2.4)$$

The formula (9.3.2.4) is known as **d'Alembert solution** of the Cauchy problem (9.3.2.2). □

Remark 9.3.2.5 (Non-smooth d'Alembert solutions) Obviously, the formula (9.3.2.4) even makes sense for discontinuous functions u_0 and v_0 , which will render $u = u(x, t)$ discontinuous in space and time.

This will not be compatible with the notion of solutions of the wave equation adopted in this section, but will comply with a weaker solution concept introduced in the context of conservation laws in Section 11.2.3, Def. 11.2.3.4. \square

§9.3.2.6 (Finite speed of propagation) We point out a simple consequence of the solution formula (9.3.2.4) for the Cauchy problem for the wave equation with constant coefficients:



$v_0 = 0 \Rightarrow$ initial data u_0 travel with speed c in opposite directions

Finite speed of propagation is a typical feature of solutions of wave equations

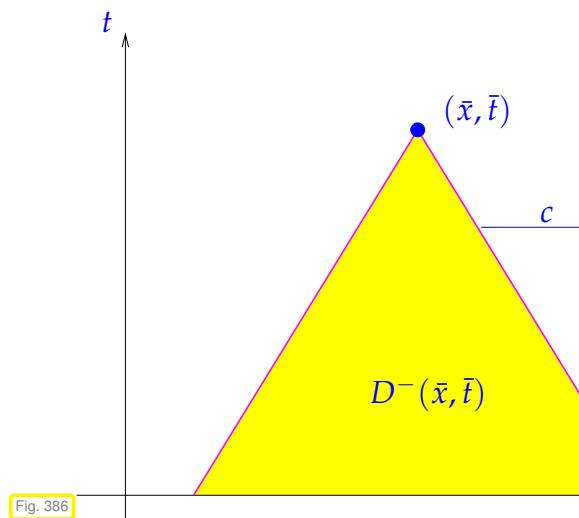
Note: (9.3.2.4) meaningful even for discontinuous u_0, v_0 , cf. Rem. 9.3.2.5!
→ “generalized solutions”!

finite speed of propagation \Rightarrow “point value” $u(\bar{x}, \bar{t})$, $(\bar{x}, \bar{t}) \in \tilde{\Omega}$, may not depend on initial values outside proper subdomain of Ω ! \square

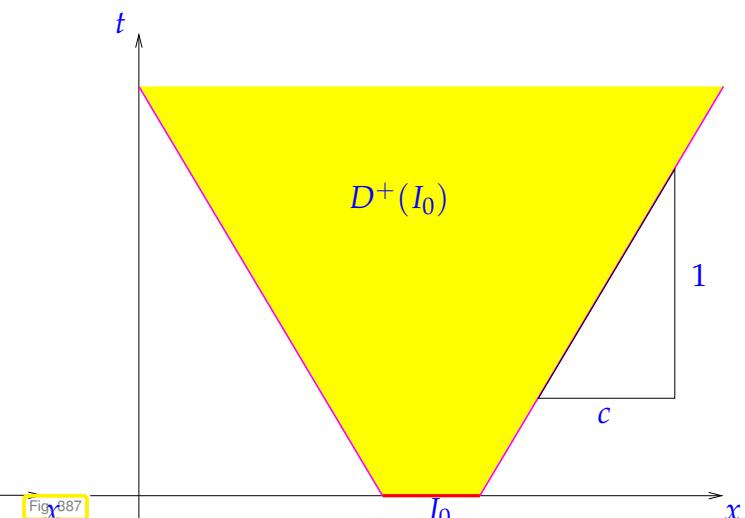
EXAMPLE 9.3.2.7 (Domain of dependence/influence for 1D wave equation, constant coefficient case) Consider $d = 1$ and the initial-boundary value problem (9.3.2.2) for wave equation:

$$c > 0: \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0, \quad u(x, 0) = u_0(x), \quad \frac{\partial u}{\partial t}(x, 0) = v_0(x), \quad x \in \mathbb{R}. \quad (9.3.2.2)$$

Intuitively we conclude from D'Alembert's formula (9.3.2.4) the following maximal domains of dependence and influence:



domain of dependence of $(\bar{x}, \bar{t}) \in \tilde{\Omega}$



domain of influence of $I_0 \subset \mathbb{R}$

Domain of dependence: the value of the solution in (\bar{x}, \bar{t}) (•) will depend only on data in the yellow triangle in Fig. 387.

Domain of influence: initial data in I_0 will be relevant for the solution only in the yellow triangle in Fig. 388. \square

The rigorous statement in any dimension is made in the following theorem.

Theorem 9.3.2.8. Domain of dependence for isotropic wave equation → [Eva98, 2.5, Thm. 6]

Let $u : \tilde{\Omega} \mapsto \mathbb{R}$ be a (classical) solution of $\frac{\partial^2 u}{\partial t^2} - c\Delta u = 0$. Then

$$\left(|x - x_0| \geq R \Rightarrow \begin{array}{l} u(x, 0) = 0 \\ \frac{\partial u}{\partial t}(x, 0) = 0 \end{array} \right) \Rightarrow u(x, t) = 0 \text{ , if } |x - x_0| \geq R + ct .$$

It tells us that, if a solution of $\frac{\partial^2 u}{\partial t^2} - c\Delta u = 0$ vanishes outside a ball of radius $R < 0$ around $x_0 \in \Omega$ at time $t \in [0, T]$, then $u(t + \tau)$ vanishes outside a ball with radius $R + c\tau$ around x_0 , provided that this larger ball is still contained in Ω . The maximal domain of influence is a **cone** in the space-time cylinder $\tilde{\Omega}$.

§9.3.2.9 (Reflective boundary conditions) We study the d'Alembert solution (9.3.2.4) of the 1D Cauchy problem (9.3.2.2) for the linear wave equation for special initial data

$$u_0(x) = w(d + x) - w(d - x) , \quad v_0(x) = 0 , \quad x \in \mathbb{R} , \quad (9.3.2.10)$$

where $w \in C^0(\mathbb{R})$ and $d > 0$. This case (9.3.2.4) yields

$$u(x, t) = \frac{1}{2}(w(d + x + ct) - w(d - x - ct) + w(d + x - ct) - w(d - x + ct)) , \quad x \in \mathbb{R} , \quad t \geq 0 .$$

For compactly supported w , this solution has the following structure:

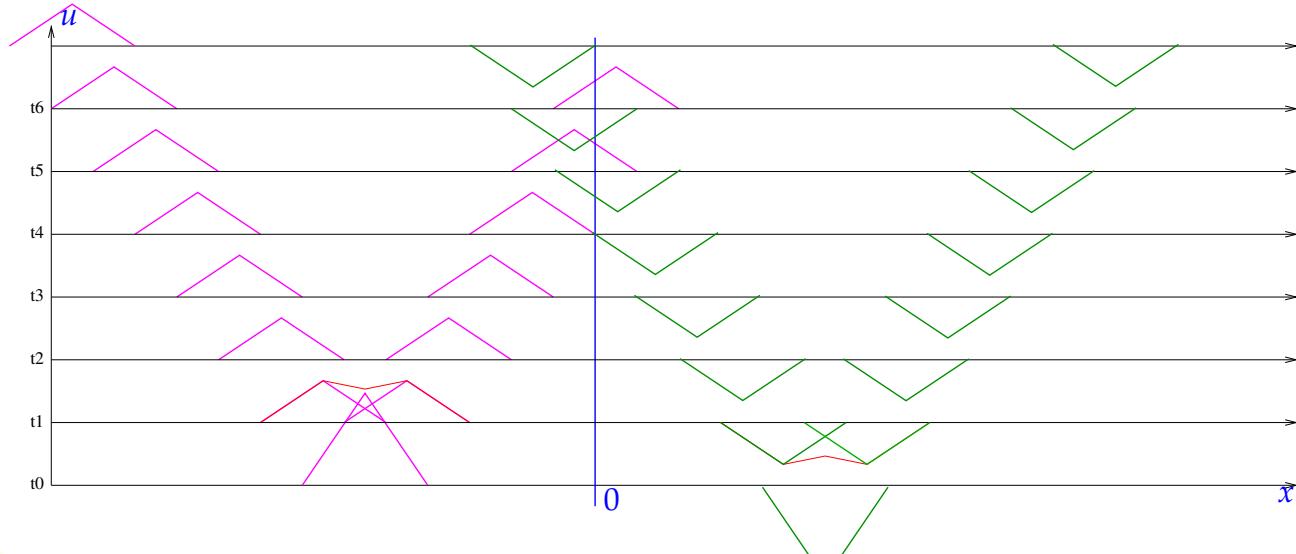


Fig. 388

► $u(x, t)$ satisfies homogeneous Dirichlet boundary conditions at $x = 0$ and, thus solves the following half-space IVP for the 1D wave equation:

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} &= 0 \quad \text{on } \mathbb{R}_- \times [0, T] , \\ u(x, 0) &= u_0(x) , \quad \frac{\partial u}{\partial t}(x, 0) = 0 , \quad x \leq 0 , \\ u(0, t) &= 0 \quad \text{for } t \geq 0 \quad (\text{Dirichlet b.c.}) . \end{aligned} \quad (9.3.2.11)$$

Reflection at boundaries

Homogeneous Dirichlet or Neumann boundary conditions on (parts of) $\partial\Omega$ model the **reflection** of waves (at those parts of $\partial\Omega$).

§9.3.2.13 (Wave propagation: conservation of energy) The solution formula (9.3.2.4) clearly indicates that in 1D and in the absence of boundary conditions the solution of the wave equation will persist undamped for all times.

This absence of damping corresponds to a **conservation of total energy**, which is a distinguishing feature of conservative wave propagation phenomena.

Now, we examine this for the following model problem, already cast in spatial variational formulation:

$$u : [0, T] \rightarrow H_0^1(\Omega): \int_{\Omega} \rho(x) \cdot \frac{\partial^2 u}{\partial t^2} v \, dx + \int_{\Omega} \sigma(x) \operatorname{grad} u \cdot \operatorname{grad} v \, dx = 0 \quad \forall v \in H_0^1(\Omega) \quad (9.3.2.14)$$

$$u \in V_0: m(\ddot{u}, v) + a(u, v) = 0 \quad \forall v \in V_0, \quad (9.3.2.15)$$

where the bilinear forms $m(\cdot, \cdot)$ and $a(\cdot, \cdot)$ are clear from (9.3.2.14), see also (9.3.1.8).

Here we do not include the case of non-homogeneous spatial Dirichlet boundary conditions through an affine trial space. This can always be taken into account by offset functions, see the remark after (9.2.1.9).

Theorem 9.3.2.16. Energy conservation in wave propagation

If $u : \tilde{\Omega} \mapsto \mathbb{R}$ solves (9.3.2.15), then we have **conservation of total energy** in the sense that

$$t \mapsto \frac{1}{2}m\left(\frac{\partial u}{\partial t}, \frac{\partial u}{\partial t}\right) + \frac{1}{2}a(u, u) \equiv \text{const.}$$

kinetic energy *elastic (potential) energy, see (1.2.1.19)*

Proof. The “formal proof” boils down to a straightforward application of the product rule (\rightarrow Rem. 9.2.3.7) together with the symmetry of the bilinear forms m and a .

Introduce the **total energy** and apply the product rule from Rem. 9.2.3.7

$$E(t) := \frac{1}{2}m\left(\frac{\partial u}{\partial t}, \frac{\partial u}{\partial t}\right) + \frac{1}{2}a(u, u).$$

► $\frac{dE}{dt}(t) = m(\ddot{u}, \dot{u}) + a(\dot{u}, u) = 0 \quad \text{for solution } u \text{ of (9.3.2.15)},$

because this is what we conclude from (9.3.2.15) for the special test function $v(x) = \dot{u}(x, t)$ for any $t \in]0, T[$. □

Of course, conservation of total energy rules out the decaying of the waves as $t \rightarrow \infty$: wave phenomena will continue unabated for all times. □

Review question(s) 9.3.2.17 (Wave propagation)

(Q9.3.2.17.A) We consider the Cauchy problem for the constant-coefficient linear wave equation in \mathbb{R}^d :

$$\frac{\partial^2 u}{\partial t^2} - \Delta u = 0 \quad \text{in } \mathbb{R}^d \times \mathbb{R}^+ . \quad (9.3.2.18)$$

For $\mathbf{k} \in \mathbb{R}^d$, $\|\mathbf{k}\| = 1$, a solution of (9.3.2.18) of the form

$$u(\mathbf{x}, t) = \operatorname{Re} \exp(i(\mathbf{k} \cdot \mathbf{x} - \omega t)) , \quad \omega \in \mathbb{R} , \quad (9.3.2.19)$$

is called a **plane-wave solution**. Find ω as a function of \mathbf{k} .

(Q9.3.2.17.B) Consider the Cauchy problem for the 1D wave equation

$$c > 0: \quad \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 , \quad u(x, 0) = u_0(x) , \quad \frac{\partial u}{\partial t}(x, 0) = v_0(x) , \quad x \in \mathbb{R} , \quad (9.3.2.2)$$

and recall the d'Alembert solution

$$u(x, t) = \frac{1}{2}(u_0(x + ct) + u_0(x - ct)) + \frac{1}{2c} \int_{x-ct}^{x+ct} v_0(s) \, ds . \quad (9.3.2.4)$$

Which relationship must be satisfied for the initial data u_0 and v_0 such that you obtain a solution that propagates only to the right?

(Q9.3.2.17.C) We study the following initial-boundary value problem for the 1D linear constant-coefficeint wave equation

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} &= 0 \quad \text{in }]0, 1[\times \mathbb{R}^+ , \\ u(0, t) &= u(1, t) = 0 \quad \forall t > 0 , \\ u(x, 0) &= u_0(x) , \quad \frac{\partial u}{\partial t}(x, 0) = v_0(x) , \quad 0 < x < 1 . \end{aligned} \quad (9.3.2.20)$$

1. In the $x - t$ -plane sketch the precise domain of influence of the interval $[\frac{1}{4}, \frac{3}{4}]$.

2. What is the domain of dependence for the point $\left[\begin{smallmatrix} 0.5 \\ 1 \end{smallmatrix}\right]$?

Hint. Remember that homogeneous Dirichlet boundary conditions for the 1D wave equation model a reflection of the wave at the boundary.

(Q9.3.2.17.D) Consider the abstract variational wave equation

$$u = u(t) \in V_0: \quad m(\ddot{u}, v) + b(\dot{u}, v) + a(u, v) = 0 \quad \forall v \in V_0 , \quad (9.3.2.21)$$

where V_0 is a vector space and m , a are symmetric positive definite bilinear forms on V_0 . Which properties of the bilinear form b ensure an **exponential decay** of the **energy**

$$\mathcal{E}(t) := \frac{1}{2}m(\dot{u}(t), \dot{u}(t)) + \frac{1}{2}a(u(t), u(t))$$

when $t \mapsto u(t)$ solves (9.3.2.21).

(Q9.3.2.17.E) Based on the d'Alembert solution (9.3.2.4) and guided by the “refelection construction” in § 9.3.2.9, construct a solution of the 1D half-space IBVP:

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} &= 0 \quad \text{on } \mathbb{R}_- \times [0, T] , \\ u(x, 0) &= u_0(x) , \quad \frac{\partial u}{\partial t}(x, 0) = 0 , \quad x \leq 0 , \\ \frac{\partial u}{\partial x}(0, t) &= 0 \quad \text{for } t \geq 0 \quad (\text{Homogeneous Neumann b.c.}) . \end{aligned}$$

△

9.3.3 Method of Lines for Wave Propagation



Video tutorial for Section 9.3.3: Method of Lines for Wave Propagation: (13 minutes)
[Download link](#), [tablet notes](#)

As we have seen in (9.3.2.14)/(9.3.1.8) the wave equation on the space-time cylinder $\tilde{\Omega} : [0, T] \times \Omega$ can be expressed through a spatial variational formulation. With $V(t) \subset H^1(\Omega)$ a Sobolev space adapted to the boundary conditions, we found

$$u \in V(t) : \int_{\Omega} \rho(x) \cdot \frac{\partial^2 u}{\partial t^2}(x, t) v(x) dx + \int_{\Omega} \sigma(x) \mathbf{grad} u(x, t) \cdot \mathbf{grad} v(x) dx = 0 \quad \forall v \in H_0^1(\Omega) \quad (9.3.1.7)$$

↑

$$u \in V(t) : m(\ddot{u}, v) + a(u, v) = 0 \quad \forall v \in V_0 . \quad (9.3.1.8)$$

As explained in § 9.3.1.13, this has to be supplemented with initial conditions $u(0) = u_0$ and $\dot{u}(0) = v_0$. The bilinear forms m and a are continuous, symmetric, and positive (semi-)definite.

Except for the second temporal derivative the structure is the same as for the abstract linear parabolic evolution equation

$$t \in]0, T[\mapsto u(t) \in V_0 : \begin{cases} m(\dot{u}(t), v) + a(u(t), v) = \ell(t)(v) & \forall v \in V_0 , \\ u(0) = u_0 \in V_0 . \end{cases} \quad (9.2.2.8)$$

Therefore, it is not surprising that spatial Galerkin discretization can be applied to (9.3.1.8) in the very same method-of-lines fashion as it was applied to (9.2.2.8) in Section 9.2.4. We elaborate the details now.

§9.3.3.1 (Spatial Galerkin semi-discretization) The method of lines approach to the wave equation (9.3.2.14), (9.3.2.15) is exactly the same as for the heat equation, see Section 9.2.4.

Idea: Apply **Galerkin discretization** (→ Section 2.2) in space to abstract linear hyperbolic variational problem (9.3.1.8).

$$t \in]0, T[\mapsto u(t) \in V_0 : \begin{cases} m\left(\frac{d^2 u}{dt^2}(t), v\right) + a(u(t), v) = \ell(t)(v) & \forall v \in V_0 , \\ u(0) = u_0 \in V_0 , \quad \frac{du}{dt}(0) = v_0 \in V_0 . \end{cases} \quad (9.3.3.2)$$

1st step: replace V_0 with a finite dimensional subspace $V_{0,h}$, $N := \dim V_{0,h} < \infty$

► Spatially discrete linear wave equation/hyperbolic evolution problem

$$t \in]0, T[\mapsto u(t) \in V_{0,h} : \begin{cases} m\left(\frac{d^2 u_h}{dt^2}(t), v_h\right) + a(u_h(t), v_h) = \ell(t)(v_h) & \forall v_h \in V_{0,h} , \\ u_h(0) = \text{projection/interpolant of } u_0 \text{ in } V_{0,h} , \\ \frac{du_h}{dt}(0) = \text{projection/interpolant of } v_0 \text{ in } V_{0,h} . \end{cases} \quad (9.3.3.3)$$

2nd step: introduce (ordered) basis $\mathfrak{B}_h := \{b_h^1, \dots, b_h^N\}$ of trial/test space $V_{0,h}$

$$(9.3.3.3) \Rightarrow \begin{cases} \mathbf{M} \left\{ \frac{d^2}{dt^2} \vec{u}(t) \right\} + \mathbf{A} \vec{u}(t) = \vec{\varphi}(t) & \text{for } 0 < t < T , \\ \vec{u}(0) = \vec{\mu}_0 , \quad \frac{d\vec{u}}{dt}(0) = \vec{v}_0 . \end{cases} \quad (9.3.3.4)$$

- ▷ s.p.d. stiffness matrix $\mathbf{A} \in \mathbb{R}^{N,N}$, $(\mathbf{A})_{ij} := a(b_h^j, b_h^i)$ (independent of time),
- ▷ s.p.d. mass matrix $\mathbf{M} \in \mathbb{R}^{N,N}$, $(\mathbf{M})_{ij} := m(b_h^j, b_h^i)$ (independent of time),
- ▷ source (load) vector $\vec{\phi}(t) \in \mathbb{R}^N$, $(\vec{\phi}(t))_i := \ell(t)(b_h^i)$ (time-dependent),
- ▷ $\vec{\mu}_0 \hat{=} \text{coefficient vector of a projection of } u_0 \text{ onto } V_{0,h}$.
- ▷ $\vec{\nu}_0 \hat{=} \text{coefficient vector of a projection of } v_0 \text{ onto } V_{0,h}$.

(9.3.3.4) is a **2nd-order** ordinary differential equation (ODE) for the time-dependent vector of basis expansion coefficients $t \mapsto \vec{\mu}(t) \in \mathbb{R}^N$.

Note:

Remark 9.3.3.5 (First-order semidiscrete hyperbolic evolution problem) Completely analogous to § 9.3.1.16 we can introduce a separate unknown function for the velocity and thus arrive at a first-order ordinary differential equation in time.

$$\begin{array}{c} \mathbf{M} \left\{ \frac{d^2}{dt^2} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = 0 \\ \downarrow \qquad \qquad \qquad \leftarrow \text{auxiliary unknown } \vec{v} = \dot{\vec{\mu}} \\ \left\{ \begin{array}{l} \frac{d}{dt} \vec{\mu}(t) = \vec{v}(t) , \\ \mathbf{M} \frac{d}{dt} \vec{v}(t) = -\mathbf{A} \vec{\mu}(t) , \end{array} \right. , \quad 0 < t < T . \end{array} \quad (9.3.3.6)$$

with initial conditions

$$\vec{\mu}(0) = \vec{\mu}_0 , \quad \vec{v}(0) = \vec{v}_0 . \quad (9.3.3.7)$$

Note that (9.3.3.6) can easily be rewritten as a (first-order) ordinary differential equation in canonical form $\dot{\mathbf{u}}(t) = \mathbf{f}(t, \mathbf{u}(t))$ by identifying

$$\mathbf{u} \leftrightarrow \begin{bmatrix} \vec{\mu} \\ \vec{v} \end{bmatrix} , \quad \mathbf{f}(t, \mathbf{u}) \leftrightarrow \left(t, \begin{bmatrix} \vec{\mu} \\ \vec{v} \end{bmatrix} \right) \mapsto \begin{bmatrix} \vec{v} \\ -\mathbf{M}^{-1} \mathbf{A} \vec{\mu} \end{bmatrix} . \quad (9.3.3.8)$$

Based on (9.3.3.8) it is straightforward how to apply any Runge-Kutta single step method according to Def. 7.3.3.1 to the semi-discrete wave equation. \square

Review question(s) 9.3.3.9 (Method of lines for wave propagation)

(Q9.3.3.9.A) The **method-of-lines semi-discretization** of the variational evolution equation

$$t \in [0, T] \mapsto u(t) \in V_0 : \quad m\left(\frac{\partial^2 u}{\partial t^2}(t), v\right) + a(u(t), v) = \ell(t)(v) \quad \forall v \in V_0 ,$$

V_0 a vector space, leads to an ordinary differential equation of the form

$$\mathbf{M} \frac{d^2 \vec{\mu}}{dt^2}(t) + \mathbf{A} \vec{\mu}(t) = \vec{\phi}(t) ,$$

with $t \mapsto \vec{\mu}(t) \in \mathbb{R}^N$, $t \mapsto \vec{\phi}(t) \in \mathbb{R}^N$, and matrices $\mathbf{A}, \mathbf{M} \in \mathbb{R}^{N,N}$.

1. Give formulas for the entries of \mathbf{A} , \mathbf{M} , and $\vec{\phi}(t)$.
2. What is the meaning of $t \mapsto \vec{\mu}(t)$?

(Q9.3.3.9.B) In order to convert a variational evolution problem into an ordinary differential equation following the policy of the method of lines, we have to choose a basis of the discrete trial and test space $V_{0,h} \subset V_0$. How does the solution $t \mapsto u_h(t) \in V_{0,h}$ of the semi-discrete evolution problem depend on the choice of basis?

(Q9.3.3.9.C) We consider the hyperbolic linear evolution problem

$$u(t) \in H_0^1(\Omega): \quad \int_{\Omega} \rho(x, t) \frac{\partial^2 u}{\partial t^2}(x, t) v(x) + \sigma(x, t) \operatorname{grad} u(x) \cdot \operatorname{grad} v(x) dx = 0 \quad \forall v \in H_0^1(\Omega),$$

$$u(x, 0) = u_0(x) \quad , \quad \frac{\partial u}{\partial t}(x, 0) = v_0(x) , \quad x \in \Omega ,$$

where $\rho : [0, T] \times \Omega \rightarrow \mathbb{R}$ and $\sigma : [0, T] \times \Omega \rightarrow \mathbb{R}$ are uniformly positive coefficient functions.

We perform a method of lines spatial semi-discretization based on $V_{0,h} \subset H_0^1(\Omega)$ equipped with a basis $\{b_h^1, \dots, b_h^N\}$, $N := \dim V_{0,h}$.

1. Write down the resulting ordinary differential equation (ODE) and characterize its building blocks.
2. Describe how one can obtain the initial values for the method-of-lines ODE, if $V_{0,h}$ is a Lagrangian finite element space.

△

9.3.4 Timestepping for Semi-Discrete Wave Equations



Video tutorial for Section 9.3.4: Timestepping for Semi-Discrete Wave Equations: (43 minutes) [Download link](#), [tablet notes](#)

We still have to discretize the initial value problem (9.3.3.4) for the method-of-lines ODE in time to arrive at a scheme that can be implemented on a computer.

§9.3.4.1 (Method-of-lines ODE) We recall that the method of lines approach gives us the semi-discrete hyperbolic evolution problem in the form of the 2nd-order ODE

$$\mathbf{M} \left\{ \frac{d^2}{dt^2} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = 0 \quad , \quad \vec{\mu}(0) = \vec{\mu}_0 \quad , \quad \frac{d\vec{\mu}}{dt}(0) = \vec{\eta}_0 . \quad (9.3.4.2)$$

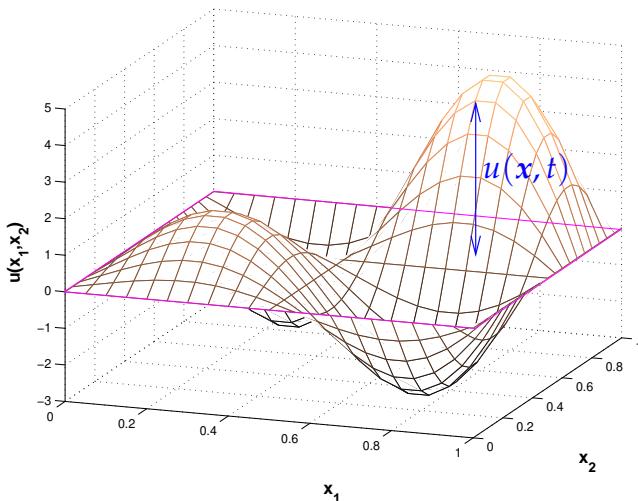
Key features of (9.3.4.2) are

◆ **reversibility:** (9.3.4.2) invariant under time-reversal $t \leftarrow -t$

◆ **energy conservation**, cf. Thm. 9.3.2.16: $E_h(t) := \frac{1}{2} \frac{d\vec{\mu}}{dt} \cdot \mathbf{M} \frac{d\vec{\mu}}{dt} + \frac{1}{2} \vec{\mu} \cdot \mathbf{A} \vec{\mu} = \text{const}$

It is highly desirable that the timestepping method respects these properties of the evolution problem, at least approximately. Timestepping schemes that achieve this are called **structure-preserving**. □

EXPERIMENT 9.3.4.3 (Euler timestepping for 1st-order form of semi-discrete wave equation) In this numerical experiment we investigate to what extent explicit and implicit Euler timestepping schemes as introduced in Ex. 9.2.6.4 and applied to the first-order form (9.3.3.6) of the semi-discrete wave equation respect the conservation of energy.



Model problem: wave propagation on a square membrane

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - \Delta u &= 0 \quad \text{on }]0, 1[^2 \times]0, 1[, \\ u(\mathbf{x}, t) &= 0 \quad \text{on } \partial\Omega \times]0, T[, \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) , \quad \frac{\partial u}{\partial t}(\mathbf{x}, 0) = 0 . \end{aligned}$$

(membrane in flat frame, displaced, but at rest at initial time)

- ◆ Initial data $u_0(\mathbf{x}) = \max\{0, \frac{1}{5} - \|\mathbf{x}\|\}$, $v_0(\mathbf{x}) = 0$,
 - ◆ $\mathcal{M} \doteq$ “structured triangular tensor product mesh”, see Fig. 257, n squares in each direction,
 - ◆ linear finite element space $V_{N,0} = \mathcal{S}_{1,0}^0(\mathcal{M})$, $N := \dim \mathcal{S}_{1,0}^0(\mathcal{M}) = (n-1)^2$,
 - ◆ All local computations (\rightarrow Section 2.7.5) rely on 3-point vertex based local quadrature formula “2D trapezoidal rule” (2.4.6.10). More explanations will be given in Rem. 9.3.4.18 below.
- ◆ $\mathbf{A} = N \times N$ Poisson matrix, see (4.1.2.8), scaled with $h := n^{-1}$,
- ◆ “lumped” diagonal mass matrix $\mathbf{M} = h\mathbf{I}$, thanks to quadrature formula, see Rem. 9.3.4.18.

Timestepping: implicit and explicit Euler method (\rightarrow Ex. 9.2.7.1, Section 6.2) for 1st-order ODE (9.3.3.6), timestep $\tau > 0$:

$$\begin{array}{ll} \vec{\mu}^{(j)} - \vec{\mu}^{(j-1)} = \tau \vec{v}^{(j-1)} , & \vec{\mu}^{(j)} - \vec{\mu}^{(j-1)} = \tau \vec{v}^{(j)} , \\ \mathbf{M}(\vec{v}^{(j)} - \vec{v}^{(j-1)}) = -\tau \mathbf{A} \vec{\mu}^{(j-1)} . & \mathbf{M}(\vec{v}^{(j)} - \vec{v}^{(j-1)}) = -\tau \mathbf{A} \vec{\mu}^{(j)} . \end{array}$$

explicit Euler

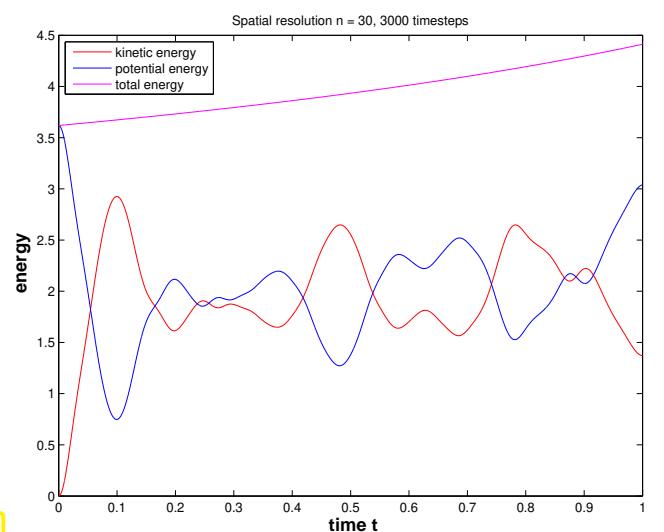
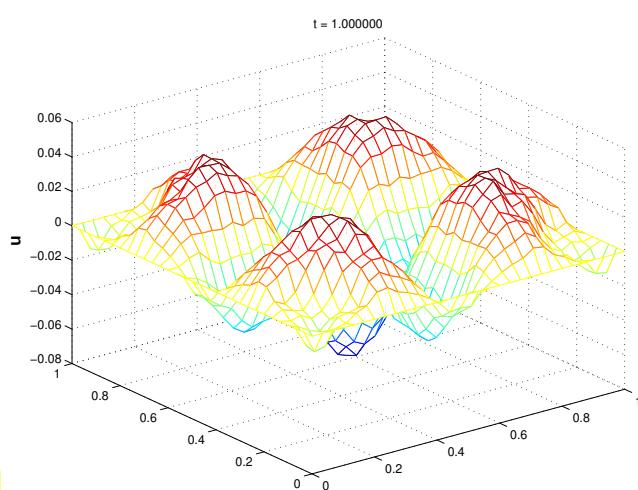
implicit Euler

Monitored: behavior of (discrete) kinetic, potential, and total energy

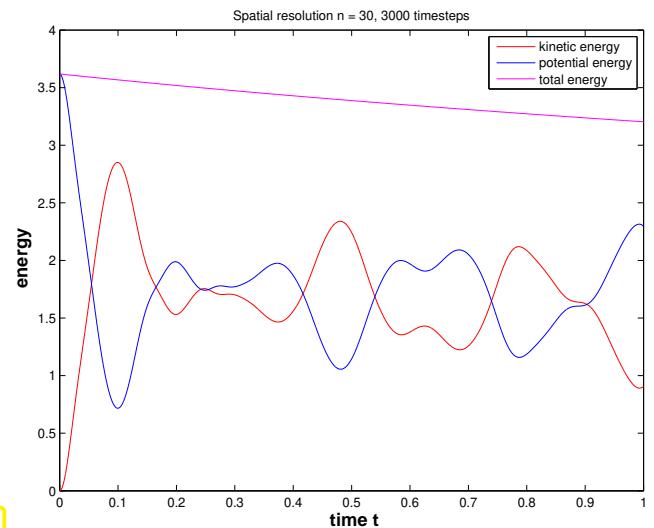
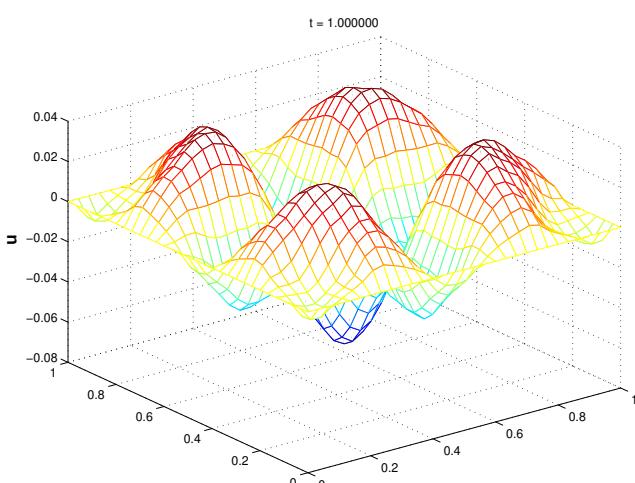
$$E_{\text{kin}}^{(j)} = \frac{1}{2} (\vec{v}^{(j)})^T \mathbf{M} \vec{v}^{(j)} , \quad E_{\text{pot}}^{(j)} = \frac{1}{2} (\vec{\mu}^{(j)})^T \mathbf{A} \vec{\mu}^{(j)} , \quad E_h^{(j)} = E_{\text{kin}}^{(j)} + E_{\text{pot}}^{(j)} . \quad (9.3.4.4)$$

for all timesteps $j = 0, 1, \dots$.

Results for explicit Euler timestepping:



Implicit Euler timestepping:

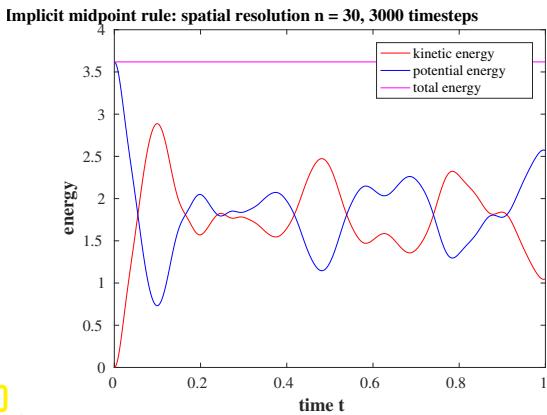


Observation: neither method conserves the total energy energy,

- ☞ explicit Euler timestepping > steady increase of total energy, solution “blows up”.
- ☞ implicit Euler timestepping > steady decrease of total energy, solution “decays”. □

Remark 9.3.4.5. What we have observed for the explicit/implicit Euler method in the previous numerical experiment is true of all explicit Runge-Kutta methods, which lead to an increase of the total energy over time, and L(π)-stable implicit Runge-Kutta methods, which make the total energy decay. □

EXPERIMENT 9.3.4.6 (Implicit midpoint rule for semi-discrete wave equation) We adopt the exact setting of Exp. 9.3.4.3, except for employing timestepping based on the implicit midpoint rule (9.2.6.7) with uniform timestep.



△ We observe *exact conservation* of the total energy for the fully discrete evolution.

It seems that then implicit midpoint rule enjoys excellent structure-preserving properties.

(An analysis will be given in the next paragraph.)

§9.3.4.7 (Crank-Nicolson timestepping) We apply the implicit midpoint single-step method, whose discrete evolution for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$ is given by

$$\Psi^{t,t+\tau} \mathbf{u} := \mathbf{w}: \quad \mathbf{w} = \mathbf{u} + \tau \mathbf{f}\left(t + \frac{1}{2}\tau, \frac{1}{2}(\mathbf{w} + \mathbf{u})\right), \quad (9.2.6.7)$$

to the first-order semi-discrete linear wave equation

$$\mathbf{u} \leftrightarrow \begin{bmatrix} \dot{\vec{\mu}}(t) \\ \dot{\vec{\nu}}(t) \end{bmatrix} = \begin{bmatrix} \vec{\nu}(t) \\ -\mathbf{M}^{-1} \mathbf{A} \vec{\mu}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{M}^{-1} \vec{\varphi}(t) \end{bmatrix} \leftrightarrow \mathbf{f}(t, \mathbf{u}), \quad (9.3.4.8)$$

which arises from Galerkin semi-discretization of (9.3.3.2) and subsequent conversion into a first-order ODE according to (9.3.3.6). The mass matrix $\mathbf{M} \in \mathbb{R}^{N,N}$ is symmetric, positive definite, the stiffness matrix $\mathbf{A} \in \mathbb{R}^{N,N}$ at least symmetric and positive semi-definite, cf. (9.3.3.4).

For the update of the states $\vec{\mu}^{j-1} \rightarrow \vec{\mu}^{(j)}$, $\vec{v}^{(j-1)} \rightarrow \vec{v}^{(j)}$ in a single timestep $t - \tau \rightarrow t$, $\tau > 0$, by combining (9.2.6.7) and (9.3.4.8) we obtain the implicit equations

$$\begin{bmatrix} \vec{\mu}^{(j)} \\ \vec{v}^{(j)} \end{bmatrix} = \begin{bmatrix} \vec{\mu}^{(j-1)} \\ \vec{v}^{(j-1)} \end{bmatrix} + \tau \begin{bmatrix} \frac{1}{2}(\vec{v}^{(j-1)} + \vec{v}^{(j)}) \\ -\frac{1}{2}\mathbf{M}^{-1}\mathbf{A}(\vec{\mu}^{(j-1)} + \vec{\mu}^{(j)}) \end{bmatrix} + \tau \begin{bmatrix} 0 \\ \mathbf{M}^{-1}\vec{\varphi}(t_j - \frac{1}{2}\tau) \end{bmatrix}. \quad (9.3.4.9)$$

This amounts to a linear system of equations for the new states

$$\begin{bmatrix} \mathbf{I} & -\frac{1}{2}\tau\mathbf{I} \\ \frac{1}{2}\tau\mathbf{A} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \vec{\mu}^{(j)} \\ \vec{v}^{(j)} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \frac{1}{2}\tau\mathbf{I} \\ -\frac{1}{2}\tau\mathbf{A} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \vec{\mu}^{(j-1)} \\ \vec{v}^{(j-1)} \end{bmatrix} + \tau \begin{bmatrix} 0 \\ \vec{\varphi}(t_j - \frac{1}{2}\tau) \end{bmatrix}, \quad (9.3.4.10)$$

$$\begin{aligned} &\Updownarrow \\ \vec{\mu}^{(j)} - \vec{\mu}^{(j-1)} &= \frac{1}{2}\tau(\vec{v}^{(j-1)} + \vec{v}^{(j)}) \\ \mathbf{M}(\vec{v}^{(j)} - \vec{v}^{(j-1)}) &= -\frac{1}{2}\tau\mathbf{A}(\vec{\mu}^{(j-1)} + \vec{\mu}^{(j)}) + \tau\vec{\varphi}(t_j - \frac{1}{2}\tau). \end{aligned} \quad (9.3.4.11)$$

We left multiply the first equation of (9.3.4.11) with $(\vec{\mu}^{(j-1)} + \vec{\mu}^{(j)})^\top \mathbf{A}^\top$ and the second equation with $(\vec{v}^{(j-1)} + \vec{v}^{(j)})^\top$, use " $(a+b)(a-b) = a^2 - b^2$ ", and then add both equations:

$$\begin{aligned} &(\vec{\mu}^{(j)})^\top \mathbf{A}(\vec{\mu}^{(j)}) - (\vec{\mu}^{(j-1)})^\top \mathbf{A}(\vec{\mu}^{(j-1)}) + (\vec{v}^{(j)})^\top \mathbf{M}(\vec{v}^{(j)}) - (\vec{v}^{(j-1)})^\top \mathbf{M}(\vec{v}^{(j-1)}) \\ &= \frac{1}{2}\tau((\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)})^\top \mathbf{A}(\vec{v}^{(j)} + \vec{v}^{(j-1)}) - (\vec{v}^{(j)} + \vec{v}^{(j-1)})^\top \mathbf{A}(\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)})) + \\ &\quad \tau(\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)})^\top \vec{\varphi}(t_j - \frac{1}{2}\tau) \\ &= \tau(\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)})^\top \vec{\varphi}(t_j - \frac{1}{2}\tau), \end{aligned}$$

because \mathbf{A} is symmetric. If $\vec{\varphi}(t) \equiv \mathbf{0}$, then

$$(\vec{\mu}^{(j)})^\top \mathbf{A}(\vec{\mu}^{(j)}) + (\vec{v}^{(j)})^\top \mathbf{M}(\vec{v}^{(j)}) = (\vec{\mu}^{(j-1)})^\top \mathbf{A}(\vec{\mu}^{(j-1)}) + (\vec{v}^{(j-1)})^\top \mathbf{M}(\vec{v}^{(j-1)}),$$

which is **conservation of total energy** in a single timestep, in perfect agreement with the observations from Exp. 9.3.4.6.

Let us eliminate the auxiliary vectors $\vec{v}^{(j)}$. Therefore subtract the first equation of (9.3.4.11) for two consecutive timesteps and left multiply them with the mass matrix \mathbf{M} :

$$\begin{aligned} \mathbf{M}(\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}) &= \frac{1}{2}\tau\mathbf{M}(\vec{v}^{(j)} - \vec{v}^{(j-1)} + \vec{v}^{(j+1)} - \vec{v}^{(j)}) \\ &= -\frac{1}{2}\tau^2\mathbf{A}(\vec{\mu}^{(j+1)} + \vec{\mu}^{(j-1)}) + \frac{1}{2}\tau^2(\vec{\varphi}(t_j - \frac{1}{2}\tau) + \vec{\varphi}(t_j + \frac{1}{2}\tau)). \end{aligned}$$

Dividing by τ^2 , we can view the resulting **2-step method** as a symmetric finite-difference approximation of the second time derivative combined with an averaging of the right hand side in (9.3.3.4):

$$\mathbf{M} \left\{ \frac{d^2}{dt^2} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = \vec{\varphi}(t) \quad (9.3.4.2)$$

\Downarrow

$$\mathbf{M} \frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\frac{1}{2}\mathbf{A}(\vec{\mu}^{(j-1)} + \vec{\mu}^{(j+1)}) + \frac{1}{2}(\vec{\varphi}(t_j - \frac{1}{2}\tau) + \vec{\varphi}(t_j + \frac{1}{2}\tau)), \quad j = 1, 2, \dots$$

(9.3.4.12)

This is called the **Crank-Nicolson** timestepping scheme for the semi-discrete wave equation.

We have seen in Ex. 9.2.6.8 that the implicit midpoint rule is a second-order timestepping scheme and also conclude that the Crank-Nicolson method is a **2nd-order method**. \square

§9.3.4.13 (Störmer-Verlet timestepping) In the case of Crank-Nicolson timestepping (9.3.4.12) we rely on an average on the right-hand side, because the second difference quotient on the left-hand side is centered at time t corresponding to timestep j .

It is natural to replace the average with a simple evaluation of the right-hand side at time $t \leftrightarrow$ timestep j , which leads to

$$\mathbf{M} \left\{ \frac{d^2}{dt^2} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = 0 \quad (9.3.4.2)$$

\Downarrow

$$\mathbf{M} \frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\mathbf{A} \vec{\mu}^{(j)}, \quad j = 1, 2, \dots \quad (9.3.4.14)$$

This is a **two-step method**, the **Störmer scheme/explicit trapezoidal rule**

By Taylor expansion, see Ex. 9.2.6.8:

Störmer scheme is a **2nd-order** method

However, from where do we get $\vec{\mu}^{(-1)}$? Two-step methods need to be kick-started by a **special initial step**: This is constructed by approximating the second initial condition by a symmetric difference quotient:

$$\frac{d}{dt} \vec{\mu}(0) = \vec{v}_0 \quad \Rightarrow \quad \frac{\vec{\mu}^{(1)} - \vec{\mu}^{(-1)}}{2\tau} = \vec{v}_0. \quad (9.3.4.15)$$

§9.3.4.16 (Leapfrog timestepping) In the case of the Crank-Nicolson timestepping we could give algebraically equivalent formulations in both the two vectors $\vec{\mu}^{(j)}, \vec{v}^{(j)}$ and in $\vec{\mu}^{(k)}$ alone. Now we do this for the explicit trapezoidal rule/Störmer scheme (9.3.4.14) applied to the semi-discrete wave equation:

$$\mathbf{M} \frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\mathbf{A} \vec{\mu}^{(j)}, \quad j = 1, \dots. \quad (9.3.4.14)$$

Inspired by Rem. 9.3.3.5 we introduce the auxiliary variable

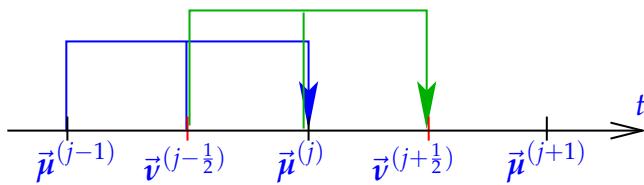
$$\vec{v}^{(j+1/2)} := \frac{\vec{\mu}^{(j+1)} - \vec{\mu}^{(j)}}{\tau},$$

which can be read as an approximation of the velocity $v := \dot{u}$ at a “half timestep”.

This leads to a timestepping scheme, which is *algebraically equivalent* to the explicit trapezoidal rule: **leapfrog timestepping** (with uniform timestep $\tau > 0$):

$$\boxed{\begin{aligned} \mathbf{M} \frac{\vec{v}^{(j+1/2)} - \vec{v}^{(j-1/2)}}{\tau} &= -\mathbf{A} \vec{\mu}^{(j)}, \\ \frac{\vec{\mu}^{(j+1)} - \vec{\mu}^{(j)}}{\tau} &= \vec{v}^{(j+1/2)}, \end{aligned}} \quad j = 0, 1, \dots \quad (9.3.4.17)$$

+ initial step $\vec{v}^{(-1/2)} + \vec{v}^{(1/2)} = 2\vec{v}_0$.



work per step:

$1 \times$ evaluation $\mathbf{A} \times$ vector,
 $1 \times$ solution of linear system for \mathbf{M}

↓

Remark 9.3.4.18 (Mass lumping) Required in each step of leapfrog timestepping: solution of linear system of equations with (large sparse) system matrix $\mathbf{M} \in \mathbb{R}^{N,N} \gg$ expensive!

Trick for (bi-)linear finite element Galerkin discretization: $V_{0,h} \subset S_1^0(\mathcal{M})$:

use *vertex based local quadrature rule*

(e.g. “2D trapezoidal rule” (2.4.6.10) on triangular mesh)

$$\int_K f(x) dx \approx \frac{|K|}{\#\mathcal{V}(K)} \sum_{p \in \mathcal{V}(K)} f(p), \quad \mathcal{V}(K) := \text{set of vertices of } K.$$

(For a comprehensive discussion of local quadrature rules see Section 2.7.5)

- Mass matrix \mathbf{M} will become a *diagonal* matrix (due to defining equation (2.4.3.5) for nodal basis functions, which are associated with nodes of the mesh).

We point out that this so-called **mass lumping** trick was used in the finite element discretization of Exp. 9.3.4.3.

Leapfrog vs. Crank-Nicolson

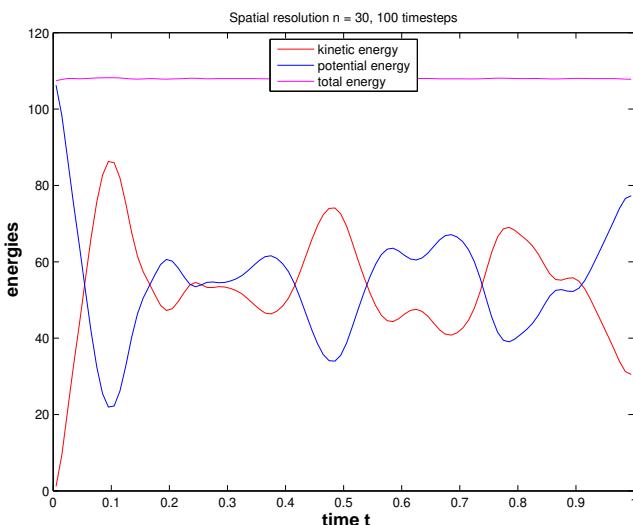
Given the possibility of mass lumping, the computational effort for leapfrog timestepping (9.3.4.17) becomes substantially smaller than that for the Crank-Nicolson scheme (9.3.4.12), because the latter will always involve the solution of a $N \times N$ sparse linear system of equations.

↓

We cannot expect exact conservation of the total energy for leapfrog timestepping. Will it be as disappointing as the Euler single-step methods in this respect?

EXPERIMENT 9.3.4.20 (Energy conservation for leapfrog) We rely on the same model problem and spatial discretization as in Exp. 9.3.4.3. We employ leapfrog timestepping with constant timestep size $\tau = 0.01$ as introduced in § 9.3.4.16.

The EIGEN-based C++ codes used to conduct this experiments can be found in → [GITLAB](#).



Leapfrog is (nearly) energy conserving
(no energy drift, only small oscillations)

This behavior is explained by the deep mathematical theory of **symplectic integrators**, see [HLW06].

Review question(s) 9.3.4.21 (Timestepping for Semi-Discrete Wave Equations)

(Q9.3.4.21.A) Remember that Crank-Nicolson timestepping for the linear ODE

$$\mathbf{M} \frac{d^2 \vec{\mu}}{dt^2}(t) + \mathbf{A} \vec{\mu}(t) = \vec{\phi}(t)$$

gives rise to the recursion

$$\mathbf{M} \frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\frac{1}{2} \mathbf{A} (\vec{\mu}^{(j-1)} + \vec{\mu}^{(j+1)}) + \frac{1}{2} (\vec{\phi}(t_j - \frac{1}{2}\tau) + \vec{\phi}(t_j + \frac{1}{2}\tau)), \quad j = 1, 2, \dots,$$

while the Störmer-Verlet timestepping yields

$$\mathbf{M} \frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\mathbf{A} \vec{\mu}^{(j)} + \vec{\phi}(t_j).$$

Generalize these timestepping schemes to the second-order ODE

$$\ddot{\mathbf{u}}(t) = \mathbf{f}(t, \mathbf{u}(t)) , \quad \mathbf{f}: \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n , \quad j = 1, 2, 3, \dots .$$

(Q9.3.4.21.B) We consider the linear non-autonomous ODE

$$\frac{d\vec{\mu}}{dt}(t) = \mathbf{A}(t) \vec{\mu}(t) \quad [\vec{\mu}(t) \in \mathbb{R}^N] , \quad (9.3.4.22)$$

with a *skew-symmetric* matrix-valued function $t \mapsto \mathbf{A}(t) \in \mathbb{R}^{N,N}$, that is, $\mathbf{A}(t)^\top = -\mathbf{A}(t)$. We apply the implicit midpoint method with uniform timestep $\tau > 0$ to discretize (9.3.4.22) in time, which produces the sequence $\vec{\mu}^{(j)}$, $j = 0, 1, 2, \dots$. Show that $\|\vec{\mu}^{(j)}\| = \|\vec{\mu}^{(0)}\|$ for all j .

Hint. The discrete evolution operator of the implicit midpoint rule when applied to the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$ is

$$\Psi^{t,t+\tau} \mathbf{u} := \mathbf{w}: \quad \mathbf{w} = \mathbf{u} + \tau \mathbf{f}(t + \frac{1}{2}\tau, \frac{1}{2}(\mathbf{w} + \mathbf{u})).$$

(Q9.3.4.21.C) What is **mass lumping** and why is it important in the context of the leapfrog timestepping scheme

$$\begin{aligned} \mathbf{M} \frac{\vec{v}^{(j+\frac{1}{2})} - \vec{v}^{(j-\frac{1}{2})}}{\tau} &= -\mathbf{A} \vec{\mu}^{(j)}, \\ \frac{\vec{\mu}^{(j+1)} - \vec{\mu}^{(j)}}{\tau} &= \vec{v}^{(j+\frac{1}{2})}, \end{aligned} \quad j = 0, 1, \dots$$

for the method-of-lines ODE

$$\mathbf{M} \left\{ \frac{d^2}{dt^2} \vec{\mu}(t) \right\} + \mathbf{A} \vec{\mu}(t) = 0 ?$$

(Q9.3.4.21.D) Explain the statement

There is *no drift* of total energy when using **leapfrog timestepping** for the method-of-lines ODE for a linear wave equation.

△

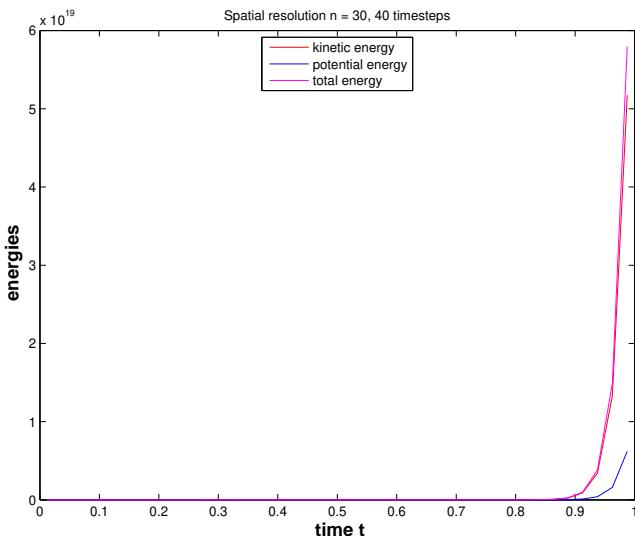
9.3.5 The Courant-Friedrichs-Levy (CFL) Condition



Video tutorial for Section 9.3.5: The Courant-Friedrichs-Levy (CFL) Condition: (46 minutes)
[Download link](#), [tablet notes](#)

Crank-Nicolson timestepping will conserve the total energy regardless of the size $\tau > 0$ of the timestep and, thus, is **unconditionally stable**. Excellent approximate energy conservation was observed in Exp. 9.3.4.20 for the leapfrog method. Does it also enjoy unconditional stability?

EXPERIMENT 9.3.5.1 (Blow-up for leapfrog timestepping)



▷ Exp. 9.3.4.20 repeated with $\tau = 0.04$
Observation:

Leapfrog suffers a **blow-up**: exponential increase of energies!
A similar behavior is observed with the explicit Euler scheme for the semi-discrete heat equation, in case the timestep constraint is violated, see Section 9.2.7.2.

§9.3.5.2 (Diagonalization of method-of-lines ODE → § 9.2.7.15) As in Section 9.2.7.2 the stability analysis of leapfrog timestepping can be based on **diagonalization**. Recall that we can find N linearly independent **generalized eigenvectors** $\vec{\psi}_1, \dots, \vec{\psi}_N \in \mathbb{R}^N$ solving the generalized eigenvalue problem

$$\mathbf{A} \vec{\psi}_i = \lambda_i \mathbf{M} \vec{\psi}_i , \quad \vec{\psi}_j^\top \mathbf{M} \vec{\psi}_i = \delta_{ij} , \quad 1 \leq i, j \leq N , \quad (9.2.7.16)$$

with positive **eigenvalues** $\lambda_i > 0$. By means of the regular square matrices

$$\mathbf{T} = [\vec{\psi}_1, \dots, \vec{\psi}_N] \in \mathbb{R}^{N,N} , \quad (9.2.7.17)$$

$$\mathbf{D} := \text{diag}(\lambda_1, \dots, \lambda_N) \in \mathbb{R}^{N,N} , \quad (9.2.7.18)$$

we can rewrite (9.2.7.16) as

$$\mathbf{AT} = \mathbf{MTD} , \quad \mathbf{T}^\top \mathbf{MT} = \mathbf{I} . \quad (9.2.7.19)$$

where the $\lambda_i > 0$ are **generalized eigenvalues** for $\mathbf{A}\vec{\xi} = \lambda \mathbf{M}\vec{\xi}$ ➤ $\lambda_i \geq \gamma$ for all i (γ is the constant introduced in (9.2.3.6)).

Now, analogous to § 9.2.7.23, we recast the 2-step formulation (9.3.4.14) in terms of the transformed coefficient vector

$$\vec{\eta}^{(j)} = \mathbf{T}^\top \mathbf{M} \vec{\mu}^{(j)} \quad \Leftrightarrow \quad \vec{\mu}^{(j)} = \mathbf{T} \vec{\eta}^{(j)}, \quad j \in \mathbb{N}_0 : \quad (9.3.5.3)$$

$$\mathbf{M} \frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\mathbf{A} \vec{\mu}^{(j)} \quad \vec{\eta} := \mathbf{T}^\top \mathbf{M} \vec{\mu} \quad \vec{\eta}^{(j+1)} - 2\vec{\eta}^{(j)} + \vec{\eta}^{(j-1)} = -\tau^2 \mathbf{D} \vec{\eta}^{(j)}.$$

Again, we have achieved a complete decoupling of the timestepping for the eigencomponents.

$$\eta_i^{(j+1)} - 2\eta_i^{(j)} + \eta_i^{(j-1)} = -\tau^2 \lambda_i \eta_i^{(j)}, \quad i = 1, \dots, N, \quad j = 1, 2, \dots. \quad (9.3.5.4)$$

In fact, (9.3.5.4) is what we end up with then applying Störmer's scheme to the *scalar* linear 2nd-order ODE $\ddot{\eta}_i = -\lambda_i \eta_i$. In a sense, the commuting diagram (9.2.7.41) remains true for 2-step methods and second-order ODEs.

(9.3.5.4) is a **linear two-step recurrence** formula for the sequences $(\eta_i^{(j)})_j$.

Try:

$$\eta_i^{(j)} = \xi^j \quad \text{for some } \xi \in \mathbb{C} \setminus \{0\}$$

Plug this into (9.3.5.4)

$$\begin{aligned} & \blacktriangleright \quad \xi^2 - 2\xi + 1 = -\tau^2 \lambda_i \xi \quad \Leftrightarrow \quad \xi^2 - (2 - \tau^2 \lambda_i) \xi + 1 = 0. \\ & \Rightarrow \quad \text{two solutions } \xi_{\pm} = \frac{1}{2} \left(2 - \tau^2 \lambda_i \pm \sqrt{(2 - \tau^2 \lambda_i)^2 - 4} \right). \end{aligned}$$

We can get a blow-up of some solutions of (9.3.5.4), if $|\xi_+| > 1$ or $|\xi_-| > 1$. From secondary school we know Vieta's formula

$$\xi_+ \cdot \xi_- = 1 \Rightarrow \begin{cases} \xi_{\pm} \in \mathbb{R} \quad \text{and} \quad \xi_+ \neq \xi_- \Rightarrow |\xi_+| > 1 \text{ or } |\xi_-| > 1 \\ \xi_- = \xi_+^* \Rightarrow |\xi_-| = |\xi_+| = 1 \end{cases},$$

where ξ_+^* designates complex conjugation. So the recurrence (9.3.5.4) has only bounded solution, if and only if

$$\text{discriminant } D := (2 - \tau^2 \lambda_i)^2 - 4 \leq 0 \quad \Leftrightarrow \quad \boxed{\tau \leq \frac{2}{\sqrt{\lambda_i}}}. \quad (9.3.5.5)$$

\longleftrightarrow stability induced timestep constraint for leapfrog timestepping \square

§9.3.5.6 (The CFL-condition)

Special setting: spatial finite element Galerkin discretization based on fixed degree Lagrangian finite element spaces (\rightarrow Section 2.6), meshes created by uniform regular refinement.

Under these conditions a generalization of Lemma 9.2.7.30, which predicts $\max_i \lambda_i = O(h_M^{-2})$,

Lemma 9.2.7.30. Behavior of generalized eigenvalues

Let \mathcal{M} be a simplicial mesh and \mathbf{A}, \mathbf{M} denote the Galerkin matrices for the bilinear forms $\mathbf{a}(u, v) = \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} v \, dx$ and $\mathbf{m}(u, v) = \int_{\Omega} u(x)v(x) \, dx$, respectively, and $V_{0,h} := \mathcal{S}_{p,0}^0(\mathcal{M})$. Then the smallest and largest generalized eigenvalues of $\mathbf{A}\vec{\mu} = \lambda\mathbf{M}\vec{\mu}$, denoted by λ_{\min} and λ_{\max} , satisfy

$$\frac{1}{\text{diam}(\Omega)^2} \leq \lambda_{\min} \leq C, \quad \lambda_{\max} \geq Ch_{\mathcal{M}}^{-2},$$

where the “generic constants” (\rightarrow Rem. 3.3.5.8) depend only on the polynomial degree p , the domain Ω , and the shape regularity measure $\rho_{\mathcal{M}}$.

combined with (9.3.5.5) implies the following timestep constraint:

Stability of leapfrog timestepping entails $\tau \leq O(h_{\mathcal{M}})$ for $h_{\mathcal{M}} \rightarrow 0$

This is known as

Courant-Friedrichs-Lowy (CFL) condition

Remark 9.3.5.7 (Geometric interpretation of CFL condition in 1D) The CFL-condition is intimately connected with the finite speed of information propagation when viewed from a geometric angle. We elaborate this in the following setting:

- ◆ 1D wave equation, (spatial) boundary conditions ignored (“Cauchy problem”),

$$c > 0: \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0, \quad u(x, 0) = u_0(x), \quad \frac{\partial u}{\partial t}(x, 0) = v_0(x), \quad x \in \mathbb{R}. \quad (9.3.2.2)$$

- ◆ Linear finite element Galerkin discretization on an infinite equidistant spatial mesh $\mathcal{M} := \{[x_{j-1}, x_j] : j \in \mathbb{Z}\}$, $x_j := h j$ (meshwidth h), see Section 2.3.
- ◆ Mass lumping for computation of mass matrix, which will become $h \cdot \mathbf{I}$, see Rem. 9.3.4.18.
- ◆ Timestepping by Störmer scheme (9.3.4.14) with constant timestep $\tau > 0$.

The considerations will all be local, which relieves us from worrying about the infinite mesh and the infinitely long basis expansion vectors $\vec{\mu}^{(j)} \in \mathbb{R}^{\mathbb{Z}}$ for which we have the linear recursion

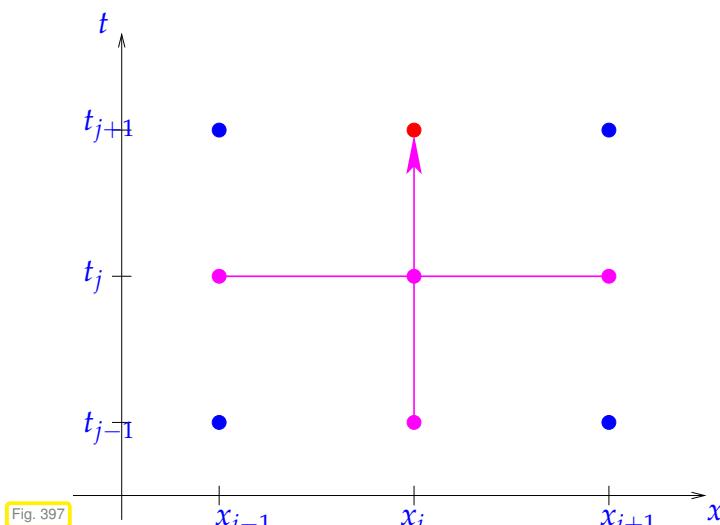
$$\mathbf{M} \frac{\vec{\mu}^{(j+1)} - 2\vec{\mu}^{(j)} + \vec{\mu}^{(j-1)}}{\tau^2} = -\mathbf{A}\vec{\mu}^{(j)}, \quad j = 1, \dots. \quad (9.3.4.14)$$

In the current setting:

- ◆ $\mathbf{M} = h \cdot \mathbf{I}$ by mass lumping,
- ◆ \mathbf{A} is *tri-diagonal*, compare (2.3.3.4).

Thus, the equations for single components $\mu_i^{(j)}$, $i \in \mathbb{Z}$, of $\vec{\mu}^{(j)}$ can easily be extracted from (9.3.4.14):

$$h \frac{\mu_i^{(j+1)} - 2\mu_i^{(j)} + \mu_i^{(j-1)}}{\tau^2} = -(\mathbf{A})_{i,i-1}\mu_{i-1}^{(j)} - (\mathbf{A})_{i,i}\mu_i^{(j)} - (\mathbf{A})_{i,i+1}\mu_{i+1}^{(j)}, \quad i \in \mathbb{Z}. \quad (9.3.5.8)$$

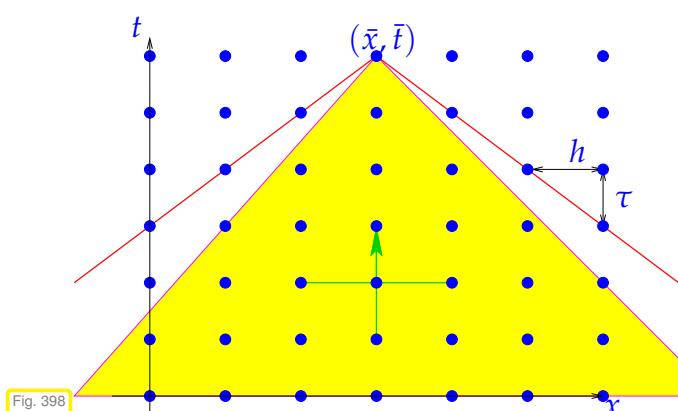


Störmer/leapfrog scheme (9.3.5.8) in stencil notation on space-time grid $\{(x_i, t_j)\}_{i \in \mathbb{Z}, j \in \mathbb{N}_0}$.

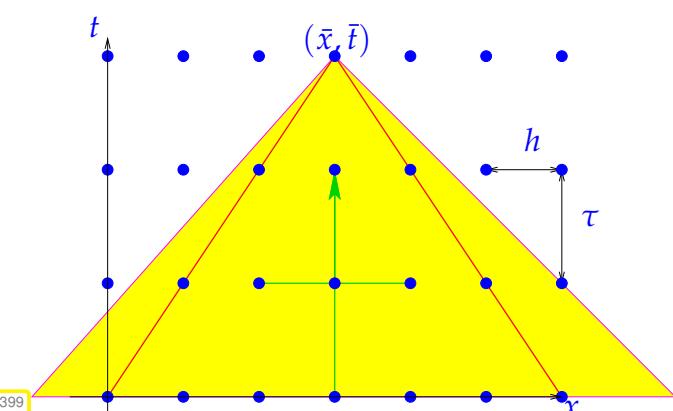
▷ flow of information in one step of Störmer scheme: $\mu_i^{(j+1)}$ from $\mu_i^{(j)}, \mu_{i-1}^{(j)}, \mu_{i+1}^{(j)}, \mu_i^{(j-1)}$.

Since the method is a two-step method, information from time-slices t_j and t_{j-1} is needed.

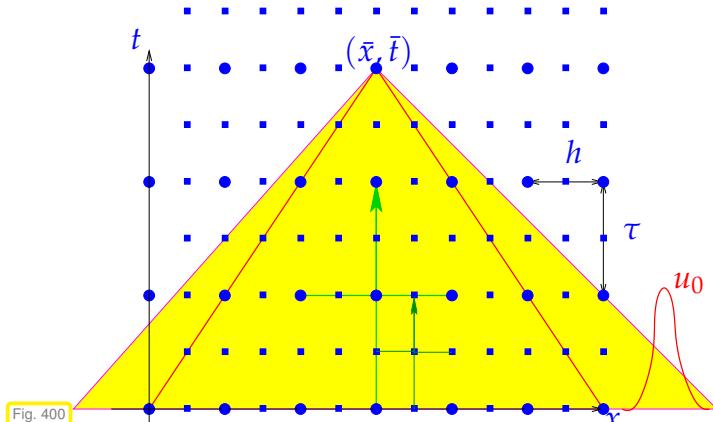
Below: yellow region $\hat{\Delta}$ $\hat{\Delta}$ domain of dependence (d.o.d.) of (\bar{x}, \bar{t})



$c\tau < h$: numerical domain of dependence
(marked with $\hat{\Delta}$) contains d.o.d. \blacksquare
CFL-condition fulfilled



$c\tau > h$: numerical domain of dependence
(marked with $\hat{\Delta}$) smaller than d.o.d. \blacksquare
CFL-condition violated



(\bullet $\hat{\Delta}$ \square $\hat{\Delta}$ $\hat{\Delta}$ coarse grid, fine grid, d.o.d.)

▷ 1D consideration:

sequence of equidistant space-time grids of $\tilde{\Omega}$ with $\tau = \gamma h$ (τ/h = meshwidth in time/space)

If $\gamma >$ CFL-constraint (here $\gamma > c^{-1}$), then
analytical domain of dependence $\not\subset$ numerical domain of dependence

▲ initial data u_0 outside numerical domain of dependence cannot influence approximation at grid point (\bar{x}, \bar{t}) on any mesh \blacktriangleright no convergence !

CFL-condition \Leftrightarrow analytical domain of dependence \subset numerical domain of dependence

§9.3.5.9 (Gauging CFL-condition) Will the CFL-condition thwart the efficient use of leapfrog, see Rem. 9.2.8.10? To this end we need an idea about the convergence of the solutions of the fully discrete method:

“Meta-theorem” 9.3.5.10. Convergence of fully discrete solutions of the wave equation

Assume that

- ◆ the solution of the IBVP for the wave equation (9.3.2.14) is “sufficiently smooth”,
- ◆ its spatial Galerkin finite element discretization relies on degree p Lagrangian finite elements (\rightarrow Section 2.6) on uniformly shape-regular families of meshes,
- ◆ timestepping is based on the leapfrog method (9.3.4.17) with uniform timestep $\tau > 0$ satisfying (9.3.5.5).

Then we can expect an asymptotic behavior of the total discretization error according to

$$\left(\tau \sum_{j=1}^M \|u - u_h(\tau j)\|_{H^1(\Omega)}^2 \right)^{\frac{1}{2}} \leq C(h_M^p + \tau^2), \quad (9.3.5.11)$$

$$\left(\tau \sum_{j=1}^M \|u - u_h(\tau j)\|_{L^2(\Omega)}^2 \right)^{\frac{1}{2}} \leq C(h_M^{p+1} + \tau^2), \quad (9.3.5.12)$$

where $C > 0$ must not depend on h_M , τ .

L.F. is 2nd-order!

“expect”: unless lack of regularity of the solution u interferes, cf. Section 3.4, § 3.4.0.11.

As in the case of Thm. 9.2.8.5 (\Rightarrow nothing new!) we find:

$\text{total discretization error} = \text{spatial error} + \text{temporal error}$

§ 3.3.5.9 still applies: (9.3.5.11) does not give information about actual error, but only about the trend of the error, when discretization parameters h_M and τ are varied.

► Nevertheless, as in the case of the a priori error estimates of Section 3.3.5, we can draw conclusions about optimal refinement strategies in order to achieve prescribed *error reduction*.

As in Section 3.3.5 we make the **assumption** that the estimates (9.3.5.11) are sharp for all contributions to the total error and that the constants are the same (!)

$$\begin{aligned} \text{contribution of spatial (energy) error} &\approx Ch_M^p, \quad h_M \hat{=} \text{mesh width} (\rightarrow \text{Def. 3.2.1.4}), \\ \text{contribution of temporal error} &\approx C\tau^2, \quad \tau \hat{=} \text{timestep size}. \end{aligned} \quad (9.3.5.13)$$

This suggests the following change of h_M , τ in order to achieve *error reduction* by a factor of $\rho > 1$:

$$\begin{array}{l} \text{reduce mesh width by factor } \rho^{1/p} \\ \text{reduce timestep by factor } \rho^{1/2} \end{array} \xrightarrow{(9.2.8.7)} \text{(energy) error reduction by } \rho > 1. \quad (9.3.5.14)$$

Guideline: spatial and temporal resolution have to be adjusted in tandem

Parallel zu Rem. 9.2.8.10 we may wonder whether the timestep constraint $\tau < O(h_M)$ (asymptotically) enforces small timesteps not required for accuracy:

When interested in error in *energy norm* ($\leftrightarrow H^1(\Omega)$ -norm):

Only for $p = 1$ (linear Lagrangian finite elements) the requirement $\tau < O(h_M)$ stipulates the use of a smaller timestep than accuracy balancing according to (9.3.5.14).

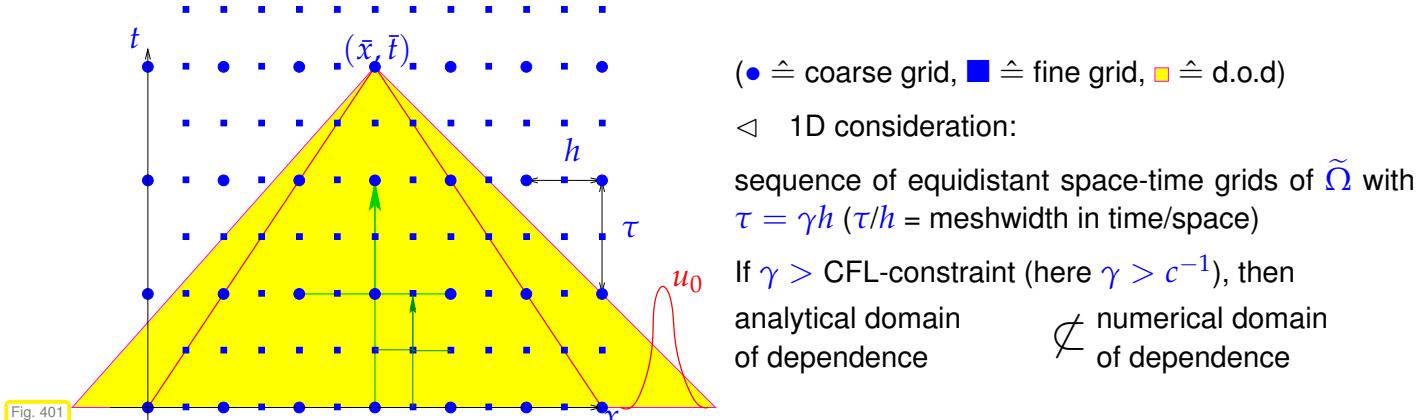
When interested in $L^2(\Omega)$ -norm:

No undue timestep constraint enforced by CFL-condition for any (h -version) of Lagrangian finite element Galerkin discretization.

The leapfrog timestep constraint $\tau \leq O(h_M)$ does not compromise (asymptotic) efficiency, if $p \geq 2$ (p $\hat{=}$ degree of spatial Lagrangian finite elements).

Review question(s) 9.3.5.15 (CFL-condition)

(Q9.3.5.15.A) In Rem. 9.3.5.7 you have seen the following drawing and caption



Explain with your own words all elements of the drawing and the statement

analytical domain of dependence $\not\subset$ numerical domain of dependence

(Q9.3.5.15.B) In the spirit of the methods of lines we perform the spatial semi-discretization of a linear wave equation on a 2D spatial domain by means of piecewise linear Lagrangian finite elements, trial/test space $S_1^0(\mathcal{M})$, on a sequence of meshes generated by uniform regular refinement.

We convert the resulting method-of-lines ODE

$$\mathbf{M} \frac{d^2 \vec{\mu}}{dt^2}(t) + \mathbf{A} \vec{\mu}(t) = 0$$

into a first-order form by introducing the auxiliary coefficient vector $\vec{\nu}(t) := \mathbf{M}^{-1} \dot{\vec{\mu}}(t)$.

Describe the stability-induced timestep constraints we face when applying explicit and implicit Euler timestepping with uniform timestep $\tau > 0$ for the temporal discretization of that first-order ODE. Give a bound for the timestep τ in terms of the meshwidth h_M of the finite-element mesh.

(Q9.3.5.15.C) Explain the following statement:

Leapfrog timestepping for the linear wave equation discretized in space by means of linear finite elements is **efficient** despite the **CFL-condition**.

“Meta-theorem” 9.3.5.10. Convergence of fully discrete solutions of the wave equation

Assume that

- ◆ the solution of the IBVP for the wave equation (9.3.2.14) is “sufficiently smooth”,
- ◆ its spatial Galerkin finite element discretization relies on degree p Lagrangian finite elements (\rightarrow Section 2.6) on uniformly shape-regular families of meshes,
- ◆ timestepping is based on the leapfrog method (9.3.4.17) with uniform timestep $\tau > 0$ satisfying (9.3.5.5).

Then we can expect an asymptotic behavior of the total discretization error according to

$$\left(\tau \sum_{j=1}^M \|u - u_h(\tau j)\|_{H^1(\Omega)}^2 \right)^{\frac{1}{2}} \leq C(h_M^p + \tau^2) \quad (9.3.5.11)$$

$$\left(\tau \sum_{j=1}^M \|u - u_h(\tau j)\|_{L^2(\Omega)}^2 \right)^{\frac{1}{2}} \leq C(h_M^{p+1} + \tau^2),$$

where $C > 0$ must not depend on h_M , τ .

L.F. is 2nd-order!

△

Learning Outcomes

After having studied this section you should

- know the transient heat equation along with suitable initial and boundary conditions for it.
- know the wave equation governing the movement of a taut elastic membrane.
- be able to state the spatial variational formulation of given second-order linear parabolic and hyperbolic IBVPs.
- understand the principle of the method of lines and how to apply it to convert a second-order linear parabolic/hyperbolic IBVP into an ordinary differential equation.
- be able to apply a Runge-Kutta timestepping scheme to a spatially semi-discrete linear IBVP.
- why semi-discrete second-order linear parabolic IBVP are “stiff” and why this calls for implicit timestepping based on efficiency considerations.
- be able to predict the convergence of a full discretization of a second-order linear parabolic/hyperbolic IBVP.
- know Störmer/leapfrog timestepping scheme for the linear wave equation.

Bibliography

- [DB02] P. Deuflhard and F. Bornemann. *Scientific Computing with Ordinary Differential Equations*. 2nd ed. Vol. 42. Texts in Applied Mathematics. New York: Springer, 2002 (cit. on pp. 564, 567, 568).
- [Eva98] L.C. Evans. *Partial differential equations*. Vol. 19. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 1998 (cit. on p. 595).
- [HLW06] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration*. 2nd ed. Vol. 31. Springer Series in Computational Mathematics. Heidelberg: Springer, 2006 (cit. on p. 606).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 550, 564, 567, 570, 573, 577, 580).
- [KC19] Christopher A. Kennedy and Mark H. Carpenter. “Diagonally implicit Runge-Kutta methods for stiff ODEs”. In: *Appl. Numer. Math.* 146 (2019), pp. 221–244. DOI: [10.1016/j.apnum.2019.07.008](https://doi.org/10.1016/j.apnum.2019.07.008) (cit. on p. 568).
- [KA03] P. Knabner and L. Angermann. *Numerical Methods for Elliptic and Parabolic Partial Differential Equations*. Vol. 44. Texts in Applied Mathematics. Heidelberg: Springer, 2003 (cit. on p. 585).

Chapter 10

Convection-Diffusion Problems

As a new aspect this chapter introduces *linear transport* into the stationary and time-dependent scalar boundary value problems that we have seen in Chapter 1 and Chapter 9. Transport is due to a *given velocity field*. Again, as in Section 1.6 and Section 9.2.1 heat conduction phenomena provide palpable model problems.

Contents

10.1 Heat conduction in a fluid	617
10.1.1 Modeling Fluid Flow	617
10.1.2 Heat Convection and Diffusion	619
10.1.3 Incompressible Fluids	622
10.1.4 Time-Dependent (Transient) Heat Flow in a Fluid	625
10.2 Stationary Convection-Diffusion Problems: Numerical Treatment	627
10.2.1 Singular Perturbation	629
10.2.2 Upwinding	632
10.3 Discretization of Time-Dependent (Transient) Convection-Diffusion IBVPs	648
10.3.1 Convection-Diffusion IBVPs: Method of Lines	649
10.3.2 Transport Equation	651
10.3.3 Lagrangian Split-Step Method	654
10.3.4 Semi-Lagrangian Method	664

§10.0.0.1 (Prerequisites for this chapter) To be able to absorb the contents of this chapter easily you should be familiar with the following subjects:

- The derivation and theory behind the variational formulation of 2nd-order elliptic BVP from Chapter 1,
- the basics of Galerkin discretization and its use for the finite-element discretization of elliptic BVPs, as covered in Section 2.2, Section 2.5, and Section 2.6,
- in particular the finite-element method in one spatial dimension from Section 2.3 and its interpretation as a finite-difference method, see Section 4.1.1,
- the method-of-lines policy for the full discretization of evolution problems as introduced in Section 9.2.4 and Section 9.2.7.

]

10.1 Heat conduction in a fluid

In the sequel denote by $\Omega \subset \mathbb{R}^d$ a bounded computational domain, $d = 1, 2, 3$, cf. the discussion in § 1.2.1.14.

To begin with we develop a mathematical model for stationary fluid flow, for instance, the steady streaming of water.

10.1.1 Modeling Fluid Flow



Video tutorial for 10.1.1: Modeling Fluid Flow: (13 minutes) [Download link](#), [tablet notes](#)

§10.1.1.1 (Flow fields and streamlines)

The key quantity is a **flow field**:

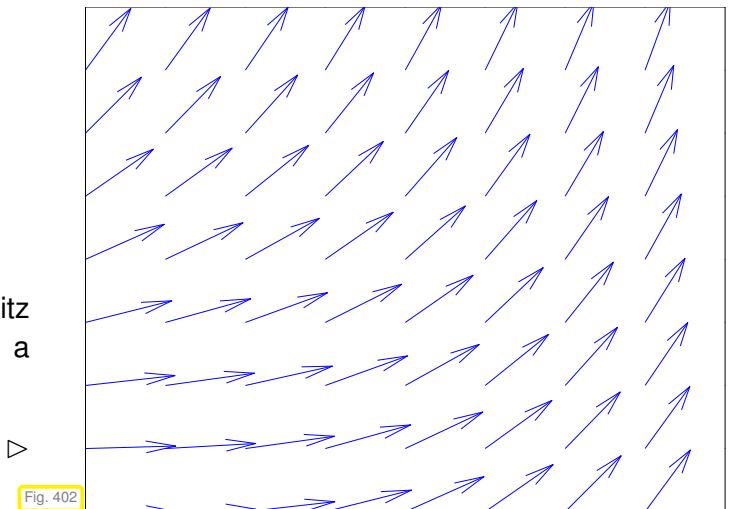
$$\mathbf{v} : \Omega \mapsto \mathbb{R}^d$$

A natural assumption is that

$$\mathbf{v} \text{ is } \text{continuous}, \mathbf{v} \in (C^0(\bar{\Omega}))^d$$

In fact, we will require that \mathbf{v} is uniformly Lipschitz continuous according to Ass. 9.2.5.4, but this is a mere technical requirement.

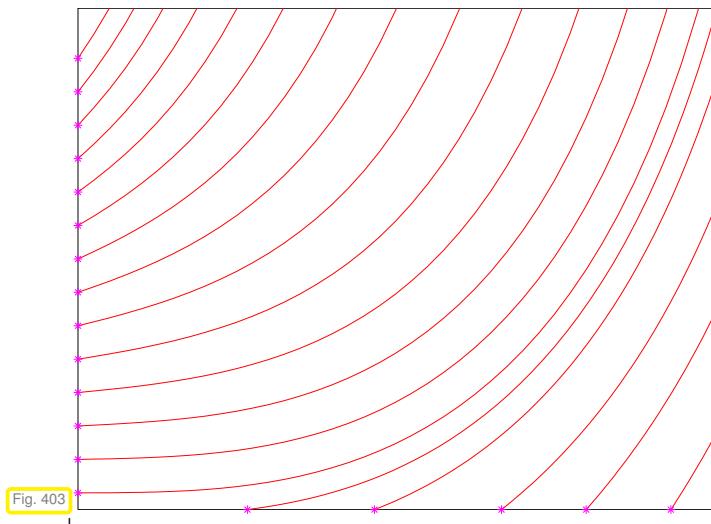
Visualization of a flow field as a vector field



Interpretation of point values of flow field:

$$\mathbf{v}(\mathbf{x}) \doteq \text{fluid velocity at point } \mathbf{x} \in \Omega$$

➤ \mathbf{v} corresponds to a **velocity field**!



Given a flow field $\mathbf{v} \in (C^0(\bar{\Omega}))^d$ we can consider the autonomous initial value problems

$$\dot{\mathbf{y}} = \mathbf{v}(\mathbf{y}) , \quad \mathbf{y}(0) = \mathbf{x}_0 . \quad (10.1.1.2)$$

Its solution $t \mapsto \mathbf{y}(t)$ defines the path travelled by a particle carried along by the fluid, a **particle trajectory**, also called a **streamline**.

- ▷ particle trajectories (streamlines) in flow field of Fig. 403.
(* \doteq initial particle positions)

§10.1.1.3 (Flow map) Via the ordinary differential equation (10.1.1.2) every flow field spawns an associated **evolution operator** according to § 9.2.5.8/[Hip19, ??]. Thus a flow field induces a transformation (mapping) of space! To explain this, let us temporarily for this § make the following assumption.

Assumption 10.1.1.4. No in/outflow

The flow does neither enter nor leave Ω .

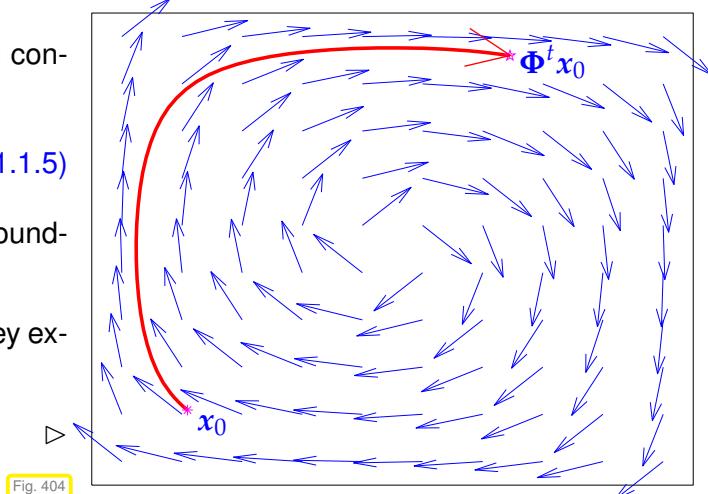
This assumption holds for fluid flow in a closed container. Mathematically, it can be expressed as

$$\mathbf{v}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \partial\Omega , \quad (10.1.1.5)$$

that is, the flow is always parallel to the wall (= boundary) of the “container” Ω .

- All particle trajectories stay inside Ω and they exist for all times!

Flow field satisfying (10.1.1.5)



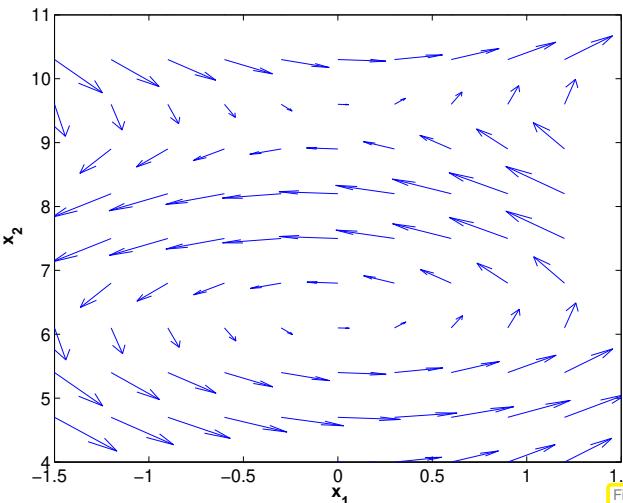
Now we fix some “time of interest” $t \in \mathbb{R}$ and, adopting the notations used when dealing with evolution operators in § 9.2.5.8, we write

$$\Phi^t : \begin{cases} \Omega & \mapsto \Omega \\ x_0 & \mapsto \mathbf{y}(t) \end{cases} , \quad t \mapsto \mathbf{y}(t) \text{ solution of IVP (10.1.1.2)} , \quad (10.1.1.6)$$

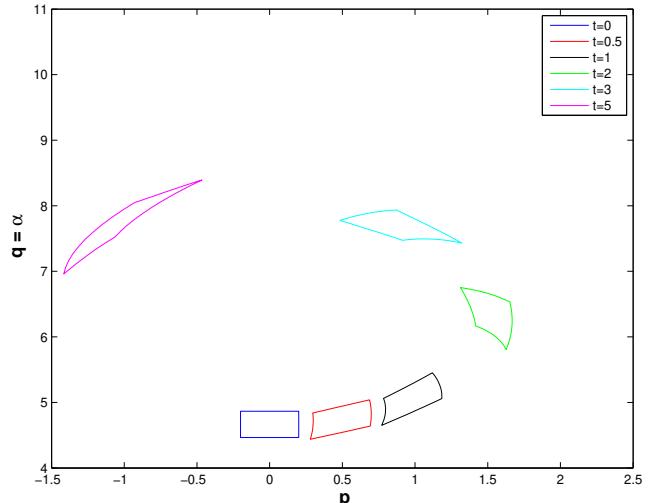
For fixed time t this is a well-defined mapping of Ω to itself. This one-parameter family of mappings $(\Phi^t)_{t \in \mathbb{R}}$ is known as **flow map**. It is a special instance of an evolution operator and inherits its basic properties like (9.2.5.11), which read in the current setting

$$\Phi^0 x_0 = x_0 , \quad \Phi^s(\Phi^t x_0) = \Phi^{s+t} x_0 \quad \forall x_0 \in \Omega , \quad \forall s, t \in \mathbb{R} . \quad (10.1.1.7)$$

Flow maps can be visualized through their effect on “control volumes”, which are subsets of Ω .



flow field $\mathbf{v} : \Omega \mapsto \mathbb{R}^2$



snapshots of $\Phi^t(V)$ for control volume V

$\Phi^\tau(V) \doteq$ volume occupied at time $t = \tau$ by particles that occupied $V \subset \Omega$ at time $t = 0$.

Review question(s) 10.1.1.8 (Modeling fluid flow)

(Q10.1.1.8.A) Assume that a flow map $\Phi : [0, T] \times \Omega \rightarrow \Omega$, $\Omega \subset \mathbb{R}^d$, is given, of which we know that it is induced by a continuous differentiable **stationary** velocity field $\mathbf{v} : \Omega \rightarrow \mathbb{R}^n$. Give a formula that recovers \mathbf{v} from Φ .

(Q10.1.1.8.B) Argue why a particle trajectory that starts inside the bounded computational domain $\Omega \subset \mathbb{R}^d$ can never leave Ω , if the driving (continuously differentiable) velocity field satisfies

$$\mathbf{v} \cdot \mathbf{n} = 0 \quad \text{on } \partial\Omega,$$

where $\mathbf{n} : \partial\Omega \rightarrow \mathbb{R}^d$ is the exterior unit normal vector field.

(Q10.1.1.8.C) Consider the unit disk domain $\Omega := \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\|_2 < 1\}$ and the velocity field

$$\mathbf{v} : \Omega \rightarrow \mathbb{R}^2, \quad \mathbf{v}(\mathbf{x}) = f(\|\mathbf{x}\|_2) \begin{bmatrix} -x_2 \\ x_1 \end{bmatrix}, \quad \mathbf{x} = [x_1 \ x_2] \in \Omega,$$

for a smooth positive function $f : [0, 1] \rightarrow \mathbb{R}^+$. Draw the induced streamlines.

△

10.1.2 Heat Convection and Diffusion



Video tutorial for Section 10.1.2: Heat Convection and Diffusion: (10 minutes) [Download link](#), [tablet notes](#)

We develop a mathematical description of heatflow in a stationary flow of a homogeneous fluid described by a *given flow/velocity field* $\mathbf{v} : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}^d$ assumed to be Lipschitz continuous. As in Section 1.6 the configuration space will a function space of temperature distributions on Ω . Hence the principal unknown in our PDE model will be a function $u : \Omega \mapsto \mathbb{R}$, which describes the stationary temperature distribution in a fluid moving according to a given stationary (= independent of time) flow field $\mathbf{v} : \Omega \mapsto \mathbb{R}^d$.

Our goal is to derive PDE-based continuum models in the form of boundary value problems whose solutions provide the temperature distribution u . The considerations will run parallel to those of Section 1.6, which deals with heat conduction in a solid.

§10.1.2.1 (Energy balance law) We adapt the considerations of Section 1.6 that led to the stationary heat equation. Write $\mathbf{j} : \Omega \rightarrow \mathbb{R}^d$ for the heat flux (\rightarrow § 1.6.0.1) and recall a fundamental physical principle governing heat flow:

Conservation of energy

$$\int_{\partial V} \mathbf{j} \cdot \mathbf{n} dS = \int_V f d\mathbf{x} \quad \text{for all "control volumes" } V. \quad (1.6.0.3)$$

↑
power flux through surface of V
↗
heat production inside V

We follow exactly the same steps as in § 1.6.0.7. From (1.6.0.3) by Gauss' theorem Thm. 1.5.2.4 we find

$$\int_V \operatorname{div} \mathbf{j}(\mathbf{x}) d\mathbf{x} = \int_V f(\mathbf{x}) d\mathbf{x} \quad \text{for all "control volumes" } V \subset \Omega. \quad (10.1.2.2)$$

Now appeal to another version of the fundamental lemma of the calculus of variations, see Lemma 1.5.3.4, this time employing piecewise constant test functions. This yields the *local form of energy conservation*:

$$\operatorname{div} \mathbf{j} = f \quad \text{in } \Omega. \quad (1.6.0.8)$$

↓

§10.1.2.3 (Heat flux law) In a moving fluid a power flux through a fixed surface is already caused by the sheer fluid flow carrying along stored thermal energy (heat), which, for simple materials, is roughly proportional to the temperature. This is reflected in a modified Fourier's law (1.6.0.5):

Fourier's law in moving fluid

$$\mathbf{j}(\mathbf{x}) = -\kappa \operatorname{grad} u(\mathbf{x}) + \mathbf{v}(\mathbf{x})\rho u(\mathbf{x}), \quad \mathbf{x} \in \Omega. \quad (10.1.2.4)$$

diffusive heat flux

(due to spatial variation of temperature)

convective heat flux

(due to fluid flow)

with $\kappa > 0 \doteq$ heat conductivity ($[\kappa] = 1 \frac{\text{W}}{\text{Km}}$),

$\rho > 0 \doteq$ volumetric heat capacity ($[\rho] = \frac{\text{J}}{\text{Km}^3}$),

both assumed to be constant (in contrast to the models of Section 1.6 and Section 9.2.1). \square

Remark 10.1.2.5. Unlike in (1.6.0.5), adding a constant offset to the temperature in (10.1.2.4) now has an impact on the heat flux. Therefore, (10.1.2.4) should be written as

$$\mathbf{j}(\mathbf{x}) = -\kappa \operatorname{grad} u(\mathbf{x}) + \mathbf{v}(\mathbf{x})(\rho u(\mathbf{x}) + U_*), \quad (10.1.2.6)$$

with $U_* > 0$, $[U_*] = \frac{\text{J}}{\text{m}^3}$, the internal energy of the fluid at temperature $u \equiv 0$. If u is the absolute temperature, we can set $U_* := 0$. \square

§10.1.2.7 (Convection-diffusion equation for temperature) Next, as in § 1.6.0.7, combine the conservation law (1.6.0.8) and the local flux law (10.1.2.4):

$$\operatorname{div} \mathbf{j} = f \quad + \quad \mathbf{j}(\mathbf{x}) = -\kappa \operatorname{grad} u(\mathbf{x}) + \mathbf{v}(\mathbf{x})\rho u(\mathbf{x})$$



$$-\operatorname{div}(\kappa \operatorname{grad} u) + \operatorname{div}(\rho \mathbf{v}(\mathbf{x})u) = f \quad \text{in } \Omega. \quad (10.1.2.8)$$

$\hat{=}$ Linear scalar **convection-diffusion equation** (CDE) (for unknown temperature $u = u(\mathbf{x})$)

$$\begin{array}{ccc} -\operatorname{div}(\kappa \operatorname{grad} u) & + & \operatorname{div}(\rho \mathbf{v}(\mathbf{x})u) = f \\ \downarrow & & \downarrow \\ \text{Terminology :} & \text{diffusive term} & \text{convective term} \\ & (2\text{nd-order}) & (1\text{st-order}) \end{array}$$

The partial differential equation (10.1.2.8) is still belongs to the class of scalar linear second-order elliptic PDEs. \square

§10.1.2.9 (Boundary conditions) The 2nd-order elliptic PDE (10.1.2.8) has to be supplemented with exactly one **boundary condition** on any part of $\partial\Omega$, see Sect. 1.7, Ex. 1.7.0.10. This can be any of the (“elliptic”) boundary conditions introduced in Sect. 1.7:

- ◆ Dirichlet boundary conditions: $u = g \in C^0(\partial\Omega)$ on $\partial\Omega$ (fixed surface temperature),
- ◆ Neumann boundary conditions: $\mathbf{j} \cdot \mathbf{n} = -h$ on $\partial\Omega$ (fixed heat flux, \mathbf{j} as in (10.1.2.4)),

- ◆ (non-linear) radiation boundary conditions: $\mathbf{j} \cdot \mathbf{n} = \Psi(\mathbf{u})$ on $\partial\Omega$ (temperature dependent heat flux, radiative heat flux).

In fact, it is worth remembering the following guideline:

Suitable boundary conditions fitting a PDE are determined by the highest-order term.

In the case of the convection-diffusion equation the highest-order term is the second-order diffusion operator. \square

Review question(s) 10.1.2.10 (Heat conduction in a fluid)

(Q10.1.2.10.A) The heat flux in a fluid moving with the stationary velocity \mathbf{v} is given by

$$\mathbf{j}(\mathbf{x}) = -\kappa \operatorname{grad} u(\mathbf{x}) + \mathbf{v}(\mathbf{x})\rho u(\mathbf{x}), \quad \mathbf{x} \in \Omega. \quad (10.1.2.4)$$

- Explain the meaning of κ , ρ , and u .
- Why can there be a heat flux even for $u \equiv \text{const}$?

(Q10.1.2.10.B) Write down the scalar linear convection-diffusion equation

$$-\operatorname{div}(\kappa \operatorname{grad} u) + \operatorname{div}(\rho \mathbf{v}(\mathbf{x})u) = f \quad \text{in } \Omega \subset \mathbb{R}^d, \quad (10.1.2.8)$$

for the special case $d = 1$, $\Omega =]a, b[$, $a < b$.

\triangle

10.1.3 Incompressible Fluids



Video tutorial for Section 10.1.3: Incompressible Fluids: (20 minutes) [Download link](#), [tablet notes](#)

For the sake of simplicity we will mainly consider **incompressible fluids**.

Definition 10.1.3.1. Incompressible flow field

A fluid flow is called **incompressible**, if the associated flow map Φ^t is **volume preserving**,

$$|\Phi^t(V)| = |\Phi^0(V)| = |V| \quad \text{for all sufficiently small } t > 0, \text{ and for all control volumes } V.$$

§10.1.3.2 (A criterion for incompressibility) Can incompressibility be read off the velocity field \mathbf{v} of the flow?

To investigate this issue, again assume the “no flow through the boundary condition” of Ass. 10.1.1.4 and recall that the flow map Φ^t from (10.1.1.6) satisfies

$$\frac{\partial}{\partial t} \Phi(t, \mathbf{x}) = \mathbf{v}(\Phi(t, \mathbf{x})), \quad \mathbf{x} \in \Omega, t > 0. \quad (10.1.3.3)$$

Here, in order to make clear the dependence on independent variables, time occurs as an argument of Φ in brackets, on par with \mathbf{x} .

