

# Numerical Methods

Andreas Stahel

Version of March 13, 2023

©Andreas Stahel, 2007–2021

All rights reserved. This work may not be translated or copied in whole or in part without the written permission by the author, except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software is forbidden.

# Contents

<b>0 Introduction</b>	<b>1</b>
0.1 Using these Lecture Notes . . . . .	1
0.2 Content and Goals of this Class . . . . .	1
0.3 Literature . . . . .	2
0.3.1 A Selection of Literature on Numerical Methods . . . . .	2
0.3.2 A Selection of Literature on the Finite Element Method . . . . .	3
Bibliography . . . . .	5
<b>1 The Model Problems</b>	<b>8</b>
1.1 Heat Equations . . . . .	8
1.1.1 Description . . . . .	8
1.1.2 The One Dimensional Heat Equation . . . . .	9
1.1.3 The Steady State Problem . . . . .	10
1.1.4 The Dynamic Problem, Separation of Variables . . . . .	10
1.1.5 The Two Dimensional Heat Equation . . . . .	11
1.1.6 The Steady State 2D–Problem . . . . .	12
1.1.7 The Dynamic 2D–Problem . . . . .	12
1.2 Vertical Deformations of Strings and Membranes . . . . .	13
1.2.1 Equation for a Steady State Deformation of a String . . . . .	13
1.2.2 Equation for a Vibrating String . . . . .	14
1.2.3 The Eigenvalue Problem . . . . .	14
1.2.4 Equation for a Vibrating Membrane . . . . .	14
1.2.5 Equation for a Steady State Deformation of a Membrane . . . . .	15
1.2.6 Eigenvalue Problem for a Membrane . . . . .	15
1.3 Horizontal Stretching of a Beam . . . . .	15
1.3.1 Description . . . . .	15
1.3.2 Poisson’s Ratio, Lateral Contraction . . . . .	17
1.3.3 Nonlinear Stress Strain Relations . . . . .	18
1.4 Bending and Buckling of a Beam . . . . .	18
1.4.1 Description . . . . .	18
1.4.2 Bending of a Beam . . . . .	20
1.4.3 Buckling of a Beam . . . . .	20
1.5 Tikhonov Regularization . . . . .	21
Bibliography . . . . .	22
<b>2 Matrix Computations</b>	<b>23</b>
2.1 Prerequisites and Goals . . . . .	23
2.2 Floating Point Operations . . . . .	23
2.2.1 Floating Point Numbers and Rounding Errors in Arithmetic Operations . . . . .	23
2.2.2 Flops, Accessing Memory and Cache . . . . .	26
2.2.3 Multi Core Architectures . . . . .	30

2.2.4 Using Multiple Cores of a CPU with <i>Octave</i> and <i>MATLAB</i> . . . . .	31
<b>2.3 The Model Matrices . . . . .</b>	<b>32</b>
2.3.1 The 1-d Model Matrix $A_n$ . . . . .	32
2.3.2 The 2-d Model Matrix $A_{nn}$ . . . . .	34
<b>2.4 Solving Systems of Linear Equations and Matrix Factorizations . . . . .</b>	<b>35</b>
2.4.1 LR Factorization . . . . .	36
2.4.2 LR Factorization and Elementary Matrices . . . . .	43
<b>2.5 The Condition Number of a Matrix, Matrix and Vector Norms . . . . .</b>	<b>45</b>
2.5.1 Vector Norms and Matrix Norms . . . . .	45
2.5.2 The Condition Number of a Matrix . . . . .	50
2.5.3 The Effect of Rounding Errors, Pivoting . . . . .	52
<b>2.6 Structured Matrices . . . . .</b>	<b>55</b>
2.6.1 Symmetric Matrices, Algorithm of Cholesky . . . . .	56
2.6.2 Positive Definite Matrices . . . . .	60
2.6.3 Stability of the Algorithm of Cholesky . . . . .	64
2.6.4 Banded Matrices and the Algorithm of Cholesky . . . . .	66
2.6.5 Computing with an Inverse Matrix is Usually Inefficient . . . . .	68
2.6.6 <i>Octave</i> Implementations of Sparse Direct Solvers . . . . .	69
2.6.7 A Selection Tree used in <i>Octave</i> for Sparse Linear Systems . . . . .	72
<b>2.7 Sparse Matrices and Iterative Solvers . . . . .</b>	<b>72</b>
2.7.1 The Model Problems . . . . .	73
2.7.2 Basic Definitions . . . . .	73
2.7.3 Steepest Descent Iteration, Gradient Algorithm . . . . .	74
2.7.4 Conjugate Gradient Iteration . . . . .	78
2.7.5 Preconditioning . . . . .	84
2.7.6 The Incomplete Cholesky Preconditioner . . . . .	87
2.7.7 Conjugate Gradient Algorithm with an Incomplete Cholesky Preconditioner . . . . .	89
2.7.8 Conjugate Gradient Algorithm with an Incomplete LU Preconditioner . . . . .	91
<b>2.8 Iterative Solvers for Non-Symmetric Systems . . . . .</b>	<b>92</b>
2.8.1 Normal Equation, Conjugate Gradient Normal Residual (CGNR) and BiCGSTAB . . . . .	92
2.8.2 Generalized Minimal Residual, GMRES and GMRES(m) . . . . .	93
<b>2.9 Iterative Solvers in MATLAB/Octave and a Comparison with Direct Solvers . . . . .</b>	<b>99</b>
2.9.1 Iterative Solvers in MATLAB/Octave . . . . .	99
2.9.2 A Comparison of Direct and Iterative Solvers . . . . .	99
<b>2.10 Other Matrix Factorizations . . . . .</b>	<b>100</b>
<b>Bibliography . . . . .</b>	<b>100</b>
<b>3 Numerical Tools . . . . .</b>	<b>102</b>
3.0.1 Prerequisites and Goals . . . . .	102
<b>3.1 Nonlinear Equations . . . . .</b>	<b>103</b>
3.1.1 Introduction . . . . .	103
3.1.2 Bisection, Regula Falsi and Secant Method to Solve one Equation . . . . .	105
3.1.3 Systems of Equations . . . . .	110
3.1.4 The Contraction Mapping Principle and Successive Substitutions . . . . .	111
3.1.5 Newton's Algorithm to Solve Systems of Equations . . . . .	115
3.1.6 Modifications of Newton's Method . . . . .	118
3.1.7 <i>Octave/MATLAB</i> Commands to Solve Equations . . . . .	120
3.1.8 Examples of Nonlinear Equations . . . . .	121
3.1.9 Optimization with MATLAB/Octave . . . . .	128
<b>3.2 Eigenvalues and Eigenvectors of Matrices, SVD, PCA . . . . .</b>	<b>130</b>
3.2.1 Matrices and Linear Mappings . . . . .	130

3.2.2	Eigenvalues and Diagonalization of Matrices . . . . .	131
3.2.3	Level Sets of Quadratic Forms . . . . .	136
3.2.4	Commands for Eigenvalues and Eigenvectors in <i>Octave/MATLAB</i> . . . . .	141
3.2.5	Eigenvalues and Systems of Ordinary Differential Equations . . . . .	144
3.2.6	SVD, Singular Value Decomposition . . . . .	149
3.2.7	From Gaussian Distribution to Covariance, and then to PCA . . . . .	152
3.3	Numerical Integration . . . . .	163
3.3.1	Integration of a Function Given by Data Points . . . . .	163
3.3.2	Integration of a Function given by a Formula . . . . .	169
3.3.3	Integration Routines Provided by MATLAB/ <i>Octave</i> . . . . .	175
3.3.4	Integration over Domains in $\mathbb{R}^2$ . . . . .	178
3.4	Solving Ordinary Differential Equations, Initial Value Problems . . . . .	182
3.4.1	Different Types of Ordinary Differential Equations . . . . .	182
3.4.2	The Basic Algorithms . . . . .	186
3.4.3	Stability of the Algorithms . . . . .	195
3.4.4	General Runge–Kutta Methods, Represented by Butcher Tables . . . . .	200
3.4.5	Adaptive Step Sizes and Extrapolation . . . . .	205
3.4.6	ODE solvers in MATLAB/ <i>Octave</i> . . . . .	207
3.4.7	Comparison of four Algorithms Available with <i>Octave/MATLAB</i> . . . . .	213
3.5	Linear and Nonlinear Regression, Curve Fitting . . . . .	220
3.5.1	Linear Regression, Method of Least Squares . . . . .	220
3.5.2	Estimation of the Variance of Parameters, Confidence Intervals, Domain of Confidence	225
3.5.3	The commands <code>LinearRegression()</code> , <code>regress()</code> and <code>lscov()</code> for <i>Octave</i> and MATLAB . . . . .	227
3.5.4	An Elementary Example . . . . .	230
3.5.5	How to Obtain Wrong Results! . . . . .	236
3.5.6	A Regression Example with Multiple Independent Variables . . . . .	238
3.5.7	Introduction to Nonlinear Regression . . . . .	239
3.5.8	Nonlinear Regression with a Logistic Function . . . . .	243
3.5.9	Additional Commands from the Package <code>optim</code> in <i>Octave</i> . . . . .	248
3.6	Resources . . . . .	248
	Bibliography . . . . .	249
<b>4</b>	<b>Finite Difference Methods</b> . . . . .	<b>253</b>
4.1	Prerequisites and Goals . . . . .	253
4.2	Basic Concepts . . . . .	253
4.2.1	Finite Difference Approximations of Derivatives . . . . .	253
4.2.2	Finite Difference Stencils . . . . .	256
4.3	Consistency, Stability and Convergence . . . . .	257
4.3.1	A Finite Difference Approximation of an Initial Value Problem . . . . .	257
4.3.2	Explicit Method, Conditional Stability . . . . .	258
4.3.3	Implicit Method, Unconditional Stability . . . . .	258
4.3.4	General Difference Approximations, Consistency, Stability and Convergence . . . . .	259
4.4	Boundary Value Problems . . . . .	263
4.4.1	Two Point Boundary Value Problems . . . . .	263
4.4.2	Boundary Values Problems on a Rectangle . . . . .	270
4.5	Initial Boundary Value Problems . . . . .	272
4.5.1	The Dynamic Heat Equation . . . . .	272
4.5.2	Construction of the Solution Using Eigenvalues and Eigenvectors . . . . .	274
4.5.3	Explicit Finite Difference Approximation to the Heat Equation . . . . .	275
4.5.4	Implicit Finite Difference Approximation to the Heat Equation . . . . .	277

4.5.5	Crank–Nicolson Approximation to the Heat Equation . . . . .	279
4.5.6	General Parabolic Problems . . . . .	281
4.5.7	A two Dimensional Dynamic Heat Equation . . . . .	282
4.5.8	Comparison of a Few More Solvers for Dynamic Problems . . . . .	284
4.6	Hyperbolic Problems, Wave Equations . . . . .	289
4.6.1	An Explicit Approximation . . . . .	289
4.6.2	An Implicit Approximation . . . . .	293
4.6.3	General Wave Type Problems . . . . .	295
4.7	Nonlinear Problems . . . . .	296
4.7.1	Partial Substitution or Picard Iteration . . . . .	296
4.7.2	Newton’s Method . . . . .	297
	Bibliography . . . . .	302
<b>5</b>	<b>Calculus of Variations, Elasticity and Tensors</b>	<b>304</b>
5.1	Prerequisites and Goals . . . . .	304
5.2	Calculus of Variations . . . . .	304
5.2.1	The Euler Lagrange Equation . . . . .	304
5.2.2	Quadratic Functionals and Second Order Linear Boundary Value Problems . . . . .	311
5.2.3	The Divergence Theorem and its Consequences . . . . .	312
5.2.4	Quadratic Functionals and Second Order Boundary Value Problems in 2 Dimensions	314
5.2.5	Nonlinear Problems and Euler–Lagrange Equations for Systems . . . . .	317
5.2.6	Hamilton’s principle of Least Action . . . . .	319
5.3	Basic Elasticity, Description of Stress and Strain . . . . .	325
5.3.1	Description of Strain . . . . .	327
5.3.2	Description of Stress . . . . .	336
5.3.3	Invariant Stress Expressions, Von Mises Stress and Tresca Stress . . . . .	340
5.4	Elastic Failure Modes . . . . .	342
5.4.1	Maximum Principal Stress Theory . . . . .	343
5.4.2	Maximum Shear Stress Theory . . . . .	343
5.4.3	Maximum Distortion Energy . . . . .	344
5.5	Scalars, Vectors and Tensors . . . . .	344
5.5.1	Change of Coordinate Systems . . . . .	344
5.5.2	Zero Order Tensors: Scalars . . . . .	344
5.5.3	First Order Tensors: Vectors . . . . .	345
5.5.4	Second Order Tensors: some Matrices . . . . .	345
5.6	Hooke’s Law and Elastic Energy Density . . . . .	348
5.6.1	Hooke’s Law . . . . .	348
5.6.2	Hooke’s law for Incompressible Materials . . . . .	349
5.6.3	Elastic Energy Density . . . . .	349
5.6.4	Volume and Surface Forces, the Bernoulli Principle . . . . .	354
5.6.5	Some Exemplary Situations . . . . .	355
5.7	More on Tensors and Energy Densities for Nonlinear Material Laws . . . . .	364
5.7.1	A few More Tensors . . . . .	364
5.7.2	Neo–Hookean Energy Density Models . . . . .	369
5.7.3	Ogden and Mooney–Rivlin Energy Density Models . . . . .	378
5.7.4	References used in the Above Section . . . . .	387
5.8	Plane Strain . . . . .	388
5.8.1	Description of Plane Strain and Plane Stress . . . . .	388
5.8.2	From the Minimization Formulation to a System of PDE’s . . . . .	391
5.8.3	Boundary Conditions . . . . .	393
5.9	Plane Stress . . . . .	394

5.9.1	From the Plane Stress Matrix to the Full Stress Matrix . . . . .	395
5.9.2	From the Minimization Formulation to a System of PDE's . . . . .	396
5.9.3	Boundary Conditions . . . . .	397
5.9.4	Deriving the Differential Equations using the Euler–Lagrange Equation . . . . .	398
Bibliography . . . . .		400
<b>6</b>	<b>Finite Element Methods</b>	<b>402</b>
6.1	From Minimization to the Finite Element Method . . . . .	402
6.2	Piecewise Linear Finite Elements . . . . .	404
6.2.1	Discretization, Approximation and Assembly of the Global Stiffness Matrix . . . . .	404
6.2.2	Integration over one Triangle . . . . .	406
6.2.3	Integration of $\nabla u \cdot \nabla u$ over one Triangle . . . . .	406
6.2.4	The Element Stiffness Matrix . . . . .	407
6.2.5	Triangularization of the Domain $\Omega \subset \mathbb{R}^2$ . . . . .	408
6.2.6	Assembly of the System of Linear Equations . . . . .	408
6.2.7	The Algorithm of Cuthill and McKee to Reduce Bandwidth . . . . .	410
6.2.8	A First Solution by the FEM . . . . .	412
6.2.9	Contributions to the Approximation Error . . . . .	415
6.3	Classical and Weak Solutions . . . . .	416
6.3.1	Weak Solutions of a System of Linear Equations . . . . .	416
6.3.2	Classical Solutions and Weak Solutions of Differential Equations . . . . .	417
6.4	Energy Norms and Error Estimates . . . . .	418
6.4.1	Basic Assumptions and Regularity Results . . . . .	419
6.4.2	Function Spaces, Norms and Continuous Functionals . . . . .	419
6.4.3	Convergence of the Finite Dimensional Approximations . . . . .	421
6.4.4	Approximation by Piecewise Linear Interpolation . . . . .	424
6.4.5	Approximation by Piecewise Quadratic Interpolation . . . . .	426
6.5	Construction of Triangular Second Order Elements . . . . .	428
6.5.1	Integration over a General Triangle . . . . .	429
6.5.2	The Basis Functions for a Second Order Element . . . . .	433
6.5.3	Integration of Functions Given at the Nodes . . . . .	435
6.5.4	Integrals to be Computed . . . . .	436
6.5.5	The Octave code <code>ElementContribution.m</code> . . . . .	436
6.5.6	Integration of $f \phi$ over one Triangle . . . . .	437
6.5.7	Integration of $b_0 u \phi$ over one Triangle . . . . .	438
6.5.8	Transformation of the Gradient to the Standard Triangle . . . . .	438
6.5.9	Integration of $u \vec{b} \cdot \nabla \phi$ over one Triangle . . . . .	442
6.5.10	Integration of $a \nabla u \cdot \nabla \phi$ over one Triangle . . . . .	443
6.5.11	Construction of the Element Stiffness Matrix . . . . .	445
6.6	Comparing First and Second Order Triangular Elements . . . . .	445
6.6.1	Observe the Convergence of the Error as $h \rightarrow 0$ . . . . .	445
6.6.2	Estimate the Number of Nodes and Triangles and the Effect on the Sparse Matrix . . . . .	446
6.6.3	Behavior of a FEM Solution within Triangular Elements . . . . .	447
6.6.4	Remaining Pieces for a Complete FEM Algorithm, the <code>FEMoctave</code> Package . . . . .	449
6.7	Applying the FEM to Other Types of Problems, e.g. Plane Elasticity . . . . .	450
6.8	Using Quadrilateral Elements . . . . .	450
6.8.1	First Order Quadrilateral Elements . . . . .	451
6.8.2	Second Order Quadrilateral Elements . . . . .	456
6.9	An Application of FEM to a Tumor Growth Model . . . . .	458
6.9.1	Introduction . . . . .	458
6.9.2	The Finite Element Method Applied to 1D Problems . . . . .	459

6.9.3 A Few Examples, Static and Dynamic . . . . .	464
6.9.4 Solving the Dynamic Tumor Growth Problem . . . . .	468
Bibliography . . . . .	475
<b>Bibliography for all Chapters</b>	<b>476</b>
<b>List of Figures</b>	<b>484</b>
<b>List of Tables</b>	<b>486</b>
<b>Index</b>	<b>487</b>

# Chapter 0

## Introduction

### 0.1 Using these Lecture Notes

These lecture notes<sup>1</sup> serve as support for the lectures. The students shall not be forced to copy many results and formulas from blackboard, beamer or projector. The notes will not replace attending the lectures, but the combination of lectures and notes should provide all necessary information in a digestible form.

In an ideal world a student will read through the lecture notes **before** the topic is presented in class. This allows the student to take full advantage of the presentation in the lectures. In class more weight is put on the **ideas** for the algorithms and methods, while the notes spell out the tedious details too. In the real world it is more likely that a student is using the notes in class. It is a good idea to supplement the notes in class with your personal notes. It should not be necessary to buy additional books for this class. A collection of additional references and a short description of the content is given in Section 0.3.

### 0.2 Content and Goals of this Class

In this class we will present the necessary background to choose and use numerical algorithms to solve problems arising in biomedical engineering applications. Obviously we have to choose a small subset of all possibly useful topics to be considered.

- Some algorithms to examine large systems of linear equations are examined.
- Some basic numerical tools are presented.
  - Basic methods to solve nonlinear equations.
  - The basic ideas and tools of numerical integration of functions given by data points or formulas.
  - The ideas and tools to generate numerical solutions of ordinary differential equations.
  - The ideas and tools to use linear and nonliner regression.
- The method of finite differences (FD) is presented and applied to a few typical problems.
- The necessary tensor calculus for the description of elasticity problems is presented.
- The principal of virtual work is applied to mechanical problems and the importance of the calculus of variations is illustrated. This will serve as basis for the Finite Element Method, i.e. FEM .
- The method of finite elements is introduced using 2D model problems. The convergence of triangular elements of order 1 and 2 is examined carefully.

The topics to be examined in this class are shown in Figure 1.

---

<sup>1</sup>Lecture notes are different from books. A book should be complete, while the combination of lectures and lecture notes should be complete.

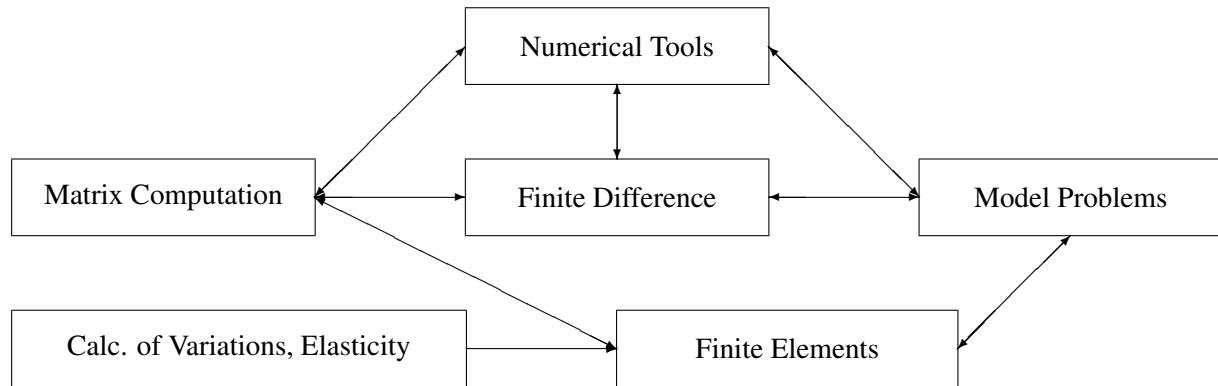


Figure 1: Structure of the topics examined in this class

The main goals of this class are:

- Get to know a few basic algorithms and techniques to solve modeling problems.
- Learn a few techniques to examine performance of numerical algorithm.
- Examine reliability of results of numerical computations.
- Examine speed and memory requirements of some algorithms.

### **The Purpose of Computing is Insight, not Numbers<sup>2</sup>**

Obviously there are many important topics numerical methods that we do not consider in this class. The methods presented should help you to read and understand the corresponding literature.

To realize that reliable numerical methods can be very important, examine a list of disasters, caused by numerical errors, see <http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html>. Among the spectacular events are Patriot missile failures, an Ariane rocket blowup and the sinking of the Sleipner offshore platform.

## 0.3 Literature

Below find a selection of books on numerical analysis and the Finite Element Method. The list is not an attempt to generate a representative selection, but is strongly influenced by my personal preference and bookshelf. The list of covered topics might help when you face problems not solvable with the methods and ideas presented in this class.

### 0.3.1 A Selection of Literature on Numerical Methods

- [Stah08] A. Stahel, Numerical Methods: the lectures notes for this class should be sufficient to follow the class. You might want to consult other books, either to catch up on prerequisite knowledge or to find other approaches to topics presented in class.
- [Schw09] H.R. Schwarz, Numerische Mathematik: this is a good introduction to numerical mathematics and might serve well as a complement to the lecture notes.
- [IsaaKell66] Isaacson and Keller, Analysis of Numerical Methods: this is an excellent (and very affordable) introduction to numerical analysis. It is mathematically very solid and I strongly recommend it as a supplement.

<sup>2</sup>Richard Hamming, 1962

- [DahmReus07] Dahmen and Reusken, Numerik für Ingenieure und Naturwissenschaftler: this is a comprehensive presentation of most of the important numerical algorithms.
- [GoluVanLoan96] Gene Golub, Charles Van Loan, Matrix Computations: this is the bible for matrix computations (see Chapter 2) and an excellent book. Use this as reference for matrix computations. There is a new, expanded edition of this marvelous book [GoluVanLoan13].
- [Smit84] G. D. Smith, Numerical Solution of Partial Differential Equations: Finite Difference Methods: this is a basic introduction to the method of finite differences.
- [Thom95] J. W. Thomas, Numerical Partial Differential Equations: Finite Difference Methods: this is an excellent up-to-date presentation of finite difference methods. Use this book if you want to go beyond the presentation in class.
- [Acto90] F. S. Acton, Numerical Methods that Work: a well written book on many basic aspect of numerical methods. Common sense advise is given out freely. Well worth reading.
- [Pres92] Press et al., Numerical Recipes in C: this is a collection of basic algorithms and some explanation of the effects and aspects to consider. There are versions of this book for the programming languages C, C++, Fortran, Pascal, Modula and Basic.
- [Knor08] M. Knorrer, Numerische Mathematik: a very short and affordable collection of results and examples. No proofs are given, just the facts stated. It might be useful when reviewing the topics and preparing an exam.
- [GhabWu16] J. Ghaboussi and X.S. Wu, Numerical Methods in Computational Mechanics: a selection of numerical results and definitions, useful for computational mechanics.

Find a list of these references and the covered topics in Table 1.

### 0.3.2 A Selection of Literature on the Finite Element Method

- [Zien13] Zienkiewicz, Taylor, and Zhu. The Finite Element Method: Its Basis and Fundamentals. A very good introduction to FEM. Includes the application to linear elasticity.
- [John87] Claes Johnson, Numerical Solution of Partial Differential Equations by the Finite Element Method: this is an excellent introduction to FEM, readable and mathematically precise. It might serve well as a supplement to the lecture notes.
- [TongRoss08] P. Tong and J. Rossettos: Finite Element Method, Basic Technique and Implementation. The book title says it all. Implementation details are carefully presented. It contains a good presentation of the necessary elasticity concepts. Since this book is available a Dover edition, the cost is minimal.
- [Schw88] H. R. Schwarz, Finite Element Method: An easily understandable book, presenting the basic algorithms and tools to use FEM.
- [Hugh87] Thomas J. R. Hughes, The Finite Element Method: Linear Static and Dynamic Finite Element Analysis: this classic book on FEM contains considerable information for elasticity problems. The presentation discusses many implementation details, also for shells, beams, plates and curved elements. It is very affordable.
- [Brae02] Dietrich Braess, Finite Elemente: this is a modern presentation of the mathematical theory for FEM and their implementation. Mathematically it is more advanced and preciser than these lecture notes. Also find the exact formulation of the Bramble-Hilbert Lemma. This book is recommended for further studies.

Reference	[Stah08]	[IsaaKell66]	[Schw09]	[DahmReus07]	[GoluVanLoan96]	[Smiti84]	[Thom95]	[Acto90]	[Knor08]	[GhabWu16]
Floating point arithmetic	x	x	x	x	x				x	x
CPU, memory and cache structure	x									
Gauss algorithm	o	x	x	x	x				x	x
LR factorization	x	x	x	x	x				x	x
Cholesky algorithm	x	x	x	x	x				x	x
Banded Cholesky	x		x	x	x					
Stability of linear algorithms	x	x	o		x					x
Iterative methods for linear systems	o	x			x	x			x	x
Conjugate gradient method	x				x					x
Preconditioners	o				x					x
Solving a single nonlinear equation	x	x	x	x					x	x
Newton's algorithm for systems	x	x	x	x					x	x
FD approximation	x	x	x	x		x	x		x	
Consistency, stability, convergence, Lax theorem	x	x	x	x		x	x			
Neumann stability analysis		x				x	x			
Boundary value problems 1D	x	x	x	x		x	x			
Boundary value problems 2D	x	x	x	x		x	x	x	x	
Stability of static problems	x	x		x		x				
Explicit, implicit, Crank-Nicolson for heat equation	x	x	x			x	x			
Explicit, implicit for wave equation	x	x				x	x			
Nonlinear FD problems	x					x				
Numerical integration, 1D		x	x	x				x	x	
Gauss integration 1D	o	x	x	x					x	
Gauss integration 2D	o	x	x	x						
ODE, initial value problems		x	x	x				x	x	x
Zeros of polynomials		x	x	x						
Interpolation		x	x	x				x		
Eigenvalues and eigenvectors		x	x	x	x			x		x
Linear regression	o		x	x	x				x	

Table 1: Literature on Numerical Methods. An extensive coverage of the topic is marked by x, while a brief coverage is marked by o.

- [Gmür00] Thomas Gmür, Méthodes des éléments finis en mécanique des structures: An introduction (in french) for FEM applied to elasticity.
- [ZienMorg06] O. C. Zienkiewicz and K. Morgan, Finite Elements and Approximation: A solid presentation of the FEM is given, preceded by a short review of the finite difference method.
- [AtkiHan09] K. Atkinson and W. Han, Theoretical Numerical Analysis: a good functional analysis framework for numerical analysis is presented in this book. This is a rather theoretical, well written book. It also shows the connection of partial differential equations and numerical methods. An excellent presentation of the FEM is given.
- [Ciar02] Philippe Ciarlet, The Finite Element Method for Elliptic Problems: Here you find a mathematical presentation of the error estimates for the FEM, including all details. The Bramble-Hilbert Lemma is carefully examined. This is a very solid mathematical foundation.
- [Prze68] J.S. Przemieniecki, Theory of Matrix Structural Analysis: this is a classical presentation of the mechanical approach to FEM. A good introduction of the keywords stress, strain and Hooke law is shown.
- [Shab08] Elasticity to FEM, plasticity
- [Koko15] Jonas Koko, Approximation numérique avec MATLAB, Programmation vectorisée, équations aux dérivées partielles. A nice introduction (in french) to MATLAB and the coding of FEM algorithms. The details are worked out. Some code for the linear elasticity problem is developed.
- [ShamDym95] Shames and Dym, Energy and Finite Element Methods in Structural Mechanics. A solid introduction to variational methods, applied to structural mechanics. Contains a good description of mechanics and FEM.
- [Froc16] Jörg Frochte, Finite–Elemente–Methode. Eine praxisbezogene Einführung mit GNU Octave/Matlab. A very readable introduction with complete codes for MATLAB/Octave. Contains a presentation of dynamic problems.
- [Li21] Gang Li, Introduction to the Finite Element Method and Implementation with MATLAB. Very detailed presentation, including documented code.

Find a list of references for the FEM and the covered topics in Table 2.

## Bibliography

- [Acto90] F. S. Acton. *Numerical Methods that Work; 1990 corrected edition*. Mathematical Association of America, Washington, 1990.
- [AtkiHan09] K. Atkinson and W. Han. *Theoretical Numerical Analysis*. Number 39 in Texts in Applied Mathematics. Springer, 2009.
- [Brae02] D. Braess. *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer, second edition, 2002.
- [Ciar02] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. SIAM, 2002.
- [DahmReus07] W. Dahmen and A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. Springer, 2007.
- [Froc16] J. Frochte. *Finite-Elemente-Methode: Eine praxisbezogene Einführung mit GNU Octave/MATLAB*. Hanser Fachbuchverlag, 2016. Octave/Matlab code available.

Reference	Intro to Calculus of Variations	Finite element method, linear 2D	Generation of stiffness matrix	Error estimates, Lemma of Cea	Second order elements in 2D	Gauss integration	Elasticity problems
[Stah08]	x	x	x	x	x	x	x
[Zien13]	x	x	x	x	x	x	x
[John87]	x	x	x	x	x	x	x
[TongRoss08]	x	x	x	x	x	x	x
[Schw88]	x	x	x	x	x	x	x
[Hugh87]	x	x	x	x	x	x	x
[Brae02]	x	x	x	x	x	x	x
[Gmur00]	x	x	x	x	x	x	x
[ZienMorg06]	x	x	x	x	x	x	x
[AtkiHan09]	x	x	x	x	x	x	x
[Ciar02]	x	x	x	x	x	x	x
[Prze68]	x	x	x	x	x	x	x
[Koko15]	x	x	x	x	x	x	x
[ShamDym95]	x	x	x	x	x	x	x
[Froc16]	x	x	x	x	x	x	x
[Li21]	x	x	x	x	x	x	x

Table 2: Literature on the Finite Element Method

- [GhabWu16] J. Ghaboussi and X. Wu. *Numerical Methods in Computational Mechanics*. CRC Press, 2016.
- [Gmür00] T. Gmür. *Méthode des éléments finis en mécanique des structures*. Mécanique (Lausanne). Presses polytechniques et universitaires romandes, 2000.
- [GoluVanLoan96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.
- [GoluVanLoan13] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2013.
- [Hugh87] T. J. R. Hughes. *The Finite Element Method, Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, 1987. Reprinted by Dover.
- [IsaaKell66] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. John Wiley & Sons, 1966. Republished by Dover in 1994.
- [John87] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, 1987. Republished by Dover.
- [Knor08] M. Knorrenschild. *Numerische Mathematik*. Carl Hanser Verlag, 2008.
- [Koko15] J. Koko. *Approximation numérique avec Matlab, Programmation vectorisée, équations aux dérivées partielles*. Ellipses, Paris, 2015.
- [Li21] G. Li. *Introduction to the Finite Element Method and Implementation with MATLAB*. Cambridge University Press, 2021.
- [Pres92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.
- [Prze68] J. Przemieniecki. *Theory of Matrix Structural Analysis*. McGraw-Hill, 1968. Republished by Dover in 1985.
- [Schw88] H. R. Schwarz. *Finite Element Method*. Academic Press, 1988.
- [Schw09] H. R. Schwarz. *Numerische Mathematik*. Teubner und Vieweg, 7. edition, 2009.
- [Shab08] A. A. Shabana. *Computational Continuum Mechanics*. Cambridge University Press, 2008.
- [ShamDym95] I. Shames and C. Dym. *Energy and Finite Element Methods in Structural Mechanics*. New Age International Publishers Limited, 1995.
- [Smit84] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, Oxford, third edition, 1986.
- [Stahel08] A. Stahel. Numerical Methods. lecture notes, BFH-TI, 2008.
- [Thom95] J. W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*, volume 22 of *Texts in Applied Mathematics*. Springer Verlag, New York, 1995.
- [TongRoss08] P. Tong and J. Rossettos. *Finite Element Method, Basic Technique and Implementation*. MIT, 1977. Republished by Dover in 2008.
- [Zien13] O. Zienkiewicz, R. Taylor, and J. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 7 edition, 2013.
- [ZienMorg06] O. C. Zienkiewicz and K. Morgan. *Finite Elements and Approximation*. John Wiley & Sons, 1983. Republished by Dover in 2006.

# Chapter 1

## The Model Problems

The main goal of the introductory chapter is to familiarize you with a small set of sample problems. These problems will be used throughout the class and in the notes to illustrate the presented methods and algorithms. Each of the selected model problems stands for a class of similar application problems.

### 1.1 Heat Equations

#### 1.1.1 Description

The **heat capacity**  $c$  of a material gives the amount of energy needed to raise the temperature  $T$  of one kilogram of the material by one degree K (Kelvin). The **thermal conductivity**  $k$  of a material indicates the amount of energy transmitted through a plate with thickness 1 m and  $1 \text{ m}^2$  area if the temperatures at the two sides differ by 1 K. In Table 1.1 find values for  $c$  and  $k$  for some typical materials. For homogeneous materials the values of  $c$  and  $k$  will not depend on the location  $\vec{x}$ . For most materials the values depend on the temperature  $T$ , but we will not consider this case (yet). The resulting equations would be nonlinear.

	heat capacity at $20^\circ C$	heat conductivity
symbol	$c$	$k$
unit	$\frac{\text{kJ}}{\text{kg K}}$	$\frac{\text{W}}{\text{m K}}$
iron	0.452	74
steel	0.42 - 0.51	45
copper	0.383	384
water	4.182	0.598

Table 1.1: Some values of heat related constants

The **flux of thermal energy** is a vector indicating the direction of the flow and the amount of thermal energy flowing per second and square meter. **Fourier's law** of heat conduction can be stated in the form

$$\vec{q} = -k \nabla T. \quad (1.1)$$

This basic law of physics indicates that the thermal energy will flow from hot spots to areas with lower temperature. For some simple situations the consequences of this equation will be examined. The only other basic physical principle to be used is **conservation of energy**. Some of the variables and symbols used in this section are shown in Table 1.2.

	symbol	unit
density of energy	$u$	$\frac{\text{J}}{\text{m}^3}$
temperature	$T$	K
heat capacity	$c$	$\frac{\text{J}}{\text{K kg}}$
density	$\rho$	$\frac{\text{kg}}{\text{m}^3}$
heat conductivity	$k$	$\frac{\text{J}}{\text{s m K}}$
heat flux	$\vec{q}$	$\frac{\text{J}}{\text{s m}^2}$
external energy source density	$f$	$\frac{\text{J}}{\text{s m}^3}$
area of the cross section	$A$	$\text{m}^2$

Table 1.2: Symbols and variables for heat conduction

### 1.1.2 The One Dimensional Heat Equation

If a temperature  $T$  over a solid (with constant cross section  $A$ ) is known to depend on one coordinate  $x$  only, then the change of temperature  $\Delta T$  measured over a distance  $\Delta x$  will lead to a flow of thermal energy  $\Delta Q$ . If the time difference is  $\Delta t$  then (1.1) reads as

$$\frac{\Delta Q}{\Delta t} = -k A \frac{\Delta T}{\Delta x}.$$

This is Fourier's law and it leads to a heat flux in direction  $x$  of

$$q = -k \frac{\partial T}{\partial x}.$$

Consider the temperature  $T$  as dependent variable find on the interval  $a \leq x \leq b$  the thermal energy

$$E(t) = \int_a^b A u(t, x) dx = \int_a^b A \rho c T(t, x) dx.$$

Now compute the rate of change of energy in the same interval. The rate of change has to equal the total flux of energy into this interval plus the input from external sources

$$\begin{aligned} \text{total change} &= \text{input through boundary} + \text{external sources}, \\ \frac{\partial E(t)}{\partial t} &= \left( -k A(a) \frac{\partial T(t,a)}{\partial x} + k A(b) \frac{\partial T(t,b)}{\partial x} \right) + \int_a^b A(x) f(t, x) dx, \\ \int_a^b A(x) \rho c \frac{\partial T(t,x)}{\partial t} dx &= \int_a^b \frac{\partial}{\partial x} \left( k A(x) \frac{\partial T(t,x)}{\partial x} \right) dx + \int_a^b A(x) f(t, x) dx. \end{aligned}$$

At this point use the conservation of energy principle. Since the above equation has to be correct for all possible values of  $a$  and  $b$  the expressions under the integrals have to be equal and we obtain the general equation for heat conduction in one variable

$$A(x) \rho c \frac{\partial T(t, x)}{\partial t} = \frac{\partial}{\partial x} \left( k A(x) \frac{\partial T(t, x)}{\partial x} \right) + A(x) f(t, x).$$

### 1.1.3 The Steady State Problem

If the steady state situation has to be examined then the temperature  $T$  can not depend on  $t$  and thus

$$-\frac{\partial}{\partial x} \left( k A(x) \frac{\partial T(x)}{\partial x} \right) = A(x) f(x).$$

This second order differential equation has to be supplemented by boundary conditions, either prescribed temperature or prescribed energy flux.

As a standard example consider a beam with constant cross section of length 1 with known temperature  $T = 0$  at the two ends and a known heating contribution  $f(x)$ . This leads to

$$-\frac{d^2}{dx^2} T(x) = \frac{1}{k} f(x) \quad \text{for } 0 \leq x \leq 1 \quad \text{and} \quad T(0) = T(1) = 0. \quad (1.2)$$

### 1.1.4 The Dynamic Problem, Separation of Variables

As standard example consider a beam with constant cross section  $A$  of length 1 with prescribed temperature at both ends for all times and a known temperature profile  $T(0, x) = T_0(x)$  at time  $t = 0$ . This leads to the partial differential equation

$$\begin{aligned} \frac{\rho c}{k} \frac{\partial}{\partial t} T(t, x) - \frac{\partial^2}{\partial x^2} T(t, x) &= \frac{1}{k} f(x) && \text{for } 0 \leq x \leq 1 \quad \text{and} \quad t \geq 0 \\ T(t, 0) = T(t, 1) &= 0 && \text{for } t \geq 0 \\ T(0, x) &= T_0(x) && \text{for } 0 \leq x \leq 1. \end{aligned} \quad (1.3)$$

For the special case  $f(x) = 0$  use the method of **separation of variables** to find a solution to this problem. With  $\alpha = \frac{k}{\rho c}$  the above simplifies to

$$\begin{aligned} \frac{\partial}{\partial t} T(t, x) &= \alpha \frac{\partial^2}{\partial x^2} T(t, x) && \text{for } 0 \leq x \leq 1 \quad \text{and} \quad t \geq 0 \\ T(t, 0) = T(t, 1) &= 0 && \text{for } t \geq 0 \\ T(0, x) &= T_0(x) && \text{for } 0 \leq x \leq 1. \end{aligned}$$

- Look for solutions of the form  $T(t, x) = h(t) \cdot u(x)$ , i.e. a product of functions of one variable each. Plugging this into the partial differential equation leads to

$$\begin{aligned} \frac{\partial}{\partial t} T(t, x) &= \alpha \frac{\partial^2}{\partial x^2} T(t, x) \\ h(t) \cdot u(x) &= \alpha h(t) \cdot u''(x) \\ \frac{1}{\alpha} \frac{\dot{h}(t)}{h(t)} &= \frac{u''(x)}{u(x)}. \end{aligned}$$

Since the left hand side depends on  $t$  only and the right hand side on  $x$  only, both have to be constant. One can verify that the constant has to be negative, e.g.  $-\lambda^2$ .

- Using the right hand side leads to the boundary value problem

$$u''(x) = -\lambda^2 u(x) \quad \text{with} \quad u(0) = u(1) = 0.$$

Use the boundary conditions  $T(t, 0) = T(t, 1) = 0$  to verify that this problem has nonzero solutions only for the values  $\lambda_n = n \pi$  and the solutions are given by

$$u_n(x) = \sin(x n \pi).$$

- The resulting differential equation for  $h(t)$  is

$$\frac{\partial}{\partial t} h(t) = -\alpha \lambda_n^2 h(t)$$

with the solutions  $h_n(t) = h_n(0) \exp(-\alpha \lambda_n^2 t)$ .

- By combining the above find solutions

$$T_n(t, x) = h_n(t) \cdot u_n(x) = h_n(0) \sin(x n \pi) \exp(-\alpha \lambda_n^2 t).$$

- This solution satisfies the initial condition  $T(0, x) = h_n(0) \sin(x n \pi)$ . To satisfy arbitrary initial conditions  $T(0, x) = T_0(x)$  use the superposition principle and Fourier Sin series

$$T_0(x) = \sum_{n=1}^{\infty} b_n \sin(n \pi x)$$

with the Fourier coefficients

$$b_n = 2 \int_0^1 T_0(x) \sin(n \pi x) dx.$$

Combining the above find the solution

$$T(t, x) = \sum_{n=1}^{\infty} b_n \sin(x n \pi) \exp(-\alpha \lambda_n^2 t).$$

Use this explicit formula to verify the accuracy of the numerical approximations to be examined.

### 1.1.5 The Two Dimensional Heat Equation

If the domain  $G \subset \mathbb{R}^2$  with boundary curve  $C$  describes a thin plate with constant thickness  $h$  then we may assume that the temperature will depend on  $t, x$  and  $y$  only and not on  $z$ . The total energy stored in that domain is given by

$$E(t) = \iint_G h u dA = \iint_G h c \rho T(t, x, y) dA.$$

Again examine the rate of change of energy  $\frac{d}{dt} E$  and arrive at

$$\frac{\partial}{\partial t} E = \iint_G h c \rho \frac{\partial T}{\partial t} dA = - \oint_C h \vec{q} \cdot \vec{n} ds + \iint_G h f dA.$$

Using the divergence theorem on the second integral and Fourier's law find

$$\begin{aligned} \iint_G h c \rho \frac{\partial T}{\partial t} dA &= - \iint_G \operatorname{div}(h \vec{q}) dA + \iint_G h f dA \\ &= \iint_G \operatorname{div}(k h \nabla T) dA + \iint_G h f dA \\ &= \iint_G \operatorname{div}(k h \nabla T(t, x, y)) dA + \iint_G h f dA. \end{aligned}$$

This equation has to be correct for all possible domains  $G$  and not only for the physical domain. Thus the expressions under the integral have to be equal and thus find the partial differential equation.

$$h c \rho \frac{\partial T}{\partial t} = \operatorname{div}(k h \nabla T(t, x, y)) + h f. \quad (1.4)$$

If  $\rho, c, h$  and  $k$  are constant then find in cartesian coordinates

$$c \rho \frac{\partial}{\partial t} T(t, x, y) = k \left( \frac{\partial^2}{\partial x^2} T(t, x, y) + \frac{\partial^2}{\partial y^2} T(t, x, y) \right) + f(t, x, y)$$

or shorter

$$c \rho \frac{\partial}{\partial t} T(t, x, y) = k \Delta T(t, x, y) + f(t, x, y),$$

where  $\Delta$  is the well known **Laplace operator**

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}.$$

The heat equation is a second order partial differential equation with respect to the space variable  $x$  and  $y$  and of first order with respect to time  $t$ .

### 1.1.6 The Steady State 2D–Problem

If the steady state has to be examined then the temperature  $T$  can not depend on  $t$  and then use the unit square  $0 \leq x, y \leq 1$  as the standard domain. Then the problem simplifies to

$$\begin{aligned} -\Delta T(x, y) &= \frac{1}{k} f(x, y) && \text{for } 0 \leq x, y \leq 1 \\ T(x, y) &= 0 && \text{for } (x, y) \text{ on boundary} . \end{aligned} \quad (1.5)$$

Find a possible solution of the above equation on an L-shaped domain in Figure 1.1. On one part of the boundary the temperature  $T(x, y) = 0$  is prescribed, but on the section on the right in Figure 1.1 the condition is thermal insulation, i.e. vanishing normal derivative  $\frac{\partial T}{\partial n} = 0$ .

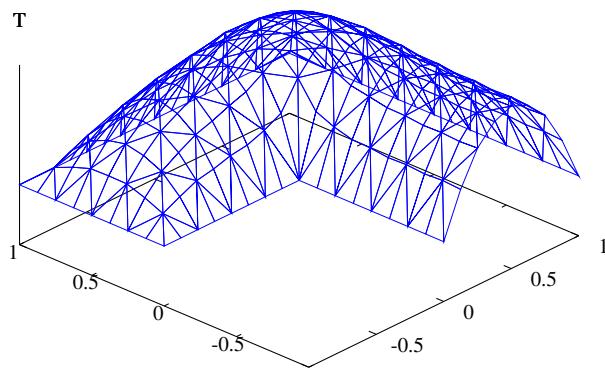


Figure 1.1: Temperature  $T$  as function of the horizontal position

### 1.1.7 The Dynamic 2D–Problem

If in the above problem the temperature  $T$  depends on time  $t$  too we have to solve the dynamic problem, supplemented with appropriate initial and boundary values. Our example problem turns into

$$\begin{aligned} \frac{\rho c}{k} \frac{\partial}{\partial t} T(t, x, y) - \Delta T(t, x, y) &= \frac{1}{k} f(t, x, y) && \text{for } 0 \leq x, y \leq 1 \text{ and } t \geq 0 \\ T(t, x, y) &= 0 && \text{for } (x, y) \text{ on boundary and } t \geq 0 \\ T(0, x, y) &= T_0(x, y) && \text{for } 0 \leq x, y \leq 1 . \end{aligned} \quad (1.6)$$

Figure 1.1 might represent the solution  $T(t, x, y)$  at a given time  $t$ .

## 1.2 Vertical Deformations of Strings and Membranes

### 1.2.1 Equation for a Steady State Deformation of a String

In Figure 1.2 a segment of a string is shown. If the segment of horizontal length  $\Delta x$  is to be at rest the sum of all the forces applied to the segment has to vanish, i.e.

$$\vec{T}(x + \Delta x) - \vec{T}(x) = -\vec{F}.$$

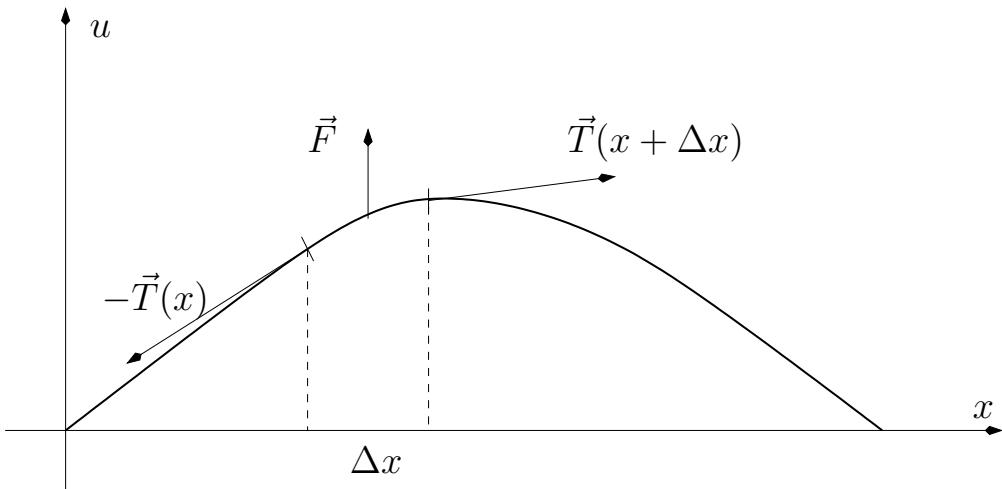


Figure 1.2: Segment of a string

Assume

- the vertical displacement of the string is given by a function  $y = u(x)$ .
- the slopes of the string are small, i.e.  $|u'(x)| \ll 1$
- the horizontal component of the tension  $\vec{T}$  is constant and equals  $T_0$ .
- the vertical external force is given by a force density function  $f(x)$

Thus the external vertical force to the segment is given by

$$F = \int_x^{x+\Delta x} f(s) ds \approx f(x) \Delta x.$$

For a string the tension vector  $\vec{T}$  is required to have the same slope as the string. Using the condition of small slope arrive at

$$\begin{aligned} T_2(x) &= u'(x) T_0 \\ T_2(x + \Delta x) &= u'(x + \Delta x) T_0 \\ T_2(x + \Delta x) - T_2(x) &= (u'(x + \Delta x) - u'(x)) T_0 \approx T_0 u''(x) \Delta x. \end{aligned}$$

By combining the above approximations and dividing by  $\Delta x$  leads to the differential equation

$$-T_0 u''(x) = f(x). \quad (1.7)$$

Supplement this with the boundary conditions of a fixed string (i.e.  $u(0) = u(L) = 0$ ) to obtain a second order boundary value problem (BVP). Using Calculus of Variations (see Section 5.2) observe that solving this BVP is equivalent to finding the minimizer of the functional

$$F(u) = \int_0^L \frac{1}{2} T_0 (u'(x))^2 - f(x) \cdot u(x) dx$$

amongst ‘all’ functions  $u$  with  $u(0) = u(L) = 0$ .

### 1.2.2 Equation for a Vibrating String

If the sum of the vertical forces in the previous section is not equal to zero, then the string will accelerate. The necessary force equals  $M \ddot{u} \approx \rho \ddot{u} \Delta x$ , where  $\rho$  is the specific mass measured in mass per length. With the boundary conditions at the two ends arrive at the initial boundary value problem (IBVP).

$$\begin{aligned} \rho(x) \ddot{u}(t, x) &= T_0 u''(t, x) + f(t, x) && \text{for } 0 < x < L \quad \text{and} \quad t > 0 \\ u(t, 0) = u(t, L) &= 0 && \text{for } 0 \leq t \\ u(0, x) &= u_0(x) && \text{for } 0 < x < L \\ \dot{u}(0, x) &= u_1(x) && \text{for } 0 < x < L. \end{aligned} \tag{1.8}$$

### 1.2.3 The Eigenvalue Problem

For an eigenvalue  $\lambda$  and an eigenfunction of the problem

$$-T_0 y''(x) = \lambda \rho(x) y(x) \quad \text{with} \quad y(0) = y(L) = 0, \tag{1.9}$$

the function

$$u(t, x) = A \cos(\sqrt{\lambda} t + \phi) \cdot y(x)$$

is a solution of the equation of the vibrating string with no external force, i.e.  $f(t, x) = 0$ . Thus the eigenvalues  $\lambda$  lead to vibrations with the frequencies  $\nu = \sqrt{\lambda}/(2\pi)$ .

### 1.2.4 Equation for a Vibrating Membrane

The situation of a vibrating membrane is similar to a vibrating string. A careful analysis of the situation (e.g. [Trim90, §1.4]) shows that the resulting PDE is given by

$$\rho \ddot{u} = \frac{\partial}{\partial x} \left( \tau \frac{\partial}{\partial x} u \right) + \frac{\partial}{\partial y} \left( \tau \frac{\partial}{\partial y} u \right) + f,$$

where the interpretation of the terms is shown in Table 1.3. Thus we are lead to the IBVP

	symbol	unit
vertical displacement	$u$	m
external force density	$f$	N/m <sup>2</sup>
horizontal tension	$\tau$	N/m
mass density	$\rho$	kg/m <sup>2</sup>

Table 1.3: Symbols and variables for a vibrating membrane

$$\begin{aligned}
\rho(x, y) \ddot{u}(t, x, y) &= -\nabla(\tau(x, y) \nabla u(t, x, y)) + f(t, x, y) && \text{for } (x, y) \in \Omega \text{ and } t > 0 \\
u(t, x, y) &= 0 && \text{for } (x, y) \in \Gamma \text{ and } t > 0 \\
u(0, x, y) &= u_0(x, y) && \text{for } (x, y) \in \Omega \\
\dot{u}(0, x, y) &= u_1(x, y) && \text{for } (x, y) \in \Omega.
\end{aligned} \tag{1.10}$$

Thus we have a second order PDE, in time and space variables.

If the external force  $f$  vanishes and  $\rho$  and  $\tau$  are constant then find the standard wave equation with velocity  $c = \sqrt{\tau/\rho}$ ,

$$\ddot{u} = \frac{\tau}{\rho} \Delta u = c^2 \Delta u.$$

### 1.2.5 Equation for a Steady State Deformation of a Membrane

The equation governing a steady state membrane are an elementary consequence of the previous section by setting  $\dot{u} = \ddot{u} = 0$ . Thus we find a second order BVP.

$$\begin{aligned}
-\nabla(\tau(x, y) \nabla u(x, y)) &= f(x, y) && \text{for } (x, y) \in \Omega \\
u(x, y) &= 0 && \text{for } (x, y) \in \Gamma.
\end{aligned} \tag{1.11}$$

Using Calculus of Variations (see Section 5.2) we will show that solving this BVP is equivalent to finding the minimizer of the functional

$$F(u) = \iint_{\Omega} \frac{\tau}{2} (\nabla u)^2 - f \cdot u \, dA$$

amongst ‘all’ functions  $u$  with  $u(x) = 0$  on the boundary  $\Gamma$ . Figure 1.1 might represent a solution.

### 1.2.6 Eigenvalue Problem for a Membrane

If  $u(x, y)$  is a solution of the eigenvalue problem

$$\begin{aligned}
\lambda \rho(x, y) u(x, y) &= -\nabla(\tau(x, y) \nabla u(x, y)) && \text{for } (x, y) \in \Omega \\
u(x, y) &= 0 && \text{for } (x, y) \in \Gamma,
\end{aligned} \tag{1.12}$$

then the function

$$A \cos(\sqrt{\lambda} t + \phi) \cdot u(x, y)$$

is a solution of the equation of a vibrating membrane. Thus the eigenvalues  $\lambda$  lead to the frequencies  $\nu = \sqrt{\lambda}/(2\pi)$ . The corresponding eigenfunction  $u(x, y)$  shows the shape of the oscillating membrane.

## 1.3 Horizontal Stretching of a Beam

### 1.3.1 Description

A beam of length  $L$  with known cross sectional area  $A$  (possibly variable) may be stretched horizontally by different methods:

- forces applied to its ends.
- extending its length.
- applying a horizontal force all along the beam.

description	symbol	SI units
horizontal position	$0 \leq x \leq L$	[m]
horizontal displacement	$u$	[m]
strain	$\varepsilon = \frac{du}{dx}$	free of units
force at right end point	$F$	[N]
density of force along beam	$f$	[N/m]
modulus of elasticity	$E$	[N/m <sup>2</sup> ]
Poisson's ratio	$0 \leq \nu < 1/2$	free of units
area of cross section	$A$	[m <sup>2</sup> ]
stress	$\sigma$	[N/m <sup>2</sup> ]

Table 1.4: Variables used for the stretching of a beam

For  $0 \leq x \leq L$  the function  $u(x)$  indicates by how much the part of the horizontal beam originally at position  $x$  will be displaced horizontally, i.e. the new position is given by  $x + u(x)$ . In Table 1.4 find a list of all relevant expressions.

The basic law of Hooke is given by

$$EA \frac{\Delta L}{L} = F$$

for a beam of length  $L$  with cross section  $A$ . A force of  $F$  will stretch the beam by  $\Delta L$ . For a short section of length  $l$  of the beam between  $x$  and  $x + l$  find

$$\frac{\Delta l}{l} = \frac{u(x+l) - u(x)}{l} \rightarrow u'(x) \quad \text{as } l \rightarrow 0+.$$

This expression is called strain and often denoted by  $\varepsilon = \frac{\partial u}{\partial x}$ . Then the force  $F(x)$  at a cross section at  $x$  is given by Hooke's law

$$F(x) = EA(x) u'(x),$$

where  $F(x)$  is the force the section on the right of the cut will apply to the left section. Now examine all forces on the section between  $x$  and  $x + \Delta x$ , whose sum has to be zero for a steady state situation (Newton's law).

$$\begin{aligned} EA(x + \Delta x) u'(x + \Delta x) - EA(x) u'(x) + \int_x^{x+\Delta x} f(s) ds &= 0 \\ \frac{EA(x + \Delta x) u'(x + \Delta x) - EA(x) u'(x)}{\Delta x} &= \frac{1}{\Delta x} \int_x^{x+\Delta x} f(s) ds. \end{aligned}$$

Taking the limit  $\Delta x \rightarrow 0$  in the above expression leads to the second order differential equation for the unknown displacement function  $u(x)$ .

$$-\frac{d}{dx} \left( EA(x) \frac{du(x)}{dx} \right) = f(x) \quad \text{for } 0 < x < L. \quad (1.13)$$

There are different boundary conditions to be considered:

- At the left end point  $x = 0$  we assume no displacement, i.e  $u(0) = 0$ .
- At the right end point  $x = L$  we can examine a given displacement  $u(L) = u_M$ , i.e we have a Dirichlet condition.
- At the right end point  $x = L$  we can examine the situation of a known force  $F$ , leading to the Neumann condition

$$EA(L) \frac{du(L)}{dx} = F.$$

### 1.3.2 Poisson's Ratio, Lateral Contraction

Poisson's ratio  $\nu$  is a material constant and indicates by what factor the lateral directions will contract, when the material is stretched. Thus the resulting cross sectional area will be reduced from its original values  $A_0$ . Assuming  $|\frac{du}{dx}| \ll 1$  the modified area is given by

$$A = A\left(\frac{du}{dx}\right) = A_0 \cdot \left(1 - \nu \frac{du}{dx}\right)^2 \approx A_0 \cdot \left(1 - 2\nu \frac{du}{dx}\right).$$

Since the area is smaller the stress will increase, leading to a further reduction of the area. The resulting stress is the **true stress**, compared to the **engineering stress**, where the force is divided by the original area  $A_0$ , not the true area  $A_0 (1 - \nu \frac{du(x)}{dx})^2$ . The linear boundary value problem (1.13) is replaced by a nonlinear problem.

$$-\frac{d}{dx} \left( EA_0 \left(1 - \nu \frac{du}{dx}\right)^2 \frac{du(x)}{dx} \right) = f(x) \quad \text{for } 0 < x < L. \quad (1.14)$$

For a given force  $F$  at the endpoint at  $x = L$  use the boundary condition

$$EA_0(L) \left(1 - \nu \frac{du(L)}{dx}\right)^2 \frac{du(L)}{dx} = F.$$

Material	Modulus of elasticity $E$ in $10^9$ N/m $^2$	Poisson's ratio $\nu$
Aluminum	73	0.34
Bone (Femur)	17	0.30
Gold	78	0.42
Rubber	3	0.50
Steel	200	0.29
Titanium	110	0.36

Table 1.5: Typical values for the elastic constants

If there is no force along the beam ( $f(x) = 0$ ) the differential equation implies that

$$EA_0 \left(1 - \nu \frac{du}{dx}\right)^2 \frac{du(x)}{dx} = \text{const}.$$

The above equation and boundary condition lead to a cubic equation for the unknown function  $w(x) = u'(x)$ .

$$\begin{aligned} EA_0(x) \left(1 - \nu \frac{du(x)}{dx}\right)^2 \frac{du(x)}{dx} &= F \\ EA_0(x) (1 - \nu w(x))^2 w(x) &= F \\ \nu^2 w^3(x) - 2\nu w^2(x) + w(x) &= \frac{F}{EA_0(x)} \\ \nu^3 w^3(x) - 2\nu^2 w^2(x) + \nu w(x) &= \frac{\nu F}{EA_0(x)}. \end{aligned} \quad (1.15)$$

This is cubic equation for the unknown  $\nu w(x)$ . Thus this example can be solved without using differential equations by solving nonlinear equations, see Example 3–13 on page 122 or by using the method of finite differences, see Example 4–9 on page 296.

### 1.3.3 Nonlinear Stress Strain Relations

If a bar is stretched by small forces  $F$  then the total change of length is proportional to the force, according to Hooke's law

$$\varepsilon = \frac{1}{E} \sigma \quad \text{or} \quad \sigma = E \varepsilon .$$

The strain  $\varepsilon = \frac{\Delta L}{L} = u'$  (relative change of length) is proportional to the stress  $\sigma = \frac{F}{A}$  (force per area). The constant of proportionality is the modulus of elasticity  $E$ . If the force exceeds a certain critical value, then the behavior of the material might change, the material might soften up or even break.

The qualitative behavior can be seen in Figure 1.3. The stress  $\sigma$  is a **nonlinear** function of the strain  $\varepsilon$ . Nonlinear material properties are often important. For the sake of simplicity most examples in these notes use linear material laws, but consider geometric nonlinearities only, e.g. the bending of a beam in Section 1.4. A few remarks on nonlinear behavior caused by large deformations are shown in Section 5.7 starting on page 364.

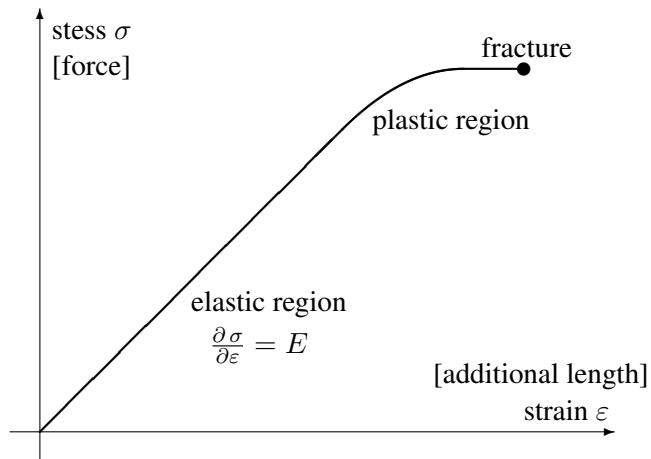


Figure 1.3: Nonlinear stress strain relation

## 1.4 Bending and Buckling of a Beam

### 1.4.1 Description

Examine a beam of length  $L$  as shown in Figure 1.4. The geometric description is given by its angle  $\alpha$  as a function of the arc length  $0 \leq s \leq L$ . In Table 1.6 find a list of all relevant expressions. Since the slope of the curve  $(x(s), y(s))$  is given by the angle  $\alpha(s)$  find

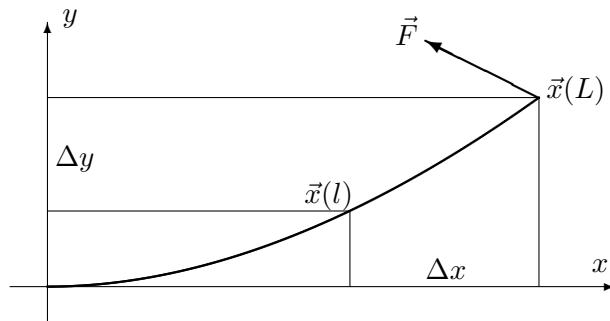


Figure 1.4: Bending of a Beam

$$\frac{d}{ds} \begin{pmatrix} x(s) \\ y(s) \end{pmatrix} = \begin{pmatrix} \cos(\alpha(s)) \\ \sin(\alpha(s)) \end{pmatrix}$$

and construct the curve from the angle function  $\alpha(s)$  with an integral

$$\vec{x}(l) = \begin{pmatrix} x(l) \\ y(l) \end{pmatrix} = \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} + \int_0^l \begin{pmatrix} \cos(\alpha(s)) \\ \sin(\alpha(s)) \end{pmatrix} ds \quad \text{for } 0 \leq l \leq L.$$

description	symbol	SI units
arc length parameter	$0 \leq s \leq L$	[m]
horizontal and vertical position	$x(s), y(s)$	[m]
curvature	$\kappa(s)$	[m <sup>-1</sup> ]
force at right end point	$\vec{F}$	[N]
modulus of elasticity	$E$	[N/m <sup>2</sup> ]
inertia of cross section	$I$	[m <sup>4</sup> ]

Table 1.6: Variables used for a bending beam

The definition of the curvature  $\kappa$  is the rate of change of the angle  $\alpha(s)$  as function of the arc length  $s$ . Thus it is given by

$$\kappa(s) = \frac{d}{ds} \alpha(s).$$

The theory of elasticity implies that the curvature at each point is proportional to the total moment generated by all forces to the right of this point. If only one force  $\vec{F} = (F_1, F_2)^T$  is applied at the endpoint this moment  $M$  is given by

$$EI \kappa = M = F_2 \Delta x - F_1 \Delta y.$$

Since

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} x(L) - x(l) \\ y(L) - y(l) \end{pmatrix} = \int_l^L \begin{pmatrix} \cos(\alpha(s)) \\ \sin(\alpha(s)) \end{pmatrix} ds$$

we can rewrite the above equation as a differential-integral equation. Then computing one derivative with respect to the variable  $l$  transforms the problem into a second order differential equation.

$$\begin{aligned} EI \alpha'(l) &= \int_l^L \begin{pmatrix} F_2 \\ -F_1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha(s)) \\ \sin(\alpha(s)) \end{pmatrix} ds \\ (EI \alpha'(l))' &= - \begin{pmatrix} F_2 \\ -F_1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha(l)) \\ \sin(\alpha(l)) \end{pmatrix} = F_1 \sin(\alpha(l)) - F_2 \cos(\alpha(l)). \end{aligned}$$

If the beam starts out horizontally we have  $a(0) = 0$  and since the moment at the right end point vanishes we find the second boundary condition as  $\alpha'(L) = 0$ . Thus we find a nonlinear, second order boundary value problem

$$(EI \alpha'(s))' = F_1 \sin(\alpha(s)) - F_2 \cos(\alpha(s)). \quad (1.16)$$

In the above general form this problem can only be solved approximately by numerical methods, see Example 4–11 on page 299.

### 1.4.2 Bending of a Beam

If the above beam is only submitted to a vertical force ( $F_1 = 0$ ) and the parameters are assumed to be constant then the equation to be examined is

$$-\alpha''(s) = \frac{F_2}{EI} \cos(\alpha(s)). \quad (1.17)$$

The boundary conditions  $\alpha(0) = \alpha'(L) = 0$  describe the situation of a beam clamped at the left edge and no moment is applied at the right end point.

#### Approximation for small angles

If we know that  $F_2 \neq 0$  and all angles remain small ( $|\alpha| \ll 1$ ) we may simplify the nonlinear term (use  $\cos \alpha \approx 1$  and  $\sin \alpha \approx 0$ ) and find equation

$$-\alpha''(s) = \frac{F_2}{EI}$$

and an integration over the interval  $s < z < L$  leads to

$$\alpha'(s) = \alpha'(L) - \int_s^L \alpha''(z) dz = 0 + \frac{F_2}{EI} (L - s)$$

and another integration from 0 to  $s$ , using  $\alpha(0) = 0$ , leads to

$$\alpha(s) = \alpha(0) + \int_0^s \alpha'(z) dz = \frac{F_2}{EI} \left( Ls - \frac{1}{2} s^2 \right).$$

Since all angles are assumed to be small ( $\sin \alpha \approx \alpha$ ) this implies

$$y(x) = \int_0^x \alpha(z) dz = \frac{F_2}{EI} \left( \frac{1}{2} L x^2 - \frac{1}{6} x^3 \right) = \frac{F_2}{6 EI} (3L - x) x^2$$

and thus we find the maximal deflection at  $x = L$  by

$$y(L) = \frac{F_2}{3 EI} L^3.$$

This result may be useful to verify the results of numerical algorithms.

### 1.4.3 Buckling of a Beam

If the above beam is horizontally compressed ( $F_1 < 0$ ,  $F_2 = 0$ ) and the parameters are assumed to be constant the equation to be solved is

$$-\alpha''(s) = k \sin(\alpha(s)) \quad \text{where} \quad k = \frac{-F_1}{EI} > 0. \quad (1.18)$$

The boundary conditions  $\alpha'(0) = \alpha'(L) = 0$  describe the situation with no moments applied at the two end points. This equation is trivially solved by  $\alpha(s) = 0$ , independent on the value of  $F_1$ .

For small displacements use  $\sin \alpha \approx \alpha$  and the equation simplifies to

$$-\alpha''(s) = k \alpha(s),$$

then use the fact that for

$$k_n = \left( \frac{n\pi}{L} \right)^2 \quad \text{where} \quad n \in \mathbb{N}$$

the functions  $u_n(s) = \cos(\sqrt{k_n} s)$  are nontrivial solutions of the linearized equation

$$u_n''(s) = -k_n u_n(s) \quad \text{with} \quad u_n'(0) = u_n'(L) = 0.$$

These solutions can be used as starting points for Newtons method applied to the nonlinear problem (1.18) .

The first ( $n = 1$ ) of the above solutions is characterized by the critical force  $F_c$

$$k_1 = \frac{\pi^2}{L^2} = \frac{F_c}{EI} \quad \Rightarrow \quad F_c = \frac{EI\pi^2}{L^2}.$$

If the actual force  $F_1$  is smaller than  $F_c$  there will be no deflection of the beam. As soon as the critical value is reached there will be a sudden, drastic deformation of the beam. The corresponding solution is

$$u_1(s) = \varepsilon \cos\left(\frac{\pi}{L} s\right).$$

Thus for small values of the parameter  $\varepsilon$  we expect approximate solutions of the form

$$\alpha(s) = \varepsilon u_1(s) = \varepsilon \cos\left(\frac{\pi}{L} s\right).$$

Using the integral formula

$$\begin{aligned} \vec{x}(l) &= \begin{pmatrix} x(l) \\ y(l) \end{pmatrix} = \int_0^l \begin{pmatrix} \cos(\alpha(s)) \\ \sin(\alpha(s)) \end{pmatrix} ds = \int_0^l \begin{pmatrix} \cos(\varepsilon \cos(\frac{\pi}{L} s)) \\ \sin(\varepsilon \cos(\frac{\pi}{L} s)) \end{pmatrix} ds \\ &\approx \int_0^l \begin{pmatrix} 1 \\ \varepsilon \cos(\frac{\pi}{L} s) \end{pmatrix} ds = \begin{pmatrix} l \\ \varepsilon \frac{L}{\pi} \sin(\frac{\pi}{L} l) \end{pmatrix}. \end{aligned}$$

In the above expression we can eliminate the variable  $l$  and find

$$y(x) = \varepsilon \frac{L}{\pi} \sin\left(\frac{\pi}{L} x\right).$$

This shape corresponds to the Euler buckling of the beam.

## 1.5 Tikhonov Regularization

Assume that you have a non-smooth function  $f(x)$  on the interval  $[a, b]$  and want to approximate it by a smoother function  $u$ . One possible approach is to use a Tikhonov functional. For a parameter  $\lambda > 0$  minimize the functional

$$T_\lambda(u) = \int_a^b |u(x) - f(x)|^2 dx + \lambda \int_a^b (u'(x))^2 dx.$$

- If  $\lambda > 0$  is very small the functional will be minimized if  $u \approx f$ , i.e. a good approximation of  $f$ .
- If  $\lambda$  is very large the integral of  $(u')^2$  will be very small, i.e. we have a very smooth function.

run  
Tikhonov-  
Slider.m

The above problem can be solved with the help of an Euler–Lagrange equation, to be examined in Section 5.2.1, starting on page 304. For smooth functions  $f(x)$  the minimizer  $u(x)$  of the Tikhonov functional  $T_\lambda(u)$  is a solution of the boundary value problem

$$-\lambda u''(x) + u(x) = +f(x) \quad \text{with} \quad u'(a) = u'(b) = 0.$$

In Figure 1.5 find a function  $f(x)$  with a few jumps and some noise added. The above problem was solved for three different values of the positive parameter  $\lambda$ .

- For small values of  $\lambda$  (e.g.  $\lambda = 10^{-5}$ ) the original curve  $f(x)$  is matched very closely, including the jumps. Caused by the noise the curve is very wiggly. In Figure 1.5 points only are displayed.
- For intermediate values of  $\lambda$  (e.g.  $\lambda = 10^{-3}$ ) the original curve  $f(x)$  is matched reasonably well and the jumps are not visible.
- For large values of  $\lambda$  (e.g.  $\lambda = 10^{-1}$ ) the jumps in  $f(x)$  completely disappear, but the result  $u(x)$  is far off the original (noisy) function  $f(x)$ .

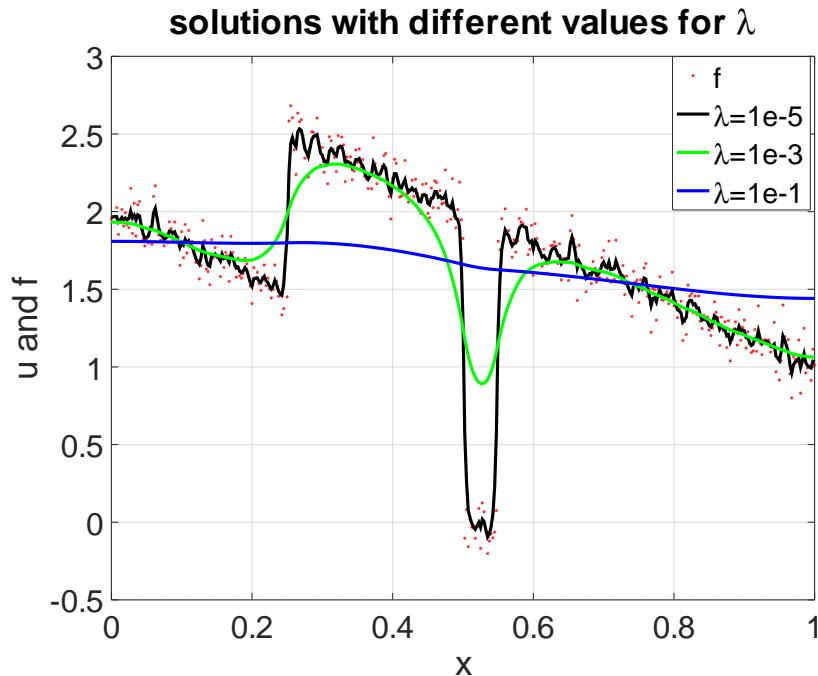


Figure 1.5: A nonsmooth function  $f$  and three regularized approximations  $u$ , using a small ( $\lambda = 10^{-5}$ ), intermediate ( $\lambda = 10^{-3}$ ) and large ( $\lambda = 0.1$ ) parameter

The above idea can be modified by using different smoothing functionals, depending on the specific requirements of the application.

Ex. 5.5

## Bibliography

- [Farl82] S. J. Farlow. *Partial Differential Equations for Scientist and Engineers*. Dover, New York, 1982.  
 [Trim90] D. W. Trim. *Applied Partial Differential Equations*. PWS–Kent, 1990.

# Chapter 2

# Matrix Computations

## 2.1 Prerequisites and Goals

After having worked through this chapter

- you should understand the basic concepts of floating point operations on computers.
- you should be familiar with the concept of the condition number of a matrix and its consequences.
- you should understand the algorithm for LR factorization and Cholesky's algorithm.
- you should be familiar with sparse matrices and the conjugate gradient algorithm.
- you should be aware of the importance of a preconditioner for the conjugate gradient algorithm.

In this chapter we assume that you are familiar with

- basic linear algebra.
- matrix and vector operation.
- the algorithm of Gauss to solve a linear system of equations.

## 2.2 Floating Point Operations

This chapter is started with a very basic introduction to the representation of floating point numbers on computers and the basic effects of rounding for arithmetic operations. For a more detailed presentation use [YounGreg72, §1, §2].

### 2.2.1 Floating Point Numbers and Rounding Errors in Arithmetic Operations

On computer floating point numbers have to be stored in binary format. As an example consider the decimal number

$$x = 178.125 = +1.78125 \cdot 10^2.$$

Since  $178 = 128 + 32 + 16 + 2$  and  $0.125 = 0 \frac{1}{2} + 0 \frac{1}{4} + 1 \frac{1}{8} = 2^{-3}$  find the binary representation

$$x = 10110010.001 = +1.0110010001 \cdot 2^{111}.$$

In this binary scientific representation the integer part will always equals 1. Thus this number could be stored in 14 bit, consisting of 1 sign bit, 3 exponent bits and 10 base bits. Obviously this type of representation is asking for standardization, which was done in 1985 with the IEEE-754 standard. As an example examine

the float format, to be stored in 32 bits. The information is given by one sign bit  $s$ , 8 exponent bits  $b_j$  and 23 bits  $a_j$  for the base.

$s$	$b_0$	$b_1$	$b_2$	$\dots$	$b_7$	$a_1$	$a_2$	$a_3$	$\dots$	$a_{23}$
-----	-------	-------	-------	---------	-------	-------	-------	-------	---------	----------

This leads to the number

$$x = \pm a \cdot 2^{b-B_0} \quad \text{where} \quad a = 1 + \sum_{j=1}^{23} a_j 2^{-j} \quad \text{and} \quad b = \sum_{k=0}^7 b_k 2^k.$$

The sign bit  $s$  indicates whether the number is positive or negative. The exponent bits  $b_k \in \{0, 1\}$  represent the exponent  $b$  in the binary basis where the bias  $B_0$  is chosen such that  $b \geq 0$ . The size of  $B_0$  depends on the exact type of real number, e.g. for the data type float the IEEE committee picked  $B_0 = 127$ . The base bits  $a_j \in \{0, 1\}$  represent the base  $1 \leq a < 2$ . Thus numbers  $x$  in the range  $1.2 \cdot 10^{-38} < |x| < 3.4 \cdot 10^{38}$  can be represented with approximately  $\log_{10}(2^{23}) = 23 \frac{\ln(2)}{\ln(10)} \approx 7$  significant decimal digits. It is important to observe that not all numbers can be represented exactly. The absolute error is at least of the order  $2^{-B_0-23}$  and the relative error is at most  $2^{-23}$ .

On the Intel 486 processor (see [Intel90, §15]) find the floating point data types in Table 2.1. The two data types `float` and `double` exist in the memory only. As soon as a number is loaded into the CPU, it is automatically converted to the extended format, the format used for all internal operations. When a number is moved from the CPU to memory it has<sup>1</sup> to be converted to `float` or `double` format. The situation on other hardware is very similar. As a consequence it is reasonable to assume that all computations will be carried out with one of those two formats. The additional accuracy of the extended format is used as guard digits. Find additional information in [DowdSeve98] or [HeroArnd01]. The reference [Gold91], also available on the internet, gives more information on the IEEE-754 standard for floating point operations.

data type	bytes	bits	base $a$	digits	exponent $b - B_0$	range
float	4	32	23 bit	7	$-126 \leq b - B_0 \leq 127$	$10^{38}$
double	8	64	52 bit	16	$-1022 \leq b - B_0 \leq 1023$	$10^{308}$
extended	10	80	63 bit	19	$-16382 \leq b - B_0 \leq 16383$	$10^{4931}$

Table 2.1: Binary representation of floating point numbers

When analyzing the performance of algorithms for matrix operations a notation to indicate the accuracy is necessary. When storing a real number  $x$  in memory some roundoff might occur and a number  $x(1 + \varepsilon)$ , where  $\varepsilon$  is bound by a number  $\mathbf{u}$ , depending on the CPU architecture used.

$$x \xrightarrow{\text{stored}} x(1 + \varepsilon) \quad \text{where} \quad |\varepsilon| \leq \mathbf{u}$$

For the above standard formats we may work with the following values for the **unit roundoff**  $\mathbf{u}$ .

$$\begin{array}{llll} \text{float:} & \mathbf{u} & \approx & 2^{-23} \approx 1.2 \cdot 10^{-7} \\ \text{double:} & \mathbf{u} & \approx & 2^{-52} \approx 2.2 \cdot 10^{-16} \end{array} \quad (2.1)$$

The notation of  $\mathbf{u}$  for the unit roundoff is adapted form [GoluVanLoan96].

### Addition and multiplication of floating point numbers

Examine two floating point numbers  $x_1 = a_1 \cdot 2^{b_1}$  and  $x_2 = a_2 \cdot 2^{b_2}$ . Use that  $1 \leq |a_i| < 2$  and assume  $b_1 \leq b_2$ . When computing the sum  $x = x_1 + x_2$  the following steps have to be performed:

<sup>1</sup>We quietly ignore that the format `long double` might be used in C. Most numerical code is using `double` or one might consider `float`, to save memory.

1. Rescale the binary representation of the smaller number, such that it will have the same exponent as the larger number..
2. Add the two new base numbers using all available digits. Assuming that  $B$  binary digits are correct and thus the error of the sum of the bases is of the size  $2^{-B}$ .
3. Convert the result into the correct binary representation.

$$x_1 + x_2 = a_1 \cdot 2^{b_1} + a_2 \cdot 2^{b_2} = a_1 \cdot 2^{b_2 - \Delta b} + a_2 \cdot 2^{b_2} = (a_1 \cdot 2^{-\Delta b} + a_2) \cdot 2^{b_2} = a \cdot 2^b$$

The absolute error is of the size  $err \approx 2^{b_2 - B}$ . The relative error  $err/(x_1 + x_2)$  can not be estimated, since the sum might be a small number if  $x_1$  and  $x_2$  are of similar size with opposite sign. If  $x_1$  and  $x_2$  are of similar size and have the same sign, then the relative error may be estimated as  $2^{-B}$ .

When computing the product  $x = x_1 \cdot x_2$  the following steps have to be performed:

1. Use the standard binary representation the two numbers.
2. Add the two exponents and multiply the two bases, using all available digits. Since  $1 \leq a_i < 2$  we know  $1 \leq a_1 \cdot a_2 < 4$ , but (at most) one shift operation will move the base back into the allowed domain. We assume  $B$  binary digits are correct and thus the error of the product of the bases is of the size  $2^{-B}$ .
3. Convert the result into the correct binary representation.

$$x_1 \cdot x_2 = a_1 \cdot 2^{b_1} \cdot a_2 \cdot 2^{b_2} = a_1 \cdot a_2 \cdot 2^{b_1 + b_2} = a \cdot 2^b$$

The absolute error is of the size  $err \approx 2^{b_1 + b_2 - B}$ . The relative error  $err/(x_1 \cdot x_2)$  is estimated by  $2^{-B}$ .

Based on the above arguments conclude that any implementation of floating point operations will necessarily lead to approximation errors in the results. For an algorithm to be useful we have to assure that those errors can not falsify the results. More information on floating point operations may be found in [Wilk63] or [YounGreg72, §2.7].

**2–1 Example :** To illustrate the above effects examine two numbers given by their decimal representation. Use  $x_1 = 7.65432 \cdot 10^3$ ,  $x_2 = 1.23456 \cdot 10^5$  and assume that all operations are carried out with 6 significant digits. There is no clever rounding scheme, all digits beyond the sixth are to be chopped off.

(a) Addition:

$$\begin{aligned} x_1 + x_2 &= 7.65432 \cdot 10^3 + 1.23456 \cdot 10^5 = (7.65432 \cdot 10^{-2} + 1.23456) \cdot 10^5 \\ &= (0.07654 + 1.23456) \cdot 10^5 = 1.31110 \cdot 10^5 \end{aligned}$$

Using a computation with more digits leads to  $x_1 + x_2 = 1.3111032 \cdot 10^5$  and thus find an absolute error of approximately 0.3 or a relative error of  $\frac{0.3}{x_1 + x_2} \approx 2 \cdot 10^{-6}$ .

(b) Subtraction:

$$\begin{aligned} x_1 - x_2 &= 7.65432 \cdot 10^3 - 1.23456 \cdot 10^5 = (7.65432 \cdot 10^{-2} - 1.23456) \cdot 10^5 \\ &= (0.07654 - 1.23456) \cdot 10^5 = -1.15802 \cdot 10^5 \end{aligned}$$

With a computation with more digits find  $x_1 - x_2 = 1.1580168 \cdot 10^5$  and an absolute error of approximately 0.3 or a relative error of  $\frac{0.3}{x_1 - x_2} \approx 3 \cdot 10^{-6}$ . Thus for this example the errors for addition and subtraction are of similar size. If we were to redo the above computations with  $x_1 = 1.23457 \cdot 10^5$ , then the difference equals  $0.00001 \cdot 10^5 = 1$ . The absolute error would not change drastically, but the relative error would be huge, caused by the division by a very small number.

## (c) Multiplication

$$x_1 \cdot x_2 = 7.65432 \cdot 10^3 \cdot 1.23456 \cdot 10^5 = (7.65432 \cdot 1.23456) \cdot 10^8 = 9.44971 \cdot 10^8$$

The absolute error is approximately  $8 \cdot 10^2$  and the relative error  $10^{-6}$ , as has to be expected for a multiplication with 6 significant digits.



**2–2 Example :** The effects of approximation errors on additions and subtractions can also be examined assuming that  $x_i$  is known with an error of  $\Delta x_i$ , i.e  $X_i = x_i \pm \Delta x_i$ . Find

$$\begin{aligned} X_1 + X_2 &= (x_1 \pm \Delta x_1) + (x_2 \pm \Delta x_2) = x_1 + x_2 \pm (\Delta x_1 \pm \Delta x_2), \\ X_1 \cdot X_2 &= (x_1 \pm \Delta x_1) \cdot (x_2 \pm \Delta x_2) = x_1 \cdot x_2 \pm (x_2 \cdot \Delta x_1 \pm x_1 \cdot \Delta x_2 \pm \Delta x_1 \cdot \Delta x_2), \\ \frac{X_1 \cdot X_2}{x_1 \cdot x_2} &\approx 1 \pm \frac{\Delta x_1}{x_1} \pm \frac{\Delta x_2}{x_2}. \end{aligned}$$

The above can be rephrased in a very compact form:

- When adding two numbers the absolute errors will be added.
- When multiplying two numbers the relative errors will be added.



### 2.2.2 Flops, Accessing Memory and Cache

When evaluating the performance of hardware or algorithm we need a good measure of the computational effort necessary to run a given code. Most numerical code is dominated by matrix and vector operations and those in turn are dominated by operations of the type

$$C(i, j) = C(i, j) + A(i, k) * B(k, j).$$

One typical operation involves one multiplication, one addition, a couple of address computations and access to the data in memory or cache. Thus the above is selected as a standard and call the effort to perform these operations one **flop**<sup>2</sup>, short for **f**loating point **o**peration. The abbreviation **FLOPS** stands for **F**loating point **O**peration **P**er **S**econd.

The computational effort for the evaluation of transcendental functions is obviously higher than for an addition or multiplication. Using a simple benchmark with **MATLAB/Octave** and **C++** on a laptop with an Intel I-7 processor Table 2.2 can be generated. The timings are normalized, such that the time to perform one addition leads to 1. No memory access is taken into account for this table. Observe that this is (at best) a rough approximation and depends on many parameters, e.g. the exact hardware and the compiler used.

<sup>2</sup>Over time the common definition of a flop has evolved. In the old days a multiplication took considerably more time than an addition or one memory access. Thus one flop used to equal one multiplication. With RISC architectures multiplications became as fast as additions and thus one flop was either an addition or multiplication. Suddenly most computers were twice as fast. On current computers the memory access takes up most of the time and the memory structure is more important than the raw multiplication/addition power. When reading performance data you have to verify which definition of flop is used. In almost all cases the multiplications are counted.

Operation	Octave	MATLAB	C++
+ , - , *	1	1	1
division	1	1	2
sqrt()	2.3	1.8	1.4
exp()	5	4.3	7.7
sin(), cos()	5.8	4.4	13

Table 2.2: Normalized timing for different operations on an Intel I7

**2–3 Example :** For a few basic operations the flop count is easy to determine.

- (a) A scalar product of two vectors  $\vec{x}, \vec{y} \in \mathbb{R}^n$  is computed by

$$s = \langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^n x_i \cdot y_i = x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 + \dots + x_n \cdot y_n$$

and requires  $n$  flops.

- (b) For a  $n \times n$ -matrix  $\mathbf{A}$  computing the product  $\vec{y} = \mathbf{A} \cdot \vec{x}$  requires  $n^2$  flops, since

$$y_j = \sum_{i=1}^n a_{j,i} \cdot x_i \quad \text{for } j = 1, 2, 3, \dots, n.$$

- (c) Multiplying a  $n \times m$  matrix  $\mathbf{A}$  with a  $m \times r$  matrix  $\mathbf{B}$  requires  $n \cdot m \cdot r$  flops to compute the  $n \times r$  matrix  $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ .

◇

On modern computers the clock rate of the CPU is considerably higher than the clock rate for memory access, i.e. it takes much longer to copy a floating point number from memory into the CPU, than to perform a multiplication. To eliminate this bottleneck in the memory bandwidth sophisticated (and expensive) cache schemes are used. Find further information in [DowdSeve98]. As a typical example examine the three level cache structure of an Alpha processor shown in Figure 2.1. The data given for the access times in Figure 2.1 are partially estimated and take the cache overhead into account. The data is given by [DowdSeve98].

A typical Pentium IV system has 16 KB of level 1 cache (L1) and 512 or 1024 KB on-chip level 2 cache (L2). The dual core Opteron processors available from AMD in 2005 have 64 KB of data cache and 64 KB of instruction cache for each core. The level 2 cache of 1 MB for each core is on-chip too and runs at the same clock rate as the CPU. These caches take up most of the area on the chip.

The importance of a fast and large cache is illustrated with the performance of a banded Cholesky algorithm. For a given number  $n$  this algorithm requires fast access to  $n^2$  numbers. For the data type `double` this leads to  $8 n^2$  bytes of fast access memory. The table below shows some typical values.

$n$	numbers	fast memory
128	16'384	128 KB
256	65'536	512 KB
512	262'144	2048 KB
750	562'500	4395 KB
1024	1'048'576	8192 KB

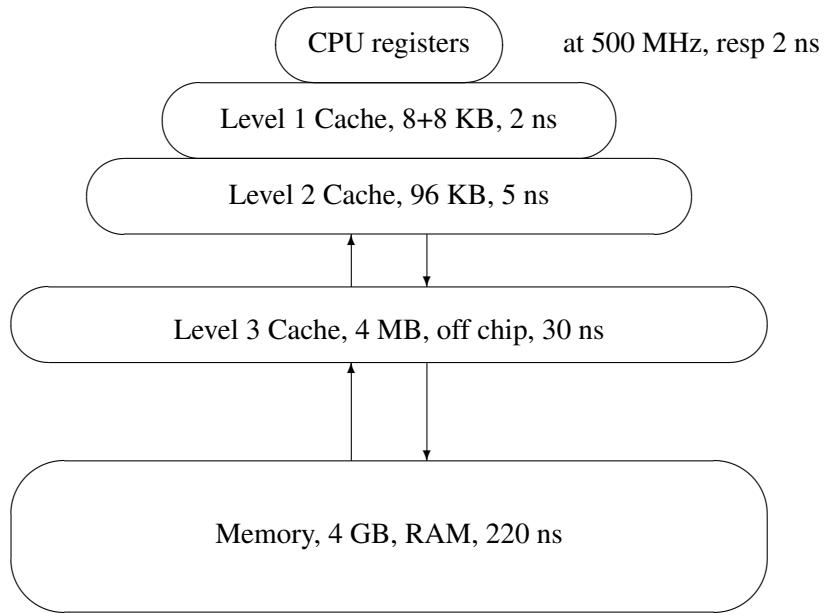


Figure 2.1: Memory and cache access times on a typical 500 MHz Alpha 21164 microprocessor system

CPU	FLOPS
NeXT (68040/25MHz)	1 M
HP 735/100	10 M
SUN Sparc ULTRA 10 (440MHz)	50 M
Pentium III 800 (out of cache)	50 M
Pentium III 800 (in cache)	185 M
Pentium 4 2.6 GHz (out of cache)	370 M
Pentium 4 2.6 GHz (in cache)	450 M
Intel I7-920 (2.67 GHz)	700 M
Intel Haswell I7-5930 (3.5 GHz)	1'800 M
AMD Ryzen 3950X (3.5 GHz)	6'000 M

Table 2.3: FLOPS for a few CPU architectures, using one core only

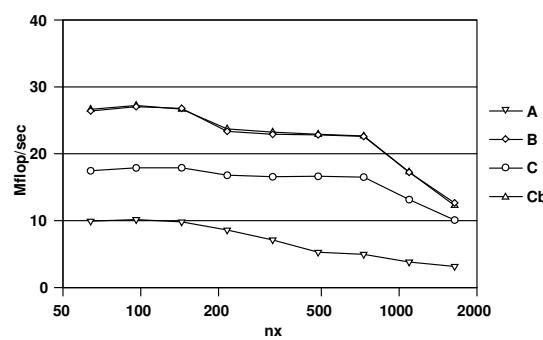


Figure 2.2: FLOPS for a 21164 microprocessor system, four implementations of one algorithm

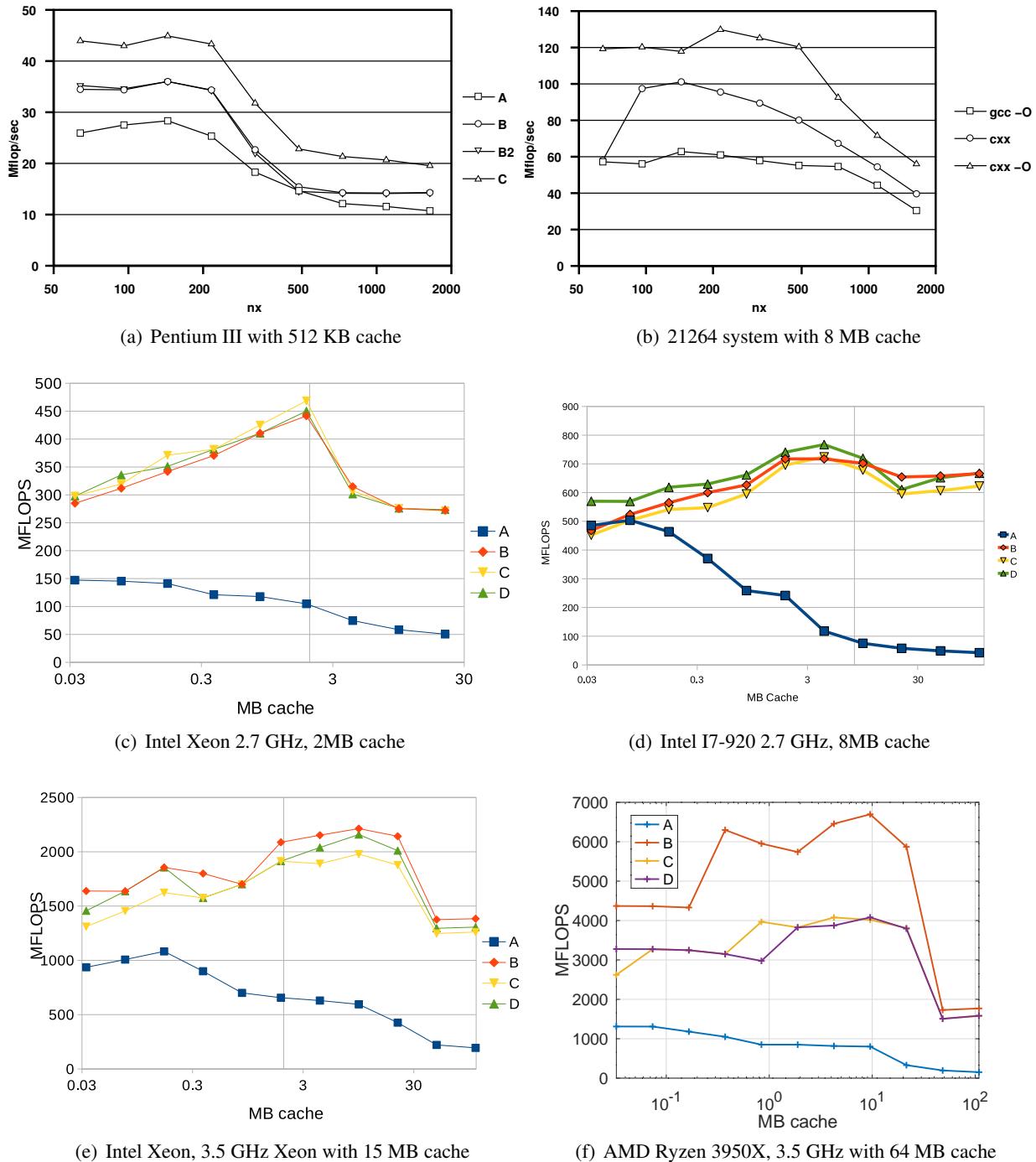


Figure 2.3: FLOPS for a few systems, four implementations of one algorithm

In Figure 2.2 find the performance result for this algorithm on a Alpha 21164 platform with the cache structure from Figure 2.1. Four slightly different codes were tested with different compilers. All codes show the typical drop of in performance if the need for fast memory exceeds the available cache (4 MB).

In Figure 2.3 the similar results for a few different systems are shown. The graphs clearly show that Intel made considerable improvements with their memory access performance. All codes and platforms show the typical drop of in performance if the need for fast memory exceeds the available cache. All results clearly indicate that the choice of compiler and very careful coding and testing is very important for good performance of a given algorithm. In Table 2.3 the performance data for a few common CPU architectures is shown.

For the most recent CPUs this data is rather difficult to determine, since the CPUs do not have a fixed clock rate any more and might execute more than one instruction per cycle.

### 2.2.3 Multi Core Architectures

In 2008 Intel introduced the Nehalem architecture. The design has an excellent interface between the CPU and the memory (RAM) and a three level cache, as shown in Figure 2.4. As a consequence the performance drop for larger problems is not as severe, visible in Figure 2.3(d), where only one core is used.

- a CPU consists of 4 cores
- each core has a separate L1 cache for data (32 KB) and code (32 KB)
- each core has a separate L2 cache for data and code (256 KB)
- the large (8 MB) L3 cache is dynamically shared between the 4 cores

This multi core architecture allows the four cores to work on the same data. To take full advantage of this it is required that the algorithms can run several sections in parallel (threading). Since CPUs with this or similar architecture are very common we will exam all algorithms on whether they can take advantage of multiple cores. More information on possible architectures is given in [DowdSeve98].

Core 1		Core 2		Core 3		Core 4	
32 KB code	32 KB data	32 KB code	32 KB data	32 KB code	32 KB data	32 KB code	32 KB data
L2 cache		L2 cache		L2 cache		L2 cache	
256KB		256KB		256KB		256KB	
8 MB L3 cache, shared							

Figure 2.4: CPU-cache structure for the Intel I7-920 (Nehalem)

- In 2014 Intel introduced the Haswell-E architecture, a clear step up from the Nehalem architecture. The size of the third level cache is larger and a better memory interface is used. As an example consider the CPU I7-5930:
- a CPU consists of 6 cores

- each core has a separate L1 cache for data (32 KB) and code (32 KB)
- each core has a separate L2 cache for data and code (256 KB)
- the large (15 MB) L3 cache is dynamically shared between the 6 cores

The performance is good, as shown in Figure 2.3(e).

- In 2017 AMD introduced the Ryzen Threadripper 1950X with 16 cores, 96KB (32KB data, 64KB code) of L1 cache per core, 8MB of L2 cache and 32MB of L3 cache.
- In the fall of 2019 AMD topped the above with the Ryzen 9 3950X with 16 cores. This processor has 64KB of L1 cache and 512KB of L2 cache dedicated for each of the 16 cores and 64MB of shared L3 cache. The performance is outstanding, see Figure 2.3(f). The structure in Figure 2.4 is still valid, just more cores and (a lot) more cache.

The above observations consider only the aspect on single CPUs, possibly with multiple cores. For supercomputing clusters the effect of a huge number of computing cores is even more essential.

It is rather surprising to observe the development: in 1976 the Cray 1 cost 8.8 M\$, weighed 5.5 tons and could perform up to 160 MFLOPS. In 2014 a NVIDIA GeForce GTX Titan Black card cost 1'000\$ and can deliver up to 1'800 GFLOPS (double precision, FP64). Thus the 2014 GPU is 11'000 times faster, very light and at a very affordable price.

## 2.2.4 Using Multiple Cores of a CPU with *Octave* and **MATLAB**

Many commands in MATLAB will take advantage of multiple cores available on the CPU. Current versions of *Octave* can use the libraries OPENBLAS or ATLAS to use multiple cores for some of the built-in commands. On Linux systems use `top` or `htop` to observe the load on the system with many cores<sup>3</sup>.

run demo  
CompCores

```
N = 2^12; % = 4096 % size of the matrix
A = eye(N) + 0.1*rand(N); Apower = A;
invA = eye(N) + Apower; t0 = cputime(); tic();
for ii = 1:10
    Apower = A * Apower; invA = invA + Apower;
end%for
WallTime = toc()
CPUtime = cputime()-t0
GFLOPS = 10*N^3/WallTime/1e9
```

The above code was used on the host karman (Ryzen 3950X, 16 cores, 32 threads with hyper-threading) and `htop` showed 32 active threads. If launching *Octave* after setting the environment variable `export OPENBLAS_NUM_THREADS=1`, then only one thread is started and the computation time is 23.7 sec. For each of the above 10 iterations more than  $N^3$  flops are required. With  $N = 2^{12} = 4096$  this leads to  $687 \cdot 10^9$  flops. A computation time of 3.16 seconds corresponds to a computational power of 217 GFLOPS for the host karman, which is considerably more than shown in Figure 2.3(f), where a single core is used with a more complex algorithm. Using only a single thread (computation time 23.7 sec) leads to 27 GFLOPS. Since the CPU has 16 cores another attempt using 16 threads shows a computation time of 2.72 seconds, i.e. to 253 GFLOPS<sup>4</sup>. With  $N = 2^{14} = 16'384$  using 16 threads the computational power is 324 GFLOPS. For the same CPU a computational power between 6 GFLOPS and 324 GFLOPS showed up.

Resulting advice:

- Use as many threads as your CPU has true cores, i.e. ignore hyper-threading.

<sup>3</sup>For a matrix  $\mathbf{A}$  with  $\|\mathbf{A}\| < 1$  the inverse of  $\mathbb{I} - \mathbf{A}$  can be determined by the Neumann series  $(\mathbb{I} - \mathbf{A})^{-1} = \mathbb{I} + \mathbf{A} + \mathbf{A}^2 + \mathbf{A}^3 + \mathbf{A}^4 + \dots$ . The code determines the first 10 contributions of this approximation.

<sup>4</sup>With  $N = 4096 = 2^{12}$  and the code `Apower*=A; invA+=Apower;` (*Octave* only) in the loop obtain 279 GFLOPS.

- Do not trust the FLOPS data published by companies (e.g AMD, Intel, ...), use your own application as benchmark. The differences can be substantial.

## 2.3 The Model Matrices

Before examining numerical algorithms for linear systems of equations and the corresponding matrices we present two typical and rather important matrices. All of the subsequent results shall be applied to these model matrices. Many problems examined in Chapter 1 lead to matrices similar to  $\mathbf{A}_n$  or  $\mathbf{A}_{nn}$ .

### 2.3.1 The 1-d Model Matrix $\mathbf{A}_n$

When using a finite difference approximation with  $n$  interior points (see Chapter 4) for the model problem (1.2)

$$-\frac{d^2}{dx^2} T(x) = \frac{1}{k} f(x) \quad \text{for } 0 \leq x \leq 1 \quad \text{and} \quad T(0) = T(1) = 0$$

the function  $T(x)$  is replaced by a set of the values  $T_i$  of the function at  $x_i = i \cdot h = i \frac{1}{n+1}$ , as shown in Figure 2.5.

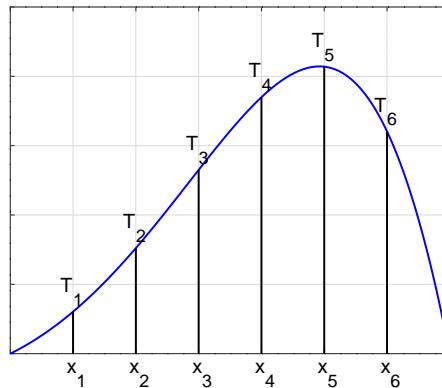


Figure 2.5: The discrete approximation of a continuous function

The second derivative is replaced by a multiplication by a  $n \times n$  matrix  $\mathbf{A}_n$ , where

$$\mathbf{A}_n = \frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix}.$$

The value of  $h = \frac{1}{n+1}$  represents the distance between two points  $x_i$ . Multiplying a vector by this matrix corresponds to computing the second derivative. The differential equation is replaced by a system of linear equations.

$$-\frac{d^2}{dx^2} u(x) = f(x) \quad \longrightarrow \quad \mathbf{A}_n \vec{u} = \vec{f}.$$

Observe the following important facts about this matrix  $\mathbf{A}_n$ :

- The matrix is symmetric, tridiagonal and positive definite
- To analyze the performance of the matrix algorithms the eigenvalues  $\lambda_j$  and eigenvectors  $\vec{v}_j$  will be useful, i.e. use  $\mathbf{A}_n \vec{v}_j = \lambda_j \vec{v}_j$ .
  - The exact eigenvalues are given by<sup>5</sup>

$$\lambda_j = \frac{4}{h^2} \sin^2\left(j \frac{\pi h}{2}\right) \quad \text{for } 1 \leq j \leq n.$$

- The eigenvector  $\vec{v}_j$  is generated by discretizing the function  $\sin(x \frac{j\pi}{n+1})$  over the interval  $[0, 1]$  and thus has  $j$  extrema within the interval.

$$\vec{v}_j = \left( \sin\left(\frac{1j\pi}{n+1}\right), \sin\left(\frac{2j\pi}{n+1}\right), \sin\left(\frac{3j\pi}{n+1}\right), \dots, \sin\left(\frac{(n-1)j\pi}{n+1}\right), \sin\left(\frac{nj\pi}{n+1}\right) \right).$$

- For  $j \ll n$  observe

$$\lambda_j = \frac{4}{h^2} \sin^2\left(j \frac{\pi h}{2}\right) = 4(n+1)^2 \sin^2\left(j \frac{\pi}{2(n+1)}\right) \approx 4(n+1)^2 \left(j \frac{\pi}{2(n+1)}\right)^2 = \pi^2 j^2$$

and in particular

$$\lambda_{min} = \lambda_1 \approx \pi^2.$$

- For the largest eigenvalue use

$$\lambda_{max} = \lambda_n = 4(n+1)^2 \sin^2\left(\frac{\pi}{2} \frac{n}{n+1}\right) \approx 4n^2.$$

The above matrix will be useful for a number of the problems in the introductory chapter.

- $\mathbf{A}_n$  will be used to solve the static heat problem (1.2).
- For each time step in the dynamic heat problem (1.3) the matrix  $\mathbf{A}_n$  will be used.
- To solve the steady state problem for the vertical displacements of a string, i.e. equation (1.7), the above matrix needed.
- For each time step of the dynamic string problem (1.8), we need the above matrix.
- The eigenvalues of the above matrix determine the solutions of the eigenvalue problem for the vibrating string problem, i.e. equation (1.9).
- The problems of the horizontal stretching of a beam (i.e. equations (1.13) and (1.14)) will lead to matrices similar to the above.
- When using Newton's method to solve the nonlinear problem of a bending beam ((1.16), (1.17) or (1.18)) we will have to use matrices similar to the above.

---

<sup>5</sup>To verify these eigenvalues and eigenvectors use the complex exponential function  $u(x) = \exp(i\alpha x)$ . The values at the grid points  $x_k = k h$  are  $u_k = \exp(i\alpha k h)$  leading to a vector  $\vec{u}$ . The matrix multiplication computes

$$\begin{aligned} -u_{k-1} + 2u_k - u_{k+1} &= \exp(i\alpha k h) (-\exp(-i\alpha h) + 2 - \exp(+i\alpha h)) \\ &= \exp(i\alpha k h) (2 - 2 \cos(\alpha h)) = \exp(i\alpha k h) 4 \sin^2\left(\frac{\alpha h}{2}\right). \end{aligned}$$

Thus the values of  $u_k$  are multiplied with the factor  $4 \sin^2(\frac{\alpha h}{2})$ . Now examine the imaginary part of these coefficients, i.e. use  $u_k = \sin(\alpha k h)$ . Based on  $u_0 = u_{n+1} = 0$  require  $\sin(\alpha_j (n+1) h) = \sin(\alpha_j 1) = 0$ , i.e.  $\alpha_j = j\pi$ . Thus the eigenvalues are  $\lambda_j = \frac{4}{h^2} \sin^2\left(j \frac{\pi h}{2}\right)$  and the corresponding eigenvectors are the discretizations of the functions  $\sin(x \frac{j\pi}{n+1})$ .

### 2.3.2 The 2-d Model Matrix $\mathbf{A}_{nn}$

When using a finite difference approximation of the Laplace operator (i.e. compute  $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ ) with  $n^2$  interior points on the unit square leads to a  $n^2 \times n^2$  matrix  $\mathbf{A}_{nn}$ . The mesh for this discretisation in the case of  $n = 4$  is shown in Figure 2.6 with a typical numbering of the nodes.

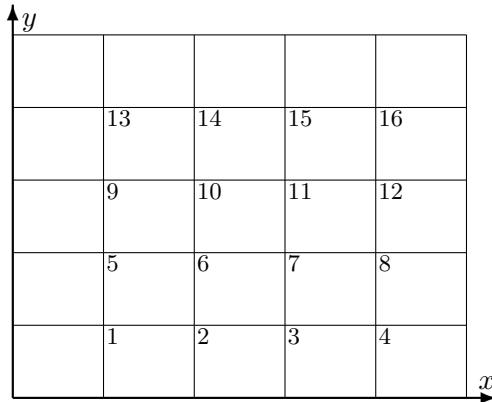


Figure 2.6: A  $4 \times 4$  grid on a square domain

The matrix is characterized by a few key properties:

- The matrix has a leading factor of  $1/h^2$ , where  $h = \frac{1}{n+1}$  represents the distance between neighboring points.
- Along the main diagonal all entries are equal to 4 .
- The  $n$ -th upper and lower diagonals are filled with  $-1$  .
- The first upper and lower diagonals are almost filled with  $-1$  . If the column/row index of the diagonal entry is a multiple of  $n$ , then the numbers to the right and below are zero.

Below find the  $16 \times 16$  matrix  $\mathbf{A}_{nn}$  for  $n = 4$ .

$$\mathbf{A}_{4,4} = \frac{1}{h^2} \begin{bmatrix} 4 & -1 & \cdot & \cdot & \cdot & -1 \\ -1 & 4 & -1 & & & -1 \\ \cdot & -1 & 4 & -1 & & -1 \\ \cdot & & -1 & 4 & & -1 \\ -1 & & & 4 & -1 & -1 \\ & -1 & & -1 & 4 & -1 \\ & & -1 & & -1 & 4 & -1 \\ & & & -1 & & 4 & -1 & -1 \\ & & & & -1 & & 4 & -1 & -1 \\ & & & & & -1 & & 4 & -1 \\ & & & & & & -1 & & -1 \\ & & & & & & & -1 & 4 & -1 \\ & & & & & & & & -1 & 4 & -1 \\ & & & & & & & & & -1 & 4 \end{bmatrix}$$

The missing numbers in the first off-diagonals can be explained by Figure 2.6. The gaps are caused by the fact, that if the node number is a multiple of  $n$  (i.e.  $k \cdot n$ ), then the node has no direct connection to the node with number  $k \cdot n + 1$ .

We replace the partial differential equation by a system of linear equations.

$$-\frac{d^2 u(x, y)}{dx^2} - \frac{d^2 u(x, y)}{dy^2} = \frac{1}{k} f(x, y) \quad \rightarrow \quad \mathbf{A}_{nn} \vec{u} = \vec{f}.$$

Observe the following important facts about this matrix  $\mathbf{A}_{nn}$ :

- The matrix is symmetric and positive definite.
- The matrix is sparse (very few nonzero entries) and has a band structure.
- To analyze the performance of the different algorithms it is convenient to know the eigenvalues.
  - The exact eigenvalues are given by

$$\lambda_{i,j} = \frac{4}{h^2} \sin^2(j \frac{\pi h}{2}) + \frac{4}{h^2} \sin^2(i \frac{\pi h}{2}) \quad \text{for } 1 \leq i, j \leq n.$$

- For  $i, j \ll n$  obtain

$$\lambda_{i,j} \approx \pi^2 (i^2 + j^2)$$

and in particular

$$\lambda_{min} = \lambda_{1,1} \approx 2\pi^2.$$

For the largest eigenvalue find

$$\lambda_{max} = \lambda_{n,n} \approx 8n^2.$$

The above matrix will be useful for a number of the problems in the introductory chapter.

- $\mathbf{A}_{nn}$  will be used to solve the static heat problem (1.5) with 2 space dimensions.
- For each time step in the dynamic heat problem (1.6) the matrix  $\mathbf{A}_{nn}$  will be used.
- To solve the steady state problem for the vertical displacements of a membrane, i.e. equation (1.11), we need the above matrix.
- For each time step of the dynamic membrane problem (1.10), the above matrix is needed.
- The eigenvalues of the above matrix determine the solutions of the eigenvalue problem for the vibrating membrane problem, i.e. equation (1.12).

One can construct the model matrix  $\mathbf{A}_{nnn}$  corresponding to the finite difference approximation of a differential equation examined on a unit cube in space  $\mathbb{R}^3$ . In Table 2.4 find the key properties of the model matrices. Observe that all matrices are symmetric and positive definite.

## 2.4 Solving Systems of Linear Equations and Matrix Factorizations

Most problems in scientific computing require solutions of linear systems of equations. Even if the problem is nonlinear you need good, reliable solvers for linear systems, since they form the building block of nonlinear solvers, e.g. Newton's algorithm (Section 3.1.5). Any algorithm for linear systems will have to be evaluated using the following criteria:

- Computational cost: how many operations are required to solve the system?

	$\mathbf{A}_n$	$\mathbf{A}_{nn}$	$\mathbf{A}_{nnn}$
differential equation on size of grid	unit interval $n$	unit square $n \times n$	unit cube $n \times n \times n$
size of matrix	$n \times n$	$n^2 \times n^2$	$n^3 \times n^3$
semi bandwidth	2	$n$	$n^2$
nonzero entries	$3n$	$5n^2$	$7n^3$
smallest eigenvalue $\lambda_{min} \approx$	$\pi^2$	$2\pi^2$	$3\pi^2$
largest eigenvalue $\lambda_{max} \approx$	$4n^2$	$8n^2$	$12n^2$
condition number $\kappa \approx$	$\frac{4}{\pi^2} n^2$	$\frac{4}{\pi^2} n^2$	$\frac{4}{\pi^2} n^2$

Table 2.4: Properties of the model matrices  $\mathbf{A}_n$ ,  $\mathbf{A}_{nn}$  and  $\mathbf{A}_{nnn}$ .

- Memory cost: how much memory of what type is required to solve the system?
- Accuracy: what type and size of errors are introduced by the algorithm? Is there an unnecessary loss of accuracy. This is an essential criterion if you wish to obtain reliable solutions and not a heap of random numbers.
- Special properties: does the algorithm use special properties of the matrix  $\mathbf{A}$ ? Some of those properties to be considered are: symmetry, positive definiteness, band structure and sparsity.
- Multi core architecture: can the algorithm take advantage of a multi core architecture?

There are three classes of solvers for linear systems of equations to be examined in these notes:

- Direct solvers: we will examine the standard LR factorization and the Cholesky factorization, Sections 2.4.1 and 2.6.1.
- Sparse direct solvers: we examine the banded Cholesky factorization, the simplest possible case, see Section 2.6.4.
- Iterative solvers: we will examine the most important case, the conjugate gradient algorithm, see Section 2.7.

One of the conclusions you should draw from the following sections is that it is almost **never** necessary to compute the inverse of a given matrix.

### 2.4.1 LR Factorization

When solving linear systems of equations the concept of **matrix factorization** is essential. For a known vector  $\vec{b} \in \mathbb{R}^n$  and a given  $n \times n$  matrix  $\mathbf{A}$  we search for solutions  $\vec{x} \in \mathbb{R}^n$  of the system  $\mathbf{A} \cdot \vec{x} = \vec{b}$ . Many algorithms use a matrix factorization  $\mathbf{A} = \mathbf{L} \cdot \mathbf{R}$  where the matrices have special properties to simplify solving the system of linear equations.

#### Solving triangular systems of linear equations

When using a matrix factorization to solve a linear system of equations we regularly have to work with triangular matrices. Thus we briefly examine the main properties of matrices with triangular structure.

**2-4 Definition :**

- A matrix  $\mathbf{R}$  is called an **upper triangular matrix** iff<sup>6</sup>  $r_{i,j} = 0$  for all  $i > j$ , i.e. all entries below the main diagonal are equal to 0 .
- A matrix  $\mathbf{L}$  is called an **lower triangular matrix** iff  $l_{i,j} = 0$  for all  $i < j$ , i.e. all entries above the main diagonal are equal to 0 .
- The two  $n \times n$  matrices  $\mathbf{L}$  and  $\mathbf{R}$  form an **LR factorization** of the matrix  $\mathbf{A}$  if
  - $\mathbf{L}$  is a lower triangular matrix and its diagonal numbers are all 1 .
  - $\mathbf{R}$  is an upper triangular matrix.
  - $\mathbf{A}$  is factorized by the two matrices, i.e.  $\mathbf{A} = \mathbf{L} \cdot \mathbf{R}$

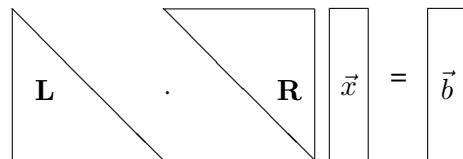
The above factorization

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ l_{2,1} & 1 & 0 & 0 & \dots & 0 \\ l_{3,1} & l_{3,2} & 1 & 0 & \dots & 0 \\ l_{4,1} & l_{4,2} & l_{4,3} & 1 & & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ l_{n,1} & l_{n,2} & \dots & l_{n,n-1} & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} & \dots & r_{1,n} \\ 0 & r_{2,2} & r_{2,3} & r_{2,4} & \dots & r_{2,n} \\ 0 & 0 & r_{3,3} & r_{3,4} & \dots & r_{3,n} \\ 0 & 0 & 0 & r_{4,4} & \dots & r_{4,n} \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & r_{n,n} \end{bmatrix}$$

can be represented graphically by



Solving the system  $\mathbf{A} \vec{x} = \mathbf{L} \mathbf{R} \vec{x} = \vec{b}$  of linear equations



can be performed in two steps

$$\mathbf{A} \vec{x} = \vec{b} \iff \mathbf{L} \cdot \mathbf{R} \vec{x} = \vec{b} \iff \begin{cases} \mathbf{L} \vec{y} = \vec{b} \\ \mathbf{R} \vec{x} = \vec{y} \end{cases} .$$

- Introduce an auxiliary vector  $\vec{y}$ . First solve the system  $\mathbf{L} \vec{y} = \vec{b}$  from top to bottom. The equation represented by row  $i$  in the matrix  $\mathbf{L}$  reads as

$$\begin{array}{rcl} 1 y_1 & = & b_1 \\ l_{2,1} y_1 + 1 y_2 & = & b_2 \\ l_{3,1} y_1 + l_{3,2} y_2 + 1 y_3 & = & b_3 \end{array} .$$

<sup>6</sup>iff: used by mathematicians to spell out “if and only if”

These equations can easily be solved. The general formula is given by

$$y_i + \sum_{j=1}^{i-1} l_{i,j} y_j = b_i \implies y_i = b_i - \sum_{j=1}^{i-1} l_{i,j} y_j.$$

```

y(1) = b(1)
for i= 2 to n
    y(i) = b(i) - L(i,1:i-1)*y(1:i-1)
end%for

```

- Subsequently we consider the linear equations with the matrix  $\mathbf{R}$ , e.g. for a  $3 \times 3$  matrix

$$\begin{aligned} r_{1,1} x_1 + r_{1,2} x_2 + r_{1,3} x_3 &= y_1 \\ r_{2,2} x_2 + r_{2,3} x_3 &= y_2 \\ r_{3,3} x_3 &= y_3 \end{aligned}.$$

These equations have to be solved bottom to top.

$$\sum_{j=i}^n r_{i,j} x_j = y_i \implies x_i = \frac{y_i}{r_{i,i}} - \frac{1}{r_{i,i}} \sum_{j=i+1}^n r_{i,j} x_j.$$

```

x(n) = y(n)/R(n,n)
for i= n-1 to 1
    x(i) = y(i) / R(i,i) - ( R(i,i+1:n)*x(i+1) ) / R(i,i)
end%for

```

- Forward and backward substitution require each approximately  $\sum_{k=1}^n k \approx \frac{1}{2} n^2$  flops. Thus the total computational effort is given by  $n^2$  flops.

The above observations show that systems of linear equations are easily solved, once the original matrix is factorized as a product of a left and right triangular matrix. In the next section we show that this factorization can be performed using the ideas of the algorithm of Gauss to solve linear systems of equations.

### LR Factorization and the Algorithm of Gauss

The algorithm of Gauss is based on the idea of row reduction of a matrix. As an example consider a system of three equations.

$$\begin{bmatrix} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{bmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

The basic idea of the algorithm is to use row operations to transform the matrix in echelon form. Using the notation of an augmented matrix this translates to

$$\left[ \begin{array}{ccc|c} 2 & 6 & 2 & 2 \\ -3 & -8 & 0 & 4 \\ 4 & 9 & 2 & 6 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 2 & 6 & 2 & 2 \\ 0 & 1 & 3 & 7 \\ 0 & -3 & -2 & 2 \end{array} \right] \rightarrow \left[ \begin{array}{ccc|c} 2 & 6 & 2 & 2 \\ 0 & 1 & 3 & 7 \\ 0 & 0 & 7 & 23 \end{array} \right].$$

In the above computations the following steps were applied:

Ex  
2.1,2.2,2.3

- Multiply the first row by  $\frac{3}{2}$  and add it to the second row.
- Multiply the first row by 2 and subtract it from the third row.
- Multiply the modified second row by 3 and add it to the third row.

The resulting system is represented by a triangular matrix and can easily be solved from bottom to top.

$$\begin{array}{l} 2x_1 + 6x_2 + 2x_3 = 2 \\ \quad 1x_2 + 3x_3 = 7 \\ \quad \quad 7x_3 = 23 \end{array}$$

The next goal is to verify that an LR factorization is a clever notation for the algorithm of Gauss. To verify this we use the notation of block matrices and a recursive scheme, i.e. we start with a problem of size  $n \times n$  and reduce it to a problem of size  $(n-1) \times (n-1)$ . For this we divide a matrix in 4 blocks of submatrices, i.e.

$$\mathbf{A} = \left[ \begin{array}{ccccc} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n} \end{array} \right] = \left[ \begin{array}{c|cccc} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ \hline a_{1,2} & & & & \\ a_{1,3} & & & & \\ \vdots & & & & \\ a_{1,n} & & & & \end{array} \right] \mathbf{A}_{n-1}.$$

The submatrix  $\mathbf{A}_{n-1}$  is a  $(n-1) \times (n-1)$  matrix. Using this notation we are searching  $n \times n$  matrices  $\mathbf{L}$  and  $\mathbf{R}$  such that  $\mathbf{A} = \mathbf{L} \cdot \mathbf{R}$ , i.e.

$$\left[ \begin{array}{c|ccccc} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ \hline a_{2,1} & & & & \\ a_{3,1} & & & & \\ \vdots & & & & \\ a_{n,1} & & & & \end{array} \right] = \left[ \begin{array}{c|ccccc} 1 & 0 & 0 & \dots & 0 \\ \hline l_{2,1} & & & & \\ l_{3,1} & & & & \\ \vdots & & & & \\ l_{n,1} & & & & \end{array} \right] \cdot \left[ \begin{array}{c|ccccc} r_{1,1} & r_{1,2} & r_{1,3} & \dots & r_{1,n} \\ \hline 0 & & & & \\ 0 & & & & \\ \vdots & & & & \\ 0 & & & & \end{array} \right] \mathbf{R}_{n-1}.$$

Using the standard matrix multiplication compute the entries in the 4 segments of  $\mathbf{A}$  separately and then examine  $\mathbf{A}$  block by block.

- Examine the top left block (one single number) in  $\mathbf{A}$

$$a_{1,1} = 1 \cdot r_{1,1}.$$

- Examine the top right block (row) in  $\mathbf{A}$

$$a_{1,j} = 1 \cdot r_{1,j} \quad \text{for } j = 2, 3, \dots, n.$$

Thus the first row of  $\mathbf{R}$  is a copy of the first row of  $\mathbf{A}$ .

- Examine the bottom left block (column) in  $\mathbf{A}$

$$\left[ \begin{array}{c} a_{2,1} \\ a_{3,1} \\ \vdots \\ a_{n,1} \end{array} \right] = \left[ \begin{array}{c} l_{2,1} \\ l_{3,1} \\ \vdots \\ l_{n,1} \end{array} \right] \cdot r_{1,1} \quad \Rightarrow \quad l_{i,1} = \frac{a_{i,1}}{r_{1,1}} = \frac{a_{i,1}}{a_{1,1}}.$$

For the standard algorithm of Gauss, this is the multiple to be used when adding a multiple of the first row to row  $i$ . This step might fail if  $a_{1,1} = 0$ . This possible problem can be avoided with the help of proper pivoting. This will be examined later in this course, see Section 2.5.3, page 52.

- Examine the bottom right block in  $\mathbf{A}$

$$\mathbf{A}_{n-1} = \begin{bmatrix} l_{2,1} \\ l_{3,1} \\ \vdots \\ l_{n,1} \end{bmatrix} \cdot \begin{bmatrix} r_{1,2} & r_{1,3} & \dots & r_{1,n} \end{bmatrix} + \mathbf{L}_{n-1} \cdot \mathbf{R}_{n-1}.$$

This can be rewritten in the form

$$\mathbf{L}_{n-1} \cdot \mathbf{R}_{n-1} = \mathbf{A}_{n-1} - \begin{bmatrix} l_{2,1} \\ l_{3,1} \\ \vdots \\ l_{n,1} \end{bmatrix} \cdot \begin{bmatrix} a_{1,2} & a_{1,3} & \dots & a_{1,n} \end{bmatrix} = \tilde{\mathbf{A}}_{n-1}.$$

This operation is a sequence of row operations on the  $(n-1) \times (n-1)$  matrix  $\mathbf{A}_{n-1}$ :

From each row a multiple (factor  $l_{i,1} = \frac{a_{i,1}}{a_{1,1}}$ ) of the first row is subtracted.

Thus the lower triangular matrix  $\mathbf{L}$  keeps track of what row operations have to be applied to transform  $\mathbf{A}_{n-1}$  into  $\tilde{\mathbf{A}}_{n-1}$ .

- Verify that these operations require  $(n-1) + (n-1)^2 = n(n-1)$  flops.
- Observe that the first row and column of  $\mathbf{A}$  will not have to be used again in the next step. Thus for a memory efficient implementation we may overwrite the first row and column of  $\mathbf{A}$  with the first row of  $\mathbf{R}$  and the first column of  $\mathbf{L}$ . The number 1 on the diagonal of  $\mathbf{L}$  does not have to be stored.

After having performed the above steps we are left with a similar question, but the size of the matrices was reduced by 1. By recursion we can restart the above process with the reduced matrices. Finally we will find the LR factorization of the matrix  $\mathbf{A}$ . The total operation count is given by

$$\text{Flop}_{\text{LR}} = \sum_{k=1}^n (k^2 - k) \approx \frac{1}{3} n^3.$$

It is possible to compute the inverse  $\mathbf{A}^{-1}$  of a given square matrix  $\mathbf{A}$  with the help of the LR factorization. It can be shown (e.g. [Schw86]) that the computational effort is approximately  $\frac{4}{3} n^3$  and thus 4 times as high as solving one system of linear equations directly.

**2–5 Observation :** Adding multiples of one row to another row in a large matrix can be implemented in parallel on a multicore architecture, as shown in Section 2.2.3. For this to be efficient the number of columns has to be considerably larger than the number CPU cores to be used. On most computers this task is taken over by a well chosen BLAS library (Basic Linear Algebra Subroutines). Excellent versions are provided by OPENBLAS or by ATLAS (Automatically Tuned Linear Algebra Software). MATLAB uses the Intel Math Kernel Library and Octave is built on OPENBLAS by default. ◇

**2–6 Example :** The above general calculations are illustrated with the numerical example used at the start of this section. For the given  $3 \times 3$  matrix  $\mathbf{A}$  we first examine the first column of the left triangular matrix  $\mathbf{L}$  and the first row of the right triangular matrix  $\mathbf{R}$ .

$$\begin{bmatrix} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{2,1} & 1 & 0 \\ l_{3,1} & l_{3,2} & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ 0 & r_{2,2} & r_{2,3} \\ 0 & 0 & r_{3,3} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ \frac{-3}{2} & 1 & 0 \\ 2 & l_{3,2} & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 6 & 2 \\ 0 & r_{2,2} & r_{2,3} \\ 0 & 0 & r_{3,3} \end{bmatrix}.$$

Then we restart the computation with the  $2 \times 2$  blocks in the lower right corner of the above matrices. From the above conclude

$$\begin{bmatrix} -8 & 0 \\ 9 & 2 \end{bmatrix} = \begin{bmatrix} \frac{-3}{2} \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 6 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ l_{3,2} & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{2,2} & r_{2,3} \\ 0 & r_{3,3} \end{bmatrix}$$

$$= \begin{bmatrix} -9 & -3 \\ 12 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ l_{3,2} & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{2,2} & r_{2,3} \\ 0 & r_{3,3} \end{bmatrix}.$$

The  $2 \times 2$  block of  $\mathbf{A}$  has to be modified first by adding the correct multiples of the first row of  $\mathbf{A}$ , i.e.

$$\begin{bmatrix} -8 & 0 \\ 9 & 2 \end{bmatrix} - \begin{bmatrix} -9 & -3 \\ 12 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ -3 & -2 \end{bmatrix}$$

and then we use an LR factorization of a  $2 \times 2$  matrix.

$$\begin{bmatrix} 1 & 3 \\ -3 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l_{3,2} & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{2,2} & r_{2,3} \\ 0 & r_{3,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 \\ 0 & r_{3,3} \end{bmatrix}.$$

The only missing value of  $r_{3,3}$  can be determined by examining the lower right corner of the above matrix product.

$$-2 = (-3) \cdot 3 + 1 \cdot r_{3,3} \implies r_{3,3} = 7.$$

Thus conclude

$$\mathbf{A} = \begin{bmatrix} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{-3}{2} & 1 & 0 \\ 2 & -3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 7 \end{bmatrix} = \mathbf{L} \cdot \mathbf{R}.$$

Instead of solving the system

$$\begin{bmatrix} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{bmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

first solve

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{-3}{2} & 1 & 0 \\ 2 & -3 & 1 \end{bmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

from top to bottom with the solution

$$y_1 = 2 \quad \Rightarrow \quad y_2 = 4 + \frac{3}{2} y_1 = 7 \quad \Rightarrow \quad y_3 = 6 - 2 y_1 + 3 y_2 = 23.$$

Instead of the original system now solve

$$\begin{bmatrix} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 7 \end{bmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \\ 23 \end{pmatrix}.$$

This is exactly the system we are left with after the matrix  $\mathbf{A}$  was reduced to echelon form. This should illustrated that the LR factorization is a clever way to formulate the algorithm of Gauss.  $\diamond$

### Implementation in Octave or MATLAB

It is rather straightforward to implement the above algorithm in *Octave/MATLAB*.

#### LRtest.m

```
function [L,R] = LRtest(A)
% [L,R] = LRtest(A) if A is a square matrix
% performs the LR decomposition of the matrix A
% !!!!!!! NO PIVOTING IS DONE !!!!!!!
% this is for instructional purposes only
[n,n] = size(A);
R = zeros(n,n); L = eye(n);

for k = 1:n
    R(k,k:n) = A(k,k:n);
    if ( R(k,k) == 0) error ("LRtest: division by 0") endif
    % divide numbers in k-th column, below the diagonal by A(k,k)
    L(k+1:n,k) = A(k+1:n,k)/R(k,k);
    % apply the row operations to A
    for j = k+1:n
        A(j,k+1:n) = A(j,k+1:n) - A(k,k+1:n)*L(j,k);
    end%for
end%for
```

The only purpose of the above code is to help the reader to understand the algorithm. It should **never** be used to solve a real problem.

- No pivoting is done and thus the code might fail on perfectly solvable problems. Running this code will also lead to unnecessary rounding errors.
- The code is not memory efficient at all. It keeps copies of 3 full size matrices around.

There are considerably better implementations, based on the above ideas. The code can be tested using the model matrix  $\mathbf{A}_n$  from Section 2.3.1.

```
n = 5;
h = 1/(n+1)

A = diag(2*ones(n,1)) -diag(ones(n-1,1),1) -diag(ones(n-1,1),-1);
A = (n+1)^2*A;
[L,R] = LRtest(A)
```

leading to the results

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5 & 1 & 0 & 0 & 0 \\ 0 & -0.667 & 1 & 0 & 0 \\ 0 & 0 & -0.75 & 1 & 0 \\ 0 & 0 & 0 & -0.8 & 1 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} 72 & -36 & 0 & 0 & 0 \\ 0 & 54 & -36 & 0 & 0 \\ 0 & 0 & 48 & -36 & 0 \\ 0 & 0 & 0 & 45 & -36 \\ 0 & 0 & 0 & 0 & 43.2 \end{bmatrix}.$$

Observe that the triangular matrices  $\mathbf{L}$  and  $\mathbf{R}$  have all nonzero entries on the diagonal and the first off-diagonal.

### 2.4.2 LR Factorization and Elementary Matrices

In this section we show a different notation for the LR factorization, using elementary matrices. This notation can be useful when applying the factorization to small matrices. In this subsection no new results are introduced, just a different notation for the LR factorization.

By definition an elementary matrix is generated by applying one row or column operation on the identity matrix. Thus their inverses are easy to construct. Consider the following examples.

1. Multiply the second row by 7.

$$\mathbf{E} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{E}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{7} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Add the 3 times the first row to the third row.

$$\mathbf{E} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{E}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix}$$

Applying row and column operations to matrices can be described by multiplications with elementary matrices, from the left or the right.

- Multiplying a matrix  $\mathbf{A}$  by an elementary matrix from the left has the same effect as applying the row operation to the matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{3}{2} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 4 & 9 & 2 \end{bmatrix}$$

- Multiplying a matrix  $\mathbf{A}$  by an elementary matrix from the right has the same effect as applying the column operation to the matrix.

$$\begin{bmatrix} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 14 & 6 & 2 \\ -19 & -8 & 0 \\ 22 & 9 & 2 \end{bmatrix}$$

Using multiple row operations we can perform an LR factorization. We use again the same example as before. The row operations to be applied are

1. Add  $\frac{3}{2}$  times the first row to the second row.
2. Subtract 2 times the first row from the third row.
3. Add 3 times the second row to the third row.

These operations are visible on the left in Figure 2.7. On the right find the corresponding elementary matrices.

$$\begin{array}{c}
 \left[ \begin{array}{ccc} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{array} \right] \quad R_2 \leftarrow R_2 + \frac{3}{2} R_1 \\
 \downarrow \qquad \qquad E_1 = \left[ \begin{array}{ccc} 1 & 0 & 0 \\ +\frac{3}{2} & 1 & 0 \\ 0 & 0 & 1 \end{array} \right], \quad E_1^{-1} = \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -\frac{3}{2} & 1 & 0 \\ 0 & 0 & 1 \end{array} \right] \\
 \left[ \begin{array}{ccc} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 4 & 9 & 2 \end{array} \right] \quad R_3 \leftarrow R_3 - 2 R_1 \\
 \downarrow \qquad \qquad E_2 = \left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{array} \right], \quad E_2^{-1} = \left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ +2 & 0 & 1 \end{array} \right] \\
 \left[ \begin{array}{ccc} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 0 & -3 & -2 \end{array} \right] \quad R_3 \leftarrow R_3 + 3 R_2 \\
 \downarrow \qquad \qquad E_3 = \left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & +3 & 1 \end{array} \right], \quad E_3^{-1} = \left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{array} \right] \\
 \left[ \begin{array}{ccc} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 7 \end{array} \right]
 \end{array}$$

Figure 2.7: LR factorization, using elementary matrices

The row operations from Figure 2.7 are used to construct the LR factorization. Start with  $\mathbf{A} = \mathbb{I} \cdot \mathbf{A}$  and use the elementary matrices for row and column operations.

Observe that we apply row operations to transform the matrix on the right to upper echelon form. The matrix on the left keeps track of the operations to be applied.

$$\begin{aligned}
\left[ \begin{array}{ccc} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{array} \right] &= \left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right] E_1^{-1} \cdot E_1 \left[ \begin{array}{ccc} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{array} \right] \\
&= \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -\frac{3}{2} & 1 & 0 \\ 0 & 0 & 1 \end{array} \right] E_2^{-1} \cdot E_2 \left[ \begin{array}{ccc} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 4 & 9 & 2 \end{array} \right] \\
&= \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -\frac{3}{2} & 1 & 0 \\ +2 & 0 & 1 \end{array} \right] E_3^{-1} \cdot E_3 \left[ \begin{array}{ccc} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 0 & -3 & -2 \end{array} \right] \\
&= \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -\frac{3}{2} & 1 & 0 \\ +2 & -3 & 1 \end{array} \right] \cdot \left[ \begin{array}{ccc} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 7 \end{array} \right]
\end{aligned}$$

Thus we constructed the LR factorization of the matrix  $\mathbf{A}$ .

$$\left[ \begin{array}{ccc} 2 & 6 & 2 \\ -3 & -8 & 0 \\ 4 & 9 & 2 \end{array} \right] = \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -\frac{3}{2} & 1 & 0 \\ +2 & -3 & 1 \end{array} \right] \cdot \left[ \begin{array}{ccc} 2 & 6 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 7 \end{array} \right] = \mathbf{L} \cdot \mathbf{R}$$

## 2.5 The Condition Number of a Matrix, Matrix and Vector Norms

### 2.5.1 Vector Norms and Matrix Norms

With a norm of a vector we usually associate its geometric length, but it is sometimes useful to use different norms. Thus we present three different norms used for the analysis of matrix computations and start out with the general definition of a norm.

**2-7 Definition :** A function is called a **norm** if for all vectors  $\vec{x}, \vec{y} \in \mathbb{R}^n$  and scalars  $\alpha \in \mathbb{R}$  the following properties are satisfied:

$$\begin{aligned}
\|\vec{x}\| &\geq 0 \quad \text{and} \quad \|\vec{x}\| = 0 \iff \vec{x} = \vec{0} \\
\|\vec{x} + \vec{y}\| &\leq \|\vec{x}\| + \|\vec{y}\| \\
\|\alpha \vec{x}\| &= |\alpha| \|\vec{x}\|
\end{aligned}$$

**2-8 Example :** It is an exercise to verify that the following three norms satisfy the above properties:

$$\begin{aligned}
\|\vec{x}\| &= \|\vec{x}\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2} = (x_1^2 + x_2^2 + \dots + x_n^2)^{1/2} = \sqrt{\langle \vec{x}, \vec{x} \rangle} \\
\|\vec{x}\|_1 &= \sum_{i=1}^n |x_i| = |x_1| + |x_2| + \dots + |x_n| \\
\|\vec{x}\|_\infty &= \max_{1 \leq i \leq n} |x_i|
\end{aligned}$$

◊

On a given vector space one can have multiple norms, e.g.  $\|\vec{x}\|_A$  and  $\|\vec{x}\|_B$ . Those norms are said to be **equivalent** if there exist positive constants  $c_1$  and  $c_2$  such that

$$c_1 \|\vec{x}\|_A \leq \|\vec{x}\|_B \leq c_2 \|\vec{x}\|_A \quad \text{for all } \vec{x} .$$

On the vector space  $\mathbb{R}^n$  all norms are equivalent and one may verify the following inequalities. If we have information on one of the possible norms of a vector  $\vec{x}$  we have some information on the size of the other norms too.

**2–9 Result :** For all vectors  $\vec{x} \in \mathbb{R}^n$  we have

$$\begin{aligned} \|\vec{x}\|_2 &\leq \|\vec{x}\|_1 \leq \sqrt{n} \|\vec{x}\|_2, \\ \|\vec{x}\|_\infty &\leq \|\vec{x}\|_2 \leq \sqrt{n} \|\vec{x}\|_\infty, \\ \|\vec{x}\|_\infty &\leq \|\vec{x}\|_1 \leq n \|\vec{x}\|_\infty. \end{aligned}$$

◊

**Proof :** For the interested reader some details of the computations are shown here.

$$\begin{aligned} \|\vec{x}\|_2^2 &= \sum_{i=1}^n x_i^2 \leq \left( \sum_{i=1}^n |x_i| \right)^2 = \|\vec{x}\|_1^2 \\ \|\vec{x}\|_1 &= \sum_{i=1}^n |x_i| = \langle \vec{\mathbb{I}}, \vec{x} \rangle \leq \|\vec{\mathbb{I}}\|_2 \cdot \|\vec{x}\|_2 = \sqrt{n} \|\vec{x}\|_2 \\ \|\vec{x}\|_\infty &= \max\{|x_i|\} = \sqrt{\max\{x_i^2\}} \leq \sqrt{\sum_{i=1}^n x_i^2} \leq \sqrt{n \max\{x_i^2\}} = \sqrt{n} \|\vec{x}\|_\infty \\ \|\vec{x}\|_\infty &= \max |x_i| \leq \sum_{i=1}^n |x_i| = \|\vec{x}\|_1 \leq n \max |x_i| = n \|\vec{x}\|_\infty \end{aligned}$$

□

A **matrix norm** of a matrix  $\mathbf{A}$  should give us some information on the length of the vector  $\vec{y} = \mathbf{A} \cdot \vec{x}$ , based on the length of  $\vec{x}$ . Thus we require the basic inequality

$$\|\mathbf{A} \cdot \vec{x}\| \leq \|\mathbf{A}\| \|\vec{x}\| \quad \text{for all } \vec{x} \in \mathbb{R}^n$$

i.e. the norm  $\|\mathbf{A}\|$  is the largest occurring amplification factor when multiplying a vector with this matrix  $\mathbf{A}$ .

**2–10 Definition :** For each vector norm there is a resulting matrix norm defined by

$$\begin{aligned} \|\mathbf{A}\| &= \max_{\vec{x} \neq \vec{0}} \frac{\|\mathbf{A} \cdot \vec{x}\|}{\|\vec{x}\|} = \max_{\|\vec{x}\|=1} \|\mathbf{A} \cdot \vec{x}\| \\ \|\mathbf{A}\|_2 &= \max_{\vec{x} \neq \vec{0}} \frac{\|\mathbf{A} \cdot \vec{x}\|_2}{\|\vec{x}\|_2} = \max_{\|\vec{x}\|_2=1} \|\mathbf{A} \cdot \vec{x}\|_2 \\ \|\mathbf{A}\|_1 &= \max_{\vec{x} \neq \vec{0}} \frac{\|\mathbf{A} \cdot \vec{x}\|_1}{\|\vec{x}\|_1} = \max_{\|\vec{x}\|_1=1} \|\mathbf{A} \cdot \vec{x}\|_1 \\ \|\mathbf{A}\|_\infty &= \max_{\vec{x} \neq \vec{0}} \frac{\|\mathbf{A} \cdot \vec{x}\|_\infty}{\|\vec{x}\|_\infty} = \max_{\|\vec{x}\|_\infty=1} \|\mathbf{A} \cdot \vec{x}\|_\infty \end{aligned}$$

where the maximum is taken over all  $\vec{x} \in \mathbb{R}^n$  with  $\vec{x} \neq \vec{0}$ , or equivalently over all vectors with  $\|\vec{x}\| = 1$ .

One may verify that all of the above norms satisfy

$$\begin{aligned}\|\mathbf{A}\| &\geq 0 \quad \text{and} \quad \|\mathbf{A}\| = 0 \iff \mathbf{A} = \mathbf{0} \\ \|\mathbf{A} + \mathbf{B}\| &\leq \|\mathbf{A}\| + \|\mathbf{B}\| \\ \|\alpha \mathbf{A}\| &= |\alpha| \|\mathbf{A}\|\end{aligned}$$

**2-11 Example :** For a given  $m \times n$  matrix  $\mathbf{A}$  the two norms  $\|\mathbf{A}\|_1$  and  $\|\mathbf{A}\|_\infty$  are rather easy to compute:

$$\begin{aligned}\|\mathbf{A}\|_1 &= \max_{1 \leq j \leq n} \left( \sum_{i=1}^m |a_{i,j}| \right) = \text{maximal column sum} \\ \|\mathbf{A}\|_\infty &= \max_{1 \leq i \leq m} \left( \sum_{j=1}^n |a_{i,j}| \right) = \text{maximal row sum}\end{aligned}$$

◊ Ex 2.4, 2.5,  
2.6

**Proof :** We examine  $\|\mathbf{A}\|_\infty$  first. Assume that the maximum value of  $\|\vec{y}\|_\infty = \|\mathbf{A} \cdot \vec{x}\|_\infty$  is attained for the vector  $\vec{x}$  with  $\|\vec{x}\|_\infty = 1$ . Then all components have to be  $x_j = \pm 1$ , otherwise we could increase  $\|\vec{y}\|_\infty$  without changing  $\|\vec{x}\|_\infty$ . If the maximal value of  $|y_i|$  is attained at the component with index  $p$  then the matrix multiplication

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{pmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,n} \\ \vdots & & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \dots & a_{m,n} \end{bmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix}$$

implies

$$y_p = \sum_{j=1}^n a_{p,j} x_j = \sum_{j=1}^n a_{p,j} (\pm 1) = \sum_{j=1}^n |a_{p,j}|.$$

This leads to the claimed result

$$\|\mathbf{A}\|_\infty = \|\vec{y}\|_\infty = \max_{1 \leq i \leq m} \left( \sum_{j=1}^n |a_{i,j}| \right).$$

To determine the norm  $\|\mathbf{A}\|_1$  examine

$$\begin{aligned}\|\vec{y}\|_1 &= \sum_{i=1}^m |y_i| = \sum_{i=1}^m \left| \sum_{j=1}^n a_{i,j} x_j \right| \leq \sum_{j=1}^n |x_j| \sum_{i=1}^m |a_{i,j}| \\ &\leq \sum_{j=1}^n |x_j| \left( \max_{1 \leq k \leq n} \sum_{i=1}^m |a_{i,k}| \right) = \left( \max_{1 \leq k \leq n} \sum_{i=1}^m |a_{i,k}| \right) \sum_{j=1}^n |x_j| \\ &= \left( \max_{1 \leq k \leq n} \sum_{i=1}^m |a_{i,k}| \right) \|\vec{x}\|_1.\end{aligned}$$

If the above column maximum is attained in column  $k$  choose  $x_k = 1$  and all other components of  $\vec{x}$  are set to zero. For this special vector then find

$$\|\mathbf{A} \cdot \vec{x}\|_1 = \sum_{i=1}^m |a_{i,k} \cdot 1| = \sum_{i=1}^m |a_{i,k}| = \|\mathbf{A}\|_1.$$

□

Unfortunately the most important norm  $\|\mathbf{A}\| = \|\mathbf{A}\|_2$  is not easily computed. But for  $n \times m$  matrices  $\mathbf{A}$  we have the following inequalities.

$$\begin{aligned} \frac{1}{\sqrt{n}} \|\mathbf{A}\|_\infty &\leq \|\mathbf{A}\|_2 \leq \sqrt{m} \|\mathbf{A}\|_\infty \\ \frac{1}{\sqrt{m}} \|\mathbf{A}\|_1 &\leq \|\mathbf{A}\|_2 \leq \sqrt{n} \|\mathbf{A}\|_1 \end{aligned}$$

and thus we might be able to estimate the size of  $\|\mathbf{A}\|_2$  with the help of the other norms. The proofs of the above statements are based on Result 2–9. A precise result on the 2-norm is given in [GoluVanLoan96, Theorem 3.2.1]. The result is stated here for sake of completeness.

**2–12 Result :** For any  $m \times n$  matrix  $\mathbf{A}$  there exists a vector  $\vec{z} \in \mathbb{R}^n$  with  $\|\vec{z}\|_2 = 1$  such that  $\mathbf{A}^T \mathbf{A} \cdot \vec{z} = \mu^2 \vec{z}$  and  $\|\mathbf{A}\|_2 = \mu$ . Since  $\mathbf{A}^T \mathbf{A}$  is symmetric and positive definite we know that all eigenvalues are real and positive. Thus  $\|\mathbf{A}\|_2$  is the square root of the largest eigenvalue of the  $n \times n$  matrix  $\mathbf{A}^T \mathbf{A}$ .  $\diamond$

We might attempt to compute all eigenvalues of the symmetric matrix  $\mathbf{A}^T \mathbf{A}$  and then compute the square root of the largest eigenvalue. Since it is computationally expensive to determine all eigenvalues the task remains difficult. There are special algorithms (power method) to estimate the largest eigenvalue of  $\mathbf{A}^T \mathbf{A}$ , used in the MATLAB/Octave functions `normest()`, `condeest()` and `eigs()`, see Section 3.2.

In Result 3–22 (page 134) find few facts on symmetric matrices and the corresponding eigenvalues and eigenvectors. Using this result one can verify that for real, symmetric matrices

$$\mathbf{A} \text{ symmetric} \implies \|\mathbf{A}\|_2 = \max_i |\lambda_i|.$$

This can also be confirmed directly. For a given symmetric matrix  $\mathbf{A}$  let  $\vec{e}_j$  be the basis of normalized eigenvectors. Then we write the arbitrary vector  $\vec{x}$  as a linear combination of the eigenvectors  $\vec{e}_j$ .

$$\vec{x} = \sum_{j=1}^n c_j \vec{e}_j \implies \mathbf{A} \cdot \vec{x} = \sum_{j=1}^n c_j \lambda_j \vec{e}_j$$

Use the orthogonality of the eigenvectors  $\vec{e}_i$  to conclude

$$\|\vec{x}\|^2 = \langle \vec{x}, \vec{x} \rangle = \left\langle \sum_{i=1}^n c_i \vec{e}_i, \sum_{j=1}^n c_j \vec{e}_j \right\rangle = \sum_{i=1}^n |c_i|^2 \langle \vec{e}_i, \vec{e}_i \rangle = \sum_{i=1}^n |c_i|^2.$$

If  $\|\vec{x}\| = 1$  then

$$1 = \|\vec{x}\|^2 = \sum_{j=1}^n |c_j|^2 \implies \|\mathbf{A} \cdot \vec{x}\|^2 = \sum_{j=1}^n |\lambda_j|^2 |c_j|^2$$

and the largest possible value will be attained if  $c_n = 1$  and all other  $c_j = 0$ , i.e. the vector  $\vec{x}$  points in the direction of the eigenvector with the largest eigenvalue.

Similarly we can determine the norm of the inverse matrix  $\mathbf{A}^{-1}$ . Since

$$\vec{x} = \sum_{j=1}^n c_j \vec{e}_j \implies \mathbf{A}^{-1} \cdot \vec{x} = \sum_{j=1}^n c_j \frac{1}{\lambda_j} \vec{e}_j$$

we find for  $\|\vec{x}\| = 1$

$$\|\mathbf{A}^{-1} \cdot \vec{x}\|^2 = \sum_{j=1}^n \frac{1}{|\lambda_j|^2} |c_j|^2$$

and the largest possible value will be attained if the vector  $\vec{x}$  points in the direction of the eigenvector with the smallest absolute value of the eigenvalue, i.e.

$$\|\mathbf{A}^{-1}\|_2 = \frac{1}{\min_j |\lambda_j|}.$$

**2–13 Example :** For the matrix  $\mathbf{A}_n$  in section 2.3.1 (page 32) the matrix norms are easily computed.

(a) The maximal column and rows sums are given by  $\frac{1}{h^2} (1 + 2 + 1)$  and thus

$$\|\mathbf{A}_n\|_1 = \|\mathbf{A}_n\|_\infty = \frac{4}{h^2} \approx 4n^2.$$

(b) The largest eigenvalue  $\lambda_n \approx 4n^2$  implies

$$\|\mathbf{A}_n\|_2 \approx 4n^2.$$

(c) Since the smallest eigenvalue is given by  $\lambda_1 \approx \pi^2$  we find

$$\|\mathbf{A}_n^{-1}\|_2 = \frac{1}{|\lambda_1|} \approx \frac{1}{\pi^2}.$$

In this case the three matrix norms are approximately equal, at least for large values on  $n$ .  $\diamond$

**2–14 Example : Orthogonal matrix**

A matrix  $\mathbf{Q} \in \mathbb{M}^{n \times n}$  is called orthogonal if  $\mathbf{Q}^T \mathbf{Q} = \mathbb{I}_n$ . Since a left inverse matrix is a right inverse we also have  $\mathbf{Q} \mathbf{Q}^T = \mathbb{I}_n$ . The norm of an orthogonal matrix  $\mathbf{Q}$  equals 1, i.e.  $\|\mathbf{Q}\|_2 = 1$ . To verify this use

$$\|\mathbf{Q}\vec{x}\|_2^2 = \langle \mathbf{Q}\vec{x}, \mathbf{Q}\vec{x} \rangle = \langle \vec{x}, \mathbf{Q}^T \mathbf{Q}\vec{x} \rangle = \langle \vec{x}, \vec{x} \rangle = \|\vec{x}\|_2^2.$$

Similary find that  $\|\mathbf{Q}^{-1}\|_2 = 1$ . To verify this use  $(\mathbf{Q}^{-1})^T = (\mathbf{Q}^T)^{-1}$  and

$$\|\mathbf{Q}^{-1}\vec{x}\|_2^2 = \langle \mathbf{Q}^{-1}\vec{x}, \mathbf{Q}^{-1}\vec{x} \rangle = \langle \vec{x}, (\mathbf{Q}^{-1})^T \mathbf{Q}^{-1}\vec{x} \rangle = \langle \vec{x}, \vec{x} \rangle = \|\vec{x}\|_2^2.$$

$\diamond$

**2–15 Observation : Frobenius norm**

The Frobenius norm of a matrix  $\mathbf{A} \in \mathbb{M}^{n \times m}$  is defined by

$$\|\mathbf{A}\|_F^2 := \sum_{i=1}^n \sum_{j=1}^m a_{i,j}^2$$

and is **not** generated by a corresponding vector norm. It is a matter of simple algebra to verify that

$$\|\mathbf{A}\|_F^2 = \text{trace}(\mathbf{A}^T \mathbf{A}).$$

Since multiplying a vector by an orthogonal matrix  $\mathbf{Q}$  does not change the length of the vector the Frobenius norm is invariant under orthogonal transformations, i.e. for orthogonal matrices  $\mathbf{Q}_l$  and  $\mathbf{Q}_r$  conclude  $\|\mathbf{A}\|_F = \|\mathbf{Q}_l \mathbf{A} \mathbf{Q}_r\|_F$ . Using the SVD (see Section 3.2.6, page 149) with the singular values  $\sigma_i$  leads thus to

$$\|\mathbf{A}\|_F^2 = \sum_{i=1}^{\min\{n,m\}} \sigma_i^2.$$

Since the 2-norm  $\|\mathbf{A}\|_2$  is invariant under orthogonal transformations too, obtain  $\|\mathbf{A}\|_2^2 = \max_i \{\sigma_i^2\}$  and hence

$$\frac{1}{\sqrt{\min\{n, m\}}} \|\mathbf{A}\|_F \leq \|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{\min\{n, m\}} \|\mathbf{A}\|_2.$$

Thus the Frobenius norm (easy to compute) allows to estimate the 2-norm (difficult to compute).  $\diamond$

## 2.5.2 The Condition Number of a Matrix

### Condition number for a matrix vector multiplication

Compare the result of  $\vec{y} = \mathbf{A} \cdot \vec{x}$  with a slightly perturbed result  $\vec{y}_p = \mathbf{A} \cdot \vec{x}_p$ . Then we want to compare the

$$\text{relative error in } \vec{x} = \frac{\|\vec{x}_p - \vec{x}\|}{\|\vec{x}\|}$$

with the

$$\text{relative error in } \vec{y} = \frac{\|\vec{y}_p - \vec{y}\|}{\|\vec{y}\|} = \frac{\|\mathbf{A} \cdot (\vec{x}_p - \vec{x})\|}{\|\mathbf{A} \cdot \vec{x}\|}.$$

The **condition number**  $\kappa$  of the matrix  $\mathbf{A}$  is characterized by the property

$$\frac{\|\vec{y}_p - \vec{y}\|}{\|\vec{y}\|} = \frac{\|\mathbf{A} \cdot (\vec{x}_p - \vec{x})\|}{\|\mathbf{A} \cdot \vec{x}\|} \leq \kappa \frac{\|\vec{x}_p - \vec{x}\|}{\|\vec{x}\|}.$$

As typical example we consider a symmetric, nonsingular matrix  $\mathbf{A}$  with eigenvalues

$$0 < |\lambda_1| \leq |\lambda_2| \leq |\lambda_3| \leq \dots \leq |\lambda_{n-1}| \leq |\lambda_n|$$

and the vectors  $\vec{x} = \vec{e}_1$  and  $\vec{x}_p = \vec{e}_1 + \varepsilon \vec{e}_n$ . Thus we examine a relative error of  $0 < \varepsilon \ll 1$  in  $\vec{x}$ . Since  $\mathbf{A} \cdot \vec{e}_1 = \lambda_1 \vec{e}_1$  and  $\mathbf{A} \cdot \vec{e}_n = \lambda_n \vec{e}_n$  we find

$$\begin{aligned} \vec{y} &= \mathbf{A} \cdot \vec{x} &= \mathbf{A} \cdot \vec{e}_1 &= \lambda_1 \vec{e}_1 \\ \vec{y}_p &= \mathbf{A} \cdot \vec{x}_p &= \mathbf{A} \cdot (\vec{e}_1 + \varepsilon \vec{e}_n) &= \lambda_1 \vec{e}_1 + \varepsilon \lambda_n \vec{e}_n \\ \frac{\|\vec{y}_p - \vec{y}\|}{\|\vec{y}\|} &= \frac{\|\mathbf{A} \cdot (\vec{e}_1 + \varepsilon \vec{e}_n) - \mathbf{A} \cdot \vec{e}_1\|}{\|\mathbf{A} \cdot \vec{e}_1\|} &= \frac{\|\varepsilon \mathbf{A} \cdot \vec{e}_n\|}{\|\mathbf{A} \cdot \vec{e}_1\|} &= \varepsilon \frac{|\lambda_n|}{|\lambda_1|}. \end{aligned}$$

In the above example the correct vector is multiplied by the smallest possible number ( $\lambda_1$ ) but the error is multiplied by the largest possible number ( $\lambda_n$ ). Thus we examined the worst case scenario.

### Condition number when solving a linear system of equations

For a given vector  $\vec{b}$  compare the solution  $\vec{x}$  of  $\mathbf{A} \cdot \vec{x} = \vec{b}$  with a slightly perturbed result  $\mathbf{A} \cdot \vec{x}_p = \vec{b}_p$ . Then we want to compare the

$$\text{relative error in } \vec{b} = \frac{\|\vec{b}_p - \vec{b}\|}{\|\vec{b}\|}$$

with the

$$\text{relative error in } \vec{x} = \frac{\|\vec{x}_p - \vec{x}\|}{\|\vec{x}\|} = \frac{\|\mathbf{A}^{-1} \cdot (\vec{b}_p - \vec{b})\|}{\|\mathbf{A}^{-1} \cdot \vec{b}\|}.$$

In this situation the **condition number**  $\kappa$  is characterized by the property

$$\frac{\|\vec{x}_p - \vec{x}\|}{\|\vec{x}\|} \leq \kappa \frac{\|\vec{b}_p - \vec{b}\|}{\|\vec{b}\|} = \kappa \frac{\|\mathbf{A} \cdot (\vec{x}_p - \vec{x})\|}{\|\mathbf{A} \cdot \vec{x}\|},$$

i.e. the relative error in the given vector  $\vec{b}$  might at worst be multiplied by the condition number to obtain the relative error in the solution  $\vec{x}$ .

As an example reconsider the above symmetric matrix  $\mathbf{A}$  and use the vectors  $\vec{b} = \vec{e}_n$  and  $\vec{b}_p = \vec{e}_n + \varepsilon \vec{e}_1$ . Thus we examine a relative error of  $0 < \varepsilon \ll 1$  in  $\vec{b}$ .

$$\frac{\|\vec{x}_p - \vec{x}\|}{\|\vec{x}\|} = \frac{\|\mathbf{A}^{-1} \cdot (\vec{e}_n + \varepsilon \vec{e}_1) - \mathbf{A}^{-1} \cdot \vec{e}_n\|}{\|\mathbf{A}^{-1} \cdot \vec{e}_n\|} = \frac{\|\varepsilon \mathbf{A}^{-1} \cdot \vec{e}_1\|}{\|\mathbf{A}^{-1} \cdot \vec{e}_n\|} = \varepsilon \frac{1/|\lambda_1|}{1/|\lambda_n|} \leq \varepsilon \frac{|\lambda_n|}{|\lambda_1|}.$$

In the above example the correct vector is divided by the largest possible number ( $\lambda_n$ ) but the error is divided by the smallest possible number ( $\lambda_1$ ). Thus we examined the worst case scenario.

Based on the above two observation we use  $|\lambda_n| = \|\mathbf{A}\|_2$  and  $\frac{1}{|\lambda_1|} = \|\mathbf{A}^{-1}\|_2$  to conclude

$$\kappa_2(A) = \frac{|\lambda_n|}{|\lambda_1|} = \|\mathbf{A}\|_2 \cdot \|\mathbf{A}^{-1}\|_2$$

for symmetric matrices  $\mathbf{A}$  and if  $\kappa_2 = 10^d$  we might loose  $d$  decimal digits of accuracy when multiplying a vector by  $\mathbf{A}$  or when solving a system of linear equations.

Using the above idea we define the **condition number** for the matrix and the result applies to multiplication by matrices and solving of linear systems systems.

**2–16 Definition :** The condition number  $\kappa(\mathbf{A})$  of a nonsingular square matrix is defined by

Ex 2.7

$$\kappa = \kappa(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$

Obviously the condition number depends on the matrix norm used.

**2–17 Example :**

- Based on Example 2–14 the condition number of an orthogonal matrix  $\mathbf{Q}$  equals 1, using the 2–norm.
- Using the singular value decomposition (SVD) (see equation (3.5) on page 149) and the above idea one can verify that for a real  $n \times n$  matrix  $\mathbf{A}$

$$\kappa_2 = \kappa_2(\mathbf{A}) = \frac{\sigma_1}{\sigma_n} = \frac{\text{largest singular value}}{\text{smallest singular value}}$$

where the singular values are given by  $\sigma_i$ .

- MATLAB/Octave provide the command `condest()` to efficiently compute a good estimate of the condition number.

◇

For any matrix and norm we have  $\kappa(\mathbf{A}) \geq 1$ . If the condition number  $\kappa$  is not too large we speak of a **well conditioned problem**.

For  $n \times n$  matrices we have the following relations between the different condition numbers.

$$\begin{aligned} \frac{1}{n} \kappa_2 &\leq \kappa_1 \leq n \kappa_2 \\ \frac{1}{n} \kappa_\infty &\leq \kappa_2 \leq n \kappa_\infty \end{aligned}$$

The verification is based on Result 2–9.

**2–18 Example :** For the model matrix  $\mathbf{A}_n$  of size  $n \times n$  in Section 2.3.1 (page 32) find

$$\kappa_2 = \frac{\lambda_{max}}{\lambda_{min}} \approx \frac{4n^2}{\pi^2}$$

and for 2D the model matrix  $\mathbf{A}_{nn}$  of size  $n^2 \times n^2$  in Section 2.3.2 (page 34) find the same result

$$\kappa_2 = \frac{\lambda_{max}}{\lambda_{min}} \approx \frac{4n^2}{\pi^2}.$$

◇

Ex 2.1, 2.2,  
2.3, 2.8

### 2.5.3 The Effect of Rounding Errors, Pivoting

Now we want to examine the effects of arithmetic operations and rounding when solving a system of linear equations of the form  $\mathbf{A} \cdot \vec{x} = \vec{b}$ . The main reference for the results in this section is the bible of matrix computations by Gene Golub and Charles van Loan [GoluVanLoan96], or the newer edition [GoluVanLoan13].

**2–19 Result :** *In an ideal situation absolutely no roundoff occurs during the solution process. Only when  $\mathbf{A}$ ,  $\vec{b}$  and  $\vec{x}$  are stored some roundoff will occur. The stored solution  $\hat{\vec{x}}$  satisfies*

$$(\mathbf{A} + \mathbf{E}) \cdot \hat{\vec{x}} = \vec{b} + \vec{e} \quad \text{with} \quad \|\mathbf{E}\|_\infty \leq \mathbf{u} \|\mathbf{A}\|_\infty \quad \text{and} \quad \|\vec{e}\|_\infty \leq \mathbf{u} \|\vec{b}\|_\infty.$$

Thus  $\hat{\vec{x}}$  solves a nearby system exactly. If now  $\mathbf{u} \kappa_\infty(\mathbf{A}) \leq 1/2$  then one can show that

$$\|\hat{\vec{x}} - \vec{x}\|_\infty \leq 4 \mathbf{u} \kappa_\infty(\mathbf{A}) \|\vec{x}\|_\infty.$$

The above bounds are the best possible. ◇

As a consequence of the above result we can not expect relative errors smaller than  $\kappa \mathbf{u}$  for any kind of clever algorithm to solve linear systems of equations. The goal has to be to achieve this accuracy.

**2–20 Definition :** For the following results we use some special, convenient notations:

$$\begin{aligned} (\mathbf{A})_{i,j} = a_{i,j} &\implies |\mathbf{A}|_{i,j} = |a_{i,j}| \\ \mathbf{A} \leq \mathbf{B} &\iff a_{i,j} \leq b_{i,j} \quad \text{for all indices } i \text{ and } j \end{aligned}.$$

The absolute value and the comparison operator are applied to each entry in the matrix.

The following theorem ([GoluVanLoan96, Theorem 3.3.2]) keeps track of the rounding errors in the back substitution process, i.e. when solving the triangular systems. The proof is considerably beyond the scope of these notes.

**2–21 Theorem :** *Let  $\hat{\mathbf{L}}$  and  $\hat{\mathbf{R}}$  be the computed LR factors of a  $n \times n$  matrix  $\mathbf{A}$ . Suppose  $\hat{\vec{y}}$  is the computed solution of  $\hat{\mathbf{L}} \cdot \vec{y} = \vec{b}$  and  $\hat{\vec{x}}$  the computed solution of  $\hat{\mathbf{R}} \cdot \vec{x} = \hat{\vec{y}}$ . Then*

$$(\mathbf{A} + \mathbf{E}) \cdot \hat{\vec{x}} = \vec{b}$$

with

$$|\mathbf{E}| \leq n \mathbf{u} (3 |\mathbf{A}| + 5 |\mathbf{L}| |\mathbf{R}|) + O(\mathbf{u}^2), \tag{2.2}$$

where  $|\mathbf{L}| |\mathbf{R}|$  is a matrix multiplication. For all practical purposes we may ignore the term  $O(\mathbf{u}^2)$ , since with  $\mathbf{u} \approx 10^{-16}$  we have  $\mathbf{u}^2 \approx 10^{-32}$ . ◇

This result implies that we find exact solutions of a perturbed system  $\mathbf{A} + \mathbf{E}$ . This is called **backward stability**.

The above result shows that large values in the triangular matrices  $\mathbf{L}$  and  $\mathbf{R}$  should be avoided whenever possible. Unfortunately we can obtain large numbers during the factorization, even for well-conditioned matrices, as shown by the following example.

**2–22 Example :** For a small, positive number  $\varepsilon$  the matrix

$$\mathbf{A} = \begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix}$$

is well-conditioned. When applying the LR factorization obtain

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{bmatrix} \cdot \begin{bmatrix} \varepsilon & 1 \\ 0 & \frac{-1}{\varepsilon} \end{bmatrix} = \mathbf{L} \cdot \mathbf{R}.$$

If  $\varepsilon > 0$  is very close to zero then the numbers in  $\mathbf{L}$  and  $\mathbf{R}$  will be large and lead to

$$|\mathbf{L}| \cdot |\mathbf{R}| = \begin{bmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{bmatrix} \cdot \begin{bmatrix} \varepsilon & 1 \\ 0 & \frac{1}{\varepsilon} \end{bmatrix} = \begin{bmatrix} \varepsilon & 1 \\ 1 & \frac{2}{\varepsilon} \end{bmatrix}$$

and thus one of the entries in the bound in Theorem 2–21 is large. Thus the error in the result might be unnecessary large. This elementary, but typical example illustrates that **pivoting is necessary**.  $\diamond$

The correct method to avoid the above problem is **pivoting**. In the LR factorization on page 38 try to factor the submatrices  $\mathbf{A}_{n-k} = \mathbf{L}_{n-k} \cdot \mathbf{R}_{n-k}$ . In the unmodified algorithm the top left number in  $\mathbf{A}_{n-k}$  is used as pivot element. Before performing a next step in the LR factorization it is better to **exchange** rows (equations) and possibly rows (variables) to avoid divisions by small numbers. There are two possible strategies:

- **partial pivoting:**

Choose the largest absolute number in the first column of  $\mathbf{A}_{n-k}$ . Exchange equations for this to become the top left number.

- **total pivoting:**

Choose the largest absolute number in the submatrix  $\mathbf{A}_{n-k}$ . Exchange equations and renumber variables for this to become the top left number.

The computational effort for total pivoting is considerably higher, since  $(n-k)^2$  numbers have to be searched for the maximal value. The bookkeeping requires considerably more effort, since equations and unknowns have to be rearranged. This additional effort is not compensated by considerably better (more reliable) results. Thus for almost all problems partial pivoting will be used. As a consequence we will only examine partial pivoting.

When using partial pivoting all entries in the left matrix  $\mathbf{L}$  will be smaller than 1 and thus  $\|\mathbf{L}\|_\infty \leq n$ . This leads to an improved error estimate (2.2) in Theorem 2–21. For details see [GoluVanLoan96, §3.4.6].

$$\|\mathbf{E}\|_\infty \leq n \mathbf{u} (3 \|\mathbf{A}\|_\infty + 5 n \|\mathbf{R}\|_\infty) + O(\mathbf{u}^2)$$

The formulation for factorization has to be modified slightly and supplemented with a **permutation matrix  $\mathbf{P}$** .

**2–23 Result :** A square matrix  $\mathbf{P}$  is a **permutation matrix** if each row and each column of  $\mathbf{P}$  contains exactly one number 1 and all other entries are zero. Multiplying a matrix or vector from the left by a permutation matrix has the effect of row permutations:

$$p_{i,j} = 1 \iff \text{the old row } j \text{ will turn into the new row } i$$

As a consequence we find  $\mathbf{P}^T = \mathbf{P}^{-1}$ .  $\diamond$

**2-24 Example :** The effects of permutation matrices are best illustrated by a few elementary examples

•

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 5 & 6 \\ 1 & 2 \\ 3 & 4 \end{bmatrix}$$

•

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 1 \end{pmatrix}$$

◊

For a given matrix  $\mathbf{A}$  we seek triangular matrices  $\mathbf{L}$  and  $\mathbf{R}$  and a permutation matrix  $\mathbf{P}$ , such that

$$\mathbf{P} \cdot \mathbf{A} = \mathbf{L} \cdot \mathbf{R}.$$

If we now want to solve the system  $\mathbf{A} \cdot \vec{x} = \vec{b}$  using the factorization replace the original system by two linear systems with triangular matrices.

$$\mathbf{A} \cdot \vec{x} = \vec{b} \iff \mathbf{P} \mathbf{A} \cdot \vec{x} = \mathbf{P} \vec{b} \iff \mathbf{L} \mathbf{R} \cdot \vec{x} = \mathbf{P} \vec{b} \iff \begin{cases} \mathbf{L} \vec{y} = \mathbf{P} \cdot \vec{b} \\ \mathbf{R} \vec{x} = \vec{y} \end{cases}.$$

Example 2-6 has to be modified accordingly, i.e. the permutation given by  $\mathbf{P}$  is applied to the right hand side.

**2-25 Example :** To solve the system

$$\mathbf{A} \cdot \vec{x} = \begin{bmatrix} 3/2 & 4 & -7/2 \\ 3 & 2 & 1 \\ 0 & -1 & 25/3 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} +1 \\ 0 \\ -1 \end{pmatrix}$$

we use the factorization

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3/2 & 4 & -7/2 \\ 3 & 2 & 1 \\ 0 & -1 & 25/3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & -1/3 & 1 \end{bmatrix} \begin{bmatrix} 3 & 2 & 1 \\ 0 & 3 & -4 \\ 0 & 0 & 7 \end{bmatrix}.$$

Then the system can be solved using two triangular systems. First solve from top to bottom

$$\begin{aligned} \mathbf{L} \vec{y} &= \mathbf{P} \vec{b} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & -1/3 & 1 \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} &= \begin{pmatrix} 0 \\ +1 \\ -1 \end{pmatrix} \end{aligned}$$

and then

$$\mathbf{R} \vec{x} = \vec{y}$$

$$\begin{bmatrix} 3 & 2 & 1 \\ 0 & 3 & -4 \\ 0 & 0 & 7 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

from bottom to top.  $\diamond$

Any good numerical library has an implementation of the LR (or LU) factorization with partial pivoting built in. As an example consider the help provided by *Octave* on the command `lu()`.

#### Octave

```
help lu
--> lu is a built-in function

Compute the LU decomposition of A.
If A is full subroutines from LAPACK are used and if A is sparse
then UMFPACK is used.
The result is returned in a permuted form, according to the optional return
value P. For example, given the matrix 'a = [1, 2; 3, 4]',
```

```
[l, u, p] = lu (A)
returns
l = 1.00000 0.00000
     0.33333 1.00000

u = 3.00000 4.00000
     0.00000 0.66667

p = 0 1
    1 0
```

The matrix is not required to be square.

Using this factorization one can solve systems of linear equations  $\mathbf{A}\vec{x} = \vec{b}$  for  $\vec{x}$ .

#### Octave

```
A = randn(3,3); % generate a random matrix
b = rand(3,1);
x1 = A\b; % a first solution
[L,U,P] = lu(A); % compute the LU factorization with pivoting
x2 = U\ (L\ (P*b)) % the solution with the help of the factorization
DifferenceSol = norm(x1-x2) % display the differences, should be zero
```

The first solution is generated with the help of the backslash operator `\`. Internally *Octave/MATLAB* use the LU (same as LR) factorization. Computing and storing the  $\mathbf{L}$ ,  $\mathbf{U}$  and  $\mathbf{P}$  matrices is only useful when multiple systems with the same matrix  $\mathbf{A}$  have to be solved. The computational effort to apply the back substitution is considerably smaller than the effort to determine the factorization, at least for general matrices.

## 2.6 Structured Matrices

Many matrices have special properties. They might be symmetric, have the nonzero entries concentrated in a narrow band around the diagonal or might have very few nonzero entries. The model matrices  $\mathbf{A}_n$  and  $\mathbf{A}_{nn}$  in Section 2.3 exhibit those properties. The basic LR factorization can be adapted to take advantage of these properties.

### 2.6.1 Symmetric Matrices, Algorithm of Cholesky

If a matrix is symmetric then we might seek a factorization  $\mathbf{A} = \mathbf{L} \cdot \mathbf{R}$  with  $\mathbf{L} = \mathbf{R}^T$ . This will lead to the classical Cholesky factorization  $\mathbf{A} = \mathbf{R}^T \cdot \mathbf{R}$ . This approach is given as an exercise. This algorithm will require the computation of square roots, which is often undesirable. Observe that on the diagonal of the factor  $\mathbf{R}$  of the standard Cholesky factorization you will find numbers different from 1, while the modified factorization below requires values of 1 along the diagonal.

Instead we examine a slight modification<sup>7</sup>. We seek a diagonal matrix  $\mathbf{D}$  and an upper triangular matrix  $\mathbf{R}$  with numbers 1 on the diagonal such that

$$\mathbf{A} = \mathbf{R}^T \cdot \mathbf{D} \cdot \mathbf{R}$$

The approach is adapted from Section 2.4.1, using block matrices again. Using standard matrix multiplications we find

$$\begin{aligned} & \left[ \begin{array}{c|cccc} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ \hline a_{1,2} & & & & \\ a_{1,3} & & & & \\ \vdots & & & & \\ a_{1,n} & & & & \end{array} \right] = \mathbf{R}^T \cdot \mathbf{D} \cdot \mathbf{R} = \\ & = \left[ \begin{array}{c|cccc} 1 & 0 & 0 & \dots & 0 \\ \hline r_{1,2} & & & & \\ r_{1,3} & & & & \\ \vdots & & & & \\ r_{1,n} & & & & \end{array} \right] \left[ \begin{array}{c|cccc} d_1 & 0 & 0 & \dots & 0 \\ \hline 0 & & & & \\ 0 & & & & \\ \vdots & & & & \\ 0 & & & & \end{array} \right] \left[ \begin{array}{c|cccc} 1 & r_{1,2} & r_{1,3} & \dots & r_{1,n} \\ \hline 0 & & & & \\ 0 & & & & \\ \vdots & & & & \\ 0 & & & & \end{array} \right] \mathbf{R}_{n-1} \\ & = \left[ \begin{array}{c|cccc} 1 & 0 & 0 & \dots & 0 \\ \hline r_{1,2} & & & & \\ r_{1,3} & & & & \\ \vdots & & & & \\ r_{1,n} & & & & \end{array} \right] \left[ \begin{array}{c|cccc} d_1 & d_1 r_{1,2} & d_1 r_{1,3} & \dots & d_1 r_{1,n} \\ \hline 0 & & & & \\ 0 & & & & \\ \vdots & & & & \\ 0 & & & & \end{array} \right] \mathbf{D}_{n-1} \cdot \mathbf{R}_{n-1} \end{aligned}$$

Now we examine the effects of the last matrix multiplication on the four subsystems. This translates to 4 subsystems.

- Examine the top left block (one single number) in  $\mathbf{A}$ . Obviously we find  $a_{1,1} = d_1$ .
- Examine the bottom left block (column) in  $\mathbf{A}$ .

$$\left[ \begin{array}{c} a_{1,2} \\ a_{1,3} \\ \vdots \\ a_{1,n} \end{array} \right] = \left[ \begin{array}{c} r_{1,2} \\ r_{1,3} \\ \vdots \\ r_{1,n} \end{array} \right] \cdot d_1 \implies r_{1,i} = \frac{a_{1,i}}{d_1} = \frac{a_{1,i}}{a_{1,1}}$$

This operation requires  $(n - 1)$  flops.

<sup>7</sup>This modification is known as modified Cholesky factorization and MATLAB provides the command `ldl()` for this algorithm.

- The top right block (row) in  $\mathbf{A}$  is then already taken care of. It is a transposed copy of the first column.
- Examine the bottom right block in  $\mathbf{A}$ . We need

$$\mathbf{A}_{n-1} = d_1 \begin{bmatrix} r_{1,2} \\ r_{1,3} \\ \vdots \\ r_{1,n} \end{bmatrix} \cdot \begin{bmatrix} r_{1,2} & r_{1,3} & \dots & r_{1,n} \end{bmatrix} + \mathbf{R}_{n-1}^T \cdot \mathbf{D}_{n-1} \cdot \mathbf{R}_{n-1}.$$

For  $2 \leq i, j \leq n$  update the entries in  $\mathbf{A}_{n-1}$  by applying

$$a_{i,j} \quad \longrightarrow \quad a_{i,j} - d_1 r_{1,i} r_{1,j} = a_{i,j} - \frac{a_{1,i} a_{1,j}}{a_{1,1}}.$$

This operation requires  $\frac{1}{2} (n-1)^2$  flops since the matrix is symmetric. Now we are left with the new factorization

$$\tilde{\mathbf{A}}_{n-1} = \mathbf{R}_{n-1}^T \cdot \mathbf{D}_{n-1} \cdot \mathbf{R}_{n-1}$$

with the updated matrix  $\tilde{\mathbf{A}}_{n-1}$ .

- Then restart the process with the reduced problem of size  $(n-1) \times (n-1)$  in the lower right block.

The total number of operations can be estimated by

$$\text{FlopChol} \approx \sum_{k=1}^{n-1} \frac{k^2}{2} \approx \frac{1}{6} n^3.$$

Thus we were able to reduce the number of necessary operations by a factor of 2 compared to the standard LR factorization ( $\text{Flop}_{\text{LR}} \approx \frac{1}{3} n^3$ ).

**2–26 Observation :** Adding multiples of one row to another row in a large matrix can be implemented in parallel on a multicore architecture, as shown in Section 2.2.3. The number of columns has to be considerably larger than the number of CPU cores to be used. ◇

### The algorithm and an implementation in Octave

The above algorithm can be implemented in any programming language. Using a MATLAB/Octave pseudo code one might write.

```

for each row:
    for each row below the current row
        find the factor for the row operation
        do the row operation
        do the column operation
for k = 1:n
    for j = k+1:n
        R(k, j) = A(j, k)/A(k, k);
        A(j, :) = A(j, :) - R(k, j)*A(k, :);
        A(:, j) = A(:, j) - R(k, j)*A(:, k);
    
```

The above may be implemented in MATLAB/Octave.

```

function [R,D] = choleskyDiag(A)
% [R,D] = choleskyDiag(A) if A is a symmetric positive definite matrix
%           returns a upper triangular matrix R and a diagonal matrix D
%           such that A = R'*D*R

% this code can only be used for didactical purposes
% it has some major flaws!

[n,m] = size(A); R = zeros(n);

for k = 1:n-1
    R(k,k) = 1;
    for j = k+1:n
        R(k,j) = A(j,k)/A(k,k);
        A(j,:) = A(j,:) - R(k,j)*A(k,:);
        A(:,j) = A(:,j) - R(k,j)*A(:,k);
    end%for
    R(n,n) = 1;
end%for
D = diag(diag(A));

```

The above code has some serious flaws:

- It does not check for correct size of the input.
- It does not check for possible divisions by 0.
- As we go through the algorithm the coefficients in **R** can replace the coefficients in **A** which will not be used any more. This cuts the memory requirement in half.
- If we do all computations in the upper right part of **A**, we already know that the result in the lower left part has to be the same. Thus we can do only half of the calculations.
- As we already know that the numbers in the diagonal of **R** have to be 1, we do not need to return them. One can use the diagonal of **R** to return the coefficients of the diagonal matrix **D**.

If we implement most<sup>8</sup> of the above points we obtain an improved algorithm, shown below.

#### choleskyM.m

```

function R = choleskyM(A)
% R = choleskyM(A) if A is a symmetric positive definite matrix
%           returns a upper triangular matrix R and a diagonal matrix D
%           such that A = R1'*D*R1
%           R1 has all diagonal entries equal to 1
%           the values of D are returned on the diagonal of R

TOL = 1e-10*max(abs(A(:))); % there certainly are better tests than this!!

[n,m] = size(A);
if (n~=m) error ('choleskyM: matrix has to be square') end%if

for k = 1:n-1
    if ( abs(A(k,k)) <= TOL) error ('choleskyM:might be a singular matrix')
        endif
    for j = k+1:n
        A(j,k) = A(j,k)/A(k,k);
        % row operations only
    end%for
end%for

```

<sup>8</sup>The memory requirements can be made considerably smaller

```

A(j,j:n) = A(j,j:n) - A(j,k)*A(k,j:n);
end%for
end%for
if ( abs(A(n,n)) <= TOL) error ('choleskyM:might be a singular matrix') end%if

% return the lower triangular part of A.
% Transpose it to obtain an upper triangular matrix
R = tril(A)';

```

The above code finds the Cholesky factorization of the matrix, but does not solve a system of linear equations. It has to be supplemented with the corresponding back-substitution algorithm.

```

function x = CholeskySolver(R,b)
% x = choleskySolver(R,b) solves A x = b
%     R has to be generated by R = choleskyM(A)

[n,m] = size(R);
if (n ~= length(b))
    error ("CholeskySovler: matrix and vector do not have same dimension ") endif

% forward substitution of R' y = b
y = zeros(size(b));
y(1) = b(1);
for k = 2:n
    y(k) = b(k);
    for j = 1:k-1 y(k) = y(k) - R(j,k)*y(j);end%for
end%for

% solve diagonal system
for k = 1:n y(k) = y(k)/R(k,k);endfor

% backward substitution of R x = y
x = zeros(size(b));
x(n) = y(n);
for k = 1:n-1
    x(n-k) = y(n-k);
    for j = n-k+1:n x(n-k) = x(n-k) - R(n-k,j)*x(j);end%for
end%for

```

The operation count for the back substitution algorithm is given by

$$\text{Flop}_{\text{Solve}} \approx n^2.$$

For large  $n$  this is small compared to the  $n^3/6$  operations for the factorization. Thus we will only keep track of the computational effort for the factorization.

Now we can solve an exemplary system of linear equations.

```

A = [1 3 -4; 3 11 0; -4 0 10];
R = choleskyM(A)
b = [1; 2; 3];
x = CholeskySolver(R,b)'
-->
R = 1      3      -4
      0      2      6
      0      0     -78

```

$\begin{bmatrix} x = & -1.16667 & 0.50000 & -0.16667 \end{bmatrix}$
---

The result says that

$$\begin{bmatrix} 1 & 3 & -4 \\ 3 & 11 & 0 \\ -4 & 0 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -4 & 6 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -78 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 & -4 \\ 0 & 1 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

and the system

$$\begin{bmatrix} 1 & 3 & -4 \\ 3 & 11 & 0 \\ -4 & 0 & 10 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

is solved by

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \approx \begin{pmatrix} -1.16667 \\ +0.50000 \\ -0.16667 \end{pmatrix}.$$

## 2.6.2 Positive Definite Matrices

In the previous section we completely ignored pivoting. This might lead to unnecessary divisions by numbers close to zero and thus large errors. For a special type of symmetric matrices one can show that pivoting is in fact not necessary to obtain numerical stability.

**2-27 Definition :** A symmetric, real matrix  $\mathbf{A}$  is called **positive definite** if and only if

$$\langle \mathbf{A} \cdot \vec{x}, \vec{x} \rangle = \langle \vec{x}, \mathbf{A} \cdot \vec{x} \rangle > 0 \quad \text{for all } \vec{x} \neq \vec{0}.$$

The matrix is called **positive semidefinite** if and only if

$$\langle \mathbf{A} \cdot \vec{x}, \vec{x} \rangle = \langle \vec{x}, \mathbf{A} \cdot \vec{x} \rangle \geq 0 \quad \text{for all } \vec{x}.$$

**2-28 Example :** To verify that the matrix  $\mathbf{A}_n$  (see page 32) is positive definite we have to show that<sup>9</sup>

$$\langle \vec{u}, \mathbf{A}_n \vec{u} \rangle > 0 \quad \text{for all } \vec{u} \in \mathbb{R}^n \setminus \{\vec{0}\}$$

$$\langle \vec{u}, \mathbf{A}_n \vec{u} \rangle = \frac{1}{h^2} \left\langle \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix}, \begin{pmatrix} 2u_1 - u_2 \\ -u_1 + 2u_2 - u_3 \\ -u_2 + 2u_3 - u_4 \\ \vdots \\ -u_{n-2} + 2u_{n-1} - u_n \\ -u_{n-1} + 2u_n \end{pmatrix} \right\rangle$$

<sup>9</sup>This verification corresponds to the integration by parts for twice differentiable functions  $u(x)$  with boundary conditions  $u(0) = u(1) = 0$ .

$$\int_0^1 u(x) \cdot (-u''(x)) dx = 0 + \int_0^1 u(x)' \cdot u'(x) dx \geq 0$$

$$\begin{aligned}
&= \frac{1}{h^2} \left\langle \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix}, \begin{pmatrix} u_1 - (u_2 - u_1) \\ (u_2 - u_1) - (u_3 - u_2) \\ (u_3 - u_2) - (u_4 - u_3) \\ \vdots \\ (u_{n-1} - u_{n-2}) - (u_n - u_{n-1}) \\ (u_n - u_{n-1}) + u_n \end{pmatrix} \right\rangle \\
&= \frac{1}{h^2} \left\langle \begin{pmatrix} u_1 \\ (u_2 - u_1) \\ (u_3 - u_2) \\ \vdots \\ (u_{n-1} - u_{n-2}) \\ (u_n - u_{n-1}) \end{pmatrix}, \begin{pmatrix} u_1 \\ (u_2 - u_1) \\ (u_3 - u_2) \\ \vdots \\ (u_{n-1} - u_{n-2}) \\ (u_n - u_{n-1}) \end{pmatrix} \right\rangle + \frac{u_n^2}{h^2} \\
&= \frac{1}{h^2} \left( u_1^2 + u_n^2 + \sum_{i=2}^n (u_i - u_{i-1})^2 \right).
\end{aligned}$$

This sum of squares is obviously positive. Only if  $\vec{u} = \vec{0}$  the expression will be zero. Thus the matrix  $\mathbf{A}_n$  is positive definite.  $\diamond$

A positive definite matrix  $\mathbf{A}$  has a few properties that are easy to verify.

**2-29 Result :** If the matrix  $\mathbf{A} = (a_{i,j})_{1 \leq i,j \leq n}$  is positive definite then

- $a_{i,i} > 0$  for  $1 \leq i \leq n$ , i.e. the numbers on the diagonal are positive.
- $\max |a_{i,j}| = \max a_{i,i}$ , i.e. the maximal value has to be on the diagonal.

$\diamond$

**Proof :**

- Choose  $\vec{x} = \vec{e}_i$  and compute  $\langle \vec{e}_i, \mathbf{A} \cdot \vec{e}_i \rangle = a_{i,i} > 0$ . For a  $4 \times 4$  matrix this is illustrated by

$$\langle \mathbf{A} \vec{x}, \vec{x} \rangle = \left\langle \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\rangle = \left\langle \begin{pmatrix} a_{13} \\ a_{23} \\ a_{33} \\ a_{34} \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\rangle = a_{33} > 0.$$

- Assume  $\max |a_{i,j}| = a_{k,l}$  with  $k \neq l$ . Choose  $\vec{x} = \vec{e}_k - \text{sign}(a_{k,l}) \vec{e}_l$  and compute  $\langle \vec{x}, \mathbf{A} \cdot \vec{x} \rangle = a_{k,k} + a_{l,l} - 2|a_{k,l}| \leq 0$ , contradicting positive definiteness. To illustrate the argument we use a small matrix again.

$$\begin{aligned}
\langle \mathbf{A} \vec{x}, \vec{x} \rangle &= \left\langle \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix} \begin{pmatrix} \pm 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \pm 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\rangle \\
&= \left\langle \begin{pmatrix} \pm a_{11} + a_{13} \\ \cdot \\ \pm a_{13} + a_{33} \\ \cdot \end{pmatrix}, \begin{pmatrix} \pm 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\rangle = a_{11} + a_{33} \pm 2a_{13} > 0
\end{aligned}$$

By choosing the correct sign we conclude  $|a_{13}| \leq \frac{1}{2} (a_{11} + a_{33})$  and thus  $|a_{13}|$  can not be larger than both of the other numbers.

□

The above allows to easily detect that a matrix is not positive definite, but it does not contain a criterion to quickly decide that  $\mathbf{A}$  is positive definite.

For a large number of applied problems the resulting matrix has to be positive definite, based on physical or mechanical observations. In many applications the generalized energy of a system is given by

$$\text{energy} = \frac{1}{2} \langle \mathbf{A} \cdot \vec{x}, \vec{x} \rangle .$$

If the object is deformed/modified the energy is often strictly increased. and based on this, the matrix  $\mathbf{A}$  has to be positive definite.

The eigenvalues contain all information about definiteness of a symmetric matrix, but this is not an efficient way to detect positive definite matrices. Finding all eigenvalues is computationally expensive. If the matrix is symmetric and the smallest eigenvalue is positive, then all eigenvalues are positive. For this the MATLAB/Octave function `eigest()` is very useful, see Section 3.2.

**2–30 Result :** *The symmetric matrix  $\mathbf{A}$  is positive definite iff all eigenvalues are strictly positive. The symmetric matrix  $\mathbf{A}$  is positive semidefinite iff all eigenvalues are positive or zero.*

◇ Ex 2.19

**Proof :** This is a direct consequence of the diagonalization result 3–22 (page 134)  $\mathbf{A} = \mathbf{Q} \mathbf{D} \mathbf{Q}^T$ , where the diagonal matrix contains the eigenvalues  $\lambda_j$  along its diagonal. The computation is based on  $\vec{y} = \mathbf{Q}^T \cdot \vec{x}$  and the equation

$$\langle \mathbf{A} \cdot \vec{x}, \vec{x} \rangle = \langle \mathbf{Q} \cdot \mathbf{D} \cdot \mathbf{Q}^T \cdot \vec{x}, \vec{x} \rangle = \langle \mathbf{D} \cdot \mathbf{Q}^T \cdot \vec{x}, \mathbf{Q}^T \cdot \vec{x} \rangle = \langle \mathbf{D} \cdot \vec{y}, \vec{y} \rangle = \sum_j \lambda_j y_i^2 .$$

□

This result is of little help to decide whether a given, large matrix is positive definite or not. Finding all eigenvalues is not an option, as it is computationally expensive. A positive answer can be given using diagonal dominance and reducible matrices, see e.g. [Axel94, §4].

**2–31 Definition :** Consider a symmetric  $n \times n$  matrix  $\mathbf{A}$ .

- $\mathbf{A}$  is called **strictly diagonally dominant** iff  $|a_{i,i}| > \sigma_i$  for all  $1 \leq i \leq n$ , where

$$\sigma_i = \sum_{j \neq i, 1 \leq j \leq n} |a_{i,j}| .$$

Along each column/row the sum of the off-diagonal elements is smaller than the diagonal element.

- $\mathbf{A}$  is called **diagonally dominant** iff  $|a_{i,i}| \geq \sigma_i$  for all  $1 \leq i \leq n$ .
- $\mathbf{A}$  is called **reducible** if there exists a permutation matrix  $\mathbf{P}$  and square matrices  $\mathbf{B}_1, \mathbf{B}_2$  and a matrix  $\mathbf{B}_3$  such that

$$\mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^T = \begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_3 \\ \mathbf{0} & \mathbf{B}_2 \end{bmatrix}$$

Since  $\mathbf{A}$  is symmetric the matrix  $\mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^T$  is also symmetric and the block  $\mathbf{B}_3$  has to vanish, i.e. we have the condition

$$\mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^T = \begin{bmatrix} \mathbf{B}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_2 \end{bmatrix} .$$

This leads to an easy interpretation of a reducible matrix  $\mathbf{A}$ : the system of linear equation  $\mathbf{A} \vec{u} = \vec{b}$  can be decomposed into two smaller, independent systems  $\mathbf{B}_1 \vec{u}_1 = \vec{b}_1$  and  $\mathbf{B}_2 \vec{u}_2 = \vec{b}_2$ . To arrive at this situation all one has to do is renumber the equations and variables.

- $\mathbf{A}$  is called **irreducible** if it is not reducible.
- $\mathbf{A}$  is called **irreducibly diagonally dominant** if  $\mathbf{A}$  is irreducible and
  - $|a_{i,i}| \geq \sigma_i$  for all  $1 \leq i \leq n$
  - $|a_{i,i}| > \sigma_i$  for at least one  $1 \leq i \leq n$

For further explanations concerning reducible matrices see [Axel94] or [VarFEM]. For our purposes it is sufficient to know that the model matrices  $\mathbf{A}_n$  and  $\mathbf{A}_{nn}$  in Section 2.3 are positive definite, diagonally dominant and irreducible.

**2–32 Result :** (see e.g. [Axel94, Theorem 4.9])

Consider a real symmetric matrix  $\mathbf{A}$  with positive numbers along the diagonal. If  $\mathbf{A}$  is strictly diagonally dominant or irreducibly diagonally dominant, then  $\mathbf{A}$  is positive definite.  $\diamond$

As a consequence of the above result the model matrices  $\mathbf{A}_n$  and  $\mathbf{A}_{nn}$  are positive definite.

**2–33 Example :** A positive definite matrix need not be diagonally dominant. As an example consider the matrix

$$\mathbf{A} = \begin{bmatrix} 5 & -4 & 1 & & & \\ -4 & 6 & -4 & 1 & & \\ 1 & -4 & 6 & -4 & 1 & \\ & 1 & -4 & 6 & -4 & 1 \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & 1 & -4 & 6 & -4 & 1 \\ & & & & 1 & -4 & 6 & -4 \\ & & & & & 1 & -4 & 5 \end{bmatrix}.$$

This matrix was generated with the help of the model matrix  $\mathbf{A}_n$ , given on page 32 by  $\mathbf{A} = h^4 \mathbf{A}_n \cdot \mathbf{A}_n$ . This matrix  $\mathbf{A}$  is positive definite, but it is clearly not diagonally dominant.  $\diamond$

The algorithm of Cholesky will not only determine the factorization, but also indicate if the matrix  $\mathbf{A}$  is positive definite. It is an efficient tool to determine positive definiteness.

**2–34 Result :** Let  $\mathbf{A} = \mathbf{R}^T \cdot \mathbf{D} \cdot \mathbf{R}$  be the Cholesky factorization of the previous section. Then  $\mathbf{A}$  is positive definite if and only if all entries in the diagonal matrix  $\mathbf{D}$  are strictly positive.  $\diamond$

**Proof :** Since the triangular matrix  $\mathbf{R}$  has only numbers 1 along the diagonal, it is invertible. If the vectors  $\vec{x} \in \mathbb{R}^n$  will cover all of  $\mathbb{R}^n$ , then the constructed vectors  $\vec{y} = \mathbf{R} \vec{x}$  will also cover all of  $\mathbb{R}^n$ . Now the identity

$$\langle \vec{x}, \mathbf{A} \vec{x} \rangle = \langle \vec{x}, \mathbf{R}^T \cdot \mathbf{D} \cdot \mathbf{R} \vec{x} \rangle = \langle \mathbf{R} \vec{x}, \mathbf{D} \cdot \mathbf{R} \vec{x} \rangle = \langle \vec{y}, \mathbf{D} \vec{y} \rangle.$$

implies

$$\begin{aligned} \langle \vec{x}, \mathbf{A} \cdot \vec{x} \rangle > 0 \quad \text{for all } \vec{x} \neq \vec{0} &\iff \langle \vec{y}, \mathbf{D} \cdot \vec{y} \rangle > 0 \quad \text{for all } \vec{y} \neq \vec{0} \\ &\iff \sum_{i=1}^n d_i y_i^2 > 0 \quad \text{for all } \vec{y} \neq \vec{0} \\ &\iff d_i > 0 \quad \text{for all } 1 \leq i \leq n. \end{aligned}$$

□

### 2.6.3 Stability of the Algorithm of Cholesky

To show that the Cholesky algorithm is stable (without pivoting) for positive definite systems two essential ingredients are used:

- Show that the entries in the factorization  $\mathbf{R}$  and  $\mathbf{D}$  are bounded by the entries in  $\mathbf{A}$ . This is only correct for positive definite matrices.
- Keep track of rounding errors for the algebraic operations to be executed during the algorithm of Cholesky.

#### The entries of $\mathbf{R}$ and $\mathbf{D}$ are bounded

For a symmetric, positive definite matrix we have the factorization

$$\mathbf{A} = \mathbf{R}^T \cdot \mathbf{D} \cdot \mathbf{R}.$$

By multiplying out the diagonal elements we obtain

$$a_{i,i} = d_i + \sum_{k=1}^{i-1} r_{k,i} d_k r_{k,i} = d_i + \sum_{k=1}^{i-1} d_k r_{k,i}^2 = \sum_{k=1}^n d_k r_{k,i}^2.$$

Since  $\mathbf{A}$  is positive definite we know that  $a_{i,i} > 0$  and  $d_i > 0$ . Thus we find bounds on the coefficients in  $\mathbf{R}$  and  $\mathbf{D}$  in terms of  $\mathbf{A}$ .

$$d_i \leq a_{i,i} \quad \text{and} \quad \sum_{k=1}^{i-1} d_k r_{k,i}^2 \leq a_{i,i} \quad \text{or similar} \quad \sum_{k=1}^n d_k r_{k,i}^2 \leq a_{i,i}.$$

Using this and the Cauchy–Schwartz inequality<sup>10</sup> we now obtain an estimate for the result of the matrix multiplication below, where the entries in  $|\mathbf{R}|$  are given by the absolute values of the entries in  $\mathbf{R}$ . Estimates of this type are needed to keep track of the ‘worst case’ situation for rounding errors and the algorithm. We will need information on the expression below.

$$\begin{aligned} (|\mathbf{R}|^T \cdot \mathbf{D} \cdot |\mathbf{R}|)_{i,j} &= \sum_{k=1}^n |r_{k,i}| d_k |r_{k,j}| = \sum_{k=1}^n (|r_{k,i}| \sqrt{d_k}) (\sqrt{d_k} |r_{k,j}|) \\ &\leq \sqrt{\sum_{k=1}^n d_k r_{k,i}^2} \cdot \sqrt{\sum_{k=1}^n d_k r_{k,j}^2} \leq \sqrt{a_{i,i}} \cdot \sqrt{a_{j,j}} \leq \max_{1 \leq j \leq n} a_{j,j}. \end{aligned} \quad (2.3)$$

Example 2–37 shows that the above is false if  $\mathbf{A}$  is not positive definite.

#### Rounding errors while solving

The following result is a modification of Theorem 2–21 for symmetric matrices.

<sup>10</sup>For vectors we know that  $\langle \vec{x}, \vec{y} \rangle = \|\vec{x}\| \|\vec{y}\| \cos \alpha$ , where  $\alpha$  is the angle between the vectors. This implies  $|\langle \vec{x}, \vec{y} \rangle| \leq \|\vec{x}\| \|\vec{y}\|$ .

**2–35 Result :** (Modification of [GoluVanLoan96, Theorem 3.3.1])

Assume that for a positive definite, symmetric  $n \times n$  matrix  $\mathbf{A}$  the algorithm of Cholesky leads to an approximate factorization

$$\hat{\mathbf{R}}^T \cdot \hat{\mathbf{D}} \cdot \hat{\mathbf{R}} = \mathbf{A} + \mathbf{E}$$

Then the error matrix  $\mathbf{E}$  satisfies

$$|\mathbf{E}| \leq 3(n-1) \mathbf{u} (|\mathbf{A}| + |\mathbf{R}|^T \cdot |\mathbf{D}| \cdot |\mathbf{R}|) + O(\mathbf{u}^2).$$

◇

The estimate (2.3) for a positive definite  $\mathbf{A}$  now implies

$$|\mathbf{E}| \leq 6(n-1) \mathbf{u} \max_i a_{i,i}.$$

**2–36 Result :** (Modification of [GoluVanLoan96, Theorem 3.3.2])

Let  $\hat{\mathbf{R}}$  and  $\hat{\mathbf{D}}$  be the computed factors of the Cholesky factorization of the  $n \times n$  matrix  $\mathbf{A}$ . Then forward and back substitution are used to solve  $\hat{\mathbf{D}} \cdot \hat{\mathbf{R}}^T \vec{y} = \vec{b}$  with the computed solution  $\hat{\vec{y}}$  and solve  $\hat{\mathbf{R}} \vec{x} = \hat{\vec{y}}$  with the computed solution  $\hat{\vec{x}}$ . Then

$$(\mathbf{A} + \mathbf{E}) \hat{\vec{x}} = \vec{b} \quad \text{with} \quad |\mathbf{E}| \leq n \mathbf{u} (3|\mathbf{A}| + 5|\mathbf{R}|^T \cdot |\mathbf{D}| \cdot |\mathbf{R}|) + O(\mathbf{u}^2).$$

◇

The estimate (2.3) for a positive definite  $\mathbf{A}$  now implies

$$|\mathbf{E}| \leq 8n \mathbf{u} \max_i a_{i,i},$$

i.e. the result of the numerical computations is the exact solution of slightly modified equations. The modification is small compared to the maximal coefficient in the original problem.

As a consequence of the above we conclude that for positive definite, symmetric matrices there is no need for pivoting when using the Cholesky algorithm.

**2–37 Example :** If the matrix is not positive definite the effect of roundoff errors may be large, even if the matrix has an ideal condition number close to 1. Consider the system

$$\begin{bmatrix} 0.0001 & 1 \\ 1 & 0.0001 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Exact arithmetic leads to the factorization

$$\begin{bmatrix} 0.0001 & 1 \\ 1 & 0.0001 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10000 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.0001 & 0 \\ 0 & -9999.9999 \end{bmatrix} \cdot \begin{bmatrix} 1 & 10000 \\ 0 & 1 \end{bmatrix}.$$

The condition number is  $\kappa = 1.0002$  and thus we expect almost no loss of precision<sup>11</sup>. The exact solution is  $\vec{x} = (0.99990001, 0.99990001)^T$ . Since all numbers in  $\mathbf{A}$  and  $\vec{b}$  are smaller than 1 one might hope

<sup>11</sup>Observe that the eigenvalues of the matrix are  $\lambda_1 = 1.0001$  and  $\lambda_2 = -0.9999$ . Thus the matrix is **not** positive definite. But the permuted matrix (row permutations)

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0.0001 & 1 \\ 1 & 0.0001 \end{bmatrix} = \begin{bmatrix} 1 & 0.0001 \\ 0.0001 & 1 \end{bmatrix}$$

has eigenvalues  $\lambda_1 = 1.0001$  and  $\lambda_2 = +0.9999$  and thus is positive definite.

for an error of the order of machine precision. The bounds on the entries in  $\mathbf{R}$  and  $\mathbf{D}$  in (2.3) are clearly violated, e.g.

$$(|\mathbf{R}|^T \cdot |\mathbf{D}| \cdot |\mathbf{R}|)_{2,2} = |r_{1,2}| |d_1| |r_{1,2}| + |r_{2,2}| |d_2| |r_{2,2}| = 10^8 \cdot 10^{-4} + 9999.9999 \approx 20000$$

Using floating point arithmetic with  $\mathbf{u} \approx 10^{-8}$  (i.e. 8 decimal digits) we obtain a factorization

$$\begin{bmatrix} 1 & 0 \\ 10000 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.0001 & 0 \\ 0 & -10000 \end{bmatrix} \cdot \begin{bmatrix} 1 & 10000 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.0001 & 1 \\ 1 & 0 \end{bmatrix}$$

and the solution is  $\hat{\mathbf{x}} = (1.0, 0.9999)^T$ . Thus the relative error of the solution is  $10^{-4}$ . This is by magnitudes larger than the machine precision  $\mathbf{u} \approx 10^{-8}$ . The effect is generated by the large numbers in the factorization. This can not occur if the matrix  $\mathbf{A}$  is positive definite since we have the bound (2.3). To overcome this type of problem a good pivoting scheme has to be used when the matrix is not positive definite, see e.g. [GoluVanLoan96, §4.4]. This will most often destroy the symmetry of the problem. It is possible to use row and column permutations to preserve the symmetry. This approach will be examined in Section 2.6.6.

◇

## 2.6.4 Banded Matrices and the Algorithm of Cholesky

Both matrices  $\mathbf{A}_n$  and  $\mathbf{A}_{nn}$  in Section 2.3 exhibit a **band structure**. This is no coincidence as most matrices generated by finite element or finite difference methods are banded matrices. We present the most elementary direct method using the band structure of the matrix  $\mathbf{A}$ . This approach is practical if the degrees of freedom in a finite element problem are numbered to minimize the bandwidth of the matrix. There are special algorithms to achieve this goal, e.g. Cuthill-McKee as described in Section 6.2.7 in the context of FEM.

If a symmetric matrix  $\mathbf{A}$  has all nonzero numbers close to the diagonal, then it is called a **banded matrix**. If  $a_{i,j} = 0$  for  $|i - j| > b$  then the integer  $b$  is called the **semibandwidth** of  $\mathbf{A}$ . For a tridiagonal matrix we find  $b = 2$ , the main diagonal and one off-diagonal. As the algorithm of Cholesky is based on row and column operations we can apply it to a banded matrix and as long as no pivoting is done the band structure of the matrix is maintained. Thus we can factor a positive definite symmetric matrix  $\mathbf{A}$  with semibandwidth  $b$  as

$$\mathbf{A} = \mathbf{R}^T \cdot \mathbf{D} \cdot \mathbf{R},$$

where  $\mathbf{R}$  is an upper triangular unity matrix with semibandwidth  $b$  and  $\mathbf{D}$  is a diagonal matrix with positive entries. This situation is visualized in Figure 2.8. For a  $n \times n$  matrix  $\mathbf{A}$  we are interested in the situation  $1 < b \ll n$ .

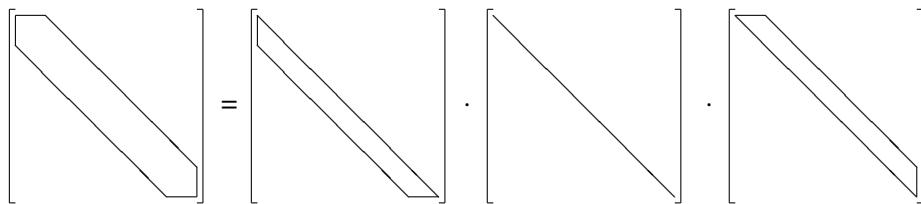


Figure 2.8: The Cholesky decomposition for a banded matrix

When implementing the algorithm of Cholesky one works along the diagonal, top to bottom. For each step only a block of size  $b \times b$  of numbers is worked on, i.e. has to be quickly accessible. Three of these situations are shown in Figure 2.9. Each of those steps requires approximately  $b^2/2$  flops and there are approximately  $n$  of those, thus we find an approximate<sup>12</sup> operational count of

<sup>12</sup>We ignored the effect that the first row in each diagonal step is left unchanged and we also do not take into account that in the lower right corner fewer computations are needed. Both effects are of lower order.

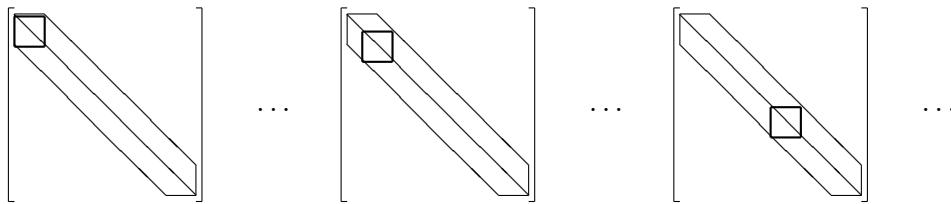


Figure 2.9: Cholesky steps for a banded matrix. The active area is marked

$$\text{FlopCholBand} \approx \frac{1}{2} b^2 n .$$

This has to be compared to the computational effort for the full Cholesky factorization

$$\text{FlopChol} \approx \frac{1}{6} n^3 = \frac{1}{6} n^2 n .$$

The additional cost to solve a system by back substitution is approximately

$$\text{FlopSolveBand} \approx 2 b n .$$

We need  $n \cdot b$  numbers to store the complete matrix  $\mathbf{A}$ . As the algorithm proceeds along the diagonal in  $\mathbf{A}$  (or its reduction  $\mathbf{R}$ ) in each step only the next  $b$  rows will be worked on. As we go to the next row the previous top row will not be used any more, but a new row at the will be added at the bottom, see Figure 2.9. Thus we have only  $b \cdot b$  active entries at any time. If these numbers can be placed in **fast memory** (e.g. in cache) then the implementation will run faster than in regular memory. Thus for good performance we like to store  $b^2$  numbers in fast memory. This has to be taken in consideration when setting up the data structure for a banded matrix, together with the memory and cache architecture of the computer to be used. Table 2.5 shows the types of fast and regular memory relevant for most problems and some typical sizes of matrices. The table assumes that 8 Bytes are used to store one number. If not enough fast memory is available the algorithm will still generate the result, but not as fast. An efficient implementation of the above idea is shown in [VarFEM].

size	band	fast memory	memory
		cache [MB]	RAM [MB]
1'000	100	0.08	0.8
10'000	100	0.08	8
100'000	100	0.08	80
100'000	200	0.32	160
100'000	500	2	400
100'000	1'000	8	800
1'000'000	1'000	8	8'000

Table 2.5: Memory requirements for the Cholesky algorithm for banded matrices

**2–38 Observation :** If the semibandwidth  $b$  is considerably larger than the number of used CPU cores, then the algorithm can efficiently be implemented on a multi core architecture, as shown in Section 2.2.3.



## 2.6.5 Computing with an Inverse Matrix is Usually Inefficient

not in class

If a linear system with matrix  $\mathbf{A} \in \mathbb{M}^{n \times n}$  has to be solved many times one might be tempted to compute the inverse matrix  $\mathbf{A}^{-1}$ . As example examine a symmetric, positive definite matrix  $\mathbf{A}$  with semibandwidth  $b \ll n$ . This example illustrates that it is very inefficient to compute with inverse matrices, it uses more memory and requires more flops, i.e. a longer computation time.

To determine the inverse matrix  $\mathbf{M} = \mathbf{R}^{-1}$  of the Cholesky factorization supplement the band matrix  $\mathbf{R}$  with the identity matrix, and then use row operations to transform the left part to the identity matrix  $\mathbb{I}_n$ . Working with elementary matrices (Section 2.4.2, page 43) examine the augmented matrix.

$$\begin{aligned}\mathbf{R} \cdot \mathbf{R}^{-1} &= \mathbb{I}_n \\ (\mathbf{M} \cdot \mathbf{R}) \cdot \mathbf{R}^{-1} &= \mathbf{M} \\ \mathbb{I}_n \cdot \mathbf{R}^{-1} &= \mathbf{M}\end{aligned}$$

As example consider a  $6 \times 6$  matrix with semibandwidth 3.

$$\left[ \begin{array}{cccccc|c} r_{1,1} & r_{1,2} & r_{1,3} & 0 & 0 & 0 & 1 \\ r_{2,2} & r_{2,3} & r_{2,4} & 0 & 0 & 0 & 1 \\ r_{3,3} & r_{3,4} & r_{3,5} & 0 & 0 & 0 & 1 \\ r_{4,4} & r_{4,5} & r_{4,6} & 0 & 0 & 0 & 1 \\ r_{5,5} & r_{5,6} & & 0 & 0 & 0 & 1 \\ r_{6,6} & & & 0 & 0 & 0 & 1 \end{array} \right]$$

and using row operations this has to be transformed to

$$\left[ \begin{array}{ccc|cccc} 1 & & & m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ 1 & & & m_{2,2} & m_{2,3} & m_{2,4} & m_{2,5} & m_{2,6} & \\ 1 & & & m_{3,3} & m_{3,4} & m_{3,5} & m_{3,6} & & \\ 1 & & & m_{4,4} & m_{4,5} & m_{4,6} & & & \\ 1 & & & m_{5,5} & m_{5,6} & & & & \\ 1 & & & m_{6,6} & & & & & \end{array} \right].$$

To transform  $\mathbf{R}$  to the diagonal matrix work bottom up.

- Divide the last row by  $r_{n,n}$ . Subtract multiples of the last row from the last  $b$  rows, such that the only nonzero entry in column  $n$  is the value 1 in the bottom right corner. This requires  $b$  flops.
- Divide row  $n - 1$  by  $r_{n-1,n-1}$ . Subtract multiples of row  $n - 1$  from rows  $n - 2$  to  $n - b$  rows, such that the only nonzero the value 1 on the diagonal. This requires  $b \cdot 2$  flops.
- Divide row  $n - 2$  by  $r_{n-2,n-2}$ . Subtract multiples of row  $n - 2$  from rows  $n - 3$  to  $n - b - 1$  rows, such that the only nonzero the value 1 on the diagonal. This requires  $b \cdot 3$  flops.
- Proceed similarly for all rows, up to the first row.

Observe that the inverse matrix  $\mathbf{M} = \mathbf{R}^{-1}$  is a full, upper triangular matrix. Thus it contains  $\frac{n^2}{2}$  entries. The number of required flops is estimated by

$$\sum_{k=1}^n b k \approx \frac{1}{2} b n^2 .$$

Since  $(\mathbf{R}^T)^{-1} = (\mathbf{R}^{-1})^T$  and  $\mathbf{A}^{-1} = (\mathbf{R}^T \mathbf{R})^{-1} = \mathbf{R}^{-1} \mathbf{R}^{-T}$  we could now solve the system  $\mathbf{A} \vec{x} = \vec{b}$  by

$$\vec{b} = \mathbf{R}^{-1} (\mathbf{R}^{-T} \vec{x}),$$

i.e. by two matrix mutiplications, each requiring approximately  $\frac{1}{2} n^2$  flops. This coincides with the number of operations required to multiply with the full matrix  $\mathbf{A}^{-1} = \mathbf{R}^{-1} \mathbf{R}^{-T}$ .

If we avoid the inverse matrix  $\mathbf{R}$  but use twice a back–substitution: first  $\mathbf{R}^T \vec{y} = \vec{b}$  and then  $\mathbf{R} \vec{x} = \vec{y}$  we require only  $2 b n$  flops, i.e. considerably less than  $\frac{1}{2} n^2$ .

## 2.6.6 Octave Implementations of Sparse Direct Solvers

In the previous sections we examined a banded Cholesky algorithm. This is only one example of a **sparse direct solver**. The above ideas of a banded Cholesky solver can be improved, with considerable additional effort. UMFPACK and CHOLMOD (by Timothy Davis) are good libraries for direct solvers for sparse matrices. Both are used by Octave. We illustrate its use by an example.

Examine the steady state heat equation (1.5) (page 12) on a unit square with  $nx$  interior grid points in  $x$  direction and  $ny$  points in  $y$  direction.

$$\begin{aligned} -\frac{\partial^2 T(x,y)}{\partial x^2} - \frac{\partial^2 T(x,y)}{\partial y^2} &= \frac{1}{k} f(x, y) && \text{for } 0 \leq x, y \leq 1 \\ T(x, y) &= 0 && \text{for } (x, y) \text{ on boundary} \end{aligned} .$$

The resulting matrix is of size  $nx \cdot ny$  by  $nx \cdot ny$  with a semi-bandwidth of  $nx$ . The matrix may be generated as a Kronecker product of two tridiagonal matrices, representing the second derivatives in  $x$  and  $y$  direction.

```
nx = 200; ny = nx; h_x = 1/(nx+1); h_y = 1/(ny+1);
Dxx = spdiags(ones(nx,1)*[-1, 2, -1], [-1, 0, 1], nx, nx) / (h_x^2);
Dyy = spdiags(ones(ny,1)*[-1, 2, -1], [-1, 0, 1], ny, ny) / (h_y^2);

A = kron(speye(ny), Dxx) + kron(Dyy, speye(nx));
b = ones(nx*ny, 1);
```

Now solve the resulting system of linear equations  $\mathbf{A} \vec{x} = \vec{b}$  with different algorithms.

run  
SpCholDem

- The above sparse matrix is converted to a full matrix, whose inverse is used to determine the solution.

```
Afull = full(A);
x1 = inv(Afull)*b;
```

This method consumes a lot of computation time and a full matrix with  $200^4$  entries has to be stored, i.e. we would need  $8 \cdot 200^4 \text{ B} = 12.8 \text{ GB}$  of memory. **This is a foolish method to use.** The computation actually failed for  $n = 200$ . A test with  $n = 80$  leads to a computation time of 150 seconds.

- The standard solver of Octave uses a good algorithm and the code

```

tic()
x2 = A\b;
SolveTime = toc()

```

takes 0.108 seconds to solve one system. Octave uses the selection tree displayed in Section 2.6.7.

- We may first compute the Cholesky factorization  $\mathbf{A} = \mathbf{R}^T \mathbf{R}$  and then determine the solution in two steps: solve  $\mathbf{R}^T \vec{y} = \vec{b}$ , then  $\mathbf{R} \vec{x} = \vec{y}$ .

```

R = chol(A);
x3 = R\ (R' \b);

```

It takes 0.667 seconds to compute  $\mathbf{R}$  and then 0.127 seconds to solve. The command `nnz(R)` shows that  $\mathbf{R}$  has approximately  $8 \cdot 10^6$  nonzero entries. This coincides with the  $n_x^3 = 200^3$  entries required by the banded Cholesky solver examined in the previous sections.

- One can modify the Cholesky algorithm with row and column permutations, seeking a factorization

$$\mathbf{P}^T \mathbf{A} \mathbf{P} = \mathbf{R}^T \mathbf{R}$$

where  $\mathbf{P}$  is a permutation matrix. Octave uses the **Approximate Minimum Degree** permutation matrix generated by the command `amd()`. Systems of linear equations can then be solved by

$$\mathbf{A} \vec{x} = \vec{b} \iff \mathbf{P}^T \mathbf{A} \mathbf{P} \mathbf{P}^T \vec{x} = \mathbf{P}^T \vec{b} \iff \mathbf{R}^T \mathbf{R} \mathbf{P}^T \vec{x} = \mathbf{P}^T \vec{b}$$

and thus by the sequence

$$\begin{aligned} \mathbf{R}^T \vec{y} &= \mathbf{P}^T \vec{b} \\ \mathbf{R} \vec{z} &= \vec{y} \\ \mathbf{P}^T \vec{x} &= \vec{z} \end{aligned}$$

With Octave/MATLAB this translates to

```

[R, m, P] = chol(A);
x4 = P*(R\ (R' \ (P' *b)));

```

The result requires 0.122 second for the factorization and only 0.014 seconds to solve the system. The matrix  $\mathbf{R}$  has only  $1081911 \approx 10^6$  nonzero entries.

To illustrate the above we use our standard matrix  $\mathbf{A}_{nn}$  with  $n = 200$ . Using Octave (version 5.0.1) on a Xeon E5-1650 system we found the numbers in Table 2.6.

algorithm	storage (numbers)	factorization time	solving time
full Cholesky	$\frac{1}{2} n^4 \approx 8 \cdot 10^8$	hopeless	??
sparse Cholesky	$\approx 8 \cdot 10^6$	0.667 sec	0.127 sec
sparse Cholesky with permutations	$\approx 1 \cdot 10^6$	0.122 sec	0.014 sec
standard \ operator			0.108 sec

Table 2.6: Comparison of direct solvers for  $\mathbf{A}_{nn}$  with  $n = 200$

The Cholesky algorithm with permutations is most efficient, concerning computation time and memory consumption. We illustrate this by a slightly modified example. We take the above standard example, but add one nonzero number to destroy most of the band structure.

```

nx = 20; ny = nx;      hx = 1/(nx+1); hy = 1/(ny+1);
Dxx = spdiags(ones(nx,1)*[-1 2 -1], [-1 0 1], nx, nx) / (hx^2);
Dyy = spdiags(ones(ny,1)*[-1 2 -1], [-1 0 1], ny, ny) / (hy^2);
A = kron(speye(ny), Dxx) + kron(Dyy, speye(nx));
A(nx, nx*ny/2) = -A(1,1)/4; A(nx*ny/2, nx) = -A(1,1)/4;
                           % add numbers to destroy the band structure
R = chol(A);             % standard Cholesky, sparse
[R2,m,P] = chol(A);    % sparse Cholesky, with permutations
nonzeroR = nnz(R)
nonzeroR2 = nnz(R2)
-->
nonzeroR = 8179
nonzeroR2 = 3758

```

Run  
CholSpy.m

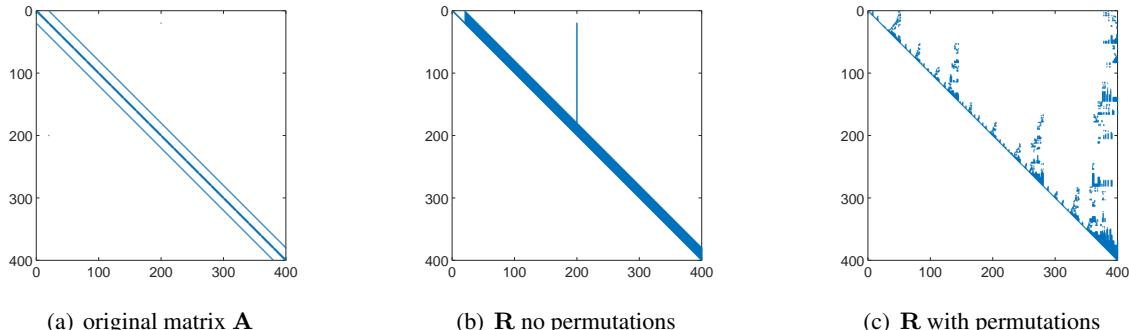


Figure 2.10: The sparsity pattern of a band matrix and two Cholesky factorizations

The matrix  $\mathbf{A}$  generated with the above code is of size  $400 \times 400$  and the semi-bandwidth is 20 (approximately). We can compute the size of the sparse matrix required to store the Cholesky factorization  $\mathbf{R}$ :

- Full Cholesky: half of a  $N \times N = 400 \times 400$  matrix, leading to 80'000 entries.
- Band Cholesky:  $N \times b = 400 \times 20$ , leading to 8'000 nonzero entries. Ignoring the single nonzero we still have a semi bandwidth of 20 and thus the banded Cholesky factorization would require  $400 \cdot 20 = 8'000$  nonzero entries.
- Sparse Cholesky: 8179 nonzero entries
- Sparse Cholesky with permutations: 3785 nonzero entries. As a consequence we need less storage and the back substitution will be about twice as fast.

The sparsity pattern in Figure 2.10 shows where the non-zeros are. In 2.10(a) find the non-zeros in the original matrix  $\mathbf{A}$ . The band structure is clearly visible. By zooming in we would find only 5 diagonals occupied by numbers, i.e. the band is far from being full. In 2.10(b) we recognize the result of the Cholesky factorization, where the additional nonzero entry leads to an isolated spike in the matrix  $\mathbf{R}$ . The band in this matrix is full. In 2.10(c) find the results with the additional permutations allowed. The band structure is replaced by a even more sparse pattern of non-zeros.

We observe:

- The `chol()` implementation in Octave is as efficient as a banded Cholesky and can deal with isolated nonzeros outside of the band.
- The `chol()` command with the additional permutations can be considerably more efficient, i.e. requires less memory and the back substitution is faster.

## 2.6.7 A Selection Tree used in Octave for Sparse Linear Systems

The banded Cholesky algorithm above shows how to use properties of the matrices to find efficient algorithms to solve systems of linear equations. There are many more tricks of the trade to be used. The goal of the previous section is to explain **one** of the essential ideas. Real world codes should use more features of the matrices. *Octave* and *MATLAB* use **sparse matrices** and more advanced algorithms.

The documentation of *Octave* contains a selection tree for solving systems of linear equations using sparse matrices. Find this information in the official *Octave* manual in the section *Linear Algebra on Sparse Matrices*. When using the command  $\mathbf{A} \backslash \mathbf{b}$  with a sparse matrix  $\mathbf{A}$  to solve a linear system the following decision tree is used to choose the algorithm to solve the system.

1. If the matrix is diagonal, solve directly and goto 8.
2. If the matrix is a permuted diagonal, solve directly taking into account the permutations. Goto 8
3. If the matrix is square, banded and if the band density is less than that given by spparms ("banden") continue, else goto 4.
  - (a) If the matrix is tridiagonal and the right-hand side is not sparse continue, else goto 3(b).
    - i. If the matrix is hermitian, with a positive real diagonal, attempt Cholesky factorization using Lapack xPTSV.
    - ii. If the above failed or the matrix is not hermitian with a positive real diagonal use Gaussian elimination with pivoting using Lapack xGTSV, and goto 8.
  - (b) If the matrix is hermitian with a positive real diagonal, attempt Cholesky factorization using Lapack xPBTRF.
  - (c) if the above failed or the matrix is not hermitian with a positive real diagonal use Gaussian elimination with pivoting using Lapack xGBTRF, and goto 8.
4. If the matrix is upper or lower triangular perform a sparse forward or backward substitution, and goto 8.
5. If the matrix is a upper triangular matrix with column permutations or lower triangular matrix with row permutations, perform a sparse forward or backward substitution, and goto 8.
6. If the matrix is square, hermitian with a real positive diagonal, attempt sparse Cholesky factorization using CHOLMOD.
7. If the sparse Cholesky factorization failed or the matrix is not hermitian with a real positive diagonal, and the matrix is square, factorize using UMFPACK.
8. If the matrix is not square, or any of the previous solvers flags a singular or near singular matrix, find a minimum norm solution using CXSPARSE.

The above clearly illustrates that a reliable and efficient algorithm to solve linear systems of equations uses more than the most elementary ideas. In particular the keywords Cholesky and band structure appear often.

## 2.7 Sparse Matrices and Iterative Solvers

All of the problems in Chapter 1 lead to linear systems  $\mathbf{A} \vec{x} + \vec{b} = \vec{0}$ , where only very few entries of the large matrix  $\mathbf{A}$  are different from zero, i.e. we have a **sparse matrix**. The Cholesky algorithm for banded matrices is using only some of this sparsity. Due to the sparsity the computational effort to compute a matrix product  $\mathbf{A} \vec{x}$  is minimal, compared to the number of operations to solve the above system with a direct method. One

is lead to search for an algorithm to solve the linear system, using matrix multiplications only. This leads to **iterative methods**, i.e. we apply computational operations until the desired accuracy is achieved. There is no reliable method to decide beforehand how many operations will have to be applied. The previously considered algorithms of LR factorization and Cholesky are both **direct methods**, since both methods will lead to the solution of the linear system using a known, finite number of operations.

Sparse matrices can very efficiently be multiplied with a vector. Thus we seek algorithms to solve linear systems of equations, using multiplications only. The trade-off is that we might have to use many multiplications of a matrix times a vector.

We will examine methods that allow to solve linear systems with  $10^6$  unknowns within a few seconds on a standard computer, see Table 2.15 on page 91.

**2-39 Observation :** It is possible to take advantage of a multi-core architecture for the multiplication of a sparse matrix with a vector.  $\diamond$

### 2.7.1 The Model Problems

In Section 2.3 we find the matrix  $\mathbf{A}_{nn}$  of size  $n^2 \times n^2$  with a semi-bandwidth of  $n + 1 \approx n$ . In each row/column only 5 entries are different from zero. For the condition number we obtain

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \approx \frac{4}{\pi^2} n^2.$$

When using a banded Cholesky algorithm to solve  $\mathbf{A} \vec{x} + \vec{b} = \vec{0}$  we need

- storage for  $n \cdot n^2 = n^3$  numbers.
- approximately  $\frac{1}{2} n^2 n^2 = \frac{1}{2} n^4$  floating point operations.

An iterative method will have to do better than this to be considered useful. To multiply the matrix  $\mathbf{A}_{nn}$  with a vector we need about  $5 n^2$  multiplications.

The above matrix  $\mathbf{A}_{nn}$  might appear when solving a two dimensional heat conduction problem. For the similar three dimensional problem we find a matrix  $\mathbf{A}$  of size  $N = n^3$  and each row has approximately 7 nonzero entries. The semi-bandwidth of the matrix is  $n^2$ . Thus the banded Cholesky solver requires approximately  $\frac{1}{2} n^3 \cdot n^4$  floating point operations. The condition number is identical to the 2-D situation.

### 2.7.2 Basic Definitions

For a given invertible  $N \times N$  matrix  $\mathbf{A}$  and a given vector  $\vec{b}$  we have the exact solution  $\vec{x}$  of  $\mathbf{A} \vec{x} + \vec{b} = \vec{0}$ . For an iteration mapping  $\Phi : \mathbb{R}^N \rightarrow \mathbb{R}^N$  we choose an initial vector  $\vec{x}_0$  and then compute  $\vec{x}_1 = \Phi(\vec{x}_0)$ ,  $\vec{x}_2 = \Phi(\vec{x}_1) = \Phi^2(\vec{x}_0)$  or

$$\vec{x}_k = \Phi^k(\vec{x}_0).$$

The mapping  $\Phi$  is called an **iterative method** with **linear convergence factor**  $q < 1$  if the error after  $k$  steps is bounded by

$$\|\vec{x}_k - \vec{x}\| \leq c q^k.$$

To improve the accuracy of the initial guess  $\vec{x}_0$  by  $D$  digits we need  $q^k \leq 10^{-D}$ . This is satisfied if

$$k \log q \leq -D \quad \text{or} \quad k \geq \frac{-D}{\log q} = \frac{-D \ln 10}{\ln q} > 0.$$

For most applications the factor  $q < 1$  will be very close to 1. Thus examine  $q = 1 - q_1$  and use the Taylor approximation  $\ln q = \ln(1 - q_1) \approx -q_1$ . Then the above computations leads to an estimate for the number of iterations necessary to decrease the error by  $D$  digits, i.e.

$$k \geq \frac{D \ln 10}{q_1}. \quad (2.4)$$

This implies that the numbers of desired correct digits is proportional to the number of required iterations and inversely proportional to the deviation  $q_1$  of the factor  $q = 1 - q_1$  from 1.

### 2.7.3 Steepest Descent Iteration, Gradient Algorithm

For a symmetric, positive definite matrix  $\mathbf{A}$  the solution of the linear system  $\mathbf{A} \vec{x} + \vec{b} = \vec{0}$  is given by the location of the minimum of the function

$$f(\vec{x}) = \frac{1}{2} \langle \vec{x}, \mathbf{A} \vec{x} \rangle + \langle \vec{x}, \vec{b} \rangle.$$

A possible graph of such a function and its level curves are shown in Figure 2.11. For symmetric matrices the gradient of this function is given by<sup>13</sup>

$$\nabla f(\vec{x}) = \mathbf{A} \vec{x} + \vec{b} = \vec{0}.$$

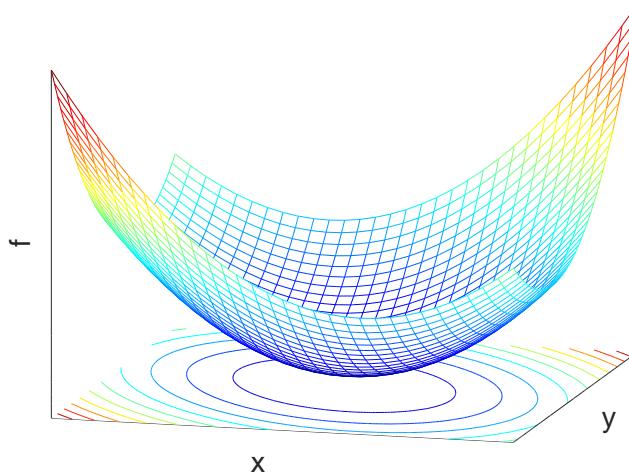


Figure 2.11: Graph of a function to be minimized and its level curves

A given point  $\vec{x}_k$  is assumed to be a good approximation of the exact solution  $\vec{x}$ . The error is given by the **residual vector**

$$\vec{r}_k = \mathbf{A} \vec{x}_k + \vec{b}.$$

---

<sup>13</sup> Use a summation notation for the scalar and matrix product and differentiate with the help of the product rule.

$$\begin{aligned} f(\vec{x}) &= \frac{1}{2} \langle \vec{x}, \mathbf{A} \vec{x} \rangle + \langle \vec{x}, \vec{b} \rangle = \frac{1}{2} \left( \sum_{i=1}^n x_i \left( \sum_{j=1}^n a_{i,j} x_j \right) \right) + \sum_{1 \leq j \leq n} b_j x_j \\ 0 = \frac{\partial}{\partial x_k} f(\vec{x}) &= \frac{1}{2} \left( 1 \sum_{j=1}^n a_{k,j} x_j + \sum_{i=1}^n x_i (a_{i,k} 1) \right) + b_k 1 = \sum_{1 \leq j \leq n} a_{k,j} x_j + b_k \end{aligned}$$

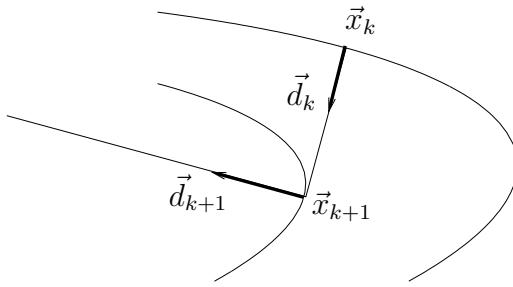


Figure 2.12: One step of a gradient iteration

The direction of steepest descent is given by

$$\vec{d}_k = -\nabla f(\vec{x}_k) = -\mathbf{A} \vec{x}_k - \vec{b} = -\vec{r}_k.$$

This is the reason for the name **steepest descent** or **gradient method**, illustrated in Figure 2.12. Thus search for a better solution in the direction  $\vec{d}_k$ , i.e. determine the coefficient  $\alpha \in \mathbb{R}$ , such that the value of the function

$$\begin{aligned} h(\alpha) &= f(\vec{x}_k + \alpha \vec{d}_k) = \frac{1}{2} \langle (\vec{x}_k + \alpha \vec{d}_k), \mathbf{A} (\vec{x}_k + \alpha \vec{d}_k) \rangle + \langle (\vec{x}_k + \alpha \vec{d}_k), \vec{b} \rangle \\ &= \frac{\alpha^2}{2} \langle \vec{d}_k, \mathbf{A} \vec{d}_k \rangle + \frac{\alpha}{2} \left( \langle \vec{d}_k, \mathbf{A} \vec{x}_k \rangle + \langle \mathbf{A} \vec{d}_k, \vec{x}_k \rangle + 2 \langle \vec{d}_k, \vec{b} \rangle \right) + \text{ independent on } \alpha \\ &= \frac{\alpha^2}{2} \langle \vec{d}_k, \mathbf{A} \vec{d}_k \rangle + \alpha \left( \langle \vec{d}_k, \mathbf{A} \vec{x}_k \rangle + \langle \vec{d}_k, \vec{b} \rangle \right) + \text{ independent on } \alpha \end{aligned}$$

is minimal. This leads to the condition

$$\begin{aligned} 0 = \frac{dh(\alpha)}{d\alpha} &= \alpha \langle \vec{d}_k, \mathbf{A} \vec{d}_k \rangle + \left( \langle \vec{d}_k, \mathbf{A} \vec{x}_k \rangle + \langle \vec{d}_k, \vec{b} \rangle \right) = \alpha \langle \vec{d}_k, \mathbf{A} \vec{d}_k \rangle + \langle \vec{d}_k, \mathbf{A} \vec{x}_k + \vec{b} \rangle \\ \alpha &= -\frac{\langle \vec{d}_k, \mathbf{A} \vec{x}_k + \vec{b} \rangle}{\langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle} = -\frac{\langle \vec{d}_k, \vec{r}_k \rangle}{\langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle} = +\frac{\langle \vec{r}_k, \vec{r}_k \rangle}{\langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle} \end{aligned}$$

and thus the next approximation  $\vec{x}_{k+1}$  of the solution is given by

$$\vec{x}_{k+1} = \vec{x}_k + \alpha \vec{d}_k = \vec{x}_k + \frac{\|\vec{r}_k\|^2}{\langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle} \vec{d}_k.$$

One step of this iteration is shown in Figure 2.12 and a pseudo code for the algorithm is shown on the left in Table 2.7.

The computational effort for one step in the algorithm seems to be: 2 matrix/vector multiplications, 2 scalar products and 2 vector additions. But the residual vector  $\vec{r}_k$  and the direction vector  $\vec{d}_k$  differ only in their sign. Since

$$\vec{r}_{k+1} = \mathbf{A} \vec{x}_{k+1} + \vec{b} = \mathbf{A} (\vec{x}_k + \alpha_k \vec{d}_k) + \vec{b} = \mathbf{A} \vec{x}_k + \vec{b} + \alpha_k \mathbf{A} \vec{d}_k = \vec{r}_k + \alpha_k \mathbf{A} \vec{d}_k$$

the necessary computations for one step of the iteration can be reduced, leading to the algorithm on the right in Table 2.7. To translate between the two implementations use a few  $\pm$  changes and

basic algorithm	$\longleftrightarrow$	improved algorithm
$\vec{d}_k$	$\longleftrightarrow$	$\vec{r}$
$\mathbf{A} \vec{d}_k$	$\longleftrightarrow$	$\vec{d}$
$\vec{x}_{k+1} = \vec{x}_k + \alpha \vec{d}_k$	$\longleftrightarrow$	$\vec{x} = \vec{x} + \alpha \vec{r}$
$\vec{r}_{k+1} = \vec{r}_k + \mathbf{A} \vec{d}_k$	$\longleftrightarrow$	$\vec{r} = \vec{r} + \alpha \vec{d}$

The improved algorithm in Table 2.7 requires

choose initial point $\vec{x}_0$ $k = 0$ while $\ \vec{r}_k\  = \ \mathbf{A} \vec{x}_k + \vec{b}\ $ too large $\vec{d}_k = -\vec{r}_k$ $\alpha = -\frac{\langle \vec{r}_k, \vec{d}_k \rangle}{\langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle}$ $\vec{x}_{k+1} = \vec{x}_k + \alpha \vec{d}_k$ $k = k + 1$ endwhile	choose initial point $\vec{x}$ $\vec{r} = \mathbf{A} \vec{x} + \vec{b}$ while $\rho = \ \vec{r}\ ^2 = \langle \vec{r}, \vec{r} \rangle$ too large $\vec{d} = \mathbf{A} \vec{r}$ $\alpha = -\frac{\rho}{\langle \vec{d}, \vec{r} \rangle}$ $\vec{x} = \vec{x} + \alpha \vec{r}$ $\vec{r} = \vec{r} + \alpha \vec{d}$ endwhile
--	--

Table 2.7: Gradient algorithm to solve  $\mathbf{A} \vec{x} + \vec{b} = \vec{0}$ , a first attempt (left) and an efficient implementation (right)

- one matrix–vector product and two scalar products
- two vector additions of the type  $\vec{x} = \vec{x} + \alpha \vec{r}$
- storage for the sparse matrix and 3 vectors

If each row of the  $N \times N$ -matrix  $\mathbf{A}$  has on average  $nz$  nonzero entries, then each iteration requires approximately  $(4 + nz)N$  flops (multiplication/addition pairs).

Since the matrix  $\mathbf{A}$  is positive definite

$$\frac{d^2}{d\alpha^2} h(\alpha) = \langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle > 0,$$

unless  $-\vec{d}_k = \mathbf{A} \vec{x}_k + \vec{b} = \vec{0}$ . This is the minimum of the function  $h(\alpha)$  and consequently  $f(\vec{x}_{k+1}) < f(\vec{x}_k)$ , unless  $\vec{x}_k$  equals the exact solution of  $\mathbf{A} \vec{x} + \vec{b} = \vec{0}$ . Since  $\vec{d}_k = -\vec{r}_k$  conclude that  $\alpha \geq 0$ , i.e. the algorithm made a step of positive length in the direction of the negative gradient.

The algorithm does not perform well if we search the minimal value in a narrow valley, as illustrated in Figure 2.13. Instead of *going down* the valley, the algorithm *jumps across* and it requires many steps to get close to the lowest point. This is reflected by the error estimate for this algorithm. One can show that (e.g. [Brae02, Satz 2.4], [Hack94, Theorem 9.2.3], [Hack16, Theorem 9.10], [LascTheo87, p. 496], [KnabAnge00, p. 212], [AxelBark84, Theorem 1.8])<sup>14</sup>

$$\|\vec{x}_k - \vec{x}\|_A \leq \left( \frac{\kappa - 1}{\kappa + 1} \right)^k \|\vec{x}_0 - \vec{x}\|_A \approx \left( 1 - \frac{2}{\kappa} \right)^k \|\vec{x}_0 - \vec{x}\|_A \quad (2.5)$$

using the energy norm  $\|\vec{x}\|_A^2 = \langle \vec{x}, \mathbf{A} \vec{x} \rangle$ .

<sup>14</sup>In [GoluVanLoan13, §11.3.2] find a complete (rather short) proof of

$$\|\vec{x}_{k+1} - \vec{x}_*\|_A^2 \leq \left( 1 - \frac{1}{\kappa_2(\mathbf{A})} \right) \|\vec{x}_k - \vec{x}_*\|_A^2.$$

Using

$$\sqrt{1 - \frac{1}{\kappa}} \approx 1 - \frac{1}{2\kappa} > 1 - \frac{2}{\kappa}$$

observe that (2.5) is a slightly better estimate. I have a note of the proof in [GoluVanLoan13, p. 627ff], adapted to the notation of these lecture notes.

For most matrices based on finite element problems we know that  $\|\vec{x}\| \leq \alpha \|\vec{x}\|_A$  and thus

$$\|\vec{x}_k - \vec{x}\| \leq c \left( \frac{\kappa - 1}{\kappa + 1} \right)^k \approx c \left( 1 - \frac{2}{\kappa} \right)^k$$

where

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}} = \text{condition number of } \mathbf{A}.$$

The resulting number of required iterations is given by

$$k \geq \frac{D \ln 10}{q_1} = \frac{D \ln 10}{2} \kappa.$$

Thus if the ratio of the largest and smallest eigenvalue of the matrix  $\mathbf{A}$  is large, then the algorithm converges slowly. Unfortunately this is most often the case, thus Figure 2.13 shows the typical situation and not the exception.

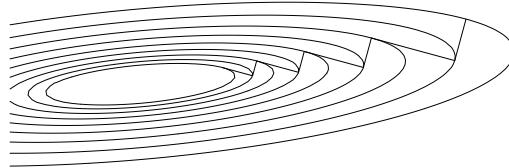


Figure 2.13: The gradient algorithm for a large condition number

### Performance on the model problem

For the problem in Section 2.7.1 we find  $\kappa \approx \frac{4}{\pi^2} n^2$  and thus

$$q = 1 - q_1 = 1 - \frac{2}{\kappa} \approx 1 - \frac{\pi^2}{2n^2}.$$

Then equation (2.4) implies that we need

$$k \geq \frac{D \ln 10}{q_1} = \frac{2 D \ln 10}{\pi^2} n^2$$

iterations to increase the accuracy by  $D$  digits. Based on the estimated operation counts

Operation with $\mathbf{A}_{nn}$	flops
$\mathbf{A}_{nn} \cdot \vec{x}$	$5n^2$
$\vec{x} = \vec{x} + \alpha \vec{r}$	$n^2$
$\langle \vec{d}, \vec{r} \rangle$	$n^2$

for the operations necessary for each step in the steepest descent iteration we arrive at the total number of flops as

$$9n^2 k \approx \frac{18 D \ln 10}{\pi^2} n^4 \approx 4.2 D n^4.$$

This is slightly worse than a banded Cholesky algorithm ( $\text{Flop}_{Chol} \approx \frac{1}{2} n^4$ ). The gradient algorithm does use less memory, but requires more flops.

## 2.7.4 Conjugate Gradient Iteration

The **conjugate gradient algorithm**<sup>15</sup> will improve the above mentioned problem of the gradient method. Instead of searching for the minimum of the function  $f(\vec{x}) = \frac{1}{2} \langle \vec{x}, \mathbf{A} \vec{x} \rangle + \langle \vec{b}, \vec{x} \rangle$  in the direction of steepest descent, combine this direction with the previous search direction and aim to reach the minimal value of the function  $f(\vec{x})$  in this plane with one step only.

The algorithm was named as one of the top ten algorithms of the 20th century, see [TopTen]. Find a detailed, readable introduction to the method of conjugate gradients in [Shew94].

### Conjugate directions

On the left in Figure 2.14 find elliptical level curves of the function  $g(\vec{x}) = \langle \vec{x}, \mathbf{A} \vec{x} \rangle$ . A first vector  $\vec{a}$  is tangential to a given level curve at a point. A second vector  $\vec{b}$  is connecting this point to the origin. The two vectors represent two subsequent search directions. When applying the transformation

$$\vec{u} = \begin{pmatrix} u \\ v \end{pmatrix} = \mathbf{A}^{1/2} \begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{A}^{1/2} \vec{x}$$

obtain

$$g(\vec{x}) = \langle \vec{x}, \mathbf{A} \vec{x} \rangle = \langle \mathbf{A}^{1/2} \vec{x}, \mathbf{A}^{1/2} \vec{x} \rangle = \langle \vec{u}, \vec{u} \rangle = h(\vec{u})$$

and the level curves of the function  $h$  in a  $(u, v)$  system will be circles, shown on the right in Figure 2.14. The two vectors  $\vec{a}$  and  $\vec{b}$  shown on in the left part will transform according to the same transformation rule. The resulting images will be orthogonal and thus

$$0 = \langle \mathbf{A}^{1/2} \vec{a}, \mathbf{A}^{1/2} \vec{b} \rangle = \langle \mathbf{A} \vec{a}, \vec{b} \rangle.$$

The vectors  $\vec{a}$  and  $\vec{b}$  are said to be **conjugate**<sup>16</sup>.

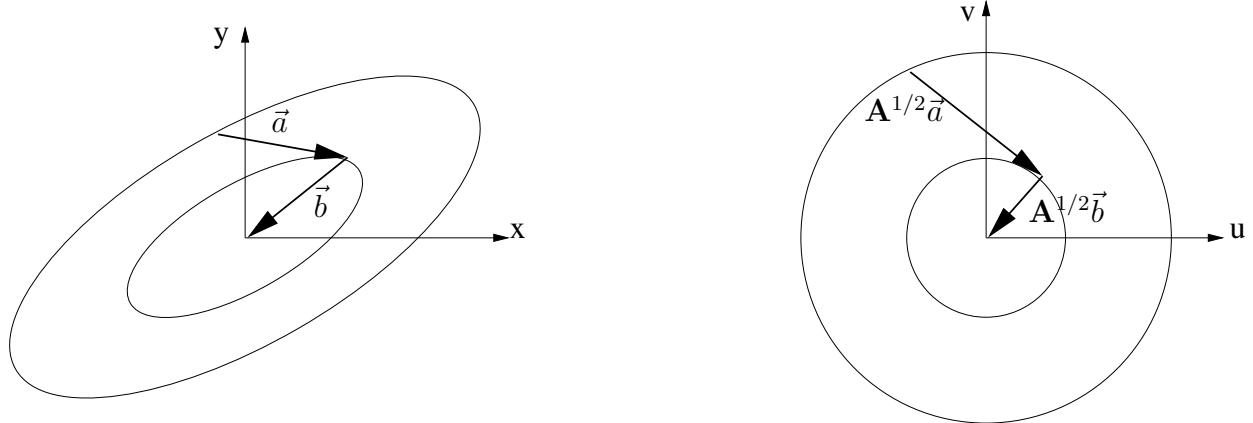


Figure 2.14: Ellipse and circle to illustrate conjugate directions

<sup>15</sup>The conjugate gradient algorithm was developed by Magnus Hestenes and Eduard Stiefel (ETHZ) in 1952.

<sup>16</sup>Using the diagonalization of the matrix  $\mathbf{A}$  (see page 134) we even have a formula for  $\mathbf{A}^{1/2}$ . Since  $\mathbf{A} = \mathbf{Q} \cdot \text{diag}(\lambda_i) \cdot \mathbf{Q}^T$  we use  $\mathbf{A}^{1/2} = \mathbf{Q} \cdot \sqrt{\text{diag}(\lambda_i)} \cdot \mathbf{Q}^T$  and conclude

$$\mathbf{A}^{1/2} \cdot \mathbf{A}^{1/2} = \mathbf{Q} \cdot \sqrt{\text{diag}(\lambda_i)} \cdot \mathbf{Q}^T \cdot \mathbf{Q} \cdot \sqrt{\text{diag}(\lambda_i)} \cdot \mathbf{Q}^T = \mathbf{Q} \cdot \text{diag}(\lambda_i) \cdot \mathbf{Q}^T = \mathbf{A}.$$

Fortunately we do **not** need the explicit formula for  $\mathbf{A}^{1/2}$ , since this would require all eigenvectors, which is computationally expensive. For the algorithm it is sufficient to know how to multiply a vector by the matrix  $\mathbf{A}$ .

### The basic conjugate gradient algorithm

The direction vectors  $\vec{d}_{k-1}$  and  $\vec{d}_k$  of two subsequent steps of the conjugate gradient algorithm should behave like the two vectors in the left part of Figure 2.14. The new direction vector  $\vec{d}_k$  is assumed to be a linear combination of the gradient  $\nabla f(\vec{x}_k) = \mathbf{A} \vec{x}_k + \vec{b} = \vec{r}_k$  and the old search direction  $\vec{d}_{k-1}$ , i.e.

$$\vec{d}_k = -\vec{r}_k + \beta \vec{d}_{k-1} \quad \text{where} \quad \vec{r}_k = \mathbf{A} \vec{x}_k + \vec{b}.$$

Since the two directions  $\vec{d}_k$  and  $\vec{d}_{k-1}$  have to be conjugate conclude

$$\begin{aligned} 0 &= \langle \vec{d}_k, \mathbf{A} \vec{d}_{k-1} \rangle = \langle -\vec{r}_k + \beta \vec{d}_{k-1}, \mathbf{A} \vec{d}_{k-1} \rangle \\ \beta &= \frac{\langle \vec{r}_k, \mathbf{A} \vec{d}_{k-1} \rangle}{\langle \vec{d}_{k-1}, \mathbf{A} \vec{d}_{k-1} \rangle}. \end{aligned}$$

Then the optimal value of  $\alpha_k$  to minimize  $h(\alpha) = f(\vec{x}_k + \alpha_k \vec{d}_k)$  can be determined with a calculation identical to the standard gradient method, i.e.

$$\alpha_k = -\frac{\langle \vec{r}_k, \vec{d}_k \rangle}{\langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle}$$

and obtain a better approximation of the solution of the linear system by  $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{d}_k$ . This algorithm is spelled out on the left in Table 2.8 and its result is illustrated in Figure 2.15. Just as in the standard gradient algorithm find  $\frac{d^2}{d\alpha^2} h(\alpha) = \langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle > 0$  and thus

- either the algorithm terminates, i.e. we found the optimal solution at this point
- or  $\alpha_k > 0$ .

This allows for division by  $\alpha_k$  in the analysis of the algorithm.

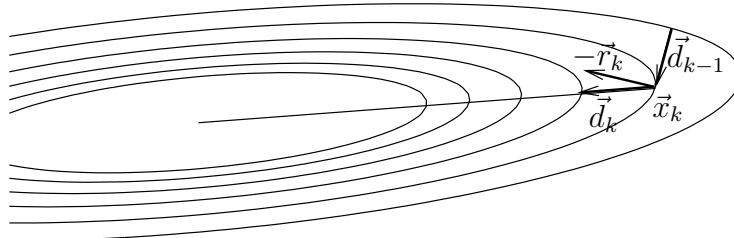


Figure 2.15: One step of a conjugate gradient iteration

### An example in $\mathbb{R}^2$

Ex 2.13

The function

$$\begin{aligned} f(x, y) &= \frac{1}{2} \left\langle \begin{bmatrix} +1 & -0.5 \\ -0.5 & +3 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \right\rangle + \left\langle \begin{pmatrix} -1 \\ -2 \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \right\rangle \\ &= \frac{1}{2} x^2 + \frac{1}{2} x y + \frac{3}{2} y^2 - x - 2 y \end{aligned}$$

is minimized at  $(x, y) \approx (1.45455, 0.90909)$ . With a starting vector at  $(x_0, y_0) = (1, 1)$  one can apply two steps of the gradient algorithm, or two steps of the conjugate gradient algorithm. The first step of the conjugate gradient algorithm coincides with the first step of the gradient algorithm, since there is no previous direction to determine the conjugate direction yet. The result is shown in Figure 2.16. The two blue arrows are the result of the gradient algorithm (steepest descent) and the green vector is the second step of the conjugate gradient algorithm. In this example the conjugate gradient algorithm finds the exact solution with two steps. This is not a coincidence, but generally correct and caused by orthogonality properties of the conjugate gradient algorithm.

choose initial point  $\vec{x}_0$

$$\vec{r}_0 = \mathbf{A} \vec{x}_0 + \vec{b}$$

$$\vec{d}_0 = -\vec{r}_0$$

$$\alpha_0 = -\frac{\langle \vec{r}_0, \vec{d}_0 \rangle}{\langle \mathbf{A} \vec{d}_0, \vec{d}_0 \rangle}$$

$$\vec{x}_1 = \vec{x}_0 + \alpha_0 \vec{d}_0$$

$$\vec{r}_1 = \mathbf{A} \vec{x}_1 + \vec{b}$$

$$k = 1$$

while  $\|\vec{r}_k\|$  too large

$$\beta_k = \frac{\langle \vec{r}_k, \mathbf{A} \vec{d}_{k-1} \rangle}{\langle \vec{d}_{k-1}, \mathbf{A} \vec{d}_{k-1} \rangle}$$

$$\vec{d}_k = -\vec{r}_k + \beta_k \vec{d}_{k-1}$$

$$\alpha_k = -\frac{\langle \vec{r}_k, \vec{d}_k \rangle}{\langle \mathbf{A} \vec{d}_k, \vec{d}_k \rangle}$$

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{d}_k$$

$$k = k + 1$$

$$\vec{r}_k = \mathbf{A} \vec{x}_k + \vec{b}$$

endwhile

choose initial point  $\vec{x}_0$

$$\vec{r} = \mathbf{A} \vec{x} + \vec{b}$$

$$\rho_0 = \|\vec{r}\|^2$$

$$\vec{d} = -\vec{r}$$

$$\vec{p} = \mathbf{A} \vec{d}$$

$$\alpha = \frac{\rho_0}{\langle \vec{p}, \vec{d} \rangle}$$

$$\vec{x} = \vec{x} + \alpha \vec{d}$$

$$\vec{r} = \vec{r} + \alpha \vec{p}$$

$$k = 1$$

while  $\rho_k$  too large

$$\beta = \frac{\rho_k}{\rho_{k-1}}$$

$$\vec{d} = -\vec{r} + \beta \vec{d}$$

$$\vec{p} = \mathbf{A} \vec{d}$$

$$\alpha = \frac{\rho_k}{\langle \vec{p}, \vec{d} \rangle}$$

$$\vec{x} = \vec{x} + \alpha \vec{d}$$

$$\vec{r} = \vec{r} + \alpha \vec{p}$$

$$\rho_k = \langle \vec{r}, \vec{r} \rangle$$

$$k = k + 1$$

endwhile

Table 2.8: The conjugate gradient algorithm to solve  $\mathbf{A} \vec{x} + \vec{b} = \vec{0}$  and an efficient implementation

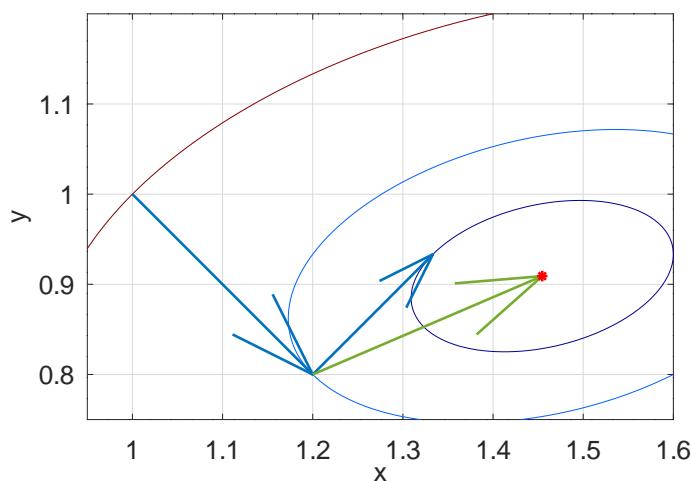


Figure 2.16: Two steps of the gradient algorithm (blue) and the conjugate gradient algorithm (green)

### Orthogonality properties

We define the Krylov subspaces generated by the matrix  $\mathbf{A}$  and the vector  $\vec{d}_0$

$$K(k, \vec{d}_0) = \text{span}\{\vec{d}_0, \mathbf{A}\vec{d}_0, \mathbf{A}^2\vec{d}_0, \dots, \mathbf{A}^{k-1}\vec{d}_0, \mathbf{A}^k\vec{d}_0\}.$$

Since  $\vec{r}_{k+1} = \vec{r}_k + \alpha_k \mathbf{A}\vec{d}_k$  and  $\vec{d}_k = -\vec{r}_k + \beta_k \vec{d}_{k-1}$  we conclude

$$\vec{r}_i \in K(k, \vec{d}_0) \quad , \quad \vec{d}_i \in K(k, \vec{d}_0) \quad \text{and} \quad \vec{x}_i \in \vec{x}_0 + K(k, \vec{d}_0) \quad \text{for } 0 \leq i \leq k.$$

The above is correct for any choice of the parameters  $\beta_k$ . Now we examine the algorithm in Table 2.8 with the optimal choice for  $\alpha_k$ , but the values of  $\beta_k$  in  $\vec{d}_k = -\vec{r}_k + \beta_k \vec{d}_{k-1}$  are to be determined by a new criterion. The theorem below shows that we minimized the function  $f(\vec{x})$  on the  $k+1$  dimensional affine subspace  $K(k, \vec{d}_0)$ , and not only on the two dimensional plane spanned by the last two search directions.

**2-40 Theorem :** Consider fixed values of  $k \in \mathbb{N}$ ,  $\vec{x}_0$  and  $\vec{r}_0 = \mathbf{A}\vec{x}_0 + \vec{b}$ . Choose the vector  $\vec{x} \in \vec{x}_0 + K(k, \vec{d}_0)$  such that the energy function  $f(\vec{x}) = \frac{1}{2} \langle \vec{x}, \mathbf{A}\vec{x} \rangle + \langle \vec{x}, \vec{b} \rangle$  is minimized on the affine subspace  $\vec{x}_0 + K(k, \vec{d}_0)$ . The subspace  $K(k, \vec{d}_0)$  has dimension  $k+1$ . The following orthogonality properties are correct

$$\begin{aligned} \langle \vec{r}_j, \vec{r}_i \rangle &= 0 && \text{for all } 0 \leq i \neq j < k \\ \langle \vec{d}_j, \mathbf{A}\vec{d}_i \rangle &= 0 && \text{for all } 0 \leq i \neq j \leq k \\ \langle \vec{r}_k, \vec{y} \rangle &= \langle \vec{x}_k - \vec{x}, \mathbf{A}\vec{y} \rangle = 0 && \text{for all } \vec{y} \in K(k, \vec{d}_0). \end{aligned}$$

The values

$$\beta_k = \frac{\langle \vec{r}_k, \mathbf{A}\vec{d}_{k-1} \rangle}{\langle \vec{d}_{k-1}, \mathbf{A}\vec{d}_{k-1} \rangle}$$

will generate the optimal solution, i.e.  $f(\vec{x})$  is minimized, with the algorithm on the left in Table 2.8.  $\diamond$

**Proof :** Since the vector  $\vec{x} \in \vec{x}_0 + K(k, \vec{d}_0)$  minimizes the function  $f(\vec{x})$  on the affine subspace  $\vec{x}_0 +$  not in class  $K(k, \vec{d}_0)$ , its gradient has to be orthogonal on the subspace  $K(k, \vec{d}_0)$ , i.e. with  $\vec{r} = \mathbf{A}\vec{x} + \vec{b} = \nabla f(\vec{x})$

$$\langle \mathbf{A}\vec{x} + \vec{b}, \vec{h} \rangle = \langle \vec{r}, \vec{h} \rangle = 0 \quad \text{for all } \vec{h} \in K(k, \vec{d}_0).$$

Since  $\vec{r} = \vec{r}_{k+1} = \mathbf{A}\vec{x} + \vec{b}$  this leads to

$$\langle \vec{r}_{k+1}, \vec{r}_i \rangle = \langle \vec{r}, \vec{r}_i \rangle = 0 \quad \text{for all } 0 \leq i \leq k$$

and  $K(k, \vec{d}_0)$  is a strict subspace of  $K(k+1, \vec{d}_0)$ . This implies  $\dim(K(k, \vec{d}_0)) = k+1$ .

Using  $\vec{r}_{k+1} = \vec{r}_k + \alpha_k \mathbf{A}\vec{d}_k$  and  $\vec{d}_i = -\vec{r}_k + \beta_i \vec{d}_{i-1}$  we conclude by recursion

$$\begin{aligned} \langle \vec{d}_i, \mathbf{A}\vec{d}_k \rangle &= \langle -\vec{r}_i + \beta_i \vec{d}_{i-1}, \frac{\vec{r}_{k+1} - \vec{r}_k}{\alpha_k} \rangle \\ &= \frac{\beta_i}{\alpha_k} \langle \vec{d}_{i-1}, \vec{r}_{k+1} - \vec{r}_k \rangle = \frac{1}{\alpha_k} \left( \prod_{j=1}^i \beta_j \right) \langle \vec{d}_0, \vec{r}_{k+1} - \vec{r}_k \rangle \\ &= \frac{-1}{\alpha_k} \left( \prod_{j=1}^i \beta_j \right) \langle \vec{r}_0, \vec{r}_{k+1} - \vec{r}_k \rangle = 0. \end{aligned}$$

The above is correct for all possible choices of  $\beta_j$  and also implies

$$0 = \langle \vec{d}_k, \mathbf{A}\vec{d}_{k-1} \rangle = \langle -\vec{r}_k + \beta_k \vec{d}_{k-1}, \mathbf{A}\vec{d}_{k-1} \rangle = -\langle \vec{r}_k, \mathbf{A}\vec{d}_{k-1} \rangle + \beta_k \langle \vec{d}_{k-1}, \mathbf{A}\vec{d}_{k-1} \rangle.$$

Thus the optimal values for  $\beta_k$  are as shown in the theorem.  $\square$

**2-41 Corollary :**

- Since  $\dim(K(k, \vec{d}_0)) = k + 1$  the conjugate gradient algorithm with exact arithmetic will terminate after at most  $N$  steps. Due to rounding errors this will not be of practical relevance for large matrices. In addition the number of steps might be prohibitively large. Thus use the conjugate gradient algorithm as an iterative method.

- Using the orthogonalities in the above theorem to conclude

$$\begin{aligned}
\langle \vec{r}_k, \vec{d}_k \rangle &= \langle \vec{r}_k, -\vec{r}_k + \beta_k \vec{d}_{k-1} \rangle = -\|\vec{r}_k\|^2 \\
\langle \vec{r}_k, \mathbf{A} \vec{d}_{k-1} \rangle &= \beta_k \langle \vec{d}_{k-1}, \mathbf{A} \vec{d}_{k-1} \rangle \\
\vec{r}_{k+1} &= \vec{r}_k + \alpha_k \mathbf{A} \vec{d}_k \\
\langle \vec{r}_{k+1}, \mathbf{A} \vec{d}_k \rangle &= \frac{1}{\alpha_k} \langle \vec{r}_{k+1}, \vec{r}_{k+1} - \vec{r}_k \rangle = \frac{1}{\alpha_k} \|\vec{r}_{k+1}\|^2 \\
\langle \vec{d}_k, \mathbf{A} \vec{d}_k \rangle &= \frac{1}{\alpha_k} \langle \vec{d}_k, \vec{r}_{k+1} - \vec{r}_k \rangle = \frac{1}{\alpha_k} \|\vec{r}_k\|^2 \\
\beta_k &= \frac{\langle \vec{r}_k, \mathbf{A} \vec{d}_{k-1} \rangle}{\langle \vec{d}_{k-1}, \mathbf{A} \vec{d}_{k-1} \rangle} = \frac{\alpha_{k-1} \|\vec{r}_k\|^2}{\alpha_{k-1} \|\vec{r}_{k-1}\|^2} = \frac{\|\vec{r}_k\|^2}{\|\vec{r}_{k-1}\|^2}.
\end{aligned}$$

◇

The above properties allow a more efficient implementation of the conjugate gradient algorithm. The algorithm on the right in Table 2.8 is taken from [GoluVanLoan96]. This improved implementation of the algorithm requires for each iteration

- one matrix–vector product and two scalar products,
- three vector additions of the type  $\vec{x} = \vec{x} + \alpha \vec{r}$ ,
- storage for the sparse matrix and 4 vectors.

If each row of the matrix  $\mathbf{A}$  has on average  $nz$  nonzero entries then we determine that each iteration requires approximately  $(5 + nz) N$  flops (multiplication/addition pairs).

**Convergence estimate**

Assume that the exact solution is given by  $\vec{z}$ , i.e.  $\mathbf{A} \vec{z} + \vec{b} = \vec{0}$ . Use the notation  $\vec{r} = \mathbf{A} \vec{y} + \vec{b}$ , resp.  $\vec{y} = \mathbf{A}^{-1}(\vec{r} - \vec{b})$  to conclude that  $\vec{y} - \vec{z} = \mathbf{A}^{-1} \vec{r}$ . Then consider the following function

$$g(\vec{y}) = \|\vec{y} - \vec{z}\|_A^2 = \langle \vec{y} - \vec{z}, \mathbf{A}(\vec{y} - \vec{z}) \rangle = \langle \vec{r}, \mathbf{A}^{-1} \vec{r} \rangle$$

and verify that

$$\begin{aligned}
\frac{1}{2} \|\vec{x} - \vec{z}\|_A^2 &= \frac{1}{2} \langle \vec{x} - \vec{z}, \mathbf{A}(\vec{x} - \vec{z}) \rangle = \frac{1}{2} \langle \vec{x} + \mathbf{A}^{-1} \vec{b}, \mathbf{A} \vec{x} + \vec{b} \rangle \\
&= \frac{1}{2} \langle \vec{x}, \mathbf{A} \vec{x} \rangle + \langle \vec{x}, \vec{b} \rangle + \frac{1}{2} \langle \mathbf{A}^{-1} \vec{b}, \vec{b} \rangle \\
&= f(\vec{x}) + \frac{1}{2} \langle \mathbf{A}^{-1} \vec{b}, \vec{b} \rangle.
\end{aligned}$$

Thus the conjugate gradient algorithm minimized the energy norm given by the function  $g(\vec{y}) = \|\vec{y} - \vec{z}\|_A$  on the subspaces  $K(k, \vec{d}_0)$ . It should be no surprise that the error estimate can be expressed in this norm.

Find the result and proofs<sup>17</sup> in [Brae02, Satz 3.7], [Hack94, Theorem 9.4.12], [Hack16, Theorem 10.14], [LascTheo87], [KnabAnge00, p. 218] or [AxelBark84]. The relevant convergence estimate is

$$\|\vec{x}_k - \vec{x}\|_A \leq 2 \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|\vec{x}_0 - \vec{x}\|_A \approx 2 \left( 1 - \frac{2}{\sqrt{\kappa}} \right)^k \|\vec{x}_0 - \vec{x}\|_A.$$

This leads to

$$\|\vec{x}_k - \vec{x}\| \leq c \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \approx c \left( 1 - \frac{2}{\sqrt{\kappa}} \right)^k. \quad (2.6)$$

Compare this with the corresponding estimate (2.5) (page 76), where  $\kappa$  appears, instead of  $\sqrt{\kappa}$ . The resulting number of required iterations is thus given by

$$k \geq \frac{D \ln 10}{q_1} = \frac{D \ln 10}{2} \sqrt{\kappa}. \quad (2.7)$$

This is considerably better than the estimate for the steepest descent method, since  $\kappa$  is replaced by  $\sqrt{\kappa} \ll \kappa$ .

### Performance on the model problems

Ex 2.17

For the problem in Section 2.7.1 we find  $\sqrt{\kappa} \approx \frac{2}{\pi} n$  and thus

$$q = 1 - q_1 = 1 - \frac{2}{\sqrt{\kappa}} \approx 1 - \frac{\pi}{n}.$$

Then equation (2.4) implies that we need

$$k \geq \frac{D \ln 10}{q_1} = \frac{D \ln 10}{\pi} n$$

iterations to increase the precision by  $D$  digits. Based on the estimate for the operations necessary to multiply the matrix with a vector we estimate the total number of flops as

$$(5 + 5) n^2 k \approx 10 \frac{D \ln 10}{\pi} n^3 \approx 7.3 D n^3.$$

This is considerably better than a banded Cholesky algorithm, since the number of operations is proportional to  $n^3$  instead of  $n^4$ . For large values of  $n$  the conjugate gradient method is clearly preferable.

Table 2.9 shows the required storage and the number of necessary flops to solve the 2-D and 3-D model problems with  $n$  free grid points in each direction. The results are illustrated<sup>18</sup> in Figure 2.17. Observe that one operation for the gradient algorithm requires more time than one operation of the Cholesky algorithm, due to the multiplication of the sparse matrix with a vector.

We may draw the following conclusions from Table 2.9 and the corresponding Figure 2.17.

- The iterative methods require less memory than direct solvers. For 3-D problem this difference is accentuated.
- For 2-D problems with small resolution the banded Cholesky algorithm is more efficient than the conjugate gradient method. For larger 2-D problems conjugate gradient will perform better.

<sup>17</sup>The simpler proof in [GoluVanLoan13, Theorem 11.3.3] does not produce the best possible estimate. There find the estimate

$$\|\vec{x}_{k+1} - \vec{x}_k\| \leq \left( 1 - \frac{1}{\kappa} \right)^{1/2} \|\vec{x}_k - \vec{x}\| \leq \left( 1 - \frac{1}{2\kappa} \right) \|\vec{x}_k - \vec{x}\|$$

leading to  $q_1 = \frac{1}{2\kappa}$ , while the better estimates leads to  $q_1 = \frac{2}{\sqrt{\kappa}}$ . The difference is essential.

<sup>18</sup>The accuracy is required to improve by 6 digits.

- For 3-D problems one should always use conjugate gradient, even for small problems.
- For small 3-D problems banded Cholesky will compute results with reasonable computation time.
- The method of steepest descent is never competitive.

Table 2.10 lists approximate computation times for a CPU capable of performing  $10^8 = 100$  MFLOPS or  $10^{11} = 100$  GFLOPS.

	2-D		3-D	
	storage	flops	storage	flops
Cholesky, banded	$n^3$	$\frac{1}{2} n^4$	$n^5$	$\frac{1}{2} n^7$
Steepest Descent	$8 n^2$	$\frac{18 D \ln 10}{\pi^2} n^4$	$10 n^3$	$\frac{22 D \ln 10}{\pi^2} n^5$
Conjugate Gradient	$9 n^2$	$\frac{10 D \ln 10}{\pi} n^3$	$11 n^3$	$\frac{12 D \ln 10}{\pi} n^4$

Table 2.9: Comparison of algorithms for the model problems

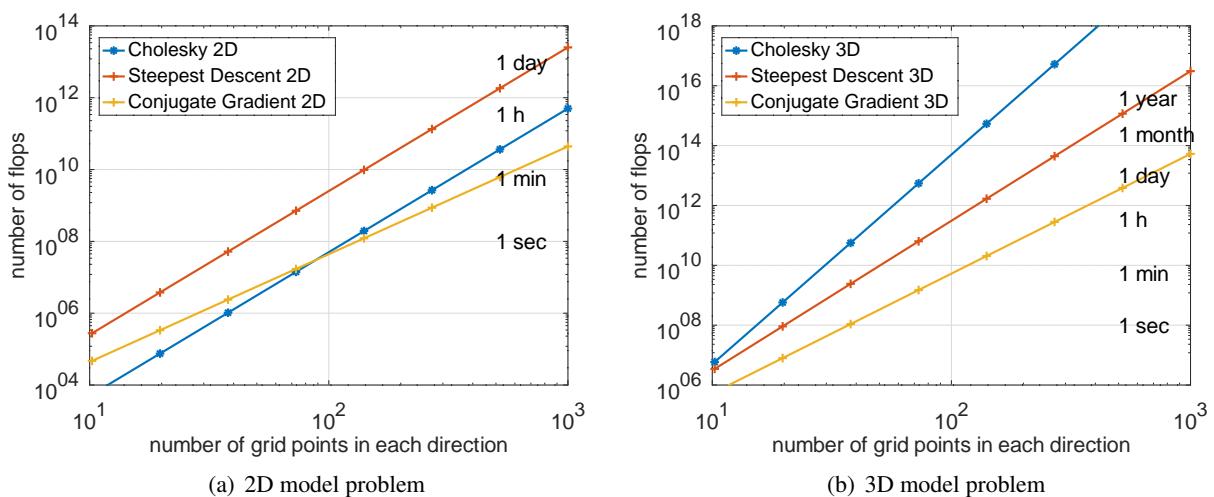


Figure 2.17: Number of operations of banded Cholesky, steepest descent and conjugate gradient algorithm on the model problem. The time estimates are for a CPU capable to perform 100 MFLOPS.

CPU \ flops	$10^8$	$10^9$	$10^{10}$	$10^{11}$	$10^{12}$	$10^{14}$	$10^{16}$	$10^{18}$
100 MFLOPS	1 sec	10 sec	1.7 min	17 min	2.8 h	11.6 days	3.2 years	320 years
100 GFLOPS	0.001 sec	0.01 sec	0.1 sec	1 sec	10 sec	16 min	28 h	116 days

Table 2.10: Time required to complete a given number of flops on a 100 MFLOPS or 100 GFLOPS CPU

## 2.7.5 Preconditioning

Based on equation (2.6) the convergence of the conjugate gradient method is heavily influenced by the condition number of the matrix  $\mathbf{A}$ . If the problem is modified such that the condition number decreases there will be faster convergence. The idea is to replace the system  $\mathbf{A}\vec{x} = -\vec{b}$  by an equivalent system with a smaller condition number. There are different options on how to proceed:

Ex 2.18

- Left preconditioning:

$$\mathbf{M}^{-1}\mathbf{A}\vec{x} = -\mathbf{M}^{-1}\vec{b}.$$

- Right preconditioning:

$$\mathbf{A}\mathbf{M}^{-1}\vec{u} = -\vec{b} \quad \text{with} \quad \vec{x} = \mathbf{M}^{-1}\vec{u}.$$

- Split preconditioning:  $\mathbf{M}$  is factored by  $\mathbf{M} = \mathbf{M}_L \cdot \mathbf{M}_R$

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\vec{u} = -\mathbf{M}_L^{-1}\vec{b} \quad \text{with} \quad \vec{x} = \mathbf{M}_R^{-1}\vec{u}.$$

The ideal condition number for  $\mathbf{M}^{-1}\mathbf{A}$  (resp.  $\mathbf{A}\mathbf{M}^{-1}$  or  $\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}$ ) would be 1, but this would require  $\mathbf{M} = \mathbf{A}$  and thus the system of linear equations is solved. The aim is to get the new matrix  $\mathbf{M}^{-1}\mathbf{A}$  (resp.  $\mathbf{A}\mathbf{M}^{-1}$  or  $\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}$ ) as close as possible to the identity matrix, but demanding little computational effort. In addition the new matrix might not be symmetric and we have to modify the above idea slightly. There are a number of different methods to implement this idea and write efficient code. Consult the literature before writing your own code, e.g. [Saad00]. This reference is available on the internet. With Octave/MATLAB many algorithms and preconditioners are available, see Table 2.18 on page 99. As a good starting reference for code use [templates] and the codes at [www.netlib.org/templates/matlab](http://www.netlib.org/templates/matlab).

As a typical example we examine a Cholesky factorization of a symmetric matrix  $\mathbf{M}$ , i.e.

$$\mathbf{M} = \mathbf{R}^T \mathbf{R}.$$

The matrices  $\mathbf{R}$  and  $\mathbf{M}$  have to be chosen such that it takes little effort to solve systems with those matrices and also as little memory as possible. Two of the possible constructions of these matrices will be shown below, the incomplete Cholesky preconditioner. Then split the preconditioner between the left and right side. Use  $\vec{x} = \mathbf{R}^{-1}\vec{u}$  to conclude

$$\tilde{\mathbf{A}}\vec{u} = \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{u} = -\mathbf{R}^{-T}\vec{b}.$$

Now verify that the new matrix is symmetric since

$$\langle \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{x}, \vec{y} \rangle = \langle \mathbf{A}\mathbf{R}^{-1}\vec{x}, \mathbf{R}^{-1}\vec{y} \rangle = \langle \vec{x}, \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{y} \rangle$$

and apply the conjugate gradient algorithm (see Table 2.8 on page 80) with the new matrix

$$\tilde{\mathbf{A}} = \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}.$$

If the matrix  $\mathbf{M} = \mathbf{R}^T\mathbf{R}$  is relatively close to the matrix  $\mathbf{A} = \mathbf{R}_e^T\mathbf{R}_e$  we conclude that  $\mathbf{R} \approx \mathbf{R}_e$  and thus

$$\tilde{\mathbf{A}} = (\mathbf{R}^{-T}\mathbf{R}_e^T)(\mathbf{R}_e\mathbf{R}^{-1}) \approx \mathbb{I} \cdot \mathbb{I} = \mathbb{I}.$$

As a consequence find a small condition number of the modified matrix  $\tilde{\mathbf{A}}$ . This leads to the basic algorithm on the left in Table 2.11.

In the algorithm on the center of Table 2.11 introduce the new vector

$$\begin{aligned} \vec{z}_k &= \mathbf{R}^{-1}\vec{r}_k = \mathbf{R}^{-1}\tilde{\mathbf{A}}\vec{x}_k + \mathbf{R}^{-1}\mathbf{R}^{-T}\vec{b} = \mathbf{R}^{-1}\mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{x}_k + \mathbf{R}^{-1}\mathbf{R}^{-T}\vec{b} \\ &= \mathbf{M}^{-1}\mathbf{A}\mathbf{R}^{-1}\vec{x}_k + \mathbf{M}^{-1}\vec{b} = \mathbf{M}^{-1}(\mathbf{A}\mathbf{R}^{-1}\vec{x}_k + \vec{b}). \end{aligned}$$

Then realize that the vectors  $\vec{d}_k$  and  $\vec{x}_k$  appear in the form  $\mathbf{R}^{-1}\vec{d}_k$  and  $\mathbf{R}^{-1}\vec{x}_k$ . This allows for a translation of the algorithm with slight changes as shown on the right in Table 2.11. This can serve as starting point for an efficient implementation.

Observe that the update of  $\vec{z}_k$  involves the matrix  $\mathbf{M}^{-1}$  and thus we have to solve the system

$$\mathbf{M}\vec{z}_k = \mathbf{A}\mathbf{R}^{-1}\vec{r}_k$$

for  $\vec{z}_k$ . Thus it is important that the structure of the matrix  $\mathbf{M}$  allows for fast solutions.

```

choose initial point  $\vec{x}_0$ 
 $\vec{r}_0 = \tilde{\mathbf{A}}\vec{x}_0 + \mathbf{R}^{-T}\vec{b}$ 
 $\vec{d}_0 = -\vec{r}_0$ 
 $\alpha_0 = -\frac{\langle \vec{r}_0, \vec{d}_0 \rangle}{\langle \tilde{\mathbf{A}}\vec{d}_0, \vec{d}_0 \rangle}$ 
 $\vec{x}_1 = \vec{x}_0 + \alpha_0 \vec{d}_0$ 
 $k = 1$ 
while  $\|\vec{r}_k\|$  too large
  choose initial point  $\vec{x}_0$ 
   $\vec{r}_0 = \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{x}_0 + \mathbf{R}^{-T}\vec{b}$ 
   $\mathbf{R}^{-1}\vec{d}_0 = -\mathbf{R}^{-1}\vec{r}_0$ 
   $\alpha_0 = -\frac{\langle \vec{r}_0, \vec{d}_0 \rangle}{\langle \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{d}_0, \vec{d}_0 \rangle}$ 
   $\vec{x}_1 = \mathbf{R}^{-1}\vec{x}_0 + \alpha_0 \mathbf{R}^{-1}\vec{d}_0$ 
   $k = 1$ 
  while  $\|\vec{r}_k\|$  too large
    choose initial point  $\vec{x}_0$ 
     $\vec{r}_0 = \mathbf{R}^{-T}\tilde{\mathbf{A}}\mathbf{R}^{-1}\vec{x}_0 + \mathbf{R}^{-T}\vec{b}$ 
     $\beta_k = \frac{\langle \vec{r}_k, \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{d}_{k-1} \rangle}{\langle \vec{d}_{k-1}, \mathbf{A}\vec{d}_{k-1} \rangle}$ 
     $\vec{d}_k = -\vec{r}_k + \beta_k \mathbf{R}^{-1}\vec{d}_{k-1}$ 
     $\alpha_k = -\frac{\langle \vec{r}_k, \vec{d}_k \rangle}{\langle \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{d}_k, \vec{d}_k \rangle}$ 
     $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \mathbf{R}^{-1}\vec{d}_k$ 
     $k = k + 1$ 
  
```

endwhile

```

 $\vec{r}_k = \tilde{\mathbf{A}}\vec{x}_k + \mathbf{R}^{-T}\vec{b}$ 
 $\mathbf{R}^{-1}\vec{r}_k = \mathbf{R}^{-1}(\mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{x}_k + \mathbf{R}^{-T}\vec{b})$ 
 $\vec{r}_k = \mathbf{A}\vec{x}_k + \vec{b}$ 
 $\vec{z}_k = \mathbf{M}^{-1}\vec{r}_k$ 
endwhile
 $\vec{x} = \mathbf{R}\vec{x}_k$  is the solution

```

Table 2.11: Preconditioned conjugate gradient algorithms to solve  $\mathbf{A}\vec{x} + \vec{b} = \vec{0}$ . On the left the original algorithm, in the center with  $\vec{z}_k = \mathbf{R}^{-1}\vec{r}_k$  and on the right using  $\mathbf{R}^{-1}\vec{d}_k$  and  $\mathbf{R}^{-1}\vec{x}_k$ . The algorithm on the right might serve as starting point for an efficient implementation.

## 2.7.6 The Incomplete Cholesky Preconditioner

An incomplete Cholesky factorization is based on the standard Cholesky factorization, but does not use all of the entries of a complete factorization. There are different ideas used to drop elements:

- Keep the sparsity pattern of the original matrix  $\mathbf{A}$ , i.e. drop the entry at a position in the matrix if the entry in the original matrix is zero. This algorithm is called  $\text{IC}(0)$  and it is presented below.
- Drop the entry if its value is below a certain threshold, the drop-tolerance. This is often called the  $\text{ICT}()$  algorithm. The results of  $\text{ICT}()$  and a similar LR factorization are examined in the following section.

This construction of a preconditioner matrix  $\mathbf{R}$  is based on the Cholesky factorization of the symmetric, positive definite matrix  $\mathbf{A}$ , i.e.  $\mathbf{A} = \mathbf{R}^T \mathbf{R}$ . But we require that the matrix  $\mathbf{R}$  has the same sparsity pattern as the matrix  $\mathbf{A}$ . Those two wishes can not be satisfied simultaneously. Give up on the exact factorization and require only

$$\mathbf{R}^T \mathbf{R} = \mathbf{A} + \mathbf{E}$$

for some perturbation matrix  $\mathbf{E}$ . This leads to the conditions

1.  $r_{i,j} = 0$  if  $a_{i,j} = 0$ ,
2.  $(\mathbf{R}^T \mathbf{R})_{i,j} = a_{i,j}$  if  $a_{i,j} \neq 0$ .

To develop the algorithm use the same idea as in Section 2.6.1. The approximate factorization

$$\mathbf{A} + \mathbf{E} = \mathbf{R}^T \cdot \mathbf{R}$$

can be written using block matrices.

$$\left[ \begin{array}{c|cccc} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{1,2} & & & & \\ a_{1,3} & & & & \\ \vdots & & & & \\ a_{1,n} & & & & \end{array} \right] + \mathbf{E} = \left[ \begin{array}{c|cccc} r_{1,1} & 0 & 0 & \dots & 0 \\ r_{1,2} & & & & \\ r_{1,3} & & & & \\ \vdots & & & & \\ r_{1,n} & & & & \end{array} \right] \cdot \left[ \begin{array}{c|cccc} r_{1,1} & r_{1,2} & r_{1,3} & \dots & r_{1,n} \\ 0 & & & & \\ 0 & & & & \\ \vdots & & & & \\ 0 & & & & \end{array} \right] \cdot \mathbf{R}_{n-1}.$$

Now examine this matrix multiplication on the four submatrices. Keep track of the sparsity pattern. This translates to 4 subsystems.

- Examine the top left block (one single number) in  $\mathbf{A}$ . Obviously  $a_{1,1} = r_{1,1} \cdot r_{1,1}$  and thus

$$r_{1,1} = \sqrt{a_{1,1}}.$$

- Examine the bottom left block (column) in  $\mathbf{A}$

$$\begin{bmatrix} a_{1,2} \\ a_{1,3} \\ \vdots \\ a_{1,n} \end{bmatrix} = \begin{bmatrix} r_{1,2} \\ r_{1,3} \\ \vdots \\ r_{1,n} \end{bmatrix} \cdot r_{1,1}$$

and thus for  $2 \leq i \leq n$  and  $a_{1,i} \neq 0$  find

$$r_{1,i} = \frac{a_{1,i}}{r_{1,1}}$$

- The top right block (row) in  $\mathbf{A}$  is then already taken care of, thanks to the symmetry of  $\mathbf{A}$ .
- Examine the bottom right block in  $\mathbf{A}$ . We need

$$\mathbf{A}_{n-1} = \begin{bmatrix} r_{1,2} \\ r_{1,3} \\ \vdots \\ r_{1,n} \end{bmatrix} \cdot \begin{bmatrix} r_{1,2} & r_{1,3} & \dots & r_{1,n} \end{bmatrix} + \mathbf{R}_{n-1}^T \cdot \mathbf{R}_{n-1}.$$

For  $2 \leq i, j \leq n$  and  $a_{i,j} \neq 0$  update the entries in  $\mathbf{A}_{n-1}$  by applying

$$a_{i,j} \quad \longrightarrow \quad a_{i,j} - r_{1,i} r_{1,j} = a_{i,j} - \frac{a_{1,i} a_{1,j}}{a_{1,1}}.$$

If  $a_{1,i} = 0$  or  $a_{1,j} = 0$  there is no need to perform this step.

- Then restart the process with the reduced problem of size  $(n-1) \times (n-1)$  in the lower right block.

The above can be translated to *Octave* code without major problems. Be aware that this implementation is very far from being efficient and do not use it on large problems. This author has a faster version, but for some real speed coding in C++ is necessary. In real applications the matrices  $\mathbf{A}$  or  $\mathbf{R}$  are rarely computed. Most often a function to evaluate the matrix products has to be provided. This allows an optimal usage of the sparsity pattern.

```

function R = cholInc(A)
% R = cholInc(A) returns the incomplete Cholesky factorization
%           of the positive definite matrix A
[n,m] = size(A);
if (n ~= m) error ('cholesky: matrix has to be square'); end%if
R = zeros(size(A));
for k = 1:n
    if A(k,k)<0 error('cholInc: failed, might be singular'); end%if
    R(k,k) = sqrt(A(k,k));
    for i = k+1:n
        if A(k,i) ~= 0 R(k,i) = A(k,i)/R(k,k); end%if
    end%for
    for j = k+1:n
        for i = j:n
            if A(j,i) ~= 0 A(j,i) = A(j,i)-A(k,j)*A(k,i)/A(k,k); end%if
        end%for
    end%for
end%for

```

The efficiency of the above preconditioner is determined by the change in condition number for the matrices  $\mathbf{A}$  and  $\tilde{\mathbf{A}} = \mathbf{R}^{-T} \mathbf{A} \mathbf{R}^{-1}$ . As an example consider the model matrix  $\mathbf{A}_{nn}$  from Section 2.3.2 for different values of  $n$ , leading to Table 2.12. Based on equation (2.6) these results might make a big difference for the computation time. In [AxelBark84] a **modified incomplete Cholesky factorization** is presented and you may find the statement that the condition number for the model matrices  $\mathbf{A}_n$  and  $\mathbf{A}_{nn}$  decreases to order  $n$  (instead of  $n^2$ ). This is considerably better than the result indicated by Table 2.12. It will also change the results in Table 2.9 and Figure 2.17 in favor of the preconditioned conjugate gradient method. In the following section find numerical results on the performance of the Cholesky preconditioner with a drop-tolerance.

$n$	size of $\mathbf{A}_{nn}$	$\kappa(\mathbf{A}) \approx$	$\kappa(\tilde{\mathbf{A}}) \approx$
10	$100 \times 100$	48.2	5.1
20	$400 \times 400$	178	16.6
40	$1'600 \times 1'600$	681	61
80	$6'400 \times 6'400$	2'658	236
160	$25'600 \times 25'600$	10'500	949
320	$102'400 \times 102'400$	41'500	3691
520	$270'400 \times 270'400$	109'600	9723

Table 2.12: The condition numbers  $\kappa$  using the incomplete Cholesky preconditioner  $\text{IC}(0)$ 

**2–42 Observation :** The preconditioner will reduce the condition number for the conjugate gradient iteration, but it does have no effect on the condition number of the original matrix  $\mathbf{A}$ . Instead of solving  $\mathbf{A}\vec{x} = \vec{b}$  the modified equation  $\tilde{\mathbf{A}}\vec{u} = \mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1}\vec{u} = \mathbf{R}^{-T}\vec{b}$  is examined, where  $\vec{u} = \mathbf{R}\vec{x}$ . Thus the sequence of solvers is

$$\mathbf{A}\vec{x} = \vec{b} \iff \begin{cases} \mathbf{R}^T \vec{b}_{\text{new}} = \vec{b} & \text{solve for } \vec{b}_{\text{new}} \text{ top to bottom} \\ (\mathbf{R}^{-T}\mathbf{A}\mathbf{R}^{-1})\vec{u} = \vec{b}_{\text{new}} & \text{solve for } \vec{u} \text{ by conjugate gradient} \\ \mathbf{R}\vec{x} = \vec{u} & \text{solve for } \vec{x} \text{ bottom up} \end{cases}$$

The first and last step of the algorithm will increase the condition number again, thus the conditioning of solving  $\mathbf{A}\vec{x} = \vec{b}$  is not improved.

As an example consider the exact Cholesky factorization  $\mathbf{R}$  of  $\mathbf{A}$ , then  $\tilde{\mathbf{A}} = \mathbf{R}^{-T}\mathbf{A}\mathbf{R} = \mathbb{I}$  and thus  $\kappa(\tilde{\mathbf{A}}) = 1$ . A test with Octave for the matrix  $\mathbf{A}_{nn}$  with  $n = 200$  shows  $\kappa(\mathbf{A}_{nn}) \approx 2.4 \cdot 10^4$  and  $\kappa(\mathbf{R}) \approx 211$ , resp.  $\kappa(\mathbf{R})^2 \approx 4.4 \cdot 10^4$ . Thus the overall condition number of the problem does not change. For an incomplete Cholesky preconditioner with a small drop tolerance the result will be similar.  $\diamond$

## 2.7.7 Conjugate Gradient Algorithm with an Incomplete Cholesky Preconditioner

The model problem with a  $520 \times 520$  grid leads to a matrix with  $270'400$  rows and columns and with MATLAB or Octave use the command `ichol()` to determine incomplete Cholesky factorizations  $\mathbf{A} \approx \mathbf{L}\mathbf{L}^T$ . Since MATLAB/Octave can estimate condition numbers and then use equation (2.6) to estimate the number of iterations required to improve the accuracy by  $D = 5$  decimal digits. Without preconditioner approximately 1900 iterations might be required, based on the condition number  $\kappa \approx 110'000$ . The computations were performed on an Intel Haswell I7-5930 with a base frequency of 3.5 GHz.

- For strictly positive drop tolerances  $\text{droptol} > 0$  the `ICT()` algorithm is applied. Only entries satisfying the condition `abs(L(i, j)) >= droptol * norm(A(j:end, j), 1)` are stored in the left triangular matrix  $\mathbf{L}$ . The command `ichol()` generates this result by

```
opts.type = 'ict';
opts.droptol = droptol;
L = ichol(A,opts);
```

For  $\text{droptol} = 0$  the incomplete Cholesky factorization  $\text{IC}(0)$  is performed by calling

```
opts.type = 'nofill';
L = ichol(A,opts);
```

- For a system  $\mathbf{A} \vec{x} = \vec{b}$  the iteration terminates if  $\|\mathbf{A} \vec{x} - \vec{b}\| \leq \text{tol} \cdot \|\vec{b}\|$ . At first a relative error of  $10^{-5}$  was asked for, then an improvement by another factor  $10^{-2}$ .