Next, formal differentiation w.r.t. \mathbf{x} and change of order of differentiation yields a differential equation for the Jacobian $D_{\mathbf{x}}\Phi^t$,

$$(10.1.3.3) \quad \Rightarrow \quad \frac{\partial}{\partial t}(D_{\mathbf{x}}\Phi)(t, \mathbf{x}) = D\mathbf{v}(\Phi(t, \mathbf{x}))(D_{\mathbf{x}}\Phi)(t, \mathbf{x}) . \quad (10.1.3.4)$$

↑
Jacobian of $\Phi \in \mathbb{R}^{d,d}$

↑
Jacobian of $\mathbf{v} \in \mathbb{R}^{d,d}$

Second strand of thought: apply transformation formula for integrals (0.3.2.32), [Str09, Satz 8.5.2]: for fixed $t > 0$

$$|\Phi(t, V)| = \int_{\Phi(t, V)} 1 d\mathbf{x} = \int_V |\det(D_{\mathbf{x}}\Phi)(t, \hat{\mathbf{x}})| d\hat{\mathbf{x}} . \quad (10.1.3.5)$$

Volume preservation by the flow map is equivalent to

$$t \mapsto |\Phi(t, V)| = \text{const.} \iff \frac{d}{dt} |\Phi(t, V)| = 0 ,$$

for any control volume $V \subset \Omega$.

$$(10.1.3.5) \Rightarrow \frac{d}{dt} |\Phi(t, V)| = \int_V \frac{\partial}{\partial t} |\det(D_{\mathbf{x}}\Phi)(t, \hat{\mathbf{x}})| d\hat{\mathbf{x}} . \quad (10.1.3.6)$$

The next theorem is a “chain rule for determinants”.

Theorem 10.1.3.7. Differentiation formula for determinants

Let $\mathbf{S} : I \subset \mathbb{R} \mapsto \mathbb{R}^{n,n}$ be a smooth matrix-valued function. If $\mathbf{S}(t_0)$ is regular for some $t_0 \in I$, then

$$\frac{d}{dt} (\det \circ \mathbf{S})(t_0) = \det(\mathbf{S}(t_0)) \operatorname{tr}\left(\frac{d\mathbf{S}}{dt}(t_0)\mathbf{S}^{-1}(t_0)\right) ,$$

where $\det : \mathbb{R}^{n,n} \rightarrow \mathbb{R}$ is the matrix determinant and tr stands for the trace of a matrix.

Proof. We start the rather formal “proof” from the Taylor expansion of $t \mapsto \mathbf{S}(t)$ around $t = t_0$:

$$\mathbf{S}(t_0 + \tau) = \mathbf{S}(t_0) + \tau \dot{\mathbf{S}}(t_0) + O(\tau^2) \quad \text{for } \tau \rightarrow 0 , \quad \dot{\mathbf{S}}(t_0) := \frac{d\mathbf{S}}{dt}(t_0) .$$

As $\mathbf{S}(t_0)$ is invertible, we can thus recast

$$\det(\mathbf{S}(t_0 + \tau)) = \det(\mathbf{S}(t_0)) \det(\mathbf{I} + \tau \mathbf{S}(t_0)^{-1} \dot{\mathbf{S}}(t_0) + O(\tau^2)) .$$

For a fixed matrix $\mathbf{M} = [\mathbf{m}_1, \dots, \mathbf{m}_n] \in \mathbb{R}^{n,n}$ with columns $\mathbf{m}_j \in \mathbb{R}^n$ we now study the expression

$$\begin{aligned} \det(\mathbf{I} + \tau \mathbf{M} + O(\tau^2)) &= \det(\mathbf{e}_1 + \tau \mathbf{m}_1, \dots, \mathbf{e}_n + \tau \mathbf{m}_n) + O(\tau^2) \\ &\stackrel{(*)}{=} 1 + \sum_{j=1}^n \tau \det(\mathbf{e}_1, \dots, \mathbf{e}_{j-1}, \mathbf{m}_j, \mathbf{e}_{j+1}, \dots, \mathbf{e}_n) + O(\tau^2) \\ &= 1 + \tau \sum_{j=1}^n (\mathbf{M})_{j,j} + O(\tau^2) \quad \text{for } \tau \rightarrow 0 . \end{aligned}$$

Here, in step (*), we exploited that the determinant is linear in each argument. \square

We apply the formula of the theorem with $\mathbf{S}(t) := D_x \Phi(t, \hat{x})$, $\hat{x} \in \Omega$:

$$\begin{aligned} \blacktriangleright \quad \frac{\partial}{\partial t} \det(D_x \Phi)(t, \hat{x}) &= \det((D_x \Phi)(t, \hat{x})) \operatorname{tr}\left(\frac{\partial}{\partial t} D_x \Phi(t, \hat{x}) D_x \Phi(t, \hat{x})^{-1}\right) \\ &\stackrel{(10.1.3.4)}{=} \det(D_x \Phi)(t, \hat{x}) \operatorname{tr}(D\mathbf{v}(\Phi(t, \hat{x}))) \underbrace{(D_x \Phi)(t, \hat{x})(D_x \Phi)^{-1}(t, \hat{x})}_{=\mathbf{I}} \\ &= \det(D_x \Phi)(t, \hat{x}) \operatorname{div} \mathbf{v}(\Phi(t, \hat{x})), \end{aligned}$$

because the divergence of a vector field \mathbf{v} is just the trace of its Jacobian $D\mathbf{v}$! From (10.1.1.7) we know that for small $t > 0$ the Jacobian $D_x \Phi(t, \hat{x})$ will be close to \mathbf{I} and, therefore, $\det(D_x \Phi)(t, \hat{x}) \neq 0$ for $t \approx 0$. Thus, for small $t > 0$ we conclude

$$\frac{d}{dt} |\Phi(t, V)| = 0 \Leftrightarrow \operatorname{div} \mathbf{v}(\Phi(t, \hat{x})) = 0 \quad \forall \hat{x} \in V.$$

Since this is to hold for **any** control volume V , the final equivalence is

$$\frac{d}{dt} |\Phi(t, V)| = 0 \quad \forall \text{control volumes } V \Leftrightarrow \operatorname{div} \mathbf{v} = 0 \quad \text{in } \Omega.$$

Theorem 10.1.3.8. Divergence-free velocity fields for incompressible flows

A stationary fluid flow in $\Omega \subset \mathbb{R}^d$ is incompressible (\rightarrow Def. 10.1.3.1), if and only if its associated velocity field $\mathbf{v} = [v_1, \dots, v_d]^\top : \Omega \rightarrow \mathbb{R}^d$ satisfies $\operatorname{div} \mathbf{v} = \sum_{j=1}^d \frac{\partial v_j}{\partial x_j} = 0$ everywhere in Ω .

§10.1.3.9 (Stationary heat equation in an incompressible fluid) Now let us assume the incompressibility condition $\operatorname{div} \mathbf{v} = 0$. Note that for $d = 1$ this boils down to $\frac{dv}{dx} = 0$ and implies $v = \text{const}$. Let us see how the linear scalar convection-diffusion equation

$$-\operatorname{div}(\kappa \operatorname{grad} u) + \operatorname{div}(\rho \mathbf{v}(x) u) = f \quad \text{in } \Omega, \quad (10.1.2.8)$$

can be transformed in this case.

We can use the product rule in higher dimensions of Lemma 1.5.2.1:

$$\operatorname{div}(\rho \mathbf{v} u) \stackrel{\text{Lemma 1.5.2.1}}{=} \rho(u \operatorname{div} \mathbf{v} + \mathbf{v} \cdot \operatorname{grad} u) \stackrel{\operatorname{div} \mathbf{v} = 0}{=} \rho \mathbf{v} \cdot \operatorname{grad} u. \quad (10.1.3.10)$$

Therefore, we can rewrite the scalar convection-diffusion equation (10.1.2.8) for an incompressible flow field as

$$-\operatorname{div}(\kappa \operatorname{grad} u) + \operatorname{div}(\rho \mathbf{v}(x) u) = f \quad \text{in } \Omega$$

 ← use $\operatorname{div} \mathbf{v} = 0$

$$-\kappa \Delta u + \rho \mathbf{v} \cdot \operatorname{grad} u = f \quad \text{in } \Omega. \quad (10.1.3.11)$$

Supplement 10.1.3.12 (Maximum principle for convection-diffusion equations) When carried along by the flow of an incompressible fluid, the temperature cannot be increased by local compression, the

effect that you can witness when pumping air. Hence, only sources/sinks can lead to local extrema of the temperature.

Now recall the discussion of the physical intuition behind the **maximum principle** of Thm. 3.7.1.2. These considerations still apply to stationary heat flow in a moving incompressible fluid.

Theorem 10.1.3.13. Maximum principle for scalar 2nd-order convection diffusion equations
 \rightarrow [Eva98, 6.4.1, Thm. I]

Let $\mathbf{v} : \Omega \mapsto \mathbb{R}^d$ be a continuously differentiable vector field and $u \in C^0(\bar{\Omega}) \cap C^2(\Omega)$. Then there holds the **maximum principle**

$$\begin{aligned}-\Delta u + \mathbf{v} \cdot \nabla u &\geq 0 \implies \min_{x \in \partial\Omega} u(x) = \min_{x \in \Omega} u(x), \\-\Delta u + \mathbf{v} \cdot \nabla u &\leq 0 \implies \max_{x \in \partial\Omega} u(x) = \max_{x \in \Omega} u(x).\end{aligned}$$

Review question(s) 10.1.3.14 (Incompressible fluids)

(Q10.1.3.14.A) Again, consider the unit disk domain $\Omega := \{x \in \mathbb{R}^2 : \|x\|_2 < 1\}$ and the velocity field

$$\mathbf{v} : \Omega \rightarrow \mathbb{R}^2, \quad \mathbf{v}(x) = f(\|x\|_2) \begin{bmatrix} -x_2 \\ x_1 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \Omega,$$

for a smooth positive function $f : [0, 1] \rightarrow \mathbb{R}^+$. For what f does \mathbf{v} satisfy the **incompressibility condition** $\operatorname{div} \mathbf{v} = 0$?

Hint: Use $\operatorname{div} \mathbf{v} = \frac{\partial v_1}{\partial x_1} + \frac{\partial v_2}{\partial x_2}$ and

$$\nabla \cdot \{x \mapsto \|x\|_2\} = \frac{x}{\|x\|_2}, \quad x \neq 0.$$

(Q10.1.3.14.B) Show that for an **incompressible, homogeneous** (none of the material coefficients depends on x) fluid whose temperature is constant in Ω , the heat flowing into Ω is equal to the heat flowing out of Ω .

△

10.1.4 Time-Dependent (Transient) Heat Flow in a Fluid



Video tutorial for Section 10.1.4: Time-Dependent (Transient) Heat Flow in a Fluid: (8 minutes) [Download link](#), [tablet notes](#)

In Section 9.2.1 we generalized the laws of stationary heat conduction derived in Section 1.6 to time-dependent temperature distributions $u = u(x, t)$ sought on a space-time cylinder $\tilde{\Omega} := \Omega \times]0, T[$. The same ideas apply to heat conduction in a fluid, whose motion is given by a time-dependent velocity field $\mathbf{v} = \mathbf{v}(x, t)$.

- Start from energy balance law

Conservation of energy:

$$\frac{d}{dt} \int_V \rho u \, dx + \int_{\partial V} \mathbf{j} \cdot \mathbf{n} \, dS = \int_V f \, dx \quad \text{for all "control volumes" } V. \quad (9.2.1.3)$$

↑ ↗ ↘
energy stored in V power flux through ∂V heat generation in V

and convert it into local form

$$\frac{\partial}{\partial t} (\rho u)(\mathbf{x}, t) + (\operatorname{div}_x \mathbf{j})(\mathbf{x}, t) = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega}. \quad (9.2.1.5)$$

- Combine it with the extended Fourier's law

$$\mathbf{j}(\mathbf{x}, t) = -\kappa \operatorname{grad} u(\mathbf{x}, t) + \mathbf{v}(\mathbf{x}, t) \rho u(\mathbf{x}, t), \quad \mathbf{x} \in \Omega. \quad (10.1.2.4)$$

Note that the velocity field is allowed to change with time: $\mathbf{v} = \mathbf{v}(\mathbf{x}, t)$.

$$\frac{\partial}{\partial t} (\rho u) - \operatorname{div}(\kappa \operatorname{grad} u) + \operatorname{div}(\rho \mathbf{v}(\mathbf{x}, t) u) = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[. \quad (10.1.4.1)$$

For details and notations refer to Section 9.2.1.

This PDE has to be supplemented with

- boundary conditions (as in the stationary case, see Sect. 1.7),
- the initial condition $u(\mathbf{x}, 0) = u_0(\mathbf{x})$, $\mathbf{x} \in \Omega$, (same as for pure diffusion, see Sect. 9.2.1).

Under the assumption $\operatorname{div}_x \mathbf{v}(\mathbf{x}, t) = 0$ of incompressibility (\rightarrow Def. 10.1.3.1 and Thm. 10.1.3.8) and in the case of constant (in space) coefficients (10.1.4.1) is equivalent to, cf. (10.1.3.10),

$$\frac{\partial}{\partial t} (\rho u) - \kappa \Delta u + \rho \mathbf{v}(\mathbf{x}, t) \cdot \operatorname{grad} u = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[. \quad (10.1.4.2)$$

Remark 10.1.4.3 (Conversion into non-dimensional form by scaling) \rightarrow Rem. 1.2.1.25 Let us elaborate how to cast (10.1.4.2) into non-dimensional form, a procedure known as **scaling**. The first step consists of fixing **reference quantities**:

- reference length l_0 , $[l_0] = 1\text{m}$,
- reference time span t_0 , $[t_0] = 1\text{s}$,
- reference temperature T_0 , $[T_0] = 1\text{K}$,
- reference volumetric heat capacity ρ_0 , $[\rho_0] = \frac{\text{J}}{\text{Km}^3}$

Here we choose l_0 and t_0 such that $v_{\max} = \frac{l_0}{t_0}$, $v_{\max} := \max_{\mathbf{x}, t} \|\mathbf{v}(\mathbf{x}, t)\|$.

A hint on how many reference quantities are at our disposal is offered by considering the number of different basic SI units relevant for the model. Here those are **1K**, **1m**, **1s**, **1J**.

Then we introduce the dimensionless temperature

$$\tilde{u}(\xi, \tau) := u(l_0 \xi, t_0 \tau), \quad \xi \in \mathbb{R}^3, \quad \tau \in \mathbb{R} \quad (\text{"pure numbers"}).$$

By the chain rule we obtain

$$\begin{aligned}\mathbf{grad}_\xi \tilde{u} &= \frac{l_0}{u_0} \mathbf{grad}_x u(x, t), \\ \Delta_\xi \tilde{u} &= \frac{l_0^2}{u_0} \Delta_x u(x, t), \\ \frac{\partial}{\partial \tau} \tilde{u} &= \frac{t_0}{u_0} \frac{\partial}{\partial t} u(x, t).\end{aligned}$$

These expressions can be inserted into (10.1.4.2). In the case of constant coefficients ρ , κ , and $\rho_0 := \rho$, after division by ρ_0 and u_0 we arrive at

$$\blacktriangleright \quad \frac{\partial}{\partial \tau} \tilde{u} - \frac{t_0 \kappa}{l_0^2 \rho_0} \Delta_\xi \tilde{u} + \frac{\mathbf{v}}{v_{\max}} \mathbf{grad}_\xi \tilde{u} = \frac{t_0}{l_0 \rho_0} f.$$

Check that $\frac{t_0 \kappa}{l_0^2 \rho_0}$ and $\frac{t_0}{l_0 \rho_0} f$ really are dimensionless! □

Review question(s) 10.1.4.4 (Transient heat flow in a fluid)

(Q10.1.4.4.A) Write down the scalar linear transient heat equation describing the evolution of a temperature distribution in a moving *incompressible, homogeneous* fluid in one spatial dimension, $d = 1$.

△

10.2 Stationary Convection-Diffusion Problems: Numerical Treatment

As model problems we use boundary value problems for the PDEs (10.1.3.11) modelling stationary heat flow in an incompressible fluid with prescribed temperature at “walls of the container” (\leftrightarrow Dirichlet boundary conditions):

$$-\kappa \Delta u + \rho \mathbf{v}(x) \cdot \mathbf{grad} u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega.$$

We first perform **scaling** equivalent to a choice of physical units. This makes the equation non-dimensional by fixing “reference length”, “reference time interval”, “reference temperature”, “reference power”, see Rem. 10.1.4.3 on page 626. As elaborated in that remark, scaling produces the following boundary value problem for non-dimensional quantities, where $\|\mathbf{v}\|_{L^\infty(\Omega)} \leq 1$, and $\epsilon := \frac{t_0 \kappa}{l_0^2 \rho_0}$, see Rem. 10.1.4.3 for the choice of reference quantities t_0, l_0, ρ_0 :

$$-\epsilon \Delta u + \mathbf{v}(x) \cdot \mathbf{grad} u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega. \quad (10.2.0.1)$$



Here, $\epsilon > 0$, $\|\mathbf{v}\|_{L^\infty(\Omega)} = 1$, and $\operatorname{div} \mathbf{v} = 0 \rightarrow$ incompressible fluid, see Def. 10.1.3.1.

§10.2.0.2 (Variational formulation for convection-diffusion BVP) The standard “4-step approach” of Sect. 1.8 can be directly applied to BVP (10.2.0.1) with one new twist:

Do not use integration by parts (Green's formula, Thm. 1.5.2.7) on convective terms!

► variational formulation for BVP (10.2.0.1):

$$u \in H_0^1(\Omega): \underbrace{\epsilon \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} w \, dx + \int_{\Omega} (\mathbf{v} \cdot \mathbf{grad} u) w \, dx}_{\text{bilinear form } \mathbf{a}(u,w)} = \underbrace{\int_{\Omega} f(x) w(x) \, dx}_{\text{linear form } \ell(w)} \quad \forall w \in H_0^1(\Omega). \quad (10.2.0.3)$$

This is a **linear variational problem** of the form (1.4.1.7), see Section 1.4.1.

Obviously, the bilinear form \mathbf{a} is **not** symmetric, see (1.2.3.4).

► \mathbf{a} does not induce an energy norm (\rightarrow Def. 1.2.3.35)

As replacement for the energy norm we use the $H^1(\Omega)$ -(semi)norm (\rightarrow Def. 1.3.4.3)

In this case we have to make sure that \mathbf{a} fits the chosen norm in the sense that it satisfies

$$\exists C > 0: |\mathbf{a}(u, v)| \leq C |u|_{H^1(\Omega)} |v|_{H^1(\Omega)} \quad \forall u, v \in H_0^1(\Omega). \quad (10.2.0.4)$$

This property is phrased as follows: (10.2.0.4) $\hat{=}$ \mathbf{a} is **continuous** on $H^1(\Omega)$, cf. (2.2.0.3).

It is not difficult to establish continuity for the concrete bilinear form $[\mathbf{a}]$ from (10.2.0.3). By the Cauchy-Schwarz inequality for integrals (1.3.4.15) we find for all $u, v \in H_0^1(\Omega)$

$$|\mathbf{a}(u, v)| \leq \|v\|_{L^\infty(\Omega)} |u|_{H^1(\Omega)} \|v\|_{L^2(\Omega)} \stackrel{\text{Thm. 1.3.4.17}}{\leq} \text{diam}(\Omega) \|v\|_{L^\infty(\Omega)} |u|_{H^1(\Omega)} |v|_{H^1(\Omega)},$$

which confirms (10.2.0.4). Next we report a surprising observation: the bilinear form \mathbf{a} from (10.2.0.3) is **positive definite** (\rightarrow Def. 1.2.3.27)! This can be verified by integration by parts

$$\begin{aligned} \int_{\Omega} (\mathbf{v} \cdot \mathbf{grad} u) u \, dx &= \int_{\Omega} (\mathbf{v} u) \cdot \mathbf{grad} u \, dx \\ &\stackrel{\text{Green's formula}}{=} - \int_{\Omega} \text{div}(\mathbf{v} u) u \, dx + \int_{\partial\Omega} \underbrace{u^2}_{=0} \mathbf{v} \cdot \mathbf{n} \, dS \\ &\stackrel{(1.5.2.2) \& \text{div } \mathbf{v} = 0}{=} - \int_{\Omega} (\mathbf{v} \cdot \mathbf{grad} u) u \, dx. \end{aligned}$$

$$\Rightarrow \mathbf{a}(u, u) = \epsilon \int_{\Omega} \|\mathbf{grad} u\|^2 \, dx > 0 \quad \forall u \in H_0^1(\Omega) \setminus \{0\}. \quad (10.2.0.5)$$

From this and (10.2.0.4) we can conclude existence and uniqueness of solutions of the BVP (10.2.0.1) in the Sobolev space $H_0^1(\Omega)$. This is a consequence of a generalization of the theory presented in Rem. 1.3.3.9 known as **inf-sup condition**, see [Bra07, Thm. 3.6]. \square

Review question(s) 10.2.0.6 (Stationary convection-diffusion problems)

(Q10.2.0.6.A) State a weak formulation of the boundary value problem

$$-\Delta u + \text{div}(\mathbf{v}(x)u) = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega.$$

(Q10.2.0.6.B) For a bounded computational domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, state the boundary value problem in strong (PDE) form induced by the variational problem

$$u \in H^1(\Omega): \underbrace{\epsilon \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} w \, dx + \int_{\Omega} (\mathbf{v} \cdot \mathbf{grad} u) w \, dx}_{\text{bilinear form } \mathbf{a}(u, w)} = \underbrace{\int_{\Omega} f(x) w(x) \, dx}_{\text{linear form } \ell(w)} \quad \forall w \in H^1(\Omega).$$

Here, $f \in C^0(\overline{\Omega})$ and $\mathbf{v} \in (C^1(\overline{\Omega}))^d$.

Hint. Through the customary two-step approach first extract the PDE by testing with compactly supported smooth functions and then the boundary conditions by testing with general w .

(Q10.2.0.6.C) Let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, be a bounded computational domain. For $f \in C^0(\overline{\Omega})$ and $\mathbf{v} \in (C^1(\overline{\Omega}))^d$ find the boundary value problem in strong (PDE) form induced by the variational problem

$$u \in H_0^1(\Omega): \epsilon \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} w \, dx + \int_{\Omega} u(\mathbf{v} \cdot \mathbf{grad} w) \, dx = \int_{\Omega} f(x) w(x) \, dx \quad \forall w \in H_0^1(\Omega).$$

Hint. Here we face essential boundary conditions.

△

10.2.1 Singular Perturbation



Video tutorial for Section 10.2.1: Singular Perturbation: (32 minutes) [Download link](#), [tablet notes](#)

We imagine a “fast-moving” incompressible fluid such that heat transport by convection dominates heat diffusion. This amounts to the situation $\epsilon \ll \|\mathbf{v}\|_{L^\infty(\Omega)}$ in the dimensionless stationary heat equation

$$-\epsilon \Delta u + \mathbf{v}(x) \cdot \mathbf{grad} u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega. \quad (10.2.0.1)$$

EXAMPLE 10.2.1.1 (1D convection-diffusion boundary value problem) We study the solution of (10.2.0.1) in one spatial dimension, $d = 1$.

$$-\epsilon \frac{d^2 u_\epsilon}{dx^2} + \frac{du_\epsilon}{dx} = 1 \quad \text{in } \Omega, \quad (10.2.1.2)$$

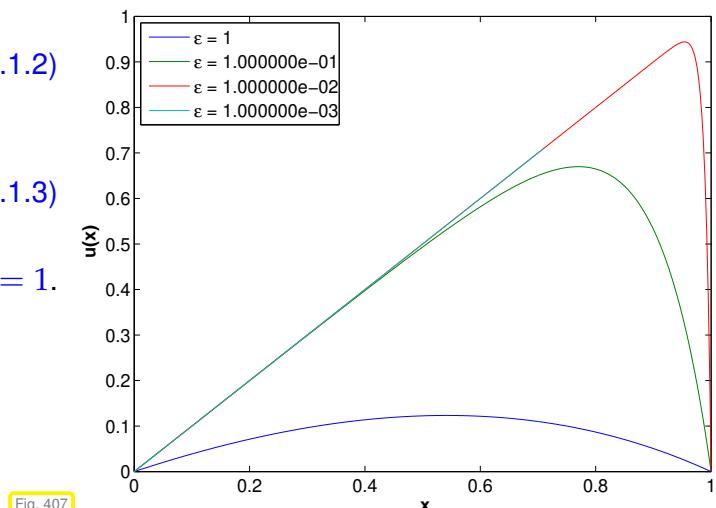
$$u_\epsilon(0) = 0, \quad u_\epsilon(1) = 0,$$

► $u_\epsilon(x) = x + \frac{\exp(-x/\epsilon) - 1}{1 - \exp(-1/\epsilon)}. \quad (10.2.1.3)$

For $\epsilon \ll 1$ we encounter a **boundary layer** at $x = 1$.

Pointwise limit:

$$\lim_{\epsilon \rightarrow 0} u_\epsilon(x) \rightarrow x \quad \forall 0 < x < 1.$$



Obviously, the pointwise limit $u_0(x) = x$ for $\epsilon \rightarrow 0$ solves the differential equation $\frac{du_0}{dx} = 1$ (“limit equation”), which can be obtained from (10.2.1.2) by simply setting $\epsilon := 0$.

“Limit problem” for (10.2.0.1) and fast flow: ignore diffusion \Rightarrow set $\epsilon = 0$:

$$(10.2.0.1) \quad \xrightarrow{\epsilon=0} \quad \mathbf{v}(x) \cdot \nabla u = f(x) \quad \text{in } \Omega . \quad (10.2.1.4)$$

Case $d = 1$ ($\Omega =]0, 1[$, $v = 1$)

$$(10.2.1.4) \quad \xrightarrow{d=1} \quad \frac{du}{dx}(x) = f(x) \quad \Rightarrow \quad u(x) = \int f \, dx + C . \quad (10.2.1.5)$$

What about this constant C ?

If $v = 1 \leftrightarrow$ fluid flows “from left to right”, so we should integrate the source from 0 to x :

$$u(x) = u(0) + \int_0^x f(s) \, ds = \int_0^x f(s) \, ds , \quad (10.2.1.6)$$

because $u(0) = 0$ by the boundary condition $u = 0$ on $\partial\Omega$. If $v = -1$ we start the integration at $x = 1$. Note that this makes the maximum principle of Thm. 10.1.3.13 hold.

§10.2.1.7 (The method of characteristics) For $d > 1$ we can solve (10.2.1.4) by the method of characteristics:

To motivate it, be aware that (10.2.1.4) describes the *pure transport* of a temperature distribution in the velocity field \mathbf{v} , that is, the heat/temperature is just carried along particle trajectories and changes only under the influence of heat sources/sinks along that trajectory.

Denote by u the solution of (10.2.1.4) and recall the differential equation (10.1.1.2) for a particle trajectory,

$$\text{streamline ODE:} \quad \frac{d\mathbf{y}}{dt}(t) = \mathbf{v}(\mathbf{y}(t)) , \quad \mathbf{y}(0) = \mathbf{x}_0 . \quad (10.1.1.2)$$

$$\xrightarrow{} \frac{d}{dt} u(\mathbf{y}(t)) = \nabla u(\mathbf{y}(t)) \cdot \frac{d}{dt} \mathbf{y}(t) = \nabla u \cdot \mathbf{v}(\mathbf{y}(t)) \stackrel{(10.2.1.4)}{=} f(\mathbf{y}(t)) .$$

\Rightarrow Compute $u(\mathbf{y}(t))$ by integrating source f along particle trajectory/streamline!

$$u(\mathbf{y}(t)) = u(\mathbf{x}_0) + \int_0^t f(\mathbf{y}(s)) \, ds \quad (10.2.1.8)$$

A rule for choosing the “starting point” \mathbf{x}_0 on the boundary of Ω will be given below in § 10.2.1.10. \square

Return to case $d = 1$. In general, a solution $u(x)$ from (10.2.1.5) will **not** satisfy the boundary condition $u(1) = 0$! Also for $u(x)$ from (10.2.1.8) the homogeneous boundary conditions may be violated where the particle trajectory leaves Ω !

In the limit case $\epsilon = 0$ not all boundary conditions of (10.2.0.1) can be satisfied.

Notion 10.2.1.9. Singularly perturbed problem

A boundary value problem depending on a parameter $\epsilon \approx \epsilon_0$ is called **singularly perturbed**, if the limit problem for $\epsilon \rightarrow \epsilon_0$ is not compatible with the boundary conditions.

Especially in the case of 2nd-order elliptic boundary value problems:

Singular perturbation = 1st-order terms become dominant for $\epsilon \rightarrow \epsilon_0$

In mathematical terms, singular perturbation for boundary values for PDEs is defined as a *change of type* of the PDE for $\epsilon = 0$: in the case of (10.2.0.1) the type changes from elliptic to hyperbolic, see Suppl. 1.1.0.1.

§10.2.1.10 (Respecting the flow of information) How can we pick the subsets of boundary data which remain relevant for the pure transport limit problem?

The direction of the flow field $\mathbf{v} : \Omega \rightarrow \mathbb{R}^d$ determines the direction of flow of information

This is clear in the context of heat conduction in a moving fluid: the temperature of fluid particles entering the domain Ω can be set, but not the temperature of those leaving Ω . We rephrase this as a general rule.

For the pure transport problem $\mathbf{v} \cdot \operatorname{grad} u = f$ Dirichlet boundary conditions can be imposed only on the **inflow boundary** part

$$\Gamma_{\text{in}} := \{x \in \partial\Omega : \mathbf{v}(x) \cdot \mathbf{n}(x) < 0\}. \quad (10.2.1.11)$$

but not on its complement in $\partial\Omega$, the **outflow boundary** part

$$\Gamma_{\text{out}} := \{x \in \partial\Omega : \mathbf{v}(x) \cdot \mathbf{n}(x) > 0\}. \quad (10.2.1.12)$$

We point out that this rule has already been applied in (10.2.1.6) for $d = 1$.

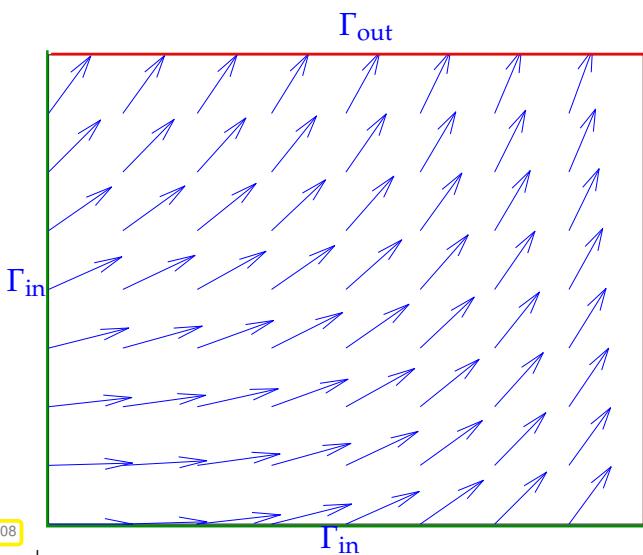


Illustration for $d = 2$:

◀ Inflow and outflow boundary parts Γ_{in} and Γ_{out} for a particular flow field.

Γ_{out}
 $\epsilon > 0$: Dirichlet boundary conditions can be imposed *everywhere* on $\partial\Omega$

$\epsilon = 0$: Dirichlet boundary conditions can be imposed *only on the inflow boundary* Γ_{in} .

Remark 10.2.1.13 (Closed streamlines)

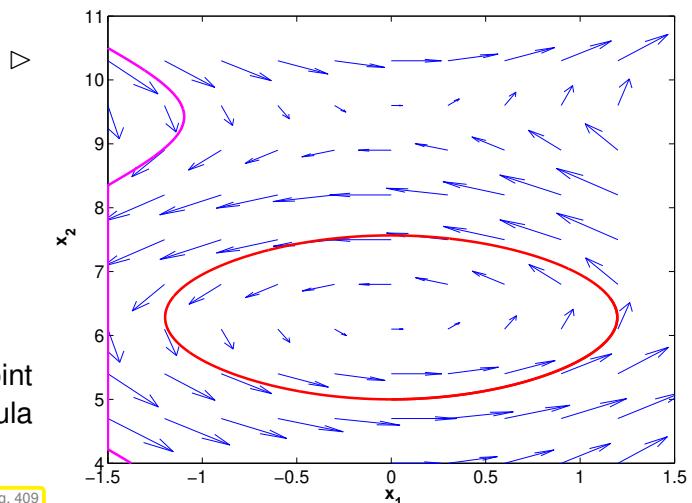
A special situation

→ velocity field

—: Streamline connecting Γ_{in} and Γ_{out}

—: Closed streamline
(recirculating flow)

On a closed streamline we cannot find a point $x_0 \in \Gamma_{\text{in}}$, which renders the solution formula (10.2.1.8) meaningless.



In the case of closed streamlines the stationary pure transport problem fails to have a unique solution: on a closed streamline and for $f \equiv 0$ the solution u can attain “any” value, because there is no boundary value to fix u . □

Review question(s) 10.2.1.14.

(Q10.2.1.14.A) What is meant by the statement that a parameter-dependent boundary value problem is **singularly perturbed**?

(Q10.2.1.14.B) Discuss to what extent the boundary value problem

$$-\epsilon \Delta u + u = f \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \partial\Omega,$$

is singularly perturbed in the limit $\epsilon \rightarrow 0$.

(Q10.2.1.14.C) We consider the pure transport problem

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \mathbf{grad} u = 0 \quad \text{in } \Omega := [0, 1]^2.$$

with Dirichlet boundary conditions on the inflow boundary part Γ_{in} .

- Describe Γ_{in} .
- Which solution does the method of characteristics yield, if we prescribed Dirichlet boundary conditions

$$u = 0 \quad \text{on } [0, 1] \times \{0\}, \quad u = 1 \quad \text{on } \{0\} \times [0, 1] ?$$

△

10.2.2 Upwinding



Video tutorial for Section 10.2.2: Upwinding: (25 minutes) [Download link](#), [tablet notes](#)

Now we tackle the discretization of stationary convection-diffusion boundary value problems. We start with examining the linear finite element Galerkin discretization for 1D model problem analytically investigated in Ex. 10.2.1.1.

$$-\epsilon \frac{d^2 u}{dx^2} + \frac{du}{dx} = f(x) \quad \text{in } \Omega := [0, 1], \quad u(0) = 0, \quad u(1) = 0. \quad (10.2.2.1)$$

Its variational formulation is, see § 10.2.0.2:

$$u \in H_0^1([0, 1]): \underbrace{\epsilon \int_0^1 \frac{du}{dx}(x) \frac{dv}{dx}(x) dx + \int_0^1 \frac{du}{dx}(x) v(x) dx}_{=:a(u, v)} = \underbrace{\int_0^1 f(x)v(x) dx}_{=:l(v)} \quad \forall v \in H_0^1([0, 1]).$$

As in Section 2.3 we use an equidistant mesh \mathcal{M} (mesh width $h > 0$), the composite trapezoidal rule (2.3.3.8) for the right hand side linear form, and the standard “tent function basis”, see (2.3.2.1).

► We obtain the following linear system of equations for the basis expansion coefficients μ_i , $i = 1, \dots, M - 1$, providing approximations for point values $u(ih)$ of exact solution u .

$$\left(-\frac{\epsilon}{h} - \frac{1}{2}\right)\mu_{i-1} + \frac{2\epsilon}{h}\mu_i + \left(-\frac{\epsilon}{h} + \frac{1}{2}\right)\mu_{i+1} = hf(ih), \quad i = 1, \dots, M - 1, \quad (10.2.2.2)$$

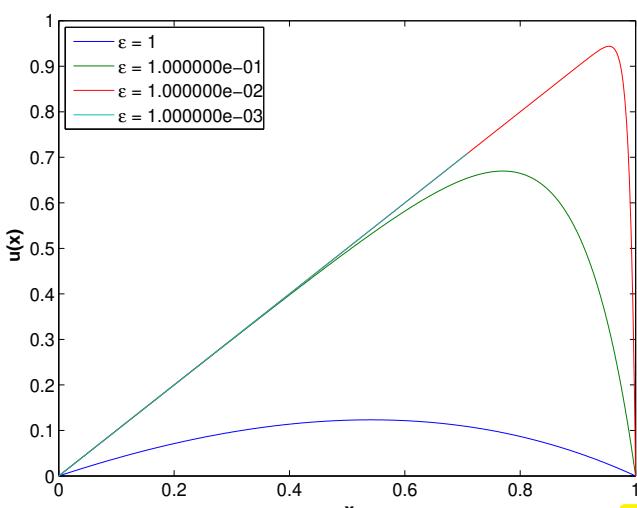
where the homogeneous Dirichlet boundary conditions are taken into account by setting $\mu_0 = \mu_M = 0$.

Remark 10.2.2.3 (Finite differences for convection-diffusion equation in 1D) As in Section 4.1.1 on the finite difference method in 1D, we can also obtain (10.2.2.2) by replacing the derivatives by suitable difference quotients:

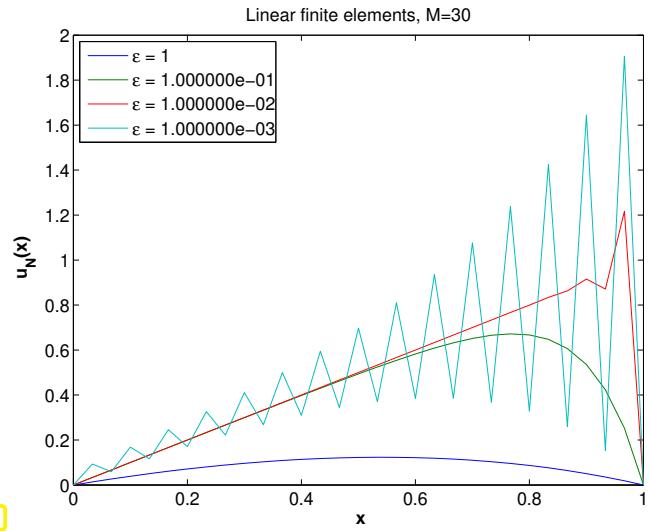
$$\begin{aligned} -\epsilon \frac{d^2u}{dx^2} + \frac{du}{dx} &= f(x) \\ \uparrow &\quad \uparrow &\quad \uparrow \\ \underbrace{\epsilon \frac{-\mu_{i+1} + 2\mu_i - \mu_{i-1}}{h^2}}_{\text{difference quotient for } \frac{d^2u}{dx^2}} + \underbrace{\frac{\mu_{i+1} - \mu_{i-1}}{2h}}_{\text{symmetric d.q. for } \frac{du}{dx}} &= f(ih). \end{aligned} \quad (10.2.2.2)$$

EXPERIMENT 10.2.2.4 (Linear FE discretization of 1D convection-diffusion problem) In this numerical test we qualitatively examine the behavior of approximate solutions of the 1D convection-diffusion boundary value problem (10.2.2.1) obtain by Galerkin finite element discretization.

- ◆ Model boundary value problem (10.2.2.1)
- ◆ linear finite element Galerkin discretization as described above
- ◆ As in Ex. 10.2.1.1: $f \equiv 1$



exact solutions



FE solutions

For very small ϵ we observe spurious *oscillations* of the FE Galerkin solution.

In order to understand the observation made in Exp. 10.2.2.4, we study the linear finite element Galerkin discretization in the limit case $\epsilon = 0$

$$(10.2.2.2) \xrightarrow{\epsilon=0} \mu_{i+1} - \mu_{i-1} = 2hf(ih), \quad i = 1, \dots, M-1. \quad (10.2.2.5)$$

► (10.2.2.5) $\hat{=}$ Linear system of equations with, for even M , *singular* system matrix!

Explanation: the difference equations (10.2.2.5) do not couple grid nodes with even and odd indices. Hence, for even M , an arbitrary constant can be added to μ_i , i odd, whereas the linear systems for the μ_j , j even, is overdetermined. The “even-odd decoupling” inherent in (10.2.2.5) causes the glaring spurious oscillations in the numerical solutions in Ex. 10.2.2.4 for very small ϵ .

For $\epsilon > 0$ the Galerkin matrix will always be regular due to (10.2.0.5), but the linear relationship (10.2.2.5) will become more and more dominant as $\epsilon > 0$ becomes smaller and smaller. In particular, (10.2.2.5) sends the message that values at even and odd numbered nodes will become decoupled, which accounts for the oscillations.

Robust discretization in the singular perturbation limit

We desire a **robust discretization** of (10.2.2.1)

= discretization that produces qualitatively correct (*) solutions for **any** $\epsilon > 0$

(*): “qualitatively correct”, e.g., satisfaction of maximum principle, Thm. 10.1.3.13]

Guideline:

Numerical methods for singularly perturbed problems must “work” for the limit problem

§10.2.2.7 (Discretization of 1D limit problem) What is a meaningful scheme for the limit problem $u' = f$ on an equidistant mesh of $\Omega :=]0, 1[$? The limit problem is an ordinary differential equation and for those

taught us many methods for discretization. Let us examine the simplest introduced in Section 6.2.1 and Section 6.2.2:

$$\text{Explicit Euler method: } \mu_{i+1} - \mu_i = hf(\xi_i) \quad i = 0, \dots, M-1,$$

$$\text{Implicit Euler method: } \mu_{i+1} - \mu_i = hf(\xi_{i+1}) \quad i = 0, \dots, M-1.$$

Both Euler methods can be regarded as finite difference discretizations of $u' = f$ based on one-sided difference quotients, recall Ex. 9.2.6.4

$$\text{Explicit Euler: } \frac{du}{dx}(x_i) \approx \frac{u(x_{i+1}) - u(x_i)}{h}, \quad \text{Implicit Euler: } \frac{du}{dx}(x_i) \approx \frac{u(x_i) - u(x_{i-1})}{h}.$$

Conversely, (10.2.2.5) can be obtained by relying on a symmetric difference quotient:

$$(10.2.2.5) \text{ by approximating } \frac{du}{dx}(x_i) \approx \frac{u(x_{i+1}) - u(x_{i-1})}{2h}.$$

Apparently, the use of a symmetric difference quotient for discretizing the convective term incurs spurious oscillations, see Exp. 10.2.2.4.

► Conclusion: use **one-sided difference quotients** for discretization of convective term!

Which type ? (Explicit or implicit Euler ?) Let us look at the resulting equations.

$$1. \text{ Linear system arising from } \text{use of backward difference quotient} \quad \frac{du}{dx}|_{x=x_i} = \frac{\mu_i - \mu_{i-1}}{h}.$$

$$\left(-\frac{\epsilon}{h}-1\right)\mu_{i-1} + \left(\frac{2\epsilon}{h}+1\right)\mu_i + -\frac{\epsilon}{h}\mu_{i+1} = hf(ih), \quad i = 1, \dots, M-1, \quad (10.2.2.8)$$

$$2. \text{ Linear system arising from } \text{use of forward difference quotient} \quad \frac{du}{dx}|_{x=x_i} = \frac{\mu_{i+1} - \mu_i}{h}.$$

$$-\frac{\epsilon}{h}\mu_{i-1} + \left(\frac{2\epsilon}{h}-1\right)\mu_i + \left(-\frac{\epsilon}{h}+1\right)\mu_{i+1} = hf(ih), \quad i = 1, \dots, M-1, \quad (10.2.2.9)$$

EXPERIMENT 10.2.2.10 (One-sided difference approximation of convective terms)

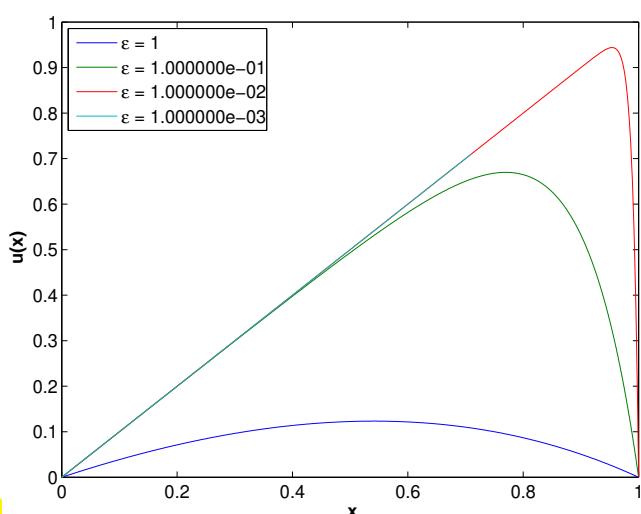
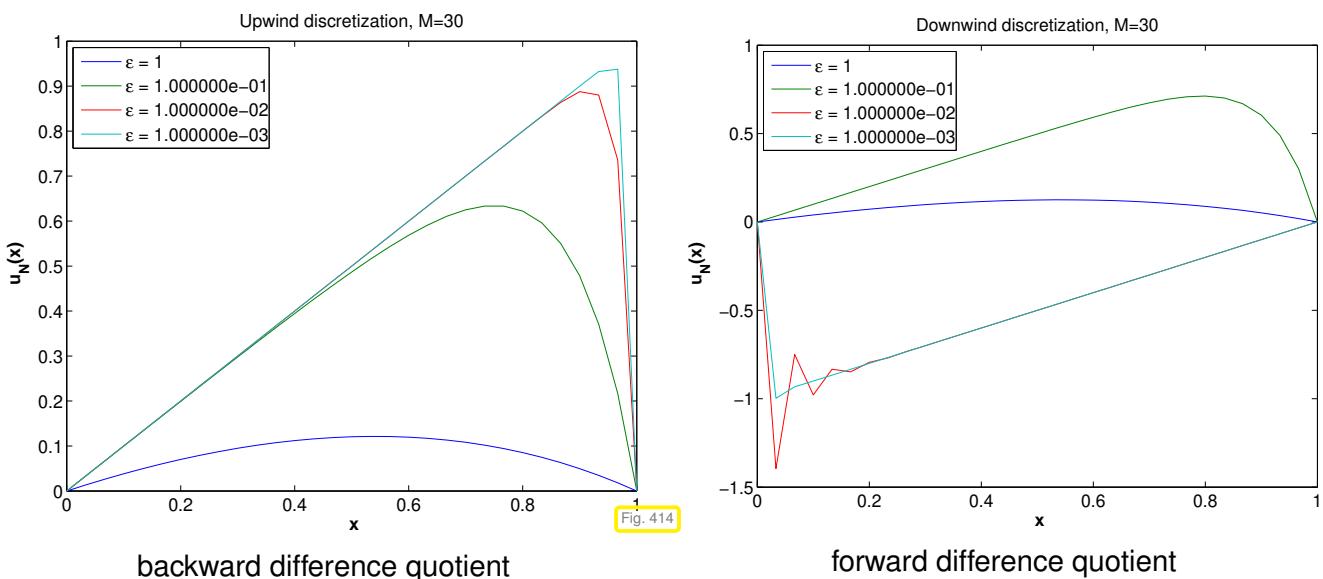


Fig. 412

We revisit the model problem of Exp. 10.2.2.4.

◀ exact solutions for different values of ϵ

We study the solutions obtained by the discretizations (10.2.2.8) and (10.2.2.9).



Only the discretization of $\frac{du}{dx}$ based on the backward difference quotient generates qualitatively correct (piecewise linear) discrete solutions (a “good method”).

If the forward difference quotient is used, the discrete solutions may violate the maximum principle of Thm. 10.1.3.13 (a “bad method”). \square

§10.2.2.11 (Selection criterion: Discrete maximum principle) How can we tell a good method from a bad method by merely examining the system matrix?

► Heuristic criterion for $\epsilon \rightarrow 0$ -robust stability of nodal finite element Galerkin discretization/finite difference discretization of *singularly perturbed* scalar linear convection-diffusion BVP (10.2.0.1) (with Dirichlet b.c.):

(Linearly interpolated) discrete solution satisfies **maximum principle** (3.7.2.1).

\Updownarrow

System matrix complies with sign-conditions (3.7.2.8)–(3.7.2.10).

We focus on **nodal** finite element Galerkin discretization for which the basis expansion coefficients μ_i of the Galerkin solution $u_h \in V_h$ double as point values of u_h at interpolation nodes. This is satisfied for Lagrangian finite element methods (\rightarrow Section 2.6) when standard nodal basis functions according to (2.6.1.4) are used.

Recall the **sign-conditions** (3.7.2.8)–(3.7.2.10) for the system matrix \mathbf{A} arising from nodal finite element Galerkin discretization or finite difference discretization in order to be able to conclude a discrete maximum principle:

- ◆ (3.7.2.8): positive diagonal entries,
- ◆ (3.7.2.9): non-positive off-diagonal entries,
- ◆ “(3.7.2.10)": diagonal dominance,

$$(\mathbf{A})_{ii} > 0 ,$$

$$(\mathbf{A})_{ij} \leq 0, \text{ if } i \neq j ,$$

$$\sum_j (\mathbf{A})_{ij} \geq 0 .$$

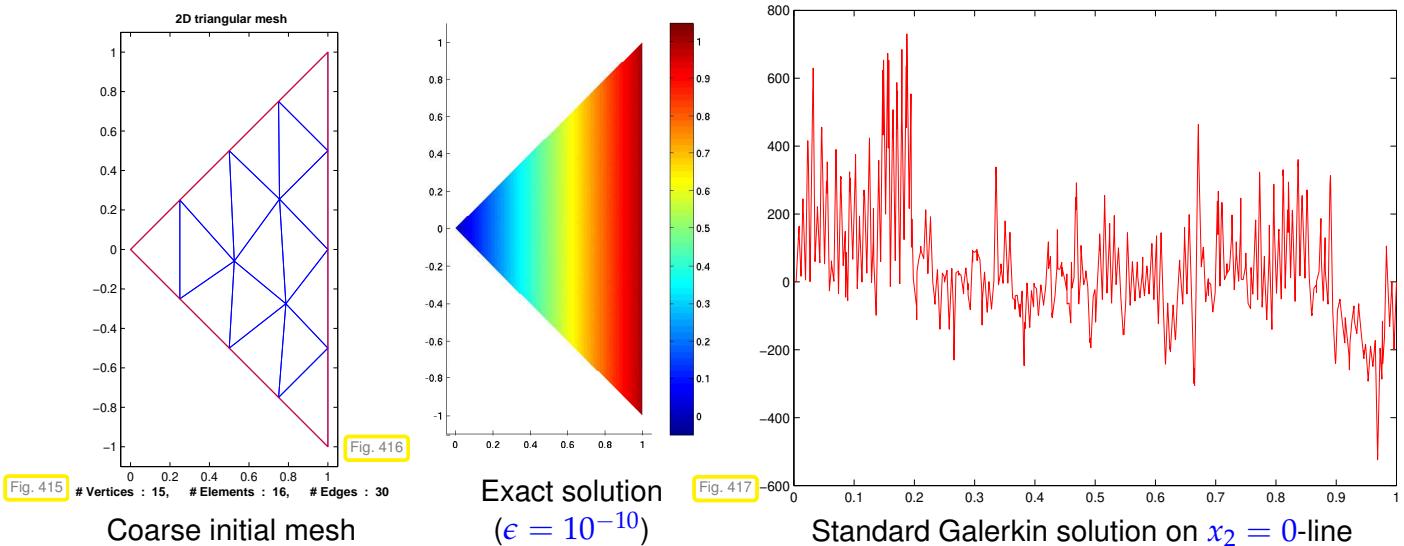
These conditions are met for *equidistant meshes in 1D*

- ◆ for the standard $\mathcal{S}_1^0(\mathcal{M})$ -Galerkin discretization (10.2.2.2), provided that $|\epsilon h^{-1}| \geq \frac{1}{2}$,
 - ◆ when using *backward* difference quotients for the convective term (10.2.2.8) for **any** choice of $\epsilon \geq 0$, $h > 0$,
 - ◆ when using *forward* difference quotients for the convective term (10.2.2.9), **provided that** $|\epsilon h^{-1}| \geq 1$.
- Only the use of a *backward* difference quotient for the convective term guarantees the (discrete) maximum principle in an $\epsilon \rightarrow 0$ -robust fashion!

Terminology: The approximation of $\frac{du}{dx}$ by *backward* difference quotients is called **upwinding**

EXPERIMENT 10.2.2.12 (Spurious Galerkin solution for 2D convection-diffusion BVP) So far all consideration were set in one spatial dimension. Are they relevant for higher dimensions? In this example we explore the danger of spurious oscillations for a 2D model problem.

- ◆ Triangle domain $\Omega = \{(x, y) : 0 \leq x \leq 1, -x \leq y \leq x\}$.
- ◆ Velocity $\mathbf{v}(x) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \geq (10.2.0.1)$ becomes $-\epsilon \Delta u + u_x = 1$.
- ◆ Exact solution: $u_\epsilon(x_1, x_2) = x - \frac{1}{1-e^{-1/\epsilon}}(e^{-(1-x_1)/\epsilon} - e^{-1/\epsilon})$, Dirichlet boundary conditions set accordingly
- ◆ Standard Galerkin discretization by means of linear finite elements on sequence of triangular mesh created by regular refinement.



As expected: spurious oscillations mar Galerkin solution

► Difficulty observed in 1D also haunts discretization in higher dimensions.

Review question(s) 10.2.2.13 (Upwinding)

(Q10.2.2.13.A) We have seen the variational formulation

$$u \in H_0^1(\Omega): \underbrace{\epsilon \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} w \, dx + \int_{\Omega} (\mathbf{v} \cdot \mathbf{grad} u) w \, dx}_{\text{bilinear form } \mathbf{a}(u, w)} = \underbrace{\int_{\Omega} f(x) w(x) \, dx}_{\text{linear form } \ell(w)} \quad \forall w \in H_0^1(\Omega), \quad (10.2.0.3)$$

of the homogeneous Dirichlet problem for a convection diffusion equation. In (10.2.0.5) we have established the estimate

$$a(u, u) = \epsilon \int_{\Omega} \|\mathbf{grad} u\|^2 dx > 0 \quad \forall u \in H_0^1(\Omega) \setminus \{0\}, \quad (10.2.0.5)$$

and we also found

$$\exists C > 0: |a(u, v)| \leq C|u|_{H^1(\Omega)}|v|_{H^1(\Omega)} \quad \forall u, v \in H_0^1(\Omega). \quad (10.2.0.4)$$

Derive an a prior error estimate for $|u - u_h|_{H^1(\Omega)}$, where $u_h \in \mathcal{S}_1^0(\mathcal{M})$ is the finite-element solution of (10.2.0.3) based on a triangular mesh \mathcal{M} of $\Omega \subset \mathbb{R}^2$. The best-approximation error for u in $\mathcal{S}_1^0(\mathcal{M})$ may enter your bound, which should be *explicit in ϵ* .

Hint. Start from (10.2.0.5) and then use *Galerkin orthogonality*.

△

10.2.2.1 Upwind Quadrature



Video tutorial for Section 10.2.2.1: Upwind Quadrature: (35 minutes) [Download link](#), [tablet notes](#)

In one spatial dimension upwinding was easily motivated by choosing the “right” one-sided difference quotient. For $d = 2, 3$ and unstructured meshes this is no longer a viable recipe. How can we extend the upwinding idea to $d > 1$?

At second glance, we may think of upwinding as “respecting the flow of information” in the sense of § 10.2.1.10. This turns out to be a fruitful perspective.

§10.2.2.14 (Upwind quadrature in one spatial dimension) Let us revisit 1D model problem

$$-\epsilon \frac{d^2 u}{dx^2} + \frac{du}{dx} = f(x) \quad \text{in } \Omega, \quad u(0) = 0, \quad u(1) = 0, \quad (10.2.2.1)$$

with variational formulation, see § 10.2.0.2:

$$u \in H_0^1([0, 1]): \underbrace{\epsilon \int_0^1 \frac{du}{dx}(x) \frac{dw}{dx}(x) dx}_{=:a(u, w)} + \underbrace{\int_0^1 \frac{du}{dx}(x) w(x) dx}_{=:l(w)} = \underbrace{\int_0^1 f(x) w(x) dx}_{\text{convective term}} \quad \forall w \in H_0^1([0, 1]).$$

We consider linear finite element Galerkin discretization on an equidistant mesh \mathcal{M} with M cells, mesh-width $h = \frac{1}{M}$, cf. Sect. 2.3.

We opt for the *global* composite trapezoidal rule

$$\int_0^1 \psi(x) dx \approx h \sum_{j=1}^{M-1} \psi(jh), \quad \text{for } \psi \in C^0([0, 1]), \psi(0) = \psi(1) = 0,$$

for the evaluation of the convective term in bilinear form a :

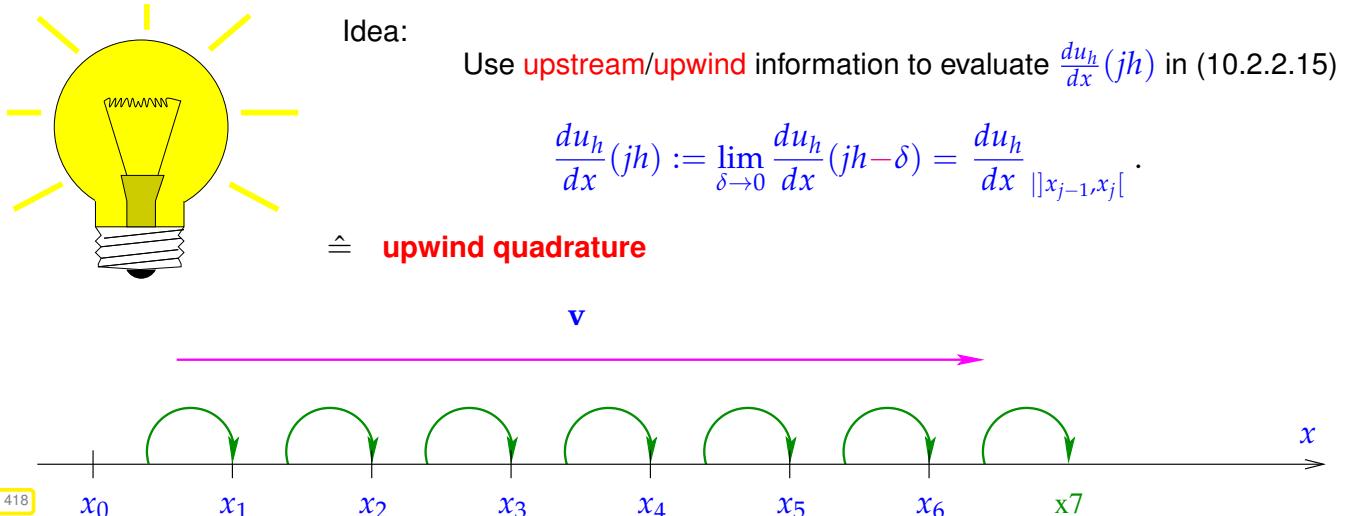
$$\int_0^1 \frac{du_h}{dx}(x) w_h(x) dx \approx h \sum_{j=1}^{M-1} \frac{du_h}{dx}(jh) w_h(hj), \quad w_h \in \mathcal{S}_{1,0}^0(\mathcal{M}). \quad (10.2.2.15)$$

Note that this is *not a valid formula*, because $\frac{du_h}{dx}(jh)$ is *ambiguous*, since $\frac{du_h}{dx}$ is discontinuous at nodes of the mesh for $u_h \in \mathcal{S}_{1,0}^0(\mathcal{M})$!

Up to now we have resolved this ambiguity by the policy of *local* quadrature, see Section 2.7.5: quadrature rule applied locally on each cell with all information taken from that cell.

However:

Convection transports information in the direction of \mathbf{v} !



Upwind quadrature yields the following contribution of the discretized convective term to the linear system using the basis expansion $u_h = \sum_{l=1}^{M-1} \mu_l b_h^l$ into *locally supported* nodal basis functions (“tent functions” b_h^j)

$$\int_0^1 \sum_{l=1}^{M-1} \mu_l \frac{db_h^l}{dx}(x) b_h^i(x) dx \stackrel{(10.2.2.15)}{\approx} h \sum_{j=1}^{M-1} \sum_{l=1}^{M-1} \mu_l \frac{db_h^l}{dx} \Big|_{[x_{j-1}, x_j]} (jh) b_h^i(jh) = h \frac{\mu_i - \mu_{i-1}}{h},$$

where we used the facts

- $\text{supp } b_h^i = [x_{i-1}, x_{i+1}]$ and $b_h^i(jh) = \delta_{ij}$, see (2.3.2.2),
- $\frac{db_h^j}{dx} \Big|_{[x_{j-1}, x_j]} = \frac{1}{h}$ and $\frac{db_h^{j-1}}{dx} \Big|_{[x_{j-1}, x_j]} = -\frac{1}{h}$, easily concluded from (2.3.2.6).

► Linear system from upwind quadrature:

$$\left(-\frac{\epsilon}{h} - 1 \right) \mu_{i-1} + \left(\frac{2\epsilon}{h} + 1 \right) \mu_i - \frac{\epsilon}{h} \mu_{i+1} = hf(ih), \quad i = 1, \dots, M-1, \quad (10.2.2.8)$$

which is the **same** as that obtained from a backward finite difference discretization of $\frac{du}{dx}$! □

§10.2.2.16 (Multi-dimensional upwind quadrature) The idea of upwind quadrature can be generalized to $d > 1$: we consider $d = 2$ and linear Lagrangian finite element Galerkin discretization of

$$u \in H_0^1(\Omega): \quad \epsilon \int_{\Omega} \mathbf{grad} u \cdot \mathbf{grad} w dx + \int_{\Omega} (\mathbf{v} \cdot \mathbf{grad} u) w dx = \int_{\Omega} f(x) w(x) dx \quad \forall w \in H_0^1(\Omega). \quad (10.2.0.3)$$

on a triangular mesh \mathcal{M} , see Section 2.4; we use the finite element space $\mathcal{S}_{1,0}^0(\mathcal{M})$.

- ① Approximation of contribution of convective terms to bilinear form by means of the *global trapezoidal rule*:

For a continuous function $\psi : \Omega \mapsto \mathbb{R}$ the (global) trapezoidal rule can easily be derived from the 2D *composite* trapezoidal rule based on

$$\int_K \psi(x) dx \approx \frac{|K|}{3} (\psi(a^1) + \psi(a^2) + \psi(a^3)), \quad (2.4.6.10)$$

where the a^i , $i = 1, 2, 3$, are the vertices of the triangle K .

$$\begin{aligned} \blacktriangleright \int_{\Omega} \psi(x) dx &= \sum_{K \in \mathcal{M}} \int_K \psi(x) dx \approx \sum_{K \in \mathcal{M}} \frac{|K|}{3} (\psi(a_K^1) + \psi(a_K^2) + \psi(a_K^3)) \\ &\approx \sum_{p \in \mathcal{V}(\mathcal{M})} \left(\frac{1}{3} \sum_{K \in \mathcal{U}_p} |K| \right) \psi(p), \end{aligned} \quad (10.2.2.17)$$

by changing the order of summation. This formula is the *global* trapezoidal rule in 2D on a triangular mesh, which we now apply to the convective term in the discrete variational formulation of the convection-diffusion problem (10.2.0.3). For $u_h, w_h \in \mathcal{S}_{1,0}^0(\mathcal{M})$ we get

$$\int_{\Omega} (\mathbf{v} \cdot \mathbf{grad} u_h) w_h dx \approx \sum_{p \in \mathcal{V}(\mathcal{M})} \left(\frac{1}{3} \sum_{K \in \mathcal{U}_p} |K| \right) \cdot \mathbf{v}(p) \cdot \mathbf{grad} u_h(p) w_h(p). \quad (10.2.2.18)$$

ambigous for $u_h \in \mathcal{S}_1^0(\mathcal{M})$!

☞ notation: $\mathcal{U}_p := \{K \in \mathcal{M} : p \in \bar{K}\}$, $p \in \mathcal{V}(\mathcal{M})$

- ② Again we resolve the ambiguity of $\mathbf{grad} u_h(p)$ by using the value of $\mathbf{grad} u_h$ “from where the flow comes”.

This means that we fix the ambiguous value of $\mathbf{v}(p) \cdot \mathbf{grad} u_h(p)$, $u_h \in \mathcal{S}_1^0(\mathcal{M})$, by taking the gradient from the triangle upstream to the node p :

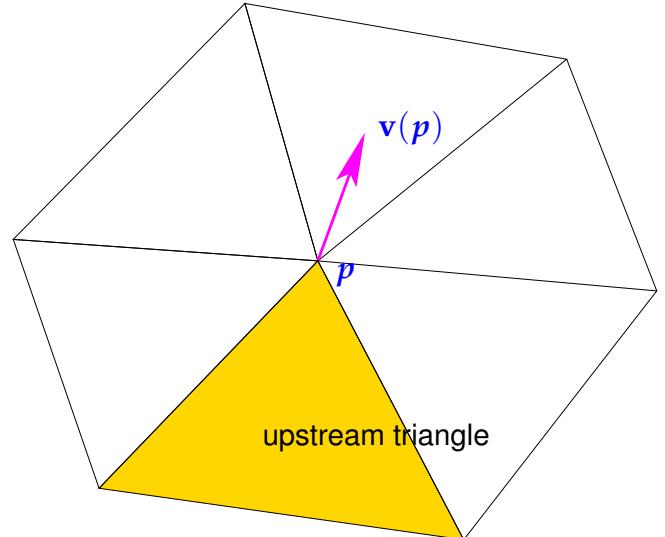
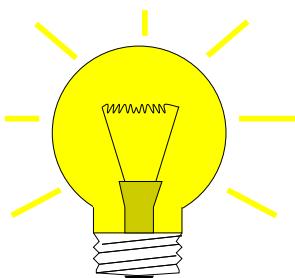


Fig. 419



Idea: Use **upstream/upwind** information to evaluate $\mathbf{grad} u_h(p)$ in (10.2.2.18)

$$\mathbf{v}(p) \cdot \mathbf{grad} u_h(p) := \lim_{\delta \rightarrow 0} \mathbf{v}(p) \cdot \mathbf{grad} u_h(p - \delta \mathbf{v}(p)). \quad (10.2.2.19)$$

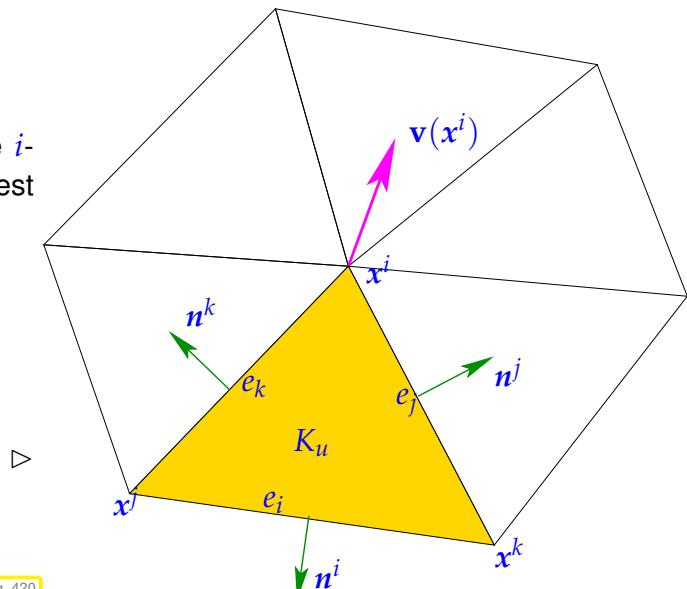
$\hat{=}$ general **upwind quadrature**.

Note that by (10.1.1.2) the vector $\mathbf{v}(p)$ supplies the direction of the streamline through p . Hence, $-\mathbf{v}(p)$ is the direction from which information is “carried into p ” by the flow.

This is the contribution of convective term to the i -th row of the final linear system of equations (test function = tent function b_h^i):

$$\underbrace{\left(\frac{1}{3} \sum_{K \in \mathcal{U}_i} |K|\right)}_{=: U_i} \mathbf{v}(x^i) \cdot \mathbf{grad} u_h|_{K_u},$$

where K_u is the upstream triangle of p .



Using the expressions for the gradients of barycentric coordinate functions from Section 2.4.5

$$\mathbf{grad} \lambda_* = -\frac{|e_i|}{2|K|} \mathbf{n}^*, \quad * = i, j, k, \quad \text{see Fig. 421 ,}$$

and the nodal basis expansion of u_h , we obtain for the convective contribution to the i -th line of the final linear system

$$\frac{U_i}{2|K_u|} \left(\underbrace{-\|x^j - x^k\| n^i \cdot \mathbf{v}(x^i) \mu_i - \|x^i - x^j\| n^k \cdot \mathbf{v}(x^i) \mu_k - \|x^i - x^k\| n^j \cdot \mathbf{v}(x^i) \mu_j}_{\leftrightarrow \text{diagonal entry}} \right)$$

By the very definition of the upstream triangle K_u we find

$$n^i \cdot \mathbf{v}(x^i) \leq 0, \quad n^k \cdot \mathbf{v}(x^i) \geq 0, \quad n^j \cdot \mathbf{v}(x^i) \geq 0.$$

► That the sign conditions (3.7.2.8), (3.7.2.9) are satisfied for the discretized convective term, (3.7.2.10) is obvious from $\lambda_i + \lambda_j + \lambda_k = 1$, which means $\mathbf{grad}(\lambda_i + \lambda_j + \lambda_k) = 0$.

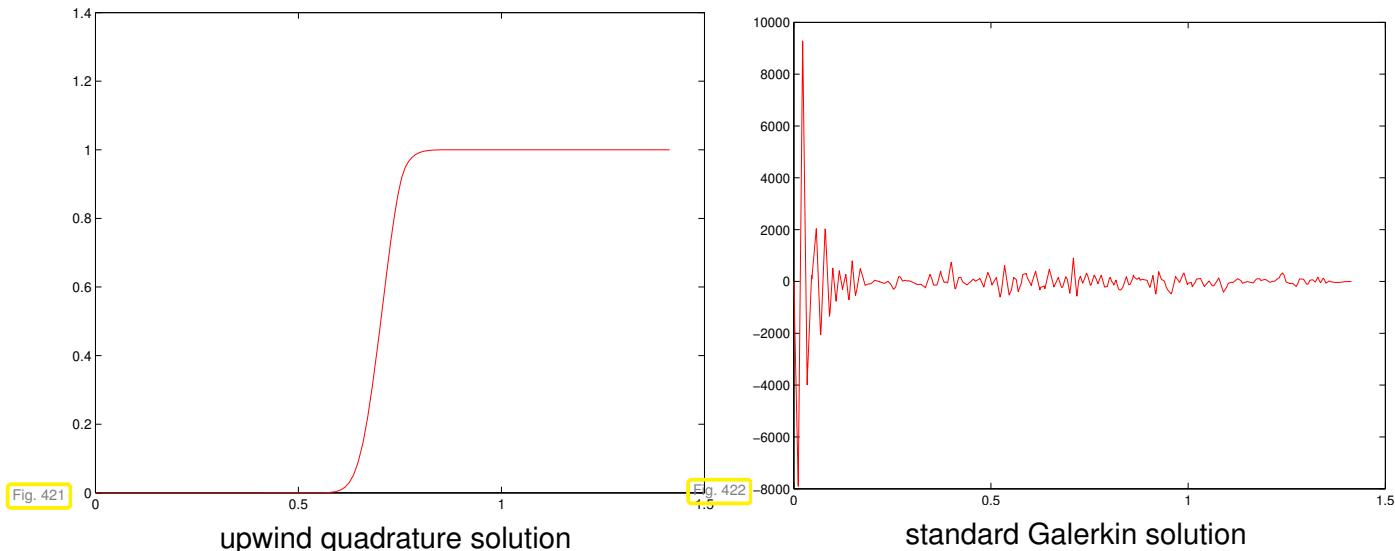
Usually, the upwind quadrature discretization of the convective term will be combined with a standard finite element Galerkin discretization of the diffusive term. In this case the finite element solution of (10.2.0.1) will satisfy the maximum principle, if this is true for the discretization of the diffusive term. Criteria for this have been established in Section 3.7, see Thm. 3.7.2.20 and Rem. 3.7.2.21, page 385. □

EXPERIMENT 10.2.2.20 (Upwind quadrature discretization)

From the above theoretical considerations we expect that the upwind quadrature finite element Galerkin discretization is immune to spurious oscillations for $\epsilon \rightarrow 0$. This is confirmed in this numerical experiment.

- ◆ Computational domain: unit square $\Omega = [0, 1]^2$
- ◆ $-\epsilon \Delta u + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \cdot \mathbf{grad} u = 0$
- ◆ Dirichlet boundary conditions: $u(x, y) = 1$ for $x > y$ and $u(x, y) = 0$ for $x \leq y$
- ◆ Limiting case ($\epsilon \rightarrow 0$): $u(x, y) = 1$ for $x > y$ and $u(x, y) = 0$ for $x \leq y$
- ◆ layer along the diagonal from $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ to $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ in the limit $\epsilon \rightarrow 0$
- ◆ 2D triangular Delaunay triangulation, see Rem. 4.2.2.3
- ◆ linear finite element upwind quadrature discretization

- Monitored: discrete solutions along diagonal from $(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix})$ to $(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix})$ for $\epsilon = 10^{-10}$.



We observe that the upwind quadrature scheme respects the maximum principle, whereas the standard Galerkin solution is rendered useless by spurious oscillations. □

Review question(s) 10.2.2.21 (Upwind quadrature)

(Q10.2.2.21.A) [] Assume that you are given a mesh \mathcal{M} of a computational domain $\Omega \subset \mathbb{R}^2$. Explain the difference between the global and the local **composite trapezoidal rule** on \mathcal{M} .

(Q10.2.2.21.B) How would you implement a C++ function based on LEHRFEM++

```
lf::mesh::utils::CodimMeshDataSet<const lf::mesh::Entity *>
findUpwindTriangle (
    const lf::mesh::Mesh &mesh,
    const lf::mesh::MeshFunction<Eigen::Vector2d> &v);
```

that creates a data structure which contains a pointer to the **upwind triangle** for every node of a 2D triangular mesh passed in the argument `mesh`. The other argument `v` provides the velocity field `v`, which is supposed to be continuous.

△

10.2.2.2 Streamline Diffusion



Video tutorial for Section 10.2.2.2: Streamline Diffusion: (35 minutes) [Download link](#), [tablet notes](#)

§10.2.2.22 (Artificial viscosity in 1D) To motivate a completely different way to achieve $\epsilon \rightarrow 0$ -robustness, we take another look at the 1D upwind discretization of

$$-\epsilon \frac{d^2 u}{dx^2} + \frac{du}{dx} = f(x) \quad \text{in } \Omega :=]0, 1[, \quad u(0) = 0 , \quad u(1) = 0 . \quad (10.2.2.1)$$

and view it from a different perspective.

Recall the 1D **upwind** (finite difference) discretization of (10.2.2.1):

$$\left(-\frac{\epsilon}{h} - 1\right)\mu_{i-1} + \left(\frac{2\epsilon}{h} + 1\right)\mu_i - \frac{\epsilon}{h}\mu_{i+1} = hf(ih) , \quad i = 1, \dots, M-1 . \quad (10.2.2.8)$$

$$\Leftrightarrow (\epsilon + h/2) \underbrace{\frac{-\mu_{i-1} + 2\mu_i - \mu_{i+1}}{h^2}}_{\triangleq \text{difference quotient for } \frac{d^2u}{dx^2}} + \underbrace{\frac{-\mu_{i-1} + \mu_{i+1}}{2h}}_{\triangleq \text{difference quotient for } \frac{du}{dx}} = f(ih),$$

for $i = 1, \dots, M-1$. This suggests a new interpretation of the stabilization achieved through upwinding:

Upwinding = h -dependent enhancement of diffusive term
 (This is widely known as **artificial diffusion/viscosity**)

We also observe that the upwinding strategy just adds the *minimal amount of diffusion* to make the resulting system matrix comply with the conditions (3.7.2.8)–(3.7.2.10), which ensure that the discrete solution satisfies the maximum principle.

? Question: How to extend the trick of adding artificial diffusion to $d > 1$?

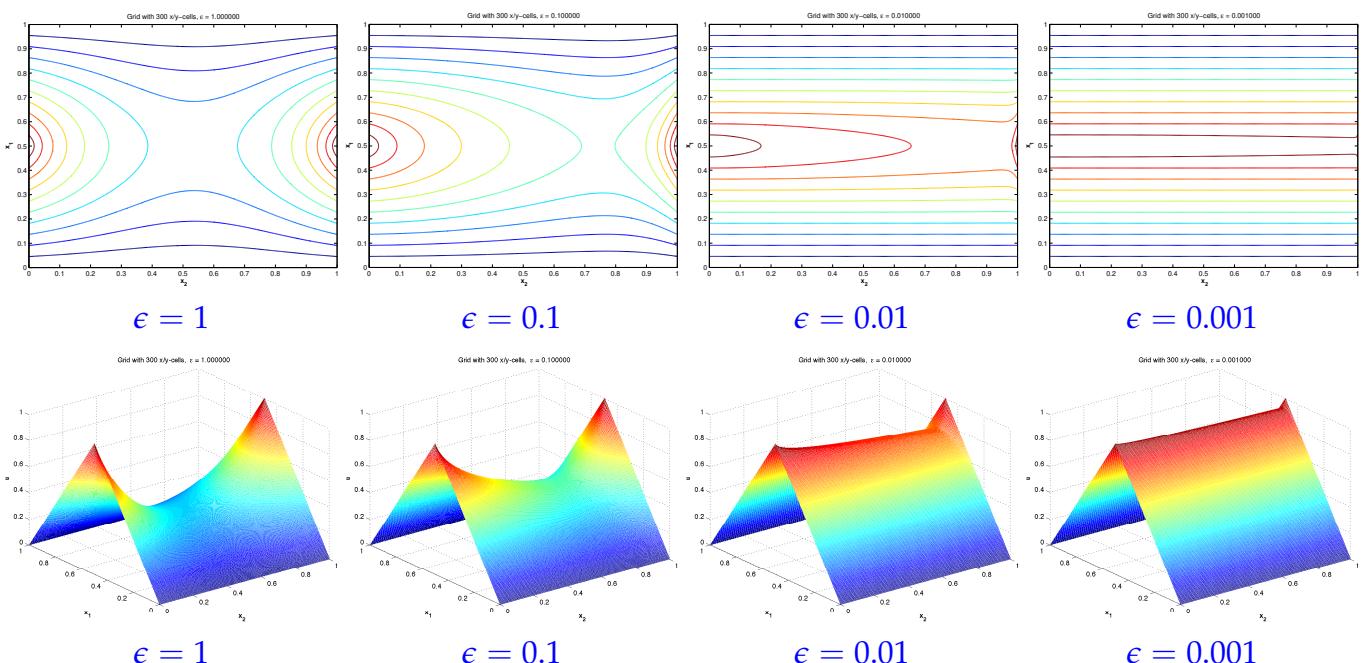
Well, just add an extra h -dependent multiple of $-\Delta$! Let's try. □

EXAMPLE 10.2.2.23 (Effect of added diffusion) We consider the convection-diffusion boundary value problem ((10.2.0.1) with $\mathbf{v} = (1, 0)$)

$$-\epsilon \Delta u + \frac{\partial u}{\partial x_1} = 0 \quad \text{in } \Omega = [0, 1]^2, \quad u = g \quad \text{on } \partial\Omega.$$

Here, the Dirichlet data are $g(x) = 1 - 2|x_2 - \frac{1}{2}|$ ("roof function").

Thus, for $\epsilon \approx 0$ we expect $u \approx g$, because the Dirichlet data are just transported in x_1 -direction and there are no boundary layers.

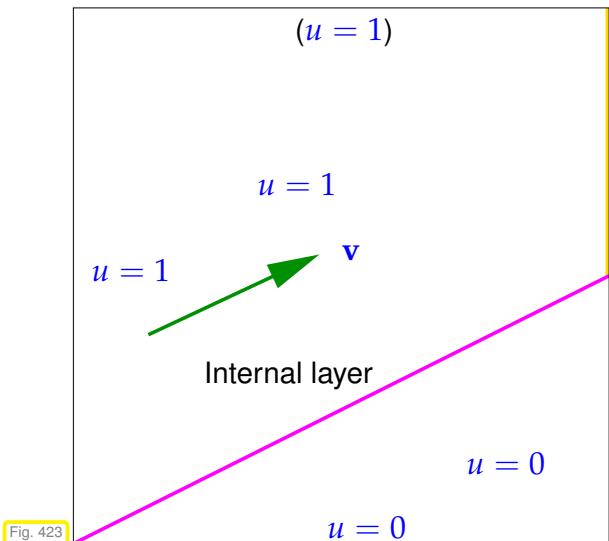


We observe that stronger diffusion leads to "smearing" of features that the flow field transports into the interior of the domain. □



(Too much) artificial diffusion \Rightarrow smearing of internal layers
(We are no longer solving the right problem!)

Remark 10.2.2.24 (Internal layers)



We see that solutions of pure transport problem with discontinuous boundary data

- display a discontinuity across the streamline emanating from the point of discontinuity on $\partial\Omega$,
- are *smooth along streamlines*.

Now let us add a little diffusion. Then we can again impose Dirichlet boundary conditions everywhere:

This is how the solution of

$$-\delta\Delta + \mathbf{v} \cdot \nabla u = 0 \quad \text{in } \Omega,$$

with $\delta > 0$ and the same boundary data looks qualitatively

\Rightarrow Observe the smearing of the internal layer !

As in Ex. 10.2.1.1, we would also find a boundary layer which is marked in gold in the figure. Inside this boundary layer the solution drops to zero abruptly.

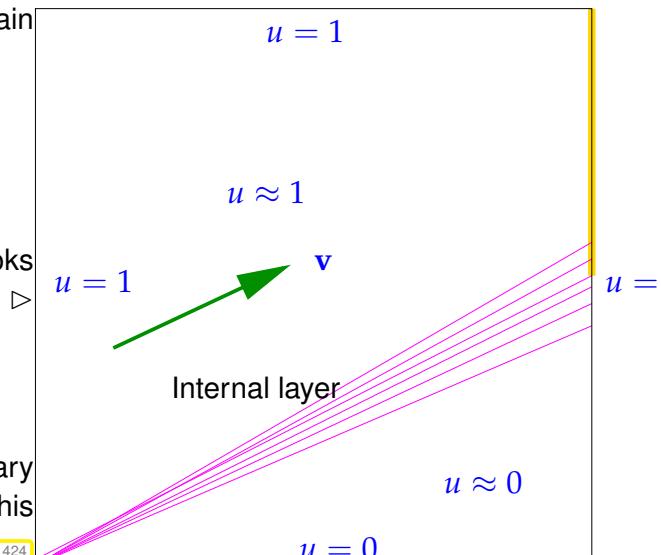
We consider the pure transport problem with constant velocity:

$$\mathbf{v} \cdot \nabla u = 0 \quad \text{in } \Omega,$$

where $\Omega = [0,1]^2$, $\mathbf{v} = (2, 1)$, $\epsilon = 10^{-4}$, and the boundary condition are given in Fig. 424.

The Dirichlet b.c. that can only be satisfied on **inflow boundary**: $u = 1$ on $\{x_1 = 0\} \cup \{x_2 = 1\}$, $u = 0$ on $\{x_1 = 1\} \cup \{x_2 = 0\}$.

\triangleleft Boundary conditions in brackets cannot be imposed for the limit problem.



Remark 10.2.2.25. Note that the above boundary conditions actually do not supply valid Dirichlet data for a second-order elliptic boundary value problem, because they jump at the corners, cf. Remark 1.9.0.6, page 130. However, they make sense for the limit problem and a finite element discretization can also be applied in this case.

Heuristics of streamline upwinding (SU)

Since the solution is *smooth along streamlines*, then adding *diffusion in the direction of streamlines* cannot do much harm.

§10.2.2.27 (Anisotropic diffusion) What does “diffusion in a direction” mean?

- Think of a generalized Fourier’s law (1.6.0.5) for $d = 2$, e.g.,

$$\mathbf{j}(\mathbf{x}) = - \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{\nabla} u(\mathbf{x}). \quad (10.2.2.28)$$

This means, only a temperature variation in \mathbf{x}_1 -direction triggers a heat flow, which is called an **anisotropic** heat conduction phenomenon. In the case of (10.2.2.28) we witness “diffusion in \mathbf{x}_1 -direction”.

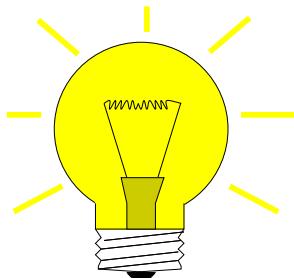
We conclude that diffusion in a general direction $\mathbf{v} \in \mathbb{R}^2$ amounts to a flux law

$$\mathbf{j}(\mathbf{x}) = -\mathbf{v}(\mathbf{x})\mathbf{v}(\mathbf{x})^\top \mathbf{\nabla} u(\mathbf{x}) \quad (10.2.2.29)$$

Such an extended Fourier’s law is an example of **anisotropic diffusion**.

Anisotropic diffusion can simply be taken into account in variational formulations and Galerkin discretization by replacing the heat conductivity κ /stiffness σ with a symmetric, positive (semi-)definite $d \times d$ -matrix, the **diffusion tensor**. In (10.2.2.29) the diffusion tensor is the rank-1 matrix $\mathbf{v}(\mathbf{x})\mathbf{v}(\mathbf{x})^\top$. \square

Armed with a concept of diffusion in direction \mathbf{v} , we can now cast the heuristics of the streamline diffusion approach into formulas.



Idea:

Anisotropic artificial diffusion in streamline direction

On cell K replace: $\epsilon \leftarrow \underbrace{\epsilon \mathbf{I} + \delta_K \mathbf{v}_K \mathbf{v}_K^\top}_{\text{new diffusion tensor}} \in \mathbb{R}^{2,2}$.

$\mathbf{v}_K \doteq$ local velocity (e.g., obtained by averaging)

$\delta_K > 0 \doteq$ method parameter controlling the strength of anisotropic diffusion

This idea spawns the class of **streamline diffusion methods** for convection-diffusion BVPs.

Thus, (for the model problem) Galerkin discretization may target the variational problem

$$\int_{\Omega} (\epsilon \mathbf{I} + \delta_K \mathbf{v}_K \mathbf{v}_K^\top) \mathbf{\nabla} u \cdot \mathbf{\nabla} w + \mathbf{v}(\mathbf{x}) \cdot \mathbf{\nabla} u w \, dx = \int_{\Omega} f w \, dx \quad \forall w \in H_0^1(\Omega). \quad (10.2.2.30)$$



This tampering affects the solution u
(solution of (10.2.2.30) \neq solution of (10.2.0.1))

Desirable:

Maintain **consistency** of variational problem!

Definition 10.2.2.31. Consistent modifications of variational problems

A variational problem is called a **consistent modification** of another, if both possess the same (unique) solution(s).

From an abstract point of view we start from a linear variational problem (\rightarrow Def. 1.4.1.6)

$$u \in V: \quad a(u, v) = \ell(v) \quad \forall v \in V_0 . \quad (1.4.1.7)$$

For an (affine) trial space $V_h \subset V$ and test space $V_{0,h} \subset V_0$ we consider a discrete variational problem with a **modified bilinear form** denoted by $\tilde{a}(\cdot, \cdot)$:

$$u_h \in V_h: \quad \tilde{a}(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h} . \quad (10.2.2.32)$$

Then (10.2.2.32) is **consistent** with (1.4.1.7) according to Def. 10.2.2.31, if

$$u \in V, \quad a(u, v) = \ell(v) \quad \forall v \in V_0 \Rightarrow \tilde{a}(u, v_h) = \ell(v_h) \quad \forall v_h \in V_{0,h} , \quad (10.2.2.33)$$

Of course, inserting u into \tilde{a} must result in well-defined numbers $\tilde{a}(u, v_h)$ for every $v_h \in V_{0,h}$.

Remark 10.2.2.34. We point out that the variational crimes investigated in Sect. 3.5 represent non-consistent modifications: they introduce a **consistency error**. \square

With the following trick we can ensure the consistency for streamline upwind variational problem:



Idea: Add anisotropic diffusion through a **residual term** that vanishes for the exact solution u
Natural candidate $-\epsilon \Delta u + \mathbf{v} \cdot \mathbf{grad} u - f$

► Streamline upwind variational problem: given mesh \mathcal{M} seek $u \in H_0^1(\Omega) \cap H^2(\mathcal{M})$

$$\begin{aligned} & \int_{\Omega} \epsilon \mathbf{grad} u \cdot \mathbf{grad} w + (\mathbf{v}(x) \cdot \mathbf{grad} u) w \, dx \\ & + \underbrace{\sum_{K \in \mathcal{M}} \delta_K \int_K (-\epsilon \Delta u + \mathbf{v} \cdot \mathbf{grad} u - f) \cdot (\mathbf{v} \cdot \mathbf{grad} w) \, dx}_{\text{stabilization term}} = \int_{\Omega} f w \, dx \quad \forall w \in H_0^1(\Omega) . \quad (10.2.2.35) \end{aligned}$$

The **colored** terms are new in (10.2.2.35) compared to (10.2.2.30). They ensure that (10.2.2.35) is a **consistent** variational formulation in the sense of Def. 10.2.2.31.

Remark 10.2.2.36. Note that enhanced smoothness of u , namely in addition $u \in H^2(K)$ for all $K \in \mathcal{M}$, is required to render (10.2.2.35) meaningful. Often authors use the notation $H^2(\mathcal{M})$ to designate the Sobolev space of functions that belong \mathcal{M} -piecewise to H^2 . \square

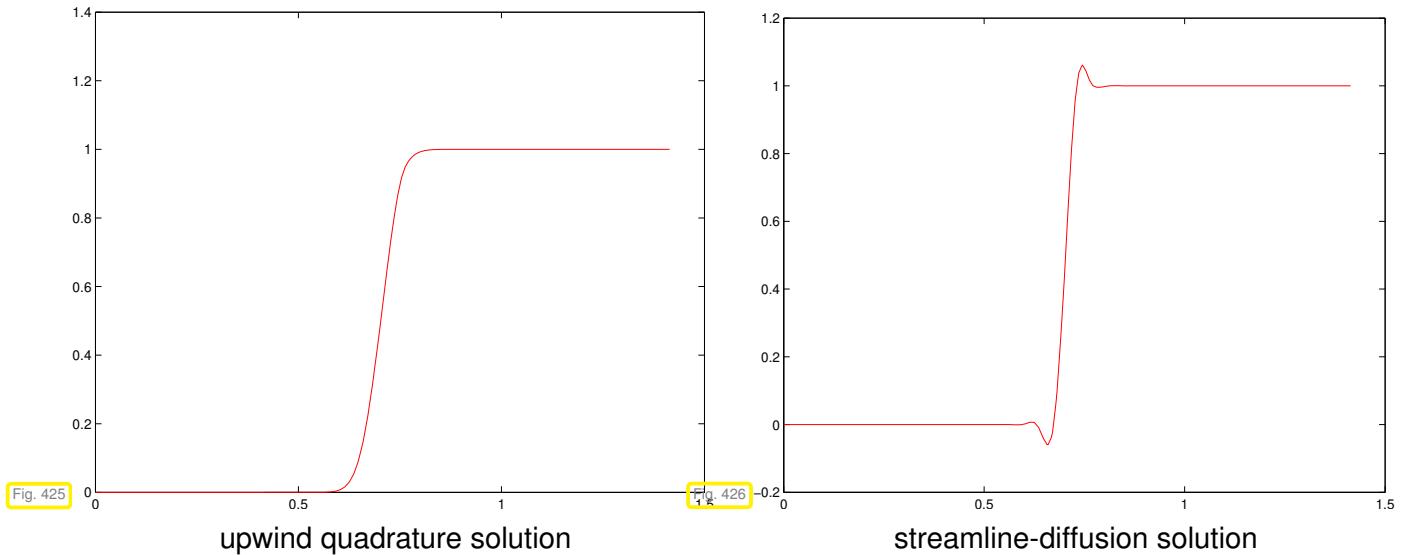
Note: In the case of Galerkin discretization based on $V_{N,0} = \mathcal{S}_1^0(\mathcal{M})$, we even find $\Delta u_h = 0$ in each $K \in \mathcal{M}$.

For the Galerkin discretization of (10.2.2.35) by means of linear Lagrangian finite elements, the local control parameters δ_K are usually chosen according to the rule

$$\delta_K := \begin{cases} \epsilon^{-1} h_K^2 & , \text{if } \frac{\|\mathbf{v}\|_{K,\infty} h_K}{2\epsilon} \leq 1 , \\ h_K & , \text{if } \frac{\|\mathbf{v}\|_{K,\infty} h_K}{2\epsilon} > 1 . \end{cases}$$

which is suggested by theoretical investigations and practical experience, cf. 1D artificial diffusion (10.2.2.8) for a reason why to choose $\delta_K \sim h_K$ for small ϵ .

EXPERIMENT 10.2.2.37 (Streamline-diffusion discretization: Internal layer) Exactly the same setting as in Ex. 10.2.2.20 with the upwind quadrature approach replaced with the streamline diffusion method.



Observations:

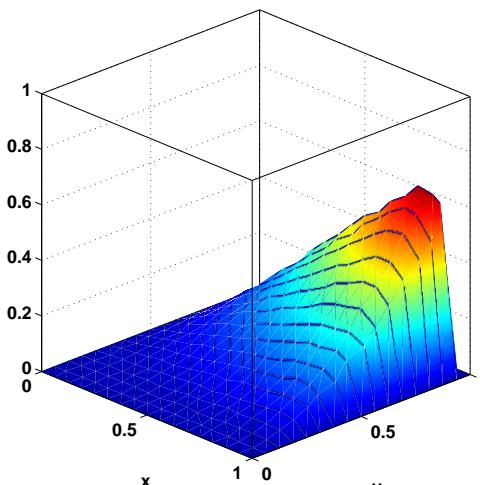
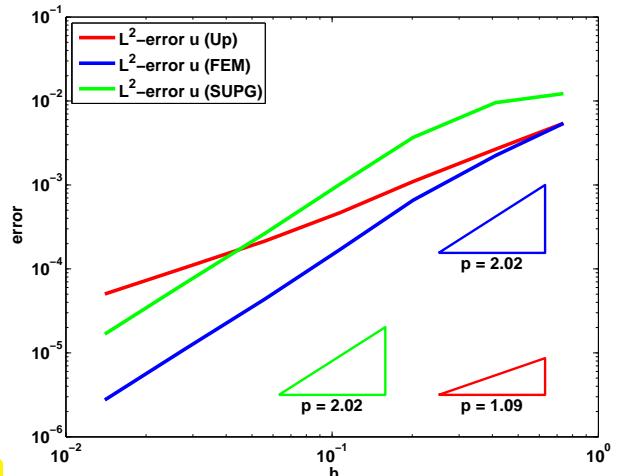
- The streamline upwind method does not exactly respect the maximum principle, but offers a better resolution of the internal layer compared with upwind quadrature (Parlance: streamline diffusion method is “less diffusive”).

EXPERIMENT 10.2.2.38 (Convergence of streamline-diffusion and upwind quadrature FEM)

- ♦ $\Omega = [0, 1]^2$, model problem (10.2.0.1), $\mathbf{v}(x) = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$, right hand side f such that

$$u_\epsilon(x, y) = xy^2 - y^2 e^{2\frac{x-1}{\epsilon}} - xe^{3\frac{y-1}{\epsilon}} + e^{2\frac{x-1}{\epsilon} + 3\frac{y-1}{\epsilon}} .$$

- ♦ Finite element discretization, $V_{0,h} = \mathcal{S}_1^0(\mathcal{M})$ und sequence of unstructured triangular “uniform” meshes, with
 - upwind quadrature stabilization from Sect. 10.2.2.1,
 - SUPG stabilization according to (10.2.2.35).
- ♦ Monitored: (Approximate) $L^2(\Omega)$ -norm of discretization error (computed with high-order local quadrature)

 u_ϵ for $\epsilon = 1$ Convergence for $\epsilon = 1$

Observation: SUPG stabilization does not affect $\mathcal{O}(h_M^2)$ -convergence of $\|u - u_h\|_{L^2(\Omega)}$ for h -refinement and $h_M \rightarrow 0$, whereas upwind quadrature leads to worse $\mathcal{O}(h_M)$ convergence of the L^2 -error norm.

Review question(s) 10.2.2.39 (Streamline diffusion)

(Q10.2.2.39.A) Write down a second-order differential operator that describes diffusion only in radial direction on the unit disk domain $\Omega := \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\|_2 < 1\}$.

△

10.3 Discretization of Time-Dependent (Transient) Convection-Diffusion IBVPs

Section 10.1.4 introduced a model for transient heat conduction in a fluid, whose motion is described by a non-stationary velocity field (\rightarrow Section 10.1.1) $\mathbf{v} : \Omega \times]0, T[\mapsto \mathbb{R}^d$

$$\frac{\partial}{\partial t}(\rho u) - \operatorname{div}(\kappa \operatorname{grad} u) + \operatorname{div}(\rho \mathbf{v}(\mathbf{x}, t) u) = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \quad (10.1.4.1)$$

where $u = u(\mathbf{x}, t) : \tilde{\Omega} \mapsto \mathbb{R}$ is the unknown temperature. Highlighted is the **convective term**. As usual, the differential operators **grad** and **div** act only on the spatial variable \mathbf{x} .

Assuming the **incompressibility condition** $\operatorname{div} \mathbf{v}(\mathbf{x}, t) = 0$, as in Section 10.2, by scaling (\rightarrow Rem. 1.2.1.25, Rem. 10.1.4.3) we arrive at the non-dimensional model equation for transient convection-diffusion.

$$\frac{\partial u}{\partial t} - \epsilon \Delta u + \mathbf{v}(\mathbf{x}, t) \cdot \operatorname{grad} u = f \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \quad (10.3.0.1)$$

to be supplemented with,

- ◆ for instance, Dirichlet boundary conditions: $u(\mathbf{x}, t) = g(\mathbf{x}, t) \quad \forall \mathbf{x} \in \partial\Omega, \quad 0 < t < T$, or other valid boundary conditions for scalar second-order BVPs, see § 10.1.2.9),
- ◆ initial conditions: $u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega$.

As explained in Section 10.1.2 other types of boundary conditions can be imposed; all boundary conditions appropriate for scalar second-order elliptic BVPs make sense for (10.3.0.1).

Of course, our goal is to devise a discretization of the initial-boundary value problem for (10.3.0.1) that performs well regardless of the local size of ϵ and \mathbf{v} . In other words, the schemes should be **robust** with respect to ϵ and \mathbf{v} .

10.3.1 Convection-Diffusion IBVPs: Method of Lines



Video tutorial for Section 10.3.1: Convection-Diffusion IBVPs: Method of Lines : (25 minutes)
[Download link](#), [tablet notes](#)

For the solution of the IBVP (10.3.0.1) we follow the general method-of-lines policy introduced in Section 9.2.4:

- ① (Finite element) discretization in space on a **fixed** mesh \Rightarrow initial value problem for ODE
- ② Discretization in time (by suitable numerical integrator = timestepping)

Spatial discretization treats time t as a parameter and can rely on any of the methods discussed in Section 10.2. The need for upwinding will be addressed below. For instance, in the case of Dirichlet boundary conditions,

$$\begin{cases} \frac{\partial u}{\partial t} - \epsilon \Delta u + \mathbf{v}(x, t) \cdot \mathbf{grad} u = f & \text{in } \tilde{\Omega} := \Omega \times]0, T[, \\ u(x, t) = g(x, t) & \forall x \in \partial\Omega, 0 < t < T , \quad u(x, 0) = u_0(x) & \forall x \in \Omega . \end{cases} \quad (10.3.1.1)$$

← spatial discretization

$$\mathbf{M} \frac{d\vec{\mu}}{dt}(t) + \epsilon \mathbf{A} \vec{\mu}(t) + \mathbf{B}(t) \vec{\mu}(t) = \vec{\phi}(t) , \quad (10.3.1.2)$$

where

- ◆ $\vec{\mu} = \vec{\mu}(t) :]0, T[\mapsto \mathbb{R}^N \hat{=} \text{coefficient vector describing approximation } u_h(t) \text{ of } u(\cdot, t)$,
- ◆ $\mathbf{A} \in \mathbb{R}^{N,N} \hat{=} \text{s.p.d. matrix of discretized } -\Delta$, e.g., (finite element) Galerkin matrix,
- ◆ $\mathbf{M} \in \mathbb{R}^{N,N} \hat{=} \text{(lumped} \rightarrow \text{Rem. 9.3.4.18) mass matrix}$
- ◆ $\mathbf{B} \in \mathbb{R}^{N,N} \hat{=} \text{possibly time-dependent matrix for discretized convective term, e.g., Galerkin matrix, upwind quadrature matrix} (\rightarrow \text{Sect. 10.2.2.1}), \text{ streamline diffusion matrix} (\rightarrow \text{Sect. 10.2.2.2}).$
- ◆ $\vec{\phi}(t) \hat{=} \text{r.h.s. vector taking into account the source function } f \text{ and the boundary data } g.$

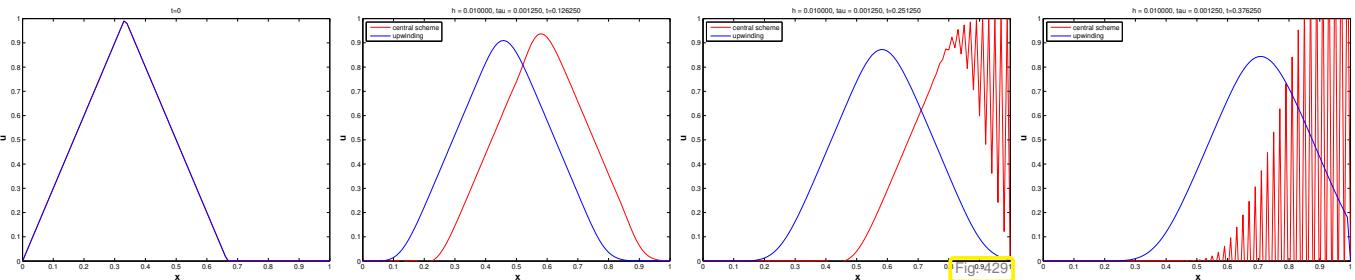
The following numerical test investigates in one spatial dimension we want to obtain evidence, which spatial discretizations and timestepping methods may be suitable for the transient convection diffusion problem.

EXPERIMENT 10.3.1.3 (Implicit Euler method of lines for transient convection-diffusion) We consider the 1D convection-diffusion IBVP:

$$\begin{aligned} \frac{\partial u}{\partial t} - \epsilon \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x} &= 0 & \text{in }]0, 1[\times [0, 1] , \\ u(x, 0) &= \max(1 - 3|x - \frac{1}{3}|, 0) , \quad 0 < x < 1 , \quad u(0, t) = u(1, t) = 0 , \quad 0 \leq t \leq 1 . \end{aligned} \quad (10.3.1.4)$$

- ◆ Spatial discretization on equidistant mesh with meshwidth $h = 1/N$:
 1. central finite difference scheme, see (10.2.2.2) (\leftrightarrow linear FE Galerkin discretization),
 2. upwind finite difference discretization, see (10.2.2.8),
- ◆ $M = hI$ (“lumped” mass matrix, see Rem. 9.3.4.18),
- ◆ Temporal discretization with uniform timestep $\tau > 0$:
 1. implicit Euler method, see (9.2.7.3),
 2. explicit Euler method, see (9.2.7.2),

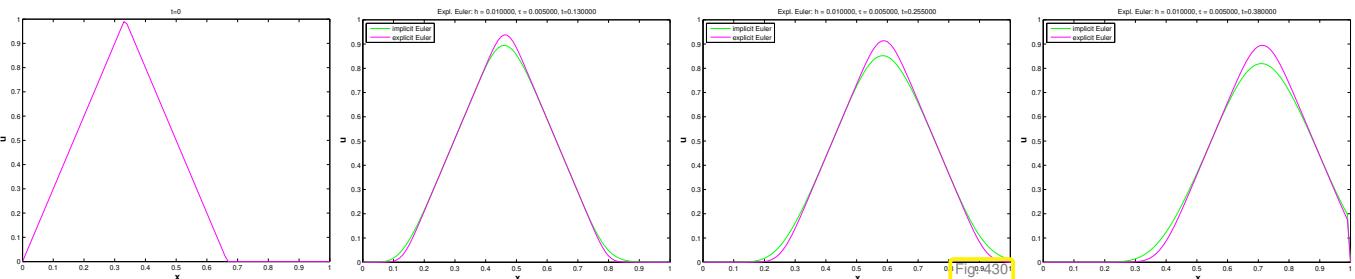
Computations with $\epsilon = 10^{-5}$, implicit Euler discretization, $h = 0.01$, $\tau = 0.00125$:



Observation:

- Central finite differences display spurious oscillations as in Ex. 10.2.2.4.
- Upwinding suppresses spurious oscillations, but introduces *spurious damping*.

Test of different 1st-order timestepping schemes in transport-dominated case; computations with $\epsilon = 10^{-5}$, spatial upwind discretization, $h = 0.01$, $\tau = 0.005$:



Observation: implicit Euler timestepping causes stronger spurious damping than explicit Euler timestepping.

However, explicit Euler is subject to tight stability induced timestep constraint for larger values of ϵ , see Section 9.2.7.2. □

Also in the transient setting we will face a singular perturbation situation (\rightarrow Notion 10.2.1.9), if $\epsilon \ll \|v(x, t)\|$ for some $(x, t) \in \bar{\Omega}$. As we have seen in Exp. 10.3.1.3, this restricts the choice of spatial discretization options.

For the spatial discretization for method of lines approach to *singularly perturbed* transient convection-diffusion IBVPs use **ϵ -robustly stable** spatial discretization of convective term.

Remark 10.3.1.5 (Choice of timestepping for m.o.l. for transient convection-diffusion) If ϵ -robustness **for all** $\epsilon > 0$ (including $\epsilon > 1$) desired \geq Arguments of Section 9.2.7.2 stipulate use of

$L(\pi)$ -stable (\rightarrow Def. 9.2.7.46) timestepping methods (implicit Euler (9.2.7.3), RADAU-3 (9.2.7.50), SDIRK-2 (9.2.7.51))

In the *singularly perturbed case* $0 < \epsilon \ll 1$ conditionally stable explicit timestepping is an option, due to a timestep constraint of the form " $\tau < O(h_M)$ ", which does not interfere with efficiency, cf. the discussion in Section 9.2.8. \downarrow

Review question(s) 10.3.1.6 (Method of Lines for transient convection-diffusion problems)

(Q10.3.1.6.A) We consider the initial-boundary value problem

$$\begin{aligned} \frac{\partial u}{\partial t} - \epsilon \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x} &= 0 \quad \text{in }]0,1[\times [0,1] , \\ u(x,0) = u_0(x), \quad 0 < x < 1, \quad u(0,t) = u(1,t) = 0, \quad 0 \leq t \leq 1, \end{aligned}$$

with $u_0 \in C_0^0([0,1])$, $u_0(0) = u_0(1) = 0$. We perform a full discretization on an equidistant mesh with spatial meshwidth $h > 0$ and timestep size $\tau > 0$,

- relying on piecewise linear finite-element Galerkin discretization with **upwind quadrature** and **mass lumping** in space,
- and implicit Euler timestepping in time.

Elaborate the discrete evolution for the basis expansion coefficient vectors using the standard tent function basis.

Hint. For the transport term the spatial discretization is equivalent to a simple upwind finite difference scheme, for the diffusion term to a simple symmetric second difference quotient.

Δ

10.3.2 Transport Equation



Video tutorial for Section 10.3.2: Transport Equation: (14 minutes) [Download link](#), [tablet notes](#)

We assume $\|\mathbf{v}\|(x,t) \approx 1$ and focus on the situation of **singular perturbation** (\rightarrow Notion 10.2.1.9): $0 < \epsilon \ll 1$. As in Section 10.2.1 the key idea is to study and understand the limit problem, obtained by setting $\epsilon := 0$.

$$\frac{\partial u}{\partial t} - \epsilon \Delta u + \mathbf{v}(x,t) \cdot \mathbf{grad} u = f \quad \text{in } \tilde{\Omega} := \Omega \times]0,T[,$$



$\leftarrow \epsilon = 0$

$$\frac{\partial u}{\partial t} + \mathbf{v}(x,t) \cdot \mathbf{grad} u = f \quad \text{in } \tilde{\Omega} := \Omega \times]0,T[. \quad (10.3.2.1)$$

The limit problem (10.3.2.1) is called the (pure) **transport equation**.

Recall that in Section 10.2.1 for the stationary pure transport problem

$$\mathbf{v}(x) \cdot \mathbf{grad} u = f(x) \quad \text{in } \Omega , \quad (10.2.1.4)$$

we found solutions by the **method of characteristics**, by integrating the source term along streamlines (following the flow direction). This leads us to study the behavior of a C^1 -solution $u = u(x,t)$ of (10.3.2.1)

“as seen from a moving fluid particle” (**Lagrangian view**). In Section 10.1.1 we have learned that particle trajectories can be found as solutions of the streamline ODE $\dot{\mathbf{y}}(t) = \mathbf{v}(\mathbf{y}(t), t)$. Therefore, we examine the function

$$t \mapsto u(\mathbf{y}(t), t) , \quad \text{where } \mathbf{y}(t) \text{ solves } \frac{d\mathbf{y}}{dt}(t) = \mathbf{v}(\mathbf{y}(t), t) , \quad \text{see (10.1.1.2).}$$

By the chain rule we see that, like in the stationary case, the change of u along streamlines is entirely due to sources:

$$\begin{aligned} \blacktriangleright \quad & \frac{d}{dt} u(\mathbf{y}(t), t) = \mathbf{grad} u(\mathbf{y}(t), t) \cdot \frac{d\mathbf{y}}{dt}(t) + \frac{\partial u}{\partial t}(\mathbf{y}(t), t) \\ &= \mathbf{grad} u(\mathbf{y}(t), t) \cdot \mathbf{v}(\mathbf{y}(t), t) + \frac{\partial u}{\partial t}(\mathbf{y}(t), t) \stackrel{(10.3.2.1)}{=} f(\mathbf{y}(t), t) . \end{aligned} \quad (10.3.2.2)$$

§10.3.2.3 (Solution formula for sourceless transport) Temporarily, we focus on the case $f \equiv 0$ (no sources) and let $u = u(\mathbf{x}, t)$ be a C^1 -solution of the pure transport equation

$$\frac{\partial u}{\partial t} + \mathbf{v}(\mathbf{x}, t) \cdot \mathbf{grad} u = 0 \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[. \quad (10.3.2.4)$$

From (10.3.2.2) we conclude that

If $f \equiv 0$, then a fluid particle “sees” a constant temperature!

We also restrict ourselves to the situation of no inflow/outflow (e.g., fluid in a container), which is characterized by velocity fields that are tangential to $\partial\Omega$:

$$\mathbf{v}(\mathbf{x}, t) \cdot \mathbf{n}(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \partial\Omega , 0 < t < T . \quad (10.1.1.5)$$

- all streamlines will “stay inside Ω ”, flow map Φ^t (10.1.1.6) defined for all times $t \in \mathbb{R}$ as we saw in § 10.1.1.3.

Next, we recall (10.3.2.2) and its message that u is constant on streamlines. This gives a **method-of-characteristics (MOC)** solution formula for (10.3.2.4).

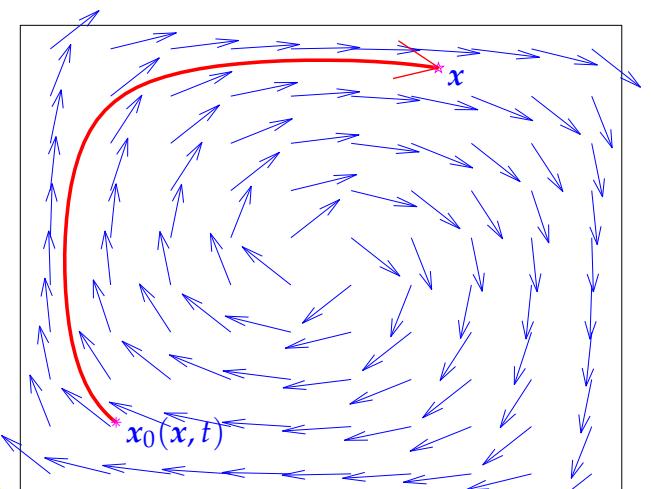
For the initial value problem:

$$\begin{aligned} \frac{\partial u}{\partial t}(\mathbf{x}, t) + \mathbf{v}(\mathbf{x}, t) \cdot \mathbf{grad} u(\mathbf{x}, t) &= 0 \quad \text{in } \tilde{\Omega} , \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega , \end{aligned}$$

we find the exact solution

$$u(\mathbf{x}, t) = u_0(x_0(\mathbf{x}, t)) , \quad (10.3.2.5)$$

where $x_0(\mathbf{x}, t)$ is the position at time 0 of the fluid particle that is located at \mathbf{x} at time t . We construct u by **backtracking along streamlines**.



§10.3.2.6 (Method-of-characteristics solution formula for transport equation) The solution formula (10.3.2.5) can be generalized to any velocity field $\mathbf{v} : \Omega \mapsto \mathbb{R}^d$ and $f \neq 0$. The new aspect is that streamlines can *enter* and *leave* the domain Ω . Therefore we have to impose Dirichlet boundary conditions on the **inflow boundary**

$$u(\mathbf{x}, t) = g(\mathbf{x}, t) \quad \text{for } \mathbf{x} \in \Gamma_{\text{in}} := \{\mathbf{x} \in \partial\Omega : \mathbf{v}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) < 0\} , \quad \text{cf. (10.2.1.11).}$$

Thus, for some parts of $\tilde{\Omega} := \Omega \times]0, T[$ the solution values $u(\mathbf{x}, t)$ are given by “transported Dirichlet data”, if the starting point of the streamline lies on Γ_{in} :

$$\frac{d}{dt} u(\mathbf{y}(t)) = f(\mathbf{y}(t), t) \quad \text{where} \quad \dot{\mathbf{y}}(t) = \mathbf{v}(\mathbf{y}(t), t) .$$

$$\Rightarrow u(\mathbf{x}, t) = \begin{cases} u_0(\mathbf{x}_0) + \int_0^t f(\mathbf{y}(s), s) ds & , \text{if } \mathbf{y}(s) \in \Omega \quad \forall 0 < s < t , \\ g(\mathbf{y}(s_0), s_0) + \int_{s_0}^t f(\mathbf{y}(s), s) ds & , \text{if } \mathbf{y}(s_0) \in \partial\Omega, \mathbf{y}(s) \in \Omega \quad \forall s_0 < s < t , \end{cases} \quad (10.3.2.7)$$

for $(\mathbf{x}, t) \in \tilde{\Omega}$.

We make the typical observation for a singularly perturbed problem as discussed in Section 10.2.1: Whereas for $\epsilon > 0$ Dirichlet boundary conditions can be imposed everywhere on $\partial\Omega$, they have to be confined to the inflow boundary for $\epsilon = 0$. \square

Review question(s) 10.3.2.8 (Transport equation)

(Q10.3.2.8.A) We consider the following initial-boundary value problem for the pure advection equation in one spatial dimension:

$$\frac{\partial u}{\partial t}(x, t) + \frac{\partial u}{\partial x}(x, t) = 0 \quad \text{on }]0, 1[\times]0, 1[,$$

$$u(0, t) = u(1, t) = 0 \quad \forall 0 < t < 1 , \quad u(x, 0) = \sin(\pi x) \quad \forall 0 < x < 1 .$$

Using the method of characteristics give a formula for the solution.

Hint. We face a situation with constant transport velocity $v = 1$.

(Q10.3.2.8.B) Let $\Omega := \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\|_2 < 1\}$ denote the unit disk, on which the following transient advection problem is posed:

$$\frac{\partial u}{\partial t} + \mathbf{v}(\mathbf{x}) \cdot \mathbf{grad} u = 0 \quad \text{on } \Omega \times]0, 1[,$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{on } \Omega ,$$

with

$$\mathbf{v}(\mathbf{x}) := \begin{bmatrix} -x_2 \\ x_1 \end{bmatrix} , \quad \mathbf{x} := \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} ,$$

and u_0 a continuous function compactly supported in Ω .

- Why can we dispense with specifying boundary values?
- Rely on the method of characteristics to find an explicit formula for the solution of this initial-value problem.

(Q10.3.2.8.C) Let $\Omega \subset \mathbb{R}^d$ stand for a bounded computational domain. Formulate a **maximum principle** for the solutions of the IVP:

$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \mathbf{grad} u = f \quad \text{in } \tilde{\Omega} \times]0, T[, \quad (10.3.2.9)$$

$$u(\mathbf{x}, t) = 0 \quad \text{on } \Gamma_{\text{in}} := \{\mathbf{x} \in \partial\Omega : \mathbf{v}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) < 0\} , \quad (10.3.2.10)$$

where the velocity field $\mathbf{v} : \tilde{\Omega} \rightarrow \mathbb{R}^d$, $\mathbf{v} \in C^1(\bar{\Omega})$ and the Dirichlet data $g \in C^0(\Gamma_{\text{in}})$ are given.

Hint. Remember that (10.3.2.9) is a mathematical model for heat transport in a moving fluid and that $u = u(\mathbf{x}, t)$ can be viewed as a time-dependent temperature field.

(Q10.3.2.8.D) Let $u = u(\mathbf{x}, t)$ be a classical solution of a sourceless ($f \equiv 0$) transient pure transport problem on the rectangular domain Ω draw below.

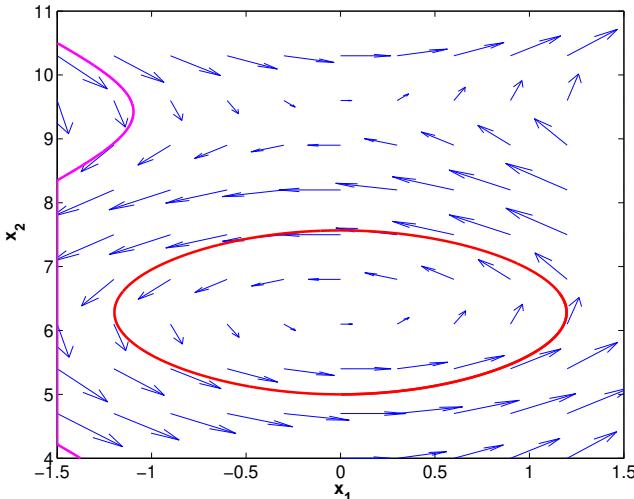


Fig. 432

The blue arrows indicate the involved velocity field $\mathbf{v} : \Omega \rightarrow \mathbb{R}^2$.

You want to use the **method-of-characteristics** solution formula to find $u(\mathbf{x}, t)$ for

- \mathbf{x} on the closed red curve,
- \mathbf{x} on the magenta curve.

Write down the formula for $u(\mathbf{x}, t)$. You may have to distinguish several cases.

△

10.3.3 Lagrangian Split-Step Method



Video tutorial for Section 10.3.3: Lagrangian Split-Step Method: (40 minutes) [Download link](#), [tablet notes](#)

In this section we develop an alternative to the method-of-lines approach, which belongs to the class of **Lagrangian discretization schemes**. Lagrangian discretization schemes for the IBVP (10.3.1.1) are inspired by the method-of-characteristics solution formulas (10.3.2.5) and (10.3.2.7).

The variant that we are going to study separates the transient convection-diffusion problem into a pure diffusion problem (heat equation → Section 9.2.1) and a pure transport problem (10.3.2.1). This is achieved by means of a particular approach to timestepping that will be introduced next.

10.3.3.1 Split-Step Timestepping, cf. Section 7.5

Unfortunately, the method-of-characteristics solution formulas (10.3.2.5) and (10.3.2.7) are available only for the pure transport problem. If $\epsilon > 0$, split-step timestepping offers a technique for dissecting a convection-diffusion evolution problem into a pure diffusion and a pure transport problem.

We first develop the method for an abstract ODE, whose right hand side is the sum of two (smooth) functions

$$\dot{\mathbf{y}} = \mathbf{g}(t, \mathbf{y}) + \mathbf{r}(t, \mathbf{y}), \quad \mathbf{g}, \mathbf{r} : [0, T] \times \mathbb{R}^m \mapsto \mathbb{R}^m. \quad (10.3.3.1)$$



Idea: In turns solve

$$(1) \quad \dot{\mathbf{y}}(t) = \mathbf{g}(t, \mathbf{y}), \\ (2) \quad \dot{\mathbf{y}}(t) = \mathbf{r}(t, \mathbf{y}),$$

over small timesteps.

This idea offers great benefits, if one has efficient methods or even analytic formulas to solve initial value problems for both ODEs $\dot{\mathbf{z}} = \mathbf{g}(t, \mathbf{z})$ and $\dot{\mathbf{w}} = \mathbf{r}(t, \mathbf{w})$. The details of a particular variant are as follows, using a temporal mesh $\{0 = t_0 < t_1 < \dots < t_M := T\}$:

Strang splitting single step method for (10.3.3.1), timestep $\tau := t_j - t_{j-1} > 0$:

compute $\mathbf{y}^{(j)} \approx \mathbf{y}(t_j)$ from $\mathbf{y}^{(j-1)} \approx \mathbf{y}(t_{j-1})$ according to

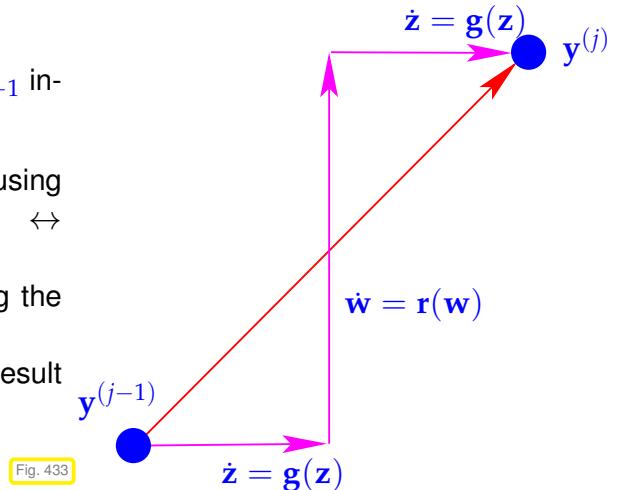
$$\tilde{\mathbf{y}} := \mathbf{z}(t_{j-1} + \frac{1}{2}\tau), \quad \text{where } \mathbf{z}(t) \text{ solves } \dot{\mathbf{z}} = \mathbf{g}(t, \mathbf{z}), \quad \mathbf{z}(t_{j-1}) = \mathbf{y}^{(j-1)}, \quad (10.3.3.2)$$

$$\hat{\mathbf{y}} := \mathbf{w}(t_j) \quad \text{where } \mathbf{w}(t) \text{ solves } \dot{\mathbf{w}} = \mathbf{r}(t, \mathbf{w}), \quad \mathbf{w}(t_{j-1}) = \tilde{\mathbf{y}}, \quad (10.3.3.3)$$

$$\mathbf{y}^{(j)} := \mathbf{z}(t_j), \quad \text{where } \mathbf{z}(t) \text{ solves } \dot{\mathbf{z}} = \mathbf{g}(t, \mathbf{z}), \quad \mathbf{z}(t_{j-1} + \frac{1}{2}\tau) = \hat{\mathbf{y}}. \quad (10.3.3.4)$$

One timestep of the split-step method of size $\tau := t_j - t_{j-1}$ involves three sub-steps:

- ① Solve $\dot{\mathbf{z}} = \mathbf{g}(t, \mathbf{z})$ over period $[t_{j-1}, t_{j-1} + \frac{1}{2}\tau]$ using the result of the previous timestep as initial value \leftrightarrow (10.3.3.2).
- ② Solve $\dot{\mathbf{w}} = \mathbf{r}(t, \mathbf{w})$ over time period $[t_{j-1}, t_j]$ using the result of ① as initial value \leftrightarrow (10.3.3.3).
- ③ Solve $\dot{\mathbf{z}} = \mathbf{g}(t, \mathbf{z})$ over time $[t_{j-1} + \frac{1}{2}\tau, t_j]$ using the result of ② as initial value \leftrightarrow (10.3.3.4).



In Section 7.5 we have already learned about the following result:

Theorem 10.3.3.5. Order of Strang splitting single step method

Assuming exact solution of the initial value problems of the sub-steps, the Strang splitting single step method for (10.3.3.1) is of second order.

This result applies to Strang splitting timestepping for initial value problems for ODEs. Now we boldly regard (10.3.1.1) as an “*ODE in function space*” for the unknown “function space valued function” $\mathbf{u} = \mathbf{u}(t) : [0, T] \mapsto H^1(\Omega)$.

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= \epsilon \Delta \mathbf{u} + \mathbf{f} - \mathbf{v} \cdot \mathbf{grad} \mathbf{u} \\ \uparrow &\quad \uparrow & \uparrow \\ \dot{\mathbf{y}} &= \mathbf{g}(\mathbf{y}) + \mathbf{r}(\mathbf{y}) \end{aligned}$$

Formally, we arrive at the following “timestepping scheme in function space” for (10.3.0.1) on a temporal mesh $0 = t_0 < t_1 < \dots < t_M := T$:

We look at a single timestep and assume that we are given approximation $\mathbf{u}^{(j-1)} \approx \mathbf{u}(t_{j-1})$. Then the concrete steps of the Strang split-step method for the IBVP

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} - \epsilon \Delta \mathbf{u} + \mathbf{v}(\mathbf{x}, t) \cdot \mathbf{grad} \mathbf{u} = \mathbf{f} & \text{in } \tilde{\Omega} := \Omega \times [0, T], \\ \mathbf{u}(\mathbf{x}, t) = \mathbf{g}(\mathbf{x}, t) \quad \forall \mathbf{x} \in \partial\Omega, 0 < t < T, \quad \mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega, \end{cases} \quad (10.3.1.1)$$

are:

① Solve (autonomous) parabolic IBVP for *pure diffusion* from t_{j-1} to $t_{j-1} + \frac{1}{2}\tau$

$$(10.3.3.2) \leftrightarrow \begin{aligned} \frac{\partial w}{\partial t} - \epsilon \Delta w &= 0 \quad \text{in } \Omega \times]t_{j-1}, t_{j-1} + \frac{1}{2}\tau[, \\ w(\mathbf{x}, t) &= g(\mathbf{x}, t) \quad \forall \mathbf{x} \in \partial\Omega, t_{j-1} < t < t_{j-1} + \frac{1}{2}\tau , \\ w(\mathbf{x}, t_{j-1}) &= u^{(j-1)}(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega . \end{aligned} \quad (10.3.3.6)$$

② Solve IBVP for *pure transport* (= **advection**), see Sect. 10.3.2,

$$(10.3.3.3) \leftrightarrow \begin{aligned} \frac{\partial z}{\partial t} + \mathbf{v}(\mathbf{x}, t) \cdot \mathbf{grad} z &= f(\mathbf{x}, t) \quad \text{in } \Omega \times]t_{j-1}, t_j[, \\ z(\mathbf{x}, t) &= g(\mathbf{x}, t) \quad \text{on inflow boundary } \Gamma_{\text{in}}, t_{j-1} < t < t_j , \\ z(\mathbf{x}, t_{j-1}) &= w(\mathbf{x}, t_{j-1} + \frac{1}{2}\tau) \quad \forall \mathbf{x} \in \Omega . \end{aligned} \quad (10.3.3.7)$$

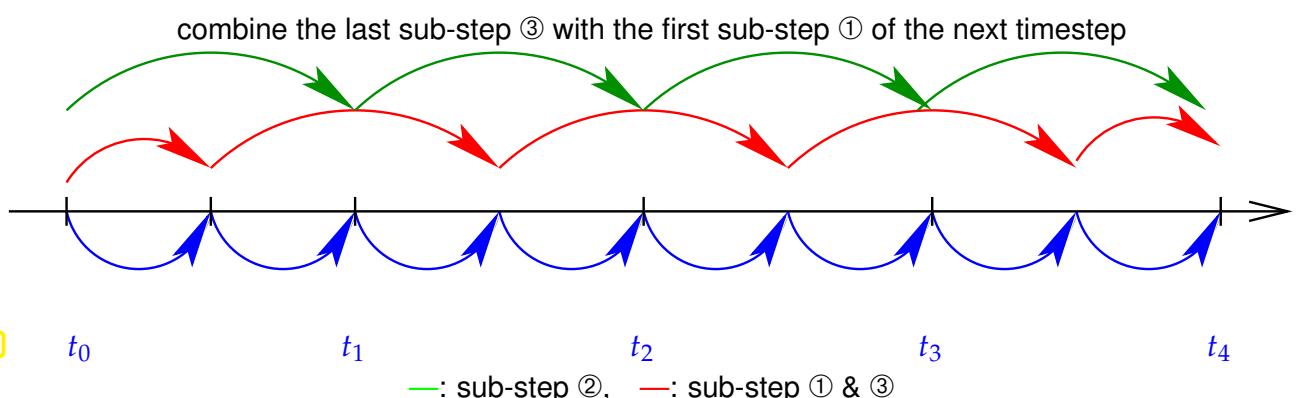
③ Solve (autonomous) parabolic IBVP for *pure diffusion* from $t_{j-1} + \frac{1}{2}\tau$ to t_j

$$(10.3.3.4) \leftrightarrow \begin{aligned} \frac{\partial w}{\partial t} - \epsilon \Delta w &= 0 \quad \text{in } \Omega \times]t_{j-1} + \frac{1}{2}\tau, t_j[, \\ w(\mathbf{x}, t) &= g(\mathbf{x}, t) \quad \forall \mathbf{x} \in \partial\Omega, t_{j-1} + \frac{1}{2}\tau < t < t_j , \\ w(\mathbf{x}, t_{j-1} + \frac{1}{2}\tau) &= z(\mathbf{x}, t_j) \quad \forall \mathbf{x} \in \Omega . \end{aligned} \quad (10.3.3.8)$$

Then set $u^{(j)}(\mathbf{x}) := w(\mathbf{x}, t_j), \mathbf{x} \in \Omega$.

Remark 10.3.3.9 (Leap-frog implementation of Strang splitting)

An efficient “implementation” of Strang splitting timestepping can merge sub-steps belonging to different timesteps. If $\mathbf{g} = \mathbf{g}(\mathbf{y})$ we can



Remark 10.3.3.10 (Approximate sub-steps for Strang splitting time) The solutions of the initial value problems in the sub-steps of Strang splitting timestepping may be computed *only approximately*.

If this is done by one step of a 2nd-order timestepping method in each case, then the resulting approximate Strang splitting timestepping will still be of second order, cf. Thm. 10.3.3.5.

10.3.3.2 Particle Method for Pure Transport (Advection)

We recall the method of characteristics for IBVPs for the pure transport (= **advection**) equation from Section 10.3.2

$$\begin{aligned} \frac{\partial u}{\partial t} + \mathbf{v}(\mathbf{x}, t) \cdot \mathbf{grad} u &= f \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \\ u(\mathbf{x}, t) &= g(\mathbf{x}, t) \quad \text{on } \Gamma_{\text{in}} \times]0, T[, \quad u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{in } \Omega , \end{aligned} \quad (10.3.3.11)$$

where the **inflow boundary** is defined as

$$\Gamma_{\text{in}} := \{x \in \partial\Omega : \mathbf{v}(x) \cdot \mathbf{n}(x) < 0\} . \quad (10.2.1.11)$$

Case $f \equiv 0$: a travelling fluid particle sees a constant solution, see (10.3.2.2)

$$\blacktriangleright u(x, t) = \begin{cases} u_0(x_0) & , \text{ if } \mathbf{y}(s) \in \Omega \quad \forall 0 < s < t , \\ g(\mathbf{y}(s_0), s_0) & , \text{ if } \mathbf{y}(s_0) \in \partial\Omega, \mathbf{y}(s) \in \Omega \quad \forall s_0 < s < t , \end{cases} \quad (10.3.3.12)$$

for $(x, t) \in \tilde{\Omega}$, where $s \mapsto \mathbf{y}(s)$ solves the initial value problem for the streamline ODE

$$\frac{d\mathbf{y}}{ds}(s) = \mathbf{v}(\mathbf{y}(s), s) , \quad \mathbf{y}(t) = x , \quad (10.3.3.13)$$

and, thus, defines a “backward particle trajectory”.

Case of general f , see § 10.3.2.3: Since $\frac{d}{dt}u(\mathbf{y}(t)) = f(\mathbf{y}(t), t)$ we can obtain the solution by integrating the source along a particle trajectory:

$$\blacktriangleright u(x, t) = \begin{cases} u_0(x_0) + \int_0^t f(\mathbf{y}(s), s) ds & , \text{ if } \mathbf{y}(s) \in \Omega \quad \forall 0 < s < t , \\ g(\mathbf{y}(s_0), s_0) + \int_{s_0}^t f(\mathbf{y}(s), s) ds & , \text{ if } \mathbf{y}(s_0) \in \partial\Omega, \mathbf{y}(s) \in \Omega \quad \forall s_0 < s < t . \end{cases} \quad (10.3.2.7)$$

The solution formula (10.3.2.7) suggests an approach for solving (10.3.3.11) approximately by following particles in the flow. This is the idea underlying **particle methods**.

We first consider the simple situation of no inflow/outflow (e.g., fluid in a container, see § 10.3.2.3)

$$\mathbf{v}(x, t) \cdot \mathbf{n}(x) = 0 \quad \forall x \in \partial\Omega , 0 < t < T . \quad (10.1.1.5)$$

In this case boundary conditions become irrelevant when solving a pure transport problem.

Then the particle method for the solution of the pure transport initial value problem proceeds as follows:

- ① Pick suitable **interpolation nodes** $\{\mathbf{p}_i\}_{i=1}^N \subset \Omega$ (initial ‘particle positions’)
- ② “**Particle pushing**”: Solve initial value problems (cf. ODE (10.3.3.13) for particle trajectories)

$$\dot{\mathbf{y}}(t) = \mathbf{v}(\mathbf{y}(t), t) , \quad \mathbf{y}(0) = \mathbf{p}_i , \quad i = 1, \dots, N ,$$

by means of a suitable single-step method with uniform timestep $\tau := T/M$, $M \in \mathbb{N}$.

\blacktriangleright sequences of solution points $\mathbf{p}_i^{(j)}, j = 0, \dots, M, i = 1, \dots, N$

- ③ **Reconstruct** approximation $u_h^{(j)} \approx u(\cdot, t_j)$, $t_j := j\tau$, by interpolation: we demand

$$u_h^{(j)}(\mathbf{p}_i^{(j)}) := u_0(\mathbf{p}_i) + \tau \sum_{l=1}^{j-1} f(\frac{1}{2}(\mathbf{p}_i^{(l)} + \mathbf{p}_i^{(l-1)}), \frac{1}{2}(t_l + t_{l-1})) , \quad i = 1, \dots, N .$$

In the third step the equidistant composite midpoint quadrature rule was used to approximate the source integral in (10.3.2.7).

Remark 10.3.3.14. This method falls into the class of

- **particle methods**, because the interpolation nodes can be regarded fluid particles tracked by the method,
- **Lagrangian methods**, which treat the IBVP in coordinate systems moving with the flow,
- **characteristic methods**, which reconstruct the solution from knowledge about its behavior along streamlines.

]

Remark 10.3.3.15 (Particle method adapted to inflow/outflow) General velocity fields $\mathbf{v} : \Omega \mapsto \mathbb{R}^d$ that penetrate $\partial\Omega$ can be dealt with by the following modifications of the particle method:

- ◆ Stop tracking i -th trajectory as soon as an interpolation nodes $\mathbf{p}_i^{(j)}$ lies outside spatial domain Ω .
- ◆ In each timestep start new trajectories from fixed locations on inflow boundary Γ_{in} (“particle injection”). These interpolation nodes will carry the boundary value.

]

EXPERIMENT 10.3.3.16 (Point particle method for pure advection) In this experiment we take a qualitative look at the ability of particle methods to simulate a rotating flow.

- ◆ IBVP (10.3.3.11) on $\Omega = [0, 1]^2$, $T = 2$, with $f \equiv 0$, $g \equiv 0$.
- ◆ Initial locally supported bump $u_0(\mathbf{x}) = \max\{0, 1 - 4\|\mathbf{x} - \begin{bmatrix} 1/2 \\ 1/4 \end{bmatrix}\|\}$.
- ◆ Two stationary divergence-free velocity fields
 - $\mathbf{v}_1(\mathbf{x}) = \begin{bmatrix} -\sin(\pi x_1) \cos(\pi x_2) \\ \cos(\pi x_1) \sin(\pi x_2) \end{bmatrix}$ satisfying (10.1.1.5),
 - $\mathbf{v}_2(\mathbf{x}) = \begin{bmatrix} -x_2 \\ x_1 \end{bmatrix}$.
- ◆ Initial positions of interpolation points on regular tensor product grid with meshwidth $h = \frac{1}{40}$.
- ◆ Approximation of trajectories by means of explicit trapezoidal rule [Hip19, ??] (method of Heun).

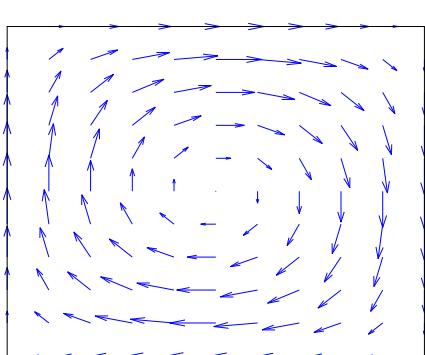


Fig. 435

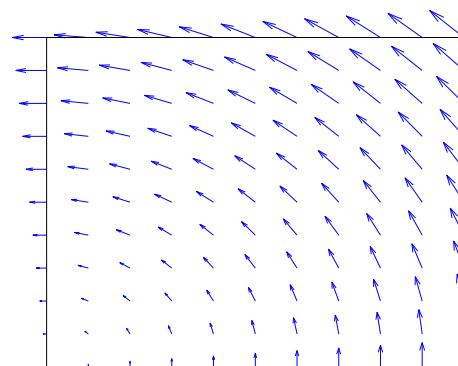
velocity field \mathbf{v}_1 (circvel)

Fig. 436

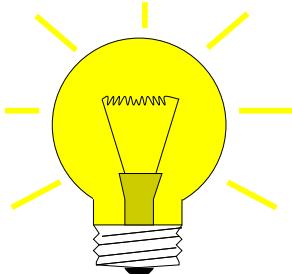
velocity field \mathbf{v}_2 (rotvel)

]

10.3.3.3 Particle Mesh Method (PMM)

The method introduced in the previous section, can be used to tackle the pure advection problem (10.3.3.7) that we face in 2nd sub-step (10.3.3.7) of the Strang splitting timestepping.

Issue: How to combine Lagrangian advection with a method for the pure diffusion problem (10.3.3.6) faced in the other sub-steps of the Strang splitting timestepping?



Idea: two views

"particle temperatures" $u(\mathbf{p}_i^{(j)})$



Nodal values of finite element function $u_h^{(j)} \in \mathcal{S}_1^0(\mathcal{M})$

► We give the outline of an algorithm for one timestep of size $\tau > 0$ of Strang splitting timestepping for the transient convection-diffusion problem with zero Dirichlet boundary conditions

$$\begin{cases} \frac{\partial u}{\partial t} - \epsilon \Delta u + \mathbf{v}(x) \cdot \mathbf{grad} u = f & \text{in } \tilde{\Omega} := \Omega \times [0, T], \\ u(x, t) = 0 & \forall x \in \partial\Omega, 0 < t < T, \quad u(x, 0) = u_0(x) \quad \forall x \in \Omega. \end{cases} \quad (10.3.1.1)$$

Note that for the sake of simplicity we assume a stationary velocity field. At the beginning of the timestep we are given

- ◆ A triangular mesh $\mathcal{M}^{(j-1)}$ of Ω ,
- ◆ a finite element approximation $u_h^{(j-1)} \in \mathcal{S}_{1,0}^0(\mathcal{M}^{(j-1)}) \leftrightarrow$ coefficient vector $\bar{\mu}^{(j-1)} \in \mathbb{R}^{N_{j-1}}$,

① *Diffusion step*, "method-of-lines half step" for

$$\begin{aligned} \frac{\partial w}{\partial t} - \epsilon \Delta w &= 0 && \text{in } \Omega \times [t_{j-1}, t_{j-1} + \frac{1}{2}\tau], \\ w(x, t) &= 0 && \forall x \in \partial\Omega, t_{j-1} < t < t_{j-1} + \frac{1}{2}\tau, \\ w(x, t_{j-1}) &= u^{(j-1)}(x) && \forall x \in \Omega. \end{aligned} \quad (10.3.3.6)$$

We approximately solve (10.3.3.6) by finite element Galerkin discretization utilizing lowest order Lagrangian finite elements on $\mathcal{M}^{(j-1)}$ and a single step of the implicit Euler method (9.2.7.3) (timestep size $\frac{1}{2}\tau$)

$$\vec{v} = (\mathbf{M} + \frac{1}{2}\tau \epsilon \mathbf{A})^{-1} \bar{\mu}^{(j-1)}, \quad (10.3.3.17)$$

where $\mathbf{A} \in \mathbb{R}^{N_{j-1} \times N_{j-1}} \doteq \mathcal{S}_{1,0}^0(\mathcal{M})$ -Galerkin matrix for $-\Delta$, $\mathbf{M} \doteq$ (possibly lumped) $\mathcal{S}_{1,0}^0(\mathcal{M})$ -mass matrix. Then \vec{v} contains the tent function basis expansion coefficients of an approximation of $w(t_{j-1} + \frac{1}{2}\tau)$.

In general, in the context of a Strang splitting scheme it is advisable to employ 2nd-order timestepping: use a 2nd-order $L(\pi)$ -stable single step method, e.g., SDIRK-2 (9.2.7.51).

② *Advection step*: use the particle method of Section 10.3.3.2 for

$$\begin{aligned} \frac{\partial z}{\partial t} + \mathbf{v}(x, t) \cdot \mathbf{grad} z &= f(x, t) && \text{in } \Omega \times [t_{j-1}, t_j], \\ z(x, t) &= 0 && \text{on inflow boundary } \Gamma_{\text{in}}, t_{j-1} < t < t_j, \\ z(x, t_{j-1}) &= w_h(x, t_{j-1} + \frac{1}{2}\tau) && \forall x \in \Omega. \end{aligned} \quad (10.3.3.7)$$

We carry out a Lagrangian advection step (of size τ) as introduced in Section 10.3.3.2 for (10.3.3.7) with

- initial “particle positions” \mathbf{p}_i given by nodes of $\mathcal{M}^{(j-1)}$, $i = 1, \dots, N_j$,
- initial “particle temperatures” given by corresponding coefficients ν_i , the components of $\vec{\nu} \in \mathbb{R}^{N_{j-1}}$ obtained from (10.3.3.17).

③ *Remeshing*, required, because the advection step has moved nodes to new positions $\tilde{\mathbf{p}}_i$ (and, maybe, introduced new nodes by “particle injection”, deleted nodes by “particle removal”).

➤ Create **new** triangular mesh $\mathcal{M}^{(j)}$ with nodes $\tilde{\mathbf{p}}_i$ (+ boundary nodes), $i = 1, \dots, N_j$

④ Repeat diffusion step ① starting with $w_h \in \mathcal{S}_{1,0}^0(\mathcal{M}^{(j)})$ = linear interpolant (\rightarrow Def. 3.3.2.1) of “particle temperatures” on $\mathcal{M}^{(j)}$. This yields the new approximate solution $u_h^{(j)}$

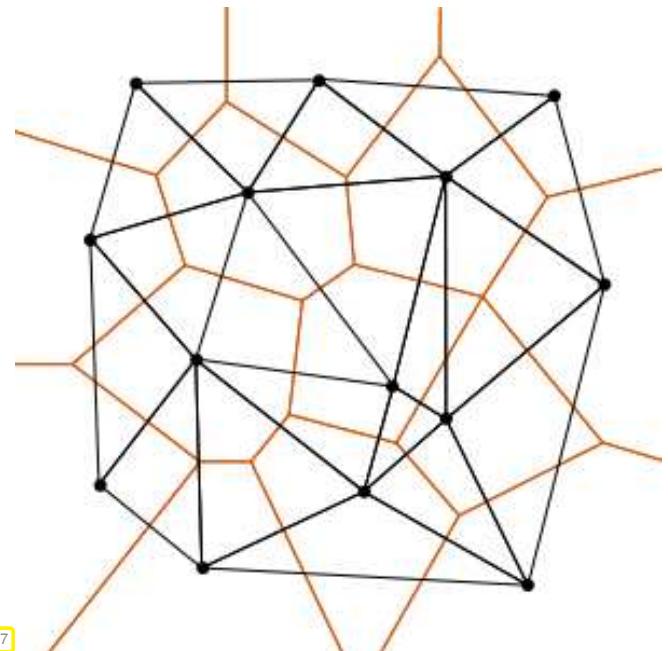
Of course, ①+③ and ② can be interleaved in leapfrog fashion, as explained in Rem. 10.3.3.9.

EXAMPLE 10.3.3.18 (Delaunay-remeshing in 2D)

Delaunay algorithm for creating a 2D triangular mesh with prescribed nodes:

- ① Compute Voronoi cells, see (4.2.2.2) & <http://www.qhull.org/>.
- ② Connect two nodes, if their associated Voronoi dual cells have an edge in common.

Remark 10.3.3.19. In MATLAB the Delaunay algorithm is available via the built-in function `TRI = delaunay(x, y)`.



EXPERIMENT 10.3.3.20 (Lagrangian method for convection-diffusion in 1D)

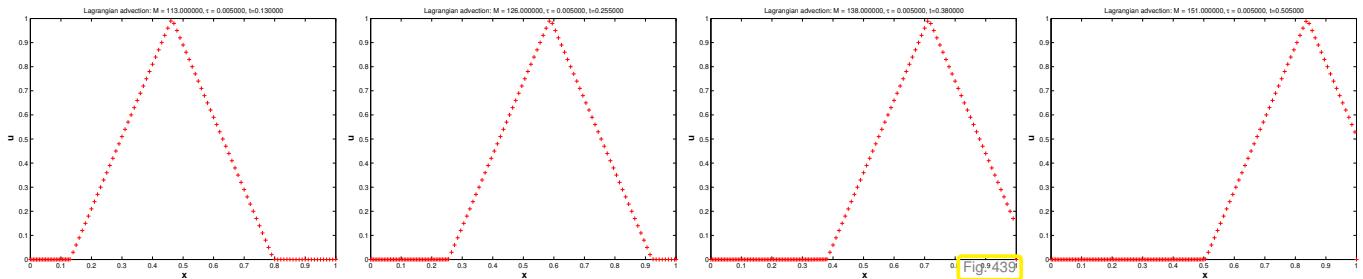
Same IBVP (10.3.1.4) as in Exp. 10.3.1.3:

$$\frac{\partial u}{\partial t} - \epsilon \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x} = 0, \\ u(x, 0) = \max(1 - 3|x - \frac{1}{3}|, 0), \\ u(0) = u(1) = 0.$$

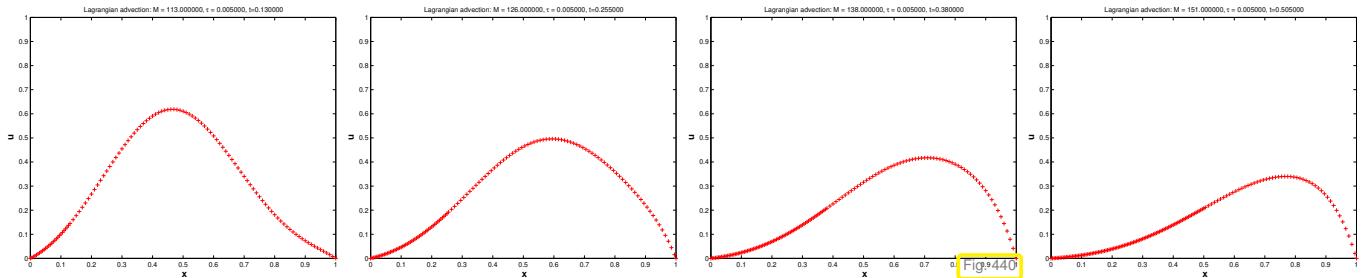
- ◆ Linear finite element Galerkin discretization with mass lumping in space
- ◆ Strang splitting applied to diffusive and convective terms
- ◆ (Sub-optimal) implicit Euler timestepping for diffusive partial timestep

$$x \mapsto u_0(x)$$

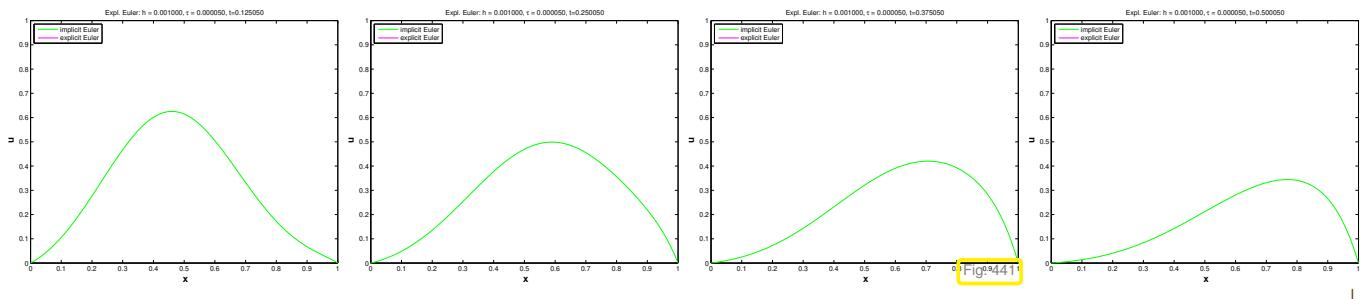
$$\epsilon = 10^{-5}:$$



$$\epsilon = 0.1:$$



“Reference solution” computed by method of lines, see Exp. 10.3.1.3, with $h = 10^{-3}$, $\tau = 5 \cdot 10^{-5}$ (“overkill resolution”):



EXPERIMENT 10.3.3.21 (Lagrangian method for convection-diffusion in 2D)

- ◆ IBVP (10.3.1.1) on $\Omega = [0, 1]^2$, $T = 1$,
- ◆ Particle mesh method based on Delaunay remeshing, see Ex. 10.3.3.18, and linear finite element Galerkin discretization for diffusion step.

We qualitatively examine the obtained solution.

Let us summarize the pros and cons of Lagrangian particle methods:



Advantage of Lagrangian (particle) methods for convection diffusion:

- ◆ No artificial diffusion required (no “smearing”)
- No stability induced timestep constraint



Drawback of Lagrangian (particle) methods for convection diffusion:

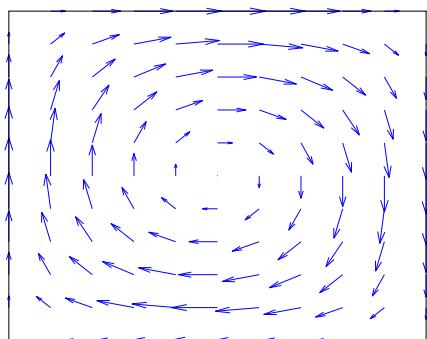
- ◆ Remeshing (may be) expensive and difficult.
- Point advection may produce “voids” in point set.

Review question(s) 10.3.3.22 (Lagrangian split-step method)

◆ **(Q10.3.3.22.A)** [Strang splitting] Explain the Strang-splitting single-step method for the IVP

$$\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}) + \mathbf{r}(\mathbf{y}) , \quad \mathbf{y}(0) = \mathbf{y}_0 .$$

(Q10.3.3.22.B) [Particle method for pure transport]



We consider the initial value problem

$$\begin{aligned} \frac{\partial u}{\partial t}(\mathbf{x}, t) + \mathbf{v}(\mathbf{x}, t) \cdot \nabla u(\mathbf{x}, t) &= 0 \quad \text{in } \tilde{\Omega}, \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega, \end{aligned}$$

with a velocity field \mathbf{v} as plotted beside.

Fig. 442

What kind of difficulty can be encountered when using the particle method for pure transport with the explicit Euler single-step method for “particle pushing”?

△

10.3.4 Semi-Lagrangian Method



Video tutorial for Section 10.3.4: Semi-Lagrangian Method: (35 minutes) [Download link](#), [tablet notes](#)

Now we study a family of methods for transient convection-diffusion that takes into account transport along streamlines, but, in contrast to genuine Lagrangian methods, relies on a *fixed* mesh.

To begin, we recall the **flow map** Φ^t induced by a (stationary) velocity field $\mathbf{v} \in C^0(\bar{\Omega})$, see (10.1.1.6). Since we admit time-dependent velocity fields, now the flow map takes two time arguments and agrees with the **evolution operator** (\rightarrow § 9.2.5.8) for the flow equation $\dot{\mathbf{y}}(t) = \mathbf{v}(\mathbf{y}(t), t)$. Therefore, we reuse the notation $\Phi^{t_0, t}$ for the flow map.

§10.3.4.1 (The material derivative) Now we introduce the Lagrangian concept of the “rate of change of a function seen by an observer moving with the flow”:

Definition 10.3.4.2. Material derivative

Given a velocity field $\mathbf{v} : \Omega \times]0, T[\mapsto \mathbb{R}^d$, the **material derivative** of a function $f = f(\mathbf{x}, t) : \Omega \times]0, T[\mapsto \mathbb{R}$ at $(\mathbf{x}, t_0) \in \Omega \times]0, T[$ is ($0 < t_0 < T$)

$$\frac{Df}{D\mathbf{v}}(\mathbf{x}, t_0) = \lim_{\tau \rightarrow 0} \frac{f(\mathbf{x}, t_0) - f(\Phi^{t_0, -\tau} \mathbf{x}, t_0 - \tau)}{\tau} = \frac{d}{d\tau} \left\{ \tau \mapsto f(\Phi^\tau \mathbf{x}, \tau) \right\} \Big|_{\tau=t_0}, \quad \mathbf{x} \in \Omega,$$

with $\Phi^{t_0, t}$ the flow map (at time t_0) associated with \mathbf{v} , that is, cf. (10.1.1.6), (10.1.1.7),

$$\frac{d\{t \mapsto \Phi^{t_0, t} \mathbf{x}\}}{dt}(t) = \mathbf{v}(\Phi^{t_0, t} \mathbf{x}, t) \quad , \quad \Phi^{t_0, t_0} \mathbf{x} = \mathbf{x}. \quad (10.3.4.3)$$

The material derivative $\frac{Df}{D\mathbf{v}}$ is the

rate of change of f experienced by a particle carried along by the flow

because $t \mapsto \Phi^{t_0, t} \mathbf{x}$ describes the trajectory of a particle located at \mathbf{x} at time t_0 ($\leftrightarrow t = 0$). \square

§10.3.4.4 (Convection-diffusion equation via material derivative) By a straightforward application of the chain rule for smooth f we find

$$\frac{Df}{D\mathbf{v}}(\mathbf{x}, t) = \mathbf{grad}_{\mathbf{x}} f(\mathbf{x}, t) \cdot \mathbf{v}(\mathbf{x}, t) + \frac{\partial f}{\partial t}(\mathbf{x}, t). \quad (10.3.4.5)$$

Thus, the transient convection-diffusion equation (10.3.0.1) can be rewritten by means of the material derivative:

$$\begin{aligned} \frac{\partial u}{\partial t} - \epsilon \Delta u + \mathbf{v}(\mathbf{x}, t) \cdot \mathbf{grad} u &= f \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \\ \Updownarrow &\leftarrow (10.3.4.5) \end{aligned}$$

$$\frac{Du}{D\mathbf{v}} - \epsilon \Delta u = f \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[. \quad (10.3.4.6)$$

§10.3.4.7 (Semi-Lagrangian temporal semi-discretization) The formulation (10.3.4.6) serves as starting point.

Idea: Employ a *backward difference* (“implicit Euler”) discretization of the material derivative



$$\frac{Du}{D\mathbf{v}}|_{(\mathbf{x},t)=(\bar{\mathbf{x}},t_0)} \approx \frac{u(\bar{\mathbf{x}}, t_0) - u(\Phi^{t_0, t_0 - \tau} \bar{\mathbf{x}}, t_0 - \tau)}{\tau}, \quad (10.3.4.8)$$

with timestep $\tau > 0$, where the trajectory $t \mapsto \Phi^{t_0, t} \bar{\mathbf{x}}$ solves the initial value problem

$$\frac{d\{t \mapsto \Phi^{t_0, t} \bar{\mathbf{x}}\}}{dt}(t) = \mathbf{v}(\Phi^{t_0, t} \bar{\mathbf{x}}, t), \quad \Phi^{t_0, t_0} \bar{\mathbf{x}} = \bar{\mathbf{x}}. \quad (10.3.4.3)$$

Obviously, in (10.3.4.8) Φ is the flow map associated with \mathbf{v} .

► This idea yields a *semi-discretization* of (10.3.4.6) *in time* (with fixed timestep $\tau > 0$)

$$\begin{aligned} \frac{u^{(j)}(\mathbf{x}) - u^{(j-1)}(\Phi^{t_j, t_j - \tau} \mathbf{x})}{\tau} - \epsilon \Delta u^{(j)}(\mathbf{x}) &= f(\mathbf{x}, t_j) \quad \text{in } \Omega, \\ + \quad \text{initial conditions at } t = t_j, \quad \text{boundary conditions on } \partial\Omega, \end{aligned} \quad (10.3.4.9)$$

where $u^{(j)} : \Omega \mapsto \mathbb{R}$ is an approximation for $u(\cdot, t_j)$, $t_j := j\tau$, $j \in \mathbb{N}$. Note the difference to the method of lines (→ Section 9.2.4, Section 9.3.3, and Section 10.3.1): in (10.3.4.9) semidiscretization in time was carried out first, now followed by discretization in space, which reverses the order adopted in the method of lines. □

§10.3.4.10 (Finite element spatial discretization) Cast (10.3.4.9) into variational form according to the recipe of Section 1.8

$$\begin{aligned} u^{(j)} \in H_0^1(\Omega): \quad & \int_{\Omega} (u^{(j)}(\mathbf{x}) - u^{(j-1)}(\Phi^{t_j, t_j - \tau} \mathbf{x})) v(\mathbf{x}) + \tau \epsilon \operatorname{grad} u^{(j)} \cdot \operatorname{grad} v \, d\mathbf{x} \\ &= \tau \int_{\Omega} f(\mathbf{x}, t_j) v(\mathbf{x}) \, d\mathbf{x} \quad \forall v \in H_0^1(\Omega), \end{aligned} \quad (10.3.4.11)$$

and apply *Galerkin discretization* (here discussed for piecewise linear finite elements, homogeneous Dirichlet boundary conditions $u = 0$ on $\partial\Omega$).

This yields one timestep (size τ) for the *semi-Lagrangian method*: the approximation $u_h^{(j)}$ for $u(j\tau)$ (equidistant timesteps) is computed from the previous timestep according to

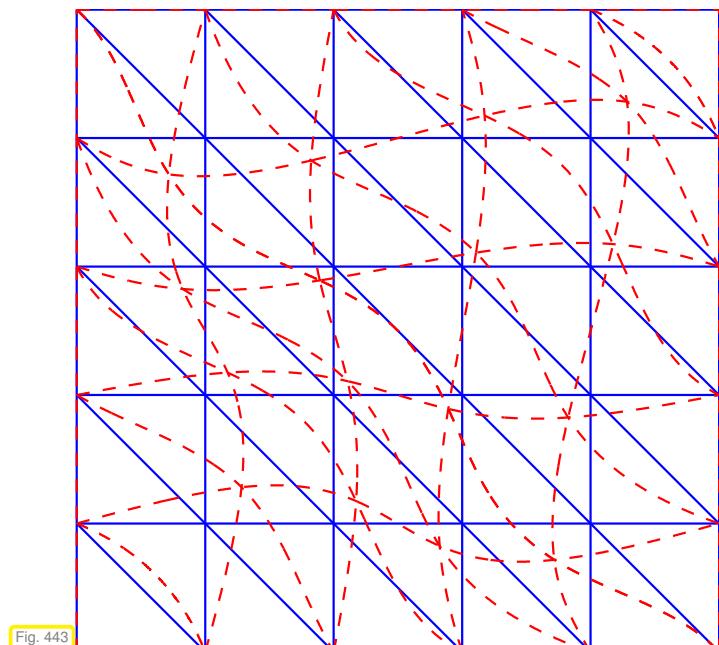
$$\begin{aligned} u_h^{(j)} \in \mathcal{S}_{1,0}^0(\mathcal{M}): \quad & \int_{\Omega} \frac{u_h^{(j)}(\mathbf{x}) - u_h^{(j-1)}(\Phi^{t_j, t_j - \tau} \mathbf{x})}{\tau} v_h(\mathbf{x}) \, d\mathbf{x} + \epsilon \int_{\Omega} \operatorname{grad} u_h^{(j)} \cdot \operatorname{grad} v_h \, d\mathbf{x} \\ &= \int_{\Omega} f(\mathbf{x}, t_j) v_h(\mathbf{x}) \, d\mathbf{x} \quad \forall v_h \in \mathcal{S}_{1,0}^0(\mathcal{M}). \end{aligned} \quad (10.3.4.12)$$

Here, \mathcal{M} is supposed to be a *fixed* triangular mesh of Ω . □

§10.3.4.13 (Interpolatory fully discrete semi-Lagrangian method) However, (10.3.4.12) cannot be implemented, because the function

$$\mathbf{x} \mapsto u_h^{(j-1)}(\Phi^{t_j, t_j - \tau} \mathbf{x}) \in H^1(\Omega)$$

is a former finite element function that has been “transported with the (reversed) flow” (in the sense of pullback, see Def. 2.8.1.2)



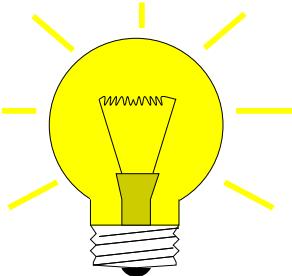
$\triangleleft \quad \text{---} \hat{=} \text{ image of } \mathcal{M} (\text{---}) \text{ under the mapping } \Phi^{t_j, t_j - \tau}$

The pullback $\mathbf{x} \mapsto v_h(\Phi^{t_j, t_j - \tau} \mathbf{x})$ of $v_h \in \mathcal{S}_{1,0}^0(\mathcal{M})$ is piecewise smooth w.r.t. the mapped mesh drawn with --- . Hence, it is not smooth inside the cells of \mathcal{M} .

► In general the transported function $\mathbf{x} \mapsto v_h(\Phi^{t_j, t_j - \tau} \mathbf{x})$ will **not** be a finite element function on \mathcal{M} ,

As a consequence, the direct “analytic” computation of entries of Galerkin matrices arising from (10.3.4.12) is not possible.

Idea:



◆ replace the transported finite element function $\mathbf{x} \mapsto u_h^{(j-1)}(\Phi^{t_j, t_j - \tau} \mathbf{x})$ with its linear interpolant (\rightarrow Def. 3.3.2.1)

$$l_1(u_h^{(j-1)} \circ \Phi^{t_j, t_j - \tau}) \in \mathcal{S}_{1,0}^0(\mathcal{M}),$$

◆ approximate $\Phi^{t_j, t_j - \tau} \mathbf{x}$ by $\mathbf{x} - \tau \mathbf{v}(\mathbf{x}, t_j)$, which amounts to the use of a single explicit Euler step.

This second approximation is called **streamline backtracking**, because we follow the streamline through (\mathbf{x}, t_j) backward in time. This yields the following definition of a single timestep $t_{j-1} \rightarrow t_j$ of the **semi-Lagrangian scheme** for the convection-diffusion IBVP (10.3.1.1):

$$\begin{aligned} u_h^{(j)} \in \mathcal{S}_{1,0}^0(\mathcal{M}): \quad & \int_{\Omega} \frac{u_h^{(j)}(\mathbf{x}) - l_1(u_h^{(j-1)}(\cdot - \tau \mathbf{v}(\cdot, t_j)))(\mathbf{x})}{\tau} w_h(\mathbf{x}) \, d\mathbf{x} + \epsilon \int_{\Omega} \mathbf{grad} u_h^{(j)} \cdot \mathbf{grad} w_h \, d\mathbf{x} \\ & = \int_{\Omega} f(\mathbf{x}, t_j) w_h(\mathbf{x}) \, d\mathbf{x} \quad \forall w_h \in \mathcal{S}_{1,0}^0(\mathcal{M}). \end{aligned}$$

Then apply local vertex based numerical quadrature (2D trapezoidal rule (2.4.6.10) = global trapezoidal rule) to the first integral. This amounts to using **mass lumping**, see Rem. 9.3.4.18.

► Implementable version of (10.3.4.12) (on a 2D triangular mesh \mathcal{M})

$$\begin{aligned} u_h^{(j)} \in \mathcal{S}_{1,0}^0(\mathcal{M}): \quad & \frac{1}{3} |U_p| (\mu_p^{(j)} - u_h^{(j-1)}(\mathbf{p} - \tau \mathbf{v}(\mathbf{p}, t_j))) + \tau \int_{\Omega} \mathbf{grad} u_h^{(j)} \cdot \mathbf{grad} b_h^p \, d\mathbf{x} \\ & = \frac{1}{3} \tau |U_p| f(\mathbf{p}), \quad \mathbf{p} \in \mathcal{N}(\mathcal{M}) \cap \Omega, \quad (10.3.4.14) \end{aligned}$$

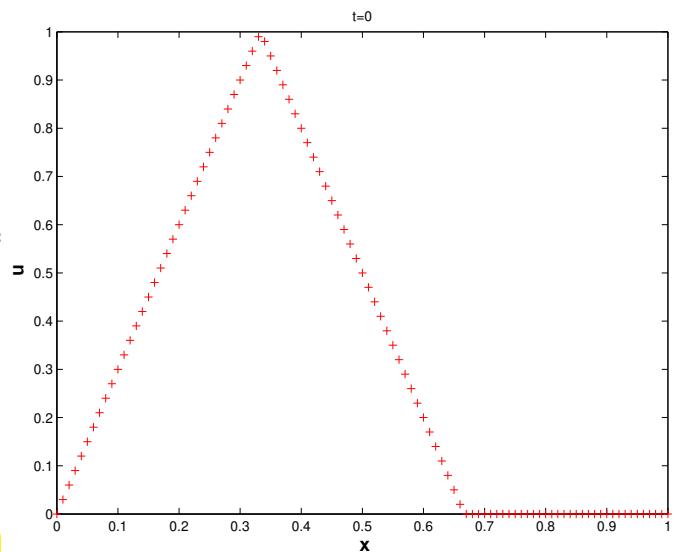
where $\mu_p^{(j)}$ are the nodal values of $u_h^{(j)} \in \mathcal{S}_{1,0}^0(\mathcal{M})$ associated with the interior nodes of the mesh \mathcal{M} , b_h^p is the “tent function” belonging to node \mathbf{p} , $|U_p|$ is the sum of the areas of all triangles adjacent to \mathbf{p} .

EXPERIMENT 10.3.4.15 (Semi-Lagrangian method for convection-diffusion in 1D) We perform a qualitative study of the semi-Lagrangian scheme in one spatial dimension:

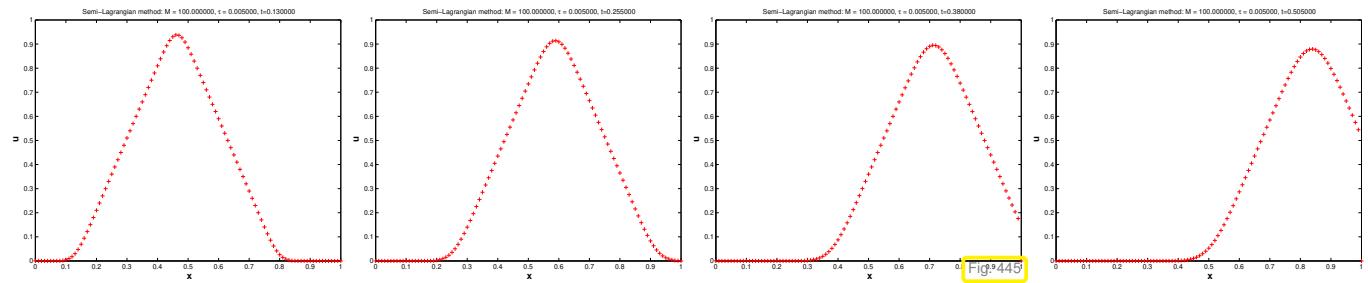
Same IBVP as in Ex. 10.3.3.20

- ◆ Linear finite element Galerkin discretization with mass lumping in space
- ◆ Semi-Lagrangian method: 1D version of (10.3.4.12)
- ◆ Explicit Euler streamline backtracking

We give snapshots of the solution for times $t = 0.124 * j, j = 1, 2, 3, 4$.

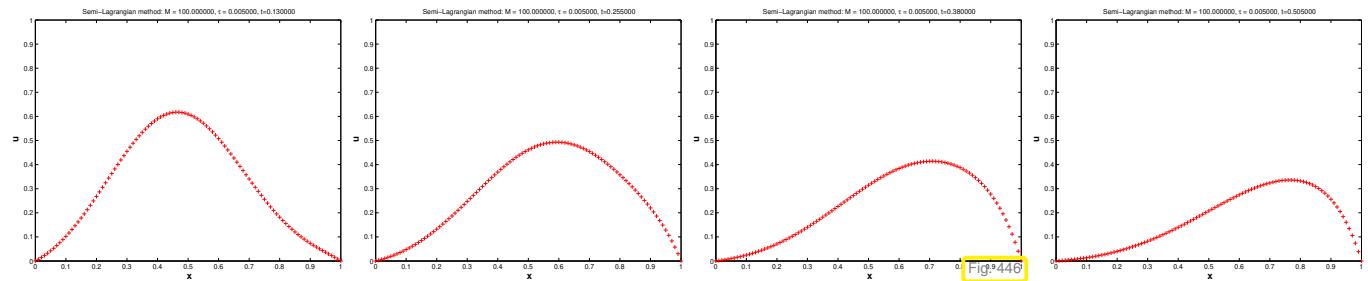


$\epsilon = 10^{-5}, \tau = 0.01, h = 0.01$:

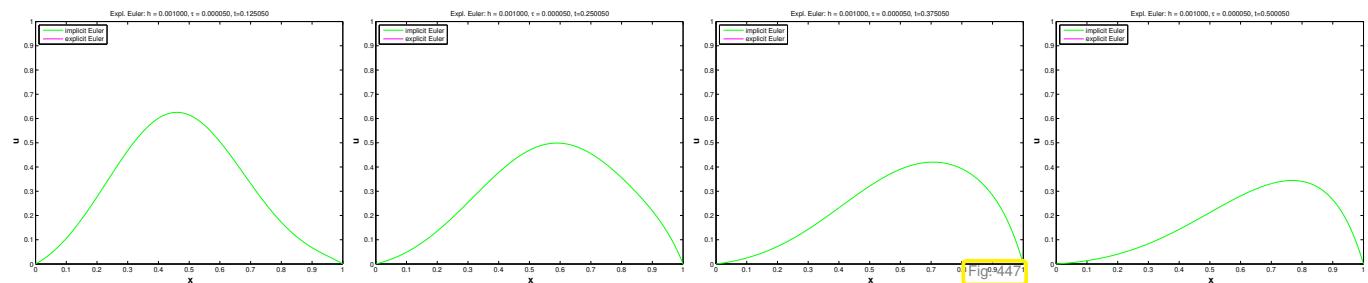


We observe rapid smearing of the sharp kink in the initial data: the semi-Lagrangian method is rather diffusive due to repeated interpolation.

$\epsilon = 0.1, \tau = 0.01, h = 0.01$:



“Reference solution” computed by method of lines, see Exp. 10.3.1.3, with “overkill resolution” $h = 10^{-3}$, $\tau = 5 \cdot 10^{-5}$:



EXPERIMENT 10.3.4.16 (Semi-Lagrangian method for convection-diffusion in 2D) We qualitatively study the solutions produced by the semi-Lagrangian scheme in 2D.

- ◆ 2nd-order scalar convection diffusion problem (10.3.1.1), $\Omega := [0, 1]^2$, $f = 0$, $g = 0$,
- ◆ velocity field

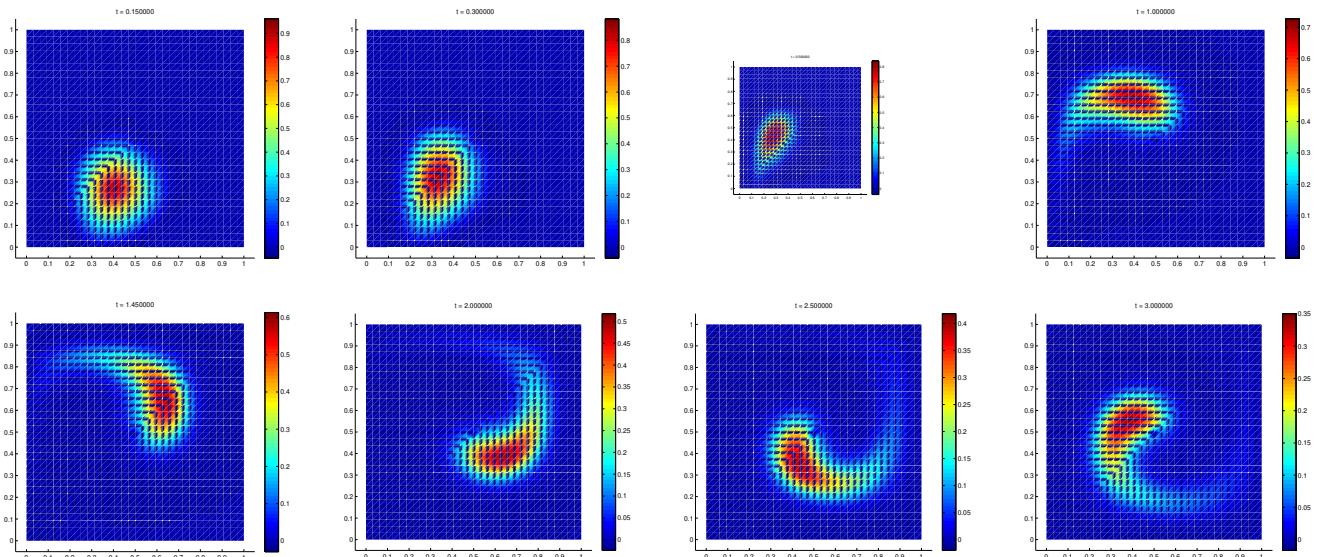
$$\mathbf{v}(\mathbf{x}) := \begin{bmatrix} -\sin(\pi x_1) \cos(\pi x_2) \\ \sin(\pi x_2) \cos(\pi x_1) \end{bmatrix}.$$

- ◆ Initial condition: “compactly supported cone shape”

$$u_0(\mathbf{x}) := \max\{0, 1 - 4 \left\| \mathbf{x} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \right\| \}.$$

- ◆ semi-Lagrangian finite element Galerkin discretization according to (10.3.4.12) on regular triangular meshes of square domain Ω , see Fig. 257.

Example with $\epsilon = 0$:



We observe smearing of initial data due to numerical diffusion inherent in the interpolation step of the semi-Lagrangian method. □

Review question(s) 10.3.4.17 (Semi-Lagrangian methods)

(Q10.3.4.17.A) We consider the pure transport IBVP on $[0, 1] \times [0, 1]$

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 \quad , \quad u(0, t) = 0 \quad , \quad u(x, 0) = \max\{(1 - 3|x - \frac{1}{3}|), 0\}.$$

Under what sufficient conditions will an interpolation-based semi-Lagrangian method that

- relies on uniform timesteps of size $\tau > 0$ and an equidistant mesh \mathcal{M} with meshwidth $h > 0$,
- uses $\mathcal{S}_1^0(\mathcal{M})$ as trial and test space,
- employs an explicit Euler approximation of the flow map,

produce snapshots of the exact solution.

(Q10.3.4.17.B) For a given smooth velocity field $\mathbf{v} \in (\mathcal{C}^1(\overline{\Omega}))^2$ and a continuous source function $f \in C^0(\overline{\Omega})$ on a computational domain $\Omega \subset \mathbb{R}^2$ we want to solve the convection-diffusion Dirichlet IBVP

$$\begin{aligned} -\epsilon \Delta u + \mathbf{v}(x) \cdot \mathbf{grad} u &= f(x, t) \quad \text{in } \Omega \times]0, T[\\ u(x, t) &= 0 \quad \text{on } \partial\Omega \times]0, T[, \quad u(x, 0) = u_0(x) . \end{aligned}$$

We use the semi-Lagrangian method with uniform timestep $\tau > 0$:

$$\begin{aligned} u_h^{(j)} \in \mathcal{S}_{1,0}^0(\mathcal{M}): \quad & \int_{\Omega} \frac{u_h^{(j)}(x) - l_1(u_h^{(j-1)}(\cdot - \tau \mathbf{v}(\cdot)))(x)}{\tau} w_h(x) dx + \epsilon \int_{\Omega} \mathbf{grad} u_h^{(j)} \cdot \mathbf{grad} w_h dx \\ &= \int_{\Omega} f(x, t_j) w_h(x) dx \quad \forall w_h \in \mathcal{S}_{1,0}^0(\mathcal{M}) . \end{aligned}$$

Selecting a basis $\mathfrak{B} := \{b_h^1, \dots, b_h^N\}$ of $\mathcal{S}_{1,0}^0(\mathcal{M})$, $N := \dim \mathcal{S}_{1,0}^0(\mathcal{M})$, this leads to the linear recursion

$$\mathbf{M} \vec{\mu}^{(j)} + \mathbf{B} \vec{\mu}^{(j-1)} + \epsilon \tau \mathbf{A} \vec{\mu}^{(j)} = \tau \vec{\varphi}^{(j)}$$

for the basis expansion coefficient vectors $\vec{\mu}^{(j)}$ of $u_h^{(j)}$, $j = 0, \dots, M$.

Give concrete formulas for the entries of the matrices $\mathbf{M}, \mathbf{B}, \mathbf{A} \in \mathbb{R}^{N,N}$.

△

Learning outcomes

After having digested the contents of this chapter you should

- know the mathematical model (“convection-diffusion equation”) for stationary and transient heat conduction in a moving (incompressible) fluid,
- understand the notion of *singular perturbation* and when convection-diffusion boundary value problems are singularly perturbed.
- know that standard Galerkin finite element discretization of convection-diffusion boundary value problem runs risk of spurious oscillations of the numerical solution in the case of singular perturbation.
- be familiar with the idea of *upwind quadrature* for a stable discretization of singularly perturbed convection-diffusion problems.
- know stabilization through *artificial diffusion/viscosity* and how it is used in the streamline diffusion method.
- remember that the method of lines approach for singularly perturbed transient convection-diffusion problems requires a stable discretization in space.
- comprehend the main idea of Lagrangian particle methods for transient advection.
- be familiar with the principle of semi-Lagrangian finite element methods for transient advection-diffusion boundary value problems.

Bibliography

- [Bra07] Dietrich Braess. *Finite elements. Theory, fast solvers, and applications in elasticity theory*. Third. Cambridge: Cambridge University Press, 2007, pp. xviii+365. DOI: [10.1017/CBO9780511618635](https://doi.org/10.1017/CBO9780511618635) (cit. on p. 628).
- [Bur+10] A. Burtscher, E. Fonn, P. Meury, and C. Wiesmayr. *LehrFEM - A 2D Finite Element Toolbox*. SAM, ETH Zürich. Zürich, Switzerland, 2010 (cit. on p. 616).
- [Eva98] L.C. Evans. *Partial differential equations*. Vol. 19. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 1998 (cit. on p. 624).
- [GG20] Deepika Garg and Sashikumaar Ganesan. *Generalized local projection stabilized finite element method for advection-reaction problems*. 2020 (cit. on p. 632).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 618, 659).
- [PW01] T. N. Phillips and A. J. Williams. “Conservative semi-Lagrangian finite volume schemes”. In: *Numer. Methods Partial Differential Equations* 17.4 (2001), pp. 403–425. DOI: [10.1002/num.1019](https://doi.org/10.1002/num.1019) (cit. on p. 664).
- [RST08] H.-G. Roos, M. Stynes, and L. Tobiska. *Numerical methods for singularly perturbed differential equations. Convection-diffusion-reaction and flow problems*. 2nd. Vol. 24. Springer Series in Computational Mathematics. Berlin: Springer, 2008 (cit. on p. 616).
- [Str09] M. Struwe. *Analysis für Informatiker*. Lecture notes, ETH Zürich. 2009 (cit. on p. 622).
- [XZ99] J. Xu and L. Zikatanov. “A monotone finite element scheme for convection diffusion equations”. In: *Math. Comp.* 68.228 (May 1999), pp. 1429–1446 (cit. on p. 648).

Chapter 11

Numerical Methods for Conservation Laws

Conservation laws describe physical phenomena governed by

- ◆ *conservation* laws for certain physical quantities (e.g., mass momentum, energy, etc.),
- ◆ *transport* of conserved physical quantities.

We have already examined problems of this type in connection with transient heat conduction in Section 10.1.4. There thermal energy was the conserved quantity and a *prescribed* external velocity field \mathbf{v} determined the transport. Familiarity with Chapter 10 is advantageous, but not essential for understanding this chapter.

A new aspect emerging for general conservation laws is that the transport velocity itself may depend on the conserved quantities themselves, which gives rise to *non-linear models*.

Contents

11.1 Conservation Laws: Examples	673
11.1.1 Linear Advection	673
11.1.2 Traffic Modeling [BD11]	678
11.1.3 Inviscid Gas Flow	683
11.2 Scalar Conservation Laws in 1D	686
11.2.1 Integral and Differential Form	686
11.2.2 Characteristics	689
11.2.3 Weak Solutions	695
11.2.4 Jump Conditions	697
11.2.5 Riemann Problem	698
11.2.6 Entropy Condition	706
11.2.7 Properties of Entropy Solutions	709
11.3 Conservative Finite Volume (FV) Discretization	712
11.3.1 Prologue: Finite-Difference Method (FDM)	712
11.3.2 Spatially Semi-Discrete Conservation Form	717
11.3.3 Discrete Conservation Property	720
11.3.4 Numerical Flux Functions	723
11.3.5 Monotone Schemes	742
11.4 Timestepping for Finite-Volume Methods	747
11.4.1 Fully Discrete Evolutions	747
11.4.2 CFL-Condition	750
11.4.3 Linear Stability Analysis	755
11.4.4 Convergence of Fully Discrete FV Method	762
11.5 Higher-Order Conservative Finite-Volume Schemes	767
11.5.1 Piecewise Linear Reconstruction	768

11.5.2 Slope Limiting	776
11.5.3 MUSCL Scheme	781
11.6 Outlook: Systems of Conservation Laws	784

11.1 Conservation Laws: Examples

In this section we will study a few important examples of linear and non-linear conservation laws. In the process we will discover key properties of their solutions. The focus here and throughout this chapter is on **Cauchy problems**, evolution problems posed on the *unbounded* spatial domain $\Omega = \mathbb{R}^d$.

- Cauchy problems are pure initial value problems (no boundary values).

Why do we restrict ourselves to Cauchy problems? Mainly for two reasons:

- ① *Finite speed of propagation* typical of conservation laws → Thm. 11.2.7.3

(This means that potential spatial boundaries will not affect the solution for some time in the case of compactly supported initial data, cf. situation for wave equation, where we also examined the Cauchy problem, see (9.3.2.2).)

- ② No spatial boundary > need not worry about (spatial) boundary conditions!

(The issues arising for spatial boundary conditions can be very intricate for conservation laws, cf. Rem. 11.2.1.8. We would like to avoid these complications.)

11.1.1 Linear Advection

The phenomena modeled by conservation laws are usually **transport-dominated**. The simplest case are models where the transport is due to a *given* velocity field $\mathbf{v} : \Omega \times [0, T] \rightarrow \mathbb{R}^d$, cf. Section 10.1.1, which leads to *linear* advection/convection evolution problems as discussed in detail in Section 10.3.

§11.1.1.1 (Heat transport in a moving fluid) Let the movement of a fluid occupying the spatial domain $\Omega \subset \mathbb{R}^d$ be described by a time-dependent continuous velocity field $\mathbf{v} : \Omega \times [0, T] \rightarrow \mathbb{R}^d$, $T > 0 \hat{=} \text{final time}$. If $u = u(\mathbf{x}, t)$, $\mathbf{x} \in \Omega$, $0 \leq t \leq T$, denotes the temperature of the fluid, then the total heat flux is

$$\mathbf{j}(\mathbf{x}, t) = -\kappa \mathbf{grad} u(\mathbf{x}, t) + \mathbf{v}(\mathbf{x}, t) \rho u(\mathbf{x}, t), \quad (\mathbf{x}, t) \in \Omega \times [0, T], \quad (11.1.1.2)$$

↑
diffusive heat flux

(due to spatial variation of temperature)

↑
convective heat flux

(due to fluid flow)

In (11.1.1.2), $\rho > 0$ stands for the heat capacity of the fluid and $\kappa > 0$ for its heat conductivity, see Section 9.2.1. The convective heat flux reflects the simple fact the moving hot material amounts to transporting thermal energy. The conservation law

$$\frac{\partial}{\partial t}(\rho u)(\mathbf{x}, t) + (\operatorname{div}_{\mathbf{x}} \mathbf{j})(\mathbf{x}, t) = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega}. \quad (9.2.1.5)$$

for thermal energy in arbitrary is not affected by the movement of the fluid. As in § 9.2.1.2 we can simply plug $\mathbf{j}(\mathbf{x}, t)$ from (11.1.1.2) into (9.2.1.5) and end up with the **convection-diffusion** equation for the temperature distribution in a fluid flowing with velocity $\mathbf{v} = \mathbf{v}(\mathbf{x}, t)$

$$\frac{\partial}{\partial t}(\rho u) - \operatorname{div}(\kappa \mathbf{grad} u) + \operatorname{div}(\rho \mathbf{v} u) = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega} := \Omega \times [0, T], \quad (10.1.4.1)$$

where $f = f(\mathbf{x}, t)$ models a heat source.

Next we assume that the fluid moves fast and conducts heat only poorly, which makes it possible to neglect the diffusive heat flux, formally achieved by setting $\kappa := 0$, which results in the linear **advection equation** → Section 10.1.4

$$\frac{\partial}{\partial t}(\rho u) + \operatorname{div}(\rho \mathbf{v}(\mathbf{x}, t)u) = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[.$$

Switching to $\Omega = \mathbb{R}^d$ this leads us to consider the the following **Cauchy problem** for the linear advection equation

$$\frac{\partial}{\partial t}(\rho u) + \operatorname{div}(\mathbf{v}(\mathbf{x}, t)(\rho u)) = f(\mathbf{x}, t) \quad \text{in } \tilde{\Omega} := \mathbb{R}^d \times]0, T[, \quad (11.1.1.3)$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathbb{R}^d \quad (\text{initial conditions}) . \quad (11.1.1.4)$$

with $u = u(\mathbf{x}, t) \hat{=} \text{temperature}$, $\rho > 0 \hat{=} \text{heat capacity}$, $\mathbf{v} = \mathbf{v}(\mathbf{x}, t) \hat{=} \text{prescribed locally Lipschitz-continuous velocity field}$, $\mathbf{v} : \mathbb{R}^d \times [0, T] \rightarrow \mathbb{R}^d$.

Terminology: (11.1.1.3) is an instance of a **linear scalar conservation law**.

Why “conservation law”? Because the derivation of the mathematical model invoked the conservation of the thermal energy (density) ρu as a key principle. We elaborate this further below.

Why “linear”? Because the flux \mathbf{j} from (11.1.1.2) depends linearly on u . In particular, the convective flux depends linearly on the value of u in a single space-time point.

In the sequel we study a much simplified problem and assume constant heat capacity, $\rho \equiv 1$ after scaling, the absence of heat sources, $f \equiv 0$, and a *stationary* velocity field $\mathbf{v} = \mathbf{v}(\mathbf{x})$. Thus we end up with the following rescaled initial value problem, here *written in conserved variables*:

$$\frac{\partial u}{\partial t} + \operatorname{div}(\mathbf{v}(\mathbf{x})u) = 0 \quad \text{in } \tilde{\Omega} := \mathbb{R}^d \times]0, T[,$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathbb{R}^d \quad (\text{initial conditions}) .$$

(11.1.1.5)

Convention: differential operator **div** acts on spatial independent variable only,

$$(\operatorname{div} \mathbf{f})(\mathbf{x}, t) := \frac{\partial f_1}{\partial x_1} + \cdots + \frac{\partial f_d}{\partial x_d} , \quad \mathbf{f}(\mathbf{x}, t) = \begin{bmatrix} f_1(\mathbf{x}, t) \\ \vdots \\ f_d(\mathbf{x}, t) \end{bmatrix} .$$

A general solution formula exists for (11.1.1.5), based on the notion of the **flow map** induced by the velocity field $\mathbf{v} = \mathbf{v}(\mathbf{x}, t)$, see also (10.1.1.6). The flow map $\Phi = \Phi(\mathbf{x}, t)$, $\mathbf{x} \in \mathbb{R}^d$, $t \in \mathbb{R}$ is a mapping

$$\Phi : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d \quad \text{defined by} \quad \begin{aligned} \frac{\partial \Phi}{\partial t}(\mathbf{x}, t) &= \mathbf{v}(\Phi(\mathbf{x}, t), t) \quad \text{in } \mathbb{R}^d \times \mathbb{R} , \\ \Phi(\mathbf{x}, 0) &= \mathbf{x} \quad \text{for all } \mathbf{x} \in \mathbb{R}^d . \end{aligned} \quad (11.1.1.6)$$

See Fig. 405, Fig. 406, and Fig. 407 for visualizations. By existence and uniqueness theorems for initial value problems for ordinary differential equations [Hip19, ??] the flow map Φ is well defined by (11.1.1.6), if \mathbf{v} is (locally) Lipschitz continuous, which we take for granted. The flow map satisfies

$$\Phi(\Phi(\mathbf{x}, t), -t) = \mathbf{x} \quad \text{for all } \mathbf{x} \in \mathbb{R}^d . \quad (11.1.1.7)$$

Theorem 11.1.1.8. Solution of linear advection problem

The solution of (11.1.1.5) is given by

$$u(\mathbf{x}, t) = |\det(D_{\mathbf{x}}\Phi)(\hat{\mathbf{x}}, t)|^{-1} u_0(\hat{\mathbf{x}}), \quad (\hat{\mathbf{x}}, t) \in \mathbb{R}^d \times \mathbb{R}, \quad \mathbf{x} = \Phi(\hat{\mathbf{x}}, t), \quad (11.1.1.9)$$

where $D_{\mathbf{x}}\Phi : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^{d,d}$ is the Jacobian of the flow map w.r.t. \mathbf{x} .

EXAMPLE 11.1.1.10 (Constant advection in 1D) We examine the simplest incarnation of the Cauchy problem (11.1.1.5), namely the special case of advection in a single spatial direction, $d = 1$ and $\Omega = \mathbb{R}$, and with a constant velocity $v = \text{const}$. This leads to

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(vu) = 0 \quad \text{in } \tilde{\Omega} = \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) \quad \forall x \in \mathbb{R}. \quad (11.1.1.11)$$

This is the 1D version of the transport equation (10.3.2.1) and its solution is given by formula (10.3.2.5), which is a special case of the result stated in Thm. 11.1.1.8.

(10.3.2.5)  $u(x, t) = u_0(x - vt), \quad x \in \mathbb{R}, \quad 0 \leq t < T. \quad (11.1.1.12)$

The solution $u = u(x, t)$ boils down to the initial data u_0 “travelling” with velocity v . For differentiable u_0 the solution property of $u(x, t)$ from (11.1.1.12) can be verified by direct computation. \square

Remark 11.1.1.13 (Discontinuous solutions of advection equations) To verify that (11.1.1.12) solves (11.1.1.11) in the sense of classical calculus we need $u_0 \in C^1(\mathbb{R})$. However, (11.1.1.11) remains meaningful even without this smoothness assumption.

The solution formula from Thm. 11.1.1.8 makes perfect sense even for *discontinuous* initial data u_0 !

→ We should not expect $u = u(x, t)$ to be differentiable in space or time.
A “weaker” concept of solution is required, see Section 11.2.3 below.

This consideration should be familiar: for second order elliptic boundary value problems, for which classical solutions are to be twice continuously differentiable, the concept of a variational solution made it possible to give a meaning to solutions $\in H^1(\Omega)$ that are merely continuous and piecewise differentiable, see Rem. 1.5.3.9.

Related to (11.1.1.12) is the d’Alembert solution formula (9.3.2.4) for 1D wave equation (9.3.2.2), which can also accommodate discontinuous solutions. \square

Proof. (of Thm. 11.1.1.8) We assume that the velocity field $\mathbf{v} : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ is continuously differentiable. Then, simply swapping differentiation in space and time, we see that the Jacobian of the flow map $D_{\mathbf{x}}\Phi$ solves the following initial value problem for the so-called **variational equation**, a differential equation for $d \times d$ -matrices:

$$\begin{cases} \frac{\partial}{\partial t} D_{\mathbf{x}}\Phi(\hat{\mathbf{x}}, t) = D_{\mathbf{x}}\mathbf{v}(\mathbf{x}, t) D_{\mathbf{x}}\Phi(\hat{\mathbf{x}}, t), & \hat{\mathbf{x}} \in \mathbb{R}^d, t \in \mathbb{R}, \quad \mathbf{x} = \Phi(\hat{\mathbf{x}}, t). \\ D_{\mathbf{x}}\Phi(\hat{\mathbf{x}}, 0) = \mathbf{I}, \end{cases} \quad (11.1.1.14)$$

The initial condition is due to the fact that $\Phi(\hat{\mathbf{x}}, 0) = \hat{\mathbf{x}}$ for all $\hat{\mathbf{x}} \in \mathbb{R}^d$.

Next, we recall the derivative of a determinant:

Theorem 10.1.3.7. Differentiation formula for determinants

Let $\mathbf{S} : I \subset \mathbb{R} \mapsto \mathbb{R}^{n,n}$ be a smooth matrix-valued function. If $\mathbf{S}(t_0)$ is regular for some $t_0 \in I$, then

$$\frac{d}{dt}(\det \circ \mathbf{S})(t_0) = \det(\mathbf{S}(t_0)) \operatorname{tr}\left(\frac{d\mathbf{S}}{dt}(t_0)\mathbf{S}^{-1}(t_0)\right),$$

where tr stands for the trace of a matrix.

We apply it to the determinant of the Jacobian of the flow map ($\mathbf{S}(t) := D_x \Phi(\hat{x}, t)$, \hat{x} regarded as parameter)

$$\begin{aligned} \frac{\partial}{\partial t} \det D_x \Phi(\hat{x}, t) &= \det D_x(\hat{x}, t) \operatorname{tr}\left(\frac{\partial}{\partial t} D_x \Phi(\hat{x}, t) \cdot D_x \Phi(\hat{x}, t)\right) \\ &\stackrel{(11.1.1.14)}{=} \det D_x \Phi(\hat{x}, t) \operatorname{div} \mathbf{v}(\Phi(\hat{x}, t), t), \end{aligned}$$

because $\operatorname{div} \mathbf{v}(x, t) = \operatorname{tr} D_x \mathbf{v}(x, t)$. Thus the determinant $\rho(\hat{x}, t) := \det D_x \Phi(\hat{x}, t)$ solves the initial-value problem

$$\begin{cases} \frac{\partial}{\partial t} \rho(\hat{x}, t) = \operatorname{div} \mathbf{v}(\Phi(\hat{x}, t), t) \rho(\hat{x}, t), & (\hat{x}, t) \in \mathbb{R}^d \times \mathbb{R}, \\ \rho(\hat{x}, 0) = 1, \end{cases} \quad (11.1.1.15)$$

By the uniqueness of solutions of initial-value problems for ODEs we infer that $\rho(\hat{x}, t) > 0$ for all $(\hat{x}, t) \in \mathbb{R}^d \times \mathbb{R}$. Hence, we have the equivalence

$$\begin{aligned} u(x, t) &= |\det(D_x \Phi)(\hat{x}, t)|^{-1} u_0(\hat{x}) \\ &\Updownarrow \\ &\rho(\hat{x}, t) u(x, t) = u_0(\hat{x}) \end{aligned} \quad (\hat{x}, t) \in \mathbb{R}^d \times \mathbb{R}, \quad x = \Phi(\hat{x}, t).$$

We differentiate the second equation with respect to time. By the product rule and the chain rule we get

$$\frac{\partial \rho}{\partial t}(\hat{x}, t) u(x, t) + \rho(\hat{x}, t) \left(\operatorname{grad} u(x, t) \cdot \frac{\partial \Phi}{\partial t}(\hat{x}, t) + \frac{\partial u}{\partial t}(x, t) \right) = 0.$$

Then plug in (11.1.1.15) and divide by $\rho(\hat{x}, t) \neq 0$:

$$\Rightarrow \operatorname{div} \mathbf{v}(x, t) u(x, t) + \operatorname{grad} u(x, t) \cdot \frac{\partial \Phi}{\partial t}(\hat{x}, t) + \frac{\partial u}{\partial t}(x, t) = 0.$$

The flow equation

$$\frac{\partial \Phi}{\partial t}(\hat{x}, t) = \mathbf{v}(\Phi(\hat{x}, t), t), \quad (\hat{x}, t) \in \mathbb{R}^d \times \mathbb{R},$$

along with the general product rule from Lemma 1.5.2.1 finish the proof. \square

Remark 11.1.1.16 (Conservation of volume content) Let the function $u : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}$ be defined as in the previous theorem

$$u(x, t) = |\det(D_x \Phi)(\hat{x}, t)|^{-1} u_0(\hat{x}), \quad (\hat{x}, t) \in \mathbb{R}^d \times \mathbb{R}, \quad x = \Phi(\hat{x}, t), \quad (11.1.1.9)$$

where $u_0 : \mathbb{R}^d \rightarrow \mathbb{R}$ and $\Phi : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ is the flow induced by the continuously differentiable velocity field $\mathbf{v} : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ according to (11.1.1.6).

Recall the transformation formula for multi-dimensional integrals

Theorem 0.3.2.31. Transformation formula for integrals

Given two domains $\widehat{\Omega}, \Omega \subset \mathbb{R}^d$ and a continuously differentiable mapping $\Phi : \widehat{\Omega} \rightarrow \Omega$, then

$$\int_{\Omega} f(x) dx = \int_{\widehat{\Omega}} f(\Phi(\widehat{x})) |\det D\Phi(\widehat{x})| d\widehat{x} \quad (0.3.2.32)$$

for any integrable function $f : \Omega \rightarrow \mathbb{R}$.

it shows that for any bounded “control volume” $V \subset \mathbb{R}^d$ and time $t \in \mathbb{R}$

$$\int_{\Phi(V,t)} u(x, t) dx = \int_V u(\Phi(\widehat{x}, t), t) |\det D_x \Phi(\widehat{x}, t)| d\widehat{x} \stackrel{(11.1.1.9)}{=} \int_V u_0(\widehat{x}) d\widehat{x} \quad \forall t \in \mathbb{R}.$$

Thus, when $u = u(x, t)$ solves (11.1.1.5),

the “content of a control volumne V ” given by the integral of u over V does not change as V moves with the flow.

Remark 11.1.1.17 (Boundary conditions for linear advection) Recall the discussion in Section 10.2.1 and Section 10.3.2, cf. solution formula (10.3.2.7):

For the scalar linear advection initial boundary value problem

$$\frac{\partial u}{\partial t} + \operatorname{div}(\mathbf{v}(x, t)u) = f(x, t) \quad \text{in } \widetilde{\Omega} := \Omega \times]0, T[, \quad (11.1.1.18)$$

$$u(x, 0) = u_0(x) \quad \text{for all } x \in \Omega , \quad (11.1.1.19)$$

on a bounded domain $\Omega \subset \mathbb{R}^d$, **boundary conditions** (e.g., prescribed temperature)

$$u(x, t) = g(x, t) \quad \text{on } \Gamma_{\text{in}}(t) \times]0, T[,$$

can be imposed only on the **inflow boundary**

$$\Gamma_{\text{in}}(t) := \{x \in \partial\Omega : \mathbf{v}(x, t) \cdot \mathbf{n}(x) < 0\} , \quad 0 < t < T . \quad (11.1.1.20)$$

Note: Γ_{in} can change with time!

Bottom line:

Knowledge of local and current direction of transport
is needed to impose meaningful boundary conditions!

11.1.2 Traffic Modeling [BD11]

We design simple mathematical models for non-stationary traffic flow on a *single long highway lane*. This situation often occurs, for instance, at bypasses of long highway construction sites.

We make simplifying *modeling assumptions* (not quite matching reality):

- ◆ Identical cars and behavior of drivers (11.1.2.1)

- ◆ Uniformity of road conditions (11.1.2.2)
- ◆ Speed of a car determined only by (its distance from) the car in front (11.1.2.3)

11.1.2.1 Particle Model

The gist of a particle model or agent based model for traffic flow is to track a finite number of individual cars over a period of time $[0, T]$. Hence, the particle model is **semi-discrete** (still continuous in time). The key state parameter of a car is its position on the road:

$x_i(t) \hat{=} \text{position of } i\text{-th car at time } t, i = 1, \dots, N$ ($N \hat{=} \text{total number of cars}$), hence the **configuration space** is \mathbb{R}^N .

We will always take for granted ordering:

$$x_i(t) < x_{i+1}(t)$$

The curve $t \mapsto x_i(t)$ in the $x - t$ -plane is the **trajectory** of the i -th car.