Find the results<sup>19</sup> in Tables 2.13 (MATLAB) and 2.14 (Octave). The following facts can be observed:

- The memory required for  $\mathbf{L}$  increases for smaller drop-tolerances. The matrix for  $\text{IC}(0)$  requires considerably less memory.
- The norm of the error matrix  $\mathbf{E} = \mathbf{L} \cdot \mathbf{L}^T - \mathbf{A}$  decreases for smaller drop-tolerances, i.e. the preconditioned matrix is a better approximation of the original matrix.
- The condition number  $\kappa$  moves closer to the ideal value of 1 for smaller drop-tolerances. MATLAB and Octave obtain identical condition numbers. This is no surprise, as the same algorithm is used.
- Using the estimated condition number we can estimate the number of iterations required to achieve the desired relative tolerance. The actually observed number of iterations is rather close to the estimate. The number of iterations for MATLAB and Octave are identical, and all considerably smaller than the  $\approx 1900$  iterations without preconditioner.
- MATLAB is faster than Octave for computing the factorization, since MATLAB uses a multithreaded version of `ichol()`. Octave is faster than MATLAB to perform the conjugate gradient iterations. The required computation time for **one iteration** increases for smaller drop-tolerances. This is caused by the larger factorization matrix  $\mathbf{L}$ .
- It takes very little time to determine the  $\text{IC}(0)$  factorization, but the condition number of the preconditioned system is large and thus it takes many iterations to achieve the desired accuracy.

drop tolerance	0.001	0.0003	0.0001	3e-5	1e-5	3e-6	1e-6	$\text{IC}(0)$
time <code>ichol</code> [sec]	0.12	0.21	0.40	0.82	1.52	3.06	6.29	0.06
size <code>ichol</code> [MB]	57.9	87.4	136.9	205.6	313.4	471.5	696.2	15.1
norm $\ \mathbf{E}\ /\ \mathbf{A}\ $	2.8e-3	1.2e-3	4.6e-4	1.9e-4	7.3e-5	2.8e-5	1.0e-5	7.4e-2
cond for PCG	343.74	148.19	56.16	24.52	10.16	4.52	2.27	9700
estim. # of runs	107	70	44	29	19	13	9	570
first run, # of runs	70	46	29	19	12	9	6	349
first run, time [sec]	2.69	2.14	1.57	1.34	1.13	1.12	1.05	10.32
restart, # of runs	27	18	10	7	5	2	2	146
restart, time [sec]	1.08	0.85	0.55	0.50	0.47	0.27	0.36	4.28

Table 2.13: Performance for MATLAB's `pcg()` with an incomplete Cholesky preconditioner

The above results show that large systems of linear equations can be solved quickly. For the above problem on a  $1000 \times 1000$  grid we have to solve a system of 1 million unknowns. Leading to the results in Table 2.15. If the same system has to be solved many times for different right hand sides this table also shows that huge systems can be solved within a few seconds. This applies to dynamic problems, where a good initial guess is provided by the previous time step.

<sup>19</sup>Most of the computational effort for these tables goes towards estimating the condition numbers! This is not shown in the tables. With new versions of Octave the command `pcg()` will estimate the extreme eigenvalues rather quickly by using the output variable `EIGEST`.

drop tolerance	0.001	0.0003	0.0001	3e-5	1e-5	3e-6	1e-6	IC(0)
time <code>ichol</code> [sec]	0.20	0.47	0.97	1.87	3.71	7.30	14.41	0.08
size <code>ichol</code> [MB]	57.9	87.4	136.9	205.6	313.4	471.5	696.2	15.1
norm $\ \mathbf{E}\ /\ \mathbf{A}\ $	2.7e-3	1.2e-3	4.5e-4	1.9e-4	7.3e-5	2.8e-5	1.0e-5	7.3e-2
cond for PCG	343.74	148.19	56.16	24.52	10.16	4.52	2.27	9700
first run, # of runs	70	46	29	19	12	9	6	349
first run, time [sec]	1.54	1.25	1.06	0.93	0.85	0.97	0.92	5.89
restart, # of runs	27	18	10	7	5	2	2	146
restart, time [sec]	0.60	0.50	0.36	0.35	0.36	0.22	0.31	2.52

Table 2.14: Performance for Octave's `pcg()` with an incomplete Cholesky preconditioner

	drop-tolerance	size $\mathbf{L}$	time <code>ichol()</code>	time PCG	# iterations
Octave	1e-3	215 MB	1.4 sec	11.1 sec	126
MATLAB	1e-3	215 MB	0.51 sec	9.9 sec	126
Octave	1e-5	1'197 MB	16.5 sec	6.3 sec	23
MATLAB	1e-5	1'197 MB	6.17 sec	6.4 sec	23

Table 2.15: Timing for solving a system with  $10^6 = 1'000'000$  unknowns

### 2.7.8 Conjugate Gradient Algorithm with an Incomplete LU Preconditioner

not in class

For the above problem one can also use an incomplete LU factorizations as preconditioners. The actual algorithm used is `ILUC` (Crout's version). This can be used if the matrix  $\mathbf{A}$  is not symmetric and positive definite. The corresponding iterative algorithms will be examined in the next section, e.g GMRES and BICG. Octave/MATLAB can compute incomplete LU factorizations with the command `ilu()`. The results in Tables 2.16 and 2.17 show the following information:

- The drop tolerance used.
- The time required to perform the incomplete LU factorization. Surprisingly the computation time seems not to depend on the drop tolerance used. Octave is faster than MATLAB.
- The memory required to perform the incomplete LU factorization. Smaller values for the drop tolerance lead to larger matrices  $\mathbf{L}$  and  $\mathbf{U}$ . Octave uses less memory than MATLAB.
- An estimate of the norm of the matrix  $\mathbf{E} = \mathbf{L} \cdot \mathbf{U} - \mathbf{A}$ . Smaller values for the drop tolerance lead to a smaller norm of the error matrix. The results for MATLAB and Octave are identical.
- An estimate of the condition number of the matrix  $\tilde{\mathbf{A}} = \mathbf{L}^{-1} \mathbf{A} \mathbf{R}^{-1}$ . Smaller values for the drop tolerance lead to a smaller condition number.
- An estimate of the required number of iterations to improve the result by 5 digits. The estimate is based on (2.7), i.e.

$$k \geq \frac{D \ln 10}{2} \sqrt{\kappa}.$$

- The effective number of iterations required to improve the result by 5 digits. The results illustrate that the theoretical bounds are rather close to the actual number of iterations. The results for MATLAB and Octave are identical.

- The time required to run the iteration. At first the CPU time decreases, then it might increase again, caused by the larger preconditioner matrices.

The computations shown were performed on an Intel Haswell I7-5930 system at 3.5 GHz.

drop tolerance	0.001	0.0003	0.0001	3e-5	1e-5	3e-6	1e-6	ILU(0)
time ILU [sec]	61.50	60.74	62.17	62.97	63.38	68.34	76.88	0.08
size ILU [MB]	124.3	191.5	298.5	458.3	699.1	1049.2	1527.8	30.3
norm $\ \mathbf{E}\ /\ \mathbf{A}\ $	2.5e-3	9.6e-4	3.8e-4	1.5e-4	5.9e-5	2.1e-5	7.7e-6	7.3e-2
cond for PCG	316	121.4	48.64	19.48	8.10	3.69	1.96	
estim. # of runs	103	64	40	25	16	11	8	
first run, iterations	67	42	26	17	11	8	6	349
first run, time [sec]	1.56	1.23	1.03	0.92	0.90	0.95	1.02	5.88
restart, iterations	26	15	10	6	4	2	1	146
restart, time [sec]	0.61	0.44	0.39	0.33	0.33	0.24	0.18	2.48

Table 2.16: Performance of Octave's `pcg()` with an `ilu()` preconditioner

drop tolerance	0.001	0.0003	0.0001	3e-5	1e-5	3e-6	1e-6	ILU(0)
time ILU [sec]	140.41	136.47	139.33	137.10	136.16	139.99	147.88	0.08
size ILU [MB]	134.7	193.5	371.8	528.6	803.2	1131.5	1678.2	30.3
norm $\ \mathbf{E}\ /\ \mathbf{A}\ $	2.5e-3	9.7e-4	3.8e-4	1.5e-4	5.9e-5	2.2e-5	7.8e-6	7.4e-2
cond for PCG	316	121.4	48.64	19.48	8.10	3.69	1.96	
first run, iterations	67	42	26	17	11	8	6	349
first run, time [sec]	2.90	2.08	1.58	1.32	1.14	1.10	1.12	10.79
restart, iterations	26	15	10	6	4	2	1	146
restart, time [sec]	1.12	0.74	0.61	0.46	0.43	0.28	0.20	4.36

Table 2.17: Performance of MATLAB's `pcg()` with an `ilu()` preconditioner

## 2.8 Iterative Solvers for Non-Symmetric Systems

The conjugate gradient algorithm of the previous section is restricted to symmetric, positive definite matrices  $\mathbf{A}$ . It is absolutely necessary to adapt the method to nonsymmetric matrices.

### 2.8.1 Normal Equation, Conjugate Gradient Normal Residual (CGNR) and BiCGSTAB

The method of the **normal equation** is based on the fact that for an invertible matrix  $\mathbf{A}$

$$\mathbf{A} \vec{x} = \vec{b} \iff \mathbf{A}^T \mathbf{A} \vec{x} = \mathbf{A}^T \vec{b}.$$

The expression can be generated by minimizing the norm of the residual  $\vec{r} = \mathbf{A} \vec{x} - \vec{b}$ .

$$\begin{aligned} f(\vec{x}) &= \|\mathbf{A} \vec{x} - \vec{b}\|^2 = \langle \mathbf{A} \vec{x} - \vec{b}, \mathbf{A} \vec{x} - \vec{b} \rangle = \langle \vec{x}, \mathbf{A}^T \mathbf{A} \vec{x} \rangle - 2 \langle \vec{x}, \mathbf{A}^T \vec{b} \rangle + \langle \vec{b}, \vec{b} \rangle \\ \nabla f(\vec{x}) &= 2(\mathbf{A}^T \mathbf{A} \vec{x} - \mathbf{A}^T \vec{b}) = \vec{0} \end{aligned}$$

Since

$$\langle \vec{x}, \mathbf{A}^T \mathbf{A} \vec{y} \rangle = \langle \mathbf{A} \vec{x}, \mathbf{A} \vec{y} \rangle = \langle \mathbf{A}^T \mathbf{A} \vec{x}, \vec{y} \rangle \quad \text{and} \quad \langle \vec{x}, \mathbf{A}^T \mathbf{A} \vec{x} \rangle = \langle \mathbf{A} \vec{x}, \mathbf{A} \vec{x} \rangle = \|\mathbf{A} \vec{x}\|^2$$

the square matrix  $\mathbf{A}^T \mathbf{A}$  is symmetric and positive definite. Thus the conjugate gradient algorithm can be applied to the modified problem

$$(\mathbf{A}^T \mathbf{A}) \vec{x} = \mathbf{A}^T \vec{b}.$$

The computational effort for each iteration step is approximately doubled, as we have to multiply the given vector by the matrix  $\mathbf{A}$  and then by its transpose  $\mathbf{A}^T$ . A more severe disadvantage stems from the fact

$$\kappa(\mathbf{A}^T \mathbf{A}) = (\kappa(\mathbf{A}))^2$$

and thus the convergence is usually slow. Using the normal equation is almost never a good idea. The above idea can be slightly modified, leading to the conjugate residual method or conjugate gradient normal residual method (CGNR), see e.g. [Saad00, §6.8], [LascTheo87, §8.6.2] or [GoluVanLoan13, §11.3.9]. The rate of convergence is related to  $\kappa(\mathbf{A})$ , but not  $\sqrt{\kappa(\mathbf{A})}$  as for the conjugate gradient method.

Possibly good choices are the Generalized Minimal Residual algorithm (GMRES) or the BiConjugate Gradients Stabilized method (BiCGSTAB). A good description is given in [Saad00, §6.5, 7.4.2]. For both algorithms preconditioning can and should be used. All of the above algorithms are implemented in Octave.

### 2.8.2 Generalized Minimal Residual, GMRES and GMRES(m)

The above disadvantage can be avoided by a method developed by Saad and Schultz in 1986. The algorithm is based on Krylov subspace methods. Below find a short outline of the basic ideas for this algorithm. A detailed presentation of these methods is beyond the scope of these notes and can be found in the literature, e.g. [Saad00]. The theoretical and practical convergence behavior is nontrivial and examined in current research projects. In [Saad00] it is shown that the condition number  $\kappa(\mathbf{A})$  enters in the estimates.

Since the GMRES algorithm uses a sizable number of tools from linear algebra, we try to bring some structure in the presentation.

- First present the basic idea of GMRES.
- The restarted version GMRES(m) is worth to be implemented
- The convergence results are given.
- To solve least square problems orthogonal matrices are used.
- The Arnoldi iteration is one of the building blocks for GMRES.
- Hessenberg matrices and Given's rotations are used as an efficient tool.

#### The GMRES algorithm

The basic idea of the GMRES (Generalized Minimum RESiduals) algorithm is to solve a least square problem at each step of the iteration. We assume that the matrix  $\mathbf{A}$  is of size  $N \times N$ . Examine an  $n$  dimensional Krylov space  $K_n$

$$K_n = \text{span}\{\vec{b}_0, \mathbf{A}\vec{b}_0, \mathbf{A}^2\vec{b}_0, \mathbf{A}^3\vec{b}_0, \dots, \mathbf{A}^{n-1}\vec{b}_0\}$$

with an orthonormal base  $\mathbf{Q}_n = [\vec{q}_1, \vec{q}_2, \vec{q}_3, \dots, \vec{q}_n]$ , i.e. the vectors  $\vec{q}_j$  are given by the columns of the  $N \times n$  matrix  $\mathbf{Q}_n$  with  $\mathbf{Q}_n^T \cdot \mathbf{Q}_n = \mathbb{I}_n$ . For a given starting vector  $\vec{x}_0$  we seek a solution in the affine subspace  $\vec{x}_0 + K_n$ . The exact solution  $\vec{x}_0 + \vec{x}^* = \mathbf{A}^{-1}\vec{b}$  is approximated by a vector  $\vec{x}_0 + \vec{x}_n$ , such that

$$\min_{\vec{x} \in K_n} \|\mathbf{A}(\vec{x}_0 + \vec{x}) - \vec{b}\| = \|\vec{r}_n\| = \|\mathbf{A}\vec{x}_n + \mathbf{A}\vec{x}_0 - \vec{b}\| = \|\mathbf{A}\vec{x}_n - \vec{b}_0\|.$$

only brief  
remark in  
class

Thus we minimize the residual on the Krylov subspace  $K_n$ , hereby justifying the name GMRES. This is different from the conjugate gradient algorithm, where the energy norm  $\langle \mathbf{A}(\vec{u}_n - \vec{u}_{exact}), (\vec{u}_n - \vec{u}_{exact}) \rangle$  is minimized.

The matrix  $\mathbf{Q}_n$  is generated by the Arnoldi algorithm, see page 96. The vector  $\vec{b}_0 = \vec{b} - \mathbf{A}\vec{x}_0$  is the first residual. Using the matrix  $\mathbf{Q}_n$  the above can be rephrased (use  $\mathbf{Q}_n\vec{y} = \vec{x}$ ) as

$$\min_{\vec{y} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{Q}_n\vec{y} - \vec{b}_0\| = \|\vec{r}_n\| = \|\mathbf{A}\vec{x}_n - \vec{b}_0\|$$

If we use the initial vector  $\vec{q}_1 = \frac{1}{\|\vec{b}_0\|} \vec{b}_0$  for the Arnoldi iteration we know that  $\langle \vec{q}_k, \vec{b}_0 \rangle = 0$  for all  $k \geq 2$  and thus  $\vec{b}_0$  has to be a multiple of  $\vec{q}_1$ , and  $\vec{q}_1$  is the first column of  $\mathbf{Q}_{n+1}$ .

$$\vec{b}_0 = \|\vec{b}_0\| \vec{q}_1 = \|\vec{b}_0\| \mathbf{Q}_{n+1} \vec{e}_1$$

With the Arnoldi iteration we have  $\mathbf{A} \mathbf{Q}_n = \mathbf{Q}_{n+1} \mathbf{H}_n \in \mathbb{M}^{N,n}$  and thus<sup>20</sup> we have a smaller least square problem to be solved.

$$\min_{\vec{y} \in \mathbb{R}^n} \left\| \mathbf{Q}_{n+1} \mathbf{H}_n \vec{y} - \|\vec{b}_0\| \mathbf{Q}_{n+1} \vec{e}_1 \right\| = \min_{\vec{y} \in \mathbb{R}^n} \left\| \mathbf{H}_n \vec{y} - \|\vec{b}_0\| \vec{e}_1 \right\|.$$

- The large least square problem with the matrix  $\mathbf{A}$  of size  $N \times N$  is replaced with an equivalent least square problem with the Hessenberg matrix  $\mathbf{H}_n$  of size  $(n+1) \times n$ .
- This least square problem can be solved by the algorithm of your choice, e.g. a QR factorization. Using Givens transformations the QR algorithm is very efficient for Hessenberg matrices ( $\approx n^2$  operations), see page 97.
- The algorithm can be stopped if the desired accuracy is achieved, e.g. if  $\|\mathbf{A}\vec{x}_n - \vec{b}_0\|/\|\vec{b}\|$  is small enough.

This leads to the basic GMRES algorithm in Figure 2.18.

---

```

1 Choose  $\vec{x}_0$  and compute  $\vec{b}_0 = \vec{b} - \mathbf{A}\vec{x}_0$ 
2  $\mathbf{Q}_1 = \vec{q}_1 = \frac{1}{\|\vec{b}_0\|} \vec{b}_0$ 
3 for  $n = 1, 2, 3, \dots$  do
4   | Arnoldi step  $n$  to determine  $\vec{q}_{n+1}$  and  $\mathbf{H}_n$ 
5   | minimize  $\left\| \mathbf{H}_n \vec{y} - \|\vec{b}\| \vec{e}_1 \right\|$ 
6   |  $\vec{x}_n = \mathbf{Q}_n \vec{y}$ 
7 end
8  $\vec{x}^* \approx \vec{x}_0 + \vec{x}_n$ 
```

---

Figure 2.18: The GMRES algorithm

### The GMRES(m) algorithm

The basic GMRES algorithm usually performs well, but has a few key disadvantages if many iterations are asked for.

- The size of the matrices  $\mathbf{Q}_n$  and  $\mathbf{H}_n$  increases, and with them the computational effort.

<sup>20</sup>

$$\|\mathbf{Q}_{n+1}\vec{z}\|^2 = \langle \mathbf{Q}_{n+1}\vec{z}, \mathbf{Q}_{n+1}\vec{z} \rangle = \langle \vec{z}, \mathbf{Q}_{n+1}^T \mathbf{Q}_{n+1}\vec{z} \rangle = \langle \vec{z}, \mathbb{I}_{n+1}\vec{z} \rangle = \|\vec{z}\|^2$$

- The orthogonality of the columns of  $\mathbf{Q}_n$  will not be perfectly maintained, caused by arithmetic errors.

To control both of these problem the GMRES algorithm is usually restarted every  $m$  steps, with typical values of  $m \approx 10 \sim 20$ . This leads to the GMRES( $m$ ) algorithm in Figure 2.19, with an outer and inner loop. Picking the optimal value for  $m$  is an art.

The above is (at best) a rough description of the basic ideas. For a good implementation the above steps of Arnoldi, Given's rotation and minimization have to be combined, and a proper termination criterion has to be used. Find a good description and pseudo code in [templates]. Find a good starting point for code at [www.netlib.org/templates/matlab/](http://www.netlib.org/templates/matlab/).

---

```

1 Choose restart parameter  $m$ 
2 Choose  $\vec{x}_0$ 
3 for  $k = 0, 1, 2, 3, \dots$  do
4    $\vec{b}_0 = \vec{b} - \mathbf{A}\vec{x}_0$ 
5    $\mathbf{Q}_1 = \vec{q}_1 = \frac{1}{\|\vec{b}_0\|} \vec{b}_0$ 
6   for  $n = 1, 2, \dots, m$  do
7     Arnoldi step  $n$  to determine  $\vec{q}_{n+1}$  and  $\mathbf{H}_n$ 
8     minimize  $\|\mathbf{H}_n \vec{y} - \|\vec{b}\| \vec{e}_1\|$ 
9      $\vec{x}_n = \mathbf{Q}_n \vec{y}$ 
10    end
11     $\vec{x}_0 = \vec{x}_0 + \vec{x}_m$ 
12 end

```

---

Figure 2.19: The GMRES( $m$ ) algorithm

---

## Convergence of GMRES

- The norm of the residuals  $\vec{r}_n = \mathbf{A}\vec{x}_n - \vec{b}$  is decreasing  $\|\vec{r}_{n+1}\| \leq \|\vec{r}_n\|$ . This is obvious since the Krylov subspace is enlarged.
- In principle GMRES will generate the exact solution after  $N$  steps, but this result is useless. The goal is to use  $n \ll N$  iterations and arithmetic errors will prevent us from using GMRES as a direct solver.
- One can construct cases where GMRES( $m$ ) does stagnate and not converge at all.
- For a fast convergence the eigenvalues of  $\mathbf{A}$  should be clustered at a point away from the origin [LiesTich05]. This is different from the conjugate gradient algorithm, where the condition number determines the rate of convergence.
- The convergence can be improved by preconditioners.

## Orthogonal matrices, QR factorization and least square problems

Assume  $\mathbf{Q}$  is a matrix with  $\mathbf{Q}^T \mathbf{Q} = \mathbb{I}$ , not necessarily square. For the least square problem with  $\mathbf{A} = \mathbf{Q}\mathbf{R}$  the expression to be minimized is

$$\begin{aligned}
\|\mathbf{A}\vec{x} - \vec{b}\|^2 &= \langle \mathbf{A}\vec{x} - \vec{b}, \mathbf{A}\vec{x} - \vec{b} \rangle = \langle \mathbf{Q}\mathbf{R}\vec{x} - \vec{b}, \mathbf{Q}\mathbf{R}\vec{x} - \vec{b} \rangle \\
&= \langle \mathbf{Q}\mathbf{R}\vec{x}, \mathbf{Q}\mathbf{R}\vec{x} \rangle - 2\langle \mathbf{Q}\mathbf{R}\vec{x}, \vec{b} \rangle + \langle \vec{b}, \vec{b} \rangle \\
&= \langle \mathbf{R}\vec{x}, \mathbf{Q}^T \mathbf{Q}\mathbf{R}\vec{x} \rangle - 2\langle \mathbf{R}\vec{x}, \mathbf{Q}^T \vec{b} \rangle + \langle \vec{b}, \vec{b} \rangle \\
&= \langle \mathbf{R}\vec{x}, \mathbf{R}\vec{x} \rangle - 2\langle \mathbf{R}\vec{x}, \mathbf{Q}^T \vec{b} \rangle + \langle \mathbf{Q}^T \vec{b}, \mathbf{Q}^T \vec{b} \rangle - \langle \mathbf{Q}^T \vec{b}, \mathbf{Q}^T \vec{b} \rangle + \langle \vec{b}, \vec{b} \rangle \\
&= \|\mathbf{R}\vec{x} - \mathbf{Q}^T \vec{b}\|^2 - \|\mathbf{Q}^T \vec{b}\|^2 + \|\vec{b}\|^2
\end{aligned}$$

and the solution  $\vec{x}$  is identical to the solution of the least square problem

$$\min_{\vec{x}} \|\mathbf{R}\vec{x} - \mathbf{Q}^T \vec{b}\|$$

with the solution

$$\vec{x} = \mathbf{R}^{-1} \mathbf{Q}^T \vec{b}.$$

For more details see Section 3.5.1.

### Arnoldi iteration

For a given matrix  $\mathbf{A} \in \mathbb{M}^{N,N}$  with  $N \geq n$  the Arnoldi algorithm generates a sequence of orthonormal vectors  $\vec{q}_n$  such that the matrix is transformed to upper Hessenberg form. The key properties are

$$\begin{aligned} \mathbf{Q}_n &= [\vec{q}_1, \vec{q}_2, \vec{q}_3, \dots, \vec{q}_n] \in \mathbb{M}^{N,n} \\ \mathbf{Q}_n^T \mathbf{Q}_n &= \mathbb{I}_n \\ \mathbf{A} \cdot \mathbf{Q}_n &= \mathbf{Q}_{n+1} \cdot \mathbf{H}_n \in \mathbb{M}^{N,n}. \end{aligned}$$

The column vectors  $\vec{q}_j$  of the matrix  $\mathbf{Q}$  have length 1 and are pairwise orthogonal. The matrix  $\mathbf{H}_n$  is of upper Hessenberg form if all entries below the first lower diagonal are zero, i.e.

$$\mathbf{H}_n = \left[ \begin{array}{cccccc} h_{11} & h_{12} & & \cdots & & h_{1n} \\ h_{21} & h_{22} & h_{23} & & \cdots & h_{2n} \\ 0 & h_{32} & h_{33} & h_{3,4} & & h_{3n} \\ 0 & 0 & h_{43} & h_{44} & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & 0 & h_{n,n-1} & h_{n,n} \\ 0 & & \cdots & & 0 & h_{n+1,n} \end{array} \right] \in \mathbb{M}^{n+1,n}.$$

The algorithm to generate the vectors  $\vec{q}_n$  and the matrix  $\mathbf{H}_n$  is not overly complicated. The last column of the matrix equation  $\mathbf{A} \cdot \mathbf{Q}_n = \mathbf{Q}_{n+1} \cdot \mathbf{H}_n \in \mathbb{M}^{N,n}$  reads as

$$\mathbf{A} \vec{q}_n = \sum_{j=1}^{n+1} h_{j,n} \vec{q}_j$$

and thus the sequence  $\vec{q}_n$  of vectors is in the Krylov space

$$K_n = \text{span}\{\vec{q}_1, \mathbf{A}\vec{q}_1, \mathbf{A}^2\vec{q}_1, \mathbf{A}^3\vec{q}_1, \dots, \mathbf{A}^{n-1}\vec{q}_1\}.$$

We also observe that

$$\vec{q}_{n+1} = \frac{1}{h_{n+1,n}} \left( \mathbf{A} \vec{q}_n - \sum_{j=1}^n h_{j,n} \vec{q}_j \right).$$

- As starting vector  $\vec{q}_1$  we may choose any vector of length 1.
- The orthogonality is satisfied if

$$0 = \langle \vec{q}_{n+1}, \vec{q}_k \rangle = \frac{1}{h_{n+1,n}} \left( \langle \mathbf{A} \vec{q}_n, \vec{q}_k \rangle - \sum_{j=1}^n h_{j,n} \langle \vec{q}_j, \vec{q}_k \rangle \right) = \frac{1}{h_{n+1,n}} (\langle \mathbf{A} \vec{q}_n, \vec{q}_k \rangle - h_{k,n} 1)$$

and thus

$$h_{k,n} = \langle \mathbf{A} \vec{q}_n, \vec{q}_k \rangle \quad \text{for } k = 1, 2, 3, \dots, n.$$

- The condition  $\|\vec{q}_{n+1}\| = 1$  then determines the value of  $h_{n+1,n}$ , i.e. use

$$h_{n+1,n} = \|\mathbf{A} \vec{q}_n - \sum_{j=1}^n h_{j,n} \vec{q}_j\|.$$

- If the above would lead to  $h_{n+1,n} = 0$  then the Krylov subspace  $K_n$  is invariant under  $\mathbf{A}$  and the GMRES algorithm will have produced to optimal solution, see [Saad00].

The algorithm can be implemented, see Figure 2.20.

- The computational effort for one Arnoldi step is given by one matrix multiplication,  $n$  scalar products for vectors of length  $N$  and the summation of  $n$  vectors of length  $N$  to generate  $\vec{q}_{n+1}$ .
- To generate all of  $\mathbf{Q}_{n+1}$  we need  $n$  matrix multiplications and  $n^2 N$  additional operations.
- For small values of  $n \ll N$  this is dominated by the matrix multiplication ( $n N^2$ ).

```

1 Choose  $\vec{q}_1$  with  $\|\vec{q}_1\| = 1$ 
2 for  $n = 1, 2, 3, \dots$  do
3   Set  $\vec{q}_{n+1} = \mathbf{A} \vec{q}_n$ 
4   for  $k = 1, 2, 3, \dots, n$  do
5      $h_{k,n} = \langle \vec{q}_{n+1}, \vec{q}_k \rangle$ 
6      $\vec{q}_{n+1} = \vec{q}_{n+1} - h_{k,n} \vec{q}_k$ 
7   end
8    $h_{n+1,n} = \|\vec{q}_{n+1}\|$  and then  $\vec{q}_{n+1} = \frac{1}{h_{n+1,n}} \vec{q}_{n+1}$ 
9 end
```

Figure 2.20: The Arnoldi algorithm

### Given's rotations applied to Hessenberg matrices

Any vector  $(x, y)$  in the plane  $\mathbb{R}^2$  can be rotated onto the horizontal axis. With the angle  $\alpha = \arctan(y/x)$  we find

$$\begin{bmatrix} \cos \alpha & +\sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \sqrt{x^2 + y^2} \\ 0 \end{pmatrix}.$$

Now we use the same idea on the upper Hessenberg matrix with an angle such that

$$\mathbf{G}^T \begin{pmatrix} h_{11} \\ h_{21} \end{pmatrix} = \begin{bmatrix} \cos \alpha & +\sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix} \begin{pmatrix} h_{11} \\ h_{21} \end{pmatrix} = \begin{pmatrix} \sqrt{h_{11}^2 + h_{21}^2} \\ 0 \end{pmatrix}$$

i.e. we apply a rotation matrix to the first two rows of  $\mathbf{H}_n$  and find a zero in the place of  $h_{2,1}$ . If applied to a matrix this is called a Givens rotation. To the modified equation we apply the same idea again to enforce  $h_{3,2} = 0$ . This can be repeated, i.e. a multiplication by  $\mathbf{G}_i$  uses a combination of rows  $i$  and  $i+1$  to replace the entry  $h_{i+1,i}$  by zero. Thus the  $\mathbf{G}_3$  Givens rotation to be applied to  $6 \times 6$  matrices is given by

$$\mathbf{G}_3 = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & c & s & \cdot & \cdot \\ \cdot & \cdot & -s & c & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{bmatrix}$$

with  $c = \cos(\alpha)$ ,  $s = \sin(\alpha)$  and the angle is  $\alpha$  such that the entry  $h_{4,3}$  will be rotated to zero. These rotation matrices are very easy to invert, since

$$\mathbf{G}^{-1} = \mathbf{G}^T \quad \text{and} \quad \mathbf{G} \mathbf{G}^T = \mathbb{I}.$$

Now multiply the Hessenberg matrix  $\mathbf{H}_n$  from the left by  $\mathbb{I}_{n+1} = \mathbf{G}_1 \mathbf{G}_1^T$ , then by  $\mathbf{G}_2 \mathbf{G}_2^T$ , then by  $\mathbf{G}_3 \mathbf{G}_3^T$ , ... Repeat until you end up with a right triangular matrix  $\mathbf{R}$ , resp  $\mathbf{H}_n = \mathbf{Q} \mathbf{R}$ .

$$\begin{aligned}
\mathbf{H}_n &= \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix} = \mathbf{G}_1 \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix} = \mathbf{G}_1 \mathbf{G}_2 \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix} \\
&= \mathbf{G}_1 \mathbf{G}_2 \mathbf{G}_3 \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix} = \mathbf{G}_1 \mathbf{G}_2 \mathbf{G}_3 \mathbf{G}_4 \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix} \\
&= \mathbf{G}_1 \mathbf{G}_2 \mathbf{G}_3 \mathbf{G}_4 \mathbf{G}_5 \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix} = \mathbf{G}_1 \mathbf{G}_2 \mathbf{G}_3 \mathbf{G}_4 \mathbf{G}_5 \mathbf{G}_6 \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix} \\
&= \mathbf{G}_1 \mathbf{G}_2 \mathbf{G}_3 \mathbf{G}_4 \mathbf{G}_5 \mathbf{G}_6 \mathbf{R} = \mathbf{Q} \mathbf{R}
\end{aligned}$$

This leads to

$$\mathbf{H}_n = \mathbf{Q} \mathbf{R} = \mathbf{G}_1^T \mathbf{G}_2^T \mathbf{G}_3^T \cdots \mathbf{G}_n^T \mathbf{R}$$

with a right matrix  $\mathbf{R} \in \mathbb{M}^{(n+1) \times n}$  and  $\mathbf{Q} \in \mathbb{M}^{(n+1) \times (n+1)}$  with  $\mathbf{Q} \cdot \mathbf{Q}^T = \mathbb{I}_{n+1}$ . Now solve the minimization problem

$$\|\mathbf{H}_n \vec{y} - \|\vec{b}_0\| \vec{e}_1\| = \|\mathbf{Q} \mathbf{R} \vec{y} - \|\vec{b}_0\| \mathbf{Q} \mathbf{Q}^T \vec{e}_1\| = \|\mathbf{R} \vec{y} - \|\vec{b}_0\| \mathbf{Q}^T \vec{e}_1\|.$$

Since the vector  $\vec{y}$  has no influence on the last component of the vector  $\mathbf{R} \vec{y}$ , minimize by using only the first  $n$  rows of  $\mathbf{R}$  and  $\mathbf{Q}$ . This leads to a solution, for more details see Section 3.5.1.

An implementation should take advantage of a few facts.

- Since

$$\mathbf{Q} = \mathbf{G}_1 \mathbf{G}_2 \mathbf{G}_3 \cdots \mathbf{G}_n \implies \mathbf{Q}^T = \mathbf{G}_n^T \cdots \mathbf{G}_3^T \mathbf{G}_2^T \mathbf{G}_1^T$$

and the vector  $\mathbf{Q}^T \vec{p}$  is computed efficiently while the Givens rotations are determined.

- The upper Hessenberg matrix  $\mathbf{H}_n$  is transformed to the right matrix  $\mathbf{R}$  row by row. Overwrite the top rows of  $\mathbf{H}_n$  by the rows of  $\mathbf{R}$ , to save some memory.
- If the original matrix is not of upper Hessenberg form, then Given's rotations is not an efficient choice for a QR factorization. A better choice is to use Householder reflections.

## 2.9 Iterative Solvers in MATLAB/Octave and a Comparison with Direct Solvers

### 2.9.1 Iterative Solvers in MATLAB/Octave

Octave and MATLAB have a few iterative solvers already implemented, see Table 2.18. The list is not complete. For most (or all) of these implementations you have to provide either the matrix for the linear system to be solved, or a function to multiply a vector by that matrix. For most algorithms you can specify a preconditioner. Use the built-in documentation of MATLAB/Octave for more information.

<code>pcg()</code>	preconditioned conjugate gradient method
<code>bicg()</code>	bi-conjugate gradient iterative method
<code>bicgstab()</code>	stabilized bi-conjugate gradient iterative method
<code>gmres()</code>	generalized minimum residual method
<code>pqr()</code>	preconditioned conjugate residual method
<code>minres()</code>	minimum residual method
<code>qmr()</code>	quasi-minimal residual method
<code>cgs()</code>	conjugate gradients squared method

Table 2.18: Iterative solvers in Octave/MATLAB

### 2.9.2 A Comparison of Direct and Iterative Solvers

In Figure 2.21 find a general comparison of different solvers for linear systems. This figure illustrates that for different problems different algorithms should be used. On the web you can find a great variety of software for linear algebra problems. An excellent starting point is given by [[www:LinAlgFree](#)].

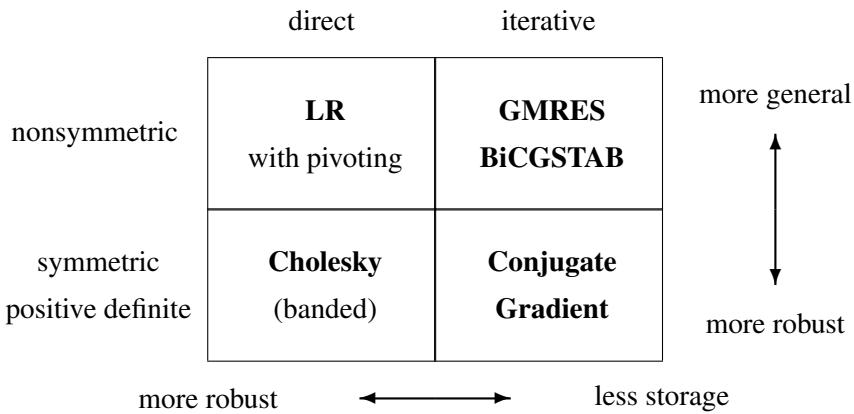


Figure 2.21: Comparison of linear solvers

The FEM (Finite Element Method) software *COMSOL* has a rather large choice of algorithms to solve the resulting systems on linear equations and some of them are capable of using multiple cores. Amongst

Algorithm	time required			
	1	2	3	4
UMFPACK, direct solver	running out of memory			
TAUCS, Cholesky based direct solver	running out of memory			
SPOOLES, direct solver	1614 s	1054 s	732 s	710 s
PARDISO, direct solver (by O.Schenk, Basel)	480 s	261 s	197 s	177 s
Conjugate Gradient, with Cholesky preconditioner	276 s	255 s	239 s	230 s
Multigrid, with PARDISO preconditioner	95 s	62 s	50 s	45 s

Table 2.19: Benchmark of different algorithms for linear systems, used with COMSOL Multiphysics

the possible model problems we picked **one**: on a cylinder with height 10 and radius 1 and a horizontal force was applied at the top and the bottom was fixed. A FEM system was set up, resulting in a system of 800'000 linear equations. Thus we have a sizable 3D problem. Then different algorithms were used to solve the system. The PC used is based on a I7-920 CPU with 4 cores and has 12 GB of RAM. Thus we have a system as presented in Section 2.2.3. Find the timing results in Table 2.19. This result is by no means representative and its main purpose is to show that **you have to choose a good algorithm**. Your conclusion will depend on the type of problems at hand, its size, the software and hardware to be used.

## 2.10 Other Matrix Factorizations

We only examined the LR (or LU) factorization and the Cholesky factorization of a matrix. There are many more factorizations and possible applications thereof.

- **SVD** Singular Value Decomposition: has many applications, see Section 3.2.6 starting on page 149.
- **QR** factorization: this factorization has many applications, including linear regression. Find a short presentation of QR used for linear regression in Section 3.5.1, starting on page 221.

In the previous chapter the codes in Table 2.20 were used.

filename	function
speed	subdirectory with C code to determine the FLOPS for a CPU
AnnGenerate.m	code to generate the a model matrix $\mathbf{A}_{nn}$
LRtest.m	code for LR factorization
cholesky.m	code for the Cholesky factorization of a matrix
choleskySolver.m	code to solve a linear system with Cholesky
cholInc.m	code for the incomplete Cholesky factorization

Table 2.20: Codes for chapter 2

## Bibliography

[Axel94] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, 1994.

- [AxelBark84] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Values Problems*. Academic Press, 1984.
- [templates] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd Edition. SIAM, Philadelphia, PA, 1994.
- [Brae02] D. Braess. *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer, second edition, 2002.
- [TopTen] B. A. Cypr. The best of the 20th century: Editors name top 10 algorithms. *SIAM News*, 2000.
- [www:LinAlgFree] J. Dongarra. Freely available software for linear algebra on the web. <http://www.netlib.org/people/JackDongarra/la-sw.html>.
- [DowdSeve98] K. Dowd and C. Severance. *High Performance Computing*. O'Reilly, 2nd edition, 1998.
- [Gold91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1), March 1991.
- [GoluVanLoan96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.
- [GoluVanLoan13] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2013.
- [Hack94] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, volume 95 of *Applied Mathematical Sciences*. Springer, first edition, 1994.
- [Hack16] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, volume 95 of *Applied Mathematical Sciences*. Springer, second edition, 2016.
- [HeroArnd01] H. Herold and J. Arndt. *C-Programmierung unter Linux*. SuSE Press, 2001.
- [Intel90] Intel Corporation. *i486 Microprocessor Programmers Reference Manual*. McGraw-Hill, 1990.
- [KnabAnge00] P. Knabner and L. Angermann. *Numerik partieller Differentialgleichungen*. Springer Verlag, Berlin, 2000.
- [LascTheo87] P. Lascaux and R. Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur, Tome 2*. Masson, Paris, 1987.
- [LiesTich05] J. Liesen and P. Tichý. Convergence analysis of Krylov subspace methods. *GAMM Mitt. Ges. Angew. Math. Mech.*, 27(2):153–173 (2005), 2004.
- [Saad00] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS, second edition, 2000. available on the internet.
- [Schw86] H. R. Schwarz. *Numerische Mathematik*. Teubner, Braunschweig, 1986.
- [Shew94] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.
- [VarFEM] A. Stahel. Calculus of Variations and Finite Elements. Lecture Notes used at HTA Biel, 2000.
- [Wilk63] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall, 1963. Republished by Dover in 1994.
- [YounGreg72] D. M. Young and R. T. Gregory. *A Survey of Numerical Analysis, Volume 1*. Dover Publications, New York, 1972.

# Chapter 3

## Numerical Tools

### 3.0.1 Prerequisites and Goals

In this chapter we will present some methods to

- solve a nonlinear equation  $f(x) = 0$  or systems of nonlinear equations  $\vec{F}(\vec{x}) = \vec{0}$ .
- work with eigenvalues and eigenvectors of matrices.
- show some aspects of SVD (singular value decomposition) and PCA (principal component analysis).
- integrate functions numerically, either given by data points or a formula.
- solve ordinary differential equations numerically.
- apply linear and nonlinear regression. This includes fitting of curves to given data.
- examine intervals and regions of confidence for parameters determined by regression.

After having worked through this chapter

- Nonlinear equations
  - you should be able to apply the methods of bisection and false positioning to solve one nonlinear equation.
  - you should be able to apply Newton's method reliably to solve one nonlinear equation.
  - you should be familiar with possible problems when using Newton's method.
  - you should be able to apply Newton's method reliably to solve systems of nonlinear equations.
- Eigenvalues and eigenvectors of matrices
  - you should understand the importance of eigenvalues and eigenvectors for linear mappings.
  - you should be able to use eigenvalues to describe the behavior of solutions of systems of linear ODEs.
  - you should understand the connection between eigenvalues and singular value decomposition.
  - you should be able to use the covariance matrix and PCA to describe the distribution of data.
- Numerical integration
  - you should be able to integrate functions given by data points using the trapezoidal rule.
  - you should understand Simpson's algorithm and Gauss integration.

- you should understand the basic idea of an adaptive integration of a given function.
- You should be able to use *Octave/MATLAB* to evaluate integrals reliably.
- Ordinary differential equations (ODE)
  - you should be able to use *MATLAB/Octave* to solve ODEs numerically, with reliable results.
  - you should understand the basic idea used for the numerical solvers and the importance of stability for the algorithms.
- Linear and nonlinear regression
  - you should be able to use the matrix notation to setup and solve linear regression problems, including the confidence intervals for the optimal parameters.
  - you should be able to setup and solve linear regression problems, including the confidence intervals for the optimal parameters.

In this chapter we assume that you are familiar with

- the definition and interpretation of eigenvalues and eigenvectors for matrices.
- the idea and computations for derivatives and linear approximations for a function of one variable.
- the idea and computations for derivatives and linear approximations for a function of multiple variables.

## 3.1 Nonlinear Equations

### 3.1.1 Introduction

When trying to solve a single equation or a system of equations

$$f(x) = 0 \quad \text{or} \quad \vec{F}(\vec{x}) = \vec{0}$$

for nonlinear functions  $f$  or  $\vec{F}$  and algebraic manipulations fail to give satisfactory results, then one has to resort to approximation methods. This will very often involve **iterative methods**. To a known value  $x_n$  apply some carefully planned operations to obtain a new value  $x_{n+1}$ . As the same operation is applied repeatedly hope for the sequence of values to converge, preferably to a solution of the original problem.

As an example pick an arbitrary value  $x_0$ , type it into your pocket calculator and then keep pushing the cos button. After a few steps you will realize that the displayed numbers converge to  $x \approx 0.73909$ . This number  $x$  solves the equation  $\cos(x) = x$ .

For all iterative methods the same questions have to be answered before launching a lengthy calculation:

- What is the computational cost of one step of the iteration?
- Will the generated sequence converge, and converge to the desired solution?
- How quickly will the sequence converge?
- How reliable is the iteration method?

This section will present some of the basic iterative methods and the corresponding results, such that you should be able to answer the above questions.

**3–1 Definition :** Let  $x_n$  be a sequence converging to  $x^*$ . This convergence is said to be of **order**  $p$  if there is a constant  $c$  such that

$$\|x_{n+1} - x^*\| \leq c \|x_n - x^*\|^p.$$

A method which produces such a sequence is said to have an **order of convergence**  $p$ .

The expression  $\log \|x_n - x^*\|$  corresponds to the number of correct (decimal) digits in the approximation  $x_n$  of the exact value  $x^*$ . Thus the order of convergence is an indication on how quickly the approximation sequence will converge to the exact solution. We examine two important cases:

- Linear convergence, convergence of order 1:

$$\begin{aligned}\|x_n - x^*\| &\leq c \|x_{n-1} - x^*\| \leq c^2 \|x_{n-2} - x^*\| \leq \dots \leq c^n \|x_0 - x^*\| \\ \log \|x_n - x^*\| &\leq \log c + \log \|x_{n-1} - x^*\| \\ \log \|x_n - x^*\| &\leq \log \|x_0 - x^*\| + n \log c\end{aligned}$$

Thus the number of accurate digits increases by a fixed number ( $\log c$ ) for each step, as long as  $c < 1$ , i.e.  $\log(c) < 0$ . In real application we do not have the exact solution to check for convergence, but we may observe the difference between subsequent values.

$$\begin{aligned}\|x_{n+1} - x_n\| &= \|(x_{n+1} - x^*) - (x_n - x^*)\| \leq \|x_{n+1} - x^*\| + \|x_n - x^*\| \\ &\leq \|x_0 - x^*\| c^{n+1} + \|x_0 - x^*\| c^n \leq \|x_0 - x^*\| (1 + c) c^n \\ \log \|x_{n+1} - x_n\| &\leq \log \|x_0 - x^*\| + \log(1 + c) + n \log(c)\end{aligned}$$

Thus expect the number of stable digits to increase by a fixed amount ( $\log c$ ) for each step.

- Quadratic convergence, convergence of order 2:

$$\begin{aligned}\|x_n - x^*\| &\leq c \|x_{n-1} - x^*\|^2 \\ \log(\|x_n - x^*\|) &\leq 2 \log \|x_{n-1} - x^*\| + \log(c)\end{aligned}$$

Thus the number of accurate digits is doubled at each step, ignoring the expression  $\log(c)$ . Once we have enough digits this simplification is justified. When observing the number of stable digits we find similarly

$$\begin{aligned}\|x_{n+1} - x_n\| &= \|(x_{n+1} - x^*) - (x_n - x^*)\| \leq c \|x_n - x^*\|^2 + \|x_n - x^*\| \\ &= (c \|x_n - x^*\| + 1) \|x_n - x^*\| \\ \log \|x_{n+1} - x_n\| &\leq \log(c \|x_n - x^*\| + 1) + \log \|x_n - x^*\| \approx 0 + \log \|x_n - x^*\|\end{aligned}$$

and consequently the number of stable digits should double at each step, at least once we are close to the actual solution<sup>1</sup>.

The effects of different orders of convergence is illustrated in Example 3–3 (see page 109), leading to Table 3.2 on page 110.

### How to stop an iteration

When a system of equations  $\vec{F}(\vec{x}) = \vec{0}$  is solved by an iterative method you end up with a sequence of vectors  $\vec{x}_n$  for  $n = 1, 2, 3, \dots$ . Then the task is to determine when to stop the iteration and accept the current result as *good enough*. Thus a good termination criterion has to be selected in advance. There are different possible options and a good choice has to be based on the concrete application and the question one has to answer<sup>2</sup>.

<sup>1</sup>Use  $\log 1 = 0$  and thus  $\log(c \|x_n - x^*\| + 1) \approx 0$  if  $c \|x_n - x^*\| \ll 1$ .

<sup>2</sup>Richard W. Hamming (1962) is said to have coined the phrase: *The purpose of computing is insight, not numbers.*

- Terminate if the absolute change in  $x$  is small enough, i.e.  $\|\vec{x}_{n+1} - \vec{x}_n\|$  is small.
- Terminate if the relative change in  $x$  is small enough, i.e. use one of the expressions  $\frac{\|\vec{x}_{n+1} - \vec{x}_n\|}{\|\vec{x}_n\|}$ ,  $\frac{\|\vec{x}_{n+1} - \vec{x}_n\|}{\|\vec{x}_{n+1}\|}$  or  $\frac{\|\vec{x}_{n+1} - \vec{x}_n\|}{\|\vec{x}_n\| + \|\vec{x}_{n+1}\|}$  as termination criterion.
- In black box solvers one should use a combination of absolute tolerance  $A$  and relative tolerance  $R$ , e.g. stop if
$$\|\vec{x}_{n+1} - \vec{x}_n\| \leq A + R \|\vec{x}_n\|$$
- Terminate if the absolute error in  $\vec{y} = \vec{F}(\vec{x})$  is small enough, i.e.  $\|\vec{F}(\vec{x}_n)\|$  is small.

### 3.1.2 Bisection, Regula Falsi and Secant Method to Solve one Equation

In this section find the basic idea for three algorithms to solve a single equation of the form  $y = f(x) = 0$ . Throughout this section assume that the function  $f$  is at least continuous, or as often differentiable as necessary. Also assume that a solution exists, i.e.  $f(x^*) = 0$ , and it is not a double zero, i.e.  $f'(x^*) \neq 0$ . Find a brief description and an illustrative graphic for each of the algorithms. Since coding of these algorithm is not too difficult, no explicit code is provided.

#### Bisection

This basic algorithm will find zeros of continuous function, once two values of  $x$  with opposite signs for  $y$  are known. The solution  $x^*$  of  $f(x^*) = 0$  will be **bracketed** by  $x_n$  and  $x_{n+1}$ , i.e.  $x^*$  is between  $x_n$  and  $x_{n+1}$ . Find the description of the algorithm below and an illustration in Figure 3.1.

- Start with two values  $x_0$  and  $x_1$  such that  $y_0 = f(x_0)$  and  $y_1 = f(x_1)$  have opposite signs, i.e.  $f(x_0) \cdot f(x_1) < 0$ . This leads to an initial interval.
- Repeat until the desired accuracy is achieved:
  - Compute the function  $y = f(x)$  at the midpoint  $x_{n+1}$  of the current interval and examine the sign of  $y$ .
  - Retain the mid point and one of the endpoints, such that the  $y$ -values have opposite signs. This is the new interval to be examined in the next iteration.

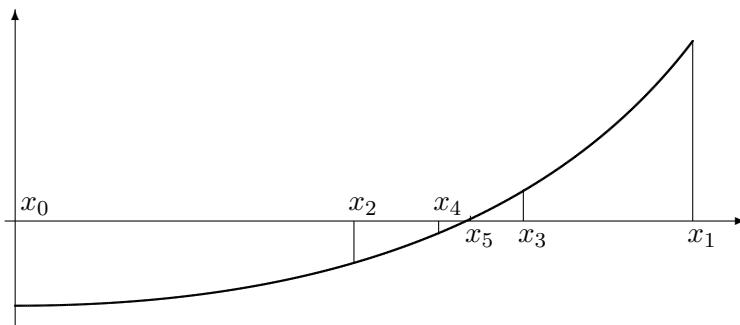


Figure 3.1: Method of bisection to solve one equation

This algorithm will always converge, since the function  $f$  is assumed to be continuous and the solution is bracketed. Obviously the maximal error is halved at each step of the iteration and we have an elementary estimate for the error

$$|x_{n+1} - x^*| \leq \frac{1}{2^n} |x_1 - x_0|.$$

Thus we find linear convergence, i.e. the number of accurate decimal digits is increased by  $\frac{\ln 2}{\ln 10} \approx 0.3$  by each step.

### False position method, Regula Falsi

This algorithm is a minor modification of the bisection method. Instead of using the midpoint of the interval we continue with the zero of the secant connecting the two endpoints. Find the illustration in Figure 3.2. It can be shown that the convergence of the algorithm is linear.

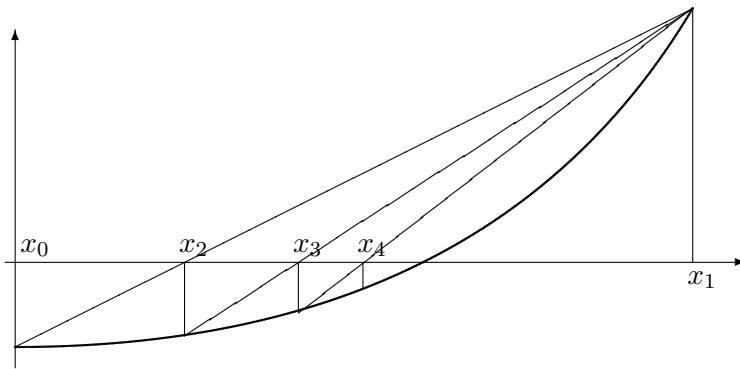


Figure 3.2: Method of false position to solve one equation

### Secant method

The two previous algorithm are guaranteed to give a solution, since the solution was bracketed. We can modify the false position method slightly and always retain the last two values, independent on the sign of the function. Find the illustration in Figure 3.3.

- Start with two values  $x_0$  and  $x_1$ , compute  $y_0 = f(x_0)$  and  $y_1 = f(x_1)$ .
- Repeat until the desired accuracy is achieved:
  - Compute the zero of the secant connecting the two given points

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n).$$

- Restart with  $x_n$  and  $x_{n+1}$ .

One can show that this algorithm has superlinear convergence.

$$|x_{n+1} - x^*| \approx c |x_n - x^*|^{1.618}.$$

This implies that the number of correct digits is multiplied by 1.6, as soon as we are close enough. This is a huge advantage over the bisection and false position methods. As a clear disadvantage we have no guaranteed convergence, even if the solution was originally bracketed. The secant might intersect the horizontal axis at a far away point and thus we might end up with a different solution than expected, or none at all. One can show that the secant method will converge to a solution  $x^*$  if the starting values are close enough to  $x^*$  and  $f'(x^*) \neq 0$ .

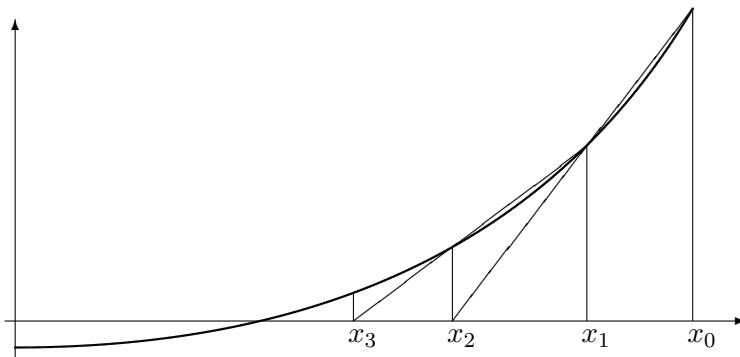


Figure 3.3: Secant method to solve one equation

### Newton's method to solve one equation

This important algorithm is based on the idea of a linear approximation. For a given estimate  $x_0$  of the zero of the function  $f(x)$  we replace the function by its linear approximation, i.e. the tangent to the curve  $y = f(x)$  at the point  $(x_0, f(x_0))$ .

$$\begin{aligned}
 f(x_0 + \Delta x) &= 0 \\
 f(x_0 + \Delta x) &\approx f(x_0) + f'(x_0) \cdot \Delta x \\
 f(x_0) + f'(x_0) \cdot \Delta x &= 0 \\
 \Delta x &= -\frac{f(x_0)}{f'(x_0)} \\
 x_1 &= x_0 + \Delta x = x_0 - \frac{f(x_0)}{f'(x_0)}
 \end{aligned}$$

The above computations lead to the algorithm of Newton, some authors call it Newton–Raphson.

- Start with a value  $x_0$  close to the zero of  $f(x)$ .
- Repeat until the desired accuracy is achieved:
  - Compute values  $f(x_n)$  and the derivative  $f'(x_n)$  at the point  $x_n$ . Apply Newton's formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

- Restart with  $x_{n+1}$

The algorithm is illustrated in Figure 3.4.

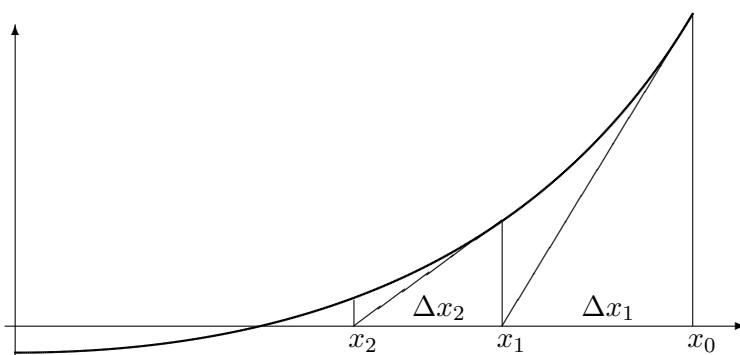


Figure 3.4: Newton's method to solve one equation

One can show that this algorithm converges quadratically

$$|x_{n+1} - x^*| \approx c |x_n - x^*|^2.$$

This implies that the number of correct digits is multiplied by 2, as soon as we are close enough. This is a huge advantage over the bisection and false position methods. As a clear disadvantage we have no guaranteed convergence. The tangent might intersect the horizontal axis at a far away point and thus we might end up with a different solution than expected, or none at all. One can show that Newton's method will converge to a solution  $x^*$  if the starting values are close enough to  $x^*$  and  $f'(x^*) \neq 0$ .

**3–2 Example :** To compute the value of  $x = \sqrt{2}$  we may try to solve the equation  $x^2 - 2 = 0$ . For this example we find

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - 2}{2x_n} = \frac{2 + x_n^2}{2x_n}.$$

With a starting value of  $x_0 = 1$  we find

$$x_1 = \frac{2 + 1}{2} = \frac{3}{2}, \quad x_2 = \frac{2 + 9/4}{3} = \frac{17}{12} \approx 1.417 \quad \text{and} \quad x_3 \approx 1.414216.$$

Thus we are very close to the actual solution with very few iteration steps.  $\diamond$

The major problem of Newton's method is based to the fact that the initial guess has to be close enough to the exact solution. If this is not the case, then we might run into severe problems. Consider the three graphs in Figure 3.5.

- The graph on the left does not have a zero (or is it a double zero?) and thus Newton's method will happily iterate along, never getting close to a solution.
- The middle graph has one solution, but if we start a Newton iteration to the right of the maximum, then the iteration will move further and further to the right. The clearly existing, unique zero will not be found.
- In the graph on the right it is easy to find starting values  $x_0$  such that Newton's method will converge, but not to the solution closest to  $x_0$ .

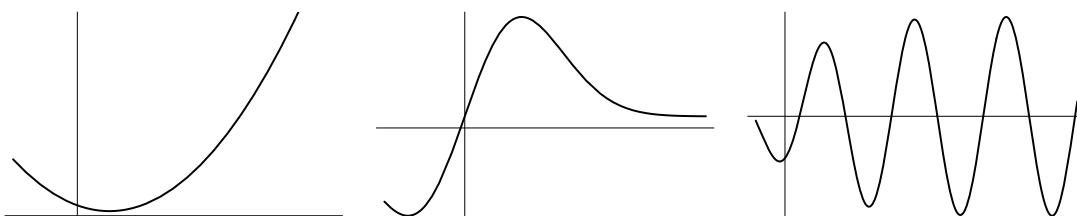


Figure 3.5: Three functions that might cause problems for Newton's methods

- Newton's method is an excellent tool to compute a zero of a function accurately, and quickly.
- Crucial for a success is the availability of a good initial guess.
- Newton's method can fail miserably when no good initial guess is available.

## Comparison

The above four algorithms have all their weak and strong points, thus a comparison is asked for:

- Bisection and False Position are guaranteed to converge to a solution, as long as the starting points  $x_0$  and  $x_1$  lead to different signs for  $f(x_0)$  and  $f(x_1)$ .
- The secant method converges faster than bisection or Regula Falsi, but the performance of Newton is hard to beat.
- The secant and Newton's method might not give the desired/expected solution or might even fail completely.
- Only Newton's methods requires values of the derivative.

Find the results in Table 3.1.

	Bisection	False Position	Secant	Newton
bracketing necessary	yes	yes	no	no
guaranteed success	yes	yes	no	no
requires derivative	no	no	no	yes
order of convergence	1	1	1.618	2

Table 3.1: Comparison of methods to solve one equation

### 3–3 Example : Performance comparison of solvers

Compute  $\sqrt{2}$  as solution of the equation  $f(x) = x^2 - 2 = 0$ . We use implementations in Octave of the above four algorithms to solve this elementary equation and keep track of the following quantities:

- The estimate of the solution at each step,  $x_n$ .
- The number of correct decimal digits: `corr`.
- The number of non-changing digits from the last iteration step: `fix`.

The results are shown in Table 3.2. There are some observations to be made:

- The number of accurate digits for the bisection method increases very slowly, but exactly in the predicted way. For 10 iterations the error has to be divided by  $2^{10} \approx 1000$ , thus we gain 3 digits only.
- The Regula Falsi method leads to linear convergence, the number of correct digits increases by a fixed number ( $\approx 0.7$ ) for each step. This is clearly superior to the method of bisection.
- The secant method leads to superlinear convergence, the number of correct digits increases by a fixed factor ( $\approx 1.6$ ) for each step. After a few steps (8) we reach machine accuracy and there is no change in the result any more.
- The Newton method converges very quickly (5 steps) up to machine precision to the exact solution. The number of correct digits is doubled at each step. This is caused by the quadratic convergence of the algorithm.
- The number of unchanged digits at each step (`fix`) is a safe estimate of the number of correct digits (`corr`). This is an important observation, since for real world problems the only available information is the value of `fix`. Computing `corr` requires the exact solution and thus there would be no need for a solution algorithm.



Bisection			Regula Falsi			Secant Method			Newton's Method		
$x_n$	corr	fix	$x_n$	corr	fix	$x_n$	corr	fix	$x_n$	corr	fix
1.000000	0.4		1.000000	0.4		1.000000	0.4		2.000000	0.2	
1.500000	1.1	0.3	1.333333	1.1	0.5	1.333333	1.1	0.5	1.500000	1.1	0.3
1.250000	0.8	0.6	1.400000	1.8	1.2	1.428571	1.8	1.0	1.416667	2.6	1.1
1.375000	1.4	0.9	1.411765	2.6	1.9	1.413793	3.4	1.8	1.414216	5.7	2.6
1.437500	1.6	1.2	1.413793	3.4	2.7	1.414211	5.7	3.4	1.414214	12	5.7
1.406250	2.1	1.5	1.414141	4.1	3.5	1.414214	9.5	5.7	1.414214	16	12
1.421875	2.1	1.8	1.414201	4.9	4.2	1.414214	15	9.5	1.414214	16	16
1.414062	3.8	2.1	1.414211	5.7	5.0	1.414214	16	16	1.414214	16	16
1.417969	2.4	2.4	1.414213	6.4	5.8	1.414214	16	16	1.414214	16	16
1.416016	2.7	2.7	1.414213	7.2	6.5	1.414214	16	16	1.414214	16	16

Table 3.2: Performance of some basic algorithms to solve  $x^2 - 2 = 0$ 

### 3.1.3 Systems of Equations

In the previous section we found that the situation to solve a single equation is rather comfortable. We have different algorithms at our disposition with different strength and weaknesses. Combined with graphical tools we should be able to examine almost all situations with reliable results.

The situation changes drastically for systems of equations and one may sum up the situation:

There is no reliable black box algorithm to solve systems of equations

- The ideas of the Bisection method, the False Position method and the Secant method can not be carried over to the situation of multiple equations.
- The method of Newton can be applied to systems of equations. This will be done in the next section. It has to be pointed out that a number of problems might occur:
  - Newton requires a good starting point to work reliably.
  - We also need the derivatives of the functions. For a system of  $n$  equations this amounts to  $n^2$  partial derivatives to be known. The computational (and programming) cost might be prohibitive.
  - For each step of Newton's method a system of  $n$  linear equations has to be solved and for very large  $n$  this might be difficult.
- There exist derivative free algorithms to solve systems of equations, e.g. Broyden's method. As a possible starting point consider [Pres92].
- If the problem is a minimization problem. i.e. you are searching for  $\vec{x} \in \mathbb{R}^n$  such that the function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  attains its minimum at  $\vec{x}$ . This leads to a system of  $n$  equations

$$\text{grad } f(\vec{x}) = \vec{0}.$$

Since  $-\text{grad } f$  is pointing in the direction of steepest descent one has good knowledge where the minimum might be. In this situation reliable and efficient algorithms are known, e.g. the Levenberg–Marquardt algorithm.

In these notes we concentrate on Newton's method and its applications. Successive substitution and partial substitution are mentioned briefly.

### 3.1.4 The Contraction Mapping Principle and Successive Substitutions

The theoretical foundation for many iterative schemes to solve systems of nonlinear equations is given by Banach's fixed point theorem, also called contraction mapping principle. This is one of the most important results in nonlinear analysis and it has many applications. We give an abstract version below and illustrate it by a few examples.

With the translation  $F(x) = G(x) + x$  it is obvious that a zero of  $G$  is a fixed point ( $F(x) = x$ ) of  $F$ .

$$G(x) = F(x) - x = 0 \iff F(x) = G(x) + x = x$$

Thus we may concentrate our efforts on efficient algorithms to locate fixed points of iterations.

**3-4 Theorem : Banach's fixed point theorem, Contraction Mapping Principle**

Let  $M$  be a closed subset of a Banach space  $E$  and the mapping  $F$  is a **contraction** from  $M$  to  $M$ , i.e. there exists a constant  $c < 1$  such that

$$F : M \longrightarrow M \quad \text{with} \quad \|F(x) - F(y)\| \leq c \|x - y\| \quad \text{for all } x, y \in M \quad . \quad (3.1)$$

Then there exists exactly one fixed point  $z \in M$  of the mapping  $F$ , i.e. one solution of  $F(z) = z$ .

For any initial point  $x_0 \in M$  the sequence formed by  $x_{n+1} = F(x_n)$  will converge to  $z$  and we have the estimate

$$\|x_{n+1} - z\| \leq c \|x_n - z\|$$

i.e. the order of convergence is at least 1. By applying the above estimate repeatedly we find the **a priori estimate**

$$\|x_n - z\| \leq c^n \|x_0 - z\|$$

i.e. we can estimate the number of necessary iterations **before** starting the algorithm. An **a posteriori estimate** is given by

$$\|x_{n+1} - z\| \leq \frac{c}{1-c} \|x_n - x_{n+1}\|$$

i.e. we can estimate the error during the computations by comparing subsequent values.  $\diamond$

The proof below is given for sake of completeness only. It is possible to work through the remainder of these notes without working through the proof, but it is advisable to understand the illustration in Figure 3.6 and the consequences of the estimates in the above theorem..

**Proof :** For an arbitrary initial point  $x_0 \in M$  we examine the sequence  $x_n = F^n(x_0)$ . not in class

$$\begin{aligned} \|F^n(x) - F^n(y)\| &\leq c \|F^{n-1}(x) - F^{n-1}(y)\| \leq c^n \|x - y\| \\ \|F^n(x_0) - F^{n+k}(x_0)\| &\leq c^n \|x_0 - F^k(x_0)\| \leq c^n \sum_{i=0}^{k-1} \|F^i(x_0) - F^{i+1}(x_0)\| \\ &\leq c^n \sum_{i=0}^{k-1} c^i \|x_0 - F(x_0)\| \leq \frac{c^n}{1-c} \|x_0 - F(x_0)\| \end{aligned}$$

Thus  $x_n$  is a Cauchy sequence and we conclude

$$x_n = F^n(x_0) \longrightarrow z \in M \quad \text{as } n \rightarrow \infty \quad .$$

Since  $F$  is continuous we conclude

$$x_{n+1} = F(x_n) \longrightarrow F(z)$$

and thus

$$F(x) = \lim x_{n+1} = \lim x_n = z.$$

If  $\bar{z}$  is also a fixed point we use the contraction property

$$\|\bar{z} - z\| = \|F(\bar{z}) - F(z)\| \leq c \|\bar{z} - z\|$$

to conclude  $\bar{z} = z$ . Thus we have a unique fixed point. To verify the linear convergence we use  $F(z) = z$  and the contraction property to conclude

$$\|x_{n+1} - z\| = \|F(x_n) - F(z)\| \leq c \|x_n - z\|.$$

To verify the à posteriori estimate we use

$$\begin{aligned}\|x_n - z\| &\leq \|x_n - x_{n+1}\| + \|x_{n+1} - z\| \leq \|x_n - x_{n+1}\| + c \|x_n - z\| \\ \|x_n - z\| &\leq \frac{1}{1-c} \|x_n - x_{n+1}\| \\ \|x_{n+1} - z\| &\leq c \|x_n - z\| \leq \frac{c}{1-c} \|x_n - x_{n+1}\|.\end{aligned}$$

□

The function  $F$  maps the set  $M$  to  $M$  and it is a **contraction**, i.e. there is a constant  $c < 1$  such that

$$\|F(x) - F(y)\| \leq c \|x - y\|$$

for all  $x, y \in M$ .

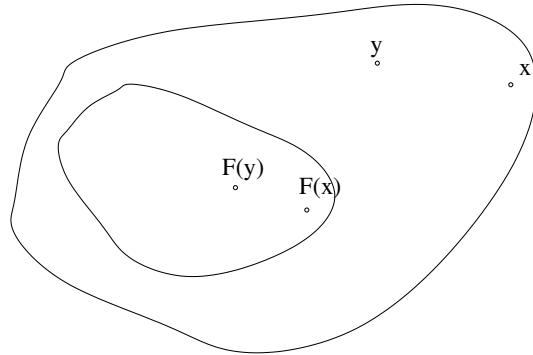


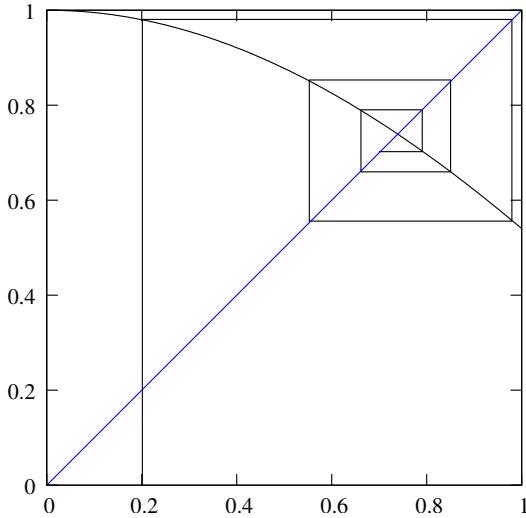
Figure 3.6: The contraction mapping principle

**3–5 Example :** The function  $f(x) = \cos(x)$  on the interval  $M = [0, 1]$  satisfies the assumptions of Banach's fixed point theorem. Obviously  $0 \leq \cos x \leq 1$  for  $0 \leq x \leq 1$  and thus  $f$  maps  $M$  into  $M$ . The contraction property is a consequence of an integral estimate.

$$\begin{aligned}\cos(x) - \cos(y) &= \int_y^x -\sin(t) dt \\ |\cos(x) - \cos(y)| &= \left| \int_y^x \sin(t) dt \right| \leq \sin(1) |x - y|.\end{aligned}$$

The contraction constant is given by  $c = \sin(1) < 1$ . As a consequence we find that the equation  $\cos(x) = x$  has exactly one solution in  $M$ . We can obtain this solution by choosing an arbitrary initial value  $x_0 \in M$  and then apply the iteration  $x_{n+1} = \cos(x_n)$ . This is illustrated in Figure 3.7. ◇

**3–6 Result :** Let  $M \subset E$  be a closed subset of a Banach space  $E$ . If a mapping  $F : M \rightarrow M$  is differentiable and the linear operator  $\mathbf{DF} \in \mathcal{L}(E, E)$  (i.e. a bounded linear operator) satisfies  $\|\mathbf{DF}(x)\| \leq c < 1$  for all  $x \in M$ , then  $F$  is a contraction. Thus the equation  $F(x) = x$  can be solved by successive substitutions  $x_{n+1} = F(x_n)$ . ◇

Figure 3.7: Successive substitution to solve  $\cos x = x$ 

**Proof :** For  $x, y \in M$  we define

$$g(\lambda) = F(x + \lambda(y - x)) \quad \text{for } 0 \leq \lambda \leq 1$$

We find  $g(0) = F(x)$  and  $g(1) = F(y)$ . The chain rule implies

$$\frac{d}{d\lambda} g(\lambda) = \mathbf{DF}(x + \lambda(y - x)) \cdot (y - x)$$

and thus

$$F(y) - F(x) = g(1) - g(0) = \int_0^1 \frac{d g(\lambda)}{d\lambda} d\lambda = \int_0^1 \mathbf{DF}(x + \lambda(y - x)) \cdot (y - x) d\lambda$$

The estimate of  $\mathbf{DF}$  now implies

$$\|F(y) - F(x)\| \leq \int_0^1 \|\mathbf{DF}(x + \lambda(y - x))\| \cdot \|y - x\| d\lambda \leq c \|y - x\|$$

and thus we have a contraction.  $\square$

### 3–7 Example : Quadratic convergence of Newton’s method

Newton’s method to solve a single equation  $f(x) = 0$  is using the iteration

$$x_{n+1} = F(x_n) = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Thus we find

$$\frac{d}{dx} F(x) = 1 - \frac{f'(x) \cdot f'(x) - f(x) \cdot f''(x)}{(f'(x))^2}.$$

If  $f(x^*) = 0$  and  $f'(x^*) \neq 0$  we conclude

$$\frac{d}{dx} F(x^*) = 1 - \frac{f'(x^*) \cdot f'(x^*) - 0}{(f'(x^*))^2} = 0.$$

If the function  $f$  is twice continuously differentiable we can conclude that in a neighborhood of  $x^*$  the derivative satisfies  $|\frac{d}{dx} F(x)| \leq \frac{1}{2}$  and thus  $F$  is a contraction. The proof shows that the contraction constant  $c$  gets closer to 0 as the approximate solution  $x_n$  approaches the exact solution  $x^*$ . Based on this idea one can prove the quadratic convergence of Newton’s method. The result remains valid in the Banach space context, see e.g. [Deim84, Theorem 15.6]. A precise result is shown in [Linz79, §5.3], without proof. The situation of  $n$  equations for  $n$  unknown is also examined carefully in [IsaaKell66].  $\diamond$

### Partial successive substitution

There are problems when it is advantageous to modify the method of successive substitutions. If we have a function  $F(x, y)$  and we want to solve  $F(x, x) = x$  we can use successive substitutions on one of the arguments only.

- Start with an initial value  $x_0$ .
- Repeat until the error is small enough
  - Use the known value of  $x_n$  and solve the equation  $F(x_n, x_{n+1}) = x_{n+1}$  for the unknown  $x_{n+1}$ .

As a trivial example try to solve the nonlinear equation  $3 + 3x = e^x$ . Given  $x_n$  you can solve  $3 + 3x_{n+1} = e^{x_n}$  for  $x_{n+1}$  by

$$x_{n+1} = \frac{1}{3}(e^{x_n} - 3).$$

A simple graph (Figure 3.8) will convince you that the equation has two solutions, one close to  $x \approx -1$  and the other close to  $x \approx 2.5$ . Choosing  $x_0 = -1$  will converge to the solution, but  $x_0 = 2.5$  will not converge at all. This shows that even for simple examples the method can fail. Run SuccSub.m

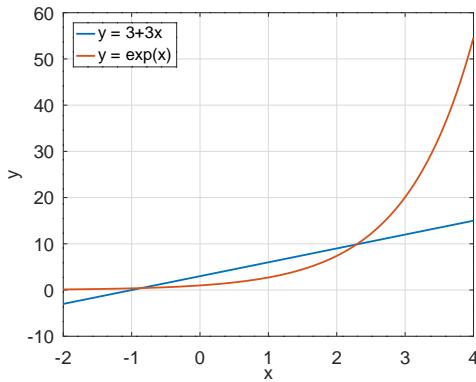


Figure 3.8: Partial successive substitution to solve  $3 + 3x = \exp(x)$

Another example is given by the stretching of a beam in equation (1.14) on page 17. To solve the nonlinear boundary value problem

$$-\frac{d}{dx} \left( E A_0(x) \left( 1 - \nu \frac{du(x)}{dx} \right)^2 \frac{du(x)}{dx} \right) = f(x) \quad \text{for } 0 < x < L$$

use a known function  $u_n(x)$  to compute the coefficient function

$$a(x) = E A_0(x) \left( 1 - \nu \frac{du_n(x)}{dx} \right)^2$$

and then solve the linear boundary value problem

$$-\frac{d}{dx} \left( a(x) \frac{du_{n+1}(x)}{dx} \right) = f(x)$$

for the next approximation  $u_{n+1}(x)$ . Using the finite difference method this will be used in Example 4-9 on page 296.

The above approach is sometimes called a Picard iteration.

### 3.1.5 Newton's Algorithm to Solve Systems of Equations

In a previous section we used Newton's method to solve a single equation. The ideas can be applied to systems of equations. We first use the algorithm to solve two equations in two unknowns.

#### Newton's algorithm to solve two equations with two unknowns

Search a solution of two equations in two unknowns

$$\begin{cases} f(x, y) = 0 \\ g(x, y) = 0 \end{cases}.$$

To simplify the problem replace the nonlinear function  $f$  and  $g$  by linear approximations about the initial point  $(x_0, y_0)$ .

$$\begin{cases} f(x_0 + \Delta x, y_0 + \Delta y) \approx f(x_0, y_0) + \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial y} \Delta y \\ g(x_0 + \Delta x, y_0 + \Delta y) \approx g(x_0, y_0) + \frac{\partial g}{\partial x} \Delta x + \frac{\partial g}{\partial y} \Delta y \end{cases}.$$

Thus replace the original equations by a set of approximate linear equations. This leads to equations for the unknowns  $\Delta x$  and  $\Delta y$ .

$$\begin{cases} f(x_0, y_0) + \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial y} \Delta y = 0 \\ g(x_0, y_0) + \frac{\partial g}{\partial x} \Delta x + \frac{\partial g}{\partial y} \Delta y = 0 \end{cases}$$

Often a shortcut notation is used

$$f_x = \frac{\partial f}{\partial x}, \quad f_y = \frac{\partial f}{\partial y},$$

and thus the approximate linear equations can be written in the form

$$\begin{pmatrix} f(x_0, y_0) \\ g(x_0, y_0) \end{pmatrix} + \mathbf{A} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

where the  $2 \times 2$  matrix  $\mathbf{A}$  of partial derivatives is given by

$$\mathbf{A} = \begin{bmatrix} f_x(x_0, y_0) & f_y(x_0, y_0) \\ g_x(x_0, y_0) & g_y(x_0, y_0) \end{bmatrix}.$$

If the matrix is invertible<sup>3</sup> the solution is given by

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -\mathbf{A}^{-1} \cdot \begin{pmatrix} f(x_0, y_0) \\ g(x_0, y_0) \end{pmatrix}.$$

Just as in the situation of a single equation find a (hopefully) better approximation of the true zero.

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

<sup>3</sup>If the determinant is different from zero use the formula

$$\mathbf{A}^{-1} = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}^{-1} = \frac{1}{f_x g_y - g_x f_y} \begin{bmatrix} g_y & -f_y \\ -g_x & f_x \end{bmatrix}$$

run demos  
Newton2D.  
TangentPl

This leads to an iterative formula for Newton's method applied to a system of two equations.

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - \mathbf{A}^{-1} \cdot \begin{pmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{pmatrix}.$$

where

$$\mathbf{A} = \begin{bmatrix} f_x(x_n, y_n) & f_y(x_n, y_n) \\ g_x(x_n, y_n) & g_y(x_n, y_n) \end{bmatrix}$$

This iteration formula is, not surprisingly, very similar to the formula for a single equation.

$$x_{n+1} = x_n - \frac{1}{f'(x_n)} f(x_n)$$

**3–8 Example :** Examine the equations

$$\begin{aligned} x^2 + 4y^2 &= 1 \\ 4x^4 + y^2 &= 1 \end{aligned}$$

with the estimated solutions  $x_0 = 1$  and  $y_0 = 1$ . We want to apply a few steps of Newton's method.

Ex. 3.2

With  $f_1(x, y) = x^2 + 4y^2 - 1$  and  $f_2(x, y) = 4x^4 + y^2 - 1$  we find the partial derivatives

$$\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x & 8y \\ 16x^3 & 2y \end{bmatrix}$$

and for  $(x_0, y_0) = (1, 1)$  we have the values  $f_1(x_0, y_0) = 4$  and  $f_2(x_0, y_0) = 4$  and we find a system of linear equations for  $x_1$  and  $y_1$ .

$$\begin{pmatrix} 4 \\ 4 \end{pmatrix} + \begin{bmatrix} 2 & 8 \\ 16 & 2 \end{bmatrix} \begin{pmatrix} x_1 - 1 \\ y_1 - 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

This can also be written as a system for the update step

$$\begin{bmatrix} 2 & 8 \\ 16 & 2 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = - \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

and thus

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{bmatrix} 2 & 8 \\ 16 & 2 \end{bmatrix}^{-1} \begin{pmatrix} 4 \\ 4 \end{pmatrix} \approx \begin{pmatrix} 0.8064516 \\ 0.5483870 \end{pmatrix}.$$

This is the result of the first Newton step. A visualization of this step can be generated with the code in `Newton2D.m`.

For the next step we use  $f_1(x_1, y_1) \approx 0.853$  and  $f_2(x_1, y_1) \approx 0.993$  and find the system for  $x_2$  and  $y_2$ .

$$\begin{pmatrix} 0.853 \\ 0.993 \end{pmatrix} + \begin{bmatrix} 1.6129 & 4.3871 \\ 8.3918 & 1.0968 \end{bmatrix} \begin{pmatrix} x_2 - 0.806 \\ y_2 - 0.548 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

This and similar calculations lead to

$x_0 = 1$	$y_0 = 1$
$x_1 = 0.8064516$	$y_1 = 0.5483870$
$x_2 = 0.7088993$	$y_2 = 0.3897547$
$x_3 = 0.6837299$	$y_3 = 0.3658653$
$x_4 = 0.6821996$	$y_4 = 0.3655839$
$\vdots$	$\vdots$
$x_7 = 0.6821941$	$y_7 = 0.3655855$

Observe the rapid convergence to a solution.

The above algorithm can be implemented in *Octave*. Below find an code segment to be stored in a file `NewtonSolve.m`. The function `NewtonSolve()` takes the function  $f$ , the function  $Df$  for the partial derivatives and the initial value  $\vec{x}_0$  as arguments and computes the solution of the system  $f(\vec{x}) = \vec{0}$ . The default accuracy of  $10^{-10}$  can be modified with a fourth argument. The code applies at most 20 iterations. The code will return the approximate solution and the number of iterations required.

#### NewtonSolve.m

```
function [x,counter] = NewtonSolve(f,Df,x0,atol)
if nargin<4 atol = 1e-10; end%if;
maxit = 20; counter = 0; xOld = x0;
x = xOld - feval(Df,xOld)\feval(f,xOld);
while ((counter<=maxit) && (norm(xOld-x)>atol))
    xOld = x;
    x = xOld-feval(Df,xOld)\feval(f,xOld);
    counter = counter+1;
end%while
end%function
```

The above problem can now be solved by

```
%% code to solve a simple system of equations
F = @(x) [x(1)^2+4*x(2)^2-1; 4*x(1)^4 + x(2)^2-1];
DF = @(x) [2*x(1), 8*x(2);
           16*x(1)^3, 2*x(2)];
x0 = [1;1]; % choose the starting value
[sol,iter] = NewtonSolve(F,DF,x0) % apply Newton's method
-->
sol =
0.68219
0.36559
iter =
5
```



### The standard result for $n$ equations for $n$ unknowns

The situation of  $n$  equation with  $n$  unknowns can be described with a vector function  $\vec{F}$  with domain of definition  $\mathbb{R}^n$ , or a subset thereof. Solving the system of  $n$  equations is then translated to the search of a

vector  $\vec{x} \in \mathbb{R}^n$  such that

$$\vec{F}(\vec{x}) = \begin{pmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \\ f_3(\vec{x}) \\ \vdots \\ f_n(\vec{x}) \end{pmatrix} = \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ f_3(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \vec{0}.$$

The linear approximation is represented with the help of the matrix  $\mathbf{DF}$  of partial derivatives.

$$\mathbf{DF} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_n} \\ \vdots & \ddots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_n}{\partial x_3} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

The Taylor approximation can now be written in the form

$$\vec{F}(\vec{x} + \vec{\Delta x}) \approx \vec{F}(\vec{x}) + \mathbf{DF}(\vec{x}) \cdot \vec{\Delta x}.$$

Newton's method is again based on the idea of replacing the nonlinear system by its linear approximation and then using a good initial guess  $\vec{x}_0 \in \mathbb{R}^n$ .

$$\vec{F}(\vec{x}_0 + \vec{\Delta x}) = \vec{0} \quad \longrightarrow \quad \vec{F}(\vec{x}_0) + \mathbf{DF}(\vec{x}_0) \cdot \vec{\Delta x} = \vec{0} \quad \longrightarrow \quad \vec{x}_1 = \vec{x}_0 + \vec{\Delta x}$$

It is important to understand this basic idea when applying the algorithm to a concrete problem. It will enable the user to give the algorithm a helping hand when necessary. Quite often Newton is not used as black box algorithm, but tuned to the concrete problem.

**3-9 Theorem :** Let  $\vec{F} \in C^2(\mathbb{R}^n, \mathbb{R}^n)$  be twice continuously differentiable and for a  $\vec{x}^* \in \mathbb{R}^n$  we have  $\vec{F}(\vec{x}^*) = \vec{0}$  and the  $n \times n$  matrix  $\mathbf{DF}(\vec{x}^*)$  of partial derivatives is invertible. Then the Newton iteration

$$\vec{x}_{n+1} = \vec{x}_n - (\mathbf{DF}(\vec{x}_n))^{-1} \cdot \vec{F}(\vec{x}_n)$$

will converge quadratically to the solution  $\vec{x}^*$ , if only the initial guess  $\vec{x}_0$  is close enough to  $\vec{x}^*$ . ◇

The critical point is again the condition that the initial guess  $\vec{x}_0$  has to be close enough to the solution for the algorithm to converge. Thus the remark on Newtons methods applied to a single equation (Section 3.1.2) remain valid.

The above result is not restricted to the space  $\mathbb{R}^n$ . Using standard analysis on Banach spaces the corresponding result remains valid.

### 3.1.6 Modifications of Newton's Method

There are many modification of the basic idea of Newton's method.

### Numerical evaluation of partial derivatives

If no analytical formula for the partial derivatives  $\frac{\partial f_i}{\partial x_j}$  is available, then one can consider a finite difference approximation to these derivatives. Since there are  $n^2$  partial derivatives this requires at least  $n^2 + 1$  evaluations of the functions  $f_i$ . This might be a delicate problem, and computationally expensive.

### The modified Newton algorithm

The computational effort to determine the  $n \times n$  matrix  $\mathbf{DF}(\vec{x}_n)$  can be considerable. Thus one can reuse the same matrix for a fixed number  $m$  of steps and only then reevaluate the matrix of partial derivatives.

$$\vec{x}_{n+j+1} = \vec{x}_{n+j} - (\mathbf{DF}(\vec{x}_n))^{-1} \cdot \vec{F}(\vec{x}_{n+j}) \quad \text{for } j = 0, 1, 2, \dots, m.$$

More iterations than with the standard method may be needed, but the computational effort for one step is smaller.

### Damped Newton's method

If the initial guess  $\vec{x}_0$  is not close enough to the actual solution, then Newton's method might jump to a completely different region and continue its search there, see e.g. the computations leading to Figure 4.28 on page 301. To avoid this effect one can at first shorten the step in Newtons method. For a parameter  $0 < \alpha \leq 1$  the iteration formula is modified to

$$\vec{x}_{n+1} = \vec{x}_n - \alpha (\mathbf{DF}(\vec{x}_n))^{-1} \cdot \vec{F}(\vec{x}_n)$$

For  $\alpha = 1$  we have the classical formula. For  $\alpha < 1$  we have a **damped Newton iteration**. In this case we loose quadratic convergence.

In recent papers [SchnWihl11], [AmreWihl14] Thomas Wihler (University of Bern) proposed a systematic approach to choosing the step-sizes, based on an ODE related to the system of equations to be solved.

The damped Newton algorithm is used by the **Levenberg–Marquardt** algorithm to solve nonlinear regression problems. At first the parameter  $\alpha$  is strictly smaller than 1. As progress is made  $\alpha$  approaches 1 to achieve quadratic convergence. This approach is used for nonlinear regression problems in Section 3.5.7 by the command `leasqr()`.

### Parameterized Newton's method

One tool available to circumvent the problem of good initial guesses is to use a parameterized Newton's method. It is often known which part of the equations causes problems for convergence.

- Start your computation with the troublesome term turned off and find a solution of the modified problem.
- Then turn the nonlinear term on step by step and use the previous solution as a initial point for Newton's method.

In Example 4–12 we will try to solve the boundary value problem

$$-\alpha''(s) = \frac{F_2}{EI} \cos(\alpha(s)) \quad \text{for } 0 < s < L \quad \text{and} \quad \alpha(0) = \alpha'(L) = 0.$$

This is the mathematical model for the bending beam, shown as shown in Section 1.4 (page 18). For large values of  $F_2$  this will not give the desired solution. For  $F_2 = 0$  the solution  $\alpha(s) = 0$  is obvious. Thus we start with  $F_2 = 0$  and then increase  $F_2$  in small steps, solving the BVP. If we arrive at the desired value of  $F_2$  we then will have a solution of the original problem. This is more efficient and (even more important) more reliable than the basic algorithm of Newton.

### 3.1.7 Octave/MATLAB Commands to Solve Equations

#### The command `fzero()` to solve a single equation

With the command `fzero()` a single equation  $f(x) = 0$  can be solved. It is advisable to give a bracketing as initial values. To solve  $x^2 - 2 = 0$  use

```
fzero(@(x)x^2-2, [0,2])
-->
ans = 1.4142
```

A sizable number of options can be used and more outputs are generated, see `help fzero`.

#### The command `fsolve()` to solve one equation or a system

With the command `fsolve()` a single equation  $f(x) = 0$  or systems  $\vec{f}(\vec{x}) = \vec{0}$  can be solved. The algorithm is based on Newton's method and thus it is essential to provide a good initial guess.

To solve  $x^2 - 2 = 0$  use

```
fzero(@(x)x^2-2, 2)
-->
ans = 1.4142
```

To solve the system in Example 3–8 use

```
f = @(x) [x(1)^2 + 4*x(2)^2-1; 4*x(1)^4 + x(2)^2-1];
fsolve(f, [1;1])
-->
ans =
    0.6822
    0.3656
```

Since Newton's methods requires the Jacobian matrix `fsolve()` uses a finite difference approach to determine the matrix of partial derivatives. One can also specify the Jacobian by creating a function<sup>4</sup> with two return arguments. In addition the option `Jacobian` has to be on.

```
function [res, Jac] = Fun_System(x)
    res = [x(1)^2 + 4*x(2)^2-1; 4*x(1)^4 + x(2)^2-1];
    if nargout ==2
        Jac = [2*x(1), 8*x(2); 4*x(1)^3, 2*x(2)];
    end%if
end%function

fsolve('Fun_System', [1;1], optimset('Jacobian','on'))
-->
ans =
    0.6822
    0.3656
```

A sizable number of options can be used and more outputs are generated, see `help fsolve`.

<sup>4</sup>With MATLAB this function has to be in a separate file `Fun_System.m`.

### 3-10 Example : Using Newton's algorithm in MATLAB/Octave

In Example 3-14 the solution of the equation  $x^2 - 1 - \cos x = 0$  is needed. In an Octave script first define a function to evaluate the function  $f(x)$  and its derivative. Then create a graph and estimate the location of the zero as  $x_0 = 1$ . Find the result in Figure 3.9. Then a simple call of `fsolve()` will compute the location of the zero as  $x \approx 1.1765019$ . By tracing the calls to the function  $f(x)$  one can observe how `fsolve()` uses a finite difference approximation to determine the values of the derivative  $f'(x)$  by computing  $f(x)$  and  $f(x + \Delta x)$  and then use  $f'(x) \approx (f(x + \Delta x) - f(x)) / \Delta x$ .

```

x = 0:0.1:3;

function [y,dy] = f(x)
    y = x.*x-1-cos(x); % value of the function
    dy = 2*x+sin(x);     % value of the derivative
    display([x,y])        % show the values of x and y
endfunction

figure(1); plot(x,f(x))
xlabel('x'); ylabel('f(x) = x^2-1-cos(x)'); grid on

[z,info,msg] = fsolve('f',1.0) % without using the derivative
options.Jacobian = 'on';          % use the given derivative
z = fsolve('f',1.0,options)
-->
1.0000  -0.5403
1.0000  -0.5403
1.190149  0.044932
1.190149  0.044932
1.1766e+00  2.1906e-04
1.1766e+00  2.1906e-04
1.1765e+00  5.3287e-09
z = 1.1765

```

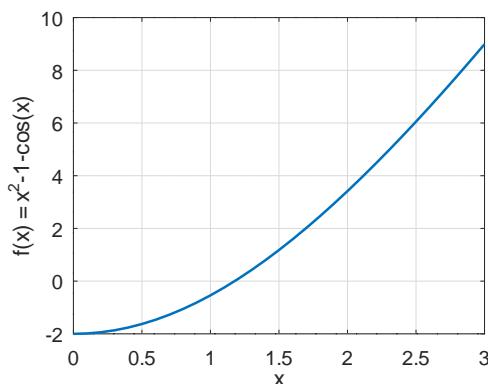


Figure 3.9: Graph of the function  $y = x^2 - 1 - \cos(x)$



#### 3.1.8 Examples of Nonlinear Equations

### 3-11 Example : Nonlinear finite difference equations

In Section 4.7, starting on page 296, some examples of nonlinear equations will be examined:

- Example 4–9 examines the stretching of a beam by a given force and variable cross section. The method of successive substitutions is used.
- Example 4–11 examines the bending of a beam. Large deformations are allowed. Newton’s method will be used.
- Example 4–12 examines a similar problem, using a parameterized version of Newton’s method.

◊

### 3–12 Example : Nonlinear methods applied to a tumor growth problem

In Section 6.9, starting on page 458, the idea of linearization or Newton’s method will be used to examine a tumor growth problem. For the space discretization the finite element method (FEM) will be used and for the time stepping a nonlinear Crank–Nicolson algorithm. ◊

### 3–13 Example : Stretching of a beam by a given force and variable cross section

The differential equation describing the longitudinal deformation of a string with cross section  $A_0(x)$  by a force density  $f(x) = 0$  and a force  $F$  at the right end at  $x = L$  is given by (see Section 1.3)

$$-\frac{d}{dx} \left( EA_0 \left( 1 - \nu \frac{du}{dx} \right)^2 \frac{du(x)}{dx} \right) = 0 \quad \text{for } 0 < x < L.$$

and the boundary conditions  $u(0) = 0$  and  $EA_0(L) (1 - \nu u'(L))^2 u'(L) = F$ . If a function  $w(x) = u'(x)$  for  $0 \leq x \leq L$  solves equation (1.15) (page 17)

$$\nu^2 w^3(x) - 2\nu w^2(x) + w(x) = \frac{F}{EA_0(x)},$$

then its integral

$$u(x) = \int_0^x w(s) ds$$

represents the horizontal deflection of a horizontal beam. A horizontal force  $F$  is applied at the right endpoint. If  $F = 0$ , then the obvious and physically correct solution is  $w(x) = 0$ . For a given function  $A_0(x)$  search for a solution of the above nonlinear equation. The solution plan to be carried out below is as follows:

- Introduce an auxiliary function  $G$  to be examined.
- Determine for which domain the equation might be solved, requiring the solution to be realistic.
- Start with a force of  $F = 0$  and increase it step by step. For each force compute the new length of the beam.
- Plot the force  $F$  as a function of the change in length  $u(L)$  to confirm Hooke’s law.

Set  $z = \nu u' = \nu w$  and consider the function

$$G(z) = z^3 - 2z^2 + z - \frac{\nu F}{EA_0(x)}.$$

The variable to be solved for is  $z$ , and  $x$  is considered as a parameter. With the help of solution of the equation  $G(z) = 0$  construct solutions of the beam problem. In general the solution  $z$  will depend on  $x$ . Before launching the computations examine possible failures of the method. Newton’s iteration will fail if the derivative vanishes. The derivative of  $G(z)$  is given by

$$\frac{d}{dz} G(z) = 3z^2 - 4z + 1 = (3z - 1)(z - 1).$$

Since  $G'(0) = 1$  we know this derivative to be positive for  $0 < z$  small. The zeros of the derivative  $G'$  are readily determined as

$$z = \frac{+4 \pm \sqrt{16 - 12}}{6} = \begin{cases} 1 \\ \frac{1}{3} \end{cases}.$$

Since the first zero of the derivative  $G'$  is at  $z = \frac{1}{3}$  expect problems if  $z \approx \frac{1}{3}$ . To confirm this examine the graph of the auxiliary function  $h$  shown in Figure 3.10.

$$\begin{aligned} h(z) &= z^3 - 2z^2 + z \\ G(z) &= h(z) - \frac{\nu F}{EA_0} \end{aligned}$$

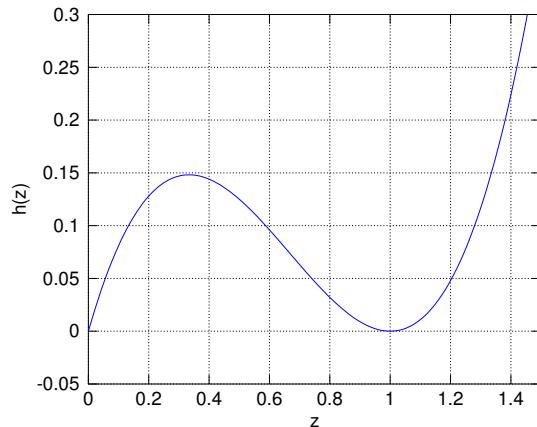


Figure 3.10: Definition and graph of the auxiliary function  $h$

We will start with the force  $F = 0$  and thus  $w(x) = 0$ . Then increase the value of  $F$  slowly and  $w$  (resp.  $z$ ) will increase too. We use Newton's method to determine the function  $w(x)$ , using the initial value  $w_0(x) = 0$  to find a solution of the above problem. If the expression

$$\frac{\nu F}{EA_0(x)}$$

is larger than  $h(1/3) = 4/27$  there is no smooth solution any more. If the critical limit for  $F$  is exceeded find that  $z = \nu u'(x)$  would have to be larger than 1. This would lead to a **negative** radius with cross sectional area  $A_0(1 - \nu u'(x))^2$ , which is obviously mechanical nonsense. This is confirmed by Figure 3.10. The beam will simply break if the critical limit is exceeded. The Octave code will happily generate numbers and graphs for larger values of  $F$ : GIGO .

The iteration formula to solve the above equation is given by

$$w_{n+1}(x) = w_n(x) - \frac{G(w_n(x))}{G'(w_n(x))}.$$

As a concrete example choose the value  $\nu = 0.3$  and the function

$$EA_0(x) = \frac{1}{2} \left( 2 - \sin\left(\frac{\pi x}{L}\right) \right) \quad \text{for } 0 \leq x \leq L.$$

This corresponds to a beam with a thiner midsection. These values will be reused in Example 4-9, where the same problem is solved with the help of a finite difference approximation. Since the minimal value of  $EA_0(x)$  is  $1/2$  the above condition on  $F$  translates to

$$F < \frac{4}{27} \frac{EA_0}{\nu} \approx 0.24691.$$

Thus expect problems beyond this critical value. To find a solution proceed as follows:

- Define the necessary constants and functions
- Choose a number  $N$  of grid points on the interval  $(0, L)$
- Choose a starting function  $w(x) = 0$ , resp. vector  $\vec{w} = \vec{0}$
- Choose the forces for which the solution is to be computed. The force should be increased slowly from 0 to the maximal possible value.
- For each value of the force:
  - Run the Newton iteration until the desired accuracy is achieved.
  - Compute the new length of the beam with the help of

$$u(L) = \int_0^L w(x) dx .$$

- Plot the length as a function of the applied force.

The MATLAB/Octave code below and the resulting Figure 3.11 confirm the above observations.

run  
testBeam.m

First define all necessary constants and functions.

```
clear EA
nu = 0.3; L = 3;

function res = EA(x)
  res = (2-sin(x/L*pi))/2;
end%function

function y = G(z,T)
  nu = 0.3;
  y = nu^2*z.^3-2*nu*z.^2 + z -T;
end%function

function y = dG(z)
  y = 3*nu^2*z.^2-4*nu*z + 1;
end%function
```

Then run the Newton iteration.

```
N = 500;
h = L/(N+1); % stepsize
x = (0:h:L)';
clf; relErrorTol=1e-10; % choose your relative error tolerance
z = zeros(size(x)); zNew = z;

FList = 0.01:0.01:0.24; maxAmp = zeros(size(FList));
k = 0;
for F = FList
  k = k+1;
  T = F./EA(x);
  relError = 2*relErrorTol;
  while relError>relErrorTol,
    zNew = z-G(z,T)./dG(z);
    relError = max(abs(z-zNew))/max(abs(zNew));
    z = zNew;
```

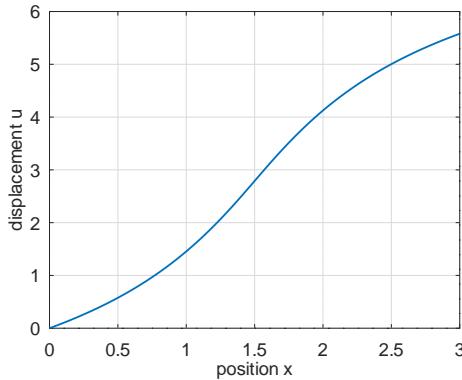
```

end%while
maxAmp(k) = trapz(x, zNew/nu);
end%for

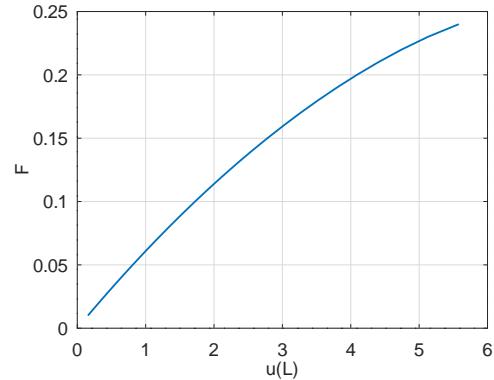
u = cumtrapz(x, zNew/nu);
figure(1); plot(x, u);
    xlabel('position x'); ylabel('displacement u'); grid on

figure(2); plot(maxAmp,FList);
    xlabel('maximal displacement u(L)'); ylabel('force F'); grid on

```



(a) displacement as function of position



(b) force as function of maximal displacement

Figure 3.11: Graphs for stretching of a beam, with Poisson contraction

The graph in Figure 3.11(b) shows the force  $F$  as function of the displacement  $u(L)$  at the right endpoint. If the lateral contraction of the beam would not be taken into account (i.e.  $\nu = 0$ ) the results would be a straight line, confirming Hooke's law. The Poisson effect weakens the beam, since the area of the cross sections is reduced and the stress thus increased, i.e. move from the engineering stress to true stress. ◇

**3–14 Example :** In [Kell92, p. 317] the boundary value problem

Ex. 3.4

$$-u''(x) = -e^{u(x)} \quad \text{with} \quad u(-1) = u(1) = 0$$

is examined. The exact solution is given by

$$u(x) = \ln \left( \frac{c^2}{1 + \cos(cx)} \right),$$

where the value of the constant  $c$  is determined as solution of the equation  $c^2 = 1 + \cos c$ . In Example 3–10 Newton's method is used to find  $c \approx 1.1765019$ . Now use Newton's again method to solve the above nonlinear boundary value problem.

With an approximate solution  $u_n(x)$  (start with  $u_0(x) = 0$ ) search a new solution of the form  $u_{n+1}(x) = u_n(x) + \phi(x)$  and examine a linear boundary value problem for the unknown function  $\phi(x)$ . Use the Taylor approximation  $e^{u+\phi} \approx e^u + e^u\phi = e^u(1 + \phi)$  and solve

$$\begin{aligned} -u_n''(x) - \phi''(x) &= -e^{u_n(x)+\phi(x)} \approx -e^{u_n(x)}(1 + \phi(x)) \\ -\phi''(x) + e^{u_n(x)}\phi(x) &= u_n''(x) - e^{u_n(x)} \quad \text{with} \quad \phi(-1) = \phi(1) = 0. \end{aligned}$$

This boundary value problem for the function  $\phi(x)$  can be solved with a finite difference approximation (see Chapter 4). Let  $h = \frac{2}{N+1}$  and  $x_i = -1 + i h$  for  $i = 1, 2, 3, \dots, N$ . With  $u_i = u(x_i)$  and  $\phi_i = \phi(x_i)$  obtain a finite difference approximation of the second order derivative

$$-\phi''(x_i) \approx \frac{-\phi_{i-1} + 2\phi_i - \phi_{i+1}}{h^2}$$

and thus a system of linear equations for the unknowns  $\phi_i$ . Use  $\phi_0 = \phi_{N+1} = 0$ .

$$\frac{-\phi_{i-1} + 2\phi_i - \phi_{i+1}}{h^2} + e^{u_i} \phi_i = -\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} - e^{u_i} = b_i \quad \text{for } i = 1, 2, 3, \dots, N$$

Using a matrix notation this leads to

$$\left[ \begin{array}{cccccc} \frac{2}{h^2} + e^{u_1} & -\frac{1}{h^2} & & & & \\ -\frac{1}{h^2} & \frac{2}{h^2} + e^{u_2} & -\frac{1}{h^2} & & & \\ & -\frac{1}{h^2} & \frac{2}{h^2} + e^{u_3} & -\frac{1}{h^2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\frac{1}{h^2} & \frac{2}{h^2} + e^{u_{N-1}} & -\frac{1}{h^2} \\ & & & & -\frac{1}{h^2} & \frac{2}{h^2} + e^{u_N} \end{array} \right] \cdot \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_{N-1} \\ \phi_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{N-1} \\ b_N \end{pmatrix}.$$

Solve this system of linear equations and then restart with  $u_{n+1}(x) = u_n(x) + \phi(x)$ . The matrix  $\mathbf{A}$  in  $\mathbf{A}\vec{\phi} = \vec{b}$  has a **tridiagonal** structure. For this type of problem special algorithms exist<sup>5</sup>. The above algorithm is implemented in Octave/MATLAB.

run  
Keller.m

```
N = 201; h = 2/(N+1); % number of grid points and stepsize

x = (-1:h:1)';
c = 1.176501940; ueexact = log(c.^2./(1+cos(c.*x)));

%%%%% Newton %%%%%%
%% build the tridiagonal matrix
di = 2*ones(N,1)/h.^2; % main diagonal
up = -ones(N-1,1)/h.^2; % upper and lower diagonal

Niterations = 5; errorNewton = zeros(Niterations,1);
u = zeros(N,1);
for k = 1:Niterations
    g = diff(diff([0;u;0]))/h.^2-exp(u);
    u = u + trisolve(di+exp(u),up,g);
    errorNewton(k) = max(abs(ueexact-[0;u;0]));
end%for
errorNewton
```

The result, shown below, illustrates that the algorithm stabilizes after the fourth step. The error does not decrease any more. The remaining error is dominated by the number of grid points  $N = 201$ . When setting  $N = 20001$  the error decreases to  $2.3 \cdot 10^{-10}$ . This effect can only be illustrated using the known exact solution. In real world problems this is not the case and we would stop the iteration as soon as enough digits do not change any more.

<sup>5</sup>An implementation is given in Octave as command `trisolve`. For MATLAB a similar code is provided in Table 3.19 in the file `tridiag.m`. With newer versions one can use sparse matrices to solve tridiagonal systems efficiently.

```

errorNewton =
1.611231387e-02
2.683166420e-05
1.913113033e-06
1.913054659e-06
1.913054659e-06

```

The above code is listed in Table 3.19 as file `Keller.m`. In this file the method of successive substitutions is also applied to the problem. A graph with the errors for Newton's method and successive substitutions is generated. With this code you can verify that both methods converge, but Newton's converge rate is two, while successive substitution converges linearly. In Table 3.3 find a comparison of Newton's method and the partial substitution approach applied to problems similar to the above. ◇

	Substitution	Newton
convergence	linear, slow	quadratic, fast
complexity of code	very simple	intermediate
good starting values necessary	yes	yes
derivative of $f(u)$ required	no	yes
solve a <b>new</b> linear system for each step	no	yes

Table 3.3: Compare partial substitution method and Newton's method

**3-15 Example :** In the previous example the BVP (Boundary Value Problem)

$$-u''(x) = -e^{u(x)} \quad \text{with} \quad u(-1) = u(1) = 0$$

was solved by the following steps:

1. Linearize the BVP.
2. Transform the linearized BVP into a system of linear equations, using finite differences.
3. Solve the resulting system of linear equations.

One may also try to apply the operations in a different order:

1. Transform the nonlinear BVP into a system of nonlinear equations, using finite differences.
2. Linearize this system of nonlinear equations.
3. Solve the resulting system of linear equations.

With finite differences the system of nonlinear equations to be solved is

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = -e^{u_i} \quad \text{for } i = 1, 2, \dots, n.$$

If  $u(x) + \phi(x)$  is a small perturbation of  $u(x)$  we use the linear approximation  $e^{u+\phi} \approx e^u (1 + \phi)$  and the fact that the linear difference operation on the left is linear. We find

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + \frac{-\phi_{i-1} + 2\phi_i - \phi_{i+1}}{h^2} = -e^{u_i}(1 + \phi_i)$$

or

$$\frac{-\phi_{i-1} + 2\phi_i - \phi_{i+1}}{h^2} + e^{u_i} \phi_i = -\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} - e^{u_i}.$$

This system of linear equations is identical to the previous problem and consequently one will find identical results. ◇

**3–16 Example :** In the previous example only the right hand side of the BVP contained a nonlinear function. The method applies also to problems with nonlinear coefficient functions. Consider

$$-(a(u(x)) u'(x))' = f(u(x)) \quad \text{with} \quad u(0) = u(1) = 0$$

and use Newton's method, i.e. for a known starting function  $u$  search for  $u + \phi$  and determine  $\phi$  as a solution of a linear problem. Then restart with the new function  $u_1 = u + \phi$ . Use the linear approximations

$$\begin{aligned} a(u + \phi) &\approx a(u) + a'(u) \cdot \phi \\ f(u + \phi) &\approx f(u) + f'(u) \cdot \phi \\ a(u + \phi)(u + \phi)' &\approx (a(u) + a'(u) \cdot \phi)(u + \phi)' \\ &\approx a(u) u' + a'(u) u' \phi + a(u) \phi' \end{aligned}$$

to replace the original nonlinear differential equation for  $u$  with a linear equation for the unknown function  $\phi$ .

$$\begin{aligned} -(a'(u) u' \phi + a(u) \phi')' &= (a(u) u')' + f(u) + f'(u) \phi \\ -(a(u) \phi)'' - f'(u) \phi &= (a(u) u')' + f(u) \end{aligned}$$

The finite difference approximation of the expression  $(a(u) u')'$  is given in Example 4–8 on page 267. There are two possible options to solve the above linear BVP for the unknown function  $\phi$ .

- Option 1: Finite difference approximation of the expression

$$(b(x) u(x))'' \approx \frac{b(x-h) u(x-h) - 2 b(x) u(x) + b(x+h) u(x+h)}{h^2}$$

or with a matrix notation

$$\frac{1}{h^2} \begin{bmatrix} -2b_1 & b_2 & & & & & & \\ b_1 & -2b_2 & b_3 & & & & & \\ & b_2 & -2b_3 & b_4 & & & & \\ & & b_3 & -2b_4 & b_5 & & & \\ & & & & \ddots & & & \\ & & & & & b_{N-2} & -2b_{N-1} & b_N \\ & & & & & b_{N-1} & -2b_N & \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_{N-1} \\ u_N \end{pmatrix}$$

- Option 2: If the coefficient function  $a(u)$  is strictly positive introduce a new function  $w(x) = a(x) \phi(x)$ . Since  $\phi = \frac{1}{a} w$  find the new differential equation

$$-w'' - \frac{f'(u)}{a(u)} w = (a(u) u')' + f(u)$$

for the unknown function  $w(x)$ . Once  $w(x)$  is computed use  $\phi(x) = \frac{w(x)}{a(u(x))}$ .

Then restart with the new approximation  $u_n(x) = u(x) + \phi(x)$ . ◊

### 3.1.9 Optimization with MATLAB/Octave

The command `fminbnd()` to find minima of a function with one variable

The function  $f(x) = \sin(x)$  has a local minimum at  $x = \frac{3\pi}{2}$  and the command `fminbnd()` will find this location between 2 and 8 by

```
fminbnd(@(x) sin(x), 2, 8)/pi
-->
ans = 1.50000
```

The function `fminbnd()` will only search between  $x = 2$  and  $x = 8$ . A sizable number of options can be used and more outputs are generated, see `help fminbnd`.

### The command `fminsearch()` to find minima of a function with multiple variables

**3-17 Example :** One can also optimize functions of multiple variables. Instead of a maximum of

$$f(x, y) = -2x^2 - 3xy - 2y^2 + 5x + 2y$$

search for a minimum of  $-f(x, y)$ . Examine the graph of  $f(x, y)$  in Figure 3.12.

#### Octave

```
[xx,yy] = meshgrid([-1:0.1:4], [-2:0.1:2]);
function res = f(x,y)
    res = -2*x.^2 -3*x.*y-2*y.^2 + 5*x+2*y;
endfunction
surf(x,y,f(x,y)); xlabel('x'); ylabel('y');
```

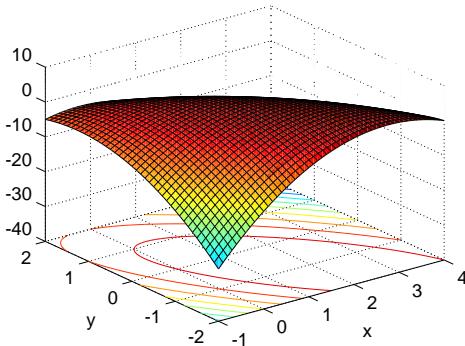


Figure 3.12: Graph of a function  $h = f(x, y)$ , with contour lines

Using the graph conclude that there is a maximum not too far away from  $(x, y) \approx (1.5, 0)$ . Now use `fminsearch()` with the function  $-f$  and the above starting values.

#### Octave

```
xMin = fminsearch(@(x)-f(x(1),x(2)), [1.5,0])
-->
xMin = 2.0000 -1.0000
```

A sizable number of options can be used and more outputs are generated, see `help fminsearch`. ◇

To examine the accuracy of the extreme point consider to have a closer look at contour levels.

## 3.2 Eigenvalues and Eigenvectors of Matrices, SVD, PCA

### 3.2.1 Matrices and Linear Mappings

Matrices of size  $m \times n$  can be used to describe linear mappings from  $\mathbb{R}^n$  into  $\mathbb{R}^m$ . This is best illustrated by an example.

Examine the matrix  $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} 1 & 0.5 \\ 0.25 & 0.75 \end{bmatrix}$$

representing a linear mapping from  $\mathbb{R}^2$  to  $\mathbb{R}^2$ , i.e. for an arbitrary vector  $\vec{x} \in \mathbb{R}^2$  the image is given by

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \mathbf{A} \vec{x} = \begin{bmatrix} 1 & 0.5 \\ 0.25 & 0.75 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1x_1 + 0.5x_2 \\ 0.25x_1 + 0.75x_2 \end{pmatrix}.$$

For the standard basis vectors observe

$$\mathbf{A} \vec{e}_1 = \begin{bmatrix} 1 & 0.5 \\ 0.25 & 0.75 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.25 \end{pmatrix} \quad \text{and} \quad \mathbf{A} \vec{e}_2 = \begin{bmatrix} 1 & 0.5 \\ 0.25 & 0.75 \end{bmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.75 \end{pmatrix}.$$

The columns of  $\mathbf{A}$  contain the images of the standard basis vectors. Figure 3.13 visualizes this linear mapping.

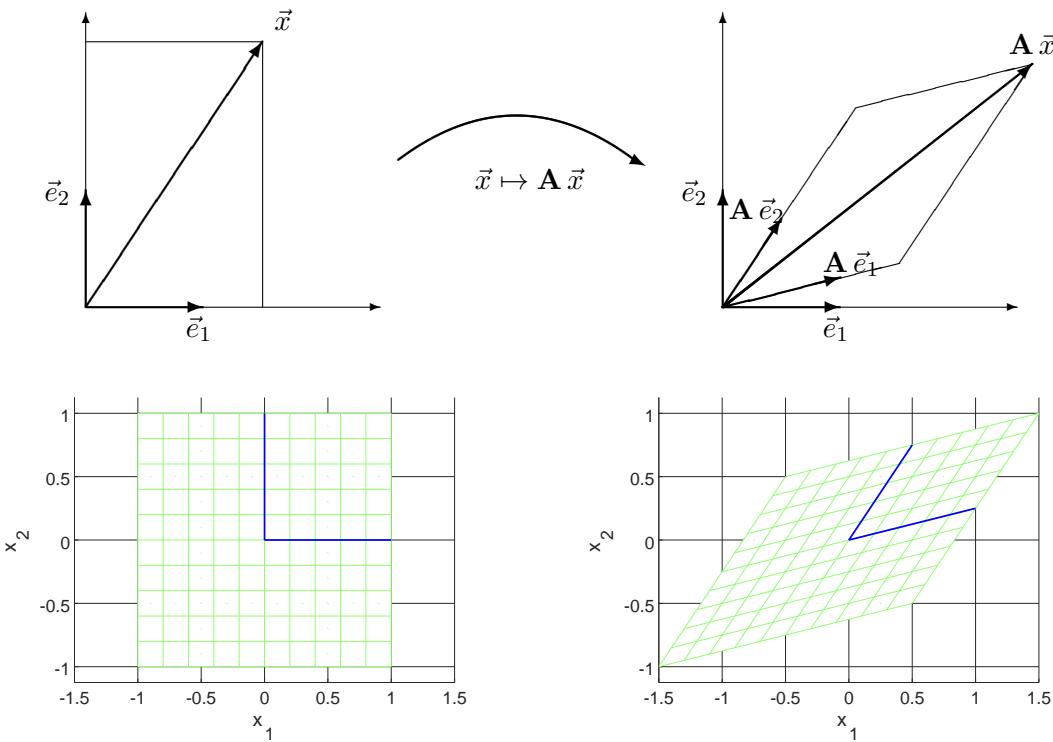


Figure 3.13: A linear mapping applied to a rectangle

### 3-18 Example : Orthogonal matrices, unitary matrices

A real matrix  $\mathbf{V} \in \mathbb{M}^{n \times n}$  is called an **orthogonal matrix**<sup>6</sup> iff

$$\mathbf{U}^T \mathbf{U} = \mathbb{I}_n.$$

<sup>6</sup>This author actually would prefer the notation of an orthonormal matrix.

For matrices with complex entries this is called an **unitary matrix**. The column vectors have length 1 and are pairwise orthogonal. To examine this property consider the columns of  $\mathbf{U}$  as vectors  $\vec{u}_i \in \mathbb{R}^n$ , i.e.  $\mathbf{U} = [\vec{u}_1, \vec{u}_2, \vec{u}_3, \dots, \vec{u}_n] \in \mathbb{M}^{n \times n}$ . Then examine the scalar products of these column vectors, as components of a matrix product.

$$\begin{aligned}\mathbf{U}^T \mathbf{U} &= \begin{bmatrix} \vec{u}_1^T \\ \vec{u}_2^T \\ \vec{u}_3^T \\ \vdots \\ \vec{u}_n^T \end{bmatrix} [\vec{u}_1, \vec{u}_2, \vec{u}_3, \dots, \vec{u}_n] \\ &= \begin{bmatrix} \langle \vec{u}_1, \vec{u}_1 \rangle & \langle \vec{u}_1, \vec{u}_2 \rangle & \langle \vec{u}_1, \vec{u}_3 \rangle & \cdots & \langle \vec{u}_1, \vec{u}_n \rangle \\ \langle \vec{u}_2, \vec{u}_1 \rangle & \langle \vec{u}_2, \vec{u}_2 \rangle & \langle \vec{u}_2, \vec{u}_3 \rangle & \cdots & \langle \vec{u}_2, \vec{u}_n \rangle \\ \langle \vec{u}_3, \vec{u}_1 \rangle & \langle \vec{u}_3, \vec{u}_2 \rangle & \langle \vec{u}_3, \vec{u}_3 \rangle & \cdots & \langle \vec{u}_3, \vec{u}_n \rangle \\ \vdots & & \ddots & & \vdots \\ \langle \vec{u}_n, \vec{u}_1 \rangle & \langle \vec{u}_n, \vec{u}_2 \rangle & \langle \vec{u}_n, \vec{u}_3 \rangle & \cdots & \langle \vec{u}_n, \vec{u}_n \rangle \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} = \mathbb{I}_n\end{aligned}$$

For an orthogonal matrix use  $\mathbf{U}^{-1} = \mathbf{U}^T$ , i.e. the inverse matrix is easily given.

Multiplying a vector  $\vec{x}$  by an orthogonal matrix  $\mathbf{U}$  does not change the length of the vector and the angles between vectors. To verify this fact use the scalar product again.

$$\begin{aligned}\|\mathbf{U}\vec{x}\|^2 &= \langle \mathbf{U}\vec{x}, \mathbf{U}\vec{x} \rangle = \langle \vec{x}, \mathbf{U}^T \mathbf{U}\vec{x} \rangle = \langle \vec{x}, \mathbb{I}\vec{x} \rangle = \|\vec{x}\|^2 \\ \cos(\angle(\mathbf{U}\vec{x}, \mathbf{U}\vec{y})) &= \frac{\langle \mathbf{U}\vec{x}, \mathbf{U}\vec{y} \rangle}{\|\vec{x}\| \|\vec{y}\|} = \frac{\langle \vec{x}, \mathbf{U}^T \mathbf{U}\vec{y} \rangle}{\|\vec{x}\| \|\vec{y}\|} = \frac{\langle \vec{x}, \vec{y} \rangle}{\|\vec{x}\| \|\vec{y}\|} = \cos(\angle(\vec{x}, \vec{y}))\end{aligned}$$

Thus multiplying by an orthogonal matrix  $\mathbf{U}$  corresponds to a rotation in  $\mathbb{R}^n$  and possibly one reflection. If  $\det(\mathbf{U}) = -1$ , there is a reflection involved, with  $\det(\mathbf{U}) = +1$  it is rotations only.

In the plane  $\mathbb{R}^2$  the orthogonal matrices with angle of rotation  $\alpha$  are the well known rotation matrices.

$$\mathbf{U} = \begin{bmatrix} +\cos(\alpha) & -\sin(\alpha) \\ +\sin(\alpha) & +\cos(\alpha) \end{bmatrix} \quad \text{or} \quad \mathbf{U} = \begin{bmatrix} +\cos(\alpha) & +\sin(\alpha) \\ +\sin(\alpha) & -\cos(\alpha) \end{bmatrix}.$$

◇

### 3.2.2 Eigenvalues and Diagonalization of Matrices

In these notes only real valued matrices are examined, i.e.  $\mathbf{A} \in \mathbb{M}^{m \times n} = \mathbb{R}^{m \times n}$ .

#### Definition of eigenvalues and eigenvectors

##### 3-19 Definition :

- A number  $\lambda \in \mathbb{C}$  is called an **eigenvalue** with corresponding **eigenvector**  $\vec{u} \neq \vec{0}$  of the real matrix  $\mathbf{A} \in \mathbb{M}^{n \times n}$  iff

$$\mathbf{A} \vec{u} = \lambda \vec{u}. \quad (3.2)$$

- A number  $\lambda \in \mathbb{C}$  is called an **generalized eigenvalue** with corresponding **eigenvector**  $\vec{u} \neq \vec{0}$  of the real matrix  $\mathbf{A} \in \mathbb{M}^{n \times n}$  and the weight matrix  $\mathbf{B} \in \mathbb{M}^{n \times n}$  iff

$$\mathbf{A} \vec{u} = \lambda \mathbf{B} \vec{u}. \quad (3.3)$$

**3-20 Observation :**

- If  $\lambda$  is an eigenvalue then  $(\mathbf{A} - \lambda \mathbb{I})\vec{u} = \vec{0}$  with  $\vec{u} \neq \vec{0}$ . Thus the matrix  $\mathbf{A} - \lambda \mathbb{I} \in \mathbb{M}^{n \times n}$  is not invertible and  $\lambda$  is a zero of the **characteristic polynomial**

$$p(\lambda) = \det(\mathbf{A} - \lambda \mathbb{I}) = 0.$$

To determine the characteristic polynomial subtract  $\lambda$  along the diagonal of the matrix  $\mathbf{A}$  and compute the determinant. This is a polynomial of degree  $n$  and consequently any  $n \times n$  matrix has exactly  $n$  eigenvalues  $\lambda_i$  for  $i = 1, 2, 3, \dots, n$ . These eigenvalues can be real or complex.

- Not all eigenvalues have their "own" eigenvector. The matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

has a double eigenvalue  $\lambda_1 = \lambda_2 = 0$  but only one eigenvector  $\vec{u} = (1, 0)^T$ . In this case  $\lambda = 0$  has algebraic multiplicity 2, but geometric multiplicity 1. This matrix can not be diagonalized. The mathematical tool to be used in this special case are **Jordan normal forms**, see e.g. [HornJohn90, §3] or [https://en.wikipedia.org/wiki/Jordan\\_normal\\_form](https://en.wikipedia.org/wiki/Jordan_normal_form). In these notes only diagonalizable matrices are examined and used.

- If  $\lambda = \alpha + i\beta \in \mathbb{C}$  with  $\beta \neq 0$  is a complex eigenvalue with eigenvector  $\vec{u} + i\vec{v}$ , then  $\lambda - i\beta$  is an eigenvalue too, with eigenvector  $\vec{u} - i\vec{v}$ . To verify this examine

$$\begin{aligned} \mathbf{A}(\vec{u} + i\vec{v}) &= (\alpha + i\beta)(\vec{u} + i\vec{v}) = (\alpha\vec{u} - \beta\vec{v}) + i(\beta\vec{u} + \alpha\vec{v}) \\ \mathbf{A}\vec{u} &= \alpha\vec{u} - \beta\vec{v} \quad \text{real part} \\ \mathbf{A}\vec{v} &= \beta\vec{u} + \alpha\vec{v} \quad \text{imaginary part} \\ \mathbf{A}(\vec{u} - i\vec{v}) &= (\alpha - i\beta)(\vec{u} - i\vec{v}) = (\alpha\vec{u} - \beta\vec{v}) - i(\beta\vec{u} + \alpha\vec{v}) \end{aligned}$$

or with a matrix notation

$$\mathbf{A} \begin{bmatrix} \vec{u} & \vec{v} \end{bmatrix} = \begin{bmatrix} +\alpha & -\beta \\ +\beta & +\alpha \end{bmatrix} \begin{bmatrix} \vec{u} & \vec{v} \end{bmatrix}.$$

The two vectors  $\vec{u} \in \mathbb{R}^n \setminus \{\vec{0}\}$  and  $\vec{v} \in \mathbb{R}^n \setminus \{\vec{0}\}$  are not zero, verified by a contradiction argument.

$$\begin{aligned} \vec{v} = \vec{0} &\implies \mathbf{A}\vec{v} = +\beta\vec{u} = \vec{0} \implies \vec{u} = \vec{0} \\ \vec{u} = \vec{0} &\implies \mathbf{A}\vec{u} = -\beta\vec{v} = \vec{0} \implies \vec{v} = \vec{0} \end{aligned}$$

The two vectors  $\vec{u}, \vec{v} \in \mathbb{R}^n$  are (real) linearly independent, again verified by contradiction.

$$\vec{v} = c\vec{u} \implies \mathbf{A}\vec{u} = \alpha\vec{u} - \beta\vec{v} = (\alpha - \beta c)\vec{u}$$

and  $\vec{u}$  would be an eigenvector with real eigenvalue  $(\alpha - \beta c)$ .

- If  $\lambda$  is a generalized eigenvector and the invertible weight matrix  $\mathbf{B}$  has a Cholesky factorization  $\mathbf{B} = \mathbf{R}^T \mathbf{R}$ , then

$$\mathbf{A}\vec{u} = \lambda \mathbf{B}\vec{u} = \lambda \mathbf{R}^T \mathbf{R}\vec{u} \implies \mathbf{R}^{-T} \mathbf{A} \mathbf{R}^{-1} \vec{u} = \lambda \vec{u}$$

and thus  $\lambda$  is a regular eigenvalue of the matrix  $\mathbf{R}^{-T} \mathbf{A} \mathbf{R}^{-1}$ . Often  $\mathbf{B} = \text{diag}([w_1, w_2, w_3, \dots, w_n]) = \text{diag}(w_i)$  is a diagonal matrix with positive entries  $w_i$  and thus  $\mathbf{R} = \text{diag}(\sqrt{w_i})$  and  $\lambda$  is an eigenvalue of  $\text{diag}(\frac{1}{\sqrt{w_i}}) \mathbf{A} \text{diag}(\frac{1}{\sqrt{w_i}})$ . Rows and columns of  $\mathbf{A}$  have to be divided by  $\sqrt{w_i}$ , this preserves the symmetry of  $\mathbf{A}$ .



**3-21 Result : Facts on real, square matrices**

Examine a real  $n \times n$  matrix  $\mathbf{A}$ . If  $\mathbf{A}$  has  $n$  distinct real eigenvalues  $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \dots \leq \lambda_n$  then the following facts can be verified (by mathematicians).

- The corresponding eigenvectors  $\vec{v}_i$  are linearly independent. Thus the matrix with the eigenvectors as columns is invertible.

$$\mathbf{V} = [\vec{v}_1, \vec{v}_2, \vec{v}_3, \dots, \vec{v}_n] \in \mathbb{M}^{n \times n}$$

- The property eigenvalue  $\mathbf{A} \vec{v}_i = \lambda_i$  can be written in the matrix form

$$\mathbf{A} \mathbf{V} = \mathbf{V} \operatorname{diag}(\lambda_i)$$

and consequently

$$\mathbf{V}^{-1} \mathbf{A} \mathbf{V} = \operatorname{diag}(\lambda_i) \quad \text{and} \quad \mathbf{A} = \mathbf{V} \operatorname{diag}(\lambda_i) \mathbf{V}^{-1}.$$

This is a **diagonalization** of the matrix  $\mathbf{A}$ .

- Any vector  $\vec{x} \in \mathbb{R}^n$  can be written as a linear combination of the eigenvectors, i.e.

$$\vec{x} = \sum_{i=1}^n c_i \vec{v}_i.$$

As a system of linear equations this reads as

$$\mathbf{V} \vec{c} = [\vec{v}_1, \vec{v}_2, \vec{v}_3, \dots, \vec{v}_n] \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \vec{x}$$

$$\vec{c} = \mathbf{V}^{-1} \vec{x}.$$

◇

If the eigenvalues are not isolated (i.e.  $\lambda_i = \lambda_j$ ) then some of the above results may fail. The eigenvectors might not  $\vec{v}_i$  be linearly independent any more. As a consequence not all vectors  $\vec{x}$  are generated as linear combinations of eigenvectors. The situation improves drastically for symmetric matrices.

### 3-22 Result : Facts on symmetric, real matrices

If  $\mathbf{A}$  is a real, symmetric  $n \times n$  matrix, then the following facts can be verified (by mathematicians).

- $\mathbf{A}$  has  $n$  real eigenvalues  $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ .
- There are  $n$  eigenvectors  $\vec{e}_j$  for  $1 \leq j \leq n$  with  $\mathbf{A} \vec{e}_j = \lambda_j \vec{e}_j$ . All eigenvectors have length 1 and they are pairwise orthogonal, i.e.  $\langle \vec{e}_j, \vec{e}_i \rangle = 0$  if  $i \neq j$ . If  $\lambda_i \neq \lambda_j$  this is easy to verify.

$$\begin{aligned} (\lambda_i - \lambda_j) \langle \vec{e}_i, \vec{e}_j \rangle &= \lambda_i \langle \vec{e}_i, \vec{e}_j \rangle - \lambda_j \langle \vec{e}_i, \vec{e}_j \rangle = \langle \lambda_i \vec{e}_i, \vec{e}_j \rangle - \langle \vec{e}_i, \lambda_j \vec{e}_j \rangle \\ &= \langle \mathbf{A} \vec{e}_i, \vec{e}_j \rangle - \langle \vec{e}_i, \mathbf{A} \vec{e}_j \rangle = \langle \mathbf{A} \vec{e}_i, \vec{e}_j \rangle - \langle \mathbf{A} \vec{e}_i, \vec{e}_j \rangle = 0 \end{aligned}$$

Even if  $\lambda_i = \lambda_j$  the orthogonality  $\langle \vec{e}_i, \vec{e}_j \rangle = 0$  can be preserved.

- Examine the orthogonal matrix  $\mathbf{Q}$ , with the normalized eigenvectors  $\vec{e}_i$  as columns, i.e.

$$\mathbf{Q} = [\vec{e}_1, \vec{e}_2, \vec{e}_3, \dots, \vec{e}_n] \quad \text{with} \quad \mathbf{Q}^T \cdot \mathbf{Q} = \mathbb{I}_n \quad \text{and} \quad \mathbf{Q}^{-1} = \mathbf{Q}^T.$$

This leads to

$$\begin{aligned} \mathbf{A} \cdot \mathbf{Q} &= \mathbf{Q} \cdot \text{diag}(\lambda_j) \\ \mathbf{Q}^T \cdot \mathbf{A} \cdot \mathbf{Q} &= \text{diag}(\lambda_j) = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \\ \mathbf{A} &= \mathbf{Q} \cdot \text{diag}(\lambda_j) \cdot \mathbf{Q}^T \end{aligned}$$

This process is called **diagonalization** of the symmetric matrix  $\mathbf{A}$ . This result is extremely useful, e.g. to simplify general stress or strain situations to principal stress or strain situations, see Section 5.3 starting on page 325.

- Each vector  $\vec{x} \in \mathbb{R}^n$  can be written as a linear combination of the eigenvectors  $\vec{e}_i$ . The result for the general matrix leads to

$$\vec{c} = \mathbf{Q}^{-1} \vec{x} = \mathbf{Q}^T \vec{x} = \begin{bmatrix} \vec{e}_1^T \\ \vec{e}_2^T \\ \vec{e}_3^T \\ \vdots \\ \vec{e}_n^T \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{bmatrix} \langle \vec{e}_1, \vec{x} \rangle \\ \langle \vec{e}_2, \vec{x} \rangle \\ \langle \vec{e}_3, \vec{x} \rangle \\ \vdots \\ \langle \vec{e}_n, \vec{x} \rangle \end{bmatrix}$$

$$\vec{x} = \sum_{i=1}^n c_i \vec{e}_i = \sum_{i=1}^n \langle \vec{x}, \vec{e}_i \rangle \vec{e}_i$$

$$\vec{x} = \sum_{i=1}^n c_i \vec{e}_i = \sum_{i=1}^n \langle \vec{x}, \vec{e}_i \rangle \vec{e}_i$$

- The inverse matrix is easily computed if all eigenvalues and vectors are already known. Then use

$$\mathbf{A}^{-1} = (\mathbf{Q} \cdot \text{diag}(\lambda_j) \cdot \mathbf{Q}^T)^{-1} = \mathbf{Q}^{-T} \cdot \text{diag}(\lambda_j)^{-1} \cdot \mathbf{Q}^{-1} = \mathbf{Q} \cdot \text{diag}\left(\frac{1}{\lambda_j}\right) \cdot \mathbf{Q}^T.$$

This is **not** an efficient way to determine the inverse matrix, since the eigenvalues and vectors are difficult to determine. The results are more useful for analytical purposes.



Another characterization of the eigenvalues of a symmetric, positive definite matrix  $\mathbf{A}$  can be based on the Rayleigh quotient

$$\rho(\vec{x}) = \frac{\langle \vec{x}, \mathbf{A} \vec{x} \rangle}{\langle \vec{x}, \vec{x} \rangle}.$$

Assume that the eigenvalues  $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \dots \leq \lambda_n$  are sorted. When looking for an extremum of the function  $\langle \vec{x}, \mathbf{A} \vec{x} \rangle$ , subject to the constraint  $\|\vec{x}\| = 1$  use the Lagrange multiplier theorem and

$$\vec{\nabla} \langle \vec{x}, \mathbf{A} \vec{x} \rangle = 2 \mathbf{A} \vec{x} \quad \text{and} \quad \vec{\nabla} \langle \vec{x}, \vec{x} \rangle = 2 \vec{x}$$

to conclude that  $\mathbf{A} \vec{x} = \lambda \vec{x}$  for some factor  $\lambda$ . Using  $\langle \vec{x}, \mathbf{A} \vec{x} \rangle = \langle \vec{x}, \lambda \vec{x} \rangle = \lambda \|\vec{x}\|^2$  conclude

$$\lambda_1 = \min_{\|\vec{x}\|=1} \langle \vec{x}, \mathbf{A} \vec{x} \rangle \quad \text{and} \quad \lambda_n = \max_{\|\vec{x}\|=1} \langle \vec{x}, \mathbf{A} \vec{x} \rangle.$$

For other eigenvalues use a slight modification of this result. If  $\vec{e}_1$  is an eigenvector to the first eigenvalue use the fact that the eigenvectors to strictly larger eigenvalues are orthogonal to  $\vec{e}_1$ . This leads to a method to determine  $\lambda_2$  by

$$\lambda_2 = \min \{ \langle \vec{x}, \mathbf{A} \vec{x} \rangle \mid \|\vec{x}\| = 1 \text{ and } \vec{x} \perp \vec{e}_1 \}.$$

This result can be extended in the obvious way to obtain  $\lambda_3$  by

$$\lambda_3 = \min \{ \langle \vec{x}, \mathbf{A} \vec{x} \rangle \mid \|\vec{x}\| = 1, \vec{x} \perp \vec{e}_1 \text{ and } \vec{x} \perp \vec{e}_2 \}.$$

The other eigenvalues can also be characterized by looking at subspaces by the **Courant-Fischer Minimax Theorem**, see [GoluVanLoan96, Theorem 8.1.2] or [Axel94, Lemma 3.13].

$$\lambda_k = \max_{\dim S=n-k} \min_{\vec{x} \in S \setminus \{\vec{0}\}} \frac{\langle \vec{x}, \mathbf{A} \vec{x} \rangle}{\langle \vec{x}, \vec{x} \rangle}$$

**3–23 Example :** The matrix  $\mathbf{A}_n$  presented in Section 2.3.1 (page 32)

$$\mathbf{A}_n = \frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix}$$

with  $h = \frac{1}{n+1}$  is the finite difference approximation of the second derivative, i.e.

$$-\frac{d^2}{dx^2} u(x) = f(x) \quad \rightarrow \quad \mathbf{A}_n \vec{u} = \vec{f}.$$

The exact eigenvalues are given by

$$\lambda_j = \frac{4}{h^2} \sin^2(j \frac{\pi h}{2}) \quad \text{for} \quad 1 \leq j \leq n.$$

and the eigenvector  $\vec{v}_j$  are generated by discretizing the function  $\sin(x \frac{j\pi}{n+1})$  over the interval  $[0, 1]$  and thus has  $j$  extrema within the interval.

$$\vec{v}_j = \left( \sin\left(\frac{1j\pi}{n+1}\right), \sin\left(\frac{2j\pi}{n+1}\right), \sin\left(\frac{3j\pi}{n+1}\right), \dots, \sin\left(\frac{(n-1)j\pi}{n+1}\right), \sin\left(\frac{nj\pi}{n+1}\right) \right)$$

◇

### 3-24 Example : The mapping generated by a symmetric matrix

For a symmetric matrix  $\mathbf{A}$  the factorization

$$\mathbf{A} = \mathbf{Q} \operatorname{diag}(\lambda_i) \mathbf{Q}^T = \mathbf{Q} \operatorname{diag}(\lambda_i) \mathbf{Q}^{-1}$$

implies that the mapping generated by the matrix has a special structure:

1.  $\vec{y} = \mathbf{Q}^{-1} \vec{x} = \mathbf{Q}^T \vec{x}$  will rotate the vector  $\vec{x}$  to obtain  $\vec{y}$ .
2.  $\vec{z} = \operatorname{diag}(\lambda_i) \vec{y}$  will stretch or compress the  $i$ -th component  $y_i$  by the factors  $\lambda_i$ .
3.  $\mathbf{A} \vec{x} = \mathbf{Q} \vec{z}$  will invert the rotation form the first step.

This behavior is consistent with the mapping of the eigenvectors  $\vec{v}_i$ , i.e. the columns of  $\mathbf{Q}$ , since  $\mathbf{A} \vec{v}_i = \lambda_i \vec{v}_i$ . Vectors in the eigenspace spanned by  $\vec{v}_i$  are stretched (or compressed) by the factor  $\lambda_i$ . The image of the unit circle in  $\mathbb{R}^2$  after a mapping by a symmetric matrix is thus an ellipse with the semi-axis in the direction of the eigenvectors and the length given by the eigenvalues. The similar result for ellipsoids as images of the unit sphere in  $\mathbb{R}^n$  is valid.  $\diamond$

### 3.2.3 Level Sets of Quadratic Forms

Ex. 3.8

#### 3-25 Example : Level sets of quadratic forms

The quadratic form generated by a symmetric, positive definite  $n \times n$  matrix  $\mathbf{A}$  is given by the function

$$f(\vec{x}) = \langle \vec{x}, \mathbf{A} \vec{x} \rangle \quad \text{for all } \vec{x} \in \mathbb{R}^n .$$

These expressions have many applications, amongst them level sets for general Gaussian distributions (e.g. Figures 3.19, 3.20 and 3.23) or the regions of confidence for the parameters determined by linear or nonlinear regression (e.g. Figures 3.49, 3.58 and 3.59).

Use  $\vec{x} = \sum_{i=1}^n \frac{1}{\sqrt{\lambda_i}} c_i \vec{e}_i$  with  $\sum_{i=1}^n c_i^2 = r^2$  to examine the quadratic form

$$\begin{aligned} \langle \vec{x}, \mathbf{A} \vec{x} \rangle &= \left\langle \sum_{i=1}^n \frac{1}{\sqrt{\lambda_i}} c_i \vec{e}_i, \mathbf{A} \sum_{j=1}^n \frac{1}{\sqrt{\lambda_j}} c_j \vec{e}_j \right\rangle = \sum_{i=1}^n \sum_{j=1}^n \frac{1}{\sqrt{\lambda_i \lambda_j}} c_i c_j \langle \vec{e}_i, \mathbf{A} \vec{e}_j \rangle \\ &= \sum_{i=1}^n \sum_{j=1}^n \frac{1}{\sqrt{\lambda_i \lambda_j}} c_i c_j \langle \vec{e}_i, \lambda_j \vec{e}_j \rangle = \sum_{i=1}^n \frac{1}{\lambda_i} c_i^2 = r^2 . \end{aligned}$$

Thus the eigenvalues and eigenvectors of  $\mathbf{A}$  characterize the contour levels.

For  $n = 2$  the values  $(c_1, c_2) = r(\cos(\alpha), \sin(\alpha))$  satisfy  $c_1^2 + c_2^2 = r^2$  and thus the vectors  $\vec{x} = \frac{r}{\sqrt{\lambda_1}} \cos(\alpha) \vec{e}_1 + \frac{r}{\sqrt{\lambda_2}} \sin(\alpha) \vec{e}_2$  satisfy  $\|\vec{x}\| = r$ . The code below generates the level curves for  $\langle \vec{x}, \mathbf{A} \vec{x} \rangle = 1$  visible in Figure 3.14(a) and the rescaled eigenvectors.

```
A = [2 1.5; 1.5 3];
[Evec, Eval] = eig(A)
alpha = linspace(0,2*pi); Points = Evec*inv(sqrt(Eval))*[cos(alpha);sin(alpha)];
figure(1); plot(Points(1,:),Points(2,:),'b'); hold on
plot([0,Evec(1,1)]/sqrt(Eval(1,1)), [0,Evec(2,1)]/sqrt(Eval(1,1)),'k')
plot([0,Evec(1,2)]/sqrt(Eval(2,2)), [0,Evec(2,2)]/sqrt(Eval(2,2)),'k')
xlabel('x_1'); ylabel('x_2'); axis([-1 1 -1 1]); axis equal; hold off
```

Similar arguments generate the level surface in  $\mathbb{R}^3$ , visible in Figure 3.14(b). The computations use spherical coordinates

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = r \begin{pmatrix} \cos(\theta) \cos(\phi) \\ \cos(\theta) \sin(\phi) \\ \sin(\theta) \end{pmatrix} \quad \text{for } 0 \leq \phi \leq 2\pi \quad \text{and} \quad -\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2} .$$

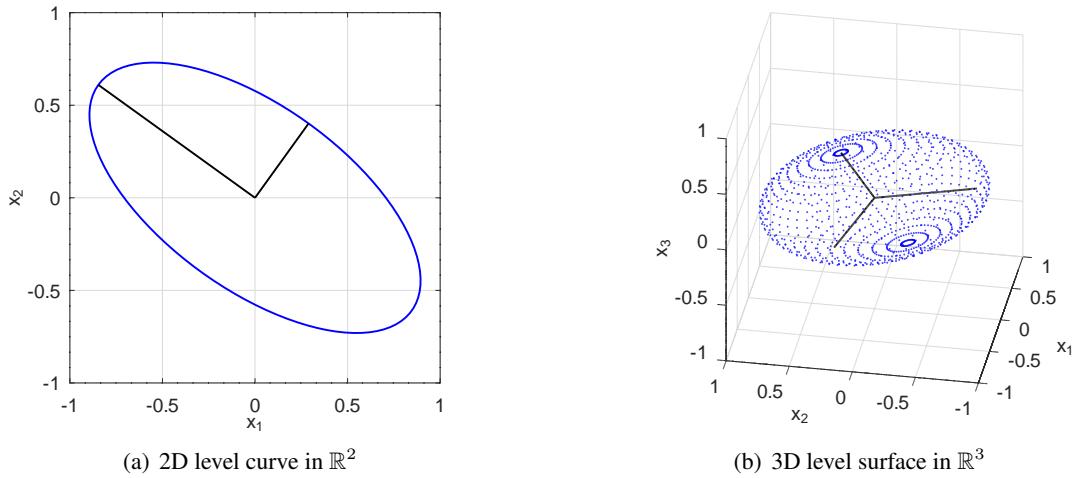


Figure 3.14: Level curve or surface of quadratic forms in  $\mathbb{R}^2$  or  $\mathbb{R}^3$ . The level curve and surface are shown in blue and in black the eigenvectors divided by  $\lambda_i$ .

```

A = [2 1.5 1; 1.5 3 0; 1 0 3];
[Evec, Eval] = eig(A)
phi = linspace(0,2*pi,51); theta = linspace(-pi,pi,51);
x = cos(phi')*cos(theta); y = sin(phi')*cos(theta); z = ones(size(phi'))*sin(theta);
Points = Evec*inv(sqrt(Eval))*[x(:),y(:),z(:)]';
figure(1); clf
    plot3(Points(1,:),Points(2,:),Points(3,:),'b')
    hold on
    plot3([0,Evec(1,1)]/sqrt(Eval(1,1)),...
           [0,Evec(2,1)]/sqrt(Eval(1,1)),...
           [0,Evec(3,1)]/sqrt(Eval(1,1)),'k')
    plot3([0,Evec(1,2)]/sqrt(Eval(2,2)),...
           [0,Evec(2,2)]/sqrt(Eval(2,2)),...
           [0,Evec(3,2)]/sqrt(Eval(2,2)),'k')
    plot3([0,Evec(1,3)]/sqrt(Eval(3,3)),...
           [0,Evec(2,3)]/sqrt(Eval(3,3)),...
           [0,Evec(3,3)]/sqrt(Eval(3,3)),'k')
    xlabel('x_1'); ylabel('x_2'); zlabel('x_3');
    axis([-1 1 -1 1 -1 1]);
    view([-80 30]);
    hold off

```

◇

### 3–26 Example : Ellipsoids as level sets

In the previous example the level curve of a quadratic function was visualized using eigenvalues and eigenvectors of a symmetric positive definite matrix  $\mathbf{A}$ . The process can be used in reverse order: given the three semi axis of an ellipsoid in  $\mathbb{R}^3$  by three orthogonal directions  $\vec{d}_i$  and the corresponding lengths  $l_i$ , find the quadratic form  $f(\vec{x}) = \langle \vec{x}, \mathbf{A} \vec{x} \rangle$  such that the ellipsoid is given as level set  $f(\vec{x}) = 1$ .