§11.1.2.4 (Velocity model) In order to describe the dynamics of the moving cars we need a *velocity model*.

► Here: **optimal velocity model**

$$\dot{x}_i(t) = v_{\text{opt}}(\Delta x_i) , \quad \Delta x_i(t) = x_{i+1}(t) - x_i(t) > 0 , \quad i = 1, \dots, N-1 . \quad (11.1.2.5)$$

↔ relies on Assumptions (11.1.2.1)–(11.1.2.3) above, in particular (11.1.2.3).

The function $\Delta x \mapsto v_{\text{opt}}(\Delta x)$ is deduced from the assumption that

each car drives as fast as possible under safety constraints.
(drive more slowly if you are close to the car in front)

$$\blacktriangleright \quad v_{\text{opt}}(\Delta x) = v_{\max} \left(1 - \frac{\Delta_0}{\Delta x}\right) , \quad (11.1.2.6)$$

with $\Delta_0 \hat{=} \text{length of a car} = \text{distance of cars in bumper to bumper traffic jam}$.

► (11.1.2.5) + (11.1.2.6): **ordinary differential equation** (ODE) on state space \mathbb{R}^N ↴

In order to get a well-posed initial value problem, the ODE has to be supplemented with **initial conditions**

$$x_i(0) = x_{i,0} \in \mathbb{R} , \quad x_{i,0} \leq x_{i+1,0} - \Delta_0 . \quad (11.1.2.7)$$

Obviously (why?): the solution of (11.1.2.5), (11.1.2.6), (11.1.2.7) satisfies $x_i(t) \leq x_{i+1}(t) - \Delta_0$.

Remark 11.1.2.8 (Acceleration based traffic modeling) The speed of a car is a consequence of drivers accelerating and breaking.

➤ acceleration based modeling of car dynamics under Assumptions (11.1.2.1)–(11.1.2.3)

$$\ddot{x}_i(t) = F(\Delta x_i(t), \Delta v_i(t)) , \quad \Delta v_i(t) = \dot{x}_{i+1}(t) - \dot{x}_i(t) . \quad (11.1.2.9)$$

Models of this type are popular in practice. ↴

EXPERIMENT 11.1.2.10 (Particle simulation of traffic flow) Usually one sets $v_{\max} = 1$ by rescaling of spatial/temporal units, cf. Rem. 1.2.1.25.

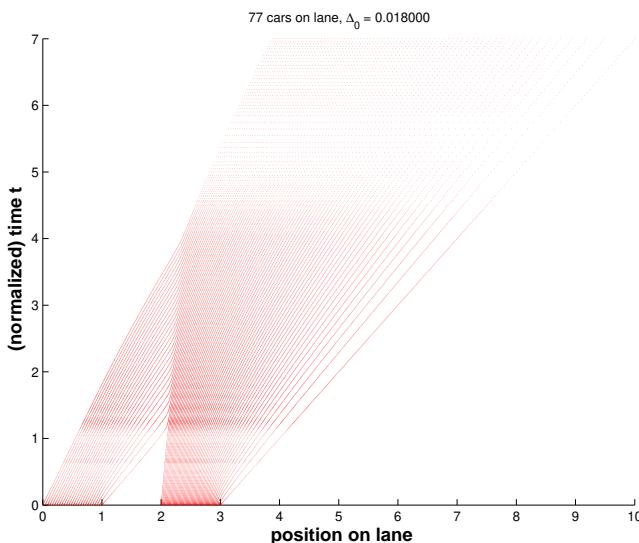


Fig. 448

We chose the following initial positions of cars:

- 26 cars equidistributed in $[0, 1]$,
- 51 cars at equidistant positions in $[2, 3]$.

This corresponds to two clusters of evenly spaced cars at different sections of the road.

▷ Simulation based on optimal velocity model (11.1.2.6) with (dimensionless) $\Delta_0 = 0.0180$.

When we launch the simulation we observe that the two clusters merge and dissolve as cars “escape” to the right. *Fan-shaped* patterns emerge, see Fig. 449. □

§11.1.2.11 (Extraction of macroscopic quantities) Our goal is to pass from the semi-discrete particle model to a *continuum model*, where the state of traffic is described by functions.

These correspond to “macroscopic quantities” $\hat{=}$ quantities describing the traffic flow detached from the existence of individual cars.

Macroscopic quantities can be obtained by *averaging* from the *microscopic* particle description.

Key macroscopic quantity:

(normalized) **density** of cars

$$u_\delta(x, t) := \frac{\Delta_0}{2\delta} \#\{i \in \{1, \dots, N\} : x - \delta \leq x_i(t) < x + \delta\}, \quad (11.1.2.12)$$

where $\delta > 0$ is the **spatial averaging length**. (The density defined in (11.1.2.12) is “normalized” because it is the ratio of the number density of cars and the maximal density Δ_0^{-1} . Hence, invariably, $0 \leq u_\delta(x, t) \leq 1$.)

Note: u_δ will crucially depend on δ

EXPERIMENT 11.1.2.13 (Particle simulation of traffic flow, cnt'd → Exp. 11.1.2.10)

We use initial car distribution

- $k/2$ cars equidistributed in $[0, 1]$,
 - k cars at equidistant positions in $[2, 3]$.
- with $k=50, 200, 800$, $\Delta_0 = 0.9/k$, see (11.1.2.6), $\delta = 3.33/k$ in (11.1.2.12), simulation using explicit Runge-Kutta timestepping of order 5.

Initial density $x \mapsto u_\delta(x, 0)$

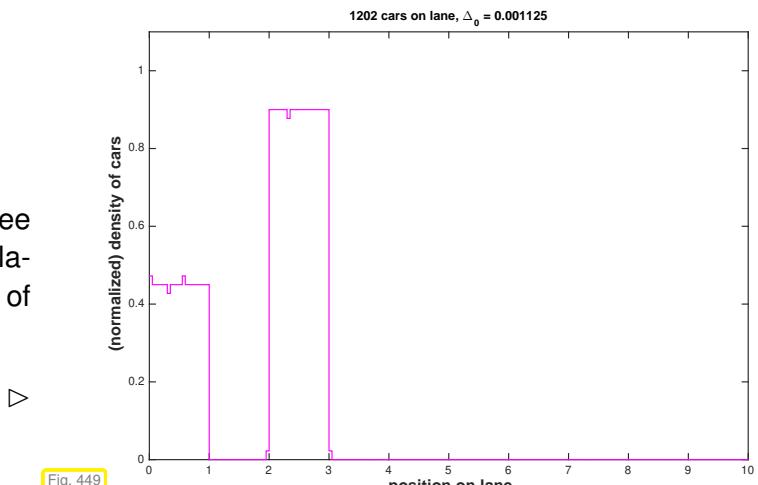
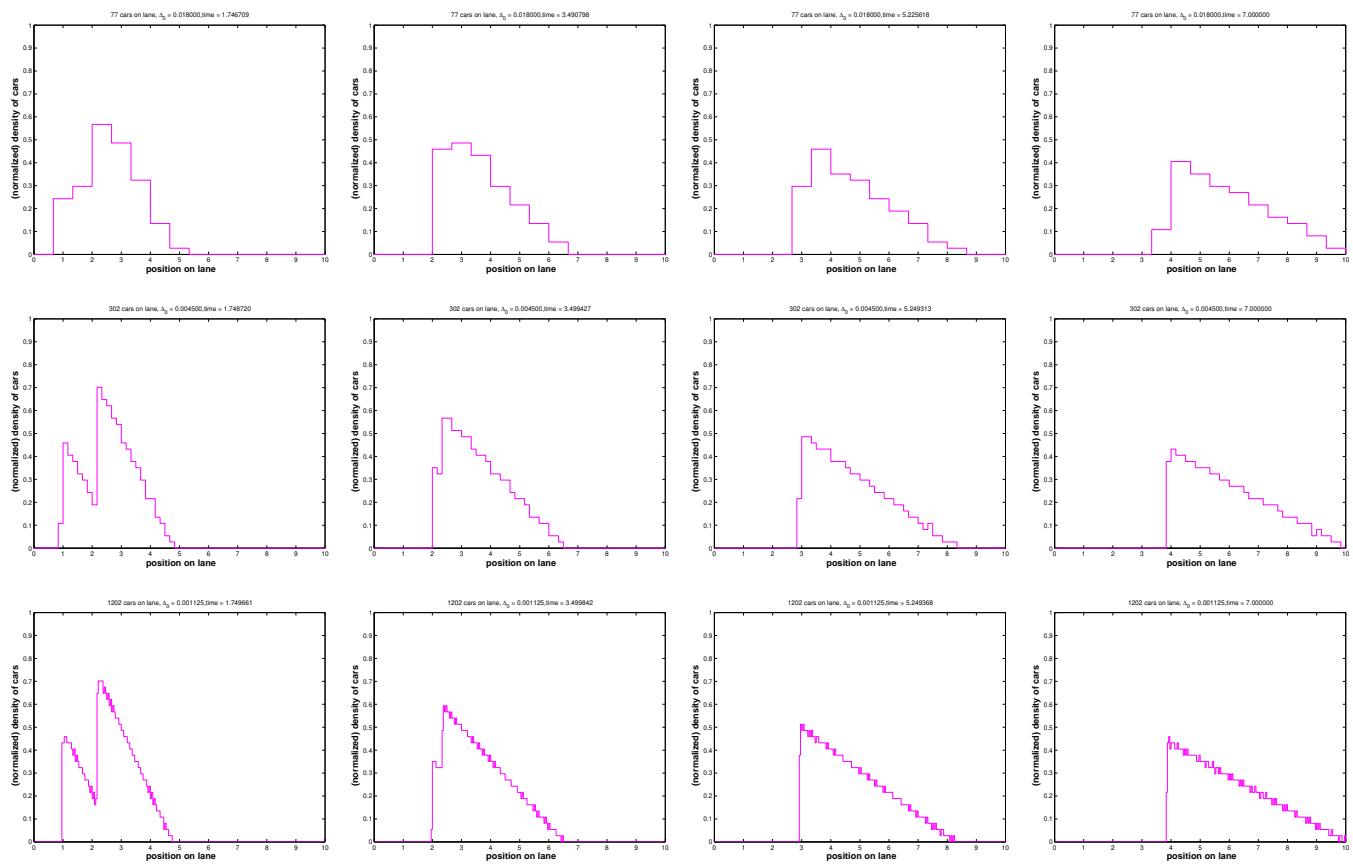


Fig. 449



Striking observation:

For $N \rightarrow \infty$, $\Delta_0 \sim N^{-1}$, $\delta \sim N^{-1}$ the normalized car densities $u_\delta(x, t)$ seem to approach a *limit density*. What is it? Can it be obtained as a solution of a “limit model”. These issues will be addressed next. \square

Note: We have made similar observation in the case of the mass-spring model of Section 5.1.1.2 in the limit $n \rightarrow \infty$.

11.1.2.2 Continuum Traffic Model

In Exp. 11.1.2.13 we observed the emergence of a stable limit density in the microscopic particle model of traffic flow according to (11.1.2.5) and (11.1.2.6), when the number of cars and their maximum density tended to ∞ in tandem, while the spatial averaging length tends to zero.

Now we derive a **macroscopic continuum model** describing this limit. This macroscopic model will be stated in terms of macroscopic quantities, which are functions of position along the road x and time t .

Note: There are many parallels with derivation of continuum elastic string model in Section 5.1.1.

Remark 11.1.2.14 (Suitability of macroscopic models for traffic flow) The limit $N \rightarrow \infty$ in traffic modeling is commonly denounced as dubious, because the number of cars on a road is way too small to render the limit a good approximation of actual traffic flow, see [BD11, Sect. 2.3].

Nevertheless, here we introduce a limit model, because

- ◆ it yields a least a qualitatively correct representation of patterns observed in real traffic flow,
- ◆ it provides an important **model problem** for scalar non-linear conservation laws, see Section 11.1.3.

Ingredients of macroscopic (continuum) traffic model:

- spatial domain $\Omega = \mathbb{R} \hat{=} \text{infinitely long single highway lane} (\rightarrow \text{Cauchy problem}),$
- traffic flow described by the macroscopic quantity

normalized density of cars $u : \Omega \times [0, T] \mapsto [0, 1]$ according to

$$u_\delta(x, t) := \frac{\Delta_0}{\delta} \#\{i \in \{1, \dots, N\} : x - \delta \leq x_i(t) < x + \delta\}, \quad (11.1.2.12)$$

- optimal velocity speed model (11.1.2.6) ($v_{\text{opt}}(\Delta x) = v_{\max}(1 - \frac{\Delta_0}{\Delta x})$).

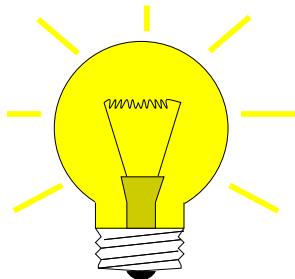
§11.1.2.15 (Macroscopic balance laws for traffic model) However, (11.1.2.6) and (11.1.2.5) do not fit the spirit of macroscopic modeling: neither Δx_i nor $\dot{x}_i(t)$ is a macroscopic quantity!

Required:

concept of a **macroscopic velocity**

Idea:

spatial averaging of velocities of cars



$$v_\delta(x, t) = \frac{\sum_{i \in \mathcal{U}_\delta(x)} \dot{x}_i(t)}{\#\mathcal{U}_\delta}, \quad (11.1.2.16)$$

$$\mathcal{U}_\delta(x) := \{i \in \{1, \dots, N\} : x - \delta \leq x_i(t) < x + \delta\}.$$

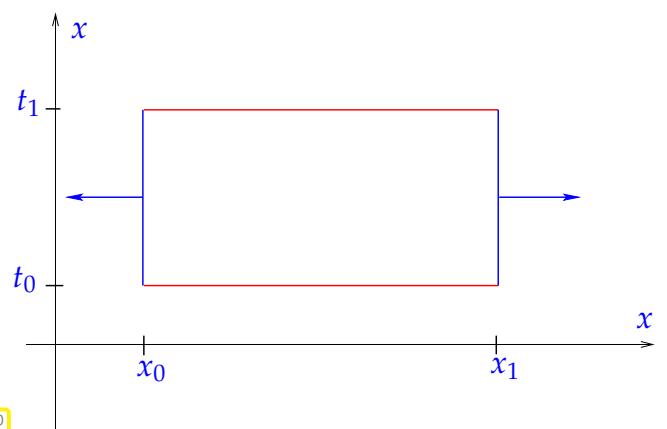


From density and velocity we derive another macroscopic quantity:

$$\text{(normalized) flux of cars: } q_\delta(x, t) = u_\delta(x, t)v_\delta(x, t). \quad (11.1.2.17)$$

Interpretation: $q_\delta(x, t) \approx \text{no. of cars passing site } x \text{ in unit time around instance } t \text{ in time.}$

approximate **balance law**
("conservation of cars" in a "space-time box")



$$\underbrace{\int_{x_0}^{x_1} u_\delta(x, t_1) dt - \int_{x_1}^{x_0} u_\delta(x, t_0) dt}_{\text{change of no. of cars on } [x_0, x_1] \text{ in } [t_0, t_1]} \approx \underbrace{\int_{t_0}^{t_1} q_\delta(x_0, t) dx - \int_{t_0}^{t_1} q_\delta(x_1, t) dx}_{\text{no. of cars entering/leaving } [x_0, x_1] \text{ in } [t_0, t_1]}. \quad (11.1.2.18)$$

§11.1.2.19 (Traffic flow: continuum limit of particle model) Now we consider $N \rightarrow \infty$ (many cars) and $\delta \sim N^{-1} \rightarrow 0$ and drop the subscript δ , which hints at the averaging.

The balance law (11.1.2.20) will remain valid in the limit and will even become exact !

$$\underbrace{\int_{x_0}^{x_1} u(x, t_1) dt - \int_{x_1}^{x_0} u(x, t_0) dt}_{\text{change of no. of cars on } [x_0, x_1] \text{ in } [t_0, t_1]} = \underbrace{\int_{t_0}^{t_1} q(x_0, t) dx - \int_{t_0}^{t_1} q(x_1, t) dx}_{\text{no. of cars entering/leaving } [x_0, x_1] \text{ in } [t_0, t_1]} . \quad (11.1.2.20)$$

In the “infinitely many cars” limit $u(x, t)$, $v(x, t)$, and $q(x, t)$ can be expected to become (piecewise) smooth functions. This justifies the transition to a **differential (PDE) macroscopic model**:

Temporarily assume that $u = u(x, t)$ is smooth in both x and t and set $x_1 = x_0 + h$, $t_1 = t_0 + \tau$. First approximate the integrals in (11.1.2.20).

$$\begin{aligned} \int_{x_0}^{x_1} u(x, t_1) - u(x, t_0) dx &= h(u(x_0, t_1) - u(x_0, t_0)) + O(h^2) \quad \text{for } h \rightarrow 0 , \\ \int_{t_0}^{t_1} q(x_1, t) - q(x_0, t) dt &= \tau(q(x_1, t_0) - q(x_0, t_0)) + O(\tau^2) \quad \text{for } \tau \rightarrow 0 . \end{aligned}$$

Then employ Taylor expansion for the differences:

$$\begin{aligned} u(x_0, t_1) - u(x_0, t_0) &= \frac{\partial u}{\partial t}(x_0, t_0)\tau + O(\tau^2) \quad \text{for } \tau \rightarrow 0 , \\ q(x_1, t_0) - q(x_0, t_0) &= \frac{\partial q}{\partial x}(x_0, t_0)h + O(h^2) \quad \text{for } h \rightarrow 0 . \end{aligned}$$

Finally, divide by h and τ and take the limit $\tau \rightarrow 0$, $h \rightarrow 0$:

$$\Rightarrow \frac{\partial u}{\partial t}(x, t) + \frac{\partial q}{\partial x}(x, t) = 0 \quad \text{in } \Omega \times]0, T[. \quad (11.1.2.21)$$

This is a first-order partial differential equation.

We still need to link u and q : From (11.1.2.6) (with $v_{\max} = 1$ after rescaling) we deduce the macroscopic **constitutive relationship** between the (averaged and normalized) density (\rightarrow (11.1.2.12)) of cars and their averaged speed (\rightarrow (11.1.2.16)):

$$v(x, t) = 1 - u(x, t) \stackrel{(11.1.2.17)}{\Rightarrow} q(x, t) = u(x, t)(1 - u(x, t)) . \quad (11.1.2.22)$$

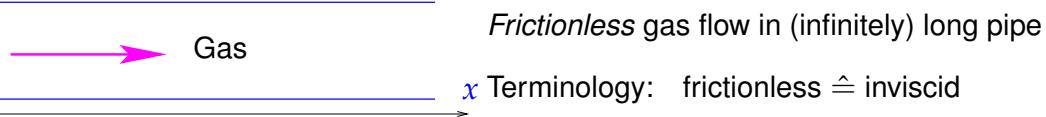
$$(11.1.2.21) \& (11.1.2.22) \& (11.1.2.17) \Rightarrow \boxed{\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u(1 - u)) = 0 \quad \text{in } \Omega \times]0, T[} . \quad (11.1.2.23)$$

+ macroscopic counterpart of initial conditions (11.1.2.7):

$$u(x, 0) = u_0(x) , \quad x \in \mathbb{R} . \quad (11.1.2.24)$$

11.1.3 Inviscid Gas Flow

Introduction. In this section we study modeling in **fluid mechanics**, a special field of continuum mechanics. In spirit this is close to the modeling of traffic flow in Section 11.1.2, because the macroscopic behavior of fluids also results from the interaction of many small particles (molecules). However, in fluid mechanics the limit model for infinitely many particles enjoys a much more solid foundation than that for traffic, because the number of particles involved is tremendous ($\approx 10^{20} - 10^{30}$).



Assumption: variation of gas density negligible (“near incompressibility”) motion of fluid driven by inertia \leftrightarrow *conservation of linear momentum*

§11.1.3.1 (Inviscid gas flow: balance law) We derive a **continuum model** for inviscid, nearly incompressible fluid in a straight infinitely long pipe $\leftrightarrow \Omega = \mathbb{R}$ (Cauchy problem).

This simple model will be based on *conservation of linear momentum*, whereas conservation of mass and energy will be neglected (and violated). Hence, the crucial conserved quantity will be the momentum.

by near incompressibility
Unknown: $u = u(x, t)$ = momentum *density* \sim local velocity $v = v(x, t)$ of fluid

Conserved quantity: (linear) **momentum** of fluid $u = u(x, t)$

➤ flux of linear momentum $f \sim v \cdot u$ (after scaling: $f(u) = \frac{1}{2}u \cdot u$)
("momentum u advected by velocity u ")

Conservation of linear momentum ($\sim u$): for all control volumes $V :=]x_0, x_1[\subset \Omega$:

$$\underbrace{\int_{x_0}^{x_1} u(x, t_1) - u(x, t_0) dx}_{\text{change of momentum in } V} + \underbrace{\int_{t_0}^{t_1} \frac{1}{2}u^2(x_1, t) - \frac{1}{2}u^2(x_0, t) dt}_{\text{outflow of momentum}} = 0 \quad \forall 0 < t_0 < t_1 < T . \quad (11.1.3.2)$$

§11.1.3.3 (Burgers equation modelling inviscid gas flow) Temporarily assume that $u = u(x, t)$ is smooth in both x and t and set $x_1 = x_0 + h$, $t_1 = t_0 + \tau$. First approximate the integrals in (11.1.3.2).

$$\begin{aligned} \int_{x_0}^{x_1} u(x, t_1) - u(x, t_0) dx &= h(u(x_0, t_1) - u(x_0, t_0)) + O(h^2) \quad \text{for } h \rightarrow 0 , \\ \int_{t_0}^{t_1} \frac{1}{2}u^2(x_1, t) - \frac{1}{2}u^2(x_0, t) dt &= \tau(\frac{1}{2}u^2(x_1, t_0) - \frac{1}{2}u^2(x_0, t_0)) + O(\tau^2) \quad \text{for } \tau \rightarrow 0 . \end{aligned}$$

Then employ Taylor expansion for the differences:

$$u(x_0, t_1) - u(x_0, t_0) = \frac{\partial u}{\partial t}(x_0, t_0)\tau + O(\tau^2) \quad \text{for } \tau \rightarrow 0 ,$$

$$\frac{1}{2}u^2(x_1, t_0) - \frac{1}{2}u^2(x_0, t_0) = \frac{\partial}{\partial x}(\frac{1}{2}u^2)(x_0, t_0)h + O(h^2) \quad \text{for } h \rightarrow 0.$$

Finally, divide by h and τ and take the limit $\tau \rightarrow 0, h \rightarrow 0$:

$$\Rightarrow \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(\frac{1}{2}u^2) = 0 \quad \text{in } \Omega \times]0, T[. \quad (11.1.3.4)$$

(11.1.3.4) = **Burgers equation**: a one-dimensional scalar conservation law (without sources) \square

Remark 11.1.3.5 (Euler equations) The above gas model blatantly ignores the fundamental laws of conservation of mass and of energy. These are taken into account in a famous more elaborate model of inviscid fluid flow:

Euler equations [Chr07], a more refined model for inviscid gas flow in an infinite pipe

$$\begin{aligned} \frac{\partial}{\partial t} \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} + \frac{\partial}{\partial x} \begin{pmatrix} \rho u \\ \rho u^2 + p \\ (E + p)u \end{pmatrix} &= 0 \quad \text{in } \mathbb{R} \times]0, T[, \\ u(x, 0) = u_0(x), \quad \rho(x, 0) = \rho_0(x), \quad E(x, 0) = E_0(x) &\quad \text{for } x \in \mathbb{R}, \end{aligned} \quad (11.1.3.6)$$

where

- ◆ $\rho = \rho(x, t) \hat{=} \text{fluid density}$, $[\rho] = \text{kg m}^{-1}$,
- ◆ $u = u(x, t) \hat{=} \text{fluid velocity}$, $[u] = \text{m s}^{-1}$,
- ◆ $p = p(x, t) \hat{=} \text{fluid pressure}$, $[p] = \text{N}$,
- ◆ $E = E(x, t) \hat{=} \text{total energy density}$, $[E] = \text{J m}^{-1}$.

+ **state equation** (material specific constitutive equations), e.g., for ideal gas

$$p = (\gamma - 1)(E - \frac{1}{2}\rho u^2), \quad \text{with adiabatic index } 0 < \gamma < 1.$$

Conserved quantities (**densities**):

$$\rho \leftrightarrow \text{mass density}, \quad \rho u \leftrightarrow \text{momentum density}, \quad E \leftrightarrow \text{energy density}.$$

Underlying physical conservation principles for individual densities:

- First equation $\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u) = 0 \leftrightarrow \text{conservation of mass}$,
- Second equation $\frac{\partial(\rho u)}{\partial t} + \frac{\partial}{\partial x}(\rho u^2 + p) = 0 \leftrightarrow \text{conservation of momentum}$,
- Third equation $\frac{\partial E}{\partial t} + \frac{\partial}{\partial x}((E + p)u) = 0 \leftrightarrow \text{conservation of energy}$.

Euler equations (11.1.3.6) = non-linear **system of conservation laws** (in 1D)

As is typical of non-linear systems of conservation laws, the analysis of the Euler equations is intrinsically difficult: hitherto not even existence and uniqueness of solutions for general initial values could be established. Moreover, solutions display a wealth of complicated structures. Therefore, this course is confined to scalar conservation laws, for which there is only one unknown real-valued function of space and time. \square

Review question(s) 11.1.3.7 (Conservation laws: Examples)

(Q11.1.3.7.A) Consider the Cauchy problem

$$\begin{aligned} \frac{\partial u}{\partial t} + \operatorname{div}(\mathbf{v}(\mathbf{x})u) &= 0 \quad \text{in } \tilde{\Omega} := \mathbb{R}^d \times]0, T[, \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathbb{R}^d \quad (\text{initial conditions}) . \end{aligned} \quad (11.1.1.5)$$

for linear advection for $d = 2$ and the velocity field $\mathbf{v}(\mathbf{x}) = \begin{bmatrix} -x_2 \\ x_1 \end{bmatrix}$. Write down the solution $u = u(\mathbf{x}, t)$ in terms of the initial data $u_0 = u_0(\mathbf{x})$.

(Q11.1.3.7.B) Let $\Phi : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ be the flow induced by a smooth velocity field $\mathbf{v} : \mathbb{R}^d \rightarrow \mathbb{R}^d$. Which partial differential equations does the function

$$u(\mathbf{x}, t) := u_0(\hat{\mathbf{x}}), \quad \mathbf{x} = \Phi(\hat{\mathbf{x}}, t), \quad \hat{\mathbf{x}} \in \mathbb{R}^d$$

satisfy. Here $u_0 : \mathbb{R}^d \rightarrow \mathbb{R}$ is a given differentiable function with bounded support.

(Q11.1.3.7.C) In an $x - t$ diagram sketch the trajectory of a car starting at $t = 0, x = 0$ and moving with *constant acceleration* to right.

(Q11.1.3.7.D) Which traffic flow conservation law arises, when the speed law

$$v_{\text{opt}}(\Delta x) = v_{\max} \left(1 - \frac{\Delta_0}{\Delta x}\right)$$

is replaced with

$$v_{\text{opt}}(\Delta x) = v_{\max} \cos\left(\frac{\pi}{2} \frac{\Delta_0}{\Delta x}\right).$$

△

11.2 Scalar Conservation Laws in 1D

This section will give a brief introduction to the mathematics of an important class of mathematical models leading to scalar conservation laws in one spatial dimension. We will first present the general form of the underlying partial differential equation, and then define a suitable notion of “solution”, the weak solutions, which can even be discontinuous. Their properties will be discussed in the remaining sub-sections, in order to understand, what features approximate numerical solutions should possess.

11.2.1 Integral and Differential Form

In Section 11.1 we have derived the following evolution equations for the real-valued unknown function $u = u(\mathbf{x}, t)$ for phenomena with dominant transport and an underlying conservation principle.

$$\text{linear advection: } \frac{\partial}{\partial t}(\rho u) + \operatorname{div}(\mathbf{v}(\mathbf{x}, t)(\rho u)) = f(\mathbf{x}, t) \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}^d) , \quad (11.1.1.3)$$

$$\text{Burgers equation: } \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} \left(\frac{1}{2} u^2 \right) = 0 \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}) , \quad (11.1.3.4)$$

$$\text{traffic flow equation: } \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} (u(1-u)) = 0 \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}) . \quad (11.1.2.23)$$

Now, we learn about a class of evolution problems to which these three belong, namely the class of **conservation laws**. In order to introduce the general concept we first need some notations and terminology:

- ◆ $\Omega \subset \mathbb{R}^d \doteq$ fixed (bounded/unbounded) spatial domain. In the case $\Omega = \mathbb{R}^d$ we deal with a **Cauchy problem** on an unbounded spatial domain without boundary.

- ◆ The computational domain is the space-time cylinder $\tilde{\Omega} := \Omega \times]0, T[$, $T > 0$ final time
- ◆ $U \subset \mathbb{R}^m$ ($m \in \mathbb{N}$) $\hat{=}$ phase space (state space) for the conserved quantities u_i (usually $U = \mathbb{R}^m$).
A vector $\in U$ is often called a state.
In most of this chapter our focus will be on the scalar case $m = 1$.

The following relationship gives the integral form of a general scalar conservation law.

Conservation law for transient state distribution $u : \tilde{\Omega} \mapsto \mathbb{R}$: $u = u(x, t)$, for $0 \leq t \leq T$:

$$\frac{d}{dt} \int_V u \, dx + \int_{\partial V} \mathbf{f}(u, x) \cdot \mathbf{n} \, dS(x) = \int_V s(u, x, t) \, dx \quad \forall \text{ "control volumes" } V \subset \Omega . \quad (11.2.1.1)$$

↑ change of amount in V ↑ inflow/outflow ↑ production term

Terminology:

- ▷ **flux function** $\mathbf{f} : U \times \Omega \mapsto \mathbb{R}^d$
- ▷ **source function** $s : U \times \Omega \times]0, T[\mapsto \mathbb{R}$ (here usually $s = 0$)

Remark 11.2.1.2 (Conservation law and flux for transient heat conduction) Note that (11.2.1.1) has the same structure as the “conservation of energy law” (9.2.1.3) for heat conduction that we recall here. If $u = u(x, t)$ denotes the temperature, $\rho > 0$ the heat capacity, and f models heat sources, then conservation of energy can be expressed as

$$\frac{d}{dt} \int_V \rho u \, dx + \int_{\partial V} \mathbf{j} \cdot \mathbf{n} \, dS = \int_V f \, dx \quad \text{for all "control volumes" } V \quad (9.2.1.3)$$

↑ energy stored in V ↑ power flux through ∂V ↑ heat generation in V

In this case the heat flux was given by

$$\text{Fourier's law} \quad \mathbf{j}(x) = -\kappa(x) \operatorname{grad} u(x), \quad x \in \Omega , \quad (1.6.0.5)$$

or its extended version (10.1.2.4). In Fourier's law (1.6.0.5) the flux is a *linear* function of first-order *derivatives* of u . \square

In contrast to Fourier's law (1.6.0.5), for the **flux function** $\mathbf{f} : U \times \Omega \mapsto \mathbb{R}^d$ in (11.2.1.1) we assume

\mathbf{f} depends on the local state u , not on derivatives of u : $f(u, x) = f(u(x), x)$.

In another respect we go far beyond Fourier's law, since

\mathbf{f} will, in general, be a *non-linear* function of u !

Remark 11.2.1.3 (Diffusive flux) Taking into account the relationship with heat “diffusion”, a flux function of the form of Fourier's law (1.6.0.5)

$$\mathbf{f}(u) = -\kappa(x) \operatorname{grad} u ,$$

is called a **diffusive flux**. \square

We can recast (11.2.1.1) in two ways. On one hand, we can integrate (11.2.1.1) over the time period $[t_0, t_1] \subset [0, T]$ and use the fundamental theorem of calculus in the time direction:

- Space-time integral form of (11.2.1.1), cf. (11.1.3.2),

$$\int_V u(\mathbf{x}, t_1) d\mathbf{x} - \int_V u(\mathbf{x}, t_0) d\mathbf{x} + \int_{t_0}^{t_1} \int_{\partial V} \mathbf{f}(u, \mathbf{x}) \cdot \mathbf{n} dS(\mathbf{x}) dt = \int_{t_0}^{t_1} \int_V s(u, \mathbf{x}, t) d\mathbf{x} dt \quad (11.2.1.4)$$

for all $V \subset \Omega$, $0 < t_0 < t_1 < T$, $\mathbf{n} \doteq$ exterior unit normal at ∂V

On the other hand we can apply Gauss' theorem Thm. 1.5.2.4 in space.

- (local) differential form of (11.2.1.1):

$$\frac{\partial}{\partial t} u + \operatorname{div}_{\mathbf{x}} \mathbf{f}(u, \mathbf{x}) = s(u, \mathbf{x}, t) \quad \text{in } \tilde{\Omega} \quad . \quad (11.2.1.5)$$

↑
div acting on spatial variable \mathbf{x} only

In the special case $d = 1$ the differential form (11.2.1.5) of a one-dimensional scalar conservation law for the “density” $u : \tilde{\Omega} \mapsto \mathbb{R}$ is

$$\frac{\partial u}{\partial t}(x, t) + \frac{\partial}{\partial x}(f(u(x, t), x)) = s(u(x, t), x, t) \quad \text{in }]\alpha, \beta[\times]0, T[, \quad \alpha, \beta \in \mathbb{R} \cup \{\pm\infty\} . \quad (11.2.1.6)$$

In any case, these evolution equations have to be supplemented with initial conditions $u(\mathbf{x}, 0) = u_0(\mathbf{x})$, $\mathbf{x} \in \Omega$

EXAMPLE 11.2.1.7 (Flux functions for simple scalar conservation laws) The examples given above,

linear advection: $\frac{\partial}{\partial t}(\rho u) + \operatorname{div}(\mathbf{v}(\mathbf{x}, t)(\rho u)) = f(\mathbf{x}, t) \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}^d) , \quad (11.1.1.3)$

Burgers equation: $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{1}{2}u^2\right) = 0 \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}) , \quad (11.1.3.4)$

traffic flow equation: $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u(1-u)) = 0 \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}) , \quad (11.1.2.23)$

are all written in the form (11.2.1.5) and, thus, the corresponding flux functions are easily found:

- ◆ For Burgers equation (11.1.3.4): $f(u, x) = f(u) = \frac{1}{2}u^2$, $s = 0$,
- ◆ For traffic flow equation (11.1.2.23): $f(u, x) = f(u) = u(1-u)$, $s = 0$,
- ◆ For linear advection (11.1.1.3): $\mathbf{f}(u, \mathbf{x}) = \mathbf{v}(\mathbf{x}, t)u$, $s = f(\mathbf{x}, t)$
(Note: in this case the conserved quantity is actually ρu , which was again denoted by u)

□

Remark 11.2.1.8 (Boundary values for conservation laws) What are suitable boundary values on $\partial\Omega \times]0, T[$? Usually this is a tricky question and the answer can be very different for different flux functions.

To see why, remember the discussion in Rem. 11.1.1.17: meaningful boundary conditions hinge on knowledge of local (in space and time) transport directions, which, in a *non-linear* conservation law, will usually depend on the unknown solution $u = u(\mathbf{x}, t)$.

This obviously compounds difficulties and to avoid these murky issues we consider only **Cauchy problems** in this chapter. \square

Review question(s) 11.2.1.9 (Integral and differential forms of scalar conservation laws)

(Q11.2.1.9.A) The differential form of a generic **scalar conservation law** is

$$\frac{\partial}{\partial t} u + \operatorname{div}_x \mathbf{f}(u, x) = s(u, x, t) \quad \text{in } \tilde{\Omega} := \Omega \times [0, T], \quad (11.2.1.5)$$

with **flux function** $\mathbf{f} : U \times \Omega \rightarrow \mathbb{R}^d$ and **source function** s . What is the integral form of (11.2.1.5)?

(Q11.2.1.9.B) Identify the flux functions and source functions for the following scalar conservation laws:

$$\text{linear advection: } \frac{\partial}{\partial t}(\rho u) + \operatorname{div}(\mathbf{v}(x, t)(\rho u)) = f(x, t) \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}^d), \quad (11.1.1.3)$$

$$\text{Burgers equation: } \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0 \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}), \quad (11.1.3.4)$$

$$\text{traffic flow equation: } \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u(1-u)) = 0 \quad \text{in } \Omega \times]0, T[, \quad (\Omega \subset \mathbb{R}). \quad (11.1.2.23)$$

(Q11.2.1.9.C) The heat equation $\frac{\partial u}{\partial t} - \Delta u = 0$ also fits (11.2.1.5) when the flux function is chosen appropriately. What is this so-called **diffusive flux**? \triangle

11.2.2 Characteristics

In this section we will come across a surprising ostensible solution formula for non-linear scalar conservation laws in one spatial dimension. Yet, at second glance, we will see that this formula has problem. Its breakdown will teach us that **discontinuous solutions** are meaningful and very common in the case of conservation laws.

We consider Cauchy problem ($\Omega = \mathbb{R}$) for one-dimensional scalar conservation law (11.2.1.6):

$$\Rightarrow \begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) &= 0 && \text{in } \mathbb{R} \times]0, T[, \\ u(x, 0) &= u_0(x) && \text{in } \mathbb{R}. \end{aligned} \quad (11.2.2.1)$$

$$\begin{aligned} \text{by chain rule: (11.2.2.1)} &\Leftrightarrow \frac{\partial u}{\partial t} + f'(u) \frac{\partial u}{\partial x} = 0, \\ \text{relate with linear advection (11.1.1.11)} &\quad \frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = 0. \end{aligned}$$

\gg The derivative $f'(u)$ plays the role of a u -dependent velocity of transport.

If this dependence was not there, the formula (11.1.1.12) would give us the solution. Now we will see how this formula can be generalized.

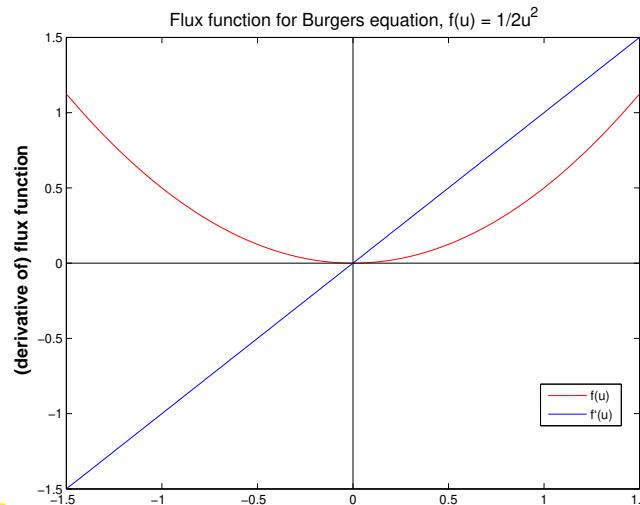
Assumption 11.2.2.2. Monotonicity of

The flux function $f : \mathbb{R} \mapsto \mathbb{R}$ is smooth ($f \in C^2$), and **convex** or **concave** [Str09, Def. 5.5.2].

Recall [Str09, Thm. 5.5.2]:

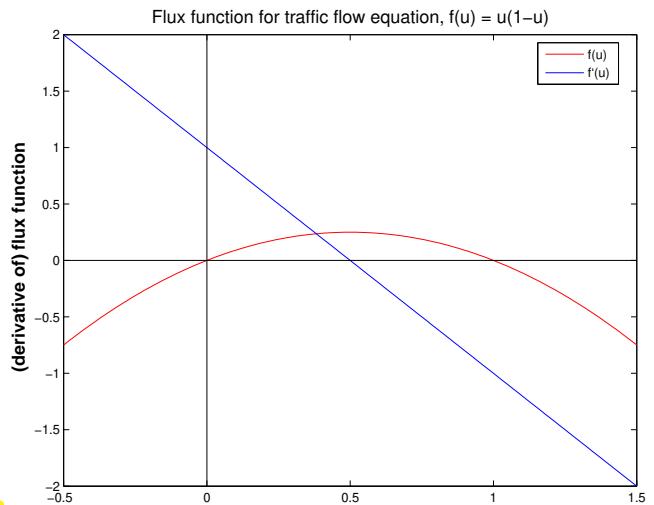
$$\begin{aligned} f \text{ convex} &\Rightarrow \text{derivative } f' \text{ increasing} \\ f \text{ concave} &\Rightarrow \text{derivative } f' \text{ decreasing} \end{aligned}$$

flux function for Burgers' equation (11.1.3.4)



f convex

flux function for traffic flow equation (11.1.2.23)



f concave

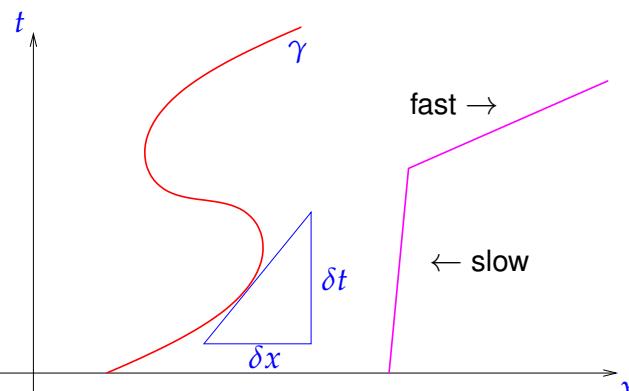
Burgers' equation (11.1.3.4) and the traffic flow equation (11.1.2.23) will serve as main examples for scalar conservation laws in one spatial dimension. The opposite curvatures of their flux functions will be reflected by a "mirror symmetric" behavior of their solutions in many cases. Below most examples will be discussed for both model problems in order to elucidate these differences, but the reader may focus on only one model problem.

Definition 11.2.2.3. Characteristic curve for one-dimensional scalar conservation law

A curve $\Gamma := (\gamma(\tau), \tau) : [0, T] \mapsto \mathbb{R} \times]0, T[$ in the (x, t) -plane is a **characteristic curve** for the conservation law (11.2.2.1), if

$$\frac{d}{d\tau} \gamma(\tau) = f'(u(\gamma(\tau), \tau)), \quad 0 \leq \tau \leq T, \quad (11.2.2.4)$$

where u is a continuously differentiable solution of (11.2.2.1).



▷ curves in an $x - t$ -diagram, described by a function $x = \gamma(t)$
 ⇔ movement of a point on the real axis.
 ▷ $x - t$ -diagram

$$\frac{d}{d\tau} \gamma(\tau) = \text{speed of interface } \gamma.$$

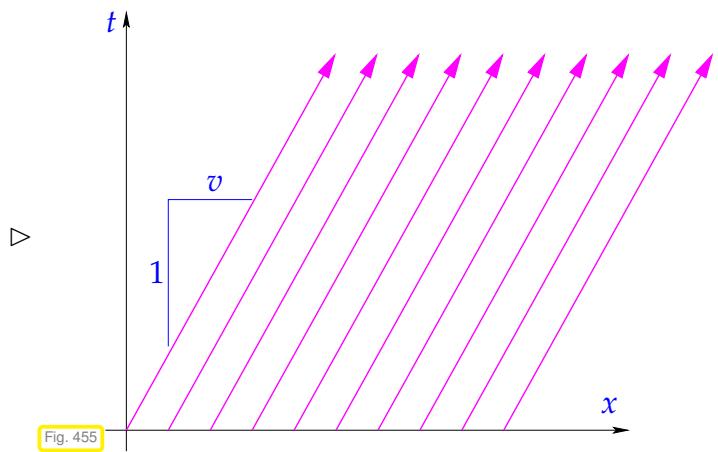
EXAMPLE 11.2.2.5 (Characteristics for advection)

Constant linear advection (11.1.1.11): $f(u) = vu$

► characteristics $\gamma(\tau) = v\tau + c, c \in \mathbb{R}$.

solution (11.1.1.12) $u(x, t) = u_0(x - vt)$

meaningful for any u_0 ! (cf. Section 10.3.2)



Ex. 11.2.2.5 reveals a close relationship between streamlines (\rightarrow Section 10.1.1) and characteristic curves. That the latter are a true generalization of the former is also reflected by the following simple observation, which generalizes the considerations in Section 10.3.2, (10.3.2.2).

Lemma 11.2.2.6. Classical solutions and characteristic curves

Smooth solutions of (11.2.2.1) are constant along characteristic curves.

Proof. Apply the chain rule twice, cf. (10.3.2.2),

(I) to $\tau \mapsto (u \circ \Psi)(\tau)$ with $u : \mathbb{R}^2 \rightarrow \mathbb{R}$, $u = u(x, t)$, $\Psi : \mathbb{R} \rightarrow \mathbb{R}^2$, $\Psi(\tau) := \begin{bmatrix} \gamma(\tau) \\ \tau \end{bmatrix}$:

$$\frac{d}{d\tau}(u \circ \Psi)(\tau) = Du(\Psi(\tau)) \frac{d\Psi}{d\tau}(\tau) = \left[\frac{\partial u}{\partial x}(\Psi(\tau)) \frac{\partial \Psi}{\partial \tau}(\tau) \right] \left[\frac{d\gamma}{d\tau}(\tau) \ 1 \right].$$

(II) to $x \mapsto (f \circ u)(x, y)$ with $f : \mathbb{R} \rightarrow \mathbb{R}$, $f = f(u)$, and $u : \mathbb{R}^2 \rightarrow \mathbb{R}$, $u = u(x, t)$:

$$\frac{\partial}{\partial x} \{x \mapsto f(u(x, t))\} = f'(u(x, t)) \frac{\partial u}{\partial x}(x, t).$$

In between we use the defining equation (11.2.2.4) for a characteristic curve:

$$\begin{aligned} \frac{d}{d\tau} u(\gamma(\tau), \tau) &\stackrel{\text{chain rule (I)}}{=} \frac{\partial u}{\partial x}(\gamma(\tau), \tau) \frac{d}{d\tau} \gamma(\tau) + \frac{\partial u}{\partial t}(\gamma(\tau), \tau) \\ &\stackrel{(11.2.2.4)}{=} \frac{\partial u}{\partial x}(\gamma(\tau), \tau) \cdot f'(u(\gamma(\tau), \tau)) + \frac{\partial u}{\partial t}(\gamma(\tau), \tau) \\ &\stackrel{\text{chain rule (II)}}{=} \left(\frac{\partial}{\partial x} f(u) \right)(\gamma(\tau), \tau) + \frac{\partial u}{\partial t}(\gamma(\tau), \tau) = 0. \end{aligned}$$

□

notation: $f' \triangleq$ derivative of flux function $f : U \subset \mathbb{R} \mapsto \mathbb{R}$

So, u is constant on a characteristic curve γ .

► $t \mapsto f'(u(\gamma(t), t))$ is constant on a characteristic curve γ

(11.2.2.4) \Rightarrow slope of characteristic curve is constant!

► Characteristic curve through $(x_0, 0)$ is a straight line $(x_0 + f'(u_0(x_0))\tau, \tau)$, $0 \leq \tau \leq T$!

Doesn't this provide an implicit solution formula for (11.2.2.1) in particular since f' is monotone?

$$u(x, t) = u_0(x - f'(u_0(x_0))t). \quad (11.2.2.7)$$

This is obviously a non-linear equation for $u(x, t)$. Will it have a solution for all (x, t) ?

§11.2.2.8 (Breakdown of characteristic solution formula) The key problem of formula (11.2.2.7) is that it may have multiple solutions:

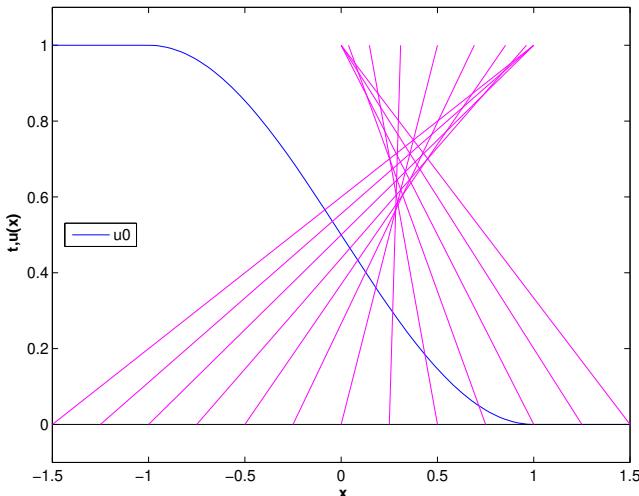


Fig. 456

For Burger's equation (11.1.3.4):
 $(f(u) = \frac{1}{2}u^2)$ smooth and strictly convex

▷ $f'(u) = u$ (increasing)

◁ if u_0 smooth and decreasing.

➤ characteristic curves intersect !

➤ solution formula (11.2.2.7) becomes invalid

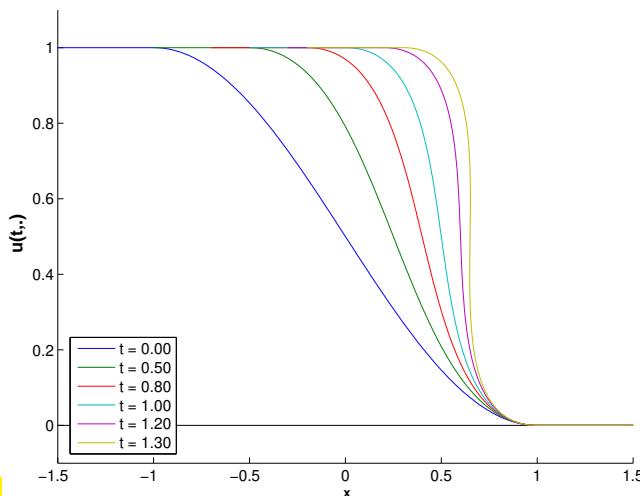


Fig. 457

$t < 1.3$: solution by (11.2.2.7)

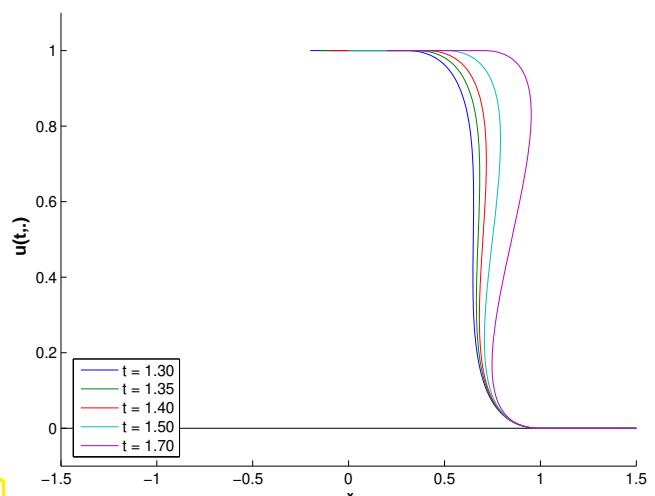


Fig. 458

The wave breaks: "multivalued solution"

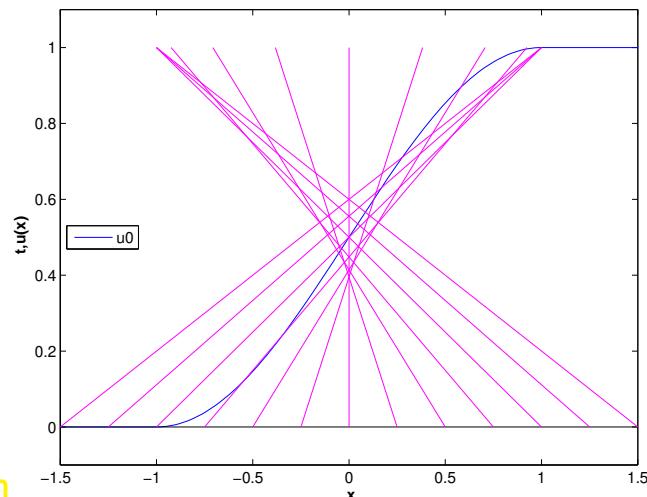


Fig. 459

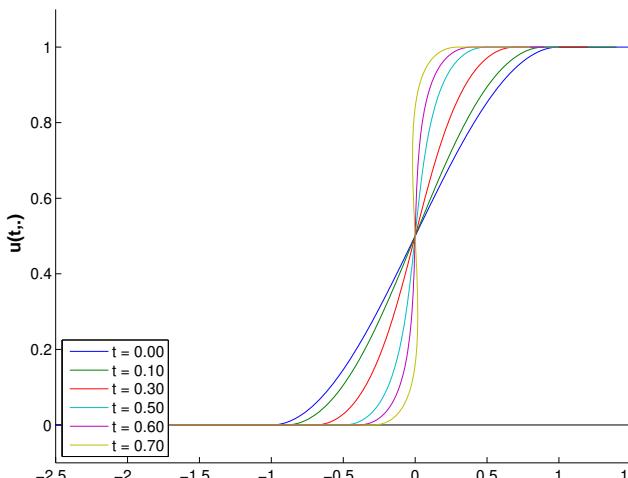
For traffic flow equation (11.1.2.23):
 $(f(u) = u(1-u))$ smooth and strictly concave

▷ $f'(u) = 1 - 2u$ (decreasing)

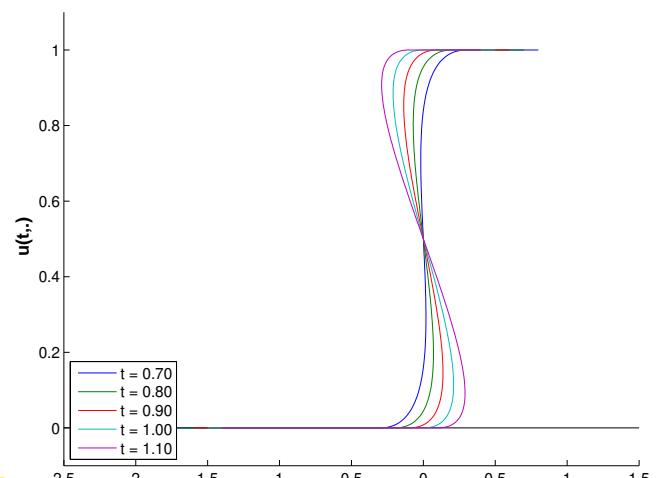
◁ if u_0 smooth and increasing.

➤ characteristic curves intersect !

➤ solution formula (11.2.2.7) becomes invalid



$t < 0.7$: solution by (11.2.2.7)



the wave breaks: “multivalued solution”

► What we have observed is a breakdown of classical solutions after some time. Also recall Ex. 11.2.2.5 and Rem. 11.1.1.13 which suggest that even discontinuous solutions can make perfect sense for scalar conservation laws. We conclude that we need a new “*weak*” concept of solutions of (11.2.2.1), similar to the situation for second-order elliptic boundary value problems, for which physically meaningful solutions with kinks are valid weak solutions, cf. Section 1.2.1.3.

Remark 11.2.2.9 (Meaning of characteristics) Concerning the interpretation of characteristics in the case of the traffic flow model (11.1.2.23) with $f(u) = u(1-u)$ we find

$$\begin{aligned} \text{Equation for characteristics} \quad & \dot{\gamma}(t) = -2u(\gamma(t), t) + 1, \\ \text{Equation for car trajectories} \quad & \dot{x}(t) = 1 - u(x(t), t). \end{aligned}$$

Hence, characteristics do not give the paths of cars; cars always drive to the right, while characteristics may be slanted to the left!

Yet, Lemma 11.2.2.6 tells us that for a smooth solution of a non-linear scalar conservation law, the characteristic running through $(x^*, t^*) \in \mathbb{R} \times]0, T[$ gives the locus of space-time points $(x, t) \in \mathbb{R} \times]0, T[$, on which the solution value $u(x^*, t^*)$ depends (for $t < t^*$) or on which it will have an influence (for $t > t^*$).

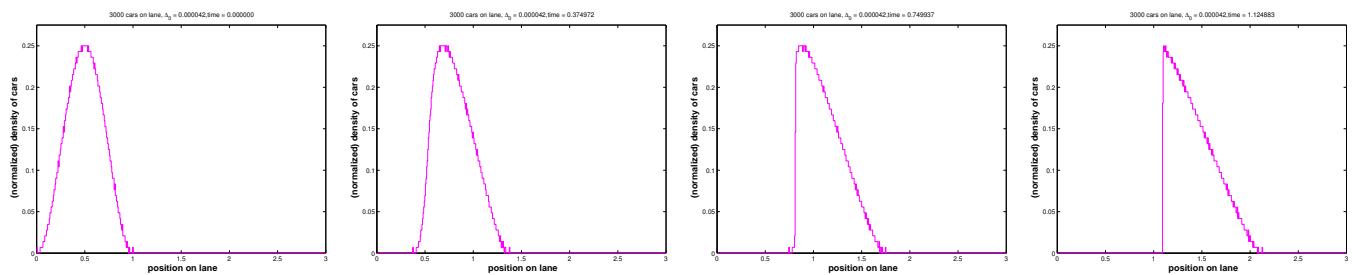
For a scalar conservation law information “flows” along characteristic curves.

EXAMPLE 11.2.2.10 (Traffic flow: Evolution of smooth initial density) For the traffic flow model we should always expect a unique car density for all times. Thus, in order to see the consequences of the breakdown of the solution formula, we return to the particle model for single lane traffic flow from Section 11.1.2.1. For a large number of cars it should give us a hint how the density will be affected by the intersection of characteristics.

We approximately solve the particle model, that is the evolution according to ODE (11.1.2.5), (11.1.2.6). for $N = 3000$ cars.

The initial car positions derived from a smooth car density u_0

$$x_i(0) = \Phi^{-1}\left(\frac{i-1}{N-1}\right), \quad i = 1, \dots, N, \quad \Phi(\xi) = \int_0^\xi u_0(x) dx, \quad u_0(x) := 2 \sin^2(\pi x).$$



After some time a *discontinuity* in the density of cars crops up (“breaking wave”, see Fig. 462). This suggests that the emergence of discontinuities despite smooth initial data is an intrinsic feature of the traffic flow model, which reflects “physical reality”. \square

Review question(s) 11.2.2.11 (Characteristics)

(Q11.2.2.11.A) In the $x - t$ -plane sketch the characteristics for the Cauchy problem for the scalar conservation law

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) &= 0 && \text{in } \mathbb{R} \times]0, T[, \\ u(x, 0) &= u_0(x) && \text{in } \mathbb{R} . \end{aligned} \quad (11.2.2.1)$$

with flux function $f(u) = u^2$ and initial data

$$u_0(x) = \begin{cases} 1+x & \text{for } -1 < x \leq 0 , \\ 1-x & \text{for } 0 < x \leq 1 , \\ 0 & \text{elsewhere.} \end{cases}$$

Definition 11.2.2.3. Characteristic curve for one-dimensional scalar conservation law

A curve $\Gamma := (\gamma(\tau), \tau) : [0, T] \mapsto \mathbb{R} \times]0, T[$ in the (x, t) -plane is a **characteristic curve** for the conservation law (11.2.2.1), if

$$\frac{d}{d\tau} \gamma(\tau) = f'(u(\gamma(\tau), \tau)) , \quad 0 \leq \tau \leq T , \quad (11.2.2.4)$$

where u is a continuously differentiable solution of (11.2.2.1).

(Q11.2.2.11.B) For a scalar 1D conservation law with flux functions

1. $f(u) = u^2$,
2. $f(u) = \sin(\pi u)$,
3. $f(u) = \cos(\pi u)$

and initial data $u_0(x) = 1$ for $-1 \leq x \leq 1$, $u_0(x) = 0$ elsewhere, sketch the family of **characteristic curves** in an $x - t$ diagram.

(Q11.2.2.11.C) Directly verify that, if well-defined, the implicitly defined function

$$u(x, t) = u_0(x - f'(u(x, t))t) , \quad (x, t) \in \mathbb{R} \times [0, T] , \quad (11.2.2.7)$$

with a strictly convex and smooth flux function $f : \mathbb{R} \rightarrow \mathbb{R}$ provides a solution of (11.2.2.1). \triangle

11.2.3 Weak Solutions

Of course, discontinuous solutions of (11.2.2.1) cannot be solutions in the sense of classical calculus. Yet, the fact that physically meaningful solutions fail to meet the smoothness requirements for classical solutions is familiar to us: we saw this already for the elastic string model, where we had to admit solutions with a kink in Ex. 1.2.1.26. This forced us to develop weak concepts of solutions. For the elastic string models these were solutions of the associated variational equation. In the case of conservation laws a similar concept of weak solutions will turn out to capture all physically meaningful solutions.

The integral form of a conservation law that we have already seen in (11.2.1.4) points the way. Let us link it with Gauss' theorem.

Theorem 1.5.2.4. Gauss' theorem

With $\mathbf{n} : \partial\Omega \mapsto \mathbb{R}^d$ denoting the **exterior unit normal vectorfield** on $\partial\Omega$ and dS indicating integration over a surface, we have

$$\int_{\Omega} \operatorname{div} \mathbf{j}(x) dx = \int_{\partial\Omega} \mathbf{j}(x) \cdot \mathbf{n}(x) dS(x) \quad \forall \mathbf{j} \in (C_{pw}^1(\bar{\Omega}))^d. \quad (1.5.2.5)$$

“Space-time Gauss theorem”

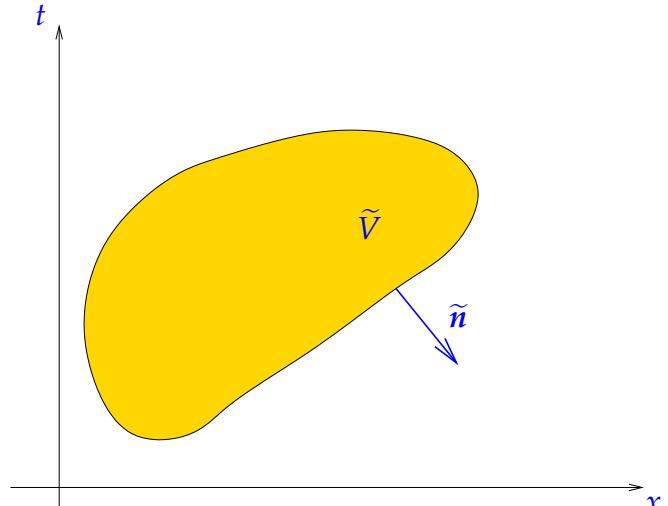
$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \quad (11.2.3.1)$$

\Updownarrow

$$\operatorname{div}_{(x,t)} \left[\begin{bmatrix} f(u) \\ u \end{bmatrix} \right] = 0 \quad \text{in } \tilde{\Omega}. \quad (11.2.3.2)$$

► ∀ “space-time control volumes” $\tilde{V} \subset \tilde{\Omega}$:

$$\int_{\partial\tilde{V}} \left[\begin{bmatrix} f(u(\tilde{x})) \\ u(\tilde{x}) \end{bmatrix} \right] \cdot \left[\begin{bmatrix} n_x(\tilde{x}) \\ n_t(\tilde{x}) \end{bmatrix} \right] dS(\tilde{x}) = 0,$$

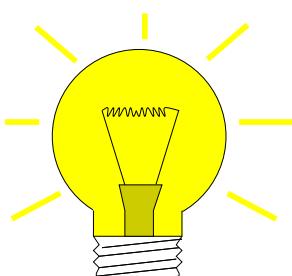


$\tilde{n} := (n_x, n_t)^T \hat{=} \text{space-time unit normal}$

(11.2.3.2) for space-time rectangle $\tilde{V} = [x_0, x_1] \times [t_0, t_1]$ ► integral form of (11.2.3.1), cf. (11.2.1.4):

$$\int_{x_0}^{x_1} u(x, t_1) dx - \int_{x_0}^{x_1} u(x, t_0) dx = \int_{t_0}^{t_1} f(u(x_0, t)) dt - \int_{t_0}^{t_1} f(u(x_1, t)) dt. \quad (11.2.3.3)$$

Still, (11.2.3.3) encounters problems, if a discontinuity of u coincides with an edge of the space-time rectangle.



The idea is similar to that behind the derivation of the weak form for 2nd-order elliptic BVPs in Section 1.8. For the Cauchy problem

- I: test the conservation law PDE with a smooth function,
- II: integrate by parts one in space & time,
- III: take into account the initial conditions.

STEP I: Test (11.2.3.2) with compactly supported smooth function $\Phi : \tilde{\Omega} \mapsto \mathbb{R}$, $\Phi(\cdot, T) = 0$, and integrate over space-time cylinder $\tilde{\Omega} = \mathbb{R} \times [0, T]$:

$$(11.2.3.2) \quad \Rightarrow \quad \int_{\tilde{\Omega}} \operatorname{div}_{(x,t)} \left[\frac{f(u)}{u} \right] \Phi(x, t) dx dt = 0 .$$

STEP II: Perform integration by parts using Green's first formula Thm. 1.5.2.7 on $\tilde{\Omega}$:

Theorem 1.5.2.7. Green's first formula

For all vector fields $\mathbf{j} \in (C_{\text{pw}}^1(\overline{\Omega}))^d$ and functions $v \in C_{\text{pw}}^1(\overline{\Omega})$ holds

$$\int_{\Omega} \mathbf{j} \cdot \operatorname{grad} v dx = - \int_{\Omega} \operatorname{div} \mathbf{j} v dx + \int_{\partial\Omega} \mathbf{j} \cdot \mathbf{n} v dS . \quad (1.5.2.8)$$

$$\begin{aligned} \int_{\tilde{\Omega}} \operatorname{div}_{(x,t)} \left[\frac{f(u)}{u} \right] \Phi(x, t) dx dt &= 0 \\ \stackrel{\text{Thm. 1.5.2.7}}{\Rightarrow} \quad \int_{\tilde{\Omega}} \left[\frac{f(u)}{u} \right] \cdot \operatorname{grad}_{(x,t)} \Phi dx dt + \int_{-\infty}^{\infty} u(x, 0) \Phi(x, 0) dx &= 0 , \end{aligned}$$

because $\partial\tilde{\Omega} = \mathbb{R} \times \{0\} \cup \mathbb{R} \times \{T\}$ with “normals” $\mathbf{n} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$ ($t = 0$ boundary) and $\mathbf{n} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ ($t = T$ boundary), which has to be taken into account in the boundary term in Green's formula. The “ $t = T$ boundary part” does not enter as $\Phi(\cdot, T) = 0$.

Note that $u(x, 0)$ is fixed by the initial condition: $u(x, 0) = u_0(x)$.

Definition 11.2.3.4. Weak solution of Cauchy problem for scalar conservation law

For $u_0 \in L^\infty(\mathbb{R})$, $u : \mathbb{R} \times]0, T[\mapsto \mathbb{R}$ is a **weak solution** of the Cauchy problem (11.2.2.1), if

$$u \in L^\infty(\mathbb{R} \times]0, T[) \quad \wedge \quad \int_{-\infty}^{\infty} \int_0^T \left\{ u \frac{\partial \Phi}{\partial t} + f(u) \frac{\partial \Phi}{\partial x} \right\} dt dx + \int_{-\infty}^{\infty} u_0(x) \Phi(x, 0) dx = 0 ,$$

for all $\Phi \in C_0^\infty(\mathbb{R} \times [0, T[)$, $\Phi(\cdot, T) = 0$.

Remark 11.2.3.5 (Properties of weak solutions) By reversing integration by parts, it is easy to see that

$$u \text{ weak solution of (11.2.2.1) \&} u \in C^1 \iff u \text{ classical solution of (11.2.2.1).}$$

Arguments from mathematical integration theory confirm

$$u \in L_{\text{loc}}^\infty(\mathbb{R} \times]0, T[) \text{ weak solution of (11.2.2.1)} \Rightarrow \begin{array}{l} u \text{ satisfies integral form (11.2.3.3)} \\ \text{for “almost all” } x_0 < x_1, 0 < t_0 < t_1 < T . \end{array}$$

11.2.4 Jump Conditions

Now we want to explore the discontinuities compatible with our concept of a weak solution from Def. 11.2.3.4.

For piecewise smooth divergence-free vectorfield $\mathbf{j} : \Omega \subset \mathbb{R}^2$ we find by a “pillbox thought experiment”:

$$\text{“div } \mathbf{j} = 0”$$

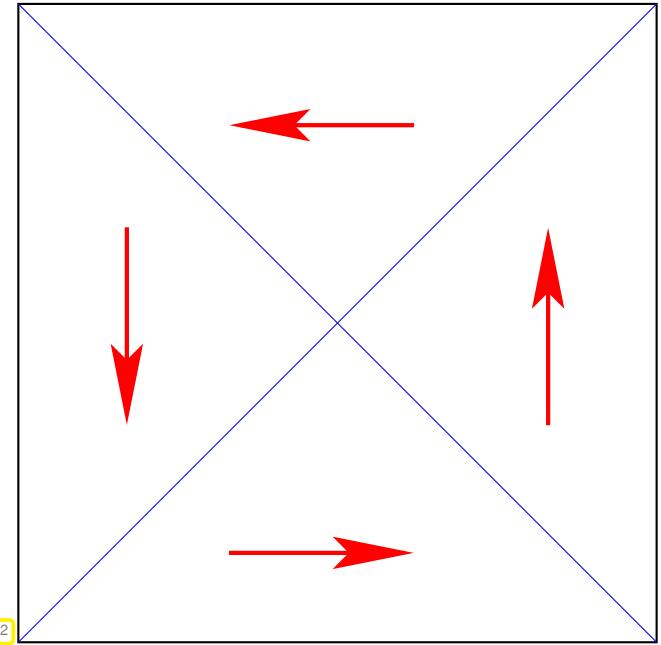
\Updownarrow

$$\int_{\partial V} \mathbf{j} \cdot \mathbf{n} dS = 0 \quad \forall \text{ control volumes } V \subset \Omega$$

Necessary condition:

Continuity of **normal components**
across discontinuities

discontinuous divergence-free vectorfield \Rightarrow



To see this, consider a slender tiny rectangle aligned with a line of discontinuity of \mathbf{j} . In the absence of normal continuity a net flux through its boundary will result, provided that the rectangle is small enough (“pillbox argument”).

Apply this insight to vectorfield on space-time domain $\tilde{\Omega} = \mathbb{R} \times]0, T[$:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \Leftrightarrow \text{div}_{(x,t)} \underbrace{\begin{bmatrix} f(u) \\ u \end{bmatrix}}_{=: \mathbf{j}} = 0 \quad \text{in } \tilde{\Omega}. \quad (11.2.3.2)$$

Normal at C^1 -curve $\Gamma := \tau \mapsto (\gamma(\tau), \tau)$ in $(\gamma(\tau), \tau)$

$$\tilde{\mathbf{n}} = \frac{1}{\sqrt{1 + |\dot{s}|^2}} \begin{bmatrix} 1 \\ -\dot{s} \end{bmatrix}, \quad \dot{s} := \frac{d\gamma}{d\tau} \quad \text{“speed of curve”}.$$

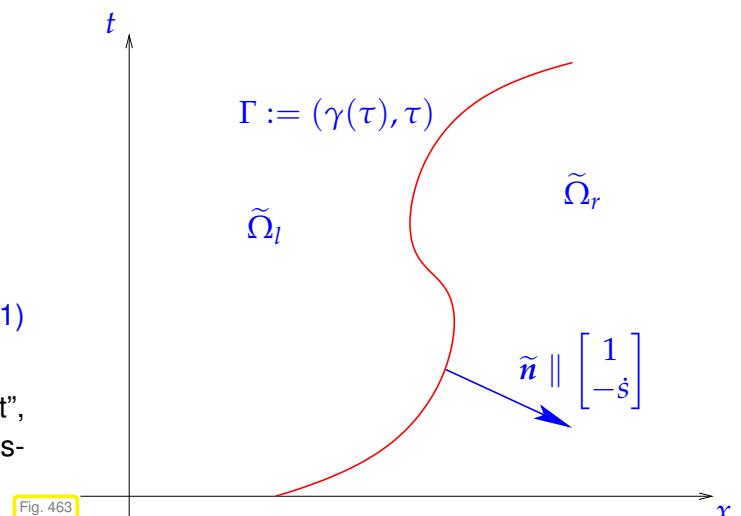
To see this, recall that the normal is orthogonal to the tangent vector $(\dot{s})^\top$ and that in 2D the direction orthogonal to (x_1) is given by $(-x_2)$.

“normal continuity” of
piecewise smooth vectorfield $(f(u), u)^T$

\Updownarrow

$$\left[\begin{array}{c} 1 \\ -\frac{d\gamma}{d\tau} \end{array} \right] \cdot \left[\begin{array}{c} \llbracket f(u) \rrbracket_\Gamma \\ \llbracket u \rrbracket_\Gamma \end{array} \right] = 0, \quad (11.2.4.1)$$

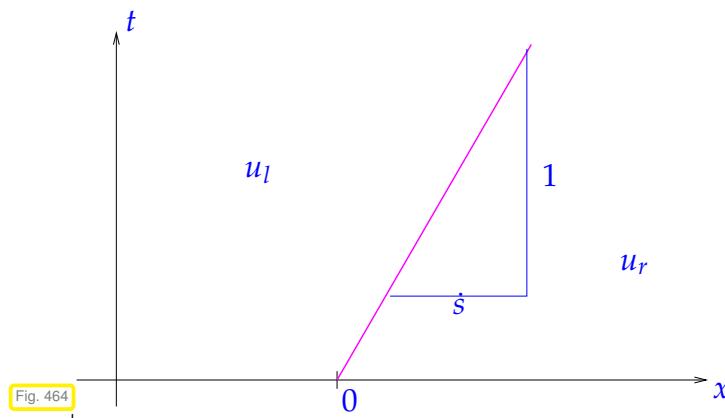
where $\llbracket \cdot \rrbracket_\Gamma \hat{=} \text{jump across } \Gamma$ (“from left to right”),
e.g. $\llbracket u \rrbracket_\Gamma = u_l - u_r$, where subscripts ‘l’ and ‘r’ designates values in $\tilde{\Omega}_l$, $\tilde{\Omega}_r$.



Terminology: (11.2.4.1) = Rankine-Hugoniot (jump) condition, shorthand notation:

$$\dot{s}(u_l - u_r) = f(u_l) - f(u_r) \quad , \quad \dot{s} := \frac{d\gamma}{d\tau} \quad \text{"propagation speed of discontinuity"} \quad (11.2.4.2)$$

§11.2.4.3 (Discontinuity connecting constant states) The simplest situation compliant with Rankine-Hugoniot jump condition: *constant states* to the left and right of the curve of discontinuity (11.2.4.1):



$$u(x, t) = \begin{cases} u_l \in \mathbb{R} & , \text{ for } x < \dot{s}t, \\ u_r \in \mathbb{R} & , \text{ for } x < \dot{s}t, \end{cases} \quad (11.2.4.4)$$

with **constant** speed \dot{s} of discontinuity, according to (11.2.4.2) given by (for $u_l \neq u_r$)

$$\dot{s} = \frac{f(u_l) - f(u_r)}{u_l - u_r}.$$

11.2.5 Riemann Problem

The situation of locally constant states discussed in § 11.2.4.3 is particularly easy.

► Consider: Cauchy-problem (11.2.2.1) for piecewise constant initial data u_0 .

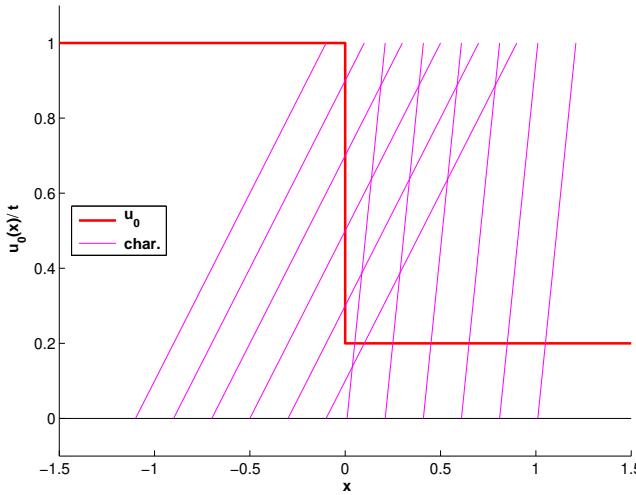
Definition 11.2.5.1. Riemann problem

$$u_0(x) = \begin{cases} u_l \in \mathbb{R} & , \text{ if } x < 0, \\ u_r \in \mathbb{R} & , \text{ if } x > 0. \end{cases} \quad \hat{=} \quad \text{Riemann problem for (11.2.2.1)}$$

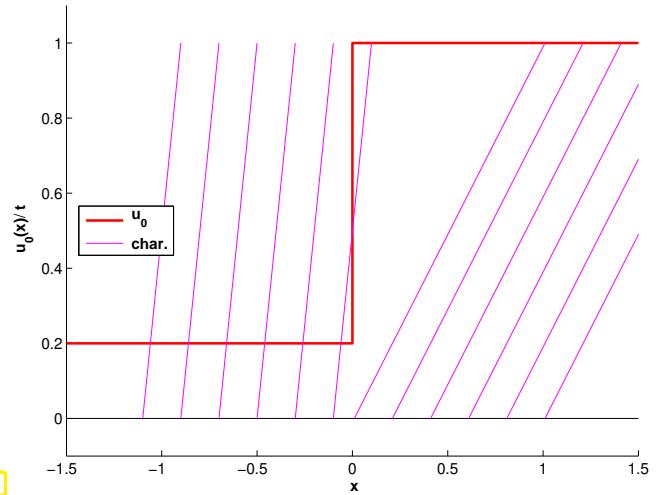
Setting, cf. Section 11.2.2:

flux function $f : \mathbb{R} \mapsto \mathbb{R}$ smooth & convex

► f' non-decreasing ► pattern of characteristic curves for Riemann problem:



$u_l > u_r$: intersecting characteristics



$u_l < u_r$: diverging characteristics

Setting, cf. Section 11.2.2:

flux function $f : \mathbb{R} \mapsto \mathbb{R}$ smooth & concave

► f' non-increasing ► pattern of characteristic curves for Riemann problem:

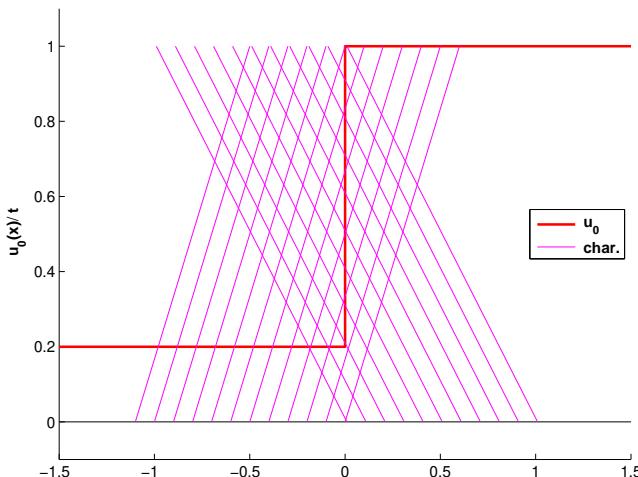


Fig. 467

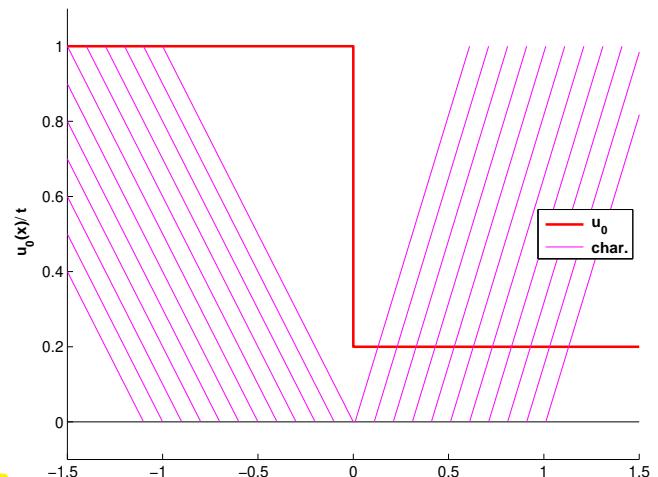


Fig. 468

$u_l < u_r$: intersecting characteristics

$u_l > u_r$: diverging characteristics

Definition 11.2.5.2. Shock

If Γ is a smooth curve in the (x, t) -plane and u a weak solution of (11.2.2.1), a discontinuity of u across Γ is called a **shock**.

By § 11.2.4.3 ► the **shock speed s** is given by the Rankine-Hugoniot jump conditions:

$$(x_0, t_0) \in \Gamma: \quad \dot{s} = \frac{f(u_l) - f(u_r)}{u_l - u_r}, \quad u_l := \lim_{\epsilon \rightarrow 0} u(x_0 - \epsilon, t_0), \quad u_r := \lim_{\epsilon \rightarrow 0} u(x_0 + \epsilon, t_0). \quad (11.2.5.3)$$

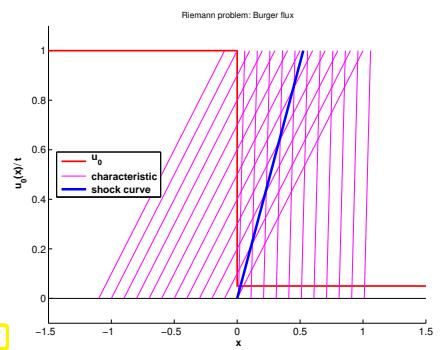
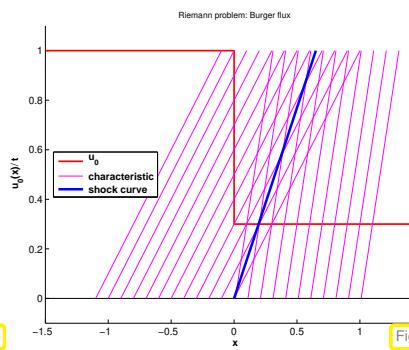
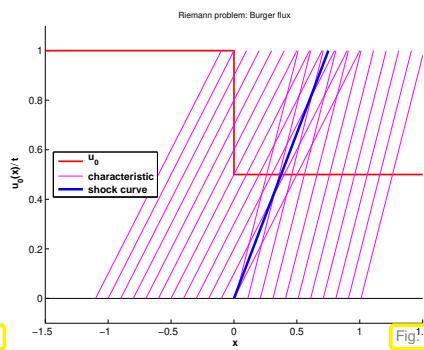
Lemma 11.2.5.4. Shock solution of Riemann problem

For any two states $u_l, u_r \in \mathbb{R}$ the piecewise constant function

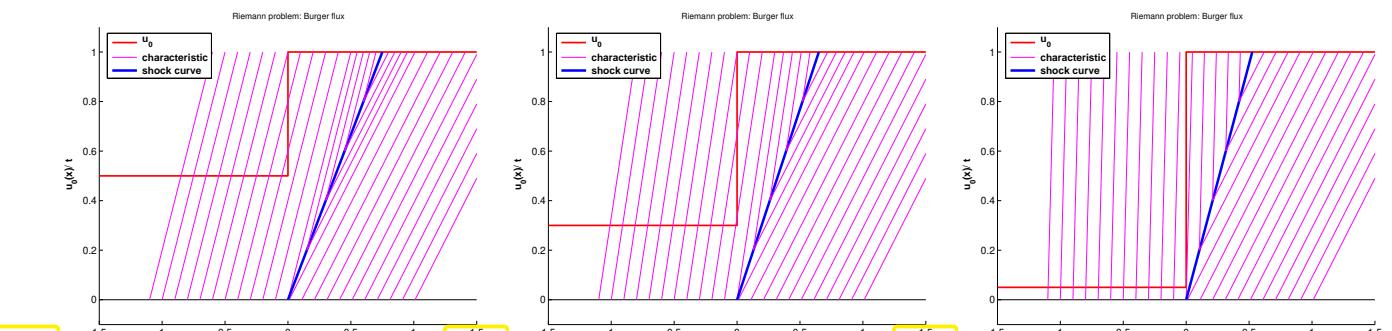
$$u(x, t) := \begin{cases} u_l & \text{for } x < \dot{s}t, \\ u_r & \text{for } x > \dot{s}t, \end{cases} \quad \dot{s} := \frac{f(u_l) - f(u_r)}{u_l - u_r}, \quad x \in \mathbb{R}, 0 < t < T,$$

is a **weak solution** (\rightarrow Def. 11.2.3.4) of the related Riemann problem (\rightarrow Section 11.2.5) for the 1D scalar conservation law (11.2.2.1).

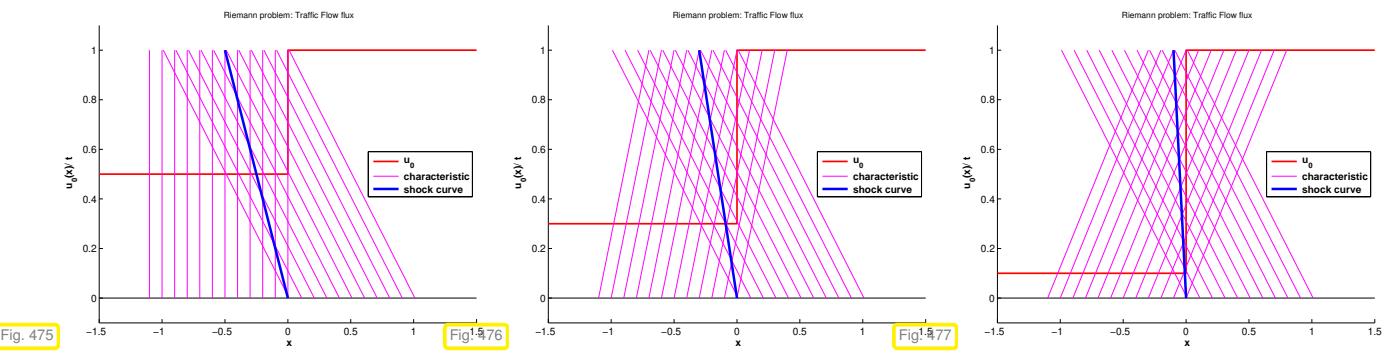
Now we study the dependence of shock solutions on the initial states u_l and u_r . We take a close look at the connection between characteristics and shocks. In the following $x - t$ diagrams, shocks are marked with —, characteristics with —, and u_0 is indicated by —.



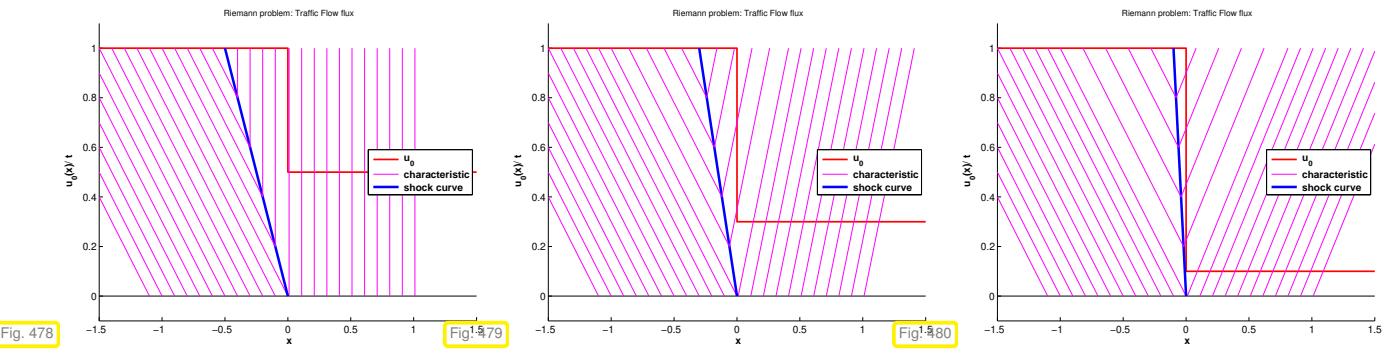
Burgers flux $f(u) = \frac{1}{2}u^2$, $u_l > u_r$: characteristic curves impinge on shock



Burgers flux $f(u) = \frac{1}{2}u^2$, $u_l < u_r$: characteristic curves emanate from shock
(expansion shock)



Traffic Flow flux $f(u) = u(1-u)$, $u_l < u_r$: characteristic curves impinge on shock



Traffic flow flux $f(u) = u(1-u)$, $u_l > u_r$: characteristic curves emanate from shock
(expansion shock)

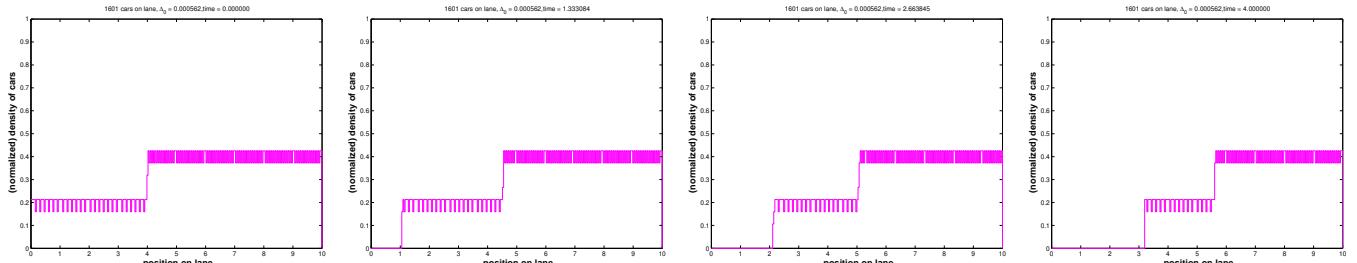
EXAMPLE 11.2.5.5 (Actual shock patterns in traffic flow) In order to tell the physical relevance of shock solutions for the car density we try to obtain them approximately from the particle model of traffic flow using many cars.

We conduct a simulation of microscopic particle model of traffic flow as in Exp. 11.1.2.13, with initial car distribution

$x0 = [(0:0.01:4), (4.005:0.005:10)]$ (MATLAB syntax),

$\Delta_0 = 0.002$, normalized car density by averaging.

Situation: column of fast going cars approaches a zone of dense traffic.



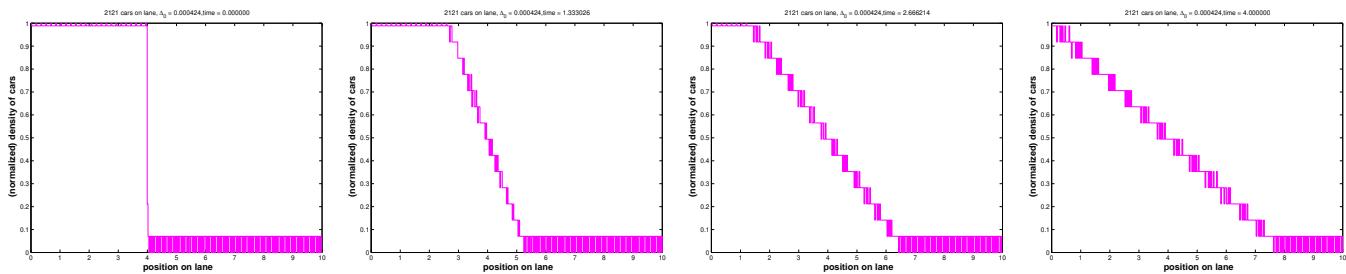
Observation: abrupt changes of car density (= **shocks**) present in initial conditions persist throughout the evolution. Sites of discontinuity travel with constant speed close to the speed predicted by the jump conditions (11.2.4.2).

EXAMPLE 11.2.5.6 (Fan patterns in traffic flow) Simulation of microscopic particle model of traffic flow as in Exp. 11.1.2.13, initial car distribution

$x0 = [(0:0.002:4), (4.05:0.05:10)]$ (MATLAB syntax),

$\Delta_0 = 0.002$, normalized car density by averaging.

Situation: front end of a traffic jam



Observation: abrupt changes of car density present in initial conditions disappear and are replaced with a zone of **linearly decreasing** car density, whose edges move with constant speed in opposite direction.

No shock solution!

EXAMPLE 11.2.5.7 (Vanishing viscosity for Burgers equation) Recall the modeling approach explained in Section 11.1.3. There is no such material as an “inviscid” fluid in nature, because in any physical system there will be a tiny amount of friction. This leads us to the very general understanding that conservation laws can usually be regarded as limit problems $\epsilon \rightarrow 0$ for singularly perturbed transport-diffusion problems with an “ ϵ -amount” of diffusion.

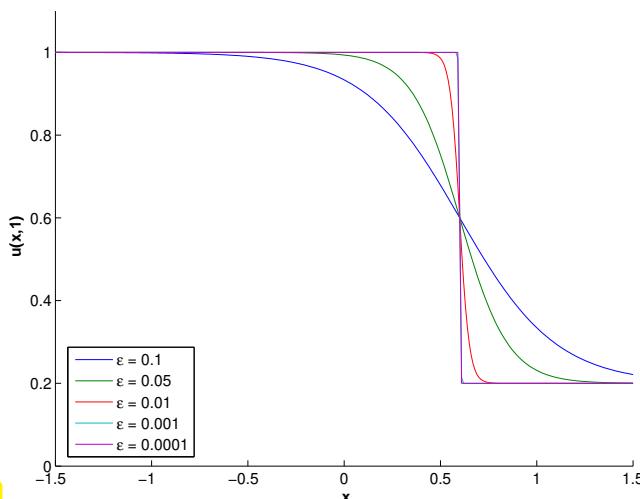
In 1D, for any $\epsilon > 0$ these transport-diffusion problems will possess a unique smooth solution. Studying its behavior for $\epsilon \rightarrow 0$ will tell us, what are “physically meaningful” solutions for the conservation law. This consideration is called the **vanishing viscosity** method to define solutions for conservation laws.

Here we pursue this idea for Burgers equation, see Section 11.1.3.

$$\text{Viscous Burgers equation: } \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} \left(\frac{1}{2} u^2 \right) = \epsilon \frac{\partial^2 u}{\partial x^2}. \quad (11.2.5.8)$$

Travelling wave solution of Riemann problem for (11.2.5.8) via Cole-Hopf transform \rightarrow [Eva98, Sect. 4.4.1]

$$u_\epsilon(x, t) = w(x - \dot{s}t) , \quad w(\xi) = u_r + \frac{1}{2}(u_l - u_r)(1 - \tanh\left(\frac{\xi(u_l - u_r)}{4\epsilon}\right)) , \quad \dot{s} = \frac{1}{2}(u_l + u_r) .$$



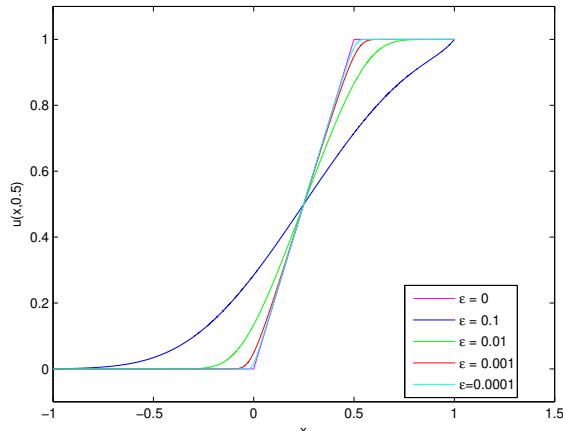
$u_\epsilon(x, t)$ = classical solution of (11.2.5.8) for all $t > 0$, $x \in \mathbb{R}$ (only for $u_l > u_r$!).

\Lsh $u_l > u_r, t = 0.5$

emerging shock for $\epsilon \rightarrow 0$

$u_\epsilon \rightarrow u$ from Lemma 11.2.5.4 in $L^\infty(\mathbb{R})$.

Highly accurate numerical solution of
Riemann problem for (11.2.5.8)
 $u_l < u_r$ $u_\epsilon(x, 0.5) \triangleright$
no shock as $\epsilon \rightarrow 0$!
 $u_\epsilon \rightarrow$ a piecewise linear function!



§11.2.5.9 (Similarity solution) Let us try to derive a (weak) solution of the homogeneous scalar conservation law (11.2.3.1) with the structure observed in Ex. 11.2.5.6 and Ex. 11.2.5.7.

Idea: conservation law (11.2.3.1) homogeneous in spatial/temporal derivatives:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \quad \text{in } \mathbb{R} \times \mathbb{R}^+ \Rightarrow \frac{\partial u_\lambda}{\partial t} + \frac{\partial}{\partial x} f(u_\lambda) = 0 \quad \text{in } \mathbb{R} \times \mathbb{R}^+,$$

where $u_\lambda(x, t) := u(\lambda x, \lambda t)$, $\lambda > 0$.

In addition, for the Riemann problem (\rightarrow Section 11.2.5) the initial condition also satisfies $u_0(\lambda x) = u_0(x)$.

This suggests that we look for solutions of the Riemann problem that are constant on all straight lines in the $x - t$ -plane that cross $(0, 0)^T$.

► try similarity solution:

$$u(x, t) = \psi(x/t)$$



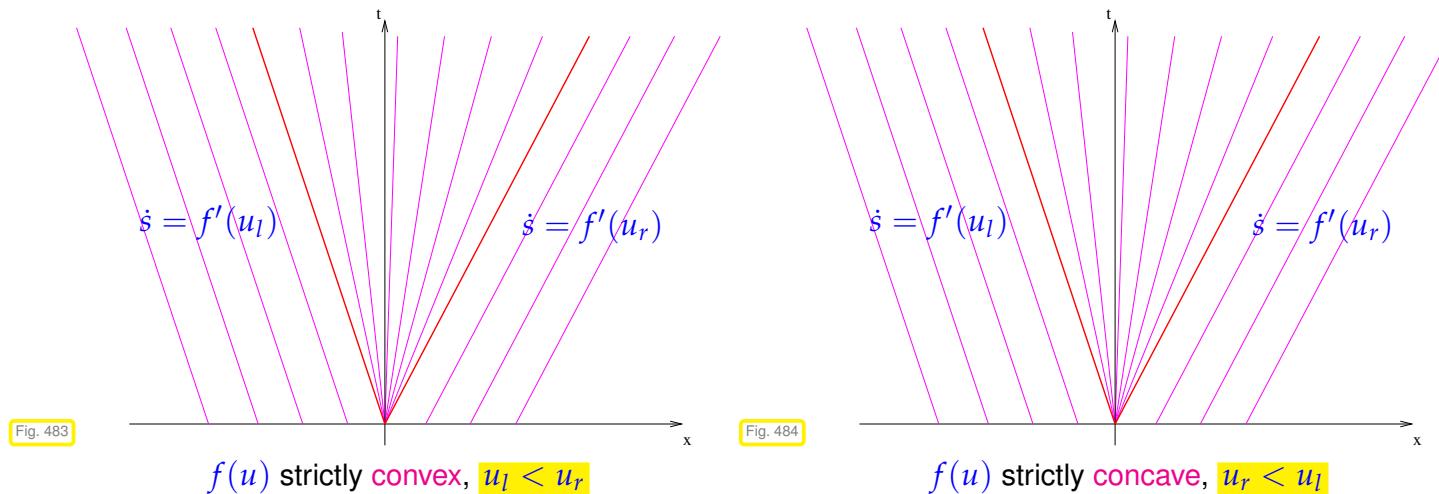
← insert in $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0$

$$f'(\psi(x/t))\psi'(x/t) = (x/t)\psi'(x/t) \quad \forall x \in \mathbb{R}, 0 < t < T.$$

$$\blacktriangleright \quad \psi' \equiv 0 \quad \vee \quad f'(\psi(w)) = w \quad \Leftrightarrow \quad \psi(w) = (f')^{-1}(w).$$

f' strictly monotone !