Assuming three directions  $\vec{d}_i$  are normalized ( $\|\vec{d}_i\| = 1$ ), then the matrix  $\mathbf{Q} = [\vec{d}_1, \vec{d}_2, \vec{d}_3]$  satisfies

$$\mathbf{Q}^T \mathbf{Q} = \mathbb{I}_3 \quad \text{and thus} \quad \mathbf{Q}^{-1} = \mathbf{Q}^T.$$

Any vector  $\vec{x} \in \mathbb{R}^3$  can be written as linear combination of the three vectors  $\vec{d}_i$ , i.e.  $\vec{x} = c_1 \vec{d}_1 + c_2 \vec{d}_2 + c_3 \vec{d}_3$

with

$$\mathbf{Q} \vec{c} = \vec{x} \quad \text{and} \quad \|\vec{c}\| = \|\mathbf{Q}^T \vec{x}\| = \|\vec{x}\|.$$

Then the matrix

$$\mathbf{A} = \mathbf{Q} \begin{bmatrix} 1/l_1^2 & 0 & 0 \\ 0 & 1/l_2^2 & 0 \\ 0 & 0 & 1/l_3^2 \end{bmatrix} \mathbf{Q}^T = (\mathbf{Q} \operatorname{diag}(1/l_i)) \operatorname{diag}(1/l_i) \mathbf{Q}^T$$

has the three eigenvectors  $\vec{d}_i$  and the eigenvalues  $\lambda_i = 1/l_i^2$ . It satisfies

$$\langle \pm l_i \vec{d}_i, \mathbf{A} (\pm l_i \vec{d}_i) \rangle = (\pm 1)^2 \langle l_i \vec{d}_i, \lambda_i (l_i \vec{d}_i) \rangle = \langle \vec{d}_i, l_i \frac{1}{l_i^2} \lambda_i l_i \vec{d}_i \rangle \langle \vec{d}_i, \vec{d}_i \rangle = \|\vec{d}_i\|^2 = 1,$$

i.e. the six endpoints of the axis of the ellipsoid are on the level set  $f(\vec{x}) = 1$ . Example 3–24 shows that multiplying by the matrix  $\mathbf{A}$  corresponds to a sequence of three mappings:

1.  $\vec{y} = \mathbf{Q}^{-1} \vec{x} = \mathbf{Q}^T \vec{x}$  will rotate the vector  $\vec{x}$  to obtain  $\vec{y}$ .
2.  $\vec{z} = \operatorname{diag}(\frac{1}{l_i^2}) \vec{y}$  will stretch or compress the  $i$ -th component  $y_i$  by the factors  $\frac{1}{l_i^2}$ .
3.  $\mathbf{A} \vec{x} = \mathbf{Q} \vec{z}$  will invert the rotation form the first step.

Thus the ellipsoid is given as level curve of  $f(\vec{x}) = \langle \vec{x}, \mathbf{A} \vec{x} \rangle = 1$ . ◇

The above examples allows to visualize level curves and surfaces in  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , but the task is more difficult when higher dimensions are necessary, i.e. for regression problems with more than three parameters, see e.g. Figure 3.51 on page 234 or Section 3.5.8 on page 243. There are (at least) two methods to reduce the dimensions to visualize the results:

not in class

- intersection with a coordinate plane with  $x_i = 0$  for some indices  $i$ .
- projection of the high dimensional ellipsoid onto a coordinate plane.

As example examine a problem with four independent variables, given by the symmetric matrix  $\mathbf{A} \in \mathbb{M}^{4 \times 4}$ , i.e.  $a_{i,j} = a_{j,i}$ . Examine reductions to expressions involving two variables only, e.g.  $x_1$  and  $x_2$ . The quadratic form to be examined is

$$f(\vec{x}) = \langle \mathbf{A} \vec{x}, \vec{x} \rangle = \left\langle \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \right\rangle = \sum_{i,j=1}^4 a_{i,j} x_i x_j.$$

To determine the intersection with the  $x_1 x_2$ -plane (i.e.  $x_3 = x_4 = 0$ ) it is convenient to write the matrix as composition of four  $2 \times 2$  block matrices

$$\mathbf{A} = \left[ \begin{array}{cc|cc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{array} \right] = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{13} \\ \mathbf{A}_{13}^T & \mathbf{A}_{33} \end{bmatrix}.$$

The restriction to the plane  $x_3 = x_4 = 0$  is characterized by

$$\begin{aligned} f(\vec{x}) &= \langle \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \\ 0 \\ 0 \end{pmatrix} \rangle \\ &= \langle \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle = \langle \mathbf{A}_{11} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle \end{aligned}$$

Thus the intersections of the level sets with the plane  $x_3 = x_4 = 0$  can be visualized with the tools from the above example.

To examine the projection on this plane requires more effort. Examine Figure 3.14(b) observe that the contour of the projection along the  $x_3$  axis onto the horizontal plane is characterized by the  $x_3$  component of the gradient (see page 74)  $\nabla f(\vec{x}) = 2 \mathbf{A} \vec{x}$  to vanish. For the four dimensional example this leads to the conditions of the last two components of the gradient to vanish., i.e.

$$\begin{aligned} \begin{pmatrix} 0 \\ 0 \end{pmatrix} &= \begin{bmatrix} a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \\ &= \begin{bmatrix} a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{bmatrix} a_{3,3} & a_{3,4} \\ a_{4,3} & a_{4,4} \end{bmatrix} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} = \mathbf{A}_{13}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \mathbf{A}_{33} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix}. \end{aligned}$$

This system can be solved for  $(x_3, x_4)$  as function of  $(x_1, x_2)$ .

$$\begin{pmatrix} x_3 \\ x_4 \end{pmatrix} = - \begin{bmatrix} a_{3,3} & a_{3,4} \\ a_{4,3} & a_{4,4} \end{bmatrix}^{-1} \begin{bmatrix} a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = -\mathbf{A}_{33}^{-1} \mathbf{A}_{31}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

If  $\mathbf{A}$  is positive definite, then  $\mathbf{A}_{11}$  and  $\mathbf{A}_{33}$  are positive definite too. With this notation the quadratic form is expressed in terms of  $(x_1, x_2)$  only.

$$\begin{aligned} f(\vec{x}) &= \langle \mathbf{A} \vec{x}, \vec{x} \rangle = \langle \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{13} \\ \mathbf{A}_{13}^T & \mathbf{A}_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \rangle \\ &= \langle \mathbf{A}_{11} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle + 2 \langle \mathbf{A}_{13} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle + \langle \mathbf{A}_{33} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix}, \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} \rangle \\ &= \langle \mathbf{A}_{11} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle - 2 \langle \mathbf{A}_{13} \mathbf{A}_{33}^{-1} \mathbf{A}_{13}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle + \\ &\quad + \langle \mathbf{A}_{33} \mathbf{A}_{33}^{-1} \mathbf{A}_{13}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \mathbf{A}_{33}^{-1} \mathbf{A}_{13}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle \\ &= \langle \mathbf{A}_{11} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle - 2 \langle \mathbf{A}_{13} \mathbf{A}_{33}^{-1} \mathbf{A}_{13}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle + \end{aligned}$$

$$\begin{aligned}
& + \langle \mathbf{A}_{13} \mathbf{A}_{33}^{-1} \mathbf{A}_{13}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle \\
= & \langle (\mathbf{A}_{11} - \mathbf{A}_{13} \mathbf{A}_{33}^{-1} \mathbf{A}_{13}^T) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rangle
\end{aligned}$$

Thus examine the level curves in  $\mathbb{R}^2$  of the quadratic form generated by the modified matrix  $\tilde{\mathbf{A}} \in \mathbb{M}^{2 \times 2}$

$$\tilde{\mathbf{A}} = \mathbf{A}_{11} - \mathbf{A}_{13} \mathbf{A}_{33}^{-1} \mathbf{A}_{13}^T.$$

In the case of a projection from  $\mathbb{R}^3$  onto  $\mathbb{R}^2$  along the  $x_3$  direction this simplifies slightly to

$$\begin{aligned}
f(\vec{x}) &= \langle \mathbf{A} \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} \rangle - \frac{1}{a_{3,3}} \left\langle \begin{bmatrix} a_{1,3} \\ a_{2,3} \end{bmatrix} \begin{bmatrix} a_{1,3} & a_{2,3} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\rangle \\
&= \left\langle \left( \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{1,2} & a_{2,2} \end{bmatrix} - \frac{1}{a_{3,3}} \begin{bmatrix} a_{1,3} \\ a_{2,3} \end{bmatrix} \begin{bmatrix} a_{1,3} & a_{2,3} \end{bmatrix} \right) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\rangle.
\end{aligned}$$

The above approach can be generalized to more than four dimensions and the variables to be removed can be chosen arbitrary. This is implemented in a MATLAB/Octave function.

#### ReduceQuadraticForm.m

```

function A_new = ReduceQuadraticForm(A, remove)
    stay = 1:length(A);
    for ii = 1:length(remove) %>>> %% remove the indicated components
        stay = stay(stay ~= remove(ii));
    end%for
    %% construct the sub matrices
    Asr = A(stay, remove); Arr = A(remove, remove);
    A_new = A(stay, stay) - Asr * (Arr \ Asr'); %% the new matrix
end%function

```

For the above matrix  $\mathbf{A} \in \mathbb{M}^{4 \times 4}$  the coordinates  $x_3$  and  $x_4$  are removed and the new matrix  $\tilde{\mathbf{A}} \in \mathbb{M}^{2 \times 2}$  is generated by calling `A_new = ReduceQuadraticForm(A, [3 4])`.

**3-27 Example : Intersection and Projection of Level Sets of a Quadratic Form onto Coordinate Plane**  
As example examine the ellipsoid in Figure 3.14(b) and determine the intersection and projection onto the coordinate plane  $x_3 = 0$ . Find the results of the code below in Figure 3.15. Observe that the axes of the two ellipses in Figure 3.15(a) are in slightly different directions.

```

A = [2 1.5 1; 1.5 3 0; 1 0 3];
[Evec, Eval] = eig(A);
phi = linspace(0, 2*pi, 51); theta = linspace(-pi, pi, 51);
x = cos(phi') * cos(theta); y = sin(phi') * cos(theta); z = ones(size(phi')) * sin(theta);
Points = Evec * inv(sqrt(Eval)) * [x(:), y(:), z(:)]';

%%% intersection with the plane x_3=0
A_intersect = A(1:2, 1:2);
[Evec_i, Eval_i] = eig(A_intersect)
x = cos(phi); y = sin(phi); z = zeros(size(phi));
Points_i = [Evec_i * inv(sqrt(Eval_i)) * [x(:), y(:)]'; z];

```

Demo\_Quadri

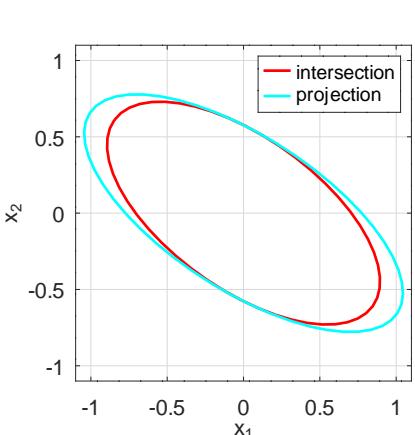
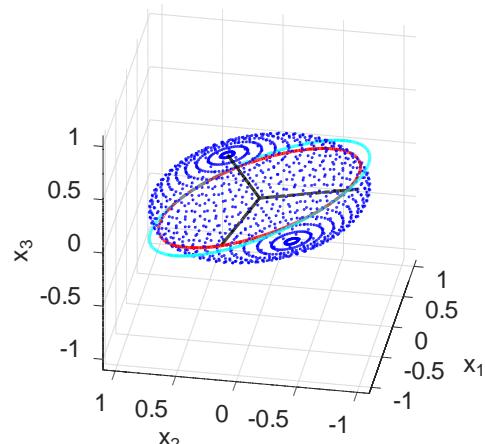
```

%%% projection onto the plane x_3=0
A_project = ReduceQuadraticForm(A, 3);
[Evec_p, Eval_p] = eig(A_project)
Points_p = [Evec_p*inv(sqrt(Eval_p))*[x(:),y(:)]';z];

figure(1); clf
plot3(Points(1,:),Points(2,:),Points(3,:),'b')
hold on
plot3([0,Evec(1,1)]/sqrt(Eval(1,1)),...
[0,Evec(2,1)]/sqrt(Eval(1,1)),...
[0,Evec(3,1)]/sqrt(Eval(1,1)),'k')
plot3([0,Evec(1,2)]/sqrt(Eval(2,2)),...
[0,Evec(2,2)]/sqrt(Eval(2,2)),...
[0,Evec(3,2)]/sqrt(Eval(2,2)),'k')
plot3([0,Evec(1,3)]/sqrt(Eval(3,3)),...
[0,Evec(2,3)]/sqrt(Eval(3,3)),...
[0,Evec(3,3)]/sqrt(Eval(3,3)),'k')
plot3(Points_i(1,:),Points_i(2,:),Points_i(3,:),'r')
plot3(Points_p(1,:),Points_p(2,:),Points_p(3,:),'c')
xlabel('x_1'); ylabel('x_2'); zlabel('x_3');
axis(1.1*[-1 1 -1 1 -1 1]); axis equal
view([-80 30]); hold off

figure(2); clf
plot(Points_i(1,:),Points_i(2,:),'r',
      Points_p(1,:),Points_p(2,:),'c')
xlabel('x_1'); ylabel('x_2');
legend('intersection','projection'); axis(1.1*[-1 1 -1 1]); axis equal

```

(a) in the coordinate plane  $x_3 = 0$ (b) in  $\mathbb{R}^3$ Figure 3.15: Level surface of a quadratic form in  $\mathbb{R}^3$  with intersection and projection onto  $\mathbb{R}^2$ 

### 3.2.4 Commands for Eigenvalues and Eigenvectors in Octave/MATLAB

MATLAB and Octave provide command to determine eigenvalues and eigenvectors for square matrices. In this section only the usage of these commands is illustrated, not attempt to examine the algorithms used is

made. Consult the bible of matrix computations [[GoluVanLoan13](#), §7, §8] for information on the algorithms to compute the eigenvalues. In [[IsaaKell66](#), §4] find a shorter presentation.

### The command `eig()` to compute all eigenvalues

With the command `eig()` all  $n$  eigenvalues or eigenvectors and eigenvalues of a  $n \times n$  can be determined. To compute the eigenvalues of the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

use

```
A = [1 2 3; 4 5 6; 7 8 9];
eval = eig(A)
-->
eval = 1.6117e+01
-1.1168e+00
-1.3037e-15
```

and to compute the eigenvectors and eigenvalues use

```
A = [1 2 3; 4 5 6; 7 8 9];
eval = eig(A)
[evec,eval] = eig(A)
-->
evec = -0.231971 -0.785830 0.408248
      -0.525322 -0.086751 -0.816497
      -0.818673  0.612328  0.408248

eval = Diagonal Matrix 1.6117e+01 0 0
                  0 -1.1168e+00 0
                  0 0 -1.3037e-15
```

The eigenvectors are all normalized to have length 1 and the eigenvalues are returned on the diagonal of a matrix. For symmetric matrices the eigenvectors are orthogonal.

The same command is used to compute generalized eigenvalues by supplying two input arguments. To examine

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \vec{v} = \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 100 \end{bmatrix} \vec{v}$$

use

```
A = [1 2 3; 4 5 6; 7 8 9]; B = diag([1 10 100]);
eval = eig(A,B)
-->
eval = 1.8221e+00
-2.3214e-01
1.5755e-19
```

### The command `eigs()` to compute a few eigenvalues

The computational effort to determine all eigenvalues of a large matrix can be huge, or it might not be possible at all. In many cases not all eigenvalues are required, but only the largest or smallest, and a good estimate is good enough. Thus Octave/MATLAB provide a command for this special task, `eigs()`. The basic idea for the algorithm used by `eigs()` is power iteration, see [GoluVanLoan13, §7.3], [IsaaKell66, §4.2] or [Demm97]. Find a detailed description in the lecture notes [Stah00, §11.5] available on my web site at <https://web.sha1.bfh.science/fem/VarFEM/VarFEM.pdf>.

As example consider the model matrices  $\mathbf{A}_n$  and  $\mathbf{A}_{nn}$  from Section 2.3 (see page 32). For these matrices the exact eigenvalues are known, `eigs()` will return estimates of some of the eigenvalues. To estimate the three largest eigenvalues of the  $1000 \times 1000$  matrix  $\mathbf{A}_n$  for  $n = 1000$  use

```
n = 1000; h = 1/(n+1);
A_n = spdiags(ones(n,1)*[-1, 2, -1], [-1, 0, 1], n, n) / (h^2);
lambda_max = 4/h^2*sin(n*pi*h/2)^2
options.issysm = true; options.tol = n^2*1e-5;
lambda = eigs(A_n, 3, 'lm', options)
-->
lambda_max = 4.0080e+06
lambda      = 4.0019e+06
            3.9569e+06
            3.8748e+06
```

and for the three smallest eigenvalues

```
lambda_min = 4/h^2*sin(pi*h/2)^2
lambda = eigs(A_n, 3, 'sm', options)
-->
lambda_min = 9.8696
lambda      = 88.8258
            39.4783
            9.8696
```

To estimate the three largest eigenvalues of the  $10'000 \times 10'000$  matrix  $\mathbf{A}_{nn}$  for  $n = 100$  use

```
n = 100; h = 1/(n+1);
A_n = spdiags(ones(n,1)*[-1, 2, -1], [-1, 0, 1], n, n) / (h^2);
A_nn = kron(speye(n), A_n) + kron(A_n, speye(n));
lambda_max = 8/h^2*sin(n*pi*h/2)^2
options.issysm = true; options.tol = n^2*1e-5;
lambda = eigs(A_nn, 3, 'lm', options)
-->
lambda_max = 8.1588e+04
lambda      = 8.1344e+04
            8.0238e+04
            7.8325e+04
```

This code takes less than 0.02 seconds to run, using `eig()` takes 46 seconds on a fast desktop computer.

Using estimates for the smallest and largest eigenvalue allows to estimate the condition number of the matrix. The command `condeest()` uses similar ideas to estimate the condition number of a matrix.

### 3.2.5 Eigenvalues and Systems of Ordinary Differential Equations

A general linear system of ordinary differential equations is given by

$$\frac{d}{dt} \vec{x}(t) = \mathbf{A}(t) \vec{x}(t) + \vec{f}(t) .$$

One can show that any solution is of the form

$$\vec{x}(t) = \vec{x}_h(t) + \vec{x}_p(t)$$

where

- $\vec{x}_p(t)$  is one particular solution
- $\vec{x}_h(t)$  is a solution of the corresponding homogeneous system  $\frac{d}{dt} \vec{x}_h(t) = \mathbf{A}(t) \vec{x}_h(t)$

Because of this structure one may examine the homogeneous problem to study the stability of numerical algorithms to solve the system of ODEs, see Section 3.4.

For constant matrices  $\mathbf{A}(t) = \mathbf{A}$  the eigenvalues and eigenvectors provide a lot of information on the solutions of the homogeneous problem  $\frac{d}{dt} \vec{x}_h(t) = \mathbf{A} \vec{x}_h(t)$ . This will lead to Result 3–29 below.

Examine the system of linear, homogeneous ODEs

$$\frac{d}{dt} \vec{x}(t) = \mathbf{A} \vec{x}(t) . \quad (3.4)$$

where the real  $n \times n$  matrix  $\mathbf{A}$  has real eigenvalues  $\lambda_i$  with corresponding eigenvectors  $\vec{v}_i \in \mathbb{R}^n$ .

- (a) Any function  $\vec{x}(t) = e^{\lambda_i t} \vec{v}_i$  solves (3.4), since

$$\frac{d}{dt} \vec{x}(t) = \left( \frac{d}{dt} e^{\lambda_i t} \right) \vec{v}_i = \lambda_i e^{\lambda_i t} \vec{v}_i \quad \text{and} \quad \mathbf{A} \vec{x}(t) = e^{\lambda_i t} \mathbf{A} \vec{v}_i = e^{\lambda_i t} \lambda_i \vec{v}_i .$$

- (b) Any linear combination of above solutions also solves the ODE (3.4), i.e.  $\vec{x}(t) = \sum_{i=1}^n c_i e^{\lambda_i t} \vec{v}_i$  is a solution. To verify this examine

$$\begin{aligned} \frac{d}{dt} \vec{x}(t) &= \sum_{i=1}^n c_i \frac{d}{dt} e^{\lambda_i t} \vec{v}_i = \sum_{i=1}^n c_i \lambda_i e^{\lambda_i t} \vec{v}_i \\ \mathbf{A} \vec{x}(t) &= \sum_{i=1}^n c_i e^{\lambda_i t} \mathbf{A} \vec{v}_i = \sum_{i=1}^n c_i e^{\lambda_i t} \lambda_i \vec{v}_i . \end{aligned}$$

- (c) If the above  $n$  eigenvectors  $\vec{v}_i$  are linearly independent, then any initial value  $\vec{x}(0) = \vec{x}_0$  can be written as a linear combination of the eigenvectors, i.e.

$$\vec{x}(0) = \vec{x}_0 = \sum_{i=1}^n c_i \vec{v}_i .$$

Verify that solving for the coefficients  $c_i$  leads to a system of linear equations. Using this solve (3.4) with an initial condition  $\vec{x}(0) = \vec{x}_0$ . With the notation

$$\vec{v}_i = \begin{pmatrix} v_{1,i} \\ v_{2,i} \\ v_{3,i} \\ \vdots \\ v_{n,i} \end{pmatrix} \quad \text{and} \quad \vec{x}_0 = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix}$$

the equation  $\vec{x}_0 = \sum_{i=1}^n c_i \vec{v}_i$  leads to

$$\begin{aligned} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} &= c_1 \begin{pmatrix} v_{1,1} \\ v_{2,1} \\ v_{3,1} \\ \vdots \\ v_{n,1} \end{pmatrix} + c_2 \begin{pmatrix} v_{1,2} \\ v_{2,2} \\ v_{3,2} \\ \vdots \\ v_{n,2} \end{pmatrix} + c_3 \begin{pmatrix} v_{1,3} \\ v_{2,3} \\ v_{3,3} \\ \vdots \\ v_{n,3} \end{pmatrix} + \cdots + c_n \begin{pmatrix} v_{1,n} \\ v_{2,n} \\ v_{3,n} \\ \vdots \\ v_{n,n} \end{pmatrix} \\ &= \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} & \cdots & v_{1,n} \\ v_{2,1} & v_{2,2} & v_{2,3} & \cdots & v_{2,n} \\ v_{3,1} & v_{3,2} & v_{3,3} & \cdots & v_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{n,1} & v_{n,2} & v_{n,3} & \cdots & v_{n,n} \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix}. \end{aligned}$$

If the eigenvectors  $\vec{v}_i$  are linearly independent, the system has a unique solution.

**3–28 Example :** Examine the system of ODEs

Ex. 3.9

$$\begin{aligned} \dot{x}(t) &= x(t) + 3y(t) \\ \dot{y}(t) &= 4x(t) + y(t) \end{aligned}$$

or using the matrix notation

$$\frac{d}{dt} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{bmatrix} 1 & 3 \\ 4 & 1 \end{bmatrix} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}.$$

For the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 4 & 1 \end{bmatrix}$$

the eigenvalues are given as solutions of a quadratic equation

$$\det \begin{bmatrix} 1 - \lambda & 3 \\ 4 & 1 - \lambda \end{bmatrix} = (1 - \lambda)^2 - 12 = \lambda^2 - 2\lambda - 11 = 0$$

with the solutions

$$\lambda_1 = \frac{1}{2} (2 + \sqrt{4 + 44}) \approx 4.4641, \quad \lambda_2 = \frac{1}{2} (2 - \sqrt{4 + 44}) \approx -2.4641$$

and the corresponding eigenvectors

$$\vec{v}_1 \approx \begin{pmatrix} 1 \\ 1.1547 \end{pmatrix}, \quad \vec{v}_2 \approx \begin{pmatrix} 1 \\ -1.1547 \end{pmatrix}.$$

Thus the general solution of the system of ODEs is given by

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \approx c_1 \begin{pmatrix} 1 \\ 1.1547 \end{pmatrix} e^{4.4641 t} + c_2 \begin{pmatrix} 1 \\ -1.1547 \end{pmatrix} e^{-2.4641 t}.$$

Find the corresponding vector field, two solutions and the eigen-directions in Figure 3.16.



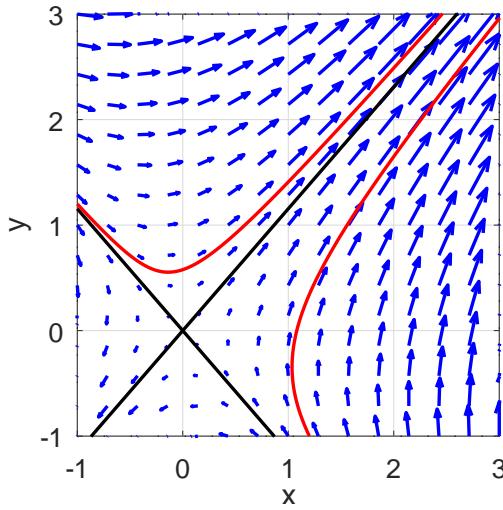


Figure 3.16: A linear vector field with the eigenvectors

If  $\lambda = \alpha + i\beta \in \mathbb{C}$  is a complex eigenvalue of a real matrix  $\mathbf{A}$  with eigenvector  $\vec{u} + i\vec{v}$  then  $\bar{\lambda} = \alpha - i\beta$  is also eigenvalue with eigenvector  $\vec{u} - i\vec{v}$ . Now use

$$\begin{aligned}\mathbf{A} \exp((\alpha + i\beta)t)(\vec{u} + i\vec{v}) &= (\alpha + i\beta) \exp((\alpha + i\beta)t)(\vec{u} + i\vec{v}) \\ \frac{d}{dt} \exp((\alpha + i\beta)t)(\vec{u} + i\vec{v}) &= (\alpha + i\beta) \exp((\alpha + i\beta)t)(\vec{u} + i\vec{v})\end{aligned}$$

and examine

$$\begin{aligned}\exp((\alpha + i\beta)t)(\vec{u} + i\vec{v}) &= \exp(\alpha t) \exp(i\beta t)(\vec{u} + i\vec{v}) \\ &= \exp(\alpha t)(\cos(\beta t) + i \sin(\beta t))(\vec{u} + i\vec{v}) \\ &= \exp(\alpha t)(\cos(\beta t)\vec{u} - \sin(\beta t)\vec{v}) + i \exp(\alpha t)(\cos(\beta t)\vec{v} + \sin(\beta t)\vec{u})\end{aligned}$$

and its conjugate complex to conclude that

$$e^{\alpha t}(\cos(\beta t)\vec{u} - \sin(\beta t)\vec{v}) \quad \text{and} \quad e^{\alpha t}(\cos(\beta t)\vec{v} + \sin(\beta t)\vec{u})$$

are linearly independent solutions of the system of ODEs  $\frac{d}{dt}\vec{x}(t) = \mathbf{A}\vec{x}(t)$ . These solutions move in the plane in  $\mathbb{R}^n$  spanned by  $\vec{u}$  and  $\vec{v}$ . They grow (or shrink) exponentially like  $e^{\alpha t}$  and rotate in the plane with angular velocity  $\beta$ .

The above observations can be collected and lead to a complete description of the solution of the homogeneous system  $\frac{d}{dt}\vec{x}(t) = \mathbf{A}\vec{x}(t)$ , assuming that all eigenvalues are isolated.

### 3-29 Result : Linear systems of ODEs with simple eigenvalues

Let  $\lambda_j \in \mathbb{R}$ ,  $j = 1, \dots, n_1$  be isolated eigenvalues of the  $n \times n$  matrix  $\mathbf{A}$  with eigenvectors  $\vec{e}_j$ . Use the isolated, complex eigenvalues  $\lambda_k = \alpha_k + i\beta_k \in \mathbb{C}$ ,  $\alpha_k, \beta_k \in \mathbb{R}$ ,  $\beta_k > 0$  for  $k = 1, \dots, n_2$  with eigenvectors  $\vec{u}_k + i\vec{v}_k$ . If these are all the eigenvalues (i.e.  $n_1 + 2n_2 = n$ ), then all solutions of the system  $\frac{d}{dt}\vec{x}(t) = \mathbf{A}\vec{x}(t)$  are given by

$$\begin{aligned}\vec{x}_h(t) &= \sum_{j=1}^{n_1} c_j e^{\lambda_j t} \vec{e}_j + \\ &\quad + \sum_{k=1}^{n_2} e^{\alpha_k t} (r_k (\cos(\beta_k t) \vec{u}_k - \sin(\beta_k t) \vec{v}_k) + s_k (\cos(\beta_k t) \vec{v}_k + \sin(\beta_k t) \vec{u}_k))\end{aligned}$$

with real constants  $c_j, r_k, s_k$ . ◇

**3-30 Example :** Examine the linear system

$$\frac{d}{dt} \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{bmatrix} 0.39 & 0.51 & 1.26 \\ -0.68 & 0 & -0.44 \\ -1.47 & -0.05 & -0.5 \end{bmatrix} \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}.$$

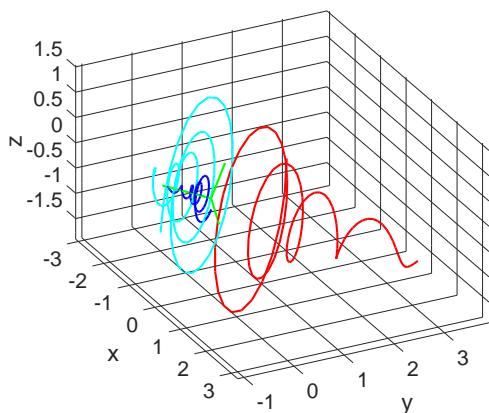
The code below generates the eigenvalues and eigenvectors

$$\lambda_{1,2} \approx -0.10 \pm i 1.41 \quad \text{with} \quad \vec{e}_{1,2} = \vec{u} \pm i \vec{v} \approx \begin{pmatrix} 0.67 \\ -0.18 \\ -0.19 \end{pmatrix} \pm i \begin{pmatrix} 0 \\ 0.28 \\ 0.64 \end{pmatrix}$$

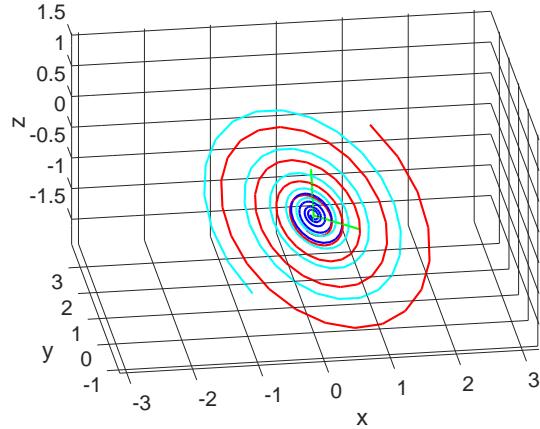
and

$$\lambda_3 \approx +0.095 \quad \text{with} \quad \vec{e}_3 \approx \begin{pmatrix} -0.13 \\ -0.91 \\ 0.40 \end{pmatrix}.$$

In the plane generated by the vectors  $\vec{u}$  and  $\vec{v}$  obtain exponential spirals with decaying radius  $c_r e^{-0.1t}$ . In the direction given by  $\vec{e}_3$  the solution is moving away from the origin, the distance described by a function  $c e^{+0.095t}$ . This behavior is confirmed by Figure 3.17 by the graphs of three solutions.



(a) view onto the direction  $\vec{e}_3$



(b) view onto the plane generated by  $\vec{u}$  and  $\vec{v}$

Figure 3.17: Spirals as solutions of a system of three differential equations

#### StabilityExample.m

```
A = [0.39 0.51 1.26; -0.68 0 -0.44; -1.47 -0.05 -0.5]
[EV,EW] = eig(A)

t = linspace(0,20);
[t1,x1] = ode45(@(t,x)A*x,t,[1;1;1]);
[t2,x2] = ode45(@(t,x)A*x,t,[0.4,-0.2,0]);
[t3,x3] = ode45(@(t,x)A*x,t,[-1,-0.5,-1]);

figure(1) % draw the eigenvectors in green
plot3([0,real(EV(1,1))],[0,real(EV(2,1))],[0,real(EV(3,1))],'g',...
[0,imag(EV(1,1))],[0,imag(EV(2,1))],[0,imag(EV(3,1))],'g',...
[0,real(EV(1,3))],[0,real(EV(2,3))],[0,real(EV(3,3))],'g')
```

```

hold on % draw three solutions in different colors
plot3(x1(:,1),x1(:,2),x1(:,3),'r',x2(:,1),x2(:,2),x2(:,3),'b',...
       x3(:,1),x3(:,2),x3(:,3),'c')
xlabel('x'); ylabel('y'); zlabel('z'); axis equal; hold off
-->
EV =
0.66972 + 0.00000i 0.66972 - 0.00000i -0.13030 + 0.00000i
-0.17882 + 0.27667i -0.17882 - 0.27667i -0.90807 + 0.00000i
-0.18947 + 0.63801i -0.18947 - 0.63801i 0.39803 + 0.00000i
EW = Diagonal Matrix
-0.10264 + 1.41104i 0 0
0 -0.10264 - 1.41104i 0
0 0 0.09529 + 0.00000i

```

◇

The situation changes slightly if second order systems are examined. Wave equations lead to this type of ODEs, see e.g. Section 4.6.

### 3–31 Result : Systems of second order, linear ordinary differential equations

Let  $\mathbf{A} \in \mathbb{R}^{n \times n}$  be a real valued matrix with positive eigenvalues  $\lambda_i = \omega_i^2 > 0$  and the corresponding Ex. 3.10 eigenvectors  $\vec{v}_i$ . Verify that the second order system of linear differential equations

$$\frac{d^2}{dt^2} \vec{x}(t) = -\mathbf{A} \vec{x}(t)$$

is solved by

$$\vec{x}(t) = \cos(\omega_i t) \vec{v}_i \quad \text{and} \quad \vec{x}'(t) = \sin(\omega_i t) \vec{v}_i.$$

Using trigonometric identities these two solutions can be combined to have one amplitude  $a_i$  and a phase shift  $\phi_i$  by

$$\vec{x}(t) = (c_1 \cos(\omega_i t) + c_2 \sin(\omega_i t)) \vec{v}_i = a_i \cos(\omega_i (t - \phi_i)) \vec{v}_i.$$

As a consequence the frequencies of these oscillations are determined by the eigenvalues  $\lambda_i = \omega_i^2$ . ◇

**Proof :** For  $\vec{x}(t) = \cos(\omega_i t) \vec{v}_i$  compute

$$\begin{aligned} \frac{d^2}{dt^2} \vec{x}(t) &= \left( \frac{d^2}{dt^2} \cos(\omega_i t) \right) \vec{v}_i = -\omega_i^2 \cos(\omega_i t) \vec{v}_i \\ -\mathbf{A} \vec{x}(t) &= -\cos(\omega_i t) \mathbf{A} \vec{v}_i = -\cos(\omega_i t) \lambda_i \vec{v}_i = -\omega_i^2 \cos(\omega_i t) \vec{v}_i \end{aligned}$$

and thus the system of differential equations is solved. The computation for the second solution  $\sin(\omega_i t) \vec{v}_i$  is similar. □

The above could also be examined by Result 3–29. Translate the system of  $n$  second order equations to a system of  $2n$  ODEs of order one. For this introduce the new variable  $\vec{v}(t) = \frac{d}{dt} \vec{x}(t) \in \mathbb{R}^n$  and examine the  $(2n) \times (2n)$  system

$$\frac{d}{dt} \begin{pmatrix} \vec{x}(t) \\ \vec{v}(t) \end{pmatrix} = \begin{pmatrix} \vec{v}(t) \\ -\mathbf{A} \vec{x}(t) \end{pmatrix} = \begin{bmatrix} 0 & \mathbb{I}_n \\ -\mathbf{A} & 0 \end{bmatrix} \begin{pmatrix} \vec{x}(t) \\ \vec{v}(t) \end{pmatrix}.$$

The generalized eigenvalues and eigenvectors satisfy  $\mathbf{A} \vec{v} = \lambda \mathbf{B} \vec{v}$  and with those ODEs with a weight matrix can be solved, i.e.

$$\mathbf{B} \frac{d}{dt} \vec{x}(t) = \mathbf{A} \vec{x}(t)$$

is solved by  $\vec{x}(t) = e^{\lambda t} \vec{v}$ . To verify this use  $\mathbf{B} \frac{d}{dt} \vec{x}(t) = \lambda e^{\lambda t} \mathbf{B} \vec{v}$  and  $\mathbf{A} \frac{d}{dt} \vec{x}(t) = \lambda e^{\lambda t} \mathbf{B} \vec{v}$ . A similar result is correct for second order ODEs

$$\mathbf{B} \frac{d^2}{dt^2} \vec{x}(t) = -\mathbf{A} \vec{x}(t),$$

which are solved by

$$\vec{x}(t) = (c_1 \cos(\omega_i t) + c_2 \sin(\omega_i t)) \vec{v}_i = a_i \cos(\omega_i (t - \phi_i)) \vec{v}_i,$$

where the generalized eigenvalue  $\mathbf{A} \vec{v}_i = \lambda_i \mathbf{B} \vec{v}_i$  is given by  $\lambda_i = \omega_i^2$ .

### 3.2.6 SVD, Singular Value Decomposition

For non-symmetric, or even non-square, matrices  $\mathbf{A}$  the idea of diagonalization of a matrix (see Result 3–22 on page 134) can be generalized, leading to the singular value decomposition (SVD) of the matrix.

**3–32 Theorem :** [GoluVanLoan13, §2.4]

If  $\mathbf{A} \in \mathbb{R}^{m \times n}$  is a real  $m \times n$  matrix, then there exist orthogonal matrices  $\mathbf{U} \in \mathbb{R}^{m \times m}$  and  $\mathbf{V} \in \mathbb{R}^{n \times n}$  and singular values  $\sigma_i$  such

$$\mathbf{U}^T \mathbf{A} \mathbf{V} = \Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_p) \in \mathbb{R}^{m \times n} \quad \text{where } p = \min\{n, m\} \quad (3.5)$$

and  $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \dots \geq \sigma_p \geq 0$ .  $\diamond$

The matrices  $\mathbf{U}$  and  $\mathbf{V}$  are orthogonal, i.e.  $\mathbf{U}^T \mathbf{U} = \mathbb{I}_m$  and  $\mathbf{V}^T \mathbf{V} = \mathbb{I}_n$  and since the left inverse matrix is a right inverse to  $\mathbf{U} \mathbf{U}^T = \mathbb{I}_m$  and  $\mathbf{V} \mathbf{V}^T = \mathbb{I}_n$ .

As consequence of the above find the SVD (Singular Value Decomposition) of the  $n \times n$  matrix  $\mathbf{A}$ .

$$\mathbf{A} = \mathbf{U} \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_p) \mathbf{V}^T = \mathbf{U} \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & \sigma_n \end{bmatrix} \mathbf{V}^T$$

With MATLAB/Octave the singular value decomposition can be computed by  $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{svd}(\mathbf{A})$ . It is not difficult to see that for square matrices  $\mathbf{A}$  and the usual 2-norm we have

$$\|\mathbf{A}\|_2 = \sigma_1, \quad \|\mathbf{A}^{-1}\|_2 = \frac{1}{\sigma_n} \quad \text{and} \quad \text{cond}(\mathbf{A}) = \frac{\sigma_1}{\sigma_n}.$$

If the matrix  $\mathbf{A}$  is symmetric and positive definite, then  $\mathbf{U} = \mathbf{V}$  and

$$\mathbf{A} \mathbf{U} = \mathbf{U} \Sigma$$

implies that the singular values are given by the eigenvalues  $\lambda_i = \sigma_i$  and in the columns of the orthogonal matrix  $\mathbf{U}$  find the normalized eigenvectors of  $\mathbf{A}$ . Thus the SVD coincides with the diagonalization of the matrix  $\mathbf{A}$ , as examined in Result 3–22 on page 134. If the real matrix  $\mathbf{A}$  is symmetric but not necessary positive definite, then  $\sigma_i = |\lambda_i|$  and for the corresponding columns (i.e. eigenvectors) find  $\vec{v}_i = -\vec{u}_i$ . Then  $\mathbf{V}^T \mathbf{U}$  is a diagonal matrix with numbers  $\pm 1$  on the diagonal.

If the  $m \times n$  matrix  $\mathbf{A}$  has more rows than columns ( $m > n$ ) then the factorization has the form

$$\mathbf{A} = \mathbf{U} \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & \sigma_n \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \mathbf{V}^T = \mathbf{U}_{econ} \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & \sigma_n \end{bmatrix} \mathbf{V}^T.$$

Since the matrix  $\mathbf{D}$  has  $m - n$  rows with zeros at the bottom, the last  $m - n$  columns of  $\mathbf{U}$  will be multiplied by zeros. There is no need to compute these columns and the command `[U,D,V] = svd(A,'econ')` generates this more economic form.

The SVD has many more applications: image processing, data compression, regression, robotics, ... Search on the internet for the keywords Professor SVD, Gene Golub and find an excellent article about Gene Golub and SVD by Cleve Moler, the founder of MATLAB.

### 3-33 Example : Approximation of a matrix by a SVD

Examine a random  $8 \times 4$  matrix  $\mathbf{A}$  and compute the SVD by

```
A = round(100*rand(8,4))
[U,D,V] = svd(A,'econ');
```

This leads to the matrix

$$\mathbf{A} = \begin{bmatrix} 45 & 91 & 51 & 25 \\ 23 & 93 & 13 & 31 \\ 32 & 66 & 43 & 21 \\ 71 & 8 & 4 & 2 \\ 30 & 27 & 78 & 61 \\ 96 & 65 & 12 & 16 \\ 96 & 59 & 18 & 69 \\ 77 & 81 & 31 & 36 \end{bmatrix}.$$

The entries in the diagonal matrix  $\mathbf{D}$  of the SVD are [279.982, 93.171, 75.224, 41.348] and thus the first entry is considerably larger than the others and one may approximate the matrix by using the first columns of  $\mathbf{U}$  and  $\mathbf{V}$  only, i.e.

$$\mathbf{A} \approx \mathbf{A}_1 = \vec{u}_1 \cdot 279.982 \cdot \vec{v}_1^T.$$

This matrix  $\mathbf{A}_1$  has a simple structure, as illustrated by the trivial example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 10 \\ 2 & 4 & 20 \\ 3 & 6 & 30 \\ -1 & -2 & -10 \end{bmatrix}.$$

Each row in  $\mathbf{A}_1$  is a multiple of the row vector  $\vec{v}_1^T$  and each column in  $\mathbf{A}_1$  is a multiple of the column vector  $\vec{u}_1$ . Using the first two columns leads to

$$\mathbf{A} \approx \mathbf{A}_2 = \begin{bmatrix} \vec{u}_1 & \vec{u}_2 \end{bmatrix} \cdot \begin{bmatrix} 279.982 & 0 \\ 0 & 93.171 \end{bmatrix} \cdot \begin{bmatrix} \vec{v}_1^T \\ \vec{v}_2^T \end{bmatrix}.$$

Each row in  $\mathbf{A}_2$  is a linear combination of the two row vectors  $\vec{v}_1^T$  and  $\vec{v}_2^T$  and each column in  $\mathbf{A}_2$  is a linear combination of the two column vectors  $\vec{u}_1$  and  $\vec{u}_2$ . The norm of these matrices and their differences are

$$\|\mathbf{A}\| = \|\mathbf{A}_1\| = \|\mathbf{A}_2\| = 279.982 , \quad \|\mathbf{A}_1 - \mathbf{A}\| = 93.171 \quad \text{and} \quad \|\mathbf{A}_2 - \mathbf{A}\| = 75.224$$

and the values are not too far from the original matrix  $\mathbf{A}$ .

$$\mathbf{A}_1 \approx \begin{bmatrix} 67.00 & 71.43 & 34.06 & 38.14 \\ 53.84 & 57.40 & 27.37 & 30.65 \\ 50.08 & 53.39 & 25.46 & 28.51 \\ 30.42 & 32.43 & 15.46 & 17.31 \\ 48.98 & 52.21 & 24.90 & 27.88 \\ 66.39 & 70.77 & 33.75 & 37.79 \\ 76.25 & 81.29 & 38.76 & 43.40 \\ 73.41 & 78.26 & 37.32 & 41.79 \end{bmatrix}, \quad \mathbf{A}_2 \approx \begin{bmatrix} 45.43 & 80.66 & 50.48 & 44.07 \\ 34.68 & 65.61 & 41.96 & 35.92 \\ 31.17 & 61.49 & 39.86 & 33.71 \\ 65.22 & 17.53 & -11.03 & 7.73 \\ 17.58 & 65.65 & 48.78 & 36.52 \\ 96.11 & 58.05 & 11.13 & 29.61 \\ 96.95 & 72.43 & 23.01 & 37.70 \\ 78.21 & 76.21 & 33.67 & 40.47 \end{bmatrix}$$

◇

### 3–34 Example : Image compression by SVD

The above idea can be used to compress images, by using only the first few contributions of the SVD. The original picture in Figure 3.18 is a  $480 \times 480$  digital picture. Using SVD only  $n$  components are used, with  $n = 5, 10, 20$  and  $40$ .

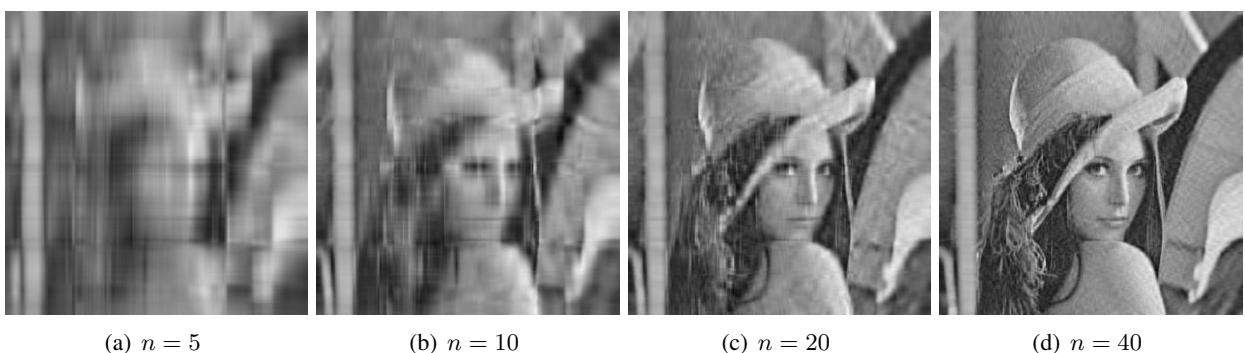


Figure 3.18: Image compression by SVD with different number  $n$  of contributions

```
pkg load image % Octave only
im = rgb2gray(imread('Lenna.jpg'));
[U,D,V] = svd(im);
n = 10;
imNew = mat2gray(U(:,1:n)*D(1:n,1:n)*V(:,1:n)');
figure(1); imshow(imNew)
```

Find a short presentation on SVD for image compression in [HuntLipsRose14, p.172–177].

◇

### 3.2.7 From Gaussian Distribution to Covariance, and then to PCA

In this section the transformation of Gaussian normal distributions by linear or affine transformations is examined. This is useful to understand the domains of confidence used by linear and nonlinear regression. As consequence the tool of principal component analysis (PCA) can be examined and the concepts and interpretation of covariance and correlation matrices are presented. Gaussian distributions and PCA are essential tools for machine learning, see e.g. [DeisFaisOng20, §6, §10].

#### Gaussian probability distributions in $\mathbb{R}^n$

The standard Gaussian PDF with mean  $\mu$  and variance  $\sigma^2$  is given by

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Examine a random variable  $\vec{x} \in \mathbb{R}^n$  with  $n$  **independent** components  $x_i$  with means  $\mu_i$  and variances  $\sigma_i^2$ . In this case the Gaussian distribution in  $\mathbb{R}^n$  is given by the PDF

$$\begin{aligned} p(\vec{x} | \vec{\mu}, \vec{\sigma}) &= \frac{1}{\sqrt{2\pi^n} \prod_{i=1}^n \sigma_i} \exp\left(-\frac{1}{2} \sum_{i=1}^n \frac{(x_i - \mu_i)^2}{\sigma_i^2}\right) \\ \ln(p(\vec{x} | \vec{\mu}, \vec{\sigma})) &= -\frac{n \ln(2\pi)}{2} - \sum_{i=1}^n \ln(\sigma_i) - \frac{1}{2} \sum_{i=1}^n \frac{(x_i - \mu_i)^2}{\sigma_i^2}. \end{aligned}$$

For the standardized Gaussian distribution with means 0 and variances 1 obtain

$$p(\vec{x} | \vec{0}, \vec{I}) = \frac{1}{\sqrt{2\pi^n}} \exp\left(-\frac{1}{2} \|\vec{x}\|^2\right) \quad \text{and} \quad \ln(p(\vec{x} | \vec{0}, \vec{I})) = -\frac{n \ln(2\pi)}{2} - \frac{1}{2} \|\vec{x}\|^2.$$

**3-35 Observation :** To determine the probability that a point  $\vec{x} \in \mathbb{R}^n$  satisfies  $\|\vec{x}\| \leq r$  one needs the surface of the unit sphere  $S_n = \{\vec{x} \in \mathbb{R}^n | \|\vec{x}\| = 1\} \subset \mathbb{R}^n$ , given by  $S_{n-1} = \frac{2\pi^{n/2}}{\Gamma(\frac{n}{2})}$ , e.g.  $S_2 = 2\pi$ ,  $S_3 = \frac{2\pi\sqrt{\pi}}{\Gamma(\frac{3}{2})} = 4\pi$ ,  $S_4 = \frac{2\pi^2}{\Gamma(2)} = \pi^2$ . To determine the confidence region for an  $n$ -dimensional case with confidence level  $100(1 - \alpha)\%$  solve the equation

$$\frac{1}{(2\pi)^{n/2}} \int_0^{r_{max}} e^{-\frac{1}{2}r^2} \frac{2\pi^{n/2}}{\Gamma(\frac{n}{2})} r^{n-1} dr = \frac{2^{1-n/2}}{\Gamma(\frac{n}{2})} \int_0^{r_{max}} e^{-\frac{1}{2}r^2} r^{n-1} dr = 1 - \alpha \quad (3.6)$$

for the upper limit of integration  $r_{max}$ . The value can also be computed using the chi-squared distribution by  $r_{max}^2 = \text{chi2inv}(1 - \alpha, n)$ . Then the domain of confidence is determined by  $r = \|\vec{x}\| \leq r_{max}$ . For  $n = 2$  this leads to

$$\begin{aligned} 1 - \alpha &= \frac{1}{\Gamma(1)} \int_0^{r_{max}} e^{-\frac{1}{2}r^2} r dr = -e^{-\frac{1}{2}r^2} \Big|_0^{r_{max}} = 1 - e^{-\frac{1}{2}r_{max}^2} \\ \ln(\alpha) &= -\frac{1}{2}r_{max}^2 \implies r_{max} = \sqrt{-2 \ln(\alpha)} \end{aligned} \quad (3.7)$$

and the domain of confidence  $\frac{1}{2}r^2 \leq -\ln(\alpha) = \ln \frac{1}{\alpha}$ .

In Table 3.4 find results for dimensions  $n$  from 1 to 10 for three values of  $\alpha$ . Listed are the radius  $r_{max}$  up to which the integral has to be performed and the values of the PDF at this radius. This table contains useful information:

- For the 1-d Gauss distribution include all values within 2 (or 1.96) standard deviations from the mean to cover 95% of the events.

- For the 4-d Gauss distribution include all values within 3 (or 3.08) standard deviations from the mean to cover 95% of the events.
- For the 8-d Gauss distribution include all values within 4 (or 3.94) standard deviations from the mean to cover 95% of the events.

This is information on the size of the region of confidence for different dimensions  $n$ .

n	$\alpha = 0.3173$		$\alpha = 0.05$		$\alpha = 0.01$	
	$r_{max}$	pdf( $r_{max}$ )	$r_{max}$	pdf( $r_{max}$ )	$r_{max}$	pdf( $r_{max}$ )
1	1	0.483941	1.95996	0.116890	2.57583	0.0289195
2	1.51517	0.480780	2.44775	0.122387	3.03485	0.0303485
3	1.87796	0.482494	2.79548	0.125287	3.36821	0.0311339
4	2.17244	0.484161	3.08022	0.127198	3.64372	0.0316669
5	2.42644	0.485534	3.32724	0.128595	3.88411	0.0320657
6	2.65300	0.486660	3.54846	0.129683	4.10023	0.0323814
7	2.85941	0.487598	3.75062	0.130565	4.29829	0.0326409
8	3.05023	0.488393	3.93793	0.131301	4.48221	0.0328600
9	3.22852	0.489078	4.11327	0.131928	4.65467	0.0330487
10	3.39647	0.489675	4.27867	0.132473	4.81760	0.0332138

Table 3.4: Standard Gaussian PDF in  $n$  dimensions

◇

### Linear and affine transformations and the resulting Gaussian PDF

Let  $\mathbf{A} \in \mathbb{M}^{n \times n}$  be an invertible matrix and examine the effect of the transformation  $\vec{y} = \mathbf{A} \vec{x}$ . Search for the PDF for  $\vec{y}$ . Using a substitution for the multidimensional integral leads to

$$\begin{aligned} P(\vec{y} \in \mathbf{A}(\Omega)) &= P(\vec{x} \in \Omega) \\ P(\vec{y} \in \mathbf{A}(\Omega)) &= \int_{\mathbf{A}(\Omega)} p_y(\vec{y}) dV_y = |\det(\mathbf{A})| \int_{\Omega} p_y(\mathbf{A}\vec{x}) dV_x \\ P(\vec{x} \in \Omega) &= \int_{\Omega} p_x(\vec{x}) dV_x . \end{aligned}$$

Since the above has to be correct for any domain  $\Omega \subset \mathbb{R}^n$  conclude

$$p_x(\vec{x}) = |\det(\mathbf{A})| p_y(\mathbf{A}\vec{x}) \quad \text{or} \quad p_y(\vec{y}) = \frac{1}{|\det(\mathbf{A})|} p_x(\mathbf{A}^{-1}\vec{y}) . \quad (3.8)$$

**3–36 Example :** Consider the example

$$\mathbf{A} = \text{diag}(\vec{\sigma}) = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}$$

with  $\det(\mathbf{A}) = \prod_{i=1}^n \sigma_i$  and

$$\|\vec{x}\|^2 = \|\mathbf{A}^{-1}\vec{y}\|^2 = \sum_{i=1}^n \frac{y_i^2}{\sigma_i^2}.$$

Use (3.8) to arrive at

$$p(\vec{y}) = \frac{1}{\sqrt{2\pi^n} \prod_{i=1}^n \sigma_i} \exp\left(-\frac{1}{2} \sum_{i=1}^n \frac{y_i^2}{\sigma_i^2}\right),$$

i.e. the recovered Gaussian distribution with different variances.  $\diamond$

With  $p_x(\vec{x}) = (2\pi)^{-n/2} \exp(-\frac{1}{2} \|\vec{x}\|^2)$  equation (3.8) can be written in the form<sup>7</sup>

$$p(\vec{y} | \mathbf{A}) = \frac{1}{\sqrt{2\pi^n} |\det(\mathbf{A})|} \exp\left(-\frac{1}{2} \vec{y}^T (\mathbf{A}\mathbf{A}^T)^{-1} \vec{y}\right).$$

and the above result can be generalized to affine transformations

$$\vec{x} \mapsto \mathbf{T}(\vec{x}) = \vec{y} = \vec{y}_0 + \mathbf{A}\vec{x}$$

with the resulting PDF

$$p(\vec{y} | \mathbf{A}, \vec{y}_0) = \frac{1}{(2\pi)^{n/2} |\det(\mathbf{A})|} \exp\left(-\frac{1}{2} (\vec{y} - \vec{y}_0)^T (\mathbf{A}\mathbf{A}^T)^{-1} (\vec{y} - \vec{y}_0)\right). \quad (3.9)$$

### Maximum likelihood estimators of $\vec{y}_0$ and $\mathbf{A}$

Assume that  $N$  independent data points  $\vec{y}_j \in \mathbb{R}^n$  are given and it is known that the distribution is an affine transformation of the standard normal distribution. You want to recover information on the affine transformation  $\mathbf{T}\vec{x} = \vec{y}_0 + \mathbf{A}\vec{x}$ . Using  $|\det(\mathbf{A})| = \sqrt{\det(\mathbf{A}\mathbf{A}^T)}$  the PDF for the combined event is

$$\begin{aligned} p(\vec{y}_1, \dots, \vec{y}_N | \mathbf{A}, \vec{y}_0) &= \prod_{j=1}^N p(\vec{y}_j | \mathbf{A}, \vec{y}_0) \\ &= \prod_{j=1}^N \frac{1}{(2\pi)^{n/2} |\det(\mathbf{A})|} \exp\left(-\frac{1}{2} (\vec{y}_j - \vec{y}_0)^T (\mathbf{A}\mathbf{A}^T)^{-1} (\vec{y}_j - \vec{y}_0)\right) \\ f(\mathbf{A}, \vec{y}_0) &:= \ln(p(\vec{y}_1, \dots, \vec{y}_N | \mathbf{A}, \vec{y}_0)) \\ &= -\frac{N}{2} n \ln(2\pi) - \frac{N}{2} \ln(\det(\mathbf{A}\mathbf{A}^T)) - \sum_{j=1}^N \frac{1}{2} (\vec{y}_j - \vec{y}_0)^T (\mathbf{A}\mathbf{A}^T)^{-1} (\vec{y}_j - \vec{y}_0). \end{aligned}$$

The above is the answer to: given  $\mathbf{A}$  and  $\vec{y}_0$ , what is the PDF of the measurement points.

Now turn the approach around and seek the answer to: given the measured data points  $\vec{y}_j$ , recover the affine transformation with  $\mathbf{A}$  and the shift  $\vec{y}_0$ . The **maximum likelihood** approach seeks the values of  $\mathbf{A}$  and  $\vec{y}_0$  to render the function  $f(\mathbf{A}, \vec{y}_0)$  maximal. Hoping for an unimodal function search for zeros of the derivatives of  $f$  with respect to  $\vec{y}_0$  and the components of  $\mathbf{A}$ .

Ex. 3.11 is the 1D situation

<sup>7</sup>

$\|\mathbf{A}^{-1}\vec{y}\|^2 = \langle \mathbf{A}^{-1}\vec{y}, \mathbf{A}^{-1}\vec{y} \rangle = \langle \vec{y}, \mathbf{A}^{-T}\mathbf{A}^{-1}\vec{y} \rangle = \langle \vec{y}, (\mathbf{A}\mathbf{A}^T)^{-1}\vec{y} \rangle$

- Compute the gradient of  $f$  with respect to  $\vec{y}_0$  and solve for  $\vec{y}_0$ .

$$\begin{aligned}\vec{0} &= \frac{\partial f}{\partial \vec{y}_0} = \sum_{j=1}^N (\mathbf{A} \mathbf{A}^T)^{-1} (\vec{y}_j - \vec{y}_0) = (\mathbf{A} \mathbf{A}^T)^{-1} \sum_{j=1}^N (\vec{y}_j - \vec{y}_0) \in \mathbb{R}^n \\ \vec{0} &= \sum_{j=1}^N (\vec{y}_j - \vec{y}_0) = -N \vec{y}_0 + \sum_{j=1}^N \vec{y}_j \\ \vec{y}_0 &= \frac{1}{N} \sum_{j=1}^N \vec{y}_j\end{aligned}$$

Thus the average value of the data points is the most likely estimator of the offset.

- As a consequence of the above you can first determine the average values and then subtract them from the data. This simplifies the function and now examine maxima of the modified function

$$\hat{f}(\boldsymbol{\Gamma}) := +N \ln(\det(\boldsymbol{\Gamma})) - \sum_{j=1}^N \vec{y}_j^T \boldsymbol{\Gamma} \vec{y}_j.$$

- To examine derivatives with respect to  $\Gamma_{rs}$  the adjugate matrix<sup>8</sup> can be used with the computational rules no proof in class

$$\boldsymbol{\Gamma}^{-1} = \frac{1}{\det(\boldsymbol{\Gamma})} \text{adj}(\boldsymbol{\Gamma}) \in \mathbb{M}^{n \times n} \quad \text{and} \quad \frac{\partial}{\partial \Gamma_{rs}} \det(\boldsymbol{\Gamma}) = (-1)^{r+s} \text{adj}(\boldsymbol{\Gamma})_{sr}.$$

This implies

$$\nabla \det(\boldsymbol{\Gamma}) = \text{adj}(\boldsymbol{\Gamma})^T = \det(\boldsymbol{\Gamma}) \cdot \boldsymbol{\Gamma}^{-T}.$$

- For the derivative of  $\hat{f}$  with respect to  $\Gamma_{rs}$  obtain

$$0 = +N \frac{\text{adj}(\boldsymbol{\Gamma})_{sr}}{\det(\boldsymbol{\Gamma})} - \sum_{j=1}^N y_{r,j} \cdot y_{s,j}.$$

Take each data point  $\vec{y}_i$  (the average  $\vec{y}_0$  already subtracted) as a column vector and stack all measurements in one matrix

$$\mathbf{Y} = \begin{bmatrix} \vec{y}_1 & \vec{y}_2 & \cdots & \vec{y}_N \end{bmatrix} \in \mathbb{M}^{n \times N}$$

and compute the **covariance matrix**

$$\mathbf{M} = \frac{1}{N} \mathbf{Y} \mathbf{Y}^T \in \mathbb{M}^{n \times n}.$$

Then the sums

$$\frac{1}{N} \sum_{j=1}^N y_{r,j} \cdot y_{s,j} = M_{r,s} \quad \text{for } 1 \leq r, s \leq n$$

appear as components of the covariance matrix. The above condition for the partial derivatives of  $\hat{f}$  to vanish is transformed to

$$\frac{\text{adj}(\boldsymbol{\Gamma})}{\det(\boldsymbol{\Gamma})} = \frac{1}{N} \mathbf{Y} \mathbf{Y}^T.$$

Thus for the matrix  $\mathbf{A}$  find the necessary condition

$$\boldsymbol{\Gamma}^{-1} = \mathbf{A} \mathbf{A}^T = \frac{1}{N} \mathbf{Y} \mathbf{Y}^T \in \mathbb{M}^{n \times n}.$$

One can not recover the components of  $\mathbf{A}$ , but only the product  $\boldsymbol{\Gamma}^{-1} = \mathbf{A} \mathbf{A}^T$ . If you insist on one of the possible constructions of a matrix  $\mathbf{A}$  you may use the Cholesky factorization  $\boldsymbol{\Gamma}^{-1} = \mathbf{R}^T \mathbf{R}$ , i.e.  $\mathbf{A} = \mathbf{R}^T$  is one of the possible reconstructions.

present  
only final  
result in  
class

<sup>8</sup>First compute the cofactor  $C_{r,s}$ . Start with the original matrix, eliminate row  $r$  and column  $s$ , compute the determinant, multiply by  $(-1)^{r+s}$ . The transpose of the matrix of cofactors is equal to  $\text{adj}(\boldsymbol{\Gamma})$ .

### Interpretation of level curves, eigenvalues and eigenvectors, PCA

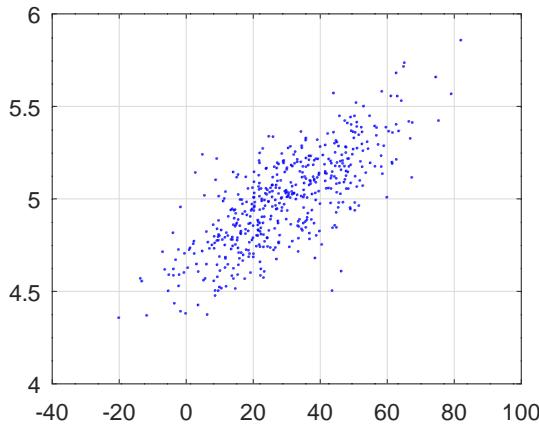
Based on equation (3.9) examine the PDF for the values of  $\vec{y}$  given by

$$p(\vec{y}) = \frac{1}{(2\pi)^{n/2}} \sqrt{\det(\mathbf{\Gamma})} \exp\left(-\frac{1}{2}(\vec{y} - \vec{y}_0)^T \mathbf{\Gamma} (\vec{y} - \vec{y}_0)\right).$$

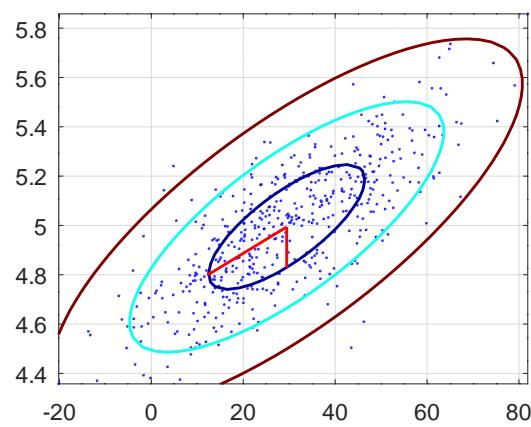
The level curves of the exponent yield useful information. It simplifies notation to work with the quadratic function

$$h(\vec{y}) = \frac{1}{2}(\vec{y} - \vec{y}_0)^T \mathbf{\Gamma} (\vec{y} - \vec{y}_0).$$

The minimal value of  $h$  is 0, attained at  $\vec{y} = \vec{y}_0$ . The level curves can be generated by the code presented in Example 3–25, using the eigenvectors and eigenvalues of  $\mathbf{\Gamma} = (\mathbf{A}\mathbf{A}^T)^{-1}$ , i.e. the inverse of the covariance matrix.



(a) raw data



(b) raw data and level curves

Figure 3.19: Raw data and level curves for the likelihood function. Inside the level curves find 99%, 87% or 39% of the data points.

This is used to determine the domains of confidence with a level of confidence  $(1 - \alpha)$ , i.e. a level of significance  $\alpha$ . For  $n = 2$  use (3.6) and (3.7) to solve

$$\int_0^{r_{max}} e^{-\frac{1}{2}r^2} r dr = 1 - e^{-\frac{1}{2}r_{max}^2} = 1 - \alpha.$$

Then  $(1 - \alpha) 100\%$  of the points are in the domain  $\|\mathbf{A}^{-1}\vec{y}\| = \|\vec{x}\| \leq r_{max}$ . Find the results for  $r_{max} = 1, 2, 3$  with the corresponding confidence levels in Figure 3.19. Assuming  $\vec{y}_0 = \vec{0}$  this domain can be determined using the covariance matrix  $\mathbf{M}$  by

$$r_{max}^2 \geq \|\mathbf{A}^{-1}\vec{y}\|^2 = \langle \mathbf{A}^{-1}\vec{y}, \mathbf{A}^{-1}\vec{y} \rangle = \langle \vec{y}, \mathbf{A}^{-T}\mathbf{A}^{-1}\vec{y} \rangle = \langle \vec{y}, (\mathbf{A}\mathbf{A}^T)^{-1}\vec{y} \rangle = \langle \vec{y}, \mathbf{M}^{-1}\vec{y} \rangle.$$

To generate graphs use the eigenvalues and eigenvectors of the covariance matrix, see Example 3–25. The result has to be compared to the usual individual intervals of confidence for  $y_1$  and  $y_2$ . Since two conditions have to be satisfied use a level of confidence of  $\sqrt{1 - \alpha}$  and read out the standard deviations as square roots of the diagonal entries in the covariance matrix. Then use the normal distribution by  $c = \text{norminv}(1 - (1 - \sqrt{1 - \alpha})) \approx \text{norminv}(1 - \alpha/2)$  to construct the intervals of confidence for  $y_i$  and  $y_2$

$$\text{mean}(y_i) - c\sigma_i \leq y_i \leq \text{mean}(y_i) + c\sigma_i.$$

The result for  $\alpha = 0.05$  is shown in Figure 3.20(a).

- 95% of the data points are inside the ellipse.
- 95% of the data points are inside the rectangle.

The ellipse provides better information on the location of the data points, since the correlation between  $y_1$  and  $y_2$  is taken into account.

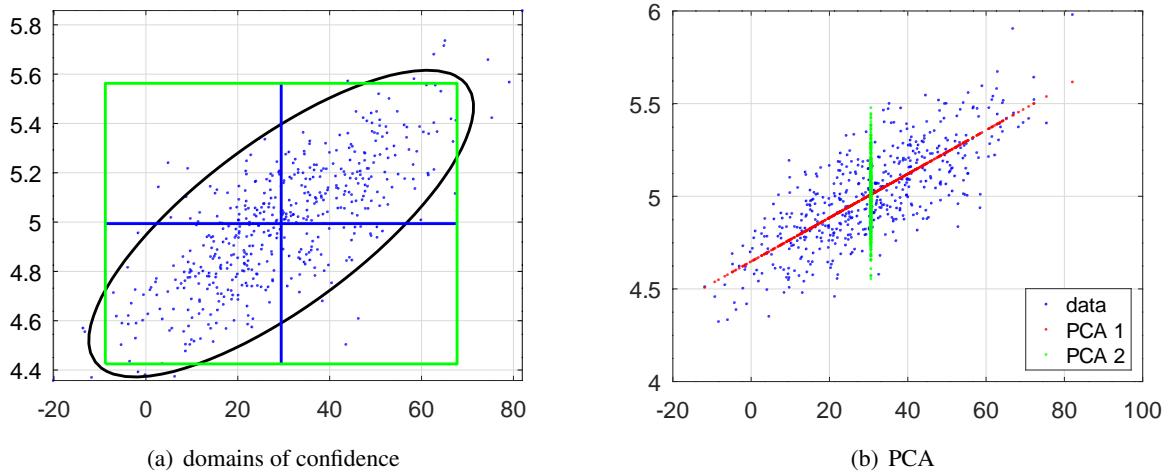


Figure 3.20: Raw data and the domain of confidence at level of significance  $\alpha = 0.05$  and the PCA

### From covariance to PCA

The positive eigenvalues  $\lambda_i$  of  $\Gamma$  and the normalized eigenvectors  $\vec{e}_i$  give a good description of the function  $h$ . Examine

$$h(\vec{y}_0 + t \vec{e}_i) = \frac{1}{2} t^2 \vec{e}_i^T \Gamma \vec{e}_i = \frac{1}{2} t^2 \lambda_i .$$

- If  $\lambda_i$  is large the function  $h$  will grow quickly in that direction and the level curves will be close together.
- If  $\lambda_i$  is small the function  $h$  will grow slowly in that direction and the level curves will be far apart.

To visualize the above one may rescale the eigenvectors  $\vec{e}_i$  to length  $\sqrt{\lambda_i}$  and then draw them starting at the center point  $\vec{y}_0$ . Find an example in Figure 3.19(b).

Since  $\mathbf{M} = \Gamma^{-1} = \mathbf{A}\mathbf{A}^T = \frac{1}{N} \mathbf{Y}\mathbf{Y}^T$  the eigenvalues  $\lambda_i$  of  $\Gamma$  lead to the eigenvalues  $\mu_i = \frac{1}{\lambda_i}$  of  $\mathbf{A}\mathbf{A}^T$ . This leads to the **Principal Component Analysis**, or short **PCA**. The main application of PCA is dimension reduction, i.e. instead of many dimensions reduce the data such that only the main (principal) components have to be examined. Such a principal component is a linear combination of the available variables.

If  $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \dots$  then  $\mu_1 \geq \mu_2 \geq \mu_3 \geq \dots$ , i.e. large eigenvalues indicated a slow growing of the function  $h$  and thus the data is spread out in the direction of the corresponding eigenvector. If  $\vec{e}_1$  is normalized then compute the principle component of the data, i.e. the projection of the data in the direction of  $\vec{e}_1$ , given by  $\langle \vec{e}_1, \vec{y} - \vec{y}_0 \rangle$ . The data can be displayed at  $\langle \vec{e}_1, \vec{y} - \vec{y}_0 \rangle \vec{e}_1 \in \mathbb{R}^2$ . The second principal component is given by  $\langle \vec{e}_2, \vec{y} - \vec{y}_0 \rangle$ . Find the graphical representation in Figure 3.20(b).

### Elementary code in Octave/MATLAB

Based on the above idea it is easy to write code in Octave/MATLAB to determine the PCA and generate figures comparable to Figure 3.20(b). The steps in the code below are:

- Use the mean values  $\vec{m}$  of the data, mainly for graphical purposes.
- With the command `cov()` the mean values  $\vec{m}$  are subtracted and the covariance computed.

$$\begin{aligned} m_i &= \frac{1}{n} \sum_{k=1}^n \text{data}_{k,i} \\ \text{cov}(\text{data})_{i,j} &= \sum_{k=1}^n (\text{data}_{k,i} - m_i)(\text{data}_{k,j} - m_j) \end{aligned}$$

- Determine the eigenvalues and eigenvectors using the command `eig()`. With the option 'descend' the eigenvalues are in decreasing order.
- Compute the length of the projection of the data into the directions of the eigenvectors, leading to the scores.
- Multiply the scores with the eigenvectors to obtain the projection of the data onto the eigen directions.
- Display the data and the first two principal components.

```

data = ..... %% generate your data as a n x 2 matrix
m = mean(data); %% mean values
[PCAAvec,PCAval] = eig(cov(data)); %% eigenvectors and values of the covariance matrix
[Pval, idx] = sort(diag(PCAval),1,'descend');
Pscore1 = (data-m)*PCAAvec(:,idx(1));
Pscore2 = (data-m)*PCAAvec(:,idx(2));
PCAdatal = Pscore1*PCAAvec(:,idx(1))';
PCAdata2 = Pscore2*PCAAvec(:,idx(2))';

figure(4)
plot(data(:,1),data(:,2),'b',m(1) + PCAdatal(:,1), m(2) + PCAdatal(:,2),'r',...
      m(1) + PCAdata2(:,1), m(2) + PCAdata2(:,2),'g')
legend('data','PCA 1','PCA 2','location','southeast')
```

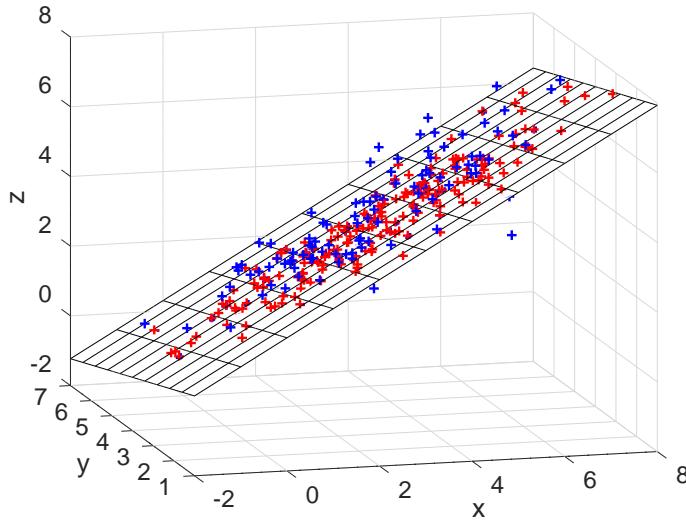
**3-37 Example :** PCA is not restricted to problems in 2 dimension, it is only the visualization that is a more challenging. In Figure 3.21 find in blue a cloud of data points in  $\mathbb{R}^3$ . The data points seem to be rather close to a plane. A PCA for the first two components was performed and the each data point was projected onto the plane spanned by the eigenvectors belonging to these two components. This leads to the red points in the figure. Observe that this approach is different from using linear regression to determine the best fitting plane.

- With a linear regression using  $z = c_0 + c_1 x + c_2 y$  the sum of the squared vertical distances is minimized.
- With PCA the sum of the squared orthogonal distances to the plane is minimized. Consult the next section, where PCA is regarded as an optimization problem.



### PCA as an optimization problem

Another view on PCA is to examine it as solution of an optimization problem. For the data matrix  $\mathbf{Y} = [\vec{y}_1, \vec{y}_2, \dots, \vec{y}_N] \in \mathbb{M}^{n \times N}$  find the direction vector  $\vec{e} \in \mathbb{R}^n$  such that the squared sum of the projections in

Figure 3.21: PCA demo for data in  $\mathbb{R}^3$ 

this direction  $\vec{e}$  is maximal, subject to the constraint  $\|\vec{e}\| = 1$ . Since the projection of a vector  $\vec{y}$  onto the direction (given by  $\vec{e}$ ) equals  $\langle \vec{e}, \vec{y} \rangle$  the function  $f(\vec{e})$  to be optimized is given by

$$f(\vec{e}) = \frac{1}{2} \sum_{j=1}^N (\langle \vec{y}_j, \vec{e} \rangle)^2 = \frac{1}{2} \sum_{j=1}^N \left( \sum_{i=1}^n y_{i,j} e_i \right)^2.$$

To find the extrema differentiate with respect to the components  $e_k$  of the normalized vector  $\vec{e}$ .

$$\begin{aligned} \frac{\partial}{\partial e_k} f(\vec{e}) &= \sum_{j=1}^N \left( \sum_{i=1}^n y_{i,j} e_i \right) y_{k,j} = \sum_{i=1}^n \left( \sum_{j=1}^N y_{i,j} y_{k,j} \right) e_i = \sum_{i=1}^n (\mathbf{Y} \mathbf{Y}^T)_{k,i} e_i \\ \nabla f(\vec{e}) &= \mathbf{Y} \mathbf{Y}^T \vec{e} \end{aligned}$$

Thus the Lagrange multiplier theorem for constrained optimization with the constraint  $\|\vec{e}\|^2 = 1$  and  $\nabla \|\vec{e}\|^2 = 2 \vec{e}$  lead to

$$\mathbf{Y} \mathbf{Y}^T \vec{e} = \lambda \vec{e},$$

i.e. the direction  $\vec{e}$  has to be an eigenvector of the matrix  $\mathbf{Y} \mathbf{Y}^T \in \mathbb{M}^{n \times n}$ . Use a directional derivative of  $f(\vec{e})$  to conclude

$$\begin{aligned} \frac{d}{dc} f(c \vec{e}) &= \langle \vec{e}, \nabla f(c \vec{e}) \rangle = c \langle \vec{e}, \mathbf{Y}^T \mathbf{Y} \vec{e} \rangle = c \langle \vec{e}, \lambda \vec{e} \rangle = c \lambda \\ f(1 \vec{e}) &= f(\vec{0}) + \int_0^1 \frac{d}{dc} f(c \vec{e}) dc = 0 + \int_0^1 c \lambda dc = \frac{1}{2} \lambda \end{aligned}$$

and thus  $f(\vec{e}) = \frac{1}{2} \lambda$ . The largest value of  $f(\vec{e})$  is attained for the eigenvector belonging to the largest eigenvalue  $\lambda_{max}$ .

This is illustrated by Figure 3.22: the direction of the first principal component is such that the sum of the squared distances of the red points from the origin is largest, while the sum of the squared lengths of the green lines is smallest. This is based on the observation that the square of the principal component and the squared length of corresponding green line add up to  $\|\vec{y}_j\|^2$  and the sum  $\sum_{j=1}^N \|\vec{y}_j\|^2$  does not depend on the direction  $\vec{e}$ .

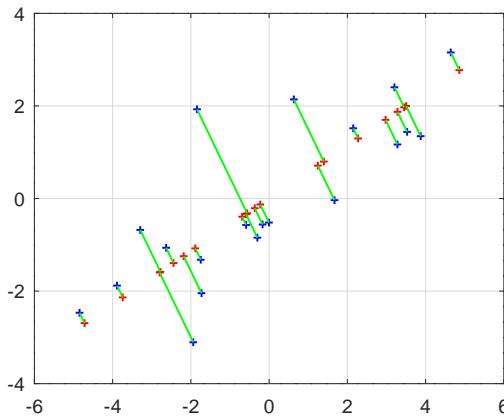


Figure 3.22: Projection of data in the direction of the first principal component. Original data in blue and the first principal component in red. The original data points and their projections are connected by green lines.

```

N = 20; data = randn(N,2); data = data - mean(data);
A = [3 1;1 1.5]; data = data*A;

[coeff,score,latent] = princomp(data);
dataPCA = score(:,1)*coeff(1,:);

figure(1); plot(data(:,1),data(:,2),'+b',dataPCA(:,1),dataPCA(:,2),'+r')
hold on
for ii = 1:length(data)
    plot([data(ii,1),dataPCA(ii,1)],[data(ii,2),dataPCA(ii,2)],'g')
end%for
hold off

```

### PCA computed by SVD

For  $\mathbf{Y} \in \mathbb{M}^{n \times N}$  with  $n < N$  compute the SVD (singular value decomposition) with unitary matrices  $\mathbf{U} \in \mathbb{M}^{n \times n}$ ,  $\mathbf{V} \in \mathbb{M}^{N \times N}$  and the diagonal matrix  $\mathbf{D} \in \mathbb{M}^{n \times N}$ .

$$\begin{aligned}
\mathbf{Y} &= \mathbf{U} \mathbf{D} \mathbf{V}^T \\
\mathbf{Y} \mathbf{Y}^T &= \mathbf{U} \mathbf{D} \mathbf{V}^T \mathbf{V} \mathbf{D}^T \mathbf{U}^T = \mathbf{U} \mathbf{D} \mathbf{D}^T \mathbf{U}^T \\
\mathbf{D} \mathbf{D}^T &= \begin{bmatrix} \sigma_1^2 & & \\ & \sigma_2^2 & \\ & & \ddots \\ & & & \sigma_n^2 \end{bmatrix} \in \mathbb{M}^{n \times n}
\end{aligned}$$

Thus the SVD of the matrix  $\mathbf{Y} \in \mathbb{M}^{n \times N}$  leads to the eigenvalues  $\sigma_i^2$  of the matrix  $\mathbf{Y} \mathbf{Y}^T$  with the  $n$  eigenvectors in the columns of  $\mathbf{U} \in \mathbb{M}^{n \times n}$ . This is the information required for the PCA. Examine the source code of `princomp.m` in Octave to realize that SVD is used to determine the PCA. Observe that the matrix  $\mathbf{Y} \mathbf{Y}^T$  does not have to be computed. This might make a difference for very large data sets, e.g. for machine learning.

### The PCA commands in Octave and MATLAB

The builtin commands `princomp()` and/or `pca()` do not use eigenvalues and eigenvectors, but a SVD to determine the PCA. With *Octave* use the command `princomp()` from the statistics package.

```
help princomp
-->
-- Function File: [COEFF] = princomp(X)
-- Function File: [COEFF,SCORE] = princomp(X)
-- Function File: [COEFF,SCORE,LATENT] = princomp(X)
-- Function File: [COEFF,SCORE,LATENT,TSQUARE] = princomp(X)
-- Function File: [...] = princomp(X,'econ')
    Performs a principal component analysis on a NxP data matrix X

* COEFF : returns the principal component coefficients
* SCORE : returns the principal component scores, the
  representation of X in the principal component space
* LATENT : returns the principal component variances, i.e., the
  eigenvalues of the covariance matrix X.
* TSQUARE : returns Hotelling's T-squared Statistic for each
  observation in X
* [...] = princomp(X,'econ') returns only the elements of latent
  that are not necessarily zero, and the corresponding columns of
  COEFF and SCORE, that is, when n <= p, only the first n-1.
  This can be significantly faster when p is much larger than n.
  In this case the svd will be applied on the transpose of the data matrix X
```

#### References

1. Jolliffe, I. T., Principal Component Analysis, 2nd Edition, Springer, 2002

With this command the PCA is easily generated by

```
[coeff,score,latent] = princomp(data);
```

and the visualization is identical to the above code.

The statistics toolbox<sup>9</sup> (i.e. \$\$\$) in MATLAB contains the command `pca()`, which generates the identical result by `[coeff,score,latent] = pca(data);`.

MATLAB and Octave provide the command `pcacov()` to determine the PCA for a given covariance matrix.

### From covariance to correlation

The above eigenvector  $\vec{e}_i$  of the symmetric matrix  $\Gamma$  are orthogonal. You will usually not find a right angle between them when graphing (e.g. Figure 3.19), since the scales in the directions in  $\mathbb{R}^n$  are usually not the same. This can be fixed. Change the scale in each coordinate direction in  $\mathbb{R}^n$  such that the new covariance matrix  $\bar{\mathbf{M}}$  has only numbers 1 on the diagonal. To achieve this divide rows and columns by the square root of the entries on the diagonal, i.e.

$$\boldsymbol{\Gamma}^{-1} = \mathbf{M} = \frac{1}{N} \mathbf{Y} \mathbf{Y}^T \in \mathbb{M}^{n \times n}$$

<sup>9</sup>On the Mathworks web site find a code `princomp.m` by B. Jones which works on MATLAB.

$$\mathbf{S}^{-1} = \begin{bmatrix} \sqrt{M_{11}} & & & \\ & \sqrt{M_{22}} & & \\ & & \ddots & \\ & & & \sqrt{M_{nn}} \end{bmatrix}$$

$$\bar{\mathbf{M}} = \mathbf{SMS}$$

The resulting matrix  $\bar{\mathbf{M}}$  is called **correlation matrix**. It has numbers 1 on the diagonal and all other entries are smaller than 1. This has the effect that a unit step in any of the **coordinate axes** will lead to the same increase for the function value.

Since the matrix is rescaled the scales of the coordinates have to be adapted, leading to

$$\begin{aligned} h(\vec{y}) &= \frac{1}{2} \langle (\vec{y} - \vec{y}_0), \mathbf{T}(\vec{y} - \vec{y}_0) \rangle = \frac{1}{2} \langle (\vec{y} - \vec{y}_0), \mathbf{M}^{-1}(\vec{y} - \vec{y}_0) \rangle \\ &= \frac{1}{2} \langle (\vec{y} - \vec{y}_0), \mathbf{S}(\mathbf{S}^{-1}\mathbf{M}^{-1}\mathbf{S}^{-1})\mathbf{S}(\vec{y} - \vec{y}_0) \rangle \\ &= \frac{1}{2} \langle (\mathbf{S}(\vec{y} - \vec{y}_0)), \bar{\mathbf{M}}^{-1}(\mathbf{S}(\vec{y} - \vec{y}_0)) \rangle = \bar{h}(\mathbf{S}(\vec{y} - \vec{y}_0)) \end{aligned}$$

Now rerun the eigenvalue/eigenvector algorithm for the correlation matrix  $\bar{\mathbf{M}}$  and display the result in a rescaled graph, using identical scales in all coordinate directions. The eigenvectors appear orthogonal, find an example in Figure 3.23.

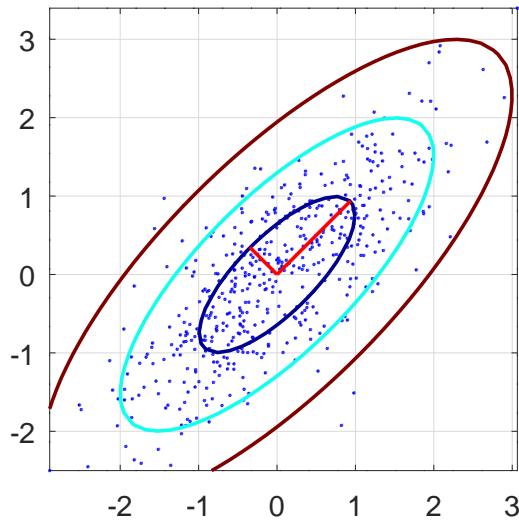


Figure 3.23: Scaled data and level curves for the likelihood function

### 3.3 Numerical Integration

In this section a few methods to evaluate definite integrals numerically are presented.

- In subsection 3.3.1 the trapezoidal rule and Simpson's rule are used to evaluate integrals for functions given by data points.
- In subsection 3.3.2 possible algorithms to evaluate integrals of functions given by an expression (formula) are examined. The brilliant idea of Gauss is presented and the method of adaptive integration is introduced.
- In subsection 3.3.3 some commands in Octave/MATLAB are used to evaluate integrals.
- In subsection 3.3.4 some ideas and Octave/MATLAB codes on how to integrate over domains in the plane  $\mathbb{R}^2$  are introduced.

#### 3.3.1 Integration of a Function Given by Data Points

In this subsection assume that a function is given by  $n + 1$  data points  $a = x_0 < x_1 < x_2 < \dots < x_n = b$  and the corresponding values  $y_i = f(x_i)$  of the function. A simple situation is shown in Figure 3.24. The goal is to find a good approximation of the definite integral  $\int_a^b f(x) dx$ .

The idea of a trapezoidal integration is to replace the function  $f(x)$  by a piecewise linear interpolation. Each segment is a trapez, whose area is easy to determine by

$$\int_{x_{i-1}}^{x_i} f(x) dx \approx \text{area of trapez} = \text{width} \cdot \text{average height} = (x_i - x_{i-1}) \cdot \frac{f(x_{i-1}) + f(x_i)}{2}.$$

For Figure 3.24 the resulting approximation of the integral is

$$n = 5 \quad h = \frac{b - a}{5} \quad x_i = a + ih \quad \text{for } i = 0, \dots, 5$$

and

$$\begin{aligned} \int_a^b f(x) dx &\approx h \left( \frac{1}{2}f(a) + f(x_1) + f(x_2) + f(x_3) + f(x_4) + \frac{1}{2}f(b) \right) \\ &= \frac{h}{2} \left( f(a) + f(b) + 2 \sum_{i=1}^4 f(x_i) \right). \end{aligned}$$

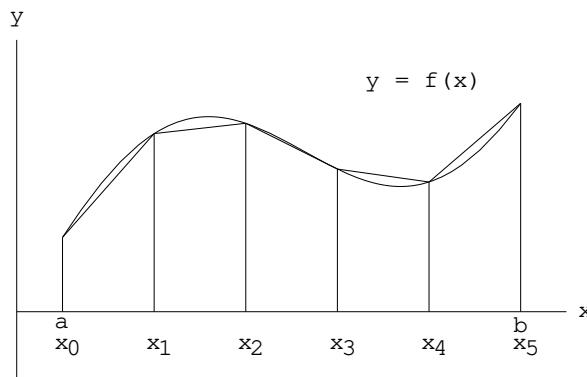


Figure 3.24: Trapezoidal integration

**3–38 Result : Trapezoidal integration**

The integral of a twice differentiable function over the standard interval  $[-h/2, +h/2]$  is approximated by

$$\int_{-h/2}^{+h/2} f(x) dx \approx \frac{h}{2} (f(-h/2) + f(+h/2))$$

and the error is estimated by

$$\text{error}_h \leq \frac{1}{6} \max_{-h/2 \leq \xi \leq +h/2} |f''(\xi)| h^3.$$

For a smooth function defined on the interval  $[a, b]$ , divided up into  $n$  subintervals of length  $h = \frac{b-a}{n}$  and  $x_i = a + i h$  this leads to

$$\int_a^b f(x) dx \approx \frac{h}{2} \left( f(a) + f(b) + 2 \sum_{i=1}^{n-1} f(x_i) \right).$$

with the error

$$\text{error}_{[a,b]} \leq \frac{b-a}{12} \max_{a \leq \xi \leq b} |f''(\xi)| h^2.$$

For polynomials up to degree 1 this leads to exact values for the integral.  $\diamond$

The pattern of coefficients for the trapezoidal rule is given by

$$\frac{b-a}{2n} (1, 2, 2, 2, \dots, 2, 2, 2, 2, 1).$$

The verification of this result is given as an exercise and the arguments to be used are very similar to Ex 3.12 Result 3–40, on the numerical integration using Simpson’s rule.

**3–39 Example : The commands `trapz()` and `cumtrapz()`**

With Octave/MATLAB the trapezoidal rule is implemented in `trapz()`. The command takes 2 arguments, the values of the independent and dependent variables. The spacing does not have to be uniform. With `cumtrapz()` (cumulative trapezoidal rule) the definite integral

$$I(x) = \int_a^x f(t) dt$$

is evaluated, using the trapezoidal rule on each of the sub-intervals. In Figure 3.25 find the results for the elementary integral  $I = \int_0^{\pi/2} \cos(x) dx = 1$ .

- For  $n = 3$  sub-intervals the trapezoidal rule leads to the answer  $I \approx 0.9770$ . Since the graph of  $\cos(x)$  is above the piecewise linear interpolation used by the trapezoidal rule, it is no surprise that the approximate answer is two low.
- Using `cumtrapz()` the integral  $I(x) = \int_0^x \cos(t) dt = \sin(x)$  is approximated, again leading to values that are too small.
- For  $n = 100$  sub-intervals the trapezoidal rule leads to the answer  $I \approx 0.999979$ , i.e. considerably more accurate. Similarly for the indefinite integral  $\int_0^x \cos(t) dt$  where true integral  $\sin(x)$  and the approximate cure are indistinguishable.

The code to generate Figure 3.25 is not too complicated.

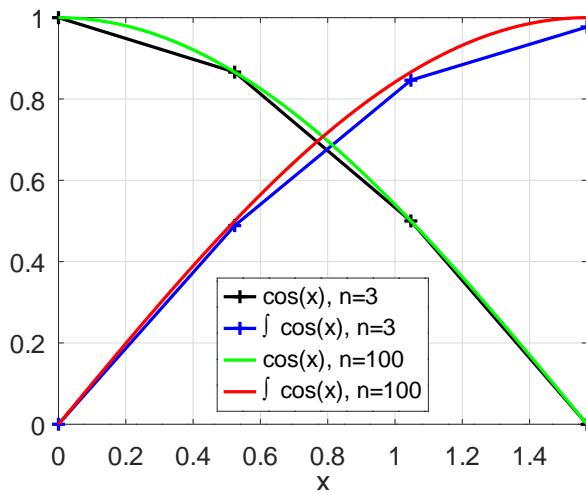


Figure 3.25: The integral  $\int_0^{\pi/2} \cos(x) dx$  by the trapezoidal rule

```

x100 = linspace(0,pi/2,100); y100 = cos(x100);
n = 3; x      = linspace(0,pi/2,n+1); y      = cos(x);
Integral = trapz(x,y)
figure(1); plot(x,y,'+-k',x,cumtrapz(x,y),'+-b',x100,y100,'-g',...
    x100,cumtrapz(x100,y100),'-r')
xlabel('x'); xlim([0,pi/2]);
legend('cos(x), n=3','\int cos(x), n=3',
    'cos(x), n=100','\int cos(x), n=100','location','south')

```

◇

### 3-40 Result : Simpson integration

The integral of a smooth function over the standard interval  $[-h, +h]$  is approximated by

$$\int_{-h}^{+h} f(x) dx \approx \frac{h}{3} (f(-h) + 4f(0) + f(+h))$$

and the error is estimated by

$$\text{error}_{2h} \leq \frac{1}{90} \max_{-h \leq \xi \leq +h} |f^{(4)}(\xi)| h^5$$

For a smooth function defined on the interval  $[a, b]$ , divided up into an even number of subintervals of length  $h = \frac{b-a}{n}$  and  $x_i = a + i h$ , this leads to

$$\int_a^b f(x) dx \approx \frac{h}{3} \left( f(a) - f(b) + 2 \sum_{i=1}^{n/2} (2f(x_{2i-1}) + f(x_{2i})) \right)$$

with the error

$$\text{error}_{[a,b]} \leq \frac{b-a}{180} \max_{a \leq \xi \leq b} |f^{(4)}(\xi)| h^4.$$

For polynomials up to degree 3 this leads to exact values for the integral. ◇

For the interval  $[-h, +h]$  the coefficients for the function values at  $-h, 0$  and  $+h$  are  $\frac{h}{3} (1, 4, 1)$ . For the integration over an interval  $[a, b]$  this leads to the pattern of coefficients

$$\frac{b-a}{3n} (1, 4, 2, 4, 2, 4, 2, 4, 2, \dots, 4, 2, 4, 2, 4, 1).$$

**Proof :** The goal is to find constants  $c_{-1}$ ,  $c_0$  and  $c_{+1}$  such that

$$\int_{-h}^{+h} u(x) dx \approx h (c_{-1} u(-h) + c_0 u(0) + c_{+1} u(+h))$$

with an error as small as possible. To arrive at this goal use a Taylor approximation at  $x = 0$ , i.e.

$$u(x) = u(0) + u'(0)x + \frac{u''(0)}{2}x^2 + \frac{u'''(0)}{6}x^3 + \frac{u^{(4)}(0)}{4!}x^4 + \frac{u^{(5)}(0)}{5!}x^5 + \frac{u^{(6)}(0)}{6!}x^6 + O(x^7). \quad (3.10)$$

For small values of  $x$  contributions of low order are of higher importance. Thus proceed contribution by contribution, starting with the small order terms, i.e. first 1, then  $x$ , then  $x^2$ , ...

1 : Using only the first contribution  $u(x) \approx u(0)$  and the integral of  $u(x) = 1$  leads to

$$\int_{-h}^{+h} 1 dx = 2h = h (c_{-1} 1 + c_0 1 + c_{+1} 1)$$

This integral is exact if the equation  $c_{-1} + c_0 + c_{+1} = 1$  is satisfied.

$x$  : Using only the first two contributions  $u(x) \approx u(0) + u'(0)x$  verify that the second contribution  $u(x) = x$  is integrated exactly, i.e.

$$\int_{-h}^{+h} x dx = 0 = h (c_{-1}(-h), 1 + c_0 0 + c_{+1} h)$$

This integral is exact if the equation  $-c_{-1} + c_{+1} = 0$  is satisfied.

$x^2$  : Using the first three contributions  $u(x) \approx u(0) + u'(0)x + \frac{u''(0)}{2}x^2$  verify that the third contribution  $u(x) = x^2$  is integrated exactly, i.e.

$$\int_{-h}^{+h} x^2 dx = \frac{2}{3}h^3 = h (c_{-1}h^2 + c_0 0 + c_{+1}h^2)$$

This integral is exact if the equation  $c_{-1} + c_{+1} = \frac{2}{3}$  is satisfied.

- This leads to a linear system of equations

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & +1 \\ 1 & 0 & 1 \end{bmatrix} \begin{pmatrix} c_{-1} \\ c_0 \\ c_{+1} \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ \frac{2}{3} \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} c_{-1} \\ c_0 \\ c_{+1} \end{pmatrix} = \begin{pmatrix} \frac{1}{3} \\ \frac{4}{3} \\ \frac{1}{3} \end{pmatrix}$$

These equations have a unique solution  $c_{-1} = c_{+1} = \frac{1}{3}$  and  $c_0 = \frac{4}{3}$  and thus the approximation formula for the integral is

$$\int_{-h}^{+h} u(x) dx = \frac{h}{3} (u(-h) + 4u(0) + u(+h)) + \text{error}_{2h}.$$

- To estimate the size of the error first verify that this sum leads to the exact integral for the function  $u(x) = x^3$ , since  $\int_{-h}^{+h} x^3 dx = 0 = \frac{h}{3}((-h)^3 + 4 \cdot 0 + (+h)^3)$ . The integration for  $x^4$  is not exact, since  $\int_{-h}^{+h} x^4 dx = \frac{2}{5}h^5 \neq \frac{h}{3}((-h)^4 + 4 \cdot 0 + (+h)^4) = \frac{2}{3}h^5$ . The error equals  $\frac{2}{3}h^5 - \frac{2}{5}h^5 = \frac{4}{15}h^5$ . The integration of the higher power  $x^5$  leads to a contribution proportional to  $h^6$ , which is considerably smaller than the above  $\frac{4}{15}h^5$  for  $0 < h \ll 1$ . Thus the error for the integration over  $[-h, +h]$  is estimated by

$$\text{error}_{2h} \approx \frac{4}{15} \frac{u^{(4)}(0)}{4!} h^5 = \frac{1}{90} u^{(4)}(0) h^5.$$

- To obtain the estimate for the integral from  $a$  to  $b$  use  $n = \frac{b-a}{2h}$  of those integrals, leading<sup>10</sup> to

$$\text{error}_{[a,b]} \leq \frac{b-a}{180} \max_{a \leq \xi \leq b} |u^{(4)}(\xi)| h^4.$$

□

The above computations can be organized in tabular form, which is easy to memorize.

$p(x)$	$\int_{-h}^{+h} p(x) dx$	$= h(c_{-1}p(-h) + c_0p(0) + c_1p(+h))$	
1	$2h$	$= h(c_{-1} + c_0 + c_1)$	equation 1
$x$	0	$= h(-c_{-1}h + c_00 + c_1h)$	equation 2
$x^2$	$\frac{2}{3}h^3$	$= h(+c_{-1}h^2 + c_00 + c_1h^2)$	equation 3
$x^3$	$\frac{2}{4}h^4$	$= h(-c_{-1}h^3 + c_00 + c_1h^3)$	OK
$x^4$	$\frac{2}{5}h^5$	$= h(+c_{-1}h^4 + c_00 + c_1h^4) = \frac{2}{3}h^5$	not OK

The Simpson integration over an interval  $[a, b]$  can only be applied if an even number  $n$  of sub-intervals is used. Thus use an odd number of data points  $x_i = a + i \frac{b-a}{n}$  for  $i = 0, 1, 2, \dots, n$ . If an odd number of intervals of equal length  $h$  is used one can use the Simpson 3/8-rule to preserve the order of convergence, i.e. the error remains proportional to  $h^4$ . Use the formula below for the first (or last) three sub-intervals of length  $h$  of the interval over which to integrate.

$$\int_{x_0}^{x_3} f(x) dx \approx \frac{3}{8} h (f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3))$$

**3-41 Example :** Examine the elementary integral  $\int_0^{\pi/2} \sin(x) dx = 1$ . The comparison of the trapezoidal rule, Simpson's formula and the Gauss method should convince you that Gauss is often extremely efficient.

- Use the trapezoidal rule with 2 sub-intervals, thus 3 evaluations of  $\sin(x)$ .

$$\int_0^{\pi/2} \sin(x) dx \approx \frac{\pi/4}{2} \left( \sin(0) + 2 \sin\left(\frac{\pi}{4}\right) + \sin\left(\frac{\pi}{2}\right) \right) = \frac{\pi}{8} \left( 0 + 2 \frac{\sqrt{2}}{2} + 1 \right) \approx 0.9481$$

- Use Simpson's rule with 1 sub-interval, thus 3 evaluations of  $\sin(x)$ .

$$\int_0^{\pi/2} \sin(x) dx \approx \frac{\pi/4}{3} \left( \sin(0) + 4 \sin\left(\frac{\pi}{4}\right) + \sin\left(\frac{\pi}{2}\right) \right) = \frac{\pi}{12} \left( 0 + 4 \frac{\sqrt{2}}{2} + 1 \right) \approx 1.0023$$

Observe that Simpson's approach generates a smaller error than the trapezoidal rule.

- Using the Gauss idea from Result 3-43 in the next section with one sub-interval with three points leads to an even more accurate result.

$$\int_0^{\pi/2} \sin(x) dx \approx \frac{\pi/4}{9} \left( 5 \sin\left(\frac{\pi}{4}(1 - \sqrt{\frac{3}{5}})\right) + 9 \sin\left(\frac{\pi}{4}\right) + 5 \sin\left(\frac{\pi}{4}(1 + \sqrt{\frac{3}{5}})\right) \right) \approx 1.000008$$

---

<sup>10</sup>The proof shown in these notes only leads to an estimate of the error, not a rigorous proof. The simplicity of the argument used justifies the lack of mathematical rigor. The statement is correct though, find a proof in [RalsRabi78, §4].

```

x0 = pi/4; h = pi/4;
xm = x0 - sqrt(3/5)*h; xp = x0 + sqrt(3/5)*h;
Gauss = h/9*(5*sin(xm) + 8*sin(x0) + 5*sin(xp))
difference = Gauss - 1
-->
Gauss = 1.0000081216
difference = 8.1216e-06

```

◊

**3-42 Example : Implementation of Simpson's rule**

Simpson's algorithm is easy to implement in Octave/MATLAB and can compute integrals rather accurately. Find one possible implementation in `simpson.m` below and determine approximative values of  $\int_0^{\pi/2} \cos(x) dx = 1$ , using 10 and 100 sub-intervals.

```

format long
n = 10 ; x = linspace(0,pi/2,n+1); Integral10 = simpson(cos(x),0,pi/2,n)
n = 100 ; x = linspace(0,pi/2,n+1); Integral100 = simpson(cos(x),0,pi/2,n)
-->
Integral10 = 1.000003392220900
Integral100 = 1.000000000338236

```

◊

**simpson.m**

```

function res = simpson(f,a,b,n)

%% simpson(f,a,b,n) compute the integral of the function f
%% on the interval [a,b] with using Simpsons rule
%% use n subintervals of equal length , n has to be even, otherwise n+1 is used
%% f is either a function handle, e.g. @sin or a vector of values

if isa(f,'function_handle')
    n = round(n/2+0.1)*2; %% assure even number of subintervals
    h = (b-a)/n;
    x = linspace(a,b,n+1);
    f_x = x;
    for k = 0:n
        f_x(k+1) = feval(f,x(k+1));
    end%for
else
    n = length(f);
    if (floor(n/2)-n/2==0)
        error('simpson: odd number of data points required');
    else
        n = n-1;
        h = (b-a)/n;
        f_x = f(:)';
    end%if
end%if

w = 2*[ones(1,n/2); 2*ones(1,n/2)]; w = w(:); % construct the Simpson weights
w = [w;1]; w(1)=1;
res = (b-a)/(3*n)*f_x*w;

```

### Problematic cases for numerical integration

The above results are all based on the Taylor approximation (3.10), which requires that the function to be integrated is often differentiable. This is not always the case and the domain of integration might not be a finite interval  $[a, b]$ . Thus the algorithms have to be adapted.

- Examine a function that is not smooth at a point inside the interval, e.g.  $f(x) = |\cos(x)|$ , which is not differentiable at  $x = \frac{\pi}{2}$ . Then determine the integral

$$\int_0^2 |\cos(x)| dx = \int_0^{\pi/2} \cos(x) dx + \int_{\pi/2}^2 -\cos(x) dx = 1 + 1 - \sin(2) \approx 1.09070 .$$

Using Simpson's method with  $n = 100$  sub-intervals leads to an error of  $\approx 3 \cdot 10^{-5}$ , while integration of the smooth function  $\cos(x)$  leads to a much smaller error of  $\approx 8 \cdot 10^{-10}$ . The problem can be avoided by splitting up the integral  $\int_0^2 = \int_0^{\pi/2} + \int_{\pi/2}^2$ . Many Octave/MATLAB integration routines use options to indicate this type of special points inside the interval of integration.

```

n = 100; x = linspace(0,2,n+1); y = abs(cos(x));
Integral100 = simpson(y,0,2)
Error = Integral100 - (2-sin(2))

Integral100A = simpson(cos(x),0,2)
ErrorA = Integral100A - sin(2)

-->
Integral100 = 1.0907
Error = 2.7390e-05

Integral100A = 0.9093
ErrorA = 8.0830e-10

```

- For a function with an infinite integrand the Simpson and the trapezoidal rule fail. Find information on how to handle functions with  $\lim_{x \rightarrow a} f(x) = \infty$  in [IsaaKell66, §6.2, p.346].
- For a function with an infinite domain the Simpson and the trapezoidal rule fail. Find information on how to handle integrals of the type  $\int_a^\infty f(x) dx$  in [IsaaKell66, §6.3, p.350].

### 3.3.2 Integration of a Function given by a Formula

In this subsection assume that a function is given by an explicit expression  $y = f(x)$  and for fixed values  $-\infty < a < b < +\infty$  the goal is to find a good approximation of the definite integral  $\int_a^b f(x) dx$ . Usually the relative or absolute tolerance for the error are specified and the integral then computed. The number and location of points where functions is evaluated can be selected.

#### Gauss integration over an interval

**3-43 Result : Gauss integration**

The integral of a smooth function over the standard interval  $[-h, +h]$  is approximated by

$$\int_{-h}^h f(x) dx \approx \frac{h}{9} \left( 5f(-\sqrt{\frac{3}{5}}h) + 8f(0) + 5f(+\sqrt{\frac{3}{5}}h) \right)$$

and the error is estimated by

$$\text{error}_{2h} \leq C h^7.$$

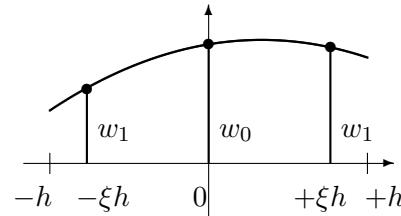
For a smooth function defined on the interval  $[a, b]$ , divided up into an even number of subintervals of length  $h = \frac{b-a}{n}$  this leads to the error

$$\text{error}_{[a,b]} \leq \tilde{C} h^6.$$

For polynomials up to degree 5 the values for the integral are exact.  $\diamond$

**Proof :** To integrate a general function  $f(x)$  over the interval  $[-h, +h]$  choose 3 symmetric integration points at  $-\xi h$ , 0 and  $+\xi h$  and integration weights  $w_0$  and  $w_1$ . Select these values such that the formula

$$\int_{-h}^{+h} f(x) dx \approx 2h (w_1 f(-\xi h) + w_0 f(0) + w_1 f(\xi h))$$



yields a result with an error as small as possible. To determine the optimal values for  $w_0$ ,  $w_1$  and  $\xi$  use a Taylor approximation and require that polynomials of increasing order are integrated exactly. This will lead to a small integration error. The computations are very similar to Result 3-40 for the verification of Simpson's rule.

Using the monomials 1,  $x$ ,  $x^2$ ,  $x^3$  and  $x^4$  leads to a system of 3 nonlinear equations, shown below. Since  $x^5$  is also integrated exactly the local integration error is proportional to  $h^7$ .

$p(x)$	$\int_{-h}^{+h} p(x) dx$	$= 2h (w_1 p(-\xi h) + w_0 p(0) + w_1 p(+\xi h))$	
1	$2h$	$= 2h (w_1 + w_0 + w_1)$	equation 1
$x$	0	$= 2h (-w_1 \xi h + w_0 0 + w_1 \xi h)$	OK
$x^2$	$\frac{2}{3} h^3$	$= 2h (+w_1 \xi^2 h^2 + w_0 0 + w_1 \xi^2 h^2)$	equation 2
$x^3$	0	$= 2h (-w_1 \xi^3 h^3 + w_0 0 + w_1 \xi^3 h^3)$	OK
$x^4$	$\frac{2}{5} h^5$	$= 2h (+w_1 \xi^4 h^4 + w_0 0 + w_1 \xi^4 h^4)$	equation 3
$x^5$	0	$= 2h (-w_1 \xi^5 h^5 + w_0 0 + w_1 \xi^5 h^5)$	OK
$x^6$	$\frac{2}{7} h^7$	$\neq 2h (+w_1 \xi^6 h^6 + w_0 0 + w_1 \xi^6 h^6)$	not OK

To solve the system

$$w_0 + 2w_1 = 1, \quad 2w_1 \xi^2 = \frac{1}{3} \quad \text{and} \quad 2w_1 \xi^4 = \frac{1}{5}$$

divide the second and third equation to conclude  $\xi^2 = \frac{3}{5}$  and thus  $\xi = \pm\sqrt{\frac{3}{5}}$ . Then the second equation leads to  $w_1 = \frac{5}{9}$ , and the first equation allows to conclude  $w_0 = \frac{9}{8}$ .

$$\int_{-h}^h f(x) dx \approx \frac{h}{9} \left( 5f(-\sqrt{\frac{3}{5}}h) + 8f(0) + 5f(+\sqrt{\frac{3}{5}}h) \right)$$

$\square$

The above is just one example of the Gauss integration idea. It is possible to use fewer or more integration points. The literature on the topic is vast, see e.g. [https://en.wikipedia.org/wiki/Gaussian\\_quadrature](https://en.wikipedia.org/wiki/Gaussian_quadrature). There are tables for the different Gauss integration schemes, e.g. the wikipedia page, [AbraSteg, Table 25.4] or the online version [DLMF15, §3.5] at <http://dlmf.nist.gov/>, or [TongRoss08, Table 6.1, page 188]. The two point approximation

Ex. 3.13

$$\int_{-h}^h f(x) dx \approx h \left( f\left(-\sqrt{\frac{1}{3}}h\right) + f\left(+\sqrt{\frac{1}{3}}h\right) \right)$$

and its error estimate  $\text{error}_{2h} \leq C h^5$  are given as an exercise.

**3-44 Example :** The Gauss algorithm for a three or five point integration is implemented Octave<sup>11</sup>. Find one possible implementation in `IntegrateGauss.m` with the built-in help.

```
help IntegrateGauss
-->
INTEGRAL = IntegrateGauss (X,F,N)

Integrate the function F over the interval X

parameters:
  * F the function, given as a function handle or string with the name
  * X the vector of x values with the domain of integration on each
    subinterval x_i to x_(i+1) a three or five point Gauss integration is used
  * N if this optional parameter equals 5, a five point Gauss formula is used,
    the default value is 3

return value: INTEGRAL the value of the integral
```

Use the following code to determine approximate values of  $\int_0^{\pi/2} \cos(x) dx = 1$ , using 10 sub-intervals and three or five Gauss points on each sub-interval.

```
n = 10 ; x = linspace(0,pi/2,n+1);
Integral3 = IntegrateGauss(x,@(x)sin(x))
Error3 = Integral3-1
Integral5 = IntegrateGauss(x,@(x)sin(x),5)
Error5 = Integral5-1
-->
Integral3 = 1.0000
Error3 = 7.4574e-12
Integral5 = 1.0000
Error5 = -1.1102e-16
```

The result shows that a Gauss integration with 10 sub-intervals and five Gauss points already generates a result whose accuracy is limited by the CPU accuracy. ◇

### IntegrateGauss.m

```
function Integral = IntegrateGauss (x,f,n)
  if nargin <= 2 n = 3; %% default is 3 Gauss points
  elseif n!=5 n = 3;
  endif
  x = x(:)';
  dx = diff(x);
  if n==3
```

<sup>11</sup>The source code has to be modified slightly to run under MATLAB.

```

pos      = 0.5 + [-sqrt(0.6)/2 0 +sqrt(0.6)/2]'; %% location of Gauss points
weight   = [5 8 5]'/18;                           %% weight for Gauss integration
else %% n=5
pos      = 0.5 + [-1/6*sqrt(5+2*sqrt(10/7)) -1/6*sqrt(5-2*sqrt(10/7))...
                 0 +1/6*sqrt(5-2*sqrt(10/7)) +1/6*sqrt(5+2*sqrt(10/7))]';
weight   = [(322-13*sqrt(70))/1800 (322+13*sqrt(70))/1800 64/225...
                 (322+13*sqrt(70))/1800 (322-13*sqrt(70))/1800 ]';
endif
IntegrationPoints = x(1:end-1) + pos*dx;
IntegrationWeights = weight*dx;
if (is_function_handle(f))
    Integral = sum( f(IntegrationPoints(:)).*IntegrationWeights(:));
else
    Integral = sum( feval(f, IntegrationPoints(:)).*IntegrationWeights(:));
endif
endfunction

```

The method of Gauss to integrate is used extensively by the method of finite elements to integrate over triangles or rectangles, see Sections 6.5.1 and 6.8.

### 3–45 Example : Convergence of the basic integration algorithms

The exact integral to be examined is given by

$$\int_0^{2\pi} \sin(2x) \exp(-x) dx = \frac{2}{5} (1 - \exp(-2\pi)) \approx 0.3993 . \quad (3.11)$$

Find the graph of this function in Figure 3.27(a). Three algorithms are used to determine the integral, using different values for the number  $n$  of sub-intervals, and consequently  $h = \frac{2\pi}{n}$ .

- trapezoidal: the error is expected to be proportional to  $h^2$ . In a double logarithmic graph this should<sup>12</sup> lead to a straight line with slope 2.
- Simpson: the error is expected to be proportional to  $h^4$  and thus a straight line with slope 4.
- Gauss: with three points, the error is expected to be proportional to  $h^6$  and thus a straight line with slope 6.

All of the above is confirmed by Figure 3.26 and Table 3.5. The surprising horizontal segments for the trapezoidal and Simpson's rule between  $n = 2$  and  $n = 4$  are caused by the zeros of  $\sin(2x)$ , which coincide with the points of evaluation. Thus the algorithms estimate the value of the integral by 0. ◇

$n$	trapez	Simpson	Gauss
2	-3.9925e-01	-3.9925e-01	-2.0145e-02
4	-3.9925e-01	-3.9925e-01	-4.7235e-04
8	-1.0334e-01	-4.7057e-03	-5.1212e-06
16	-2.5715e-02	1.6044e-04	-7.1676e-08
32	-6.4176e-03	1.5004e-05	-1.0885e-09
64	-1.6036e-03	1.0076e-06	-1.6886e-11

Table 3.5: Approximation errors of three integration algorithms. The examined integral is shown in (3.11) and the width of the sub-intervals is  $h = \frac{2\pi}{n}$ .

<sup>12</sup>Use the laws of logarithms and  $z = C \cdot h^p$  to conclude  $\log(z) = \log(C) + p \cdot \log(h)$ .

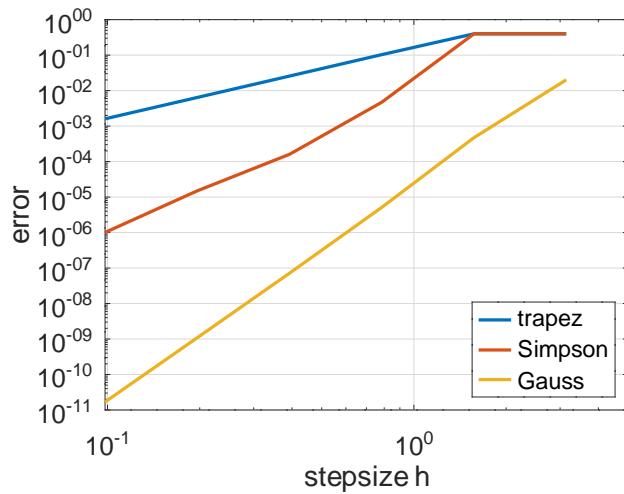


Figure 3.26: The approximation errors of three integration algorithms as function of the stepsize

### Richardson extrapolation for improved convergence

For the trapezoidal rule and Simpson's method the order of convergence (for smooth functions) is known. Using the result  $I_h$  with interval length  $h$  and  $I_{h/2}$  with interval length  $\frac{h}{2}$  can be used to generate an even more accurate estimate by extrapolation.

$$\begin{aligned} I_h - I &\approx c h^p \\ I_{h/2} - I &\approx c \left(\frac{h}{2}\right)^p \end{aligned} \implies 2^p (I_{h/2} - I) \approx c h^p \implies (2^p - 1) I \approx 2^p I_{h/2} - I_h .$$

- For the trapezoidal rule use  $p = 2$  to obtain the improved estimate

$$I \approx \frac{2^2 I_{h/2} - I_h}{2^2 - 1} = \frac{4 I_{h/2} - I_h}{3} .$$

- For Simpson's method use  $p = 4$  to obtain the improved estimate

$$I \approx \frac{2^4 I_{h/2} - I_h}{2^4 - 1} = \frac{16 I_{h/2} - I_h}{15} .$$

A similar approach is used in Section 3.4.5 to approximate solutions of ordinary differential equations.

### The idea of adaptive integration

To estimate the error when computing an integral  $I = \int_a^b f(x) dx$  one can use a subdivision in  $N$  sub-intervals, leading to the approximative  $I_N$ , then with  $2N$  sub-intervals, leading to  $I_{2N}$ . If the difference is small enough accept the result, otherwise determine  $I_{4N}$ .

To apply this idea use the trapezoidal rule with  $N$  sub-intervals.

$$T_N = \left( f(a) + 2 \sum_{i=1}^{N-1} f\left(a + i \frac{b-a}{N}\right) + f(b) \right) \frac{b-a}{2N}$$

For  $N = 6$  a graphical representation is given by

$$\begin{array}{ccccccccccccc} & \bullet & & \bullet \\ N = 6 & (1 & & 2 & & 2 & & 2 & & 2 & & 1) & & \frac{b-a}{12} \end{array}$$

The summations for  $T_N$  and  $T_{2N}$  are not independent, since all points used by  $T_N$  are also used by  $T_{2N}$ , and some more. The two formulas

$$\begin{aligned} T_N &= \left( f(a) + 2 \sum_{i=1}^{N-1} f(a + i \frac{b-a}{N}) + f(b) \right) \frac{b-a}{2N} \\ T_{2N} &= \left( f(a) + 2 \sum_{i=1}^{2N-1} f(a + i \frac{b-a}{2N}) + f(b) \right) \frac{b-a}{4N} \end{aligned}$$

can be visualized by

$\bullet$												
$N = 6$	(1	2	2	2	2	2	2	2	2	1)	$\frac{b-a}{12}$	
$N = 12$	(1	2	2	2	2	2	2	2	2	2	1)	$\frac{b-a}{24}$

Thus conclude

$$T_{2N} = \frac{1}{2} T_N + \frac{b-a}{2N} \sum_{i=1}^N f(a + (2i-1) \frac{b-a}{2N}).$$

The function has only to be evaluated at the new points. Using the sequence

$$T_1, \quad T_2, \quad T_4, \quad T_8, \quad T_{16}, \quad T_{32}, \quad \dots$$

it is possible to stop as soon as the desired accuracy is achieved.

Since the error for Simpson's approach is expected to be much smaller, the above has to be adapted. Use

$$S_{2N} = \frac{b-a}{3 \cdot 2N} \left( f(a) + 4 \sum_{i=1}^N f(x_{2i-1}) + 2 \sum_{i=1}^{N-1} f(x_{2i}) + f(b) \right),$$

where  $x_i = a + i \frac{b-a}{2N}$ . A graphical representation is given by

$\bullet$	$\bullet$											
$T_6$	(2	4	4	4	4	4	4	4	2)	$\frac{b-a}{2 \cdot 12}$		
$T_{12}$	(1	2	2	2	2	2	2	2	2	1)	$\frac{b-a}{2 \cdot 12}$	
$S_{12}$	(1	4	2	4	2	4	2	4	2	4	1)	$\frac{b-a}{3 \cdot 12}$

Thus for Simpson's rule use

$$S_{2N} = \frac{2}{3} (2 T_{2N} - \frac{1}{2} T_N) = \frac{1}{3} (4 T_{2N} - T_N).$$

Based on Gauss integration a similar saving of evaluations of the function  $f$  is not possible, since only points inside the sub-intervals are used.

For graphs similar to Figure 3.27(a) it works well to use finer meshes on all of the interval, but for graphs similar to Figure 3.27(b) a **local adaptation** is asked for, i.e. use more points where the function

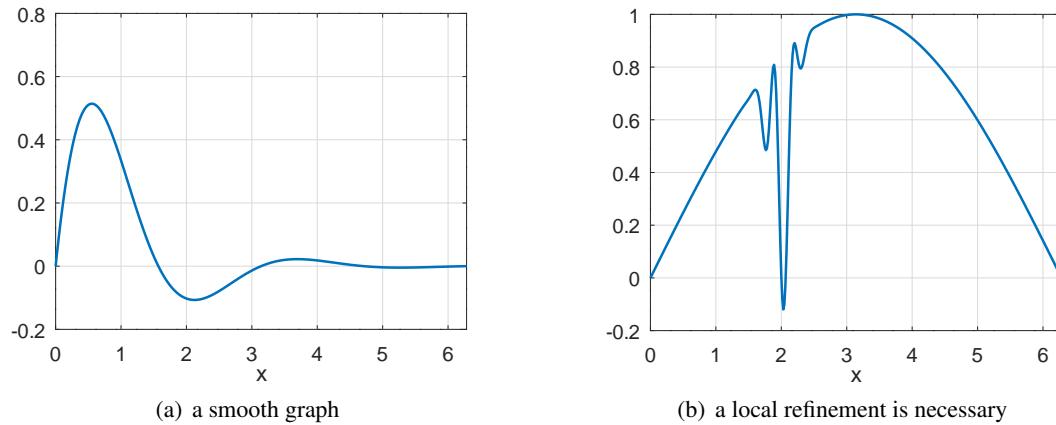


Figure 3.27: Two graphs of functions for integration

values vary severely. As example examine Simpson's approach on a sub-interval of length  $4 h$  starting at  $x_i$  and compare the two approximations of the integral.

$$\begin{aligned} \int_{x_i}^{x_i+4h} f(x) dx &\approx R_1 &= \frac{2h}{3} (f(x_i) + 4f(x_{i+2}) + f(x_{i+4})) \\ \int_{x_i}^{x_i+4h} f(x) dx &\approx R_2 &= \frac{h}{3} (f(x_i) + 4f(x_{i+1}) + 2f(x_{i+2}) + 4f(x_{i+3}) + f(x_{i+4})) \end{aligned}$$

If the difference  $|R_1 - R_2|$  is too large the interval has to be divided, otherwise proceed with the given result. The integration algorithms provided by MATLAB/Octave use local adaptation.

### 3.3.3 Integration Routines Provided by MATLAB/Octave

In this section some of the functions provided by Octave and/or MATLAB are illustrated. For more details consult the appropriate manuals.

## How to pass the function as argument to the numerical integration commands

With Octave and MATLAB there are different ways to pass a function to the integration routine.

- As a string with the function name  
A function can be defined in a separate file, e.g.

```
function res = ff(x)
    res = x.*sin (1./x).*sqrt (abs (x-1));
end
```

This defines the function

$$\text{ff}(x) = x \cdot \sin\left(\frac{1}{x}\right) \cdot \sqrt{|x - 1|}.$$

Observe that the code for the function is written with vectorization in mind, i.e. it can be evaluated for many arguments with one function call.

```
ff([0.2:0.2:2])
-->
-0.1715 0.1854 0.3777 0.3395 0 0.3972 0.5800 0.7251 0.8491 0.9589
```

This is essential for the integration routines and required for speed reasons.

- With *Octave* the definition of this function can be before the function call in an *Octave* script or in a separate file `ff.m`.
- With *MATLAB* the definition of this function has to be in a separate file `ff.m` or at the end of a *MATLAB* script.

An integration over the interval  $[0, 3]$  can then be performed by

```
result = quad('ff', 0, 3)
-->
ans = 1.9819
```

- As a function handle

The above integral can also be computed by using a function handle.

```
ff = @(x) x.*sin(1./x).*sqrt(abs(x-1));
quad(ff, 0, 3)
-->
ans = 1.9819
```

- function handles are very convenient to determine integrals depending on parameters. To compute the integrals

$$\int_0^2 \sin(\lambda t) dt$$

for  $\lambda$  values between 1 and 2 use

```
for lambda = [1:0.1:2]
    quad(@(t)sin(lambda*t), 0, 2)
end%for
```

### Using the function `integral()`

For the integration by `integral()` the function has to be passed as a handle. The limits of integration can be  $-\infty$  or  $+\infty$ . To determine

$$\int_0^{+\infty} \cos(t) \exp(-t) dt$$

use

```
integral(@(t)cos(t).*exp(-t), 0, Inf)
-->
ans = 0.5000
```

The function `integral()` can be used with optional parameters, specified as pairs of the string with the name of the option and the corresponding value. The most often used options are:

- `AbsTol` to specify the absolute tolerance. The default value is `AbsTol` =  $10^{-10}$ .
- `RelTol` to specify the relative tolerance. The default value is `AbsTol` =  $10^{-6}$ .
- The adaptive integration is refined to determine the value  $Q$  of the integral, until the condition

$$\text{error} \leq \max\{\text{AbsTol}, \text{RelTol} \cdot |Q|\}$$

is satisfied, i.e. either the relative or absolute error are small enough.

- `Waypoints` to specify a set of points at which the function to be integrated might not be continuous. This can be used instead of multiple calls of `integral()` on sub-intervals.

As an example for the function `integral()` examine the integral

$$\int_0^2 |\cos(x)| dx = 2 - \sin(2) .$$

Observe that this function is not differentiable at  $x = \frac{\pi}{2}$ . Thus the high order of approximation for Simpson and Gauss are not valid and problems are to be expected.

```
integral_exact = 2 - sin(2);
integral_1 = integral(@(x)abs(cos(x)),0,2)
integral_2 = integral(@(x)abs(cos(x)),0,2,'AbsTol',1e-12,'RelTol',1e-12)
integral_3 = integral(@(x)abs(cos(x)),0,2,'Waypoints',pi/2)
Log10_Errors = log10(abs([integral_1, integral_2 integral_3] - integral_exact))
-->
integral_1      =    1.0907
integral_2      =    1.0907
integral_3      =    1.0907
Log10_Errors   =  -7.9982  -14.3313  -15.6536
```

The results illustrate the usage of the tolerance parameters. Specifying the special point  $x = \frac{\pi}{2}$  generates a more accurate results with less computational effort.

### Using the function `quad()`

This function uses an recursive, adaptive Simpson scheme, as presented in the previous sections. This function will be removed from MATLAB in a future release.

- To integrate a function  $f$  over the interval  $[a, b]$  use `quad(f, a, b)`, where `f` is a string with the name of the function or a function handle.
- With the optional third argument `TOL` a vector with the absolute and relative tolerances can be specified.
- With Octave the optional fourth argument `SING` is a vector of values where singularities are expected. In addition options can be read or set by calling `quad_options`.

The two functions leading to Figure 3.27 are

$$f_1(x) = \sin(2x) \exp(-x) \quad \text{and} \quad f_2(x) = \sin\left(\frac{x}{2}\right) - \exp(-20(x-2)^2) \sin(10x) .$$

To integrate these functions over the interval  $[0, 2\pi]$  use (Octave only)

```
a = 0; b = 2*pi;
[q, ier, nfun, err] = quad (@(x) sin(2*x).*exp(-x), a, b)
[q, ier, nfun, err] = quad (@(x) sin(x/2)-exp(-20*(x-2).^2).*sin(10*x).^2, a, b)
```

The results show that the first function required 21 evaluations of the function  $f_1(x)$  while  $f_2(x)$  had to be evaluated 273 times. This is caused by the local adaptation around  $x \approx 2$ , obvious in Figure 3.27(b).

### Using the function `quadv()`

The function `quadv()` is similar to `quad()`, but can evaluate vector valued integrals. The basic call is `quadv(f, a, b)`, where `f` is a string with the name of the function or a function handle. If a second return argument is asked for the number of function evaluations is given. This function will be removed from MATLAB in a future release, use `integral()` with the option `ArrayValued`.

To evaluate

$$\vec{Q} = \int_0^{\pi/2} \exp(-t) \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix} dt$$

use

```
[Q,num_eval] = quadv(@(t)[sin(t);cos(t)]*exp(-t),0,0.5*pi)
-->
Q =
    0.3961
    0.6039

num_eval = 17
```

The same can be obtained by calling `integral()` with the option `ArrayValued`.

```
Q = integral(@(t)[sin(t);cos(t)]*exp(-t),0,0.5*pi, 'ArrayValued',true)
```

### More functions available in Octave

In Section 23.1 of the Octave manual a few more integration functions are documented.

`quad` Numerical integration based on Gaussian quadrature.

`quadv` Numerical integration using an adaptive vectorized Simpson's rule.

`quadl` Numerical integration using an adaptive Lobatto rule.

`quadgk` Numerical integration using an adaptive Gauss-Kronrod rule.

`quadcc` Numerical integration using adaptive Clenshaw-Curtis rules.

`integral` A compatibility wrapper function that will choose between `quadv` and `quadgk` depending on the integrand and options chosen.

### 3.3.4 Integration over Domains in $\mathbb{R}^2$

Integrals over rectangular domains in Figure 3.28(a)  $\Omega = [a, b] \times [c, d] \subset \mathbb{R}^2$  are computed by nested 1-D integrations

$$Q = \iint_{\Omega} f(x, y) dA = \int_a^b \left( \int_c^d f(x, y) dy \right) dx .$$

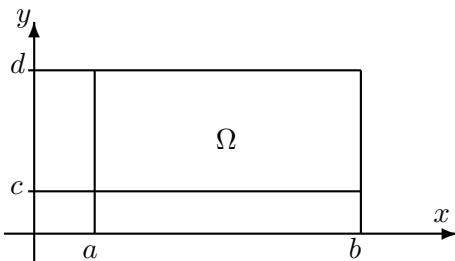
MATLAB/Octave provides commands to perform this double integration. The function  $f(x, y)$  depends on two arguments, the corresponding MATLAB/Octave code has to accept matrices for  $x$  and  $y$  as arguments and return a matrix of the same size.

- **integral2:** The basic call is  $Q = \text{integral2}(f, a, b, c, d)$ . Possible options have to be specified by the string with the name of the option and the corresponding value.
- **quad2d:** The basic call is  $Q = \text{quad2d}(f, a, b, c, d)$ . Possible options have to be specified by the string with the name of the option and the corresponding value.

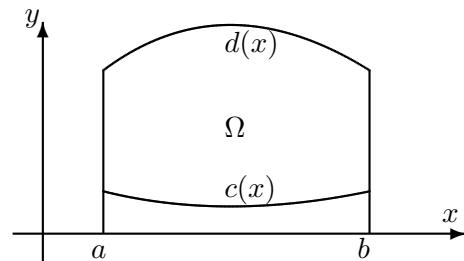
**3-46 Example :** To integrate the function  $f(x, y) = x^2 + y$  over the rectangular domain  $1 \leq x \leq 2$  and  $0 \leq y \leq 3$  use

```
quad2d(@(x,y)x.^2+y,1,2,0,3)
integral2(@(x,y)x.^2+y,1,2,0,3)
-->
ans = 11.500
ans = 11.500
```

◇



(a) a rectangular domain



(b) a general domain

Figure 3.28: Domains in the plane  $\mathbb{R}^2$ 

Integrals over non rectangular domains in Figure 3.28(b) can be given by  $a \leq x \leq b$  and  $x$  dependent limits for the  $y$  values  $c(x) \leq y \leq d(x)$ . The integral are computed by nested 1-D integrations

$$Q = \iint_{\Omega} f(x, y) dA = \int_a^b \left( \int_{c(x)}^{d(x)} f(x, y) dy \right) dx .$$

MATLAB/Octave provides commands to perform this double integration. The only change is that the upper and lower limits are provided as functions of  $x$ .

Integrals over domains where the left and right limit are given as functions of  $y$  (i.e.  $a(y) \leq x \leq b(y)$ ) are covered by swapping the order of integration,

$$Q = \iint_{\Omega} f(x, y) dA = \int_c^d \left( \int_{a(y)}^{b(y)} f(x, y) dx \right) dy .$$

**3-47 Example :** Consider the triangular domain with corners  $(0, 0)$ ,  $(1, 0)$  and  $(0, 1)$ . Thus the limits are  $0 \leq x \leq 1$  and  $0 \leq y \leq 1 - x$ . To integrate the function

$$f(x, y) = \frac{(1 + x + y)^2}{\sqrt{x^2 + y^2}}$$

use

$$\iint_{\Omega} f(x, y) dA = \int_0^1 \left( \int_0^{1-x} \frac{(1 + x + y)^2}{\sqrt{x^2 + y^2}} dy \right) dx .$$

This is readily implemented using `integral2()` or `quad2d()`.

```
fun = @(x,y) 1./sqrt(x+y).* (1+x+y).^2 ;
ymax = @(x) 1 - x;
Q1 = quad2d(fun,0,1,0,ymax)
Q2 = integral2(fun,0,1,0,ymax)
-->
Q1 = 0.2854
Q2 = 0.2854
```

◇

**3-48 Example :** The previous example can also be solved using polar coordinates  $r$  and  $\phi$ . Express the area element  $dA$  in polar coordinates by

$$dA = dx \cdot dy = r \cdot dr d\phi .$$

For the function use

$$f_p(r, \phi) = f(x, y) = f(r \cos(\phi), r \sin(\phi)) .$$

The upper limit  $y = 1 - x$  of the domain has to be expressed in terms of  $r$  and  $\phi$  by

$$y = 1 - x \implies r \sin(\phi) = 1 - r \cos(\phi) \implies r_{\max}(\phi) = \frac{1}{\cos(\phi) + \sin(\phi)} .$$

Then the double integral

$$\iint_{\Omega} f_p(r, \phi) dA = \int_0^{\pi/2} \left( \int_0^{r_{\max}(\phi)} f_p(r, \phi) r dr \right) d\phi .$$

is computed, using `integral2()` or `quad2d()`.

```
polarfun = @(theta,r) fun(r.*cos(theta),r.*sin(theta)).*r;
rmax = @(theta) 1./(sin(theta) + cos(theta));
Q1 = quad2d(polarfun,0,pi/2,0,rmax)
Q2 = integral2(polarfun,0,pi/2,0,rmax)
-->
Q1 = 0.2854
Q2 = 0.2854
```

◇

**3–49 Observation :**

- MATLAB/*Octave* provide another function `dblquad()`.
- For triple integrals MATLAB/*Octave* provide the command `integral3()` with a syntax similar to `integral2()`.
- To integrate over triangles or rectangles is a task required for the method of finite elements. Find results in this direction in Sections 6.5.1 and 6.8.



## 3.4 Solving Ordinary Differential Equations, Initial Value Problems

In this section a few types of ordinary differential equations are solved by numerical methods. The goal is to

- understand on how to setup ordinary differential equations for numerical algorithms.
- understand the basics of some algorithms used to solve ODEs.
- be able to use *Octave/MATLAB* to solve ODEs and systems of ODEs reliably.

To achieve this goal the section is organized as follows:

- In subsection 3.4.1 different types of ODEs are shown and *Octave/MATLAB* commands used to generate numerical approximations. The provided examples should help to use *MATLAB/Octave* to solve ODEs.
- In subsection 3.4.2 find the basic ideas for three ODE solvers: Euler, Heun and a Runge–Kutta method. The essential error estimates are shown. Simple codes for algorithms with fixed step size are given. The stability of the algorithms is examined.
- In subsection 3.4.4 general Runge–Kutta schemes and their Butcher tables are introduced.
- In subsection 3.4.5 the idea of local extrapolation and adaptive step sizes is presented and illustrated by one example.
- In subsection 3.4.6 the usage of the solvers provided by *Octave/MATLAB* is presented.
- In subsection 3.4.7 four algorithms in *Octave/MATLAB* are examined carefully. Particular attention is given to stiff problems.

### 3.4.1 Different Types of Ordinary Differential Equations

#### Introduction

The simplest form of an ODE (Ordinary Differential Equation) is

$$\frac{d}{dt} u(t) = f(t, u(t))$$

for a given function  $f : [t_{\text{init}}, t_{\text{end}}] \times \mathbb{R} \rightarrow \mathbb{R}$ . An additional initial time  $t_0$  and an initial value  $u_0 = u(t_0)$  lead to an IVP (Initial Value Problem). A solution has to be a function, satisfying the differential equation and the initial condition. One can show that for differentiable functions  $f$  any IVP has a unique solution on an interval  $a < t_0 < b$ . The final time  $t < b$  might be smaller than  $+\infty$ , since the solution could blow up in finite time.

#### 3–50 Example : The logistic differential equation

The behavior of the size of a population  $0 \leq p(t)$  with a limited nutrition supply can be modeled by the logistic differential equation

$$\frac{d}{dt} p(t) = (\alpha - p(t)) p(t) .$$

In this case the differential equation with  $f(p) = (\alpha - p)p$  is autonomous, i.e.  $f$  does not explicitly depend on the time  $t$ . In Figure 3.29 three solutions for  $\alpha = 2$  with different initial values are shown. The vector field is generated by displaying many (rescaled) vectors  $(1, f(p))$  attached at points  $(t, p)$ . The function  $p(t)$  being a solution of the ordinary differential equation is equivalent to the slope of the curve to coincide with the directions of the vector field. Thus vector fields can be useful to understand the qualitative behavior of solutions of ODEs. Figure 3.29 is generated by the code below.

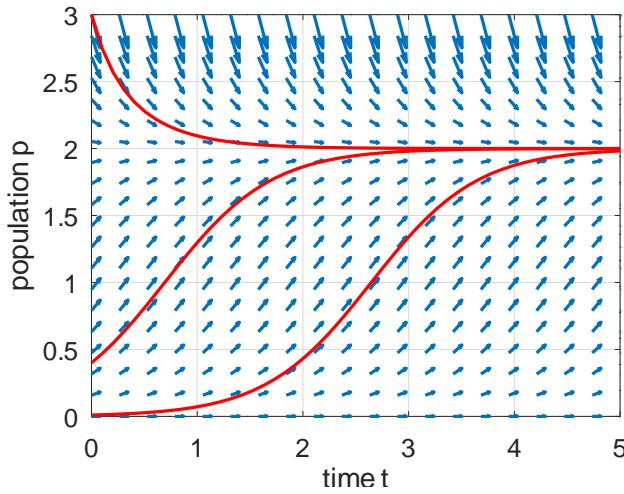


Figure 3.29: Vector field and three solutions for a logistic equation

```
t_max = 5; p_max = 3;
[t,p] = meshgrid(linspace(0, t_max, 20),linspace(0,p_max, 20));
v1 = ones(size(t)); v2 = p.* (2-p);
figure(1); quiver(t,p,v1,v2)
xlabel('time t'); ylabel('population p')
axis([0 t_max, 0 p_max])

[t1,p1] = ode45(@ (t,p)p.* (2-p),linspace(0,t_max,50),0.4);
[t2,p2] = ode45(@ (t,p)p.* (2-p),linspace(0,t_max,50),3.0);
[t3,p3] = ode45(@ (t,p)p.* (2-p),linspace(0,t_max,50),0.01);
hold on
plot(t1,p1,'r',t2,p2,'r', t3,p3,'r')
hold off
```

◇

### Systems of ordinary differential equations

The above idea can be applied to a system of ODEs. As example consider the famous predator–pray model by Volterra–Lotka<sup>13</sup>.

#### 3–51 Example : Volterra–Lotka predator–prey model

Consider two different species with the size of their population given by  $x(t)$  and  $y(t)$ .

Ex. 3.14

$$\begin{aligned}x(t) &\quad \text{population size of pray at time } t \\y(t) &\quad \text{population size of predator at time } t\end{aligned}$$

The predators  $y$  (e.g. sharks) are feeding of the pray  $x$  (e.g. small fish). The food supply for the pray is limited by the environment. The behavior of these two populations can be described by a system of two first order differential equations.

$$\begin{aligned}\frac{d}{dt} x(t) &= (c_1 - c_2 y(t)) x(t) \\ \frac{d}{dt} y(t) &= (c_3 x(t) - c_4) y(t)\end{aligned}$$

<sup>13</sup>Proposed by Alfred J. Lotka in 1910 and Vito Volterra in 1926.

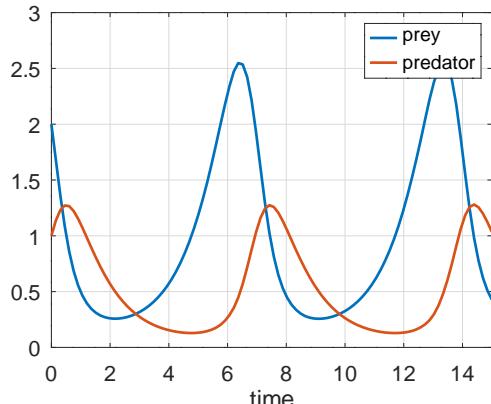
where  $c_i$  are positive constants. This function can be implemented in a function file `VolterraLotka.m` in MATLAB/Octave.

**VolterraLotka.m**

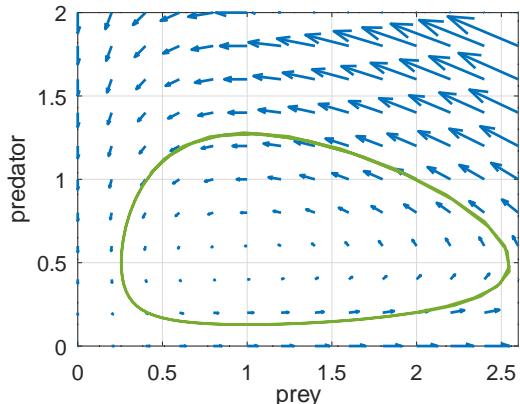
```
function res = VolterraLotka(x)
c1 = 1; c2 = 2; c3 = 1; c4 = 1;
res = [(c1-c2*x(2))*x(1);
        (c3*x(1)-c4)*x(2)];
end%function
```

With the help of the above function generate information about the solutions of this system of ODEs.

- Generate the data for the vector field and then use the command `quiver` to display the vector field, shown in Figure 3.30(b).
- Use `ode45()` to generate numerical solutions, in this case with initial values  $(x(0), y(0)) = (2, 1)$  and for 100 times  $0 \leq t_i \leq 15$ . Display the result in Figure 3.30.



(a) as function of time



(b) vector field and solution

Figure 3.30: One solution and the vector field for the Volterra-Lotka problem

```
x = 0:0.2:2.6; % define the x values to be examined
y = 0:0.2:2.0; % define the y values to be examined

n = length(x); m = length(y);
Vx = zeros(n,m); Vy = Vx; % create zero vectors for the vector field

for i = 1:n
    for j = 1:m
        v = VolterraLotka([x(i),y(j)],0); % compute the vector
        Vx(i,j) = v(1); Vy(i,j) = v(2);
    end%for
end%for

t = linspace(0,15,100);
[XY] = ode45(@(t,x)VolterraLotka(x),t,[2;1]);

figure(1); plot(t,XY)
xlabel('time'); legend('prey','predator'); axis([0,15,0,3]); grid on
```

```
figure(2); quiver(x,y,Vx',Vy',2); hold on
plot(XY(:,1),XY(:,2));
axis([min(x),max(x),min(y),max(y)]);
grid on; xlabel('prey'); ylabel('predator'); hold off
```

◊

### Converting an ODE of higher order to a system of order 1

Ordinary differential equation of higher order can be converted to systems of order 1. Thus most numerical algorithm are applicable to system of order 1, but not to higher order ODEs. The method is illustrated by the example of a pendulum equation.

#### 3–52 Example : ODE for a damped pendulum

The equation

$$\ddot{x}(t) + \alpha \dot{x}(t) + k x(t) = f(t)$$

Ex. 3.16

describes a mass attached to a spring with an additional damping term  $\alpha \dot{x}$ . Introducing the new variables  $y_1(t) = x(t)$  and  $y_2(t) = \dot{x}(t)$  leads to

$$\frac{d}{dt} y_1(t) = \dot{x}(t) = y_2(t)$$

and

$$\frac{d}{dt} y_2(t) = \frac{d}{dt} \dot{x}(t) = \ddot{x}(t) = f(t) - \alpha \dot{x}(t) - k x(t) = f(t) - \alpha y_2(t) - k y_1(t)$$

This can be written as a system of first order equations

$$\frac{d}{dt} \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{pmatrix} = \begin{pmatrix} y_2(t) \\ f(t) - k y_1(t) - \alpha y_2(t) \end{pmatrix}$$

or

$$\frac{d}{dt} \vec{y}(t) = \vec{F}(\vec{y}(t))$$

and with the help of a function file the problem can be solved with computations very similar to the above Volterra–Lotka example. The code below will compute a solution with the initial displacement  $x(0) = 0$  and initial velocity  $\frac{d}{dt} x(0) = 1$ . Then Figure 3.31 will be generated.