We can apply the formula for a similarity solution to the situation of a Riemann problem, because the initial data are compatible with it. Assuming monotonicity of the derivative of the (smooth) flux function f , we obtain the following similarity solutions:



Lemma 11.2.5.10. Rarefaction solution of Riemann problem

If $f \in C^2(\mathbb{R})$ is strictly $\begin{cases} \text{convex and } u_l < u_r, \\ \text{concave and } u_r < u_l, \end{cases}$ then

$$u(x, t) := \begin{cases} u_l & \text{for } x < \min\{f'(u_l), f'(u_r)\} \cdot t, \\ g\left(\frac{x}{t}\right) & \text{for } \min\{f'(u_l), f'(u_r)\} < \frac{x}{t} < \max\{f'(u_l), f'(u_r)\}, \\ u_r & \text{for } x > \max\{f'(u_l), f'(u_r)\} \cdot t, \end{cases}$$

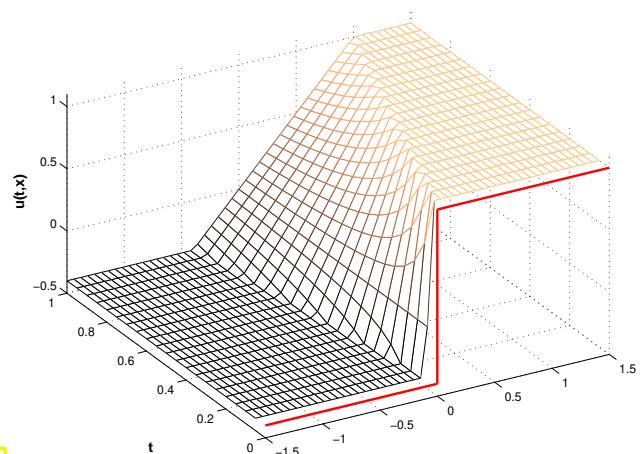
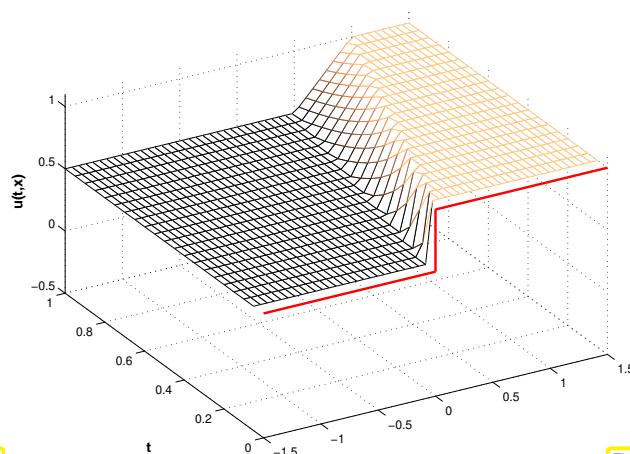
$g := (f')^{-1}$, is a continuous weak solution of the Riemann problem (\rightarrow Section 11.2.5).

Proof. We show that the rarefaction solution is a weak solution according to Def. 11.2.3.4 \Rightarrow for $\Phi \in C_0^\infty(\mathbb{R} \times]0, T[)$

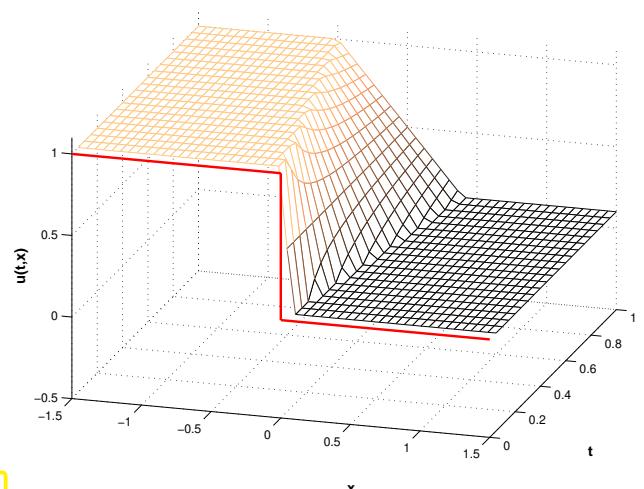
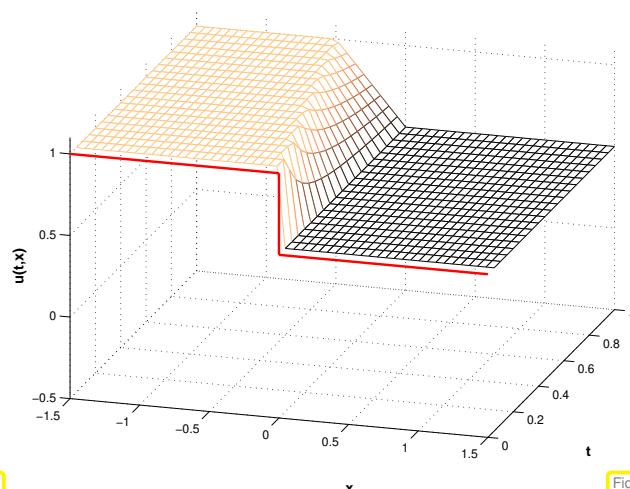
$$\begin{aligned} & \int_0^T \left\{ \int_{-\infty}^{f'(u_l)t} u_l \frac{\partial \Phi}{\partial t} + f(u_l) \frac{\partial \Phi}{\partial x} dx + \int_{f'(u_l)t}^{f'(u_r)t} g\left(\frac{x}{t}\right) \frac{\partial \Phi}{\partial t} + f(g\left(\frac{x}{t}\right)) \frac{\partial \Phi}{\partial x} dx + \int_{f'(u_r)t}^\infty u_r \frac{\partial \Phi}{\partial t} + F(u_r) \frac{\partial \Phi}{\partial x} dx \right\} dt \\ &= \int_0^T \int_{f'(u_l)t}^{f'(u_r)t} g'\left(\frac{x}{t}\right) \frac{x}{t^2} \Phi - f'(g\left(\frac{x}{t}\right)) \frac{1}{t} g'\left(\frac{x}{t}\right) \Phi dx dt = 0, \end{aligned}$$

because $(f' \circ g)(x/t) = x/t$ and by fundamental theorem of calculus. \square

Terminology: solution of Lemma 11.2.5.10 = **rarefaction wave**: *continuous solution!*



Burger flux function $f(u) = \frac{1}{2}u^2$, $u_l < u_r$: rarefaction wave solutions



Traffic flow flux function $f(u) = \frac{1}{2}u(1-u)$, $u_l > u_r$: rarefaction wave solutions

Review question(s) 11.2.5.11 (Scalar conservation laws: weak solutions and Riemann problems)

(Q11.2.5.11.A) Given a continuous, *compactly supported* function $u_0 \in C_0^0(\mathbb{R})$ show that $u(x,t) := u_0(x - vt)$ is a weak solution of the Cauchy problem ($T > 0$)

$$\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = 0 \quad \text{in } \mathbb{R} \times [0, T] , \quad u(x, 0) = u_0(x) , \quad x \in \mathbb{R} .$$

Definition 11.2.3.4. Weak solution of Cauchy problem for scalar conservation law

For $u_0 \in L^\infty(\mathbb{R})$, $u : \mathbb{R} \times]0, T[\mapsto \mathbb{R}$ is a **weak solution** of the Cauchy problem

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) &= 0 && \text{in } \mathbb{R} \times]0, T[, \\ u(x, 0) &= u_0(x) && \text{in } \mathbb{R} , \end{aligned} \tag{11.2.2.1}$$

if

$$u \in L^\infty(\mathbb{R} \times]0, T[) \quad \wedge \quad \int_{-\infty}^{\infty} \int_0^T \left\{ u \frac{\partial \Phi}{\partial t} + f(u) \frac{\partial \Phi}{\partial x} \right\} dt dx + \int_{-\infty}^{\infty} u_0(x) \Phi(x, 0) dx = 0 ,$$

for all $\Phi \in C_0^\infty(\mathbb{R} \times [0, T])$, $\Phi(\cdot, T) = 0$.

(Q11.2.5.11.B) [Rankine-Hugoniot jump condition] Explain what the **Rankine-Hugoniot jump condition**

$$\dot{s}(u_l - u_r) = f(u_l) - f(u_r) \quad , \quad \dot{s} := \frac{d\gamma}{d\tau} \quad \text{"propagation speed of discontinuity"} \quad (11.2.4.2)$$

has to do with the **normal continuity** of a vectorfield $\mathbb{R}^2 \mapsto \mathbb{R}^2$.

(Q11.2.5.11.C) [Shock solutions] We consider the traffic flow equation

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u(1-u)) = 0 \quad \text{in } \mathbb{R} \times \mathbb{R}^+ .$$

Formulate a sufficient and necessary condition on the left and right states u_l and u_r of a **Riemann problem** such that it has a **right-moving** shock as a weak solution (not necessarily an entropy solution).

(Q11.2.5.11.D) [Shock speed] Give an example of a scalar 1D conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}f(u) = 0 \quad \text{in } \mathbb{R} \times \mathbb{R}^+$$

with a strictly convex smooth **flux function** $f : \mathbb{R} \rightarrow \mathbb{R}$ for which every Riemann problem has a **right-moving** shock as a weak solution.

(Q11.2.5.11.E) [Rarefaction wave] We consider the **Riemann problem** for the traffic flow equation

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u(1-u)) = 0 \quad \text{in } \mathbb{R} \times \mathbb{R}^+ .$$

with left state $u_l = 1$ and right state $u_r = 0$. Give the expressions for the resulting **rarefaction solution**.

Lemma 11.2.5.10. Rarefaction solution of Riemann problem

If $f \in C^2(\mathbb{R})$ is strictly $\begin{cases} \text{convex and } u_l < u_r, \\ \text{concave and } u_r < u_l, \end{cases}$ then

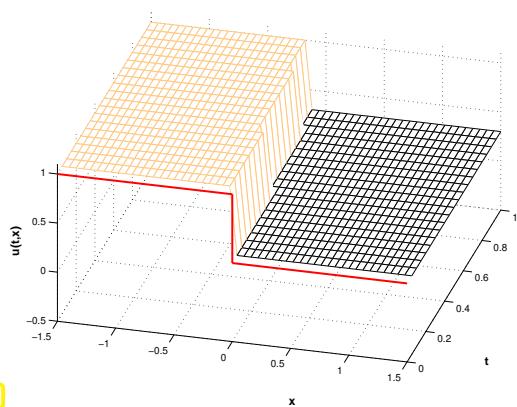
$$u(x,t) := \begin{cases} u_l & \text{for } x < \min\{f'(u_l), f'(u_r)\} \cdot t, \\ g\left(\frac{x}{t}\right) & \text{for } \min\{f'(u_l), f'(u_r)\} < \frac{x}{t} < \max\{f'(u_l), f'(u_r)\}, \\ u_r & \text{for } x > \max\{f'(u_l), f'(u_r)\} \cdot t, \end{cases}$$

$g := (f')^{-1}$, is a continuous weak solution of the Riemann problem.

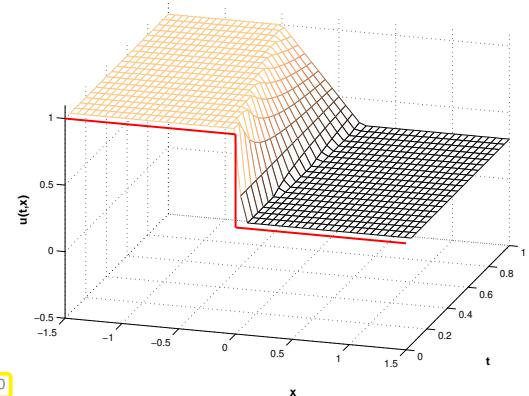
△

11.2.6 Entropy Condition

In Section 11.2.5 we discovered that weak solutions of a scalar conservation law need not be unique. If f' is decreasing as in the traffic flow equation (11.1.2.23) and $u_l > u_r$ both a shock and a rarefaction wave provide valid weak solutions.



Riemann solution: shock



Riemann solution: rarefaction wave

How to select “physically meaningful” = admissible solution ?

- ① Comparison with results from microscopic models, see Ex. 11.2.5.6 for the case of traffic flow.
- ② **Vanishing viscosity technique** (\rightarrow Ex. 11.2.5.7 for Burgers’ equation): add an “ ϵ -amount” of diffusion (“friction”) and study solution for $\epsilon \rightarrow 0$.

However, desirable: simple selection criteria (**entropy conditions**)

Definition 11.2.6.1. Lax entropy condition

u $\hat{=}$ weak solution of (11.2.2.1), piecewise classical solution in a neighborhood of C^2 -curve $\Gamma := (\gamma(\tau), \tau)$, $0 \leq \tau \leq T$, discontinuous across Γ .

u satisfies the **Lax entropy condition** in $(x_0, t_0) \in \Gamma$: $\Leftrightarrow f'(u_l) > \dot{s} := \frac{f(u_l) - f(u_r)}{u_l - u_r} > f'(u_r)$.



Characteristic curves must not emanate from shock \leftrightarrow no “generation of information”

► The **expansion shocks** from Fig. 473–Fig. 475, Fig. 479–Fig. 481 are not allowed.

Parlance: shock satisfying Lax entropy condition = **physical shock**

Note: f' increasing
decreasing \blacktriangleright by Def. 11.2.6.1 necessary for physical shock

$u_l > u_r$
$u_l < u_r$

Physically meaningful weak solution of conservation law = **entropy solution**

For scalar conservation laws with locally Lipschitz-continuous flux function f [Eva98, Sect. 11.4.3]:

Existence & uniqueness of entropy solutions

Remark 11.2.6.2 (General entropy solution for 1D scalar Riemann problem \rightarrow [Osh84]) In fact there is a general formula for the entropy solution of the Riemann problem (\rightarrow Section 11.2.5) for (11.2.2.1) with

arbitrary $f \in C^1(\mathbb{R})$:

$$u(x, t) = \psi(x/t) \quad , \quad \psi(\xi) := \begin{cases} \underset{u_l \leq u \leq u_r}{\operatorname{argmin}}(f(u) - \xi u) & , \text{ if } u_l < u_r , \\ \underset{u_r \leq u \leq u_l}{\operatorname{argmax}}(f(u) - \xi u) & , \text{ if } u_l \geq u_r . \end{cases} \quad (11.2.6.3)$$

□

EXAMPLE 11.2.6.4 (Entropy solution of Burgers equation) An analytic solution is available for Burgers equation (11.1.3.4) with initial data, see [Eva98, Sect. 3.4, Ex. 3]

$$u_0(x) = \begin{cases} 0 & , \text{ if } x < 0 \text{ or } x > 1 , \\ 1 & , \text{ if } 0 \leq x \leq 1 . \end{cases}$$

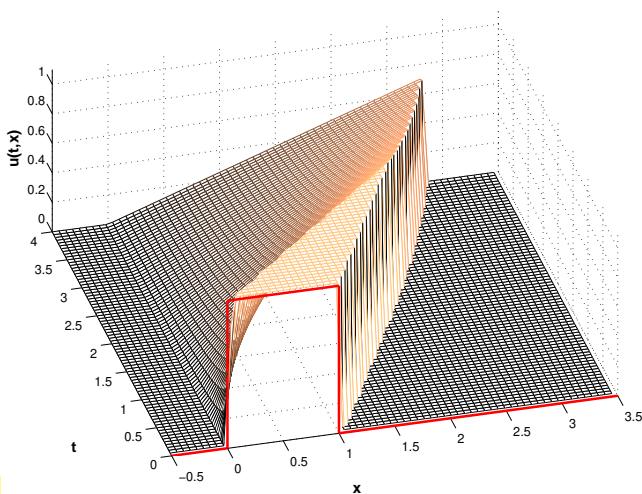


Fig. 491

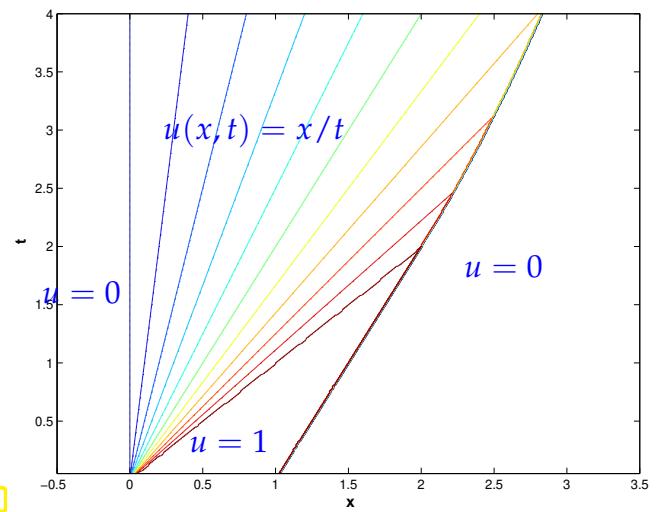


Fig. 492

Vector field in $x - t$ -plane

$$\begin{bmatrix} f(u(x, t)) \\ u(x, t) \end{bmatrix}$$

for entropy solution $u = u(x, t)$

▷

Observe the normal continuity across the shock: the vector field is tangential to the shock curve.

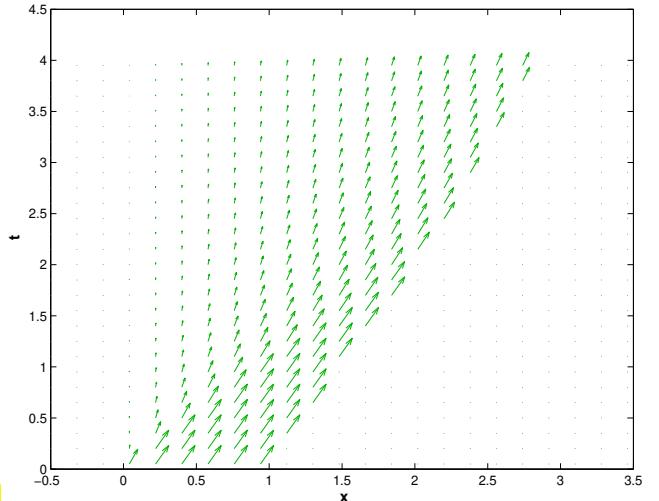


Fig. 493

□

EXAMPLE 11.2.6.5 (Entropy solution of Traffic Flow equation) An analytic solution is also available for the traffic flow equation (11.1.2.23) with initial data, see [Eva98, Sect. 3.4, Ex. 3]

$$u_0(x) = \begin{cases} 0.5 & , \text{ if } x < 0 \text{ or } x > 1 , \\ 1 & , \text{ if } 0 \leq x \leq 1 . \end{cases}$$

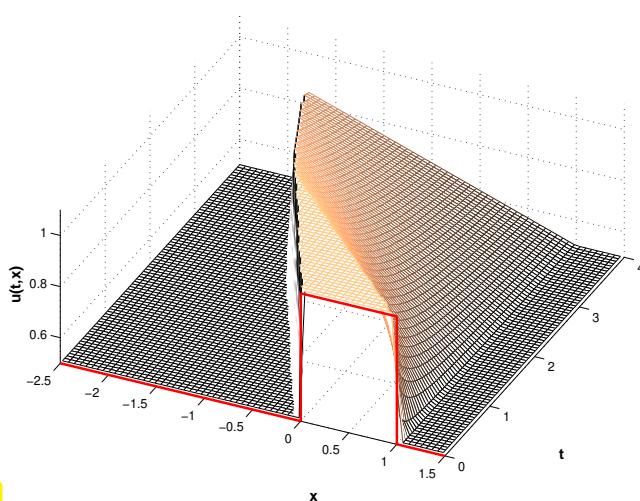


Fig. 494

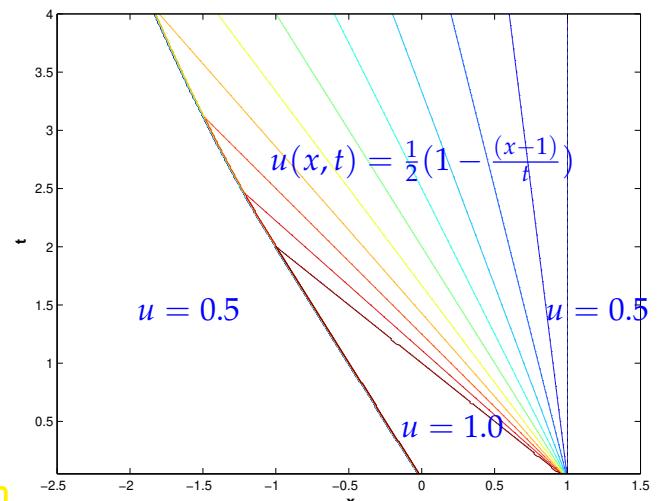


Fig. 495

Vector field in $x - t$ -plane

$$\begin{bmatrix} f(u(x, t)) \\ u(x, t) \end{bmatrix}$$

for entropy solution $u = u(x, t)$

Observe the normal continuity across the shock!

▷.

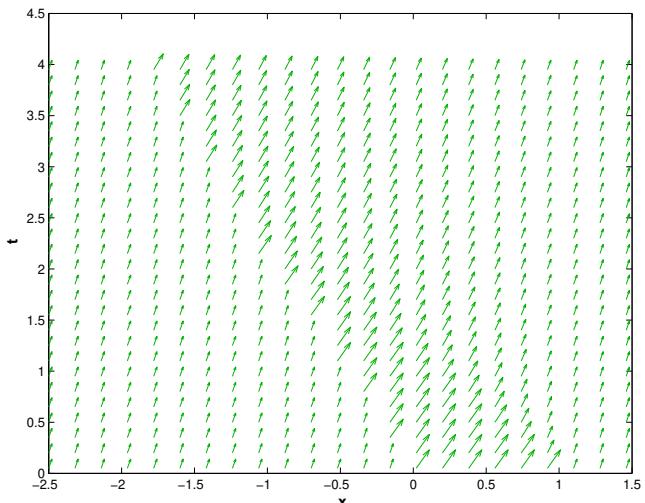


Fig. 496

□

11.2.7 Properties of Entropy Solutions

Existence and uniqueness of entropy solutions for 1D scalar conservation laws is guaranteed by theory.

Setting: $u \in L^\infty(\mathbb{R} \times]0, T[)$ weak (\rightarrow Def. 11.2.3.4) entropy solution of Cauchy problem

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) , \quad x \in \mathbb{R} . \quad (11.2.2.1)$$

with flux function $f \in C^1(\mathbb{R})$ (not necessarily convex/concave).

Notation: $\bar{u} \in L^\infty(\mathbb{R} \times]0, T[)$ $\hat{=}$ entropy solution w.r.t. initial data $\bar{u}_0 \in L^\infty(\mathbb{R})$.

Theorem 11.2.7.1. Comparison principle for scalar conservation laws

$$\text{If } u_0 \leq \bar{u}_0 \text{ a.e. on } \mathbb{R} \Rightarrow u \leq \bar{u} \text{ a.e. on } \mathbb{R} \times]0, T[$$

With obvious consequences, because we get constant solutions for constant initial values:



$$u_0(x) \in [\alpha, \beta] \text{ on } \mathbb{R} \Rightarrow u(x, t) \in [\alpha, \beta] \text{ on } \mathbb{R} \times]0, T[$$

Note: this guarantees the normalization condition $0 \leq u(x, t) \leq 1$ for the traffic flow model, if it is satisfied for the initial data u_0 .

► L^∞ -stability (\Rightarrow no blow-up can occur!)

$$\forall 0 \leq t \leq T: \|u(\cdot, t)\|_{L^\infty(\mathbb{R})} \leq \|u_0\|_{L^\infty(\mathbb{R})}. \quad (11.2.7.2)$$

Theorem 11.2.7.3. L^1 -contractivity of evolution for scalar conservation law

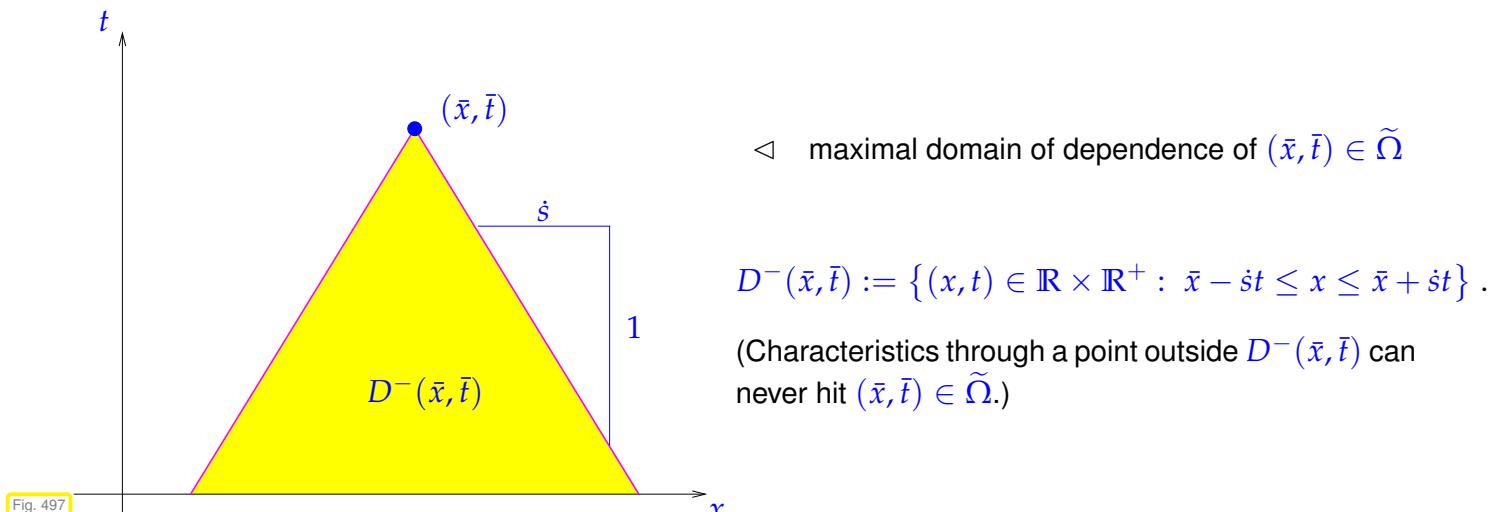
$$\forall t \in]0, T[, R > 0: \int_{|x| < R} |u(x, t)| dx \leq \int_{|x| < R + \dot{s}t} |u_0(x)| dx,$$

with *maximal speed of propagation*

$$\dot{s} := \max\{|f'(\xi)| : \inf_{x \in \mathbb{R}} u_0(x) \leq \xi \leq \sup_{x \in \mathbb{R}} u_0(x)\}. \quad (11.2.7.4)$$

Thm. 11.2.7.3 ► finite speed of propagation in conservation law, bounded by \dot{s} from (11.2.7.4):

► As in the case of the wave equation → Section 9.3.2:

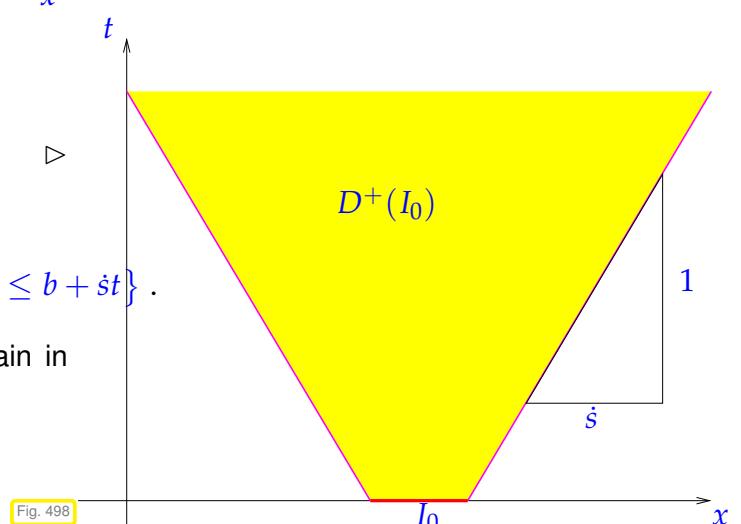


maximal domain of influence of $I_0 \subset \mathbb{R}$

For $I_0 = [a, b]$

$$D^+([a, b]) := \{(x, t) \in \mathbb{R} \times \mathbb{R}^+ : a - \dot{s}t \leq x \leq b + \dot{s}t\}.$$

(Characteristics starting in I_0 will always remain in $D^+(I_0)$.)



Analogous to Thm. 9.3.2.8:

Corollary 11.2.7.5. Domain of dependence for scalar conservation law → [Daf00, Cor. 6.2.2]

The value of the entropy solution at $(\bar{x}, \bar{t}) \in \tilde{\Omega}$ depends only on the restriction of the initial data to $\{x \in \mathbb{R}: |x - \bar{x}| < \dot{s}\bar{t}\}$, where \dot{s} is defined in (11.2.7.4).

Another strand of theoretical results asserts that the solution of a 1D scalar conservation law cannot develop oscillations:

u solves (11.2.2.1) ➤ No. of local extrema (in space) of $u(\cdot, t)$ decreasing with time

Review question(s) 11.2.7.6 (1D conservation laws: entropy solutions and their properties)

(Q11.2.7.6.A) [Lax entropy condition] Explain the “meaning” of the Lax entropy condition relying on the following two figures:

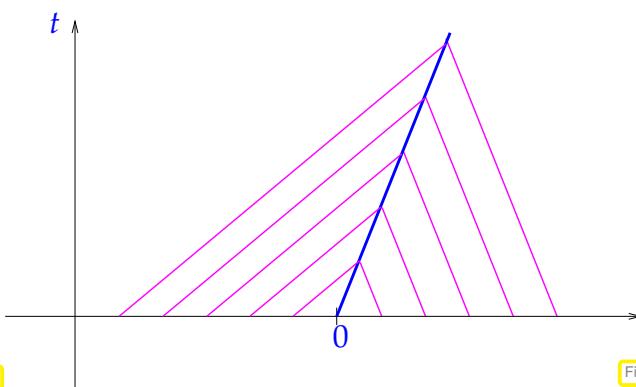


Fig. 499

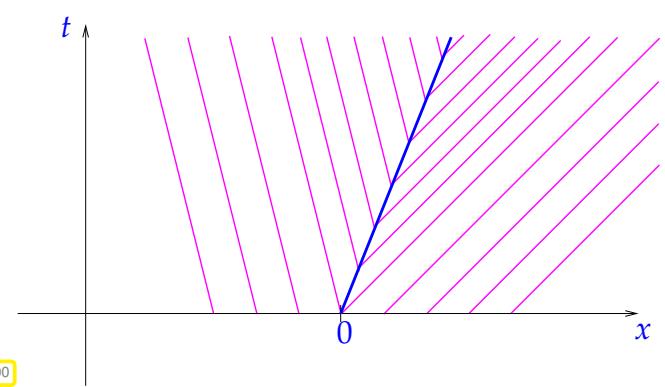


Fig. 500

(Q11.2.7.6.B) [Entropy solutions of Riemann problems] For $u_0(x) = 0$ for $x < 0$, $u_0(x) = 1$ for $x \geq 0$ give the formulas for the entropy solutions of the **Riemann problems** for the scalar 1D conservation laws with **flux functions**

1. $f(u) = u^4$, $u \in \mathbb{R}$,
2. $f(u) = \log(1 + u)$, $u > -1$,
3. $f(u) = 1 - e^u$, $u \in \mathbb{R}$,
4. $f(u) = \frac{1}{1+u}$, $u > -1$.

(Q11.2.7.6.C) [Domains of dependence and influence] We consider a 1D scalar conservation law with smooth flux function $f: \mathbb{R} \rightarrow \mathbb{R}$ and compactly supported initial data u_0 satisfying $\alpha \leq u_0(x) \leq \beta$ for almost all $x \in \mathbb{R}$.

Explain the concepts of

- (maximal) **domain of dependence** of a point (x, t)
- (maximal) **domain of influence** of an interval $I_0 \subset \mathbb{R}$

and how they can be computed based on the following two results about **entropy solutions**:

Theorem 11.2.7.1. Comparison principle for scalar conservation laws

If $u_0 \leq \bar{u}_0$ a.e. on \mathbb{R} ⇒ $u \leq \bar{u}$ a.e. on $\mathbb{R} \times]0, T[$

Theorem 11.2.7.3. L^1 -contractivity of evolution for scalar conservation law

$$\forall t \in]0, T[, R > 0: \int_{|x| < R} |u(x, t)| dx \leq \int_{|x| < R + \dot{s}t} |u_0(x)| dx ,$$

with *maximal speed of propagation*

$$\dot{s} := \max\{|f'(\xi)| : \inf_{x \in \mathbb{R}} u_0(x) \leq \xi \leq \sup_{x \in \mathbb{R}} u_0(x)\} . \quad (11.2.7.4)$$

(Q11.2.7.6.D) We consider the Cauchy problem for a one-dimensional scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \quad \text{in } \mathbb{R} \times \mathbb{R}^+, \quad u(x, 0) = u_0(x) \quad \forall x \in \mathbb{R} . \quad (11.2.2.1)$$

Formulate most general conditions on the flux function $f : \mathbb{R} \rightarrow \mathbb{R}$ such that

$$u \text{ solves (11.2.2.1)} \Rightarrow u + c \text{ solves (11.2.2.1) for all } c \in \mathbb{R}.$$

Here, “solves” is to be read in the sense of weak solutions.

△

11.3 Conservative Finite Volume (FV) Discretization

After we have some understanding of solutions of scalar conservation laws and of their properties, we can tackle the issue of how to discretize the Cauchy problems. Parallel to the approach to linear evolution problems of Chapter 9 we first deal with discretization in space and focus on a mesh/grid-based approach. We pay particular attention to the capability of the spatial discretization to capture essential qualitative properties of the solutions.

11.3.1 Prologue: Finite-Difference Method (FDM)

A temptingly simple way to discretize partial differential in a single spatial dimension is the **finite difference** approach. It is based on the strong (PDE) form of the boundary value problem/evolution problem, cf. Section 4.1.

All finite difference methods (FDMs) rely on spatial **grids** (meshes), for the finite one-dimensional domain $\Omega =]a, b[, a < b$, defined through the sets of its **nodes**

$$\begin{aligned} \mathcal{V}(\mathcal{M}) &:= \{a = x_0 < x_1 < \dots < x_{M-1} < x_M = b\}, \quad M \in \mathbb{N} , \\ \blacktriangleright \quad \text{grid/mesh} \quad \mathcal{M} &:= \{]x_{i-1}, x_i[, i = 1, \dots, M\} . \end{aligned}$$

A grid is called **equidistant** with **meshwidth** $h_{\mathcal{M}} := \frac{b-a}{M}$, if $x_i = a + ih$, $i = 0, \dots, M$.

The finite difference method is inspired by the definition of derivatives through limits of difference quotients, for instance, for $u \in C^{[a,b]}(\Omega)$

$$\frac{df}{dx}(x_0) := \lim_{h \rightarrow 0} \frac{f(x_0 + h) - f(x_0)}{h} , \quad x_0 \in]a, b[.$$

This suggests that for small width/span $h > 0$ a difference quotient supplies a good approximation for the derivative. This idea is widely exploited for numerical differentiation, see, e.g., [Hip19, ??].

Idea: Construction of finite-difference discretization


- (I) : Replace derivatives in PDE with difference quotients.
- (II): Anchor the difference quotients at the nodes of a mesh and choose their widths so that they connect the values of a function at the nodes ("nodal values").

There are several options for choosing difference quotients that approximate a first derivative:

- ◆ The **symmetric difference quotient** at the anchor point x_0

$$\frac{df}{dx}(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}, \quad \text{with span/width } h > 0. \quad (11.3.1.2)$$

- ◆ The **backward difference quotient** at the anchor point x_0

$$\frac{df}{dx}(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h}, \quad \text{with span/width } h > 0. \quad (11.3.1.3)$$

- ◆ The **forward difference quotient** at the anchor point x_0

$$\frac{df}{dx}(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}, \quad \text{with span/width } h > 0. \quad (11.3.1.4)$$

§11.3.1.5 (Spatial finite-difference scheme for 1D linear advection) From Ex. 11.1.1.10 we know the linear advection evolution problem in one spatial dimension described by the PDE

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(v(x)u) = 0 \quad \text{in } \tilde{\Omega} := \Omega \times]0, T[, \quad (11.3.1.6)$$

where $v = v(x) : \Omega \rightarrow \mathbb{R}$ is a continuous velocity field.

We consider the Cauchy problem, that is, $\Omega = \mathbb{R}$, over a finite time interval $[0, T]$, $T > 0$. We equip $\Omega = \mathbb{R}$ with an infinite equidistant grid with nodes $x_i := ih$, $i \in \mathbb{Z}$, and meshwidth $h := h_M > 0$. The unknowns will be the values $\mu_i(t) \approx u(x_i, t)$, $i \in \mathbb{Z}$, collected in the infinite time-dependend vector $\vec{\mu} : [0, T] \rightarrow \mathbb{R}^{\mathbb{Z}}$.

Then we approximate the spatial derivative $\frac{\partial}{\partial x}$ by difference quotients and we end up with one of the following **spatially semi-discrete** evolution problems, ordinary differential equations with state space $\mathbb{R}^{\mathbb{Z}}$.

- ◆ If we use a backward finite difference quotient, we obtain

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(v(x)u) = 0 &\rightarrow \frac{\partial u}{\partial t}(x_i) + \frac{v(x_i)u(x_i, t) - v(x_{i-1})u(x_{i-1}, t)}{h} = 0, \\ &\rightarrow \dot{\mu}_i(t) + \frac{v(x_i)\mu_i(t) - v(x_{i-1})\mu_{i-1}(t)}{h}. \end{aligned} \quad (11.3.1.7)$$

- ◆ Using centered difference quotients we get

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(v(x)u) = 0 &\rightarrow \frac{\partial u}{\partial t}(x_i) + \frac{v(x_{i+1})u(x_{i+1}, t) - v(x_{i-1})u(x_{i-1}, t)}{2h} = 0, \\ &\rightarrow \dot{\mu}_i(t) + \frac{v(x_{i+1})\mu_{i+1}(t) - v(x_{i-1})\mu_{i-1}(t)}{2h}. \end{aligned} \quad (11.3.1.8)$$

- ◆ Forward difference quotients yield

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(v(x)u) = 0 &\longrightarrow \frac{\partial u}{\partial t}(x_i) + \frac{v(x_{i+1})u(x_{i+1}, t) - v(x_i)u(x_i, t)}{h} = 0, \\ &\longrightarrow \dot{\mu}_i(t) + \frac{v(x_{i+1})\mu_{i+1}(t) - v(x_{i-1})\mu_{i-1}(t)}{h}. \end{aligned} \quad (11.3.1.9)$$

For the Cauchy problem we have to specify an initial condition $u(x, 0) = u_0(x)$, $x \in]bbR$. This helps us define the initial vector for the spatially semi-discrete evolution problem

$$\vec{\mu}(0) = [u_0(x_i)]_{i \in \mathbb{Z}}.$$

↓

EXPERIMENT 11.3.1.10 (Finite-difference discretization of 1D linear advection with constant velocity) We use finite-difference discretization in space to compute an approximate solution of a Cauchy problem for the linear advection equation $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(v(x)u) = 0$ for constant velocity $v \equiv 1$. We consider the temporal interval $[0, 1]$ and initial data u_0 supported in $[0, 1]$.

From Ex. 11.1.1.10 we recall that the exact solution of the present Cauchy problem is given by $u(x, t) = u_0(x - t)$, $(x, t) \in \mathbb{R} \times [0, 1]$: the solution is the initial distribution traveling in positive direction with speed 1. In particular, the space-times support of the solution $u = u(x, t)$ is contained in $[0, 2] \times [0, 1]$. This makes possible the truncation of the spatial domain to $\Omega^* :=]-1, 3[$.

We rely on an equidistant spatial mesh with node set

$$\mathcal{V}(\mathcal{M}) := \{x_i := ih, i = -n, \dots, 3n\}, \quad h := \frac{1}{n}, \quad n := 100.$$

For the nodal values μ_i we impose

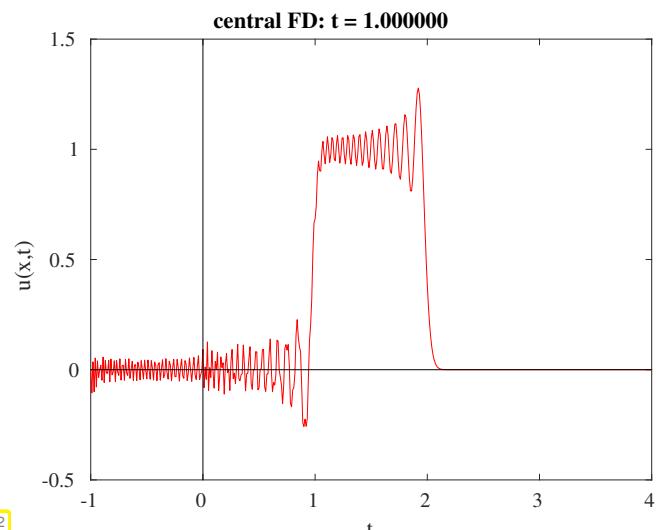
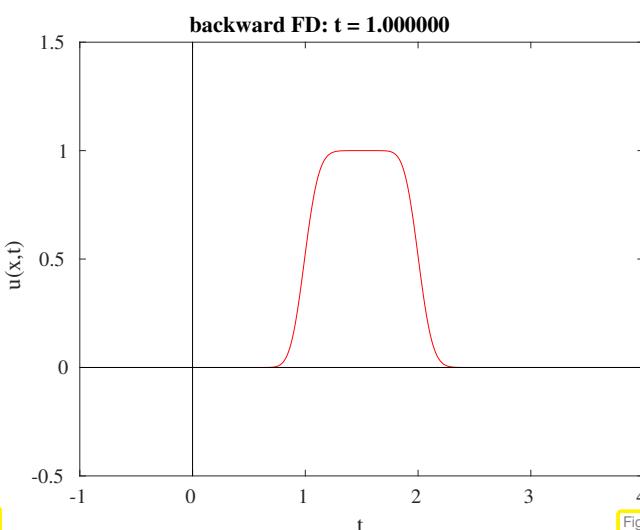
$$\mu_i(t) := 0 \quad \text{for } i \leq -n \quad \text{or} \quad i \geq 3n, \quad 0 \leq t \leq 1.$$

This leaves $\mu_{-n+1}(t), \dots, \mu_{3n-1}(t)$ as the $4n - 1$ unknowns to be evolved according to the ODEs (11.3.1.7), (11.3.1.8), and (11.3.1.9). For spatial discretization we use a 5th-order explicit Runge-Kutta with adaptive timestep control, **Ode45** from § 6.5.3.3.

The initial distribution was given by a discontinuous box function

$$u_0(x) = \begin{cases} 1 & \text{for } 0 \leq x \leq 1, \\ 0 & \text{elsewhere.} \end{cases}$$

We plot the solutions obtained at final time $T = 1$, rendering uses piecewise linear interpolation.



Backward difference quotient, (11.3.1.7)

Symmetric difference quotient, (11.3.1.8)

If we use the forward difference quotient as in (11.3.1.9), we observe *blow-up*. No meaningful plot can be produced.

We observe, a qualitatively correct solution for (11.3.1.7), the finite difference method based on backward difference quotients. Spurious oscillations arise when symmetric difference quotients are used as in (11.3.1.8).

The same observations were made in Section 10.2 and Section 10.3 for both stationary and transient one-dimensional linear convection-diffusion diffusion problems, see Exp. 10.2.2.4 and Exp. 10.2.2.10 for stationary transport problems, and Exp. 10.3.1.3 for the time-dependent case. This can be expected, because the transport equation (10.3.2.1) boils down to (11.3.1.6) for $d = 1$ and constant velocity. \square

§11.3.1.11 (Fully discrete finite-difference method for linear advection) As in Exp. 11.3.1.10 for a constant velocity $v > 0$ we consider the Cauchy problem

$$\frac{\partial u}{\partial t} + v \frac{\partial}{\partial x}(u) = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) , \quad x \in \mathbb{R} , \quad (11.3.1.12)$$

with exact solution $u(x, t) = u_0(x - vt)$, $(x, t) \in \mathbb{R} \times]0, T[$.

The characteristics are parallel lines in the $x - t$ -plane with slope v , see Ex. 11.2.2.5. Thus, according to the interpretation given in Rem. 11.2.2.9, information in the model propagates with speed v in positive x -direction.

We combine spatial finite-difference semi-discretization on a an (infinite) equidistant grid as introduced in § 11.3.1.5 with meshwidth $h > 0$ with **explicit Euler timestepping** (9.2.6.5) with uniform timestep $\tau > 0$ and, thus, arrive at a fully discrete evolution problems defining sequences of vectors $(\vec{\mu}^{(j)})_{j=1}^m$, $m = T/\tau$, $\vec{\mu}^{(j)} \in \mathbb{R}^Z$, $\vec{\mu}^{(0)} = \vec{\mu}(0)$:

- ◆ If $\frac{\partial}{\partial x}$ is approximated by a backward difference quotient as in (11.3.1.7), we get

$$\mu_i^{(j+1)} = \mu_i^{(j)} - v \frac{\tau}{h} (\mu_i^{(j)} - \mu_{i-1}^{(j)}) , \quad i \in \mathbb{Z}, j = 0, \dots, m-1 . \quad (11.3.1.13)$$

Note that in the case of the “magic timestep” $\tau = \frac{h}{v}$, the recursion simplifies to

$$\mu_i^{(j+1)} = \mu_{i-1}^{(j)} .$$

In this case the fully discrete evolution **exactly** captures transport with velocity v and we have $\mu_i^{(j)} = u(ih, j\tau)$!

- ◆ From (11.3.1.8), which arose from symmetric difference quotients, we get

$$\mu_i^{(j+1)} = \mu_i^{(j)} - v \frac{\tau}{2h} (\mu_{i+1}^{(j)} - \mu_{i-1}^{(j)}) , \quad i \in \mathbb{Z}, j = 0, \dots, m-1 . \quad (11.3.1.14)$$

The choice $\tau = \frac{h}{v}$ leads to the simpler evolution

$$\begin{aligned} \mu_i^{(j+1)} &= \frac{1}{2} (\mu_{i-1}^{(j)} + 2\mu_i^{(j)} - \mu_{i+1}^{(j)}) \\ \Leftrightarrow \quad \mu_i^{(j+1)} - \mu_{i-1}^{(j)} &= \frac{1}{2} (-\mu_{i-1}^{(j)} + 2\mu_i^{(j)} - \mu_{i+1}^{(j)}) , \quad i \in \mathbb{Z}, j = 0, \dots, m-1 . \end{aligned}$$

We observe that the exact advection of the initial value is perturbed by another difference quotient that corresponds to a discretization of $\frac{d^2}{dx^2}$, cf. (2.3.3.4) and Rem. 10.2.2.3. Thus, this term will introduce large errors, once the solution develops oscillations.

- ◆ If we use forward difference quotients to approximate $\frac{\partial u}{\partial x}$ as in (11.3.1.9) we end up with

$$\mu_i^{(j+1)} = \mu_i^{(j)} - v \frac{\tau}{h} (\mu_{i+1}^{(j)} - \mu_i^{(j)}) , \quad i \in \mathbb{Z}, j = 0, \dots, m-1 . \quad (11.3.1.15)$$

This scheme is completely wrong about the direction of propagation of information for $v > 0$. Blow-up is an inevitable consequence.

□

EXAMPLE 11.3.1.16 (Naive finite difference scheme for Burgers equation) Now we tackle the Cauchy Problem for a non-linear conservation law, namely Burgers equation (11.1.3.4), with a simple finite-difference method.



This will present a warning example that simply replacing derivatives with difference quotients to discretize conservation laws may yield spurious schemes.

We rewrite Burgers equation (11.1.3.4) using the product rule:

$$\frac{\partial u}{\partial t}(x, t) + u(x, t) \frac{\partial u}{\partial x}(x, t) = 0 \quad \text{in } \mathbb{R} \times]0, T[.$$

Now this looks like a linear advection equation with velocity $v(x, t) = u(x, t)$:

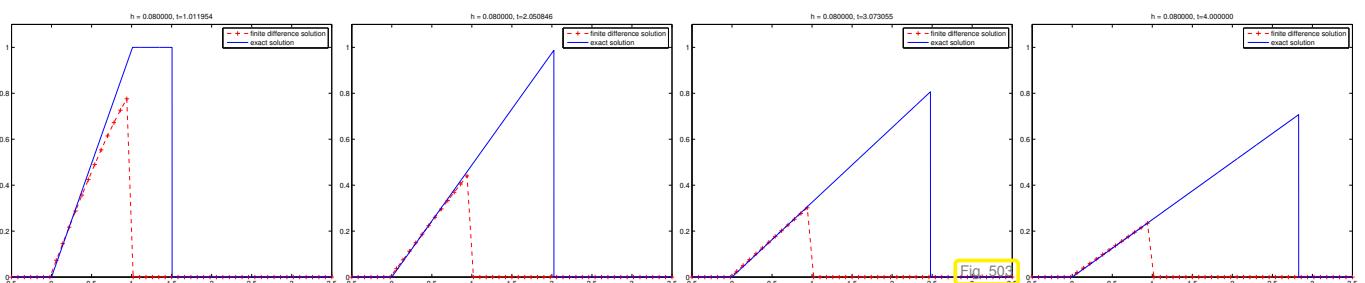
$$\begin{aligned} \text{Burgers' equation: } \frac{\partial u}{\partial t}(x, t) + u(x, t) \frac{\partial u}{\partial x}(x, t) &= 0 \quad \text{in } \mathbb{R} \times]0, T[. \\ \Downarrow &\qquad \Downarrow \\ \text{linear advection: } \frac{\partial u}{\partial t}(x, t) + v(x, t) \frac{\partial u}{\partial x}(x, t) &= 0 \quad \text{in } \mathbb{R} \times]0, T[. \end{aligned}$$

We assume $u_0(x) \geq 0$. Then, by Thm. 11.2.7.1, $u(x, t) \geq 0$ for all $0 < t < T$, that is, transport is always going in positive x -direction. In this first part of this section we have learned that in this case we should approximate $\frac{\partial u}{\partial x}$ by means of a backward difference quotient (11.3.1.3). This also heeds the advice from Section 10.3.1 because it yields an **upwind discretization** in space.

Thus, on an (infinite) equidistant spatial grid with meshwidth $h > 0$, that is, $x_j := hj, j \in \mathbb{Z}$, we obtain a semi-discrete evolution problem for the nodal values $\mu_j = \mu_j(t) \approx u(x_j, t)$.

$$\begin{aligned} \frac{\partial u}{\partial t}(x, t) + u(x, t) \frac{\partial u}{\partial x}(x, t) &= 0 \quad \text{in } \mathbb{R} \times]0, T[. \\ \Downarrow &\qquad \Downarrow \\ \dot{\mu}_j(t) + \mu_j \frac{\mu_j - \mu_{j-1}}{h} &= 0, \quad j \in \mathbb{Z}, \quad 0 < t < T . \end{aligned} \quad (11.3.1.17)$$

Our numerical experiment tackles the Cauchy problem from Ex. 11.2.6.4, “box shaped” initial data u_0 , $h = 0.08$, integration of (11.3.1.17) with adaptive explicit Runge-Kutta method `ode45`.



Observation from numerical experiment: OK for rarefaction wave, but *scheme cannot capture speed of shock correctly!*

To understand the behavior of the scheme, we consider the Riemann problem with $u_0(x) = 1$ for $x < 0 - \epsilon$, and $u_0(x) = 0$ for $x > 0 - \epsilon$, $\epsilon \ll 1$. Accordingly, we choose as initial value for the semidiscrete evolution

$$\mu_j(0) = \begin{cases} 1 & , \text{ if } j < 0 , \\ 0 & , \text{ if } j \geq 0 , \end{cases}$$

Then, it is easy to see that $\dot{\mu}_j = 0$ for all $j \in \mathbb{Z}$.

Entropy solution (for this u_0) = travelling shock (\rightarrow Lemma 11.2.5.4), speed
 $\dot{s} = \frac{1}{2} > 0$



Numerical solution:
 $\vec{\mu}(t) = \vec{\mu}_0$ for all $t > 0$!

► 3-point FDM (11.3.1.17) “converges” to wrong solution !

□

In the next section we will learn an approach to the discretization of 1D conservation laws that has some built-in safeguards against failures as confronted in the above example.

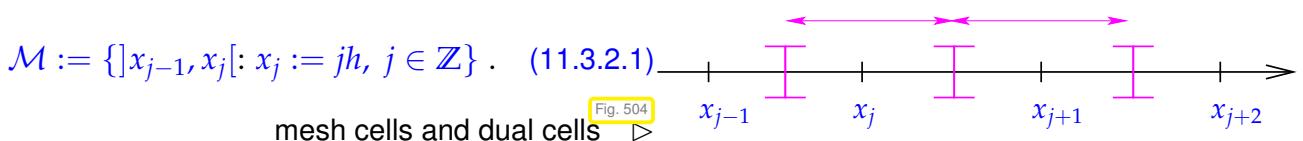
11.3.2 Spatially Semi-Discrete Conservation Form

Objective: spatial semi-discretization of a Cauchy problem for a general scalar conservation law in one spatial dimension:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) , \quad x \in \mathbb{R} . \quad (11.2.2.1)$$

on an (infinite) equidistant spatial mesh with mesh width $h > 0$.

Remember: We have already seen spatial semi-discretization in the context of the **method of lines**, see Section 9.2.4. In a sense, our treatment of conservation laws follows a method of lines approach.



The time-dependent unknowns of the semi-discrete scheme will be denoted by $\mu_j = \mu_j(t)$, $j \in \mathbb{Z}$. They play a similar role as the time-dependent basis expansion coefficients occurring as components of the vector $\vec{\mu} = \vec{\mu}(t)$ in the method of lines ODE Eq. (9.2.4.4).

We adopt a **finite volume interpretation** of the coefficients/unknowns $\mu_j(t)$, $j \in \mathbb{Z}$):

$\mu_j \leftrightarrow$ conserved quantities in **dual cells** $[x_{j-1/2}, x_{j+1/2}]$, midpoints $x_{j-1/2} := \frac{1}{2}(x_j + x_{j-1})$:

$$\mu_j(t) \approx \frac{1}{h} \int_{x_{j-1/2}}^{x_{j+1/2}} u(x, t) dx . \quad (11.3.2.2)$$

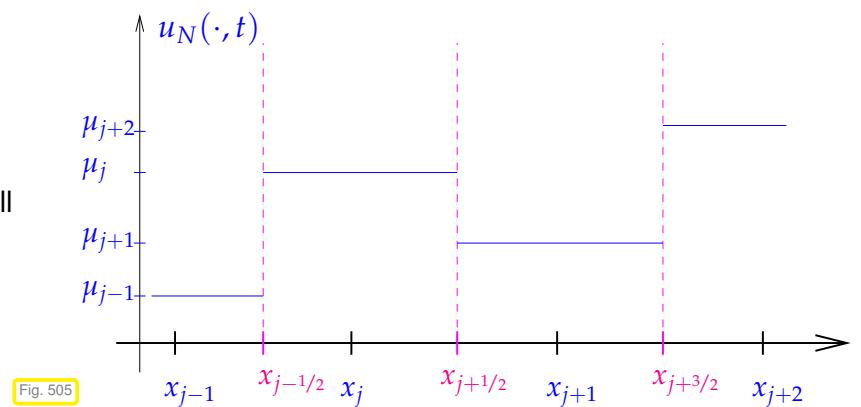
Relate $\vec{\mu}(t) := (\mu_j(t))_{j \in \mathbb{Z}} \in \mathbb{R}^{\mathbb{Z}}$ \longleftrightarrow $u_N(x, t) = \sum_{j \in \mathbb{Z}} \mu_j(t) \chi_{[x_{j-1/2}, x_{j+1/2}]}(x)$. (11.3.2.3)

☞ notation: **characteristic function** $\chi_{[x_{j-1/2}, x_{j+1/2}]}(x) = \begin{cases} 1 & , \text{ if } x_{j-1/2} < x \leq x_{j+1/2} , \\ 0 & \text{elsewhere.} \end{cases}$

☞ $(\mu_j(t))_{j \in \mathbb{Z}}$ \longleftrightarrow **piecewise constant** approximation $u_N(t) \approx u(\cdot, t)$

Note:

$u_N(t)$ is discontinuous at dual cell boundaries $x_{j+1/2}$!



By spatial integration over dual cells, which now play the role of the control volumes in (11.2.1.1), and applying the fundamental theorem of calculus, we obtain

$$\begin{aligned} \frac{\partial u}{\partial t}(x, t) + \frac{\partial}{\partial x}(f(u(x, t))) &= 0, \quad x \in \mathbb{R}, \\ \Rightarrow \frac{d}{dt} \int_{x_{j-1/2}}^{x_{j+1/2}} u(x, t) dx + f(u(x_{j+1/2}, t)) - f(u(x_{j-1/2}, t)) &= 0, \quad j \in \mathbb{Z}, \end{aligned} \quad (11.3.2.4)$$

$$\Rightarrow \frac{d\mu_j}{dt}(t) + \frac{1}{h} \left(\underbrace{f(u_N(x_{j+1/2}, t))}_{?} - \underbrace{f(u_N(x_{j-1/2}, t))}_{?} \right) = 0, \quad j \in \mathbb{Z}. \quad (11.3.2.5)$$

Problem: owing to the jumps of $u_N(t)$ we face the ambiguity of the values $u_N(x_{j+1/2}, t), u_N(x_{j-1/2}, t)$. (We encountered a similar situation it in the context of upwind quadrature in Section 10.2.2.1.)

Abstract “solution”:

Approximation $f(u_N(x_{j+1/2}, t)) \approx f_{j+1/2}(t) := F(\mu_{j-m_l+1}(t), \dots, \mu_{j+m_r}(t)), \quad j \in \mathbb{Z},$

with numerical flux function $F : \mathbb{R}^{m_l+m_r} \mapsto \mathbb{R}, \quad m_l, m_r \in \mathbb{N}_0$.

Note: If $f = f(u)$, then the **same** numerical flux function is usually used for all dual cells!

When we plug this approximation into (11.3.2.5) we end up with the following (formally infinite) system of ODEs:

Finite volume semi-discrete evolution for (11.2.2.1) in conservation form

$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h} (F(\mu_{j-m_l+1}(t), \dots, \mu_{j+m_r}(t)) - F(\mu_{j-m_l}(t), \dots, \mu_{j+m_r-1}(t))) , \quad j \in \mathbb{Z} . \quad (11.3.2.7)$$

numerical flux (function) $F : \mathbb{R}^{m_l+m_r} \mapsto \mathbb{R}$

Special case: **2-point numerical flux** ($m_l = m_r = 1$): $F = F(v, w)$
 $(v \hat{=} \text{left state}, w \hat{=} \text{right state})$

$$(11.3.2.7) \quad \blacktriangleright \quad \frac{d\mu_j}{dt}(t) = -\frac{1}{h} (F(\mu_j(t), \mu_{j+1}(t)) - F(\mu_{j-1}(t), \mu_j(t))) , \quad j \in \mathbb{Z} . \quad (11.3.2.8)$$

Assumption on numerical flux functions: F Lipschitz-continuous in each argument.

The following code implements the right-hand-side of (11.3.2.8) for a generic 2-point numerical flux function passed through a functor object. Of course, it can deal with finite-size vectors $\vec{\mu} \in \mathbb{R}^N$, $N \in \mathbb{N}$, only. Therefore, it assumes

$$\mu_j = (\vec{\mu})_1 \quad \text{for } j < 1 \quad \text{and} \quad \mu_\ell = (\vec{\mu})_N \quad \text{for } j > N ,$$

and computes the right-hand-side values only for the indices $j = 1, \dots, N$.

C++ & EIGEN code 11.3.2.9: Right hand side function for MOL-ODE (11.3.2.8) → GITLAB

```

2 // arguments: (Finite) state vector  $\mu$  of cell averages, see
3 // (11.3.2.2)
4 // Functor  $F : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ , 2-point
5 // numerical flux
6 // return value: Vector with differences of numerical fluxes, which
7 // provides
8 // the right hand side of (11.3.2.8)
9 template <typename FunctionF>
10 VectorXd fluxdiff(const VectorXd &mu, FunctionF &&F) {
11     unsigned n = mu.size(); // length of state vector
12     VectorXd fd = VectorXd::Zero(n); // return vector
13
14     // constant continuation of data for  $x \leq a$ !
15     fd[0] = F(mu[0], mu[1]) - F(mu[0], mu[0]);
16     for (unsigned j = 1; j < n - 1; ++j) {
17         fd[j] = F(mu[j], mu[j + 1]) - F(mu[j - 1], mu[j]); // see (11.3.2.8)
18     }
19     // constant continuation of data for  $x \geq b$ !
20     fd[n - 1] = F(mu[n - 1], mu[n - 1]) - F(mu[n - 2], mu[n - 1]);
21     // Efficient thanks to return value optimization (RVO)
22     return fd;
}

```

The next code demonstrates the use of an explicit Runge-Kutta single-step method for solving (11.3.2.8) after truncation to a finite spatial interval $[a, b]$.

C++ & EIGEN code 11.3.2.10: Wrapper code for finite volume evolution with 2-point flux
→ GITLAB

2 // arguments:

```

3 // Real numbers  $a, b$ , the boundaries of the interval,
4 // unsigned int  $N$ , the number of cells,
5 // Functor  $u_0 : \mathbb{R} \mapsto \mathbb{R}$ , initial value,
6 // Final time  $T > 0$ ,
7 // Functor  $F = F(v, w)$  for 2-point numerical flux function.
8 //
9 // return value:
10 // Vector with cell values at final time  $T$ 
11 //
12 // Finite volume discrete evolution in conservation form with 2-point
13 // flux, see (11.3.2.8); Cauchy problem over time  $[0, T]$ ,
14 // timestepping with adaptive explicit Runge-Kutta single-step method
15 // of order 5(4).
16 template <typename FunctionU0, typename FunctionF>
17 VectorXd conformevl(double a, double b, unsigned N, FunctionU0 u0, double T,
18                      FunctionF &&F) {
19     double h = (b - a) / N; // meshwidth
20     // centers of dual cells
21     VectorXd x = VectorXd::LinSpaced(N, a + 0.5 * h, b - 0.5 * h);
22
23     // vector  $\vec{\mu}_0$  of initial cell averages
24     // obtained by point sampling of  $u_0$  in cell centers
25     VectorXd mu0 = x.unaryExpr(u0);
26
27     // right hand side function for ode solver
28     auto odefun = [&](const VectorXd &mu, VectorXd &dmdt, double t) {
29         dmdt = -1. / h * fluxdiff<FunctionF>(mu, F);
30     };
31
32     // Method of lines approach, c.f. Sect. 9.2.4: timestepping by
33     // Boost integrator (adaptive explicit embedded Runge-Kutta method
34     // of order 5, see also Def. 7.3.3.1)
35     double abstol = 1E-8, reltol = 1E-6; // integration control parameters
36     // std::vector<double> t; Returns temporal grid
37     // std::vector<Eigen::VectorXd> MU; Returns states  $\vec{\mu}^{(k)}$ 
38     // Use this C++17 syntax only, if you are well aware of the return
39     // types
40     auto [t, MU] =
41         ode45(odefun, 0, T, mu0, abstol, reltol); // 
42     // Retrieve approximate state at final time.
43     return MU.back();
44 }

```

Note that in Code 11.3.2.10, Line 40, we rely on high-order explicit Runge-Kutta timestepping in order to solve (11.3.2.7) approximately.

11.3.3 Discrete Conservation Property

We consider a Cauchy problem for a scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) , \quad x \in \mathbb{R} . \quad (11.2.2.1)$$

and its conservative finite volume discretization on an (infinite) equidistant spatial mesh with mesh width $h > 0$:

$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h} (F(\mu_{j-m_l+1}(t), \dots, \mu_{j+m_r}(t)) - F(\mu_{j-m_l}(t), \dots, \mu_{j+m_r-1}(t))) , \quad j \in \mathbb{Z} . \quad (11.3.2.7)$$

We abbreviate

$$f_{j+1/2}(t) := F(\mu_{j-m_l+1}(t), \dots, \mu_{j+m_r}(t)).$$

§11.3.3.1 (Preservation of constant data) An evident first property of finite volume methods in conservation form:

$$\mu_j(0) = \mu_0 \in \mathbb{R} \quad \forall j \in \mathbb{Z} \quad \Rightarrow \quad \mu_j(t) = \mu_0 \quad \forall j \in \mathbb{Z}, \quad \forall t > 0. \quad (11.3.3.2)$$

that is, constant solutions are preserved by the method. Such methods are called **well-balanced discretizations**. \square

§11.3.3.3 (Discrete flux balance) For conservation laws we found the fundamental local balance relation, see (9.2.1.3):

$$\frac{d}{dt} \int_a^b u(x, t) dx = -(f(u(b, t)) - f(u(a, t))). \quad (11.3.3.4)$$

A “telescopic sum argument” combined with the interpretation (11.3.2.3) shows that the conservation form (11.3.2.7) of the semi-discrete conservation law implies

$$\begin{aligned} \frac{d}{dt} \int_{x_{k-1/2}}^{x_{m+1/2}} u_N(x, t) dx &= h \sum_{l=k}^m \frac{d\mu_j}{dt}(t) = -(f_{m+1/2}(t) - f_{k-1/2}(t)) \quad \forall k, m \in \mathbb{Z}. \\ &\Updownarrow \\ \frac{d}{dt} \int_{x_{k-1/2}}^{x_{m+1/2}} u(x, t) dx &= -(f(u(x_{m+1/2}, t)) - f(u(x_{k-1/2}, t))), \end{aligned}$$

► With respect to unions of dual cells and numerical fluxes, the semidiscrete solution $u_N(t)$ satisfies a balance law of the same structure as a (weak) solution of (11.2.2.1). \square

Of course, the numerical flux function F has to fit the flux function f of the conservation law; the following is a minimal requirement for a viable numerical flux function.

Definition 11.3.3.5. Consistent numerical flux function

A numerical flux function $F : \mathbb{R}^{m_l+m_r} \mapsto \mathbb{R}$ is **consistent** with the flux function $f : \mathbb{R} \mapsto \mathbb{R}$, if

$$F(u, \dots, u) = f(u) \quad \forall u \in \mathbb{R}.$$

§11.3.3.6 (Discrete shock speed) Focus: solution of Riemann problem (\rightarrow Section 11.2.5) by finite volume method in conservation form (11.3.2.7):

Initial data “constant at $\pm\infty$ ”: $\mu_{-j}(0) = u_l$, $\mu_j(0) = u_r$ for large j .

Consistency of the numerical flux function implies for large $m \gg 1$

$$\frac{d}{dt} \int_{-x_{-m-1/2}}^{x_{m+1/2}} u_N(x, t) dx = -(F(u_r, \dots, u_r) - F(u_l, \dots, u_l)) = -(f(u_r) - f(u_l)). \quad (11.3.3.7)$$

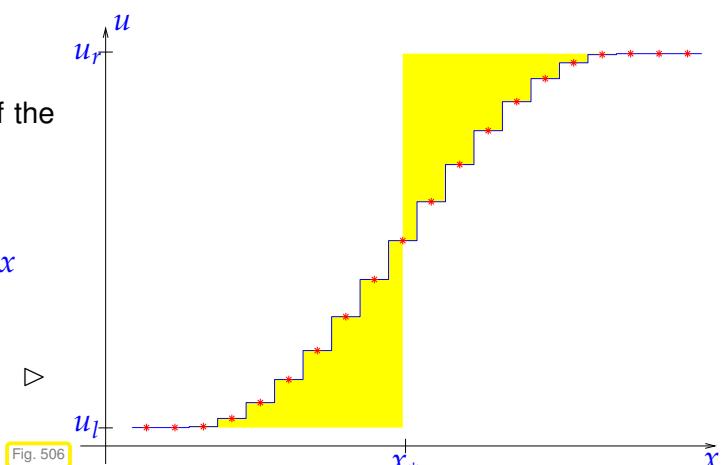
Exactly the same balance law holds for any weak solutions of the Riemann problem!

Situation : $u_r > u_l \geq$ shock in traffic flow, discrete solution $u_N(t)$ increasing & supposed to approximate a shock; we cannot expect that u_N will also feature a sharp discontinuity, rather we may see a “smeared” transition from u_l to u_r .

Write $x_*(t) \in \mathbb{R}$ for the approximate location of the shock at time t , defined as

$$\int_{-\infty}^{x_*(t)} u_N(x, t) - u_l \, dx = \int_{x_*(t)}^{\infty} u_r - u_N(x, t) \, dx$$

equality of yellow areas



► $\int_{x_{-m-1/2}}^{x_{m+1/2}} u_N(x, t) \, dx = (x_*(t) + x_{-m-1/2})u_l + (x_{m+1/2} - x_*(t))u_r .$

$$(11.3.3.7) \quad \frac{dx_*}{dt}(t) = \frac{1}{u_l - u_r} \sum_{j \in \mathbb{Z}} \frac{d\mu_j}{dt}(t) = \frac{f(u_l) - f(u_r)}{u_l - u_r} \stackrel{(11.2.4.2)}{=} \dot{s} .$$

Conservation form with consistent numerical flux yields correct “discrete shock speed”
(immune to spurious shock speeds as observed in Ex. 11.3.1.16)

Review question(s) 11.3.3.8 (Spatial semi-discretization of 1D conservation laws)

(Q11.3.3.8.A) [Finite difference method for linear advection] We discretize the linear advection equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 \quad \text{on } \mathbb{R} \times \mathbb{R}^+$$

in space using finite differences on an equidistant mesh. What is the domain of dependence of the solution component $\mu_0(t) \approx u(0, t)$ if $\frac{\partial}{\partial x}$ is approximated by

1. the forward difference quotient,
2. the backward difference quotient,
3. or a symmetric difference quotient.

(Q11.3.3.8.B) [Semi-discrete conservative evolution] State the semi-discrete evolution (“FV-MOL ODE”) that arises from the spatial conservative finite-volume discretization of the 1D scalar conservation law $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0$, if we use a two-point numerical flux. What is the state space of the resulting “infinite ODE” and what is the meaning of the components of its state vector?

(Q11.3.3.8.C) [Finite-volume semi-discretization with periodic boundary conditions] We consider the Cauchy problems for a 1D scalar conservation law $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0$ with 1-periodic initial data u_0 , that is, $u_0(x) = u_0(x - 1)$ for all $x \in \mathbb{R}$, which leads to a 1-periodic entropy solution.

Sketch then implementation of a C++ function

```
template <typename NumFlux>
Eigen::VectorXd fluxdiff_per(const Eigen::VectorXd &mu,
NumFlux &&F);
```

that provides the right-hand-side functional (up to scaling with h^{-1}) for the conservative finite-volume method-of-lines ODE, an equidistant spatial mesh with mesh width $h > 0$, and adjusted to the **1-periodic** setting. The argument **mu** passes the vector of cell averages, whereas the functor **F** provides a **two-point numerical flux**.

△

11.3.4 Numerical Flux Functions

In this section concrete choices of consistent (\rightarrow Def. 11.3.3.5) numerical flux functions will be presented and discussed. We restrict ourselves to 2-point numerical fluxes $F = F(v, w)$, $v \hat{=} \text{"left state"}$, $w \hat{=} \text{"right state"}$, see page 719.

It will turn out that finding appropriate numerical flux functions is by no means straightforward, because both instability and numerical solutions that violate the entropy condition (*to* Section 11.2.6) have to be avoided.

11.3.4.1 Central Flux

A very simple choice for numerical flux functions relies on arithmetic averaging and yields the two **central numerical fluxes**

$$F_1(v, w) := \frac{1}{2}(f(v) + f(w)) , \quad F_2(v, w) := f\left(\frac{1}{2}(v + w)\right) . \quad (11.3.4.1)$$

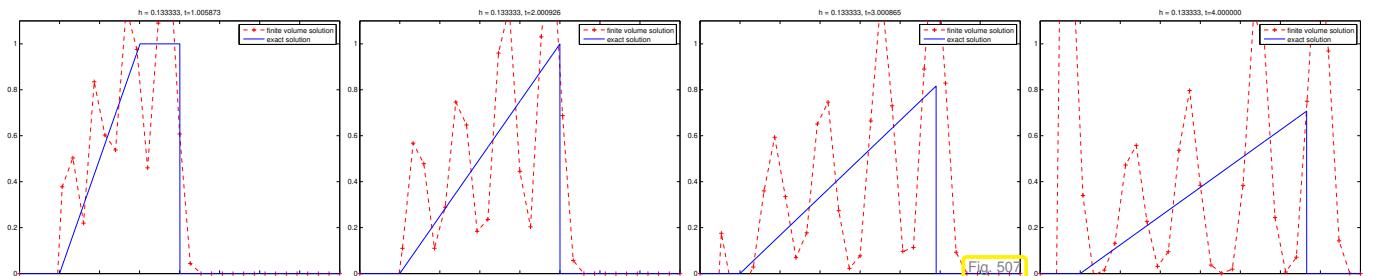
Obviously the 2-point numerical fluxes F_1 and F_2 are consistent according to Def. 11.3.3.5. The resulting spatially semi-discrete schemes are given by, see (11.3.2.8),

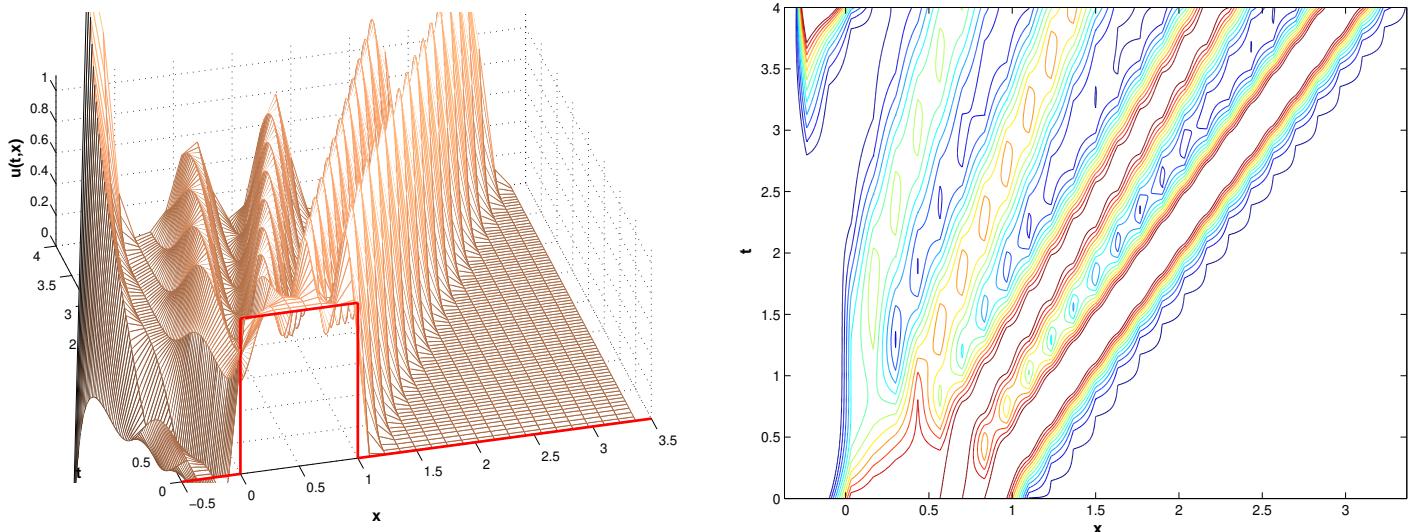
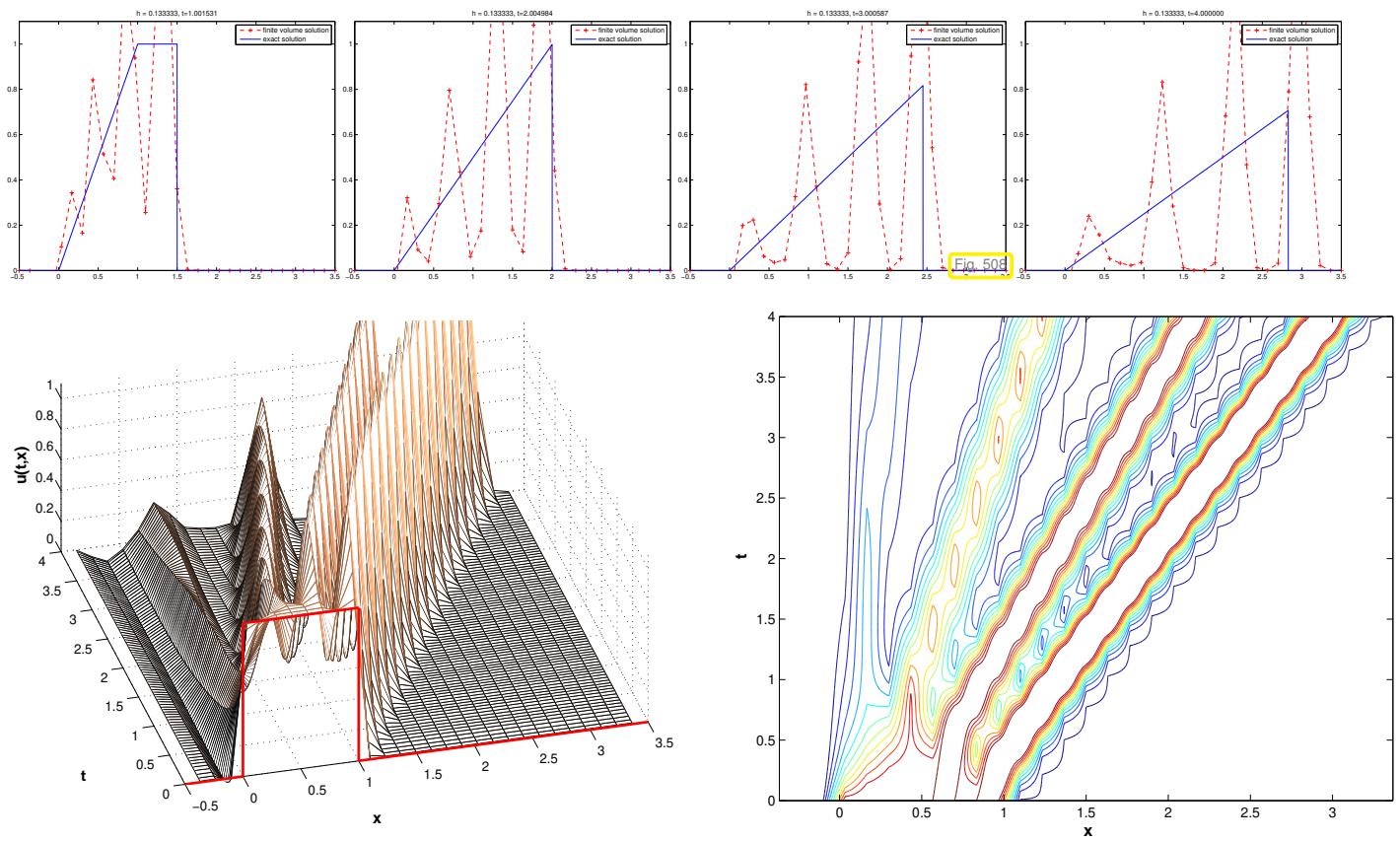
$$\begin{aligned} F_1: \quad \frac{d\mu_j}{dt}(t) &= -\frac{1}{2h}(f(\mu_{j+1}(t)) - f(\mu_{j-1}(t))) , \\ F_2: \quad \frac{d\mu_j}{dt}(t) &= -\frac{1}{h}(f\left(\frac{1}{2}(\mu_j(t) + \mu_{j+1}(t))\right) - f\left(\frac{1}{2}(\mu_j(t) + \mu_{j-1}(t))\right)) . \end{aligned}$$

EXPERIMENT 11.3.4.2 (Central flux for Burgers equation)

- ◆ Cauchy problem for Burgers equation (11.1.3.4) (flux function $f(u) = \frac{1}{2}u^2$) from Ex. 11.2.6.4 (“box” initial data)
- ◆ Spatial finite volume discretization in conservation form (11.3.2.7) with central numerical fluxes according to (11.3.4.1).
- ◆ timestepping based on adaptive explicit Runge-Kutta method `ode45`, see [Hip19, ??] (with absolute tolerance `atol=10^-7`, relative tolerance `reltol=10^-6`)

Fully discrete evolution for central numerical flux F_1 : $h = 0.03$

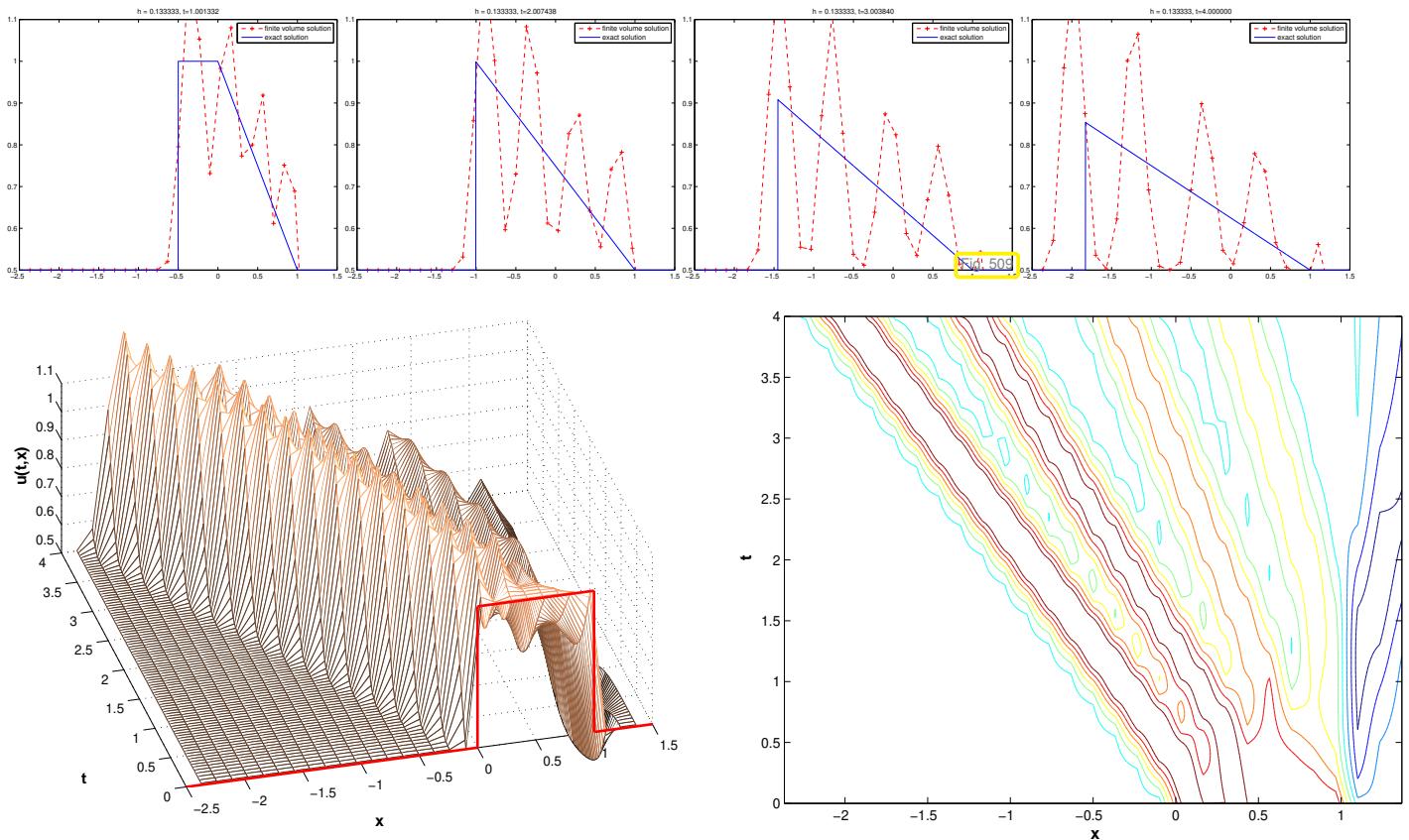
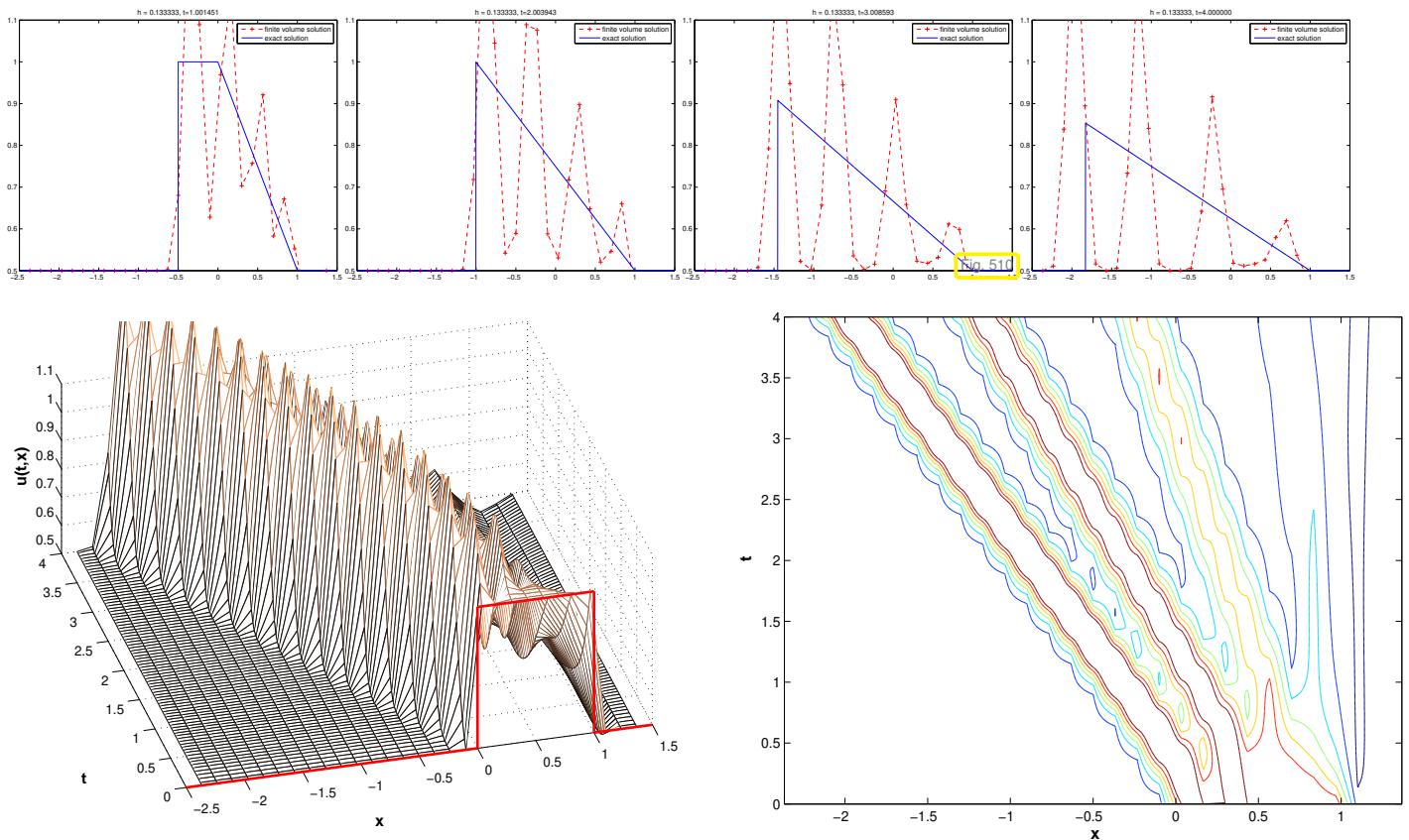


Fully discrete evolution for central numerical flux F_2 : $h = 0.017$ 

We observe massive *spurious oscillations* utterly pollute numerical solution. \square

EXPERIMENT 11.3.4.3 (Central flux for Traffic Flow equation)

- ◆ Cauchy problem for Traffic Flow equation (11.1.2.23) (flux function $f(u) = u(1 - u)$) from Ex. 11.2.6.5 (“box” initial data, $u_0 = \chi_{[0,1]}$)
- ◆ Spatial finite volume discretization in conservation form (11.3.2.7) with central numerical fluxes according to (11.3.4.1).
- ◆ timestepping based on adaptive explicit Runge-Kutta method `ode45` with absolute tolerance 10^{-7} and relative tolerance 10^{-6} .

Fully discrete evolution for central numerical flux F_1 : $h = 0.03$ Fully discrete evolution for central numerical flux F_2 : $h = 0.017$ 

Observation: massive spurious oscillations utterly pollute numerical solution

EXPERIMENT 11.3.4.4 (Central flux for linear advection) In order to see whether the emergence of spurious oscillations is an inherent weakness of central fluxes we apply them to the simplest scalar conservation law, linear advection Section 11.1.1 with constant velocity.

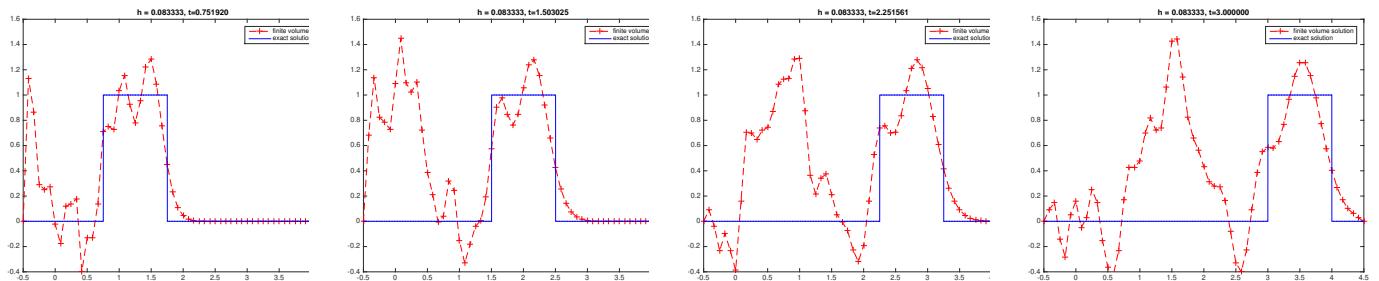
We consider the Cauchy problem (11.1.1.11): constant velocity scalar linear advection, $c = 1$, flux function $f(u) = cu$

$$\frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} = 0 \quad \text{in } \tilde{\Omega} = \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) \quad \forall x \in \mathbb{R} . \quad (11.1.1.11)$$

Finite volume spatial discretization in conservation form (11.3.2.7) with central numerical fluxes from (11.3.4.1):

$$\begin{aligned} F_1(v, w) &:= \frac{1}{2}(f(v) + f(w)) \\ F_2(v, w) &:= f\left(\frac{1}{2}(v + w)\right) \end{aligned} \Rightarrow \frac{d\mu_j}{dt}(t) = -\frac{c}{2h}(\mu_{j+1}(t) - \mu_{j-1}(t)) , \quad j \in \mathbb{Z} . \quad (11.3.4.5)$$

For the numerical experiment we use “box shaped” initial data $u_0 = \chi_{[0,1]}$, an equidistant spatial mesh with meshwidth $h = 0.083$, ode45 adaptive explicit Runge-Kutta timestepping.



Fig

Again, we observe tremendous spurious oscillations that render the computed solution completely useless. □

Remark 11.3.4.6 (Connection with convection-diffusion IBVPs → Chapter 10) Note that the Cauchy problem (11.1.1.11) is an initial value problem for the 1D transport equation (10.3.2.1)!

From Section 10.2.2, (10.2.2.2) we see that the semi-discrete evolution

$$\frac{d\mu_j}{dt}(t) = -\frac{c}{2h}(\mu_{j+1}(t) - \mu_{j-1}(t)) , \quad j \in \mathbb{Z} , \quad (11.3.4.5)$$

agrees with what we obtain from straightforward spatial *linear finite element Galerkin semi-discretization*.

In Section 10.3.1 we learned that this method is *prone to spurious oscillations*, see Exp. 10.3.1.3. This offers an explanation also for its failure for Burgers equation/traffic flow equation, see Exp. 11.3.4.2. □

11.3.4.2 Lax-Friedrichs/Rusanov Flux

§11.3.4.7 (Fighting oscillations with diffusion) According to § 11.1.1.1 the simple linear advection Cauchy problem

$$\frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} = 0 \quad \text{in } \tilde{\Omega} = \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) \quad \forall x \in \mathbb{R} . \quad (11.1.1.11)$$

models heat transport in a fluid moving with constant velocity c .

If u_0 is oscillatory (many local extrema), then these will just be carried along. However, if there is a non-zero heat conductivity $\kappa > 0$, then local extrema of the temperature can be expected to decay exponentially, while they are moving with the flow. For instance, for $c = \kappa = 1$ (dimensionless equations), we get

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} - \frac{\partial^2 u}{\partial x^2} = 0, \quad u_0(x) = \sin(x) \quad \Rightarrow \quad u(x, t) = e^{-t} \sin(x - t), \quad x \in \mathbb{R}, t \geq 0. \quad (11.3.4.8)$$

Hence, let us consider the advection equation with *extra added diffusion*, whose strength can be controlled by the diffusion coefficient $\kappa > 0$,

$$\frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} - \kappa \frac{\partial^2 u}{\partial x^2} = 0, \quad (11.3.4.9)$$

which amounts to a 1D scalar conservation law with the flux function (\rightarrow Rem. 11.2.1.3)

$$f(u) = cu - \kappa \frac{\partial u}{\partial x}. \quad (11.3.4.10)$$

A related numerical flux on an equidistant mesh with meshwidth $h > 0$ can rely on a central flux (11.3.4.1) for the advective part, and on a simple difference quotient approximation for the derivative

$$\begin{aligned} f(u) &= cu - \kappa \frac{\partial u}{\partial x}, \\ F(v, w) &= \frac{c}{2}(v + w) - \kappa \frac{w - v}{h}. \end{aligned}$$

central numerical flux diffusive numerical flux

With this choice of numerical flux the semi-discrete evolution (11.3.2.8) becomes:

$$\dot{\mu}_j(t) + c \frac{\mu_{j+1}(t) - \mu_{j-1}(t)}{2h} + \kappa \frac{-\mu_{j+1}(t) + 2\mu_j(t) - \mu_{j-1}(t)}{h^2} = 0. \quad (11.3.4.11)$$

The computations of (10.2.2.2) show that (11.3.4.11) agrees with the method-of-lines ODE obtained from the standard linear finite element Galerkin discretization of (11.3.4.9) on an equidistant mesh!

Caution: the extra diffusion amounts to a *perturbation* of the Cauchy problem that must be kept as small as possible and, in any case, vanish for $h \rightarrow 0$, which entails $\kappa = \kappa(h)$.

Guideline: prevent diffusive flux from dominating central flux \Rightarrow

$$\kappa = \frac{ch}{2} \quad (11.3.4.12)$$

Remark 11.3.4.13 (Connection with artificial viscosity \rightarrow Section 10.2.2.2) As already pointed out in Rem. 11.3.4.6, the developments in this section are closely connected with similar considerations in Section 10.2.2, Section 10.3.1 in the context of stable spatial discretization of convection-diffusion problems (11.3.4.9).

In Section 10.2.2.2 we saw that *artificial diffusion* cures instability of central difference quotients. In (10.2.2.8) we found a new interpretation of the *upwind* discretization based on one-sided difference quotients:

$$\begin{aligned} \frac{\partial u}{\partial t} &+ c \frac{\partial u}{\partial x} = 0 \quad \text{in } \mathbb{R} \times [0, T], \\ \Updownarrow & \\ \frac{\partial u}{\partial t} + \left(\frac{ch}{2} \right) \underbrace{\frac{-\mu_{j-1} + 2\mu_j - \mu_{j+1}}{h^2}}_{\hat{\triangleq} \text{ difference quotient for } \frac{d^2 u}{dx^2}} + \underbrace{c \frac{\mu_{j+1} - \mu_{j-1}}{2h}}_{\hat{\triangleq} \text{ difference quotient for } c \frac{du}{dx}} &= 0, \quad j \in \mathbb{Z}. \end{aligned}$$

Can this be rewritten in conservation form (11.3.2.7)? YES!

$$(\text{ch}/2) \frac{-\mu_{j-1} + 2\mu_j - \mu_{j+1}}{h^2} + c \frac{\mu_{j+1} - \mu_{j-1}}{2h} = \frac{1}{h} (F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)),$$

with $F(v, w) := \frac{c}{2}(v + w) - \frac{c}{2}(w - v)$.

central numerical flux h -weighted diffusive/viscous numerical flux

Recall from Rem. 11.2.1.3: the flux function $f(u) = -\frac{\partial u}{\partial x}$ models diffusion. Hence, the diffusive numerical flux amounts to a central finite difference discretization of the partial derivative in space:

$$-\frac{\partial u}{\partial x}(x, t) \Big|_{x=x_{j+1/2}} \approx -\frac{1}{h} (u(x_{j+1}, t) - u(x_j, t)).$$

Thus, starting from upwind discretization, we also arrive at the scheme heuristically derived in § 11.3.4.7.

How to adapt the idea of extra diffusion to general scalar conservation laws? A simple manipulation connects these with linear advection:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = \frac{\partial u}{\partial t} + f'(u) \frac{\partial u}{\partial x} = 0$$

local speed of transport $\leftrightarrow c$

However, the speed $f'(u)$ of transport will depend on x , which suggests that the strength of artificial diffusion should vary. We choose it according to (11.3.4.12), but large enough to fit the maximal local velocity: we set $k = \frac{h}{2} \max\{|f'(u)| : \min\{v, w\} \leq u \leq \max\{v, w\}\}$ in the diffusive part of the numerical flux.

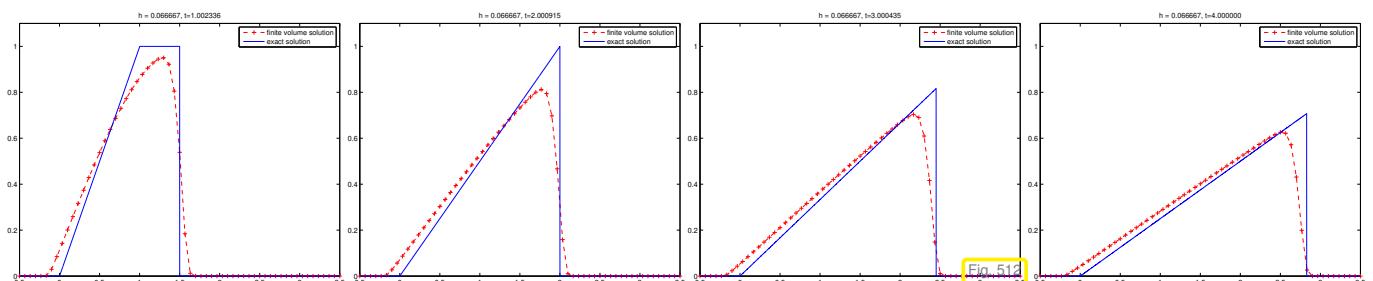
► (local) Lax-Friedrichs/Rusanov flux

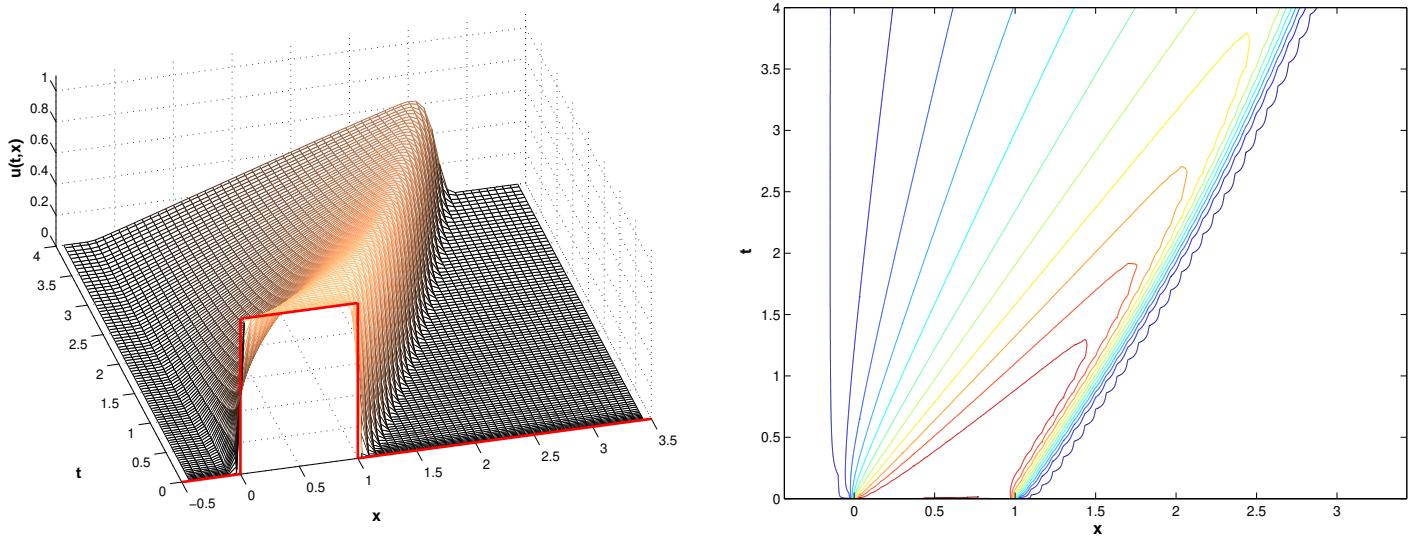
$$F_{\text{LF}}(v, w) = \frac{1}{2}(f(v) + f(w)) - \frac{1}{2}(w - v) \cdot \max_{\min\{v, w\} \leq u \leq \max\{v, w\}} |f'(u)|.$$

The next two experiments investigate the performance of the (local) Lax-Friedrichs/Rusanov numerical flux for our model non-linear scalar conservation laws.

EXPERIMENT 11.3.4.17 (Lax-Friedrichs flux for Burgers equation)

- ☞ Same setting and conservative discretization as in Exp. 11.3.4.2
- ☞ Numerical flux function: Lax-Friedrichs flux (11.3.4.16)



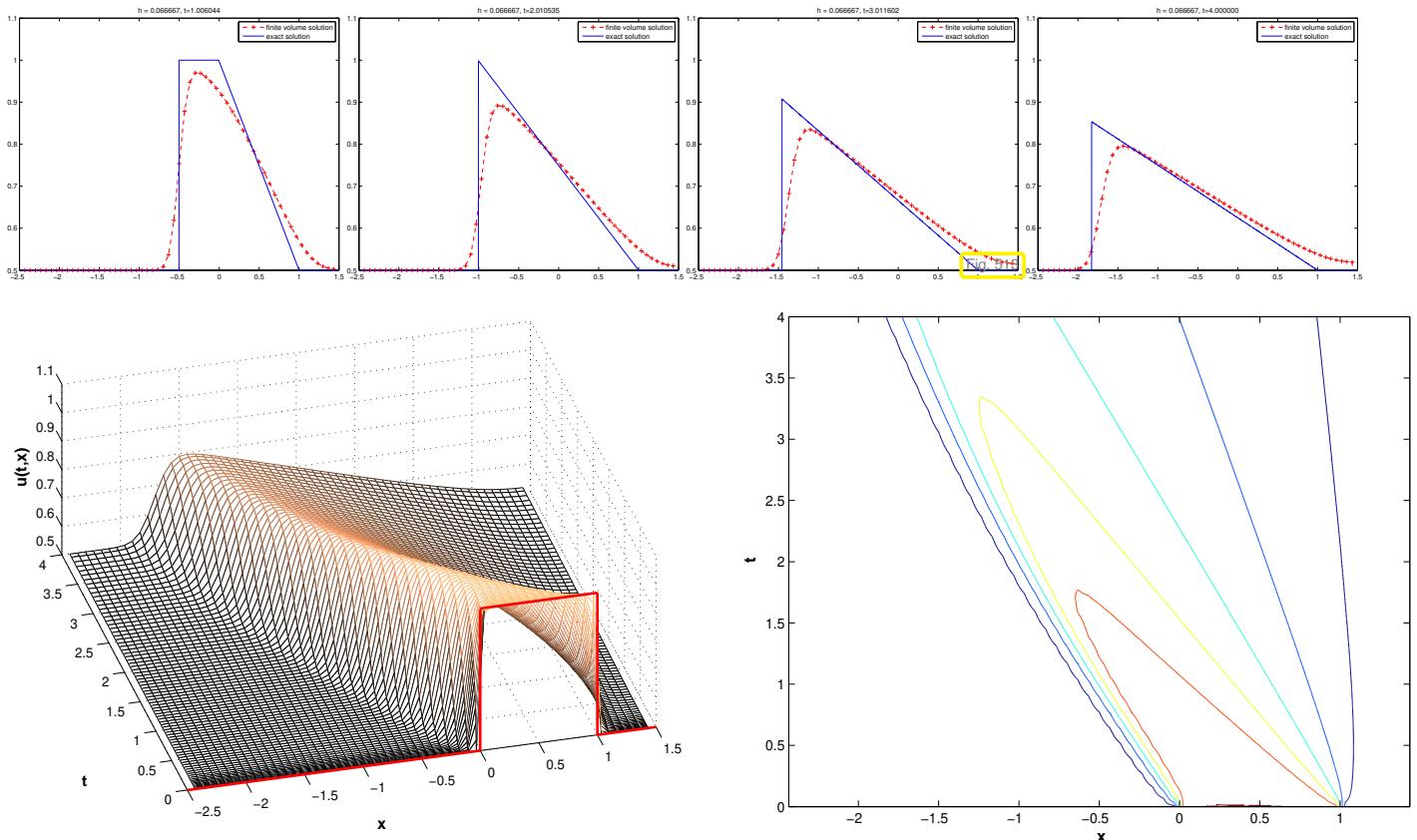


Observation: spurious oscillations are suppressed completely, qualitatively good resolution of both shock and rarefaction.

However, an undesirable effect of artificial diffusion is also evident. It leads to a *smearing* of the shock, cf. the discussion in Ex. 10.2.2.23. Instead of a jump the numerical solution suggests a smooth transition between two states. □

EXPERIMENT 11.3.4.18 (Lax-Friedrichs flux for traffic flow equation) same setting and conservative discretization as in Exp. 11.3.4.2

Numerical flux function: Lax-Friedrichs flux (11.3.4.16)



Same observations as in Exp. 11.3.4.17: no spurious oscillations, qualitatively correct solution, but strong smearing of shock. □

11.3.4.3 Upwind Flux

Another idea for stable spatial discretization of stationary transport in Section 10.2.2.1 (“upwind quadrature”):

“**upwinding**” = obtain information from where transport brings it

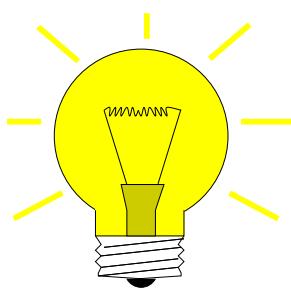
☞ remedy for ambiguity of evaluation of discontinuous gradient in upwind quadrature

Owing to the discontinuity of u_N at $x_{k+1/2}$, ambiguity is also faced in the evaluation of the fluxes $f(u_N(x_{j+1/2}), t), f(u_N(x_{j-1/2}), t)$, see (11.3.2.5), which forced us to introduce numerical flux functions in (11.3.2.7). We may also seek to select the value of u_N from that side of $x_{k+1/2}$ where information comes from. In light of Rem. 11.2.2.9 we should examine the direction of the characteristic running through $(x_{k+1/2}, t)$.

Def. 11.2.2.3, (11.3.4.15) ➤ The local slope of the characteristic curve (velocity of transport) at $(x, t) \in \tilde{\Omega}$ is given by $f'(u(x, t))$.



local velocity of transport $f'(u_N(x_{k+1/2}, t))$ is ambiguous too!



Idea: There is a “velocity of propagation” even at discontinuities of u !
Deduce it from **Rankine-Hugoniot jump condition** (11.2.4.2).

► local velocity of transport $= \begin{cases} f'(u) & \text{for unique state, } u = u_l = u_r \\ \frac{f(u_r) - f(u_l)}{u_r - u_l} & \text{at discontinuity.} \end{cases}$

$(u_l, u_r) \hat{=} \text{states to left and right of discontinuity})$

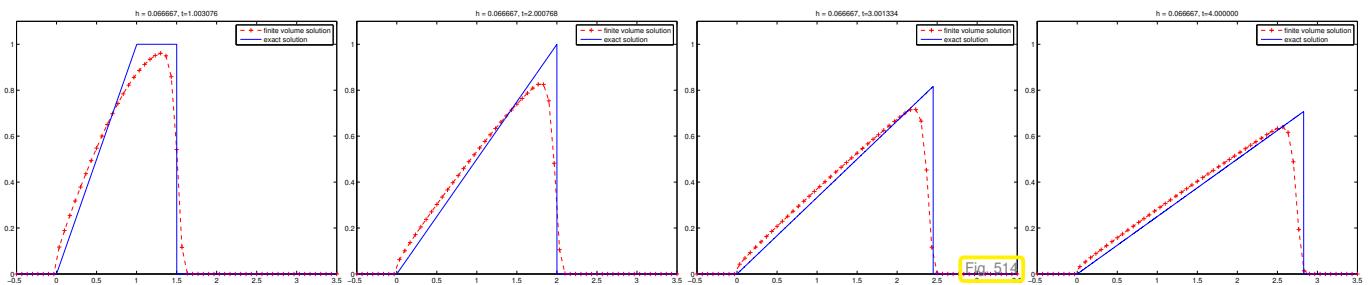
► **upwind numerical flux** for scalar conservation law with flux function f :

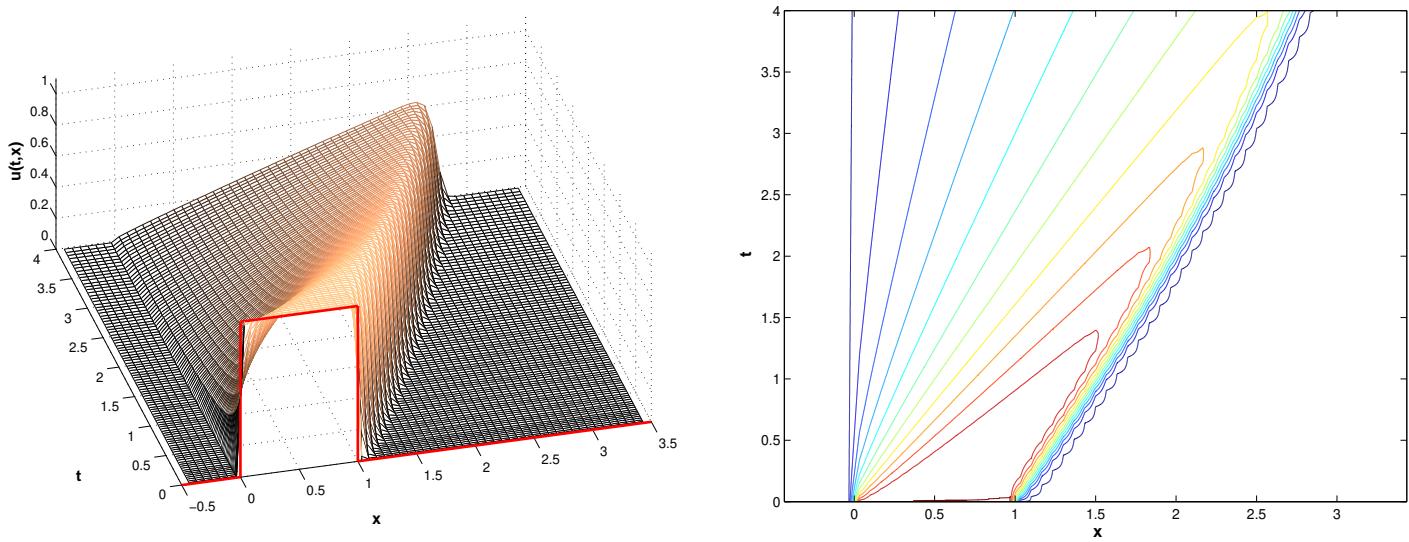
$$F_{uw}(v, w) = \begin{cases} f(v) & , \text{if } \dot{s} \geq 0, \\ f(w) & , \text{if } \dot{s} < 0, \end{cases} \quad \dot{s} := \begin{cases} \frac{f(w) - f(v)}{w - v} & \text{for } v \neq w, \\ f'(v) & \text{for } v = w. \end{cases} \quad (11.3.4.19)$$

Now we investigate empirically the performance of the upwind numerical flux for our model non-linear scalar conservation laws.

EXPERIMENT 11.3.4.20 (Upwind flux for Burgers equation) ☞ same setting and conservative discretization as in Exp. 11.3.4.2

☞ Numerical flux function: upwind flux (11.3.4.19)

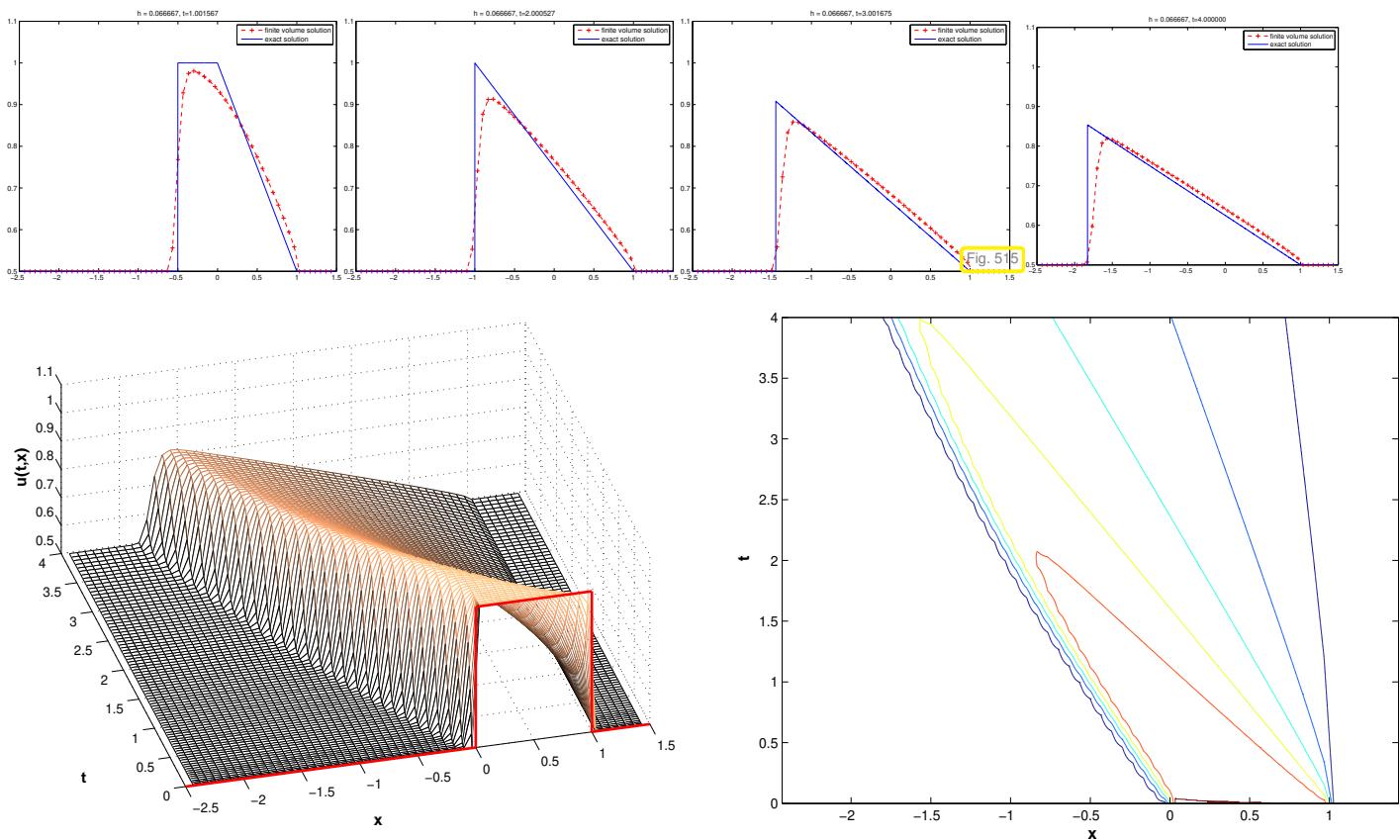




EXPERIMENT 11.3.4.21 (Upwind flux for traffic flow simulation)

- ☞ Conservative finite volume discretization of Cauchy problem for traffic flow equation (11.1.2.23), flux function $f(u) = u(1-u)$
- ☞ Equidistant spatial mesh with meshwidth $h = 0.03$, adaptive explicit Runge-Kutta timestepping (MATLAB `ode45`)
- ☞ Numerical flux function: upwind flux (11.3.4.19)
- ☞ “Box shaped” initial data $u_0(x) = \begin{cases} 1 & \text{for } 0 \leq x \leq 1, \\ 0.5 & \text{elsewhere.} \end{cases}$

The solution will comprise a stationary shock and a rarefaction fan, which will merge eventually.



We observe a satisfactory resolution of the shock and the rarefaction fan.

EXAMPLE 11.3.4.22 (Upwind flux and transsonic rarefaction) In this example we will witness a situation in which the use of the upwind numerical flux function produces a non-physical shock.

We consider the Cauchy problem (11.2.2.1) for Burgers equation (11.1.3.4), i.e., $f(u) = \frac{1}{2}u^2$ and initial data

$$u_0(x) = \begin{cases} -1 & \text{for } x < 0 \text{ or } x > 1, \\ 1 & \text{for } 0 < x < 1. \end{cases} \quad (11.3.4.23)$$

The analytic solution for this Cauchy problem is given in Ex. 11.2.6.4.

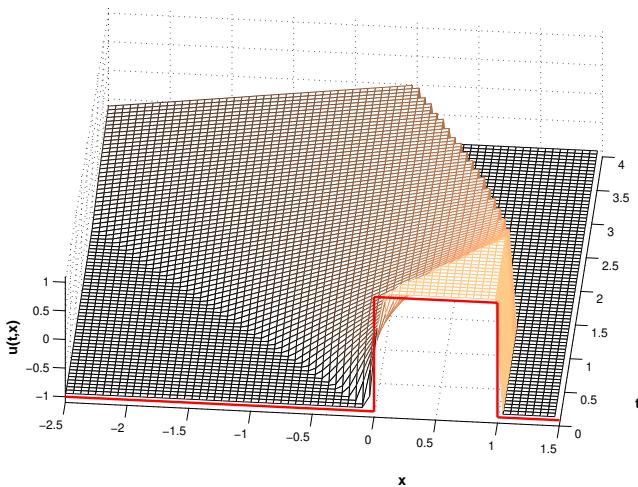


Fig. 516

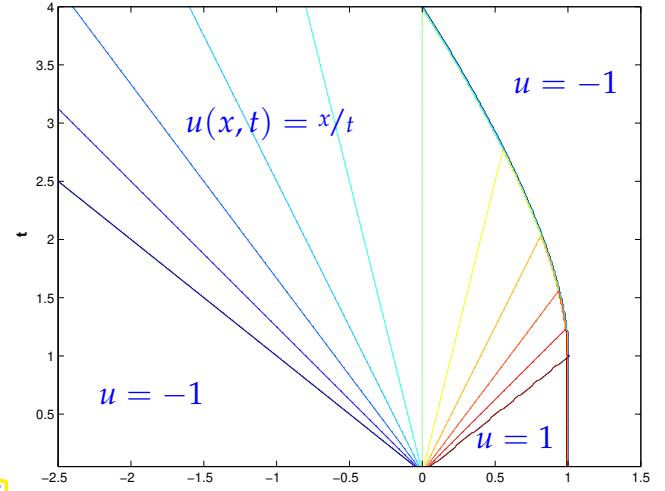


Fig. 517

There is a related Cauchy problem (11.2.2.1) for the traffic flow equation (11.1.2.23), i.e., $f(u) = u(1-u)$ and initial data

$$u_0(x) = \begin{cases} 0 & \text{for } x < 0 \text{ or } x > 1, \\ 1 & \text{for } 0 < x < 1. \end{cases} \quad (11.3.4.24)$$

Its analytic solution is plotted in Fig. 519 and given in Fig. 520.

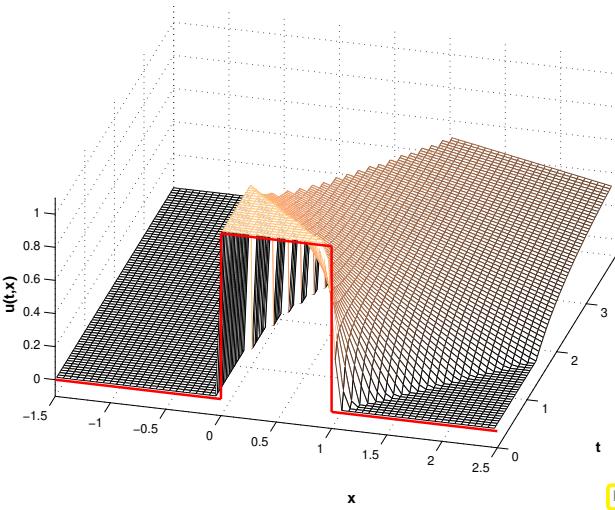


Fig. 518

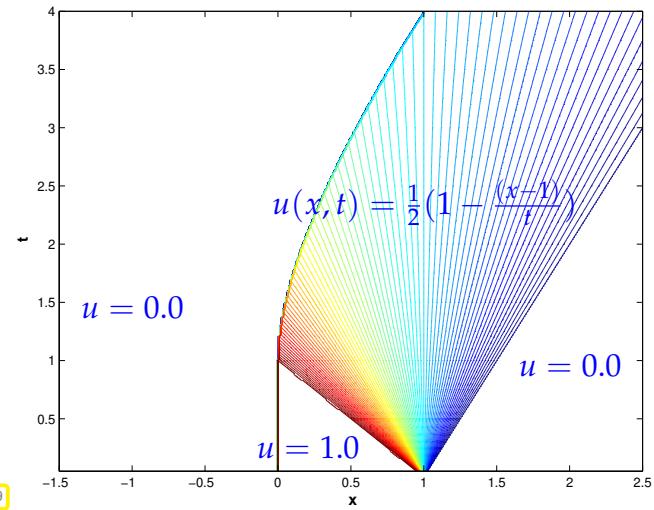
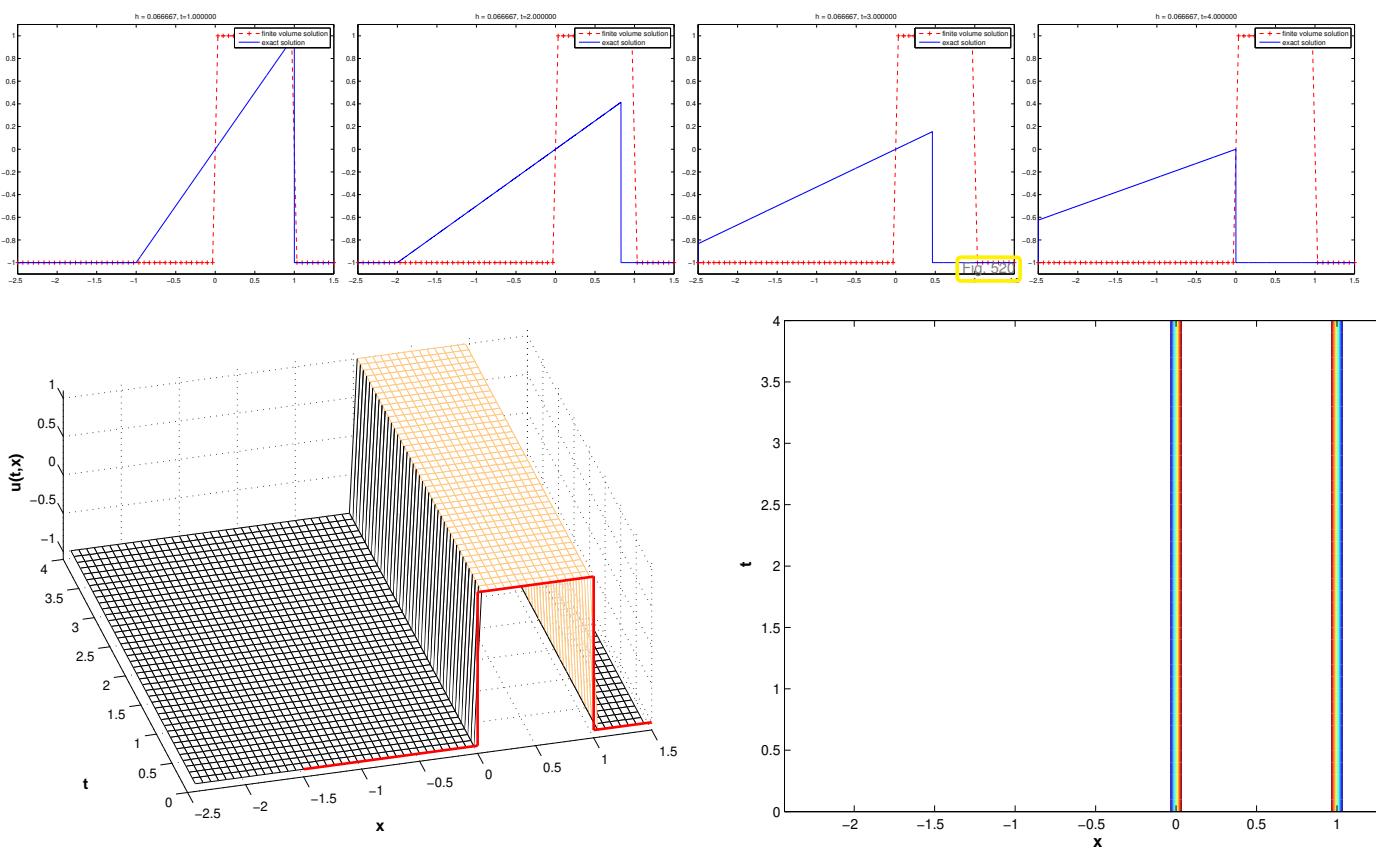


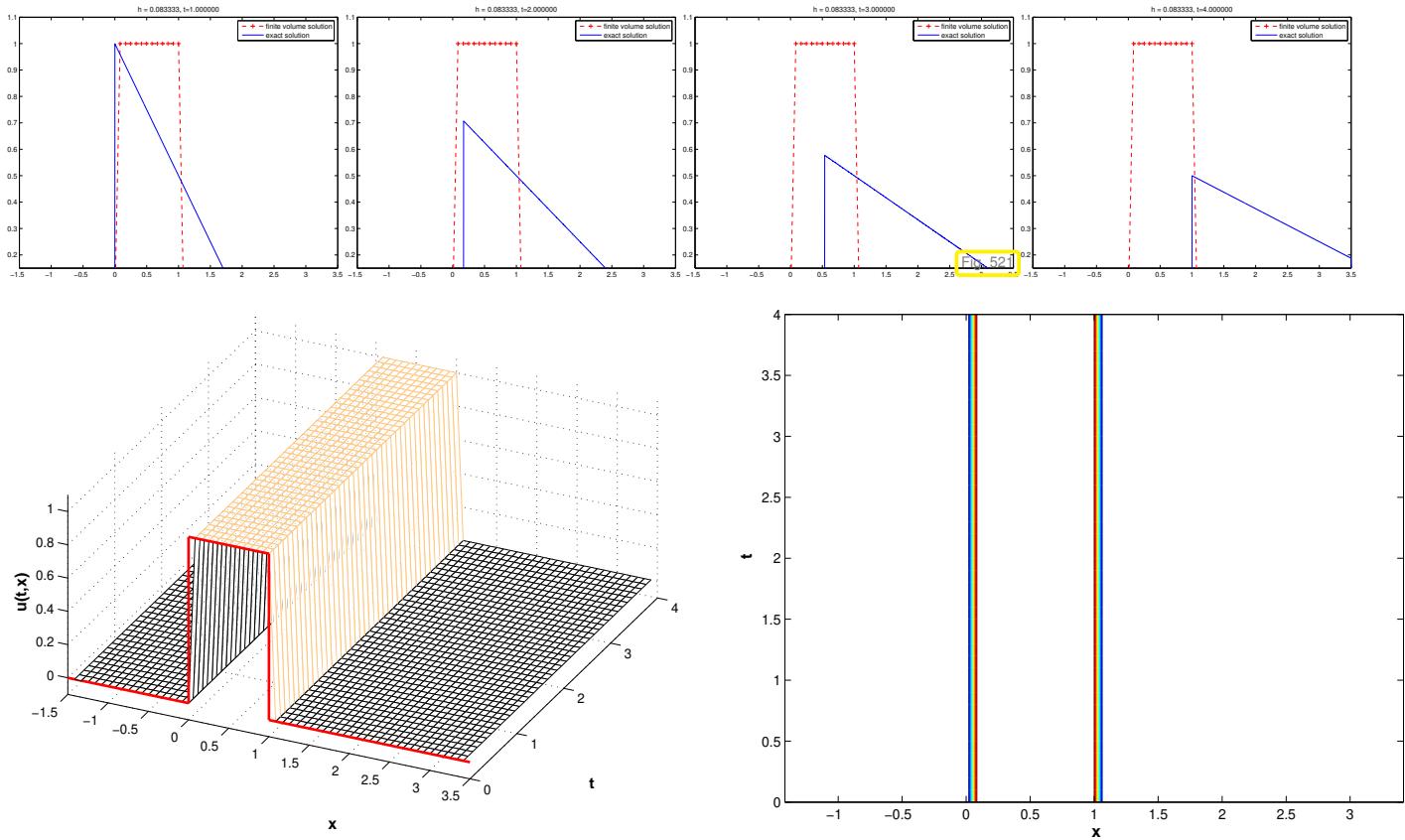
Fig. 519

The *entropy solution* (\rightarrow Section 11.2.6) of these Cauchy problems features a **transonic rarefaction fan** at $x = 1$: this is a rarefaction solution (\rightarrow Lemma 11.2.5.10) whose “edges” move in opposite directions.

Burgers’ equation, initial density (11.3.4.23): numerical solution with finite volume method with upwind flux (11.3.4.19).



Traffic flow equation, initial data (11.3.4.24): numerical solution with finite volume method with upwind flux (11.3.4.19).



Conservative finite volume discretization with upwind flux produces (stationary) *expansion shock* instead of transonic rarefaction!

Section 11.2.6: this is a weak solution, but it violates the entropy condition, “non-physical shock”. □

EXAMPLE 11.3.4.25 (Upwind flux: Convergence to expansion shock) In Ex. 11.3.4.22 we have seen that the use of the upwind flux can make a conservative finite volume discretization converge to non-physical expansion shocks. In this example simple computations will show how this can happen. The setting is the following:

- ◆ Cauchy problem (11.2.2.1) for Burgers equation (11.1.3.4), i.e., $f(u) = \frac{1}{2}u^2$
- ◆ $u_0(x) = 1$ for $x > 0$, $u_0(x) = -1$ for $x < 0$
 - entropy solution = rarefaction wave (\rightarrow Lemma 11.2.5.10)
- ◆ FV in conservation form, upwind flux (11.3.4.19), on equidistant grid, $x_j = (j + \frac{1}{2})h$, meshwidth $h > 0$
 - initial nodal values $\mu_j(0) = \begin{cases} -1 & \text{for } j < 0, \\ 1 & \text{for } j \geq 0. \end{cases}$
 - Semi-discrete evolution equation:
$$\frac{d\mu_j}{dt}(t) = -\frac{1}{2h} \cdot \begin{cases} \mu_{j+1}^2(t) - \mu_j^2(t) & \text{for } j \geq 0, \\ \mu_j^2(t) - \mu_{j-1}^2(t) & \text{for } j < 0. \end{cases}$$
 - $\mu_j(t) = \mu_j(0)$ for all t ➤ for $h \rightarrow 0$, convergence to entropy violating expansion shock !
 - conservative finite volume method may converge to non-physical weak solutions ! □

11.3.4.4 Godunov Flux

Ex. 11.3.4.22 strikingly illustrated the failure of the a conservative finite volume discretization based on upwind flux to deal with transsonic rarefactions. In this section a different perspective on upwind fluxes will suggest a remedy.

(The following discussion is for *convex* flux functions only, that occur, for instance in Burgers equation (11.1.3.4). The reader is encouraged to figure out the modifications necessary if the flux function is concave, as in the traffic flow equation (11.1.2.23).)

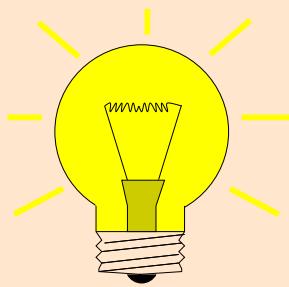
The upwind flux (11.3.4.19) is a numerical flux of the form

$F(v, w) = f(u^\downarrow(v, w))$

with an **intermediate state** $u^\downarrow(v, w) \in \mathbb{R}$.

For the upwind flux the intermediate state is not really “intermediate”, but coincides with one of the states v, w depending on the sign of the “local shock speed” $s := \frac{f(w) - f(v)}{w - v}$.

§11.3.4.26 (Local Riemann problems) We note that the intermediate state for the upwind numerical flux at the dual cell boundary $x_{j+1/2}$ agrees with the state produced for short times at $x_{j+1/2}$ by an *all-shock solution* of the conservation law with initial data $u_N(\cdot, t)$, with u_N the M -piecewise constant function defined by the dual cell averages according to (11.3.2.3). This solution may feature non-physical (expansion) shocks, while rarefaction waves are missing. For this reason the simple upwind flux fails to capture rarefaction waves as we have witnessed in Ex. 11.3.4.22.



Idea: obtain suitable intermediate state as

$$u^\downarrow(v, w) = \psi(0), \quad (11.3.4.27)$$

where $\bar{u}(x, t) = \psi(x/t)$ solves the Riemann problem (\rightarrow Section 11.2.5)

$$\frac{\partial \bar{u}}{\partial t} + \frac{\partial}{\partial x} f(\bar{u}) = 0, \quad \bar{u}(x, 0) = \begin{cases} v & \text{, for } x < 0, \\ w & \text{, for } x \geq 0. \end{cases} \quad (11.3.4.28)$$

Remember Lemma 11.2.5.4, Lemma 11.2.5.10, and the reasons why we can count on the entropy solution of the Riemann problem to be a **similarity solution** of the form $\bar{u}(x, t) = \psi(x/t)$, see page 703.

We focus on $f : \mathbb{R} \mapsto \mathbb{R}$ strictly convex & smooth (e.g. Burgers equations (11.1.3.4))

\blacktriangleright Riemann problem (11.3.4.28) (\rightarrow Section 11.2.5) has the **entropy solution** (\rightarrow Section 11.2.6):

- ① If $v > w$ \blacktriangleright discontinuous solution, **shock** (\rightarrow Lemma 11.2.5.4)

$$\bar{u}(t, x) = \begin{cases} v & \text{if } x < \dot{s}t, \\ w & \text{if } x > \dot{s}t, \end{cases} \quad \dot{s} = \frac{f(v) - f(w)}{v - w}. \quad (11.3.4.29)$$

- ② If $v \leq w$ \blacktriangleright continuous solution, **rarefaction wave** (\rightarrow Lemma 11.2.5.10)

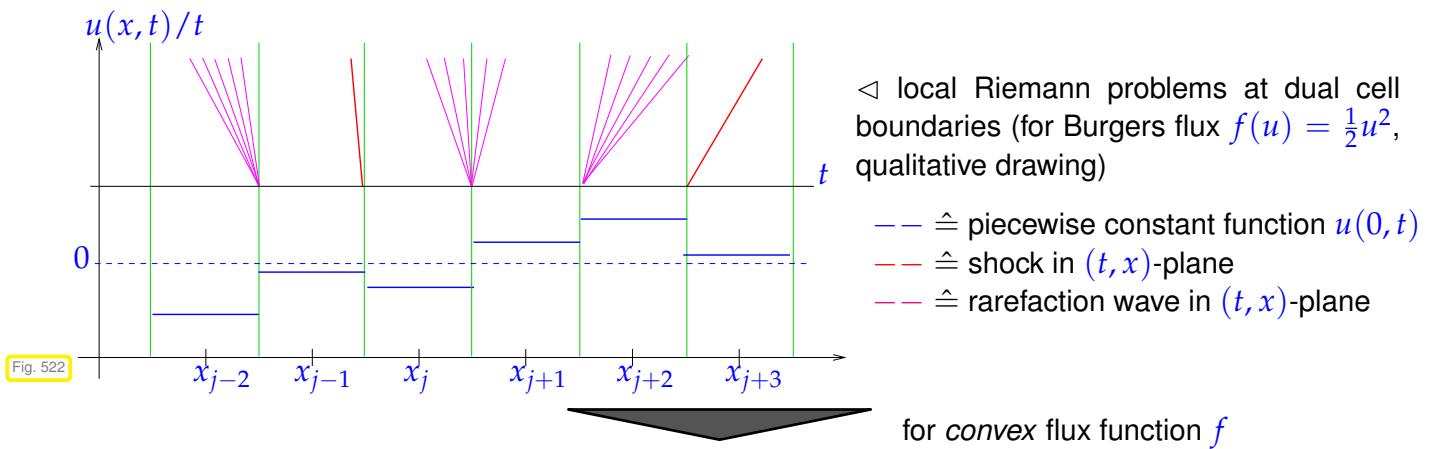
$$\bar{u}(t, x) = \begin{cases} v & \text{if } x < f'(v)t, \\ g(x/t) & \text{if } f'(v) \leq x/t \leq f'(w), \\ w & \text{if } x > f'(w)t, \end{cases} \quad g := (f')^{-1}. \quad (11.3.4.30)$$

- \blacktriangleright Also from these formulas we see that all weak solutions of a Riemann problem are of the form $u(x, t) = \psi(x/t)$ (similarity solution) with a suitable function ψ , which is
 - ◆ piecewise constant with a jump at $\dot{s} := \frac{f(w) - f(v)}{w - v}$ for a shock solution (11.3.4.29),
 - ◆ the continuous function (in the case of strictly convex flux function f)

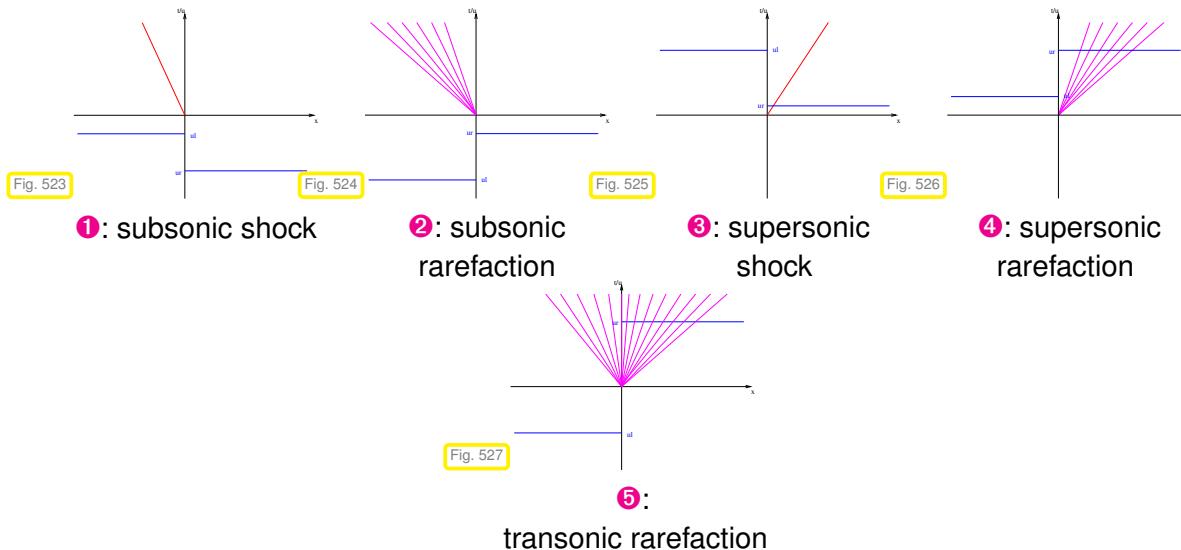
$$\psi(\xi) := \begin{cases} v & \text{, if } \xi < f'(v), \\ (f')^{-1}(\xi) & \text{, if } f'(v) < \xi < f'(w), \\ w & \text{, if } \xi > f'(w), \end{cases}$$

provided that $w > v$ = situation of a rarefaction solution (11.3.4.30), see Lemma 11.2.5.10.

A graphical illustration of the various local Riemann solutions that can be found at dual cell boundaries is given next:



$$u^\downarrow(v, w) = \begin{cases} w & , \text{if } v = w \text{ (constant solution)}, \\ v & , \text{if } v > w \wedge \dot{s} < 0 \text{ (shock ①)}, \\ (f')^{-1}(0) & , \text{if } v < w \wedge f'(w) < 0 \text{ (rarefaction ②)}, \\ & , \text{if } v > w \wedge \dot{s} > 0 \text{ (shock ③)}, \\ & , \text{if } v < w \wedge f'(v) > 0 \text{ (rarefaction ④)}, \\ & , \text{if } v < w \wedge f'(v) \leq 0 \leq f'(w) \text{ (rarefaction ⑤)}. \end{cases} \quad (11.3.4.31)$$



§11.3.4.32 (Formulas for Godunov numerical flux function) A detailed analysis of (11.3.4.31) yields fairly explicit formulas:

$$v > w \quad (\text{shock case}): \quad f(u^\downarrow(v, w)) = \begin{cases} f(v) & , \text{if } \frac{f(w) - f(v)}{w - v} > 0 \Leftrightarrow f(w) < f(v), \\ f(w) & , \text{if } \frac{f(w) - f(v)}{w - v} \leq 0 \Leftrightarrow f(w) \geq f(v). \end{cases}$$

► $f(u^\downarrow(v, w)) = \max\{f(v), f(w)\}.$

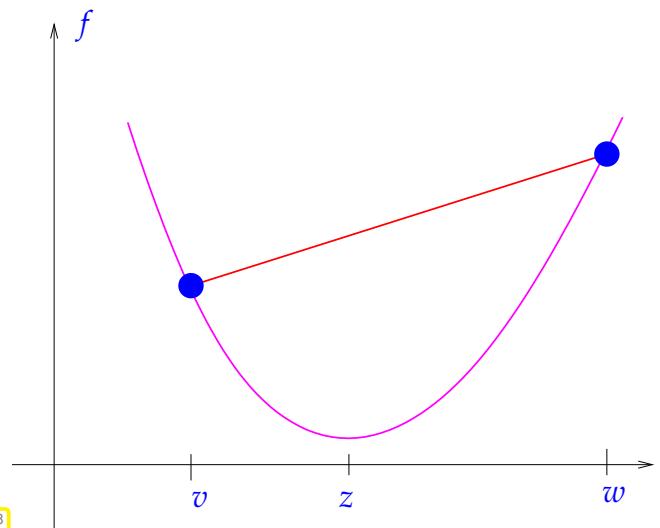
For a convex flux function f :

$$v < w \Rightarrow f'(v) \leq \frac{f(w) - f(v)}{w - v} \leq f'(w).$$

► For $v < w$ (rarefaction case)

$$f(u^\downarrow(v, w)) = \begin{cases} f(v) & , \text{ if } f'(v) > 0, \\ f(z) & , \text{ if } f'(v) < 0 < f'(w), \\ f(w) & , \text{ if } f'(w) < 0, \end{cases}$$

where $f'(z) = 0 \Leftrightarrow f$ has a global minimum in z .



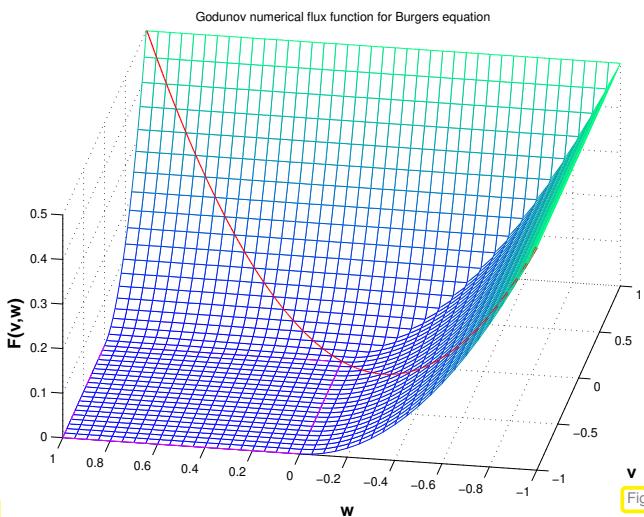
2-point numerical flux function according to (11.3.4.27) and (11.3.4.28): **Godunov numerical flux**

Using general Riemann solution (11.2.6.3) we get for **any** flux function:

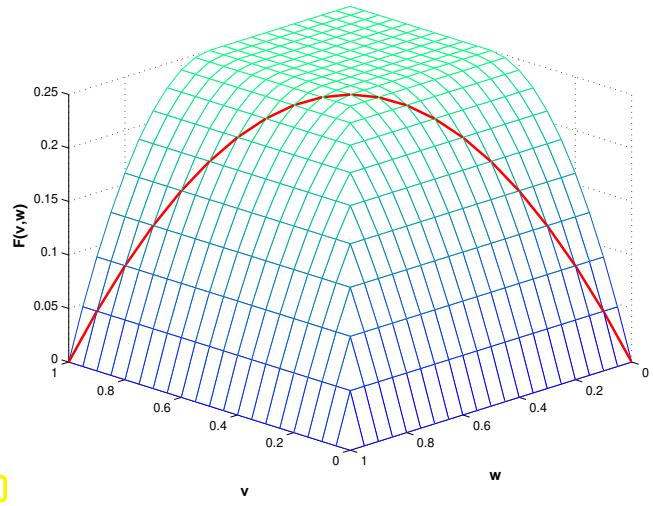
► Godunov numerical flux function

$$F_{GD}(v, w) = \begin{cases} \min_{v \leq u \leq w} f(u) & , \text{ if } v < w, \\ \max_{w \leq u \leq v} f(u) & , \text{ if } w \leq v. \end{cases} \quad (11.3.4.33)$$

Obviously the Godunov numerical flux is consistent according to Def. 11.3.3.5.

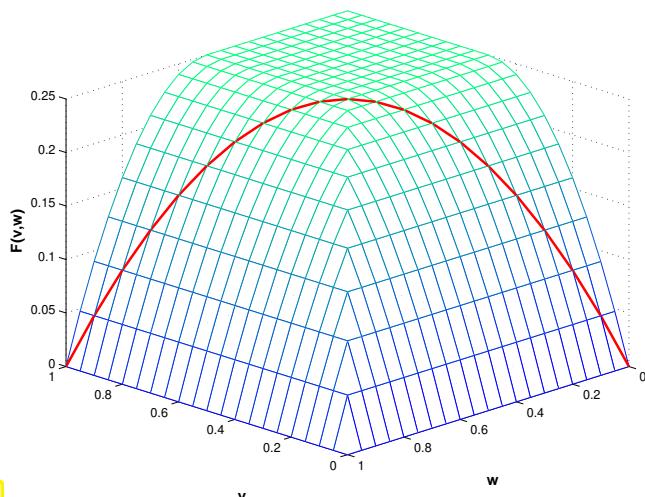
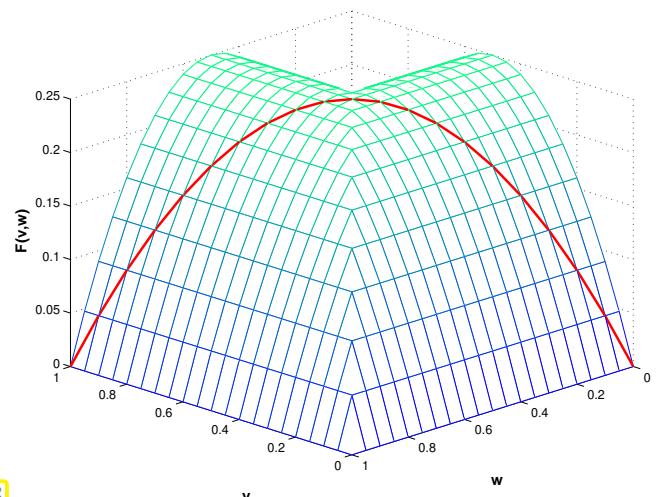


for Burgers' equation (11.1.3.4)



for traffic flow equation (11.1.2.23)

Remark 11.3.4.34 (Upwind flux and expansion shocks) For traffic flow equation (11.1.2.23) ($f(u) = u(1-u)$)

Godunov flux $F_{DG}(v, w)$ upwind flux $F_{UW}(v, w)$

$F_{uw}(v, w) = F_{GD}(v, w)$, except for the case of *transsonic rarefaction*!

(transsonic rarefaction = rarefaction fan with edges moving in opposite direction, see Ex. 11.3.4.22)

What does the upwind flux $F_{uw}(v, w)$ from (11.3.4.19) yield in the case of transsonic rarefaction?

If f convex, $v < w$, $f'(v) < 0 < f'(w)$,

$$\Rightarrow F_{uw}(v, w) = f(\psi(0)),$$

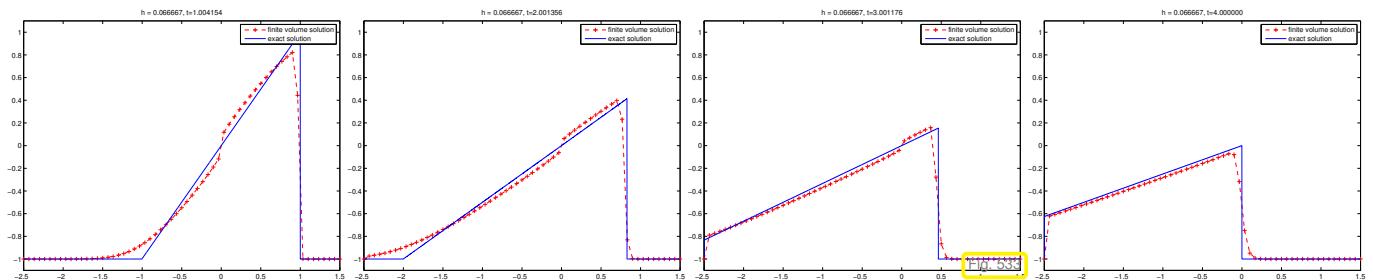
where $u(x, t) = \psi(x/t)$ is a non-physical *entropy-condition violating* (\rightarrow Def. 11.2.6.1) expansion shock weak solution of (11.3.4.28).

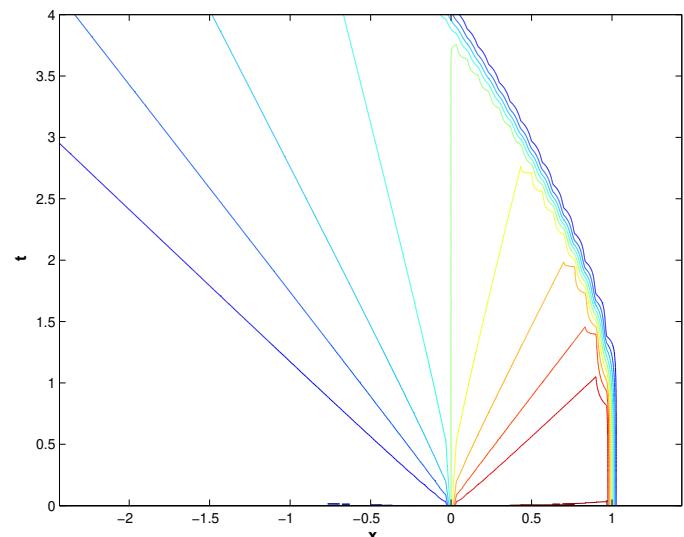
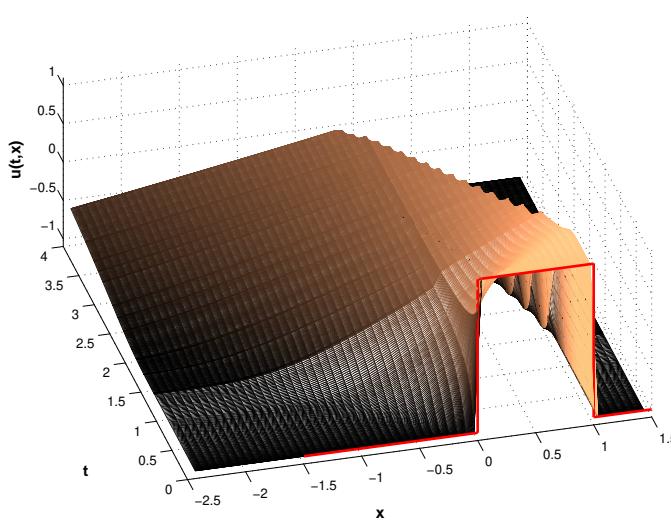
Upwind flux treats transsonic rarefaction as expansion shock!

➤ Explanation for observation made in Ex. 11.3.4.22.

EXPERIMENT 11.3.4.35 (Godunov flux for Burgers equation) ↗ same setting and conservative discretization as in Ex. 11.3.4.22

☞ Numerical flux function: Godunov numerical flux (11.3.4.33)

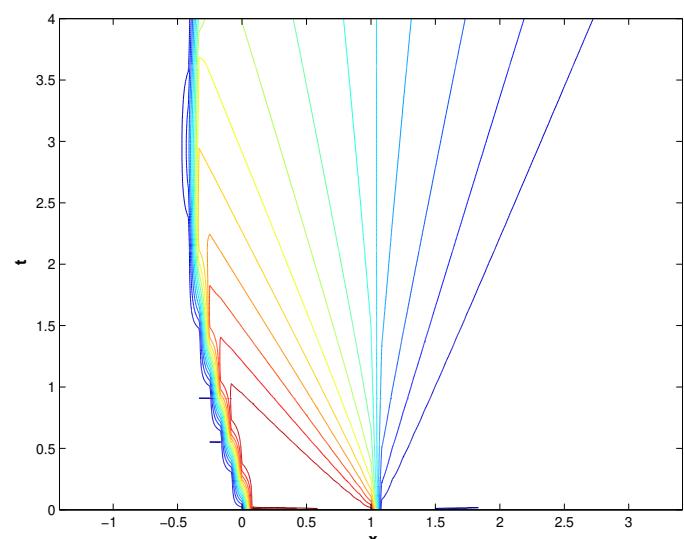
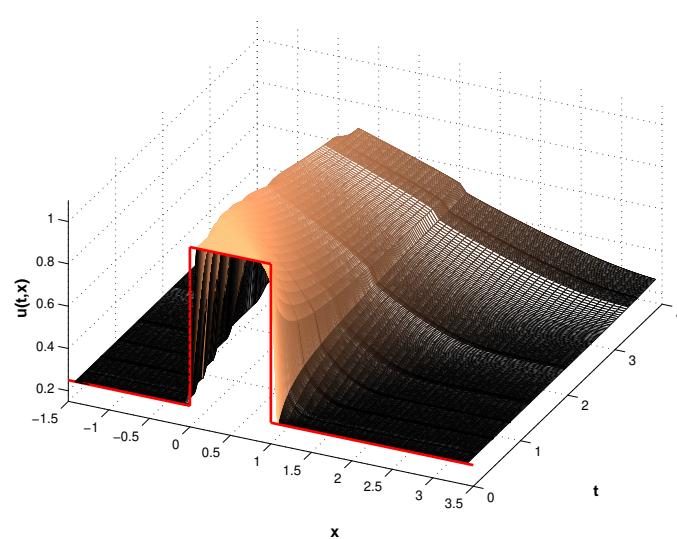
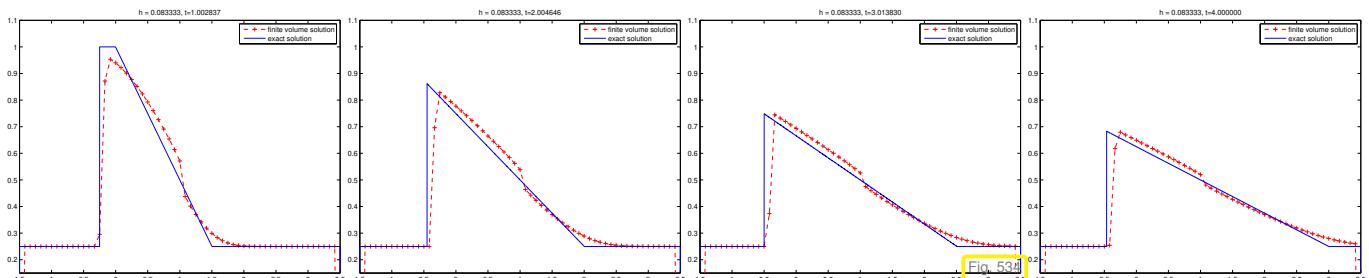




Observation: Transonic rarefaction captured by discretization, but small remnants of an expansion shock still observed. □

EXPERIMENT 11.3.4.36 (Godunov flux for traffic flow equation) ☞ same setting and conservative discretization as in Ex. 11.3.4.22

☞ Numerical flux function: Godunov numerical flux (11.3.4.33)



Observation: Transonic rarefaction captured by discretization, but small remnants of an expansion shock still observed. □

Review question(s) 11.3.4.37 (Numerical flux functions)

(Q11.3.4.37.A) State concrete formulas for the following **two-point numerical flux functions**, when applied

to a 1D scalar conservation law with flux function

1. $f(u) = \exp(u)$, $u \in \mathbb{R}$,
2. $f(u) = u^2$, $u \in \mathbb{R}$:

① Central flux

$$F_C(v, w) = f\left(\frac{1}{2}(v + w)\right), \quad v, w \in \mathbb{R},$$

② (local) Lax-Friedrichs flux:

$$F_{LF}(v, w) = \frac{1}{2}(f(v) + f(w)) - \frac{1}{2}(w - v) \cdot \max_{\min\{v, w\} \leq u \leq \max\{v, w\}} |f'(u)|, \quad (11.3.4.16)$$

③ upwind flux

$$F_{uw}(v, w) = \begin{cases} f(v) & , \text{if } \dot{s} \geq 0, \\ f(w) & , \text{if } \dot{s} < 0, \end{cases} \quad \dot{s} := \begin{cases} \frac{f(w) - f(v)}{w - v} & \text{for } v \neq w, \\ f'(v) & \text{for } v = w, \end{cases} \quad (11.3.4.19)$$

④ Godunov flux:

$$F_{GD}(v, w) = \begin{cases} \min_{v \leq u \leq w} f(u) & , \text{if } v < w, \\ \max_{w \leq u \leq v} f(u) & , \text{if } w \leq v. \end{cases} \quad (11.3.4.33)$$

(Q11.3.4.37.B) [Upwind flux vs. Godunov flux] Give an example of a strictly concave flux function $f : \mathbb{R} \rightarrow \mathbb{R}$ and of two states $v, w \in \mathbb{R}$, for which the upwind flux

$$F_{uw}(v, w) = \begin{cases} f(v) & , \text{if } \dot{s} \geq 0, \\ f(w) & , \text{if } \dot{s} < 0, \end{cases} \quad \dot{s} := \begin{cases} \frac{f(w) - f(v)}{w - v} & \text{for } v \neq w, \\ f'(v) & \text{for } v = w, \end{cases} \quad (11.3.4.19)$$

and the Godunov flux:

$$F_{GD}(v, w) = \begin{cases} \min_{v \leq u \leq w} f(u) & , \text{if } v < w, \\ \max_{w \leq u \leq v} f(u) & , \text{if } w \leq v, \end{cases} \quad (11.3.4.33)$$

yield different results.

Give an example of a strictly concave flux function f , for which they agree for all input states.

(Q11.3.4.37.C) [Expansion shock and transonic rarefaction] For Burgers equation with flux function $f(u) = u^2$ characterize all pairs of left and right states $u_l, u_r \in \mathbb{R}$ for which a conservative finite volume discretization of the 1D Riemann problem based on the upwind flux produces an expansion shock that violates the Lax entropy condition.

Hint. The upwind flux fails in situations where transonic rarefactions emerge as entropy solutions.

△

11.3.5 Monotone Schemes

We made the following observations made for some piecewise constant solutions $u_N(t)$ of semi-discrete evolutions arising from spatial finite volume discretization in conservation form (11.3.2.8):

$$\begin{array}{ll} \text{Exp. 11.3.4.17 (Lax-Friedrichs numerical flux (11.3.4.16))} & \diamond \min_{x \in \mathbb{R}} u_0(x) \leq u_N(x, t) \leq \max_{x \in \mathbb{R}} u_0(x) \\ \text{Exp. 11.3.4.35 (Godunov numerical flux (11.3.4.33))} & \diamond \text{no new local extrema in numerical solution} \end{array}$$

In these respects the conservative finite volume discretizations based on either the Lax-Friedrichs numerical flux (\rightarrow Section 11.3.4.2) or the Godunov numerical flux (\rightarrow Section 11.3.4.4) inherit crucial structural properties of the exact solution, see Section 11.2.7, in particular,

Theorem 11.2.7.1. Comparison principle for scalar conservation laws

$$\text{If } u_0 \leq \bar{u}_0 \text{ a.e. on } \mathbb{R} \Rightarrow u \leq \bar{u} \text{ a.e. on } \mathbb{R} \times]0, T[$$

Thus, in the numerical experiments these schemes display **structure preservation**, cf. Section 3.7.

Is this coincidence, just valid for the special settings examined in Exp. 11.3.4.17 and Exp. 11.3.4.35?

§11.3.5.1 (Discrete comparison principle) Focus: semi-discrete evolution (11.3.2.8) resulting from finite volume discretization in conservation form with 2-point numerical flux on an equidistant infinite mesh

$$(11.3.2.7) \quad \blacktriangleright \quad \frac{d\mu_j}{dt}(t) = -\frac{1}{h}(F(\mu_j(t), \mu_{j+1}(t)) - F(\mu_{j-1}(t), \mu_j(t))), \quad j \in \mathbb{Z}, \quad (11.3.2.8)$$

for Cauchy problem

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x), \quad x \in \mathbb{R}, \quad (11.2.2.1)$$

induced by Lax-Friedrichs numerical flux (11.3.4.16)

$$F_{LF}(v, w) = \frac{1}{2}(f(v) + f(w)) - \frac{1}{2} \max_{\min\{v, w\} \leq u \leq \max\{v, w\}} |f'(u)|(w - v). \quad (11.3.4.16)$$

$$\blacktriangleright \quad \frac{d\mu_j}{dt} = -\frac{1}{2h} \left(f(\mu_{j+1}) - f(\mu_{j-1}) - \max_{u \in [\mu_j, \mu_{j+1}]} |f'(u)|(\mu_{j+1} - \mu_j) + \max_{u \in [\mu_{j-1}, \mu_j]} |f'(u)|(\mu_j - \mu_{j-1}) \right). \quad (11.3.5.2)$$

Goal: show that $u_N(t)$ linked to $\vec{\mu}(t)$ from (11.3.5.2) through piecewise constant reconstruction (11.3.2.3) satisfies

$$\min_{x \in \mathbb{R}} u_N(x, 0) \leq u_N(x, t) \leq \max_{x \in \mathbb{R}} u_N(x, 0) \quad \forall x \in \mathbb{R}, \quad \forall t \in [0, T]. \quad (11.3.5.3)$$

Recall from Section 11.2.7: estimate (11.3.5.3) for the exact solution $u(x, t)$ of (11.2.2.1) is a consequence of the comparison principle of Thm. 11.2.7.1 and the fact that constant initial data are preserved during the evolution. The latter property is straightforward for conservative finite volume spatial semi-discretization, see (11.3.3.2).

➤ Goal: Establish comparison principle for finite volume semi-discrete solutions based on Lax-Friedrichs numerical flux:

$$\left\{ \begin{array}{l} \vec{\mu}(t), \vec{\eta}(t) \text{ solve (11.3.5.2)}, \\ \eta_j(0) \leq \mu_j(0) \quad \forall j \in \mathbb{Z} \end{array} \right\} \Rightarrow \eta_j(t) \leq \mu_j(t) \quad \forall j \in \mathbb{Z}, \quad \forall 0 \leq t \leq T.$$

Assumption: $\vec{\mu} = \vec{\mu}(t)$ and $\vec{\eta} = \vec{\eta}(t)$ solve (11.3.5.2) and satisfy for some $t \in [0, T]$

$$\eta_k(t) \leq \mu_k(t) \quad \forall k \in \mathbb{Z} \quad , \quad \xi := \eta_j(t) = \mu_j(t) \quad \text{for some } j \in \mathbb{Z} .$$

Can η_j raise above μ_j ?

$$\frac{d}{dt}(\mu_j - \eta_j) = -\frac{1}{h} \left(F_{\text{LF}}(\xi, \mu_{j+1}) - F_{\text{LF}}(\xi, \eta_{j+1}) + F_{\text{LF}}(\eta_{j-1}, \xi) - F_{\text{LF}}(\mu_{j-1}, \xi) \right) .$$

To show: $\frac{d}{dt}(\mu_j - \eta_j) \geq 0 \Rightarrow \mu_j(t) \text{ will stay above } \eta_j(t)$.

This can be concluded, if

$$F_{\text{LF}}(\xi, \mu_{j+1}) - F_{\text{LF}}(\xi, \eta_{j+1}) \leq 0 \quad \text{and} \quad F_{\text{LF}}(\eta_{j-1}, \xi) - F_{\text{LF}}(\mu_{j-1}, \xi) \leq 0 . \quad (11.3.5.4)$$

The only piece of information we are allowed to use is

$$\mu_{j+1} \geq \eta_{j+1} \quad \text{and} \quad \mu_{j-1} \geq \eta_{j-1} .$$

This would imply (11.3.5.4), if F_{LF} was increasing in the first argument and decreasing in the second argument. Such a trait of a two-point numerical flux is considered in the next definition.

Definition 11.3.5.5. Monotone numerical flux function

A 2-point numerical flux function $F = F(v, w)$ is called **monotone**, if

F is an **increasing** function of its **first** argument v ($\forall w$)

and

F is a **decreasing** function of its **second** argument w ($\forall v$).

Corollary 11.3.5.6. Simple criterion for monotone flux function

A continuously differentiable 2-point numerical flux function $F = F(v, w)$ is monotone, if and only if

$$\frac{\partial F}{\partial v}(v, w) \geq 0 \quad \text{and} \quad \frac{\partial F}{\partial w}(v, w) \leq 0 \quad \forall (v, w) . \quad (11.3.5.7)$$

The important 2-point numerical fluxes that we have studied in Section 11.3.4.2 and Section 11.3.4.4 enjoy the monotonicity property.

Lemma 11.3.5.8. Monotonicity of Lax-Friedrichs/Rusanov numerical flux and Godunov flux

For any continuously differentiable flux function f the associated Lax-Friedrichs/Rusanov flux (11.3.4.16) and Godunov flux (11.3.4.33) are monotone.

Proof.

① (Local) Lax-Friedrichs/Rusanov numerical flux:

$$F_{\text{LF}}(v, w) = \frac{1}{2}(f(v) + f(w)) - \frac{1}{2}(w - v) \cdot \max_{\min\{v,w\} \leq u \leq \max\{v,w\}} |f'(u)| .$$

Application of the criterion (11.3.5.7) is straightforward:

$$\begin{aligned}\frac{\partial F_{\text{LF}}}{\partial v}(v, w) &= \frac{1}{2}f'(v) + \frac{1}{2} \max_{\min\{v,w\} \leq u \leq \max\{v,w\}} |f'(u)| \geq 0 , \\ \frac{\partial F_{\text{LF}}}{\partial w}(v, w) &= \frac{1}{2}f'(w) - \frac{1}{2} \max_{\min\{v,w\} \leq u \leq \max\{v,w\}} |f'(u)| \leq 0 .\end{aligned}$$

For the genuine Lax-Friedrichs numerical flux (11.3.4.16) the proof of monotonicity entails treating numerous cases separately, because the factor in front of the diffusive flux will also depend on v and w .

② Godunov numerical flux

$$F_{\text{GD}}(v, w) = \begin{cases} \min_{v \leq u \leq w} f(u) & , \text{if } v < w , \\ \max_{w \leq u \leq v} f(u) & , \text{if } w \leq v . \end{cases} \quad (11.3.4.33)$$

$v < w$: If v increases, then the range of values over which the minimum is taken will shrink, which makes $F_{\text{GD}}(v, w)$ increase.

If w is raised, then the minimum is taken over a larger interval, which causes $F_{\text{GD}}(v, w)$ to become smaller.

$v \geq w$: If v increases, then the range of values over which the maximum is taken will grow, which makes $F_{\text{GD}}(v, w)$ increase.

If w is raised, then the maximum is taken over a smaller interval, which causes $F_{\text{GD}}(v, w)$ to decrease. \square

Lemma 11.3.5.9. Comparison principle for monotone semi-discrete conservative evolutions

Let the 2-point numerical flux function $F = F(v, w)$ be monotone (\rightarrow Def. 11.3.5.5) and $\vec{\mu} = \vec{\mu}(t)$, $\vec{\eta} = \vec{\eta}(t)$ solve (11.3.2.8). Then

$$\eta_k(0) \leq \mu_k(0) \quad \forall k \in \mathbb{Z} \Rightarrow \eta_k(t) \leq \mu_k(t) \quad \forall k \in \mathbb{Z}, \quad \forall 0 \leq t \leq T .$$

The assertion of Lemma 11.3.5.9 means that for monotone numerical flux, the semi-discrete evolution satisfies the **comparison principle** of Thm. 11.2.7.1.

Proof (of Lemma 11.3.5.9, following the above considerations for the Lax-Friedrichs flux).

The two sequences of nodal values satisfy (11.3.2.8)

$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h}(F(\mu_j(t), \mu_{j+1}(t)) - F(\mu_{j-1}(t), \mu_j(t))), \quad j \in \mathbb{Z}, \quad (11.3.5.10)$$

$$\frac{d\eta_j}{dt}(t) = -\frac{1}{h}(F(\eta_j(t), \eta_{j+1}(t)) - F(\eta_{j-1}(t), \eta_j(t))), \quad j \in \mathbb{Z}. \quad (11.3.5.11)$$

Let t_0 be the *earliest* time, at which $\vec{\eta}$ “catches up” with $\vec{\mu}$ in at least one node $x_j, j \in \mathbb{Z}$, of the mesh, that is

$$\eta_k(t_0) \leq \mu_k(t_0) \quad \forall k \in \mathbb{Z}, \quad \xi := \eta_j(t_0) = \mu_j(t_0) .$$

By subtracting (11.3.5.10) and (11.3.5.11) we get

$$\frac{d}{dt}(\mu_j - \eta_j)(t_0) = -\frac{1}{h} \left(F(\xi, \mu_{j+1}(t_0)) - F(\xi, \eta_{j+1}(t_0)) + F(\eta_{j-1}(t_0), \xi) - F(\mu_{j-1}(t_0), \xi) \right) \geq 0 ,$$

because for a *monotone* numerical flux function (\rightarrow Def. 11.3.5.5)

$$\begin{aligned} \eta_{j-1}(t_0) &\leq \mu_{j-1}(t_0) & \text{increasing in first argument} \Rightarrow F(\eta_{j-1}(t_0), \xi) - F(\mu_{j-1}(t_0), \xi) \leq 0 , \\ \eta_{j+1}(t_0) &\leq \mu_{j+1}(t_0) & \text{decreasing in second argument} \Rightarrow F(\xi, \mu_{j+1}(t_0)) - F(\xi, \eta_{j+1}(t_0)) \leq 0 . \end{aligned}$$

This means that “ η_j cannot overtake μ_j ”: no value η_j can ever raise above μ_j . \square

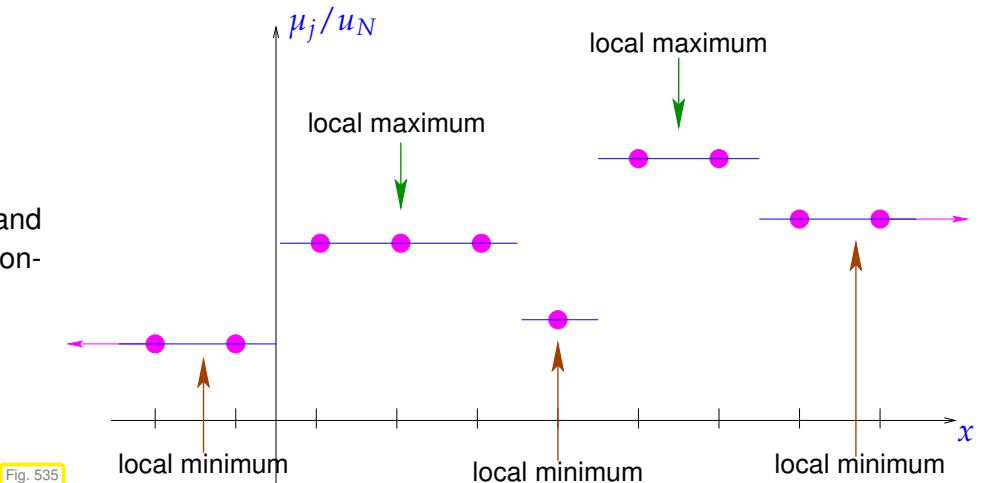
§11.3.5.12 (No creation of discrete local extrema) Now we want to study the “decrease of the number of local extrema” during a semi-discrete evolution, another *structural property* of exact solutions of conservations laws, see Section 11.2.7.

Intuitive terminology: $\vec{\mu}$ has a **local maximum** $u_m \in \mathbb{R}$, if

$$\exists j \in \mathbb{Z}: \mu_j = u_m \quad \text{and} \quad \exists k_l < j < k_r \in \mathbb{N}: \max_{k_l < l < k_r} \mu_l = u_m \quad \text{and} \quad \mu_{k_l} < u_m, \mu_{k_r} < u_m .$$

In analogous fashion, we define a local minimum. If $\vec{\mu}$ is constant for large indices, these values are also regarded as local extrema.

Counting local extrema of $\vec{\mu}$ and the associated piecewise constant reconstruction.



Lemma 11.3.5.13. Non-oscillatory monotone semi-discrete evolutions

If $\vec{\mu} = \vec{\mu}(t)$ solves (11.3.2.8) with a *monotone* numerical flux function $F = F(v, w)$ and $\vec{\mu}(0)$ has finitely many local extrema, then the number of local extrema of $\vec{\mu}(t)$ cannot be larger than that of $\vec{\mu}(0)$.

Proof. $i \doteq$ index of local maximum of $\vec{\mu}(t)$, t fixed

$$\begin{aligned} \mu_{i-1}(t) &\leq \mu_i(t) & \xrightarrow{\text{monotone flux}} F(\mu_i, \mu_{i+1}) &\geq F(\mu_i, \mu_i) \geq F(\mu_{i-1}, \mu_i) , \\ \mu_{i+1}(t) &\leq \mu_i(t) \\ \Rightarrow \quad \frac{d}{dt} \mu_i(t) &= -\frac{1}{h} (F(\mu_i, \mu_{i+1}) - F(\mu_{i-1}, \mu_i)) \leq 0 . \end{aligned}$$

➤ maxima of $\vec{\mu}$ subside, (minima of $\vec{\mu}$ rise !)

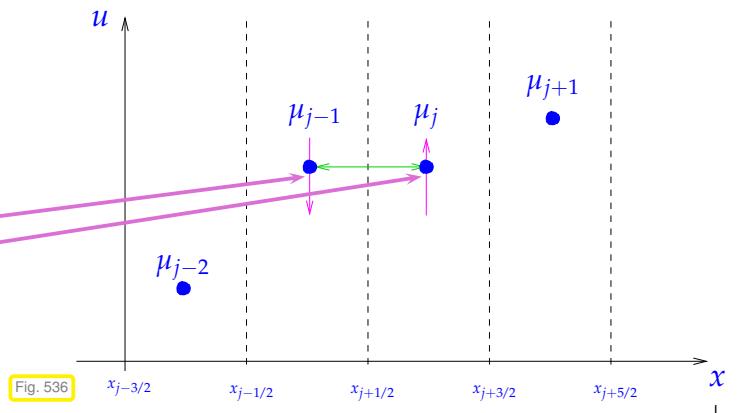
Idea of proof:

No new (local) extrema can arise !

Adjacent values cannot “overtake”:

local maximum: cannot move up

local minimum: cannot move down



Review question(s) 11.3.5.14 (Monotone schemes)

(Q11.3.5.14.A) Show that for smooth and convex $f : \mathbb{R} \rightarrow \mathbb{R}$ the Godunov numerical flux

$$F_{\text{GD}}(v, w) = \begin{cases} \min_{v \leq u \leq w} f(u) & , \text{if } v < w , \\ \max_{w \leq u \leq v} f(u) & , \text{if } w \leq v , \end{cases} \quad (11.3.4.33)$$

satisfies

$$\frac{\partial F_{\text{GD}}}{\partial v}(v, w) \geq 0 \quad \text{and} \quad \frac{\partial F_{\text{GD}}}{\partial w}(v, w) \leq 0 \quad \forall (v, w) , \quad (11.3.5.7)$$

$$\left| \frac{\partial F_{\text{GD}}}{\partial v}(v, w) \right|, \left| \frac{\partial F_{\text{GD}}}{\partial w}(v, w) \right| \leq \max_{\min\{v, w\} \leq u \leq \max\{v, w\}} |f'(u)| \quad \forall (v, w) .$$

(Q11.3.5.14.B) A mapping $\mathcal{H} : \mathbb{R}^{\mathbb{Z}} \rightarrow \mathbb{R}^{\mathbb{Z}}$ ($\mathbb{R}^{\mathbb{Z}}$ is the vector space of real-valued sequences $(\mu_j)_{j \in \mathbb{Z}}$, equivalently, the vector space of functions $\mathbb{Z} \mapsto \mathbb{R}$) is called **monotone***, if

$$\zeta_j \geq \mu_j \quad \forall j \in \mathbb{Z} \Rightarrow (\mathcal{H}((\zeta_k)_{k \in \mathbb{Z}}))_j \geq (\mathcal{H}((\mu_k)_{k \in \mathbb{Z}}))_j \quad \forall j \in \mathbb{Z} .$$

- Show that $\mathcal{H} : \mathbb{R}^{\mathbb{Z}} \rightarrow \mathbb{R}^{\mathbb{Z}}$ which is continuously differentiable in each component is monotone, if

$$\left(\frac{\partial \mathcal{H}}{\partial \mu_k} \right)_j \geq 0 \quad \forall j, k \in \mathbb{Z} .$$

- Applying **explicit Euler timestepping** with timestep $\tau > 0$ to the semi-discrete evolution arising from the conservative finite-volume discretization of a scalar conservation law $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0$ on an equidistant spatial mesh (mesh width $h > 0$) and based on the **Godunov numerical flux** yields a mapping $\mathcal{H} : \mathbb{R}^{\mathbb{Z}} \rightarrow \mathbb{R}^{\mathbb{Z}}$. Show that this mapping is monotone provided that

$$\frac{\tau}{h} \leq \frac{1}{2M} , \quad M := \max\{|f'(u)| : \min_{x \in \mathbb{R}} u_0(x) \leq u \leq \max_{x \in \mathbb{R}} u_0(x)\} .$$

Hint. Depends on Question (Q11.3.5.14.A).

Remark. Mappings like \mathcal{H} will be investigated more closely in § 11.4.1.8.

△

11.4 Timestepping for Finite-Volume Methods

In the spirit of the method of lines (MOL) approach, we next pursue the temporal discretization of the ordinary differential equation (FV-MOL-ODE)

$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h} (F(\mu_{j-m_l+1}(t), \dots, \mu_{j+m_r}(t)) - F(\mu_{j-m_l}(t), \dots, \mu_{j+m_r-1}(t))) , \quad j \in \mathbb{Z} , \quad (11.3.2.7)$$

which arises from the conservative finite-volume (FV) spatial semi-discretization of the Cauchy problem for a generic 1D scalar conservation law (without sources)

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) \quad \text{in } \mathbb{R} . \quad (11.2.2.1)$$

Remark 11.4.0.1. Note that (11.3.2.7) is an ODE on the infinite-dimensional state space $\mathbb{R}^{\mathbb{Z}}$, but formally we can treat it like a regular ODE in \mathbb{R}^N . In any case, in actual computations the spatial domain will always be truncated to a finite interval, and the initial value $u_0 = u_0(x)$ will be constant for large $|x|$. Such a truncation is implemented in Code 11.3.2.9. \square

11.4.1 Fully Discrete Evolutions

§11.4.1.1 (Explicit timestepping for FV-MOL-ODE) We have learned that *explicit* timestepping methods for initial value problems for the generic autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ generate a sequence $\mathbf{y}^{(0)} := \mathbf{y}_0, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots$ of approximations of $\mathbf{y}(t)$ at particular points t_j in time (forming a temporal grid) by computing $\mathbf{y}^{(j+1)}$ from $\mathbf{y}^{(j)}$ by means of a fixed small number of \mathbf{f} -evaluations.

For the method-of-lines ordinary differential equations (11.3.2.7) we exclusively focus on explicit timestepping, for good reasons: Apart from the simplest case of linear advection (\rightarrow Section 11.1.1), the right-hand-side in (11.3.2.7) will invariably be *non-linear* and might even be *non-smooth*, for instance when choosing the Godunov numerical flux (11.3.4.33). Thus, anything that goes beyond plain evaluation can be difficult and unpredictable; imagine using an iterative method for solving a non-linear system of equations involving that right-hand-side. This leaves little other option than using explicit timestepping. \square

§11.4.1.2 (Runge-Kutta single-step timestepping) In a straightforward way explicit single-step timestepping methods can be applied to (11.3.2.7).

Our focus: *Explicit* Runge-Kutta single-step methods (RK-SSMs, \rightarrow Def. 7.3.3.1)

Remember the the definition of an RK-SSM for a general ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$ on state space V_0 :

Definition 7.3.3.1. General Runge-Kutta single-step method

For coefficients $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, the discrete evolution $\Psi^{s,t}$ of an **s-stage Runge-Kutta single step method** (RK-SSM) for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$, is defined by

$$\mathbf{k}_i := \mathbf{f}\left(t + c_i \tau, \mathbf{u} + \tau \sum_{j=1}^s a_{ij} \mathbf{k}_j\right), \quad i = 1, \dots, s , \quad \Psi^{t,t+\tau} \mathbf{u} := \mathbf{u} + \tau \sum_{i=1}^s b_i \mathbf{k}_i .$$

The $\mathbf{k}_i \in V_0$ are called **increments**.

We have also seen that the coefficients $a_{ij} \in \mathbb{R}$ and $b_i \in \mathbb{R}$ are usually given in the form of a **Butcher scheme**

$$\begin{array}{c|ccccc} & c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ & c_2 & a_{21} & \ddots & & a_{2s} \\ \hline \mathbf{c} & \mathfrak{A} & \hat{=} & \vdots & \ddots & \vdots \\ & \mathbf{b}^T & & \vdots & & \vdots \\ & c_s & a_{s1} & \dots & & a_{ss} \\ \hline & b_1 & b_2 & \dots & \dots & b_s \end{array} , \quad \mathbf{c}, \mathbf{b} \in \mathbb{R}^s, \quad \mathfrak{A} \in \mathbb{R}^{s,s} , \quad (7.3.3.3)$$

here for a general Runge-Kutta method.

Next recall from Def. 6.4.0.9 that *explicit* s -stage Runge-Kutta single step methods are distinguished by the fact that

the coefficients a_{ij} vanish for $j \geq i, 1 \leq i, j \leq s$.

This means, that for explicit RK-SSM the coefficient matrix \mathfrak{A} in the Butcher scheme is strictly lower triangular. Butcher schemes for *explicit* RK-SSMs look like

$$\frac{\mathbf{c} \mid \mathfrak{A}}{\mathbf{b}^T} \hat{=} \begin{array}{c|cccc} 0 & 0 & \dots & \dots & 0 \\ c_2 & a_{21} & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ c_s & a_{s1} & \dots & a_{s,s-1} & 0 \\ \hline b_1 & b_2 & \dots & \dots & b_s \end{array}, \quad \mathbf{c}, \mathbf{b} \in \mathbb{R}^s, \quad \mathfrak{A} \in \mathbb{R}^{s,s},$$

Obviously, in this case the increments \mathbf{k}_i can be computed in turns (without solving a non-linear system of equations): For the abstract autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ an explicit s -stage Runge-Kutta single step method reads (for uniform timestep size $\tau > 0$)

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(\mathbf{y}^{(k)}) , \\ \mathbf{k}_2 &= \mathbf{f}(\mathbf{y}^{(k)} + \tau a_{21} \mathbf{k}_1) , \\ &\vdots \\ \mathbf{k}_s &= \mathbf{f}(\mathbf{y}^{(k)} + \tau a_{s1} \mathbf{k}_1 + \dots + \tau a_{s,s-1} \mathbf{k}_{s-1}) , \end{aligned} \quad , \quad \mathbf{y}^{(k+1)} := \mathbf{y}^{(k)} + \tau \sum_{l=1}^s b_l \mathbf{k}_l . \quad (11.4.1.3)$$

Returning to (11.3.2.7) we now consider an initial value problem for an abstract semi-discrete evolution in $\mathbb{R}^{\mathbb{Z}}$:

$$\frac{d\vec{\mu}}{dt}(t) = \mathcal{L}_h(\vec{\mu}(t)) , \quad 0 \leq t \leq T , \quad \vec{\mu}(0) = \vec{\mu}_0 \in \mathbb{R}^{\mathbb{Z}} . \quad (11.4.1.4)$$

Here $\mathcal{L}_h : \mathbb{R}^{\mathbb{Z}} \mapsto \mathbb{R}^{\mathbb{Z}}$ is a (non-linear) **finite-difference operator**. For instance, for a finite volume semi-discretization in conservation form with 2-point numerical flux is reads

$$(11.3.2.8) \Rightarrow (\mathcal{L}_h \vec{\mu})_j := -\frac{1}{h} (F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)) . \quad (11.4.1.5)$$

Note that for conservative finite volume discretizations \mathcal{L}_h is **local**: $(\mathcal{L}_h(\vec{\mu}))_j$ depends only on “neighboring values” $\mu_{j-m_l}, \dots, \mu_{j+m_r}$:

$$(\mathcal{L}_h \vec{\mu})_j = \mathcal{L}_j(\mu_{j-m_l}, \dots, \mu_{j+m_r}) , \quad j \in \mathbb{Z} , \quad (11.4.1.6)$$

with suitable functions $\mathcal{L}_j : \mathbb{R}^{1+m_l+m_r} \rightarrow \mathbb{R}$.

From (11.4.1.3) we deduce the formulas for a single step of an explicit s -stage Runge-Kutta single step

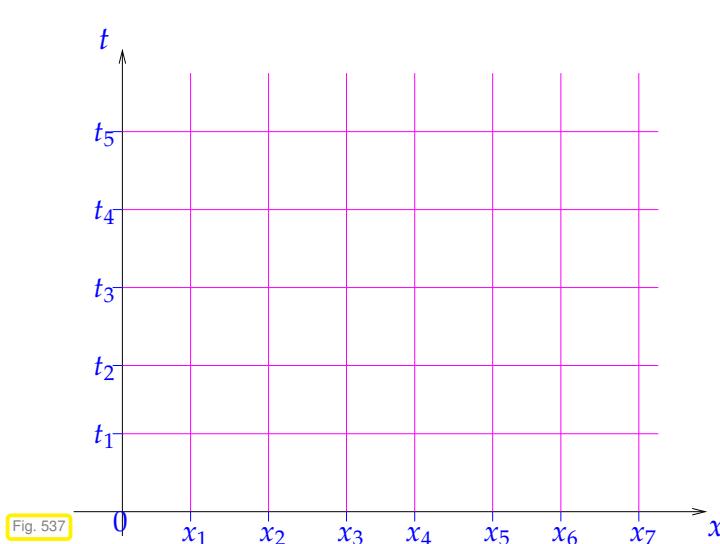
method applied to (11.4.1.4) with timestep $\tau > 0$:

$$\begin{aligned}
 \vec{\kappa}_1 &= \mathcal{L}_h(\vec{\mu}^{(k)}) , \\
 \vec{\kappa}_2 &= \mathcal{L}_h(\vec{\mu}^{(k)} + \tau a_{21} \vec{\kappa}_1) , \\
 \vec{\kappa}_3 &= \mathcal{L}_h(\vec{\mu}^{(k)} + \tau a_{31} \vec{\kappa}_1 + \tau a_{32} \vec{\kappa}_2) , \\
 &\vdots \\
 \vec{\kappa}_s &= \mathcal{L}_h(\vec{\mu}^{(k)} + \tau \sum_{j=1}^{s-1} a_{sj} \vec{\kappa}_j) , \\
 \vec{\mu}^{(k+1)} &= \vec{\mu}^{(k)} + \tau \sum_{l=1}^s b_l \vec{\kappa}_l .
 \end{aligned} \tag{11.4.1.7}$$

All increments $\vec{\kappa}_i$, $i = 1, \dots, s$, belong to the state space \mathbb{R}^Z .

The formulas (11.4.1.7) are “explicit” in the sense that timestepping just relies on more evaluations of the operator \mathcal{L}_h . This greatly facilitates implementation, because, as remarked already, \mathcal{L}_h will, in general, be a non-linear and even non-smooth mapping. Thus it might be very difficult and expensive to solve a system of non-linear equations involving \mathcal{L}_h . \dashv

§11.4.1.8 (Fully discrete evolution) The application of a timestepping scheme converts (11.4.1.4) into a family of equations for functions on an infinite space-time grid. We give a formal description:



Setting: equidistant spatial mesh \mathcal{M} , meshwidth $h > 0$, nodes $x_j := h j$, $j \in \mathbb{Z}$, uniform timestep $\tau > 0$, $t_k := \tau k$, $k \in \mathbb{N}_0$.

Single step timestepping for (11.4.1.4) produces a sequence $(\vec{\mu}^{(k)})_{k \in \mathbb{N}_0}$

$$\mu_j^{(k)} \approx u(x_j, t_k), \quad j \in \mathbb{Z}, k \in \mathbb{N}_0 .$$

► Fully discrete evolution

$$\vec{\mu}^{(k+1)} = \mathcal{H}_h(\vec{\mu}^{(k-1)}) , \quad k \in \mathbb{N}_0 .$$

$\mathcal{H}_h : \mathbb{R}^Z \mapsto \mathbb{R}^Z$: denotes the fully discrete evolution operator, arising from applying the single step timestepping (11.4.1.7) to (11.4.1.4). \dashv

EXAMPLE 11.4.1.9 (Fully discrete evolutions arising from conservative discretizations) We examine simple fully discrete evolutions \mathcal{H}_h arising from finite volume semi-discretization in conservation form with 2-point numerical flux $F = F(v, w)$, that is, from timestepping for a semi-discrete evolution with finite-difference operator

$$(11.3.2.8) \Rightarrow (\mathcal{L}_h \vec{\mu})_j := -\frac{1}{h} (F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)) , \quad \vec{\mu} \in \mathbb{R}^Z, j \in \mathbb{Z} . \tag{11.4.1.5}$$

Using *explicit Euler* timestepping ($\hat{=} 1$ -stage explicit RK-method) we get

$$\vec{\mu}^{(k+1)} = \vec{\mu}^{(k)} + \tau \mathcal{L}_h(\vec{\mu}^{(k)}) .$$

$$\Rightarrow (\mathcal{H}_h(\vec{\mu}))_j = \mu_j - \frac{\tau}{h} (F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)) . \quad (11.4.1.10)$$

In the case of *explicit trapezoidal rule* timestepping Eq. (6.4.0.6) (2-stage RK-SSM, method of Heun)

$$\vec{\kappa} = \vec{\mu}^{(k)} + \tau \mathcal{L}_h(\vec{\mu}^{(k)}) , \quad \vec{\mu}^{(k+1)} = \vec{\mu}^{(k)} + \frac{\tau}{2} (\mathcal{L}_h(\vec{\mu}^{(k)}) + \mathcal{L}_h(\vec{\kappa})) .$$

$$\Rightarrow \begin{cases} \kappa_j := (\vec{\kappa})_j = \mu_j - \frac{\tau}{h} (F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)) , \\ (\mathcal{H}_h(\vec{\mu}))_j = \mu_j - \frac{\tau}{2h} (F(\kappa_j, \kappa_{j+1}) - F(\kappa_{j-1}, \kappa_j) + F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)) . \end{cases} \quad (11.4.1.11)$$

For the *explicit midpoint rule*, another 2-stage RK-SSM, we get the recursion

$$\vec{\kappa} = \vec{\mu}^{(k)} + \frac{\tau}{2} \mathcal{L}_h(\vec{\mu}^{(k)}) , \quad \vec{\mu}^{(k+1)} = \vec{\mu}^{(k)} + \tau \mathcal{L}_h(\vec{\kappa}) .$$

$$\Rightarrow \begin{cases} \kappa_j := (\vec{\kappa})_j = \mu_j - \frac{\tau}{2h} (F(\mu_j, \mu_{j+1}) - F(\mu_{j-1}, \mu_j)) , \\ (\mathcal{H}_h(\vec{\mu}))_j = \mu_j - \frac{\tau}{h} (F(\kappa_j, \kappa_{j+1}) - F(\kappa_{j-1}, \kappa_j)) . \end{cases} \quad (11.4.1.12)$$

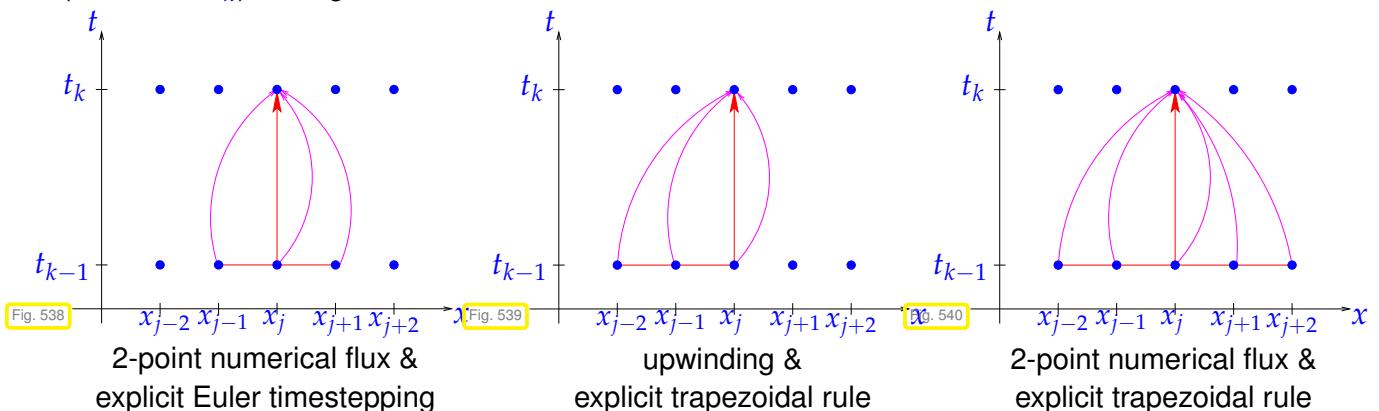
□

11.4.2 CFL-Condition

As we have seen in Section 9.2.3 and Section 9.3.5, the use of explicit timestepping in the context of a method-of-lines approach to an initial boundary value problem for a PDE often faces a mesh-dependent *timestep constraint* in order to avoid catastrophic blow-up. This will also be the case for the conservative finite volume discretization of conservation laws.

§11.4.2.1 (Difference stencils) We have already observed in (11.4.1.6) that the operators \mathcal{L}_h process cell averages μ_j locally. This allows a catchy representation of the structure of fully discrete evolutions.

We employ a *stencil notation* in order to visualize the flow of information in fully discrete *explicit* evolutions (action of \mathcal{H}_h), cf. Fig. 398.



The arrows indicate which values $\mu_i^{(k-1)}$ contribute in the computation of $\mu_j^{(k)}$. □

§11.4.2.2 (Common properties of conservative fully discrete evolutions) A consequence of the locality of \mathcal{L}_h combined with *explicit* timestepping is the *locality* of fully discrete evolution operator:

$$\exists m_l, m_r \in \mathbb{N}_0: \quad (\mathcal{H}(\vec{\mu}))_j = \mathcal{H}_j(\mu_{j-m_l}, \dots, \mu_{j+m_r}) . \quad (11.4.2.3)$$

If the flux function f does not depend on x , $f = f(u)$ as in (11.2.2.1), we can expect that

$$\mathcal{H}_h \text{ is translation-invariant: } \mathcal{H}_j = \mathcal{H} \quad \forall j \in \mathbb{Z} .$$

This is the case for (11.4.1.10) and (11.4.1.11).

By inspection of (11.4.1.7) we realize that, if \mathcal{L}_h is translation-invariant

$$(\mathcal{L}_h(\vec{\mu}))_j = \mathcal{L}(\mu_{j-n_l}, \dots, \mu_{j+n_r}), \quad j \in \mathbb{Z},$$

for a *single* function $\mathcal{L} : \mathbb{R}^{n_l+n_r+1} \rightarrow \mathbb{R}$, $n_l, n_r \in \mathbb{N}$, and timestepping relies on an s -stage explicit Runge-Kutta method, then we conclude for m_l, m_r in (11.4.2.3)

$$m_l \leq s \cdot n_l, \quad m_r \leq s \cdot n_r.$$

This gives us bounds for the width of the stencil of \mathcal{H} . □

§11.4.2.4 (Domains of dependence) Now we revisit a concept from Section 9.3.5, see, in particular, Rem. 9.3.5.7:

Definition 11.4.2.5. Numerical domain of dependence

Consider explicit translation-invariant fully discrete evolution $\vec{\mu}^{(k+1)} := \mathcal{H}(\vec{\mu}^{(k)})$ on uniform spatio-temporal mesh $(x_j = h j, j \in \mathbb{Z}, t_k = k \tau, k \in \mathbb{N}_0)$ with

$$\exists m \in \mathbb{N}_0: \quad (\mathcal{H}(\vec{\mu}))_j = \mathcal{H}(\mu_{j-m}, \dots, \mu_{j+m}), \quad j \in \mathbb{Z}. \quad (11.4.2.6)$$

Then the **numerical domain of dependence** is given by

$$D_h^-(x_j, t_k) := \{(x_n, t_l) \in \mathbb{R} \times [0, t_k]: j - m(k-l) \leq n \leq j + m(k-l)\}.$$

From Thm. 11.2.7.3 recall the **maximal analytical domain of dependence** for a solution of (11.2.2.1) is

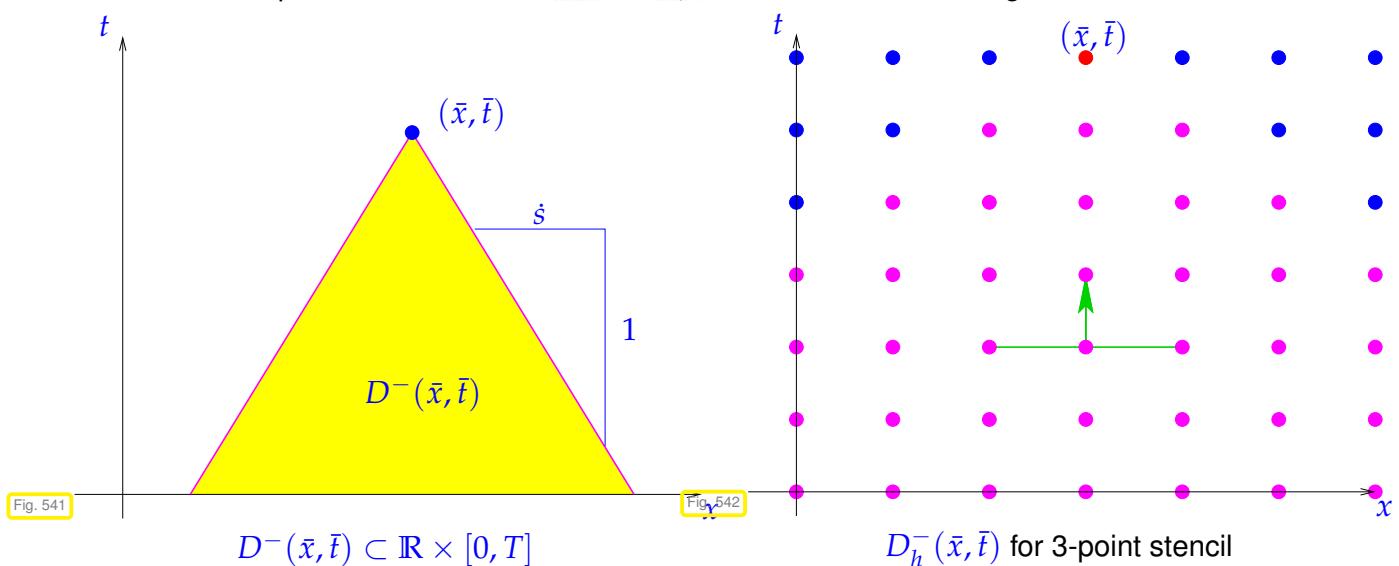
$$D^-(\bar{x}, \bar{t}) := \{(x, t) \in \mathbb{R} \times [0, \bar{t}]: \dot{s}_{\min}(\bar{t} - t) \leq x - \bar{x} \leq \dot{s}_{\max}(\bar{t} - t)\}. \quad (11.4.2.7)$$

with **maximal speeds of propagation** in either direction

$$\dot{s}_{\min} := \min\{f'(\xi): \inf_{x \in \mathbb{R}} u_0(x) \leq \xi \leq \sup_{x \in \mathbb{R}} u_0(x)\}, \quad (11.4.2.8)$$

$$\dot{s}_{\max} := \max\{f'(\xi): \inf_{x \in \mathbb{R}} u_0(x) \leq \xi \leq \sup_{x \in \mathbb{R}} u_0(x)\}. \quad (11.4.2.9)$$

A domain of dependence with $\dot{s} := \dot{s}_{\min} = \dot{s}_{\max}$ is sketched below in Fig. 542.