- In Figure 3.31(a) find the graphs of  $x(t)$  and  $v(t)$  as function of the time  $t$ . The effect of the damping term  $-\alpha v(t) = -0.1 v(t)$  is clearly visible.
- In Figure 3.31(b) find the vector field and the computed solution. The horizontal axis represents the displacement  $x$  and the vertical axis indicates the velocity  $v = \dot{x}$ . This is the **phase portrait** of the second order ODE.

```
y = -1:0.2:1; v = -1:0.2:1; n = length(y); m = length(v);
Vx = zeros(n,m); Vy = Vx; % create zero vectors for the vector field

function ydot = Spring(y)
    ydot = zeros(size(y));
    k = 1; al = 0.1;
    ydot(1) = y(2);
    ydot(2) = -k*y(1)-al*y(2);
end%function

for i = 1:n
```

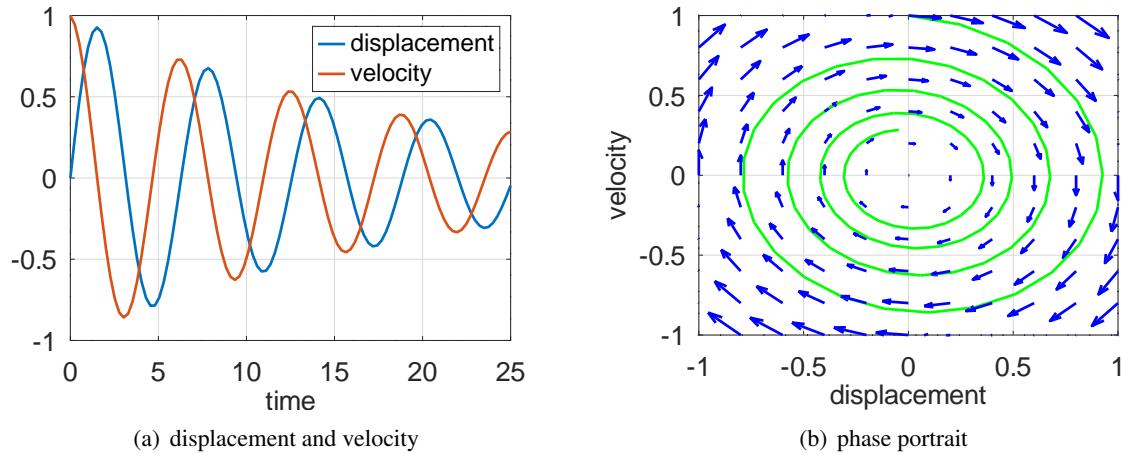


Figure 3.31: Vector field and a solution for a spring-mass problem

```

for j = 1:m
    z = Spring([y(i),v(j)]);
    % compute the vector
    Vx(i,j) = z(1); Vy(i,j) = z(2); % store the components
end%for
end%for

t = linspace(0,25,100);
[XY] = ode45(@(t,y)Spring(y),t,[0;1]);

figure(1); plot(t,XY)
xlabel('time'); legend('displacement','velocity')
axis(); grid on

figure(2); plot(XY(:,1),XY(:,2),'g'); % plot solution in phase portrait
axis([min(y),max(y),min(v),max(v)]);
hold on
quiver(y,v,Vx',Vy','b');
xlabel('displacement'); ylabel('velocity');
grid on; hold off

```

◊

### 3.4.2 The Basic Algorithms

In this subsection three basic algorithms are presented. The purpose is to understand how these algorithms work, the resulting code is only useful for didactical end demo purposes. For real world problem use the codes presented in subsection 3.4.6, starting on page 207. The explications are given for single ODEs, but the methods apply to systems of ODEs too, without major modifications.

#### The Euler method

To understand the basic idea on numerical ODE solvers the method of Euler is easiest to understand. The idea carries over to more sophisticated and efficient approaches.

As a first example examine the IVP (Initial Value Problem)

$$\frac{d}{dt}x(t) = x(t)^2 - 2t \quad \text{with} \quad x(0) = 0.75 .$$

Thus search for a curve following the corresponding vector field in Figure 3.32. Use the definition of the

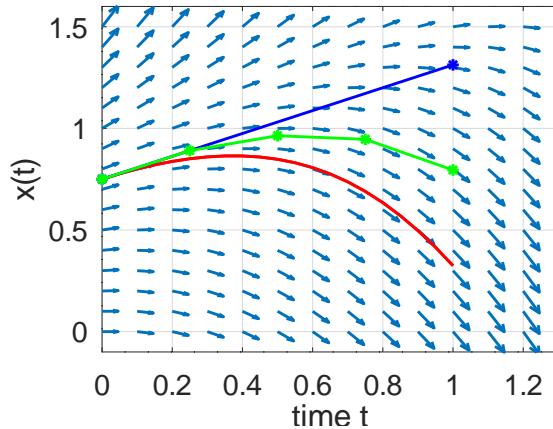


Figure 3.32: Vector field and solution of the ODE  $\frac{d}{dt}x(t) = x(t)^2 - 2t$ . The true solution is displayed in red, the solution generated by one Euler step in blue and the result using four Euler steps in green.

derivative

$$\frac{d}{dt}x(0) = \lim_{h \rightarrow 0} \frac{x(h) - x(0)}{h}$$

at  $t = 0$ . Instead of the limes use a "small" value for  $h$ . The differential equation is transformed into an algebraic equation

$$\frac{x(h) - x(0)}{h} = f(0, x(0)) .$$

This can easily be solved for  $x(h)$

$$x(h) = x(0) + h f(0, x(0)) .$$

Thus the first step of this method leads to the straight line in Figure 3.32. A bit more general, to move from time  $t$  to time  $t + h$  use

$$x(t + h) = x(t) + h f(t, x(t)) .$$

This is **Euler's method** or also the **explicit** method.

Above the stepsize  $h = 1$  was used to determine  $x(1)$ . To obtain a better approximation use smaller values for  $h$ . With the stepsize  $h = \frac{1}{4} = 0.25$  find

$$\begin{aligned} x(0.25) &\approx 0.75 + 0.25 (0.75^2 - 2 \cdot 0) &= 0.890625 \\ x(0.50) &= 0.890625 + 0.25 (0.890625^2 - 2 \cdot 0.25) &\approx 0.963928 \\ x(0.75) &\approx 0.963928 + 0.25 (0.963928^2 - 2 \cdot 0.5) &\approx 0.9462176 \\ x(1.00) &\approx 0.9462176 + 0.25 (0.9462176^2 - 2 \cdot 0.75) &\approx 0.7950496 \end{aligned}$$

This leads to the four straight line segments in Figure 3.32 and this approximate solution is already closer to the exact solution.

### The Heun method

This slightly more advanced method is known as method of Heun, or also Runge–Kutta of order 2. To discretize

$$\dot{x} = f(t, x) \quad \text{with} \quad x(t_0) = x_0$$

with step size  $h$  from  $t = t_i$  to  $t = t_{i+1} = t_i + h$  use the computational scheme:

$$\begin{aligned} k_1 &= f(t_i, x_i) \\ k_2 &= f(t_i + h, x_i + k_1 h) \\ x_{i+1} &= x_i + h \frac{k_1 + k_2}{2} \\ t_{i+1} &= t_i + h \end{aligned}$$

At two different positions the function  $f(t, x)$  is evaluated, leading to two slopes  $k_1$  and  $k_2$  for the solution of the ODE. Then one time step is performed with the average of the two slopes. This is visualized in Figure 3.33.

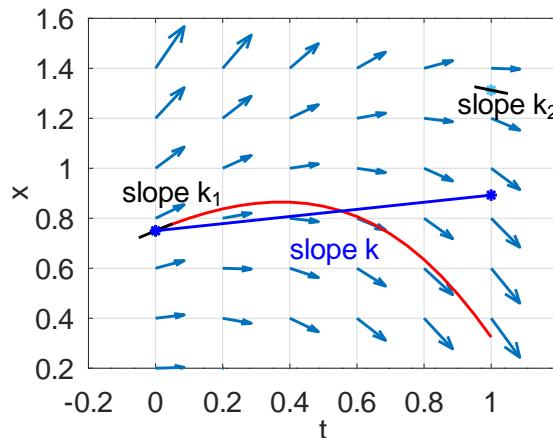


Figure 3.33: One step of the Heun Method for the ODE  $\frac{dx}{dt} = -x^2(t) - 2t$  with  $x(0) = 0.75$  and step size  $h = 1$

For the initial value problem  $\frac{dx}{dt} = x(t)^2 - 2t$  with  $x(0) = 0.75$  the calculations for one Heun step of length  $h = 1$  are given by

$$\begin{aligned} k_1 &= f(t_0, x_0) &= f(0, 0.75) &= 0.5625 \\ k_2 &= f(t_0 + h, x_0 + h, k_1) &= f(1, 1.3125) &= -0.27734375 \\ k &= \frac{1}{2}(k_1 + k_2) &= 0.142578125 \\ x(1) &\approx x(0) + h k &= 0.75 + 1(0.142578125) &= 0.892578125 \end{aligned}$$

This is a better approximation than the one generated by one Euler step. The above computations can be performed by MATLAB/Octave:

```
x0 = 0.75; h = 1; % set initial value and step size
f = @(t,x)x^2-2*t; % define the function f(t,x) = x^2 - 2t
h = 1;
k1 = f(0,x0)
k2 = f(h,x0+h*k1)
k = (k1+k2)/2
x1 = x0+h*k
```

### The classical Runge–Kutta method

One of the most often used methods is a Runge–Kutta method of order 4. It is often called the classical Runge–Kutta method. To apply one time step for the IVP

$$\dot{x} = f(t, x) \quad \text{with} \quad x(t_0) = x_0$$

with step size  $h$  from  $t = t_i$  to  $t = t_{i+1} = t_i + h$  use the following computational scheme:

$$\begin{aligned} k_1 &= f(t_i, x_i) \\ k_2 &= f(t_i + h/2, x_i + k_1 h/2) \\ k_3 &= f(t_i + h/2, x_i + k_2 h/2) \\ k_4 &= f(t_i + h, x_i + k_3 h) \\ x_{i+1} &= x_i + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) \\ t_{i+1} &= t_i + h \end{aligned}$$

At four different positions the function  $f(t, x)$  is evaluated, leading to four slopes  $k_i$  for the solution of the ODE. Then one time step is performed with a weighted average of the four slopes. This is visualized in Figure 3.34.

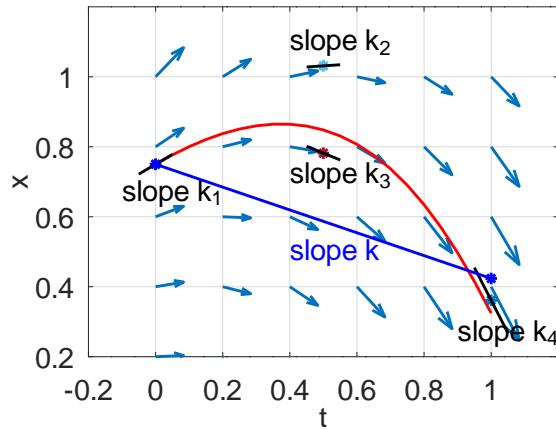


Figure 3.34: One step of the Runge–Kutta Method of order 4 for the ODE  $\frac{dx}{dt} = -x^2(t) - 2t$  with  $x(0) = 0.75$  and step size  $h = 1$

For the initial value problem  $\frac{dx}{dt} = x(t)^2 - 2t$  with  $x(0) = 0.75$  the calculations for one Runge–Kutta step of length  $h = 1$  are given by

$$\begin{aligned} k_1 &= f(t_0, x_0) &= f(0, 0.75) &= 0.5625 \\ k_2 &= f(t_0 + \frac{h}{2}, x_0 + \frac{h}{2} k_1) &= f(\frac{1}{2}, 1.03125) &\approx 0.0634766 \\ k_3 &= f(t_0 + \frac{h}{2}, (x_0 + \frac{h}{2} k_1)) &\approx f(\frac{1}{2}, 0.781738) &\approx -0.388885 \\ k_4 &= f(t_0 + h, x_0 + h k_3) &\approx f(1, 0.36111474) &\approx -1.869596 \\ k &= \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4) &\approx -0.326319 \\ x(1) &\approx x(0) + h k &\approx 0.75 + 1 (-0.326319) &\approx 0.423681 . \end{aligned}$$

The advanced numerical solver `ode23()` generated the answer  $x(1) \approx 0.32449$ . Thus the answer of the RK algorithm is considerably closer than the answer of  $x(1) \approx 0.7975$  generated by four Euler steps. For both approaches the right hand side of the ODE was evaluated four times, i.e. a comparable computational effort. The above computations can be performed by MATLAB or Octave:

```

x0 = 0.75; h = 1; % set initial value and step size
f = @(t,x)x^2-2*t; % define the function f(x) = x^2 - 2 t
h = 1;
k1 = f(0,x0)
k2 = f(h/2,x0+h/2*k1)
k3 = f(h/2,x0+h/2*k2)
k4 = f(h,x0+h*k3)
k = (k1+2*k2+2*k3+k4)/6
x1 = x0+h*k

```

### Discretization errors for Euler, Heun and Runge–Kutta

For all three of the above approximation methods to solutions of ODEs one can (and should) use estimates for the discretization errors. For smaller values of the step length  $h$  the error is expected to be smaller. As one example examine the error of the Euler method. Assuming that a solution  $x(t)$  of an ODE  $\frac{d}{dt}y(t) = f(t, y(t))$  which can be represented by a Taylor approximation

$$\begin{aligned} y(t_0 + h) &= y(t_0) + y'(t_0)h + \frac{y''(t_0)}{2!}h^2 + \frac{y'''(t_0)}{3!}h^3 + \dots + \frac{y^{(k)}(t_0)}{k!}h^k + \dots \\ &= y(t_0) + y'(t_0) \cdot h + h^2 \cdot \left[ \frac{y''(t_0)}{2!} + \frac{y'''(t_0)}{3!}h + \dots + \frac{y^{(k)}(t_0)}{k!}h^{k-2} + \dots \right]. \end{aligned}$$

Using  $y'(t_0) = f(t_0, y(t_0))$  and the Euler approximation

$$y_E(t_0 + h) = y(t_0) + h f(t_0, y(t_0))$$

leads to

$$y(t_0 + h) - y_E(t_0 + h) = h^2 \cdot \left[ \frac{y''(t_0)}{2!} + \frac{y'''(t_0)}{3!}h + \dots + \frac{y^{(k)}(t_0)}{k!}h^{k-2} + \dots \right].$$

Thus for small values of  $h$  obtain the local error estimate for one Euler step

$$|y_E(t_0 + h) - y(t_0 + h)| \leq C_E h^2 \quad (3.12)$$

where the “constant”  $C_E$  is related to second derivatives of the function  $f(t, y)$ . Thus the local discretization error of Euler method is of order 2. To arrive at a final time  $t_0 + T$  one has to apply  $n = T/h$  steps and each step might add some error. This leads to the global discretization error

$$|y_E(t_0 + T) - y(t_0 + Y)| \leq \tilde{C}_E h. \quad (3.13)$$

Thus the global discretization error of the Euler method is of order 1.

Similar arguments (with more tedious computations) can be performed for the method of Heun and Runge–Kutta, leading to Table 3.6.

Verfahren	step size $h$	local error	global error
Euler	$h = \frac{T}{n}$	$\approx C_E \cdot h^2$	$\approx C_E \cdot n \cdot h^2 = \tilde{C}_E h$
Heun	$h = \frac{T}{n}$	$\approx C_H \cdot h^3$	$\approx C_H \cdot n \cdot h^3 = \tilde{C}_H h^2$
Runge–Kutta	$h = \frac{T}{n}$	$\approx C_{RK} \cdot h^5$	$\approx C_{RK} \cdot n \cdot h^5 = \tilde{C}_{RK} h^4$

Table 3.6: Discretization errors for the methods of Euler, Heun and Runge–Kutta

For most problems Runge–Kutta is the most efficient algorithm to generate approximate solutions to ODEs. There are multiple aspects to be taken into account, i.e. to compare Euler’s method to Runge–Kutta:

### 1. Computational effort for one step

The computational effort for most applications is dominated by the evaluation of the RHS  $f(t, x)$  of the ODE. For the Euler method one call of the RHS is required, while Runge–Kutta requires 4 calls of the RHS.

Advantage: Euler

### 2. Differentiability of $f(t, x)$

For the above error estimates to be correct Euler requires that the function  $f$  be twice differentiable, while Runge–Kutta requires that  $f$  is four times differentiable.

Advantage: Euler

### 3. Order of consistency

The global discretization error for Euler is proportional to  $h$ , while it is proportional to  $h^4$  for Runge–Kutta.

Advantage: Runge–Kutta.

### 4. Number of time steps

Based on the higher order of convergence one usually gets away with fewer time steps for Runge–Kutta, for a given total error. The smaller  $h$  the better Runge–Kutta will perform.

Advantage: Runge–Kutta.

**3–53 Example :** The last of the above arguments is by far the most important. This is illustrated by an example, see [MeybVach91]. The initial value problem  $\frac{dy}{dt} = 1 + (y(t) - t)^2$  with  $y(0) = 0.5$  is solved by  $y(t) = t + \frac{1}{2-t}$ , e.g.  $y(1.8) = 6.8$ . Use  $n$  time steps of length  $n = \frac{1.8}{n}$  to approximate  $y(1.8)$  numerically. This leads to Table 3.7. With 72 calls of the RHS by Runge–Kutta the global error is of the same size as with 7200 calls by Euler. ◇

method	$h$	$n$	number of calls	global error
Euler	0.1	18	18	$2.23 \cdot 10^{-0}$
Runge–Kutta	0.1	18	72	$3.40 \cdot 10^{-3}$
Euler	0.01	180	180	$4.91 \cdot 10^{-1}$
Runge–Kutta	0.01	180	720	$4.20 \cdot 10^{-7}$
Euler	0.001	1800	1800	$5.66 \cdot 10^{-2}$
Runge–Kutta	0.001	1800	7200	$4.32 \cdot 10^{-11}$

Table 3.7: A comparison of Euler and Runge–Kutta

The ordinary differential equation

$$\frac{d}{dx} u(x) = f(u(x)) \quad \text{with} \quad u(x_0) = u_0$$

is equivalent<sup>14</sup> to the integral equation

$$u(x) = u_0 + \int_{x_0}^x f(u(s)) ds .$$

<sup>14</sup>

$\frac{d}{dx} \left( u_0 + \int_{x_0}^x f(u(s)) ds \right) = f(u(x)) \quad \text{and} \quad u(x_0) = u_0$

Thus it is no surprise that there is a close connection between numerical integration and solving ODEs. Compare the order of convergence of the different methods. It stands out that the order of the error for Runge–Kutta is higher than expected.

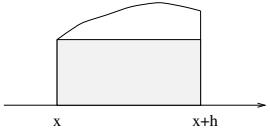
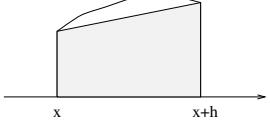
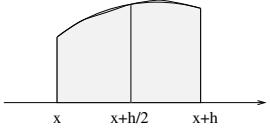
integral	differential equation
exact solutions	
$I = \int_x^{x+h} f(t) dt$	$u'(x) = f(u(x))$ $u(x + h) = u(x) + \int_x^{x+h} f(u(t)) dt$
approximate solutions	
rectangular rule  $I \approx h f(x)$ error = $O(h^2)$	Euler method $k_1 = f(u(x))$ $u(x + h) \approx u(x) + h k_1$ local error = $O(h^2)$
trapezoidal rule  $I \approx \frac{h}{2} (f(x) + f(x + h))$ error = $O(h^3)$	method of Heun $k_1 = f(u(x))$ $k_2 = f(u(x) + k_1 h)$ $u(x + h) \approx u(x) + \frac{h}{2} (k_1 + k_2)$ local error = $O(h^3)$
Simpson's rule  $I \approx \frac{h}{6} (f(x) + 4 f(x + h/2) + f(x + h))$ error = $O(h^4)$	method of Runge–Kutta (RK4) $k_1 = f(u(x))$ $k_2 = f(u(x) + k_1 h/2)$ $k_3 = f(u(x) + k_2 h/2)$ $k_4 = f(u(x) + k_3 h)$ $u(x + h) \approx u(x) + \frac{h}{6} (k_1 + 2 k_2 + 2 k_3 + k_4)$ local error = $O(h^5)$

Table 3.8: Comparing integration and solving ODEs

### Codes for Runge–Kutta, Heun and Euler with fixed step size

An ODE usually has to be solved on a given time interval. To apply the Runge–Kutta approach with a fixed step size use the code `ode_RungeKutta.m` in Figure 3.35. The function takes several arguments:

- `FunFcn`: a string with the function name for the RHS of the ODE.
- `t`: a vector of scalar time values at which the solution is returned. `t(1)` is the initial time.
- `y0`: the initial values.
- `steps`: the number of Runge–Kutta steps to be taken between the output times.

The function will return the output times in `Tout` and the values of the solution in `Yout`.

The code has the following structure:

1. name of the function, declaration of the parameters
2. documentation
3. initialization
4. main loop
  - (a) determine length of steps  $h$
  - (b) apply the correct number of Runge–Kutta steps
  - (c) save the result
5. return the result

Very similar codes `ode_Euler.m` and `ode_Heun.m` implement the methods of Euler and Heun with fixed step size. The usage is illustrated by the following equation of a simple pendulum.

**3-54 Example :** The second order ODE

$$\frac{d^2}{dt^2} y(t) = -k \sin(y(t))$$

describes the angle  $y(t)$  of a pendulum, possibly with large angles, since the approximation  $\sin(y) \approx y$  is not used. This second order ODE is transformed to a system of order 1 by

$$\frac{d}{dt} \begin{pmatrix} y(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ -k \sin(y(t)) \end{pmatrix}.$$

This ODE leads to a function `pend()`, which is then used by `ode_RungeKutta()` to generate the solution for times  $[0, 30]$  for different initial angles.

#### Pendulum.m

```
%> demo file to solve a pendulum equation
Tend = 30;
%> on Matlab put the definition of the function in a separate file pend.m
function y = pend(t,x)
    k = 1;
    y = [x(2);-k*sin(x(1))];
end%function

y0 = [0.1;0];           % small angle
% y0=[pi/2;0];          % large angle
% y0 = [pi-0.01;0]; % very large angle

t = linspace(0,Tend,100);
[t,y] = ode_RungeKutta('pend',t,y0,10); % Runge–Kutta
% [t,y] = ode_Euler('pend',t,y0,10); % Euler
% [t,y] = ode_Heun('pend',t,y0,10); % Heun

figure(1); plot(t,180/pi*y(:,1))
xlabel('time'); ylabel('angle [Deg]')

```



**ode\_RungeKutta.m**

```

function [tout, yout] = ode_RungeKutta(Fun, t, y0, steps)
% [Tout, Yout] = ode_RungeKutta(fun, t, y0, steps)
%
%      Integrate a system of ordinary differential equations using
%      4th order Runge-Kutta formula.
%
% INPUT:
% Fun    - String containing name of user-supplied problem description.
%          Call: yprime = Fun(t,y)
%          T      - Vector of times (scalar).
%          y      - Solution column-vector.
%          yprime - Returned derivative column-vector; yprime(i) = dy(i)/dt.
% T(1)   - Initial value of t.
% y0     - Initial value column-vector.
% steps  - steps to take between given output times
%
% OUTPUT:
% Tout   - Returned integration time points (column-vector).
% Yout   - Returned solution, one solution column-vector per tout-value.
%
% The result can be displayed by: plot(tout, yout) .

% Initialization
y = y0(:); yout = y'; tout = t(:);

% The main loop
for i = 2:length(t)
    h = (t(i)-t(i-1))/steps;
    tau = t(i-1);
    for j = 1:steps
        % Compute the slopes
        s1 = feval(Fun, tau, y);           s1 = s1(:);
        s2 = feval(Fun, tau+h/2, y+h*s1/2); s2 = s2(:);
        s3 = feval(Fun, tau+h/2, y+h*s2/2); s3 = s3(:);
        s4 = feval(Fun, tau+h, y+h*s3);    s4 = s4(:);
        tau = tau + h;
        y = y + h*(s1 + 2*s2+ 2*s3 + s4)/6;
    end%for
    yout = [yout; y.'];
end%for

```

Figure 3.35: Code for Runge–Kutta with fixed step size

### 3.4.3 Stability of the Algorithms

Examine the ODE

$$\frac{d}{dt} y(t) = \lambda y(t)$$

with the exact solution  $y(t) = y(0) \exp(\lambda t)$ . For  $\lambda < 0$  this solution converges to zero as  $t \rightarrow \infty$ . Thus numerical algorithms are expected to have this feature too.

Based in Section 3.2.5 and in particular Result 3–29 the stability behavior for the elementary ODE  $\frac{d}{dt} y(t) = \lambda y(t)$  is also valid for systems of linear ODEs. Using linearization most of the results remain valid for nonlinear systems of ODEs.

**3–55 Definition :** A numerical method to solve ODEs is called **stable** iff for  $\operatorname{Re}(\lambda) < 0$  the numerical solution of  $\frac{d}{dt} y(t) = \lambda y(t)$  satisfies

$$\lim_{t \rightarrow \infty} y(t) = 0 .$$

When using approximation methods to solve  $\frac{d}{dt} y(t) = \lambda y(t)$  one arrives in many cases at an iteration formula of the type

$$y_{i+1} = y(t_i + h) = g(\lambda h) y_i ,$$

i.e. at each time step the current value of the solution  $y(t_i)$  is multiplied by  $g(\lambda h)$  to arrive at  $y(t_i + h) = y(t_{i+1})$ . In this case  $\lim_{t \rightarrow \infty} y(t) = 0$  is equivalent to  $\lim_{n \rightarrow \infty} (g(\lambda h))^n = 0$ , which is equivalent to  $|g(\lambda h)| < 1$ . The stability can be formulated using the stability function  $g(z)$  for  $z \in \mathbb{C}$ .

- The computational scheme is stable in the domain in  $\mathbb{C}$  where  $|g(\lambda h)| < 1$ .
- A computational scheme is called **A-stable** iff

$$|g(z)| \leq 1 \quad \text{for all } \{z \in \mathbb{C} : \operatorname{Re}(z) < 0\} .$$

- A computational scheme is called **L-stable** iff it is A-stable and in addition

$$\lim_{z \rightarrow -\infty} g(z) = 0 \quad \text{as } z \longrightarrow -\infty .$$

Based on Result 3–29 the stability carries over to linear systems of ODEs and by linearization to nonlinear systems.

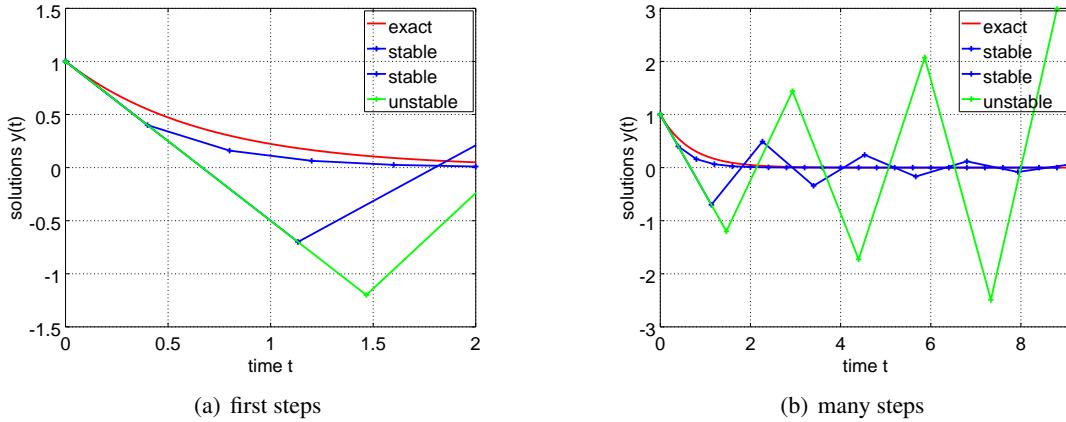


Figure 3.36: Conditional stability of Euler's approximation to  $\frac{d}{dt} y(t) = \lambda y(t)$  with  $\lambda < 0$

### 3–56 Result : Conditional stability of Euler's explicit method

When using Euler's forward difference method to solve  $\frac{dy}{dt} = \lambda y(t)$  with  $y_i = y(t_i)$  and  $y_{i+1} = y(t_i + h)$  find

$$\frac{d}{dt} y(t_i) - \lambda y(t_i) \approx \frac{y_{i+1} - y_i}{h} - \lambda y_i = 0.$$

The differential equation is replaced by the difference equation

$$\frac{y_{i+1} - y_i}{h} = \lambda y_i \quad \implies \quad y_{i+1} = y_i + h \lambda y_i = (1 + h \lambda) y_i .$$

Verify that this difference equation is solved by

$$y_i = y_0 (1 + h \lambda)^i.$$

For this expression to remain bounded independent on  $i$  the condition  $|1 + h \lambda| \leq 1$  is necessary. For complex values of  $\lambda$  this condition is satisfied if  $z = \lambda h$  is inside a circle of radius 1 with center at  $-1 \in \mathbb{C}$ . This domain is visualized in Figure 3.38. For real, negative values of  $\lambda$  the condition simplifies to

$$h|\lambda| < 2 \quad \iff \quad h < \frac{2}{|\lambda|}.$$

This is an example of **conditional stability**, i.e. the schema is only stable if the above condition on the step size  $h$  is satisfied. To visualize the behavior examine the results in Figure 3.36 for solutions of the differential equation  $\frac{dy}{dt} = \lambda y(t)$ .

- At the starting point the differential equation determines the slope of the straight line approximation of the solution. The slope is independent on the length of the step size  $h$ .
  - If the step size is small enough then the numerical solution will not overshoot but converge to zero, as expected.
  - If the step size is too large then the numerical solution will overshoot and will move further and further away from zero by each step.

□

**3-57 Result : Unconditional stability the backward Euler method**

When using backward Euler's method to solve  $\frac{d}{dt} y(t) = \lambda y(t)$  with  $y_i = y(t_i)$  and  $y_{i+1} = y(t_i + h)$  find

$$\frac{d}{dt} y(t_{i+1}) - \lambda y(t_{i+1}) \approx \frac{y_{i+1} - y_i}{h} - \lambda y_{i+1} = 0.$$

The differential equation is replaced by the difference equation

$$\frac{y_{i+1} - y_i}{h} = +\lambda y_{i+1} \quad \Rightarrow \quad (1 - h \lambda) y_{i+1} = y_i.$$

One can verify that this difference equation is solved by

$$y_i = y_0 \frac{1}{(1 - h \lambda)^i}.$$

For this expression to remain bounded independent on  $i$  we need  $|1 - h \lambda| > 1$ . For complex values of  $\lambda$  this condition is satisfied if  $z = \lambda h$  is outside a circle of radius 1 with center at  $+1 \in \mathbb{C}$ . This domain is visualized in Figure 3.38. For real, negative values of  $\lambda$  the condition leads to  $1 - h \lambda > +1$ , which is automatically satisfied and we have **unconditional stability**. The method is A-stable and L-stable.

To visualize the behavior we examine the results in Figure 3.37 for solutions of the differential equation  $\frac{d}{dt} y(t) = \lambda y(t)$ .

- The slope of the straight line approximation is determined by the differential equation at the **end point** of the straight line segment. Consequently the slope will depend on the step size  $h$ .
- If the step size is small enough then the numerical solution will not overshoot but converge to zero.
- Even if the step size is large the numerical solution will not overshoot zero, but converge to zero.

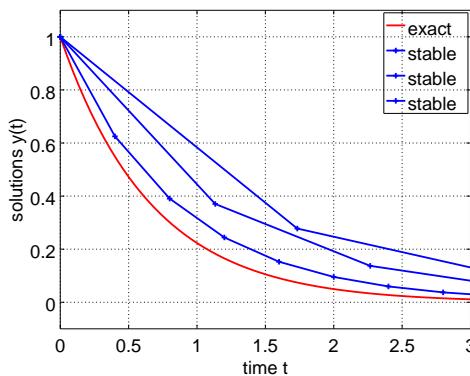


Figure 3.37: Unconditional stability of the implicit approximation to  $\frac{d}{dt} y(t) = \lambda y(t)$  with  $\lambda < 0$

◇

**3–58 Result : Conditional stability for the methods of Heun and Runge–Kutta**

Using one step of the method of Heun solve  $\frac{d}{dt} y(t) = \lambda y(t)$  with  $y_i = y(t_i)$  and  $y_{i+1} = y(t_i + h)$  leads to

$$\begin{aligned} k_1 &= \lambda y_i \\ k_2 &= \lambda(y_i + h \lambda y_i) = (\lambda + h \lambda^2) y(0) \\ k &= \frac{1}{2}(k_1 + k_2) = \frac{1}{2}(2\lambda + h \lambda^2) y_i \\ y_{i+1} &= y(0) + h k = (1 + h \lambda + \frac{1}{2} h^2 \lambda^2) y_i \\ y_i &= (1 + h \lambda + \frac{1}{2} h^2 \lambda^2)^i y_0 \end{aligned}$$

For this expression to remain bounded independent on  $i$  we need

$$|1 + h \lambda + \frac{1}{2} h^2 \lambda^2| = |1 + z + \frac{1}{2} z^2| \leq 1 .$$

To examine this set in  $\mathbb{C}$  use  $|\exp(i \alpha)| = 1$  and solve

$$\begin{aligned} e^{i \alpha} &= 1 + z + \frac{1}{2} z^2 \\ 2e^{i \alpha} &= 2 + 2z + z^2 \\ z^2 + 2z + 2 - 2e^{i \alpha} &= 0 \\ z_{1,2} &= -1 \pm \sqrt{-1 + 2e^{i \alpha}} . \end{aligned}$$

This generates an ellipse like curve between  $-2$  and  $0$  on the real axis and  $\pm i\sqrt{3}$  along the imaginary direction, visible in Figure 3.38. Heun's method is stable inside this domain, i.e. this is a conditional stability.

For the Runge–Kutta method the corresponding inequality is<sup>15</sup>

$$|1 + z + \frac{1}{2} z^2 + \frac{1}{6} z^3 + \frac{1}{24} z^4| = |1 + (h \lambda) + \frac{1}{2} (h \lambda)^2 + \frac{1}{6} (h \lambda)^3 + \frac{1}{24} (h \lambda)^4| \leq 1$$

and the domain is displayed in Figure 3.38. The classical Runge–Kutta method is stable inside this domain, i.e. this is a conditional stability.  $\diamond$

---

<sup>15</sup>Use elementary, tedious computations or software for symbolic calculations.

**3–59 Result : Unconditional stability for the method of Crank–Nicolson**

In Section 4.5.5 the method of Crank–Nicolson will be examined. Using one step of this method to solve  $\frac{dy}{dt} = \lambda y(t)$  with  $y_i = y(t_i)$  and  $y_{i+1} = y(t_i + h)$  leads to

$$\begin{aligned}\frac{y_{i+1} - y_i}{h} &= \lambda \frac{y_i + y_{i+1}}{2} \\ y_{i+1} &= \frac{2 + h\lambda}{2 - h\lambda} y_i \\ y_i &= \left( \frac{2 + h\lambda}{2 - h\lambda} \right)^i y_0.\end{aligned}$$

For this expression to remain bounded independent on  $i$  the necessary condition is

$$\left| \frac{2 + h\lambda}{2 - h\lambda} \right| = \left| \frac{2 + z}{2 - z} \right| \leq 1 \quad \iff \quad |2 + z| \leq |2 - z|.$$

Examine the two points  $2 \pm z = 2 \pm h\lambda \in \mathbb{C}$ :

- For  $\operatorname{Re} \lambda < 0$  the two points have the same size imaginary part and  $|\operatorname{Re}(2 + h\lambda)|$  is smaller than  $|\operatorname{Re}(2 - h\lambda)|$ . Thus the method is stable in the left half plane  $\operatorname{Re} \lambda < 0$ .
- For  $\operatorname{Re} \lambda = 0$  observe  $|2 + h\lambda| = |2 - h\lambda|$ , i.e. the method is stable.
- For  $\operatorname{Re} \lambda > 0$  the method is unstable, as should be. The exact solution  $\exp(\lambda t)$  grows exponentially for  $t > 0$ .

The domain of stability is displayed in Figure 3.38. The method is unconditionally A-stable, but not L-stable, since  $\lim_{\operatorname{Re} z \rightarrow -\infty} |g(z)| = 1$ .  $\diamond$

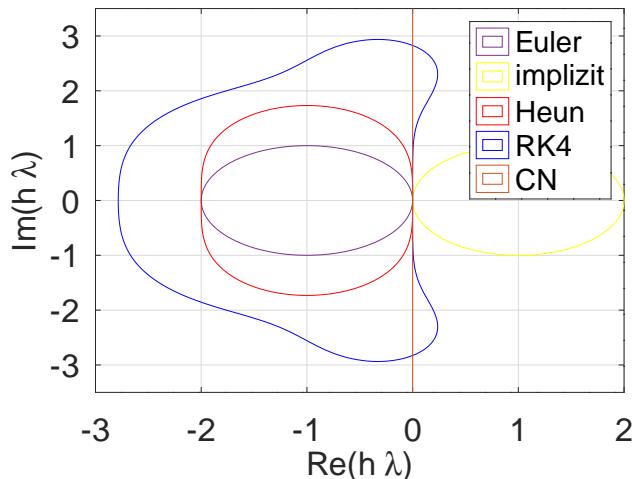


Figure 3.38: Domains of stability in  $\mathbb{C}$  for a few algorithms

### 3–60 Result : Domains of stability for a few algorithms

The domains of stability of the above methods to solve ODEs is visible in Figure 3.38.

- Euler: stable in a circle with radius 1 and center at  $-1 + i0 \in \mathbb{C}$ . For  $\operatorname{Re}(\lambda) < 0$  and large time steps  $h$  the method is unstable, i.e **conditional stability**.
- implicit: stable in **outside** a circle with radius 1 and center at  $+1 + i0 \in \mathbb{C}$ . For  $\operatorname{Re}(\lambda) < 0$  the method is stable for any time step  $h > 0$ , i.e. **unconditional stability**.
- Heun: stable in an ellipse like domain with real parts larger than  $-2$ . For  $\operatorname{Re}(\lambda) < 0$  and large time steps  $h$  the method is unstable, i.e **conditional stability**.
- RK4: stable in an odd shaped domain with real parts larger than  $-2.8$ . For  $\operatorname{Re}(\lambda) < 0$  and large time steps  $h$  the method is unstable, i.e **conditional stability**.
- CN: Crank–Nicolson, the method is stable in the complex half plane  $\operatorname{Re}(z) < 0$ , i.e **unconditional stability**.

◇

#### 3.4.4 General Runge–Kutta Methods, Represented by Butcher Tables

##### Explicit Runge–Kutta schemes

An explicit Runge–Kutta method with  $s$  stages is given by

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + c_2 h, y_n + h(a_{21}k_1)) \\ k_3 &= f(t_n + c_3 h, y_n + h(a_{31}k_1 + a_{32}k_2)) \\ k_4 &= f(t_n + c_4 h, y_n + h(a_{41}k_1 + a_{42}k_2 + a_{43}k_3)) \\ &\vdots \\ k_s &= f(t_n + c_s h, y_n + h(a_{s1}k_1 + a_{s2}k_2 + \dots + a_{s,s-1}k_{s-1})) \\ y_{n+1} &= y_n + h \sum_{i=1}^s b_i k_i \end{aligned}$$

The computational scheme is conveniently represented by a Butcher table.

0						
$c_2$	$a_{21}$					
$c_3$	$a_{31} \quad a_{32}$					
$c_4$	$a_{41} \quad a_{42} \quad a_{43}$					
$\vdots$	$\vdots \quad \ddots$					
$c_s$	$a_{s1} \quad a_{s2} \quad a_{s3} \quad \dots \quad a_{s,s-1}$					
$y_{n+1}$	$b_1$	$b_2$	$b_3$	$\dots$	$b_{s-1}$	$b_s$

(3.14)

Working from top to bottom for the general, explicit Runge–Kutta scheme one never has to solve an equation, just evaluate the function  $f(t, y)$  and plug in. Thus this is an explicit scheme. One step of length  $h$  requires  $s$  evaluations of  $f(t, y)$ .

**3–61 Example : Butcher tables for Heun and the classical Runge–Kutta**

The method of Heun is a 2 stage Runge–Kutta method of order 2 and since

$$k_1 = f(t_n, y_n) \quad , \quad k_2 = f(t_n + h, y_n + h k_1) \quad , \quad y_{n+1} = y_n + \frac{1}{2} k_1 + \frac{1}{2} k_2$$

its Butcher table is given by

$$\begin{array}{c|cc} 0 & & \\ \hline 1 & 1 & \\ \hline y_{n+1} & \frac{1}{2} & \frac{1}{2} \end{array} . \quad (3.15)$$

The classical Runge–Kutta method of order 4 is a 4 stage method with the Butcher table

$$\begin{array}{c|cccc} 0 & & & & \\ \hline \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ \hline 1 & 0 & 0 & 1 & \\ \hline y_{n+1} & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array} . \quad (3.16)$$

◊

**Implicit Runge–Kutta schemes**

A general implicit Runge–Kutta method with  $s$  stages is given by

$$\begin{aligned} k_1 &= f(t_n + c_1 h, y_n + h (a_{1,1} k_1 + a_{1,2} k_2 + \dots + a_{1,s} k_s)) \\ k_2 &= f(t_n + c_2 h, y_n + h (a_{2,1} k_1 + a_{2,2} k_2 + \dots + a_{2,s} k_s)) \\ k_3 &= f(t_n + c_3 h, y_n + h (a_{3,1} k_1 + a_{3,2} k_2 + \dots + a_{3,s} k_s)) \\ &\vdots \quad \vdots \\ k_s &= f(t_n + c_s h, y_n + h (a_{s,1} k_1 + a_{s,2} k_2 + \dots + a_{s,s} k_s)) \\ y_{n+1} &= y_n + h \sum_{i=1}^s b_i k_i \end{aligned}$$

Observe that this leads to a system of equation for the slopes  $\vec{k} \in \mathbb{R}^s$ , thus it is an implicit scheme. For nonlinear functions  $f(t, y)$  it is a nonlinear system of equations. For a system of  $n$  equation with  $\vec{y} \in \mathbb{R}^n$  it leads to a nonlinear system of  $n \cdot s$  equations for  $n \cdot s$  unknowns. Since Newton's algorithm is used to solve the system it is a good idea to provide the Jacobian matrix  $\mathbf{J}$  to the algorithms. Use Example 3–69 on how to use the Jacobian matrix  $\mathbf{J}$ .

$$\mathbf{J} = \frac{\partial f(t, \vec{y})}{\partial \vec{y}} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \dots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \dots & \frac{\partial f_2}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \dots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}$$

An implicit Runge–Kutta scheme is also represented by a Butcher table.

$$\begin{array}{c|ccccc}
 c_1 & a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,s} \\
 c_2 & a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,s} \\
 c_3 & a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,s} \\
 \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
 c_s & a_{s,1} & a_{s,2} & a_{s,3} & \cdots & a_{s,s} \\
 \hline
 y_{n+1} & b_1 & b_2 & b_3 & \cdots & b_s
 \end{array} \iff \begin{array}{c|c}
 \vec{c} & \mathbf{A} \\
 \hline
 y_{n+1} & \vec{b}^T
 \end{array} \quad (3.17)$$

### 3–62 Example : Stability analysis for an implicit scheme

For the linear ODE  $\frac{d}{dt} y(t) = \lambda y(t)$  the system of equations for the slopes  $\vec{k}$  is

$$\vec{k} = \lambda (y_n \vec{1} + h \mathbf{A} \vec{k}) \iff (\mathbb{I} - \lambda h \mathbf{A}) \vec{k} = \lambda y_n \vec{1}.$$

Thus the next time step is given by

$$y_{n+1} = y_n + h \langle \vec{b}, \vec{k} \rangle = y_n + y_n \lambda h \langle \vec{b}, (\mathbb{I} - \lambda h \mathbf{A})^{-1} \vec{1} \rangle = y_n \left( 1 + \lambda h \langle \vec{b}, (\mathbb{I} - \lambda h \mathbf{A})^{-1} \vec{1} \rangle \right).$$

The essential information is given by the stability function

$$g(z) = 1 + z \langle \vec{b}, (\mathbb{I} - z \mathbf{A})^{-1} \vec{1} \rangle = \frac{\det(\mathbb{I} - z \mathbf{A} + z \vec{1} \vec{b}^T)}{\det(\mathbb{I} - z \mathbf{A})} \quad (3.18)$$

where  $z = \lambda h$ . For a proof see [Butc03, Lemma 351A, p. 230]. The stability condition is  $|g(z)| < 1$ . This stability function  $g(z)$  is a rational function where numerator and denominator are polynomials of degree  $s$ .

◇

For an explicit scheme the matrix of coefficients has a lower triangular form

$$\mathbf{A} = \begin{bmatrix} 0 & & & & \\ a_{21} & 0 & & & \\ a_{31} & a_{32} & 0 & & \\ \vdots & & & \ddots & \\ a_{s,1} & a_{s,2} & \cdots & a_{s,s-1} & 0 \end{bmatrix}$$

and thus  $\det(\mathbb{I} - z \mathbf{A}) = 1$  and the stability function  $g(z)$  is a polynomial of degree  $s$ . An explicit scheme of this type is conditionally stable, never unconditionally stable.

### 3–63 Example : Butcher table for the implicit Euler and Crank–Nicolson method

The implicit Euler method is given by

$$\begin{aligned}
 k_1 &= f(t_n + h, y_n + h k_1) \\
 y_{n+1} &= y_n + h k_1 = y_n + h f(t_n + h, y_n + h k_1) \\
 \frac{y_{n+1} - y_n}{h} &= f(t_n, y_{n+1})
 \end{aligned}$$

Thus the Butcher table is

$$\begin{array}{c|c}
 1 & 1 \\
 \hline
 y_{n+1} & 1
 \end{array}$$

and the stability function is

$$g(z) = 1 + z \langle \vec{b}, (\mathbb{I} - z \mathbf{A})^{-1} \vec{1} \rangle = 1 + z \frac{1}{1-z} = \frac{1}{1-z}.$$

As a consequence the implicit Euler Method is stable for ODEs  $\frac{d}{dt} y(t) = \lambda y(t)$  with  $\operatorname{Re} \lambda < 0$ , i.e **unconditional stability**.

The Butcher table for the Crank–Nicolson method is

0	0	0
1	$\frac{1}{2}$	$\frac{1}{2}$
$y_{n+1}$	$\frac{1}{2}$	$\frac{1}{2}$

and the method is given by

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + h, y_n + h(\frac{1}{2}k_1 + \frac{1}{2}k_2)) \\ y_{n+1} &= y_n + h \left( \frac{1}{2}k_1 + \frac{1}{2}k_2 \right) \\ &= y_n + \frac{h}{2} \left( f(t_n, y_n) + f(t_n + h, y_n + h(\frac{1}{2}k_1 + \frac{1}{2}k_2)) \right) \\ &= y_n + \frac{h}{2} (f(t_n, y_n) + f(t_n + h, y_{n+1})) \\ \frac{y_{n+1} - y_n}{h} &= \frac{1}{2} (f(t_n, y_n) + f(t_n + h, y_{n+1})). \end{aligned}$$

This is caused by the identical rows in the Butcher table. If  $c_j = 1$  and  $a_{j,i} = b_i$  for  $i = 1, 2, \dots, s$  then

$$\begin{aligned} y_{n+1} &= y_n + h \sum_{i=1}^s b_i k_i \\ k_j &= f(t_n + 1h, y_n + h \sum_{i=1}^s a_{j,i} k_i) = f(t_n + h, y_{n+1}) \end{aligned}$$

The stability function is

$$\begin{aligned} g(z) &= 1 + z \langle \vec{b}, (\mathbb{I} - z \mathbf{A})^{-1} \vec{1} \rangle = 1 + z \langle \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}, \begin{bmatrix} 1 & 0 \\ -\frac{z}{2} & 1 - \frac{z}{2} \end{bmatrix}^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \rangle \\ &= 1 + \frac{z}{2} \langle \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \frac{1}{1 - \frac{z}{2}} \begin{bmatrix} 1 - \frac{z}{2} & 0 \\ +\frac{z}{2} & 1 \end{bmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \rangle \\ &= 1 + \frac{z}{2 - z} \langle \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 - \frac{z}{2} \\ 1 + \frac{z}{2} \end{pmatrix} \rangle = 1 + \frac{z}{2 - z} 2 = \frac{2 + z}{2 - z}. \end{aligned}$$

As a consequence the Crank–Nicolson Method is stable for ODEs  $\frac{d}{dt} y(t) = \lambda y(t)$  with  $\operatorname{Re} \lambda < 0$ , i.e find **unconditional stability**.  $\diamond$

### Embedded Runge–Kutta schemes

The embedded methods are designed to produce an estimate of the local truncation error of a single Runge–Kutta step, and as result, allow to control the error with adaptive stepsize. This is done by having two

methods in one single table, one method of order  $p$  and one of order  $p - 1$ . The higher order approximation is given by

$$y_{n+1} = y_n + h \sum_{i=1}^s b_i k_i$$

and the lower order approximation by

$$y_{n+1}^* = y_n + h \sum_{i=1}^s b_i^* k_i .$$

This is represented by an extended Butcher table.

$c_1$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$\cdots$	$a_{1,s}$
$c_2$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$\cdots$	$a_{2,s}$
$c_3$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$\cdots$	$a_{3,s}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$c_s$	$a_{s,1}$	$a_{s,2}$	$a_{s,3}$	$\cdots$	$a_{s,s}$
$y_{n+1}$	$b_1$	$b_2$	$b_3$	$\cdots$	$b_s$
$y_{n+1}^*$	$b_1^*$	$b_2^*$	$b_3^*$	$\cdots$	$b_s^*$

The error is then estimated by

$$y_{n+1} - y_{n+1}^* = h \sum_{i=1}^s (b_i - b_i^*) k_i$$

and used to possibly adapt the step size  $h$ . The key point of an embedded method is to use the same slopes  $k_i$  to generate the estimates of order  $p$  and  $p - 1$ . This uses fewer evaluations of the RHS  $f(t, y)$  than performing one step of size  $h$  and two of size  $h/2$  and then compare.

### 3–64 Example : Butcher table for the Bogacki–Shampine method in `ode23.m`

This explicit method is used in MATLAB/Octave by the command `ode23()`. The name indicates that the result is based on approximations of orders 2 and 3. The Butcher table is

$0$				
$\frac{1}{2}$		$\frac{1}{2}$		
$\frac{3}{4}$	0	$\frac{3}{4}$		
1	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{4}{9}$	
$y_{n+1}$	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{4}{9}$	0
$y_{n+1}^*$	$\frac{7}{24}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{8}$

and one embedded step requires 4 evaluations of  $f(t, y)$ . Observe that the last row of the coefficient matrix  $\mathbf{A}$  and the vector  $\vec{b}^T$  coincide, thus

$$k_4 = f(t_n + h, y_n + h \sum_{i=1}^3 a_{4,i} k_i) , \quad y_{n+1} = y_n + h \sum_{i=1}^3 a_{4,i} k_i$$

and  $k_4$  can be used during the next step as "new"  $k_1$ . Consequently only 3 evaluations of  $f(t, y)$  are required.

For Octave find the Butcher table for the command `ode23()` in the file `runge_kutta_23.m` and for MATLAB in the file `ode23.m`.  $\diamond$

**3–65 Example : Butcher table for the Dormand–Prince method in `ode45.m`**

This explicit method is used in MATLAB/Octave by the command `ode45()`. The name indicates that the result is based on approximations of orders 4 and 5. The Butcher table is ([HairNorsWann08, p. 178], [Butc03, p. 211])

0						
$\frac{1}{5}$						
$\frac{3}{10}$						
$\frac{4}{5}$						
$\frac{8}{9}$						
1						
1						
$y_{n+1}$						
$y_{n+1}^*$						

(3.21)

and one embedded step requires 7 evaluations of  $f(t, y)$ . Observe that the last row of the coefficient matrix  $\mathbf{A}$  and the vector  $\vec{b}^T$  coincide, thus

$$k_7 = f(t_n + h, y_n + h \sum_{i=1}^6 a_{7,i} k_i) \quad , \quad y_{n+1} = y_n + h \sum_{i=1}^6 a_{7,i} k_i$$

and  $k_7$  can be used during the next step as "new"  $k_1$ . Consequently only 6 evaluations of  $f(t, y)$  are required.

For Octave find the Butcher table for the command `ode45()` in the file `runge_kutta_45_doprri.m` and for MATLAB in the file `ode45.m`.  $\diamond$

### 3.4.5 Adaptive Step Sizes and Extrapolation

One of the critical points when using numerical tools to approximate solutions of ODEs is a good choice of the step size  $h$ :

- for  $h$  too small the computational effort might be to large, i.e. it takes too long or the rounding errors caused by the arithmetic on the CPU lead to numerical problems.
- for  $h$  too large the discretization errors will be too large.

A nice way out would be if the algorithms could determine a "good" step size. Fortunately this is possible in most cases. Here one approach of automatic step size control is presented: Compute the solution at  $t + h$  twice, once with one step of length  $h$  and the two steps of length  $h/2$ . Use the difference of the two results to estimate the discretization error and then adapt the step size  $h$  accordingly.

Use Table 3.6 to estimate the error when stepping from  $t$  with  $y(t)$  to  $t + h$  with solution  $y(t + h)$ , one with one step of size  $h$  leading to the first result  $r_1$ , then with two steps of size  $h/2$  leading to the second result  $r_2$ . Then use the approximations

$$\begin{aligned} y(t + h) - r_1 &\approx C h^{p+1} \\ y(t + 2 \cdot \frac{h}{2}) - r_2 &\approx 2 C (\frac{h}{2})^{p+1} \end{aligned}$$

as exact equations (not completely true, but good enough) and solve for  $y(t + h)$  and  $C$ . Subtraction of the above two expressions leads to

$$r_2 - r_1 = C \left( h^{p+1} - 2 \left( \frac{h}{2} \right)^{p+1} \right) = C \frac{2^p - 1}{2^p} h^{p+1} .$$

This implies

$$\begin{aligned} Ch^{p+1} &= \frac{2^p}{2^p - 1} (r_2 - r_1) \\ y(t+h) &= r_1 + \frac{2^p}{2^p - 1} (r_2 - r_1) = \frac{2^p r_2 - r_1}{2^p - 1} \end{aligned} \quad (3.22)$$

and equation (3.22) leads to two useful results:

1. **control of step size  $h$ :** (3.22) contains a (often very good) estimate of the local discretization error at time  $t$ :

$$y(t+h) - r_1 \approx \frac{2^p}{2^p - 1} (r_2 - r_1).$$

This deviation should not be larger than a given bound  $0 < \varepsilon$ , typically very small. Consider three outcomes:

- If the estimated error

$$\frac{2^p}{2^p - 1} |r_2 - r_1| \geq \varepsilon$$

is too large, then restart at  $t$  with a smaller step size  $h$ .

- If the estimated error is just about right, go on with the same step size and move from  $t+h$  to  $t+2h$ .
- If the estimated error is "considerably too small", advance by one step and for the next step choose a large step size  $h$ .

The choice of the error bound  $\varepsilon$  and the way to adapt the step size have to be made very carefully.

2. **local extrapolation:** using  $r_1$  and  $r_2$  generate a new, better approximation for the solution at  $t+h$  by

$$y(t+h) = \frac{2^p r_2 - r_1}{2^p - 1}$$

This "new" method will have a local discretization error proportional to  $h^{p+2}$ , i.e. the order of convergence is improved by 1 .

The above error estimates and extrapolation method assume that the Taylor approximation of the ODE is correct, which requires the RHS function  $f(t, y)$  to be many times differentiable. If the function  $f(t, y)$  is smooth (e.g. jumps of the values or of derivatives), then the above estimates are not correct. This will cause the adaptive step size approaches in Section 3.4.5 to try step sizes smaller and smaller and the algorithm might come to a screeching halt, usually with a warning message. Higher order methods (e.g. `ode45()`) are more susceptible to this problem than lower order methods (e.g. `ode23()`). Another reason for the algorithms to stop could be blowup of the solution. As example consider the ODE  $\frac{dy}{dt} = 1 + y(t)^2$ , solved by  $y(t) = \tan(t)$ , which blows up at  $t = \frac{\pi}{2}$ .

Using Table 3.6 for Runge–Kutta shows  $p = 4$  and thus

$$\begin{aligned} y(t+h) - r_1 &\approx \frac{16}{15} (r_2 - r_1) \approx r_2 - r_1 \\ y(t+h) &\approx \frac{16 r_2 - r_1}{15} \end{aligned}$$

The local discretization error is of the form  $O(h^{p+1}) = O(h^5)$ , i.e. by using  $h/2$  instead of  $h$  the error is expected to be divided by  $2^5 = 32$ . As a consequence the step size will be modified by factors closer to 1, e.g. by  $0.8h$  for smaller steps and by  $1.2h$  for larger steps.

The above idea based on the Runge–Kutta method is implemented in a code `rk45.m`. The name indicates the order of convergence is 4, improved to 5 by the extrapolation. In the function `rk23.m` the idea is implemented based on the method of Heun, which is also called a Runge–Kutta method of order 2. In both codes an absolute and relative tolerance for the error can be specified.

The algorithm in `ode45.m` does not use the above mentioned classical Runge–Kutta with half step size, where one step requires 11 evaluations of the function  $f(t, y)$ . Instead the Dormand–Prince embedded method in Example 3–65 is used. It requires only 7 evaluations of  $f(t, y)$  whith the error also proportional to  $h^5$ .

**3–66 Example :** As an illustrative example examine the ODE

$$\ddot{y}(t) = -y^3(t) \quad \text{with} \quad y(0) = 0, \quad \dot{y}(0) = 1$$

on the interval  $[0, 3\pi]$ . Asking for the same relative and absolute tolerance ( $10^{-2}$  or  $10^{-6}$ ) with the two adaptive methods based on Heun and Runge–Kutta leads to the results in Table 3.9 and Figure 3.39. The Runge–Kutta based approach uses clearly fewer time steps. Figure 3.39 also shows that in section of the solution with a large curvature, a smaller time step is used. The results for the tolerance of  $10^{-6}$  clearly show that the difference is more significant for higher accuracy results.

	Heun	Runge–Kutta	Heun	Runge–Kutta
absolute and relative tolerance		$10^{-2}$		$10^{-6}$
globale order of convergence	$h^2$	$h^4$	$h^2$	$h^4$
number of time steps	213	34	10235	274
number of function calls	1160	440	51400	3487

Table 3.9: Comparison of a Heun based method (`rk23.m`) and Runge–Kutta method (`rk45.m`)

Observe that the number of function evaluations is considerably higher than the number of time steps.

- Each time step consists of one step of length  $h$  and two steps of length  $h/2$ . For Runge–Kutta this leads to 11 function evaluations for each time step and for Heun to 5 evaluations. Observe that the first evaluation is share between the two computations.
- If a step size is rejected and the computation redone with a shorter time step, some of the evaluations are "thrown away".

◇

To illustrate the results in this section a few MATLAB/Octave codes are used, see Table 3.19 on page 250.

### 3.4.6 ODE solvers in MATLAB/Octave

Most of the ODE solvers in Octave and MATLAB follow a very similar syntax. Thus it is very easy to switch the solvers and find the one most suitable for your problem. In the next subsection four of the available algorithms will be described with more details.

- Octave 6.2.0: `ode15i`, `ode15s`, `ode23`, `ode23s`, `ode45`
- Matlab R2019a: `ode113`, `ode15i`, `ode15s`, `ode23`, `ode23s`, `ode23t`, `ode23tb`, `ode45`

The goal of this subsection is to illustrate the application of the commands `ode??` to a single ODE or systems of ODEs. The usage of the options is explained.

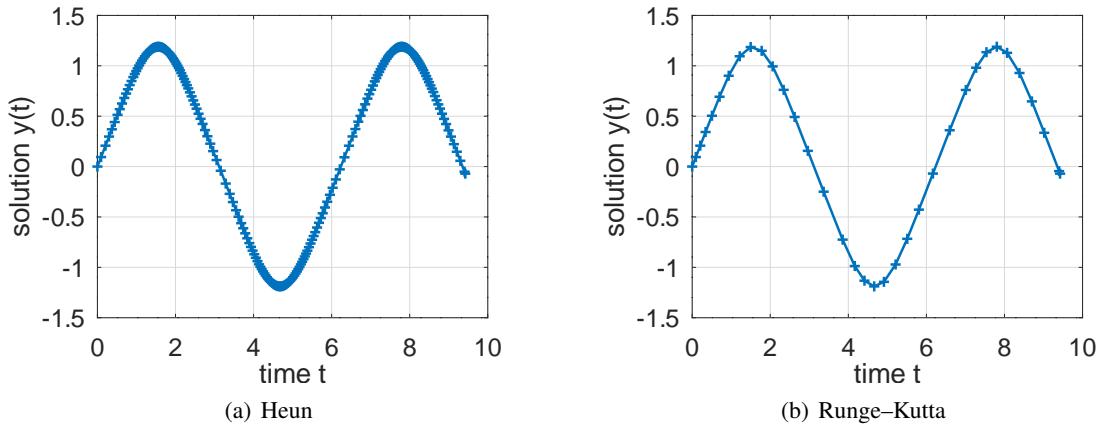


Figure 3.39: Graphical results for a Heun based method and Runge–Kutta method, with tolerances  $10^{-2}$

## The basic usage of `ode??`

Typing the command `help ode45` in Octave generates on the first few lines the calling options for `ode45()`, but all other `ode??()` commands are very similar, as are the MATLAB commands.

```
help ode45
-->
-- [T, Y] = ode45 (FUN, TSPAN, Y0, ODE_OPT)
-- [T, Y] = ode45 (FUN, TSPAN, Y0, ODE_OPT)
-- [T, Y, TE, YE, IE] = ode45 (...)
-- SOLUTION = ode45 (...)
-- ode45 (...)
```

The input arguments have to be given by the user. A call of the function `ode45()` will return results.

- input arguments

`FUN` is a function handle, inline function, or string containing the name of the function that defines the ODE:  $\frac{d}{dt} y(t) = f(t, y(t))$  (or  $\frac{d}{dt} \vec{y}(t) = f(t, \vec{y}(t))$ ). The function must accept two inputs, where the first is time  $t$  and the second is a column vector (or scalar) of unknowns  $y$ .

**TRANGE** specifies the time interval over which the ODE will be evaluated. Typically, it is a two-element vector specifying the initial and final times ( $[t_{init}, t_{final}]$ ). If there are more than two elements, then the solution will be evaluated at these intermediate time instances.

Observe that the algorithms `ode??()` will always first choose the intermediate times, using the adapted step sizes, see Section 3.4.5. If `TRANGE` is a two-element vector, these values are returned. If more intermediate times are asked for the algorithm will use a special interpolation algorithm to return the solution at the desired times. Asking for more (maybe many) intermediate times will **not** increase the accuracy. For increased accuracy use the options `RelTol` and `AbsTol`, see page 211.

`INIT` contains the initial value for the unknowns. If it is a row vector then the solution `Y` will be a matrix in which each column is the solution for the corresponding initial value in  $t_{init}$ .

`ODE_OPT` The optional fourth argument `ODE_OPT` specifies non-default options to the ODE solver. It is a structure generated by `odeset()`, see page 211.

- return arguments

- If the function `[T, Y] = ode??()` is called with two return arguments, the first return argument is column vector T with the times at which the solution is returned. The output Y is a matrix in which each column refers to a different unknown of the problem and each row corresponds to a time in T.
- If the function `SOL = ode??()` is called with one return argument, a structure with three fields: `SOL.x` are the times, `SOL.y` is the matrix of solution values and the string `SOL.solver` indicated which solver was used.
- If the function `ode??()` is called with no return arguments, a graphic is generated. Try `ode45(@(t,y)y,[0,1],1)`.
- If using the Events option, then three additional outputs may be returned. `TE` holds the time when an Event function returned a zero. `YE` holds the value of the solution at time `TE`. `IE` contains an index indicating which Event function was triggered in the case of multiple Event functions.

Solving the ODE  $\frac{dy}{dt} = (1-t)y(t)$  for  $0 \leq t \leq 5$  with initial condition  $y(0) = 1$  is a one-liner.

```
[t,y] = ode45(@(t,y) (1-t)*y, [0,5],1);
plot(t,y,'+-')
```

The plot on the left in Figure 3.40 shows that the time steps used by `ode45()` are rather large and thus the solution seems to be inaccurate. The Dormand–Prince method in `ode45()` used large time steps to achieve the desired accuracy and then returned the solution at those times only. It might be better to return the solution at more intermediate times, uniformly spaced. This can be specified in `trange` when calling `ode45()`, see the code below. Find the result on the right in Figure 3.40.

```
[t,y] = ode45(@(t,y) (1-t)*y, [0:0.1:5],1);
plot(t,y,'+-')
```

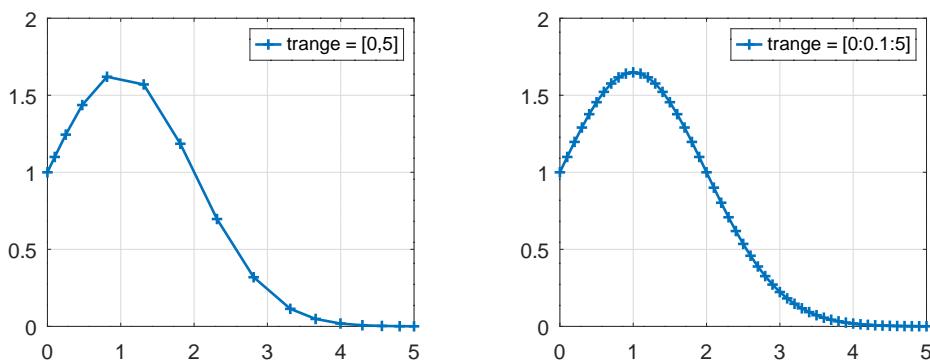


Figure 3.40: Solution of an ODE by `ode45()` at the computed times or at preselected times

### An illustrative example, the SIR model for the spreading of an infection

A pandemic might start with a few infected individuals, but the infection can spread very quickly. One of the mathematical models for this is the SIR model (Susceptible, Infected, Recovered). Find a description at <https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model> and a YouTube video on the SIR model at <https://www.youtube.com/watch?v=NKMHHm2Zbkw>.

This (overly) simple model for the spreading of a virus uses three time dependent variables:

- $S(t)$  = the susceptible fraction of the population
- $I(t)$  = the infected fraction of the population
- $R(t)$  = the recovered fraction of the population

and  $S(t) + I(t) + R(t) = 1$  implies

$$\frac{dS(t)}{dt} + \frac{dI(t)}{dt} + \frac{dR(t)}{dt} = 0.$$

Assuming there are  $N$  individuals in the population, use two parameters to describe the spreading of the virus.

- $b$  = number of contacts per day of an infected individual.  
 $b N I(t)$  new attempted infections, but only the fraction  $S(t)$  is susceptible.
- $k$  = fraction of infected individuals that will recover in one day.  
 $k N I(t)$  individuals will recover in one day.

Interpretation of the parameters:

- The value  $\frac{1}{k}$  can be considered as number of days an individual can spread the virus to new patients.
- Every day an infected individual will make contact to  $b$  other individuals, possibly infecting them with the virus. Only the fraction  $0 < S(t) < 1$  is susceptible, thus  $b S(t)$  will actually be infected by this individual.
- During his sick period an individual will thus infect  $\frac{b}{k} S(t)$  newly infected patients.
- As we have a total of  $N I(t)$  sick individuals we will observe  $b N I(t) S(t)$  new infection every day.

This leads to a system of ODEs for the three ratios  $S(t)$ ,  $I(t)$  and  $R(t)$ .

$$\begin{aligned}\frac{d}{dt} S(t) &= -b S(t) I(t) \\ \frac{d}{dt} I(t) &= -\frac{dS(t)}{dt} - \frac{dR(t)}{dt} = +b S(t) I(t) - k I(t) \\ \frac{d}{dt} R(t) &= +k I(t)\end{aligned}$$

Rewrite this as an ODE for  $I(t)$  and  $R(t)$ , using  $S(t) = 1 - I(t) - R(t)$ .

$$\begin{aligned}\frac{d}{dt} I(t) &= +b(1 - I(t) - R(t)) I(t) - k I(t) = (+b(1 - I(t) - R(t) - k) I(t) \\ &= (+b - k - b I(t) - b R(t)) I(t) \\ \frac{d}{dt} R(t) &= +k I(t)\end{aligned}$$

This system of ODE is solved numerically using e.g `ode45()`. Find the results of the code below in Figure 3.41. The additional code in the file `SIR_Model.m` generates the vector fields in Figure 3.42.

**SIR Model.m**

```
I0 = 1e-4; S0 = 1 - I0; R0 = 0; %% the initial values
b = 1/3; k = 1/10; %% the model parameters
%% b = 1/8; %% use this for a smaller infection rate

%% for MATLAB comment out this function and put it a file SIR.m
function res = SIR(t,I,R,b,k) %%x = IR
res = [(+b-k-b*I-b*R).*I; k*I];
end%function

[t,IR] = ode45(@(t,x)SIR(t,x(1),x(2),b,k),linspace(0,600,601),[I0,R0]);

figure(1); plot(t,IR)
xlabel('time [days]'); ylabel('fraction of Infected and Recovered')
ylim([-0.05 1.05])
legend('infected','recovered', 'location','east')
```

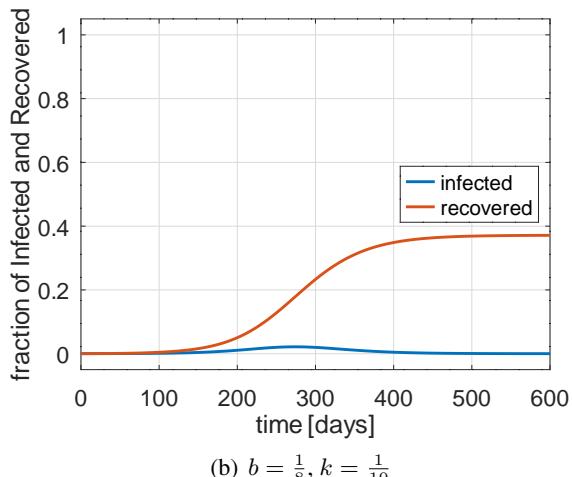
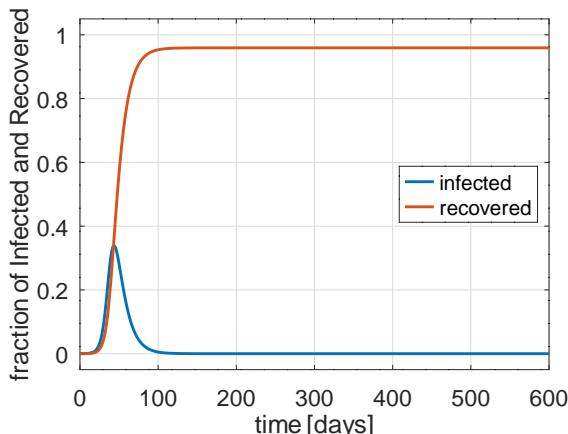


Figure 3.41: SIR model, with infection rate  $b$  and recovery rate  $k$ . The positive effect of a small infection rate  $b$  is obvious.

**Using options for the commands `ode??`**

For these ODE solvers many options can and should be used. The command `odeset()` will generate a list of the available options and their default values. With `help odeset` you obtain more information on these options. The available options differ slightly for Octave and MATLAB. Below find a list of the options, including the default values.

**Octave odeset()**

```
odeset()
-->
List of the most common ODE solver options.
Default values are in square brackets.

AbsTol: scalar or vector, >0, [1e-6]
RelTol: scalar, >0, [1e-3]
BDF: binary, {[ "off"], "on" }
Events: function_handle, []
```

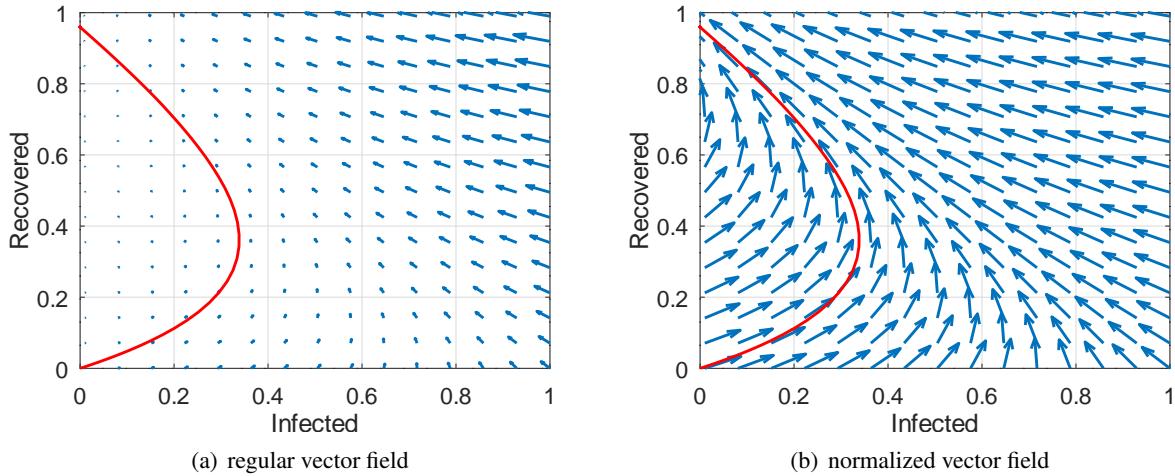


Figure 3.42: SIR model vector field with  $b = \frac{1}{3}$  and  $k = \frac{1}{10}$

```

InitialSlope: vector, []
InitialStep: scalar, >0, []
    Jacobian: matrix or function_handle, []
JConstant: binary, {[ "off"], "on" }
JPattern: sparse matrix, []
    Mass: matrix or function_handle, []
MassSingular: switch, {[ "maybe"], "no", "yes" }
    MaxOrder: switch, {[5], 1, 2, 3, 4, }
    MaxStep: scalar, >0, []
MStateDependence: switch, {[ "weak"], "none", "strong" }
    MvPattern: sparse matrix, []
NonNegative: vector of integers, []
NormControl: binary, {[ "off"], "on" }
    OutputFcn: function_handle, []
    OutputSel: scalar or vector, []
    Refine: scalar, integer, >0, []
    Stats: binary, {[ "off"], "on" }
Vectorized: binary, {[ "off"], "on" }

```

### Matlab odeset()

```
odeset()
-->
    AbsTol: [ positive scalar or vector {1e-6} ]
    RelTol: [ positive scalar {1e-3} ]
    NormControl: [ on | {off} ]
    NonNegative: [ vector of integers ]
    OutputFcn: [ function_handle ]
    OutputSel: [ vector of integers ]
    Refine: [ positive integer ]
    Stats: [ on | {off} ]
    InitialStep: [ positive scalar ]
    MaxStep: [ positive scalar ]
    BDF: [ on | {off} ]
    MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
    Jacobian: [ matrix | function_handle ]
    JPATTERN: [ sparse matrix ]
    Vectorized: [ on | {off} ]
```

```

    Mass: [ matrix | function_handle ]
MStateDependence: [ none | {weak} | strong ]
    MvPattern: [ sparse matrix ]
MassSingular: [ yes | no | {maybe} ]
InitialSlope: [ vector ]
    Events: [ function_handle ]

```

The most frequently used options are `AbsTol` (default value  $10^{-6}$ ) and `RelTol` (default value  $10^{-3}$ ), used to specify the absolute and relative tolerances for the solution. At each time step the algorithm estimates the error( $i$ ) of the  $i$ -th component of the solution. Then the condition

$$|\text{error}(i)| \leq \max\{\text{RelTol} * \text{abs}(y(i)), \text{AbsTol}(i)\}$$

has to be satisfied. Thus at least one of the absolute or relative error has to be satisfied. As a consequence it is rather useless to ask for a very small relative error, but keep the absolute error large. Both have to be made small.

The ODE  $\frac{d}{dt}x(t) = 1 + x(t)^2$  with  $x(0) = 0$  is solved by  $x(t) = \tan(t)$ . Thus we know the exact value of  $x(\pi/4) = 1$ . With the default values obtain

```

[t,x] = ode23(@(t,x)1+x^2,[0,pi/4],0);
Error_Steps_default = [x(end)-1, length(t)-1]
-->
Error_Steps_default = [-3.6322e-05 25]

```

i.e. with 25 Heun steps the error is approximately  $3.6 \cdot 10^{-5}$ . With the options `AbsTol` and `RelTol` the error can be made smaller. The price to pay are more steps, i.e. a higher computational effort.

```

ode_opt = odeset('AbsTol',1e-9,'RelTol',1e-9);
[t,x] = ode23(@(t,x)1+x^2,[0,pi/4],0,ode_opt);
Error_Steps_opt = [x(end)-1 ,length(t)-1]
-->
Error_Steps_opt = [-2.3420e-10    495]

```

With the command `odeget()` one can read out specific options for the ode solvers.

### 3.4.7 Comparison of four Algorithms Available with Octave/MATLAB

In important aspect to consider when selecting a solve for an ODE is **stiffness**. A linear system is considered stiff if the ratio between the smallest and largest eigenvalue is large. Thus the corresponding eigen solutions have a very different time scale. Due to stability problems different algorithms have to be used. As a general rule are explicit methods not suitable for stiff problems, while implicit methods might work. For an introduction use the Wikipedia article [https://en.wikipedia.org/wiki/Stiff\\_equation](https://en.wikipedia.org/wiki/Stiff_equation).

Below four algorithms available on Octave are applied to a few sample problems to illustrate the differences. The results by very MATLAB are similar.

`ode45` : an implementation of a Runge–Kutta (4,5) formula, the Dormand–Prince method of order 4 and 5.  
This algorithm works well on most non-stiff problems and is a good choice as a starting algorithm.

`ode23` : an implementation of an explicit Runge–Kutta (2,3) method, the explicit Bogacki–Shampine method of order 3. It might be more efficient than `ode45` at crude tolerances for moderately stiff problems.

`ode15s` : this command solves stiff ODEs. It uses a variable step, variable order BDF (Backward Differentiation Formula) method that ranges from order 1 to 5. Use `ode15s` if `ode45` fails or is very inefficient.

`ode23s` : this command solves stiff ODEs with a Rosenbrock method of order (2,3). The `ode23s` solver evaluates the Jacobian during each step of the integration, so supplying it with the Jacobian matrix is critical to its reliability and efficiency.

To see the statistics of the different solvers use the option<sup>16</sup> `opt = odeset('Stats','on')`.

### 3–67 Example : The ODE

$$\frac{dy}{dt} = -y(t) + 3 \quad \text{with} \quad y(0) = 0$$

with the exact solution

$$y(t) = 3 - 3 \exp(-t)$$

is an example of a non stiff problem. The solution should not be a problem at all for either algorithm. This is confirmed by the results in Table 3.10. Observe that the algorithm in `ode45` generates the most accurate results, see Figure 3.43. The example was solved by the code below.

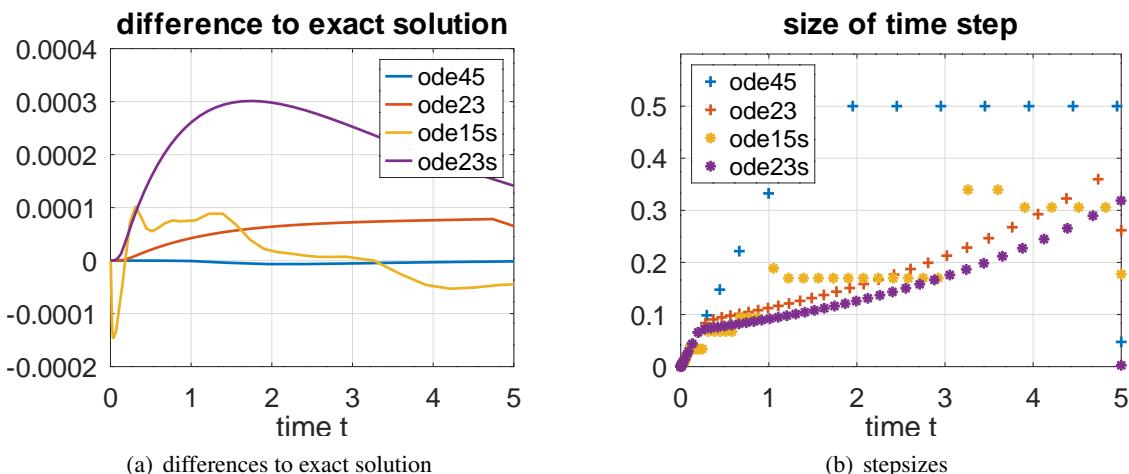


Figure 3.43: Solution of a non–stiff ODE with different algorithms

	ode45	ode23	ode15s	ode23s
Number of steps	30	45	48	52
Number of evaluations	183	138	91	418

Table 3.10: Data for a non–stiff ODE problem with different algorithms

Some of the results in Table 3.10 can be explained.

- For `ode45()` the number of evaluations is barely more than 6 times the number of time steps. This is consistent with the information in Example 3–65. Thus the algorithm never had to go to smaller time steps  $h$ . The time step is increased, until it reaches to upper limit<sup>17</sup> `MaxStep` of  $h = 0.5$ .

<sup>16</sup>This author generated the numbers by using a counter within the function, and obtained slightly different numbers.

<sup>17</sup>Examine the source file `odedefaults.m` to confirm this default value. The option `MaxStep` can be modified.

- For `ode23()` the number of evaluations is barely more than 3 times the number of time steps. This is consistent with the information in Example 3–64. Thus the algorithm never had to go to smaller time steps  $h$ . In Figure 3.43 observe that the step size is strictly increasing for `ode23()`.

```

opt = odeset ("RelTol", 1e-8, "AbsTol", 1e-4);
f = @(t,y)-(y-3); y_exact = @(t) 3-3*exp(-t);

[t45, y45] = ode45 (f, [0, 5], 0, opt);
[t23, y23] = ode23 (f, [0, 5], 0, opt);
[t15s, y15s] = ode15s (f, [0, 5], 0, opt);
[t23s, y23s] = ode23s (f, [0, 5], 0, opt);

figure(1); plot(t45,y_exact(t45),t45,y45,t23,y23,t15s,y15s,t23s,y23s)
    xlabel('time t'); ylabel('solution y(t)'); title('solution')
    legend('exact','ode45','ode23','ode15s','ode23s','location','southeast')
figure(2); plot(t45,y45-y_exact(t45),t23,y23-y_exact(t23),...
    t15s,y15s-y_exact(t15s),t23s,y23s-y_exact(t23s))
    xlabel('time t'); title('difference to exact solution')
    legend('ode45','ode23','ode15s','ode23s')
figure(3); plot(t45(2:end),diff(t45),'+',t23(2:end),diff(t23),'+',...
    t15s(2:end),diff(t15s),'*', t23s(2:end),diff(t23s),'*')
    xlabel('time t'); title('size of time step')
    legend('ode45','ode23','ode15s','ode23s','location','northwest')
NumberOfStepsNonStiff = [length(t45),length(t23),length(t15s),length(t23s)]-1

```

◇

### 3–68 Example : The ODE

$$\frac{dy}{dt} = -1000 y(t) + 3000 - 2000 \exp(-t) \quad \text{with } y(0) = 0$$

with the exact solution

$$y(t) = 3 - 0.998 \exp(-1000 t) - 2.002 \exp(-t)$$

is an example of a stiff problem. For  $0 \leq t \ll 1$  the term  $\exp(-1000 t)$  (generated by  $-1000 y(t)$ ) dominates, then  $\exp(-t)$  takes over. Those effects occur on a different time scale. Find the graphical results in Figure 3.44 and the number of steps and evaluations in Table 3.11. The stiffness of the ODE is confirmed by the number of time steps and evaluation of the RHS in Table 3.11. Observe that the algorithm in `ode15s` generates the most accurate results, see Figure 3.44. The Octave code is very similar to the previous example.

Some of the results in Table 3.11 can be explained.

- For `ode45()` the number of evaluations equals 6.65 times the number of time steps. One step of Dormand–Prince uses 6 evaluations, thus the step size had to be made shorter many times.
- The number of evaluations for the stiff solvers `ode15s()` and `ode23s()` is considerably smaller than for the explicit solvers `ode45()` and `ode23()`.

	ode45	ode23	ode15s	ode23s
Number of steps	1522	2011	99	222
Number of evaluations	10125	6045	178	1778

Table 3.11: Data for a stiff ODE problem with different algorithms

◇

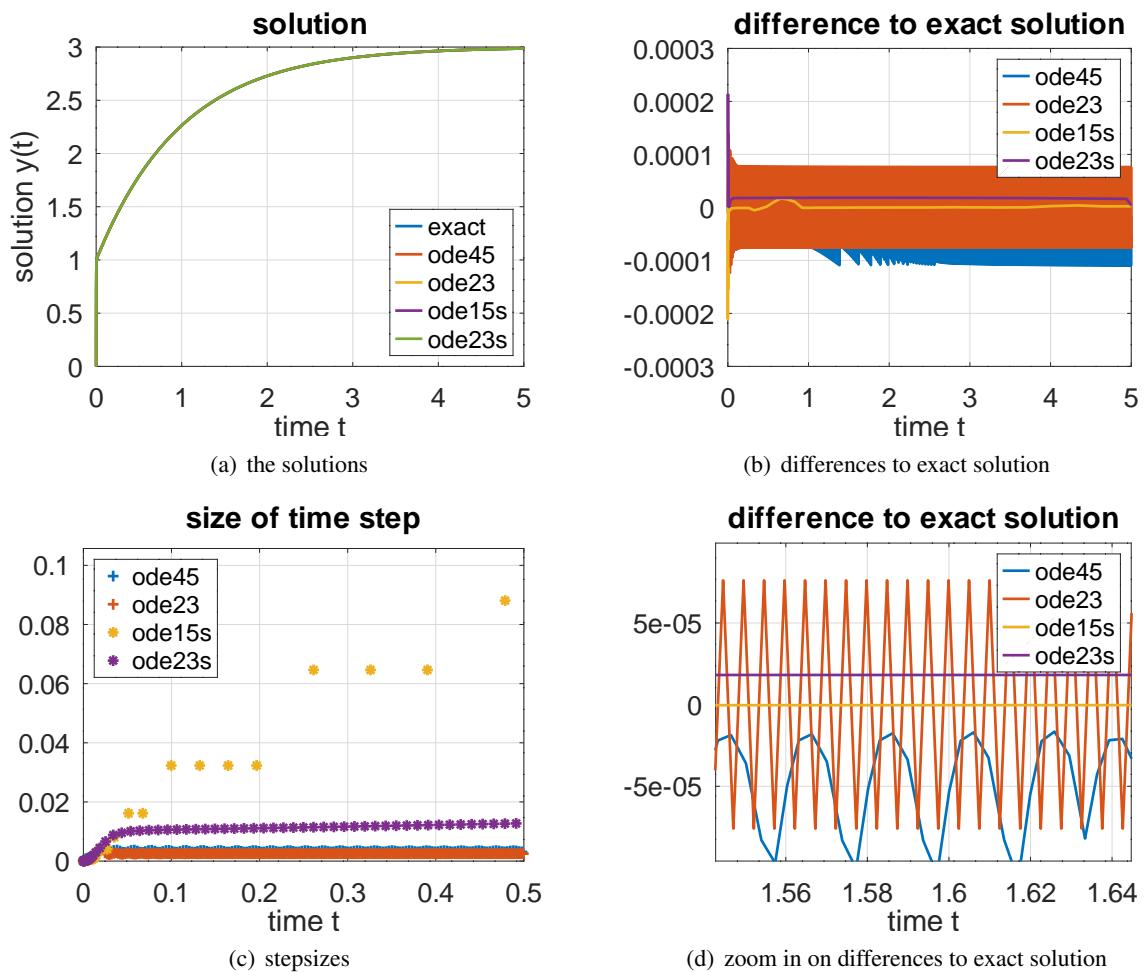


Figure 3.44: Solution of a stiff ODE with different algorithms

**3–69 Example :** The system of ODEs

$$\frac{d}{dt} \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{bmatrix} 98 & 198 \\ -99 & -199 \end{bmatrix} \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} + \begin{pmatrix} -98 \\ +99 \end{pmatrix} \quad \text{with} \quad \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

is solved by

$$\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \exp(-t) \begin{pmatrix} 2 \\ -1 \end{pmatrix} + \exp(-100t) \begin{pmatrix} -1 \\ +1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The result is based on the eigenvalues  $\lambda_1 = -100$  and  $\lambda_2 = -1$  of the matrix. Since the eigenvalues have different magnitudes this is a (moderately) stiff system.

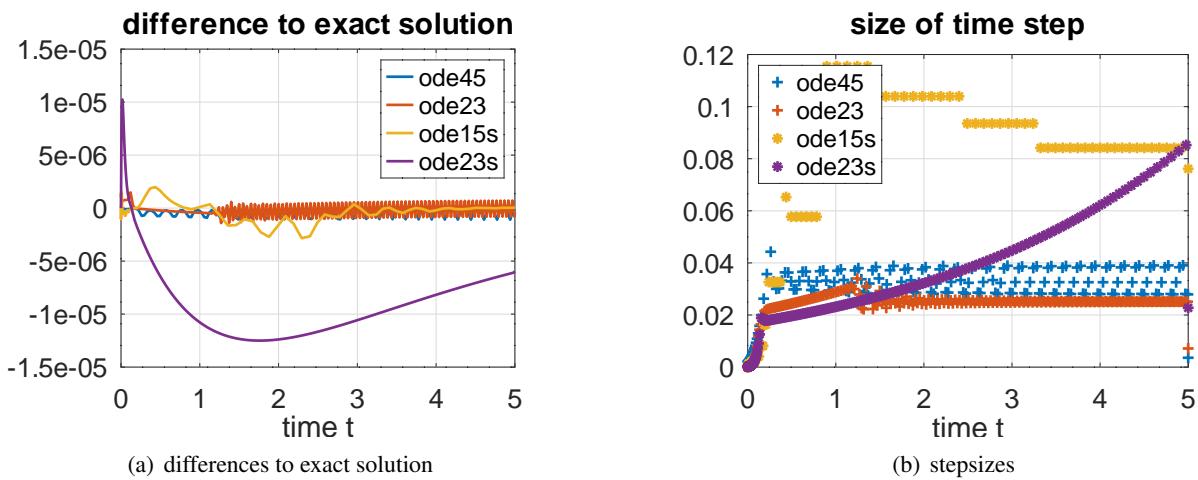


Figure 3.45: Solution of a stiff ODE system with different algorithms

Some of the results in Table 3.12 can be explained.

- For `ode45()` the number of evaluations equals 6.9 times the number of time steps. One step of Dormand–Prince uses 6 evaluations, thus the step size had to be made shorter many times.
- The number of evaluations for the stiff solvers `ode15s()` and `ode23s()` is considerably smaller than for the explicit solvers `ode45()` and `ode23()`.

	ode45	ode23	ode15s	ode23s
Number of steps	174	308	166	283
Number of evaluations	1198	950	284	2832

Table 3.12: Data for a stiff ODE system with different algorithms

The results are generated by the code

```

opt = odeset ("RelTol", 1e-8, "AbsTol", 1e-6);
f = @(t,y) [98 198;-99, -199]* (y-[1;0]);
y_exact = @(t) 2*exp(-t)-exp(-100*t)+1; %% only first component

[t45, y45] = ode45 (f, [0, 5], [2;0], opt); y45 = y45(:,1);
[t23, y23] = ode23 (f, [0, 5], [2;0], opt); y23 = y23(:,1);
[t15s, y15s] = ode15s (f, [0, 5], [2;0], opt); y15s = y15s(:,1);

```

```
[t23s, y23s] = ode23s (f, [0, 5], [2;0], opt); y23s = y23s(:,1);

figure(1); plot(t45,y_exact(t45),t45,y45,t23,y23,t15s,y15s,t23s,y23s)
xlabel('time t'); ylabel('solution y(t)');
legend('exact','ode45','ode23','ode15s','ode23s')
title('solution')

figure(2); plot(t45,y45-y_exact(t45),t23,y23-y_exact(t23),...
    t15s,y15s-y_exact(t15s), t23s,y23s-y_exact(t23s))
xlabel('time t'); title('difference to exact solution')
legend('ode45','ode23','ode15s','ode23s')

t_lim = 5; t23_short = t23(t23 < t_lim);
figure(3); plot(t45(2:end),diff(t45),'+',t23(2:end),diff(t23),'+',...
    t15s(2:end),diff(t15s),'*',t23s(2:end),diff(t23s),'*')
xlabel('time t'); title('size of time step')
legend('ode45','ode23','ode15s','ode23s','location','northwest')

NumberOfSteps = [length(t45),length(t23),length(t15s),length(t23s)]-1
```

In the above code the Jacobian for the ODE was not used. The constant Jacobian is given by

$$\mathbf{J} = \frac{\partial \vec{f}}{\partial \vec{y}} = \begin{bmatrix} 98 & 198 \\ -99 & -199 \end{bmatrix}.$$

Pass this information on to the implicit algorithms `ode15s()` and `ode23s()` by setting the option.

```
opt = odeset ("RelTol", 1e-8, "AbsTol", 1e-6, 'Jacobian',[98 198;-99, -199]);
```

Then obtain the results in Table 3.13. Observe that the number of steps does not change, but the number of evaluations by `ode15s()` and `ode23s()` is substantially smaller. This is due to the algorithms not having to determine the Jacobian numerically by evaluating the function  $f(t, \vec{y})$ .

	ode45	ode23	ode15s	ode23s
Number of steps	174	308	166	283
Number of evaluations	1198	950	222	1417

Table 3.13: Data for a stiff ODE system with different algorithms, using the Jacobian matrix

◇

**3-70 Example :** The system of ODEs

$$\frac{dy}{dt} \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{bmatrix} 79 & 72 \\ -90 & -82 \end{bmatrix} \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} + \begin{pmatrix} -79 \\ +90 \end{pmatrix} \quad \text{with} \quad \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

is solved by

$$\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \exp(-t) \begin{pmatrix} +9 \\ -10 \end{pmatrix} + \exp(-2t) \begin{pmatrix} -8 \\ +9 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The result is based on the eigenvalues  $\lambda_1 = -1$  and  $\lambda_2 = -2$  of the matrix. Since the eigenvalues have similar magnitudes this is a non-stiff system. In Figure 3.46 and Table 3.14 verify that the algorithm `ode45` generates the most accurate results with the fewest number of evaluations of the RHS of the ODE.

Some of the results in Table 3.14 can be explained.

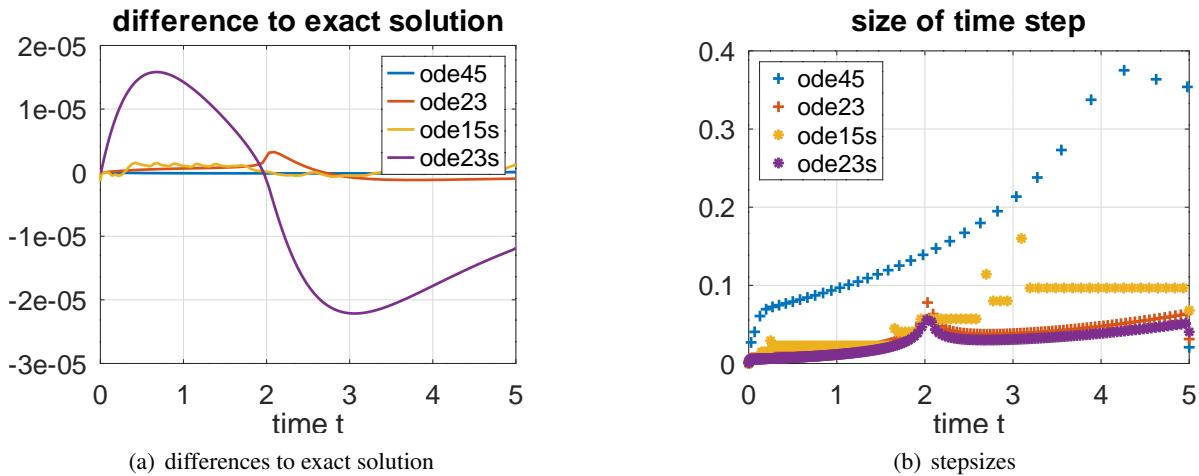


Figure 3.46: Solution of a non-stiff ODE system with different algorithms

- For `ode45()` the number of evaluations equals 6.26 times the number of time steps. One step of Dormand–Prince uses 6 evaluations, thus the step size had to be made shorter only a few times. This is visible in Figure 3.46.
- For `ode23()` the number of evaluations equals 3.01 times the number of time steps. One step of Bogacki–Shampine uses 3 evaluations, thus the step size was increased most of the times. This is visible in Figure 3.46. Considerably more time steps are used by `ode23()`, compared to `ode45()`.

	<code>ode45</code>	<code>ode23</code>	<code>ode15s</code>	<code>ode23s</code>
Number of steps	35	216	155	271
Number of evaluations	219	651	270	2711

Table 3.14: Data for non-stiff ODE system with different algorithms

The Octave code is very similar to the previous example. ◇

## 3.5 Linear and Nonlinear Regression, Curve Fitting

In this section the basics of linear and nonlinear regression are presented. Linear regression is one of the basic tools for machine learning (ML), e.g. [Agga20] or [DeisFaisOng20, §9]. Using Octave/MATLAB a few examples for linear and nonlinear regression are provided.

### 3.5.1 Linear Regression, Method of Least Squares

#### Linear regression for a straight line

For  $n$  given points  $(x_i, y_i)$  in a plane try to determine a straight line  $y(x) = p_1 \cdot 1 + p_2 \cdot x$  to match those points as good as possible. One good option is to examine the residuals  $r_i = p_1 \cdot 1 + p_2 \cdot x_i - y_i$ . Using matrix notation write

$$\vec{r} = \mathbf{F} \cdot \vec{p} - \vec{y} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \cdot \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}.$$

Linear regression corresponds to minimization of the norm of  $\vec{r}$ , i.e. minimize

$$\sum_{i=1}^n r_i^2 = \|\vec{r}\|^2 = \|\mathbf{F} \cdot \vec{p} - \vec{y}\|^2 = \langle \mathbf{F} \cdot \vec{p} - \vec{y}, \mathbf{F} \cdot \vec{p} - \vec{y} \rangle.$$

This is the reason for the often used name **method of least squares**.

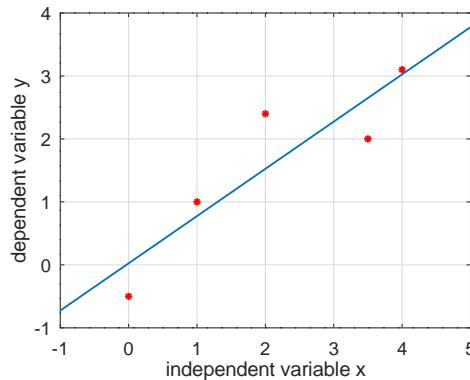


Figure 3.47: Regression of a straight line

Consider  $\|\vec{r}\|^2$  as a function of  $p_1$  and  $p_2$ . At the minimal point the two partial derivatives with respect to  $p_1$  and  $p_2$  have to vanish. This leads to a system of linear equations for the vector  $\vec{p}$ , the **normal equation**.

$$(\mathbf{F}^T \mathbf{F}) \cdot \vec{p} = \mathbf{F}^T \vec{y}$$

This can easily be implemented in Octave, leading to the result in Figure 3.47 and a residual of  $\|\vec{r}\|^2 \approx 1.23$ .

**Octave**

```
x = [0; 1; 2; 3.5; 4]; y = [-0.5; 1; 2.4; 2.0; 3.1];
F = [ones(size(x)) x];
p = (F' * F) \ (F' * y);
```

```

residual = norm(F*p-y)

xn = [-1 5]; yn = p(1)+p(2)*xn;
plot(x,y,'*r',xn,yn);
xlabel('independent variable x'); ylabel('dependent variable y')

```

### Linear regression with a matrix notation

The above idea carries over to a linear combination of functions  $f_j(x)$  for  $1 \leq j \leq m$ . For a vector  $\vec{x} = (x_1, x_2, \dots, x_k)^T$  examine a function of the form

$$f(\vec{x}) = \sum_{j=1}^m p_j \cdot f_j(\vec{x}).$$

The optimal values of the parameter vector  $\vec{p} = (p_1, p_2, \dots, p_m)^T$  have to be determined. Thus minimize the expression

$$\chi^2 = \|\vec{r}\|^2 = \sum_{i=1}^n r_i^2 = \sum_{i=1}^n (f(x_i) - y_i)^2 = \sum_{i=1}^n \left( \left( \sum_{j=1}^m p_j \cdot f_j(x_i) \right) - y_i \right)^2$$

Using a vector and matrix notation

$$\vec{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_m \end{pmatrix}, \quad \vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{F} = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \dots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \dots & f_m(x_n) \end{bmatrix}$$

the expression to be minimized is given by

$$\|\vec{r}\|^2 = \|\mathbf{F} \cdot \vec{p} - \vec{y}\|^2 = \langle \mathbf{F} \cdot \vec{p} - \vec{y}, \mathbf{F} \cdot \vec{p} - \vec{y} \rangle.$$

Computing the partial derivatives with respect to the parameters leads to a necessary condition.

$$(\mathbf{F}^T \mathbf{F}) \cdot \vec{p} = \mathbf{F}^T \vec{y}$$

This system of  $n$  linear equations for the unknown vector of parameters  $\vec{p} \in \mathbb{R}^m$  is called a **normal equation**. With the optimal parameter vector  $\vec{p}$ , compute the values of the regression curve with a matrix multiplication

$$(\mathbf{F} \vec{p})_i = \sum_{j=1}^m p_j \cdot f_j(x_i).$$

### QR factorization and linear regression

For a  $n \times m$  matrix  $\mathbf{F}$  with more rows than columns ( $n > m$ ) a **QR** factorization of the matrix can be computed

$$\mathbf{F} = \mathbf{Q} \cdot \mathbf{R}$$

where the  $n \times n$  matrix  $\mathbf{Q}$  is orthogonal ( $\mathbf{Q}^{-1} = \mathbf{Q}^T$ ) and the  $n \times m$  matrix  $\mathbf{R}$  has an upper triangular structure. Now consider the block matrix notation

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_l & \mathbf{Q}_r \end{bmatrix} \quad \text{and} \quad \mathbf{R} = \begin{bmatrix} \mathbf{R}_u \\ \mathbf{0} \end{bmatrix}.$$

The  $m \times m$  matrix  $\mathbf{R}_u$  is square and upper triangular. The left part  $\mathbf{Q}_l$  of the square matrix  $\mathbf{Q}$  is of size  $n \times m$  and satisfies  $\mathbf{Q}_l^T \mathbf{Q}_l = \mathbb{I}_n$ . Use the zeros in the lower part of  $\mathbf{R}$  to verify that

$$\mathbf{F} = \mathbf{Q} \cdot \mathbf{R} = \mathbf{Q}_l \cdot \mathbf{R}_u.$$

MATLAB/Octave can compute the QR factorization by  $[\mathbf{Q}, \mathbf{R}] = \text{qr}(\mathbf{F})$  and the reduced form by the command  $[\mathbf{Q}_l, \mathbf{R}_u] = \text{qr}(\mathbf{F}, 0)$ . This factorization is very useful to implement linear regression.

Multiplying a vector  $\vec{r} \in \mathbb{R}^n$  with the orthogonal matrix  $\mathbf{Q}$  or its inverse  $\mathbf{Q}^T$  corresponds to a rotation of the vector and thus will not change its length<sup>18</sup>. This observation can be used to rewrite the linear regression problem from Section 3.5.1.

$$\begin{aligned} \mathbf{F} \cdot \vec{p} - \vec{y} &= \vec{r} \quad \text{length to be minimized} \\ \mathbf{Q} \cdot \mathbf{R} \cdot \vec{p} - \vec{y} &= \vec{r} \quad \text{length to be minimized} \\ \mathbf{R} \cdot \vec{p} - \mathbf{Q}^T \cdot \vec{y} &= \mathbf{Q}^T \cdot \vec{r} \\ \begin{bmatrix} \mathbf{R}_u \cdot \vec{p} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{Q}_l^T \cdot \vec{y} \\ \mathbf{Q}_r^T \cdot \vec{y} \end{bmatrix} &= \begin{bmatrix} \mathbf{Q}_l^T \cdot \vec{r} \\ \mathbf{Q}_r^T \cdot \vec{r} \end{bmatrix} \end{aligned}$$

Since the vector  $\vec{p}$  does not change the lower part of the above system, the problem can be replaced by a smaller system of  $m$  equations for  $m$  unknowns, namely the upper part only of the above system.

$$\mathbf{R}_u \cdot \vec{p} - \mathbf{Q}_l^T \cdot \vec{y} = \mathbf{Q}_l^T \cdot \vec{r} \quad \text{length to be minimized}$$

Obviously this length is minimized if  $\mathbf{Q}_l^T \cdot \vec{r} = \vec{0}$  and thus find the reduced equations for the vector  $\vec{p}$ .

$$\begin{aligned} \mathbf{R}_u \cdot \vec{p} &= \mathbf{Q}_l^T \cdot \vec{y} \\ \vec{p} &= \mathbf{R}_u^{-1} \cdot \mathbf{Q}_l^T \cdot \vec{y} \end{aligned}$$

In Octave the above algorithm can be implemented with two commands only.

---

**Octave**


---

```
[Q,R] = qr(F,0);
p = R\ (Q'*y);
```

---

It can be shown that the condition number for the QR algorithm is much smaller than the condition number for the algorithm based on  $(\mathbf{F}^T \mathbf{F}) \cdot \vec{p} = \mathbf{F}^T \vec{y}$ . Thus there are fewer accuracy problems to be expected and the results are more reliable<sup>19</sup>.

As a simple example fit a function  $f(x) = p_1 \cdot 1 + p_2 \cdot x + p_3 \cdot \sin(x)$  to a given set of data points  $(x_i, y_i)$  for  $1 \leq i \leq n$ , as seen in Figure 3.48. In this example the  $n \times 3$  matrix  $\mathbf{F}$  is given by

$$\mathbf{F} = \begin{bmatrix} 1 & x_1 & \sin(x_1) \\ 1 & x_2 & \sin(x_2) \\ 1 & x_3 & \sin(x_3) \\ \vdots & \vdots & \vdots \\ 1 & x_n & \sin(x_n) \end{bmatrix}.$$

The code below first generates random data and then uses the reduced QR factorization to apply the linear regression.

---

<sup>18</sup>

$\|\mathbf{Q} \vec{x}\|^2 = \langle \mathbf{Q} \vec{x}, \mathbf{Q} \vec{x} \rangle = \langle \vec{x}, \mathbf{Q}^T \mathbf{Q} \vec{x} \rangle = \langle \vec{x}, \mathbb{I} \vec{x} \rangle = \|\vec{x}\|^2$

<sup>19</sup>A careful computation shows that using the QR factorization  $\mathbf{F} = \mathbf{Q} \mathbf{R}$  in  $\mathbf{F}^T \mathbf{F} \vec{p} = \mathbf{F}^T \vec{y}$  also leads to  $\mathbf{R}_u \vec{p} = \mathbf{Q}_l^T \vec{y}$ .

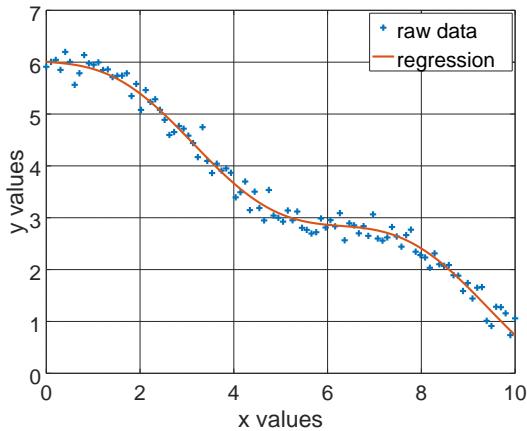


Figure 3.48: An example for linear regression

```
% generate the random data
n = 100; x = linspace(0,10,n);
y = 6-0.5*x+0.4*sin(x) + 0.2*randn(1,n);

% perform the linear regression, using the QR factorization
F = [ones(n,1) x(:) sin(x(:))];
[Q1,Ru] = qr(F,0); % apply the reduced QR factorization
p = Ru\ (Q1'*y(:)) % determine the optimal parameters
Ru % display the upper right matrix
y_reg = F*p; % determine the linear regression curve

figure(1); plot(x,y,'+',x,y_reg)
legend('raw data','regression')
xlabel('x values'); ylabel('y values')
-->
p =
6.00653
-0.50358
0.43408
Ru =
-10.00000 -50.00000 -1.79193
0.00000 29.15765 -0.50449
0.00000 0.00000 6.62802
```

### SVD, singular value decomposition and linear regression

For  $\mathbf{F} \in \mathbb{R}^{n \times m}$  with  $n > m$  it is possible to use the SVD  $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}$  to analyze the linear regression problem. For this split up the matrix  $\mathbf{U} \in \mathbb{R}^{n \times n}$  in a left part  $\mathbf{U}_l \in \mathbb{R}^{n \times m}$  and a right part  $\mathbf{U}_r \in \mathbb{R}^{n \times (n-m)}$ .

$$\mathbf{F} = \mathbf{U} \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & \sigma_n \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \mathbf{V}^T = \mathbf{U}_l \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & \sigma_n \end{bmatrix} \mathbf{V}^T = \mathbf{U}_l \Sigma \mathbf{V}$$

Now the linear regression problem can be examined. The computations are rather similar to the linear regression approach using the QR factorization.

$$\begin{aligned} \mathbf{F} \cdot \vec{p} - \vec{y} &= \vec{r} \quad \text{length to be minimized} \\ \mathbf{U} \cdot \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_n) \cdot \mathbf{V}^T \cdot \vec{p} - \vec{y} &= \vec{r} \quad \text{length to be minimized} \\ \Sigma \cdot \mathbf{V}^T \cdot \vec{p} - \mathbf{U}^T \vec{y} &= \mathbf{U}^T \vec{r} \quad \text{length to be minimized} \\ \begin{bmatrix} \Sigma \cdot \mathbf{V}^T \cdot \vec{p} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{U}_l^T \vec{y} \\ \mathbf{U}_r^T \vec{y} \end{bmatrix} &= \begin{bmatrix} \mathbf{U}_l^T \vec{r} \\ \mathbf{U}_r^T \vec{r} \end{bmatrix} \quad \text{length to be minimized} \\ \text{optimize upper part only} &, \quad \text{set } \mathbf{U}_l^T \vec{r} = \vec{0}, \text{ with smallest possible norm} \\ \Sigma \cdot \mathbf{V}^T \cdot \vec{p} - \mathbf{U}_l^T \vec{y} &= \vec{0} \\ \Sigma \cdot \mathbf{V}^T \cdot \vec{p} &= \mathbf{U}_l^T \vec{y} \end{aligned}$$

If  $\sigma_n > 0$  then the above problem has a unique solution. The ratio  $\sigma_1/\sigma_n$  contains information about the sensitivity of this least square problem. This allows to detect ill conditioned linear regression problems. For further information consult [GoluVanLoan13, §5.3].

MATLAB/Octave provide a command to generate the reduced SVD: `[U, S, V] = svd(F, 'econ')` or `[U, S, V] = svd(F, 0)`. The above algorithm is applied with just two commands.

```
[U1,S,V] = svd(F,0);
p = (S*V') \ (U1'*y(:))
```

The above linear regression example is solved by

```
[U1,S,V] = svd(F,0); % compute the reduced SVD factorization
p = (S*V') \ (U1'*y(:)) % determine the optimal parameters
y_reg = F*p; % determine the linear regression curve

figure(1); plot(x,y,'+',x,y_reg)
legend('raw data','regression')
xlabel('x values'); ylabel('y values')
```

The result will be identical to the one generated by the the QR factorization and also leads to Figure 3.48.

### 3.5.2 Estimation of the Variance of Parameters, Confidence Intervals, Domain of Confidence

For regression problems it is essential to not only determine the optimal values for the parameters, but also gain information on the accuracy and reliability of the result. Thus the domains of confidence for the parameters have to be determined.

Using the above results (for the parabola fit) determine the residual vector

$$\vec{r} = \mathbf{F} \cdot \vec{p} - \vec{y}$$

and then mean and variance  $V = \sigma^2$  of the  $y$ -errors can be estimated. The estimation is valid if the  $y$ -errors are independent and assumed to be of equal size, i.e. **assume a-priori that the errors are given by a normal distribution**. Then the variance  $\sigma^2$  of the  $y$  values can be estimated by

$$\sigma^2 = \frac{1}{n-m} \sum_{i=1}^n r_i^2$$

In most applications the values of the parameters  $p_j$  contain the essential information. It is often important to know how reliable the obtained results are, i.e. we want to know the variance of the determined parameter values  $p_j$ . To this end consider the normal equation

$$(\mathbf{F}^T \cdot \mathbf{F}) \cdot \vec{p} = \mathbf{F}^T \cdot \vec{y}$$

and thus the explicit expression for  $\vec{p}$

$$\vec{p} = (\mathbf{F}^T \cdot \mathbf{F})^{-1} \cdot \mathbf{F}^T \cdot \vec{y} = \mathbf{M} \cdot \vec{y} \quad (3.23)$$

or

$$p_j = \sum_{i=1}^n m_{j,i} y_i \quad \text{for } 1 \leq j \leq m \quad \text{with} \quad \mathbf{M} = [m_{j,i}]_{1 \leq j \leq m, 1 \leq i \leq n} = (\mathbf{F}^T \cdot \mathbf{F})^{-1} \cdot \mathbf{F}^T$$

is a  $m \times n$ -matrix, where  $m < n$  (more columns than rows). For a reliable evaluation the same argument has to be computed based on the **QR** factorization

$$\vec{p} = \mathbf{R}_u^{-1} \cdot \mathbf{Q}_l^T \cdot \vec{y}.$$

This explicit representation of  $p_j$  allows<sup>20</sup> to compute the variance  $\text{var}(p_j)$  of the parameters  $p_j$ , using the estimated variance  $\sigma^2$  of the  $y$ -values. The result is given by

$$\text{var}(p_j) = \sum_{i=1}^n m_{j,i}^2 \sigma^2 \quad \text{where} \quad \sigma^2 = \frac{1}{n-m} \sum_{i=1}^n r_i^2$$

Knowing this standard deviation and **assuming a normal distribution** one can (see [MontRung03, §12-3.1]) readily<sup>21</sup> determine the 95% **confidence intervals** for the individual parameters<sup>22</sup>, i.e. with a proba-

<sup>20</sup>If  $z_k$  are **independent** random variables given by a normal distribution with variances  $\text{var}(z_k)$ , then a linear combination of the  $z_i$  also leads to a normal distribution. The variances are given by the computational rules:

$$\begin{aligned} \text{var}(z_1 + z_2) &= \text{var}(z_1) + \text{var}(z_2) \\ \text{var}(\alpha_1 z_1) &= \alpha_1^2 \text{var}(z_1) \\ \text{var}\left(\sum_i \alpha_i z_i\right) &= \sum_i \alpha_i^2 \text{var}(z_i) \end{aligned}$$

<sup>21</sup>Use

$$\int_{-1.96\sigma}^{+1.96\sigma} \frac{1}{\sqrt{2\pi}} \exp(-x^2/(2\sigma^2)) dx \approx 0.95.$$

<sup>22</sup>A more careful approach is to determine the confidence region for  $\vec{p} \in \mathbb{R}^m$ , using a general  $m$ -dimensional  $F$ -distribution, see [https://en.wikipedia.org/wiki/Confidence\\_region](https://en.wikipedia.org/wiki/Confidence_region). The domain of confidence will be an ellipsoid in  $\mathbb{R}^m$ , best computed using a PCA. The difference can be substantial if the off-diagonal entries in the correlation matrix for the parameters  $\vec{p}$  are not small. Find more information in [RawlPantuDick98, §4.6.3] or examine Example 3-71 below.

bility of 95% the actual value of the parameter is between  $p_i - 1.96 \sqrt{\text{var}_i}$  and  $p_i + 1.96 \sqrt{\text{var}_i}$ <sup>23</sup>. The above assumes that the distribution of the parameters are normal distributions. Actually the distribution to use is a Student's t-distribution with  $n - m$  degrees of freedom. This can be computed in MATLAB/Octave by

```
p_CI = p + tinv(1-0.05/2,length(x)-m) * [-sigma +sigma]
```

### 3-71 Example : Confidence intervals and domains of confidence

As a simple (and academic) regression example examine the fit of a straight line to the curve  $y = \sin(x)$  for  $0 \leq x \leq \frac{\pi}{2}$ . Create 31 uniformly spaced points  $x_i = i \frac{\pi}{2.30}$  for  $i = 0, 1, 2, \dots, 30$  and the corresponding  $y_i = \sin(x_i)$ . Then two regressions are performed:

- Fit  $f_1(x) = p_1 1 + p_2 x$  by minimizing the length of the residual vector  $\vec{r}$ .

$$\mathbf{F}_1 \vec{p} - \vec{y} = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_{30} \end{bmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} - \begin{pmatrix} \sin(x_0) \\ \sin(x_1) \\ \sin(x_2) \\ \vdots \\ \sin(x_{30}) \end{pmatrix} = \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_{30} \end{pmatrix}$$

- Fit  $f_2(x) = p_1 1 + p_2 (x - 10)$  by minimizing the length of the residual vector  $\vec{r}$ .

$$\mathbf{F}_2 \vec{p} - \vec{y} = \begin{bmatrix} 1 & x_0 - 10 \\ 1 & x_1 - 10 \\ 1 & x_2 - 10 \\ \vdots & \vdots \\ 1 & x_{30} - 10 \end{bmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} - \begin{pmatrix} \sin(x_0) \\ \sin(x_1) \\ \sin(x_2) \\ \vdots \\ \sin(x_{30}) \end{pmatrix} = \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_{30} \end{pmatrix}$$

Both attempts fit a straight line through the same data points, but represented by slightly different parametrizations. The optimal values of the parameters and their estimated standard deviations are given by

$$\begin{aligned} p_1 &= 0.112357 \pm 0.024183 & p_1 &= 6.729570 \pm 0.244002 \\ p_2 &= 0.661721 \pm 0.026446 & p_2 &= 0.661721 \pm 0.026446 \end{aligned}$$

This leads to the 95% confidence intervals

$$\begin{aligned} 0.062898 &\leq p_1 \leq 0.161816 & 6.2305 &\leq p_1 \leq 7.2286 \\ 0.607634 &\leq p_2 \leq 0.715809 & 0.6076 &\leq p_2 \leq 0.7158 \end{aligned}$$

Observe that  $0.112357 + 0.661721 x \approx 6.729570 + 0.661721 (x - 10)$ . The two straight lines coincide, but the standard deviation of the parameter  $p_1$  is much larger for the second attempt. This is caused by the strong correlation between the functions 1 and  $(x - 10)$  for the second approach. This can be made visible by examining the correlation matrices

$$\text{corp}_1 = \begin{bmatrix} 1 & -0.8589 \\ -0.8589 & 1 \end{bmatrix} \quad \text{and} \quad \text{corp}_2 = \begin{bmatrix} 1 & 0.9987 \\ 0.9987 & 1 \end{bmatrix}.$$

<sup>23</sup>Observe that the CIs (Confidence Interval) are the probabilistic event, i.e. they will change from one random drawing to the next. Thus the statement is not "With probability 0.95 the true value of  $p_i$  is in the CI", but better "With probability 0.95 the generated CI contains the true value of  $p_i$ ". See [en.wikipedia.org/wiki/Confidence\\_interval](https://en.wikipedia.org/wiki/Confidence_interval).

With MATLAB/Octave<sup>24</sup> use `lscov()` to obtain the covariance matrix `covp` and then the correlation matrix by

$$\text{corp} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 \\ 0 & \frac{1}{\sigma_2} \end{bmatrix} \cdot \text{covp} \cdot \begin{bmatrix} \frac{1}{\sigma_1} & 0 \\ 0 & \frac{1}{\sigma_2} \end{bmatrix}.$$

- To construct a rectangular domain of confidence at confidence level  $1 - \alpha$  the two confidence intervals have to be determined with confidence level  $\sqrt{1 - \alpha} \approx 1 - \alpha/2$ . Then the parameters are in the rectangle with a confidence level of  $1 - \alpha$ .
- To construct a better, ellipsoidal domain of confidence use the covariance matrix. To determine the dimensions of the ellipses the F-distribution has to be used. For more information see [DrapSmit98, §5.4,§5.5], [RawlPantuDick98, §4.6.3] or use Section 3.2.7 with results on Gaussian distributions.

Find the graphical results in Figure 3.49. Observe that the domain of confidence in Figure 3.49(a) contains preciser information, mainly by the narrower range  $0.06 \leq p_1 \leq 0.16$  in horizontal direction. The domain of confidence in Figure 3.49(a) clearly points out that 95% is in an ellipse, not spread out over all of the rectangle. In Figure 3.49(b) for the regression  $f_2(x) = p_1 1 + p_2 (x - 10)$  the ellipse is extremely narrow. This is caused by the strong correlation between  $p_1$  and  $p_2$ , as computed in the correlation matrix `corp2`. The situation is better in Figure 3.49(a) for the regression  $f_1(x) = p_1 1 + p_2 x$ . An even slightly better solution would be to fit a function  $f_3(x) = p_1 1 + p_2 (x - \frac{\pi}{4})$ , where the two parameters are not correlated at all, i.e. the correlation matrix would be very close to the identity matrix  $\mathbb{I}_2$ . The result in Figure 3.49 would be an ellipse with horizontal axis. To generate the ellipses in Figure 3.49 use eigenvalues and eigenvectors, see Example 3-25.  $\diamond$

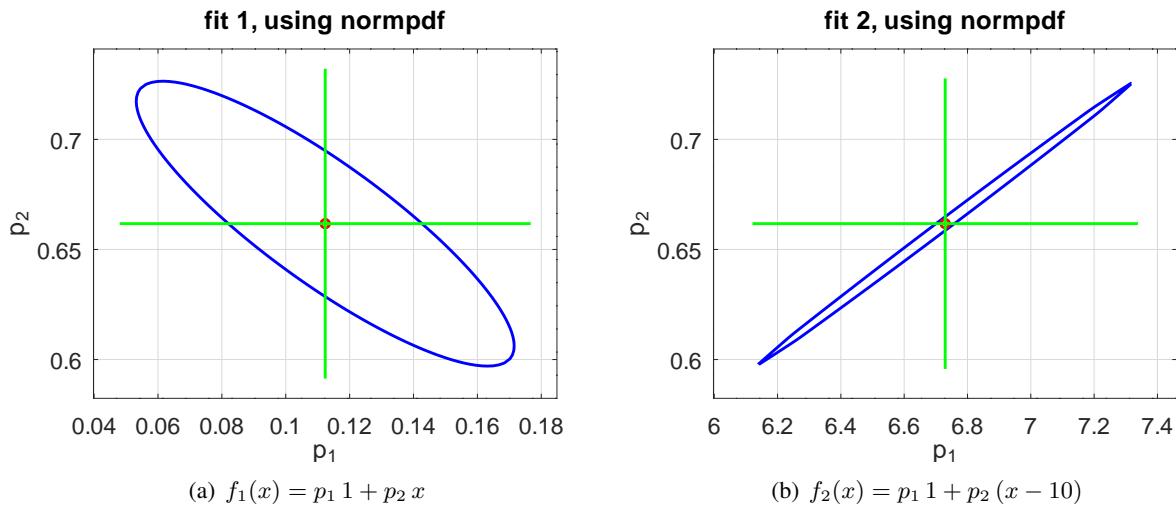


Figure 3.49: Regions of confidence for two similar regressions with identical data

### 3.5.3 The commands `LinearRegression()`, `regress()` and `lscov()` for Octave and MATLAB

In MATLAB and/or Octave there are a few commands to useful for linear regression, see Table 3.15. In these notes three are presented : `LinearRegression()`, `regress()` and `lscov()`.

<sup>24</sup>Based on [GoluVanLoan96, §5.6.3], which is also available in [GoluVanLoan13, §6.1.2].

Command	Properties
LinearRegression()	standard and weighted linear regression returns standard deviations for parameters
regress()	standard linear regression returns confidence intervals for parameters
lscov()	generalized least square estimation, with weights
ols()	ordinary least square estimation
gls()	generalized least square estimation
polyfit()	regression with for polynomials only
lsqnonneg()	regression with positivity constraint

Table 3.15: Commands for linear regression

**The command** LinearRegression()

The code for LinearRegression() is available for MATLAB and Octave.

- Octave: either load the optimization package with the command `pkg load optim` or download the file from my web site at [LinearRegression.m for Octave](#).
- MATLAB: download the file from my web site at [LinearRegression.m for Matlab](#).

The builtin help LinearRegression leads to

```

LinearRegression (F, Y)
LinearRegression (F, Y, W)
[P, E_VAR, R, P_VAR, FIT_VAR] = LinearRegression(...)

general linear regression

determine the parameters p_j (j=1,2,...,m) such that the function
f(x) = sum_(j=1,...,m) p_j*f_j(x) is the best fit to the given
values y_i by f(x_i) for i=1,...,n, i.e. minimize
sum_(i=1,...,n) (y_i - sum_(j=1,...,m) p_j*f_j(x_i))^2 with respect to p_j

parameters:
* F is an n*m matrix with the values of the basis functions at
the support points. In column j give the values of f_j at the
points x_i (i=1,2,...,n)
* Y is a column vector of length n with the given values
* W is a column vector of length n with the weights of the data points.
1/w_i is expected to be proportional to the estimated
uncertainty in the y values. Then the weighted expression
sum_(i=1,...,n) (w_i^2 * (y_i - f(x_i))^2) is minimized.

return values:
* P is the vector of length m with the estimated values of the parameters
* E_VAR is the vector of estimated variances of the provided y values.
If weights are provided, then the product e_var_i*w^2_i is assumed
to be constant.
* R is the weighted norm of the residual
* P_VAR is the vector of estimated variances of the parameters p_j
* FIT_VAR is the vector of the estimated variances of the fitted

```

```
function values f(x_i)
```

To estimate the variance of the difference between future y values and fitted y values use the sum of E\_VAR and FIT\_VAR

Caution: do NOT request FIT\_VAR for large data sets, as a n by n matrix is generated

The command `LinearRegression()` allows to use a weighted least square algorithm, i.e. instead of minimizing the standard  $\|\vec{r}\|^2 = \sum_{i=1}^n r_i^2$  the weighted expression

$$\|\vec{r}\|_W^2 = \sum_{i=1}^n w_i^2 r_i^2$$

is minimized, with given weights  $w_i$ . This should be used if some data is known to be more reliable than others, or to give outliers a lesser weight.

### The command regress ()

The command `regress()` is contained in the statistics toolbox in MATLAB (i.e. \$\$\$) and in the statistics package in Octave, i.e. `pkg load statistics`. The command `help regress` in Octave leads to

```
[B, BINT, R, RINT, STATS] = regress (Y, X, [ALPHA])
Multiple Linear Regression using Least Squares Fit of Y on X with
the model 'y = X * beta + e'.
```

Here,

- \* 'y' is a column vector of observed values
- \* 'X' is a matrix of regressors, with the first column filled with the constant value 1
- \* 'beta' is a column vector of regression parameters
- \* 'e' is a column vector of random errors

Arguments are

- \* Y is the 'y' in the model
- \* X is the 'X' in the model
- \* ALPHA is the significance level used to calculate the confidence intervals BINT and RINT (see 'Return values' below). If not specified, ALPHA defaults to 0.05

Return values are

- \* B is the 'beta' in the model
- \* BINT is the confidence interval for B
- \* R is a column vector of residuals
- \* RINT is the confidence interval for R
- \* STATS is a row vector containing:
  - \* The R^2 statistic
  - \* The F statistic
  - \* The p value for the full model
  - \* The estimated error variance

R and RINT can be passed to 'rcoplot' to visualize the residual intervals and identify outliers.

Nan values in Y and X are removed before calculation begins.

### The command `lscov()`

The command `lscov()` is available in MATLAB/*Octave*. It returns the covariance matrix for the optimal parameters, which is required to generate the ellipses of confidence for the optimal parameters. The command `help lscov` in *Octave* leads to

```
-- X = lscov (A, B)
-- X = lscov (A, B, V)
-- X = lscov (A, B, V, ALG)
-- [X, STDX, MSE, S] = lscov (...)
```

Compute a generalized linear least squares fit.

Estimate  $X$  under the model  $B = AX + W$ , where the noise  $W$  is assumed to follow a normal distribution with covariance matrix  $\{\sigma^2\} V$ .

If the size of the coefficient matrix  $A$  is  $n$ -by- $p$ , the size of the vector/array of constant terms  $B$  must be  $n$ -by- $k$ .

The optional input argument  $V$  may be an  $n$ -element vector of positive weights (inverse variances), or an  $n$ -by- $n$  symmetric positive semi-definite matrix representing the covariance of  $B$ . If  $V$  is not supplied, the ordinary least squares solution is returned.

The  $ALG$  input argument, a guidance on solution method to use, is currently ignored.

Besides the least-squares estimate matrix  $X$  ( $p$ -by- $k$ ), the function also returns  $STDX$  ( $p$ -by- $k$ ), the error standard deviation of estimated  $X$ ;  $MSE$  ( $k$ -by-1), the estimated data error covariance scale factors ( $\{\sigma^2\}$ ); and  $S$  ( $p$ -by- $p$ , or  $p$ -by- $p$ -by- $k$  if  $k > 1$ ), the error covariance of  $X$ .

Reference: Golub and Van Loan (1996), 'Matrix Computations (3rd Ed.)', Johns Hopkins, Section 5.6.3

See also: `ols`, `gls`, `lsqnonneg`.

### 3.5.4 An Elementary Example

As an illustrative example examine an exact curve  $y = x - x^2$  for  $0 \leq x \leq 1$ . Then some normally distributed noise is added, leading to data points  $(x_i, y_i)$ , shown in Figure 3.50. Using the command `LinearRegression()` a parabola  $y(x) = p_1 1 + p_2 x + p_3 x^2$  is fitted to the generated values  $y_i$ . The result are the estimated values for the parameters  $p_i$  and the estimated standard deviations for the parameters.

Ex. 3.17

#### Finding the optimal parameters, their standard deviations and the confidence intervals

On the first few lines in the code below the data is generated, using a normally distributed noise contribution. Then `LinearRegression()` is applied to determine the solutions.

```
N = 20 ; % number of data points
x = linspace(0,1); y = x.* (1-x); % the "true" data
x_d = rand(N,1); % the (random) data points
noise_small = 0.02*randn(N,1); % the small (random) noise
y_d = x_d.* (1-x_d) + noise_small; % random data points
```

```

F = [ones(size(x_d)) x_d x_d.^2]; % construct the regression matrix
[p, e_var, r, p_var, fit_var] = LinearRegression(F,y_d);
sigma = sqrt(p_var);
parameters = [p sigma] % show the parameters and their standard deviation

y_fit = p(1) + p(2)*x + p(3)*x.^2; % compute the fitted values

figure(1); plot(x,y,'k',x_d,y_d,'b+',x,y_fit,'g')
xlabel('independent variable x')
ylabel('dependent variable y')
title('regression with small noise')
legend('true curve','data','best fit','location','south')
-->
parameters = -7.0696e-03 7.3803e-03
1.1126e+00 3.6093e-02
-1.1446e+00 3.7854e-02

```

Using the standard deviations of the parameters one can then determine the confidence intervals for a chosen level of significance  $\alpha = 5\% = 0.05$ .

```

alpha = 0.05 ; % level of significance
p95_n = p + norminv(1-alpha/2)*[-sigma +sigma] % normal distribution
p95_t = p + tinv(1-alpha/2,length(x_d)-3)*[-sigma +sigma] % Student-t distribution
-->
p95_n = -2.1535e-02 7.3955e-03
1.0419e+00 1.1834e+00
-1.2188e+00 -1.0704e+00

p95_t = -2.2641e-02 8.5014e-03
1.0365e+00 1.1888e+00
-1.2244e+00 -1.0647e+00

```

The numerical result for the Student-t distribution implies that with a level of confidence of 95% the confidence intervals for the parameters  $p_i$  in  $y = p_1 1 + p_2 x + p_3 x^2$  satisfy

$$\begin{aligned} -0.023 &\leq p_1 \leq +0.009 \\ +1.04 &\leq p_2 \leq +1.19 \\ -1.22 &\leq p_3 \leq -1.06 . \end{aligned}$$

This confirms the “exact” values of  $p_1 = 0$ ,  $p_2 = +1$  and  $p_3 = -1$ . The best fit parabola in Figure 3.50(a) is rather close to the “true” parabola.

### The effect of large noise contributions

The above can be repeated with a considerably larger noise contribution.

```

noise_big = 0.5*randn(N,1); % the large (random) noise
...
parameters = [p sigma]
...
p95 = p + norminv(1-alpha/2)*[-sigma +sigma]
p95 = p + tinv(1-alpha/2,length(x_d)-3)*[-sigma +sigma]
-->
parameters = -0.082664 0.387973

```

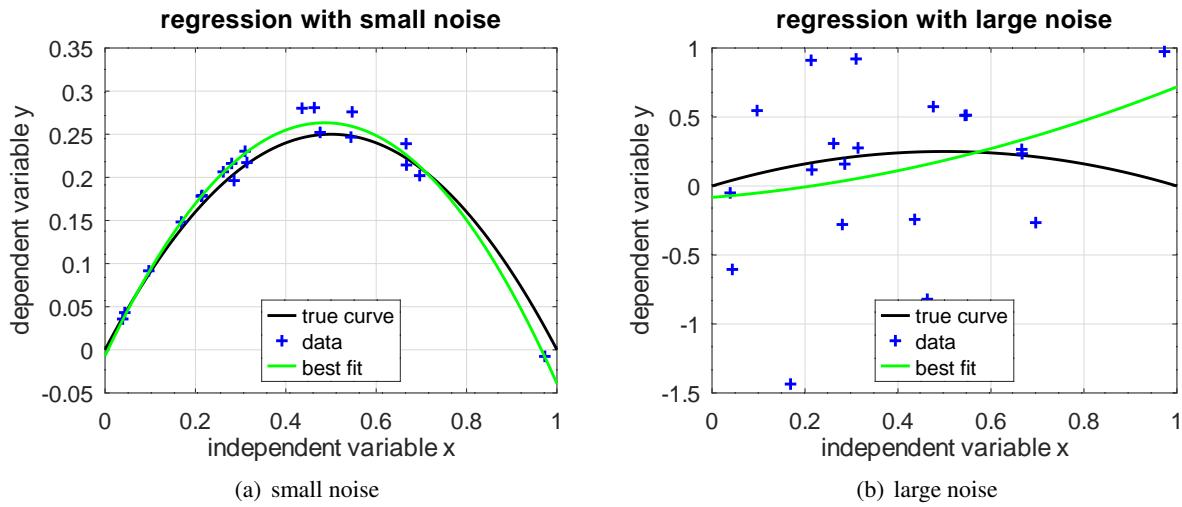


Figure 3.50: Linear regression for a parabola, with a small or a large noise contribution

0.272995 1.897374  
0.527731 1.989961

p95\_n = -0.8431 0.6777  
-3.4458 3.9918  
-3.3725 4.4280

p95\_t = -0.9012 0.7359  
-3.7301 4.2761  
-3.6707 4.7262

These numbers imply

$$p_1 \approx -0.08 \quad , \quad p_2 \approx 0.27 \quad \text{and} \quad p_3 \approx 0.53 \, .$$

This is obviously far from the “true” values and confirmed by the very wide confidence intervals.

$$\begin{array}{lclcl} -0.9 & \leq & p_1 & \leq & +0.7 \\ -3.7 & \leq & p_2 & \leq & +4.2 \\ -3.7 & \leq & p_3 & \leq & +4.7 \end{array}$$

The best fit parabola in Figure 3.50(b) is poor approximation of the “true” parabola.

### Using the command `regress()`

The problem above is solved using the function `LinearRegression()`, but one may use the function `regress()` instead. For the small noise case the code below shows the identical results.

```
[p_regress, p95_t_regress] = regress(y_d,F,alpha)
-->
p_regress      = -7.0696e-03
                  1.1126e+00
                  -1.1446e+00

p95_t_regress = -2.2641e-02   8.5014e-03
                  1.0365e+00   1.1888e+00
                  -1.2244e+00 -1.0647e+00
```

For the large noise case obtain again no surprise.

```
[p_regress, p95_t_regress] = regress(y_d,F,alpha)
-->
p_regress      = -0.082664
                  0.272995
                  0.527731

p95_t_regress = -0.9012    0.7359
                  -3.7301   4.2761
                  -3.6707   4.7262
```

### Using the command `lscov()`

The problem above is solved using the functions `LinearRegression()` or `regress()`, but one may use the function `lscov()` instead. For the small noise case the code below shows the identical results.

```
[p_lscov, sigma_lscov, ~, covp_lscov] = lscov(F,y_d);
p_lscov
p95_t_lscov = p_lscov + tinv(1-alpha/2,length(x_d)-3) * [-sigma_lscov +sigma_lscov]
-->
p_lscov      = -7.0696e-03
                  1.1126e+00
                  -1.1446e+00

p95_t_lscov = -2.2641e-02    8.5014e-03
                  1.0365e+00   1.1888e+00
                  -1.2244e+00  -1.0647e+00
```

The command `lscov` can also determine the covariance matrix for the parameters.

```
covp_lscov
-->
covp_lscov =  5.4469e-05  -2.3421e-04  2.0710e-04
              -2.3421e-04  1.3027e-03  -1.3021e-03
              2.0710e-04  -1.3021e-03  1.4330e-03
```

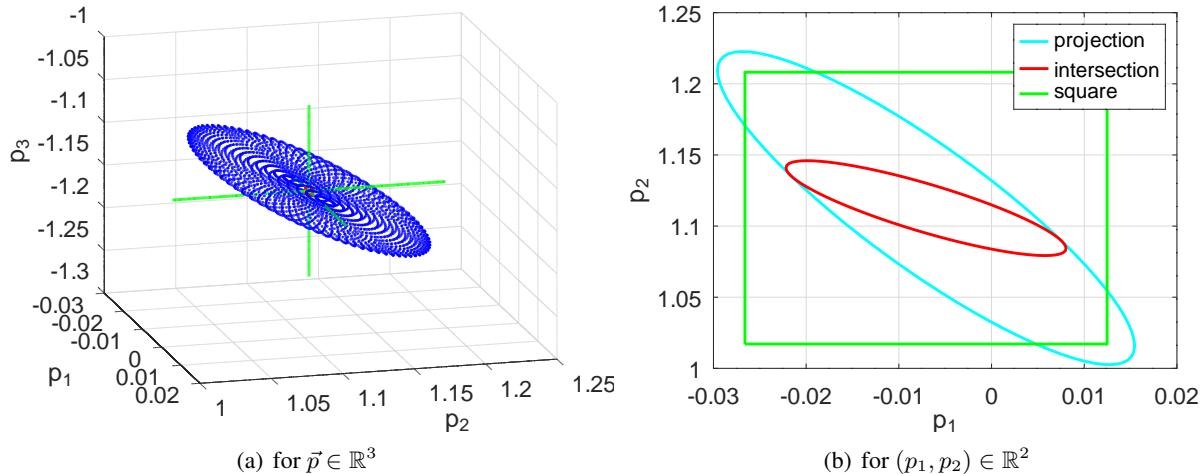
The rather large values off the diagonal in the correlation matrix indicate that the three parameters  $p_i$  are not independent.

```
corp = diag(1./sqrt(diag(covp_lscov))) * covp_lscov * diag(1./sqrt(diag(covp_lscov)))
-->
corp =  1.0000  -0.8792  0.7413
          -0.8792  1.0000  -0.9530
          0.7413  -0.9530  1.0000
```

With the covariance matrix the ellipsoidal region of confidence for  $\vec{p} \in \mathbb{R}^3$  can be determined. To start choose a level of significance, in this case  $\alpha = 0.05$ .

- Use Example 3–71 to determine the ellipsoidal domain of confidence in  $\mathbb{R}^3$  and Example 3–25 to visualize, see Figure 3.51(a). This figure can be rotated on screen if generated locally by Octave/MATLAB.

- To obtain printable visualizations restrictions to planes are useful. Use the tools from Example 3–27 to display the intersection and projection of the 3D domain of confidence with the plane  $p_3 = \text{const}$ . Find the result in Figure 3.51(b).
- If working with individual intervals of confidence the level of significance has to be adapted, such that  $(1 - \alpha_3)^3 = 1 - \alpha = 0.95$ , i.e.  $\alpha_3 = 1 - (1 - \alpha)^{1/3} \approx \frac{1}{3} \alpha$ . The projection of the “box” of confidence in  $\mathbb{R}^3$  leads to the rectangle in Figure 3.51(b).

Figure 3.51: Region of confidence for the parameters  $p_1$ ,  $p_2$  and  $p_3$ 

```

inv_cov = inv(covp_lscov);
%% the ellipsoid in space
[Evec,Eval] = eig(inv_cov);
phi = linspace(0,2*pi,81); theta = linspace(-pi,pi,81);
x = cos(phi')*cos(theta); y = sin(phi')*cos(theta); z = ones(size(phi'))*sin(theta);
Points = Evec*inv(sqrt(Eval))*[x(:),y(:),z(:)]';

%% the intersection with p3=0
A_intersect = inv_cov(1:2,1:2);
[Evec_i, Eval_i] = eig(A_intersect);
x = cos(phi); y = sin(phi); z = zeros(size(phi));
Points_i = [Evec_i*inv(sqrt(Eval_i))*[x(:),y(:)]'];

%%% projection onto the plane p3=0
A_project = ReduceQuadraticForm(inv_cov, 3);
[Evec_p, Eval_p] = eig(A_project);
Points_p = [Evec_p*inv(sqrt(Eval_p))*[x(:),y(:)]'];

p1 = p_lscov(1); p2 = p_lscov(2); p3 = p_lscov(3); sigma = sigma_lscov;
alpha3 = 1-(1-alpha)^(1/3); t_lim = tinv(1-alpha3/2,length(x_d)-3);
f_limit = 3*finv(1-alpha,3,length(x_d));

figure(2);
plot3(p1,p2,p3,'or', p1+sqrt(f_limit)*Points(1,:),...
       p2 + sqrt(f_limit)*Points(2,:),p3 + sqrt(f_limit)*Points(3,:),'.b',...
       [p1-t_lim*sigma(1),p1+t_lim*sigma(1)], [p2,p2], [p3,p3],'g',...

```

```
[p1,p1], [p2-t_lim*sigma(2),p2+t_lim*sigma(2)], [p3,p3],'g',...
[p1,p1], [p2,p2], [p3-t_lim*sigma(3),p3+t_lim*sigma(3)],'g')
xlabel('p_1'); ylabel('p_2'); zlabel('p_3'); view([75,20]);

figure(3);
plot(p1+sqrt(f_limit)*Points_p(1,:),p2+sqrt(f_limit)*Points_p(2,:),'c',...
    p1+sqrt(f_limit)*Points_i(1,:),p2+sqrt(f_limit)*Points_i(2,:),'r',...
    p1+t_lim*sigma(1)*[-1 1 1 -1 -1],p2+t_lim*sigma(2)*[-1,-1,1,1,-1],'g')
xlabel('p_1'); ylabel('p_2'); legend('projection','intersection','square')
```

For the large noise case obtain again no surprise, i.e. the same results as for `LinearRegression()` and `regress()`.

```
[p_lscov, sigma_lscov, ~, covp_lscov] = lscov (F,y_d);
p_lscov
p95_t_lscov = p_lscov + tinv(1-alpha/2,length(x_d)-3)*[-sigma_lscov +sigma_lscov]
-->
p_lscov = -0.082664
           0.272995
           0.527731

p95_t_lscov = -0.9012   0.7359
               -3.7301   4.2761
               -3.6707   4.7262
```

### Improving the confidence interval by using more data points

In the above Figure 3.50(b) the size of the noise was large and as a consequence the results for the parameters  $p_i$  was rather unreliable, i.e. a large standard variation and wide confidence intervals. This can be improved by using more data points, assuming that the noise is given by a normal distribution. By multiplying the number of data points by 100 ( $N \rightarrow 100*N$ ) theory predicts that the standard deviation  $\sigma$  should be  $\sqrt{100} = 10$  times smaller. This is caused by the matrix used to solve for the parameters. More data points lead to larger entries in  $\mathbf{F}$  and thus to smaller entries in the matrix  $(\mathbf{F}^T \mathbf{F})^{-1} \mathbf{F}$  or in the matrix  $\mathbf{R}_u^{-1}$ . The results confirm this expectation.

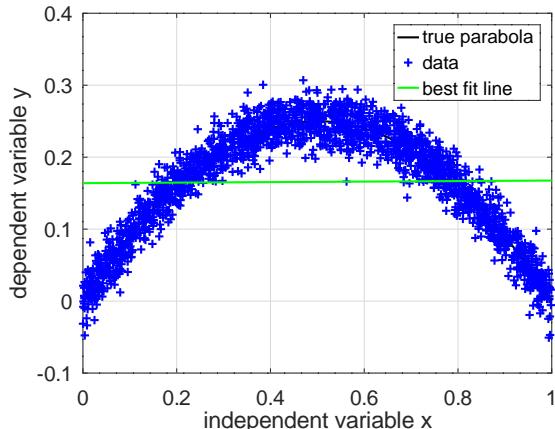
```
noise_big = 0.5*randn(100*N,1);
x_d = rand(100*N,1); y_d = x_d.* (1-x_d) + noise_big;

F = [ones(size(x_d)) x_d x_d.^2]; % construct the regression matrix
[p, e_var, r, p_var, fit_var] = LinearRegression(F,y_d);
sigma = sqrt(p_var);
parameters = [p sigma] % show the parameters and their standard deviation
p95_t=p+tinv(1-alpha/2,length(x_d)-3)*[-sigma +sigma] % Student-t distribution
-->
parameters = -0.0086227   0.0334205
               1.1139283   0.1555626
               -1.1457702   0.1514078

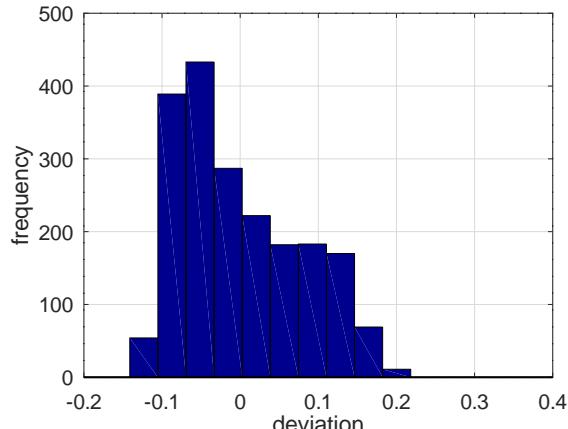
p95_t = -0.074165   0.056920
          0.808846   1.419010
          -1.442704  -0.848836
```

Using more data points will increase the reliability of the result, but only if the residuals are normally distributed. If you are fitting a straight line to data on a parabola the results can not improve. Thus one has

to make a visual check of the fitting (Figure 3.52(a)), to realize the the result is impossibly correct, even if the standard deviations of the parameters  $p_i$  are small. The problem is also visible in a histogram of the residuals (Figure 3.52(b)). Obviously the distribution of the residuals is far from a normal distribution, and thus the essential working assumption of normally distributed deviations is violated.



(a) fitting a straight line to a parabola



(b) histogram of the residuals

Figure 3.52: Fitting a straight line to data close to a parabola

```

noise_small = 0.02*randn(100*N,1); y_d = x_d.* (1-x_d) + noise_small;
F = [ones(size(x_d)) x_d]; % construct the regression matrix for a straight line
[p, e_var, r, p_var, fit_var] = LinearRegression(F,y_d);
sigma = sqrt(p_var);
parameters = [p sigma] % show the parameters and their standard deviation
p95_t = p+tinv(1-alpha/2,length(x_d)-3)*[-sigma +sigma] % Student-t distribution

y_fit = p(1) + p(2)*x; % compute the fitted values
figure(3); plot(x,y,'k',x_d,y_d,'b+',x,y_fit,'g')
    xlabel('independent variable x')
    ylabel('dependent variable y')
    title('regression with small noise')
    legend('true parabola','data','best fit line')

residuals = F*p-y_d;
figure(4); hist(residuals)
    xlabel('deviation'); ylabel('frequency')
-->
parameters =  0.1639191   0.0034668
              0.0035860   0.0059815

p95_t =  0.1571200   0.1707181
        -0.0081447   0.0153166

```

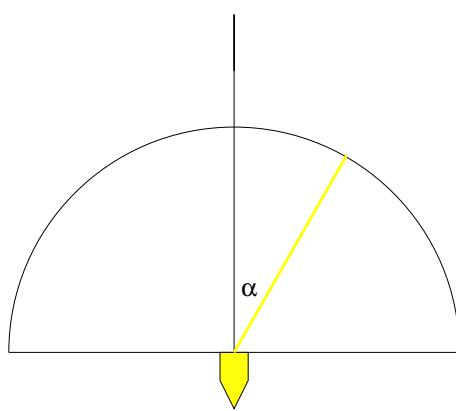
### 3.5.5 How to Obtain Wrong Results!

In Figure 3.53 find the data for the intensity of the light emitted by an LED, as function of the angle of emmission. For the design of the optical system the data points had to be “translated” to a function. Using linear regression leads to the obviously wrong result in Figure 3.53(b). The following computations were performed:

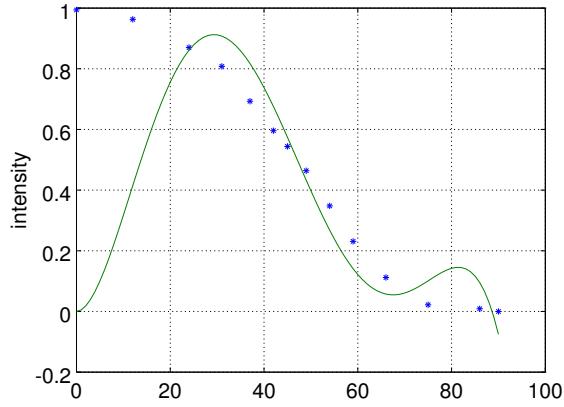
- The angle was given in degrees, i.e.  $0^\circ \leq \alpha \leq 90^\circ$
- A polynomial of degree 5 was used, i.e. regression for  $T(\alpha) = p_1 + p_2 \alpha + p_3 \alpha^2 + p_4 \alpha^3 + p_5 \alpha^4 + p_6 \alpha^5$ .
- The matrix  $\mathbf{F}$  was constructed, then the normal equation solved by

$$\vec{p} = \text{inv}(\mathbf{F}^T \mathbf{F}) \cdot (\mathbf{F}^T \vec{y}).$$

- The warning message about a singular matrix was ignored. Thus the computations performed with a matrix with a very large condition number, e.g.  $\kappa \approx 10^{17}$ .



(a) setup of LED



(b) intensity profile

Figure 3.53: Intensity of light as function of the angle  $\alpha$ 

The problem is perfectly solvable, when using better approaches.

- Since the intensity profile is symmetric (i.e.  $T(-\alpha) = T(\alpha)$ ), there can be no odd contributions to the polynomial<sup>25</sup>. Use the simpler function  $T(\alpha) = p_1 + p_2 \alpha^2 + p_3 \alpha^4$ , based on monomials of even degree only.
- Instead of the normal equation with the matrix  $\mathbf{F}^T \mathbf{F}$  use the **QR** factorization.
- The angle  $\alpha = 90^\circ$  leads to numbers  $90^5 \approx 6 \cdot 10^9$  and thus to numbers  $3 \cdot 10^{19}$  in the matrix  $\mathbf{F}^T \mathbf{F}$ . In the same matrix there are also numbers close to 1. Thus the condition number of this matrix is extremely large. Using radian instead of degrees helps, i.e. use an appropriate rescaling of the data.

Using just one of these three improvements leads to reasonable answers. The most important aspect to consider when using the linear regression method is to select the correct type of function for the fitting. This decision has to be made based on insight into the problem at hand.

Choose your basis functions for linear regression very carefully,  
based on information about the system to be examined.

<sup>25</sup>In my (long) career I have seen **no** application requiring a regression with a polynomial of high degree. I have seen many problems when using a polynomial of high degree.

### 3.5.6 A Regression Example with Multiple Independent Variables

It is as well possible perform linear regression with functions of multiple independent variables. The function

$$z = p_1 \cdot 1 + p_2 \cdot x + p_3 \cdot y$$

describes a plane in 3D space. A surface of this type is fit to a set of given points  $(x_j, y_j, z_j)$  by the code below, resulting in Figure 3.54. The columns of the matrix  $\mathbf{F}$  have to contain the values of the basis functions 1,  $x$  and  $y$  at the given data points.

$$\mathbf{F} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ \vdots \\ 1 & x_N & y_N \end{bmatrix}$$

```
N = 100; x = 2*rand(N,1); y = 3*rand(N,1);
z = 2 + 2*x - 1.5*y + 0.5*randn(N,1);

F = [ones(size(x)), x, y];
p = LinearRegression(F,z)

[x_grid, y_grid] = meshgrid([0:0.1:2], [0:0.2:3]);
z_grid = p(1) + p(2)*x_grid + p(3)*y_grid;

figure(1); plot3(x,y,z,'*')
hold on
mesh(x_grid,y_grid,z_grid)
xlabel('x'); ylabel('y'); zlabel('z');
hold off
-->
p = 1.7689 2.0606 -1.4396
```

Since only very few ( $N=100$ ) points were used the exact parameter values  $\vec{p} = (+2, +2, -1.5)$  are not very accurately reproduced. Increasing  $N$  will lead to more accurate results for this simulation, or decrease the size of the random noise in  $+0.5*\text{randn}(N, 1)$ .

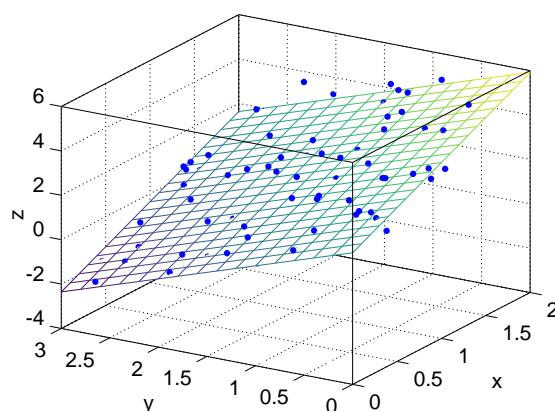


Figure 3.54: Result of a 3D linear regression

The command `LinearRegression()` does not determine the confidence intervals for the parameters, but it returns the estimated standard deviations, resp. the variances. With these the confidence intervals can be computed, using the Student-t distribution. To determine the CI modify the above code slightly.

```
[p,~,~,p_var] = LinearRegression(F,z); alpha = 0.05;
p_CI = p + tinv(1-alpha/2,N-3) * [-sqrt(p_var) +sqrt(p_var)]
-->
p_CI =
+1.6944 +2.2357
+1.8490 +2.2222
-1.5869 -1.3495
```

The results implies that the 95% confidence intervals for the parameters  $p_i$  are given by

$$\begin{aligned} +1.6944 &< p_1 < +2.2357 \\ +1.8490 &< p_2 < +2.2222 \quad \text{with a confidence level of 95\% .} \\ -1.5869 &< p_3 < -1.3495 \end{aligned}$$

### 3.5.7 Introduction to Nonlinear Regression

#### Introduction and first examples

The commands in the above section are well suited for linear regression problems, but there are many important **nonlinear** regression problems. Examine Table 3.16 to distinguish linear and nonlinear regression problems. Unfortunately nonlinear regression problems are considerably more delicate to work with and special algorithm have to be used. For many problems it is critical to find good initial guesses for the parameters to be determined. Linear and nonlinear regression problems may also be treated as minimization problems. This is often not a good idea, as regression problems have special properties that one can, and has to, take advantage of. Find a list of commands for nonlinear regression in Table 3.17. Observe that the syntax and algorithm of these commands might differ between MATLAB and Octave. You definitely have to consult the manuals and examine example applications.

function	parameters	type of regression
$y = a + mx$	$a, m$	linear
$y = ax^2 + bx + c$	$a, b, c$	linear
$y = ae^{cx}$	$a, c$	nonlinear
$y = d + ae^{cx}$	$a, c, d$	nonlinear
$y = d + ae^{cx}$	$a, d$	linear
$y = a \sin(\omega t + \delta)$	$a, \omega, \delta$	nonlinear
$y = a \cos(\omega t) + b \sin(\omega t)$	$a, b$	linear

Table 3.16: Examples for linear and nonlinear regression

#### Nonlinear least square fit with `leasqr()`

The **optimization package** of Octave provides the command `leasqr()`<sup>26</sup>. It is an excellent implementation of the Levenberg–Marquardt algorithm. The basic idea is a damped Newton’s method (see Section 3.1.6 on

<sup>26</sup>In Octave call `pkg load optim`. MATLAB users may use the codes on my web page `leasqr.m` and `dfdp.m`. The version of `leasqr.m` available on the internet seems to have a small bug computing the covariance matrix `covp`. The original source code contains a remark on an alternative method to determine `covp`. My web page contains the modified version.

Command	Properties
<code>leasqr()</code>	standard non linear regression, Levenberg–Marquardt
<code>fsolve()</code>	can be used for nonlinear regression too
<code>nlinfit()</code>	nonlinear regression
<code>lsqcurvefit()</code> <code>nonlin_curvefit()</code>	nonlinear curve fitting frontend, <i>Octave</i> only
<code>lsqnonlin()</code> <code>nonlin_residmin()</code>	nonlinear minimization of residue frontend, <i>Octave</i> only
<code>nlpaci()</code>	determine confidence intervals of parameters, MATLAB only
<code>expfit()</code>	regression with exponential functions

Table 3.17: Commands for nonlinear regression

page 119) to solve the corresponding system of nonlinear equations for the parameters. Thus `leasqr()` needs the matrix of partial derivatives with respect to the parameters. This matrix can be provided as function or estimated by the auxiliary function `dfdp.m`, which uses a finite difference approximation. The *Octave* package also provides one example in `leasqr_demo.m` and you can examine its source code.

As a first example try to fit a function

$$f(t) = A e^{-\alpha t} \cos(\omega t + \phi)$$

through a number of measured points  $(t_i, y_i)$ . Then search the values for the parameters  $A$ ,  $\alpha$ ,  $\omega$  and  $\phi$  to minimize

$$\sum_i |f(t_i) - y_i|^2.$$

Since the function is nonlinear with respect to the parameters  $\alpha$ ,  $\omega$  and  $\phi$  one can **not** use linear regression.

In *Octave* the command `leasqr()` will solve nonlinear regression problems. To set up an example one may:

1. Choose “exact” values for the parameters.
2. Generate normally distributed random numbers as perturbation of the “exact” result.
3. Define the appropriate function and generate the data.

Find the code below and the generated data points are shown in Figure 3.55, together with the best possible approximation by a function of the above type.

#### Octave

```
Ae = 1.5; ale = 0.1; omegae = 0.9 ; phie = 1.5;
noise = 0.1;
t = linspace(0,10,50)';
n = noise*randn(size(t));
function y = f(t,p)
y = p(1)*exp(-p(2)*t).*cos(p(3)*t + p(4));
endfunction
y = f(t,[Ae,ale,omegae,phie])+n;
plot(t,y,'+;data;')
```

You have to provide the function `leasqr()` with good initial estimates for the parameters. The algorithm in `leasqr()` uses a damped Newton method (see Section 3.1.5) to find the optimal solution. Examining the selection of points in Figure 3.55 estimate

- $A \approx 1.5$ : this might be the amplitude at  $t = 0$ .
- $\alpha \approx 0$ : there seems to be very little damping.
- $\omega \approx 0.9$ : the period seems to be slightly larger than  $2\pi$ , thus  $\omega$  slightly smaller than 1.
- $\phi \approx \pi/2$ : the graph seems to start out like  $-\sin(\omega t) = \cos(\omega t + \frac{\pi}{2})$

The results of your simulation might vary slightly, caused by the random numbers involved.

---

**Octave**


---

```
A0 = 2; a10 = 0; omega0 = 1; phi0 = pi/2;
[fr,p] = leasqr(t,y,[A0,a10,omega0,phi0],'f',1e-10);
p'

yFit = f(t,p);
plot(t,y,'+', t,yFit)
legend('data','fit')
-->
p = 1.523957 0.098949 0.891675 1.545294
```

---

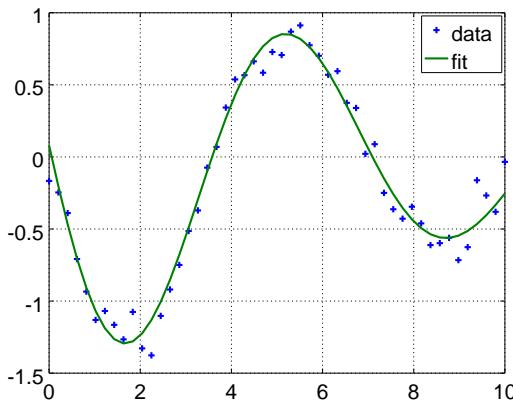


Figure 3.55: Least square approximation of a damped oscillation

The above result contains the estimates for the parameters. For many problems the deviations from the true curve are randomly distributed, with a normal distribution and small variance. In this case the parameters are also randomly distributed with a normal distribution. The diagonal of the covariance matrix contains the variances of the parameters and thus estimate the standard deviations by taking the square root of the variances.

---

**Octave**


---

```
pkg load optim % load the optimization package in Octave
[fr,p,kvg,iter,corp,covp,covr,stdresid,Z,r2] =...
    leasqr(t,y,[A0,a10,omega0,phi0],'f',1e-10);
pDev = sqrt(diag(covp))'
-->
pDev = 0.0545981 0.0077622 0.0073468 0.0307322
```

---

With the above results obtain Table 3.18. Observe that the results are consistent, i.e. the estimated parameters are rather close to the “exact” values. To obtain even better estimates, rerun the simulation with less noise or more points.

parameter	estimated value	standard dev.	”exact” value
$A$	1.52	0.055	1.5
$\alpha$	0.099	0.0078	0.1
$\omega$	0.892	0.0073	0.9
$\phi$	1.54	0.031	1.5

Table 3.18: Estimated and exact values of the parameters

### Nonlinear regression with `fsolve()`

The command `fsolve()` is used to solve systems of nonlinear equations, using Newton’s method. Assume that a function depends on parameters  $\vec{p} \in \mathbb{R}^m$  and the actual variable  $x$ , i.e.

$$y = f(\vec{p}, x).$$

A few ( $n$ ) points are given, thus  $\vec{x} \in \mathbb{R}^n$ , and the same number of values of  $\vec{y}_d \in \mathbb{R}^n$  are measured. For precise measurements we expect  $\vec{y}_d \approx \vec{y} = f(\vec{p}, \vec{x})$ . Then we can search for the optimal parameters  $\vec{p} \in \mathbb{R}^m$  such that

$$f(\vec{p}, \vec{x}) - \vec{y}_d = \vec{0}.$$

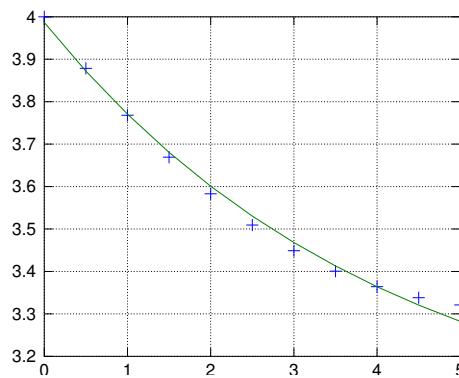
If  $m < n$  this is an **over determined** system of  $n$  equation for the  $m$  unknowns  $\vec{p} \in \mathbb{R}^m$ . In this case the command `fsolve()` will convert the system of equations to a minimization problem

$$\|f(\vec{p}, \vec{x}) - \vec{y}_d\| \text{ is minimized with respect to } \vec{p} \in \mathbb{R}^m .$$

It is also possible to estimate the variances of the optimal parameters, using the techniques from Section 3.5.2. This will be done very carefully in the following Section 3.5.8.

As an illustrative example data points on the curve  $y = \exp(-0.2x) + 3$  are generated and then some random noise is added. As initial parameters use the naive guess  $y(x) = \exp(0 \cdot x) + 0$ . The best possible fit is determined and displayed in Figure 3.56.

```
b0 = 3; a0 = 0.2; % choose the data
x = 0:.5:5; noise = 0.1 * sin (100*x); y = exp (-a0*x) + b0 + noise;
[p, fval, info, output] = fsolve (@(p) (exp (-p(1)*x) + p(2) - y), [0, 0]);
plot(x,y,'+', x,exp(-p(1)*x)+p(2))
```

Figure 3.56: Nonlinear least square approximation with `fsolve()`

### Nonlinear regression with lsqnonlin()

The optimization toolbox in MATLAB (\$\$\$) and the optim package in Octave provide the command `lsqnonlin()` to solve nonlinear regression problems. The Syntax is very similar to `fsolve()` and the result is identical to Figure 3.56.

```
b0 = 3; a0 = 0.2; % choose the data
x = 0:.5:5; noise = 0.1 * sin (100*x); y = exp (-a0*x) + b0 + noise;

p = lsqnonlin (@(p) (exp(-p(1)*x) + p(2) - y), [0, 0]);
plot(x,y,'+', x,exp(-p(1)*x)+p(2))
```

### 3.5.8 Nonlinear Regression with a Logistic Function

In this section a more concrete example of nonlinear regression is examined carefully. Special attention is given to the individual intervals of confidence and the combined region of confidence for the optimal parameters.

Many growth phenomena can be described by rescaling and shifting the basic logistic<sup>27</sup> growth function  $g(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}$ . It is easy to see that this function is monotonically increasing and

$$\lim_{x \rightarrow -\infty} g(x) = 0 \quad , \quad g(0) = \frac{1}{2} \quad \text{and} \quad \lim_{x \rightarrow +\infty} g(x) = 1 .$$

By shifting and rescaling examine the modified logistic function

$$f(x) = p_1 + p_2 g(p_3 (x - p_4)) = p_1 + \frac{p_2}{1 + \exp(-p_3 (x - p_4))} \quad (3.24)$$

with the four parameters  $p_i$ ,  $i = 1, 2, 3, 4$ . An example is shown in Figure 3.57. For the given data points (in Ex. 3.18 red) the optimal values for the parameters  $p_i$  have to be determined. This is a nonlinear regression problem.

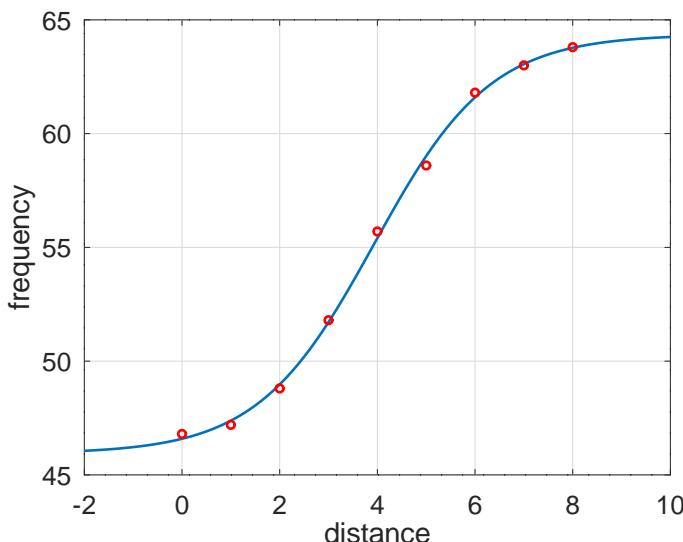


Figure 3.57: Data points and the optimal fit by a logistic function

<sup>27</sup>Also called sigmoid function

An essential point for a nonlinear regression problems is to find good estimates for the values of the parameters. Thus examine the graph of the logistic function (3.24) carefully:

- At the midpoint  $x = p_4$  find  $f(p_4) = p_1 + p_2 \frac{1}{2}$ .
- For the extreme values observe  $\lim_{x \rightarrow -\infty} f(x) = p_1$  and  $\lim_{x \rightarrow +\infty} f(x) = p_1 + p_2$ .
- The maximal slope is at the midpoint and given by<sup>28</sup>  $f'(p_4) = \frac{p_2 p_3}{4}$ .

Assuming  $p_2, p_3 > 0$  now find good estimates for the parameter values.

- $p_1$  offset: minimal height of the data points
- $p_2$  amplitude: difference of maximal and minimal value
- $p_3$  slope: the maximal slope is  $m = \frac{p_2 p_3}{4}$  and thus  $p_3 = \frac{4m}{p_2}$
- $p_4$  midpoint: average of  $x$  values

Based on this use the code below to determine the estimated values.

```

x_data = [0 1 2 3 4 5 6 7 8]';
y_data = [46.8 47.2 48.8 51.8 55.7 58.6 61.8 63 63.8]';

p1 = min(y_data);
p2 = max(y_data)-min(y_data);
p3 = 4*max(diff(y_data))./diff(x_data))/p2;
p4 = mean(x_data);

```

This result can now be used to apply a nonlinear regression, using the functions `leasqr()`, `fsolve()` or `lsqcurvefit()`.

### Solution by `leasqr()`, Octave and MATLAB

To determine the optimal values of the parameters:

- Define the logistic function, with the four parameters  $p_i$ .
- Call `leasqr()`, returning the values and the covariance matrix. On the diagonal of the covariance matrix find the estimated variances of the parameters  $p_i$ .

Find the result in Figure 3.57. As numerical result the optimal values of  $p_i$  and their standard deviations are shown. These can be used to determine confidence intervals for the parameters. In addition the number of required iterations and the resulting residual ( $\sum_{i=1}^n (f(x_i) - y_i)^2$ )<sup>1/2</sup> is displayed.

```

f = @(x,p) p(1) + p(2)*exp(p(3)*(x-p(4)))./(1+exp(p(3)*(x-p(4)))); 
[fr, p,~, iter, corp, covp] = leasqr(x_data,y_data,[p1,p2,p3,p4],f);
sigma = sqrt(diag(covp));
optimal_values = [p'; sigma']
iter_residual = [iter,norm(fr-y_data)]
x = linspace(-2,10);
figure(1); plot(x,f(x,p),x_data,y_data,'or')
 xlabel('distance'); ylabel('frequency')
-->
optimal_values =
    45.931829    18.428664    0.838742    3.932786
    0.380858    0.645210    0.062353    0.080993
iter_residual =
    4    0.64832

```

<sup>28</sup>For  $g(x) = \frac{1}{1+\exp(-x)}$  use  $g'(0) = \frac{1}{4}$  and then some rescaling to determine  $f'(p_4)$ .

Using the estimated variances of the individual parameters  $p_i$  using the  $t$ -distribution the 95% confidence interval for the individual parameters can be estimated.

```
p_CI_95 = p + tinv(1-0.05/2,length(x_data)-4) * [-sigma +sigma]
-->
p_CI_95 =
  44.9528   46.9109
  16.7701   20.0872
  0.6785    0.9990
  3.7246    4.1410
```

But since the correlation matrix `corp` contains large off-diagonal entries the result might not be reliable.

```
corp =
  1.000000 -0.873224  0.787091  0.453505
 -0.873224  1.000000 -0.904347 -0.043361
  0.787091 -0.904347  1.000000  0.034882
  0.453505 -0.043361  0.034882  1.000000
```

The large off-diagonal entries indicate that the four parameters  $p_i$  are not independent.

### Domain of confidence

To examine the joint domain of confidence for the four parameters  $p_i$  some more computations have to be performed. There are two methods to visualize the joint domain of confidence;

1. as a rectangular box in  $\mathbb{R}^4$
2. as an ellipsoid in  $\mathbb{R}^4$

For the individual confidence intervals with level of significance  $\alpha = 0.05$  use the Student  $t$ -distribution by calling `tinv(1-0.05/2, n-4)`. If the "box" in  $\mathbb{R}^4$  would be constructed with these widths, then the confidence of the true parameter to be inside the box would be  $(1 - \alpha)^4 = (1 - 0.05)^4 \approx 0.81$ . For a level of confidence  $1 - \alpha$  for all four parameters together one needs  $p_4^4 = (1 - \alpha)$  or  $p_4 = (1 - \alpha)^{1/4}$ . This leads to the level of significance

$$\alpha_4 = 1 - p_4 = 1 - (1 - \alpha)^{1/4} \approx \alpha/4 .$$

```
alpha4 = 1-(1-alpha)^0.25;
p_CI_95_joint = p + tinv(1-alpha4/2,length(x_data)-4) * [-sigma +sigma]
-->
p_CI_95_joint =
  44.4879   47.3758
  15.9825   20.8748
  0.6023    1.0751
  3.6257    4.2399
```

The result is a clearly larger domain of confidence for the joint parameters, it is a rectangular "box" in  $\mathbb{R}^4$ .

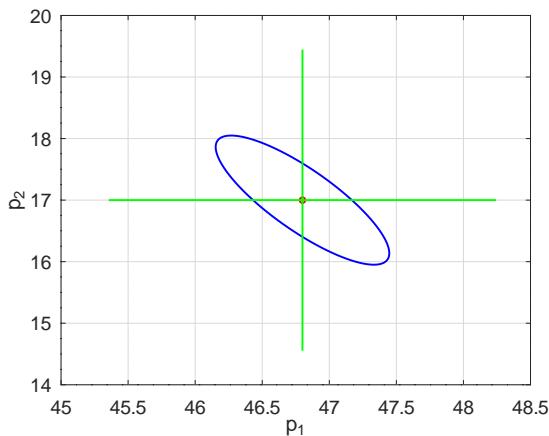
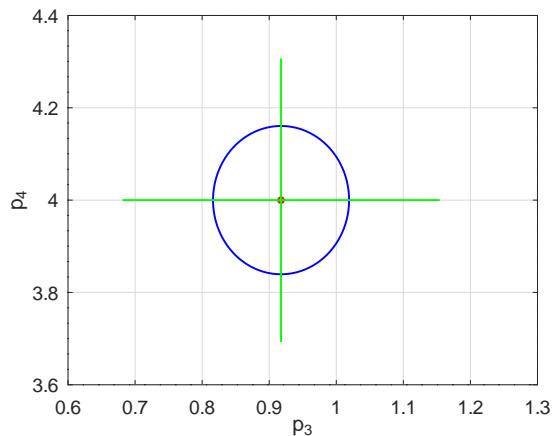
Caused by the correlation of the parameters  $p_i$  one should examine the ellipsoidal domain of confidence in  $\mathbb{R}^4$ . The intersection of the domain in  $\mathbb{R}^4$  with the plane  $p_3 = p_4 = \text{const}$  is an ellipse in the plane with  $p_1$  and  $p_2$ . To determine the dimensions of the ellipses the F-distribution has to be used, see [DrapSmit98] or [RawlPantuDick98]. Find the result in Figure 3.58(a). The result of the intersection with  $p_1 = p_2 = \text{const}$  is shown in Figure 3.58(b). Also shown are the central axis for the rectangular domains of confidence in green. Use the results in Section 3.2.7 and Example 3-25 to draw the ellipses.

```

inv_cov = inv(covp); inv_cov = inv_cov(1:2,1:2); % examine p1 and p2
[Evec,Eval] = eig(inv_cov); angle = linspace(0,2*pi);
Points = Evec*inv(sqrt(Eval))*[cos(angle);sin(angle)];

f_limit = 4*finv(1-alpha,4,length(x_data));
figure(3); plot(p1+sqrt(f_limit)*Points(1,:),p2+sqrt(f_limit)*Points(2,:),'b',...
    'linewidth',2)
t_lim = tinv(1-alpha/2,length(x_data)-4)
hold on
plot(p1,p2,'or')
plot([p1-t_lim*sigma(1),p1+t_lim*sigma(1)], [p2,p2], 'g')
plot([p1,p1], [p2-t_lim*sigma(2),p2+t_lim*sigma(2)], 'g')
xlabel('p_1'); ylabel('p_2'); hold off

```

(a) intersection with  $p_3 = p_4 = \text{const}$ (b) intersection with  $p_1 = p_2 = \text{const}$ Figure 3.58: The intersection of the 95% confidence ellipsoid in  $\mathbb{R}^4$  with 2D-planes

The intersection of the confidence domain in  $\mathbb{R}^4$  with the plane  $p_4 = \text{const}$  leads to an ellipsoid in  $\mathbb{R}^3$ , shown in Figure 3.59. The intersection of this ellipsoid with the horizontal plane  $p_3 = \text{const}$  generated by the green markers leads to the ellipse in Figure 3.58(a). On occasion it might also be useful to use the projections of the general ellipsoids on coordinate planes, using the ideas leading to Example 3–27 on page 140.

```

inv_cov = inv(covp); inv_cov = inv_cov([1 2 3],[1 2 3]);
[Evec,Eval] = eig(inv_cov);
phi = linspace(0,2*pi,81); theta = linspace(-pi,pi,81);
x = cos(phi')*cos(theta); y = sin(phi')*cos(theta); z = ones(size(phi'))*sin(theta);
Points = Evec*inv(sqrt(Eval))*[x(:,1),y(:,1),z(:,1)]';

f_limit = 4*finv(1-alpha,4,length(x_data));
figure(4); plot3(p1+sqrt(f_limit)*Points(1,:),p2 + sqrt(f_limit)*Points(2,:),...
    p3 + sqrt(f_limit)*Points(3,:),'b')
hold on
plot3(p1,p2,p3,'or')
plot3([p1-t_lim*sigma(1),p1+t_lim*sigma(1)], [p2,p2], [p3,p3], 'g')
plot3([p1,p1], [p2-t_lim*sigma(2),p2+t_lim*sigma(2)], [p3,p3], 'g')
plot3([p1,p1], [p2,p2], [p3-t_lim*sigma(3),p3+t_lim*sigma(3)], 'g')
xlabel('p_1'); ylabel('p_2'); zlabel('p_3'); view([-115,15]); hold off

```

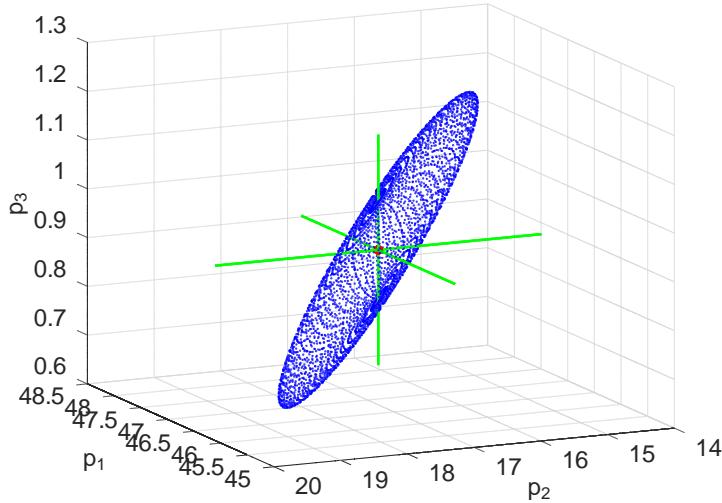


Figure 3.59: The intersection of the 95% confidence ellipsoid in  $\mathbb{R}^4$  with the  $p_4 = \text{const}$  plane

### Solution by `nonlin_curvefit()`, Octave only

With the command `nonlin_curvefit()` the method of nonlinear least squares can be used to fit a function to data points. A solution for the above problem is given by

#### Octave

```
f = @(p,x) p(1) + p(2)*exp(p(3)*(x-p(4)))./(1+exp(p(3)*(x-p(4))));  
[p,fr,convergence_flag,outp] = nonlin_curvefit (f,[p1;p2;p3;p4], x_data, y_data);  
optimal_values = p'  
iter_residual = [outp.niter,norm(fr-y_data)]  
-->  
optimal_values = 4.5932e+01 1.8429e+01 8.3874e-01 3.9328e+00  
iter_residual = 4 0.6483
```

The result for the optimal parameters  $p_i$  is identical to the previous results. In addition the number of required iterations and the resulting residual are displayed. The command `curvefit_stat()` will determine the covariance matrix `covp` and the correlation matrix `corp`.

```
settings = optimset ('ret_covp',true,'ret_corp',true,'objf_type','wls')  
FitInfo = curvefit_stat (f,p,x_data,y_data,settings)  
sigma = sqrt(diag(FitInfo.covp))  
-->  
FitInfo = scalar structure containing the fields:  
covp = 1.4505e-01 -2.1458e-01 1.8692e-02 1.3989e-02  
       -2.1458e-01 4.1630e-01 -3.6383e-02 -2.2660e-03  
       1.8692e-02 -3.6383e-02 3.8879e-03 1.7616e-04  
       1.3989e-02 -2.2660e-03 1.7616e-04 6.5599e-03  
corp = 1.000000 -0.873224 0.787091 0.453505  
       -0.873224 1.000000 -0.904347 -0.043361  
       0.787091 -0.904347 1.000000 0.034882  
       0.453505 -0.043361 0.034882 1.000000  
sigma = 3.8086e-01 6.4521e-01 6.2353e-02 8.0993e-02
```

With this information the analysis of the domain of confidence can be performed, identical to the results by `leasqr()`.

### Solution by `fsolve()`, MATLAB and Octave

The command `fsolve()` is used to solve systems of nonlinear equations. If more data points than parameters are given (more equations than unknowns), then a nonlinear least square solution is determined. Thus solve the above problem using `fsolve()`.

```
f2 = @(p) p(1) + p(2)*exp(p(3)*(x_data-p(4)))./(1+exp(p(3)*(x_data-p(4))))-y_data;
[p,fval] = fsolve(f2,[p1,p2,p3,p4]);
optimal_values = p
residual = norm(fval)
-->
optimal_values = 45.93183 18.42866 0.83874 3.93279
residual = 0.64832
```

It is no surprise that the same result is found. `fsolve()` does not estimate standard deviations for the parameters. One might use `nlpaci()` to determine confidence intervals.

### Solution by `lsqcurvefit()`, MATLAB and Octave

With the command `lsqcurvefit()` the method of nonlinear least squares can be used to fit a function to data points. A solution for the above problem is given by

```
f3 = @(p,x_data) p(1) + p(2)*exp(p(3)*(x_data-p(4)))./(1+exp(p(3)*(x_data-p(4))));
[p,residual] = lsqcurvefit(f3,[p1,p2,p3,p4],x_data,y_data)
optimal_values = p'
residual = sqrt(residual)
-->
optimal_values = 45.93183 18.42866 0.83874 3.93279
residual = 0.64832
```

It is no surprise that the same result is found. `lsqcurvefit()` does not estimate standard deviations for the parameters. The command `lsqcurvefit()` can return more results, e.g. the residual or the Jacobian matrix with the partial derivatives with respect to the parameters.

### 3.5.9 Additional Commands from the Package `optim` in Octave

The package `optim` in Octave (see <https://gnu-octave.github.io/packages/optim/>) provides additional commands for linear and nonlinear regression problems.

- `lsqlin`: linear least square with linear constraints
- `expfit`: Prony's method for non-linear exponential fitting
- `polyfit`: polynomial fitting, MATLAB and Octave
- `wpolyfit`: polynomial fitting
- `polyconf`: confidence and prediction intervals for polynomial fitting, uses `wpolyfit`
- `polyfitinf`: polynomial fitting with the maximum norm

## 3.6 Resources

- **Nonlinear Equations** : based on my notes

- **Eigenvalues and Eigenvectors**

The bible of matrix computations is clearly [GoluVanLoan13] or one of the earlier editions. For general matrix analysis [HornJohn90] could be useful.

- **Numerical Integration**

- Some information from [IsaaKell66, §7] was used.
- In [RalsRabi78] rigorous proofs for some of the error estimates are shown.
- Find more information on basic integration methods in [YounGreg72].

- **Solving Ordinary Differential Equations, Initial Value Problems**

- Detailed information on ODE solvers can be found in the book [Butc03] by John Butcher ([Butc16] is a newer edition) or in [HairNorsWann08], [HairNorsWann96].
- Use the article `ode_suite.pdf` ([ShamReic97]) by Shampine and Reichelt for excellent information on the MATLAB ODE solvers. The document is available at the Mathwork web site at [https://www.mathworks.com/help/pdf\\_doc/otherdocs/ode\\_suite.pdf](https://www.mathworks.com/help/pdf_doc/otherdocs/ode_suite.pdf).
- Information on different Runge–Kutta methods is available on Wikipedia.  
[https://en.wikipedia.org/wiki/List\\_of\\_Runge-Kutta\\_methods](https://en.wikipedia.org/wiki/List_of_Runge-Kutta_methods)  
[https://en.wikipedia.org/wiki/Runge-Kutta\\_methods](https://en.wikipedia.org/wiki/Runge-Kutta_methods)
- Marc Compere provided codes that (should) work with MATLAB and Octave. It is available in a repository at [https://gitlab.com/comperem/ode\\_solvers](https://gitlab.com/comperem/ode_solvers).
  - \* `rk2fixed.m`, `rk4fixed.m` and `rk8fixed.m` are explicit Runge–Kutta methods with fixed step size of order 2, 4 and 8 .
  - \* `ode23.m`, `ode45.m` and `ode78.m` are explicit Runge–Kutta methods with adaptive step sizes.

- **Linear and Nonlinear Regression, Curve Fitting**

Some of the information in these notes is taken from the notes [Octave07] for a class on MATLAB/Octave, available at [OctaveAtBFH.pdf](#) or from a statistics class with the supporting notes [Stah16], available at [StatisticsWithMatlabOctave.pdf](#).

In the previous chapter the codes in Table 3.19 were used.

## Bibliography

- [AbraSteg] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, 1972.
- [Agga20] C. Aggarwal. *Linear Algebra and Optimization for Machine Learning*. Springer, first edition, 2020.
- [AmreWihl14] M. Amrein and T. Wihler. An adaptive Newton-method based on a dynamical systems approach. *Communications in Nonlinear Science and Numerical Simulation*, 19(9):2958–2973, 2014.
- [Axel94] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, 1994.
- [Butc03] J. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, second edition, 2003.
- [Butc16] J. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, third edition, 2016.

filename	description
<b>nonlinear equations</b>	
NewtonSolve.m	function file to apply Newton's method
exampleSystem.m	first example of a system of equations, Example 3–8
Newton2D.m	code to visualize Example 3–8
testBeam.m	code to solve Example 3–13
Keller.m	script file to solve Example 3–14
tridiag.m	MATLAB function to solve tridiagonal systems
<b>eigenvalues and eigenvectors</b>	
ReduceQuadraticForm.m	function used in Example 3–27
<b>numerical integration</b>	
simpson.m	integration using Simpson's rule
IntegrateGauss.m	integration using the Gauss approach
<b>ordinary differential equations, ODEs</b>	
ode_RungeKutta	Runge–Kutta method with fixed step size
ode_Heun	Heun method with fixed step size
ode_Euler	Euler method with fixed step size
Pendulum.m	Pendulum demo for the algorithms with fixed step size
rk23.m	an adaptive algorithm, based on the method of Heun
rk45.m	an adaptive algorithm, based on the method of Runge–Kutta
Test_rk23_rk45.m	demo code for rk23 and rk45 in Example 3–66
<b>linear and nonlinear regression</b>	
LinearRegression.m	linear regression, Octave version
LinearRegression.m.Matlab	linear regression, MATLAB version
ExampleLinReg.m	code to generate Section 3.5.4
leasqr.m	MATLAB version of the command leasqr()
dfdp.m	MATLAB version of the support file for leasqr()

Table 3.19: Codes for chapter 3

- [Deim84] K. Deimling. *Nonlinear Functional Analysis*. Springer Verlag, 1984.
- [DeisFaisOng20] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. pre-publication.
- [Demm97] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.
- [DrapSmit98] N. Draper and H. Smith. *Applied Regression Analysis*. Wiley, third edition, 1998.
- [GoluVanLoan96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.
- [GoluVanLoan13] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2013.
- [HairNorsWann08] E. Hairer, S. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Non-stiff Problems*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, second edition, 1993. third printing 2008.
- [HairNorsWann96] E. Hairer, S. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Lecture Notes in Economic and Mathematical Systems. Springer, second edition, 1996.
- [HornJohn90] R. Horn and C. Johnson. *Matrix Analysis*. Cambridge University Press, 1990.
- [HuntLipsRose14] B. R. Hunt, R. L. Lipsman, and J. M. Rosenberg. *A Guide to MATLAB: For Beginners and Experienced Users*. Cambridge University Press, New York, NY, USA, third edition, 2014.
- [IsaaKell66] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. John Wiley & Sons, 1966. Republished by Dover in 1994.
- [Kell92] H. B. Keller. *Numerical Methods for Two-Point Boundary Value Problems*. Dover, 1992.
- [Linz79] P. Linz. *Theoretical Numerical Analysis*. John Wiley& Sons, 1979. Republished by Dover.
- [MeybVach91] K. Meyberg and P. Vachenauer. *Höhere Mathematik II*. Springer, Berlin, 1991.
- [MontRung03] D. Montgomery and G. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, third edition, 2003.
- [DLMF15] N. I. of Standards. NIST Digital Library of Mathematical Functions, at <http://dlmf.nist.gov>, 2015.
- [Pres92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.
- [RalsRabi78] A. Ralston and P. Rabinowitz. *A first Course in Numerical Analysis*. McGraw–Hill, second edition, 1978. republished by Dover in 2001.
- [RawlPantuDick98] J. Rawlings, S. Pantula, and D. Dickey. *Applied regression analysis*. Springer texts in statistics. Springer, New York, 2. ed edition, 1998.
- [SchnWihl11] H. R. Schneebeli and T. Wihler. The Netwon-Raphson Method and Adaptive ODE Solvers. *Fractals*, 19(1):87–99, 2011.
- [ShamReic97] L. Shampine and M. W. Reichelt. The MATLAB ODE Suite. *SIAM Journal on Scientific Computing*, 18:1–22, 1997.
- [Stah00] A. Stahel. Calculus of Variations and Finite Elements. supporting notes, 2000.

- [Octave07] A. Stahel. Octave and Matlab for Engineering Applications. lecture notes, 2007.
- [Stahel16] A. Stahel. Statistics with Matlab/Octave. supporting notes, BFH-TI, 2016.
- [TongRoss08] P. Tong and J. Rossettos. *Finite Element Method, Basic Technique and Implementation*. MIT, 1977. Republished by Dover in 2008.
- [YounGreg72] D. M. Young and R. T. Gregory. *A Survey of Numerical Analysis, Volume 1*. Dover Publications, New York, 1972.

## Chapter 4

# Finite Difference Methods

### 4.1 Prerequisites and Goals

In this chapter we examine one of the methods to replace differential equations by approximating systems of difference equations. We replace the continuous equation by a discrete system of equations. These equations are then solved, using the techniques from previous chapters. The goal is to find approximate solutions that are close to the exact solution. One of the possible standard references is [Smit84]. A more detailed presentation is given in [Thom95], where you can find state of the art techniques.

After having worked through this chapter

- you should understand the basic concept of a finite difference approximation and finite difference stencils.
- should be familiar with the concepts of consistency, stability and convergence of a finite difference approximation.
- should know about conditional and unconditional stability of solvers.
- should be able to set up and solve second order linear boundary value problems on intervals and rectangles.
- should be able to set up and solve second order linear initial boundary value problems on intervals.
- should be able to set up and solve some nonlinear boundary value problems with the help of a finite difference approximation.

In this chapter we assume that you are familiar with

- the basic idea and definition of a derivative.
- the concept of ordinary differential equations, in particular with  $\dot{y}(t) = -\lambda y(t)$ .
- the representation of a vector as a linear combination of eigenvectors.

### 4.2 Basic Concepts

#### 4.2.1 Finite Difference Approximations of Derivatives

Instead of solving a differential equation replace the derivatives by approximate difference formulas, based on the definition of a derivative

$$\frac{d}{dt} y(t) = \lim_{h \rightarrow 0} \frac{y(t+h) - y(t)}{h},$$

by selecting a finite value of  $0 < h \ll 1$ , leading to

$$\frac{d}{dt} y(t) \approx \frac{y(t+h) - y(t)}{h}.$$

Other approximations to the first derivative are possible, using similar ideas and computations. This leads to the formulas and stencils in Figure 4.1 .

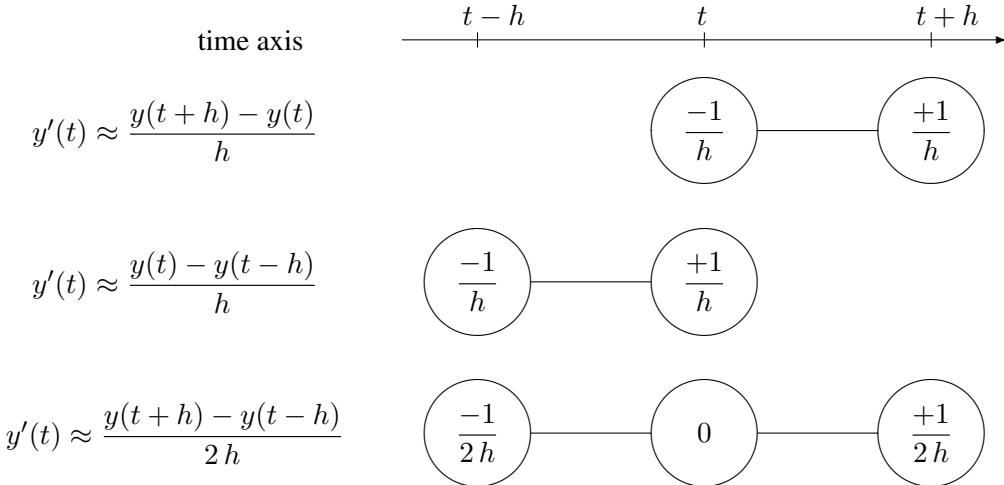


Figure 4.1: FD stencils for  $y'(t)$ , forward, backward and centered approximations

In Figure 4.2 use the values of the function at the grid points  $t - h$ ,  $t$  and  $t + h$  to find formulas for the first and second order derivatives. The second derivative is examined as derivative of the derivative. The above observations hint towards the following approximate formulas.

$$\begin{aligned} \frac{d}{dt} y(t) &\approx \frac{y(t+h) - y(t)}{h} \approx \frac{y(t) - y(t-h)}{h} \approx \frac{y(t+h) - y(t-h)}{2h} \\ \frac{d}{dt} y(t) &\approx \frac{y(t+h/2) - y(t-h/2)}{h} \\ \frac{d^2}{dt^2} y(t) &\approx \frac{y'(t+h/2) - y'(t-h/2)}{h} \approx \frac{1}{h} \left( \frac{y(t+h) - y(t)}{h} - \frac{y(t) - y(t-h)}{h} \right) \\ &= \frac{y(t-h) - 2y(t) + y(t+h)}{h^2} \end{aligned}$$

The quality of the above approximations is determined by the error. For smaller values of  $h > 0$  the error should be as small as possible. To determine the size of this error use the Taylor approximation

$$y(t+x) = y(t) + y'(t) \cdot x + \frac{y''(t)}{2} x^2 + \frac{y'''(t)}{3!} x^3 + \frac{y^{(4)}(t)}{4!} x^4 + O(x^5)$$

with different values for  $x$  (use  $x = \pm h$  with  $|h| \ll 1$ ) and verify that

$$\begin{aligned} y(t+h) &= y(t) + y'(t) \cdot h + \frac{y''(t)}{2} h^2 + \frac{y'''(t)}{3!} h^3 + O(h^4) \\ y(t-h) &= y(t) - y'(t) \cdot h + \frac{y''(t)}{2} h^2 - \frac{y'''(t)}{3!} h^3 + O(h^4) \\ y(t+h) - y(t-h) &= 2y'(t) \cdot h + 2 \frac{y''(t)}{3!} h^3 + O(h^4) \\ \frac{y(t+h) - y(t-h)}{2h} - y'(t) &= \frac{y''(t)}{3!} h^2 + O(h^3) = O(h^2) \end{aligned}$$

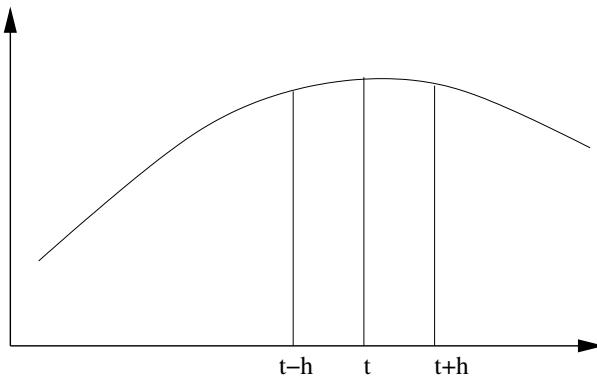


Figure 4.2: Finite difference approximations of derivatives

and thus conclude<sup>1</sup>

$$y'(t) = \frac{y(t+h) - y(t-h)}{2h} + O(h^2).$$

With computations very similar to the above find the finite difference approximations for the second order derivative by

$$y''(t) = \frac{y(t-h) - 2y(t) + y(t+h)}{h^2} + \frac{2}{4!} y^{(4)}(t) h^2 + O(h^3). \quad (4.1)$$

Similarly find the formulas for other derivatives in Table 4.1 . This table also indicates that the error of the centered difference formula is smaller than for the forward or backward formulas. These finite difference approximations are often visualized with the help of stencils, as shown in Figure 4.1 .

Ex 4.3–4.5

forward difference	$y'(t) = \frac{y(t+h) - y(t)}{h} + O(h)$
backward difference	$y'(t) = \frac{y(t) - y(t-h)}{h} + O(h)$
centered difference	$y'(t) = \frac{y(t+h/2) - y(t-h/2)}{h} + O(h^2)$
	$y''(t) = \frac{y(t-h) - 2y(t) + y(t+h)}{h^2} + O(h^2)$
	$y'''(t) = \frac{-y(t-h) + 3y(t) - 3y(t+h) + y(t+2h)}{h^3} + O(h)$
	$y'''(t) = \frac{-y(t-3h/2) + 3y(t-h/2) - 3y(t+h/2) + y(t+3h/2)}{h^3} + O(h^2)$
	$y'''(t) = \frac{-y(t-2h) + 2y(t-h) - 2y(t+h) + y(t+2h)}{2h^3} + O(h^2)$
	$y^{(4)}(t) = \frac{y(t-2h) - 4y(t-h) + 6y(t) - 4y(t+h) + y(t+2h)}{h^4} + O(h^2)$

Table 4.1: Finite difference approximations

With the above finite difference stencils replace derivatives by approximate finite differences, accepting a discretization error. As  $h$  converges to 0 we expect this error to approach 0. But in most cases small values of  $h$  will lead to larger arithmetic errors when performing the operations and this contribution will get larger as  $h$  approaches 0. For the total error the two contributions have to be added. This basic rule

$$\text{total error} = \text{discretization error} + \text{arithmetic error}$$

is illustrated in Figure 4.3. As a consequence do **not** expect to get arbitrary close to an error of 0. In this chapter only the discretization errors are carefully examined, assuming that the arithmetic error is negligible. This does not imply that rounding errors can safely be ignored, as illustrated in an exercise.

Ex 4.2

<sup>1</sup>Use the notation  $f(h) = O(h^n)$  to indicate  $|f(h)| \leq C h^n$  for some constant  $C$ . This indicates that the expression  $f(h)$  is of order  $h^n$  or less.

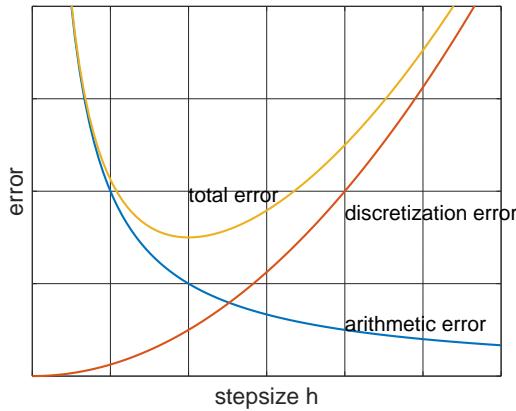


Figure 4.3: Discretization and arithmetic error contributions

#### 4.2.2 Finite Difference Stencils

Based on the above finite difference approximations define **finite difference stencils** for differential operators.

##### Finite difference stencil for a steady state problem

In Chapter 2.7.1 find the differential operator

$$-\Delta u = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2}$$

for most 2-dimensional steady state problems. Based on Table 4.1 obtain a simple finite difference approximation

$$-\Delta u(x, y) = \frac{-u(x-h, y) + 2u(x, y) - u(x+h, y)}{h^2} + \frac{-u(x, y-h) + 2u(x, y) - u(x, y+h)}{h^2} + O(h^2).$$

For the rectangular grid in Figure 4.4 set

$$u_{i,j} = u(x_j, y_i) = u(jh, ih)$$

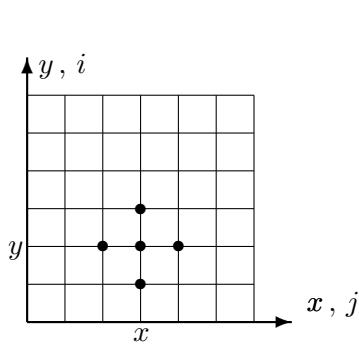
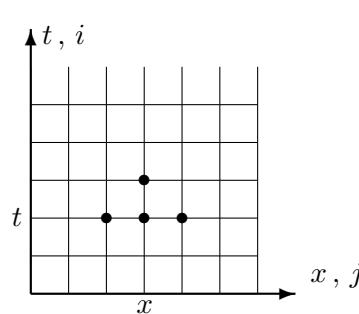
and then find

$$(-\Delta u)_{i,j} \approx \frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2}$$

##### Finite difference stencil for a dynamic heat problem

When trying to discretize the dynamic heat equation  $\frac{d}{dt}u(t, x) - u''(t, x) = f(t, x)$  use the notation  $u_{i,j} = u(t_i, x_j) = u(ih_t, jh_x)$  and the forward difference approximation for  $\frac{d}{dt}u$ .

$$\begin{aligned} \frac{d}{dt}u(t_i, x_j) &= \frac{u_{i+1,j} - u_{i,j}}{h_t} + O(h_t) \\ u''(t_i, x_j) &= \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_x^2} + O(h_x^2) \\ (\frac{d}{dt}u - u'')_{i,j} &\approx -\frac{1}{h_x^2}u_{i,j-1} + \left(\frac{2}{h_x^2} - \frac{1}{h_t}\right)u_{i,j} - \frac{1}{h_x^2}u_{i,j+1} + \frac{1}{h_t}u_{i+1,j} \end{aligned}$$

Figure 4.4: Finite difference stencil for  $-u_{xx} - u_{yy}$  if  $h = h_x = h_y$ Figure 4.5: Finite difference stencil for  $u_t - u_{xx}$ , explicit, forward

This leads to the stencil in Figure 4.5, the **explicit** finite difference stencil for the heat equation.

If the backward difference approximation is used for the time derivative  $\frac{d}{dt} u$  find the **implicit** finite difference stencil in Figure 4.6.

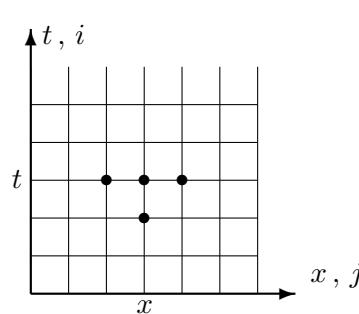
### 4.3 Consistency, Stability and Convergence

In this section we first examine a finite difference approximation for an elementary ordinary differential equation. The results and consequences will apply to considerably more difficult problems.

#### 4.3.1 A Finite Difference Approximation of an Initial Value Problem

Consider the ordinary differential equation for  $\lambda > 0$ .

$$\frac{d}{dt} y(t) = -\lambda y(t) \quad \text{with} \quad y(0) = y_0 .$$

Figure 4.6: Finite difference stencil for  $u_t - u_{xx}$ , implicit, backward

The exact solution is given by  $y(t) = y_0 e^{-\lambda t}$ . Obviously the solution is bounded on any interval on  $\mathbb{R}_+$  and we expect its numerical approximation to remain bounded too, independent on the final time  $T$ .

To visualize the context consider a similar problem  $\frac{d}{dt} y(t) + \lambda y(t) = f(t)$  for a given function  $f(t)$ . This problem can be discretized with stepsize  $h$  at the grid points  $t_i = i h$  for  $0 \leq i \leq N - 1$ . This will lead to an approximation on the interval  $[0, T] = [0, (N - 1) h]$ . The unknown function  $y(t)$  in the interval  $t \in [0, T]$  is replaced by a vector  $\vec{y} \in \mathbb{R}^N$  and the function  $f(t)$  is replaced by a vector  $\vec{f} \in \mathbb{R}^N$ .

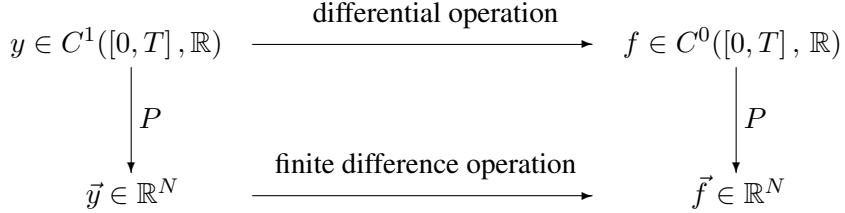


Figure 4.7: A finite difference approximation of an initial value problem

### 4.3.2 Explicit Method, Conditional Stability

When using the forward difference method in Table 4.1 to solve  $\frac{d}{dt} y(t) + \lambda y(t) = 0$  with  $y_i = y(t)$  and  $y_{i+1} = y(t + h)$  find

$$\frac{d}{dt} y(t) + \lambda y(t) \approx \frac{y_{i+1} - y_i}{h} + \lambda y_i = 0$$

and the difference will converge to 0 as the stepsize  $h$  approaches 0. This will be called **consistency** of the finite difference approximation. The differential equation is replaced by the difference equation

$$\begin{aligned} \frac{y_{i+1} - y_i}{h} &= -\lambda y_i \\ y_{i+1} &= y_i - h \lambda y_i = (1 - h \lambda) y_i. \end{aligned}$$

In Result 3–56 (page 196) it is verified the solution of this difference equation satisfies  $\lim_{i \rightarrow \infty} y_i = 0$  if and only if  $|1 - h \lambda| < 1$ . Since  $\lambda$  and  $h$  are positive this leads to the condition

$$h \lambda < 2 \iff h < \frac{2}{\lambda}.$$

This is an example of **conditional stability**, i.e. the schema is only stable if the above condition on the stepsize  $h$  is satisfied.

### 4.3.3 Implicit Method, Unconditional Stability

We may also use the backward difference method in Table 4.1

$$\frac{d}{dt} y(t) + \lambda y(t) \approx \frac{y_i - y_{i-1}}{h} + \lambda y_i = 0$$

and the difference will converge to 0 as the stepsize  $h$  approaches 0. Thus this scheme is also consistent. The differential equation is replaced by

$$\frac{y_i - y_{i-1}}{h} = -\lambda y_i \implies (1 + h \lambda) y_i = y_{i-1}$$

One can verify that this difference equation is solved by

$$y_i = y_0 \frac{1}{(1 + h \lambda)^i}.$$

In Result 3–57 (page 197) it is verified the solution of this difference equation satisfies  $\lim_{i \rightarrow \infty} y_i = 0$  for  $\lambda > 0$  and we have **unconditional stability**.

### 4.3.4 General Difference Approximations, Consistency, Stability and Convergence

To explain the approximation behavior of finite difference schemes use the example problem

$$-u''(x) = f(x) \quad \text{for } 0 < x < L \quad \text{with boundary conditions } u(0) = u(L) = 0. \quad (4.2)$$

Assume that for a given function  $f(x)$  the exact solution is given by the function  $u(x)$ . The differential equation is replaced by a difference equation. For  $n \in \mathbb{N}$  discretize the interval by  $x_k = k \cdot h = k \frac{L}{n+1}$  and then consider an approximate solution  $u_k \approx u(k \cdot h)$  for  $k = 0, 1, 2, \dots, n, n+1$ . The finite difference approximation of the second derivative in Table 4.1 leads for interior points to

$$-\frac{u_{k-1} - 2u_k + u_{k+1}}{h^2} = f_k = f(k \cdot h) \quad \text{for } k = 1, 2, 3, \dots, n. \quad (4.3)$$

The boundary conditions are taken into account by  $u_0 = u_{n+1} = 0$ . These linear equations can be written in the form

$$\frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix}.$$

The solution of this linear system will create the values of the approximate solution at the grid points. Exact and approximate solution are shown in Figure 4.8. As  $h \rightarrow 0$  we hope that  $u$  will converge to the exact solution  $u(x)$ .

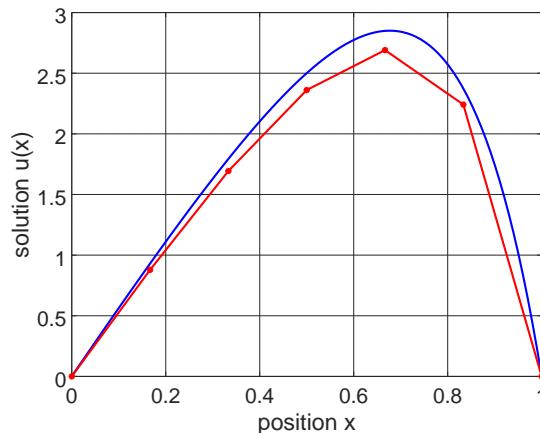


Figure 4.8: Exact and approximate solution of a boundary value problem

To examine the behavior of the approximate solution use a general framework for finite difference approximations to boundary value problems. Examine Figure 4.9 to observe how a differential equation is replaced by an approximate system of linear equations. The similar approach for general problems is shown in Figure 4.10.

Consider functions defined on a domain  $\Omega \subset \mathbb{R}^N$  and for a fixed mesh size  $h$  cover the domain with a discrete set of points  $x_k \in \Omega$ . This leads to the following vector spaces:

- $E_1$  is a space of functions defined on  $\Omega$ . In the above example consider  $u \in C^2([0, L], \mathbb{R})$  with  $u(0) = u(L) = 0$ . On this space use the norm  $\|u\|_{E_1} = \max\{|u(x)| : 0 \leq x \leq L\}$ .

$$\begin{array}{ccc}
 u \in C^2([0, L], \mathbb{R}) & \xrightarrow{-\frac{\partial^2}{\partial x^2}} & f \in C^0([0, L], \mathbb{R}) \\
 \downarrow P_h & & \downarrow P_h \\
 \vec{u}_h \in \mathbb{R}^N & \xrightarrow{\mathbf{A}_h \cdot} & \vec{f}_h \in \mathbb{R}^N
 \end{array}$$

Figure 4.9: An approximation scheme for  $-u''(x) = f(x)$ 

- $E_2$  is a space of functions defined on  $\Omega$ . In the above example consider  $f \in C^0([0, L], \mathbb{R})$  with the norm  $\|f\|_{E_2} = \max\{|f(x)| : 0 \leq x \leq L\}$ .
- $E_1^h$  is a space of discretized functions. In the above example consider  $\vec{u} \in \mathbb{R}^n = E_1^h$ , where  $u_k = u(k \cdot h)$ . The vector space  $E_1^h$  is equipped with the norm  $\|\vec{u}\|_{E_1^h} = \max\{|u_k| : 1 \leq k \leq n\}$ .
- $E_2^h$  is also a space of discretized functions. In the above example consider  $\vec{f} \in \mathbb{R}^n = E_2^h$ , where  $f_k = f(k \cdot h)$ . The vector space  $E_2^h$  is equipped with the norm  $\|\vec{f}\|_{E_2^h} = \max\{|f_k| : 1 \leq k \leq n\}$ .

On these spaces we examine the following linear operations:

- For  $u \in E_1$  let  $\mathbf{F} : E_1 \rightarrow E_2$  be the linear differential operator. In the above example  $\mathbf{F}(u) = -u''$ .
- For  $\vec{u} \in E_1^h$  let  $\mathbf{F}_h : E_1^h \rightarrow E_2^h$  be the linear difference operator. In the above example

$$\mathbf{F}_h(\vec{u})_k = \frac{u_{k-1} - 2u_k + u_{k+1}}{h^2}$$

- For  $u \in E_1$  let  $\vec{u} = P_1^h(u) \in E_1^h$  be the projection of the function  $u \in E_1$  onto  $E_1^h$ . It is determined by evaluation the function at the points  $x_k$ .
- For  $f \in E_2$  let  $\vec{f} = P_2^h(f) \in E_2^h$  be the projection of the function  $f \in E_2$  onto  $E_2^h$ . It is determined by evaluation the function at the points  $x_k$ .

The above operations are illustrated in Figure 4.10. There is a recent article [KhanKhan18] on the importance of this structure of the fundamental spaces of numerical analysis.

$$\begin{array}{ccc}
 \mathbf{F} & & \\
 u \in E_1 \xrightarrow{} f \in E_2 & & h \rightarrow 0 \\
 \downarrow P_1^h & & \|P_1^h u\|_{E_1^h} \rightarrow \|u\|_{E_1} \\
 u_h \in E_1^h \xrightarrow{\mathbf{F}_h} f_h \in E_2^h & & \|P_2^h f\|_{E_2^h} \rightarrow \|f\|_{E_2} \\
 & & P_2^h(\mathbf{F}(u)) \approx \mathbf{F}_h(P_1^h(u))
 \end{array}$$

Figure 4.10: A general approximation scheme for boundary value problems

**4-1 Definition :** For a given  $f \in E_2$  let  $u \in E_1$  be the solution of  $\mathbf{F}(u) = f$  and  $\vec{u}_h$  the solution of  $\mathbf{F}_h(\vec{u}_h) = P_2^h(f)$ .

- The above approximation scheme is said to be **convergent** of order  $p$  if

$$\|P_1^h(u) - \vec{u}_h\|_{E_1^h} \leq c_1 h^p,$$

where the constant  $c_1$  is independent on  $h$ , but it may depend on  $u$ .

- The above approximation scheme is said to be **consistent** of order  $p$  if

$$\|\mathbf{F}_h(P_1^h(u)) - P_2^h(\mathbf{F}(u))\|_{E_2^h} \leq c_2 h^p,$$

where the constant  $c_2$  is independent on  $h$ , but it may depend on  $u$ . This implies that the diagram in Figure 4.10 is almost commutative as  $h$  approaches 0.

- The above approximation scheme is said to be **stable** if the linear operator  $\mathbf{F}_h \in \mathcal{L}(E_1^h, E_2^h)$  is invertible and there exists a constant  $M$ , independent on  $h$ , such that

$$\|u_h\|_{E_1^h} \leq M \|\mathbf{F}_h(u_h)\|_{E_2^h} \quad \text{for all } u_h \in E_1^h$$

This is equivalent to  $\|\mathbf{F}_h^{-1}\| \leq M$ , i.e. the inverse linear operators of the approximate problems are uniformly bounded.

For the above example the stability condition reads as  $\|\vec{u}\| \leq M \|\mathbf{A}_h \vec{f}\|$  or equivalently  $\|\mathbf{A}_h^{-1} \vec{f}\| \leq M \|\vec{u}\|$ , independent on  $h$ . This is thus the condition on the matrix norm of the inverse matrix to be independent on  $h$ , i.e.  $\|\mathbf{A}_h^{-1}\| \leq M$ .

Now state a fundamental result for finite difference approximations to differential equations. The theorem is also known as **Lax equivalence theorem**<sup>2</sup>. The result applies to a large variety of problems. We will examine only a few of them.

**4-2 Theorem :** If a finite difference scheme is consistent of order  $p$  and stable, then it is convergent of order  $p$ . A short formulation is:

*consistency and stability imply convergence*



**Proof :** Let  $u$  be the solution of  $\mathbf{F}(u) = f$  and  $\vec{u}$  the solution of  $\mathbf{F}_h(\vec{u}) = P_2^h(f) = P_2^h(\mathbf{F}(u))$ . Since the scheme is stable and consistent of order  $p$  we find

$$\begin{aligned} \|P_1^h(u) - \vec{u}\|_{E_1^h} &= \|\mathbf{F}_h^{-1} (\mathbf{F}_h(P_1^h(u)) - \vec{u})\|_{E_1^h} \\ &\leq \|\mathbf{F}_h^{-1}\| \|\mathbf{F}_h(P_1^h(u)) - \mathbf{F}_h(\vec{u})\|_{E_2^h} \\ &\leq M \|\mathbf{F}_h(P_1^h(u)) - P_2^h(\mathbf{F}(u))\|_{E_2^h} \\ &\leq M c h^p. \end{aligned}$$

Thus the finite difference approximation scheme is convergent. □

Table 4.2 illustrates the abstract concept using the example equation (4.2).

**4-3 Result :** To verify convergence of the solution of the finite difference of approximation of equation (4.2) to the exact solution we have to assure that the scheme is consistent and stable. Use the finite difference approximation

$$-u''(x) = f(x) \quad \rightarrow \quad \frac{-u_{k-1} + 2u_k - u_{k+1}}{h^2} = f_k.$$

- Consistency:** According to equation (4.1) or Table 4.1 (page 255) the scheme is consistent of order 2.
- Stability:** Let  $\vec{u}$  be the solution of the equation (4.3) with right hand side  $\vec{f}$ . Then

$$\|\vec{u}\|_\infty = \max_{1 \leq k \leq n} \{|u_k|\} \leq \frac{L^2}{2} \max_{1 \leq k \leq n} \{|f_k|\} = \frac{L^2}{2} \|\vec{f}\|_\infty \quad \text{independent on } h. \quad (4.4)$$



<sup>2</sup>We only use the result that a consistent and stable scheme has to be convergent. Lax also showed that a consistent and convergent scheme has to be stable. Find a proof in [AtkiHan09].

	general problem	sample problem (4.2)
exact equation	$\mathbf{F}(u) = f$	$-u''(x) = f(x)$
approximate left hand side	$P_1^h(u) \in E_1^h$	$u_k = u(k \cdot h)$
approximate right hand side	$P_2^h(f) \in E_2^h$	$f_k = f(k \cdot h)$
difference expression	$\mathbf{F}_h(\vec{u})$	$\frac{-u_{k-1} + 2u_k - u_{k+1}}{h^2}$
approximate equation	$\mathbf{F}_h(\vec{u}) = P_2^h(f)$	$\frac{-u_{k-1} + 2u_k - u_{k+1}}{h^2} = f(k \cdot h)$
stability	$\ u_h\ _{E_1^h} \leq M \ \mathbf{F}_h(u_h)\ _{E_2^h}$	$\max\{ u_k \} \leq M \max\{ f_k \}$
convergence, as $h \rightarrow 0$	$\ P_1^h(u) - \vec{u}\ _{E_1^h} \rightarrow 0$	$\max\{ u(k \cdot h) - u_k \} \rightarrow 0$

Table 4.2: Exact and approximate boundary value problem

**Proof :** The proof of stability of this finite difference scheme is based on a discrete maximum principle<sup>3</sup>. Proceed in two stages.

- As a first step verify a discrete maximum principle. If  $f_k \leq 0$  for  $k = 0, 1, 2, \dots, n, (n+1)$  and

$$\frac{-u_{k-1} + 2u_k - u_{k+1}}{h^2} = f_k = f(k \cdot h) \leq 0 \quad \text{for } k = 1, 2, 3, \dots, n$$

then

$$\max_{0 \leq k \leq n+1} \{u_k\} = \max\{u_0, u_{n+1}\}.$$

For the continuous case this corresponds to functions with  $u''(x) \geq 0$  attaining the largest value on the boundary.

To verify the discrete statement assume that  $\max_{1 \leq k \leq n} \{u_k\} = u_i$  for some index  $1 \leq i \leq n$ . Then

$$\begin{aligned} -u_{i-1} + 2u_i - u_{i+1} &= +h^2 f_i \leq 0 \\ u_i &= \frac{1}{2} (u_{i-1} + u_{i+1}) + \frac{1}{2} h^2 f_i \leq u_i + 0. \end{aligned}$$

Thus find  $u_{i-1} = u_i = u_{i+1}$  and  $f_i = 0$ . The process can be repeated with indices  $i-1$  and  $i+1$  to finally obtain the desired estimate. The computations also imply that  $\vec{u} = \vec{0}$  is the only solution of the homogeneous problem, i.e. the square matrix has a trivial kernel. Using linear algebra this implies that the matrix representing  $\mathbf{F}_h$  is invertible.

- Use the vector  $\vec{v} \in \mathbb{R}^n$  defined by  $v_k = (k/h)^2 = \left(\frac{kL}{n+1}\right)^2$ . The vector corresponds to the discretization of the function  $v(x) = x^2$ . Verify that

$$\frac{-v_{k-1} + 2v_k - v_{k+1}}{h^2} = -2 \quad \text{for } k = 1, 2, 3, \dots, n.$$

Let  $C = \|\vec{f}\|_\infty = \max\{|f_k| : 1 \leq k \leq n\}$  and  $f_k^+ = f_k - C \leq 0$ . Then  $\vec{w}^+ = \vec{u} + \frac{C}{2} \vec{v}$  is the solution of  $\mathbf{F}_h(\vec{w}^+) = \vec{f}^+$  and based on the first part of the proof and  $u_0 = u_{n+1} = 0$  find

$$\max_{1 \leq k \leq n} \{w_k^+\} = \max_{1 \leq k \leq n} \{u_k + \frac{C}{2} v_k\} \leq \frac{C}{2} \max\{v_0, v_{n+1}\} = \frac{C}{2} L^2.$$

Since  $v_k \geq 0$ , this implies  $u_k \leq \frac{C}{2} L^2$ .

<sup>3</sup>Readers familiar with partial differential equations will recognize the maximum principle and the construction of sub- and super-solutions to obtain à priori bounds.

A similar argument with  $f_k^- = f_k + C \geq 0$  and  $\vec{w}^- = \vec{u} - \frac{C}{2} \vec{v}$  implies

$$\min_{1 \leq k \leq n} \{u_k - \frac{C}{2} v_k\} \geq -\frac{C}{2} L^2.$$

These two inequalities imply

$$-\frac{C L^2}{2} \leq u_k \leq \frac{C L^2}{2} \quad \text{for } k = 1, 2, 3, \dots, n$$

and thus the stability estimate (4.4). □

In this section some basic concepts were introduced and illustrated using one sample application. The above proof for stability of finite difference approximations to elliptic boundary value problems can be applied to two or higher dimensional problems, e.g. [Smit84, p. 255]. Further information can be found in many books on numerical methods to solve PDE's and also in [IsaaKell66, §9.5] and [Wlok82].

## 4.4 Boundary Value Problems

In a first section examine differential equations defined on an interval, then solve partial differential equations on rectangles in  $\mathbb{R}^2$ .

### 4.4.1 Two Point Boundary Value Problems

#### 4-4 Example : An elementary example

To examine the boundary value problem

Ex 4.12

$$-u''(x) = 2 - x \quad \text{on } 0 < x < 2 \quad \text{with } u(0) = u(2) = 0$$

use 4 internal points  $x_1 = 0.4$ ,  $x_2 = 0.8$ ,  $x_3 = 1.2$  and  $x_4 = 1.6$ . With  $u_i = u(x_i)$  the finite difference approximation is

$$-u''(x_i) \approx \frac{-u(x_{i-1}) + 2u(x_i) - u(x_{i+1})}{h^2} = \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2},$$

where  $h = \frac{2}{5} = 0.4$  and  $u(x_0) = u_0 = u(x_5) = u_5 = 0$ . Thus find four equations for 4 unknowns.

$$\begin{aligned} \frac{1}{0.4^2} (-0 + 2u_1 - u_2) &= 2 - x_1 = 1.6 \\ \frac{1}{0.4^2} (-u_1 + 2u_2 - u_3) &= 2 - x_2 = 1.2 \\ \frac{1}{0.4^2} (-u_2 + 2u_3 - u_4) &= 2 - x_3 = 0.8 \\ \frac{1}{0.4^2} (-u_3 + 2u_4 - 0) &= 2 - x_4 = 0.4 \end{aligned}$$

With a matrix notation this leads to

$$\frac{1}{0.4^2} \begin{bmatrix} +2 & -1 & 0 & 0 \\ -1 & +2 & -1 & 0 \\ 0 & -1 & +2 & -1 \\ 0 & 0 & -1 & +2 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 1.6 \\ 1.2 \\ 0.8 \\ 0.4 \end{pmatrix}.$$

This system can now be solved using an algorithm from chapter 2. Since  $h = 0.4$  is not small we only obtain a very crude approximation of the exact solution. To obtain better approximations choose  $h$  small, leading to larger systems of equations. Since the approximation is consistent and stable, we have convergence, i.e. the deviation from the exact solution can be made as small as we wish, by choosing  $h$  small enough<sup>4</sup>. ◇

<sup>4</sup>Ignoring possible arithmetic error contributions, i.e. Figure 4.3 on page 256.

**4–5 Example : A nonlinear boundary value problem**

For the nonlinear problem

Ex 4.13

$$-u''(x) = x + \cos(u(x)) \quad \text{on} \quad 0 < x < 2 \quad \text{with} \quad u(0) = u(2) = 0$$

one can apply the above procedures again to obtain a **nonlinear** system of equations.

$$\frac{1}{0.4^2} \begin{bmatrix} +2 & -1 & 0 & 0 \\ -1 & +2 & -1 & 0 \\ 0 & -1 & +2 & -1 \\ 0 & 0 & -1 & +2 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 0.4 + \cos(u_1) \\ 0.8 + \cos(u_2) \\ 1.2 + \cos(u_3) \\ 1.6 + \cos(u_4) \end{pmatrix}$$

To solve this nonlinear system use methods from chapter 3.1, i.e. partial substitution or Newton's method. Using obvious notations denote the above system by

$$\mathbf{A} \vec{u} = \vec{x} + \cos(\vec{u}).$$

- To use the method of partial substitution choose a starting vector  $\vec{u}_0$ , e.g.  $\vec{u}_0 = (0, 0, 0, 0)^T$ . Then use the iteration scheme

$$\begin{aligned} \mathbf{A} \vec{u}_{k+1} &= \vec{x} + \cos(\vec{u}_k) \\ \vec{u}_{k+1} &= \mathbf{A}^{-1} (\vec{x} + \cos(\vec{u}_k)) \end{aligned}$$

and start to iterate and hope for convergence.

- To use Newton's method build on the linearization  $\cos(\vec{u} + \vec{\phi}) \approx \cos(\vec{u}) - \sin(\vec{u}) \cdot \vec{\phi}$ . Then examine

$$\begin{aligned} \mathbf{A}(\vec{u}_k + \vec{\phi}) &= \vec{x} + \cos(\vec{u}_k) - \sin(\vec{u}_k) \cdot \vec{\phi} \\ (\mathbf{A} + \text{diag}(\sin(\vec{u}_k))) \cdot \vec{\phi} &= -\mathbf{A} \vec{u}_k + \vec{x} + \cos(\vec{u}_k). \end{aligned}$$

The last expression is a system of equations for the vector  $\vec{\phi}$ . The matrix  $\mathbf{A} + \text{diag}(\sin(\vec{u}_k))$  on the left hand side has to be modified by adding the values of  $\sin(\vec{u}_k)$  along the diagonal. Thus for each iteration a new system of linear equations has to be solved. With the solution  $\vec{\phi}$  update  $\vec{u}_{k+1} = \vec{u}_k + \vec{\phi}$  and restart the iteration.

◇

**4–6 Example : Stretching of a beam, with fixed endpoints**

In Section 1.3 we found that the boundary value problem in equation (1.13) (see page 16)

$$-\frac{d}{dx} \left( EA \frac{du(x)}{dx} \right) = f(x) \quad \text{for} \quad 0 < x < L \quad \text{with} \quad u(0) = 0 \quad \text{and} \quad u(L) = u_M$$

corresponds to the stretching of a beam. We consider, at first, constant cross sections only and we will work with the constant  $EA$ . The interval  $[0, L]$  is divided into  $N + 1$  subintervals of equal length  $h = \frac{L}{N+1}$ . Using the notations

$$x_i = i \cdot h \quad , \quad u_i = u(x_i) \quad \text{and} \quad f_i = f(x_i) \quad \text{for} \quad 0 \leq i \leq N + 1$$

and the finite difference formula for  $u''$  in Table 4.1 we replace the differential equation at all interior points by the difference equation

$$-\frac{EA}{h^2} (u_{i-1} - 2u_i + u_{i+1}) = f_i \quad \text{for} \quad 1 \leq i \leq N$$

for the unknowns  $u_i$ . The boundary conditions lead to  $u_0 = 0$  and  $u_{N+1} = u_M$ . Using a matrix notation we find a linear system of equations.

$$\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & \ddots & \ddots & \ddots \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_{N-1} \\ u_N \end{pmatrix} = \frac{h^2}{EA} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_{N-1} \\ f_N \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ u_M \end{pmatrix}$$

This system can be written in the form

$$\mathbf{A} \cdot \vec{u} = \vec{g}$$

with appropriate definition for the matrix  $\mathbf{A}$  and the vectors  $\vec{u}$  and  $\vec{g}$ .

Observe that the  $N \times N$  matrix  $\mathbf{A}$  is symmetric, positive definite and tridiagonal. Thus this system of equations can be solved very quickly, even for large values of  $N$ . First choose a specific example

$$EA = 1 \quad , \quad L = 3 \quad , \quad u_M = 0.2 \quad , \quad f(x) = \sin(x/2) \quad \text{with} \quad N = 20$$

and set the corresponding variables in *Octave*. Then set up the matrix  $\mathbf{A}$ , solve the system and plot the solution, leading to Figure 4.11(a).

[Run demo](#)

#### BeamStretch.m

```
EA = 1.0; L = 3; uM = 0.2; N = 20;
fRHS = @(x) sin(0.5*x); % define the function for the RHS

h = L/(N+1); % stepsize
x = (h:h:L-h)'; f = fRHS(x);
g = h^2/EA*f; g(N) = g(N)+uM;

%% build the tridiagonal, symmetric matrix
di = 2*ones(N,1); % diagonal
up = -ones(N-1,1); % upper and lower diagonal

u = trisolve(di,up,g); % use the special solver
figure(1); plot([0;x;L], [0;u;uM]) % plot the displacement
xlabel('distance'); ylabel('displacement'); grid on
```

The force on the beam at position  $x$  is given by

$$F(x) = EA \frac{du(x)}{dx}.$$

This can be approximated by a centered difference formula

$$F(x_i + \frac{h}{2}) \approx EA \frac{u_{i+1} - u_i}{h}.$$

Thus plot the force  $F$ , as seen in Figure 4.11(b). This graph shows that the left part of the beam is stretched ( $u' > 0$ ), while the right part is compressed ( $u' < 0$ ).

```
du = diff([0;u;uM])/h;
plot([0;x]+h/2,EA*du); grid on
```



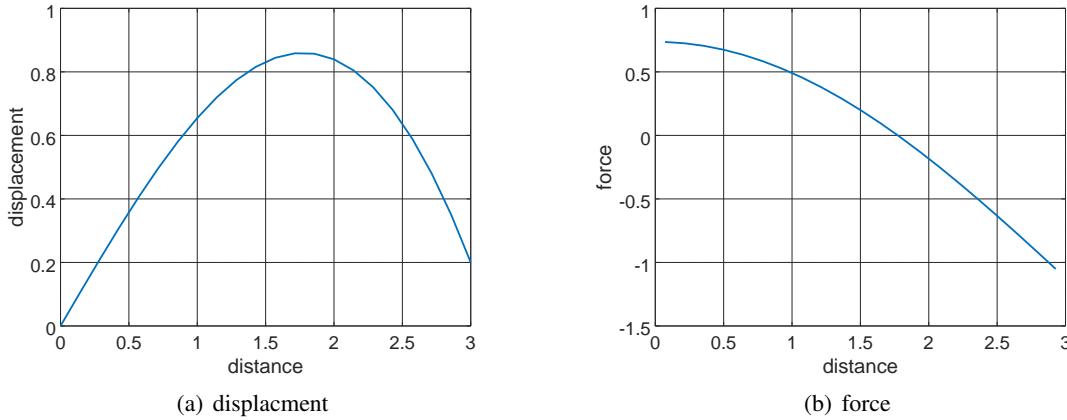


Figure 4.11: Stretching of a beam, displacement and force

- The above example also contains the solution to the steady state heat equation (1.2) on page 10.
  - The above example also contains the solution to the steady state of the vertical deformation of a horizontal string, equation (1.7) on page 13.

## 4-7 Example : Stretching of a beam by a given force

According to Section 1.3 a known force  $F$  at the right endpoint is described by the boundary condition

### Ex 4.14

$$EA \frac{d u(L)}{dx} = F.$$

This new boundary condition replaces the old condition  $u(L) = u_M$ . To handle this case introduce two new unknowns  $u_{N+1} = u(L)$  and  $u_{N+2} = u(L + h)$ . Using a centered difference approximation find<sup>5</sup>

$$\begin{aligned}\frac{d u(L)}{dx} &= \frac{u_{N+2} - u_N}{2 h} + O(h^2) \\ \frac{u_{N+2} - u_N}{2 h} &= \frac{F}{EA} \\ u_{N+2} &= u_N + 2 h \frac{F}{EA}\end{aligned}$$

and using the differential equation at the boundary point  $x = L$  we find

$$\begin{aligned} \frac{-u_N + 2u_{N+1} - u_{N+2}}{h^2} &= \frac{1}{EA} f_{N+1} \\ -u_N + 2u_{N+1} - (u_N + 2h \frac{F}{EA}) &= \frac{h^2}{EA} f_{N+1} \\ -u_N + u_{N+1} &= \frac{h^2}{EA} \frac{f_{N+1}}{2} + h \frac{F}{EA}. \end{aligned}$$

<sup>5</sup>A simpler approach uses

$$u'(L) \approx \frac{u_{N+1} - u_N}{h} = \frac{F}{EA}$$

and thus  $-u_N + u_{N+1} = \frac{h F}{EA}$ . The approximation error is of order  $h$ . For the above approach find an error of  $h^2$ . The additional accuracy is well worth the small additional coding. The only change in the final result is a missing  $\frac{f_{N+1}}{2}$  in the last component of the right hand side vector.

This additional equation can be added to the previous system of equations, leading to a system of  $N + 1$  linear equations.

$$\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & \ddots & \ddots & \ddots \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 & -1 \\ & & & & & & -1 & +1 \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_{N-1} \\ u_N \\ u_{N+1} \end{pmatrix} = \frac{h^2}{EA} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_{N-1} \\ f_N \\ \frac{f_{N+1}}{2} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \frac{hF}{EA} \end{pmatrix}$$

This matrix is again symmetric, positive definite and tridiagonal. For the simple case  $f(x) = 0$  the exact solution  $u(x) = \frac{F}{EA}x$  is known. This is confirmed by the Octave/MATLAB computations below and the resulting straight line in Figure 4.12.

```
EA = 1.0; L = 3; F = 0.2; N = 20;

h = L/(N+1); % stepsize

x = (h:h:L)';
f = zeros(size(x)); % f(x) = 0
g = h^2/EA*f; g(N+1) = g(N+1)/2+h*F/EA;

%% build the tridiagonal, symmetric matrix
di = 2*ones(N+1,1); di(N+1) = 1; % diagonal
up = -ones(N,1); % upper and lower diagonal

u = trisolve(di, up, g);
plot([0;x], [0;u])
```

◇

#### 4-8 Example : Stretching of a beam by a given force and variable cross section

If the cross section  $A$  in the previous example 4-7 is not constant, the above algorithm has to be modified. The differential equation

$$-\frac{d}{dx} \left( EA(x) \frac{du(x)}{dx} \right) = f(x) \quad \text{for } 0 < x < L$$

now uses a variable coefficient  $a(x) = EA(x)$ . The boundary conditions remain  $u(0) = 0$  and  $EA(L) \frac{du(L)}{dx} = F$ . To determine the derivative of  $g(x) = a(x)u'(x)$  use the centered difference formula  $g'(x) = \frac{1}{h}(g(x + \frac{h}{2}) - g(x - \frac{h}{2})) + O(h^2)$  and the approximations

$$\begin{aligned} u'(x - h/2) &= \frac{u(x) - u(x - h)}{h} + O(h^2) \\ u'(x + h/2) &= \frac{u(x + h) - u(x)}{h} + O(h^2) \\ \frac{d}{dx} (a(x) \cdot u'(x)) &= \frac{1}{h} (a(x + h/2) \cdot u'(x + h/2) - a(x - h/2) \cdot u'(x - h/2)) + O(h^2) \\ &\approx \frac{1}{h} \left( a(x + h/2) \frac{u(x + h) - u(x)}{h} - a(x - h/2) \frac{u(x) - u(x - h)}{h} \right) \\ &= \frac{1}{h^2} \left( a(x - \frac{h}{2}) u(x - h) - (a(x - \frac{h}{2}) + a(x + \frac{h}{2})) u(x) + a(x + \frac{h}{2}) u(x + h) \right). \end{aligned}$$

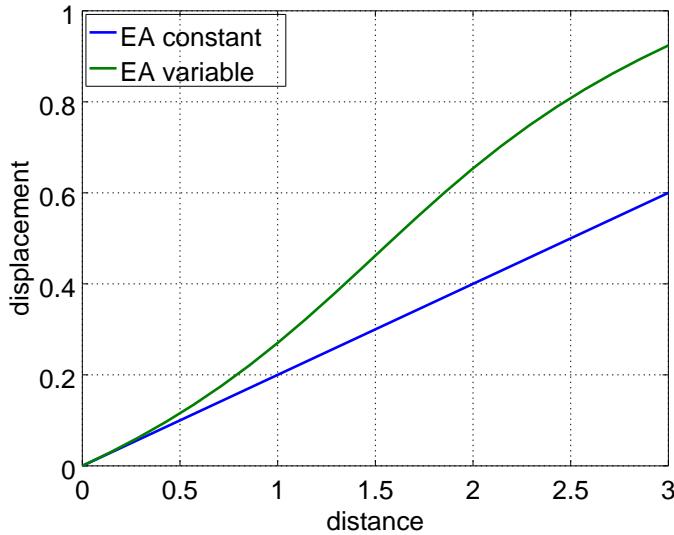


Figure 4.12: Stretching of a beam with constant and variable cross section

One can verify that the error of this finite difference approximation is of the order  $h^2$ . Observe that the values of the coefficient function  $a(x) = EA(x)$  are used at the midpoints of the intervals of length  $h$ . For  $0 \leq i \leq N$  set  $a_i = a(i h + \frac{h}{2})$  to find the difference scheme

$$-a_{i-1} u_{i-1} + (a_{i-1} + a_i) u_i - a_i u_{i+1} = h^2 f_i \quad \text{for } 1 \leq i \leq N$$

To take the boundary condition  $EA(L) u'(L) = a(L) u'(L) = F$  into account proceed just as in Example 4–7.

$$\begin{aligned} F = a(L) \frac{du(L)}{dx} &\approx \frac{1}{2} (a(L - h/2) u'(L - h/2) + a(L + h/2) u'(L + h/2)) + O(h^2) \\ &\approx \frac{a_N (-u_N + u_{N+1})}{2h} + \frac{a_{N+1} (-u_{N+1} + u_{N+2})}{2h} + O(h^2) \\ a_{N+1} u_{N+2} &= +a_N u_N - (a_N - a_{N+1}) u_{N+1} + 2h F + O(h^3) \end{aligned}$$

Using this information for the finite difference approximation of the differential equation at  $x = L$  this leads to

$$\begin{aligned} -a_N u_N + (a_N + a_{N+1}) u_{N+1} - a_{N+1} u_{N+2} &= h^2 f_{N+1} + O(h^4) \\ -2a_N u_N + (2a_N + a_{N+1}) u_{N+1} &= h^2 f_{N+1} + 2h F + O(h^4) + O(h^3) \\ -a_N u_N + a_N u_{N+1} &= h^2 \frac{f_{N+1}}{2} + h F + O(h^4) + O(h^3). \end{aligned}$$

Thus use the approximation

$$\frac{-a_N u_N + a_N u_{N+1}}{h^2} = \frac{f_{N+1}}{2} + \frac{1}{h} F. \quad (4.5)$$

which is consistent of order  $h$ . The elementary approach

$$a(L) u'(L) \approx a(L - h/2) \frac{u(L) - u(L - h)}{h} = a_N \frac{u_{N+1} - u_N}{h} = F$$

would generate a similar equation, without the contribution  $\frac{f_{N+1}}{2}$ , which is consistent of order  $h$  too. A more detailed analysis shows that the approach in (4.5) is consistent of order  $h^2$  for constant functions  $a(x)$ . Thus

equation (4.5) should be used. With this additional equation arrive at a system of  $N + 1$  linear equations.

$$\begin{bmatrix} a_0 + a_1 & -a_1 \\ -a_1 & a_1 + a_2 & -a_2 \\ & -a_2 & a_2 + a_3 & -a_3 \\ & & -a_3 & a_3 + a_4 & -a_4 \\ & & & \ddots & \ddots & \ddots \\ & & & & -a_{N-1} & a_{N-1} + a_N & -a_N \\ & & & & & -a_N & a_N \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_N \\ u_{N+1} \end{pmatrix} = h^2 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_N \\ \frac{f_{N+1}}{2} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ h F \end{pmatrix}$$

This is again a linear system of the form

$$\mathbf{A} \cdot \vec{u} = \vec{g},$$

where the matrix  $\mathbf{A}$  is symmetric, positive definite and tridiagonal.

Reconsider the previous example, but with a thinner cross section  $A(x)$  in the middle section of the beam. Use

$$a(x) = EA(x) = \frac{1}{2} \left( 2 - \sin \frac{\pi x}{L} \right).$$

First define all expressions in Octave and then construct and solve the tridiagonal system of equations. In the code below the sparse, tridiagonal matrix is with the command `spdiags()` and then the linear system solved with the usual backslash operator.

#### BeamStretchVariable.m

```
L = 3; F = 0.2; N = 20;
fRHS = @ (x) zeros (size (x)) % no external forces along the beam
EA = @ (x) (2-sin (x/L*pi))/2;

h = L/(N+1); x = (h:h:L)'; f = fRHS (x);
g = h^2*f; g(N+1) = g(N+1)/2+h*F;

%% build the tridiagonal, symmetric matrix
di = [EA(x-h/2)+EA(x+h/2)]; % diagonal
di(N+1) = EA(L-h/2); % last entry modified
up = -EA([3*h/2:h:L-h/2]'); % upper and lower diagonal
Mat = spdiags ([[up;0],di,[0;up]], [-1 0 1], N+1, N+1); % build the sparse matrix
u = Mat\g; % solve the linear system

plot ([0;x], [0;u], [0;x], [0;u]) % xB, uB from previous computations
legend ('EA constant', 'EA variable', 'location', 'northwest')
xlabel ('distance'); ylabel ('displacement')
```

Demo first  
spdiags,  
then this

The result in Figure 4.12 (page 268) confirms the fact that the thinner beam is weaker, i.e. it will stretch more than the beam with constant, larger cross section. ◇

#### 4.4.2 Boundary Values Problems on a Rectangle

To solve the heat equation on a unit square (equation (1.5) on page 12) one has to solve

$$\begin{aligned}-\Delta u(x, y) &= f(x, y) \quad \text{for } 0 \leq x, y \leq 1 \\ u(x, y) &= 0 \quad \text{for } (x, y) \text{ on boundary}\end{aligned}$$

using the grid size  $h = \frac{1}{n+1}$  and  $u_{i,j} = u(jh, ih)$  and the finite difference stencil in Section 4.2.2 find the equations

$$\frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2} = f_{i,j}.$$

The corresponding grid (for  $n = 7$ ) is shown in Figure 4.13.

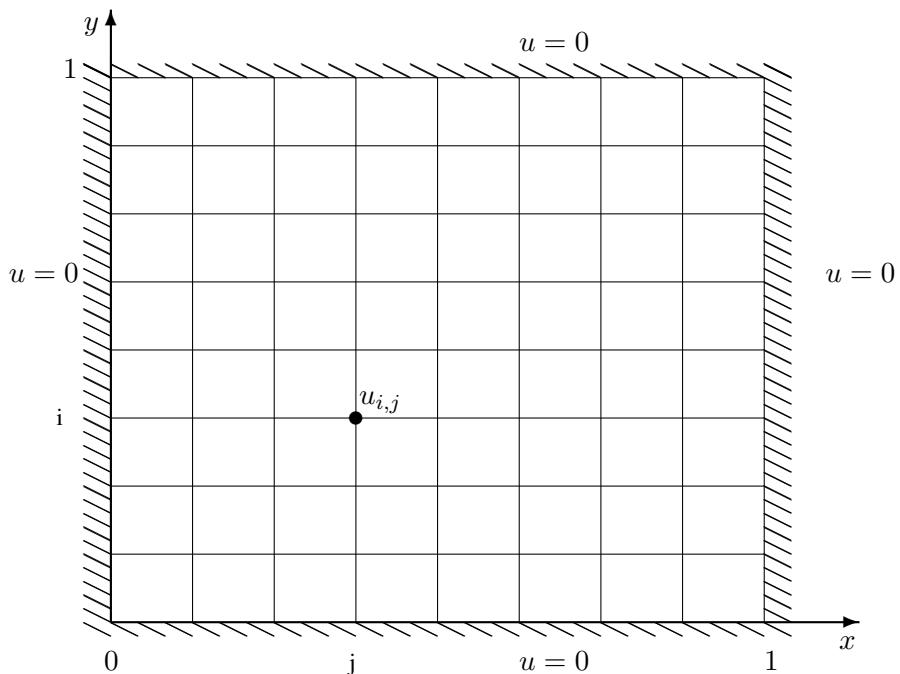


Figure 4.13: A finite difference grid for a steady state heat equation

The unknown values of  $u_{i,j}$  have to be numbered, and there are different options. Here is one of them:

- First number the nodes in the lowest row with numbers 1 through  $n$ .
- Then number the nodes in the second row with numbers  $n + 1$  through  $2n$ .
- Proceed through all the rows. The top right corner will obtain the number  $n^2$ .

The above finite difference approximation of the PDE then reads as

$$\frac{4u_{(i-1)n+j} - u_{(i-2)n+j} - u_{in+j} - u_{(i-1)n+j-1} - u_{(i-1)n+j+1}}{h^2} = f_{(i-1)n+j}.$$

This approximation is consistent of order 2. Arguments very similar to Result 4-3 show that the scheme is stable and thus it is convergent of order 2. Using the model matrix  $\mathbf{A}_{nn}$  from Section 2.3.2 (page 34) this leads to a system of linear equations.

$$\mathbf{A}_{nn} \vec{u} = \vec{f}$$

with a banded, symmetric, positive definite matrix  $\mathbf{A}_{nn}$ . The above is implemented in Octave to solve the system of linear equations and generate the graphics. As an example solve the problem with the right hand side  $f(x, y) = x^2$ .

run demo

**Plate.m**

```
%%%%% script file to solve the heat equation on a unit square
n = 7;
f = @(x,y) x.^2; % describe the heating contribution
%%%%%%%%%%%%%% no modifications necessary beyond this line
h = 1/(n+1);
Dxx = spdiags(ones(n,1)*[-1 2 -1], [-1 0 1], n, n)/h^2;
A = kron(Dxx, eye(n)) + kron(eye(n), Dxx);

x = h:h:1-h; y = x;
[xx,yy] = meshgrid(x,y); % generate the mesh for the graphics
fvec = f(xx(:),yy(:)); % compute the function
tic(); % start the stop watch
u = A\fvec; % solve the system of equations
toc(); % display the solution time
mesh(xx,yy,reshape(fvec,n,n)) % generate the graphics
xlabel('x'); ylabel('y');
```

The result of the above code is not completely satisfying, since the zero values of the function on the boundary are not displayed. The code below adds these values and will generate Figure 4.14. The graph clearly displays the higher temperature in the section with large values of  $x$ . This is caused by the heating term  $f(x, y) = x^2$ .

```
%%% add on the zero boundary values for a nicer graphics
x = 0:h:1; y = x;
[xx,yy] = meshgrid(x,y); uu = zeros(size(xx));
uu(2:n+1,2:n+1) = reshape(u,n,n);
mesh(xx,yy,uu)
xlabel('x'); ylabel('y');
```

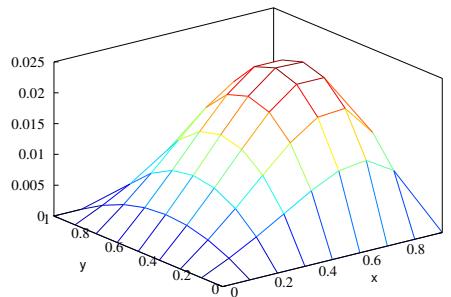


Figure 4.14: Solution of the steady state heat equation on a square

The model matrix  $\mathbf{A}_{nn}$  on page 34 is symmetric of size  $n^2 \times n^2$  and it has a band structure with semi-bandwidth  $n$ . Thus it is a very sparse matrix. In the above code the system of linear equation is solved with the command  $u=A\fvec$ , and this will take advantage of the symmetry and sparseness. It is not a good idea to use  $u = \text{inv}(A) * \fvec$  since this creates the full matrix  $\mathbf{A}^{-1}$ . A better idea is to use Octave/MATLAB commands to create  $\mathbf{A}$  as a sparse matrix, then the built-in algorithm will take advantage of this. Thus we can solve problems with much finer grids, see the modified code below. In addition we may allow a different number of grid points in the two directions.

- Use  $nx$  interior points in  $x$  direction and  $ny$  points in  $y$  direction for a matrix of size  $(nx \cdot ny) \times (nx \cdot ny)$ . Numbering in the  $x$  direction first will lead to a semi-bandwidth of  $nx$ , but numbering in the  $y$  direction first will lead to a semi-bandwidth of  $ny$ .
- To construct the matrices representing the derivatives in  $x$  and  $y$  direction independently use the command `spdiags()`. Then use the Kronecker product (command `kron()`) to construct the sparse matrix  $\mathbf{A}$ .
- The backslash operator `\` in MATLAB or Octave will take full advantage of the sparsity structure of this matrix, using algorithms presented in Chapter 2, in particular Section 2.6.7.

[run demo](#)**Heat2DStatic.m**

```

nx = 55; ny = 50;
f = @(x,y)x.^2;
%%%%%%%%%%%%%
hx = 1/(nx+1); hy = 1/(ny+1);
Dxx = spdiags(ones(nx,1)*[-1 2 -1], [-1 0 1], nx, nx) / (hx^2);
Dyy = spdiags(ones(ny,1)*[-1 2 -1], [-1 0 1], ny, ny) / (hy^2);
A = kron(Dxx, speye(ny)) + kron(speye(nx), Dyy);
x = hx:hx:1-hx; y = hy:hy:1-hy;
[xx,yy] = meshgrid(x,y);
fvec = f([xx(:),yy(:)]);
tic()
u = A\fvec;
solutionTime = toc()

%%% add on the zero boundary values for a nicer graphics
x = 0:hx:1; y = 0:hy:1;
[xx,yy] = meshgrid(x,y); uu = zeros(size(xx));
uu(2:ny+1, 2:nx+1) = reshape(u, ny, nx);
mesh(xx, yy, uu)
xlabel('x'); ylabel('y');

```

[Ex 4.16](#)

## 4.5 Initial Boundary Value Problems

### 4.5.1 The Dynamic Heat Equation

A one dimensional heat equation is given by the partial differential equation

$$\begin{aligned}
\frac{\partial}{\partial t} u(t, x) &= \kappa \frac{\partial^2}{\partial x^2} u(t, x) && \text{for } 0 < x < 1 \quad \text{and} \quad t > 0 \\
u(t, 0) &= u(t, 1) &= 0 & \text{for } t > 0 \\
u(0, x) &= u_0(x) & & \text{for } 0 < x < 1
\end{aligned} \tag{4.6}$$

The maximum principle<sup>6</sup> implies that for all  $t \geq 0$  we find

$$\max\{|u(x, t)| : 0 \leq x \leq 1\} \leq \max\{|u_0(x)| : 0 \leq x \leq 1\}.$$

The approximate solutions generated by a finite difference scheme should satisfy this property too, leading to the stability condition.

The two dimensional domain  $(t, x) \in \mathbb{R}_+ \times [0, 1]$  is discretized as illustrated in Figure 4.15. For step sizes  $h = \Delta x = \frac{1}{n+1}$  and  $\Delta t$  let

$$u_{i,j} = u(i \cdot \Delta t, j \cdot \Delta x) \quad \text{for } j = 0, 1, 2, \dots, n, n+1 \quad \text{and} \quad i \geq 0.$$

<sup>6</sup>Find the precise statement and proofs in any good book on partial differential equations.

The boundary condition  $u(t, 0) = u(t, 1) = 0$  implies  $u_{i,0} = u_{i,n+1} = 0$  and the initial condition  $u(0, x) = u_0(x)$  leads to  $u_{0,j} = u_0(j \cdot \Delta x)$ . The PDE (4.6) is replaced by a finite difference approximation on the grid shown in Figure 4.15 and the result is examined.

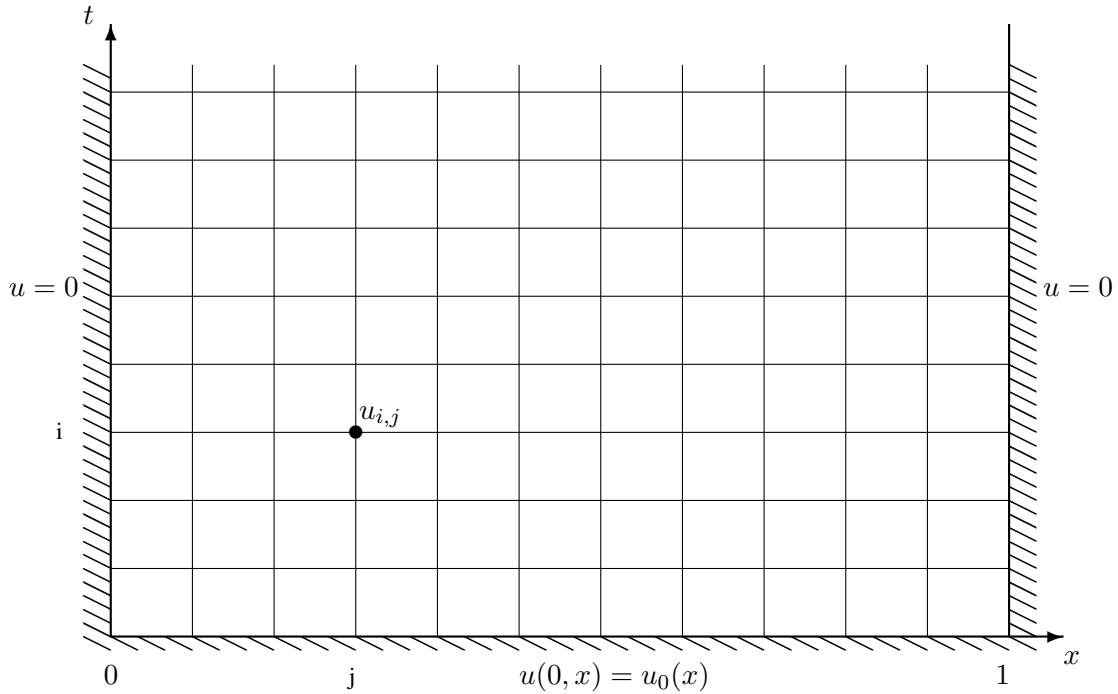


Figure 4.15: A finite difference grid for a dynamic heat equation

The solution of the finite difference equation will be computed with the help of time steps, i.e. use the values at one time level  $t = i \cdot \Delta t$  and then compute the values at the next level  $t + \Delta t = (i + 1) \Delta t$ . Thus put all values at one time level  $t = i \Delta t$  into a vector  $\vec{u}_i$  by

$$\vec{u}_i = (u_{i,1}, u_{i,2}, u_{i,3}, \dots, u_{i,n-1}, u_{i,n})^T.$$

A finite difference approximation to the second order space derivative is given by (see Table 4.1 on page 255)

$$\kappa \frac{\partial^2}{\partial x^2} u(t, x) = \kappa \frac{u(t, x - \Delta x) - 2u(t, x) + u(t, x + \Delta x)}{\Delta x^2} + O((\Delta x)^2). \quad (4.7)$$

Thus the values of the second order space derivatives at one time level can be approximated by  $-\kappa \mathbf{A}_n \cdot \vec{u}_i$  where the symmetric  $n \times n$  matrix  $\mathbf{A}_n$  is given by

$$\mathbf{A}_n = \frac{1}{\Delta x^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}.$$

Now find the approximation of the PDE (Partial Differential Equation) by a linear system of ordinary differential equations

$$\frac{d}{dt} \vec{u}(t) = -\kappa \mathbf{A}_n \vec{u}(t) \quad \text{with} \quad \vec{u}(0) = \vec{u}_0.$$

### 4.5.2 Construction of the Solution Using Eigenvalues and Eigenvectors

To examine the different possible approximations of the dynamic heat equation (4.6) use eigenvalues and eigenvectors of the matrix  $\mathbf{A}_n$  given by

$$\lambda_k = \frac{1}{\Delta x^2} (2 + 2 \cos \frac{k \pi}{n+1}) = \frac{4}{\Delta x^2} \sin^2 \frac{k \pi}{2(n+1)}$$

and

$$\vec{v}_k = \left( \sin \frac{1 k \pi}{n+1}, \sin \frac{2 k \pi}{n+1}, \sin \frac{3 k \pi}{n+1}, \dots, \sin \frac{(n-1) k \pi}{n+1}, \sin \frac{n k \pi}{n+1} \right)^T$$

where  $k = 1, 2, 3, \dots, n$ . Thus the eigenvectors are discretizations of the functions  $\sin(k \pi x)$  on the interval  $[0, 1]$ . These functions have exactly  $k$  local extrema in the interval. The higher the value of  $k$  the more the eigenfunction will oscillate. In these notes find more information on matrices of the above type in Section 2.3 (page 32) and Result 3–22 (page 134). For another proof of the above see [Smit84, p. 154].

Since the matrix  $\mathbf{A}_n$  is symmetric the eigenvectors are orthogonal and form a basis. Since the eigenvectors satisfy

$$\langle \vec{v}_k, \vec{v}_j \rangle = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases}$$

(i.e. orthonormalized), then any vector  $\vec{u}$  can be written as linear combination of normalized eigenvectors  $\vec{v}_k$  of the matrix  $\mathbf{A}_n$ , i.e.

$$\vec{u} = \sum_{k=1}^n \alpha_k \vec{v}_k \quad \text{with} \quad \alpha_k = \langle \vec{u}, \vec{v}_k \rangle.$$

For arbitrary  $t \geq 0$  consider the vector  $\vec{u}(t)$  of the discretized (in space) solution. The differential equation (4.7) reads as

$$\frac{d}{dt} \vec{u}(t) = -\kappa \mathbf{A}_n \cdot \vec{u}(t).$$

The solution  $\vec{u}(t)$  of this system of ODEs (Ordinary Differential Equation) can be written as linear combination of eigenvectors, i.e.

$$\begin{aligned} \vec{u}(t) &= \sum_{k=1}^n \alpha_k(t) \vec{v}_k \\ \frac{d}{dt} \vec{u}(t) &= \sum_{k=1}^n \dot{\alpha}_k(t) \vec{v}_k \\ -\kappa \mathbf{A}_n \vec{u}(t) &= -\kappa \sum_{k=1}^n \alpha_k(t) \mathbf{A}_n \vec{v}_k = -\kappa \sum_{k=1}^n \alpha_k(t) \lambda_k \vec{v}_k \\ \sum_{k=1}^n \dot{\alpha}_k(t) \vec{v}_k &= -\sum_{k=1}^n (\kappa \alpha_k(t) \lambda_k) \vec{v}_k. \end{aligned}$$

Examine the scalar product of the above with a vector  $\vec{v}_j$  and use the orthogonality to conclude

$$\begin{aligned} \langle \vec{v}_j, \sum_{k=1}^n \dot{\alpha}_k(t) \vec{v}_k \rangle &= \langle \vec{v}_j, -\sum_{k=1}^n (\kappa \alpha_k(t) \lambda_k) \vec{v}_k \rangle \\ \sum_{k=1}^n \dot{\alpha}_k(t) \langle \vec{v}_j, \vec{v}_k \rangle &= -\sum_{k=1}^n (\kappa \alpha_k(t) \lambda_k) \langle \vec{v}_j, \vec{v}_k \rangle \\ \dot{\alpha}_j(t) &= -\kappa \lambda_j \alpha_j(t) \quad \text{for } j = 1, 2, 3, \dots, n. \end{aligned}$$

The above system of  $n$  linear equations is converted to  $n$  linear, first order differential equations. The initial values for the coefficient functions are given by  $\alpha_j(0) = \langle \vec{u}_0, \vec{v}_j \rangle$ . For these equations use the methods and results in Section 4.3.1. The approximation scheme to the system of differential equations  $\frac{d}{dt} \vec{x}(t) = -\kappa \mathbf{A} \vec{x}(t)$  is stable if and only if the scheme applied to all of the above ordinary differential equations is stable. Three different approaches will be examined: explicit, implicit and Crank–Nicolson.

Since the above ordinary differential equation can be solved analytically find a formula for the solution

$$\vec{u}(t) = \sum_{k=1}^n \alpha_k(t) \vec{v}_k = \sum_{k=1}^n \langle \vec{u}_0, \vec{v}_k \rangle \exp(-\kappa \lambda_k t) \vec{v}_k.$$

For numerical purposes this formula is not very useful, since the effort to determine the eigenvalues and eigenvectors is too large.

### 4.5.3 Explicit Finite Difference Approximation to the Heat Equation

The time derivative in the PDE (4.6) can be approximated by a forward difference

$$\frac{\partial}{\partial t} u(t, x) = \frac{u(t + \Delta t, x) - u(t, x)}{\Delta t} + O(\Delta t).$$

This can be combined with the space derivatives in equation (4.7) to obtain the scheme illustrated in Figure 4.16. The corresponding stencil is shown in Figure 4.5 on page 257. The results in Table 4.1 imply that the scheme is consistent with an error of the order  $O(\Delta t) + O((\Delta x)^2)$ .

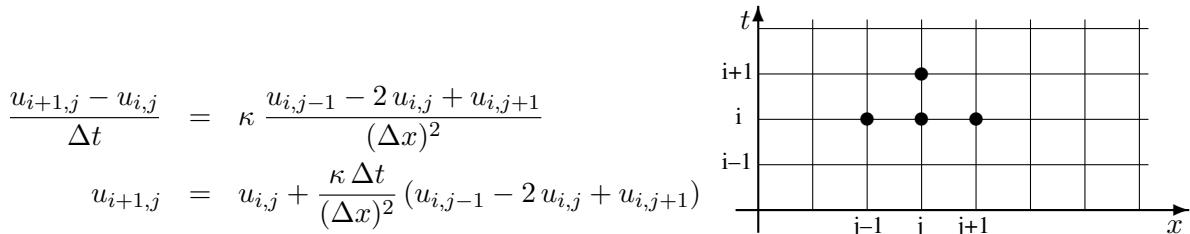


Figure 4.16: Explicit finite difference approximation

Using a matrix notation the finite difference equation can be written as

$$\frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} = -\kappa \mathbf{A}_n \vec{u}_i$$

or solving for the next timelevel  $\vec{u}_{i+1}$

$$\vec{u}_{i+1} = \vec{u}_i - \kappa \Delta t \mathbf{A}_n \cdot \vec{u}_i = (\mathbb{I}_n - \kappa \Delta t \mathbf{A}_n) \cdot \vec{u}_i.$$

If the vector  $\vec{u}_i$  is known the values at the next time level  $\vec{u}_{i+1}$  can be computed without solving a system of linear equations, thus this is called an **explicit** method. Starting with the discretization of the initial values  $\vec{u}_0$  and applying the above formula repeatedly we find the solution

$$\vec{u}_i = (\mathbb{I}_n - \kappa \Delta t \mathbf{A}_n)^i \cdot \vec{u}_0.$$

The goal is to examine the stability of this finite difference scheme. Since for eigenvalues  $\lambda_k$  and eigenvectors  $\vec{v}_k$  we have

$$(\mathbb{I}_n - \kappa \Delta t \mathbf{A}_n)^i \cdot \vec{v}_k = (1 - \kappa \Delta t \lambda_k)^i \cdot \vec{v}_k$$

and the solution will remain bounded as  $i \rightarrow \infty$  only if  $\kappa \Delta t \lambda_k < 2$  for all  $k = 1, 2, 3, \dots, n$ . This corresponds to the stability condition.

Since we want to use the results of Section 4.3.2 on solutions of the ordinary differential equation we translate to the coefficient functions  $\alpha_k(t)$  and find

$$\begin{aligned}\frac{d}{dt} \alpha_k(t) &= -\kappa \lambda_k \alpha_k(t) \\ \frac{\alpha_k(t + \Delta t) - \alpha_k(t)}{\Delta t} &= -\kappa \lambda_k \alpha_k(t) \quad \text{finite difference approximation} \\ \alpha_k(t + \Delta t) &= (1 - \Delta t \kappa \lambda_k) \alpha_k(t) \\ \alpha_k(i \cdot \Delta t) &= (1 - \Delta t \kappa \lambda_k)^i \alpha_k(0).\end{aligned}$$

The scheme is stable if the absolute value of the bracketed expression is smaller than 1, i.e.

$$\kappa \lambda_k \Delta t < 2.$$

Since the largest eigenvalue of  $\mathbf{A}_n$  is (see Section 2.3.1 starting on page 32)  $\lambda_n = \frac{4}{\Delta x^2} \sin^2 \frac{n\pi}{2(n+1)} \approx \frac{4 \sin^2 \frac{\pi}{2}}{\Delta x^2} = \frac{4}{\Delta x^2}$  find the stability condition

$$\kappa \frac{\Delta t}{\Delta x^2} < \frac{1}{2} \iff \Delta t < \frac{1}{2\kappa} (\Delta x)^2.$$

This is a situation of **conditional stability**. The restriction on the size of the timestep  $\Delta t$  is severe, since for small values of  $\Delta x$  the  $\Delta t$  will need to be much smaller.

In Figure 4.17 a solution of the dynamic heat problem

$$\begin{aligned}\frac{\partial}{\partial t} u(t, x) &= \kappa \frac{\partial^2}{\partial x^2} u(t, x) \quad \text{for } 0 < x < 1 \text{ and } t > 0 \\ u(t, 0) = u(t, 1) &= 0 \quad \text{for } t > 0 \\ u(0, x) &= f(x) \quad \text{for } 0 < x \leq 1\end{aligned}$$

is shown for values of  $r = \kappa \frac{\Delta t}{\Delta x^2}$  slightly smaller or larger than the critical value of 0.5. The initial value used is

$$f(x) = \begin{cases} 2x & \text{for } 0 < x \leq 0.5 \\ 2 - 2x & \text{for } 0.5 \leq x < 1 \end{cases}.$$

Since the largest eigenvalue of  $\mathbf{A}_n$  will be the first to exhibit instability examine the corresponding eigen-

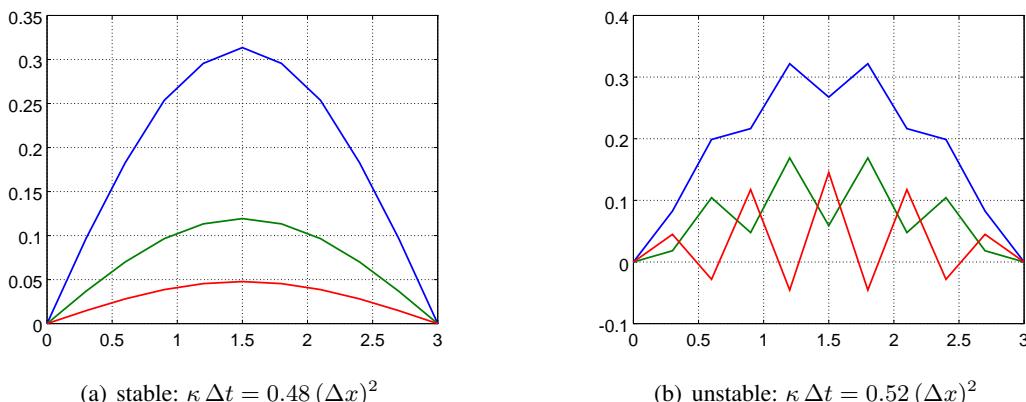


Figure 4.17: Solution of 1-d heat equation, stable and unstable algorithms with  $r \approx 0.5$

vector

$$\vec{v}_n = \left( \sin \frac{1 n \pi}{n+1}, \sin \frac{2 n \pi}{n+1}, \sin \frac{3 n \pi}{n+1}, \dots, \sin \frac{(n-1) n \pi}{n+1}, \sin \frac{n n \pi}{n+1} \right)^T.$$

The corresponding eigenfunction has  $n$  extrema in the interval. Thus the instability should exhibit  $n$  extrema, which is confirmed by Figure 4.17(b) where the calculation is done with  $n = 9$ , as shown in the Octave code below. The deviation from the correct solution exhibits 9 local extrema in the interval. This is an example of a consistent and non-stable finite difference approximation. Obviously the scheme is not convergent.

run demo

---

### HeatDynamic.m

---

```
L = 1; % length of the space interval
n = 9; % number of interior grid points
%n = 29; % number of interior grid points
r = 0.45; % ratio to compute time step
%r = 0.52; % ratio to compute time step
T = 0.1; % final time

iv = @(x) min([2*x/L, 2-2*x/L]')';
dx = L/(n+1); dt = r*dx^2; x = linspace(0,L,n+2)';

y = iv(x);
ynew = y;
legend('off')
for t = 0:dt:T+dt;
    % for k = 2:n+1 % code with loops
    % ynew(k) = (1-2*r)*y(k)+r*(y(k-1)+y(k+1));
    % endfor
    % y = ynew;
    y(2:n+1) = (1-2*r)*y(2:n+1)+r*(y(1:n)+y(3:n+2)); % no loops
    plot(x,y)
    axis([0,1,0,1]); grid on
    text(0.1,0.9,['t = ',num2str(t,3)]);
    pause(0.02);
end%for
```

---

In the above code we verify that for each time step approximately  $2 \cdot n$  multiplications/additions are necessary. Thus the computational cost of one time step is  $2n$ .

If the differential equation to be solved contains an inhomogeneous term, i.e.

$$\frac{\partial}{\partial t} u(t, x) = \kappa \frac{\partial^2}{\partial x^2} u''(t, x) + f(t, x)$$

then use the difference approximation

$$\vec{u}_{i+1} - \vec{u}_i = -\Delta t \kappa \mathbf{A}_n \vec{u}_i + \Delta t \vec{f}_i.$$

This system can be solved similarly.

#### 4.5.4 Implicit Finite Difference Approximation to the Heat Equation

The time derivative in the PDE (4.6) can be approximated by a backward difference

$$\frac{\partial}{\partial t} u(t, x) = \frac{u(t, x) - u(t - \Delta t, x)}{\Delta t} + O(\Delta t).$$

This will lead to the finite difference scheme shown in Figure 4.18. The corresponding stencil is shown in Figure 4.6 on page 257. The results in Table 4.1 again imply that the scheme is consistent with the error of the order  $O(\Delta t) + O((\Delta x)^2)$ .

$$\frac{u_{i+1,j} - u_{i,j}}{\Delta t} = \kappa \frac{u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}}{(\Delta x)^2}$$

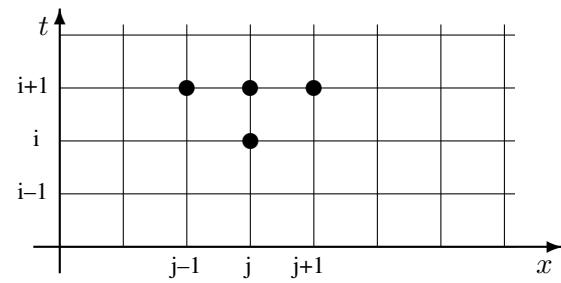


Figure 4.18: Implicit finite difference approximation

Using a matrix notation find

$$\vec{u}_{i+1} - \vec{u}_i = -\kappa \Delta t \mathbf{A}_n \cdot \vec{u}_{i+1}$$

or

$$(\mathbb{I}_n + \kappa \Delta t \mathbf{A}_n) \cdot \vec{u}_{i+1} = \vec{u}_i.$$

If the values  $\vec{u}_i$  at a given time  $t_i = i \Delta t$  are known solve a system of linear equations to determine the values  $\vec{u}_{i+1}$  at the next time level. This is an **implicit** method. As in the previous section use the eigenvalues and vectors of  $\mathbf{A}_n$  to examine stability of the scheme. Using the known initial value  $\vec{u}_0$  we are lead to the iteration scheme

$$\vec{u}_i = (\mathbb{I}_n + \kappa \Delta t \mathbf{A}_n)^{-i} \cdot \vec{u}_0$$

and thus (use  $\mathbf{A}_n \vec{v}_k = \lambda_k \vec{v}_k$ )

$$(\mathbb{I}_n + r \mathbf{A}_n)^{-i} \cdot \vec{v}_k = \left( \frac{1}{1 + \kappa \Delta t \lambda_k} \right)^i \cdot \vec{v}_k.$$

Since  $\lambda_k > 0$  we find that this scheme is **unconditionally stable**, i.e. there are no restrictions on the ratio of the step sizes  $\Delta x$  and  $\Delta t$ . This is confirmed by the results in Figure 4.19. It was generated by code similar to the one below.

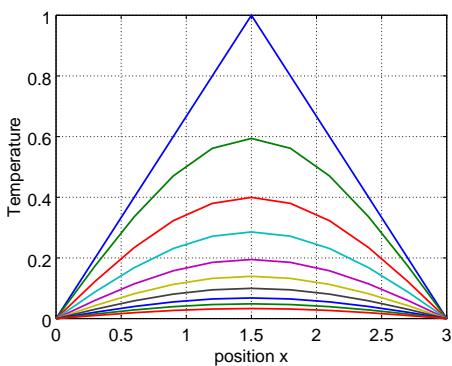
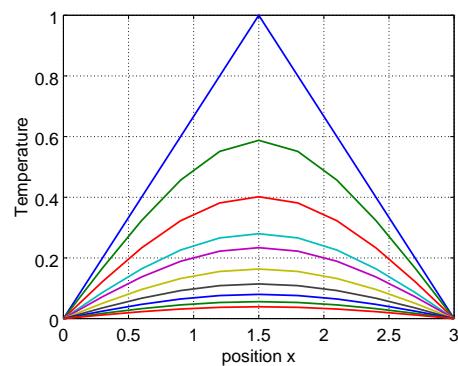
(a) stable:  $\kappa \Delta t = 0.5 (\Delta x)^2$ (b) stable:  $\kappa \Delta t = 2 (\Delta x)^2$ 

Figure 4.19: Solution of 1-d heat equation, implicit scheme with small and large step sizes

[run demo](#)

### HeatDynamicImplicit.m

```
L = 1; % length of the space interval
n = 29; % number of interior grid points
r = 0.2; % ratio to compute time step
%r = 2.0;% ratio to compute time step
```

```

T = 0.5; % final time
plots = 5;% number of plots to be saved

iv = @(x)min([2*x/L,2-2*x/L]')';
dx = L/(n+1); dt = 2*r*dx^2; x = linspace(0,L,n+2)';
initval = iv(x(2:n+1));

yplot = zeros(plots,n+2);
plotc = 1; tplot = linspace(0,T,plots);

Adiag = ones(n,1)*(1+2*r);
Aoffdiag = -ones(n-1,1)*r;

y = initval;
for t = 0:dt:T+dt;
    if min(abs(tplot-t))<dt/2
        yplot(plotc,2:n+1) = y'; plotc = plotc +1;
    end%if
    y = trisolve(Adiag,Aoffdiag,y);
end%for
plot(x,yplot)
grid on
xlabel('position x'); ylabel('Temperature')

```

To perform one time step one has to solve a system of  $n$  linear equations where the matrix is symmetric, tridiagonal and positive definite. There are efficient algorithms (`trisolve()`) for this type of problem (e.g. [GoluVanLoan96], [GoluVanLoan13]), requiring only  $5n$  multiplications. If the matrix decomposition and the back-substitution are separately coded, then this can even be reduced to an operation count for one time-step of only  $3n$  multiplication. Thus the computational effort for one explicit step is similar to the cost for one implicit step, but we gain unconditional stability.

If the differential equation to be solved contains an inhomogeneous term, i.e.

$$\dot{u}(t, x) = \kappa u''(t, x) + f(t, x),$$

then we may use the difference approximation

$$\vec{u}_{i+1} - \vec{u}_i = -\Delta t \kappa \mathbf{A}_n \vec{u}_{i+1} + \Delta t \vec{f}_{i+1}.$$

This system can be solved similarly.

#### 4.5.5 Crank–Nicolson Approximation to the Heat Equation

When using a centered difference approximation

$$\frac{\partial}{\partial t} u(t, x) = \frac{u(t + \Delta t/2, x) - u(t - \Delta t/2, x)}{\Delta t} + O((\Delta t)^2)$$

at the midpoint between time levels the finite difference scheme in Figure 4.20 is generated. It is an approximation of the differential equation  $\frac{d}{dt} u = \kappa u''$  at the midpoint  $(\frac{t_i+t_{i+1}}{2}, x_j)$ . The results in Table 4.1 imply that the scheme is consistent with the error of the order  $O((\Delta t)^2) + O((\Delta x)^2)$ . The order of convergence in time is improved by one..

The matrix notation leads to

$$\vec{u}_{i+1} - \vec{u}_i = -\frac{\kappa \Delta t}{2} (\mathbf{A}_n \cdot \vec{u}_{i+1} + \mathbf{A}_n \cdot \vec{u}_i)$$

$$\begin{aligned}\frac{u_{i+1,j} - u_{i,j}}{\kappa \Delta t} &= \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{2(\Delta x)^2} \\ &+ \frac{u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}}{2(\Delta x)^2}\end{aligned}$$

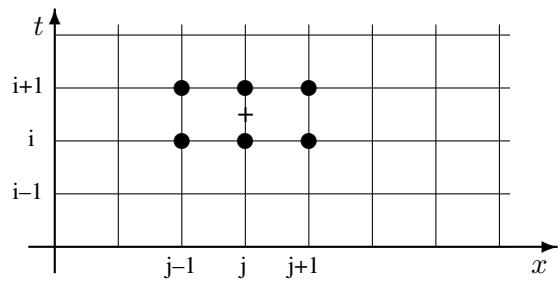


Figure 4.20: Crank–Nicolson finite difference approximation

or

$$\left( \mathbb{I}_n + \frac{\kappa \Delta t}{2} \mathbf{A}_n \right) \cdot \vec{u}_{i+1} = \left( \mathbb{I}_n - \frac{\kappa \Delta t}{2} \mathbf{A}_n \right) \cdot \vec{u}_i.$$

With the values  $\vec{u}_i$  at a given time multiply the vector with a matrix and then solve a system of linear equations to determine the values  $\vec{u}_{i+1}$  at the next time level. This is an **implicit** method. As in the previous section we use the eigenvalues and vectors of  $\mathbf{A}_n$  to examine stability of the scheme. Examine the inequality

$$\left| \frac{2 - \kappa \Delta t \lambda_k}{2 + \kappa \Delta t \lambda_k} \right|^i < 1.$$

Since  $\lambda_k > 0$  this scheme is **unconditionally stable**.

In Table 4.3 find a comparison of the three different finite difference approximations to equation (4.6).

- For the explicit method one multiplication by a matrix  $\mathbb{I} - \alpha \mathbf{A}$  is required. Thus we need approximately  $3n$  multiplications. If one would take advantage of the symmetry it could be reduced to  $2n$  multiplications.
- For the implicit method one system of linear equations with a matrix  $\mathbb{I} - \alpha \mathbf{A}$  is required. Using the standard Cholesky factorization with band structure approximately  $4n$  multiplications are required. Working with the modified Cholesky factorization one could reduce to  $3n$  multiplications.
- For the Crank–Nicolson method one matrix multiplication is paired with one system to be solved. Thus we need approximately  $7n$  multiplication, or only  $5n$  with the optimized algorithms.

Using an inverse matrix is a bad idea in the above context, as this will lead to a full matrix and thus at least  $n^2$  multiplications. Even for relatively large numbers  $n$ , the time required to do one time step will be minimal for all of the above methods. This will be different for the 2D situation, as examined in Table 4.4. As a consequence one should use either an implicit method or Crank–Nicolson for this type of problem.

If the differential equation to be solved contains an inhomogeneous term, i.e.

$$\frac{\partial}{\partial t} u(t, x) = \kappa \frac{\partial^2}{\partial x^2} u''(t, x) + f(t, x)$$

then use the difference approximation<sup>7</sup>

$$\vec{u}_{i+1} - \vec{u}_i = -\frac{\Delta t \kappa}{2} \mathbf{A}_n (\vec{u}_i + \vec{u}_{i+1}) + \frac{\Delta t}{2} (\vec{f}_i + \vec{f}_{i+1}).$$

This leads to

$$(\mathbb{I} + \frac{\Delta t \kappa}{2} \mathbf{A}_n) \vec{u}_{i+1} = (\mathbb{I} - \frac{\Delta t \kappa}{2} \mathbf{A}_n) \vec{u}_i + \frac{\Delta t}{2} (\vec{f}_i + \vec{f}_{i+1}).$$

This system can be solved similarly.

<sup>7</sup>Instead of  $\frac{1}{2} (\vec{f}_i + \vec{f}_{i+1})$  one can (or even should) use  $\vec{f}_{i+1/2}$ , i.e. the discretetization of  $f(t_i + \frac{\Delta t}{2}, x)$ .

method	order of consistency	stability condition	flops	optimal flops
explicit	$O(\Delta t) + O((\Delta x)^2)$	$\Delta t < \frac{1}{2\kappa} (\Delta x)^2$	$3n$	$2n$
implicit	$O(\Delta t) + O((\Delta x)^2)$	unconditional	$4n$	$3n$
Crank–Nicolson	$O((\Delta t)^2) + O((\Delta x)^2)$	unconditional	$7n$	$5n$
advantage	Crank–Nicolson	implicit and CN	none	none

Table 4.3: Comparison of finite difference schemes for the 1D heat equation

#### 4.5.6 General Parabolic Problems

In the previous section we considered only a special case of the space discretization operator  $\mathbf{A} = \kappa \mathbf{A}_n$ . A more general situation may be described by the equation

$$\frac{d}{dt} \vec{u}(t) = -\mathbf{A} \cdot \vec{u}(t) + \vec{f}(t)$$

where the symmetric, positive definite matrix  $\mathbf{A}$  has eigenvalues  $0 \leq \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \dots \leq \lambda_n$ . When using either Crank–Nicolson or the fully implicit method the resulting finite difference scheme will be unconditionally stable. The explicit method leads to

$$\vec{u}(t + \Delta t) = \vec{u}(t) - \Delta t \mathbf{A} \vec{u}(t) + \vec{f}(t) = (\mathbb{I} - \Delta t \mathbf{A}) \vec{u}(t) + \Delta t \vec{f}(t).$$

As in the previous sections we examine  $\vec{u}$  as a linear combination of the eigenvectors  $\vec{v}_k$ . For the largest eigenvalue  $\lambda_n$  the factor has to be smaller than 1 and thus  $|1 - \Delta t \lambda_n| < 1$ . This leads to the stability condition

$$\Delta t \cdot \lambda_n < 2 \quad \iff \quad \Delta t < \frac{2}{\lambda_n}.$$

This condition remains valid, also for problems with more than one space dimension. To use the explicit method for these type of problems one needs to estimate the largest eigenvalue of the space discretization. Estimates of this type can be given, based on the condition number of the discretization matrix, e.g. [KnabAnge00, Satz 3.45]. For higher space dimensions the effort to solve one linear system of equations for the implicit methods will increase drastically, as the resulting matrices will not be tridiagonal, but we find a band structure. Nonetheless this structure can be used in efficient implementations, all will be shown in the next section. The relevant results on matrix computations are given in Chapter 2.

For many dynamic problems a mass matrix  $\mathbf{M}$  has to be taken into account too. Consider a discretized systems of the form

$$\mathbf{M} \frac{d}{dt} \vec{u}(t) = -\mathbf{A} \cdot \vec{u}(t) + \vec{f}(t).$$

Often linear systems of equations with the matrix  $\mathbf{M}$  are easily solved, e.g.  $\mathbf{M}$  might be a diagonal matrix with positive entries. The **generalized eigenvalues**  $\lambda$  and **eigenvectors**  $\vec{v}$  are nonzero solutions of

$$\mathbf{A} \cdot \vec{v} = \lambda \mathbf{M} \vec{v}.$$

- The explicit discretization scheme leads to

$$\begin{aligned} \frac{1}{\Delta t} \mathbf{M} (\vec{u}(t + \Delta t) - \vec{u}(t)) &= -\mathbf{A} \cdot \vec{u}(t) + \vec{f}(t) \\ \mathbf{M} \vec{u}(t + \Delta t) &= \mathbf{M} \vec{u}(t) - \Delta t \mathbf{A} \cdot \vec{u}(t) + \Delta t \vec{f}(t) = (\mathbf{M} - \Delta t \mathbf{A}) \cdot \vec{u}(t) + \Delta t \vec{f}(t). \end{aligned}$$

Using an expansion with eigenvectors of the generalized eigenvalue problem the homogeneous problem ( $\vec{f}(t) = \vec{0}$ ) leads to

$$\begin{aligned} \alpha_k(t + \Delta t) \mathbf{M} \vec{v}_k &= \alpha_k(t) (\mathbf{M} - \Delta t \mathbf{A}) \cdot \vec{v}_k \\ &= \alpha_k(t) (\mathbf{M} - \Delta t \lambda_k \mathbf{M}) \cdot \vec{v}_k \\ \alpha_k(t + \Delta t) &= \alpha_k(t) (1 - \Delta t \lambda_k). \end{aligned}$$

Thus the stability condition is again  $\Delta t < 2/\lambda_n$ , where  $\lambda_n$  is the largest generalized eigenvalue.

- The fully implicit scheme will lead to

$$\begin{aligned}\frac{1}{\Delta t} \mathbf{M} (\vec{u}(t + \Delta t) - \vec{u}(t)) &= -\mathbf{A} \cdot \vec{u}(t + \Delta t) + \vec{f}(t) \\ (\mathbf{M} + \Delta t \mathbf{A}) \cdot \vec{u}(t + \Delta t) &= \mathbf{M} \cdot \vec{u}(t) + \Delta t \vec{f}(t)\end{aligned}$$

and is unconditionally stable.

- The Crank–Nicolson scheme will lead to

$$\begin{aligned}\frac{1}{\Delta t} \mathbf{M} (\vec{u}(t + \Delta t) - \vec{u}(t)) &= -\frac{1}{2} (\mathbf{A} \cdot \vec{u}(t + \Delta t) + \mathbf{A} \cdot \vec{u}(t)) + \frac{1}{2} (\vec{f}(t) + \vec{f}(t + \Delta t)) \\ \left( \mathbf{M} + \frac{\Delta t}{2} \mathbf{A} \right) \cdot \vec{u}(t + \Delta t) &= \left( \mathbf{M} - \frac{\Delta t}{2} \mathbf{A} \right) \cdot \vec{u}(t) + \frac{\Delta t}{2} (\vec{f}(t) + \vec{f}(t + \Delta t))\end{aligned}$$

and is unconditionally stable.

Ex 4.18

### 4.5.7 A two Dimensional Dynamic Heat Equation

Equation (1.6) on page 12 describes the temperature distribution as a function of the space coordinates  $x$  and  $y$ , and the time variable  $t$ .

$$\begin{aligned}\frac{\partial}{\partial t} u(t, x, y) - \kappa \Delta u(t, x, y) &= f(t, x, y) && \text{for } 0 \leq x, y \leq 1 \text{ and } t \geq 0 \\ u(t, x, y) &= 0 && \text{for } (x, y) \text{ on boundary and } t \geq 0 \\ u(0, x, y) &= u_0(x, y) && \text{for } 0 \leq x, y \leq 1\end{aligned}\tag{4.8}$$

#### Explicit approximation

The explicit (with respect to time) finite difference approximation is determined by

$$\frac{1}{\Delta t} (\vec{u}_{i+1} - \vec{u}_i) = -\kappa \mathbf{A}_{nn} \vec{u}_i + \vec{f}_i$$

or

$$\vec{u}_{i+1} = \vec{u}_i - \Delta t (\kappa \mathbf{A}_{nn} \vec{u}_i - \vec{f}_i).$$

For each time step we have to multiply the matrix  $\mathbf{A}_{nn}$  with a vector. Due to the severe sparsity of the matrix this requires approximatley  $5 n^2$  multiplications. Since the largest eigenvalue is given by  $\kappa \lambda_{n,n} \approx \kappa 8 n^2 \approx \frac{8 \kappa}{(\Delta x)^2}$  we have the stability condition

$$\Delta t \leq \frac{2}{\kappa \lambda_{n,n}} \approx \frac{1}{4 \kappa} (\Delta x)^2.$$

The algorithm is conditionally stable only.

#### Implicit approximation

The implicit (with respect to time) finite difference approximation is determined by

$$\frac{1}{\Delta t} (\vec{u}_{i+1} - \vec{u}_i) = -\kappa \mathbf{A}_{nn} \vec{u}_{i+1} + \vec{f}_{i+1}$$

or

$$(\mathbb{I} + \Delta t \kappa \mathbf{A}_{nn}) \vec{u}_{i+1} = \vec{u}_i + \Delta t \vec{f}_{i+1}.$$

The algorithm is unconditionally stable. For each time-step a system of linear equations has to be solved, but the matrix is constant. Thus we can factorize the matrix once (Cholesky) and then do the back substitution steps only. The symmetric, positive definite matrix  $\mathbf{A}_{nn}$  has size  $n^2 \times n^2$  and a semi-bandwidth of  $b = n$ . Using the results in Section 2.6.4 the computational effort for one banded Cholesky factorization is approximated by  $\frac{1}{2} n^2 n^2$ . Each subsequent solving of a system of equation requires  $2 n^3$  multiplications.

### Crank–Nicolson approximation

The CN finite difference approximation is determined by

$$\frac{1}{\Delta t} (\vec{u}_{i+1} - \vec{u}_i) = -\frac{\kappa}{2} \mathbf{A}_{nn} (\vec{u}_i + \vec{u}_{i+1}) + \frac{1}{2} (\vec{f}_i + \vec{f}_{i+1})$$

or

$$\left( \mathbb{I} + \Delta t \frac{\kappa}{2} \mathbf{A}_{nn} \right) \vec{u}_{i+1} = \left( \mathbb{I} - \Delta t \frac{\kappa}{2} \mathbf{A}_{nn} \right) \vec{u}_i + \frac{\Delta t}{2} \left( \vec{f}_i + \vec{f}_{i+1} \right).$$

The algorithm is unconditionally stable too and the computational effort is comparable to the implicit method.

### Comparison

A comparison for the explicit, implicit and CN approximation is given in Table 4.4. For the implicit scheme each time step requires more computations than an explicit time step. The time steps for the implicit scheme may be larger. The choice of best algorithm thus depends on the time interval on which you want to compute the solution: for very small times the explicit scheme is more efficient, for very large times the implicit scheme is more efficient. This differs from the 1D situation in Table 4.3 where the computational effort for each time step was small and of the same order for the three algorithms examined. The Crank–Nicolson scheme can be applied to the 2D heat equation, leading to a higher order of consistency.

	explicit	implicit	Crank–Nicolson
order of consistency	$O(\Delta t) + O((\Delta x)^2)$	$O(\Delta t) + O((\Delta x)^2)$	$O((\Delta t)^2) + O((\Delta x)^2)$
condition on time step $\Delta t$	$\Delta t \leq \frac{1}{4\kappa} (\Delta x)^2$	no condition	no condition
linear system to be solved	no	yes	yes
flops for matrix factorization	none	$\frac{1}{2} n^4$	$\frac{1}{2} n^4$
flops for each time step	$5 n^2$	$2 n^3$	$2 n^3$

Table 4.4: Comparison of finite difference schemes for 2D dynamic heat equations

### A sample code in Octave/MATLAB

Below find Octave code to solve the initial boundary value problem with  $u_0(x, y) = 0$  and  $f(t, x, y) = x^2$  on the interval  $[0, T] = [0, 0.5]$  with  $dt = 0.02$  and  $nx \cdot ny = 34 \cdot 35$  interior grid points. Figure 4.21(b) shows that the temperature at an interior point converges towards a final value. The result in Figure 4.21(a) is, not surprisingly, very similar to Figure 4.14, i.e. the solution of the steady state problem  $-\Delta u = x^2$ . The results from Section 2.6.6 (page 69) are used to first determine the Cholesky factorization of the matrix and then for each time step use the back substitution only, this is efficient.

run demo

#### PlateDynamic.m

```
%%%%% script file to solve the dynamic heat equation on a unit square
%%%%% using an implicit finite difference scheme
T = 0.5; dt = 0.02;

nx = 34; ny = 35;
f = @(x,y) x.^2;

%%%%%%%%%%%%%
t = 0:dt:T; utrace = zeros(size(t));
hx = 1/(nx+1); hy = 1/(ny+1);
```

```

Dxx = spdiags(ones(nx,1)*[-1 2 -1], [-1 0 1], nx,nx) / (hx^2);
Dyy = spdiags(ones(ny,1)*[-1 2 -1], [-1 0 1], ny,ny) / (hy^2);
A = kron(Dxx, speye(ny)) + kron(speye(nx), Dyy);
Astep = speye(nx*ny,nx*ny) + dt*A;
[R,p,P] = chol(Astep); % Cholesky factorization, with permutations
% R = chol(A);
x = hx:hx:1-hx; y = hy:hy:1-hy;
u = zeros(nx*ny,1); % define the initial temperature
[xx,yy] = meshgrid(x,y); fvec = dt*f(xx(:),yy(:));

for k = 2:length(t);
    u = P*(R'\(P'*(u+fvec))); % one time step
%    u = R\ (R'\(u+fvec)); % one time step
    utrace(k) = u(55);
end%for

%%% add on the zero boundary values for a nicer graphics
x = 0:hx:1; y = 0:hy:1;
[xx,yy] = meshgrid(x,y); uu = zeros(size(xx));
uu(2:ny+1,2:nx+1) = reshape(u,ny,nx);

figure(1); surf(xx,yy,uu); xlabel('x'); ylabel('y');
figure(2); plot(t,utrace); grid on ; xlabel('Time t'); ylabel('Temp u');

```

In the above code we used a sparse Cholesky factorization to solve the system of linear equations at each time step. Since  $\mathbf{A}$  is a sparse, symmetric, positive definite matrix we may also use an iterative solver, e.g. the conjugate gradient algorithm. According to the results in Section 2.7 and in particular Figure 2.17 (page 84), this might be a faster solution for fine meshes. Since this is a time step algorithm we have good initial guesses of the solution of the linear system, using the result at the previous time step.

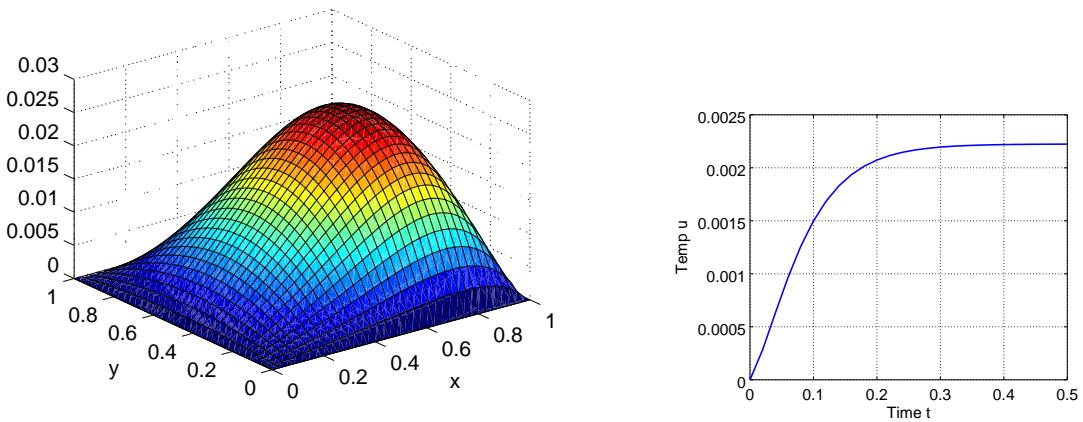


Figure 4.21: Solution of the dynamic heat equation on a square

#### 4.5.8 Comparison of a Few More Solvers for Dynamic Problems

not in class

We list some facts for a few more numerical solvers for the linear system of ODEs of the form

$$\frac{d}{dt} \vec{u}(t) = -\mathbf{A} \vec{u}(t) + \vec{f}(t) \quad (4.9)$$

or for the corresponding stability question

$$\frac{d}{dt} u(t) = -\lambda u(t) \quad \text{with} \quad u(0) = 1 \quad (4.10)$$

for values  $\lambda > 0$ . This was examined in Section 3.4.3 on page 195. The exact solution is obviously  $u(t) = u(0) \exp(-\lambda t)$ . Of interest are two aspects of the solvers:

1. stability condition:  $|u_i|$  has to remain bounded. This is the case for all  $\text{Re}(\lambda) < 0$  if the method is A-stable.
2. decay condition: for large values of  $\lambda$ , the solution should converge to 0 very fast. This is the case if the method is L-stable.