The right figure highlights the numerical domain of dependence on a uniform space-time grid for $m = 1$, that is the case of a 3-point stencil as depicted in Fig. 539. \square

§11.4.2.10 (CFL-condition)

Definition 11.4.2.11. Courant-Friedrichs-Lowy (CFL-)condition \rightarrow Rem. 9.3.5.7

An explicit translation-invariant local fully discrete evolution $\vec{\mu}^{(k+1)} := \mathcal{H}(\vec{\mu}^{(k)})$ on uniform spatio-temporal mesh ($x_j = h j$, $j \in \mathbb{Z}$, $t_k = k\tau$, $k \in \mathbb{N}_0$) as in Def. 11.4.2.5 satisfies the **Courant-Friedrichs-Lowy (CFL)-condition**, if the convex hull of its numerical domain of dependence contains the maximal analytical domain of dependence:

$$D^-(x_j, t_k) \subset \text{convex}(D_h^-(x_j, t_k)) \quad \forall j, k.$$

By the definition (11.4.2.7) of $D^-(\bar{x}, \bar{t})$ and $D_h^-(x_j, t_k)$ from Def. 11.4.2.5 a sufficient condition for the CFL-condition to be satisfied is

$$\frac{\tau}{h} \leq \frac{m}{\max\{|\dot{s}_{\min}|, |\dot{s}_{\max}|\}} \quad \longleftrightarrow \quad \text{a } \textit{timestep constraint!}. \quad (11.4.2.12)$$

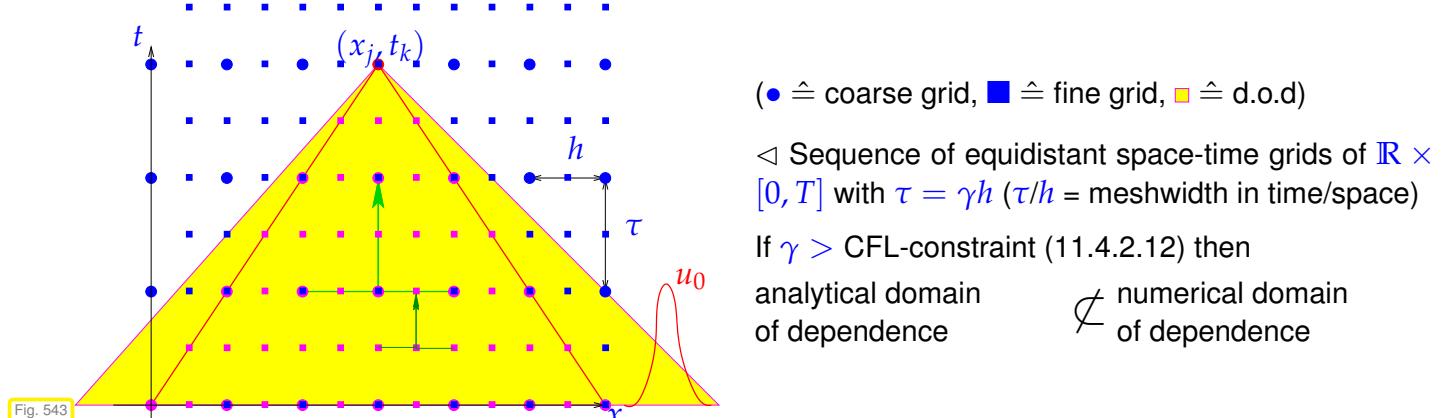
This is a timestep constraint similar to the one encountered in Section 9.3.5 in the context of leapfrog timestepping for the semi-discrete wave equation.

Remember Rem. 9.2.8.10, page 585: stability induced timestep constraints can lead to an inefficient discretization. Also in the case of the “ODE” (11.3.2.7) implicit timestepping can circumvent the CFL-condition. Yet, at the price of having to solve *non-linear systems* of equations, which may be prohibitive and makes people gladly put up with the moderate timestep constraint (11.4.2.12).

As discussed in Rem. 9.3.5.7,

we cannot expect convergence for *fixed ratio* $\tau : h$, for $h \rightarrow 0$ in case the CFL-condition is violated.

Refer to Fig. 544 for a “graphical argument”:



Heuristic reasoning: Initial data u_0 supported outside the numerical domain of dependence can influence the exact solution in the point (x_j, t_k) , which is contained in all spatio-temporal grids of the sequence. However, $\mu_j^{(k)}$ will never be influenced by initial data inside the support of u_0 . Hence, there can be cases, when $\mu_j^{(k)} \not\rightarrow u(x_j, t_k)$ though $h, \tau \rightarrow 0$. \square

Review question(s) 11.4.2.13 (Fully discrete evolutions and CFL-condition)

For the following review questions you may need the definition of a Runge-Kutta single-step method (RK-SSM) and their encoding through Butcher schemes.

Definition 7.3.3.1. General Runge-Kutta single-step method

For coefficients $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^s a_{ij}$, $i, j = 1, \dots, s$, $s \in \mathbb{N}$, the discrete evolution $\Psi^{s,t}$ of an **s -stage Runge-Kutta single step method** (RK-SSM) for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t, \mathbf{u})$, is defined by

$$\mathbf{k}_i := \mathbf{f}\left(t + c_i \tau, \mathbf{u} + \tau \sum_{j=1}^s a_{ij} \mathbf{k}_j\right), \quad i = 1, \dots, s, \quad \Psi^{t,t+\tau} \mathbf{u} := \mathbf{u} + \tau \sum_{i=1}^s b_i \mathbf{k}_i.$$

The $\mathbf{k}_i \in V_0$ are called **increments**.

$$\frac{\mathbf{c}}{\mathbf{b}^T} \hat{=} \begin{array}{c|ccccc} c_1 & a_{11} & a_{12} & \dots & \dots & a_{1s} \\ c_2 & a_{21} & \ddots & & & a_{2s} \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ c_s & a_{s1} & \dots & & & a_{ss} \\ \hline b_1 & b_2 & \dots & \dots & \dots & b_s \end{array}, \quad \mathbf{c}, \mathbf{b} \in \mathbb{R}^s, \quad \mathfrak{A} \in \mathbb{R}^{s,s}. \quad (7.3.3.3)$$

(Q11.4.2.13.A) We consider an abstract semi-discrete evolution in $\mathbb{R}^{\mathbb{Z}}$:

$$\frac{d\vec{\mu}}{dt}(t) = \mathcal{L}_h(\vec{\mu}(t)), \quad \mathcal{L}_h : \mathbb{R}^{\mathbb{Z}} \mapsto \mathbb{R}^{\mathbb{Z}}.$$

We perform timestepping based on an s -stage RK-SSM described by the following butcher scheme:

$$\frac{\mathbf{c}}{\mathbf{b}^T} \hat{=} \begin{array}{c|ccccc} 0 & 0 & \dots & & \dots & 0 \\ c_2 & a_{21} & \ddots & & & \vdots \\ \vdots & 0 & \ddots & & & \vdots \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ c_s & 0 & \dots & \dots & 0 & a_{s,s-1} & 0 \\ \hline b_1 & b_2 & \dots & & \dots & \dots & b_s \end{array}, \quad \mathbf{c}, \mathbf{b} \in \mathbb{R}^s, \quad \mathfrak{A} \in \mathbb{R}^{s,s}.$$

Elaborate the fully discrete evolution equations.

(Q11.4.2.13.B) [Multi-step timestepping] The 2-step Adams-Basforth **multi-step** method with uniform timestep $\tau > 0$ applied to the ODE $\dot{\mathbf{u}} = \mathbf{F}(t, \mathbf{u})$ generates a sequence of approximate states $\mathbf{u}^{(j)} \approx \mathbf{u}(t_j)$, $t_j := t_0 + \tau j$, by the **3-term recursion**

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + \tau \left(\frac{3}{2} \mathbf{f}(t_j, \mathbf{u}^{(j)}) - \frac{1}{2} \mathbf{f}(t_{j-1}, \mathbf{u}^{(j-1)}) \right). \quad (11.4.2.14)$$

Give the stencil of the fully discrete evolution when this timestepping scheme is used for the FV-MOL-ODE

$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h} (F(\mu_j(t), \mu_{j+1}(t)) - F(\mu_{j-1}(t), \mu_j(t))), \quad j \in \mathbb{Z}, \quad (11.3.2.8)$$

on a uniform space-time grid and with some numerical flux function $F : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$.

(Q11.4.2.13.C) We consider a Cauchy problem for 1D linear advection with $v > 0$

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(vu) = 0 \quad \text{in } \tilde{\Omega} = \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) \quad \forall x \in \mathbb{R} . \quad (11.1.1.11)$$

For spatial discretization we use a conservative finite volume method with Lax-Friedrichs/Rusanov numerical flux. For timestepping we employ

1. the explicit Euler single-step method,
2. the implicit Euler single-step method.

In both cases derive the equations for the resulting fully discrete evolutions on a uniform space-time grid with spatial mesh width $h > 0$ and timestep $\tau > 0$.

For both choices characterize the **numerical domain of dependence** for $v = 1$.

Hint. The (local) Lax-Friedrichs/Rusanov numerical flux function for the scalar conservation law $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}f(u) = 0$ is

$$F_{LF}(v, w) = \frac{1}{2}(f(v) + f(w)) - \frac{1}{2}(w - v) \cdot \max_{\min\{v, w\} \leq u \leq \max\{v, w\}} |f'(u)| . \quad (11.3.4.16)$$

(Q11.4.2.13.D) Give reasons why explicit timestepping methods are preferred for the semi-discrete evolution problems arising from conservative spatial finite-volume discretization of 1D scalar conservation laws.

(Q11.4.2.13.E) We consider the fully discrete evolution for solving a Cauchy problem for a 1D scalar conservation law based on

- (i) a conservative finite-volume spatial discretization with 2-point numerical flux,
- (ii) an s -stage explicit Runge-Kutta single step method described by the Butcher scheme

$$\begin{array}{c|ccccc} & 0 & 0 & \dots & \dots & 0 \\ \hline \mathbf{c} & c_2 & a_{21} & \ddots & & \vdots \\ \hline \mathbf{b}^T & \vdots & \vdots & \ddots & \ddots & \vdots \\ & \vdots & \vdots & \ddots & \ddots & \vdots \\ \hline & c_s & a_{s1} & \dots & a_{s,s-1} & 0 \\ \hline & b_1 & b_2 & \dots & \dots & b_s \end{array} \hat{=} \quad , \quad \mathbf{c}, \mathbf{b} \in \mathbb{R}^s, \quad \mathfrak{A} \in \mathbb{R}^{s,s} .$$

We obtain a solution $\vec{\mu}^{(k)} \in \mathbb{R}^{\mathbb{Z}}$, $k \in \mathbb{N}_0$.

Assume that the initial data $u_0 : \mathbb{R} \rightarrow \mathbb{R}$ are constant outside a bounded interval. Which conditions on the RK-SSM ensure the discrete conservation property

$$\sum_{j \in \mathbb{Z}} \mu_j^{(k)} = \sum_{j \in \mathbb{Z}} \mu_j^{(0)} \quad \forall k \in \mathbb{N}_0 , \quad \mu_j^{(k)} := (\vec{\mu}^{(k)})_j ?$$

△

11.4.3 Linear Stability Analysis

In Section 9.2.7.2 (parabolic evolutions) and Section 9.3.5 (linear wave equations) we found that for the method of lines combined with **explicit** timestepping

timestep constraints $\tau \leq O(h^r)$, $r \in \{1, 2\}$, *necessary* to avoid exponential blow-up (*instability*)

Is the timestep constraint (11.4.2.12) suggested by the CFL-condition also stipulated by stability requirements?

§11.4.3.1 (Focus on linear advection) We are going to investigate the question only for the Cauchy problem for scalar *linear* advection in 1D with constant velocity $v > 0$:

$$\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) \quad \forall x \in \mathbb{R}. \quad (11.1.1.11)$$

Method of lines approach: Semi-discretization in space on equidistant mesh with meshwidth $h > 0$ leads to a

➤ *linear, local, and translation-invariant* semi-discrete evolution

$$\frac{d\vec{\mu}}{dt}(t) = \mathcal{L}_h(\vec{\mu}(t)) , \quad \text{with} \quad (\mathcal{L}_h(\vec{\mu}))_j = \sum_{l=-m}^m c_l \mu_{j+l} , \quad j \in \mathbb{Z}, \quad (11.4.3.2)$$

for suitable weights $c_l \in \mathbb{R}$. This is also called a *stencil* formula, cf. § 11.4.2.1, m is the width of the stencil.

Explanation of terminology:

- *linear*: The finite difference operator $\mathcal{L}_h : \mathbb{R}^{\mathbb{Z}} \mapsto \mathbb{R}^{\mathbb{Z}}$ is linear.
- *local*: $(\mathcal{L}_h(\vec{\mu}))_j$ depends only on a few coefficients μ_{j+l} for small $|l|$, cf. page 750.
- *translation-invariant*: if $\eta_j := \mu_{j+1}$, then $(\mathcal{L}_h(\vec{\eta}))_j = (\mathcal{L}_h(\vec{\mu}))_{j+1}$ (the finite difference operator commutes with shifts of the coefficient vector, cf. page 750).

—

EXAMPLE 11.4.3.3 (Upwind difference operator for linear advection) Finite volume semi-discretization of (11.1.1.11) in conservation form with Godunov numerical flux (11.3.4.33) (, which agrees with the upwind flux (11.3.4.19) in this case)

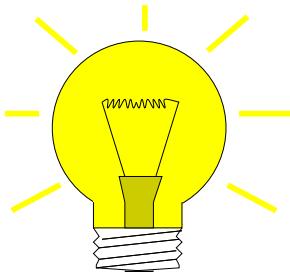
$$(\mathcal{L}_h(\vec{\mu}))_j = -\frac{v}{h}(\mu_j - \mu_{j-1}) . \quad (11.4.3.4)$$

► Coefficients in (11.4.3.2): $c_0 = -\frac{v}{h}$, $c_{-1} = \frac{v}{h}$.

Note: In this case the (loca) Lax-Friedrichs/Rusanov numerical flux (11.3.4.16) yields the same \mathcal{L}_h . —

As in Section 9.2.7.2 and Section 9.3.5 we employ a **diagonalization technique** (with a new twist). The policy was to expand the vector of unknowns of the semi-discrete evolution into eigenvectors of the “right-hand-side operator” of the method-of-lines ODE.

Now the new twist is that \mathcal{L}_h acts on the *sequence space* $\mathbb{R}^{\mathbb{Z}}$!



Idea: trial expression for “eigenvectors” (in $\mathbb{C}^{\mathbb{Z}}$)

“complex waves” $(\vec{\psi}_{\xi})_j := \exp(i\xi j)$, $j \in \mathbb{Z}$, $-\pi < \xi \leq \pi$. (11.4.3.5)

We have obtained infinite set of eigenvectors, matching $\dim \mathbb{C}^{\mathbb{Z}} = \infty$! This set is also complete.

Remark 11.4.3.6 (Diagonalization in \mathbb{C}) Why do we have to consider complex-valued eigenvectors? Well, remember from linear algebra that purely real matrices may have complex eigenvalues. Here, purely real finite difference operators have a complex spectrum! \square

By straightforward computations, using $\exp(x+y) = \exp(x)\exp(y)$, we verify the eigenvector property and compute the corresponding eigenvalues:

$$(\mathcal{L}_h(\vec{\mu}))_j = \sum_{l=-m}^m c_l \mu_{j+l} \Rightarrow \mathcal{L}_h \psi^\xi = \left(\underbrace{\sum_{l=-m}^m c_l \exp(i\xi l)}_{\text{"eigenvalue" } \hat{c}_h(\xi)} \right) \psi^\xi .$$

► spectrum of \mathcal{L}_h : $\sigma(\mathcal{L}_h) = \{\hat{c}_h(\xi) := \sum_{l=-m}^m c_l \exp(i\xi l) : -\pi < \xi \leq \pi\}$. (11.4.3.7)

Terminology: The function $\hat{c}_h(\xi)$ is known as the **symbol** of the difference operator \mathcal{L}_h , cf. the concept of **symbol of a differential operator**.

Remark 11.4.3.8 (Eigenvectors of translation invariant linear operators) In [Hip19, ??] periodic linear time-invariant filters are introduced as linear operators on the space of n -periodic sequences that commute with translation, see [Hip19, ??]. They are described by *circulant matrices*, see [Hip19, ??]. The linear difference operator \mathcal{L}_h from (11.4.3.2) generalizes this concept, because it still features translation invariance, but acts on infinite sequences. In fact, \mathcal{L}_h can be represented by means of an infinite banded circulant matrix with respect to the “unit vector basis” of the space of sequences $\mathbb{R}^{\mathbb{Z}}$

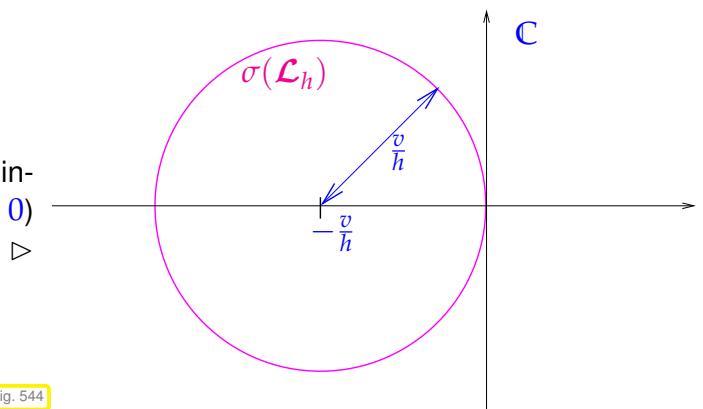
$$\mathcal{L}_h \sim \begin{bmatrix} \ddots & & & \\ & \ddots & & \\ & & \ddots & 0 \\ & & c_0 & \dots & c_m & 0 \\ 0 & c_{-m} & \dots & c_0 & \dots & c_m & 0 \\ 0 & c_{-m} & \dots & c_0 & \dots & c_m & 0 \\ & & & \ddots & & \ddots & \ddots \end{bmatrix}$$

According to [Hip19, ??] the columns of the Fourier matrix [Hip19, ??], the vectors $(\exp(\frac{2\pi jk}{n}))_{j=0}^{n-1} \in \mathbb{C}^n$, $k = 0, \dots, n-1$, provide the eigenvectors of any circulant matrix $\in \mathbb{C}^n$. The generalization of these “Fourier harmonics” to $\mathbb{R}^{\mathbb{Z}}$ are the complex waves defined in (11.4.3.5). Therefore we can expect them to furnish eigenvectors for \mathcal{L}_h . \square

EXAMPLE 11.4.3.9 (Spectrum of upwind difference operator) Apply formula (11.4.3.7) with $c_0 = -\frac{v}{h}$, $c_{-1} = \frac{v}{h}$ (from (11.4.3.4)):

For \mathcal{L}_h from (11.4.3.4): $\sigma(\mathcal{L}_h) = \left\{ \frac{v}{h} (\exp(-i\xi) - 1) : -\pi < \xi \leq \pi \right\}$

Spectrum of upwind finite difference operator for linear advection with velocity $v > 0$ (meshwidth $h > 0$) as a subset of \mathbb{C}



↓

§11.4.3.10 (Diagonalization of semi-discrete evolution) The eigenvalue $\hat{c}_h(\xi)$ will govern the evolution when we choose $\vec{\psi}_\xi$ as initial value:

$$\begin{aligned} \mathcal{L}\vec{\psi}_\xi &= \hat{c}(\xi) \vec{\psi}_\xi , \\ \frac{d\vec{\mu}}{dt}(t) &= \mathcal{L}_h(\vec{\mu}(t)) , \quad \Rightarrow \quad \vec{\mu}(t) = \exp(\hat{c}(\xi)t) \vec{\xi}_\xi , \\ \vec{\mu}(0) &= \vec{\psi}_\xi \end{aligned} \tag{11.4.3.11}$$

as can be seen by simply differentiation.

In § 9.2.7.15 the principal idea was an expansion of the time-dependent vector of unknown coefficients as a finite linear combination of eigenvector of the spatially discrete evolution operator. However, now we have to deal with uncountably many “eigenvectors” $\vec{\psi}_\xi$, $-\pi < \xi \leq \pi$, so that linear combination becomes integration over $[-\pi, \pi]$:

$$\vec{\mu}(t) = \int_{-\pi}^{\pi} \hat{\mu}(t, \xi) \vec{\psi}_\xi d\xi \Leftrightarrow \mu_j(t) = \int_{-\pi}^{\pi} \hat{\mu}(t, \xi) \exp(i\xi j) d\xi . \tag{11.4.3.12}$$

$$\Rightarrow \frac{d\vec{\mu}}{dt}(t) = \mathcal{L}_h(\vec{\mu}(t)) \Rightarrow \boxed{\frac{\partial \hat{\mu}}{\partial t}(t, \xi) = \hat{c}_h(\xi) \hat{\mu}(t, \xi)} . \tag{11.4.3.13}$$

This is a family of *decoupled* scalar, linear ODEs parameterized by $\xi \in]-\pi, \pi]$.

↓

Remark 11.4.3.14 (Fourier series → [Hip19, ??]) Up to normalization the relationship

$$\vec{\mu}^{(0)} \in \mathbb{R}^{\mathbb{Z}} \Leftrightarrow \hat{\mu}^{(0)} :]-\pi, \pi] \mapsto \mathbb{C}$$

from (11.4.3.12) is the **Fourier series transform**, which maps a sequence to a 2π -periodic function. It has the important isometry property

$$\sum_{j=-\infty}^{\infty} |\mu_j|^2 = 2\pi \int_{-\pi}^{\pi} |\hat{\mu}(\xi)|^2 d\xi .$$

➤ The symbol \hat{c}_h can be viewed as the *representation of a difference operator in Fourier domain*. ↓

The decoupling manifest in (11.4.3.13) carries over to Runge-Kutta timestepping in the sense of the commuting diagram (9.2.7.41). If Ψ^τ is the discrete evolution operator (→ Def. 9.2.6.2) induced by an s -stage

Runge-Kutta single step method according to Def. 7.3.3.1 with timestep $\tau > 0$ for the ODE $\dot{\vec{\mu}} = \mathcal{L}_h(\vec{\mu})$, \mathcal{L}_h from (11.4.3.2), then straightforward computations yield

$$\Psi^\tau \vec{\psi}_\xi = \Psi_\xi^\tau \hat{c}(\xi) \vec{\xi}_\xi , \quad (11.4.3.15)$$

where $\Psi_\xi^\tau \in \mathbb{C}$ is the (multiplication) discrete evolution operator describing the application of the same RK-SSM to the scalar ODE $\dot{\mu} = \hat{c}(\xi)\mu$.

To put these considerations into the diagonalization framework, we introduce the Fourier transforms of the members of the sequence $(\vec{\mu}^{(k)})_k$ created by Runge-Kutta timestepping

$$\vec{\mu}^{(k)} = \int_{-\pi}^{\pi} \hat{\mu}^{(k)}(\xi) \vec{\psi}_\xi d\xi \Leftrightarrow \mu_j^{(k)} = \int_{-\pi}^{\pi} \hat{\mu}^{(k)}(\xi) \exp(i\xi j) d\xi . \quad (11.4.3.16)$$

Then from (11.4.3.15), formally appealing to the linearity of \mathcal{L}_h , we conclude that

$$\vec{\mu}^{(k+1)} = \Psi^\tau \vec{\mu}^{(k)} = \int_{-\pi}^{\pi} \hat{\mu}^{(k)}(\xi) \Psi^\tau \vec{\psi}_\xi d\xi = \int_{-\pi}^{\pi} \hat{\mu}^{(k)} \Psi_\xi^\tau \vec{\psi}_\xi d\xi . \quad (11.4.3.17)$$

Hence, $\xi \mapsto \hat{\mu}^{(k)}(\xi) \Psi_\xi^\tau$ has been identified as the Fourier transform of $\vec{\mu}^{(k+1)}$ and we find

$$\hat{\mu}^{(k)} = (\Psi_\xi^\tau)^k \hat{\mu}^{(0)} , \quad k \in \mathbb{N} . \quad (11.4.3.18)$$

EXAMPLE 11.4.3.19 (Explicit Euler in Fourier domain) Let us apply the above formulas to explicit Euler timestepping (6.2.1.4) for semi-discrete evolution (11.4.3.2), see also (11.4.1.10),

$$\begin{aligned} \vec{\mu}^{(k+1)} &= \vec{\mu}^{(k)} + \tau \mathcal{L}_h \vec{\mu}^{(k)} . \\ \Rightarrow \int_{-\pi}^{\pi} \hat{\mu}^{(k+1)}(\xi) \vec{\psi}_\xi d\xi &= (\text{Id} + \tau \mathcal{L}_h) \int_{-\pi}^{\pi} \hat{\mu}^{(k)}(\xi) \vec{\psi}_\xi d\xi = \int_{-\pi}^{\pi} \hat{\mu}^{(k)}(\xi) (1 + \tau \hat{c}_h(\xi)) d\xi . \\ \Rightarrow \hat{\mu}^{(k+1)}(\xi) &= \hat{\mu}^{(k)}(\xi) (1 + \tau \hat{c}_h(\xi)) . \end{aligned}$$

In Fourier domain a single explicit Euler timestep corresponds to a multiplication of $\hat{\mu} :]-\pi, \pi] \mapsto \mathbb{C}$ with the function $(1 + \tau \hat{c}_h) :]-\pi, \pi] \mapsto \mathbb{C}$.

We get the same result when applying an explicit Euler step to the ODE $\frac{\partial \hat{\mu}}{\partial t}(t, \xi) = \hat{c}_h(\xi) \hat{\mu}(t, \xi)$ from (11.4.3.13) with parameter ξ :

$$\hat{\mu}^{(k+1)}(\xi) = (1 + \tau \hat{c}_h(\xi)) \hat{\mu}^{(k)}(\xi) .$$

□

We summarize the observation made in the previous example: For the sequence $(\vec{\mu}^{(k)})_{k \in \mathbb{N}_0}$ generated by an RK-SSM for the linear MOL-ODE (11.4.3.2) holds

$$\vec{\mu}^{(k)} = \int_{-\pi}^{\pi} \hat{\mu}^{(k)}(\xi) \vec{\psi}_\xi d\xi ,$$

where $\left(\hat{\mu}^{(k)}(\xi)\right)_{k \in \mathbb{N}_0}$ is the sequence of approximations created by the Runge-Kutta method when applied to the scalar linear initial value problem

$$\dot{y} = \hat{c}(\xi) y \quad , \quad y(0) = \hat{\mu}^{(0)}(\xi) .$$

Clearly, timestepping can only be stable, if blowup $|\hat{\mu}^{(k)}(\xi)| \rightarrow \infty$ for $k \rightarrow \infty$ can be avoided **for all** $-\pi < \xi \leq \pi$.

From Thm. 7.1.0.17 we know a rather explicit formula for the (complex) numbers Ψ_ξ^τ :

Theorem 11.4.3.20. Stability function of explicit Runge-Kutta methods

The execution of one step of size $\tau > 0$ of an explicit s -stage Runge-Kutta single step method (\rightarrow Def. 7.3.3.1) with Butcher scheme $\begin{array}{c|cc} \mathbf{c} & \mathfrak{A} \\ \hline \mathbf{b}^T & \end{array}$ (see (7.3.3.3)) for the scalar linear ODE $\dot{y} = \lambda y, \lambda \in \mathbb{C}$, amounts to a multiplication with the number

$$\Psi_\lambda^\tau = \underbrace{1 + z \mathbf{b}^T (\mathbf{I} - z \mathfrak{A})^{-1} \mathbf{1}}_{\text{stability function } S(z)} = \det(\mathbf{I} - z \mathfrak{A} + z \mathbf{1} \mathbf{b}^T), \quad z := \lambda \tau, \quad \mathbf{1} = (1, \dots, 1)^T \in \mathbb{R}^s .$$

EXAMPLE 11.4.3.21 (Stability functions of explicit RK-methods)

- Explicit Euler method (11.4.1.10) : $\begin{array}{c|c} 0 & 0 \\ \hline 1 & \end{array} \Rightarrow S(z) = 1 + z .$

- Explicit trapezoidal rule (11.4.1.11) : $\begin{array}{c|cc} 0 & 0 & 0 \\ \hline 1 & 1 & 0 \\ \hline \frac{1}{2} & \frac{1}{2} & \end{array} \Rightarrow S(z) = 1 + z + \frac{1}{2}z^2 .$

- Classical RK4-method Ex. 6.4.0.15 : $\begin{array}{c|cccc} 0 & 0 & 0 & 0 \\ \hline \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \hline \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \hline 1 & 0 & 0 & 1 \\ \hline \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array} \Rightarrow S(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4 .$

Thm. 7.1.0.17 together with the combinatorial formula for the determinant means that $\Psi_\lambda^\tau(z)$ is a polynomial of degree $\leq s$ in $z \in \mathbb{C}$. □

So we conclude for the evolution of the “Fourier transforms” $\hat{\mu}^{(k)}(\xi)$:

$$\hat{\mu}^{(k+1)}(\xi) = S(\tau \hat{c}(\xi)) \cdot \hat{\mu}^{(k)}(\xi), \quad k \in \mathbb{N}_0, \quad -\pi < \xi \leq \pi ,$$

where $z \mapsto S(z)$ is the **stability function** of the Runge-Kutta timestepping method, see Thm. 7.1.0.17. For the explicit Euler method we recover the formula of Ex. 11.4.3.19.

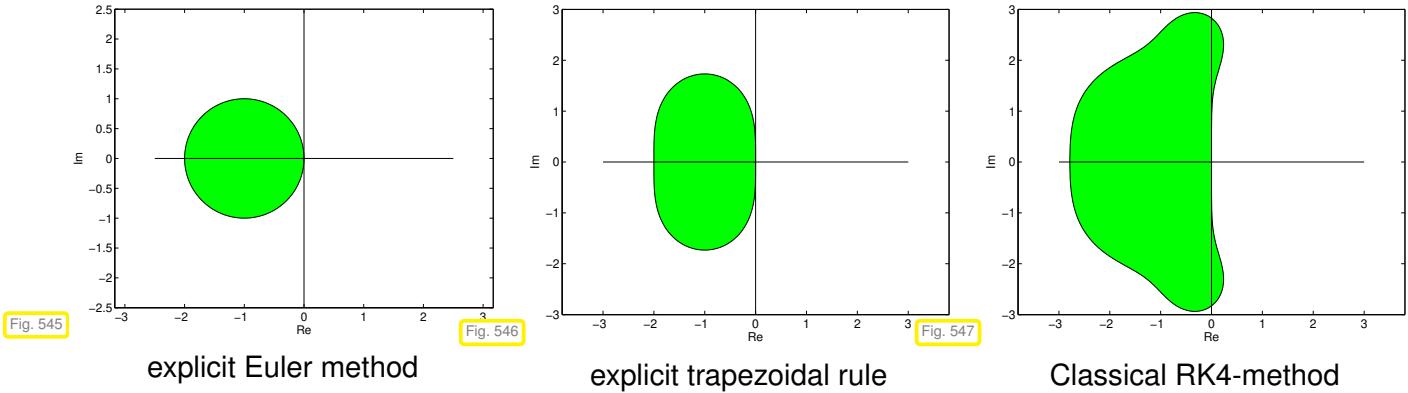
$$\text{Stability of RK-timestepping of linear semi-discrete evolution} \iff \max_{-\pi < \xi \leq \pi} |S(\tau\hat{\zeta}(\xi))| \leq 1$$

The linear stability analysis based on Fourier symbols of difference operators for Cauchy problems is often referred to as **von Neumann stability analysis**.

Remark 11.4.3.22 (Stability domains) Terminology in the theory of Runge-Kutta single step methods Def. 7.1.0.51:

$$\text{Stability domain: } \{z \in \mathbb{C}: |S(z)| \leq 1\}.$$

Stability domains:



For explicit RK-SSM the stability function $S(z)$ is a polynomial, see § 7.1.0.49. Therefore, their stability domains will invariably be **bounded** sets in \mathbb{C} .

► Necessary stability condition for RK-SSM for linear evolutions in $\mathbb{R}^{\mathbb{Z}}$:

$$\{\tau\hat{\zeta}(\xi), -\pi < \xi \leq \pi\} \subset \text{stability domain of RK-SSM}$$

EXAMPLE 11.4.3.23 (Stability and CFL condition) Consider: upwind spatial discretization (11.4.3.4) & explicit Euler timestepping

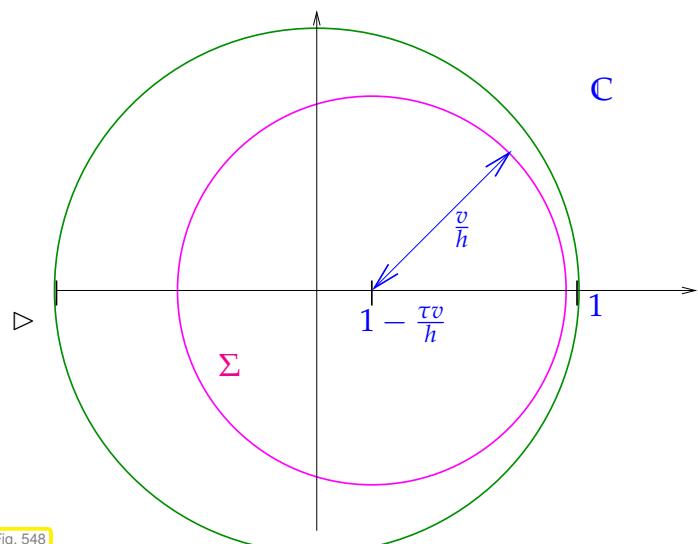
- symbol of difference operator (\rightarrow Ex. 11.4.3.9): $\hat{c}_h(\xi) = \frac{v}{h}(\exp(-i\xi) - 1)$,
- stability function: $S(z) = 1 + z$.

Locus of

$$\Sigma := S(\tau \hat{c}(\xi)), \quad -\pi < \xi \leq \pi,$$

in the complex plane

(Unit circle in green)

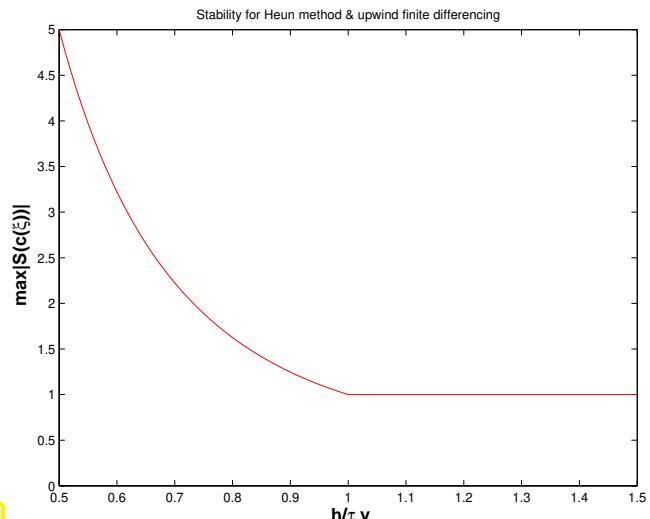
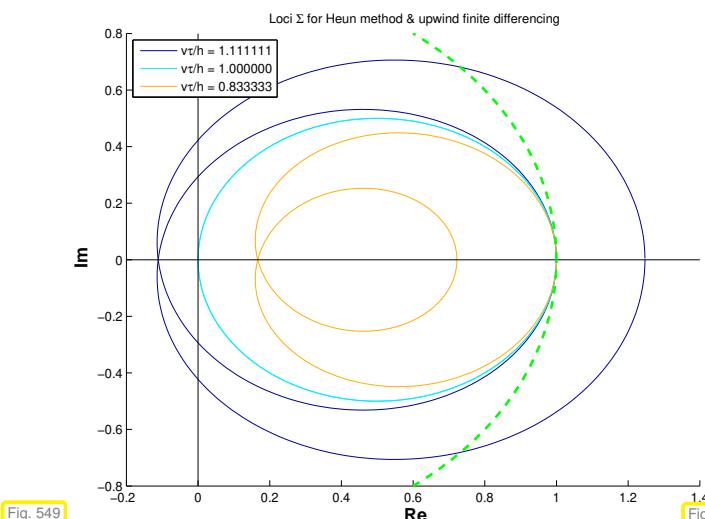


► $|S(\tau \hat{c}(\xi))| \leq 1 \quad \forall -\pi < \xi \leq \pi \iff v \frac{\tau}{h} \leq 1.$

= CFL-condition of Def. 11.4.2.11! Note that the maximal analytic region of dependence for constant velocity v linear advection is merely a line with slope v in the $x - t$ -plane, see Ex. 11.2.2.5.

Consider: upwind spatial discretization (11.4.3.4) & explicit trapezoidal rule: stability function $S(z) = 1 + z + \frac{1}{2}z^2$

Plots for $v = 1, \tau = 1$



► $|S(\tau \hat{c}(\xi))| \leq 1 \quad \forall -\pi < \xi \leq \pi \iff v \frac{\tau}{h} \leq 1.$

= *tighter timestep constraint* than stipulated by mere CFL-condition (11.4.2.12). To see this note that the explicit trapezoidal rule is a 2-stage Runge-Kutta method. Hence, the spatial stencil has width 2 in upwind direction, see Fig. 540. □

Stability induced timestep constraint

For an *explicit* Runge-Kutta single-step method applied to a linear semi-discrete evolution (11.4.3.2) the necessary stability condition $\max_{-\pi \leq \xi \leq \pi} |S(\tau \hat{c}(\xi))| \leq 1$ implies a *timestep constraint*.

Review question(s) 11.4.3.25 (Linear stability analysis of fully discrete evolutions)

(Q11.4.3.25.A) Let $\mathcal{L}_h : \mathbb{R}^{\mathbb{Z}} \rightarrow \mathbb{R}^{\mathbb{Z}}$ be a linear **translation-invariant** operator.

1. What does it mean that \mathcal{L}_h is linear?
2. Give a definition of what is meant by “translation-invariant”.
3. What is the **symbol** of \mathcal{L}_h ?

(Q11.4.3.25.B) Conduct a von Neumann stability analysis for the linear evolution

$$\dot{\mu}_j = \frac{\mu_{j+1} - 2\mu_j + \mu_{j-1}}{h^2}, \quad j \in \mathbb{Z}, \quad h > 0, \quad (11.4.3.26)$$

when

1. explicit Euler timestepping,
2. the explicit trapezoidal rule

with uniform timestep $\tau > 0$ is used for discretization in time.

△

11.4.4 Convergence of Fully Discrete FV Method

In this section we examine the asymptotic convergence of fully discrete conservative finite volume methods for scalar conservation laws as they have been introduced in Section 11.4.1. We restrict ourselves to Cauchy problems and uniform space-time grids, and study the limit $h \rightarrow 0$ ($h \doteq$ spatial meshwidth) and $\tau \rightarrow 0$ ($\tau \doteq$ timestep). Our treatment focuses on numerical tests and some heuristic local considerations.

EXPERIMENT 11.4.4.1 (Convergence of fully discrete finite volume methods for Burgers equation)

This example presents a comprehensive *empirical* investigation of the convergence of simple finite volume methods.

- ◆ Cauchy problem for Burgers equation (11.1.3.4)

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} \left(\frac{1}{2} u^2 \right) = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x), \quad x \in \mathbb{R} .$$

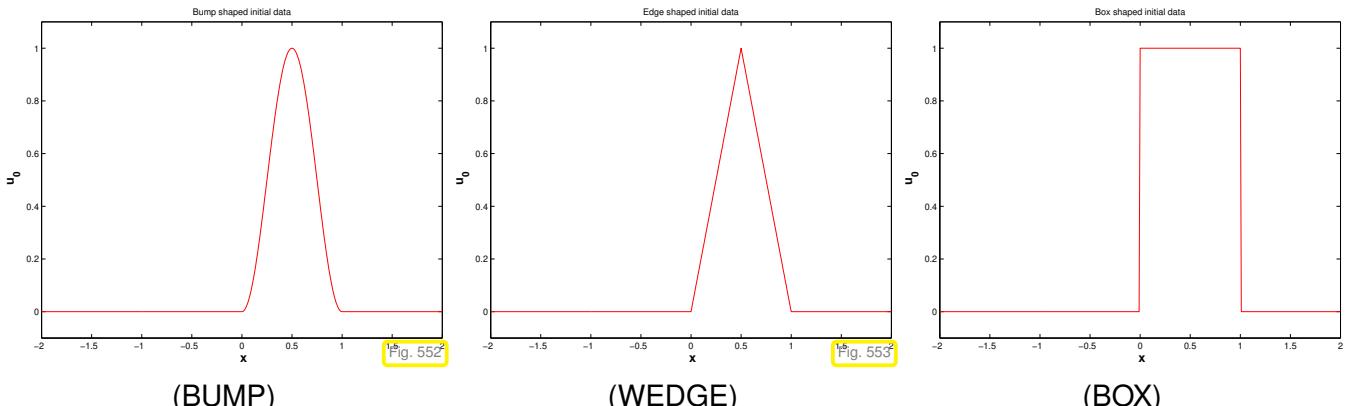
- ◆ smooth, non-smooth and discontinuous initial data, supported in $[0, 1]$:

$$u_0(x) = 1 - \cos^2(\pi x), \quad 0 \leq x \leq 1, \quad 0 \text{ elsewhere}, \quad (\text{BUMP})$$

$$u_0(x) = 1 - 2 * |x - \frac{1}{2}|, \quad 0 \leq x \leq 1, \quad 0 \text{ elsewhere}, \quad (\text{WEDGE})$$

$$u_0(x) = 1, \quad 0 \leq x \leq 1, \quad 0 \text{ elsewhere}. \quad (\text{BOX})$$

➤ maximum speed of propagation $s = 1$.



- ◆ Spatial discretization on equidistant mesh with meshwidth $h > 0$ based on finite volume method in conservation form with
 - ① (local) Lax-Friedrichs numerical flux (11.3.4.16),
 - ② Godunov numerical flux (11.3.4.33).
- ◆ Initial values $\vec{\mu}^{(0)}$ obtained from dual cell averages.
- ◆ Explicit Runge-Kutta (order 4) timestepping with uniform timestep $\tau > 0$.
- ◆ Fixed ratio: $\tau : h = 1$ (\geq CFL-condition satisfied)
- ◆ Monitored: error norms (log-log plots)

$$\text{err}_1(h) := \max_{k>0} h \sum_j |\mu_j^{(k)} - u(x_j, t_k)| \approx \max_{k>0} \|u_h^{(k)} - u(\cdot, t_k)\|_{L^1(\mathbb{R})}, \quad (11.4.4.2)$$

$$\text{err}_\infty(h) := \max_{k>0} \max_{j \in \mathbb{Z}} |\mu_j^{(k)} - u(x_j, t_k)| \approx \max_{k>0} \|u_h^{(k)} - u(\cdot, t_k)\|_{L^\infty(\mathbb{R})}. \quad (11.4.4.3)$$

for different final times $T = 0.3, 4$, $h \in \{\frac{1}{20}, \frac{1}{40}, \frac{1}{80}, \frac{1}{160}, \frac{1}{320}, \frac{1}{640}, \frac{1}{1280}\}$.

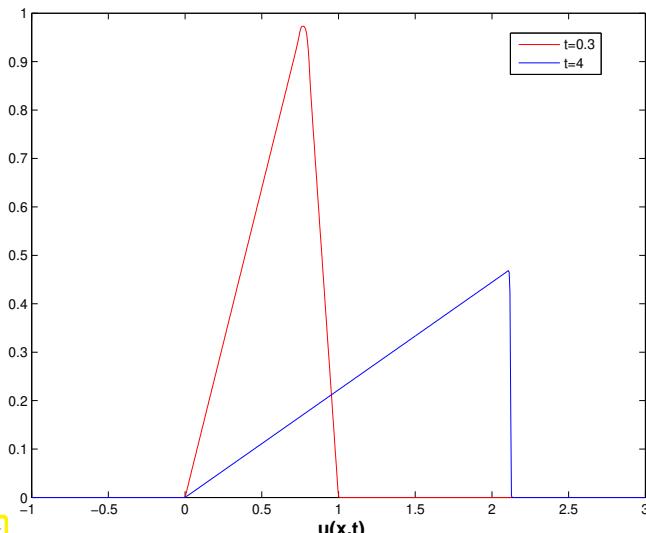


Fig. 554

“Exact solution”: Initial data (WEDGE)

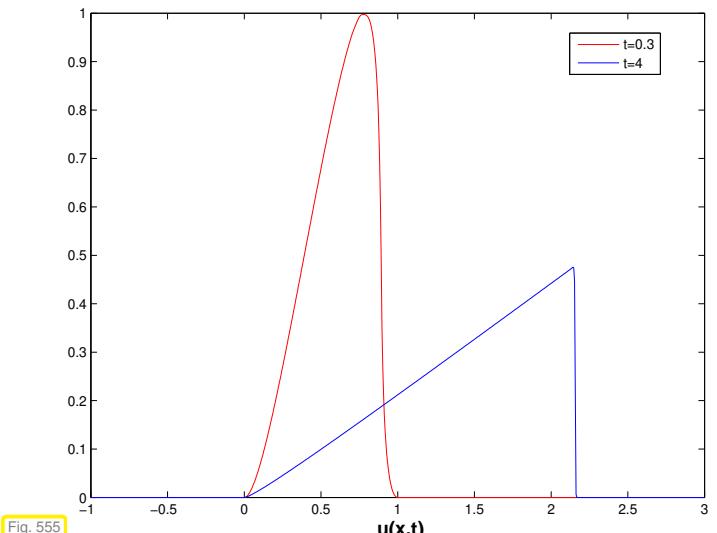


Fig. 555

“Exact solution”: Initial data (BUMP)

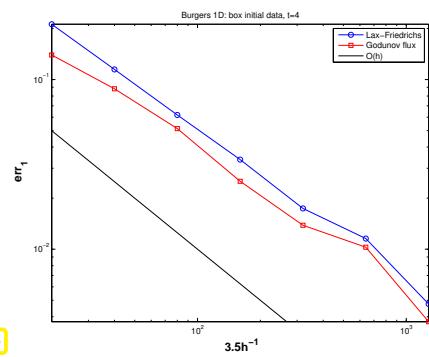
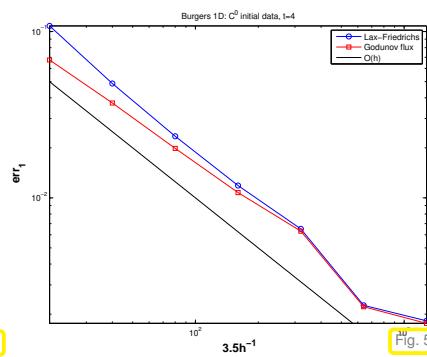
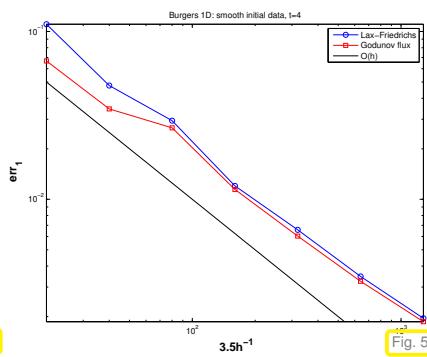
These ‘exact solutions’ were computed with a MUSCL scheme (\rightarrow Section 11.5.3) on an equidistant mesh with $h = 10^{-4}$

Note: for bump initial data (BUMP) we can still expect $u(\cdot, 0.3)$ to be smooth, because characteristics will not intersect before that time, cf. (11.2.2.7) and § 11.2.2.8.

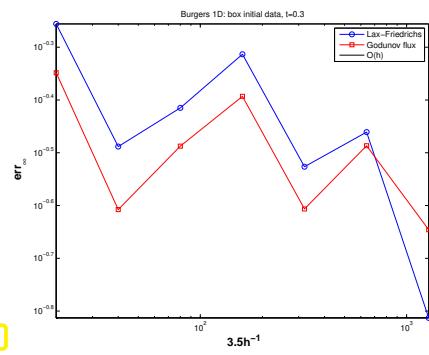
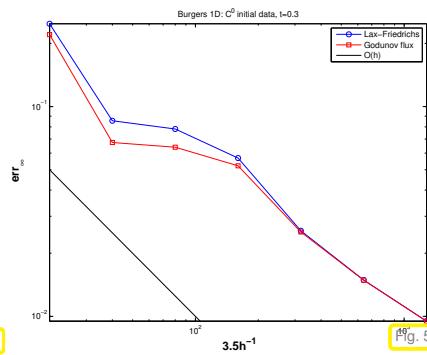
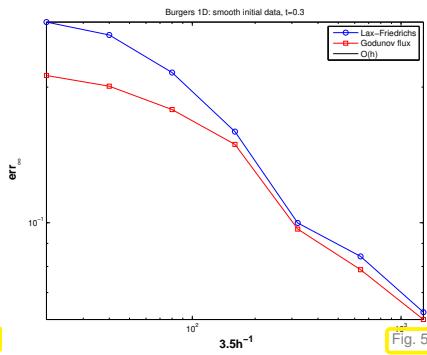
Why do we study the particular error norms (11.4.4.2) and (11.4.4.3)?

From Thm. 11.2.7.1 and Thm. 11.2.7.3 we know that the evolution for a scalar conservation law in 1D enjoys stability on the norms $\|\cdot\|_{L^1(\mathbb{R})}$ and $\|\cdot\|_{L^\infty(\mathbb{R})}$. Hence, these norms are the natural norms for measuring discretization errors, cf. the use of the energy norm for measuring the finite element discretization error for 2nd order elliptic BVP.

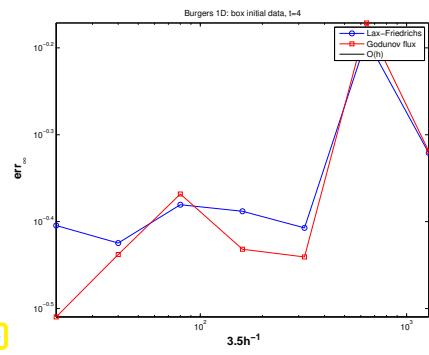
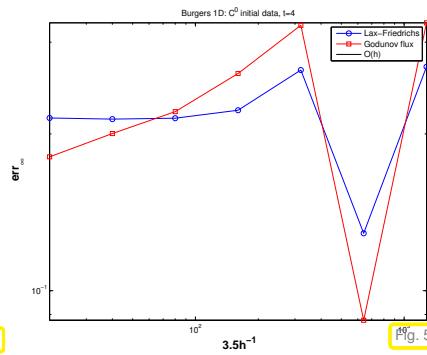
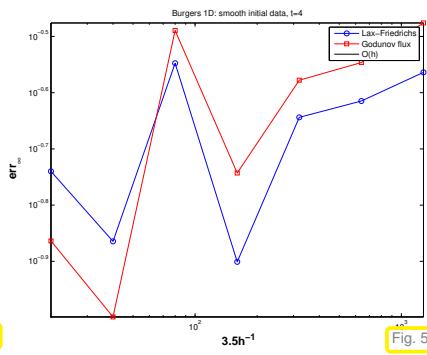
$T = 0.3$, error err_1



$T = 0.3$: error err_∞



$T = 4$: error err_∞



Error obtained by comparison with numerical “reference solution” obtained on a very fine spatio-temporal grid.

Observations: for either numerical flux function

- ◆ (near) first order algebraic convergence (\rightarrow Def. 3.2.2.1) w.r.t. mesh width h in err_1 ,
- ◆ algebraic convergence w.r.t. mesh width h in err_∞ **before** the solution develops discontinuities (shocks),
- ◆ no convergence in norm err_∞ after shock formation.



Observed throughout in numerical experiments:

The best we get is **merely first order** algebraic convergence: $O(h)$ for $h \rightarrow 0$

§11.4.4.4 (Order of consistency of numerical fluxes) Now we give a heuristic explanation for this low order of convergence:

Let $u = u(x, t)$ be a *smooth* entropy solution of the Cauchy problem

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = 0 \quad \text{in } \mathbb{R} \times]0, T[, \quad u(x, 0) = u_0(x) , \quad x \in \mathbb{R} . \quad (11.2.2.1)$$

We study the so-called **local consistency error** of the numerical flux $F = F(v, w)$

$$(\vec{\tau}_F(t))_j = F(u(x_j, t), u(x_{j+1}, t)) - f(u(x_{j+1/2}, t)) , \quad j \in \mathbb{Z} , \quad (11.4.4.5)$$

which measures the deviation of the approximate flux and the true flux, when the approximate solution agreed with the exact solution at the nodes of the mesh. Specifically, we are interested in the behavior of $(\vec{\tau}_F(t))_j$ as for meshwidth $h \rightarrow 0$, where an equidistant spatial mesh is assumed. We make use of the following terminology:

$$\max_{j \in \mathbb{Z}} (\vec{\tau}_F(t))_j = O(h^q) \quad \text{for } h \rightarrow 0 \quad \leftrightarrow \quad \text{numerical flux is consistent of order } q \in \mathbb{N} . \quad (11.4.4.6)$$

Heuristic reasoning: The order of consistency of the numerical flux function limits the (algebraic) order of convergence of (semi-discrete and fully discrete) finite volume schemes. \square

EXAMPLE 11.4.4.7 (Consistency error of upwind numerical flux) We make the assumption that the flux function is f continuously differentiable, $u_0 \geq 0$ and $f'(u) \geq 0$ for all $u \geq 0$. This rules out transonic rarefactions, that is, rarefaction fans with edges moving in opposite directions, see Fig. 528.

In this case the upwind numerical flux (11.3.4.19) agrees with the Godunov flux (11.3.4.33), see Rem. 11.3.4.34 and we have

$$F_{uw}(u(x_j, t), u(x_{j+1}, t)) = f(u(x_j), t) , \quad j \in \mathbb{Z} . \quad (11.4.4.8)$$

By successive Taylor expansion of f and u we obtain

$$\begin{aligned} \blacktriangleright (\vec{\tau}_{F_{uw}}(t))_j &= f(u(x_j, t)) - f(u(x_{j+1/2}, t)) \\ &= f'(u(x_{j+1/2}, t))(u(x_j, t) - u(x_{j+1/2}, t)) + O(|u(x_j, t) - u(x_{j+1/2}, t)|^2) \\ &= -f'(u(x_{j+1/2}, t)) \frac{\partial u}{\partial x}(x_{j+1/2}, t) \frac{1}{2}h + O(h^2) \quad \text{for } h \rightarrow 0 , \end{aligned}$$

This means that the upwind/Godunov numerical flux is (only) *first order consistent*. \square

EXAMPLE 11.4.4.9 (Consistency error of Lax-Friedrichs/Rusanov numerical flux) Again, we assume a smooth flux function f and sufficiently smooth solutions u of the Cauchy problem. From Section 11.3.4.2 we recall the definition of the (local) Lax-Friedrichs numerical flux

$$F_{LF}(v, w) = \frac{1}{2}(f(v) + f(w)) - \frac{1}{2} \max_{\min\{v, w\} \leq u \leq \max\{v, w\}} |f'(u)|(w - v) , \quad (11.3.4.16)$$

which is composed of the central flux and a diffusive flux.

We examine the consistency error for both parts separately, using Taylor expansion:

① central flux:

$$\begin{aligned} \frac{1}{2}(f(u(x_j, t)) + f(u(x_{j+1}, t))) - f(u(x_{j+1/2}, t)) \\ &= \frac{1}{2}f'(u(x_{j+1/2}, t))(u(x_j, t) - u(x_{j+1/2}, t) + u(x_{j+1}, t) - u(x_{j+1/2}, t)) + O(h^2) \quad (11.4.4.10) \\ &= \frac{1}{2}f'(u(x_{j+1/2}, t)) \left(\frac{\partial u}{\partial x}(x_{j+1/2}, t) (-\frac{1}{2}h + \frac{1}{2}h) + O(h^2) \right) + O(h^2) \\ &= O(h^2) \quad \text{for } h \rightarrow 0 . \end{aligned}$$

>

The central flux is *second order consistent*.

However, due to instability the central flux on its own is useless, see Section 11.3.4.1.

② diffusive flux part:

$$u(x_{j+1}, t) - u(x_j, t) = \frac{\partial u}{\partial x}(x_{j+1/2}, t)h + O(h^2) \quad \text{for } h \rightarrow 0 .$$



$$F_{\text{LF}}(u(x_j, t), u(x_{j+1}, t)) - f(u(x_{j+1/2}, t)) = O(h) \quad \text{for } h \rightarrow 0 ,$$

that is the Lax-Friedrichs/Rusanov numerical flux is only first order consistent, because the consistency error is dominated by the diffusive flux, which is necessary for the sake of stability. □

The observations made in the above examples are linked to a general fact:

Order barrier for monotone numerical fluxes

Monotone numerical fluxes (→ Def. 11.3.5.5) are at most first order consistent.

Review question(s) 11.4.4.12 (Convergence of fully discrete FV methods)

(Q11.4.4.12.A) [Relevant empiric error norms] When conducting convergence studies of conservative finite volume methods for Cauchy problems for scalar conservation laws on uniform space-time grids (spatial mesh width $h > 0$, timestep size $\tau > 0$) one usually examines the errors

$$\begin{aligned} \text{err}_1(h) &:= \max_{k>0} h \sum_j |\mu_j^{(k)} - u(x_j, t_k)| \approx \max_{k>0} \|u_h^{(k)} - u(\cdot, t_k)\|_{L^1(\mathbb{R})} , \\ \text{err}_\infty(h) &:= \max_{k>0} \max_{j \in \mathbb{Z}} |\mu_j^{(k)} - u(x_j, t_k)| \approx \max_{k>0} \|u_h^{(k)} - u(\cdot, t_k)\|_{L^\infty(\mathbb{R})} . \end{aligned}$$

1. In these formulas, what do $\mu_j^{(k)}$, t_k , u_h , and u stand for?
2. Why are the above error expressions considered relevant for scalar conservation laws?

(Q11.4.4.12.B) [Local consistency error] The local consistency error of a 2-point numerical flux $F = F(v, w)$ associated with the flux function $f : \mathbb{R} \rightarrow \mathbb{R}$ is defined as

$$(\vec{\tau}_F(t))_j = F(u(x_j, t), u(x_{j+1}, t)) - f(u(x_{j+1/2}, t)) , \quad j \in \mathbb{Z} , \quad (11.4.4.5)$$

What is the consistency order of F , how is it computed, and what assumption limits its predictive power for the actual convergence of a conservative finite-volume method for a Cauchy problem for a scalar conservation law?

<

11.5 Higher-Order Conservative Finite-Volume Schemes

All the finite-volume schemes that we have seen so far are of first order, that is, at best they achieve asymptotic algebraic convergence $O(h + \tau)$ for $h, \tau \rightarrow 0$. From earlier considerations we know that method of

higher order are desirable from the perspective of efficiency. This section pursues idea that will lead to finite-volume method that often enjoy faster asymptotic algebraic convergence.

Formally, high-order conservative finite volume methods are distinguished by numerical flux functions that are consistent of order ≥ 2 , see (11.4.4.6).

However, solutions of (systems of) conservation laws will usually not even be continuous (because of shocks emerging even in the case of smooth u_0 , see (11.2.2.8)), let alone smooth, so that the formal order of consistency may not have any bearing for the (rate of) convergence observed for the method for a concrete Cauchy problem.

Therefore in the field of numerics of conservation laws “high-order” is desired not so much for the promise of higher rates of convergence, but for the following advantages:

- ◆ for the same spatial resolution, high-order methods frequently provide more accurate solutions in the sense of global error norms as first-order methods,
- ◆ high-order methods often provide *better resolution of local features* of the solution (shocks, etc.).

In standard semi-discrete finite volume schemes in conservation form for 2-point numerical flux function $F = F(v, w)$,

$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h}(F(\mu_j(t), \mu_{j+1}(t)) - F(\mu_{j-1}(t), \mu_j(t))), \quad j \in \mathbb{Z}, \quad (11.3.2.8)$$

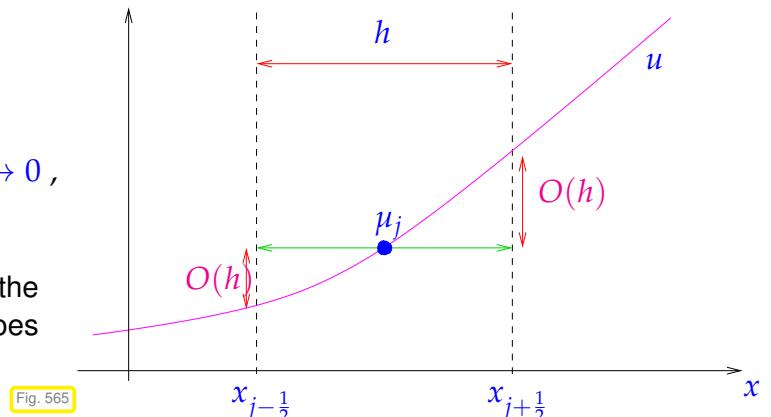
the numerical flux function is evaluated for the cell averages μ_j , which can be read as approximate values of a projection of the exact solution onto piecewise constant functions (on dual cells)

$$\mu_j(t) \approx \frac{1}{h} \int_{x_{j-1/2}}^{x_{j+1/2}} u(x, t) dx. \quad (11.3.2.2)$$

By Taylor expansion we find for $u(\cdot, t) \in C^1(\mathbb{R})$

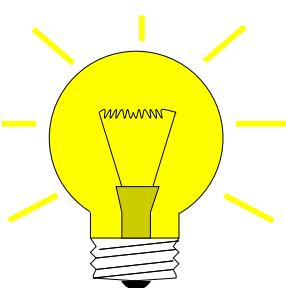
$$u(x_{j+1/2}, t) - \frac{1}{h} \int_{x_{j-1/2}}^{x_{j+1/2}} u(x, t) dx = O(h) \quad \text{for } h \rightarrow 0,$$

and, unless some lucky cancellation occurs as in the case of the central flux, see Ex. 11.4.4.9, this does not allow more than first order consistency.



11.5.1 Piecewise Linear Reconstruction

How to upgrade (11.3.2.8) without destroying the conservative structure?



Idea: Plug “better” approximations of $u(x_{j\pm 1/2}, t)$ into numerical flux function in (11.3.2.8)

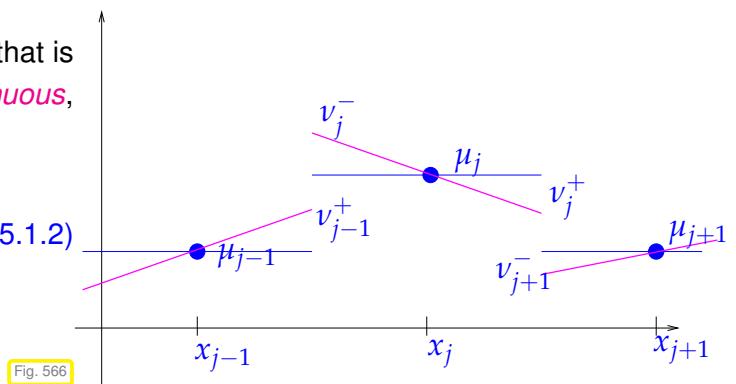
$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h}(F(v_j^+(t), v_{j+1}^-(t)) - F(v_{j-1}^+(t), v_j^-(t))), \quad j \in \mathbb{Z}, \quad (11.5.1.1)$$

where v_j^\pm are obtained by **piecewise linear reconstruction** from the (dual) cell values μ_j .

The values v_j^- and v_j^+ are those of a function that is *piecewise linear* on the dual cells, but *discontinuous*, plotted as — in Fig. 567.

$$\begin{aligned} v_j^-(t) &:= \mu_j(t) - \frac{1}{2}h\sigma_j(t), \\ v_j^+(t) &:= \mu_j(t) + \frac{1}{2}h\sigma_j(t), \end{aligned} \quad j \in \mathbb{Z}, \quad (11.5.1.2)$$

with suitable **slopes** $\sigma_j(t) = \sigma(\vec{\mu}(t))$.



Analogy: piecewise cubic Hermite interpolation with reconstructed slopes, discussed in the context of *shape preserving interpolation* in [Hip19, ??]. However, we do not aim for smooth functions now as is clear from Fig. 567.

Definition 11.5.1.3. Linear reconstruction

Given an (infinite) mesh $\mathcal{M} := \{[x_{j-1}, x_j]\}_{j \in \mathbb{Z}}$ ($x_{j-1} < x_j$), a **linear reconstruction operator** $R_{\mathcal{M}}$ is a mapping (not necessarily linear)

$$R_{\mathcal{M}} : \mathbb{R}^{\mathbb{Z}} \mapsto \{v \in L^\infty(\mathbb{R}) : v \text{ linear on }]x_{j-1/2}, x_{j+1/2}[\forall j \in \mathbb{Z}\},$$

taking a sequence $\vec{\mu} \in \mathbb{R}^{\mathbb{Z}}$ of cell averages to a possibly *discontinuous* function $R_{\mathcal{M}}\vec{\mu}$ that is piecewise linear on dual cells.

Combining linear reconstruction and (11.5.1.1) we get a semi-discrete evolution in conservation form, similar to (11.3.2.7). In particular, for a 2-point numerical flux $F = F(v, w)$ we obtain the following semi-discrete evolution ($\hat{=}$ method-of-lines ODE)

$$\begin{aligned} \frac{d\mu_j}{dt}(t) &= -\frac{1}{h} \left(F(R_{\mathcal{M}}(\vec{\mu}(t))(x_{j+1/2}^+), R_{\mathcal{M}}(\vec{\mu}(t))(x_{j+1/2}^-)) - \right. \\ &\quad \left. F(R_{\mathcal{M}}(\vec{\mu}(t))(x_{j-1/2}^-), R_{\mathcal{M}}(\vec{\mu}(t))(x_{j-1/2}^+)) \right), \quad j \in \mathbb{Z}. \end{aligned} \quad (11.5.1.4)$$

The superscripts '+' and '-' designate limits from below/left and above/right. Compare this formula with the basic semi-discrete conservation law

$$\frac{d\mu_j}{dt}(t) = -\frac{1}{h} (F(\mu_j(t), \mu_{j+1}(t)) - F(\mu_{j-1}(t), \mu_j(t))), \quad j \in \mathbb{Z}, \quad (11.3.2.8)$$

and (11.5.1.1), (11.5.1.2).

C++/EIGEN code 11.5.1.5: Operator \mathcal{L}_h for spatial semidiscretization with conservative FV with linear reconstruction and 2-point numerical flux \rightarrow GITLAB

```

2 // arguments:
3 // double h: meshwidth of equidistant spatial grid
4 // Vector mu: (finite) vector  $\vec{\mu}$  of cell averages
5 // Functor F: 2-point numerical flux function  $F = F(v, w)$ 
6 // Functor slope: slope reconstruction function
7 //  $h \cdot \sigma_j = \text{slopes}(\mu_{j-1}, \mu_j, \mu_{j+1})$ 
8 //
9 // returns a vector with differences of numerical fluxes,
10 // which supplies the right-hand side for the FV-MOL ODE
11 //
12 // Function that realizes the right hand side operator  $\mathcal{L}_h$  for the

```

```

13 // ODE (11.4.1.4) arising from conservative finite volume
14 // semidiscretization of the Cauchy problem for a 1D scalar
15 // conservation law (11.2.2.1).
16 template <typename FunctionF, typename FunctionSlopes>
17 Eigen::VectorXd slopelimfluxdiff(const Eigen::VectorXd &mu, FunctionF &&F,
18                                     FunctionSlopes &&slopes) {
19     unsigned n = mu.size(); // Number of active dual grid cells
20     Eigen::VectorXd sigma = Eigen::VectorXd::Zero(n); // Vector of slopes
21     Eigen::VectorXd fd = Eigen::VectorXd::Zero(n);
22
23     // Computation of slopes  $\sigma_j$ , uses  $\mu_0 = \mu_1$ ,
24     //  $m_{N+1} = \mu_N$ , which amounts to constant extension of states
25     // beyond domain of influence  $[a, b]$  of non-constant initial data. Same
26     // technique has been applied in Code 11.3.2.9
27     sigma[0] = slopes(mu[0], mu[0], mu[1]);
28     for (unsigned j = 1; j < n - 1; ++j)
29         sigma[j] = slopes(mu[j - 1], mu[j], mu[j + 1]);
30     sigma[n - 1] = slopes(mu[n - 2], mu[n - 1], mu[n - 1]);
31
32     // Compute linear reconstruction at endpoints of dual cells (11.5.1.2)
33     Eigen::VectorXd nup = mu + 0.5 * sigma;
34     Eigen::VectorXd num = mu - 0.5 * sigma;
35
36     // As in Code 11.3.2.10: employ constant continuation of data
37     // outside  $[a, b]$ !
38     fd[0] = F(nup[0], num[1]) - F(mu[0], num[0]);
39     for (unsigned j = 1; j < n - 1; ++j)
40         fd[j] =
41             F(nup[j], num[j + 1]) - F(nup[j - 1], num[j]); // see (11.3.2.8)
42     fd[n - 1] = F(nup[n - 1], mu[n - 1]) - F(nup[n - 2], num[n - 1]);
43     return fd;
44 }

```

We stress that the `slopes` functor has to return the slopes σ_j scaled by the (global) meshwidth. Hence, this implementation is suitable only for equidistant meshes.

C++ EIGEN code 11.5.1.6: Conservative FV with linear reconstruction: Explicit adaptive Runge-Kutta “ode45” timestepping → GITLAB

```

2 // Arguments:
3 // real numbers  $a, b$ .  $a < b$ , the boundaries of the domain
4 // unsigned int  $N$  the number of grid cells
5 // Functor  $u_0 : \mathbb{R} \mapsto \mathbb{R}$ : initial data
6 // real number  $T > 0$ : final time
7 // Functor  $F = F(v, w)$ : 2-point numerical flux function
8 // Functor  $slopes = \sigma(v, u, w)$ : 3-point slope reconstruction rule
9 // (Note: no division by  $h$  needs to be done in slope computation
10 //
11 // returns:
12 // Vector of cell averages at final time
13 //
14 // finite volume discrete evolution in conservation form with linear
15 // reconstruction, see (11.5.1.4).
16 // Cauchy problem over time  $[0, T]$ , equidistant mesh
17 template <typename FunctionU0, typename FunctionF, typename FnSlopes>
18 Eigen::VectorXd highreavl(double a, double b, unsigned N, FunctionU0 &&u0,
19                           double T, FunctionF &&F, FnSlopes &&slopes) {
20     double h = (b - a) / N; // mesh width
21     // positions of grid points

```

```

22 Eigen::VectorXd x = Eigen::VectorXd::LinSpaced(N, a + 0.5 * h, b - 0.5 * h);
23 // vector of initial cell averages (column vector) from sampling  $u_0$ 
24 Eigen::VectorXd mu0 = x.unaryExpr(u0);
25
26 // right hand side lambda function for ODE solver
27 auto odefun = [&](const Eigen::VectorXd &mu, Eigen::VectorXd &dmdt,
28                    double t) {
29     dmdt = -1. / h * slopeLimFluxDiff<FunctionF, FnSlopes>(mu, F, slopes);
30 }
31
32 // timestepping by explicit adaptive Runge-Kutta single-step
33 // method of order 5. Adaptivity control according to [Hip19, ??]
34 double abstol = 1E-8, reltol = 1E-6; // integration control parameters
35 // std::vector<double> t; Temporal adaptive integration mesh
36 // std::vector<Eigen::VectorXd> MU; Returns states  $\vec{\mu}^{(k)}$ 
37 auto [t, MU] = ode45(odefun, 0, T, mu0, abstol, reltol);
38 // Extract state vector for final time
39 return MU.back();
40 }
```

Now, let us continue with more concrete linear reconstructions. We start with a “natural” choice, the so-called **central slope** (averaged slope)

$$\sigma_j(t) = \frac{1}{2} \left(\frac{\mu_{j+1}(t) - \mu_j(t)}{h} + \frac{\mu_j(t) - \mu_{j-1}(t)}{h} \right) = \frac{1}{2} \frac{\mu_{j+1}(t) - \mu_{j-1}(t)}{h}. \quad (11.5.1.7)$$

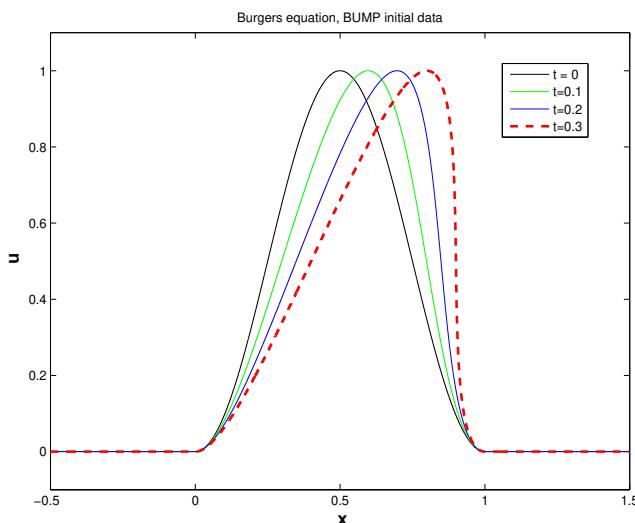
By Taylor expansion: for $u \in C^2$ (that is, u sufficiently smooth), central slope (11.5.1.7), v_j^\pm according to (11.5.1.2)

$$|v_j^-(t) - u(x_{j-1/2}, t)|, |v_j^+(t) - u(x_{j+1/2}, t)| = O(h^2).$$

EXPERIMENT 11.5.1.8 (Convergence of FV with linear reconstruction)

- ◆ Cauchy problem for Burgers equation (11.1.3.4) (flux function $f(u) = \frac{1}{2}u^2$) from Ex. 11.2.6.4 with C^1 bump initial data (BUMP)
- ◆ Equidistant spatial mesh with meshwidth h of spatial interval $[-0.5, 1.5]$
- ◆ Linear reconstruction with central slope (11.5.1.7)
- ◆ Godunov numerical flux (11.3.4.33): $F = F_{\text{GD}}$
- ◆ 2n-order Runge-Kutta timestepping (method of Heun), timestep $\tau = 0.5h$ (“CFL = 0.5”)

Monitored: Approximate L^1 - and L^∞ -norms of error at final time $T = 0.3$ (exact solution still *smooth* at this time, see Exp. 11.4.4.1)



▷ “exact solution”

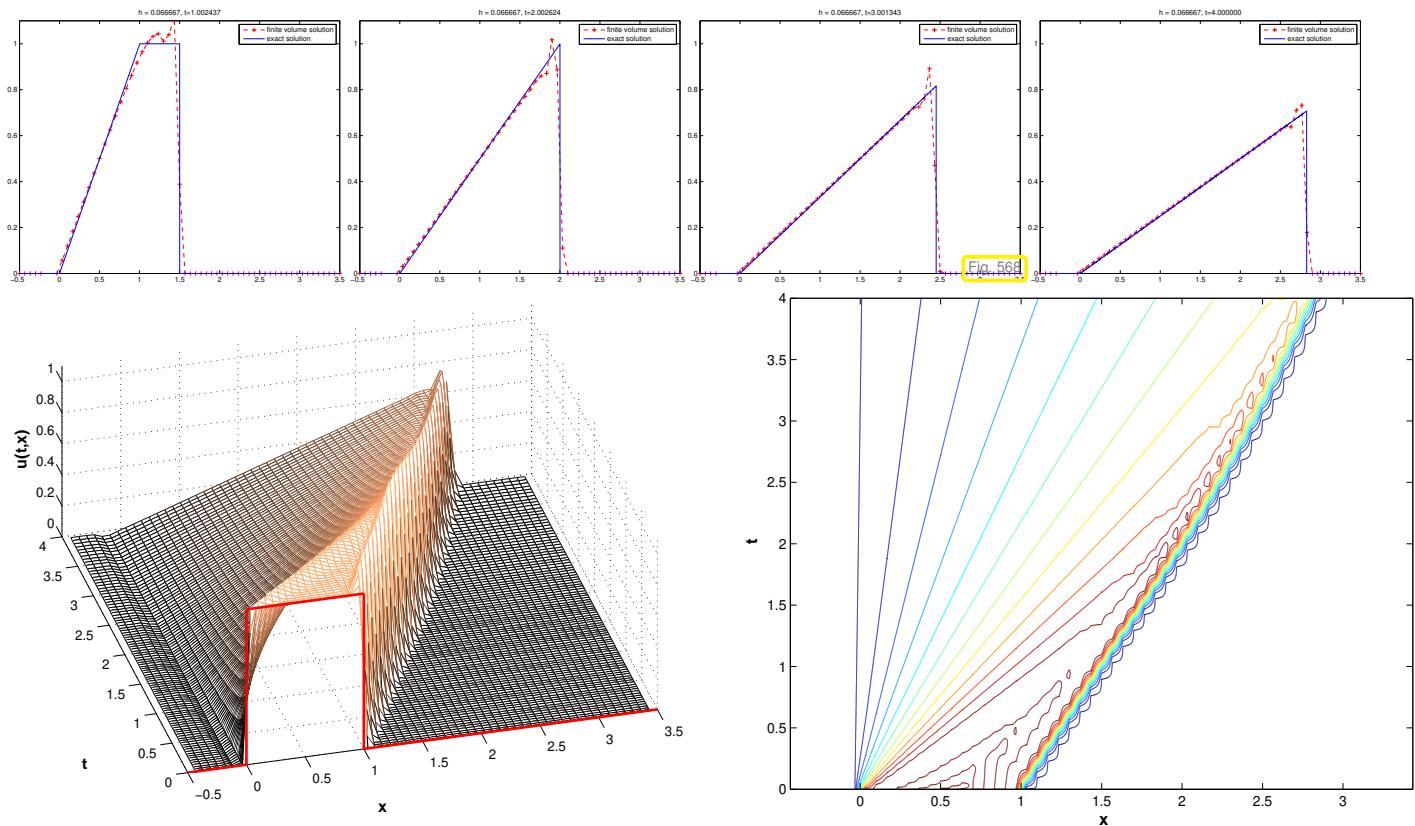
computed by means of a high-order finite volume method (WENO) on a equidistant mesh with 2^{14} points., U. Fjordholm (SAM)

Observation: 2nd-order convergence in both norms

EXPERIMENT 11.5.1.9 (Linear reconstruction with central slope (Burgers' equation))

We revisit the Cauchy problem of Exp. 11.3.4.2:

- ◆ Cauchy problem for Burgers equation (11.1.3.4) (flux function $f(u) = \frac{1}{2}u^2$) from Ex. 11.2.6.4 (“box” initial data)
- ◆ Equidistant spatial mesh with meshwidth $h = \frac{1}{15}$ covering spatial interval $[-0.5, 3.5]$
- ◆ Linear reconstruction with central slope (11.5.1.7)
- ◆ Godunov numerical flux (11.3.4.33): $F = F_{\text{GD}}$
- ◆ timestepping based on adaptive 5th-order explicit Runge-Kutta method with “overkill tolerances”



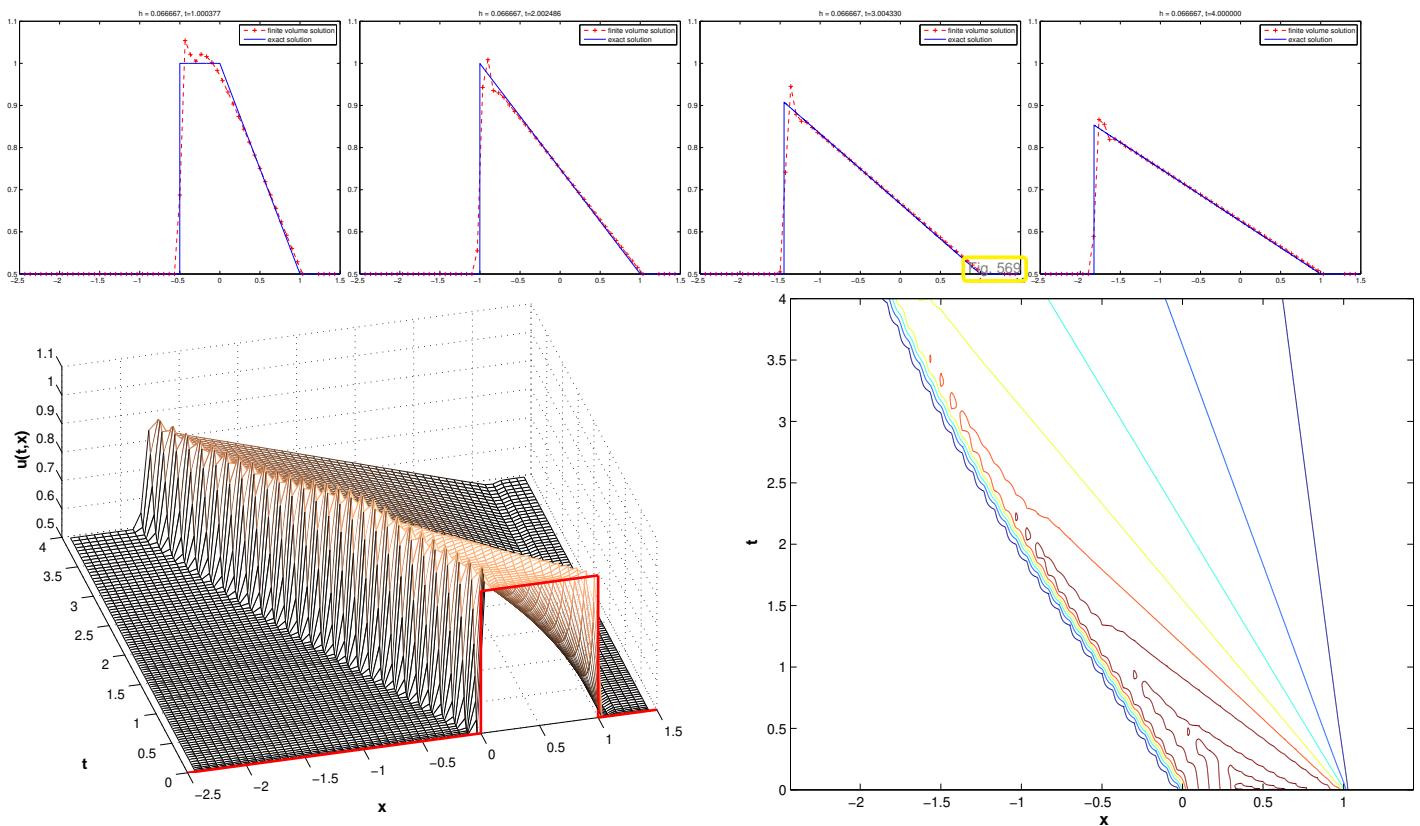
Emergence of spurious oscillations in the vicinity of shock (in violation of structural properties of the exact solution, see (11.2.7.2).)

Compare: Oscillations occurring in FV schemes relying on central flux, see Exp. 11.3.4.2. □

EXPERIMENT 11.5.1.10 (Linear reconstruction with central slope (traffic flow))

We adapt the previous experiment to the Cauchy problem from Exp. 11.3.4.3:

- ◆ Cauchy problem for Traffic Flow equation (11.1.2.23) (flux function $f(u) = u(1-u)$) from Ex. 11.2.6.5 (“box” intial data)
- ◆ Equidistant spatial mesh of $-0.5, 3.5$ with meshwidth $h = \frac{1}{15}$
- ◆ Linear reconstruction with central slope (11.5.1.7)
- ◆ Godunov numerical flux (11.3.4.33): $F = F_{\text{GD}}$
- ◆ timestepping based on adaptive 5th-order Runge-Kutta method (“ode45”). “overkill tolerances”



Emergence of spurious oscillations in the vicinity of shock (in violation of structural properties of the exact solution, see (11.2.7.2).)

Compare: Oscillations occurring in FV schemes relying on central flux, see Exp. 11.3.4.3. □

In Exp. 11.3.4.2, Exp. 10.2.2.4, the spurious oscillations can be blamed on the unstable central flux/central finite differences. Maybe, this time the central slope formula is the culprit. Thus, we investigate slope reconstruction connected with backward and forward difference quotients.

EXPERIMENT 11.5.1.11 (Linear reconstruction with one-sided slopes (Burgers' equation))

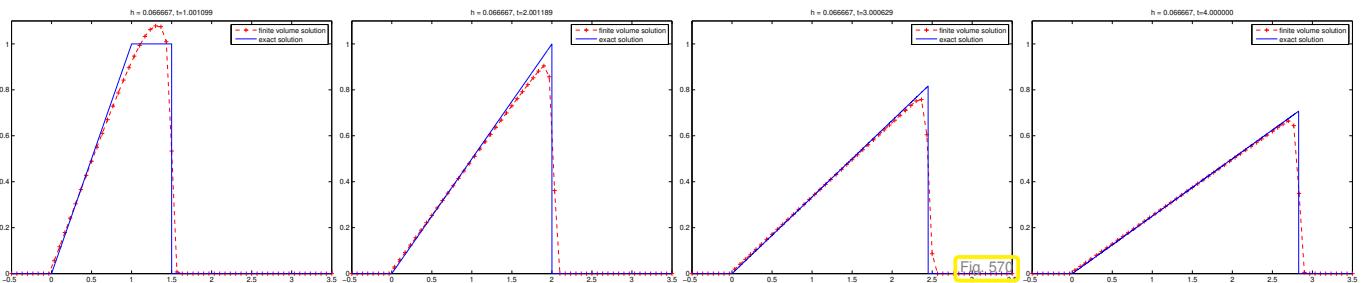
One-sided slopes for use in (11.5.1.2)

$$\text{Right slope: } \sigma_j(t) = \frac{\mu_{j+1}(t) - \mu_j(t)}{h}, \quad (11.5.1.12)$$

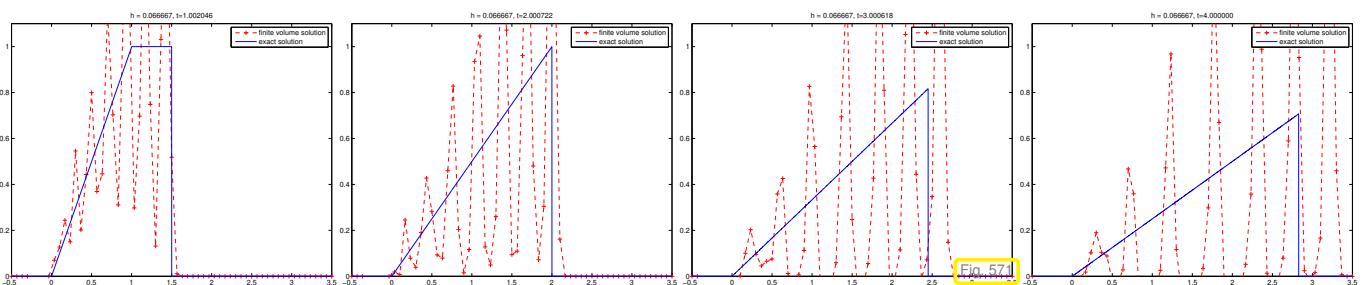
$$\text{Left slope: } \sigma_j(t) = \frac{\mu_j(t) - \mu_{j-1}(t)}{h} . \quad (11.5.1.13)$$

Same setting as in Exp. 11.5.1.9, with central slope replaced with one-sided slopes.

Left slope:

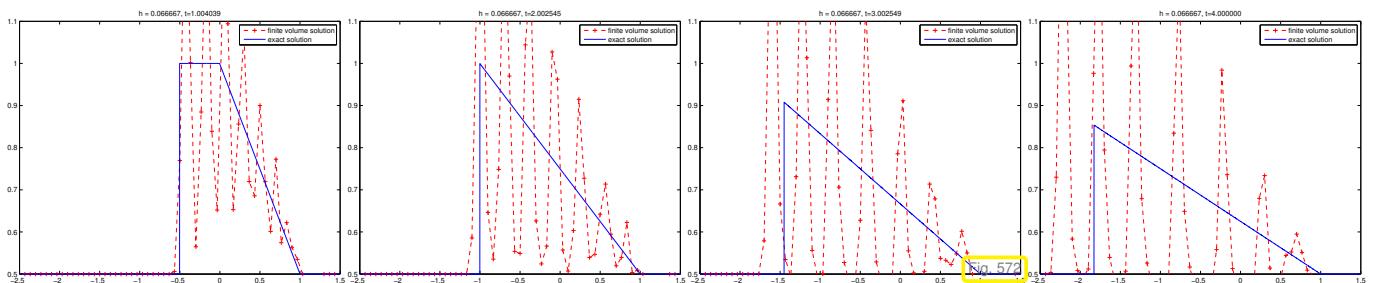


Right slope:

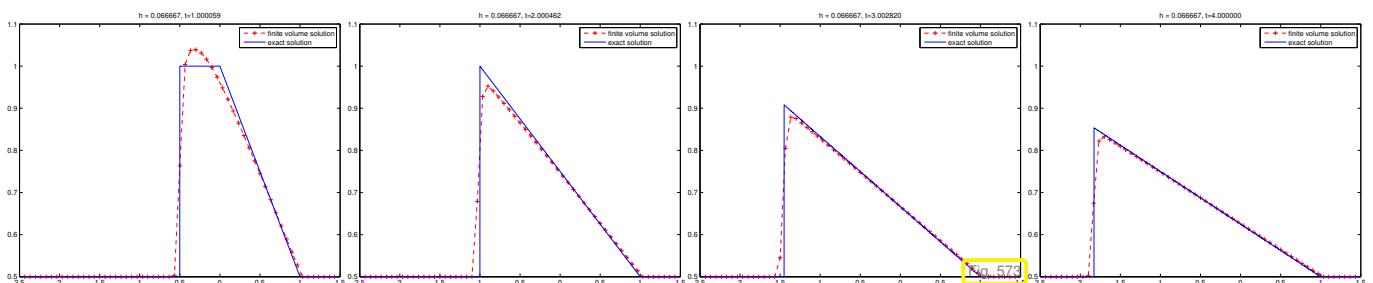


Observation: spurious oscillations/overshoots, massive and global for (11.5.1.12), moderate close to shock for (11.5.1.13). □

EXPERIMENT 11.5.1.14 (Linear reconstruction with one-sided slopes (traffic flow)) Left slope:



Right slope:



Observation: spurious oscillations/overshoots, massive and global for (11.5.1.12), moderate close to shock for (11.5.1.13). □

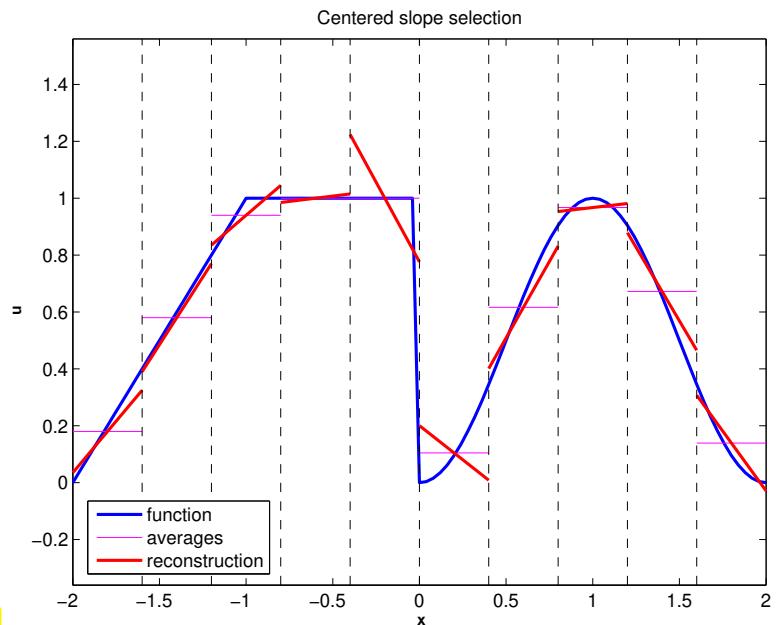
It seems to be the very process of linear reconstruction that triggers oscillations near shocks. These oscillations can be traced back to “overshooting” of linear reconstruction at jumps.

EXAMPLE 11.5.1.15 (Over-/Undershoots in linear reconstruction) In this example we apply the slope formulas proposed above to particular “synthetic cell averages” derived from a function (blue graph in plots) featuring a jump, a kink, and a local maximum.

Slope from central differencing:

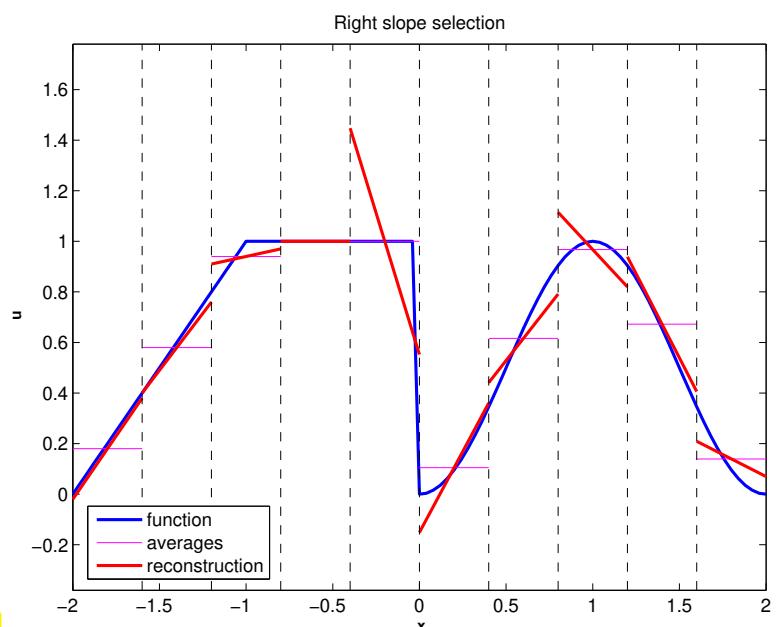
$$\sigma_j = \frac{1}{2h}(\mu_{j+1} - \mu_{j-1}) . \quad (11.5.1.7)$$

(— $\hat{=}$ cell averages, — $\hat{=}$ piecewise linear reconstruction)



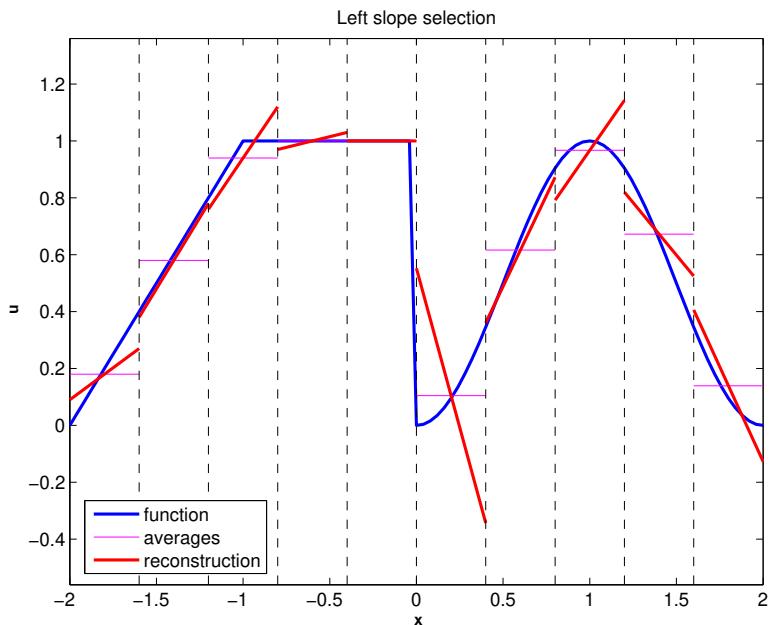
Slope from forward differencing:

$$\sigma_j = \frac{1}{h}(\mu_{j+1} - \mu_j) . \quad (11.5.1.12)$$



Slope from backward differencing:

$$\sigma_j = \frac{1}{h}(\mu_j - \mu_{j-1}) . \quad (11.5.1.13)$$



We observe that the piecewise linear reconstruction develops over- and undershoots regardless of the slope formula used. \square

11.5.2 Slope Limiting

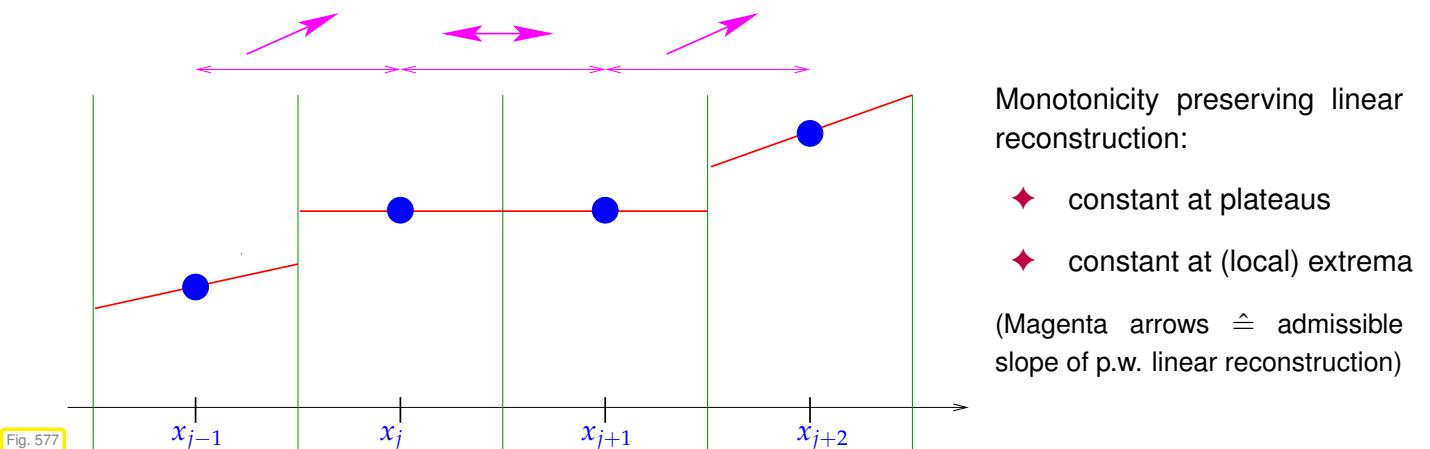
We want to find a piecewise linear reconstruction method with a guarantee for the suppression of “over-/undershoots” (\rightarrow Fig. 575, Fig. 576, Fig. 577).

Use local **monotonicity preservation** of linear reconstruction

Definition 11.5.2.1. Monotonicity-preserving linear reconstruction (MPLR)

An linear reconstruction operator R_M (\rightarrow Def. 11.5.1.3) is **monotonicity preserving**, if

$$(R_M \vec{\mu})(x_j) = \mu_j \wedge \begin{cases} \mu_j \leq \mu_{j+1} & \Rightarrow R_M \vec{\mu} \text{ non-decreasing in }]x_j, x_{j+1}[, \\ \mu_j \geq \mu_{j+1} & \Rightarrow R_M \vec{\mu} \text{ non-increasing in }]x_j, x_{j+1}[. \end{cases}$$



Related: **shape preserving** Hermite interpolation, see [Hip19, ??], achieved by using

- ◆ zero slope, in case of local slopes with opposite sign, see [Hip19, ??],
- ◆ harmonic averaging of local slopes, see [Hip19, ??].

Remark 11.5.2.2 (Consequence of monotonicity preservation) A monotonicity preserving linear reconstruction operator R_M (\rightarrow Def. 11.5.2.1)

- respects the range of cell averages

$$\min\{\mu_k, \mu_{k+1}, \dots, \mu_m\} \leq (R_M \vec{\mu})(x) \leq \max\{\mu_k, \mu_{k+1}, \dots, \mu_m\}, \quad x_k < x < x_m. \quad (11.5.2.3)$$

\leftrightarrow “range preservation” by entropy solutions, see Thm. 11.2.7.1.

- does not allow the creation of new extrema

$$\#\{\text{extrema of } R_M \vec{\mu}\} \leq \#\{\text{extrema of } \vec{\mu}\}. \quad (11.5.2.4)$$

\leftrightarrow preservation of number of extrema in entropy solution, Section 11.2.7.

□

Remark 11.5.2.5 (Linearity and monotonicity preservation) The linear reconstruction operators (\rightarrow Def. 11.5.1.3) based on the slope formulas (11.5.1.7) (central slope), (11.5.1.12) (forward slope), (11.5.1.13) (backward slope) are *linear* in the sense that

$$R_M(\alpha \vec{\mu} + \beta \vec{v}) = \alpha R_M(\vec{\mu}) + \beta R_M(\vec{v}) \quad \forall \vec{\mu}, \vec{v} \in \mathbb{R}^Z, \alpha, \beta \in \mathbb{R}. \quad (11.5.2.6)$$

Lemma 11.5.2.7. Linear monotonicity preserving reconstruction trivial

Every linear, monotonicity preserving (\rightarrow Def. 11.5.2.1) linear reconstruction yields piecewise constant functions.