In a typical situation we find after one step  $u(\Delta t) \approx g(\lambda \Delta t)$ , where the function  $g()$  depends on the algorithm to be examined. For the exact solver use  $g(\lambda \Delta t) = \exp(-\lambda \Delta t)$ . The stability requires  $|g(\lambda \Delta t)| \leq 1$  and the decay condition translates to  $|g(\lambda \Delta t)| \ll 1$  for  $\lambda \Delta t \gg 1$ .

### The explicit solver

For the ODE (4.9) consider the basic explicit method

$$\begin{aligned} \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} &= -\mathbf{A} \vec{u}_i + \vec{f}_i \\ \vec{u}_{i+1} &= \vec{u}_i + \Delta t (-\mathbf{A} \vec{u}_i + \vec{f}_i). \end{aligned}$$

The method is consistent of order 1. For the stability (4.10) examine

$$\begin{aligned} \frac{u_{i+1} - u_i}{\Delta t} &= -\lambda u_i \\ u_{i+1} &= (1 - \Delta t \lambda) u_i \end{aligned}$$

and thus the function  $g_{Ex}(\lambda \Delta t) = 1 - \lambda \Delta t$ . This leads to the conditional stability  $\Delta t < \frac{2}{\lambda}$  and we find no decay at all.

### The basic implicit solver

For the ODE (4.9) consider the basic implicit method

$$\begin{aligned} \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} &= -\mathbf{A} \vec{u}_{i+1} + \vec{f}_{i+1} \\ (\mathbb{I} + \Delta t \mathbf{A}) \vec{u}_{i+1} &= \vec{u}_i + \Delta t \vec{f}_{i+1} \end{aligned}$$

The method is consistent of order 1. For the stability (4.10) examine

$$\begin{aligned} \frac{u_{i+1} - u_i}{\Delta t} &= -\lambda u_{i+1} \\ (1 + \Delta t \lambda) u_{i+1} &= u_i \\ u_{i+1} &= \frac{u_i}{1 + \Delta t \lambda} \end{aligned}$$

and thus the stability function with  $z = \lambda \Delta t$  is

$$g_{Im}(z) = \frac{1}{1 + z}$$

which leads to the unconditional stability and unconditional decay. The method as A-stable and L-stable.

### The Crank–Nicolson solver

For the ODE (4.9) consider

$$\begin{aligned}\frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} &= -\mathbf{A} \frac{\vec{u}_{i+1} + \vec{u}_i}{2} + \frac{\vec{f}_{i+1} + \vec{f}_i}{2} \\ (\mathbb{I} + \frac{\Delta t}{2} \mathbf{A}) \vec{u}_{i+1} &= (\mathbb{I} - \frac{\Delta t}{2} \mathbf{A}) \vec{u}_i + \frac{\Delta t}{2} (\vec{f}_{i+1} + \vec{f}_i)\end{aligned}$$

The method is consistent of order 2<sup>8</sup>. For the stability examine the model problem (4.10).

$$\begin{aligned}\frac{u_{i+1} - u_i}{\Delta t} &= -\lambda \frac{u_i + u_{i+1}}{2} \\ (1 + \frac{\Delta t}{2} \lambda) u_{i+1} &= (1 - \frac{\Delta t}{2} \lambda) u_i \\ u_{i+1} &= \frac{2 - \Delta t \lambda}{2 + \Delta t \lambda} u_i\end{aligned}$$

and thus the stability function with  $z = \lambda \Delta t$  is

$$g_{CN}(z) = \frac{2 - z}{2 + z}$$

which leads to the unconditional stability and no decay, since  $\lim_{z \rightarrow \infty} g(z) = -1$ . The method is A-stable but not L-stable.

### The BDF2 solver

This is an implicit method with good stability properties. Find a good presentation in [Butc03, §225]. The BDF2 (backwards differentiation formula) uses the approximations

$$\begin{aligned}\frac{d}{dt} u_i &\approx \frac{u_{i+1} - u_i}{\Delta t} \approx \frac{u_i - u_{i-1}}{\Delta t} \\ \frac{2}{3} \frac{d}{dt} u_i &\approx \frac{3}{3} \frac{u_{i+1} - u_i}{\Delta t} - \frac{1}{3} \frac{u_i - u_{i-1}}{\Delta t} = \frac{u_{i+1} - 4u_i + u_{i-1}}{3\Delta t}\end{aligned}$$

to realize that the ODE (4.9) can be discretized by

$$\begin{aligned}\frac{1}{\Delta t} \left( \vec{u}_{i+1} - \frac{4}{3} \vec{u}_i + \frac{1}{3} \vec{u}_{i-1} \right) &= -\frac{2}{3} \mathbf{A} \vec{u}_{i+1} + \frac{2}{3} \vec{f}_{i+1} \\ \vec{u}_{i+1} - \frac{4}{3} \vec{u}_i + \frac{1}{3} \vec{u}_{i-1} &= -\frac{2\Delta t}{3} \mathbf{A} \vec{u}_{i+1} + \frac{2\Delta t}{3} \vec{f}_{i+1} \\ (\mathbb{I} + \frac{2\Delta t}{3} \mathbf{A}) \vec{u}_{i+1} &= +\frac{4}{3} \vec{u}_i - \frac{1}{3} \vec{u}_{i-1} + \frac{2\Delta t}{3} \vec{f}_{i+1} \\ (3\mathbb{I} + 2\Delta t \mathbf{A}) \vec{u}_{i+1} &= +4\vec{u}_i - \vec{u}_{i-1} + 2\Delta t \vec{f}_{i+1}\end{aligned}$$

---

<sup>8</sup>Can be verified by comparing

$$\begin{aligned}g_{CN}(z) &= \frac{2-z}{2+z} = \frac{2+z}{2+z} - \frac{2z}{2+z} = 1 - 2z \frac{1/2}{1+z/2} = 1 - z \frac{1}{1+z/2} \\ &= 1 - z \left( 1 - \frac{z}{2} + \frac{z^2}{2^2} - \frac{z^3}{2^3} + \dots \right) = 1 - z + \frac{1}{2} z^2 - \frac{1}{4} z^3 + \frac{1}{8} z^4 - \dots \\ \exp(-z) &= 1 - z + \frac{1}{2} z^2 - \frac{1}{6} z^3 + \dots\end{aligned}$$

The method is consistent of order 2. Observe that 2 time levels are used to advance by one time step. Thus one needs another algorithm to start BDF2, e.g. one RK step, which is also of order 2. To examine the stability use the model equation (4.10), i.e. examine  $\frac{d}{dt} u(t) = -\lambda u(t)$ .

$$\begin{aligned} u_{i+1} - \frac{4}{3} u_i + \frac{1}{3} u_{i-1} &= -\lambda \frac{2 \Delta t}{3} u_{i+1} \\ (3 + 2 \lambda \Delta t) u_{i+1} &= 4 u_i - u_{i-1} \\ u_{i+1} &= \frac{4}{3 + 2 \lambda \Delta t} u_i - \frac{1}{3 + 2 \lambda \Delta t} u_{i-1} \\ \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix} &= \begin{bmatrix} 0 & 1 \\ \frac{-1}{3+2\lambda\Delta t} & \frac{4}{3+2\lambda\Delta t} \end{bmatrix} \begin{pmatrix} u_{i-1} \\ u_i \end{pmatrix} \end{aligned}$$

To examine the stability of this scheme use the eigenvalues of this matrix.

$$\begin{aligned} 0 &= \det \begin{bmatrix} -\mu & 1 \\ \frac{-1}{3+2\lambda\Delta t} & \frac{4}{3+2\lambda\Delta t} - \mu \end{bmatrix} = \mu^2 - \frac{4}{3+2\lambda\Delta t} \mu + \frac{1}{3+2\lambda\Delta t} \\ 0 &= (3 + 2 \lambda \Delta t) \mu^2 - 4 \mu + 1 \\ \mu_{1,2} &= \frac{1}{2(3 + 2 \lambda \Delta t)} \left( +4 \pm \sqrt{16 - 4(3 + 2 \lambda \Delta t)} \right) \\ &= \frac{1}{3 + 2 \lambda \Delta t} \left( +2 \pm \sqrt{4 - (3 + 2 \lambda \Delta t)} \right) = \frac{+2 \pm \sqrt{1 - 2 \lambda \Delta t}}{3 + 2 \lambda \Delta t} \end{aligned}$$

To examine this expression consider two different cases for  $z = \lambda \Delta t$ :

- $z \leq \frac{1}{2}$ : Use  $g_{1,2}(z) = \frac{2 \pm \sqrt{1 - 2 z}}{3 + 2 z}$  and

$$g_{BDF2}(z) = \max\{g_{1,2}(z)\} = \frac{2 + \sqrt{1 - 2 z}}{3 + 2 z}$$

- $z > \frac{1}{2}$ : Use  $g_{1,2}(z) = \frac{2 \pm i\sqrt{2 z - 1}}{3 + 2 z} \in \mathbb{C}$  and

$$g_{BDF2}(z) = \max\{|g_{1,2}(z)|\} = \frac{\sqrt{2^2 + 2 z - 1}}{3 + 2 z} = \frac{\sqrt{3 + 2 z}}{3 + 2 z} = \frac{1}{\sqrt{3 + 2 z}}$$

Verify that for  $z > 0$  the absolute values are smaller than 1, and thus find unconditional stability. For the decay condition observe that  $\lim_{z \rightarrow \infty} |g_{1,2}(z)| = 0$ . This is based on  $g_{BDF2}(z) \approx \frac{1}{\sqrt{2 z}}$  for  $z \gg 1$ . The method is A-stable and L-stable. To verify the second order consistency compare  $\exp(-z) \approx 1 - z + \frac{1}{2} z^2 - \frac{1}{6} z^3$  with the Taylor approximation

$$g(z) = \frac{2 + \sqrt{1 - 2 z}}{3 + 2 z} \approx 1 - z + \frac{1}{2} z^2 - \frac{1}{2} z^3.$$

### An L-stable Runge–Kutta solver

This is a *diagonally implicit Runge–Kutta* or DIRK method, find a good presentation in [Butc03, §361]. The ODE (4.9) is discretized by

$$\begin{aligned} (\mathbb{I} + \theta \Delta t \mathbf{A}) \vec{u}_n &= (\mathbb{I} - (1 - \theta) \Delta t \mathbf{A}) \vec{u}_i \\ (\mathbb{I} + \theta \Delta t \mathbf{A}) \vec{u}_{i+1} &= \left( \mathbb{I} - \frac{1}{2} \Delta t \mathbf{A} \right) \vec{u}_i - \left( \frac{1}{2} - \theta \right) \Delta t \mathbf{A} \vec{u}_n \end{aligned}$$

where  $\theta = 1 - 1/\sqrt{2}$ . The method is consistent of order 2. The Butcher table (see Section 3.4.4) of this algorithm is

$$\begin{array}{c|cc} 1 - \frac{1}{\sqrt{2}} & 1 - \frac{1}{\sqrt{2}} & 0 \\ \hline 1 & \frac{1}{\sqrt{2}} & 1 - \frac{1}{\sqrt{2}} \\ \hline y_{n+1} & \frac{1}{\sqrt{2}} & 1 - \frac{1}{\sqrt{2}} \end{array} .$$

Observe that two systems of linear equations have to be solved for each step, but using the same matrix  $\mathbb{I} - \theta \Delta t \mathbf{A}$ . To examine the stability use (4.10) and

$$\begin{aligned} (1 + \theta \lambda \Delta t) u_n &= (1 - (1 - \theta) \lambda \Delta t) u_i \\ (1 + \theta \lambda \Delta t) u_{i+1} &= (1 - \frac{\lambda}{2} \Delta t) u_i - (\frac{1}{2} - \theta) \lambda \Delta t u_n \\ &= (1 - \frac{\lambda}{2} \Delta t) u_i - (\frac{1}{2} - \theta) \frac{1 - (1 - \theta) \lambda \Delta t}{1 + \theta \lambda \Delta t} \lambda \Delta t u_i \\ &= \left( (1 - \frac{\lambda}{2} \Delta t) - \frac{(\frac{1}{2} - \theta)(1 - (1 - \theta) \lambda \Delta t)}{1 + \theta \lambda \Delta t} \lambda \Delta t \right) u_i \end{aligned}$$

with  $\theta = 1 - 1/\sqrt{2} \approx 0.29$ . Thus examine the stability function ( $z = \lambda \Delta t$ )

$$g_{RK}(z) = \frac{1}{1 + \theta z} \left( (1 - \frac{z}{2}) - \frac{(\frac{1}{2} - \theta)(1 - (1 - \theta)z)}{1 + \theta z} z \right) .$$

A Taylor approximation for  $|z| \ll 1$  of this function is given by

$$g_{RK}(z) = 1 - z + \frac{1}{2} z^2 - \frac{\sqrt{2} - 1}{2} z^3 + O(z^4)$$

and a comparisons with  $\exp(-z) = 1 - z + \frac{1}{2} z^2 - \frac{1}{6} z^3 + O(z^4)$  verifies that this ODE solver is consistent of order 2. Use a plot of this function for  $z > 0$  to observe that  $|g(z)| < 1$  and thus the method is unconditionally stable. For  $z \gg 1$  use  $1 - (1 - \theta)z \approx -(1 - \theta)z$  and symbolic math program to verify

$$g_{RK}(z) \approx \frac{1}{1 + \theta z} \left( (1 - \frac{z}{2}) + \frac{(\frac{1}{2} - \theta)(1 - \theta)z}{1 + \theta z} z \right) = \frac{4 + 2(1 - \sqrt{2})z}{(2 + (2 - \sqrt{2})z)^2} .$$

For  $z \gg 1$  find

$$\begin{aligned} g_{RK}(z) &\approx \frac{2(1 - \sqrt{2})z}{(2 - \sqrt{2})^2 z^2} = \frac{2(2 + \sqrt{2})^2 (1 - \sqrt{2})}{(2 + \sqrt{2})^2 (2 - \sqrt{2})^2 z} \\ &= \frac{2(6 + 4\sqrt{2})(1 - \sqrt{2})}{(4 - 2)^2 z} = \frac{6 - 8 - 2\sqrt{2}}{2z} = \frac{-1 - \sqrt{2}}{z} . \end{aligned}$$

Consequently we have unconditional decay. The method is A-stable and L-stable.

### Comparison of the solvers

The above algorithms should be compared using different criteria. Table 4.5 shows the key properties and in Figure 4.22 find the graphs of the stability functions. The values of  $g(z)$  are shown and should be compared to  $\exp(-z)$ , which represents the exact solution of  $\frac{d}{dt} u(t) = -\lambda u(t)$ .

algorithm	order	$g(z)$ for $z \gg 1$	time levels	systems to solve
implicit	1	$\approx \frac{1}{z}$	1	1
Crank–Nicolson	2	$\approx -1$	1	1
BDF2	2	$\approx \frac{1}{\sqrt{2}z} \approx \frac{0.71}{\sqrt{z}}$	2	1
Runge–Kutta, L-stable	2	$\approx \frac{-1-\sqrt{2}}{z}$	1	2

Table 4.5: For different ODE solvers find the order of consistency, the asymptotic approximation of the stability function  $g(z)$ , the number of time levels used and the number of linear systems to be solved to perform one time step.

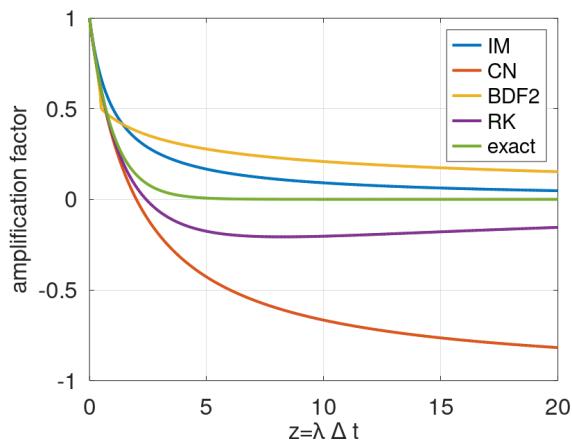


Figure 4.22: Stability functions  $g(z)$  for four algorithms with  $z = \lambda \Delta t$

## 4.6 Hyperbolic Problems, Wave Equations

The simplest form of a wave equation is

$$\begin{aligned} \frac{\partial^2}{\partial t^2} u(t, x) &= c^2 \frac{\partial^2}{\partial x^2} u(t, x) && \text{for } 0 < x < 1 \quad \text{and} \quad t > 0 \\ u(t, 0) = u(t, 1) &= 0 && \text{for } t > 0 \\ u(0, x) &= u_0(x) && \text{for } 0 < x < 1 \\ \dot{u}(0, x) &= u_1(x) && \text{for } 0 < x < 1 \end{aligned} . \quad (4.11)$$

The equation of a vibrating string (1.8) on page 14 is of this form. Examine an explicit and an implicit finite difference approximation.

### 4.6.1 An Explicit Approximation

Examine the finite difference approximation on a grid given by Figure 4.23. Given  $\vec{u}_{i-1}$  and  $\vec{u}_i$  compute the solution  $\vec{u}_{i+1}$  with a multiplication by a matrix, thus this is an explicit scheme. The finite difference scheme is consistent of order  $(\Delta x)^2 + (\Delta t)^2$ .

$$\frac{\vec{u}_{i+1} - 2\vec{u}_i + \vec{u}_{i-1}}{(\Delta t)^2} = -c^2 \mathbf{A}_n \cdot \vec{u}_i$$

$$\begin{aligned} \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta t)^2} &= c^2 \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta x)^2} \\ \frac{\vec{u}_{i+1} - 2\vec{u}_i + \vec{u}_{i-1}}{(\Delta t)^2} &= -c^2 \mathbf{A}_n \vec{u}_i \\ \vec{u}_{i+1} &= (2\mathbb{I} - (\Delta t)^2 c^2 \mathbf{A}_n) \vec{u}_i - \vec{u}_{i-1} \end{aligned}$$

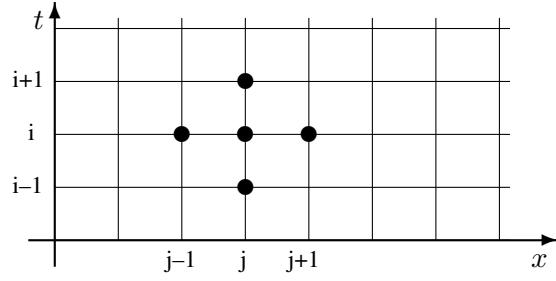


Figure 4.23: Explicit finite difference approximation for the wave equation

### Stability of the explicit scheme

The next point to be considered is the stability of the finite difference scheme. The technique used is very similar to the procedure used in Section 4.3.2 (page 258) to examine stability of the finite difference approximation to the dynamic heat equation. Since the time derivatives are of order 2 it is convenient to first examine the ordinary differential equation

$$\frac{d^2}{dt^2} \alpha(t) = -\lambda \alpha(t)$$

with exact solutions  $\alpha(t) = A \cos(\sqrt{\lambda}t + \delta)$ . Thus the solution remains bounded for all times  $t$ . Insist on the solutions of the approximate equation

$$\frac{\alpha(t+h) - 2\alpha(t) + \alpha(t-h)}{h^2} = -\lambda \alpha(t)$$

to remain bounded too. Solve the above difference equation for  $\alpha(t+h)$  to find

$$\alpha(t+h) = 2\alpha(t) - \alpha(t-h) - h^2 \lambda \alpha(t).$$

Using a matrix notation write this in the form

$$\begin{pmatrix} \alpha(t) \\ \alpha(t+h) \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 2 - \lambda h^2 \end{bmatrix} \cdot \begin{pmatrix} \alpha(t-h) \\ \alpha(t) \end{pmatrix}$$

and with an iteration find

$$\begin{pmatrix} \alpha(ih) \\ \alpha((i+1)h) \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 2 - \lambda h^2 \end{bmatrix}^i \cdot \begin{pmatrix} \alpha(0) \\ \alpha(h) \end{pmatrix}.$$

These solutions remain bounded as  $i \rightarrow \infty$  if the eigenvalues  $\mu$  of the matrix have absolute values smaller or equal to 1<sup>9</sup>. Thus we examine the solutions of the characteristic equation

$$\det \begin{bmatrix} 0 - \mu & 1 \\ -1 & 2 - \lambda h^2 - \mu \end{bmatrix} = \mu^2 - \mu(2 - \lambda h^2) + 1 = 0.$$

Using a factorization of this polynomial find

$$\mu^2 - \mu(2 - \lambda h^2) + 1 = (\mu - \mu_1)(\mu - \mu_2) = \mu^2 - (\mu_1 + \mu_2)\mu + \mu_1 \mu_2.$$

<sup>9</sup>If  $\mu$  is an eigenvalue of the matrix  $\mathbf{A}$  with eigenvector  $\vec{v}$  then  $\mathbf{A}^k \vec{v} = \mu^k \vec{v}$ . This expression remains bounded iff  $|\mu| \leq 1$ . We quietly assume that the matrix  $\mathbf{A}$  is diagonalizable.

Since the constant term equals 1, conclude that  $\mu_1 \cdot \mu_2 = 1$ . If both values  $\mu_{1,2}$  would be real and  $\mu_1 \cdot \mu_2 = 1$  then  $\mu_2 = 1/\mu_1$  and one of the absolute values would be larger than 1. If the values are conjugate complex use  $1 = \mu_1 \cdot \mu_2 = \mu_1 \cdot \overline{\mu_1} = |\mu_1|^2$ . Thus the condition for  $|\mu_{1,2}| \leq 1$  to be correct is given by: conjugate complex values on the unit circle with nonzero imaginary part. The solutions  $\mu_{1,2}$  are given by

$$\begin{aligned}\mu_{1,2} &= \frac{1}{2} \left( 2 - \lambda h^2 \pm \sqrt{(2 - \lambda h^2)^2 - 4} \right) = \frac{1}{2} \left( 2 - \lambda h^2 \pm \sqrt{\lambda^2 h^4 - 4\lambda h^2} \right) \\ &= \frac{2 - \lambda h^2 \pm \sqrt{\lambda h^2} \sqrt{\lambda h^2 - 4}}{2}.\end{aligned}$$

Thus as a necessary and sufficient condition for stability use a negative discriminant.

$$\lambda^2 h^4 - 4\lambda h^2 < 0 \iff \lambda h^2 \leq 4 \iff h^2 \leq \frac{4}{\lambda}$$

### Setting up the initial values

To start the iteration the values of  $u(0)$  and  $u(\Delta t)$  have to be available. A simple approach is to use the initial velocity  $\dot{u}(0) = v_0$  and thus  $u(\Delta t) \approx u(0) + v_0 \Delta t$ . This approximation is consistent of order  $\Delta t$ . A better approach is to use the centered approximation and the differential equation at  $t = 0$ . The idea used to improve the consistency is similar to the approach in Example 4–7 used to discretize the boundary condition  $u'(L) = F$ .

$$\begin{aligned}\frac{\partial}{\partial t} u(0) &\approx \frac{u(\Delta t) - u(-\Delta t)}{2\Delta t} = v_0 \\ u(-\Delta t) &= u(\Delta t) - 2v_0\Delta t \\ \ddot{u}(0) &\approx \frac{u(-\Delta t) - 2u(0) + u(\Delta t)}{(\Delta t)^2} = -\lambda u(0) \\ \frac{(u(\Delta t) - 2v_0\Delta t) - 2u(0) + u(-\Delta t)}{(\Delta t)^2} &= -\lambda u(0) \\ u(\Delta t) &= u(0) + v_0\Delta t - \frac{1}{2}\lambda u(0)(\Delta t)^2.\end{aligned}$$

With the additional term the approximation is consistent of order  $(\Delta t)^2$ .

When applied to the equation  $\frac{\partial^2}{\partial t^2} \vec{u}(t) = -c^2 \mathbf{A} \vec{u}(t) + \vec{f}(t)$  with the initial conditions  $\vec{u}(0) = \vec{u}_0$  and  $\frac{\partial}{\partial t} \vec{u}(0) = v_0$  use

$$\begin{aligned}\frac{\partial}{\partial t} \vec{u}(0) &\approx \frac{\vec{u}(\Delta t) - \vec{u}(-\Delta t)}{2\Delta t} = \frac{\vec{u}_{+1} - \vec{u}_{-1}}{2\Delta t} = \vec{v}_0 \\ u_{-1} &= u_{+1} - 2\Delta t v_0 \\ \frac{u_{-1} - 2u_0 + u_{+1}}{(\Delta t)^2} &\approx \frac{\partial^2}{\partial t^2} \vec{u}(0) = -c^2 \mathbf{A} \vec{u}_0 + \vec{f}_0 \\ \frac{(u_{+1} - 2\Delta t v_0) - 2u_0 + u_{-1}}{(\Delta t)^2} &= \frac{2u_{+1} - 2\Delta t v_0 - 2u_0}{(\Delta t)^2} = -c^2 \mathbf{A} \vec{u}_0 + \vec{f}_0 \\ u_{+1} &= u_0 + \Delta t \vec{v}_0 + \frac{(\Delta t)^2}{2} (-c^2 \mathbf{A} \vec{u}_0 + \vec{f}_0).\end{aligned}$$

This approximation<sup>10</sup> is consistent of order  $(\Delta t)^2$ .

---

<sup>10</sup>Currently this is not implemented yet in some of my sample codes.

### Solving the equation by time stepping

Now return to the wave equation. With the notation from the previous section write the discretization scheme in Figure 4.23 in the form

$$\vec{u}_{i+1} - 2\vec{u}_i + \vec{u}_{i-1} = -c^2 (\Delta t)^2 \mathbf{A}_n \cdot \vec{u}_i$$

or, when solved for  $\vec{u}_{i+1}$ ,

$$\vec{u}_{i+1} = (2\mathbb{I}_n - c^2 (\Delta t)^2 \mathbf{A}_n) \cdot \vec{u}_i - \vec{u}_{i-1}.$$

With a block matrix notation this can be transformed in a form similar to the ODE situation above.

$$\begin{pmatrix} \vec{u}_i \\ \vec{u}_{i+1} \end{pmatrix} = \begin{bmatrix} 0 & \mathbb{I}_n \\ -\mathbb{I}_n & 2\mathbb{I}_n - c^2 (\Delta t)^2 \mathbf{A}_n \end{bmatrix} \cdot \begin{pmatrix} \vec{u}_{i-1} \\ \vec{u}_i \end{pmatrix}.$$

Then write the solution as a linear combination of the eigenvectors of the matrix  $\mathbf{A}_n$ , i.e.

$$\vec{u}(t) = \sum_{k=1}^n \alpha_k(t) \vec{v}_k \quad \text{where} \quad \mathbf{A}_n \vec{v}_k = \lambda_k \vec{v}_k.$$

The above matrix is replaced by

$$\begin{bmatrix} 0 & 1 \\ -1 & 2 - c^2 (\Delta t)^2 \lambda_k \end{bmatrix}$$

and powers of this matrix have to remain bounded, for all eigenvalues  $\lambda_k$ . The stability condition for the ODE leads to  $c^2 (\Delta t)^2 \lambda_k \leq 4$  for  $k = 1, 2, 3, \dots, n$ . Since the largest eigenvalue is given by

$$\lambda_n = \frac{4}{\Delta x^2} \sin^2 \frac{n\pi}{2(n+1)} \approx \frac{4}{\Delta x^2} \sin^2 \pi/2 = \frac{4}{\Delta x^2}$$

find the stability condition

$$c^2 \frac{(\Delta t)^2}{(\Delta x)^2} \leq 1 \iff c^2 (\Delta t)^2 \leq (\Delta x)^2 \iff c \Delta t \leq \Delta x.$$

The solution at the first two time levels has to be known to get the finite difference scheme started. We have to use the initial conditions to construct the vectors  $u_0$  and  $u_1$ . The first initial condition in equation (4.11) obviously implies that  $\vec{u}_0$  should be the discretization of  $u(x, 0) = u_0(x)$ . The Octave code below is an elementary implementation of the presented finite difference scheme. In the example below the initial velocity  $v_0(x) = 0$  is used and then determined  $\vec{u}_1$  by the method in the previous section. If the ratio  $r = \frac{\Delta t}{\Delta x}$  is increased beyond the critical value of  $1/c$  then the algorithm is unstable and the solution will be far away from the true solution. The instability is again (as in Section 4.5.3) in the direction of the eigenvector belonging to the largest eigenvalue.

[run demo](#)

#### Wave.m

```
L = 3; % length of the space interval
n = 150; % number of interior grid points
r = 0.99; % ratio to compute time step
T = 6; % final time
iv = @(x)max([min([2*x'; 2-2*x']); 0*x']); % initial value

dx = L/(n+1); dt = r*dt; x = linspace(0,L,n+2)';

y0 = iv(x); y0(1) = 0; y0(n+2) = 0;
% use zero initial speed
y1 = y0 + dt^2/2*[0;diff(y0,2);0]/dx^2; % improved initialization
y2 = y0; % reserve the memory
```

```

figure(1); clf;
for t = 0:dt:T+dt;
    plot(x,y0); axis([0,L,-1,1]); drawnow(); %% graphics uses most of the time
    if 0 %% for loops, slow
        for k = 2:n+1
            y2(k)=(2-2*r^2)*y1(k)+r^2*(y1(k-1)+y1(k+1))-y0(k);
        end%for
    else %% no loop, fast
        y2(2:n+1) = (2-2*r^2)*y1(2:n+1) + r^2*(y1(1:n)+y1(3:n+2))-y0(2:n+1);
    end%if
    y0 = y1; y1 = y2;
end%for

figure(2); plot(x,y0,x,iv(x))

```

### Conditional stability, based on the cone of dependence, D'Alembert's solution

In the above section we examined the conditional stability of the explicit scheme based on eigenvalues of the corresponding matrix. One may also use a more physical argument to examine the stability condition.

Using calculus one may verify that the unique solution of the initial value problem

$$\begin{aligned}
 \frac{\partial^2}{\partial t^2} u(t, x) &= c^2 \frac{\partial^2}{\partial x^2} u(t, x) && \text{for } -\infty < x < \infty \text{ and } t \in \mathbb{R} \\
 u(0, x) &= u_0(x) && \text{for } -\infty < x < \infty \\
 \frac{\partial}{\partial t} u(0, x) &= u_1(x) && \text{for } -\infty < x < \infty
 \end{aligned} \tag{4.12}$$

is given by D'Alembert's formula.

$$u(t, x) = \frac{1}{2} (u_0(x - ct) + u_0(x + ct)) + \frac{1}{2c} \int_{x-ct}^{x+ct} u_1(\xi) d\xi \tag{4.13}$$

This implies that the solution of the wave equation at time  $t$  and position  $x$  is determined by the values in the **cone of dependence**, i.e. all times  $\tau < t$  and positions  $\tilde{x}$  such that  $|\tilde{x} - x| \leq c(t - \tau)$ . This is visualized in Figure 4.24. On the left find the domain having an influence on the solution at time  $t$  and position  $x$  and on the right the domain influenced by the values at time  $t = 0$  and position  $x$ . This formula and figure also confirm that information is traveling at most with speed  $c$ .

A quick look at Figure 4.23 (page 290) will confirm that for the explicit finite difference scheme the cone of dependence has a slope of  $\frac{\Delta t}{\Delta x}$ , while the slope for the exact solution in Figure 4.24 is given by  $1/c$ . Since the numerical cone has to contain the exact cone we have the condition

$$\frac{\Delta t}{\Delta x} \leq \frac{1}{c} \implies \Delta t \leq \frac{1}{c} \Delta x.$$

This confirms the stability condition obtained using eigenvalues.

### 4.6.2 An Implicit Approximation

Since the explicit method is again conditionally stable we consider an implicit method, which turns out to be unconditionally stable. The space discretization at time level  $i$  in the previous section is replaced by a weighted average of discretizations at levels  $i - 1$ ,  $i$  and  $i + 1$ .

$$\frac{\vec{u}_{i+1} - 2\vec{u}_i + \vec{u}_{i-1}}{(\Delta t)^2} = -\frac{c^2}{4} (\mathbf{A}_n \cdot \vec{u}_{i+1} + 2\mathbf{A}_n \cdot \vec{u}_i + \mathbf{A}_n \cdot \vec{u}_{i-1}).$$

One can verify (tedious computations) that this difference scheme is consistent of order  $(\Delta x)^2 + (\Delta t)^2$ . As a consequence we obtain a linear system of equations for  $\vec{u}_{i+1}$ . Thus this is an implicit scheme.

$$\left( \mathbb{I} + c^2 \frac{(\Delta t)^2}{4} \mathbf{A}_n \right) \vec{u}_{i+1} = \left( 2\mathbb{I} - 2c^2 \frac{(\Delta t)^2}{4} \mathbf{A}_n \right) \vec{u}_i - \left( \mathbb{I} + c^2 \frac{(\Delta t)^2}{4} \mathbf{A}_n \right) \vec{u}_{i-1}$$

if time permits

Ex 4.17

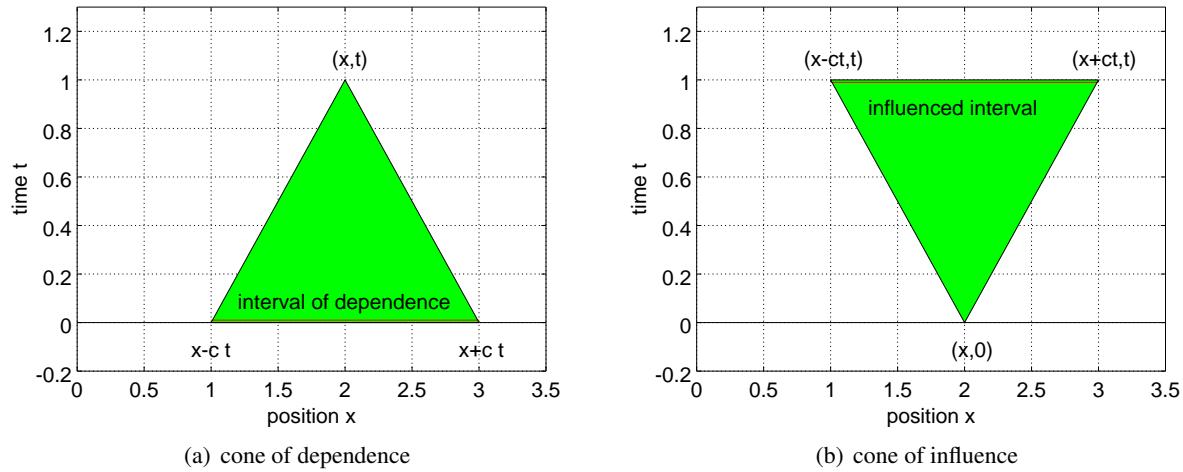


Figure 4.24: D'Alembert's solution of the wave equation

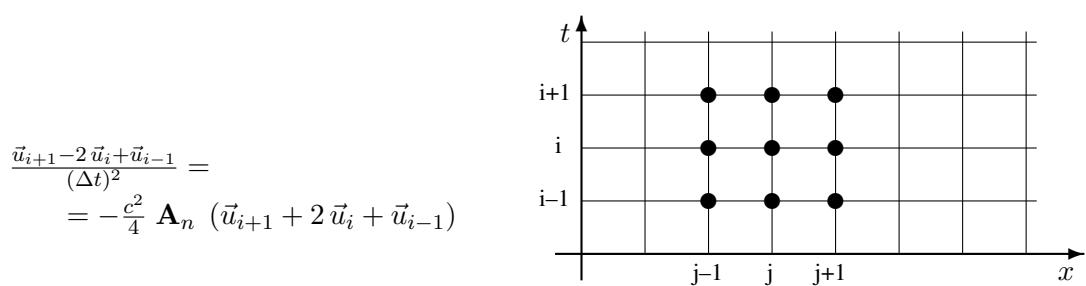


Figure 4.25: Implicit finite difference approximation for the wave equation

### Stability of the implicit scheme

To examine the stability consider eigenvalues  $\lambda > 0$  and the corresponding eigen vectors and use the notation

$$\gamma = c^2 \lambda \frac{(\Delta t)^2}{4} > 0.$$

Now examine the time discretization of  $\frac{d^2}{dt^2} \alpha(t) = -c^2 \lambda \alpha(t)$ .

$$\begin{aligned} \alpha(t + \Delta t) - 2\alpha(t) + \alpha(t - \Delta t) &= -c^2 \lambda \frac{(\Delta t)^2}{4} (\alpha(t + \Delta t) + 2\alpha(t) + \alpha(t - \Delta t)) \\ (1 + \gamma) \alpha(t + \Delta t) &= (2 - 2\gamma) \alpha(t) - (1 + \gamma) \alpha(t - \Delta t) \\ \begin{pmatrix} \alpha(t) \\ \alpha(t + \Delta t) \end{pmatrix} &= \begin{bmatrix} 0 & 1 \\ -1 & \frac{2-2\gamma}{1+\gamma} \end{bmatrix} \begin{pmatrix} \alpha(t - \Delta t) \\ \alpha(t) \end{pmatrix} \end{aligned}$$

Examine the eigenvalues  $\mu_{1,2}$  of this matrix by solving the quadratic equation

$$\det \begin{bmatrix} -\mu & 1 \\ -1 & 2 \frac{1-\gamma}{1+\gamma} - \mu \end{bmatrix} = \mu^2 - \frac{2-2\gamma}{1+\gamma} \mu + 1 = 0$$

and observe that  $\mu_1 \cdot \mu_2 = 1$ . Using the discriminant

$$4 \left( \frac{1-\gamma}{1+\gamma} \right)^2 - 4 < 0$$

conclude that two values  $\mu_{1,2}$  are complex conjugate. This implies  $|\mu_1| = |\mu_2| = 1$  and the scheme is **unconditionally stable**.

### 4.6.3 General Wave Type Problems

A more general form of a wave type, dynamic boundary value problem may be given in the form

$$\frac{d}{dt^2} \mathbf{M} \vec{u}(t) = -\mathbf{A} \vec{u}(t) + \vec{f}(t)$$

and the corresponding explicit scheme is

$$\begin{aligned} \mathbf{M} (\vec{u}_{i+1} - 2\vec{u}_i + \vec{u}_{i-1}) &= -(\Delta t)^2 \mathbf{A} \vec{u}_i + (\Delta t)^2 \vec{f}_i \\ \mathbf{M} \vec{u}_{i+1} &= (2\mathbf{M} - (\Delta t)^2 \mathbf{A}) \vec{u}_i - \mathbf{M} \vec{u}_{i-1} + (\Delta t)^2 \vec{f}_i. \end{aligned}$$

The scheme will be stable if

$$(\Delta t)^2 < 4/\lambda_n$$

where  $\lambda_n$  is the largest of the generalized eigenvalues, i.e. nonzero solutions of

$$\mathbf{A} \vec{v} = \lambda \mathbf{M} \vec{v}.$$

Thus the explicit scheme is **conditionally stable**.

An implicit scheme is given by

$$\mathbf{M} \frac{\vec{u}_{i+1} - 2\vec{u}_i + \vec{u}_{i-1}}{(\Delta t)^2} = -\frac{1}{4} (\mathbf{A} \vec{u}_{i+1} + 2\mathbf{A} \vec{u}_i + \mathbf{A} \vec{u}_{i-1}) + \vec{f}_i$$

or equivalently

$$\left( \mathbf{M} + \frac{(\Delta t)^2}{4} \mathbf{A} \right) \vec{u}_{i+1} = 2 \left( \mathbf{M} - \frac{(\Delta t)^2}{4} \mathbf{A} \right) \vec{u}_i - \left( \mathbf{M} + \frac{(\Delta t)^2}{4} \mathbf{A} \right) \vec{u}_{i-1} + (\Delta t)^2 \vec{f}_i.$$

For a generalized eigenvalue  $\lambda$  with  $\mathbf{A} \vec{v} = -\lambda \mathbf{M} \vec{v}$  and vectors  $\vec{u}_i = \alpha_i \vec{v}$  this translates to

$$\left(1 - \frac{(\Delta t)^2}{4} \lambda\right) \alpha_{i+1} = 2 \left(1 + \frac{(\Delta t)^2}{4} \lambda\right) \alpha_i - \left(1 - \frac{(\Delta t)^2}{4} \lambda\right) \alpha_{i-1}$$

With the abbreviation  $\gamma = \frac{\Delta t^2}{4} \lambda > 0$  this can be written as a system

$$\begin{pmatrix} \alpha_i \\ \alpha_{i+1} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & \frac{2+2\gamma}{1-\gamma} \end{bmatrix} \begin{pmatrix} \alpha_{i-1} \\ \alpha_i \end{pmatrix}$$

and this scheme is **unconditionally stable**, as expected.

## 4.7 Nonlinear Problems

### 4.7.1 Partial Substitution or Picard Iteration

#### 4–9 Example : Stretching of a Beam by a given Force and Variable Cross Section

In the above example we take into account that the cross section will change, due to the stretching of the beam, i.e. we take Poisson contraction into account. The mathematical description is given in equation (1.14) (see page 17).

$$-\frac{d}{dx} \left( E A_0(x) \left(1 - \nu \frac{du}{dx}\right)^2 \frac{du}{dx} \right) = f(x) \quad \text{for } 0 < x < L$$

with boundary conditions  $u(0) = 0$  and

$$E A_0(L) \left(1 - \nu \frac{du(L)}{dx}\right)^2 \frac{du(L)}{dx} = F.$$

This is a nonlinear boundary value problem for the unknown displacement function  $u(x)$ . We will use the method of successive substitution from Section 3.1.4. Thus we proceed as follows:

- Pick a starting function  $u_0(x)$ . If possible use a good guess to the solution. For this example  $u_0(x) = 0$  will do.
- While changes are too large
  - Compute the coefficient function

$$a(x) = E A_0(x) \left(1 - \nu \frac{du(x)}{dx}\right)^2$$

and then solve the linear boundary problem

$$-\frac{d}{dx} \left( a(x) \frac{du(x)}{dx} \right) = f(x).$$

This is the problem in Example 4–8 on page 267.

- Take this solution as your current solution and estimate its error by comparing with the previous solution.
- Show your final solution.

The above algorithm is implemented in Octave/MATLAB.

**BeamNL.m**

run demo

```

nu = 0.3; L = 3; F = 0.2;
EA = @(x) (2-sin(x/L*pi))/2;
fRHS = @(x) zeros(size(x));
%%%%%%%%%%%%%
N = 500; h = L/(N+1); % stepsize
x = (h:h:L)'; f = [fRHS(x)];
u = zeros(size(f)); g = h^2*f; g(N+1) = g(N+1)/2 + h*F;

cc = 1; cstring = 'rgbcmykrgbcmyk'; %% color counter and color sequence

figure(1); clf; hold on; grid on; axis([0 3 0 1.4]) % setup of graphics
xlabel('position x'); ylabel('displacement u');
relDiffTol = 1e-5; % choose the relative difference tolerance
Differences = []; relDiff = 2*relDiffTol;
while relDiff > relDiffTol
    a = EA(x-h/2).* (1-nu*diff([0;u])/h).^2; % compute coefficients
    di = [a(1:N)+a(2:N+1); a(N+1)]; % diagonal entries
    up = -a(2:N+1); % upper diagonal entries
    uNew = trisolve(di,up,g); % solve the linear symmetric system
    plot([0;x],[0;u],'linewidth',2,'color',cstring(cc)); cc = cc+1;
    pause(0.5)
    relDiff = max(abs(u-uNew))/max(abs(uNew)) % determine relative difference
    Differences = [Differences;relDiff]; % store the relative differences
    u = uNew; % prepare for restart
end%while
axis([0 3 0 1.4])
xlabel('position x'); ylabel('displacement u'); hold off

figure(2); semilogy(Differences)
xlabel('iterations'); ylabel('relative difference')

```

The above code required 12 iterations until the relative difference was smaller than  $10^{-5}$ . Find the graphical results in Figure 4.26(a). The final result has to be compared with Example 4–8 to verify that the beam is even weaker than before. The logarithm of the relative difference can be plotted as a function of the iteration number. The result in Figure 4.26(b) shows a straight line and this is consistent with a linear convergence<sup>11</sup>.

◇

## 4.7.2 Newton's Method

To illustrate the use of Newton's method we use a single example, the problem of the bending beam in Section 1.4.2 on page 20.

### 4–10 Example : Bending of a beam for small angles

To bend a horizontal beam apply a small vertical force  $F_2$  at the right end point. Use equation (1.17)

$$-\alpha''(s) = \frac{F_2}{EI} \cos(\alpha(s)) \quad \text{for } 0 < s < L \quad \text{and} \quad \alpha(0) = \alpha'(L) = 0$$

shown in Section 1.4.2 (page 20). The boundary conditions  $\alpha(0) = \alpha'(L) = 0$  describe the situation of a beam clamped at the left edge and no moment is applied at the right end. For small angles  $\alpha$  use  $\cos(\alpha) \approx 1$  and find a linear problem with constant coefficients.

$$-\alpha''(s) = \frac{F_2}{EI} \quad \text{with} \quad \alpha(0) = \alpha'(L) = 0.$$

<sup>11</sup>

$$|\text{diff}| \approx \alpha_0 q^n \implies \ln |\text{diff}| \approx \ln \alpha_0 + n \cdot \ln q$$

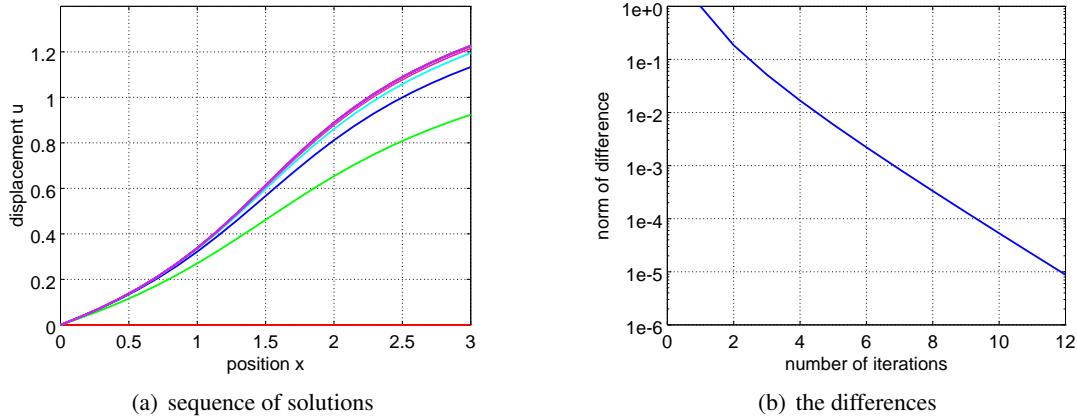


Figure 4.26: Nonlinear beam stretching problem solved by successive substitution and the logarithmic differences as function of the number of iterations

Use the discretization  $x_i = i \frac{L}{n}$  and  $\alpha_i = \alpha(x_i)$  for  $i = 1, 2, 3, \dots, n$  and the boundary conditions  $\alpha(0) = \alpha'(L) = 0$  to find a system of the form

$$\frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 1 \end{bmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_{n-2} \\ \alpha_{n-1} \\ \alpha_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-2} \\ f_{n-1} \\ \frac{f_n}{2} \end{pmatrix}$$

To take the boundary condition  $\alpha'(L) = 0$  into account proceed as in Example 4-7 on page 266. For this elementary problem use the known, the exact solution  $\alpha(s) = \frac{F_2}{EI} (Ls - \frac{1}{2}s^2)$  leading to a maximal angle of  $\alpha(L) = \frac{F_2}{EI} L^2$ . The exact solution is a polynomial of degree 2 and for this problem the approximate solution will coincide with the exact solution, i.e. no approximation error. According to Section 1.4.2 the maximal vertical deflection is given by  $y(L) = \frac{F_2}{3EI} L^3$ . With the angle function  $\alpha(s)$  the shape of the beam is given by

$$\begin{pmatrix} x(l) \\ y(l) \end{pmatrix} = \int_0^l \begin{pmatrix} \cos(\alpha(s)) \\ \sin(\alpha(s)) \end{pmatrix} ds.$$

Caused by the numerical integration by the trapezoidal rule (`cumtrapz()`) the maximal displacement will not be reproduced exactly. One error contribution is the approximate integration by the trapezoidal rule and another effect is using  $\sin \alpha$  for the integration, instead of only  $\alpha$ . The code below implements the above algorithm and verifies the result.

## BeamLinear.m

run demo

```

EI = 1.0; L = 3; F = [0 0.1];
% F = [0 2] % large force
N = 200;
%%%%%%% no modifications necessary beyond this line
h = L/N; s = (h:h:L)';
%% build and solve the tridiagonal system
A = spdiags(ones(N,1)*[-1 2 -1], [-1 0 1], N, N)/h^2; A(N,N) = 1/h^2;

```

```

g = F(2)/EI*ones(size(s)); g(N) = g(N)/2;
alpha = A\g;;
%% display the solution
x = cumtrapz([0;s],[1; cos(alpha)]); y = cumtrapz([0;s],[0; sin(alpha)]);
plot(x,y); xlabel('x'); ylabel('y');

MaximalAngles = [alpha(N), F(2)/(2*EI)*L^2]
MaximalDeflections = [max(y), F(2)/(3*EI)*L^3,trapz([0;s],[0;alpha])]
```

◊

One may try to solve the above problem using partial substitution. Start with an initial angle  $\alpha_0(s)$  and then solve iteratively the linear problem

$$-\alpha''_{k+1}(s) = \frac{F_2}{EI} \cos(\alpha_k(s)).$$

For small forces  $F_2$  this will be successful, but for larger angles the answers are of no value. One has to use Newton's method.

#### 4-11 Example : Bending of a beam, with Newton's method

Since equation (1.17) on page 20

$$-\alpha''(s) = \frac{F_2}{EI} \cos(\alpha(s)) \quad \text{for } 0 < s < L \quad \text{and} \quad \alpha(0) = \alpha'(L) = 0$$

is a nonlinear equation use Newton's method (see Section 3.1.5) to find an approximate solution. Build on the linear approximation

$$\cos(\alpha + \phi) \approx \cos(\alpha) - \sin(\alpha) \cdot \phi$$

and for a known function  $\alpha(s)$  search a solution  $\phi(s)$  of the boundary value problem

$$-\phi''(s) = \alpha''(s) + \frac{F_2}{EI} \cos(\alpha(s)) - \frac{F_2}{EI} \sin(\alpha(s)) \phi(s) \quad \text{for } 0 < s < L.$$

With the definitions  $f(s) = \alpha''(s) + \frac{F_2}{EI} \cos(\alpha(s))$  and  $b(s) = \frac{F_2}{EI} \sin(\alpha(s))$  this is a differential equation of the form

$$-\phi''(s) + b(s) \phi(s) = f(s).$$

The boundary conditions to be satisfied are  $\alpha(0) + \phi(0) = 0$  and  $\alpha'(L) + \phi'(L) = 0$ . Since  $\alpha(0) = \alpha'(L) = 0$  this translates to  $\phi(0) = \phi'(L) = 0$ . To keep the second order consistency we again the idea from Example 4-7 on page 266<sup>12</sup>. The resulting system can be written in the form

$$\frac{1}{h^2} \begin{bmatrix} 2 & -1 \\ -1 & 2 & -1 \\ & -1 & 2 & -1 \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & -1 & 2 & -1 \\ & & & -1 & 2 & -1 \end{bmatrix} \cdot \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_{n-2} \\ \phi_{n-1} \\ \phi_n \end{pmatrix} + \begin{pmatrix} b_1 \phi_1 \\ b_2 \phi_2 \\ b_3 \phi_3 \\ \vdots \\ b_{n-2} \phi_{n-2} \\ b_{n-1} \phi_{n-1} \\ \frac{b_n}{2} \phi_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-2} \\ f_{n-1} \\ \frac{f_n}{2} \end{pmatrix}.$$

<sup>12</sup>

$$0 = \phi'(L) = \frac{\phi(L+h) - \phi(L-h)}{2h} + O(h^2) \implies \phi_{n+1} = \phi_{n-1}$$

$$\frac{-\phi_{n-1} + 2\phi_n - \phi_{n+1}}{h^2} + b_n \phi_n = f_n \implies \frac{-\phi_{n-1} + \phi_n}{h^2} + \frac{b_n}{2} \phi_n = \frac{f_n}{2}$$

The contributions of the form  $b_i \phi_i$  have to be integrated into the matrix and thus on the diagonal find the expressions  $\frac{2}{h^2} + b_i$ . This matrix is symmetric, but not necessarily positive definite since the values of  $b_i$  might be negative. The new solution  $\alpha_{\text{new}}(s)$  can then be computed by

$$\alpha_{\text{new}}(s) = \alpha(s) + \phi(s) \quad \text{resp.} \quad \alpha_i \rightarrow \alpha_i + \phi_i.$$

With this new approximation then start the next iteration step for Newton's method. This has to be repeated until a solution is found with the desired accuracy.

This algorithm is implemented in MATLAB/Octave and the result shown in Figure 4.27. Use the previous example as a reference problem and for very small forces  $F_2$  and the resulting angles for the two answers should be close.

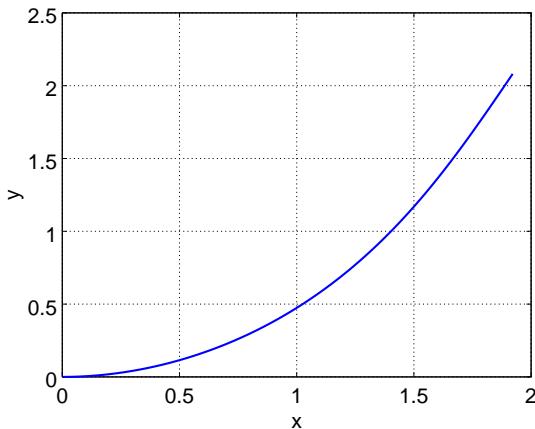


Figure 4.27: Bending of a beam, solved by Newton's method

[run demo](#)

#### BeamNewton.m

```

EI = 1.0; L = 3; F = [0 0.1]; % try values of 0.5 1.5 and 2
N = 200;
%%%%%% no modifications necessary beyond this line
h = L/N; % stepsize
s = (h:h:L)'; alpha = zeros(size(s));

%% build the tridiagonal matrix
A = spdiags(ones(N,1)*[-1 2 -1], [-1 0 1], N, N)/h^2;
A(N,N) = 1/h^2;
DiffTol = 1e-10; DiffAbs = 2*DiffTol;
while DiffAbs>DiffTol
    b = F(2)/EI*sin(alpha); b(N) = b(N)/2;
    f = F(2)/EI*cos(alpha); f(N) = f(N)/2;
    phi = (A+spdiags(b, 0, N, N)) \ (-A*alpha+f);
    alpha = alpha+phi;
    disp(sprintf('maximal angle = %7.4f, difference = %7.4e',
        max(abs(alpha)), max(abs(phi))));
end%while
x = cumtrapz([0;s], [0; cos(alpha)]); y = cumtrapz([0;s], [0; sin(alpha)]);
plot(x,y); xlabel('x'); ylabel('y'); grid on

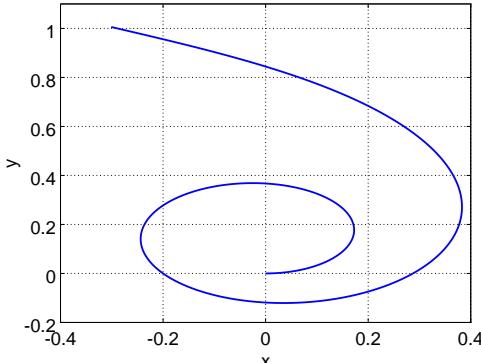
```

The values of the differences in the above iterative algorithm are given by

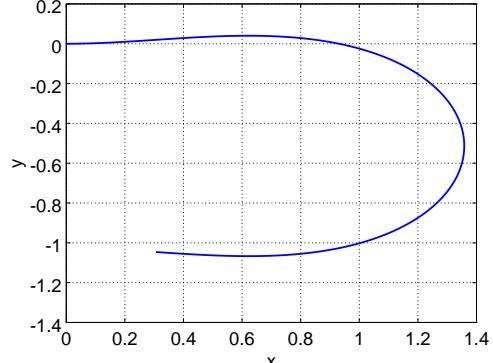
$$4.5 \cdot 10^{-1}, \quad 2.9 \cdot 10^{-2}, \quad 1.1 \cdot 10^{-4}, \quad 1.4 \cdot 10^{-9} \quad \text{and} \quad 3.1 \cdot 10^{-15}$$

and verify that the number of stable digits is doubled at each step, after an initial search for the solution. This is consistent with the quadratic convergence of Newton's method.  $\diamond$

When the codes in Examples 4–10 and 4–11 are used again with a larger value for the vertical force  $F_2 = 2.0$  obtain the (at first) surprising results in Figure 4.28. This obvious problem is created by the



(a) solved as a linear problem



(b) solved as a nonlinear problem

Figure 4.28: Bending of a beam with large force, solved as linear problem (a) and by Newton's method (b), using a zero initial displacement

**geometric nonlinearity** in the differential equation, i.e. the nonlinear expression  $\cos(\alpha(s))$ .

- The computations in Example 4–10 are based on the assumptions of small angles and use the approximation  $\cos \alpha \approx 1$ . For this computation this is certain to be false and thus the results are invalid. Thus the result in Figure 4.28(a) can not be correct.
- The solution with Newton's method from Example 4–11 is folding down and pulled backward. This might well be a physical solution, but not the one we were looking for. Thus the result in Figure 4.28(b) is correct, but useless. This is an illustration of the fact that nonlinear problems might have multiple solutions and we have to assure work with the desired solution. When Newton's algorithm is applied to this problem the errors will at first get considerably larger and only after a few searching steps the iteration will start to converge towards one of the possible solutions. This illustrates again that Newton is **not** a good algorithm to search a solution, but a good method to determine a known solution with good accuracy.

#### 4–12 Example : Bending of a beam, solved by a parameterized Newton's method

To solve the problem of a bending beam with a large vertical force and find the solution of a beam bent upwards use a **parameterization method**. Instead of searching immediately the solution with the desired force ( $F_2 = 2.0$ ) increase the force step by step from 0 to the desired value. Newton's method will find the solution for one given force and this solution will then be used as starting function for Newton's method for the next higher force. Find the intermediate and final results of the code below in Figure 4.29.

run demo

##### BeamParam.m

```

EI = 1.0; L = 3; N = 200; FList = 0.25:0.25:2;
%%%%%% no modifications necessary beyond this line
h = L/N; % stepsize
s = (h:h:L)';
alpha1 = zeros(size(s));

%% build the tridiagonal matrix
A = spdiags(ones(N,1)*[-1 2 -1], [-1 0 1], N, N)/h^2; A(N,N) = 1/h^2;

errTol = 1e-10;

```

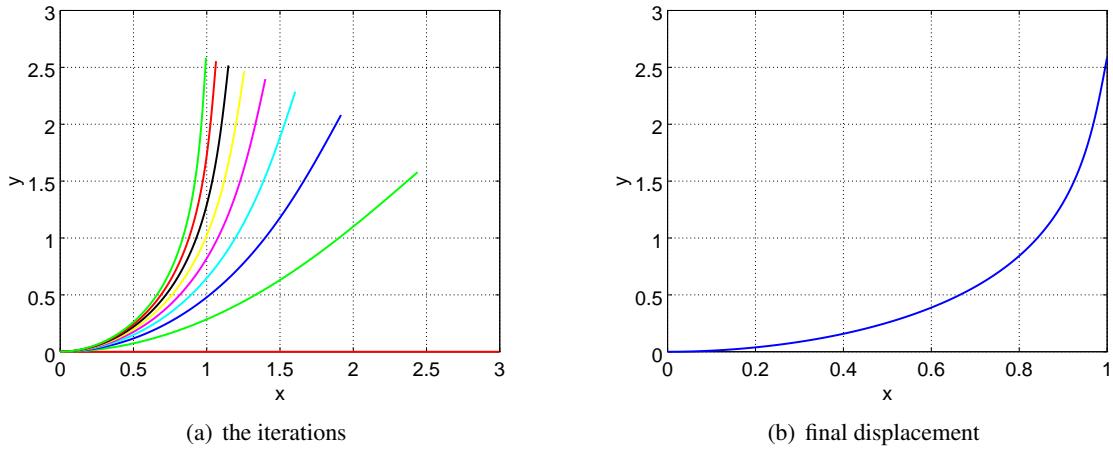


Figure 4.29: Nonlinear beam problem for a large force, solved by a parameterized Newton's method

```

yPlot = zeros(length(s)+1,1); xPlot = [0;s];

for F2 = FList
    disp(sprintf('Use force F_2 = %4.2g',F2))
    errAbs = 2*errTol;
    while errAbs>errTol;
        b = F2/EI*sin(alpha1); b(N) = b(N)/2;
        f = F2/EI*cos(alpha1); f(N) = f(N)/2;
        phi = -(A+spdiags(b,0,N,N))\ (A*alpha1-f);
        alpha1 = alpha1 + phi;
        errAbs = max(abs(phi))
    end%while
    x = cumtrapz([0;s],[0; cos(alpha1)]); y = cumtrapz([0;s],[0; sin(alpha1)]);
    xPlot = [xPlot x]; yPlot = [yPlot y];
end%for
figure(1); plot(xPlot,yPlot)
            grid on; xlabel('x'); ylabel('y'); axis equal

x = cumtrapz([0;s],[0; cos(alpha1)]); y = cumtrapz([0;s],[0; sin(alpha1)]);
figure(2); plot(x,y);
            grid on; xlabel('x'); ylabel('y');

```

◆

In the previous chapter the codes in Table 4.6 were used.

## Bibliography

- [AtkiHan09] K. Atkinson and W. Han. *Theoretical Numerical Analysis*. Number 39 in Texts in Applied Mathematics. Springer, 2009.

[Butc03] J. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, second edition, 2003.

[GoluVanLoan96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

filename	function
BeamStretch.m	code to solve Example 4–6
BeamStretchVariable.m	code to solve Example 4–8
Plate.m	code to solve the BVP in Section 4.4.2
Heat2DStatic.m	code to solve the BVP in Section 4.4.2
HeatDynamic.m	code to solve the IVP in Section 4.5.3
HeatDynamicImplicit.m	code to solve the IVP in Section 4.5.4
PlateDynamic.m	code to solve the IVP in Section 4.5.7
Wave.m	code to solve the IVP in Section 4.6.1
BeamNL.m	code to solve Example 4–9
BeamLinear.m	code to solve the bending beam problem, Example 4–10
BeamNewton.m	code to solve the bending beam problem, Example 4–11
BeamParam.m	code to solve the bending beam problem, Example 4–12

Table 4.6: Codes for chapter 4

[GoluVanLoan13] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2013.

[IsaaKell66] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. John Wiley & Sons, 1966. Republished by Dover in 1994.

[KhanKhan18] O. Khanmohamadi and E. Khanmohammadi. Four fundamental spaces of numerical analysis. *Mathematics Magazin*, 91(4):243–253, 2018.

[KnabAnge00] P. Knabner and L. Angermann. *Numerik partieller Differentialgleichungen*. Springer Verlag, Berlin, 2000.

[Smit84] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, Oxford, third edition, 1986.

[Thom95] J. W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*, volume 22 of *Texts in Applied Mathematics*. Springer Verlag, New York, 1995.

[Wlok82] J. Wloka. *Partielle Differentialgleichungen*. Teubner, Stuttgart, 1982.

# Chapter 5

# Calculus of Variations, Elasticity and Tensors

## 5.1 Prerequisites and Goals

After having worked through this chapter

- you should be familiar with the basic idea of the calculus of variations.
- you should be able to apply the Euler–Lagrange equations to problems in one or multiple variables.
- you should understand the notations of stress and strain and Hooke’s law.
- should be able to formulate elasticity equations as minimization problems.
- recognize plane strain and plane stress situations.
- should know about the stress and strain invariants.

Show  
PowerMet

In this chapter we assume that you are familiar with the following:

- Basic calculus for one and multiple variables.
- Taylor approximations of order one, i.e. linear approximations.
- Some classical mechanics.

## 5.2 Calculus of Variations

### 5.2.1 The Euler Lagrange Equation

The goal of this section is to “discover” the Euler–Lagrange equation and then apply it to some sample problems. If the functions  $u(x)$  and  $f(x, u, u')$  are given then the definite integral

$$F(u) = \int_a^b f(x, u(x), u'(x)) dx$$

is well defined. The main idea is now to examine the behavior of  $F(u)$  for different functions  $u$ . Search for functions  $u(x)$ , which will minimize the value of  $F(u)$ . Technically speaking try to minimize the functional  $F$ .

start with  
shortest  
connection  
problem

The basic idea is quite simple: if a function  $F(u)$  has a minimum, then its derivative has to vanish. But there is a major technical problem: the variable  $u$  is actually a function  $u(x)$ , i.e. we have to minimize a functional. The techniques of the calculus of variations<sup>1</sup> deal with this type of problem.

**5–1 Definition :** If a mapping  $F$  is defined for a set of functions  $X$  and returns a number as a result then  $F$  is called a **functional** on the function space  $X$ .

Thus a functional is nothing but a function with a set of functions as domain of definition. It might help to compare typical functions and functionals.

	domain of definition	range
function	interval $[a, b]$	numbers $\mathbb{R}$
functional	continuous functions defined on $[a, b]$ , i.e. $C([a, b], \mathbb{R})$	numbers $\mathbb{R}$

Here are a few examples of functionals.

$$\begin{array}{ll} F(u) = \int_0^\pi a(x) u^2(x) dx & \text{defined on } C([0, 1], \mathbb{R}) \text{ with } u(0) = u(\pi) = 1 \\ F(u) = \int_0^1 \sqrt{1 + u'(x)^2} dx & \text{defined on } C^1([0, 1], \mathbb{R}) \text{ with } u(0) = 1, u(1) = \pi \\ F(u) = \int_0^1 (u'(x)^2 - 1)^2 + u(x)^2 dx & \text{defined on } C^1([0, 1], \mathbb{R}) \text{ with } u(0) = u(1) = 0 \\ F(u) = \int_0^1 a(x) u''(x)^2 dx & \text{defined on } C^2([0, 1], \mathbb{R}) \end{array}$$

The principal goal of the calculus of variations is to find extrema of functionals.

The fundamental lemma below is related to Hilbert space methods. As very simple example examine vectors in  $\mathbb{R}^n$  to visualize the basic idea. A vector  $\vec{u} \in \mathbb{R}^n$  equals  $\vec{0}$  if and only if the scalar product with all vectors  $\vec{\phi} \in \mathbb{R}^n$  vanishes, i.e.

$$\langle \vec{u}, \vec{\phi} \rangle = 0 \quad \text{for all } \vec{\phi} \in \mathbb{R}^n \iff \vec{u} = \vec{0}.$$

Similarly a continuous function vanishes on an interval  $[a, b]$  iff its product with all functions  $\phi$  integrates to 0, i.e. the role of the scalar product is taken over by an integration.

$$\langle \vec{f}, \vec{g} \rangle \rightarrow \int_a^b f(x) \cdot g(x) dx$$

**5–2 Lemma :** If  $u(x)$  is a continuous function on the interval  $a \leq x \leq b$  and

$$\int_a^b u(x) \cdot \phi(x) dx = 0$$

for all differentiable functions  $\phi$  with  $\phi(a) = \phi(b) = 0$  then

$$u(x) = 0 \quad \text{for all } a \leq x \leq b.$$

◇

---

<sup>1</sup>The calculus of variations was initiated with the problem of a brachistochrone by Johann Bernoulli's (1667–1748) in 1696, see [HenrWann17]. Contributions by Jakob Bernoulli (1654–1705) and Leonhard Euler (1707–1783) followed. Joseph-Louis Lagrange (1736–1813) contributed extensively to the method.

**Proof :** Proceed by contradiction. Assume that  $u(x_0) > 0$  for some  $x_0$  between  $a$  and  $b$ . Since the function  $u(x)$  is continuous we know that  $u(x) > 0$  on a (possibly small) interval  $x_1 < x < x_2$ . Now choose

$$\phi(x) = \begin{cases} 0 & \text{for } x \leq x_1 \\ (x - x_1)^2(x - x_2)^2 & \text{for } x_1 \leq x \leq x_2 \\ 0 & \text{for } x_2 \leq x \end{cases} .$$

Then conclude  $u(x)\phi(x) \geq 0$  for all  $a \leq x \leq b$  and  $u(x_0)\phi(x_0) > 0$  and thus

$$\int_a^b u(x) \cdot \phi(x) dx = \int_{x_1}^{x_2} u(x) \cdot \phi(x) dx > 0.$$

This is a contradiction to the condition in the Lemma. Thus  $u(x) = 0$  for  $a < x < b$ . As the function  $u$  is continuous conclude that  $u(a) = u(b) = 0$ .  $\square$

With a few more mathematical ideas the above result can be improved to obtain an important result for the calculus of variations.

### 5–3 Theorem : *The fundamental lemma of calculus of variations*

- If  $u(x)$  is a continuous function for  $a \leq x \leq b$  and

$$\int_a^b u(x) \cdot \phi(x) dx = 0$$

for all infinitely often differentiable functions  $\phi(x)$  with  $\phi(a) = \phi(b) = 0$  then

$$u(x) = 0 \quad \text{for all } a \leq x \leq b$$

- If  $u(x)$  is a differentiable function for  $a \leq x \leq b$  and

$$\int_a^b u(x) \cdot \phi'(x) dx = 0$$

for all infinitely often differentiable functions  $\phi(x)$  then

$$u'(x) = 0 \quad \text{for all } a \leq x \leq b \quad \text{and} \quad u(a) \cdot \phi(a) = u(b) \cdot \phi(b) = 0$$

$\diamond$

**Proof :** Find the proof of the first statement in any good book on functional analysis or calculus of variations. For the second part use integration by parts, i.e.

$$0 = \int_a^b u(x) \cdot \phi'(x) dx = u(b) \cdot \phi(b) - u(a) \cdot \phi(a) - \int_a^b u'(x) \cdot \phi(x) dx.$$

Considering all test function  $\phi(x)$  with  $\phi(a) = \phi(b) = 0$  leads to the condition  $u'(x) = 0$ . We are free to choose test functions with arbitrary values at the end points  $a$  and  $b$ , thus conclude that  $u(a) \cdot \phi(a) = u(b) \cdot \phi(b) = 0$ .  $\square$

For a given function  $f(x, u, u')$  search for a function  $u(x)$  such that the functional

$$F(u) = \int_a^b f(x, u(x), u'(x)) dx$$

has a critical value for the function  $u$ . For sake of a readability use the notations<sup>2</sup>

$$\begin{aligned} f_x(x, u, u') &= \frac{\partial}{\partial x} f(x, u, u') , \\ f_u(x, u, u') &= \frac{\partial}{\partial u} f(x, u, u') , \\ f_{u'}(x, u, u') &= \frac{\partial}{\partial u'} f(x, u, u') . \end{aligned}$$

If the functional  $F$  attains its minimal value at the function  $u(x)$  conclude that

$$g(\varepsilon) = F(u + \varepsilon \phi) \geq F(u) \quad \text{for all } \varepsilon \in \mathbb{R} \text{ and arbitrary functions } \phi(x) .$$

Thus the scalar function  $g(\varepsilon)$  has a minimum at  $\varepsilon = 0$  and thus the derivative should vanish, i.e.

$$\frac{d g(0)}{d\varepsilon} = \frac{d}{d\varepsilon} F(u + \varepsilon \phi) \Big|_{\varepsilon=0} = 0 \quad \text{for all functions } \phi .$$

To find the equations to be satisfied by the solution  $u(x)$  use linear approximations. For small values of  $\Delta u$  and  $\Delta u'$  use a Taylor approximation to conclude

$$\begin{aligned} f(x, u + \Delta u, u' + \Delta u') &\approx f(x, u, u') + \frac{\partial f(x, u, u')}{\partial u} \Delta u + \frac{\partial f(x, u, u')}{\partial u'} \Delta u' \\ &= f(x, u, u') + f_u(x, u, u') \Delta u + f_{u'}(x, u, u') \Delta u' \\ f(x, u(x) + \varepsilon \phi(x), u'(x) + \varepsilon \phi'(x)) &= f(x, u(x), u'(x)) + \varepsilon f_u(x, u(x), u'(x)) \phi(x) + \\ &\quad + \varepsilon f_{u'}(x, u(x), u'(x)) \phi'(x) + O(\varepsilon^2) . \end{aligned}$$

Now examine the functional in question

$$\begin{aligned} g(0) = F(u) &= \int_a^b f(x, u(x), u'(x)) dx \\ g(\varepsilon) &= F(u + \varepsilon \phi) = \int_a^b f(x, u(x) + \varepsilon \phi(x), u'(x) + \varepsilon \phi'(x)) dx \\ &\approx \int_a^b f(x, u(x), u'(x)) + \varepsilon f_u(x, u(x), u'(x)) \phi(x) + \varepsilon f_{u'}(x, u(x), u'(x)) \phi'(x) dx \\ &= F(u) + \varepsilon \int_a^b f_u(x, u(x), u'(x)) \phi(x) + f_{u'}(x, u(x), u'(x)) \phi'(x) dx \end{aligned}$$

---

<sup>2</sup>Observe the difference between total derivatives and partial derivatives, as illustrated by the example.

$$\begin{aligned} f(x, u, u') &= x^2 (u')^2 + \cos(x) \cdot u \\ \frac{\partial}{\partial x} f(x, u(x), u'(x)) &= 2x (u'(x))^2 - \sin(x) \cdot u(x) \\ \frac{\partial}{\partial u} f(x, u(x), u'(x)) &= \cos(x) \\ \frac{\partial}{\partial u'} f(x, u(x), u'(x)) &= 2x^2 u'(x) \\ \frac{d}{dx} f(x, u(x), u'(x)) &= 2x (u'(x))^2 + 2x^2 u'(x) u''(x) - \sin(x) \cdot u(x) + \cos(x) \cdot u'(x) \end{aligned}$$

or

$$\frac{d}{d\varepsilon} F(u + \varepsilon\phi) \Big|_{\varepsilon=0} = \int_a^b f_u(x, u(x), u'(x)) \phi(x) + f_{u'}(x, u(x), u'(x)) \phi'(x) dx.$$

This integral has to vanish for all function  $\phi(x)$  and using the Fundamental Lemma 5-3, leading to a necessary condition. An integration by parts leads to

$$\begin{aligned} 0 &= \int_a^b f_u(x, u(x), u'(x)) \phi(x) + f_{u'}(x, u(x), u'(x)) \phi'(x) dx \\ &= f_{u'}(x, u(x), u'(x)) \phi(x) \Big|_{x=a}^b \\ &\quad + \int_a^b \left( f_u(x, u(x), u'(x)) - \frac{d}{dx} f_{u'}(x, u(x), u'(x)) \right) \phi(x) dx. \end{aligned}$$

Since this expression has to vanish for all function  $\phi(x)$  the necessary conditions are

$$\int_a^b f(x, u(x), u'(x)) dx \text{ extremal} \implies \begin{cases} \frac{d}{dx} f_{u'}(x, u(x), u'(x)) = f_u(x, u(x)) \\ f_{u'}(a, u(a), u'(a)) \cdot \phi(a) = 0 \\ f_{u'}(b, u(b), u'(b)) \cdot \phi(b) = 0 \end{cases}.$$

The first condition is the **Euler–Lagrange** equation, the second and third condition are **boundary conditions**. If the value  $u(a)$  is given and we are not free to choose, then we need  $\phi(a) = 0$  and the first boundary condition is automatically satisfied. If we are free to choose  $u(a)$ , then  $\phi(a)$  need not vanish and we have the condition

$$f_{u'}(a, u(a), u'(a)) = 0.$$

This is a **natural boundary condition**. A similar argument applies at the other endpoint  $x = b$ .

Now we have the central result for the calculus of variations in one variable.

#### 5-4 Theorem : Euler–Lagrange equation

If a smooth function  $u(x)$  leads to a critical value of the functional

$$F(u) = \int_a^b f(x, u(x), u'(x)) dx$$

the differential equation

$$\frac{d}{dx} f_{u'}(x, u(x), u'(x)) = f_u(x, u(x), u'(x)) \quad (5.1)$$

has to be satisfied for  $a < x < b$ . This is usually a second order differential equation.

- If it is a critical value amongst all functions with prescribed boundary values  $u(a)$  and  $u(b)$ , use these to solve the differential equation.
- If you are free to choose the values of  $u(a)$  and/or  $u(b)$ , then the **natural boundary conditions**

$$f_{u'}(a, u(a), u'(a)) = 0 \quad \text{and/or} \quad f_{u'}(b, u(b), u'(b)) = 0$$

can be used.

◇

If the functional is modified by boundary contributions

$$F(u) = \int_a^b f(x, u(x), u'(x)) dx - K_1 u(b) - \frac{K_2}{2} u^2(b)$$

the Euler–Lagrange is not modified, but the natural boundary condition at  $x = b$  is given by

$$f_{u'}(b, u(b), u'(b)) = K_1 + K_2 u(b).$$

The verification follows exactly the above procedure and is left as an exercise.

Ex. 5.2

### 5–5 Example : Shortest connection between two points

Given two points  $(a, y_1)$  and  $(b, y_2)$  in a plane determine the function  $y = u(x)$ , such that its graph connects the two points and the length of this curve is a short as possible. The length  $L$  of the curve is given by the integral

$$L(u) = \int_a^b \sqrt{1 + (u'(x))^2} dx.$$

Using the notations of the above results determine the partial derivatives

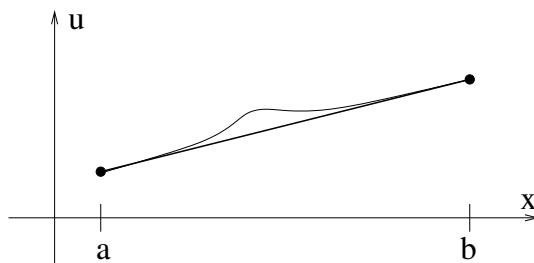


Figure 5.1: Shortest connection between two points

$$\begin{aligned} f(x, u, u') &= \sqrt{1 + (u')^2} \\ f_x(x, u, u') &= f_u(x, u, u') = 0 \\ f_{u'}(x, u, u') &= \frac{u'}{\sqrt{1 + (u')^2}} \end{aligned}$$

and the Euler–Lagrange equation (5.1) applied to this example leads to

$$\frac{d}{dx} \frac{u'(x)}{\sqrt{1 + (u'(x))^2}} = 0.$$

The derivative of a function being zero everywhere implies that the function has to be constant and thus

$$\frac{u'(x)}{\sqrt{1 + (u'(x))^2}} = c$$

and conclude that  $u'(x)$  has to be constant. Thus the optimal solution is a straight line. This should not be a surprise.

If we are free to choose the point of contact along the vertical line at  $x = b$  we may use  $\phi(b) \neq 0$  and thus find the natural boundary condition

$$\frac{u'(b)}{\sqrt{1 + (u'(b))^2}} = 0.$$

This implies  $u'(b) = 0$  and thus  $u'(x) = 0$  for all  $x$ . This leads to a horizontal line, which is obviously the shortest connection from the given height at  $x = a$  to the vertical line at  $x = b$ .  $\diamond$

### 5–6 Example : String under transversal load

The vertical deformation of a horizontal string can be given by a function  $y = u(x)$  for  $0 \leq x \leq L$ . Due to this deformation  $u(x)$  of the string it will be lengthened by

$$\Delta L = \int_0^L \sqrt{1 + (u'(x))^2} dx - L$$

and due to the constant horizontal force  $T$  this requires an energy of  $T \Delta L$ . The applied external, vertical force density  $f(x)$  can be modeled by a corresponding potential energy density of  $-f(x) u(x)$ . Now the total energy is given by

$$E(u) = \int_0^L F(u(x), u'(x)) dx = \int_0^L T \sqrt{1 + (u'(x))^2} - f(x) \cdot u(x) dx.$$

For this functional find the partial derivatives

$$\begin{aligned} F(u, u') &= T \sqrt{1 + (u')^2} - f \cdot u \\ F_u(u, u') &= -f \\ F_{u'}(u, u') &= T \frac{u'}{\sqrt{1 + (u')^2}} \end{aligned}$$

and thus the Euler–Lagrange equation (5.1) applied to this example lead to

$$-\frac{d}{dx} \left( \frac{T}{\sqrt{1 + (u'(x))^2}} u'(x) \right) = f(x).$$

Since the string is attached at both ends, supplement this differential equation with the boundary conditions  $u(0) = u(L) = 0$ .

If we know a priori that the slope  $u'(x)$  along the string is small, use a linear approximation<sup>3</sup>

$$\sqrt{1 + (u'(x))^2} \approx 1 + \frac{1}{2} (u'(x))^2.$$

With this the change of length  $\Delta L$  of the string is given by

$$\Delta L = \int_0^L \sqrt{1 + (u'(x))^2} dx - L \approx \int_0^L \frac{1}{2} (u'(x))^2 dx.$$

Now the total energy can be written in the form

$$E(u) = T \Delta L + E_{pot} = \int_0^L \frac{1}{2} T (u'(x))^2 - f(x) \cdot u(x) dx$$

and the resulting Euler–Lagrange equation is given by

$$-T u''(x) = f(x).$$

◇

---

<sup>3</sup>Use the Taylor approximation  $\sqrt{1 + z} \approx 1 + \frac{1}{2} z$ .

### 5–7 Example : Bending of a beam

In Section 1.4 find the description of a bending beam. If  $\alpha(s)$  is the angle at a position  $(x(s), y(s))$  construct the curve from the function  $\alpha(s)$  with an integral

$$\vec{x}(l) = \begin{pmatrix} x(l) \\ y(l) \end{pmatrix} = \int_0^l \begin{pmatrix} \cos(\alpha(s)) \\ \sin(\alpha(s)) \end{pmatrix} ds \quad \text{for } 0 \leq l \leq L.$$

The elastic energy stored in the bent beam is given by

$$U_{elast} = \int_0^L \frac{1}{2} EI(\alpha'(s))^2 ds.$$

An external force  $\vec{F} = (F_1, F_2)$  at the right end point  $\vec{x}(L)$  has to be determined by

$$\vec{F} = -\operatorname{grad} U_{pot} = -\left( \frac{\partial U_{pot}}{\partial x}, \frac{\partial U_{pot}}{\partial y} \right)$$

and are thus given by the potential energy  $U_{pot}(x, y)$

$$U_{pot}(x(L), y(L)) = -F_1 x(L) - F_2 y(L) = -F_1 \int_0^L \cos(\alpha(s)) ds - F_2 \int_0^L \sin(\alpha(s)) ds.$$

The total energy  $U_{tot}$  as a functional of the angle function  $\alpha(s)$  is given by

$$\begin{aligned} U_{tot}(\alpha) &= U_{elast}(\alpha) + U_{pot}(\vec{x}(L)) = \int_0^L \frac{1}{2} EI(\alpha'(s))^2 ds + U_{pot}(x(L), y(L)) \\ &= \int_0^L \frac{1}{2} EI(\alpha'(s))^2 - F_1 \cos(\alpha(s)) - F_2 \sin(\alpha(s)) ds. \end{aligned}$$

The physical situation is characterized as a minimum of this functional, using Bernoulli's principle. For the expression to be integrated find the partial derivatives

$$\begin{aligned} F(\alpha, \alpha') &= \frac{1}{2} EI(\alpha')^2 - F_1 \cos(\alpha) - F_2 \sin(\alpha) \\ F_\alpha(\alpha, \alpha') &= F_1 \sin(\alpha) - F_2 \cos(\alpha) \\ F_{\alpha'}(\alpha, \alpha') &= EI\alpha' \end{aligned}$$

and the Euler–Lagrange equation for this problem is given by

$$(EI \alpha'(s))' = F_1 \sin(\alpha(s)) - F_2 \cos(\alpha(s)).$$

This is identical equation (1.16) (page 19). For a beam clamped at the left end and no moments at the right end point find the boundary conditions  $\alpha(0) = \alpha'(L) = 0$ . The second is a natural boundary condition, as  $u(L)$  is not prescribed. This is a nonlinear, second order boundary value problem.  $\diamond$

### 5.2.2 Quadratic Functionals and Second Order Linear Boundary Value Problems

If for given functions  $a(x)$ ,  $b(x)$  and  $g(x)$  the functional

$$F(u) = \int_{x_0}^{x_1} \frac{1}{2} a(x) (u'(x))^2 + \frac{1}{2} b(x) u(x)^2 + g(x) \cdot u(x) dx \quad (5.2)$$

has to be minimised, then obtain the Euler–Lagrange equation

$$\begin{aligned} \frac{d}{dx} f_{u'} &= f_u \\ \frac{d}{dx} \left( a(x) \frac{d u(x)}{dx} \right) &= b(x) u(x) + g(x). \end{aligned}$$

This is a linear, second order differential equation which has to be supplemented with appropriate boundary conditions. If the value at one of the endpoints is given then this is called a **Dirichlet boundary condition**. If we are free to choose the value at the boundary then this is called a **Neumann boundary condition**. Theorem 5–4 implies that the second situation leads to a **natural boundary condition**

$$a(x) \frac{d^2 u}{dx^2} = 0 \quad \text{for } x = x_0 \quad \text{or} \quad x = x_1.$$

If we wish to consider a non-homogeneous boundary conditions

$$a(x) \frac{d u}{dx} = r(x) \quad \text{for } x = x_0 \quad \text{or} \quad x = x_1$$

then the functional has to be supplemented by

Ex. 5.2

$$F(u) + r(x_0) u(x_0) - r(x_1) u(x_1).$$

Thus the above approach shows that many second order differential equation correspond to extremal points for a properly chosen functional. Many physical, mechanical and electrical problems lead to this type of equation as can be seen in Table 5.1 (Source: [OttoPete92, p. 63]).

### 5.2.3 The Divergence Theorem and its Consequences

The well known fundamental theorem of calculus for functions of one variable

$$\int_a^b f'(x) dx = -f(a) + f(b)$$

can be extended to functions of multiple variables. If  $\Omega \subset \mathbb{R}^n$  is a "nice" domain with boundary  $\partial\Omega$  and outer unit normal vector  $\vec{n}$  then the corresponding result is called the **divergence theorem**. For domains  $\Omega \subset \mathbb{R}^2$  use

$$\iint_{\Omega} \operatorname{div} \vec{v} dA = \iint_{\Omega} \frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y} dA = \oint_{\partial\Omega} \langle \vec{v}, \vec{n} \rangle ds$$

and if  $\Omega \subset \mathbb{R}^3$  then the notation is

$$\iiint_{\Omega} \operatorname{div} \vec{v} dV = \oint_{\partial\Omega} \langle \vec{v}, \vec{n} \rangle dA$$

where  $dV$  is the standard volume element and  $dA$  the surface element. The usual rule to differentiate products of two functions leads to

$$\begin{aligned} \nabla \cdot (f \vec{v}) &= (\nabla f) \cdot \vec{v} + f (\nabla \cdot \vec{v}) \\ \operatorname{div}(f \vec{v}) &= \langle \operatorname{grad} f, \vec{v} \rangle + f (\operatorname{div} \vec{v}). \end{aligned}$$

Using the divergence theorem conclude

$$\begin{aligned} \iint_{\Omega} f (\operatorname{div} \vec{v}) dA &= \iint_{\Omega} \operatorname{div}(f \vec{v}) - \langle \operatorname{grad} f, \vec{v} \rangle dA \\ &= \oint_{\partial\Omega} f \langle \vec{v}, \vec{n} \rangle ds - \iint_{\Omega} \langle \operatorname{grad} f, \vec{v} \rangle dA. \end{aligned}$$

This formula is referred to as **Green–Gauss theorem** or **Green's identity** and is similar to integration by parts for functions of one variable

$$\int_a^b f \cdot g' dx = -f(a) \cdot g(a) + f(b) \cdot g(b) - \int_a^b f' \cdot g dx$$

differential equation	problem description	constitutive law		
$\frac{d}{dx} (A k \frac{dT}{dx}) + Q = 0$	one-dimensional heat flow	$T =$ temperature $A =$ area $k =$ thermal conductivity $Q =$ heat supply	$q = -k \frac{dT}{dx}$	Fourier's law
$\frac{d}{dx} (A E \frac{du}{dx}) + b = 0$	axially loaded elastic bar	$u =$ displacement $A =$ area $E =$ Young's modulus $b =$ axial loading		Hooke's law $\sigma = E \frac{du}{dx}$ $\sigma =$ stress
$\frac{d}{dx} (S \frac{dw}{dx}) + p = 0$	transversely loaded flexible string	$w =$ deflection $S =$ string force $p =$ lateral loading		
$\frac{d}{dx} (A D \frac{dc}{dx}) + Q = 0$	one dimensional diffusion	$c =$ concentration $A =$ area $D =$ Diffusion coefficient $Q =$ external supply		Fick's law $q = -D \frac{dc}{dx}$ $q =$ flux
$\frac{d}{dx} (A \gamma \frac{dV}{dx}) + Q = 0$	one dimensional electric current	$V =$ voltage $A =$ area $\gamma =$ electric conductivity $Q =$ charge supply		Ohm's law $q = -\gamma \frac{dV}{dx}$ $q =$ charge flux
$\frac{d}{dx} \left( A \frac{D^2}{32\mu} \frac{dp}{dx} \right) + Q = 0$	laminar flow in a pipe (Poisseuille flow)	$p =$ pressure $A =$ area $D =$ diameter $\mu =$ viscosity $Q =$ fluid supply		$q = \frac{D^2}{32\mu} \frac{dp}{dx}$ $q =$ volume flux

Table 5.1: Examples of second order differential equations

or if spelled out for the derivative  $g'$

$$\int_a^b f \cdot g'' dx = -f(a) \cdot g'(a) + f(b) \cdot g'(b) - \int_a^b f' \cdot g' dx.$$

For finite elements and calculus of variations the divergence theorem is most often used in the form below.

$$\begin{aligned} \iint_{\Omega} f (\operatorname{div} \operatorname{grad} g) dA &= \oint_{\partial\Omega} f \langle \operatorname{grad} g, \vec{n} \rangle ds - \iint_{\Omega} \langle \operatorname{grad} f, \operatorname{grad} g \rangle dA \\ \iint_{\Omega} f \Delta g dA &= \oint_{\partial\Omega} f \langle \nabla g, \vec{n} \rangle ds - \iint_{\Omega} \langle \nabla f, \nabla g \rangle dA \end{aligned}$$

### 5.2.4 Quadratic Functionals and Second Order Boundary Value Problems in 2 Dimensions

We want to modify the functional in (5.2) to a 2 dimensional setting and examine the boundary value problem resulting from the Euler–Lagrange equations.

Consider a domain  $\Omega \subset \mathbb{R}^2$  with a boundary  $\partial\Omega = \Gamma_1 \cup \Gamma_2$  consisting of two disjoint parts  $\Gamma_1$  and  $\Gamma_2$ . For given functions  $a, b, f, g_1$  and  $g$  (all depending on  $x$  and  $y$ ) we search a yet unknown function  $u$ , such that the functional

$$F(u) = \iint_{\Omega} \frac{1}{2} a \langle \nabla u, \nabla u \rangle + \frac{1}{2} b u^2 + f \cdot u dA - \int_{\Gamma_2} g_2 u ds \quad (5.3)$$

is minimal amongst all functions  $u$  which satisfy

$$u(x, y) = g_1(x, y) \quad \text{for } (x, y) \in \Gamma_1.$$

To find the necessary equations assume that  $\phi$  and  $\nabla\phi$  are small and use the approximations

$$\begin{aligned} (u + \phi)^2 &= u^2 + 2u\phi + \phi^2 \approx u^2 + 2u\phi \\ \langle \nabla(u + \phi), \nabla(u + \phi) \rangle &= \langle \nabla u, \nabla u \rangle + 2 \langle \nabla u, \nabla \phi \rangle + \langle \nabla \phi, \nabla \phi \rangle \\ &\approx \langle \nabla u, \nabla u \rangle + 2 \langle \nabla u, \nabla \phi \rangle \end{aligned}$$

and Green's identity to conclude

$$\begin{aligned} F(u + \phi) - F(u) &\approx \iint_{\Omega} a \langle \nabla u, \nabla \phi \rangle + b u \phi + f \cdot \phi dA - \int_{\Gamma_2} g_2 \phi ds \\ &= \iint_{\Omega} (-\nabla(a \nabla u) + b u + f) \cdot \phi dA + \int_{\Gamma} a \langle \vec{n}, \nabla u \rangle \phi ds - \int_{\Gamma_2} g_2 \phi ds \\ &= \iint_{\Omega} (-\nabla(a \nabla u) + b u + f) \cdot \phi dA + \int_{\Gamma_2} (a \langle \vec{n}, \nabla u \rangle - g_2) \phi ds. \end{aligned}$$

The test-function  $\phi$  is arbitrary, but has to vanish on  $\Gamma_1$ . If the functional  $F$  is minimal for the function  $u$  then the above integral has to vanish for all test-functions  $\phi$ . First consider only test-functions that vanish on  $\Gamma_2$  too and use the fundamental lemma (a modification of Theorem 5–3) to conclude that the expression in the parenthesis in the integral over the domain  $\Omega$  has to be zero. Then use arbitrary test functions  $\phi$  to conclude that the expression in the integral over  $\Gamma_2$  has to vanish too. Thus the resulting linear partial differential equation with boundary conditions is given by

$$\begin{aligned} \nabla \cdot (a \nabla u) - b u &= f && \text{for } (x, y) \in \Omega \\ u &= g_1 && \text{for } (x, y) \in \Gamma_1 \\ a \langle \nabla u, \vec{n} \rangle &= g_2 && \text{for } (x, y) \in \Gamma_2 \end{aligned} \quad (5.4)$$

The functions  $a, b, f$  and  $g_i$  are known functions and we have to determine the solution  $u$ , all depending on the independent variables  $(x, y) \in \Omega$ . The vector  $\vec{n}$  is the **outer unit normal vector**. The expression

$$\langle \nabla u, \vec{n} \rangle = n_1 \frac{\partial u}{\partial x} + n_2 \frac{\partial u}{\partial y} = \frac{\partial u}{\partial \vec{n}}$$

determines the directional derivative of the function  $u$  in the direction of the outer normal  $\vec{n}$ .

A list of typical applications of elliptic equations of second order is shown in Table 5.2, see [Redd84]. The static heat conduction problem in Section 1.1.5 (page 11) is another example. A description of the ground water flow problem is given in [OttoPete92]. This table clearly illustrates the importance of the above type of problem.

### 5–8 Example : Deformation of a membrane

When a small vertical displacement of a thin membrane is given by  $z = u(x, y)$ , where  $(x, y) \in \Omega \subset \mathbb{R}^2$ , we can compute the elastic energy stored in the membrane by

$$E_{\text{elast}} = \iint_{\Omega} \frac{\tau}{2} \|\nabla u\|^2 dA = \iint_{\Omega} \frac{\tau}{2} (u_x^2 + u_y^2) dA,$$

where we assume that  $u = 0$  on the boundary  $\partial\Omega$ . Now apply a vertical force to the membrane given by a force density function  $f(x, y)$  (units: N/m<sup>2</sup>). To formulate this we introduce a potential energy

$$E_{\text{pot}} = - \iint_{\Omega} f \cdot u dA.$$

Based on the previous results minimizing the total energy

$$E = E_{\text{elast}} + E_{\text{pot}} = \iint_{\Omega} \frac{\tau}{2} (u_x^2 + u_y^2) - f u dA$$

leads to the Euler–Lagrange equation

$$\tau \Delta u = \nabla \cdot (\tau \nabla u) = -f.$$

This corresponds to the model problem with equation (1.11) on page 15.  $\diamond$

### 5–9 Example : Vibration of a membrane

Use Newtons law in the above problem for the vertical acceleration  $\ddot{u}$ . If an external device is applying a force  $f$  on the membrane in a static situation, then the stretched membrane is applying the opposite force to the external device. If there is no external device this force leads to an acceleration. Thus conclude

$$f = -\rho \ddot{u}$$

where  $\rho$  is the mass density (units kg/m<sup>2</sup>). The resulting equation is then

$$\rho \ddot{u} - \nabla \cdot (\tau \nabla u) = 0$$

which corresponds to equation (1.10) on page 15.

The corresponding eigenvalue equation (1.12) leads to harmonic oscillations as solutions.  $\diamond$

Field of application	Primary variable	Material constant	Source variable	Secondary variables
General situation	$u$	$a$	$f$	$\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}$
Heat transfer	Temperature $T$	Conductivity $k$	Heat source $Q$	Heat flow density $\vec{q}$ $\vec{q} = -k \nabla T$
Diffusion	Concentration $c$	Diffusion coefficient	external supply $Q$	flux $\vec{q} = -D \nabla c$
Electrostatics	Scalar potential $\Phi$	Dielectric constant $\varepsilon$	Charge density $\rho$	Electric flux density $D$
Magnetostatics	Magnetic potential $\Phi$	Permeability $\nu$		Magnetic flux density $B$
Transverse deflection of elastic membrane	Transverse deflection $u$	Tension of membrane $T$	Transversely distributed load	Normal force $q$
Stress $\tau$				
Torsion of a bar	Warping function $\phi$	1	0	$\tau_{xz} = \frac{E \alpha}{2(1+\nu)} (-y + \frac{\partial \phi}{\partial x})$ $\tau_{yz} = \frac{E \alpha}{2(1+\nu)} (x + \frac{\partial \phi}{\partial y})$
Irrational flow of an ideal fluid	Stream function $\Psi$	Density $\rho$	Mass production $\sigma$ (usually zero)	Velocity $(u, v)^T$
				$\frac{d\Psi}{dx} = -u$ $\frac{d\Psi}{dy} = v$
	Velocity potential $\Phi$			$\frac{d\Phi}{dx} = u$ $\frac{d\Phi}{dy} = v$
Ground-water flow	Piezometric head $\Phi$	Permeability $K$	Recharge $Q$ (or pumping $-Q$ )	seepage $q = K \frac{\partial \Phi}{\partial n}$ velocities

Table 5.2: Some examples of Poisson's equation  $-\nabla(a \nabla u) = f$

### 5.2.5 Nonlinear Problems and Euler–Lagrange Equations for Systems

not in class

If a functional  $J(u)$  is given in the form

$$J(u) = \iint_{\Omega} F(u, \nabla u) dA$$

apply a small perturbation  $\phi$  to the argument  $u$  and use a linear approximation to find

$$\begin{aligned} J(u + \varphi) &= \iint_{\Omega} F(u + \varphi, \nabla u + \nabla \varphi) dA \\ &= \iint_{\Omega} F(u, \nabla u) + F_u(u, \nabla u) \varphi + F_{\nabla u}(u, \nabla u) \nabla \varphi dA + O(|\varphi|^2, \|\nabla \varphi\|^2) \end{aligned}$$

with the notations

$$F_{\nabla u} = \left( \frac{\partial F}{\partial \frac{\partial u}{\partial x}}, \frac{\partial F}{\partial \frac{\partial u}{\partial y}} \right) \quad \text{and} \quad F_{\nabla u} \nabla \varphi = \frac{\partial F}{\partial \frac{\partial u}{\partial x}} \frac{\partial \phi}{\partial x} + \frac{\partial F}{\partial \frac{\partial u}{\partial y}} \frac{\partial \phi}{\partial y}.$$

If the minimum of the functional  $J(u)$  is attained at the function  $u$  conclude that for all permissible test functions  $\varphi$  find the necessary condition

$$\begin{aligned} 0 &= \iint_{\Omega} F_{\nabla u}(u, \nabla u) \nabla \varphi + F_u(u, \nabla u) \varphi dA \\ &= \oint_{\partial\Omega} \varphi F_{\nabla u}(u, \nabla u) \cdot \vec{n} ds + \iint_{\Omega} -\nabla(F_{\nabla u}(u, \nabla u)) \varphi + F_u(u, \nabla u) \varphi dA. \end{aligned}$$

Since this expression vanishes for all test functions  $\varphi$  use the fundamental lemma to find the Euler–Lagrange equation

$$-\nabla(F_{\nabla u}(u, \nabla u)) + F_u(u, \nabla u) = 0 \quad \text{in } \Omega \quad (5.5)$$

and on the sections of the boundary where the test function  $\varphi$  does not vanish the natural boundary condition

$$\langle F_{\nabla u}(u, \nabla u), \vec{n} \rangle = 0.$$

For the example 5–8 of a deformed membrane the above leads to

$$\begin{aligned} F(u, \nabla u) &= \frac{\tau}{2} (u_x^2 + u_y^2) - f \cdot u \\ F_u(u, \nabla u) &= -f \\ F_{\nabla u}(u, \nabla u) &= \left( \frac{\partial}{\partial u_x} F, \frac{\partial}{\partial u_y} F \right) = \tau (u_x, u_y) = \tau \nabla u \end{aligned}$$

and the Euler–Lagrange equation is given by

$$-\nabla(F_{\nabla u}(u, \nabla u)) + F_u(u, \nabla u) = -\nabla \cdot (\tau \nabla u) - f = 0.$$

### 5–10 Example : Plateau problem

If a surface in  $\mathbb{R}^3$  is described by a function  $z = u(x, y)$  where  $(x, y) \in \Omega \subset \mathbb{R}^2$ , then the total area is given by the functional

$$J(u) = \iint_{\Omega} \sqrt{1 + \|\nabla u\|^2} dA = \iint_{\Omega} \sqrt{1 + u_x^2 + u_y^2} dA.$$

If the goal is to minimize the total area use calculus of variations. To generate the Euler–Lagrange equations use the Taylor approximation  $\sqrt{u+z} \approx \sqrt{u} + z/(2\sqrt{u})$  and conclude

$$\begin{aligned} J(u + \varphi) &= \iint_{\Omega} \sqrt{1 + (u_x + \varphi_x)^2 + (u_y + \varphi_y)^2} dA \\ &\approx \iint_{\Omega} \sqrt{1 + u_x^2 + 2u_x\varphi_x + u_y^2 + 2u_y\varphi_y} dA \\ &\approx J(u) + \iint_{\Omega} \frac{1}{\sqrt{1 + u_x^2 + u_y^2}} (u_x\varphi_x + u_y\varphi_y) dA. \end{aligned}$$

If the functional  $J$  attains its minimum at the function  $u$  conclude that for all test functions  $\varphi$

$$\begin{aligned} 0 &= + \iint_{\Omega} \frac{1}{\sqrt{1 + u_x^2 + u_y^2}} \nabla u \cdot \nabla \varphi dA \\ &= \oint_{\partial\Omega} \frac{1}{\sqrt{1 + u_x^2 + u_y^2}} \vec{n} \nabla u \cdot \varphi ds - \iint_{\Omega} \nabla \left( \frac{1}{\sqrt{1 + u_x^2 + u_y^2}} \nabla u \right) \varphi dA. \end{aligned}$$

If the values  $z = u(x, y)$  are known on the boundary  $\partial\Omega \subset \mathbb{R}^2$  then the test functions  $\varphi$  vanish on the boundary and the Euler–Lagrange equation are given by

$$-\nabla \left( \frac{1}{\sqrt{1 + u_x^2 + u_y^2}} \nabla u \right) = 0.$$

This is a nonlinear second order differential equation. The identical result may be generated by

$$\begin{aligned} F(u, \nabla u) &= \sqrt{1 + u_x^2 + u_y^2} \\ F_u(u, \nabla u) &= 0 \\ F_{\nabla u}(u, \nabla u) &= \frac{1}{\sqrt{1 + u_x^2 + u_y^2}} (u_x, u_y) \end{aligned}$$

and then use the Euler–Lagrange equation in the form (5.5).  $\diamond$

The above idea can also be applied to functionals depending on more than one independent variable. Examine a domain  $\Omega \subset \mathbb{R}^2$  with a boundary  $\partial\Omega$  consisting of two parts. On  $\Gamma_1$  the values of  $u_1$  and  $u_2$  are given and on  $\Gamma_2$  these values are free. Then minimize a functional of the form

$$J(u_1, u_2) = \iint_{\Omega} F(u_1, u_2, \nabla u_1, \nabla u_2) dA - \int_{\Gamma_2} u_1 \cdot g_1 + u_2 \cdot g_2 ds. \quad (5.6)$$

Apply small perturbations  $\varphi_1$  and  $\varphi_2$ , use a linear approximation to find<sup>4</sup>

$$J = J(u_1 + \varphi_1, u_2 + \varphi_2)$$

<sup>4</sup>Use the notation  $F_{\nabla u} = (\frac{\partial}{\partial u_x} F, \frac{\partial}{\partial u_y} F)^T$ .

$$\begin{aligned}
&= \iint_{\Omega} F(u_1 + \varphi_1, \nabla u_1 + \nabla \varphi_1, u_2 + \varphi_2, \nabla u_2 + \nabla \varphi_2) dA + \\
&\quad - \int_{\Gamma_2} (u_1 + \varphi_1) \cdot g_1 + (u_2 + \varphi_2) \cdot g_2 ds \\
&= J(u_1, u_2) + \iint_{\Omega} F_{u_1}(\dots) \varphi_1 + \langle F_{\nabla u_1}(\dots), \nabla \varphi_1 \rangle + F_{u_2}(\dots) \varphi_2 + \langle F_{\nabla u_2}(\dots), \nabla \varphi_2 \rangle dA - \\
&\quad - \int_{\Gamma_2} \varphi_1 \cdot g_1 + \varphi_2 \cdot g_2 ds + O(|\varphi_1|^2, \|\nabla \varphi_1\|^2, |\varphi_2|^2, \|\nabla \varphi_2\|^2).
\end{aligned}$$

Find necessary conditions if the minimum of the functional  $J(u_1, u_2)$  is attained at  $(u_1, u_2)$ . Conclude that for all permissible test functions  $\varphi_1$  and  $\varphi_2$  vanishing on the boundary  $\partial\Omega$  the following integral has to equal zero.

$$\begin{aligned}
0 &= \iint_{\Omega} F_{u_1}(\dots) \varphi_1 + \langle F_{\nabla u_1}(\dots), \nabla \varphi_1 \rangle + F_{u_2}(\dots) \varphi_2 + \langle F_{\nabla u_2}(\dots), \nabla \varphi_2 \rangle dA \\
&= \iint_{\Omega} (F_{u_2}(\dots) - \operatorname{div}(F_{\nabla u_1}(\dots))) \varphi_1 + (F_{u_2}(\dots) - \operatorname{div}(F_{\nabla u_2}(\dots))) \varphi_2 dA
\end{aligned}$$

Since this expression to vanish for all test functions  $\varphi_1$  and  $\varphi_2$  use the fundamental lemma to arrive at a system of Euler–Lagrange equations.

**5–11 Result :** A minimizer  $u_1$  and  $u_2$  of the functional  $J(u_1, u_2)$  of the form (5.6) solves the system of Euler–Lagrange equations

$$\operatorname{div}(F_{\nabla u_1}(u_1, u_2, \nabla u_1, \nabla u_2)) = F_{u_1}(u_1, u_2, \nabla u_1, \nabla u_2) \quad (5.7)$$

$$\operatorname{div}(F_{\nabla u_2}(u_1, u_2, \nabla u_1, \nabla u_2)) = F_{u_2}(u_1, u_2, \nabla u_1, \nabla u_2). \quad (5.8)$$

◇

Using these equations with test functions  $\varphi_1$  and  $\varphi_2$  vanishing on  $\Gamma_1$ , but not necessarily on  $\Gamma_2$  find

$$0 = \int_{\Gamma_2} \varphi_1 F_{\nabla u_1}(\dots) \cdot \vec{n} - \varphi_1 \cdot g_1 + \varphi_2 F_{\nabla u_2}(\dots) \cdot \vec{n} - \varphi_2 \cdot g_2 ds.$$

This leads to the natural boundary conditions on  $\Gamma_2$ .

$$\begin{aligned}
\langle F_{\nabla u_1}(u_1, u_2, \nabla u_1, \nabla u_2), \vec{n} \rangle &= g_1 \\
\langle F_{\nabla u_2}(u_1, u_2, \nabla u_1, \nabla u_2), \vec{n} \rangle &= g_2
\end{aligned}$$

This method can be used to derive the differential equations governing elastic deformations of solids, see Section 5.9.4 for the PDE governing the plane stress situation.

## 5.2.6 Hamilton's principle of Least Action

The notes in this section are mostly taken from [VarFEM]. The starting point was the classical book by Weinberger [Wein74, p. 72].

Examine a system of particles subject to given geometric constraints and otherwise influenced by forces which are functions of the positions of the particles only. In addition we require the system to be conservative, i.e. the forces can be written as the gradient of a **potential energy**  $V$  of the system. We denote the  $n$  **degrees of freedom** of the system with  $\vec{q} = (q_1, q_2, \dots, q_n)^T$ . The **kinetic energy**  $T$  of the system is the extension of the basic formula  $E = \frac{1}{2} m v^2$ . With those form the **Lagrange function**  $L$  of the system by

$$L(\vec{q}, \dot{\vec{q}}) = T(\vec{q}, \dot{\vec{q}}) - V(\vec{q}).$$

The fundamental principle of Hamilton can now be formulated:

The actual motion of a system with the above Lagrangian  $L$  is such as to render the (Hamilton's) integral

$$I = \int_{t_1}^{t_2} (T - V) dt = \int_{t_1}^{t_2} L(\vec{q}, \dot{\vec{q}}) dt$$

an extremum with respect to all twice differentiable functions  $\vec{q}(t)$ . Here  $t_1$  and  $t_2$  are arbitrary times.

This is a situation where we (usually) have multiple dependent variables  $q_i$  and thus the Euler–Lagrange equations imply

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} = \frac{\partial L}{\partial q_i} \quad \text{for } i = 1, 2, \dots, n.$$

These differential equations apply to many mechanical setups, as the following examples will illustrate.

### 5–12 Example : Newton's law as consequence of Hamilton's principle

Suppose a force  $F$  is given by a potential energy  $V(x)$  by  $F(x) = -\frac{\partial}{\partial x} V(x)$  and a particle with mass  $m$  is moving with velocity  $v(t) = \frac{d}{dt} x(t)$ , then the Lagrange function is given by

$$L(x, \dot{x}) = T(x, \dot{x}) - V(x) = \frac{1}{2} m (\dot{x})^2 - V(x).$$

Thus determine

$$\begin{aligned} \frac{d}{dt} \frac{\partial}{\partial \dot{x}} L(x, \dot{x}) &= \frac{d}{dt} \left( \frac{1}{2} m 2 \dot{x} \right) = m \ddot{x} \\ \frac{\partial}{\partial x} L(x, \dot{x}) &= -\frac{\partial}{\partial x} V(x) = +F(x) \end{aligned}$$

and the resulting Euler–Lagrange equation is given by

$$m \ddot{x}(t) = -\frac{\partial}{\partial x} V(x) = F(x),$$

i.e. the well known form of Newton's equation is a consequence of Hamilton's principle.  $\diamond$

### 5–13 Example : Single pendulum

For a single pendulum of length  $l$  find the kinetic and potential energy

$$T(\varphi, \dot{\varphi}) = \frac{1}{2} m l^2 (\dot{\varphi})^2 \quad \text{and} \quad V(\varphi) = -m l g \cos \varphi$$

and thus the Lagrange function

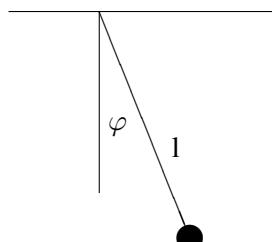
$$L = T - V = \frac{1}{2} m l^2 (\dot{\varphi})^2 + m l g \cos \varphi.$$

The only degree of freedom is  $q_1 = \varphi$  and the functional to be minimised is

$$\int_a^b \frac{1}{2} m l^2 (\dot{\varphi})^2 + m l g \cos \varphi dt.$$

The Euler–Lagrange equation leads to

$$\begin{aligned} \frac{d}{dt} \frac{\partial L}{\partial \dot{\varphi}} &= \frac{\partial L}{\partial \varphi} \\ \frac{d}{dt} m l^2 \dot{\varphi} &= -m l g \sin \varphi \\ \ddot{\varphi} &= -\frac{g}{l} \sin \varphi. \end{aligned}$$



This is the well known differential equation describing a pendulum. One can certainly derive the same equation using Newton's law.  $\diamond$

### 5–14 Example : A double pendulum

The calculations for this problem are shown in many books on classical mechanics, e.g. [Gree77]. A double pendulum consists of two particles with mass  $m$  suspended by mass-less rods of length  $l$ . Assuming that all takes place in a vertical plane we have two degrees of freedom: the two angles  $\varphi$  and  $\theta$ . The potential energy is not too hard to find as

$$V(\varphi, \theta) = -m l g (2 \cos \varphi + \cos \theta)$$

The velocity of the upper particle is  $v_1 = l \dot{\varphi}$ .

To find the kinetic energy we need the velocity of the lower mass. The velocity vector is equal to the vector sum of the velocity of the upper mass and the velocity of the lower particle relative to the upper mass.

$$\begin{aligned} \vec{v}_1 &= \text{length } l \dot{\varphi} \text{ and angle } \varphi \pm \frac{\pi}{2} \\ \vec{v}_2 &= \text{length } l \dot{\theta} \text{ and angle } \theta \pm \frac{\pi}{2} \\ \varphi - \theta &= \text{angle between } \vec{v}_1 \text{ and } \vec{v}_2 \end{aligned}$$

Since the two vectors differ in direction by an angle of  $\varphi - \theta$  we can use the law of cosine to find the absolute velocity as<sup>5</sup>

$$\text{speed of second mass} = l \sqrt{\dot{\varphi}^2 + \dot{\theta}^2 + 2 \dot{\varphi} \dot{\theta} \cos(\varphi - \theta)}.$$

Thus the total kinetic energy is

$$T(\varphi, \theta, \dot{\varphi}, \dot{\theta}) = \frac{m l^2}{2} \left( 2 \dot{\varphi}^2 + \dot{\theta}^2 + 2 \dot{\varphi} \dot{\theta} \cos(\varphi - \theta) \right)$$

and the Lagrange function is

$$L = T - V = \frac{m l^2}{2} \left( 2 \dot{\varphi}^2 + \dot{\theta}^2 + 2 \dot{\varphi} \dot{\theta} \cos(\varphi - \theta) \right) + m l g (2 \cos \varphi + \cos \theta).$$

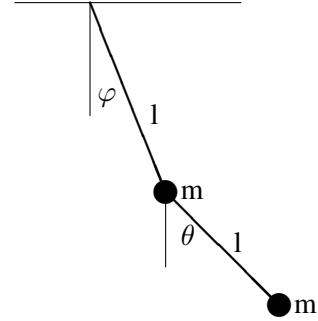
The Euler–Lagrange equation for the free variable  $\varphi$  is obtained by

$$\begin{aligned} \frac{\partial L}{\partial \dot{\varphi}} &= m l^2 \left( 2 \dot{\varphi} + \dot{\theta} \cos(\varphi - \theta) \right) \\ \frac{d}{dt} \frac{\partial L}{\partial \dot{\varphi}} &= m l^2 \left( 2 \ddot{\varphi} + \ddot{\theta} \cos(\varphi - \theta) - \dot{\theta} (\dot{\varphi} - \dot{\theta}) \sin(\varphi - \theta) \right) \\ \frac{\partial L}{\partial \varphi} &= -m l^2 \dot{\varphi} \dot{\theta} \sin(\varphi - \theta) - m l g 2 \sin \varphi \end{aligned}$$

which, upon substitution into the Euler–Lagrange equation, yields

$$m l^2 \left( 2 \ddot{\varphi} + \ddot{\theta} \cos(\varphi - \theta) + \dot{\theta}^2 \sin(\varphi - \theta) \right) = -m l g 2 \sin \varphi.$$

<sup>5</sup>Another approach is to use cartesian coordinates  $x(\varphi, \theta) = l(\sin \varphi + \sin \theta)$ ,  $y(\varphi, \theta) = -l(\cos \varphi + \cos \theta)$  and a few calculations.



In a similar fashion the Euler–Lagrange equation for the variable  $\theta$  is obtained by

$$\begin{aligned}\frac{\partial L}{\partial \dot{\theta}} &= m l^2 (\dot{\theta} + \dot{\varphi} \cos(\varphi - \theta)) \\ \frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}} &= m l^2 (\ddot{\theta} + \ddot{\varphi} \cos(\varphi - \theta) - \dot{\varphi} (\dot{\varphi} - \dot{\theta}) \sin(\varphi - \theta)) \\ \frac{\partial L}{\partial \theta} &= +m l^2 \dot{\varphi} \dot{\theta} \sin(\varphi - \theta) - m l g \sin \theta\end{aligned}$$

leading to

$$m l^2 (\ddot{\theta} + \ddot{\varphi} \cos(\varphi - \theta) - \dot{\varphi}^2 \sin(\varphi - \theta)) = -m l g \sin \theta.$$

Those two equations can be divided by  $m l^2$  and then lead to a system of ordinary differential equations of order 2.

$$\begin{aligned}2 \ddot{\varphi} + \ddot{\theta} \cos(\varphi - \theta) + \dot{\theta}^2 \sin(\varphi - \theta) &= -\frac{g}{l} 2 \sin \varphi \\ \ddot{\theta} + \ddot{\varphi} \cos(\varphi - \theta) - \dot{\varphi}^2 \sin(\varphi - \theta) &= -\frac{g}{l} \sin \theta\end{aligned}$$

By isolating the second order terms on the left arrive at

$$\begin{bmatrix} 2 & \cos(\varphi - \theta) \\ \cos(\varphi - \theta) & 1 \end{bmatrix} \begin{pmatrix} \ddot{\varphi} \\ \ddot{\theta} \end{pmatrix} = \sin(\varphi - \theta) \begin{pmatrix} -\dot{\theta}^2 \\ \dot{\varphi}^2 \end{pmatrix} - \frac{g}{l} \begin{pmatrix} 2 \sin \varphi \\ \sin \theta \end{pmatrix}.$$

The matrix on the left hand side is invertible and thus this differential equation can reliably be solved by numerical procedures, see Section 3.4 starting on page 182.

Assuming that all angles and velocities are small one can use the approximations  $\cos(\varphi - \theta) \approx 1$  and  $\sin x \approx x$  to obtain the **linearized** system of differential equations

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{pmatrix} \ddot{\varphi} \\ \ddot{\theta} \end{pmatrix} = -\frac{g}{l} \begin{pmatrix} 2 \varphi \\ \theta \end{pmatrix}.$$

Solving for the second order derivatives obtain

$$\begin{pmatrix} \ddot{\varphi} \\ \ddot{\theta} \end{pmatrix} = -\frac{g}{l} \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} \begin{pmatrix} 2 \varphi \\ \theta \end{pmatrix} = -\frac{g}{l} \begin{bmatrix} 2 & -1 \\ -2 & 2 \end{bmatrix} \begin{pmatrix} \varphi \\ \theta \end{pmatrix}.$$

This linear system of equations could be solved explicitly, using eigenvalues and eigenvectors, see Section 3.4. For the above matrix find

$$\lambda_1 = 2 - \sqrt{2} \approx 0.59 \quad , \quad \vec{v}_1 = \begin{pmatrix} 1 \\ \sqrt{2} \end{pmatrix} \quad \text{and} \quad \lambda_2 = 2 + \sqrt{2} \approx 3.41 \quad , \quad \vec{v}_2 = \begin{pmatrix} 1 \\ -\sqrt{2} \end{pmatrix}$$

Thus the solutions for small angles there of the form

$$\begin{pmatrix} \phi(t) \\ \theta(t) \end{pmatrix} \approx A_1 \cos(\sqrt{0.59 \frac{g}{l}} t + \delta_1) \begin{pmatrix} 1 \\ \sqrt{2} \end{pmatrix} + A_2 \cos(\sqrt{3.41 \frac{g}{l}} t + \delta_2) \begin{pmatrix} 1 \\ -\sqrt{2} \end{pmatrix}$$

Thus there is one type of in-phase solution with a small frequency and a high frequency solution where the two angles are out of phase.  $\diamond$

Run  
DoublePen

not in class

### 5-15 Example : A pendulum with moving support

A chariot of mass  $m_1$  with an attached pendulum of length  $l$  and mass  $m_2$  is moving freely. The situation is shown in Figure 5.2. In this example the independent variable is time  $t$  and the two general coordinates (degrees of freedom) are  $x$  and  $\theta$ , i.e.

$$\vec{u} = \begin{pmatrix} x \\ \theta \end{pmatrix}.$$

The position and velocity of the pedulum are

$$\vec{p} = \begin{pmatrix} x + l \sin(\theta) \\ -l \cos(\theta) \end{pmatrix}, \quad \vec{v} = \begin{pmatrix} \dot{x} + l \dot{\theta} \cos(\theta) \\ l \dot{\theta} \sin(\theta) \end{pmatrix}$$

and potential and kinetic energy are given by

$$\begin{aligned} V(x, \theta) &= -m_2 l g \cos \theta - F x \\ T(x, \theta, \dot{x}, \dot{\theta}) &= \frac{m_1}{2} \dot{x}^2 + \frac{m_2}{2} ((\dot{x} + l \cos \theta \dot{\theta})^2 + (l \sin \theta \dot{\theta})^2) \\ &= \frac{m_1}{2} \dot{x}^2 + \frac{m_2}{2} (\dot{x}^2 + 2l \dot{x} \cos \theta \dot{\theta} + l^2 \dot{\theta}^2). \end{aligned}$$

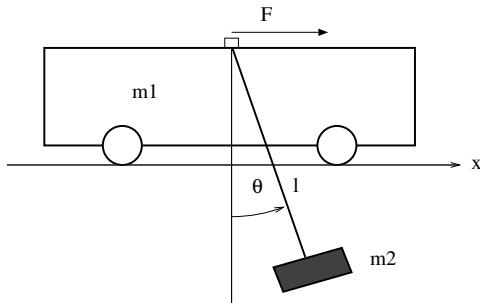


Figure 5.2: Pendulum with moving support

First examine the case  $F = 0$  and for the Lagrange function  $L = T - V$  derive two Euler–Lagrange equations. The first equation deals with the dependence on the function  $x(t)$  and its derivative  $\dot{x}(t)$ .

$$\begin{aligned} \frac{d}{dt} L_{\dot{x}}(x, \theta, \dot{x}, \dot{\theta}) &= L_x(x, \theta, \dot{x}, \dot{\theta}) \\ \frac{d}{dt} ((m_1 + m_2) \dot{x} + m_2 l \cos \theta \dot{\theta}) &= 0. \end{aligned}$$

From this conclude that the momentum in  $x$  direction is conserved. The second equation deals with the dependence on the function  $\theta(t)$ .

$$\begin{aligned} \frac{d}{dt} L_{\dot{\theta}}(x, \theta, \dot{x}, \dot{\theta}) &= L_{\theta}(x, \theta, \dot{x}, \dot{\theta}) \\ m_2 \frac{d}{dt} (l \dot{x} \cos \theta + l^2 \dot{\theta}) &= -m_2 l \dot{x} \dot{\theta} \sin \theta - m_2 l g \sin \theta \\ \frac{d}{dt} (\dot{x} \cos \theta + l \dot{\theta}) &= -\dot{x} \dot{\theta} \sin \theta - g \sin \theta \\ \ddot{x} \cos \theta - \dot{x} \dot{\theta} \sin \theta + l \ddot{\theta} &= -\dot{x} \dot{\theta} \sin \theta - g \sin \theta \\ \ddot{x} \cos \theta + l \ddot{\theta} &= -g \sin \theta. \end{aligned}$$

This is a second order differential equation for the functions  $x(t)$  and  $\theta(t)$ . The two equations can be combined, leading to the system

$$\begin{aligned}(m_1 + m_2) \ddot{x} + m_2 l \cos \theta \ddot{\theta} &= m_2 l \sin \theta (\dot{\theta})^2 \\ \ddot{x} \cos \theta + l \ddot{\theta} &= -g \sin \theta.\end{aligned}$$

With the help of a matrix the system can be solved for the highest occurring derivatives. A straight forward computation shows that the determinant of the matrix does not vanish and thus we can always find the inverse matrix.

$$\frac{d^2}{dt^2} \begin{pmatrix} x \\ \theta \end{pmatrix} = \begin{bmatrix} m_1 + m_2 & m_2 l \cos \theta \\ \cos \theta & l \end{bmatrix}^{-1} \begin{pmatrix} m_2 l \sin \theta (\dot{\theta})^2 \\ -g \sin \theta \end{pmatrix}$$

This is a convenient form to generate numerical solutions for the problem at hand.

The above model does not consider friction. Now we want to include some friction on the moving chariot. This is not elementary, as the potential  $V$  can not depend on the velocity  $\dot{x}$ , but there is a trick to be used.

1. Introduce a constant force  $F$  applied to the chariot. This is done by modifying the potential  $V$  accordingly.
2. Find the corresponding differential equations.
3. Set the force  $F = -\alpha \dot{x}$

To take the additional force  $F$  into account modify the potential energy

$$V(x, \theta) = -m_2 l g \cos \theta - x \cdot F.$$

The Euler–Lagrange equation for the variable  $\theta$  will not be affected by this change, but the equation for  $x$  turns out to be

$$\begin{aligned}\frac{d}{dt} \left( (m_1 + m_2) \dot{x} + m_2 l \cos \theta \dot{\theta} \right) &= F \\ (m_1 + m_2) \ddot{x} + m_2 l \cos \theta \ddot{\theta} &= m_2 l \sin \theta (\dot{\theta})^2 + F\end{aligned}$$

and the full system is now given by

$$\frac{d^2}{dt^2} \begin{pmatrix} x \\ \theta \end{pmatrix} = \begin{bmatrix} m_1 + m_2 & m_2 l \cos \theta \\ \cos \theta & l \end{bmatrix}^{-1} \begin{pmatrix} m_2 l \sin \theta (\dot{\theta})^2 + F \\ -g \sin \theta \end{pmatrix}.$$

Now replace  $F$  by  $-\alpha \dot{x}$  and one obtains

$$\frac{d^2}{dt^2} \begin{pmatrix} x \\ \theta \end{pmatrix} = \begin{bmatrix} m_1 + m_2 & m_2 l \cos \theta \\ \cos \theta & l \end{bmatrix}^{-1} \begin{pmatrix} m_2 l \sin \theta (\dot{\theta})^2 - \alpha \dot{x} \\ -g \sin \theta \end{pmatrix}.$$

Below find the complete code to solve this example and the resulting Figure 5.3.

Run demo

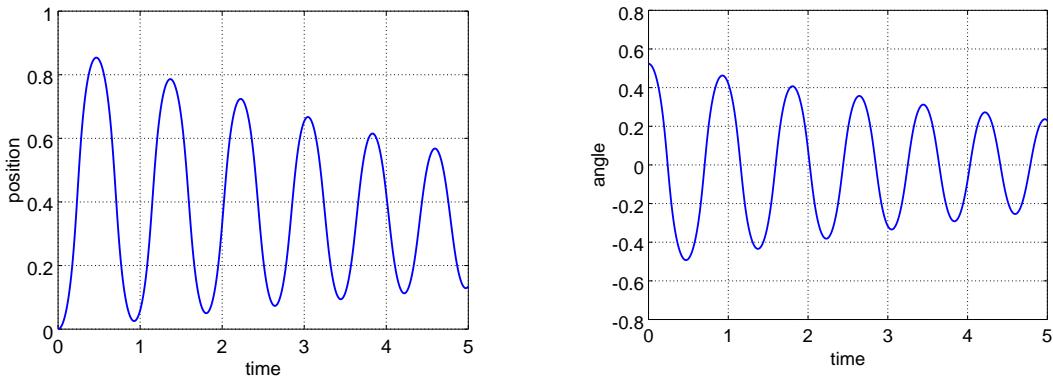


Figure 5.3: Numerical solution for a pendulum with moving support

**MovingPendulum.m**

```

function MovingPendulum()
t = 0:0.01:5; Y0 = [0;pi/6;0;0];

function dy = MovPend(y)
l = 1; m1 = 1; m2 = 8; g = 9.81; al = 0.5;
ddy = [m1+m2, m2*l*cos(y(2));
        cos(y(2)), 1]\[m2*l*sin(y(2))*y(4)^2-al*y(3);-g*sin(y(2))];
dy = [y(3);y(4);ddy];
end%function

[t,Y] = ode45(@(t,y)MovPend(y),t,Y0);
figure(2); plot(t,Y(:,1));
xlabel('time'); ylabel('position'); grid on;
figure(3); plot(t,Y(:,2));
xlabel('time'); ylabel('angle'); grid on
end%function

```



### 5.3 Basic Elasticity, Description of Stress and Strain

In the following sections we give a **very basic** introduction to the description of elastic deformations. Find this and more information in the vast literature. One good introductory book is [Bowe10] and the corresponding web page [solidmechanics.org](http://solidmechanics.org). The book [GhabPeckWu17] gives a broad presentation of the basics for computational analysis. It goes clearly beyond the scope of these introductory notes.

In Figure 5.4 the experimental laws of elasticity are illustrated. A beam of original length  $l$  with cross-sectional area  $A = w \cdot h$  is stretched by applying a force  $F$ . Many experiments lead to the following two basic laws of elasticity.

- **Hooke's law**

$$\frac{\Delta l}{l} = \frac{1}{E} \frac{F}{A},$$

where the material constant  $E$  is called **modulus of elasticity** (Young's modulus).

- **Poisson's law**

$$\frac{\Delta h}{h} = \frac{\Delta w}{w} = -\nu \frac{\Delta l}{l},$$

where the material constant  $\nu$  is called **Poisson's ratio**.

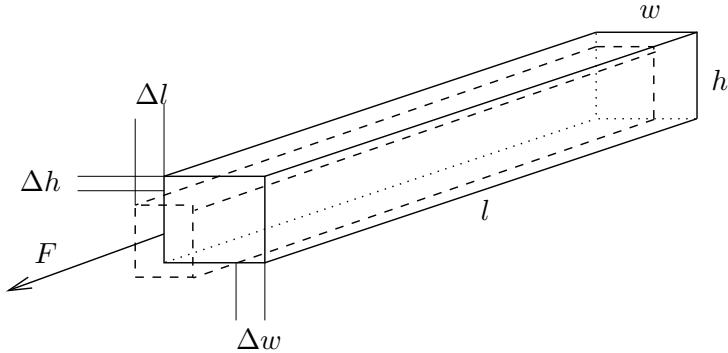


Figure 5.4: Definition of the modulus of elasticity  $E$  and the Poisson number  $\nu$

In this section the above basic mechanical facts are formulated for general situations, i.e. introduce the basic equations of elasticity. The procedure is as follows:

- Description of deformed solid: strain.
- Description of the forces within deformed solid: stress.
- Introduction to scalars, vectors, tensors.
- State the connection between deformations and forces, leading to Hooke's law.

An elastic solid can be fixed at its left edge and be pulled on at the right edge by a force. Figure 5.5 shows a simple situation. The original shape (dotted line) will change into a deformed state (full line). The goal is to give a mathematical description of the deformation of the solid (strain) and the forces that will occur in the solid (stress). A point originally at position  $\vec{x}$  in the solid is moved to its new position  $\vec{x} + \vec{u}(\vec{x})$ , i.e. displaced by  $\vec{u}(\vec{x})$ .

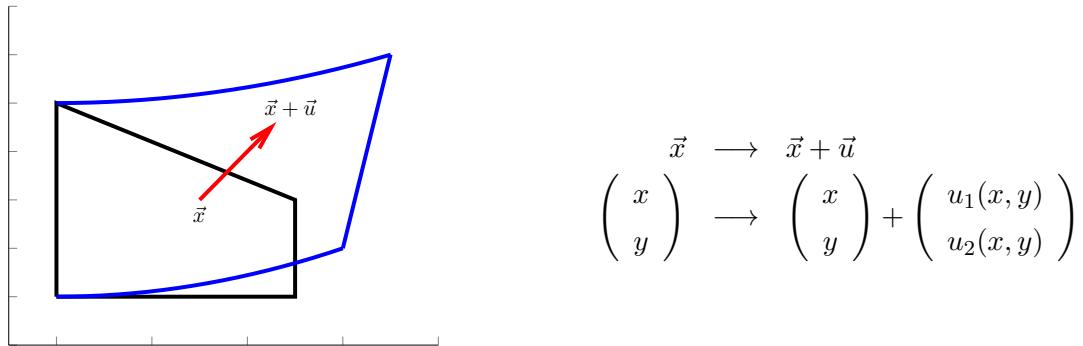


Figure 5.5: Deformation of an elastic solid

The notation will be used to give a formula for the elastic energy stored in the deformed solid. Based on this information we will construct a finite element solution to the problem. For a given force we search the displacement vector field  $\vec{u}(\vec{x})$ .

In order to simplify the treatment enormously we assume that the displacement of the structure are very small compared to the dimensions of the solid.

### 5.3.1 Description of Strain

The **strain** will give us a mathematical description of the deformation of a given object. It is a purely geometrical description and at this point not related to elasticity. Find a readable description for the construction of strain, using displacement, in [ChouPaga67, §2, p.34ff]. First examine the strain for the deformation of an object in a plane. Later we will extend the construction to objects in space.

Of a large object to be deformed and moved in a plane (see Figure 5.5) examine a small rectangle of width  $\Delta x$  and height  $\Delta y$  and its behavior under the deformation. The original rectangle  $ABCD$  and the deformed shape  $A'B'C'D'$  are shown in Figure 5.6.

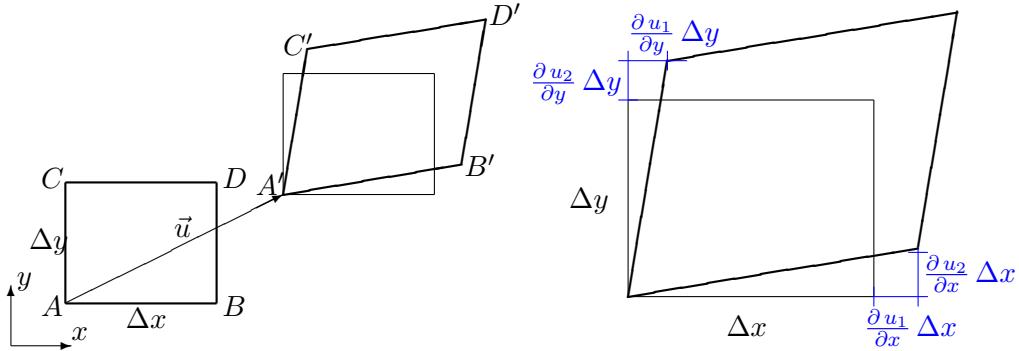


Figure 5.6: Definition of strain: rectangle before and after deformation

Since  $\Delta x$  and  $\Delta y$  are assumed to be very small, the deformation is very close to an affine deformation, i.e. a linear deformation and a translation. Since the deformations are small we also know that the deformed rectangle has to be almost horizontal, thus Figure 5.6 is correct. A straightforward Taylor approximation leads to expressions for the positions of the four corners of the rectangle.

$$\begin{aligned} A &= \begin{pmatrix} x \\ y \end{pmatrix} \quad \rightarrow \quad A' = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} \\ B &= \begin{pmatrix} x + \Delta x \\ y \end{pmatrix} \quad \rightarrow \quad B' = \begin{pmatrix} x + \Delta x \\ y \end{pmatrix} + \begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} + \begin{pmatrix} \frac{\partial u_1(x, y)}{\partial x} \Delta x \\ \frac{\partial u_2(x, y)}{\partial x} \Delta x \end{pmatrix} \\ C &= \begin{pmatrix} x \\ y + \Delta y \end{pmatrix} \quad \rightarrow \quad C' = \begin{pmatrix} x \\ y + \Delta y \end{pmatrix} + \begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} + \begin{pmatrix} \frac{\partial u_1(x, y)}{\partial y} \Delta y \\ \frac{\partial u_2(x, y)}{\partial y} \Delta y \end{pmatrix} \\ D &= \begin{pmatrix} x + \Delta x \\ y + \Delta y \end{pmatrix} \quad \rightarrow \quad D' = \begin{pmatrix} x + \Delta x \\ y + \Delta y \end{pmatrix} + \begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} + \begin{pmatrix} \frac{\partial u_1(x, y)}{\partial x} \Delta x + \frac{\partial u_1(x, y)}{\partial y} \Delta y \\ \frac{\partial u_2(x, y)}{\partial x} \Delta x + \frac{\partial u_2(x, y)}{\partial y} \Delta y \end{pmatrix} \end{aligned}$$

The last equation can be rewritten in the form

$$\begin{aligned} \begin{pmatrix} \Delta u_1 \\ \Delta u_2 \end{pmatrix} &:= \begin{pmatrix} u_1(x + \Delta x, y + \Delta y) \\ u_2(x + \Delta x, y + \Delta y) \end{pmatrix} - \begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial x} & \frac{\partial u_2}{\partial y} \end{bmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \\ &= \frac{1}{2} \begin{bmatrix} 2 \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \\ \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} & 2 \frac{\partial u_2}{\partial y} \end{bmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} + \frac{1}{2} \begin{bmatrix} 0 & \frac{\partial u_1}{\partial y} - \frac{\partial u_2}{\partial x} \\ \frac{\partial u_2}{\partial x} - \frac{\partial u_1}{\partial y} & 0 \end{bmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \\ &= \mathbf{A} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} + \mathbf{R} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}. \end{aligned}$$

Observe that the matrix  $\mathbf{A}$  is symmetric and  $\mathbf{R}$  is antisymmetric<sup>6</sup>.

Assuming that our structure is only slightly deformed conclude<sup>7</sup> that  $\Delta u_1$  and  $\Delta u_2$  are considerably smaller than  $\Delta x$  and  $\Delta y$ . Based on this ignore quadratic contributions  $(\Delta u)^2$ . Now compute the distance of the points  $A'$  and  $D'$  in the deformed body

$$\begin{aligned} |A'D'|^2 &= (\Delta x + \Delta u_1)^2 + (\Delta y + \Delta u_2)^2 = \langle \begin{pmatrix} \Delta x + \Delta u_1 \\ \Delta y + \Delta u_2 \end{pmatrix}, \begin{pmatrix} \Delta x + \Delta u_1 \\ \Delta y + \Delta u_2 \end{pmatrix} \rangle \\ &\approx \langle \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \rangle + \langle \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \mathbf{A} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} + \mathbf{R} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \rangle + \\ &\quad \langle \mathbf{A} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} + \mathbf{R} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \rangle \\ &= \langle \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \rangle + 2 \langle \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \mathbf{A} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \rangle \\ &= |AD|^2 + 2 \langle \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \mathbf{A} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \rangle. \end{aligned}$$

Observe that the matrix  $\mathbf{R}$  does not lead to changes of distances in the body. They correspond to rotations. Only the contributions by  $\mathbf{A}$  lead to stretching of the material.

Setting  $\Delta y = 0$  in the above formula compute the distance  $|A'B'|$  as

$$\begin{aligned} |A'B'|^2 &= (\Delta x)^2 + 2 \frac{\partial u_1}{\partial x} (\Delta x)^2 \approx (\Delta x)^2 (1 + \frac{\partial u_1}{\partial x})^2 \\ |A'B'| &= \sqrt{1 + 2 \frac{\partial u_1}{\partial x}} \Delta x \approx \Delta x + \frac{\partial u_1}{\partial x} \Delta x. \end{aligned}$$

Now examine the ratio of the change of length over the original length to obtain the **normal strains**  $\varepsilon_{xx}$  and  $\varepsilon_{yy}$  in the direction of the two axes.

$$\begin{aligned} \varepsilon_{xx} &= \frac{\text{change of length in } x \text{ direction}}{\text{length in } x \text{ direction}} = \frac{\frac{\partial u_1(x,y)}{\partial x} \Delta x}{\Delta x} = \frac{\partial u_1(x,y)}{\partial x} \\ \varepsilon_{yy} &= \frac{\text{change of length in } y \text{ direction}}{\text{length in } y \text{ direction}} = \frac{\frac{\partial u_2(x,y)}{\partial y} \Delta y}{\Delta y} = \frac{\partial u_2(x,y)}{\partial y}. \end{aligned}$$

To find the geometric interpretation of the **shear strain**

$$\varepsilon_{xy} = \varepsilon_{yx} = \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right)$$

assume that the rectangle  $ABCD$  is not rotated, as shown in Figure 5.6. Let  $\gamma_1$  be the angle formed by the line  $A'B'$  with the  $x$  axis and  $\gamma_2$  the angle between the line  $A'C'$  and the  $y$  axis. The sign convention is such that both angles in Figure 5.6 are positive. Since  $\tan \phi \approx \phi$  for small angles find

$$\begin{aligned} \tan \gamma_1 &= \frac{\frac{\partial u_2(x,y)}{\partial x} \Delta x}{\Delta x} = \frac{\partial u_2(x,y)}{\partial x} \\ \tan \gamma_2 &= \frac{\frac{\partial u_1(x,y)}{\partial y} \Delta y}{\Delta y} = \frac{\partial u_1(x,y)}{\partial y} \\ 2 \varepsilon_{xy} &= \tan \gamma_1 + \tan \gamma_2 \approx \gamma_1 + \gamma_2. \end{aligned}$$

<sup>6</sup>This implies  $\langle \vec{v}, \mathbf{A} \cdot \vec{w} \rangle = \langle \mathbf{A}^T \cdot \vec{v}, \vec{w} \rangle = \langle \mathbf{A} \cdot \vec{v}, \vec{w} \rangle$  and  $\langle \vec{v}, \mathbf{R} \cdot \vec{w} \rangle = \langle \mathbf{R}^T \cdot \vec{v}, \vec{w} \rangle = -\langle \mathbf{R} \cdot \vec{v}, \vec{w} \rangle$ .

<sup>7</sup>Due to this simplification we will later encounter a problem with rotations about large angles. A possible rescue is shown in Section 5.7 using the Cauchy–Green tensor.

Thus the number  $\varepsilon_{xy}$  indicates by how much a right angle between the  $x$  and  $y$  axis would be diminished by the given deformation.

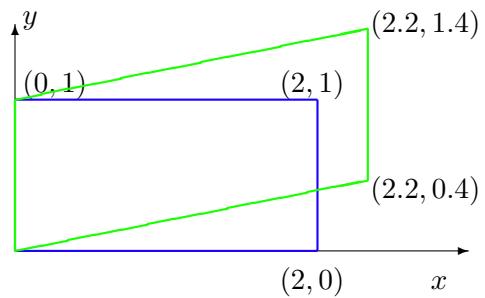
**5–16 Definition :** The matrix

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) & \frac{\partial u_2}{\partial y} \end{bmatrix}$$

is the (infinitesimal) **strain tensor**.

**5–17 Example :**

Examine a small section in a deformed solid and compare the original and deformed shape of a small rectangle. A block ( $\Delta x = 2$  and  $\Delta y = 1$ ) is deformed in the  $xy$  plane according to the figure on the right. The original shape is shown in blue and the deformed shape in green. Use this picture to read out the three strains.



- Along the  $x$ -axis observe  $\Delta u_1 = 0.2$  and  $\Delta u_2 = 0.4$ . This leads to

$$\frac{\partial u_1}{\partial x} = \frac{\Delta u_1}{\Delta x} = \frac{0.2}{2} = 0.1 \quad \text{and} \quad \frac{\partial u_2}{\partial x} = \frac{\Delta u_2}{\Delta x} = \frac{0.4}{2} = 0.2 .$$

- Along the  $y$ -axis observe  $\Delta u_1 = \Delta u_2 = 0$ . This leads to

$$\frac{\partial u_1}{\partial y} = \frac{\Delta u_1}{\Delta y} = 0 \quad \text{and} \quad \frac{\partial u_2}{\partial y} = \frac{\Delta u_2}{\Delta y} = 0 .$$

- Thus the strain tensor is given by

$$\begin{bmatrix} e_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & e_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2} \left( \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} \right) \\ \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) & \frac{\partial u_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0 \end{bmatrix} .$$

◇

**5–18 Example :** It is a good exercise to compute the strain components for a few simple deformations.

- pure translation: If the displacement vector  $\vec{u}$  is constant we have the situation of a pure translation, without deformation. Since all derivatives of  $u_1$  and  $u_2$  vanish we find  $\varepsilon_{xx} = \varepsilon_{yy} = \varepsilon_{xy} = 0$ , i.e. the strain components are all zero.
- pure rotation: A pure rotation by angle  $\phi$  is given by

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos \phi x - \sin \phi y \\ \sin \phi x + \cos \phi y \end{pmatrix}$$

and thus the displacement vector is given by

$$\begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} = \begin{pmatrix} \cos \phi x - \sin \phi y - x \\ \sin \phi x + \cos \phi y - y \end{pmatrix} .$$

Since the overall displacement has to be small compute only with small angles  $\phi^8$ . This leads to

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) & \frac{\partial u_2}{\partial y} \end{bmatrix} = \begin{bmatrix} \cos \phi - 1 & 0 \\ 0 & \cos \phi - 1 \end{bmatrix} \approx \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Again all components of the strain vanish.

- stretching in both directions: The displacement

$$\begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$$

corresponds to a stretching of the solid by the factor  $1 + \lambda$  in both directions. The components of the strain are given by

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) & \frac{\partial u_2}{\partial y} \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

i.e. there is no shear strain in this situation.

- stretching in  $x$  direction only: The displacement

$$\begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} = \lambda \begin{pmatrix} x \\ 0 \end{pmatrix}$$

corresponds to a stretching by the factor  $1 + \lambda$  along the  $x$  axis. The components of the strain are given by

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) & \frac{\partial u_2}{\partial y} \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & 0 \end{bmatrix}.$$

- stretching in  $45^\circ$  direction: The displacement

$$\begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} = \frac{\lambda}{2} \begin{pmatrix} x + y \\ x + y \end{pmatrix}$$

corresponds to a stretching by the factor  $1 + \lambda$  along the axis  $x = y$ . The straight line  $y = -x$  is left unchanged. To verify this observe

$$\begin{pmatrix} u_1(x, x) \\ u_2(x, x) \end{pmatrix} = \lambda \begin{pmatrix} x \\ x \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} u_1(x, -x) \\ u_2(x, -x) \end{pmatrix} = \lambda \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

The components of the strain are given by

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) & \frac{\partial u_2}{\partial y} \end{bmatrix} = \begin{bmatrix} \lambda/2 & \lambda/2 \\ \lambda/2 & \lambda/2 \end{bmatrix}.$$

- The two previous examples both stretch the solid in one direction by a factor  $\lambda$  and leave the orthogonal direction unchanged. Thus it is the same type of deformation, the difference being the coordinate system used to examine the result. Observe that the expressions

$$\begin{array}{ll} \varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy} & \text{depend on the coordinate system} \\ \varepsilon_{xx} + \varepsilon_{yy} \quad \text{and} \quad \frac{\partial u_1}{\partial y} - \frac{\partial u_2}{\partial x} & \text{do not depend on the coordinate system} \end{array}$$

This observation will be confirmed and proven in the next result.



<sup>8</sup>This can be improved by working with the Green strain tensor, see Section 5.7 on page 364.