Proof. Define $\vec{e}^k \in \mathbb{R}^Z$, $k \in \mathbb{Z}$, by

$$\epsilon_j^k = \begin{cases} 1 & \text{for } k = j, \\ 0 & \text{else.} \end{cases}$$

The \vec{e}^k form a basis of \mathbb{R}^Z . Thus, due to linearity, R_M is fixed by its action on the basis vectors \vec{e}^k and its image is spanned by $\{R_M \vec{e}^k\}_{k \in \mathbb{Z}}$.

However, monotonicity preservation entails that $R_M \vec{e}^k$ is piecewise constant, see Fig. 578. □



Necessary (for monotonicity preservation):

Non-linear linear reconstruction



□

A simple consideration, see Fig. 578

$$\mu_{j-1} \leq \mu_j \quad \text{and} \quad \mu_j \geq \mu_{j+1} \Rightarrow R_M \vec{\mu} \equiv \text{const} \quad \text{on } [x_{j-1/2}, x_{j+1/2}], \quad (11.5.2.8)$$

for any monotonicity preserving (\rightarrow Def. 11.5.2.1) linear reconstruction operator R_M (\rightarrow Def. 11.5.1.3).

➤ monotonicity preserving linear reconstruction $R_M \vec{\mu}$ must be constant at local extrema of $\vec{\mu}$!

Definition 11.5.2.9. Minmod reconstruction

The **minmod reconstruction** R_{mm} is a piecewise linear reconstruction (\rightarrow Def. 11.5.1.3) defined by

$$(R_{\text{mm}} \vec{u})(x) = \mu_j + \sigma_j(x - x_j)$$

for $x_{j-1/2} < x < x_{j+1/2}, j \in \mathbb{Z}$,

$$\sigma_j := \text{minmod}\left(\frac{\mu_{j+1} - \mu_j}{x_{j+1} - x_j}, \frac{\mu_j - \mu_{j-1}}{x_j - x_{j-1}}\right),$$

$$\text{minmod}(v, w) := \begin{cases} v & , vw > 0, |v| < |w|, \\ w & , vw > 0, |w| < |v|, \\ 0 & , vw \leq 0 . \end{cases}$$

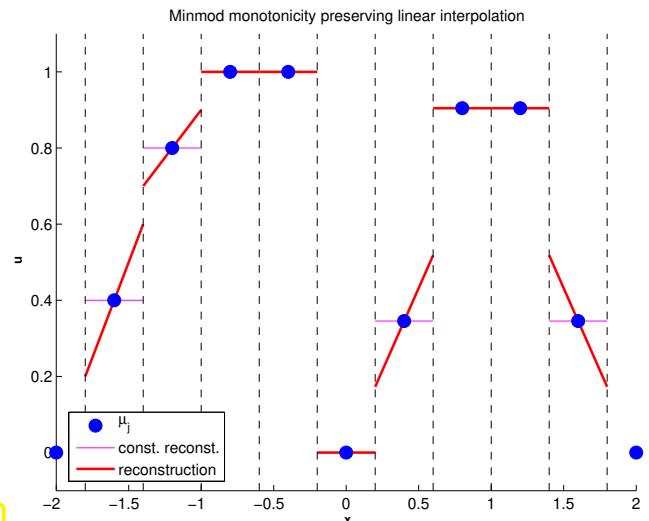


Fig. 578

Lemma 11.5.2.10. Monotonicity preservation of minmod reconstruction

Minmod reconstruction (\rightarrow Def. 11.5.2.9) is monotonicity preserving (\rightarrow Def. 11.5.2.1)

Proof. w.l.o.g. assume $\mu_{j+1} \geq \mu_j \Rightarrow \sigma_j \geq 0 \wedge \sigma_{j+1} \geq 0$

$$\Rightarrow \mu_j + \frac{1}{2}h\sigma_j \leq \frac{1}{2}(\mu_j + \mu_{j+1}) \leq \mu_{j+1} - \frac{1}{2}h\sigma_{j+1}$$

□

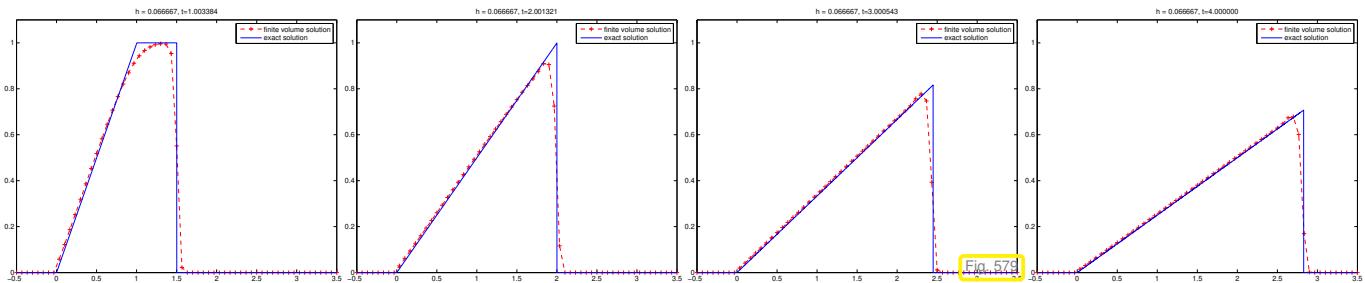
Terminology: effect of **minmod**-function in R_{mm} : slope **limiting**: **minmod** = **slope limiter**

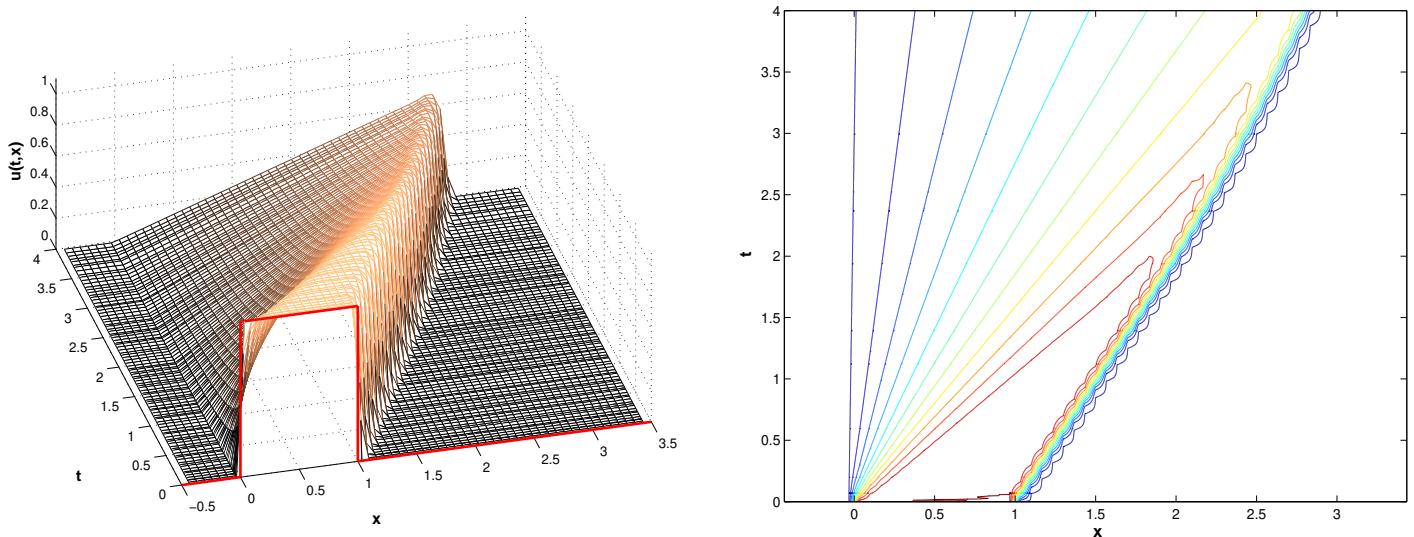
EXPERIMENT 11.5.2.11 (Linear reconstruction with minmod limiter (Burgers' equation)) Same setting as in Exp. 11.5.1.9, Cauchy problem as in Exp. 11.3.4.2:

- ◆ Cauchy problem for Burgers equation (11.1.3.4) (flux function $f(u) = \frac{1}{2}u^2$) from Ex. 11.2.6.4 ("box" initial data)
- ◆ Equidistant spatial mesh with meshwidth $h = \frac{1}{15}$
- ◆ Linear reconstruction with minmod limited slope (\rightarrow Def. 11.5.2.9)

$$\sigma_j := \text{minmod}\left(\frac{\mu_j - \mu_{j-1}}{h}, \frac{\mu_{j+1} - \mu_j}{h}\right).$$

- ◆ Godunov numerical flux (11.3.4.33): $F = F_{\text{GD}}$
- ◆ timestepping based on adaptive 5th-order Runge-Kutta method "ode45" with "overkill tolerances"





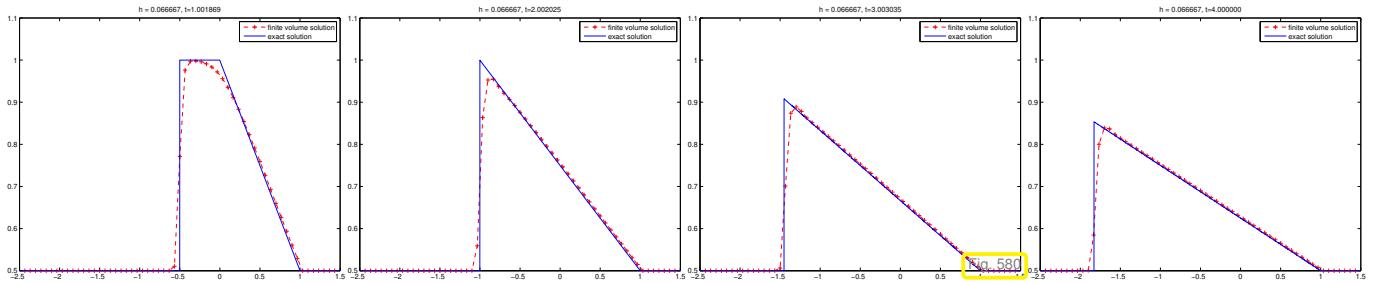
Observation: spurious oscillations successfully suppressed! □

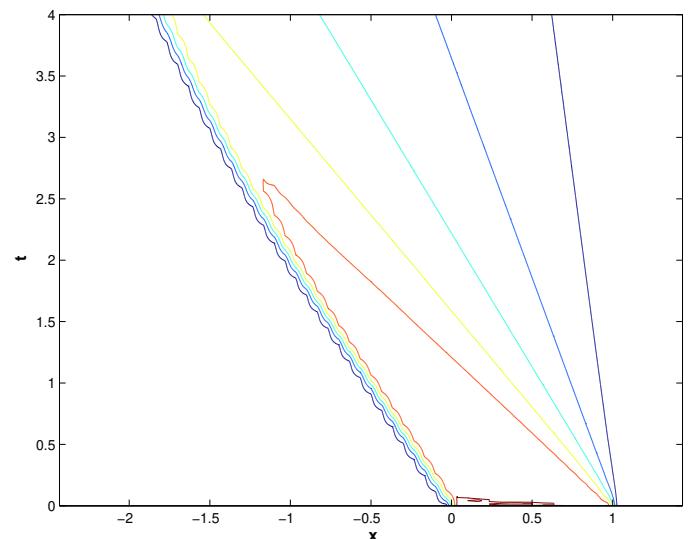
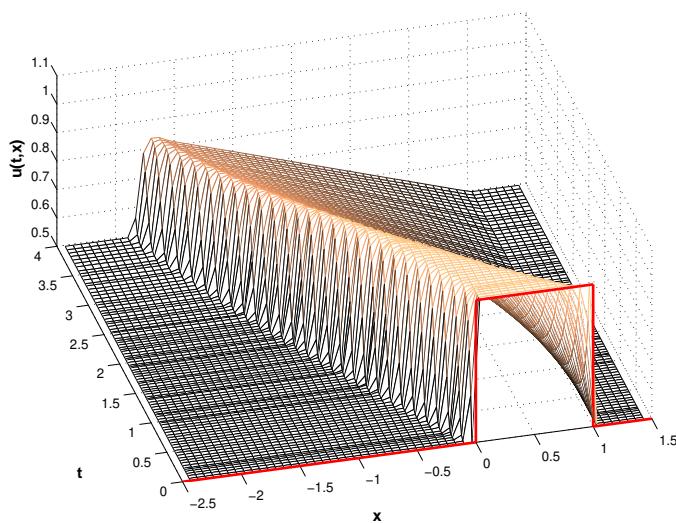
EXPERIMENT 11.5.2.12 (Linear reconstruction with minmod limiter) Same setting as in Exp. 11.5.1.9, Cauchy problem as in Exp. 11.3.4.3:

- ◆ Cauchy problem for Traffic Flow equation (11.1.2.23) (flux function $f(u) = u(1-u)$) from Ex. 11.2.6.5 (“box” intial data)
- ◆ Equidistant spatial mesh with meshwidth $h = \frac{1}{15}$
- ◆ Linear reconstruction with minmod limited slope (\rightarrow Def. 11.5.2.9)

$$\sigma_j := \text{minmod} \left(\frac{\mu_j - \mu_{j-1}}{h}, \frac{\mu_{j+1} - \mu_j}{h} \right).$$

- ◆ Godunov numerical flux (11.3.4.33): $F = F_{\text{GD}}$
- ◆ timestepping based on adaptive Runge-Kutta method `ode45` with absolute tolerance 10^{-7} and relative tolerance 10^{-6} .





Observation: spurious oscillations successfully suppressed!

EXPERIMENT 11.5.2.13 (Improved resolution by limited linear reconstruction)

- ◆ Same setting as in Ex. 11.3.4.22: Cauchy problem for Burgers equation (11.1.3.4) (flux function $f(u) = \frac{1}{2}u^2$) from Ex. 11.2.6.4 (shifted “box” initial data, $u_0(x) = -1$ for $x \notin [0, 1]$, $u_0(x) = 1$ for $x \in [0, 1]$)
- ◆ Equidistant spatial mesh with meshwidth $h = \frac{1}{15}$
- ◆ “High-order” method based on linear reconstruction with minmod limited slope (\rightarrow Def. 11.5.2.9)

$$\sigma_j := \text{minmod}\left(\frac{\mu_j - \mu_{j-1}}{h}, \frac{\mu_{j+1} - \mu_j}{h}\right).$$

- ◆ Godunov numerical flux (11.3.4.33): $F = F_{\text{GD}}$
- ◆ timestepping based on adaptive Runge-Kutta method `ode45` with absolute tolerance 10^{-10} and relative tolerance 10^{-8} .

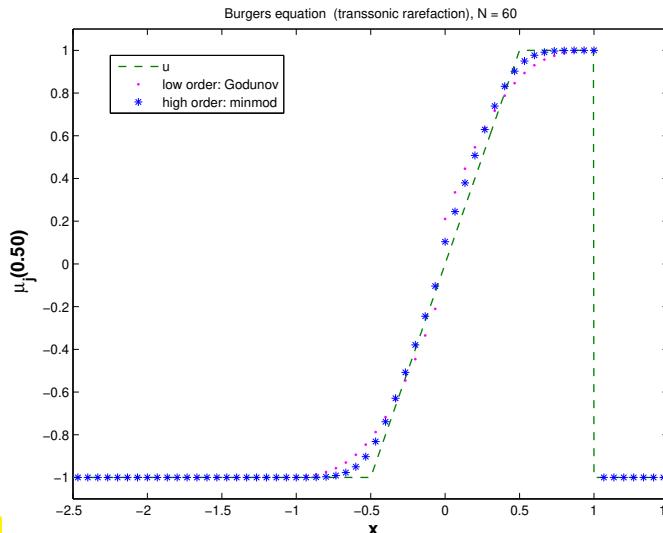


Fig. 581

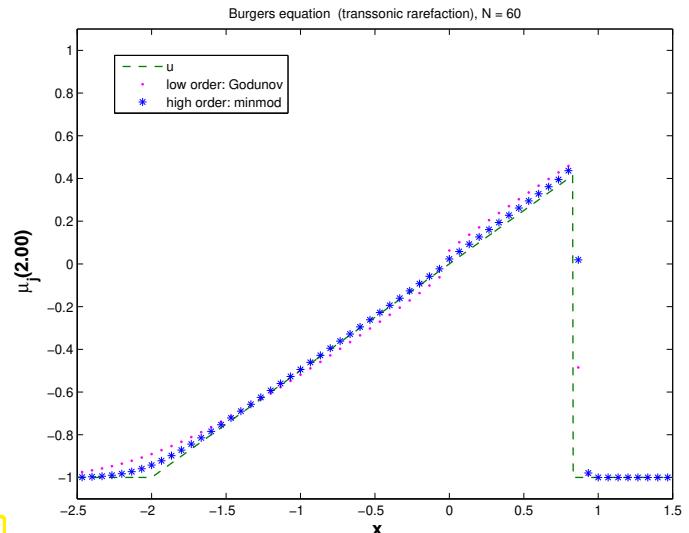


Fig. 582

Observation: *Better resolution* of rarefaction fan compared with the conservative finite volume method based on of Godunov numerical flux without linear reconstruction. Good resolution of shock.

This improved resolution is the main rationale for the use of piecewise linear reconstruction.

11.5.3 MUSCL Scheme

= Monotone Upwind Scheme for Conservation Laws

Also in the case of the method-of-lines approach to Cauchy problems for conservation laws with *smooth solutions* it is commonly observed that

$$\max_{k>0} \|u_h^{(k)} - u(\cdot, t_k)\|_{L^1(\mathbb{R})}, \max_{k>0} \|u_h^{(k)} - u(\cdot, t_k)\|_{L^\infty(\mathbb{R})} = O(h^p + \tau^q) \quad \text{for } h, \tau \rightarrow 0,$$

if we use a scheme of (formal) order $p \in \mathbb{N}$ in space combined with explicit timestepping of order $q \in \mathbb{N}$. We already saw error bounds of this kind for parabolic evolution problems in Section 9.2.8 and for the linear wave equation in Thm. 9.3.5.10.

However, the **CFL-condition** will always force us to choose $\tau = O(h)$, see Section 11.4.2. Therefore, in light of the considerations in Rem. 9.2.8.10, there is nothing gained by choosing $q \geq p$, which suggests that we opt for $q = p$. Thus, in the case of a minmod-limited conservative finite volume scheme with piecewise linear reconstruction, the use of a second-order explicit Ruge-Kutta single-step method is advisable. This will yield the popular MUSCL scheme.

Case of equidistant spatial mesh with meshwidth $h > 0$:

- ◆ Conservative finite volume spatial discretization (11.5.1.1) with monotone consistent 2-point flux, e.g., Godunov numerical flux (11.3.4.33)
- ◆ Piecewise linear reconstruction (\rightarrow Def. 11.5.1.3) with **minmod** slope limiting (\rightarrow Def. 11.5.2.9):

$$v_j^\pm := \mu_j \pm \tfrac{1}{2} \text{minmod}(\mu_{j+1} - \mu_j, \mu_j - \mu_{j-1}). \quad (11.5.3.1)$$

- ◆ 2nd-order Runge-Kutta timestepping for (11.5.1.1): **method of Heun**, cf. (11.4.1.11):

If the right hand side of (11.5.1.1) is abbreviated by

$$\mathcal{L}_h(\vec{\mu}) := -\frac{1}{h} (F(v_j^+(t), v_{j+1}^-(t)) - F(v_{j-1}^+(t), v_j^-(t))),$$

then the fully discrete scheme (uniform timestep $\tau > 0$) reads (11.5.1.1)

$$\begin{aligned} \vec{\kappa} &:= \vec{\mu}^{(k)} + \tau \mathcal{L}_h(\vec{\mu}^{(k)}) , \\ \vec{\mu}^{(k+1)} &:= \vec{\mu}^{(k)} + \tfrac{1}{2} \tau (\mathcal{L}_h(\vec{\mu}^{(k)}) + \mathcal{L}_h(\vec{\kappa})) . \end{aligned} \quad (11.5.3.2)$$

EXAMPLE 11.5.3.3 (Adequacy of 2nd-order timestepping)

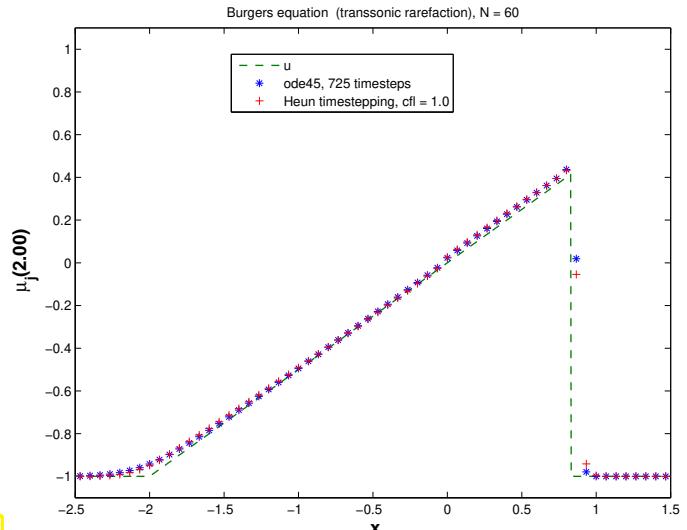
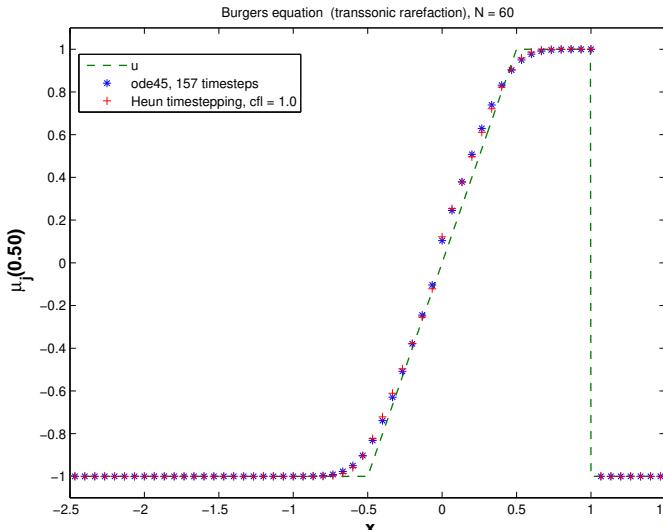
- ◆ Same setting as in Ex. 11.3.4.22: Cauchy problem for Burgers equation (11.1.3.4) (flux function $f(u) = \frac{1}{2}u^2$) from Ex. 11.2.6.4 (shifted “box” intial data, $u_0(x) = -1$ for $x \notin [0, 1]$, $u_0(x) = 1$ for $x \in [0, 1]$)
- ◆ Equidistant spatial mesh with meshwidth $h = \frac{1}{15}$
- ◆ Linear reconstruction with minmod limited slope (\rightarrow Def. 11.5.2.9)

$$\sigma_j := \text{minmod}\left(\frac{\mu_j - \mu_{j-1}}{h}, \frac{\mu_{j+1} - \mu_j}{h}\right).$$

- ◆ Godunov numerical flux (11.3.4.33): $F = F_{GD}$

- ◆ Two options for timestepping:

1. timestepping based on 5th-order adaptive Runge-Kutta method with tight tolerances (“exact timestepping”),
2. Heun timestepping (11.5.3.2) with uniform timestep $\tau = h$



Observation: 2nd-order Runge-Kutta method (11.5.3.2) provides same accuracy as “overkill integration” by means of `ode45` with tight tolerances.

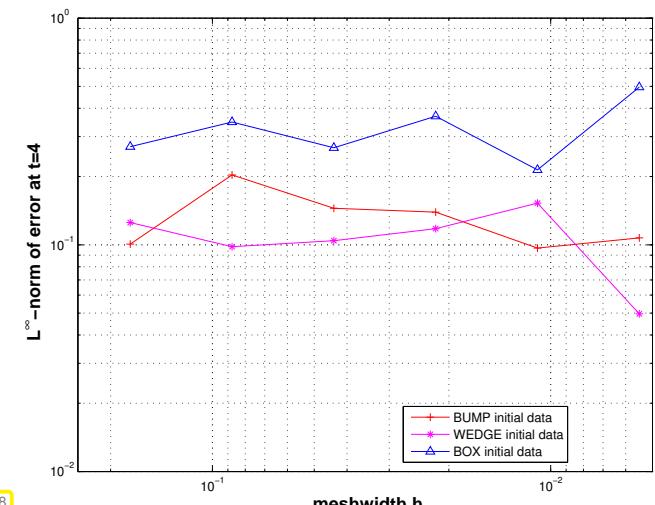
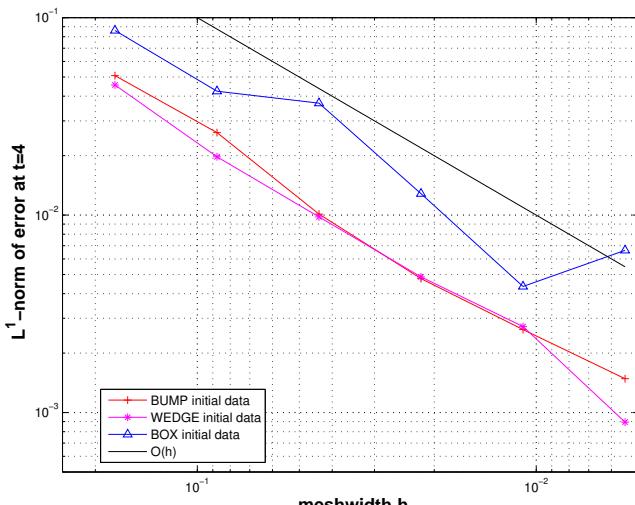
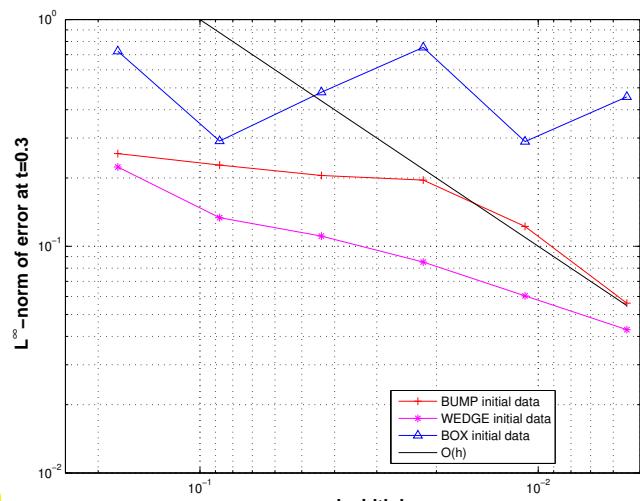
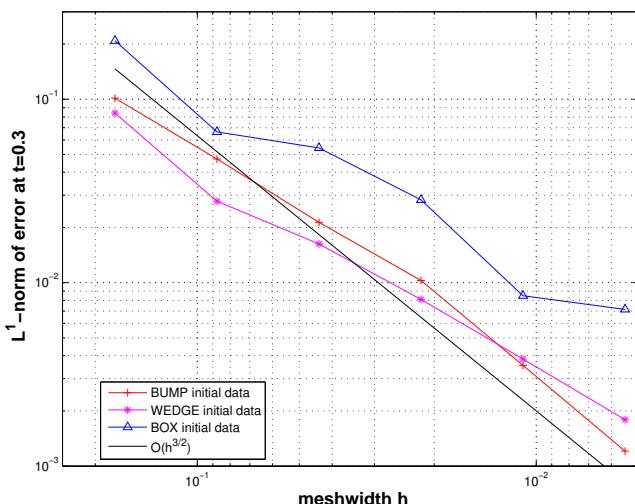
- For the sake of efficiency balance order of spatial and temporal discretizations and use Heun timestepping.

EXPERIMENT 11.5.3.4 (Convergence of MUSCL scheme) We repeat the numerical experiments of Exp. 11.4.4.1 for

- ◆ a conservative finite volume discretization with Godunov numerical flux and `minmod`-limited linear reconstruction, see Exp. 11.5.2.11 (`ode45` timestepping),
- ◆ the MUSCL scheme as introduced above with fixed timestep $\tau = 0.5h$.

We monitor the “discrete” error norms (11.4.4.2) and (11.4.4.3)

$$\begin{aligned} \text{err}_1(h) &:= \max_{k>0} h \sum_j |\mu_j^{(k)} - u(x_j, t_k)| \approx \max_{k>0} \|u_h^{(k)} - u(\cdot, t_k)\|_{L^1(\mathbb{R})}, \\ \text{err}_\infty(h) &:= \max_{k>0} \max_{j \in \mathbb{Z}} |\mu_j^{(k)} - u(x_j, t_k)| \approx \max_{k>0} \|u_h^{(k)} - u(\cdot, t_k)\|_{L^\infty(\mathbb{R})}. \end{aligned}$$



Observation: 2nd-order Heun method produces solutions whose convergence and accuracy matches those of solutions obtained by highly accurate high-order Runge-Kutta timestepping.

>

Review question(s) 11.5.3.5 (Higher order conservative finite volume methods)

We recall two key concepts for the design of higher-order finite volume methods:

Definition 11.5.1.3. Linear reconstruction

Given an (infinite) mesh $\mathcal{M} := \{[x_{j-1}, x_j]\}_{j \in \mathbb{Z}}$ ($x_{j-1} < x_j$), a **linear reconstruction operator** $R_{\mathcal{M}}$ is a mapping

$$R_{\mathcal{M}} : \mathbb{R}^{\mathbb{Z}} \mapsto \{v \in L^{\infty}(\mathbb{R}) : v \text{ linear on } [x_{j-1/2}, x_{j+1/2}] \forall j \in \mathbb{Z}\},$$

taking a sequence $\vec{\mu} \in \mathbb{R}^{\mathbb{Z}}$ of cell averages to a possibly *discontinuous* function $R_{\mathcal{M}}\vec{\mu}$ that is piecewise linear on dual cells.

Definition 11.5.2.1. Monotonicity-preserving linear reconstruction (MPLR)

An linear reconstruction operator R_M (\rightarrow Def. 11.5.1.3) is **monotonicity preserving**, if

$$(R_M \vec{\mu})(x_j) = \mu_j \quad \wedge \quad \begin{aligned} \mu_j \leq \mu_{j+1} &\Rightarrow R_M \vec{\mu} \text{ non-decreasing in }]x_j, x_{j+1}[, \\ \mu_j \geq \mu_{j+1} &\Rightarrow R_M \vec{\mu} \text{ non-increasing in }]x_j, x_{j+1}[. \end{aligned}$$

(Q11.5.3.5.A) Argue why a linear monotonicity preserving piecewise linear reconstruction must impose vanishing slopes throughout.

(Q11.5.3.5.B) Explain the meanings of “linear” in the phrase “non-*linear linear* reconstruction” (in the context of high-order finite volume methods for 1D conservation laws).

(Q11.5.3.5.C) Fully discrete evolutions for scalar conservation laws on uniform space-time meshes can be described by means of stencils.

What will be the width of the stencil for the fully discrete evolution operator, when using two-point numerical fluxes, piecewise linear reconstruction based on min-mod limiting, and a (generic) s -stage explicit RK-SSM for timestepping for the discretization of a scalar conservation law in 1D.

△

11.6 Outlook: Systems of Conservation Laws

Learning outcomes

In this chapter about the numerical treatment of conservation laws you should have learned

- the general form of a scalar conservation law in one spatial dimension, and the balance law expressed by it.
- the notions of weak solutions, shock solutions, entropy conditions and entropy solutions.
- the general policy of constructing a conservative finite volume spatial semi-discretization.
- important consistent numerical flux functions, in particular the Godunov flux.
- the structure preservation inherent in conservative finite volume methods based on monotone numerical fluxes.
- the concept and significance of the CFL-condition for fully discrete conservative finite volume schemes.
- the construction of “high-order” spatial discretizations based on slope limited piecewise linear reconstruction.

Bibliography

- [BD11] Nicola Bellomo and Christian Dogbe. “On the Modeling of Traffic and Crowds: A Survey of Models, Speculations, and Perspectives”. English. In: *SIAM REVIEW* 53.3 (2011), 409–463. DOI: [10.1137/090746677](https://doi.org/10.1137/090746677) (cit. on pp. 678, 681).
- [Can+15] Suncica Canic, Benedetto Piccoli, Jing-Mei Qiu, and Tan Ren. “Runge-Kutta discontinuous Galerkin method for traffic flow model on networks”. In: *J. Sci. Comput.* 63.1 (2015), pp. 233–255. DOI: [10.1007/s10915-014-9896-z](https://doi.org/10.1007/s10915-014-9896-z) (cit. on p. 678).
- [Chi05] S. Childress. *Notes on traffic flow*. Online notes, <http://chaton.inf.ethz.ch/cselabJoomla/images/NPDE2005.pdf> (2005) (cit. on p. 673).
- [Chr07] D. Christodoulou. “The Euler equations of compressible fluid flow”. In: *Bull. American Math. Soc.* 44.4 (2007), pp. 581–602 (cit. on p. 685).
- [Daf00] C.M. Dafermos. *Hyperbolic conservation laws in continuum physics*. Vol. 325. Grundlehren der mathematischen Wissenschaften. Berlin: Springer, 2000 (cit. on pp. 673, 711).
- [Eva98] L.C. Evans. *Partial differential equations*. Vol. 19. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 1998 (cit. on pp. 702, 707, 708).
- [Hip19] R. Hiptmair. *Numerical Methods for Computational Science and Engineering*. 2019. URL: https://www.sam.math.ethz.ch/~grsam/NCSE20/NumCSE_Lecture_Document.pdf (cit. on pp. 675, 712, 724, 756–758, 768, 770, 776, 777).
- [Krö97] D. Kröner. *Numerical Schemes for Conservation Laws*. Chichester: Wiley-Teubner, 1997 (cit. on p. 673).
- [LeV02] R. J. LeVeque. *Finite volume methods for hyperbolic problems*. Cambridge Texts in Applied Mathematics. Cambridge: Cambridge University Press, 2002, pp. xx+558. DOI: [10.1017/CBO9780511791253](https://doi.org/10.1017/CBO9780511791253) (cit. on p. 673).
- [Osh84] S. Osher. “Riemann Solvers, The Entropy Condition, and Difference Approximations”. In: *SIAM J. Numer. Anal.* 21.2 (1984), pp. 217–235 (cit. on p. 707).
- [Str09] M. Struwe. *Analysis für Informatiker*. Lecture notes, ETH Zürich. 2009 (cit. on pp. 689, 690).

Chapter 12

Finite Elements for the Stokes Equation

Main Index: Terms and Keywords

- H^1 -conforming, 184
- L^2 -norm, 32
- (Bi-)linear forms, 23, 60
- (Generalized) Linear variational problem (LVP), 88
- (Local) quadrature rule, 238
- Sparse matrix**, 157
- 1-norm
 - on \mathbb{R}^n , 24
- 2-norm
 - on \mathbb{R}^n , 24
- 2-regularity
 - of Dirichlet problem, 355
- Landau-Livshits equations, 38
- a priori estimates, 307
- a-orthogonal, 287
- A-stability of a Runge-Kutta single step method, 500
- A-stable single step method, 500
- absolute tolerance, 460
- acoustic waves, 43
- advection equation, 643
- Affine (linear) transformation, 239
- Affine (sub)space, 22
- affine equivalent, 255, 258
- affine linear function
 - in 2D, 151
- affine mapping, 239
- Affine space, 62
- algebraic convergence, 294, 442
- algorithm
 - numerical, 126
- analytic solution, 126
- angle condition
 - for Delaunay mesh, 393
- anisotropic diffusion, 617
- artificial diffusion, 615
- artificial viscosity, 615
- Ass: 2-regularity of homogeneous Dirichlet problem, 355
- Ass: Continuity of output functional → Def. 1.2.3.42, 344
- Ass: Continuity of r.h.s. functional, 285
- Ass: Extra smoothness requirements, 99
- Ass: Global Lipschitz continuity of vectorfield, 531
- Ass: Global solutions, 427
- Ass: Linearity of output functional, 344
- Ass: Monotonicity of f' , 658
- Ass: No in/outflow, 590
- Ass: Non-degeneracy of Φ_K , 264
- Ass: Non-warped shape of 2D membrane, 50
- Ass: S.p.d. bilinear form, 284
- Ass: Smoothness of right-hand side vectorfield, 447, 534
- Ass: Smoothness required for two-point boundary value problems, 93
- Ass: Smoothness requirement for stiffness coefficient, 144
- Ass: Warp-free/loop-free shape of string, 49
- assembly, 158
 - cell oriented, 224
 - in FEM, 216
 - linear finite elements, 163
- autonomization, 423
- Autonomous ODE, 419
- autonomous ODE, 416
- balance law, 651
- balanced finite volume discretization, 689
- Banach space, 75
- barycenter, 393
- barycentric coordinate functions, 158
- barycentric coordinate representation
 - of local shape functions, 235
- barycentric coordinates, 158
- basis
 - change of, 135
 - of a vector space, 22
- Basis of a finite dimensional vector space, 22, 132
- Belousov-Zhabotinsky reaction, 456

- best approximation error, 288
- beta function, 236
- bilinear form, 23, 60
 - continuous, 600
 - positive definite, 64
 - positive semi-definite, 63
 - symmetric, 61
- bilinear transformation, 261
- Black-Scholes equation, 40
- blow-up, 457
- boundary conditions
 - Dirichlet, 100
 - boundary conditions, 50, 100
 - Dirichlet, 107
 - essential, 119, 248
 - homogeneous, 108
 - natural, 119
 - Neumann, 103, 107
 - radiation, 107
 - boundary fitting, 276
 - parabolic, 276
 - boundary flux
 - computation of, 347
 - boundary layer, 601
 - boundary value, 46
 - boundary value problem
 - elliptic, 107
 - boundary value problem (BVP), 46
 - Bounded linear operator, 115
 - bounded linear operator, 115
 - Burgers equation, 654, 670
 - Butcher scheme, 453, 496
 - Cahn-Hilliard equation, 38
 - cardinal basis, 140, 185, 190
 - Cauchy problem, 643, 655, 665
 - for one-dimensional conservation law, 658
 - for wave equation, 567
 - Cauchy sequence, 75
 - Cauchy-Schwarz inequality, 25
 - cell, 148, 176
 - cell contributions, 217
 - central flux, 691, 692
 - central slope, 737
 - CFL-condition, 583, 719
 - characteristic coordinates, 567
 - characteristic curve, 659
 - Characteristic curve for one-dimensional scalar conservation law, 659
 - characteristic method, 630
 - chemical reaction kinetics, 486
 - circuit simulation
 - transient, 422
 - circumcenter, 392
 - Classical Runge-Kutta method
 - Butcher scheme, 455
 - classical solution, 100, 101, 111
 - co-dimension
 - relative, 176
 - coefficient vector, 134
 - collocation, 493
 - for ODEs, 539
 - with splines, 410
 - collocation conditions, 493, 539
 - collocation method, 407
 - spectral, 408
 - collocation nodes, 407
 - collocation points, 407, 493
 - collocation single step methods, 492
 - compact embedding, 116
 - compatibility condition, 112, 127
 - for FDM, 380
 - compatibility conditions
 - for $H^1(\Omega)$, 84, 139, 184
 - Complete normed vector spaces and Hilbert spaces, 75
 - complete space, 75, 285
 - composite midpoint rule, 144
 - composite trapezoidal rule, 144
 - Compressed Column Storage (CCS), 216
 - Compressed Row Storage (CRS), 216
 - computational domain, 104
 - computational effort, 325
 - condition number
 - of a matrix, 401
 - conditionally stable, 560
 - Configuration space, 49
 - configuration space, 49
 - conforming
 - finite element space, 184
 - Congruent matrices, 136
 - congruent matrices, 136
 - conservation
 - of energy, 574
 - conservation form, 686
 - conservation law, 656
 - differential form, 657
 - integral form, 657
 - one-dimensional, 657
 - scalar, 657
 - conservation of energy, 105
 - consistency
 - of a variational problem, 617

- consistency error, 533, 732
 Consistent modifications of variational problems, 617
 Consistent numerical flux function, 689
 Consistent single step methods, 438
 continuity
 of a bilinear form, 129
 of a linear form, 82, 129
 of a linear functional, 67
 of linear functional, 344
 Continuity of a linear and bilinear form, 67
 Continuous linear operator, 319
 continuous problem, 131
 control volume, 390, 656
 convection-diffusion equation, 593
 convective cooling, 108
 convective term, 593, 599
 convective terms, 593
 convergence, 292
 algebraic, 294, 442
 asymptotic, 292
 exponential, 294, 442
 convex combination, 363
 convex domain, 114
 convex function, 65
 Convexity of a real-valued function, 65
 COO format
 of a sparse matrix, 215
 corner singular function, 332
 Corollary: H^1 -norm of piecewise smooth functions, 85
 Corollary: Admissible loading/source functions linear 2nd-order elliptic problems, 82
 Corollary: Consistent Runge-Kutta single step methods, 454
 Corollary: Derivative of piecewise smooth functions, 32
 Corollary: Domain of dependence for scalar conservation law, 679
 Corollary: Error estimate for piecewise linear interpolation in 2D, 318
 Corollary: Necessary condition for existence of minimizer, 64
 Corollary: Point evaluation on $H^1(\Omega)$, 79
 Corollary: Polynomial stability function of explicit RK-SSM, 478
 Corollary: Rational stability function of general RK-SSM, 500
 Corollary: Simple criterion for monotone flux function, 710
 Corollary: Stages limit order of explicit RK-SSM, 479
 Courant-Friedrichs-Levy condition (CFL), 583
 Courant-Friedrichs-Lowy (CFL-)condition, 719
 Crank-Nicolson method, 544
 Crank-Nicolson timestepping, 578
 Cubic spline, 410
 cubic spline, 410
 curl, 30
 curse of dimension, 328
 cut-off function, 350
 d'Alembert solution, 567
 d.o.f. handler, 218
 d.o.f. mapper, 218
 debugging
 of FE codes, 369
 degree of freedom (DOF), 134
 degrees of freedom, 126, 179
 Delaunay mesh
 angle condition, 393
 Delaunay triangulation, 393
 delta distribution, 68
 Dense subset, 77
 dense subspace, 77
 diagonal dominance, 363
 diagonalization, 581
 for solving linear ODEs, 480
 diagonally implicit Runge-Kutta method, 497
 dielectric tensor, 58
 difference quotient, 681
 backward, 433, 681
 forward, 433, 681
 one-sided, 377
 symmetric, 377, 435, 681
 difference scheme, 433
 differential operator, 46
 diffusion tensor, 617
 diffusive flux, 656
 diffusive term, 593, 599
 diffusive terms, 593
 dimension
 of a vector space, 22
 Dirac delta function, 68
 Dirichlet boundary conditions, 95, 100, 107
 for linear FE, 250
 Dirichlet data, 90
 admissible, 119
 Dirichlet problem, 100
 variational formulation, 90
 DIRK-SSM, 497
 discrete evolution, 437
 consistency error, 533

- Discrete evolution operator, 533
- Discrete evolution operator → Section 6.3.1, § 6.3.1.1, 533
- discrete maximum principle, 360
- Discrete model, 126
- discrete model, 126
- discrete problem, 131
- discrete variational problem, 130, 134
- discretization, 126, 127
- discretization error, 288, 441
- discretization parameter, 293
- displacement, 49
- dissipation, 526
- DistMesh, 202
- divergence, 30
 - of a vectorfield, 95
- dof mapper, 164
- domain, 46
 - computational, 104
 - spatial, 50
- domain of dependence, 568
- domain of influence, 568
- dot product, 23
- doxygen, 194
- dual mesh, 391
- dual problem, 345
- dual variational problem, 345
- duality estimate, 345
- eddy current model, 37
- eddy currents, 37
- edge, 175
- Eigen, 215
 - sparse matrices, 216
- elastic energy, 52
- electric field, 57
- electric scalar potential, 57
- electromagnetic field energy, 57
- electrostatic field energy, 57
- electrostatics, 56
- element, 176
- Element (stiffness) matrix and element (load) vector, 217
- element load vector, 217
- element stiffness matrix, 217
- elliptic
 - boundary value problem, 46
 - linear scalar second order PDE, 106
- elliptic boundary value problem, 107
- embedded Runge-Kutta methods, 466
- energy
 - conservation, 105
- of electrostatic field, 57
- energy conservation
 - for wave equation, 570
- Energy norm, 66
- energy norm, 66, 286
- energy space, 73
- entity, 175
 - co-dimension, 175
- equidistant mesh, 139, 180
- equilibrium condition, 52
- equipotential surface, 57
- equivalence
 - of norms, 317
- essential boundary conditions, 119, 248
- essential constraints, 251
- Euclidean norm, 24
- Euler equations, 654
- Euler method, 543
 - explicit, 432
 - implicit, 434
 - implicit, stability function, 499
 - semi implicit, 505
- Euler polygon, 432
- evolution operator, 428
 - for an ODE, 531
 - fully discrete, 716
 - semi-discrete, 715
- Evolution operator/mapping, 428
- evolution problem, 516
 - semi-discrete, 528
 - spatial variational formulation, 522
- expansion shock, 669
- explicit Euler method, 432, 534, 543
 - Butcher scheme, 454
- explicit midpoint rule
 - Butcher scheme, 455
 - for ODEs, 452
- explicit Runge-Kutta method, 453
- Explicit Runge-Kutta single-step method, 453
- explicit trapezoidal rule
 - Butcher scheme, 454
- exponential convergence, 294
- extended state space
 - of an ODE, 423
- face, 175
- factory object, 202
- field energy
 - electromagnetic, 57
- finite difference method (FDM), 680
- finite difference methods, 375
- Finite element mesh/triangulation, 175

- finite element space
 - H^1 -conforming, 184
 - Lagrangian, 184
- finite elements
 - parametric, 263
- finite volume methods (FVM), 389
- finite-difference operator, 715
- flow field, 590
- flow map, 591, 644
- flux function, 656, 658
- Fourier expansion, 524
- Fourier's law, 105, 395, 519
 - in fluid, 593
- Frobenius norm, 314
- fully discrete evolutions, 716
- functional, 343
 - linear, 344
- functor, 128
- fundamental lemma of calculus of variations, 99
- fundamental theorem of calculus, 94
- Galerkin matrix, 217
- Galerkin orthogonality, 287
- Galerkin solution
 - quasi-optimality, 288
- Gamma function, 236
- Gauss collocation single step method, 495, 541
- Gauss' theorem, 106, 592, 664
- Gauss-Legendre quadrature, 244
- Gauss-Lobatto quadrature, 244
- Gauss-Radau quadrature formulas, 503
- generalized (= weak) gradient, 85
- generalized eigenvalue problem, 581
- generalized eigenvalues, 581
- generic constants, 325
- geometry layer, 204, 213
- Gibbs free energy, 38
- global shape functions, 179
- global solution
 - of an IVP, 426
- Gmsh, 196
 - .geo-file, 197
 - .msh-file, 199
 - geometric modeling, 196
 - geometry file, 197
 - mesh file, 199
 - physical groups, 200, 203
- gmsh
 - mesh generation, 198
- Godunov numerical flux, 705
- graded mesh, 146
- gradient, 30, 53
- in weak sense, 85
- of a function, 53
- transformation, 266, 279
- Gramian determinant, 241
- Green's first formula, 98
- grid, 381, 680
 - 1D, 139
- grid point
 - of a 1D mesh, 376
- h-refinement, 289, 328
- hanging node, 148, 177
- hat function, 85, 154
- heartbeat model, 420
- heat capacity, 518
- heat conductivity, 105
- heat equation, 518
- heat flux, 104, 105
 - computation of, 347
 - convective, 593, 643
 - diffusive, 593, 643
- heat source, 105
- heat transport, 643
- Helmholtz equation, 43
- Hessian, 36, 313
- Heun method, 717
- Higher order Sobolev spaces/norms, 317
- Higher-order Sobolev semi-norms, 318
- Hilbert space, 75, 285
- homogeneous boundary conditions, 108
- homogeneous Dirichlet problem
 - linear finite element space, 154
- hyperbolic evolution problem, 565
 - discrete case, 572
- impedance boundary conditions, 108
- implicit Euler method, 434, 534, 543
- implicit function theorem, 435
- implicit midpoint method, 435, 534, 576
- implicit midpoint rule, 544
- incidence relations, 206
- incompressible, 594
- Incompressible flow field, 594
- increment
 - for collocation SSM, 540
- increment equations
 - linearized, 506
- increments
 - Runge-Kutta, 453, 496
- index mapping matrix, 232
- indexing
 - of entities, 205

inexact splitting methods, 510
 inflow, 656
 inflow boundary, 603, 624, 647
 initial conditions, 516
 initial value problem
 stiff, 488, 547
 initial value problem (IVP), 423
 initial-boundary value problems (IBVP), 517
 parabolic, 519
 initial-value problem, 530
 Inner product, 25
 inner product, 25
 integrated Legendre polynomials, 400
 integration by parts
 1D, 94
 multidimensional, 98
 intermediate state, 702
 interpolant
 piecewise linear, 395
 interpolation error, 308, 312
 interpolation error estimates
 anisotropic, 319
 in 1D, 307
 interpolation nodes, 185, 190
 inviscid, 653
 IVP, 423
 IVP $\hat{=}$ initial value problem, 530

 Jacobian, 29, 36

 kinetic energy, 570
 kinetics
 of chemical reaction, 486
 Kronecker product, 545

 L(π)-stability, 555
 L(π)-stability, 555
 L-shaped domain, 302
 L-stability, 554
 L-stable, 502
 L-stable Runge-Kutta method, 502
 Lagrangian finite elements, 184
 on quadrilaterals, 260
 Lagrangian method, 630
 for advection, 626
 Lagrangian view, 624
 lambda function, 128
 Laplace equation, 100
 Laplace operator, 100
 Laplacian, 30
 Lax entropy condition, 675
 Lax-Friedrichs flux, 696

layer
 boundary, 601
 layers
 internal, 616
 of mesh functionality, 204
 leapfrog, 578
 Legendre polynomials, 400
 integrated, 400
 LehrFEM++, 194
 gmsh reader, 202
 mesh factory, 202
 Lemma:
 fundamental lemma of the calculus of variations, 94
 Lemma: Affine equivalence of Lagrangian finite elements on simplicial meshes, 257
 Lemma: Affine transformation of triangles, 239
 Lemma: Auxiliary estimate on sector, 314
 Lemma: Behavior of generalized eigenvalues, 551
 Lemma: Boundedness condition on linear form, 67
 Lemma: Classical solutions and characteristic curves, 660
 Lemma: Comparison principle for monotone semi-discrete conservative evolutions, 711
 Lemma: Congruent Galerkin matrices, 136
 Lemma: Criterion for local Lipschitz continuity, 425
 Lemma: Decay of solutions of parabolic evolutions, 525
 Lemma: Dimension of spaces of polynomials, 178
 Lemma: Dimension of spaces of tensor product polynomials, 179
 Lemma: Effect of change of basis on Galerkin matrix, 135
 Lemma: Energy norm and energy deviation, 286
 Lemma: Existence of solutions of collocation equations, 539
 Lemma: Fundamental lemma of calculus of variations in higher dimensions, 99
 Lemma: Fundamental lemma of calculus of variations on boundaries, 102
 Lemma: General product rule, 95
 Lemma: Integration of powers of barycentric coordinate functions, 236
 Lemma: Invertibility of “averaging matrices”, 364
 Lemma: Linear monotonicity preserving reconstruction trivial, 743

- Lemma: Local interpolation error estimates for
 2D linear interpolation, 315
 Lemma: Monotonicity of Lax-Friedrichs/Rusanov
 numerical flux and Godunov flux, 710
 Lemma: Monotonicity preservation of minmod
 reconstruction, 744
 Lemma: Non-oscillatory monotone semi-discrete
 evolutions, 712
 Lemma: Preservation of polynomials under
 affine pullback, 257
 Lemma: Rarefaction solution of Riemann
 problem, 672
 Lemma: Shock solution of Riemann problem,
 668
 Lemma: Smoothness of solutions of ODEs, 417
 Lemma: Space of solutions of linear ODEs, 418
 Lemma: Sparsity of Galerkin matrix, 156
 Lemma: Stability function as approximation of
 \exp for small arguments, 479
 Lemma: Testing with basis vectors, 134
 Lemma: Transformation formula for gradients,
 266
 lexicographic ordering, 381, 382
 Lie-Trotter splitting, 508
 limit cycle, 487
 linear boundary fitting, 339
 linear evolution, 523
 Linear first-order ODE, 417
 linear form, 23, 60
 continuity, 82
 Linear forms, 23, 60
 linear function
 in 2D, 151
 linear functional, 23, 60
 linear interpolation
 in 1D, 307
 in 2D, 311, 312
 Linear interpolation in 2D, 312
 linear ODE, 417, 532
 linear operator
 bounded, 115
 Linear reconstruction, 735
 linear regression, 297
 linear variational problem, 88
 Linearity, 108
 Lipschitz continuity, 531
 Lipschitz continuous function, 424, 425
 load vector, 134, 217
 local operations, 224
 local quadrature rule
 transformation, 240
 local quasi-uniformity, 326
 local shape function, 181
 barycentric representation, 259
 local shape functions
 quadratic, 186
 Local shape functions (LSF), 181
 local→global index mapping, 218
 logistic differential equation, 418
 Lotka-Volterra ODE, 419
 M-matrix, 364, 365
 macroscopic quantities, 649
 magnetization, 38
 manufactured solution, 299
 manufactured solutions, 370
 mass lumping, 547, 579
 Material derivative, 635
 material derivative, 635
 material tensor, 90
 matrix
 condition number, 401
 matrix exponential, 480
 maximum norm
 on \mathbb{R}^n , 24
 maximum principle, 358, 597
 discrete, 360
 Maxwell's equations
 static case, 57
 Mean square norm/ L^2 -norm, 32
 mean value formula, 313
 mean-value theorem, 114
 mesh, 175
 1D, 139
 cell, 148
 equidistant, 139
 in time, 437
 node, 148
 non-conforming, 177
 quadrilateral, 176
 simplicial, 177
 temporal, 431
 topology, 206
 triangular, 176
 mesh data structure, 203
 mesh file format, 195
 mesh functionality
 layers, 204
 mesh generator, 195
 Mesh width, 293
 mesh width, 139, 293
 method of characteristics, 602, 624
 method of lines, 528

- micromagneticcs, 38
- micromagnetics, 38
- midpoint method
 - implicit, stability function, 499
- midpoint rule, 452
 - composite, 144
- minmod, 744
- Minmod reconstruction, 744
- mixed boundary conditions, 108
- Mixed Neumann–Dirichlet problem, 103
- model
 - continuous, 126
 - discrete, 126
- monomial basis, 400
- Monotone numerical flux function, 710
- monotonicity preserving linear interpolation, 742
- Monotonicity-preserving linear reconstruction (MPLR), 742
- multi-index, 317
- multi-index notation, 178
- multiplicative trace inequality, 120
- Multivariate polynomials, 178
- MUSCL scheme, 747
- natural boundary conditions, 119
- Navier-Stokes equations, 37
- nested meshes, 289
- NETGEN, 202
- Neumann boundary conditions, 103, 107
- Neumann data
 - admissibility conditions, 119
- Neumann problem, 111
 - compatibility condition, 112
 - variational form, 111
- Newton's second law of motion, 563
- Newton-Cotes formula, 244
- nodal analysis
 - transient, 422
- nodal basis, 154
- nodal interpolation operator, 271
- nodal interpolation operators, 323
- node, 148
 - 1D, 139, 680
 - hanging, 148
 - of a 1D mesh, 376
 - quadrature, 143
- Norm, 24
- norm, 24
 - on function space, 32
- Numerical domain of dependence, 718
- numerical domain of dependence, 718
- numerical flux, 391, 396
- numerical flux (function), 686
- numerical flux function, 391
- numerical quadrature
 - nodes, 238
 - weights, 238
- ODE, 36, 423
 - autonomous, 416
 - linear, 417, 532
 - scalar, 417, 432
- $\text{ODE} \triangleq$ ordinary differential equation, 530
- ODE, right-hand-side function, 416
- offset function trick, 130
- offset function, 249
 - for linear FE, 249
 - for linear finite elements in 1D, 145
- offset function trick, 62, 130
- one-sided difference quotient, 377
- one-step error, 444
- option pricing, 40
- order
 - of a discrete evolution, 447
 - of an ODE, 424
 - of discrete evolution, 533
- Order of a discrete evolution operator, 447
- Order of a local quadrature rule, 242
- Order of a single step method, 443
- order of quadrature rule, 242
- ordinary differential equation, 530
- ordinary differential equation (ODE), 423
- oregonator, 456
- orientation
 - of edges, 208
- outflow, 656
- outflow boundary, 603
- output functional, 343
- p-refinement, 328
- Parametric finite elements, 264
- parametric finite elements, 263, 264
- Paraview, 203
- partial differential equation, 36
- particle mesh method, 631
- particle method, 630
- particle model
 - of traffic flow, 648
- PDE, 36
 - ILinear scalar second order elliptic, 106
- Peano
 - Theorem of, 425
- perpendicular bisector, 392
- Petrov-Galerkin discretization, 226

- phase space, 656
 - of an ODE, 423
- phenomenological model, 420
- Picard-Lindelöf
 - Theorem of, 425
- Piecewise continuously differentiable functions,
 - 31, 54
- piecewise linear interpolant, 395
- piecewise linear reconstruction, 734
- piecewise quadratic interpolation, 324
- point force, 54
- Poisson equation, 100
- Poisson equation, 297
- Poisson matrix, 382
- polar coordinates, 69
- polygon, 147
- polynomials
 - degree, 178
 - multivariate, 178
- Positive (semi-)definite bilinear form, 25
- positive definite, 25
 - bilinear form, 64
 - uniformly, 58
- positive definite (s.p.d.), 25
- Positive definite bilinear form, 64
- Positive definite matrix, 25
- positive semi-definite, 25
 - bilinear form, 63
- Positive semi-definite bilinear form, 63
- potential energy, 52, 570
- predator-prey model, 419
- predicate, 231, 251
- primal mesh, 391
- principal, 417
- problem size, 325
- procedural form
 - of data functions, 128
- product rule, 525
 - in higher dimensions, 95
- production term, 656
- propagated error, 444
- Pullback, 256
- pullback, 256
- Punkt
 - stationär, 420
- Pythagoras' theorem, 287
- Quadratic functional, 61
- quadratic functional, 61
- quadratic local shape functions, 186
- Quadratic minimization problem, 62
- quadratic minimization problem, 62, 75
- quadratic minimization problems, 60
- quadrature formula
 - 1D, 143
 - general, 238
- quadrature nodes, 143, 238
- quadrature rule, 238
 - on triangle, 243
 - order, 242
- quadrature rules
 - Gauss-Legendre, 244
 - Gauss-Lobatto, 244
- quadrature weights, 143, 238
- quadrilateral mesh, 176
- quasi-optimality, 288
- quasi-uniformity, 369
- Radau RK-method
 - order 3, 503
 - order 5, 503
- Radau timestepping, 555
- radiation boundary conditions, 107
- rarefaction
 - subsonic, 704
 - supersonic, 704
 - transonic, 704
- rarefaction wave/fan, 672
- rate
 - of algebraic convergence, 442
- Rate of convergence, 294
- reaction coefficient, 127
- reaction term
 - in 2nd-order BVP, 147
- recirculating flow, 604
- reference element, 238, 239
- reference elements, 264
- reference triangle, 239
- refinement layer, 204
- Region of (absolute) stability, 485
- region of absolute stability, 485
- regular refinement, 289
- relative tolerance, 460
- Rellich theorem, 116
- reversibility, 574
- Riccati differential equation, 431
- Riccati differential equation, 432
- Riemann problem, 667
 - local, 702
- Riesz representation theorem, 285
- right hand side
 - of an ODE, 423
- right hand side vector, 217
- Robin boundary conditions, 108

robust discretization, 606
 Rodriguez formula, 400
 ROW methods, 507
 Runge-Kutta
 increments, 453, 496
 Runge-Kutta method, 453, 496
 L-stable, 502
 Runge-Kutta methods
 embedded, 466
 semi-implicit, 504
 stability function, 478, 499, 726
 Runge-Kutta single-step method, 496
 scalar ODE, 432
 scalar product, 25
 scaling, 53, 598
 Schrödinger equation
 electronic, 41
 SDIRK timestepping, 555
 semi-discrete evolution problem, 528
 semi-implicit Euler method, 505
 semi-norm, 80
 separation of variables, 524
 shape functions
 global, 179
 shape regularity
 uniform, 369
 Shape regularity measure, 316
 shape regularity measure, 316, 329
 Shock, 668
 shock, 668
 physical, 675
 subsonic, 704
 supersonic, 704
 shock speed, 668
 similarity solution, 671
 Simplicial Lagrangian finite element spaces, 184
 simplicial mesh, 177
 single step method
 A-stability, 500
 Single-step method, 437
 single-step method, 437
 singular perturbation, 602
 Singularly perturbed boundary value problem,
 602
 SIR model, 421
 slope limiter, 744
 slope limiting, 742
 slopes, 735
 Sobolev norms, 317
 Sobolev semi-norms, 318
 Sobolev space $H_0^1(\Omega)$, 78
 Sobolev space $H^1(\Omega)$, 79
 Sobolev space $H_0^1(\Omega)$, 78
 Sobolev spaces, 71, 317
 solution
 approximate, 126
 Solution of an ordinary differential equation, 417
 source term, 46
 Space $L^2(\Omega)$, 74
 space-time-cylinder, 516
 span, 205
 sparse matrix
 initialization, 167
 sparsity pattern, 157
 spatial domain, 50
 spectral collocation method, 408
 spectral Galerkin methods, 289
 spectrum, 723
 spline
 cubic, 410
 spline collocation, 410
 split-step timestepping, 626
 splitting
 Lie-Trotter, 508
 Strang, 508
 splitting methods, 508
 inexact, 510
 Störmer scheme, 578
 stability, 90
 region of, 485
 unconditional, 553
 stability domain, 727
 stability function
 of explicit Runge-Kutta methods, 478, 726
 of RK-SSM, 554
 of Runge-Kutta methods, 499
 stages, 497
 star-shaped, 120
 state space, 656
 of an ODE, 423
 stencil, 383, 717
 Stiff IVP, 488, 547
 stiffness, 52
 stiffness matrix, 134, 217
 sparsity, 180
 Strang splitting, 508, 626, 627
 Strang's lemma, 337
 streamline, 590
 closed, 603
 streamline backtracking, 637
 streamline diffusion, 614
 strong form, 46, 101

- structure-preserving timestepping, 574
 sub-entities, 176
 subsonic rarefaction, 704
 subsonic shock, 704
 supersonic rarefaction, 704
 supersonic shock, 704
 support
 of a function, 140
 Support of a function, 140
 Supremum norm, 32
 supremum norm, 32
 surface, 34
 symbol
 of a difference operator, 723
 Symmetric bilinear form, 24
 symmetric bilinear form, 24, 61
 symmetric difference quotient, 377
 Symmetric positive definite matrices, 26
- T-matrix, 232
 Töplitz matrix, 143
 tangent field, 431
 Tensor product Lagrangian finite element spaces, 190
 Tensor product polynomials, 179
 tensor product polynomials, 179
 tent function, 140, 152, 154
 test space, 88
 TETGEN, 202
 Theorem: $L^2(\Omega)$ by completion, 77
 Theorem: L^1 -contractivity of evolution for scalar conservation law, 678
 Theorem: $H^1(\Omega)$ -Norm interpolation error estimates for Lagrangian finite elements, 342
 Theorem: $L^2(\Omega)$ -Norm interpolation error estimates for Lagrangian finite elements, 342
 Theorem: (Absolute) stability of explicit RK-SSM for linear systems of ODEs, 484
 Theorem: Maximum principle for 2nd-order elliptic BVP, 358
 Theorem: Angle condition for Voronoi dual meshes, 393
 Theorem: Assembly through index mapping matrices, 232
 Theorem: Best approximation error estimates for Lagrangian finite elements, 324
 Theorem: Cauchy-Schwarz inequality, 25
 Theorem: Cea's lemma, 288
 Theorem: Classical solutions are weak solutions, 111
- Theorem: Comparison principle for scalar conservation laws, 677
 Theorem: Compatibility conditions for piecewise smooth functions in $H^1(\Omega)$, 84
 Theorem: Completion of a normed vector space, 77
 Theorem: Convergence of single-step methods, 538
 Theorem: Convergence of solutions of fully discrete parabolic evolution problems, 559
 Theorem: Convergence of fully discrete solutions of the wave equation, 585
 Theorem: Corner singular function decomposition, 334
 Theorem: Differentiation formula for determinants, 595
 Theorem: Divergence-free velocity fields for incompressible flows, 596
 Theorem: Domain of dependence for isotropic wave equation, 569
 Theorem: Duality estimate for linear functional output, 345
 Theorem: Elliptic lifting theorem on convex domains, 334
 Theorem: Energy conservation in wave propagation, 570
 Theorem: Equivalence of quadratic minimization problem and linear variational problem, 88
 Theorem: Error estimate for piecewise linear interpolation, 316
 Theorem: Existence and uniqueness of solution of linear variational problem, 285
 Theorem: Existence and uniqueness of solutions of discrete variational problems, 131
 Theorem: Existence and uniqueness of solutions of IVPs, 531
 Theorem: Existence of minimizers in Hilbert spaces, 76
 Theorem: Existence of unique minimizer in finite dimensions, 68
 Theorem: Gauss' theorem, 96
 Theorem: Green's first formula, 98
 Theorem: Implicit function theorem, 435
 Theorem: Independence of Galerkin solution of choice of basis, 134
 Theorem: Inverse positivity, 365
 Theorem: Maximum principle for linear FE solution of Poisson equation, 366
 Theorem: Maximum principle for scalar

- 2nd-order convection diffusion equations, 597
- Theorem: Multiplicative trace inequality, 120
- Theorem: Norms from inner products, 25, 65
- Theorem: Order of collocation single step method, 496, 541
- Theorem: Order of simple splitting methods, 510
- Theorem: Order of Strang splitting single step method, 627
- Theorem: Partial derivatives commute, 29
- Theorem: Poincaré-Friedrichs inequality, 81, 113
- Theorem: Poincaré-Friedrichs-type estimate, 116
- Theorem: Real diagonalization of symmetric matrices, 26
- Theorem: Region of stability of Gauss collocation single step methods, 501
- Theorem: Rellich's Theorem: Compact embedding of $H^1(\Omega)$ in $L^2(\Omega)$, 116
- Theorem: Smooth elliptic lifting theorem, 331
- Theorem: Sobolev embedding theorem, 319
- Theorem: Sobolev spaces by completion, 80
- Theorem: Solution of linear advection problem, 645
- Theorem: Stability function of explicit Runge-Kutta methods, 726
- Theorem: Stability function of general Runge-Kutta methods, 499
- Theorem: Stability function of some explicit Runge-Kutta methods, 478
- Theorem: Strang's second lemma, 337
- Theorem: Taylor's theorem, 28
- Theorem: Theorem of Peano & Picard-Lindelöf [Ama83, Satz II(7.6)], [Str09, Satz 6.5.1], [DR08, Thm. 11.10], [Han02, Thm. 73.1], 425
- Theorem: Transformation formula for integrals, 34
- Theorem: Uniqueness of solutions of quadratic minimization problems, 65
- Theorem: Variation-of-constants formula, 529
- timestep (size), 433
- timestep constraint, 479
 - explicit Euler, 552
- timestepping, 432, 543
- tolerance
 - absolute, 460
 - for adaptive timestepping for ODEs, 459
 - relative, 460
- topology
 - of a mesh, 206
- topology layer, 204
- trace theorem, 120
- traffic flow
 - velocity model, 648
- trajectory, 420, 590
- transformation
 - bilinear, 261
- transformation of functions, 256
- transformation techniques, 263
- translation-invariant, 717
- transonic rarefaction, 704
- transport equation, 623, 643
- transport-dominated, 643
- transsonic rarefaction fan, 700
- trapezoidal rule, 452
 - composite, 143, 144
 - for ODEs, 452
 - global, 612
- trial space, 88
 - for collocation, 493
- triangle inequality, 24
- Triangle mesh generator, 202
- triangular mesh, 176
- triangulation, 175
 - in 2D, 148
- two-point boundary value problem, 95
- two-step method, 578
- Types of convergence, 294
- unconditional stability, 553
- uniform shape regularity, 369
- uniform shape-regularity, 326
- uniformly positive, 106
- Uniformly positive (definite) tensor field, 58
- unit triangle, 181, 239
- upwind quadrature, 610–612
- upwinding, 609
- validation
 - of FE codes, 369
- variational crime, 336
- variational formulation
 - spatial, 522
- variational problem
 - discrete, 130, 134
 - perturbed, 336
- vector field, 423
- velocity field, 590
- vertex, 175
- vertical force, 49
- von Neumann stability analysis, 550, 727
- Voronoi cell, 392

Voronoi dual mesh, 391

VTK, 203

wave equation, 564

weak (= generalized) gradient, 85

weak form, 101

weak solution, 665

Weak solution of Cauchy problem for scalar
conservation law, 665

Weak/generalized gradient, 85

weight

quadrature, 143

width

of a mesh, 293

Abbreviations and Acronyms

ASP $\hat{=}$ affine space, 22

BDC $\hat{=}$ boundary conditions, 50

BLF $\hat{=}$ bilinear form, 23, 60

BVP $\hat{=}$ boundary value problem, 46

CAD $\hat{=}$ computer-aided design, 276

CDE $\hat{=}$ convection-diffusion equation, 593

CFG-SPC $\hat{=}$ configuration space, 48

CSI $\hat{=}$ Cauchy-Schwarz inequality, 80

CSI $\hat{=}$ Cauchy-Schwarz inequality, 25

DOF $\hat{=}$ degree of freedom, 134, 179

DP $\hat{=}$ discretization parameter, 293

DQ $\hat{=}$ difference quotient, 376, 681

DVP $\hat{=}$ discrete variational problem, 130

E&U $\hat{=}$ existence and uniqueness, 63

FDM $\hat{=}$ finite difference method, 680

FDM $\hat{=}$ finite-difference method, 375

FEM $\hat{=}$ finite element method, 125, 129

FSP $\hat{=}$ function space, 22

FV $\hat{=}$ finite volume (scheme), 680

FV-MOL-ODE $\hat{=}$ semi-discrete evolution arising
conservative finite-colume

semi-discretization, 714

FVM $\hat{=}$ finite-volume method, 390

GALMAT $\hat{=}$ Galerkin matrix, 134

GD $\hat{=}$ Galerkin Discretization, 129

GO $\hat{=}$ Galerkin orthogonality, 287

GSF $\hat{=}$ global shape functions, 181

IbP $\hat{=}$ integration by parts, 94

IBP $\hat{=}$ integration by parts, 94

IBVP $\hat{=}$ initial-boundary value problem, 517

IRK-SSM $\hat{=}$ implicit Runge-Kutta single-step
method, 498

IVP $\hat{=}$ initial-value problem, 415

LagrFE $\hat{=}$ Lagrangian finite element space

$S_p^0(\mathcal{M})$, 184

LF $\hat{=}$ linear form, 23, 60

LSE $\hat{=}$ linear system of equations, 87, 133

LSF $\hat{=}$ local shape functions, 181

LVP $\hat{=}$ linear variational problem, 88

LWE $\hat{=}$ linear wave equation, 564

MOC $\hat{=}$ method of characteristics, 624

MPLR $\hat{=}$ monotonicity-preserving linear
reconstruction, 742

MW $\hat{=}$ mesh width, 293

NFF $\hat{=}$ numerical flux (function), 686

ODE $\hat{=}$ ordinary differential equation, 95, 415

p.d. $\hat{=}$ positive definite, 25

p.s.d. $\hat{=}$ positive semi-definite, 25

PDE $\hat{=}$ partial differential equation, 36

PMM $\hat{=}$ particle mesh method, 631

r.h.s $\hat{=}$ right-hand side, 416

RHS $\hat{=}$ right-hand side, 134

s.p.d. $\hat{=}$ positive definite, 25

SF $\hat{=}$ stability function, 478

SSM $\hat{=}$ single-step method, 437

SU $\hat{=}$ streamline upwinding, 616

UP $\hat{=}$ uniformly positive, 52

UPD $\hat{=}$ uniformly positive definite, 58

VS $\hat{=}$ vector space, 22

VSP $\hat{=}$ vector space, 22

List of Symbols

$C_0^\infty(\Omega) \doteq$	smooth functions with support inside Ω , 81	$Hf \doteq$	Hessian of a scalar valued function, 36
$C_{\text{pw}}^k([a, b]) \doteq$	piecewise k -times continuously differentiable functions on an interval, 31	$Hu \doteq$	Hessian of a scalar valued function, 30
$D^-(\bar{x}, \bar{t}) \doteq$	maximal analytical domain of dependence of (\bar{x}, \bar{t}) , 718	$H^m(\Omega) \doteq$	m -th order Sobolev space, 317
$D^\alpha u \doteq$	multiple partial derivatives, 317	$S_1^0(\mathcal{M}) \doteq$	151
$Df \doteq$	general derivative of a function f , 28	$I_1 \doteq$	piecewise linear interpolation on finite element mesh, 395
$H_*^1(\Omega) \doteq$	function is $H^1(\Omega)$ with vanishing mean, 113	$P_n \doteq$	n -th Legendre polynomial, 400
$H^2(\mathcal{M}) \doteq$	Sobolev space of functions \mathcal{M} -piecewise $\in H^2$, 618	$S_p^0(\mathcal{M}) \doteq$	$H^1(\Omega)$ -conforming Lagrangian FE space, 184
$J(t_0, \mathbf{y}_0) \doteq$	maximal domain of definition of a solution of an IVP, 425	$L^\infty(\Omega) \doteq$	space of (essentially) bounded functions on Ω , 32
$O(f(N)) \doteq$	Landau- O for $N \rightarrow \infty$, 294	$L^\infty(\Omega) \doteq$	vector space of essentially bounded functions on Ω , 32
$S(z) \doteq$	stability function of Runge-Kutta method, 726	$L^2(\Omega) \doteq$	space of square-integrable functions on Ω , 74
$A, B, \dots \doteq$	real or complex matrices, 22	$\ u\ _{H^m(\Omega)} \doteq$	m -th order Sobolev norm, 317
$a, b, \dots \doteq$	“small” coordinate vectors, 22	$\ u\ _{L^\infty(\Omega)} \doteq$	supremum norm of $u : \Omega \mapsto \mathbb{R}^n$, 32
n	107	$\ \cdot\ _{L^2(\Omega)} \doteq$	L^2 -norm of a function, 32
$n \doteq$	exterior unit normal vectorfield, 96	$\ \cdot\ _{L^2(\Omega)} \doteq$	norm on $L^2(\Omega)$, 74
$\mathcal{H}_h \doteq$	fully discrete evolution operator, 716	$\ \cdot\ _0 \doteq$	L^2 -norm of a function, 32
$\mathcal{L}(V, W) \doteq$	space of linear mappings $V \mapsto W$, 28	$\mathcal{V}(\mathcal{M}) \doteq$	176
$\mathcal{L}_h \doteq$	semi-discrete evolution operator doe 1D conservation law, 715	$\ \cdot\ _0 \doteq$	norm on $L^2(\Omega)$, 74
$C^0(\Omega) \doteq$	vector space of continuous functions on $\Omega \subset \mathbb{R}^d$, 31	$\ \cdot\ _\infty \doteq$	supremum norm of a function/maximum norm of a vector, 32
$C^k(\Omega) \doteq$	k -times continuous differentiable real-valued functions on Ω , 31	$\ \cdot\ _\infty \doteq$	supremum norm (of function), maximum norm (for a vector), 32
$\mathcal{P}_p(\mathbb{R}^d) \doteq$	space of d -variate polynomials, 178	$\Omega \doteq$	(spatial) domain, 46
$\mathcal{Q}_p(\mathbb{R}^d) \doteq$	179	Φ^*	256
$\mathcal{S}_{3,\mathcal{T}} \doteq$	cubic spline space, knot set \mathcal{T} , 410	$\Phi^{s,t} \mathbf{u} \doteq$	evolution operator for an ODE, 531
$\mathcal{V}(\mathcal{M}) \doteq$	set of vertices of a mesh, 148	$\Psi^h \mathbf{y} \doteq$	discrete evolution for autonomous ODE, 437
$\Delta \doteq$	Laplace operator, 100	$ u _{H^m(\Omega)}$	m -th order Sobolev semi-norm, 318
$Df \doteq$	Jacobian of a differentiable function, 29, 36	$\mathbf{A}, \mathbf{B}, \mathbf{C}, \dots$ (matrices)	(matrices), 134
$D_{\mathbf{y}} \mathbf{f} \doteq$	Derivative of \mathbf{f} w.r.t. \mathbf{y} (Jacobian), 425	$\mathbf{S}^{-\top}$	hat= inverse transposed of matrix \mathbf{S} , 266
$\text{div } \mathbf{j} \doteq$	divergence of a vector field, 96	$\mathbf{1} \doteq$	vector with entries all equal to 1, 22
I_1	312	$\mathbf{0} \doteq$	zero vector $\in \mathbb{R}^n$, 22
$\Gamma_{\text{in}} \doteq$	inflow boundary for advection BVP, 603	$\approx \doteq$	two-sided uniform estimate, 326
		$H_{\Gamma_D}^1(\Omega) \doteq$	functions in $H^1(\Omega)$ with zero trace on Γ_D , 368
		$\mathbb{C}^- := \{z \in \mathbb{C}: \text{Re } z < 0\}$, 500

$S^2 \hat{=} \text{unit sphere}, 42$	$\text{differentiably functions}, 31$
$a_K \hat{=} \text{restriction of bilinear form } a \text{ to cell } K, 158$	$\rho_K \hat{=} \text{shape regularity measure of cell } K, 316, 329$
$\chi_I \hat{=} \text{characteristic function of an interval } I \subset \mathbb{R}, 686$	$\rho_{\mathcal{M}} \hat{=} \text{shape regularity measure of a mesh } \mathcal{M}, 316, 329$
$\cdot \hat{=} \text{differentiation with respect to time } t, 530$	$f _D \hat{=} \text{restriction to a function to a set } D, 32$
$\ddot{u} := \frac{\partial u}{\partial t^2}, 563$	$\vec{e}^k \hat{=} k\text{-th unit vector in } \mathbb{R}^n, 22$
$\delta_{ij} \hat{=} \text{Kronecker symbol}, 22$	$\vec{\mu}, \vec{\eta} \hat{=} \text{"large" coefficient vectors}, 22$
$\dot{u}(t) \hat{=} (\text{partial}) derivative w.r.t. time, 522$	$\vec{\mu}, \vec{\phi}, \vec{\xi}, \dots \hat{=} (\text{coefficient vectors}), 134$
$\ell_K \hat{=} \text{restriction of linear form } \ell \text{ to cell } K, 168$	$S_{p,0}^0(\mathcal{M}) \hat{=} \text{Degree } p \text{ Lagrangian finite element space with zero Dirichlet boundary conditions.}, 191$
$\frac{Df}{D\mathbf{v}}(t) \hat{=} \text{material derivative w.r.t. velocity field } \mathbf{v}, 635$	$S_{1,0}^0(\mathcal{M}) \hat{=} \text{space of p.w. linear } C^0\text{-finite elements}, 139$
grad $\hat{=} \text{gradient of a scalar valued function}, 53$	$H_0^1(\Omega) \hat{=} \text{Sobolev space}, 78$
$\hat{c}(\xi) \hat{=} \text{symbol of a finite difference operator}, 723$	$f', f'', f^{(k)} \hat{=} \text{derivatives of a function in 1D}, 28$
$(\vec{\mu})_j \hat{=} j\text{-th component of vector } \vec{\mu}, 22$	$f(\cdot) = O(g(\cdot)) \hat{=} \text{Landau "O"-notation}, 28$
$\lesssim \hat{=} \leq c \cdot, \text{with a "generic constant" } C > 0, 92$	$h_{\mathcal{M}} \hat{=} \text{mesh width of mesh } \mathcal{M}, 293$
$\mathbf{1} = (1, \dots, 1)^T, 726$	$h_{\mathcal{M}} \hat{=} \text{meshwidth of a grid}, 139$
$\mathbf{1} = [1, \dots,]^\top, 478$	$x_{j-\frac{1}{2}} := \frac{1}{2}(x_j + x_{j-1}) \hat{=} \text{midpoint of cell in 1D}, 685$
$dS \hat{=} \text{integration over a surface}, 96$	$\mathbf{x}, \mathbf{p}, \mathbf{y}, \dots \hat{=} \text{small vectors, coordinate vectors of points}, 50$
$\mathcal{M}, 176$	$ v _{H^1(\Omega)} = v _{1,\Omega} = v _1 \hat{=} \text{semi-norm on } H^1(\Omega), 78$
$\nabla F(\mathbf{x}) := \mathbf{grad} F(\mathbf{x}) \hat{=} \text{nabla notation for gradient}, 53$	$\mathbf{f} \hat{=} \text{right hand side of an ODE}, 423$
$\text{cond}(\mathbf{A}), 401$	$\partial\Omega \hat{=} \text{boundary of the domain } \Omega, 50$
$\text{diam}(\Omega) \hat{=} \text{diameter of } \Omega \subset \mathbb{R}^d, 51$	$\cdot \hat{=} \text{Derivative w.r.t. time } t, 416$
$\text{nnz}, 157$	$\text{TOL tolerance}, 459$
$\overline{D} \hat{=} \text{the closure of a set } D \subset \mathbb{R}^d, 148$	
$\overline{\Omega} \hat{=} \text{closure of domain } \Omega, 51$	
$C_{\text{pw},0}^k(\Omega) \hat{=} \text{piecewise } k\text{-times continuously differentiably functions vanishing on the boundary}, 32$	
$C_{\text{pw}}^k(\Omega) \hat{=} \text{piecewise } k\text{-times continuously}$	

Examples and Remarks

- L^2 -convergence of FE solutions, 353
 L^2 -estimates on non-convex domain, 355
 h -convergence of Lagrangian FEM on L-shaped domain, 302
 \mathbb{R} by completion of \mathbb{Q} , 77
(Bi)-linear Lagrangian finite elements on hybrid meshes, 191
 $|\cdot|_{H^1(\Omega)}$ -seminorm, 80
Gmsh – meshing more complex geometries, 201
Gmsh file format for storing meshes, 198
Gmsh geometry description file, 197
LEHRFEM++ – A simple but flexible C++ finite element library, 194
LEHRFEM++ – building a mesh from Gmsh mesh file, 202
LEHRFEM++ interface to local computations, 227
LEHRFEM++ numbering *convention* for global shape functions, 222
“Butcher barriers” for explicit RK-SSM, 455
“Energy-geometry”, 287
“Explosion equation”: finite-time blow-up, 426
“Failure” of adaptive timestepping, 463
“Generic constants”, 324
“Wrapped rock on a stove”, 108
L-stable implicit Runge-Kutta methods, 503
1D convection-diffusion boundary value problem, 601
6-Node triangles in LEHRFEM++, 277
- Gmsh** – marking parts of a mesh by tags, 200
- A function that is not locally Lipschitz continuous, 425
Acceleration based traffic modeling, 648
Accessing corner coordinates in LEHRFEM++, 213
Accessing sub-entities in LEHRFEM++, 207
Actual shock patterns in traffic flow, 669
Adaptive explicit high-order Runge-Kutta method for discrete parabolic evolution, 547
- Adaptive explicit RK-SSM for scalar linear decay ODE, 473
Adaptive timestepping for mechanical problem, 469
Adequacy of 2nd-order timestepping, 747
Alternative (“legacy”) terminology, 134
Alternative computation of element matrix for $-\Delta$, 160
Approximate computation of error norms, 297
Approximate Dirichlet boundary conditions, 249
Approximate sub-steps for Strang splitting time, 628
Approximation of mean temperature, 344, 345
Assembly of boundary contributions, 230
Assembly of Galerkin linear system for homogeneous Neumann problem, 229
Assembly of Galerkin matrices in LEHRFEM++, 225
Assembly of right hand side vector for linear finite elements, 169
Asymptotic nature of convergence, 305
- Behavior of generalized eigenvalues of $\mathbf{A}\vec{\mu} = \lambda\mathbf{M}\vec{\mu}$, 550
Benefit of variational formulations of BVPs, 141
Bi-linear tensor-product Lagrangian finite element functions, 190
Bilinear forms on \mathbb{R}^n , 23
Bilinear Lagrangian finite elements, 188
Blow-up, 457
Blow-up for leapfrog timestepping, 581
Blow-up of explicit Euler method, 474
Boundary conditions and $L^2(\Omega)$, 74
Boundary conditions and density, 77
Boundary conditions for linear advection, 647
Boundary conditions for wave equation, 565
Boundary conditions in $H_0^1(\Omega)$, 78
Boundary values for conservation laws, 657
Butcher scheme for some explicit RK-SSM, 454
- Cell-dependent evaluations for bilinear transformations, 268

- Cell-local shape functions for $\mathcal{S}_1^0(\mathcal{M})$ in 2D, 181
 Central flux for Burgers equation, 691
 Central flux for linear advection, 694
 Central flux for Traffic Flow equation, 692
 Characteristics for advection, 659
 Characteristics of stiff IVPs, 491
 Choice of bases for polynomial spectral Galerkin methods, 400
 Choice of timestepping for m.o.l. for transient convection-diffusion, 622
 Circumcenter-based construction: Geometric obstruction, 392
 Closed streamlines, 603
 Commonly used embedded explicit Runge-Kutta methods, 466
 Compatibility conditions for 1D FVM for Neumann two-point BVP, 380
 Compatible boundary and initial data, 519
 Computation of heat flux, 348, 351
 Computation of norms of finite element discretization errors in LEHRFEM++, 298
 Computing element matrices for parametric FEM in LEHRFEM++, 273
 Conditioning of polynomial spectral Galerkin matrices, 401
 Connection with artificial viscosity, 695
 Connection with convection-diffusion IBVPs → Chapter 10, 694
 Consequence of monotonicity preservation, 743
 Conservation law and flux for transient heat conduction, 656
 Conservation of volume content, 646
 Consistency error of Lax-Friedrichs/Rusanov numerical flux, 732
 Consistency error of upwind numerical flux, 732
 Consistency of implicit midpoint method, 438
 Constant advection in 1D, 645
 Construction of higher order Runge-Kutta single step methods, 455
 Continuity of interpolation operators, 318
 Continuity up to the boundary, 51
 Convective cooling, 108
 Convergence for conditionally stable Runge-Kutta timestepping, 560
 Convergence for linear and quadratic Lagrangian finite elements in energy norm, 300
 Convergence of Euler timestepping for MOL ODE, 545
 Convergence of finite-difference single-step methods → Exp. 6.3.2.5, 536
 Convergence of fully discrete finite volume methods for Burgers equation, 729
 Convergence of fully discrete timestepping in one spatial dimension, 557
 Convergence of FV with linear reconstruction, 737
 Convergence of inexact simple splitting methods, 510
 Convergence of Lagrangian FEM for p -refinement, 304
 Convergence of linear and quadratic Lagrangian finite elements in L^2 -norm, 301
 Convergence of linear finite element method for two-point BVP, 299
 Convergence of MUSCL scheme, 748
 Convergence of naive semi-implicit Radau method, 507
 Convergence of polynomial spectral Galerkin method in 1D, 403
 Convergence of semi-implicit midpoint method, 506
 Convergence of simple Runge-Kutta methods, 452
 Convergence of simple splitting methods, 509
 Convergence of spectral collocation in 1D, 409
 Convergence of streamline-diffusion and upwind quadrature FEM, 619
 Conversion into non-dimensional form by scaling, 598
 Corner singular functions, 332
 Crank-Nicolson timestepping, 544
 Curse of dimension, 328
 Data in procedural form, 128
 Degenerate elliptic boundary value problem, 37
 Delaunay-remeshing in 2D, 632
 Diagonalization in \mathbb{C} , 723
 Differentiating bilinear forms with time-dependent arguments, 525
 Diffusive flux, 656
 Discontinuous solutions of advection equations, 645
 Discrete evolutions for non-autonomous ODEs, 438
 Discrete quadratic minimization problems, 130
 Discrete variational problems in affine spaces, 132
 Domain of dependence/influence for 1D wave equation, constant coefficient case, 568
 Duality estimate for modified heat flux functional, 351

- Effect of added diffusion, 615
 Efficient assembly of sparse Galerkin matrices, 167
 Eigenvectors of translation invariant linear operators, 723
 Element matrix for quadratic Lagrangian finite elements, 236
 Elliptic lifting result for rectangular domains, 346
 Elliptic lifting result in 1D, 331
 Empiric Convergence of collocation single step methods, 494, 540
 Empiric convergence of semi-implicit Euler single-step method, 505
 Energy conservation for leapfrog, 579
 Energy norm and $H^1(\Omega)$ -norm, 317
 Entropy solution of Burgers equation, 676
 Entropy solution of Traffic Flow equation, 676
 Equivalent formulations of elliptic BVPs, 100
 Estimation of “wrong” error?, 462
 Euler equations, 654
 Euler methods for stiff decay IVP, 491
 Euler timestepping, 543
 Euler timestepping for 1st-order form of semi-discrete wave equation, 574
 Evolution operator for Lotka-Volterra ODE, 428
 Evolution problems for PDEs and ODEs, 520
 Existence of minimizers in finite dimensions, 68
 Explicit adaptive RK-SSM for stiff IVP, 472
 Explicit Euler in Fourier domain, 725
 Explicit Euler method as a difference scheme, 433
 Explicit Euler method for damped oscillations, 483
 Explicit trapezoidal rule for decay equation, 476
 Extra smoothness of source function in finite difference approach, 383
 Extra smoothness requirement for PDE formulation, 100
 Fan patterns in traffic flow, 670
 Feasibility of implicit Euler timestepping, 434
 Finding continuous replacement functionals, 352
 Finite difference single-step methods, 534
 Finite differences for convection-diffusion equation in 1D, 605
 Finite element meshes with hanging nodes, 177
 Finite element methods on surfaces, 270
 Finite-difference discretization of 1D linear advection with constant velocity, 682
 First-order semidiscrete hyperbolic evolution problem, 573
 Flux functions for simple scalar conservation laws, 657
 Fourier series, 724
 Fourier spectral Galerkin methods, 404
 From higher order ODEs to first order systems, 424
 Fully discrete evolutions arising from conservative discretizations, 716
 Function space valued functions, 522
 FVM: Incorporation of Dirichlet boundary conditions, 394
 Gain through adaptivity, 462
 Gap between interpolation error and best approximation error, 322
 Gauss-Radau collocation SSM for stiff IVP, 504
 General asymptotic estimates, 328
 General entropy solution for 1D scalar Riemann problem, 675
 Geometric interpretation of CFL condition in 1D, 583
 Geometric modeling with **Gmsh**, 196
 Global assembly of right-hand-side vector in LEHRFEM++, 227
 Global regular refinement in LEHRFEM++, 289
 Godunov flux for Burgers equation, 706
 Godunov flux for traffic flow equation, 707
 Good accuracy on “bad” meshes, 321
 Group property of autonomous evolutions, 429
 Growth with limited resources, 418
 Guessing timestep constraint, 560
 Heartbeat model, 420
 Higher order timestepping for 1D heat equation, 558
 Impact of efficient initialization of sparse Galerkin matrix, 167
 Impact of linear boundary approximation on FE convergence, 339
 Impact of numerical quadrature on finite element discretization error, 338
 Implicit Euler method of lines for transient convection-diffusion, 621
 Implicit midpoint rule for semi-discrete wave equation, 576
 Implicit nature of collocation single step methods, 494
 Implicit RK-SSMs for stiff IVP, 501
 Implicit sharpness of asymptotic convergence, 294
 Importance of discrete maximum principle, 362

- Importance of global numbering of geometric entities, 204
- Important Banach spaces and Hilbert spaces, 76
- Improved resolution by limited linear reconstruction, 746
- Index mapping by d.o.f. mapper, 165
- Index mapping matrix for linear Lagrangian finite elements on triangular mesh, 232
- Indexing and numbering of (sub-)entities in LEHRFEM++, 211
- Initial time, 517
- Inspecting mesh topology in LEHRFEM++, 211
- Internal array representation of 2D triangular mesh, 150
- Internal layers, 616
- Justification for teaching Sobolev spaces, 83
- Kinetics of chemical reactions, 486
- $L(\pi)$ -stable Runge-Kutta single step methods, 555
- Lagrangian finite elements on hybrid meshes, 192
- Lagrangian method for convection-diffusion in 1D, 632
- Lagrangian method for convection-diffusion in 2D, 633
- Laplace operator, 100
- Lax-Friedrichs flux for Burgers equation, 696
- Lax-Friedrichs flux for traffic flow equation, 697
- Leap-frog implementation of Strang splitting, 628
- Learning LEHRFEM++, 194
- Linear BVP, 108
- Linear FE discretization of 1D convection-diffusion problem, 605
- Linear forms from bilinear forms, 24
- Linear forms on \mathbb{R}^n , 23
- Linear reconstruction with central slope (Burgers equation), 738
- Linear reconstruction with central slope (traffic flow), 739
- Linear reconstruction with minmod limiter, 745
- Linear reconstruction with minmod limiter (Burgers' equation), 744
- Linear reconstruction with one-sided slopes (Burgers equation), 739
- Linear reconstruction with one-sided slopes (traffic flow), 740
- Linear variational problems on affine spaces, 130
- Linearity and monotonicity preservation, 743
- Local computations based on affine equivalence, 258
- Local interpolation nodes for cubic ($p = 3$) and quartic ($p = 4$) Lagrangian FE in 2D, 187
- Local quadrature rules on quadrilaterals, 244
- Local quadrature rules on triangles, 243
- Local→global index mapping and index array, 219
- Local→global mapping for linear Lagrangian finite elements on triangular mesh, 219
- Low-dimensional configuration spaces, 49
- LSFs for $\mathcal{S}_1^0(\mathcal{M})$ on edges, 182
- M-matrices, 364
- Mass lumping, 579
- Mathematical notion of $L^2(\Omega)$, 74
- Maximum principle for higher order Lagrangian FEM, 366
- Maximum principle for linear FE for 2nd-order elliptic BVPs, 366
- Meaning of characteristics, 662
- Mixed boundary conditions, 108
- Modules of LEHRFEM++, 195
- More general finite-volume methods, 397
- Naive finite difference scheme for Burgers equation, 684
- Necessary condition for L-stability, 502
- Non-degenerate parametric mappings, 264
- Non-differentiable function in $H_0^1(0,1)$, 85
- Non-dimensional equations, 53
- Non-existence of solutions of positive definite quadratic minimization problem, 71
- Non-polynomial “bilinear” local shape functions, 262
- Non-smooth d'Alembert solutions, 568
- Notation for “discrete entities”, 130
- Notation for single step methods, 439
- Numbering of global shape functions, 223
- Numbering of local shape functions for quadratic Lagrangian finite elements, 220
- Offset function for elastic string model, 62
- Offset function for finite element Galerkin discretization, 145
- Offset functions for linear Lagrangian FE, 249
- One-sided difference approximation of convective terms, 607
- Ordering of global shape functions required?, 252
- Orders of finite-difference single-step methods, 447, 534
- Oregonator reaction, 456

- Orientation of sub-entities, 208
- Other tools for mesh generation, 202
- Output functionals, 343
- Output of explicit Euler method, 433
- Over-/Undershoots in linear reconstruction, 741
- Parametric bilinear finite elements in LEHRFEM++, 263
- Particle method adapted to inflow/outflow, 630
- Particle simulation of traffic flow, 649
- Piecewise gradient, 152
- Piecewise linear functions (not) in $H_0^1(0,1)$, 84
- Piecewise quadratic interpolation, 324
- Point load, 68
- Point loading of elastic string, 54
- Point particle method for pure advection, 630
- Positive definite matrices, 64
- Potential inefficiency of conditionally stable single step methods, 560
- Predator-prey model, 419
- Predicting stiffness of non-linear IVPs, 490
- Processing extra information in Gmsh mesh file with LEHRFEM++, 203
- Properties of the Galerkin matrix inherited from a, 136
- Properties of weak solutions, 665
- Pythagoras' theorem, 287
- Quadratic functionals on \mathbb{R}^N , 61
- Quadratic functionals with positive definite bilinear form in 2D, 64
- Quadratic minimization problem in $L^2(\Omega)$, 76
- Quadratic tensor product Lagrangian finite elements, 190
- Radiative cooling, 108
- Rationale for high-order single step methods, 450
- Recasting quadratic minimization problems on \mathbb{R}^N , 86
- Regions of stability for simple implicit RK-SSM, 499
- Regions of stability of some explicit RK-SSM, 485
- Regular/uniform refinement of triangular mesh in 2D, 289
- Repetition: Well-posed 2nd-order linear elliptic variational problems, 285
- RK-SSM and quadrature rules, 454
- Second-order geometry approximation in GMSH, 276
- Semi-implicit Euler single-step method, 505
- Semi-Lagrangian method for convection-diffusion in 1D, 638
- Semi-Lagrangian method for convection-diffusion in 2D, 639
- Simple adaptive stepsize control, 462
- Simple adaptive timestepping for fast decay, 476
- Simple collocation single-step methods, 540
- Simple Runge-Kutta methods by quadrature & bootstrapping, 452
- SIR model, 421
- Solution of a Dirichlet BVP with LEHRFEM++, 253
- Solving the stage equations for implicit RK-SSMs, 497
- Sparse stiffness matrices, 157
- Spatial discretization options, 528
- Special case: Linear system of equations for linear finite element discretization on equidistant mesh, 142
- Spectral collocation on a square, 408
- Spectrum of elliptic operators, 551
- Spectrum of upwind difference operator, 723
- Speed of convergence of polygonal methods, 442
- Splitting linear and decoupling terms, 511
- Splitting off stiff components, 510
- Spurious Galerkin solution for 2D convection-diffusion BVP, 609
- Stability and CFL condition, 727
- Stability domains, 727
- Stability function and exponential function, 478
- Stability functions of explicit RK-methods, 726
- Stability functions of explicit Runge-Kutta single step methods, 478
- Stage form equations for increments, 497
- Stencils on more general meshes, 384
- Stepsize control detects instability, 480
- Storing topology of triangular mesh in 2D, 209
- Streamline-diffusion discretization: Internal layer, 619
- Strongly attractive limit cycle, 487
- Suitability of macroscopic models for traffic flow, 650
- Summary: Impact of choice of basis, 136
- Supports of global shape functions in 1D, 179
- Supports of global shape functions on triangular mesh, 180
- Symmetric difference quotient for second derivative, 378
- Tangent field and solution curves, 431
- Taut membrane with free boundary values, 101

- Temporally varying spatial domains, 516
Testing for topological type, 207
The quadrature challenge on general domains, 404
The rationale behind learning LEHRFEM++, 195
Tolerances and accuracy, 468
Traffic flow: Evolution of smooth initial density, 662
Transient circuit simulation, 422
Transient simulation of RLC-circuit, 480
Treatment of Neumann boundary conditions in finite volume schemes, 394
Triangular quadratic ($p = 2$) Lagrangian finite elements, 185
Truncation of unbounded domain, 37
Uniqueness of solutions of the Neumann problem, 112
Upwind difference operator for linear advection, 722
Upwind flux and expansion shocks, 705
Upwind flux and transsonic rarefaction, 700
Upwind flux for Burgers equation, 698
Upwind flux for traffic flow simulation, 699
Upwind flux: Convergence to expansion shock, 702
Upwind quadrature discretization, 613
Use of methods of `If::assemble::DofHandler`, 221
Usefulness of L^2 -estimates, 356
Using entity iterators in LEHRFEM++, 205
Vanishing viscosity for Burgers equation, 670
Variational formulation for heat conduction with Dirichlet boundary conditions, 110
Variational formulation for Neumann problem, 111
Variational formulation: heat conduction with general radiation boundary conditions, 111
Variational problems with different trial and test spaces, 226
Visualization of explicit Euler method, 432
von Neumann stability analysis, 550
Why “numerical integration”?, 415

C++ Codes and More

If::mesh::Entity, 204

If::io::GmshReader, 202

If::mesh::Mesh, 204

If::mesh::Mesh::Contains(), 205

If::mesh::Mesh::Entities(), 205

If::mesh::MeshFactory, 202

nostd::span, 205

If::geometry::Corners, 213

If::geometry::Geometry::Global, 240

If::geometry::Geometry::IntegrationElement, 241

If::mesh::Entity::Geometry (), 213

If::mesh::Mesh::EntityByIndex(), 205

If::mesh::Mesh::Index(), 205

If::mmesh::Mesh::NumEntities(), 205

ReactionDiffusionElementMatrixProvider::Eval(),

274