**5-19 Observation :** Consider two coordinate systems, where one is generated by rotating the first coordinate axes by an angle  $\phi$ . The situation is shown in Figure 5.7 with  $\phi = \frac{\pi}{6} = 30^\circ$ . Now express a vector  $\vec{u}$  (components in  $(xy)$ -system) also in the  $(x'y')$ -system. To achieve this rotate the vector  $\vec{u}$  by  $-\phi$  and read out the components. In our example find  $\vec{u} = (1, 1)^T$  and thus

$$\vec{u}' = \mathbf{R}^T \cdot \vec{u} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \approx \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.366 \\ 0.366 \end{pmatrix}.$$

The numbers are confirmed by Figure 5.7.  $\diamond$

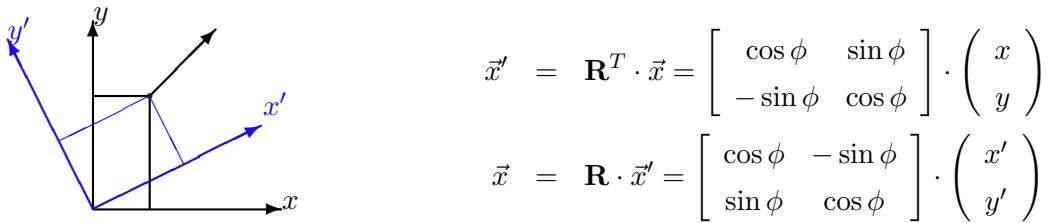


Figure 5.7: Rotation of the coordinate system

**5-20 Result :** A given strain situation is examined in two different coordinate system, as show in Figure 5.7. Then we have

$$\begin{aligned} \varepsilon_{xx} + \varepsilon_{yy} &= \varepsilon'_{x'x'} + \varepsilon'_{y'y'} \\ \frac{\partial u_1}{\partial y} - \frac{\partial u_2}{\partial x} &= \frac{\partial u'_1}{\partial y'} - \frac{\partial u'_2}{\partial x'} \end{aligned}$$

and the strain components transform according to the formula

$$\begin{bmatrix} \varepsilon'_{x'x'} & \varepsilon'_{x'y'} \\ \varepsilon'_{x'y'} & \varepsilon'_{y'y'} \end{bmatrix} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}.$$

**Proof :** Since the deformations at a given point are identical we have

$$\begin{aligned} \vec{u}'(\vec{x}') &= \mathbf{R}^T \cdot \vec{u}(\vec{x}) = \mathbf{R}^T \cdot \vec{u}(\mathbf{R} \cdot \vec{x}') \\ u'_1(\vec{x}') &= [\cos \phi \quad \sin \phi] \cdot \vec{u}(\mathbf{R} \cdot \vec{x}') = \cos \phi u_1(\vec{x}) + \sin \phi u_2(\vec{x}) \\ &= +\cos \phi u_1(\cos \phi x' - \sin \phi y', \sin \phi x' + \cos \phi y') + \\ &\quad +\sin \phi u_2(\cos \phi x' - \sin \phi y', \sin \phi x' + \cos \phi y') \\ u'_2(\vec{x}') &= [-\sin \phi \quad \cos \phi] \cdot \vec{u}(\mathbf{R} \cdot \vec{x}') = -\sin \phi u_1(\vec{x}) + \cos \phi u_2(\vec{x}) \\ &= -\sin \phi u_1(\cos \phi x' - \sin \phi y', \sin \phi x' + \cos \phi y') + \\ &\quad +\cos \phi u_2(\cos \phi x' - \sin \phi y', \sin \phi x' + \cos \phi y') \end{aligned}$$

With elementary, but lengthy application of the chain rule we find

$$\begin{aligned} \frac{\partial}{\partial x'} u'_1(\vec{x}') &= \cos \phi \left( \frac{\partial u_1}{\partial x} \cos \phi + \frac{\partial u_1}{\partial y} \sin \phi \right) + \sin \phi \left( \frac{\partial u_2}{\partial x} \cos \phi + \frac{\partial u_2}{\partial y} \sin \phi \right) \\ &= \cos^2 \phi \frac{\partial u_1}{\partial x} + \sin^2 \phi \frac{\partial u_2}{\partial y} + \cos \phi \sin \phi \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \end{aligned}$$

show  
 $\vec{u}'(\vec{x}')$  and  
 $u'_1(\vec{x}')$

show  
 $\frac{\partial}{\partial x'} u_1(\vec{x})$

$$\begin{aligned}
\frac{\partial}{\partial y'} u'_1(\vec{x}') &= \cos \phi \left( -\frac{\partial u_1}{\partial x} \sin \phi + \frac{\partial u_1}{\partial y} \cos \phi \right) + \sin \phi \left( -\frac{\partial u_2}{\partial x} \sin \phi + \frac{\partial u_2}{\partial y} \cos \phi \right) \\
&= -\cos \phi \sin \phi \frac{\partial u_1}{\partial x} + \cos \phi \sin \phi \frac{\partial u_2}{\partial y} + \cos^2 \phi \frac{\partial u_1}{\partial y} - \sin^2 \phi \frac{\partial u_2}{\partial x} \\
\frac{\partial}{\partial x'} u'_2(\vec{x}') &= -\sin \phi \left( \frac{\partial u_1}{\partial x} \cos \phi + \frac{\partial u_1}{\partial y} \sin \phi \right) + \cos \phi \left( \frac{\partial u_2}{\partial x} \cos \phi + \frac{\partial u_2}{\partial y} \sin \phi \right) \\
&= -\cos \phi \sin \phi \frac{\partial u_1}{\partial x} + \cos \phi \sin \phi \frac{\partial u_2}{\partial y} - \sin^2 \phi \frac{\partial u_1}{\partial y} + \cos^2 \phi \frac{\partial u_2}{\partial x} \\
\frac{\partial}{\partial y'} u'_2(\vec{x}') &= -\sin \phi \left( -\frac{\partial u_1}{\partial x} \sin \phi + \frac{\partial u_1}{\partial y} \cos \phi \right) + \cos \phi \left( -\frac{\partial u_2}{\partial x} \sin \phi + \frac{\partial u_2}{\partial y} \cos \phi \right) \\
&= \sin^2 \phi \frac{\partial u_1}{\partial x} + \cos^2 \phi \frac{\partial u_2}{\partial y} - \cos \phi \sin \phi \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right).
\end{aligned}$$

Now verify that

show  $\varepsilon'_{x'x'}$

$$\begin{aligned}
\varepsilon'_{x'x'} + \varepsilon'_{y'y'} &= \frac{\partial u'_1}{\partial x'} + \frac{\partial u'_2}{\partial y'} = \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} = \varepsilon_{xx} + \varepsilon_{yy}, \\
\frac{\partial u'_1}{\partial y'} - \frac{\partial u'_2}{\partial x'} &= \frac{\partial u_1}{\partial y} - \frac{\partial u_2}{\partial x}.
\end{aligned}$$

These two expressions are thus independent on the orientation of the coordinate system.

If the matrix multiplication below is carried one step further, then the claimed transformation formula will appear.

$$\begin{aligned}
&\mathbf{R}^T \cdot \begin{bmatrix} 2\varepsilon_{xx} & 2\varepsilon_{xy} \\ 2\varepsilon_{xy} & 2\varepsilon_{yy} \end{bmatrix} \cdot \mathbf{R} = \\
&= \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} 2 \frac{\partial u_1}{\partial x} & \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} & 2 \frac{\partial u_2}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \\
&= \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} 2 \cos \phi \frac{\partial u_1}{\partial x} + \sin \phi (\frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y}) & -2 \sin \phi \frac{\partial u_1}{\partial x} + \cos \phi (\frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y}) \\ \cos \phi (\frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y}) + 2 \sin \phi \frac{\partial u_2}{\partial y} & -\sin \phi (\frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y}) + 2 \cos \phi \frac{\partial u_2}{\partial y} \end{bmatrix}
\end{aligned}$$

□

**5-21 Example :** To read out the strain in a direction given by the normalized directional vector  $\vec{d} = (d_1, d_2)^T = (\cos \alpha, \sin \alpha)^T$  compute the normal strain  $\frac{\Delta l}{l}$  in that direction<sup>9</sup> by

$$\frac{\Delta l}{l} = \langle \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}, \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \rangle.$$

To verify this result

- Construct a rotation matrix  $\mathbf{R}$ , such that the new  $x'$  direction coincides with the direction given by  $\vec{d}$ .

$$\mathbf{R} = \begin{bmatrix} d_1 & -d_2 \\ d_2 & d_1 \end{bmatrix}$$

- Apply the above transformation rule for the strain.

$$\begin{bmatrix} \varepsilon'_{x'x'} & \varepsilon'_{x'y'} \\ \varepsilon'_{x'y'} & \varepsilon'_{y'y'} \end{bmatrix} = \mathbf{R}^T \cdot \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \mathbf{R}$$

<sup>9</sup>This result might be useful when working with strain gauges to measure deformations.

- Then the top left entry  $\varepsilon'_{x'x'}$  shows the normal strain  $\frac{\Delta l}{l}$  in that direction.

$$\begin{aligned}\frac{\Delta l}{l} &= \varepsilon'_{x'x'} = \langle \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{bmatrix} \varepsilon'_{x'x'} & \varepsilon'_{x'y'} \\ \varepsilon'_{x'y'} & \varepsilon'_{y'y'} \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rangle \\ &= \langle \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \mathbf{R}^T \cdot \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \mathbf{R} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rangle = \langle \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}, \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \rangle\end{aligned}$$

As an example consider the  $45^\circ$  direction, thus  $\vec{d} = \frac{1}{\sqrt{2}}(1, 1)^T$  and conclude

$$\begin{aligned}\frac{\Delta l}{l} &= \langle \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \rangle \\ &= \frac{1}{2} \langle \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} + \varepsilon_{xy} \\ \varepsilon_{xy} + \varepsilon_{yy} \end{pmatrix} \rangle \\ &= \frac{1}{2} (\varepsilon_{xx} + 2\varepsilon_{xy} + \varepsilon_{yy}) = \frac{1}{2} \left( \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial y} \right).\end{aligned}$$

◊

Since the strain matrix is symmetric, there always exists<sup>10</sup> an angle  $\phi$  such that the strain matrix in the new coordinate system is diagonal, i.e.

$$\begin{bmatrix} \varepsilon'_{x'x'} & 0 \\ 0 & \varepsilon'_{y'y'} \end{bmatrix} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}.$$

Thus at least close to the examined point the deformation consists of stretching (or compressing) the  $x'$  axis and stretching (or compressing) the  $y'$  axis. No shear strain is observed in this new coordinate system. One of the possible displacements is given by

$$\begin{pmatrix} x' \\ y' \end{pmatrix} \rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} + \begin{pmatrix} \varepsilon'_{x'x'} x' \\ \varepsilon'_{y'y'} y' \end{pmatrix}.$$

The values of  $\varepsilon'_{x'x'}$  and  $\varepsilon'_{y'y'}$  can be found as eigenvalues of the original strain matrix, i.e. solutions of the quadratic equation

$$f(\lambda) = \det \begin{bmatrix} \varepsilon_{xx} - \lambda & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} - \lambda \end{bmatrix} = 0.$$

The eigenvectors indicate the directions of pure strain, i.e. in that coordinate system you find no shear strain. The eigenvalues correspond to the **principal strains**.

<sup>10</sup>The eigenvector  $\vec{e}_1$  to the first eigenvalue  $\lambda_1$  can be normalized and thus written in the form

$$\vec{e}_1 = \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix} \implies \mathbf{A} \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix} = \lambda_1 \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix}.$$

The second eigenvector  $\vec{e}_2$  is orthogonal to the first and thus

$$\mathbf{A} \begin{pmatrix} -\sin \phi \\ \cos \phi \end{pmatrix} = \lambda_2 \begin{pmatrix} -\sin \phi \\ \cos \phi \end{pmatrix} \implies \mathbf{A} \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}.$$

Multiply the last equation from the left by the transpose of the rotation matrix to arrive at the diagonalization result.

**5-22 Example :** Examine the strain matrix

$$\mathbf{A} = \begin{bmatrix} 0.04 & 0.01 \\ 0.01 & 0 \end{bmatrix}$$

This corresponds to a solid stretched by 4% in the  $x$  direction and the angle between the  $x$  and  $y$  axis is diminished by 0.02. To diagonalize this matrix we determine the zeros of

$$\det(\mathbf{A} - \lambda \mathbb{I}) = \det \begin{bmatrix} 0.04 - \lambda & 0.01 \\ 0.01 & 0 - \lambda \end{bmatrix} = \lambda^2 - 0.04\lambda - 0.01^2 = (\lambda - 0.02)^2 - 0.02^2 - 0.01^2 = 0$$

with the solutions  $\lambda_1 = 0.02 + \sqrt{0.0005} = 0.01(2 + \sqrt{5}) \approx 0.04236$  and  $\lambda_2 = 0.02 - \sqrt{0.0005} = 0.01(2 - \sqrt{5}) \approx -0.00236$ . The first eigenvector is then found as solution of the linear system

$$\begin{bmatrix} 0.04 - \lambda_1 & 0.01 \\ 0.01 & 0 - \lambda_1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \approx \begin{bmatrix} -0.00236 & 0.01 \\ 0.01 & -0.04236 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

The second of the above equations is a multiple of the first and thus use only the first equation

$$-0.00236x + 0.01y = 0.$$

Since only the direction matters generate an easy solution by

$$\vec{e}_1 = \begin{pmatrix} 1 \\ 0.236 \end{pmatrix} \quad \text{with } \lambda_1 = 0.04236.$$

The second eigenvector  $\vec{e}_2$  is orthogonal to the first and thus

$$\vec{e}_2 = \begin{pmatrix} 0.236 \\ -1 \end{pmatrix} \quad \text{with } \lambda_2 = -0.00236.$$

As a consequence the above strain corresponds to a pure stretching by 4.2% in the direction of  $\vec{e}_1$  and a compression of 0.2% in the orthogonal direction.  $\diamond$

### Strain for solids in space

So far all calculations were made in the plane, but they can readily be adapted to solids in space. If the deformation of a solid is given by the deformation vector field  $\vec{u}$ , i.e.

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \longrightarrow \vec{x} + \vec{u} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

then we can compute the three normal and three strain components by the formulas in Table 5.3<sup>11</sup>.

The above results about transformation of strains in a rotated coordinate system do also apply. Thus for a given strain there is a rotation of the coordinate system, given by the orthonormal matrix  $\mathbf{R}$  such that

$$\begin{bmatrix} \varepsilon'_{x'x'} & 0 & 0 \\ 0 & \varepsilon'_{y'y'} & 0 \\ 0 & 0 & \varepsilon'_{z'z'} \end{bmatrix} = \mathbf{R}^T \cdot \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_{zz} \end{bmatrix} \cdot \mathbf{R}.$$

The entries on the diagonal are called **principal strains**.

<sup>11</sup>In part of the literature (e.g. [Prze68]) the shear strains are defined without the division by 2. All results can be adapted accordingly.

symbol	formula	interpretation
$\varepsilon_{xx}$	$\frac{\partial u_1}{\partial x}$	ratio of change of length divided by length in $x$ direction
$\varepsilon_{yy}$	$\frac{\partial u_2}{\partial y}$	ratio of change of length divided by length in $y$ direction
$\varepsilon_{zz}$	$\frac{\partial u_3}{\partial z}$	ratio of change of length divided by length in $z$ direction
$\varepsilon_{xy} = \varepsilon_{yx}$	$\frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right)$	the angle between the $x$ and $y$ axis is diminished by $2\varepsilon_{xy}$
$\varepsilon_{xz} = \varepsilon_{zx}$	$\frac{1}{2} \left( \frac{\partial u_1}{\partial z} + \frac{\partial u_3}{\partial x} \right)$	the angle between the $x$ and $z$ axis is diminished by $2\varepsilon_{xz}$
$\varepsilon_{yz} = \varepsilon_{zy}$	$\frac{1}{2} \left( \frac{\partial u_2}{\partial z} + \frac{\partial u_3}{\partial y} \right)$	the angle between the $y$ and $z$ axis is diminished by $2\varepsilon_{yz}$

Table 5.3: Normal and shear strains in space

### Invariant expressions of the strain tensor

Many physical expressions do not depend on the coordinate system used to describe the system, e.g. the energy density of the system. Thus they should be invariant under rotations of the above type. To examine this invariant expressions are the essential tool. For this use the characteristic polynomial of the above matrices. Let  $\mathbf{S}$  and  $\mathbf{S}'$  be the strain matrix in the original and rotated coordinate system. Thus examine

$$\begin{aligned}\mathbf{S}' &= \mathbf{R}^T \mathbf{S} \mathbf{R} \\ \det(\mathbf{S}' - \lambda \mathbb{I}) &= \det(\mathbf{R}^T \mathbf{S} \mathbf{R} - \lambda \mathbf{R}^T \mathbf{R}) = \det(\mathbf{R}^T (\mathbf{S} - \lambda \mathbb{I}) \mathbf{R}) \\ &= \det(\mathbf{R}^T) \det(\mathbf{S} - \lambda \mathbb{I}) \det(\mathbf{R}) = \det(\mathbf{S} - \lambda \mathbb{I}).\end{aligned}$$

The two characteristic polynomials are identical and the coefficients for  $\lambda^3$ ,  $\lambda^2$ ,  $\lambda^1$  and  $\lambda^0 = 1$  have to coincide. This leads to three invariants. Compute the determinant to find

$$\begin{aligned}\det(\mathbf{S} - \lambda \mathbb{I}) &= \det \begin{bmatrix} \varepsilon_{xx} - \lambda & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} - \lambda & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_{zz} - \lambda \end{bmatrix} \\ &= -\lambda^3 + \lambda^2 (\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}) - \\ &\quad -\lambda (\varepsilon_{yy}\varepsilon_{zz} - \varepsilon_{yz}^2 + \varepsilon_{xx}\varepsilon_{zz} - \varepsilon_{xz}^2 + \varepsilon_{xx}\varepsilon_{yy} - \varepsilon_{xy}^2) + \det(\mathbf{S}).\end{aligned}$$

As a consequence find three invariant strain expressions.

$$\begin{aligned}I_1 &= \text{trace}(\mathbf{S}) = \varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz} \\ I_2 &= \varepsilon_{yy}\varepsilon_{zz} - \varepsilon_{yz}^2 + \varepsilon_{xx}\varepsilon_{zz} - \varepsilon_{xz}^2 + \varepsilon_{xx}\varepsilon_{yy} - \varepsilon_{xy}^2 \\ I_3 &= \det(\mathbf{S}) = \varepsilon_{xx}\varepsilon_{yy}\varepsilon_{zz} + 2\varepsilon_{xy}\varepsilon_{yz}\varepsilon_{yz} - \varepsilon_{xx}\varepsilon_{yz}^2 - \varepsilon_{yy}\varepsilon_{xz}^2 - \varepsilon_{zz}\varepsilon_{xy}^2\end{aligned}$$

- These invariant expressions can be expressed in terms of the eigenvalues of the strain matrix  $\mathbf{S}$ :  $I_1 = \lambda_1 + \lambda_2 + \lambda_3$ ,  $I_2 = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1$  and  $I_3 = \lambda_1 \cdot \lambda_2 \cdot \lambda_3$ . To verify this fact diagonalize the matrix, i.e. compute with  $\lambda_1 = \varepsilon_{xx}$ ,  $\lambda_2 = \varepsilon_{yy}$ ,  $\lambda_3 = \varepsilon_{zz}$  and vanishing shearing strains.
- Since  $\det(\mathbf{S}' - \lambda \mathbb{I}) = \det(\mathbf{S} - \lambda \mathbb{I})$  the three eigenvalues  $\lambda_i$  are invariant. But there is no easy, explicit expression for the eigenvalues in terms of coefficients of the tensor, since a cubic equation has to be solved to determine the values of  $\lambda_i$ .

We will see (page 351) that the elastic energy density can be expressed in terms of the invariants  $I_i$ .

### 5.3.2 Description of Stress

For sake of simplicity first examine planar situations and at the end of the section apply the obvious extensions to the more realistic situation in space.

Consider an elastic body where all forces are parallel to the  $xy$  plane and independent on  $z$  and the contour of the solid is independent on  $z$ . Consider a small rectangular box of this solid with width  $\Delta x$ , height  $\Delta y$  and depth  $\Delta z$ . A cut parallel to the  $xy$  plane is shown in Figure 5.8. Based on the formula

$$\text{stress} = \frac{\text{force}}{\text{area}}$$

examine the **normal stress** and **tangential stress** components on the surfaces of this rectangle. Assume that the small box is a static situation and there are no external body forces. Balancing all components of forces and moments<sup>12</sup> leads to the conditions

$$\sigma_x^2 = \sigma_x^1 \quad , \quad \sigma_y^3 = \sigma_y^4 \quad , \quad \tau_{yx}^1 = \tau_{xy}^2 = \tau_{xy}^3 = \tau_{xy}^4 .$$

Thus the situation simplifies as shown on the right in Figure 5.8.

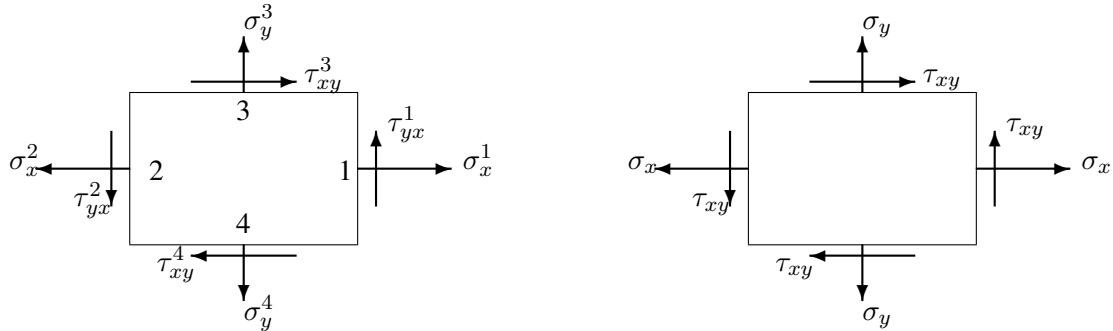


Figure 5.8: Definition of stress in a plane, initial (left) and simplified (right) situation

The stress situation of a solid is described by all components of the stress, typically as functions of the location.

#### Normal and tangential stress in an arbitrary direction

Figure 5.9 shows a virtual cut of a solid such that the normal vector  $\vec{n} = (\cos \phi, \sin \phi)^T$  forms an angle  $\phi$  with the  $x$  axis. Now examine the normal stress  $\sigma$  and the tangential stress  $\tau$  along side  $A$ .

Since  $A_x = A \sin \phi$  and  $A_y = A \cos \phi$  the condition of balance of force leads to

$$\begin{aligned} s_x A &= \sigma_x A_y + \tau_{xy} A_x = \sigma_x \cos \phi + \tau_{xy} \sin \phi \\ s_y A &= \sigma_y A_x + \tau_{xy} A_y = \tau_{xy} \cos \phi + \sigma_y \sin \phi \end{aligned}$$

where  $\vec{s} = (s_x, s_y)^T$ . Using matrices write the above in the form

$$\begin{pmatrix} s_x \\ s_y \end{pmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix} \quad \text{or} \quad \vec{s} = \mathbf{S} \cdot \vec{n} \quad (5.9)$$

<sup>12</sup>Balancing the forces in  $x$  and  $y$  direction and the moment leads to

$$\begin{aligned} (\sigma_x^1 - \sigma_x^2) \Delta y + (\tau_{xy}^3 - \tau_{xy}^4) \Delta x &= 0 \\ (\sigma_y^3 + \sigma_y^4) \Delta x + (\tau_{yx}^1 - \tau_{yx}^2) \Delta y &= 0 \\ (\tau_{yx}^1 + \tau_{yx}^2) \Delta y - (\tau_{xy}^3 + \tau_{xy}^4) \Delta x &= 0 \end{aligned}$$

for all positive values of  $\Delta x$  and  $\Delta y$ . Change the values of  $\Delta x$  and  $\Delta y$  independently to arrive at the desired conclusion.

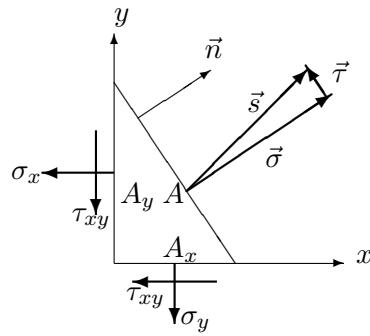


Figure 5.9: Normal and tangential stress in an arbitrary direction

where the symmetric **stress matrix** is given by

$$\mathbf{S} = \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix}.$$

The stress vector  $\vec{s}$  may be decomposed in a normal component  $\sigma$  and a tangential component  $\tau$ . Determine the component of  $\vec{\sigma}$  in the direction of  $\vec{n}$  by

$$\begin{aligned} \sigma &= \langle \vec{n}, \vec{s} \rangle = \vec{n}^T \cdot \vec{s} = \langle \vec{n}, \vec{s} \rangle = (\cos \phi, \sin \phi) \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix} \\ \tau &= (-\sin \phi, \cos \phi) \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix}. \end{aligned}$$

The value of  $\sigma$  is positive if  $\vec{\sigma}$  is pointing out of the solid and  $\tau$  is positive if  $\vec{\tau}$  is pointing upward in Figure 5.9.

This allows to consider a new coordinate system, generated by rotation of the  $xy$  system by an angle  $\phi$  (see Figure 5.7, page 331). The result is

$$\begin{aligned} \sigma_{x'} &= (\cos \phi, \sin \phi) \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix} \\ \sigma_{y'} &= (-\sin \phi, \cos \phi) \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} -\sin \phi \\ \cos \phi \end{pmatrix} \\ \tau_{x'y'} &= (-\sin \phi, \cos \phi) \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix}. \end{aligned}$$

An elementary matrix multiplication shows that this is equivalent to

$$\begin{bmatrix} \sigma_{x'} & \tau_{x'y'} \\ \tau_{x'y'} & \sigma_{y'} \end{bmatrix} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \cdot \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \quad (5.10)$$

This transformation formula should be compared with result 5-20 on page 331. It shows that the behavior under coordinate rotations for the stress matrix and the strain matrix is identical.

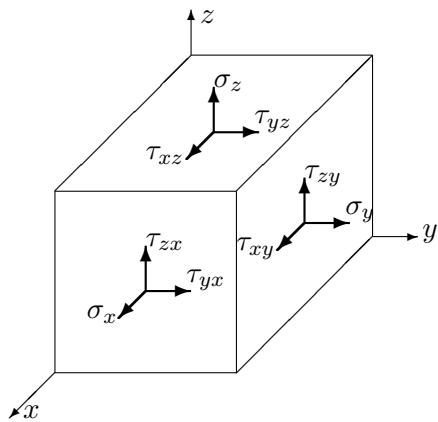


Figure 5.10: Components of stress in space

symbol	description
$\sigma_x$	normal stress at a surface orthogonal to $x = \text{const}$
$\sigma_y$	normal stress at a surface orthogonal to $y = \text{const}$
$\sigma_z$	normal stress at a surface orthogonal to $z = \text{const}$
$\tau_{xy} = \tau_{yx}$	tangential stress in $y$ direction at surface orthogonal to $x = \text{const}$ tangential stress in $x$ direction at surface orthogonal to $y = \text{const}$
$\tau_{xz} = \tau_{zx}$	tangential stress in $z$ direction at surface orthogonal to $x = \text{const}$ tangential stress in $x$ direction at surface orthogonal to $z = \text{const}$
$\tau_{yz} = \tau_{zy}$	tangential stress in $z$ direction at surface orthogonal to $y = \text{const}$ tangential stress in $y$ direction at surface orthogonal to $z = \text{const}$

Table 5.4: Description of normal and tangential stress in space

### Normal and tangential stress in space

All the above observations can be adapted to the situation in space. Figure 5.10 shows the notational convention and Table 5.4 gives a short description.

The symmetric **stress matrix**  $\mathbf{S}$  is given by

$$\mathbf{S} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix}$$

and the stress vector  $\vec{s}$  at a plane orthogonal to  $\vec{n}$  is given by

$$\vec{s} = \mathbf{S} \cdot \vec{n}.$$

The behavior of  $\mathbf{S}$  under rotation of the coordinate system  $\vec{x} = \mathbf{R}^T \cdot \vec{x}'$  or  $\vec{x}' = \mathbf{R} \cdot \vec{x}$  is given by

$$\mathbf{S}' = \begin{bmatrix} \sigma'_x & \tau'_{xy} & \tau'_{xz} \\ \tau'_{xy} & \sigma'_y & \tau'_{yz} \\ \tau'_{xz} & \tau'_{yz} & \sigma'_z \end{bmatrix} = \mathbf{R}^T \cdot \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} \cdot \mathbf{R}.$$

When solving the cubic equation

$$\det(\mathbf{S} - \lambda \mathbb{I}_3) = \det \begin{bmatrix} \sigma_x - \lambda & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y - \lambda & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z - \lambda \end{bmatrix} = 0$$

for the three eigenvalues  $\lambda_{1,2,3}$  and the corresponding orthonormal eigenvectors  $\vec{e}_1$ ,  $\vec{e}_2$  and  $\vec{e}_3$ , we find a coordinate system in which all tangential stress components vanish. Since there are only normal stresses the stress matrix  $\mathbf{S}'$  has the form

$$\begin{bmatrix} \sigma'_x & 0 & 0 \\ 0 & \sigma'_y & 0 \\ 0 & 0 & \sigma'_z \end{bmatrix}.$$

The numbers on the diagonal are the **principal stresses**. This is very useful to extract results out of stress computations. When asked to find the stress at a given point in a solid many different forms of answers are possible:

- Give all six components of the stress in a given coordinate system.
- Find the three principal stresses and use those as a result. One might also give the corresponding directions.
- Give the maximal principal stress.
- Give the maximal and minimal principal stress.
- Give the von Mises stress, or the Tresca stress

The “correct” form of the answer depends on the context.

### 5-23 Result : Mohr's circle

Using the above transformation law start out with a situation of principal stresses  $\sigma_1$  and  $\sigma_2$  and then rotate the coordinate system by an angle  $\phi$ .

$$\begin{aligned} \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} &= \begin{bmatrix} \cos \phi & +\sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi \\ +\sin \phi & \cos \phi \end{bmatrix} \\ &= \begin{bmatrix} \cos \phi & +\sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \sigma_1 \cos \phi & -\sigma_1 \sin \phi \\ +\sigma_2 \sin \phi & \sigma_2 \cos \phi \end{bmatrix} \\ &= \begin{bmatrix} \sigma_1 \cos^2 \phi + \sigma_2 \sin^2 \phi & (\sigma_2 - \sigma_1) \cos \phi \sin \phi \\ (\sigma_2 - \sigma_1) \cos \phi \sin \phi & \sigma_1 \sin^2 \phi + \sigma_2 \cos^2 \phi \end{bmatrix} \end{aligned}$$

Using trigonometric identities this leads to

$$\begin{aligned} \sigma_x &= \sigma_1 \cos^2 \phi + \sigma_2 \sin^2 \phi = \sigma_1 \frac{1 + \cos 2\phi}{2} + \sigma_2 \frac{1 - \cos 2\phi}{2} = \frac{\sigma_1 + \sigma_2}{2} - \frac{\sigma_2 - \sigma_1}{2} \cos 2\phi \\ \sigma_y &= \sigma_1 \sin^2 \phi + \sigma_2 \cos^2 \phi = \sigma_1 \frac{1 - \cos 2\phi}{2} + \sigma_2 \frac{1 + \cos 2\phi}{2} = \frac{\sigma_1 + \sigma_2}{2} + \frac{\sigma_2 - \sigma_1}{2} \cos 2\phi \\ \tau_{xy} &= (\sigma_2 - \sigma_1) \cos \phi \sin \phi = \frac{\sigma_2 - \sigma_1}{2} \sin 2\phi \end{aligned}$$

Since  $(R \cos 2\phi, R \sin 2\phi)$  is the parameterization of a circle observe that  $(\sigma_x, -\tau_{xy})$  and  $(\sigma_y, +\tau_{xy})$  are on a circle in the  $(\sigma, \tau)$ -plane with center at  $\frac{\sigma_1 + \sigma_2}{2}$  on the  $\sigma$ -axis and radius  $\frac{\sigma_2 - \sigma_1}{2}$ . This is Mohr's circle shown in Figure 5.11(a).

- If the values of  $\sigma_x$ ,  $\sigma_y$  and  $\tau_{xy}$  are known one can easily construct Mohr's circle and then read out the principal stresses  $\sigma_1$  and  $\sigma_2$ . This is equivalent to solving the corresponding quadratic equation.

$$\begin{aligned} 0 &= \det(\mathbf{S} - \sigma \mathbb{I}) = (\sigma_x - \sigma)(\sigma_y - \sigma) + \tau_{xy}^2 = \sigma^2 - (\sigma_x + \sigma_y)\sigma + \sigma_x \sigma_y + \tau_{xy}^2 \\ \sigma_{1,2} &= \frac{1}{2} \left( (\sigma_x + \sigma_y) \pm \sqrt{(\sigma_x + \sigma_y)^2 - 4\sigma_x \sigma_y + 4\tau_{xy}^2} \right) = \frac{\sigma_x + \sigma_y}{2} \pm \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2} \end{aligned}$$

- The rotation angle  $\phi$  is determined by

$$\tan(2\phi) = \frac{\tau_{xy}}{\sigma_y - \frac{\sigma_x + \sigma_y}{2}} = \frac{2\tau_{xy}}{\sigma_y - \sigma_x}$$

- There is an adaption of Mohr's circle to a 3D stress setup, shown in Figure 5.11(b). The Wikipedia page [en.wikipedia.org/wiki/Mohr's\\_circle](https://en.wikipedia.org/wiki/Mohr%27s_circle) shows more information on Mohr's circle. In [ChouPaga67, §1.5, §1.8] find a derivation of Mohr's circle in 2D and 3D.
- Since the transformation law for strains is identical one can generate a similar circle with the normal strains on the horizontal axis and the shear strains on the vertical axis.

◇

### 5.3.3 Invariant Stress Expressions, Von Mises Stress and Tresca Stress

The **von Mises stress**  $\sigma_M$  (also called octahedral shear stress) is a scalar expression and often used to examine failure modes of solids, see also the following section.

$$\begin{aligned} \sigma_M^2 &= \sigma_x^2 + \sigma_y^2 + \sigma_z^2 - \sigma_x \sigma_y - \sigma_y \sigma_z - \sigma_z \sigma_x + 3\tau_{xy}^2 + 3\tau_{yz}^2 + 3\tau_{zx}^2 \\ &= \frac{1}{2} ((\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_z - \sigma_x)^2) + 3(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2) . \end{aligned}$$

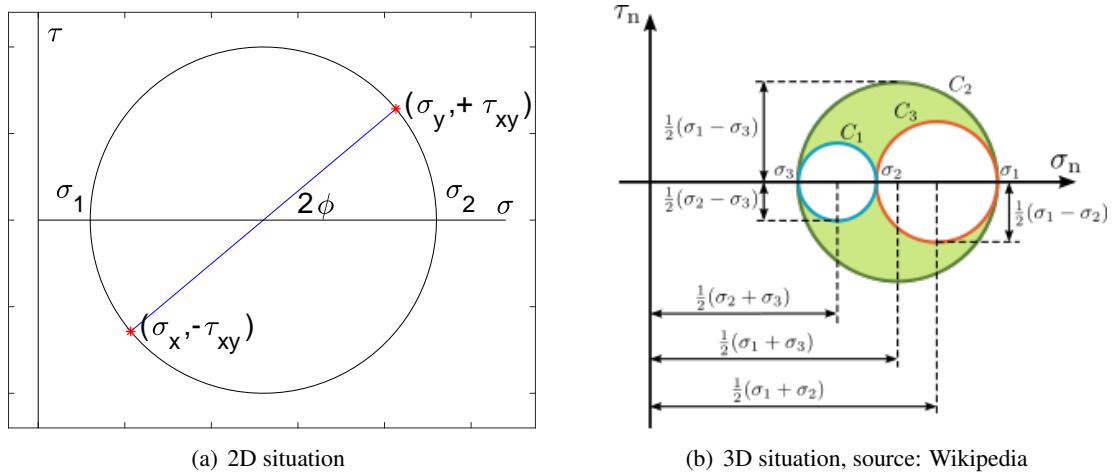


Figure 5.11: Mohr's circle for the 2D and 3D situations

It is important that the above expression for the von Mises stress does not depend on the orientation of the coordinate system. On page 335 find the invariants for the strain matrix. Use identical arguments determine three invariants for the stress matrix.

$$\begin{aligned} I_1 &= \sigma_x + \sigma_y + \sigma_z \\ I_2 &= \sigma_y \sigma_z + \sigma_x \sigma_z + \sigma_x \sigma_y - \tau_{yz}^2 - \tau_{xz}^2 - \tau_{xy}^2 \\ I_3 &= \det \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} = +\sigma_x \sigma_y \sigma_z + 2 \tau_{xy} \tau_{xz} \tau_{yz} - \sigma_x \tau_{yz}^2 - \sigma_y \tau_{xz}^2 - \sigma_z \tau_{xy}^2 \end{aligned}$$

Obviously any function of these invariants is invariant too and consequently independent of the orientation of the coordinate system. Many physically important expressions have to be invariant, e.g. the energy density.

With elementary (but tedious) algebra find

$$I_1^2 - 3I_2 = \sigma_x^2 + \sigma_y^2 + \sigma_z^2 - \sigma_y \sigma_z - \sigma_x \sigma_z - \sigma_x \sigma_y + 3\tau_{yz}^2 + 3\tau_{xz}^2 + 3\tau_{xy}^2 = \sigma_M^2$$

and consequently the von Mises stress is invariant under rotations. If reduced to principal stresses (no shear stress) find

$$2\sigma_M^2 = (\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2.$$

Thus the von Mises stress is a measure for the **differences** among the three principal stresses. In the simplest possible case of stress in one direction only, i.e.  $\sigma_2 = \sigma_3 = 0$ , find

$$\sigma_M^2 = \frac{1}{2} ((\sigma_1 - 0)^2 + (0 - 0)^2 + (0 - \sigma_1)^2) = \sigma_1^2.$$

The **Tresca stress**  $\sigma_T$  is defined by

$$\sigma_T = \max\{|\sigma_1 - \sigma_2|, |\sigma_2 - \sigma_3|, |\sigma_3 - \sigma_1|\}$$

and thus a measure of the differences amongst the principal stresses, similar to the von Mises stress.

**5-24 Corollary :** *The von Mises stress is smaller than the Tresca stress, i.e.*

$$0 \leq \sigma_M \leq \sigma_T .$$

◇

**Proof :** Without loss of generality examine the principal stress situation and assume  $\sigma_1 \leq \sigma_2 \leq \sigma_3$ .

$$\begin{aligned} 2\sigma_M^2 &= (\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2 \\ 2\sigma_T^2 &= 2(\sigma_3 - \sigma_1)^2 \\ 2(\sigma_M^2 - \sigma_T^2) &= (\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 - (\sigma_3 - \sigma_1)^2 = 2\sigma_2^2 - 2\sigma_1\sigma_2 - 2\sigma_3\sigma_2 + 2\sigma_1\sigma_3 \\ &= 2(\sigma_2 - \sigma_3)(\sigma_2 - \sigma_1) \leq 0 \end{aligned}$$

This implies  $\sigma_M^2 \leq \sigma_T^2$ . □

To decide whether a material will fail you will need the maximal principal stress, the von Mises and Tresca stress. You need the definition of the different stresses and strains, Hooke's law (Section 5.6) and possibly the plane stress and plane strain description (Sections 5.8 and 5.9). For a given situation there are multiple paths to compute these:

- If the  $3 \times 3$  stress matrix is known, then first determine the eigenvalues, i.e. the principal stresses. Once you have these it is easy to read out the desired values.
- If the  $3 \times 3$  strain matrix is known, then you have two options to determine the principal stresses.
  1. First use Hooke's law to determine the  $3 \times 3$  stress matrix, then proceed as above.
  2. Determine the eigenvalues of the strain matrix to determine the principal strains. Then use Hooke's law to determine the principal stresses.
- If the situation is a plane stress situation and you know the  $2 \times 2$  stress matrix, then you may first generate the full  $3 \times 3$  stress matrix, and then proceed as above.
- If the situation is a plane strain situation and you know the  $2 \times 2$  strain matrix, then you may first generate the full  $3 \times 3$  strain matrix, and then proceed as above.

These computational paths are illustrated in Figure 5.12.

## 5.4 Elastic Failure Modes

The results in this section are inspired by [Hear97]. The criterion for elastic failure depend on the type of material to be examined: ductile<sup>13</sup> or brittle<sup>14</sup>. To simplify the formulation assume that the stress tensor is given in principal form, i.e.

$$\mathbf{S} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} .$$

<sup>13</sup>In German: dehnbar, zäh

<sup>14</sup>In German: spröd, brüchig

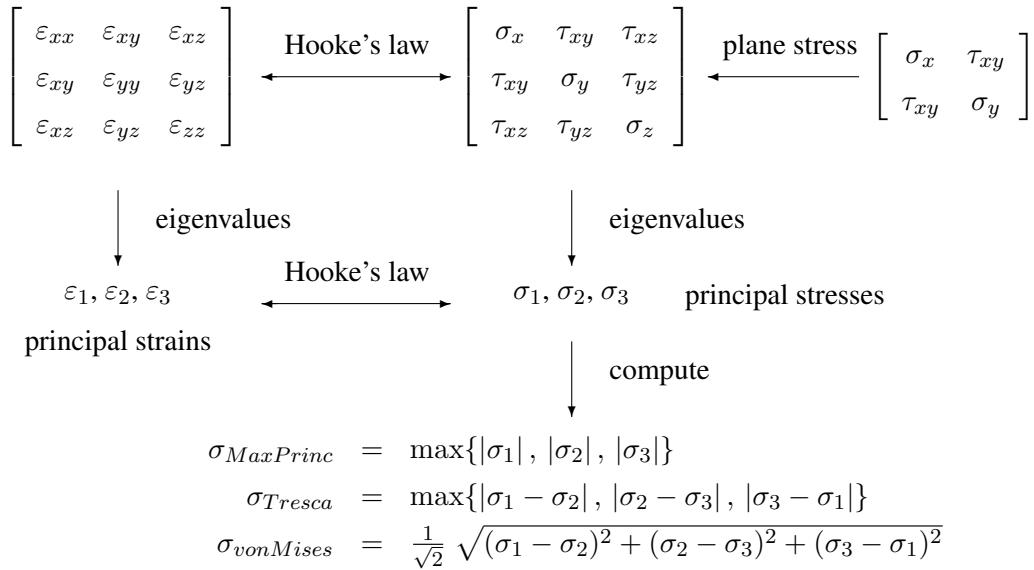


Figure 5.12: How to determine the maximal principal stress, von Mises stress and Tresca stress

#### 5.4.1 Maximum Principal Stress Theory

When the maximum principal stress exceeds a critical **yield stress**  $\sigma_Y$  the material might fail. For the material not to fail the condition

$$\max\{|\sigma_1|, |\sigma_2|, |\sigma_3|\} \leq \sigma_Y$$

has to be satisfied. This theory can be shown to apply well to brittle materials, while it should not be applied to ductile materials. Even in the case of a pure tension test, a ductile material failing is caused by large shearing, as examined in the next section. Homogeneous materials can withstand huge hydrostatic pressures, indicating that a maximum principal stress criterion might not be a wise choice.

#### 5.4.2 Maximum Shear Stress Theory

For the 2D situation we recognize from Mohr's circle in Result 5–23 that the shear stress at a plane with angle  $\phi$  is given by

$$\tau_{xy} = (\sigma_2 - \sigma_1) \cos \phi \sin \phi.$$

The maximal value is attained at  $45^\circ$  angles and the maximal value is  $\frac{1}{2}(\sigma_2 - \sigma_1)$ . This leads to the **Tresca stress**

$$\sigma_T = \max\{|\sigma_1 - \sigma_2|, |\sigma_2 - \sigma_3|, |\sigma_3 - \sigma_1|\}.$$

If working under the condition that the material fails because of shear stresses, we are lead to the condition

$$\sigma_T \leq \sigma_Y.$$

Most cracks in metals are caused by severe shearing forces, i.e. they are ductile. If a metal is submitted to a traction test (pull until it breaks) the direction of the initial break and the force show an angle of  $45^\circ$ . This is very different for brittle materials<sup>15</sup>.

Show result of traction test

Twist chalk Ex 5.9

<sup>15</sup>Twisting a piece of chalk until it breaks usually leads to an angle of  $45^\circ$  for the initial crack. It is an exercise to verify that for twisting the maximal normal stress occurs at this angle. Since chalk is a brittle material it breaks because of the maximal normal stress is too large.

### 5.4.3 Maximum Distortion Energy

The stress may be written as a sum of a hydrostatic stress (identical in all directions) and shape changing stresses.

$$\begin{aligned}\sigma_1 &= \frac{1}{3}(\sigma_1 + \sigma_2 + \sigma_3) + \frac{1}{3}(\sigma_1 - \sigma_2) + \frac{1}{3}(\sigma_1 - \sigma_3) \\ \sigma_2 &= \frac{1}{3}(\sigma_1 + \sigma_2 + \sigma_3) + \frac{1}{3}(\sigma_2 - \sigma_1) + \frac{1}{3}(\sigma_2 - \sigma_3) \\ \sigma_3 &= \frac{1}{3}(\sigma_1 + \sigma_2 + \sigma_3) + \frac{1}{3}(\sigma_3 - \sigma_1) + \frac{1}{3}(\sigma_3 - \sigma_2).\end{aligned}$$

The elastic energy density  $W$  can be decomposed into a volume changing component and a shape changing part.

$$W = \frac{1-2\nu}{6E}(\sigma_1 + \sigma_2 + \sigma_3)^2 + \frac{1+\nu}{3E}\sigma_M^2$$

This is verified in an exercise. A similar argument can be based on strain instead of stress, see Section 5.6.3 Ex 5.14 on page 352. When computing the energy contribution of the shape changing stresses we are thus lead to the **von Mises stress**

$$2\sigma_M^2 = (\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2$$

and the corresponding criterion

$$\sigma_M \leq \sigma_Y.$$

## 5.5 Scalars, Vectors and Tensors

This section is a very brief introduction to Cartesian tensors. It consists of the main definitions, the transformation rules and a few examples. For a readable, elementary introduction you may consult [Sege77]. A  $n$ -th order tensor in  $\mathbb{R}^3$  is an object whose specification requires a collection of  $3^n$  numbers, called component of the tensor. Scalars are tensors of order 0 with  $3^0 = 1$  components, vectors are tensors of order 1 with  $3^1 = 3$  components and second order tensors have  $3^2 = 9$  components. For an object to be called a tensor the components have to satisfy specific transformation rules when the coordinate system is changed.

To simplify the presentation examine the situation in  $\mathbb{R}^2$  only, but it has to be pointed out that all results and examples remain valid in  $\mathbb{R}^3$ .

### 5.5.1 Change of Coordinate Systems

In Figure 5.7 (page 331) the basis vectors of a coordinate system are rotated by a fixed angle  $\alpha$  to obtain the new basis vectors. Then the components of a vectors are transformed according the transformation rules below.

$$\begin{aligned}\begin{pmatrix} x' \\ y' \end{pmatrix} &= \mathbf{R}^T \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \\ \begin{pmatrix} x \\ y \end{pmatrix} &= \mathbf{R} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix}\end{aligned}$$

### 5.5.2 Zero Order Tensors: Scalars

A scalar function  $u(x, y)$  determines one scalar value at given points in space. It might be given in the original or the transformed system, the resulting values have to coincide. Scalars are **invariant** under coordinate transformations.

$$u'(x', y') = u(x, y).$$

Examples of scalars include: temperature, hydrostatic pressure, density, concentration. Observe that not all expressions leading to a number are invariant under transformations. As an example consider the partial derivative of a scalar with respect to the first coordinate, i.e.  $\frac{\partial f}{\partial x_1}$ . The transformation rule for this expression will be examined below.

### 5.5.3 First Order Tensors: Vectors

To determine a vector  $\vec{u}$  two numbers are required  $\vec{u} = (u, v)^T$ . In the new coordinate system the same vector is given by  $\vec{u}' = (u', v')^T$ . The transformation needs to satisfy the property

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \mathbf{R}^T \cdot \begin{pmatrix} u \\ v \end{pmatrix} .$$

**5–25 Example :** Well known and often used examples of vectors are position vectors, velocity and force vectors. ◇

### 5–26 Example : Gradient as first order tensor

The gradient of a scalar  $f$  is a first order tensor. This is a consequence of the chain rule.

$$\begin{aligned} f'(x', y') &= f(x, y) = f(x' \cos \alpha - y' \sin \alpha, x' \sin \alpha + y' \cos \alpha) \\ \frac{\partial}{\partial x'} f'(x', y') &= +\frac{\partial f}{\partial x} \cos \alpha + \frac{\partial f}{\partial y} \sin \alpha \\ \frac{\partial}{\partial y'} f'(x', y') &= -\frac{\partial f}{\partial x} \sin \alpha + \frac{\partial f}{\partial y} \cos \alpha \end{aligned}$$

and thus find

$$\begin{pmatrix} \frac{\partial}{\partial x'} f' \\ \frac{\partial}{\partial y'} f' \end{pmatrix} = \begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix} \cdot \begin{pmatrix} \frac{\partial}{\partial x} f \\ \frac{\partial}{\partial y} f \end{pmatrix} = \mathbf{R}^T \cdot \begin{pmatrix} \frac{\partial}{\partial x} f \\ \frac{\partial}{\partial y} f \end{pmatrix} .$$

The gradient is often used as a row vector and thus transpose the above identity to conclude

$$\left( \frac{\partial}{\partial x'} f', \frac{\partial}{\partial y'} f' \right) = \left( \frac{\partial}{\partial x} f, \frac{\partial}{\partial y} f \right) \cdot \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} = \left( \frac{\partial}{\partial x} f, \frac{\partial}{\partial y} f \right) \cdot \mathbf{R} .$$

◇

Observe that not all pairs of scalar expressions transform according to a first order tensor. As examples consider stress and strain. The transformation rules for

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \sigma_x \\ \sigma_y \end{pmatrix}$$

will be examined below.

### 5.5.4 Second Order Tensors: some Matrices

A second order tensor  $\mathbf{A}$  requires 4 components, conveniently arranged in the form of a  $2 \times 2$ -matrix.

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$$

When a new coordinate system is introduced the required transformation rule, for  $\mathbf{A}$  to be a tensor, is

$$\begin{aligned} \begin{bmatrix} a'_{1,1} & a'_{1,2} \\ a'_{2,1} & a'_{2,2} \end{bmatrix} &= \begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix} \cdot \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \cdot \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} . \\ \mathbf{A}' &= \mathbf{R}^T \cdot \mathbf{A} \cdot \mathbf{R} \end{aligned}$$

To decide whether a  $2 \times 2$  matrix is a tensor this transformation rule has to be verified. When all details are carried out this leads to the following formula, where  $C = \cos \alpha$  and  $S = \sin \alpha$ .

$$\begin{aligned} \begin{bmatrix} a'_{1,1} & a'_{1,2} \\ a'_{2,1} & a'_{2,2} \end{bmatrix} &= \begin{bmatrix} C & S \\ -S & C \end{bmatrix} \cdot \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \cdot \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \\ &= \begin{bmatrix} C & S \\ -S & C \end{bmatrix} \cdot \begin{bmatrix} a_{1,1}C + a_{1,2}S & -a_{1,1}S + a_{1,2}C \\ a_{2,1}C + a_{2,2}S & -a_{2,1}S + a_{2,2}C \end{bmatrix} \\ &= \begin{bmatrix} a_{1,1}C^2 + (a_{1,2} + a_{2,1})CS + a_{2,2}S^2 & (-a_{1,1} + a_{2,2})SC + a_{1,2}C^2 - a_{2,1}S^2 \\ (-a_{1,1} + a_{2,2})SC - a_{1,2}S^2 + a_{2,1}C^2 & a_{1,1}S^2 - (a_{1,2} + a_{2,1})CS + a_{2,2}C^2 \end{bmatrix} \end{aligned}$$

### 5–27 Example : Stress and strain as tensors

According to Result 5–20 the components of the strain are transformed by

$$\begin{bmatrix} \varepsilon'_{x'x'} & \varepsilon'_{x'y'} \\ \varepsilon'_{x'y'} & \varepsilon'_{y'y'} \end{bmatrix} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} \cdot \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}$$

and based on equation (5.10) we find for the stress

$$\begin{bmatrix} \sigma_{x'} & \tau_{x'y'} \\ \tau_{x'y'} & \sigma_{y'} \end{bmatrix} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \cdot \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} .$$

Thus stress and strain, as examined in the previous sections, are second order tensors.  $\diamond$

### 5–28 Example : Linear mapping as tensor

A linear mapping from  $\mathbb{R}^2$  to  $\mathbb{R}^2$  is completely determined by the image of the basis vectors, see Section 3.2. Use the components of these images as columns for a matrix  $\mathbf{A}$ , then the action of the linear mapping can be described by a matrix multiplication.

$$\begin{pmatrix} x \\ y \end{pmatrix} \longrightarrow \mathbf{A} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{e.g.} \quad \begin{pmatrix} x \\ y \end{pmatrix} \longrightarrow \begin{bmatrix} 2 & 0.5 \\ 1 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} .$$

Find a numerical example and the corresponding picture in Figure 5.13.

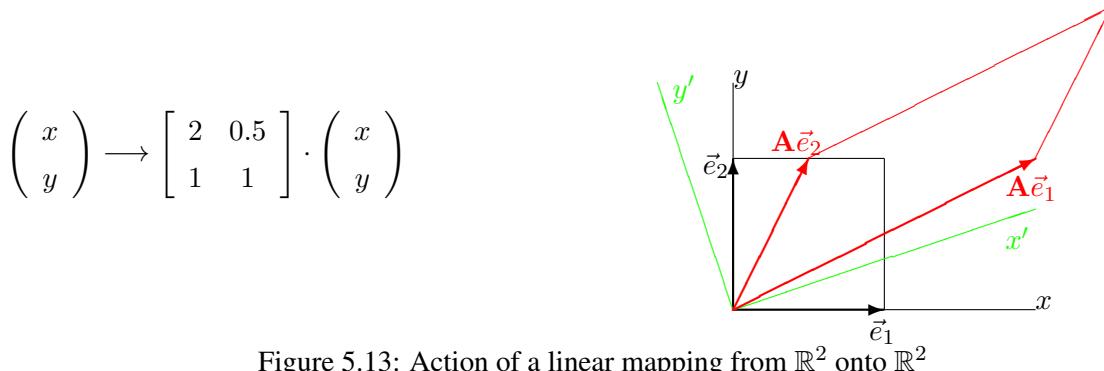
If the same linear mapping is to be examined in a new coordinate system  $(x', y')$  the matrix will obviously change. To determine this new matrix  $\mathbf{A}'$  the following steps have to be performed:

1. Determine the original components  $x, y$  based on the new components  $x'$  and  $y'$ .
2. Compute the original components of the image by multiplying with the matrix  $\mathbf{A}$ .
3. Determine the new components of the image.

This leads to

$$\begin{pmatrix} x' \\ y' \end{pmatrix} \longrightarrow \mathbf{A}' \cdot \begin{pmatrix} x' \\ y' \end{pmatrix} = \mathbf{R}^T \cdot \mathbf{A} \cdot \mathbf{R} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix} .$$

Thus linear mappings can be considered second order tensors.  $\diamond$

Figure 5.13: Action of a linear mapping from  $\mathbb{R}^2$  onto  $\mathbb{R}^2$ **5–29 Example : Displacement gradient tensor**

Use Figure 5.6 (page 327) to conclude

$$\begin{pmatrix} \Delta u_1 \\ \Delta u_2 \end{pmatrix} = \begin{pmatrix} u_1(x + \Delta x, y + \Delta y) \\ u_2(x + \Delta x, y + \Delta y) \end{pmatrix} - \begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} \approx \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial x} & \frac{\partial u_2}{\partial y} \end{bmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \mathbf{DU} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}.$$

**DU** is the **displacement gradient tensor**. Now examine this matrix if generated using a rotated coordinate system. Since the above is a linear mapping we can use the previous results and conclude that the transformation rule

$$\mathbf{DU}' = \mathbf{R}^T \mathbf{DU} \mathbf{R} \quad (5.11)$$

is correct, i.e. the displacement gradient is a tensor of order 2 .  $\diamond$

**5–30 Example : Second order partial derivatives**

The computation for the gradient in Example 5–26 can be extended to

$$\begin{aligned} f'(x', y') &= f(x' \cos \alpha - y' \sin \alpha, x' \sin \alpha + y' \cos \alpha) \\ \frac{\partial}{\partial x'} f'(x', y') &= +\frac{\partial f}{\partial x} \cos \alpha + \frac{\partial f}{\partial y} \sin \alpha \\ \frac{\partial}{\partial y'} f'(x', y') &= -\frac{\partial f}{\partial x} \sin \alpha + \frac{\partial f}{\partial y} \cos \alpha \\ \frac{\partial^2}{\partial x'^2} f'(x', y') &= +\left(+\frac{\partial^2 f}{\partial x^2} \cos \alpha + \frac{\partial^2 f}{\partial x \partial y} \sin \alpha\right) \cos \alpha + \left(+\frac{\partial^2 f}{\partial x \partial y} \cos \alpha + \frac{\partial^2 f}{\partial y^2} \sin \alpha\right) \sin \alpha \\ \frac{\partial^2}{\partial x' \partial y'} f'(x', y') &= +\left(-\frac{\partial^2 f}{\partial x^2} \sin \alpha + \frac{\partial^2 f}{\partial x \partial y} \cos \alpha\right) \cos \alpha + \left(-\frac{\partial^2 f}{\partial x \partial y} \sin \alpha + \frac{\partial^2 f}{\partial y^2} \cos \alpha\right) \sin \alpha \\ \frac{\partial^2}{\partial y'^2} f'(x', y') &= -\left(-\frac{\partial^2 f}{\partial x^2} \sin \alpha + \frac{\partial^2 f}{\partial x \partial y} \cos \alpha\right) \sin \alpha + \left(-\frac{\partial^2 f}{\partial x \partial y} \sin \alpha + \frac{\partial^2 f}{\partial y^2} \cos \alpha\right) \cos \alpha \end{aligned}$$

and thus the symmetric Hesse matrix of second order partial derivatives satisfies

$$\begin{bmatrix} \frac{\partial^2 f'}{\partial x'^2} & \frac{\partial^2 f'}{\partial x' \partial y'} \\ \frac{\partial^2 f'}{\partial x' \partial y'} & \frac{\partial^2 f'}{\partial y'^2} \end{bmatrix} = \begin{bmatrix} +\cos \alpha & +\sin \alpha \\ -\sin \alpha & +\cos \alpha \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \cdot \begin{bmatrix} +\cos \alpha & -\sin \alpha \\ +\sin \alpha & +\cos \alpha \end{bmatrix}.$$

This implies that the matrix of second order derivatives satisfies the transformation rule of a second order tensor.  $\diamond$

More examples of second order tensors are given in [Aris62] or [BoriTara79].

## 5.6 Hooke's Law and Elastic Energy Density

Using the strains in Table 5.3 (page 335) and the stresses in Table 5.4 (page 338) formulate the basic connection between the geometric deformations (strain) and the resulting forces (stress). It is a basic physical law<sup>16</sup>, confirmed by many measurements. The shown formulation is valid as long as all stress and strains are small. For large strains we would have to enter the field of nonlinear elasticity. First steps are shown in Section 5.7, by listing a few nonlinear material laws.

### 5.6.1 Hooke's Law

This is the general form of **Hooke's law** for a homogeneous (independent on position), isotropic (independent on direction) materials. This law is the foundation of linear elasticity and any book on elasticity will show a formulation, e.g. [Prze68, §2.2]<sup>17</sup>, [Sout73, §2.7], or [Wein74, §10.1].

$$\begin{aligned} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} &= \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \cdot \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \\ \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} &= \frac{1+\nu}{E} \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix} \end{aligned} \quad (5.12)$$

or by inverting the matrix

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & & & \\ \nu & 1-\nu & \nu & & & 0 \\ \nu & \nu & 1-\nu & & & \\ & & & 1-2\nu & & \\ 0 & & & & 1-2\nu & \\ & & & & & 1-2\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix}. \quad (5.13)$$

With the obvious notation equation (5.13) may be written in the form

$$\vec{\sigma} = \mathbf{H} \cdot \vec{\varepsilon}.$$

Observe that the equations decouple and we can equivalently write

$$\begin{aligned} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} &= \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix}, \\ \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix} &= \frac{E}{(1+\nu)} \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix}. \end{aligned} \quad (5.14)$$

<sup>16</sup>One can verify that for homogeneous, isotropic materials a linear law must have this form, e.g. [Sege77]

<sup>17</sup>The missing factors 2 are due to the different definition of the shear strains.

### 5.6.2 Hooke's law for Incompressible Materials

Observe that Hooke's law in this form can not be used for incompressible materials, i.e.  $\nu = \frac{1}{2}$ . Examine

$$\sigma_x + \sigma_y + \sigma_z = \frac{E}{1 - 2\nu} (\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz})$$

to conclude that  $\nu = \frac{1}{2}$  leads to  $\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz} = 0$ . This is a constraint to be satisfied for incompressible materials. Equation (5.12) can also be written in the form

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} = \frac{1 + \nu}{E} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} - \frac{\nu}{E} (\sigma_x + \sigma_y + \sigma_z) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

and in this form it is easy to read off that

$$\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz} = \frac{1 - 2\nu}{E} (\sigma_x + \sigma_y + \sigma_z).$$

Using Hooke's law in the form of equation (5.12) for  $\nu = \frac{1}{2}$  leads to

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix} \cdot \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} \quad (5.15)$$

and the determinant of this matrix equals zero. Thus one can not determine the inverse matrix.

### 5.6.3 Elastic Energy Density

The elastic energy density will be rewritten in many different forms, suitable for different interpretations. This is useful when extending the material properties to nonlinear laws, e.g. [Bowe10, §3.5.3, §3.5.5]. In this section many tedious, algebraic steps are spelled out with intermediate steps, even if they are trivial. This should be helpful when using literature on linear and nonlinear elasticity.

To start out we want to find the formula for the **energy density** of a deformed body. For this we consider a small block with width  $\Delta x$ , height  $\Delta y$  and depth  $\Delta z$ , located at the fixed origin. For a fixed displacement vector  $\vec{u}$  of the corner  $P = (\Delta x, \Delta y, \Delta z)$  we deform the block by a sequence of affine deformations, such that point  $P$  moves along straight lines. The displacement vector of  $P$  is given by the formula  $t \vec{u}$  where the parameter  $t$  varies from 0 to 1. If the final strain is denoted by  $\vec{\varepsilon}$  then the strains during the deformation are given by  $t \vec{\varepsilon}$ . Accordingly the stresses are given by  $t \vec{\sigma}$  where the final stress  $\vec{\sigma}$  can be computed by Hooke's law (e.g. equation (5.14)). Now we compute the total work needed to deform this block, using the basic formula  $\text{work} = \text{force} \cdot \text{distance}$ . There are six contributions:

- The face  $x = \Delta x$  moves from  $\Delta x$  to  $\Delta x(1 + \varepsilon_{xx})$ . For a parameter step  $dt$  at  $0 < t < 1$  the traveled distance is thus  $\varepsilon_{xx} \Delta x dt$ . The force is determined by the area  $\Delta y \cdot \Delta z$  and the normal stress  $t \sigma_x$ , where  $\sigma_x$  is the stress in the final position  $t = 1$ . The first energy contribution can now be integrated by

$$\int_0^1 (\Delta y \cdot \Delta z \cdot \sigma_x) \cdot t \varepsilon_{xx} \Delta x dt = \Delta y \cdot \Delta z \cdot \Delta x \cdot \sigma_x \cdot \varepsilon_{xx} \int_0^1 t dt = \frac{1}{2} \Delta V \cdot \sigma_x \cdot \varepsilon_{xx}.$$

- Similarly normal displacement of the faces at  $y = \Delta y$  and  $z = \Delta z$  lead to contributions

$$\frac{1}{2} \Delta V \cdot \sigma_y \cdot \varepsilon_{yy} \quad \text{and} \quad \frac{1}{2} \Delta V \cdot \sigma_z \cdot \varepsilon_{zz}.$$

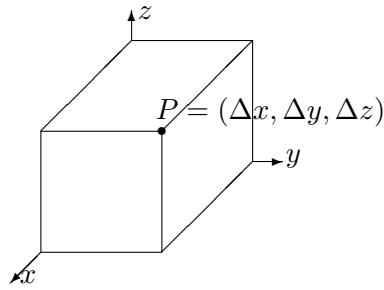


Figure 5.14: Block to be deformed, used to determine the elastic energy density  $W$

- To examine the situation of pure shearing strain observe that the face at  $x = \Delta x$  is moved by  $\frac{\partial u_2}{\partial x} \Delta x$  in  $y$ -direction. The shear stress in that direction is  $\tau_{xy}$  and thus the resulting energy

$$\frac{1}{2} \frac{\partial u_2}{\partial x} \Delta x \cdot \tau_{xy} \Delta y \Delta z$$

The face at  $y = \Delta y$  is moved by  $\frac{\partial u_1}{\partial y} \Delta y$  in  $x$ -direction. The shear stress in that direction is  $\tau_{xy}$ , leading to an energy contribution

$$\frac{1}{2} \frac{\partial u_1}{\partial y} \Delta y \cdot \tau_{xy} \Delta x \Delta z$$

Adding these two leads to a contribution to the energy due to shearing in the  $xy$  plane.

$$\frac{1}{2} \left( \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} \right) \tau_{xy} \Delta x \Delta y \Delta z = \varepsilon_{xy} \tau_{xy} \Delta V$$

The similar contributions due to shearing in the  $xz$  and  $yz$  planes lead to energies  $\varepsilon_{xz} \tau_{xz} \Delta V$  and  $\varepsilon_{yz} \tau_{yz} \Delta V$ .

Adding all six contributions and then dividing by the volume  $\Delta V$  we obtain the energy density

$$W = \frac{\text{energy}}{\Delta V} = \frac{1}{2} (\sigma_x \varepsilon_{xx} + \sigma_y \varepsilon_{yy} + \sigma_z \varepsilon_{zz} + 2 \tau_{xy} \varepsilon_{xy} + 2 \tau_{xz} \varepsilon_{xz} + 2 \tau_{yz} \varepsilon_{yz}) .$$

This can be written as scalar product in the form

$$W = \frac{1}{2} \left\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle + \left\langle \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \right\rangle \quad (5.16)$$

or according to Hooke's law in the form of equation (5.14) also as

$$W = \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle + \frac{E}{(1+\nu)} \left\langle \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \right\rangle . \quad (5.17)$$

The above formula for the energy density  $W$  can be written in the form  $W = \frac{1}{2} \langle \vec{\varepsilon}, \mathbf{A} \vec{\varepsilon} \rangle$  with a suitably chosen symmetric matrix  $\mathbf{A}$ .

$$\mathbf{A} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \\ & & 2(1-2\nu) \\ 0 & & 2(1-2\nu) \\ & & 2(1-2\nu) \end{bmatrix}$$

The gradient of this expression is given by  $\nabla \langle \mathbf{A} \vec{\varepsilon}, \vec{\varepsilon} \rangle = 2 \mathbf{A} \vec{\varepsilon}$  and thus conclude

$$\sigma_x = \frac{\partial W}{\partial \varepsilon_{xx}}, \quad \sigma_y = \frac{\partial W}{\partial \varepsilon_{yy}}, \quad \sigma_z = \frac{\partial W}{\partial \varepsilon_{zz}},$$

and

$$\tau_{xy} = \frac{1}{2} \frac{\partial W}{\partial \varepsilon_{xy}}, \quad \tau_{xz} = \frac{1}{2} \frac{\partial W}{\partial \varepsilon_{xz}}, \quad \tau_{yz} = \frac{1}{2} \frac{\partial W}{\partial \varepsilon_{yz}} = \frac{E}{1+\nu} \varepsilon_{yz}.$$

Thus obtain normal and shear stresses as partial derivatives of the energy density  $W$  with respect to the normal and shear strains.

### Energy density as function of invariant strain expressions

The energy density is expressed in terms of the invariant expression on page 335. To achieve this goal first generate a few invariant strain expressions. Since

$$\mathbf{S}'^2 = \mathbf{S}' \mathbf{S}' = \mathbf{R}^T \mathbf{S} \mathbf{R} \mathbf{R}^T \mathbf{S} \mathbf{R} = \mathbf{R}^T \mathbf{S} \mathbf{S} \mathbf{R}$$

use that the matrix  $\mathbf{S}^2$  is a tensor too, and this leads to more invariant expressions.

$$\begin{aligned} \mathbf{S} &= \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{xy} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{xz} & \varepsilon_{yz} & \varepsilon_{zz} \end{bmatrix} \\ I_1 &= \text{trace}(\mathbf{S}) = \varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz} \\ I_1^2 &= \text{trace}(\mathbf{S})^2 = \varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2 + 2\varepsilon_{xx}\varepsilon_{yy} + 2\varepsilon_{xx}\varepsilon_{zz} + 2\varepsilon_{yy}\varepsilon_{zz} \\ \mathbf{S}^2 &= \begin{bmatrix} \varepsilon_{xx}^2 + \varepsilon_{xy}^2 + \varepsilon_{xz}^2 & \cdot & \cdot \\ \cdot & \varepsilon_{xy}^2 + \varepsilon_{yy}^2 + \varepsilon_{yz}^2 & \cdot \\ \cdot & \cdot & \varepsilon_{xz}^2 + \varepsilon_{yz}^2 + \varepsilon_{zz}^2 \end{bmatrix} \\ I_2 &= \varepsilon_{yy}\varepsilon_{zz} - \varepsilon_{yz}^2 + \varepsilon_{xx}\varepsilon_{zz} - \varepsilon_{xz}^2 + \varepsilon_{xx}\varepsilon_{yy} - \varepsilon_{xy}^2 = \frac{1}{2} (\text{trace}(\mathbf{S})^2 - \text{trace}(\mathbf{S}^2)) \\ I_3 &= \det(\mathbf{S}) \end{aligned}$$

Elementary algebra leads to

$$\begin{aligned} I_4 &= I_1^2 - 2I_2 = \varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2 + 2\varepsilon_{xy}^2 + 2\varepsilon_{xz}^2 + 2\varepsilon_{yz}^2 \\ \frac{2(1+\nu)(1-2\nu)}{E} W &= (1-\nu)(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2) + 2\nu(\varepsilon_{xx}\varepsilon_{yy} + \varepsilon_{xx}\varepsilon_{zz} + \varepsilon_{yy}\varepsilon_{zz}) \\ &\quad + 2(1-2\nu)(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) \\ &= (1-\nu)(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2 + 2\varepsilon_{xy}^2 + 2\varepsilon_{xz}^2 + 2\varepsilon_{yz}^2) \end{aligned}$$

$$\begin{aligned}
& + 2\nu (\varepsilon_{xx}\varepsilon_{yy} + \varepsilon_{xx}\varepsilon_{zz} + \varepsilon_{yy}\varepsilon_{zz} - \varepsilon_{xy}^2 - \varepsilon_{xz}^2 - \varepsilon_{yz}^2) \\
& = (1-\nu)(I_1^2 - 2I_2) + 2\nu I_2 = (1-\nu)I_1^2 - 2(1-2\nu)I_2 \\
W & = \frac{E(1-\nu)}{2(1+\nu)(1-2\nu)} I_1^2 - \frac{E}{1+\nu} I_2
\end{aligned}$$

and thus conclude that the energy density  $W$  is invariant under rotations, as it should be.

The above expression can also be written in terms of the principal strains. Using

$$I_2 = \varepsilon_{11}\varepsilon_{22} + \varepsilon_{11}\varepsilon_{33} + \varepsilon_{22}\varepsilon_{33} \quad \text{and} \quad I_4 = \varepsilon_{11}^2 + \varepsilon_{22}^2 + \varepsilon_{33}^2$$

find

$$\begin{aligned}
W & = \frac{E}{2(1+\nu)(1-2\nu)} ((1-\nu)I_4 + 2\nu I_2) \\
& = \frac{E}{2(1+\nu)(1-2\nu)} ((1-\nu)I_1^2 - 2(1-2\nu)I_2) \\
& = \frac{E}{2(1+\nu)(1-2\nu)} ((1-\nu)(\varepsilon_{11}^2 + \varepsilon_{22}^2 + \varepsilon_{33}^2) + 2\nu(\varepsilon_{11}\varepsilon_{22} + \varepsilon_{11}\varepsilon_{33} + \varepsilon_{22}\varepsilon_{33})) .
\end{aligned}$$

Observe that different expressions are available for the energy density  $W$  as function of the invariants.

### Volume changing and shape changing energy

There are many more invariant expressions. In [Bowe10, p. 89] find

$$\begin{aligned}
I_5 & = I_4 - \frac{1}{3}I_1^2 \\
& = \varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2 + 2(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) - \frac{1}{3}(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2 + 2\varepsilon_{xx}\varepsilon_{yy} + 2\varepsilon_{xx}\varepsilon_{zz} + 2\varepsilon_{yy}\varepsilon_{zz}) \\
3I_5 & = 2\varepsilon_{xx}^2 + 2\varepsilon_{yy}^2 + 2\varepsilon_{zz}^2 - 2\varepsilon_{xx}\varepsilon_{yy} - 2\varepsilon_{xx}\varepsilon_{zz} - 2\varepsilon_{yy}\varepsilon_{zz} + 6(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) \\
& = (\varepsilon_{xx} - \varepsilon_{yy})^2 + (\varepsilon_{xx} - \varepsilon_{zz})^2 + (\varepsilon_{yy} - \varepsilon_{zz})^2 + 6(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) .
\end{aligned}$$

This invariant corresponds to the von Mises stress  $\sigma_M$  on page 340, but formulated with strains instead of stresses. Expressed in principal strains find

$$3I_5 = (\varepsilon_{11} - \varepsilon_{22})^2 + (\varepsilon_{11} - \varepsilon_{33})^2 + (\varepsilon_{22} - \varepsilon_{33})^2 .$$

For the energy density this leads to (using elementary, tedious algebra)

$$\begin{aligned}
I_6 & = \frac{1+\nu}{3}I_1^2 + (1-2\nu)I_5 \\
& = \frac{1+\nu}{3}(\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz})^2 + \frac{1-2\nu}{3}((\varepsilon_{xx} - \varepsilon_{yy})^2 + (\varepsilon_{xx} - \varepsilon_{zz})^2 + (\varepsilon_{yy} - \varepsilon_{zz})^2) \\
& \quad + \frac{1-2\nu}{3}6(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) \\
& = \frac{1+\nu}{3}(\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz})^2 + \frac{1-2\nu}{3}((\varepsilon_{xx} - \varepsilon_{yy})^2 + (\varepsilon_{xx} - \varepsilon_{zz})^2 + (\varepsilon_{yy} - \varepsilon_{zz})^2) \\
& \quad + 2(1-2\nu)(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) \\
& = \frac{1+\nu+2(1-2\nu)}{3}(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2) + \frac{2(1+\nu)-2(1-2\nu)}{3}(\varepsilon_{xx}\varepsilon_{yy} + \varepsilon_{xx}\varepsilon_{zz} + \varepsilon_{yy}\varepsilon_{zz}) \\
& \quad + 2(1-2\nu)(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) \\
& = (1-\nu)(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2) + 2\nu(\varepsilon_{xx}\varepsilon_{yy} + \varepsilon_{xx}\varepsilon_{zz} + \varepsilon_{yy}\varepsilon_{zz}) + 2(1-2\nu)(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) \\
& = \frac{2(1+\nu)(1-2\nu)}{E}W .
\end{aligned}$$

Thus write the energy density  $W$  as sum of two contributions

$$W = W_{vol} + W_{shape} = \frac{E}{6(1-2\nu)} I_1^2 + \frac{E}{2(1+\nu)} I_5.$$

- a volume changing contribution, caused by hydrostatic pressure:

$$W_{vol} = \frac{E}{6(1-2\nu)} (\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz})^2 = \frac{E}{6(1-2\nu)} (\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33})^2$$

- and a shape changing contribution, caused by shearing:

$$\begin{aligned} W_{shape} &= \frac{E}{6(1+\nu)} ((\varepsilon_{xx} - \varepsilon_{yy})^2 + (\varepsilon_{xx} - \varepsilon_{zz})^2 + (\varepsilon_{yy} - \varepsilon_{zz})^2 + 6(\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2)) \\ &= \frac{E}{6(1+\nu)} ((\varepsilon_{11} - \varepsilon_{22})^2 + (\varepsilon_{11} - \varepsilon_{33})^2 + (\varepsilon_{22} - \varepsilon_{33})^2) \end{aligned}$$

- The above can also be expressed using the shear modulus  $\mu$  and the bulk modulus  $K$ , see Example 5–35 and Table 5.5 (page 360).

$$\begin{aligned} W &= \frac{E}{6(1-2\nu)} (\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33})^2 + \frac{E}{6(1+\nu)} ((\varepsilon_{11} - \varepsilon_{22})^2 + (\varepsilon_{11} - \varepsilon_{33})^2 + (\varepsilon_{22} - \varepsilon_{33})^2) \\ &= \frac{K}{2} (\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33})^2 + \frac{\mu}{3} ((\varepsilon_{11} - \varepsilon_{22})^2 + (\varepsilon_{11} - \varepsilon_{33})^2 + (\varepsilon_{22} - \varepsilon_{33})^2) \\ &= W_{vol} + W_{shape} \end{aligned}$$

The above is a starting point for nonlinear material laws, e.g. hyperelastic materials as described in [Bowe10, §3.3.3]. This approach allows to distinguish shear stress and hydrostatic stress. In principle the energy density  $W$  can be any function of invariants, e.g.  $W = f(I_1, I_5)$ . The validity of the model has to be justified by experiments or by other arguments. As examples find in the FEM software COMSOL the nonlinear material models Neo–Hookean, Mooney–Rivlin and Murnaghan, amongst others.

### Energy density as function of stress

In the above sections the energy density was expressed in terms of strain. We can combine (5.16) and Hooke's law in the form (5.12) to arrive at

$$\begin{aligned} W &= \frac{1}{2E} \langle \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} \rangle + \frac{1+\nu}{E} \langle \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix}, \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix} \rangle \\ &= \frac{1}{2E} (\sigma_x^2 + \sigma_y^2 + \sigma_z^2 - 2\nu(\sigma_x\sigma_y + \sigma_x\sigma_z + \sigma_y\sigma_z)) + \frac{1+\nu}{E} (\tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2). \end{aligned}$$

Since invariants of the stress tensor are given by

$$\begin{aligned} I_1 &= \sigma_x + \sigma_y + \sigma_z \\ I_2 &= \sigma_y\sigma_z - \tau_{yz}^2 + \sigma_x\sigma_z - \tau_{xz}^2 + \sigma_x\sigma_y - \tau_{xy}^2 \\ I_4 = I_1^2 - 2I_2 &= \sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2 + 2\tau_{xy}^2 + 2\tau_{xz}^2 + 2\tau_{yz}^2 \end{aligned}$$

we conclude

$$\begin{aligned} W &= \frac{1}{2E} (\sigma_x^2 + \sigma_y^2 + \sigma_z^2) - \frac{\nu}{E} (\sigma_x\sigma_y + \sigma_x\sigma_z + \sigma_y\sigma_z) + \frac{1+\nu}{E} (\tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2) \\ &= \frac{1}{2E} (\sigma_x^2 + \sigma_y^2 + \sigma_z^2 + 2\tau_{xy}^2 + 2\tau_{xz}^2 + 2\tau_{yz}^2) - \frac{\nu}{E} (\sigma_x\sigma_y + \sigma_x\sigma_z + \sigma_y\sigma_z - \tau_{xy}^2 - \tau_{xz}^2 - \tau_{yz}^2) \\ &= \frac{1}{2E} I_4 - \frac{\nu}{E} I_2. \end{aligned}$$

In terms of principal stresses this leads to the energy density

$$W = \frac{1}{2E} (\sigma_1^2 + \sigma_2^2 + \sigma_3^2) - \frac{\nu}{E} (\sigma_1\sigma_2 + \sigma_1\sigma_3 + \sigma_2\sigma_3).$$

### 5.6.4 Volume and Surface Forces, the Bernoulli Principle

The previous section presented expressions for the elastic energy stored in a deformed solid. When using calculus of variations (leading to FEM algorithms) we also need to take external forces into account. This is best done by introducing matching potentials energies, representing volume and surface forces.

#### Volume forces

A force applied to the volume of the solid can be introduced by means of a volume force density  $\vec{f}$  (units:  $\frac{\text{N}}{\text{m}^3}$ ). By adding the potential energy

$$U_{Vol} = - \iiint_{\Omega} \vec{f} \cdot \vec{u} \, dV$$

to the elastic energy and then minimizing we are lead to the correct force term.

As an example consider the weight (given by the density  $\rho$ ) of the solid and the resulting gravitational force. This leads to a force density of

$$\vec{f} = \begin{pmatrix} 0 \\ 0 \\ -\rho g \end{pmatrix}$$

and thus find the corresponding potential energy as

$$U_{Vol} = + \iiint_{\Omega} \rho g u_3 \, dV.$$

This potential energy decreases if the solid is moved downwards.

#### Surface forces

By adding a surface potential energy, using the surface force density  $\vec{g}$  (units:  $\frac{\text{N}}{\text{m}^2}$ ),

$$U_{Surf} = - \iint_{\partial\Omega} \vec{g} \cdot \vec{u} \, dA$$

examine forces applied to the surface only.

As an example consider a constant surface pressure  $p$  on the surface of the solid. This surface force density is

$$\vec{g} = -p \vec{n}$$

where  $\vec{n}$  is the outer unit normal vector. Thus find the corresponding potential energy as

$$U_{Surf} = + \iint_{\partial\Omega} p \vec{u} \cdot \vec{n} \, dA.$$

This potential energy decreases if the solid is compressed, i.e.  $\vec{u} \cdot \vec{n} < 0$ .

#### The Bernoulli Principle

Using the above we can now give a formula for the total energy in a deformed solid.

$$U(\vec{u}) = U_{elast} + U_{Vol} + U_{Surf} \quad (5.18)$$

$$\begin{aligned}
&= \iiint_{\Omega} \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle dV + \\
&\quad + \iiint_{\Omega} \frac{E}{1+\nu} \left\langle \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \right\rangle dV - \iiint_{\Omega} \vec{f} \cdot \vec{u} dV - \iint_{\partial\Omega} \vec{g} \cdot \vec{u} dA.
\end{aligned}$$

As in many other situations we use again a principle of least energy to find the equilibrium state of a deformed solid. This result is known under the name Bernoulli<sup>18</sup> principle.

If the above solid is in equilibrium, then the displacement function  $\vec{u}$  is a minimizer of the above energy functional, subject to the given boundary conditions.

This is the basis for a finite element (FEM) solution to elasticity problems.

### 5.6.5 Some Exemplary Situations

In this section Hooke's law is illustrated by considering a few simple examples.

#### 5-31 Example : Hooke's basic law

Consider the situation in Figure 5.15 with the following assumptions:

- The solid of length  $L$  has constant cross section perpendicular to the  $x$  axis, with area  $A = \Delta y \cdot \Delta z$ .
- The left face is fixed in the  $x$  direction, but free to move in the other directions.
- The constant normal stress  $\sigma_x$  at the right face is given by  $\sigma_x = \frac{F}{A}$ .
- There are no forces in the  $y$  and  $z$  directions.

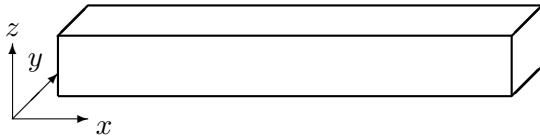


Figure 5.15: Situation for the basic version of Hooke's law

This leads to the consequences:

- All stresses in  $y$  and  $z$  direction vanish, i.e.

$$\sigma_y = \sigma_z = \tau_{xy} = \tau_{xz} = \tau_{yz} = 0.$$

<sup>18</sup>Proposed by Daniel Bernoulli (1700–1782), the son of Johann Bernoulli (1667–1748) and nephew of Jakob Bernoulli (1654–1705).

- Hooke's law (5.12) implies

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \cdot \begin{pmatrix} \frac{F}{A} \\ 0 \\ 0 \end{pmatrix} = \frac{1}{E} \frac{F}{A} \begin{pmatrix} 1 \\ -\nu \\ -\nu \end{pmatrix}.$$

- The first component of the above equations leads to the classical, basic form of Hooke's law

$$\varepsilon_{xx} = \frac{\Delta L}{L} = \frac{1}{E} \frac{F}{A}.$$

This is the standard definition of Young's **modulus of elasticity**. The solid is stretched by a factor  $(1 + \frac{1}{E} \frac{F}{A})$ .

- In the  $y$  and  $z$  direction the solid is contracted by a factor of  $(1 - \frac{\nu}{E} \frac{F}{A})$ . This is an interpretation of **Poisson's ratio**  $\nu$ .

$$\varepsilon_{yy} = -\nu \varepsilon_{xx}$$

Multiply the relative change of length in the  $x$  direction by  $\nu$  to obtain the relative change of length in the  $y$  and  $z$  directions. One expects  $\nu \geq 0$ .

- The energy density  $W$  can be found by equation (5.17).

$$\begin{aligned} W &= \frac{1}{2} \left\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle + \left\langle \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \right\rangle \\ &= \frac{1}{2} \frac{1}{E} \left( \frac{F}{A} \right)^2 \left\langle \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -\nu \\ -\nu \end{pmatrix} \right\rangle + 0 = \frac{1}{2} \frac{1}{E} \left( \frac{F}{A} \right)^2 = \frac{1}{2} E \varepsilon_{xx}^2 \end{aligned}$$

- Assuming that we have the principal strain situation the energy density is given by

$$W = \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle$$

For the case of uniaxial loading in  $x$ -direction use  $\varepsilon_{yy} = \varepsilon_{zz}$  and find

$$\begin{aligned} W &= \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} ((1-\nu)(\varepsilon_{xx}^2 + 2\varepsilon_{yy}^2) + \nu(4\varepsilon_{xx}\varepsilon_{yy} + 2\varepsilon_{yy}^2)) \\ \frac{\partial W}{\partial \varepsilon_{xx}} &= \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} ((1-\nu)2\varepsilon_{xx} + \nu 4\varepsilon_{yy}) \\ &= \frac{E}{(1+\nu)(1-2\nu)} ((1-\nu)\varepsilon_{xx} + \nu 2\varepsilon_{yy}) \\ \frac{\partial W}{\partial \varepsilon_{yy}} &= \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} ((1-\nu)4\varepsilon_{yy} + \nu 4(\varepsilon_{xx} + \varepsilon_{yy})) \\ &= \frac{4}{2} \frac{E}{(1+\nu)(1-2\nu)} (\nu\varepsilon_{xx} + \varepsilon_{yy}) \end{aligned}$$

- Based on the Bernoulli principle the energy density  $W$  will be minimized with respect to  $\varepsilon_{yy}$ . Thus conclude  $\varepsilon_{yy} = -\nu\varepsilon_{xx}$ , i.e. obtain the Poisson contraction as consequence of minimizing the energy.

- Since the stress in  $x$ -direction is given by

$$\begin{aligned}\sigma_x = \frac{\partial W}{\partial \varepsilon_{xx}} &= \frac{E}{(1+\nu)(1-2\nu)} ((1-\nu)\varepsilon_{xx} + \nu 2\varepsilon_{yy})) \\ &= \frac{E}{(1+\nu)(1-2\nu)} ((1-\nu)\varepsilon_{xx} - \nu 2\nu\varepsilon_{xx}) = E\varepsilon_{xx}\end{aligned}$$

we also obtain Hooke's elementary law as consequence of minimizing the energy.

◊

### 5-32 Example : Solid under hydrostatic pressure

If a rectangular block is submitted to a constant pressure  $p$ , then we know all components of the stress (assuming they are constant throughout the solid), namely

$$\sigma_x = \sigma_y = \sigma_z = -p \quad \text{and} \quad \tau_{xy} = \tau_{xz} = \tau_{yz} = 0.$$

Submerging a solid deep into water will lead to this hydrostatic pressure situation. Hooke's law now implies

$$\varepsilon_{xy} = \varepsilon_{xz} = \varepsilon_{yz} = 0$$

and

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} = -\frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \cdot \begin{pmatrix} p \\ p \\ p \end{pmatrix} = -\frac{p(1-2\nu)}{E} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

i.e. in each direction the solid is compressed by a factor of  $1 - \frac{p(1-2\nu)}{E}$ . Since putting a solid under pressure will make it shrink, the Poisson's ratio must satisfy the condition  $0 \leq \nu \leq \frac{1}{2}$ . The case  $\nu = \frac{1}{2}$  corresponds to an incompressible object.

Since the length in each direction is multiplied by the same factor  $(1 + \varepsilon_{xx})$  determinr the relative change of volume by

$$\frac{\Delta V}{V} = \left(1 - \frac{p(1-2\nu)}{E}\right)^3 - 1 \approx -\frac{3(1-2\nu)}{E} p = -\frac{1}{K} p$$

if the pressure  $p$  is small. The appearing coefficient  $K = \frac{E}{3(1-2\nu)}$  is called the **bulk modulus** of the material.

The energy density  $W$  is given by

$$\begin{aligned}W &= \frac{1}{2} \left\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle + \left\langle \begin{pmatrix} \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \right\rangle \\ &= -\frac{1}{2} \frac{p(1-2\nu)}{E} \left\langle \begin{pmatrix} -p \\ -p \\ -p \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\rangle + 0 = \frac{1}{2} \frac{1-2\nu}{E} 3 p^2.\end{aligned}$$

This can also be expressed in terms of normal strains.

$$W = \frac{1}{2} \frac{3(1-2\nu)}{E} p^2 = \frac{1}{2} \frac{3E}{1-2\nu} \varepsilon_{xx}^2$$

Then the pressure is determined by

$$-p = \sigma_x = \frac{1}{3} \frac{\partial W}{\partial \varepsilon_{xx}} = \frac{E}{1-2\nu} \varepsilon_{xx}.$$

◊

### 5-33 Example : Shear modulus

To the block in Figure 5.14 apply a force of strength  $F$  in direction of the  $x$  axis to the top (area  $\Delta x \cdot \Delta y$ ). No normal forces in the  $y$  direction are applied. The corresponding forces have to be applied to the faces at  $x = 0$  and  $x = \Delta x$  for the block to be in equilibrium. No other forces apply, thus

$$\tau_{xz} = \frac{F}{A} \quad \text{and} \quad \tau_{xy} = \tau_{yz} = \sigma_x = \sigma_y = \sigma_z = 0.$$

Hooke's law (5.12) leads to

$$\varepsilon_{xx} = \varepsilon_{yy} = \varepsilon_{zz} = \varepsilon_{xy} = \varepsilon_{yz} = 0$$

and

$$\varepsilon_{xz} = \frac{1+\nu}{E} \frac{F}{A} = \frac{1}{2G} \frac{F}{A} = \frac{1}{2G} \tau_{xy}.$$

This is the reason why some presentations use the **shear modulus**  $G = \mu = \frac{E}{2(1+\nu)}$ . The energy density is given by

$$W = \tau_{xy} \varepsilon_{xy} = 2G \varepsilon_{xy}^2 = \frac{E}{1+\nu} \varepsilon_{xy}^2.$$

One can determine Young modulus  $E$  by a uniaxial loading and then measuring the shear modulus  $G = \frac{E}{2(1+\nu)}$  allows to determine Poisson's ratio  $\nu$ .

One can also start with the general formula for the energy density based on Hooke's law.

$$\begin{aligned} W &= \frac{E}{6(1-2\nu)} (\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz})^2 + \\ &\quad + \frac{E}{6(1+\nu)} ((\varepsilon_{xx} - \varepsilon_{yy})^2 + (\varepsilon_{xx} - \varepsilon_{zz})^2 + (\varepsilon_{yy} - \varepsilon_{zz})^2 + 6(\varepsilon_{xy}^2 + \varepsilon_{yz}^2 + \varepsilon_{xz}^2)). \end{aligned}$$

Use Bernoulli's principle to determine all strains. Minimizing this energy density leads to

$$0 = \frac{\partial W}{\partial \varepsilon_{yz}} = \frac{E}{1+\nu} 2\varepsilon_{yz} \quad \text{and} \quad 0 = \frac{\partial W}{\partial \varepsilon_{xz}} = \frac{E}{1+\nu} 2\varepsilon_{xz}$$

and thus the only nonzero shearing is  $\varepsilon_{xy}$ . Similarly

$$\begin{aligned} 0 &= \frac{\partial W}{\partial \varepsilon_{xx}} = \frac{E}{3(1-2\nu)} (\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}) + \frac{E}{3(1+\nu)} ((\varepsilon_{xx} - \varepsilon_{yy}) + (\varepsilon_{xx} - \varepsilon_{zz})) \\ 0 &= \frac{\partial W}{\partial \varepsilon_{yy}} = \frac{E}{3(1-2\nu)} (\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}) + \frac{E}{3(1+\nu)} ((-\varepsilon_{xx} + \varepsilon_{yy}) + (\varepsilon_{yy} - \varepsilon_{zz})) \\ 0 &= \frac{\partial W}{\partial \varepsilon_{zz}} = \frac{E}{3(1-2\nu)} (\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}) + \frac{E}{3(1+\nu)} ((-\varepsilon_{xx} + \varepsilon_{zz}) + (\varepsilon_{yy} - \varepsilon_{zz})) \end{aligned}$$

Then conlude

- The sum of the three equations leads to  $\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz} = 0$ .
- The sum of the first two equations leads to  $\varepsilon_{xx} + \varepsilon_{yy} = 2\varepsilon_{zz} = -2(\varepsilon_{xx} + \varepsilon_{yy})$ . This implies  $\varepsilon_{xx} + \varepsilon_{yy} = 0$  and thus  $\varepsilon_{zz} = 0$ .
- The sum of the first and third equation leads to  $\varepsilon_{xx} + \varepsilon_{zz} = 2\varepsilon_{yy} = -2(\varepsilon_{xx} + \varepsilon_{zz})$ . This implies  $\varepsilon_{xx} + \varepsilon_{zz} = 0$  and thus  $\varepsilon_{yy} = 0$ .
- Now  $\varepsilon_{xx} = 0$  is easy to see.

Thus the only non vanishing strain is  $\varepsilon_{xy}$ . ◊

**5–34 Example : Biaxial loading**

not in class

The situation of biaxial loading is given by

$$\sigma = \sigma_x = \sigma_y = \quad \text{and} \quad \sigma_z = \tau_{xy} = \tau_{xz} = \tau_{yz} = 0.$$

This is a situation with identical stress in two directions and no stress in the third direction and no shearing. Hooke's law leads to

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \cdot \begin{pmatrix} \sigma \\ \sigma \\ 0 \end{pmatrix} = \frac{\sigma}{E} \begin{pmatrix} 1-\nu \\ 1-\nu \\ -2\nu \end{pmatrix} = \begin{pmatrix} \varepsilon \\ \varepsilon \\ \varepsilon_{zz} \end{pmatrix}$$

or by solving for  $\sigma = \frac{E}{1-\nu} \varepsilon$ . This leads to the energy density

$$W = \frac{1}{2} \left\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle = \frac{E}{2(1-\nu)} \left\langle \begin{pmatrix} \varepsilon \\ \varepsilon \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon \\ \varepsilon \\ \varepsilon_{zz} \end{pmatrix} \right\rangle = \frac{E}{1-\nu} \varepsilon^2.$$

Hooke's law is confirmed, since by the partial derivative of the energy density  $W$  with respect to the strain  $\varepsilon$  find

$$\sigma = \frac{1}{2} \frac{\partial W}{\partial \varepsilon} = \frac{1}{2} \frac{\partial}{\partial \varepsilon} \left( \frac{E}{1-\nu} \varepsilon^2 \right) = \frac{E}{1-\nu} \varepsilon.$$

◇

**5–35 Example : A few different elastic moduli**

not in class

For homogeneous, isotropic materials there are many different ways to describe the linear laws of elasticity.

- Young's modulus  $E$
- Poisson's ratio  $\nu$
- Shear modulus  $G$  or  $\mu$
- Lamé's first parameter  $\lambda$
- Bulk modulus  $K$

Given any two of these the others are determined<sup>19</sup>, leading to Table 5.5.

◇

**5–36 Example : Principal stress and principal strain**

Since the strain tensor is symmetric select a coordinate system such that

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_{zz} \end{bmatrix} = \begin{bmatrix} \varepsilon_{xx} & 0 & 0 \\ 0 & \varepsilon_{yy} & 0 \\ 0 & 0 & \varepsilon_{zz} \end{bmatrix}$$

i.e. vanishing shear strains. Based on Hooke's law (5.14) conclude that the shear stresses vanish too, i.e.

$$\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} = \begin{bmatrix} \sigma_x & 0 & 0 \\ 0 & \sigma_y & 0 \\ 0 & 0 & \sigma_z \end{bmatrix}$$

<sup>19</sup>See [https://en.wikipedia.org/wiki/Elastic\\_modulus](https://en.wikipedia.org/wiki/Elastic_modulus).

Modul	Young's $E$	Poisson $\nu$	Shear $G = \mu$	Lamé $\lambda$	bulk $K$	notes
$(E, \nu)$	$E$	$\nu$	$\frac{E}{2(1+\nu)}$	$\frac{\nu E}{(1+\nu)(1-2\nu)}$	$\frac{E}{3(1-2\nu)}$	
$(E, G)$	$E$	$\frac{E}{2G} - 1$	$G$	$\frac{G(E-2G)}{3G-E}$	$\frac{EG}{3(3G-E)}$	
$(\nu, G)$	$2G(1+\nu)$	$\nu$	$G$	$\frac{2\nu G}{1-2\nu}$	$\frac{2G(1+\nu)}{3(1-2\nu)}$	
$(E, \lambda)$	$E$	$\frac{2\lambda}{E+\lambda+R}$	$\frac{E-3\lambda+R}{4}$	$\lambda$	$\frac{E+3\lambda+R}{6}$	$R = \sqrt{E^2 + 9\lambda^2 + 2\lambda E}$
$(\nu, \lambda)$	$\frac{\lambda(1+\nu)(1-2\nu)}{\nu}$	$\nu$	$\frac{\lambda(1-2\nu)}{2\nu}$	$\lambda$	$\frac{\lambda(1+\nu)}{3\nu}$	useless if $\nu = 0$
$(G, \lambda)$	$\frac{G(3\lambda+2G)}{\lambda+G}$	$\frac{\lambda}{2(\lambda+G)}$	$G$	$\lambda$	$\lambda + \frac{2G}{3}$	
$(E, K)$	$E$	$\frac{3K-E}{6K}$	$\frac{3EK}{9K-E}$	$\frac{3K(3K-E)}{9K-E}$	$K$	
$(\nu, K)$	$3K(1-2\nu)$	$\nu$	$\frac{3K(1-2\nu)}{2(1+\nu)}$	$\frac{3\nu K}{1+\nu}$	$K$	
$(G, K)$	$\frac{9GK}{3K+G}$	$\frac{3K-2G}{2(3K+G)}$	$G$	$K - \frac{2G}{3}$	$K$	
$(\lambda, K)$	$\frac{9K(K-\lambda)}{3K-\lambda}$	$\frac{\lambda}{3K-\lambda}$	$\frac{3(K-\lambda)}{2}$	$\lambda$	$K$	

Table 5.5: Elastic moduli and their relations

and find the resulting normal stresses

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} (1-\nu)\varepsilon_{xx} + \nu(\varepsilon_{yy} + \varepsilon_{zz}) \\ (1-\nu)\varepsilon_{yy} + \nu(\varepsilon_{xx} + \varepsilon_{zz}) \\ (1-\nu)\varepsilon_{zz} + \nu(\varepsilon_{xx} + \varepsilon_{yy}) \end{pmatrix}.$$

Using this, the elastic energy density is given by

$$\begin{aligned} W &= \frac{1}{2} \left\langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle \\ &= \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left( (1-\nu)(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2) + 2\nu(\varepsilon_{xx}\varepsilon_{yy} + \varepsilon_{yy}\varepsilon_{zz} + \varepsilon_{zz}\varepsilon_{xx}) \right). \end{aligned}$$

The deformed volume can be estimated by

$$V + \Delta V = V(1 + \varepsilon_{xx})(1 + \varepsilon_{yy})(1 + \varepsilon_{zz}) \approx V(1 + \varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz})$$

or

$$\Delta V \approx (\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}) V.$$

Using

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \cdot \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}$$

find

$$\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz} = \frac{1-2\nu}{E} (\sigma_x + \sigma_y + \sigma_z)$$

and

$$\frac{\Delta V}{V} \approx \frac{1-2\nu}{E} (\sigma_x + \sigma_y + \sigma_z).$$

This implies that the sum of the three principal stresses determines the volume change. Based on this consider the volume changing hydrostatic stress (pressure)

$$\sigma_h = \frac{1}{3} (\sigma_x + \sigma_y + \sigma_z)$$

and the shape changing stresses

$$\hat{\sigma}_x = \sigma_x - \sigma_h, \quad \hat{\sigma}_y = \sigma_y - \sigma_h, \quad \hat{\sigma}_z = \sigma_z - \sigma_h.$$



### 5-37 Example : Torsion of a tube

not in class

In this example examine the torsion of a circular, hollow shaft, as shown in Figure 5.16. Since the computations are lengthy the computational path shall be spelled out first.

- Express the displacement in terms of radial and angular contributions.
- Determine the normal and shear strains.
- Use Hooke's law to find the elastic energy density and by an integration find the total elastic energy.
- Use Euler–Lagrange equations to determine the boundary value problems for the radial and angular displacement functions and solve these.
- Use these solutions to determine the actual energy stored in the twisted tube and determine the required torque.

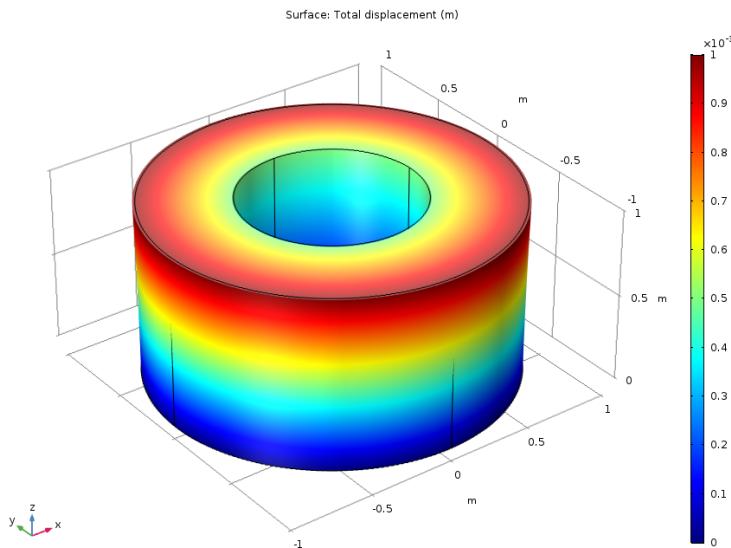


Figure 5.16: Torsion of a tube

When twisting a circular tube on the top surface one can decompose the deformation in a radial component  $u_r$  and an angular component  $u_\varphi$ . The vertical component is assumed to vanish<sup>20</sup>, i.e.  $u_3 = 0$ .

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = u_r(r, z) \begin{pmatrix} \cos \varphi \\ \sin \varphi \\ 0 \end{pmatrix} + u_\varphi(r, z) \begin{pmatrix} -\sin \varphi \\ +\cos \varphi \\ 0 \end{pmatrix}$$

<sup>20</sup>This is just for the sake of simplicity of the presentation. The vertical displacement  $u_3$  can be used as an unknown function too, with zero displacement at the top and the bottom. The resulting three Euler–Lagrange equations will then lead to  $u_3 = 0$ . This author has the detailed computations.

Based on the rotational symmetry<sup>21</sup> one can examine the expression in the  $xz$  plane only, i.e.  $y = 0$ . Let  $r = x$  and find the normal strains

$$\varepsilon_{11} = \frac{\partial u_1}{\partial x} = \frac{\partial u_r}{\partial r}, \quad \varepsilon_{22} = \frac{\partial u_2}{\partial y} = \frac{1}{r} u_r, \quad \varepsilon_{33} = \frac{\partial u_3}{\partial z} = 0$$

and the shear strains

$$2\varepsilon_{xy} = \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} = -\frac{1}{r} u_\varphi + \frac{\partial u_\varphi}{\partial r}, \quad 2\varepsilon_{xz} = \frac{\partial u_1}{\partial z} + \frac{\partial u_3}{\partial x} = \frac{\partial u_r}{\partial z}, \quad 2\varepsilon_{yz} = \frac{\partial u_2}{\partial z} + \frac{\partial u_3}{\partial y} = \frac{\partial u_\varphi}{\partial z}.$$

Observe that this is neither a plane strain nor a plane stress situation. The energy density  $W$  is given by equation (5.17) (page 350)

$$\begin{aligned} W &= \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \right\rangle + \\ &\quad + \frac{E}{(1+\nu)} \left\langle \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \right\rangle \end{aligned}$$

and leads to the energy density along the  $x$  axis with  $r = x$ .

$$\begin{aligned} W(r, z) &= \frac{E}{2(1+\nu)(1-2\nu)} ((1-\nu)(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2) + 2\nu(\varepsilon_{xx}\varepsilon_{yy} + \varepsilon_{xx}\varepsilon_{zz} + \varepsilon_{yy}\varepsilon_{zz})) + \\ &\quad + \frac{E}{(1+\nu)} (\varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) \\ &= \frac{E(1-\nu)}{2(1+\nu)(1-2\nu)} \left( \left( \frac{\partial u_r}{\partial r} \right)^2 + \frac{1}{r^2} u_r^2 \right) + \\ &\quad + \frac{E}{4(1+\nu)} \left( \left( \frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r} \right)^2 + \left( \frac{\partial u_r}{\partial z} \right)^2 + \left( \frac{\partial u_\varphi}{\partial z} \right)^2 \right). \end{aligned}$$

With this the functional  $U$  for the total elastic energy is given by an integration over the tube  $R_0 < r < R_1$  and  $0 < z < H$ , using cylindrical coordinates.

$$\begin{aligned} U(\vec{u}) &= \int_0^H \int_{R_0}^{R_1} W(r, z) 2\pi r dr dz \\ &= 2\pi \int_0^H \int_{R_0}^{R_1} \frac{E(1-\nu)r}{2(1+\nu)(1-2\nu)} \left( \left( \frac{\partial u_r}{\partial r} \right)^2 + \frac{u_r^2}{r^2} \right) + \\ &\quad + \frac{Er}{4(1+\nu)} \left( \left( \frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r} \right)^2 + \left( \frac{\partial u_r}{\partial z} \right)^2 + \left( \frac{\partial u_\varphi}{\partial z} \right)^2 \right) dr dz \\ &= \frac{\pi E}{2(1+\nu)} \int_0^H \int_{R_0}^{R_1} \frac{2(1-\nu)}{(1-2\nu)} \left( r \left( \frac{\partial u_r}{\partial r} \right)^2 + \frac{u_r^2}{r} \right) + \\ &\quad + \frac{r}{2} \left( \left( \frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r} \right)^2 + \left( \frac{\partial u_r}{\partial z} \right)^2 + \left( \frac{\partial u_\varphi}{\partial z} \right)^2 \right) dr dz \\ &= \frac{\pi E}{2(1+\nu)} \int_0^H \int_{R_0}^{R_1} F(r, z, u_r, \frac{\partial u_r}{\partial r}, \frac{\partial u_r}{\partial z}, u_\varphi, \frac{\partial u_\varphi}{\partial r}, \frac{\partial u_\varphi}{\partial z}) dr dz. \end{aligned}$$

<sup>21</sup>Using the chain rule one can express the partial derivatives with respect to Cartesian coordinates  $x$  and  $y$  in terms of derivatives with respect cylindrical coordinates  $r$  and  $\varphi$ .

$$f(x, y) = F(r, \varphi) \implies \frac{\partial f}{\partial x} = \cos \varphi \frac{\partial F}{\partial r} - \frac{\sin \varphi}{r} \frac{\partial F}{\partial \varphi} \quad \text{and} \quad \frac{\partial f}{\partial y} = \sin \varphi \frac{\partial F}{\partial r} + \frac{\cos \varphi}{r} \frac{\partial F}{\partial \varphi}$$

Along the  $x$  axis we have  $\varphi = 0$  and thus  $\cos \varphi = 1$  and  $\sin \varphi = 0$ .

Thus the elastic energy is given as a quadratic expression in terms of the two displacement functions  $u_r$  and  $u_\varphi$  and their partial derivatives. The physical solution will minimize this energy based on the Bernoulli principle. Use computations very similar to Section 5.2.4 (page 314) to generate the Euler–Lagrange equations for the two unknown functions  $u_r$  and  $u_\varphi$ .

- For the radial displacement function  $u_r(r, z)$ :

$$\begin{aligned}\frac{\partial F}{\partial u_r} &= \frac{4(1-\nu)}{1-2\nu} \frac{u_r}{r} \\ \frac{\partial F}{\partial \frac{\partial u_r}{\partial r}} &= \frac{4(1-\nu)}{1-2\nu} r \frac{\partial u_r}{\partial r} \\ \frac{\partial F}{\partial \frac{\partial u_r}{\partial z}} &= r \frac{\partial u_r}{\partial z} \\ \frac{\partial}{\partial r} \frac{\partial F}{\partial \frac{\partial u_r}{\partial r}} + \frac{\partial}{\partial z} \frac{\partial F}{\partial \frac{\partial u_r}{\partial z}} &= \frac{\partial F}{\partial u_r} \\ \frac{4(1-\nu)}{1-2\nu} \left( r \frac{\partial^2 u_r}{\partial r^2} + \frac{\partial u_r}{\partial r} \right) + r \frac{\partial^2 u_r}{\partial z^2} &= \frac{4(1-\nu)}{1-2\nu} \frac{u_r}{r}\end{aligned}$$

on the domain  $R_0 < r < R_1$  and  $0 < z < H$ . At the bottom  $z = 0$  and the top  $z = H$  the boundary conditions are

$$u_r(r, 0) = 0 \quad \text{and} \quad u_r(r, H) = 0 \quad \text{for } R_0 < r < R_1$$

and on the sides  $r = R_i$  we use the natural boundary condition  $\frac{\partial F}{\partial \frac{\partial u_r}{\partial r}} = 0$ , leading to

$$\frac{\partial u_r(R_0, z)}{\partial r} = \frac{\partial u_r(R_1, z)}{\partial r} = 0 \quad \text{for } 0 < z < H .$$

This boundary value problem is solved by  $u_r(r, z) = 0$ , i.e. no radial displacement.

- For the angular displacement function  $u_\varphi(r, z)$ :

$$\begin{aligned}\frac{\partial F}{\partial u_\varphi} &= -\left(\frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r}\right) \\ \frac{\partial F}{\partial \frac{\partial u_\varphi}{\partial r}} &= +r\left(\frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r}\right) \\ \frac{\partial F}{\partial \frac{\partial u_\varphi}{\partial z}} &= r \frac{\partial u_\varphi}{\partial z} \\ \frac{\partial}{\partial r} \frac{\partial F}{\partial \frac{\partial u_\varphi}{\partial r}} + \frac{\partial}{\partial z} \frac{\partial F}{\partial \frac{\partial u_\varphi}{\partial z}} &= \frac{\partial F}{\partial u_\varphi} \\ \frac{\partial}{\partial r} \left(r \left(\frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r}\right)\right) + \frac{\partial}{\partial z} \left(r \frac{\partial u_\varphi}{\partial z}\right) &= -\left(\frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r}\right) \\ r \frac{\partial^2 u_\varphi}{\partial r^2} + r \frac{\partial^2 u_\varphi}{\partial z^2} &= -\frac{\partial u_\varphi}{\partial r} + \frac{u_\varphi}{r}\end{aligned}$$

on the domain  $R_0 < r < R_1$  and  $0 < z < H$ . At the bottom  $z = 0$  and the top  $z = H$  the boundary conditions are

$$u_\varphi(r, 0) = 0 \quad \text{and} \quad u_r(r, H) = r\alpha \quad \text{for } R_0 < r < R_1$$

and on the sides  $r = R_i$  use the natural boundary condition  $\frac{\partial F}{\partial \frac{\partial u_\varphi}{\partial r}} = r\left(\frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r}\right) = 0$ , leading to

$$R_0 \frac{\partial u_\varphi(R_0, z)}{\partial r} = u_\varphi(R_0, z) \quad \text{and} \quad R_1 \frac{\partial u_\varphi(R_1, z)}{\partial r} = u_\varphi(R_1, z) \quad \text{for } 0 < z < H$$

This boundary value problem is solved by  $u_\varphi(r, z) = r z^{\frac{\alpha}{H}}$ .

Using the above solutions  $u_r(r, z) = 0$  and  $u_\varphi(r, z) = r z \frac{\alpha}{H}$  compute the energy density

$$\begin{aligned} W(r, z) &= \frac{E(1-\nu)}{2(1+\nu)(1-2\nu)} \left( \left( \frac{\partial u_r}{\partial r} \right)^2 + \frac{1}{r^2} u_r^2 \right) + \frac{E}{4(1+\nu)} \left( \left( \frac{\partial u_\varphi}{\partial r} - \frac{u_\varphi}{r} \right)^2 + \left( \frac{\partial u_r}{\partial z} \right)^2 + \left( \frac{\partial u_\varphi}{\partial z} \right)^2 \right) \\ &= \frac{E(1-\nu)}{2(1+\nu)(1-2\nu)} 0 + \frac{E}{4(1+\nu)} \left( 0^2 + 0^2 + \left( r \frac{\alpha}{H} \right)^2 \right) = \frac{E r^2}{4(1+\nu) H^2} \alpha^2 \end{aligned}$$

and thus the elastic energy by an integration

$$\begin{aligned} U(\vec{u}) &= \int_0^H \int_{R_0}^{R_1} W(r, z) 2\pi r dr dz = \frac{2\pi E \alpha^2}{4(1+\nu) H^2} \int_0^H \int_{R_0}^{R_1} r^3 dr dz \\ &= \frac{2\pi E \alpha^2}{4 \cdot 4(1+\nu) H} (R_1^4 - R_0^4). \end{aligned}$$

The elastic energy is expressed as a function of the angle of rotation  $\alpha$ . The torque  $T$  required to twist this circular tube is given by

$$T = \frac{\partial U}{\partial \alpha} = \frac{\pi E}{4(1+\nu)} (R_1^4 - R_0^4) \frac{\alpha}{H}.$$

This result can also be obtained by **assuming** that the cut at height  $z$  is rotated by an angle  $\alpha \frac{z}{H}$  and then determine the resulting shear stresses along those planes. The computations are rather easy. The above approach verifies that this simplification is correct.  $\diamond$

## 5.7 More on Tensors and Energy Densities for Nonlinear Material Laws

### 5.7.1 A few More Tensors

In most parts of these lecture notes we only use infinitesimal strains. This restricts the applications to small strain and displacement situations only. One important example that can not be examined using infinitesimal strains only is the large bending of slender beams. There are nonlinear extensions allowing to describe more general situations. In this section we provide a starting point for further investigations. Find more information in [Bowe10], [Redd13] and [Redd15].

Use the displacement gradient tensor  $\mathbf{DU}$  (Example 5–29) and examine Figure 5.6 (page 327) to verify that for a deformation  $\vec{u}(x, y)$  the vector

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} + \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial x} & \frac{\partial u_2}{\partial y} \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} + \mathbf{DU} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = (\mathbb{I} + \mathbf{DU}) \vec{\Delta x}$$

connects the points  $A'$  to  $D'$ .  $\mathbb{I} + \mathbf{DU}$  is the **deformation gradient tensor**. For two vectors

$$\vec{\Delta x}_1 = \begin{pmatrix} \Delta x_1 \\ \Delta y_1 \end{pmatrix} \quad \text{and} \quad \vec{\Delta x}_2 = \begin{pmatrix} \Delta x_2 \\ \Delta y_2 \end{pmatrix}$$

examine the scalar product of the image of the two vectors.

$$\begin{aligned} \langle (\mathbb{I} + \mathbf{DU}) \vec{\Delta x}_1, (\mathbb{I} + \mathbf{DU}) \vec{\Delta x}_2 \rangle &= \langle \vec{\Delta x}_1, (\mathbb{I} + \mathbf{DU}^T)(\mathbb{I} + \mathbf{DU}) \vec{\Delta x}_2 \rangle \\ &= \langle \vec{\Delta x}_1, (\mathbb{I} + \mathbf{DU} + \mathbf{DU}^T + \mathbf{DU}^T \mathbf{DU}) \vec{\Delta x}_2 \rangle \\ &= \langle \vec{\Delta x}_1, \mathbf{C} \vec{\Delta x}_2 \rangle \end{aligned}$$

The matrix

$$\mathbf{C} = \mathbb{I} + \mathbf{DU} + \mathbf{DU}^T + \mathbf{DU}^T \mathbf{DU}$$

is the **Cauchy–Green deformation tensor**. In particular obtain in Figure 5.6 (page 327)

$$|A'D'|^2 = \langle (\mathbb{I} + \mathbf{DU}) \vec{\Delta x}, (\mathbb{I} + \mathbf{DU}) \vec{\Delta x} \rangle = \langle \vec{\Delta x}, \mathbf{C} \vec{\Delta x} \rangle.$$

For the 2D situation use

$$\begin{aligned} \mathbf{C} &= \mathbb{I} + \mathbf{DU} + \mathbf{DU}^T + \mathbf{DU}^T \mathbf{DU} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 2 \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \\ \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} & 2 \frac{\partial u_2}{\partial y} \end{bmatrix} + \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_2}{\partial x} \\ \frac{\partial u_1}{\partial y} & \frac{\partial u_2}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial x} & \frac{\partial u_2}{\partial y} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 2 \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \\ \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} & 2 \frac{\partial u_2}{\partial y} \end{bmatrix} + \begin{bmatrix} \left(\frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 & \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} \\ \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} & \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(\frac{\partial u_2}{\partial y}\right)^2 \end{bmatrix} \\ &= \mathbb{I} + 2 \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} + \begin{bmatrix} \left(\frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 & \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} \\ \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} & \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(\frac{\partial u_2}{\partial y}\right)^2 \end{bmatrix}. \end{aligned}$$

Use the transformation rule (5.11) for the displacement gradient tensor  $\mathbf{DU}$  to examine the Cauchy–Green deformation tensor in a rotated coordinate system.

$$\begin{aligned} \mathbf{C}' &= \mathbb{I} + \mathbf{DU}' + \mathbf{DU}'^T + \mathbf{DU}'^T \mathbf{DU}' \\ &= \mathbb{I} + \mathbf{R}^T \mathbf{D} \mathbf{U} \mathbf{R} + \mathbf{R}^T \mathbf{D} \mathbf{U}^T \mathbf{R} + \mathbf{R}^T \mathbf{D} \mathbf{U}^T \mathbf{R} \mathbf{R}^T \mathbf{D} \mathbf{U} \mathbf{R} \\ &= \mathbb{I} + \mathbf{R}^T \mathbf{D} \mathbf{U} \mathbf{R} + \mathbf{R}^T \mathbf{D} \mathbf{U}^T \mathbf{R} + \mathbf{R}^T \mathbf{D} \mathbf{U}^T \mathbf{D} \mathbf{U} \mathbf{R} \\ &= \mathbf{R}^T (\mathbb{I} + \mathbf{DU} + \mathbf{DU}^T + \mathbf{DU}^T \mathbf{DU}) \mathbf{R} \\ &= \mathbf{R}^T \mathbf{C} \mathbf{R} \end{aligned}$$

This is the transformation rule for a second order tensor.

The **Green strain tensor** is given by

$$\mathbf{E} = \frac{1}{2} (\mathbf{C} - \mathbb{I})$$

and thus

$$\mathbf{E} = \begin{bmatrix} E_{xx} & E_{xy} \\ E_{xy} & E_{yy} \end{bmatrix} = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \left(\frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 & \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} \\ \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} & \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(\frac{\partial u_2}{\partial y}\right)^2 \end{bmatrix}. \quad (5.19)$$

When dropping the quadratic contributions obtain the previous (infinitesimal) strain tensor. In Table 5.6 find the definitions of the tensors defined in these lecture notes.

Geometric interpretation of the entries in the Green strain tensor:

$E_{xx}$ : Consider a deformation with fixed origin. The point  $(\Delta x, 0)$  is moved to  $(1 + \frac{\partial u_1}{\partial x}, \frac{\partial u_2}{\partial x}) \Delta x$  and thus the new length  $l$  of the original segment from  $(0, 0)$  to  $(\Delta x, 0)$  is given by<sup>22</sup>

$$\begin{aligned} l^2 &= \left( (1 + \frac{\partial u_1}{\partial x})^2 + (\frac{\partial u_2}{\partial x})^2 \right) (\Delta x)^2 \\ &= \left( 1 + 2 \frac{\partial u_1}{\partial x} + (\frac{\partial u_1}{\partial x})^2 + (\frac{\partial u_2}{\partial x})^2 \right) (\Delta x)^2 \end{aligned}$$

<sup>22</sup>For  $|z| \ll 1$  use the linear approximation  $\sqrt{1+z} \approx 1 + \frac{1}{2} z$ .

$$\begin{aligned}
l &= \sqrt{1 + 2 \frac{\partial u_1}{\partial x} + (\frac{\partial u_1}{\partial x})^2 + (\frac{\partial u_2}{\partial x})^2} \Delta x \\
&\approx \left( 1 + \frac{\partial u_1}{\partial x} + \frac{1}{2} (\frac{\partial u_1}{\partial x})^2 + \frac{1}{2} (\frac{\partial u_2}{\partial x})^2 \right) \Delta x \\
\frac{\Delta l}{\Delta x} &\approx \left( \frac{\partial u_1}{\partial x} + \frac{1}{2} (\frac{\partial u_1}{\partial x})^2 + \frac{1}{2} (\frac{\partial u_2}{\partial x})^2 \right) = E_{xx}.
\end{aligned}$$

Thus  $E_{xx}$  shows the relative change of length in  $x$  direction. Use Figure 5.6 (page 327) for a visualization of the result. Observe that the displaced segment need not be vertical any more.

$E_{yy}$ : Similar to  $E_{xx}$ , but in  $y$  direction.

$E_{xy}$ : Use the two orthogonal vectors  $\vec{v}_1 = (1, 0)^T$  and  $\vec{v}_2 = (0, 1)^T$ , attach these at a point, deform the solid and then determine the angle  $\phi$  between the two deformed vectors. Assume that the entries in  $\mathbf{DU}$  are small. Using

$$\begin{aligned}
(\mathbb{I} + \mathbf{DU}) \vec{v}_1 &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} \frac{\partial u_1}{\partial x} \\ \frac{\partial u_2}{\partial x} \end{pmatrix}, \quad (\mathbb{I} + \mathbf{DU}) \vec{v}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial y} \end{pmatrix} \\
\cos(\phi) &= \frac{\langle \vec{v}_1, \mathbf{C} \vec{v}_2 \rangle}{\|(\mathbb{I} + \mathbf{DU}) \vec{v}_1\| \|(\mathbb{I} + \mathbf{DU}) \vec{v}_2\|} = \frac{C_{xy}}{\sqrt{1 + \text{small}} \sqrt{1 + \text{small}}} \approx 2 E_{xy}
\end{aligned}$$

conclude

$$\frac{\pi}{2} - \phi \approx \sin\left(\frac{\pi}{2} - \phi\right) = \cos \phi \approx 2 E_{xy}.$$

Thus  $2 E_{xy}$  indicates by how much the angle between the two coordinates axis is diminished by the deformation. This interpretation is identical to the interpretation of the infinitesimal strain tensor on page 329.

### 5-38 Example : Pure rotation

For a pure rotation

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos \phi x - \sin \phi y \\ \sin \phi x + \cos \phi y \end{pmatrix}$$

we find the displacement vector

$$\begin{pmatrix} u_1(x, y) \\ u_2(x, y) \end{pmatrix} = \begin{pmatrix} \cos \phi x - \sin \phi y - x \\ \sin \phi x + \cos \phi y - y \end{pmatrix}.$$

Now determine the partial derivatives of the displacements with respect to  $x$  and  $y$ . This leads to the Cauchy–Green deformation tensor

$$\begin{aligned}
\mathbf{C} &= \mathbb{I} + \begin{bmatrix} 2 \cos \phi - 2 & -\sin \phi + \sin \phi \\ -\sin \phi + \sin \phi & 2 \cos \phi - 2 \end{bmatrix} + \\
&+ \begin{bmatrix} (\cos \phi - 1)^2 + \sin^2 \phi & -(1 - \cos \phi) \sin \phi + \sin \phi (1 - \cos \phi) \\ -(1 - \cos \phi) \sin \phi + \sin \phi (1 - \cos \phi) & \sin^2 \phi + (\cos \phi - 1)^2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 2 \cos \phi - 2 & 0 \\ 0 & 2 \cos \phi - 2 \end{bmatrix} + \begin{bmatrix} 2 - 2 \cos \phi & 0 \\ 0 & 2 - 2 \cos \phi \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\end{aligned}$$

and we find

$$|A'D'|^2 = \langle \mathbf{C} \vec{AD}, \vec{AD} \rangle = \langle \vec{AD}, \vec{AD} \rangle = \|\vec{AD}\|^2 = |AD|^2.$$

Thus no section of the solid is stretched, even for large angles  $\phi$ . All components of the Green strain tensor  $\mathbf{E}$  vanish. Thus the small angle restriction from Example 5-18 disappeared. With this approach we can examine situations with large deformations, but still small strains, e.g. bending of slender rods.  $\diamond$

infinitesimal strain tensor $\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{xx} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2} \left( \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} \right) \\ \frac{1}{2} \left( \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} \right) & \frac{\partial u_2}{\partial y} \end{bmatrix}$
displacement gradient tensor $\mathbf{DU} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial x} & \frac{\partial u_2}{\partial y} \end{bmatrix}$
deformation gradient tensor $\mathbb{I} + \mathbf{DU} = \begin{bmatrix} 1 + \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial x} & 1 + \frac{\partial u_2}{\partial y} \end{bmatrix}$
Cauchy–Green deformation tensor $\mathbf{C} = \mathbb{I} + \mathbf{DU} + \mathbf{DU}^T + \mathbf{DU}^T \mathbf{DU} = \mathbb{I} + 2 \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{xx} \end{bmatrix} + \begin{bmatrix} \left( \frac{\partial u_1}{\partial x} \right)^2 + \left( \frac{\partial u_2}{\partial x} \right)^2 & \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} \\ \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} & \left( \frac{\partial u_1}{\partial y} \right)^2 + \left( \frac{\partial u_2}{\partial y} \right)^2 \end{bmatrix}$
Green strain tensor $\begin{bmatrix} E_{xx} & E_{xy} \\ E_{xy} & E_{xx} \end{bmatrix} = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{xx} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \left( \frac{\partial u_1}{\partial x} \right)^2 + \left( \frac{\partial u_2}{\partial x} \right)^2 & \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} \\ \frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} & \left( \frac{\partial u_1}{\partial y} \right)^2 + \left( \frac{\partial u_2}{\partial y} \right)^2 \end{bmatrix}$
infinitesimal stress tensor $\begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix}$

Table 5.6: Different tensors in 2D

Since the Green strain tensor  $\mathbf{E}$  satisfies the usual transformation rule for a second order tensor it can be diagonalized by rotating the coordinate system and we find in the rotated coordinate system

$$\begin{bmatrix} E_{xx} & 0 \\ 0 & E_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & 0 \\ 0 & \frac{\partial u_2}{\partial y} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \left(\frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 & 0 \\ 0 & \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(\frac{\partial u_2}{\partial y}\right)^2 \end{bmatrix}.$$

This will be useful to determine energy formulas for selected deformation problems, or you may use the invariant expressions of the Green strain tensor, comparable to the observations on page 351ff.

### 5-39 Example : More invariants of the Cauchy–Green deformation tensor, used to describe nonlinear material laws

The Cauchy–Green deformation tensor  $\mathbf{C}$  is often used to describe the elastic energy density  $W(\mathbf{C})$  for large deformations. It is easiest to work with the tensor in principal form, i.e. the Cauchy–Green deformation tensor in a principal system.

$$\begin{aligned} \mathbf{C} &= \mathbb{I} + \mathbf{D}\mathbf{U} + \mathbf{D}\mathbf{U}^T + \mathbf{D}\mathbf{U}^T\mathbf{D}\mathbf{U} \\ &= \begin{bmatrix} 1 + 2\frac{\partial u_1}{\partial x} & & & \\ & \ddots & & \\ & & 1 + 2\frac{\partial u_2}{\partial y} & \\ & & & \ddots \\ & & & & 1 + 2\frac{\partial u_3}{\partial z} \end{bmatrix} + \\ &\quad + \begin{bmatrix} \left(\frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 + \left(\frac{\partial u_3}{\partial x}\right)^2 & & & \\ & \ddots & & \\ & & \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(\frac{\partial u_2}{\partial y}\right)^2 + \left(\frac{\partial u_3}{\partial y}\right)^2 & \\ & & & \ddots \\ & & & & \left(\frac{\partial u_1}{\partial z}\right)^2 + \left(\frac{\partial u_2}{\partial z}\right)^2 + \left(\frac{\partial u_3}{\partial z}\right)^2 \end{bmatrix} \\ &= \begin{bmatrix} \left(1 + \frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 + \left(\frac{\partial u_3}{\partial x}\right)^2 & & & \\ & \ddots & & \\ & & \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(1 + \frac{\partial u_2}{\partial y}\right)^2 + \left(\frac{\partial u_3}{\partial y}\right)^2 & & \\ & & & \ddots \\ & & & & \left(\frac{\partial u_1}{\partial z}\right)^2 + \left(\frac{\partial u_2}{\partial z}\right)^2 + \left(1 + \frac{\partial u_3}{\partial z}\right)^2 \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1^2 & 0 & 0 \\ 0 & \lambda_2^2 & 0 \\ 0 & 0 & \lambda_3^2 \end{bmatrix} \end{aligned}$$

The diagonal entries  $\lambda_i^2$  are squares of the the **principal stretches**  $\lambda_i$ .

$$\begin{aligned} \lambda_1 &= \sqrt{\left(1 + \frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 + \left(\frac{\partial u_3}{\partial x}\right)^2} = \text{factor by which } x \text{ axis will be stretched} \\ \lambda_2 &= \sqrt{\left(1 + \frac{\partial u_2}{\partial y}\right)^2 + \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(\frac{\partial u_3}{\partial y}\right)^2} = \text{factor by which } y \text{ axis will be stretched} \\ \lambda_3 &= \sqrt{\left(1 + \frac{\partial u_3}{\partial z}\right)^2 + \left(\frac{\partial u_1}{\partial z}\right)^2 + \left(\frac{\partial u_2}{\partial z}\right)^2} = \text{factor by which } z \text{ axis will be stretched} \end{aligned}$$

A geometrical reasoning for the above is shown in Figure 5.6 (on page 327).

The elastic energy density  $W$  is usually expressed in terms of invariants of the tensors. There are many different expressions for the energy density, corresponding to different types of materials, always expressed in term of invariants. A few of the invariants of the Cauchy–Green deformation tensor are

$$\begin{aligned} I_1 &= \text{trace}(\mathbf{C}) = \lambda_1^2 + \lambda_2^2 + \lambda_3^2 \\ &= \left(1 + \frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 + \left(\frac{\partial u_3}{\partial x}\right)^2 + \left(1 + \frac{\partial u_2}{\partial y}\right)^2 + \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(\frac{\partial u_3}{\partial y}\right)^2 + \end{aligned}$$

$$\begin{aligned}
& + \left(1 + \frac{\partial u_3}{\partial z}\right)^2 + \left(\frac{\partial u_1}{\partial z}\right)^2 + \left(\frac{\partial u_2}{\partial z}\right)^2, \\
I_2 &= \lambda_1^2 \lambda_2^2 + \lambda_2^2 \lambda_3^2 + \lambda_1^2 \lambda_3^2 = \frac{1}{2} (\text{trace}(\mathbf{C})^2 - \text{trace}(\mathbf{C}^2)), \\
I_3 &= \det(\mathbf{C}) = \lambda_1^2 \cdot \lambda_2^2 \cdot \lambda_3^2, \\
J &= \sqrt{\det(\mathbf{C})} = \lambda_1 \cdot \lambda_2 \cdot \lambda_3 \quad \text{factor of volume change.} \\
\bar{I}_1 &= \frac{I_1}{J^{2/3}} = \lambda_1^{4/3} \lambda_2^{-2/3} \lambda_3^{-2/3} + \lambda_1^{-2/3} \lambda_2^{4/3} \lambda_3^{-2/3} + \lambda_1^{-2/3} \lambda_2^{-2/3} \lambda_3^{4/3}.
\end{aligned}$$

Observe that these expressions can be determined by algebraic operations, once the entries in  $\mathbf{C}$  are known. It is not necessary to compute the eigenvalues. This is used in Example 5–44. These expressions can be used to examine different models for the energy density  $W$ , mainly for nonlinear material laws.  $\diamond$

Table 5.7 shows some nonlinear material laws commonly used. The following examples analyze some of these definitions. The energy density  $W$  for the nonlinear material is shown, and then a small strain analysis is performed to examine the connection to the linear law of Hooke. For most cases uniaxial loading in  $x$ -direction and hydrostatic loading is examined.

energy density $W$	loading	small strain approximation	Example
Neo-Hookean, incompressible $C_{10} \cdot (I_1 - 3) = C_{10} \cdot (\bar{I}_1 - 3)$	uniaxial	$C_{10} = \frac{1}{6} E$	5–40
	biaxial		5–41
Neo-Hookean, compressible $C_{10} \cdot (\bar{I}_1 - 3) + \frac{1}{D} (J - 1)^2$	uniaxial	$C_{10} = \frac{E}{4(1+\nu)} = \frac{\mu}{2}$	5–42
	hydrostatic	$\frac{1}{D} = \frac{E}{6(1-2\nu)} = \frac{K}{2}$	5–43
	shearing	$C_{10} = \frac{E}{4(1+\nu)} = \frac{\nu}{2}$	5–44
$\frac{\mu}{2} (I_1 - 3) - \mu \ln(J) + \frac{\lambda}{2} \ln(J)^2$	uniaxial	$\mu = \frac{E}{2(1+\nu)}$	5–45
	hydrostatic	$\lambda = \frac{\mu E}{(1+\nu)(1-2\nu)}$	5–46
	shearing		5–47
Mooney–Rivlin, incompressible $C_{10} \cdot (\bar{I}_1 - 3) + C_{01} \cdot (\bar{I}_2 - 3)$	uniaxial	$C_{10} + C_{01} = \frac{1}{6} E$	5–48
	hydrostatic		
Mooney–Rivlin, compressible $C_{10} \cdot (\bar{I}_1 - 3) + C_{01} \cdot (\bar{I}_2 - 3) + \frac{1}{D} (J - 1)^2$	hydrostatic	$\frac{1}{D} = \frac{E}{6(1-2\nu)} = \frac{K}{2}$	5–49
	uniaxial	$C_{10} + C_{01} \approx \frac{1}{6} E$	5–49
Ogden, incompressible $\frac{\mu}{\alpha} (\lambda_1^\alpha + \lambda_2^\alpha + \lambda_3^\alpha - 3)$	uniaxial	$E = \frac{3}{2} \mu \alpha$	5–50
Ogden, compressible $\frac{\mu}{\alpha} (\bar{\lambda}_1^\alpha + \bar{\lambda}_2^\alpha + \bar{\lambda}_3^\alpha - 3) + \frac{1}{D} (\lambda_1 \lambda_2 \lambda_3 - 1)$	hydrostatic	$\frac{1}{D} = \frac{E}{6(1-2\nu)} = \frac{K}{2}$	5–51
	uniaxial	$E = \frac{18\mu\alpha}{12+\mu\alpha D}$	5–52

Table 5.7: Some nonlinear material laws, given by the energy density  $W$ . The connection of the small strain approximation to Hooke's linear law is shown.

### 5.7.2 Neo-Hookean Energy Density Models

#### 5–40 Example : Neo-Hookean energy density for incompressible materials, uniaxial loading

not in class

Now try to connect the Neo-Hookean form for the energy density with the result for small strains based on Hooke's law. To do this investigate a uniaxial loading in  $x$  direction only with

$$\varepsilon_{xx} = \frac{\partial u_1}{\partial x} \quad \text{very small} \quad .$$

Start by examining an incompressible material, i.e.  $J = \lambda_1 \lambda_2 \lambda_3 = 1$  and the energy density given by

$$W = C_{10} \cdot (I_1 - 3) = C_{10} \cdot (\lambda_1^2 + \lambda_2^2 + \lambda_3^2 - 3), \quad (5.20)$$

where  $C_{10}$  is a material constant. This is called the **Neo-Hookean** energy density, for incompressible materials. To examine uniaxial loading use the symmetry  $\lambda_2 = \lambda_3$  and the incompressibility  $\lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1$  to conclude

$$\lambda_2 = \lambda_3 = \frac{1}{\sqrt{\lambda_1}} \quad \text{and} \quad W = C_{10} \cdot \left( \lambda_1^2 + \frac{2}{\lambda_1} - 3 \right).$$

With the notations  $z_2 = \frac{\partial u_2}{\partial x}$  and  $z_3 = \frac{\partial u_3}{\partial x}$  minimize  $W(z_2, z_3)$ .

$$\begin{aligned} 0 &= \frac{\partial}{\partial z_2} W = C_{10} \cdot \left( 2\lambda_1 - \frac{2}{\lambda_1^2} - 3 \right) \frac{\partial \lambda_1}{\partial z_2} = C_{10} \cdot \left( 2\lambda_1 - \frac{2}{\lambda_1^2} - 3 \right) \frac{1}{\lambda_1} \frac{\partial u_2}{\partial x} \implies \frac{\partial u_2}{\partial x} = 0 \\ 0 &= \frac{\partial}{\partial z_3} W = C_{10} \cdot \left( 2\lambda_1 - \frac{2}{\lambda_1^2} - 3 \right) \frac{\partial \lambda_1}{\partial z_3} = C_{10} \cdot \left( 2\lambda_1 - \frac{2}{\lambda_1^2} - 3 \right) \frac{1}{\lambda_1} \frac{\partial u_3}{\partial x} \implies \frac{\partial u_3}{\partial x} = 0 \end{aligned}$$

There is no shearing in this situation, as expected. Using

$$\lambda_1 = \sqrt{\left(1 + \frac{\partial u_1}{\partial x}\right)^2} = \left|1 + \frac{\partial u_1}{\partial x}\right| = 1 + \frac{\partial u_1}{\partial x} = 1 + \varepsilon_{xx}$$

find the elastic energy density  $W = \frac{1}{2} E \varepsilon_{xx}^2$  (based on equation (5.16) on page 350, generated by small deformations and Hooke's law (5.15) for incompressible materials) and compare to the result based on the above formula.

$$\begin{aligned} W &= C_{10} \cdot \left( \lambda_1^2 + \frac{2}{\lambda_1} - 3 \right) = C_{10} \cdot \left( (1 + \varepsilon_{xx})^2 + \frac{2}{1 + \varepsilon_{xx}} - 3 \right) \\ &\approx C_{10} \cdot \left( 1 + 2\varepsilon_{xx} + \varepsilon_{xx}^2 + 2(1 - \varepsilon_{xx} + \varepsilon_{xx}^2 - \varepsilon_{xx}^3 + \varepsilon_{xx}^4) - 3 \right) = C_{10} \cdot (3\varepsilon_{xx}^2 - 2\varepsilon_{xx}^3 + 2\varepsilon_{xx}^4) \end{aligned}$$

For small  $\varepsilon_{xx}$  this should be similar to  $W = \frac{1}{2} E \varepsilon_{xx}^2$ , leading to

$$C_{10} = \frac{1}{6} E = \frac{1}{2} \mu \quad \text{with shear modulus} \quad \mu = \frac{E}{2(1 + \nu)} = \frac{E}{3}.$$

Using this generate an approximating stress-strain curve based on  $W \approx \frac{1}{6} E (3\varepsilon_{xx}^2 - 2\varepsilon_{xx}^3)$ .

$$\sigma_x = \frac{\partial W}{\partial \varepsilon_{xx}} \approx E \varepsilon_{xx} - E \varepsilon_{xx}^2$$

Since the material is assumed to be incompressible, confirm Poisson's ratio of  $\nu = \frac{\varepsilon_{yy}}{\varepsilon_{xx}} = -\frac{1}{2}$  by

$$\lambda_2 = \lambda_3 = \frac{1}{\sqrt{\lambda_1}} = \frac{1}{\sqrt{1 + \varepsilon_{xx}}} \approx 1 - \frac{1}{2} \varepsilon_{xx}.$$

In Figure 5.17(a) find the stress-strain curve for the linear Hooke's law and the Neo-Hookean material under uniaxial loading. This is only valid if  $|\varepsilon_{xx}| \ll 1$ , but the stress-strain curve can be generated for larger values of  $\varepsilon_{xx}$  too.

$$\begin{aligned} \sigma_x &= \frac{\partial W}{\partial \varepsilon_{xx}} = C_{10} \frac{\partial}{\partial \varepsilon_{xx}} \left( (1 + \varepsilon_{xx})^2 + \frac{2}{1 + \varepsilon_{xx}} - 3 \right) = C_{10} \cdot \left( 2(1 + \varepsilon_{xx}) - \frac{2}{(1 + \varepsilon_{xx})^2} \right) \\ &= C_{10} 2 \left( 1 + \varepsilon_{xx} - 1 + 2\varepsilon_{xx} - 3\varepsilon_{xx}^2 + 4\varepsilon_{xx}^3 - 5\varepsilon_{xx}^4 + O(\varepsilon_{xx}^5) \right) \\ &= C_{10} \left( 6\varepsilon_{xx} - 6\varepsilon_{xx}^2 + 8\varepsilon_{xx}^3 - 10\varepsilon_{xx}^4 + O(\varepsilon_{xx}^5) \right) \\ \frac{\partial \sigma_x}{\partial \varepsilon_{xx}} &= C_{10} \cdot \left( 2 + \frac{4}{(1 + \varepsilon_{xx})^3} \right) \end{aligned}$$

With  $C_{10} = \frac{1}{6} E$  the stress-strain curve is

- a straight line with slope  $\frac{\partial \sigma_x}{\partial \varepsilon_{xx}} \approx C_{10} 6 = E$  for  $0 \leq |\varepsilon_{xx}| \ll 1$ , identical to Hooke's law.
- a straight line  $\sigma_x \approx \frac{E}{3} (1 + \varepsilon_{xx})$  with slope  $\frac{1}{3} E$  for  $\varepsilon_{xx} \gg 1$  very large.
- for  $0 < \varepsilon_{xx}$  the slope is smaller than  $E$  and for  $-1 < \varepsilon_{xx} < 0$  the slope is larger than  $E$ .

This is visualized in Figure 5.17(a).

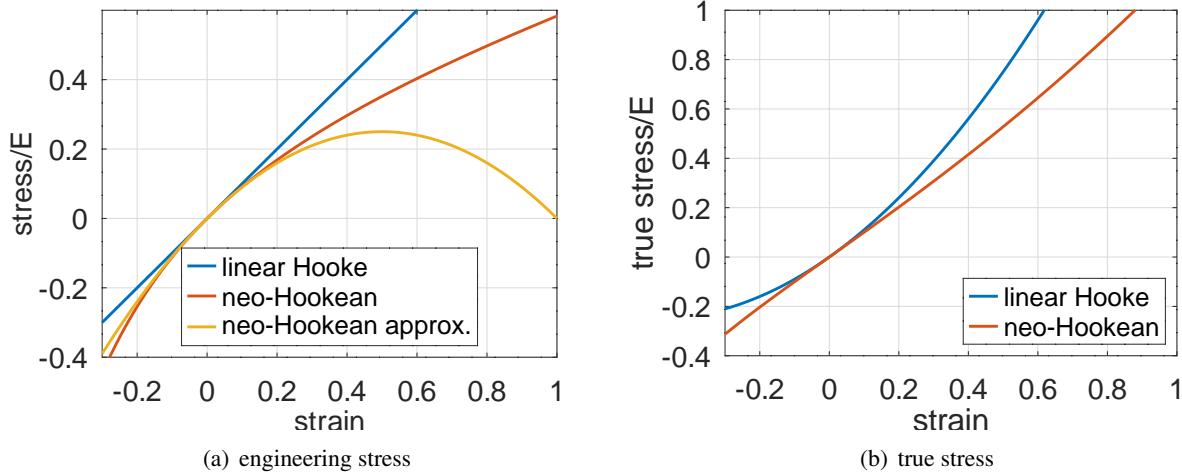


Figure 5.17: Stress–strain curve for Hooke's linear law and an incompressible neo–Hookean material under uniaxial loading

The above arguments use the engineering stress, i.e. the force is divided by the area of the undeformed solid. Since  $\lambda_2 = \lambda_3 = \frac{1}{\sqrt{\lambda_1}} = \frac{1}{\sqrt{1+\varepsilon_{xx}}}$  the cross sectional area is diminished by a factor of  $\lambda_2 \cdot \lambda_3 = \frac{1}{1+\varepsilon_{xx}}$ , leading to a true stress of

$$\begin{aligned}\sigma_{x,true} &= \sigma_x (1 + \varepsilon_{xx}) = C_{10} \cdot \left( 2(1 + \varepsilon_{xx}) - \frac{2}{(1 + \varepsilon_{xx})^2} \right) (1 + \varepsilon_{xx}) \\ &= C_{10} \cdot \left( 2(1 + \varepsilon_{xx})^2 - \frac{2}{(1 + \varepsilon_{xx})} \right)\end{aligned}$$

The true stress based on Hooke's law is given by  $E(\varepsilon_{xx} + \varepsilon_{xx}^2)$ , shown in Figure 5.17(b).  $\diamond$

A uniform **biaxial loading** is characterized by  $\sigma = \sigma_x = \sigma_y, \sigma_z = 0$  and no shearing, see also Example 5–34 on page 359. To derive the Hookean energy density for biaxial loading use Hooke's law (5.15) for incompressible materials.

$$\begin{aligned}\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} &= \frac{1}{2E} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} = \frac{1}{2E} \begin{pmatrix} \sigma_x \\ \sigma_y \\ -2\sigma_x \end{pmatrix} \\ W &= \frac{1}{2} \langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \rangle = \frac{2E}{2} \langle \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \rangle = 2E \varepsilon_{xx}^2\end{aligned}$$

The stress–strain curve for the biaxial loading with Hooke's law is determined by

$$\sigma = \frac{1}{2} \frac{\partial W}{\partial \varepsilon} = \frac{1}{2} \frac{\partial}{\partial \varepsilon} (2E \varepsilon^2) = 2E \varepsilon.$$

The factor  $\frac{1}{2}$  is used since the solid is stretched in two directions.

**5-41 Example : Neo-Hookean energy density for incompressible materials, biaxial loading**

not in class

Investigate a biaxial stretching in  $x$  and  $y$  directions with  $\varepsilon = \varepsilon_{xx} = \frac{\partial u_1}{\partial x} = \varepsilon_{yy} = \frac{\partial u_2}{\partial y}$ . For an incompressible material use  $J = \lambda_1 \lambda_2 \lambda_3 = 1$  and the Neo-Hookean energy density given by

$$W = C_{10} \cdot (I_1 - 3) = C_{10} \cdot (\lambda_1^2 + \lambda_2^2 + \lambda_3^2 - 3)$$

with the material constant  $C_{10}$ . The symmetry  $\lambda_1 = \lambda_2$  and the incompressibility  $\lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1$  lead to  $\lambda_3 = \frac{1}{\lambda_1^2}$  and the energy density

$$\begin{aligned} W &= C_{10} \cdot (2\lambda_1^2 + \frac{1}{\lambda_1^4} - 3) = C_{10} \cdot \left( 2(1 + \varepsilon_{xx})^2 + \frac{1}{(1 + \varepsilon_{xx})^4} - 3 \right) \\ &\approx C_{10} \cdot (2 + 4\varepsilon_{xx} + 2\varepsilon_{xx}^2 + (1 - 4\varepsilon_{xx} + 10\varepsilon_{xx}^2 - 20\varepsilon_{xx}^3 + 35\varepsilon_{xx}^4 - \dots) - 3) \\ &\approx C_{10} \cdot (12\varepsilon_{xx}^2 - 20\varepsilon_{xx}^3 + 34\varepsilon_{xx}^4) . \end{aligned}$$

For small  $\varepsilon_{xx}$  this should be similar to  $W = 2E\varepsilon_{xx}^2$ , leading to  $C_{10} = \frac{1}{6}E$  again, identical to the uniaxial loading in the previous example. The stress-strain curve is based on

$$\sigma = \frac{1}{2} \frac{\partial W}{\partial \varepsilon} = \frac{1}{2} C_{10} \cdot \left( 4(1 + \varepsilon) - \frac{4}{(1 + \varepsilon)^5} \right) = \frac{E}{3} (+6\varepsilon - 15\varepsilon^2 + 35\varepsilon^3 - 70\varepsilon^4 + \dots) .$$

Find the result in Figure 5.18 and observe the slopes to be twice as steep as in Figure 5.17 for uniaxial loading. The expressions for the stress  $\sigma$  as function of the strain  $\varepsilon$  are different for the uniaxial and biaxial loading situations.  $\diamond$

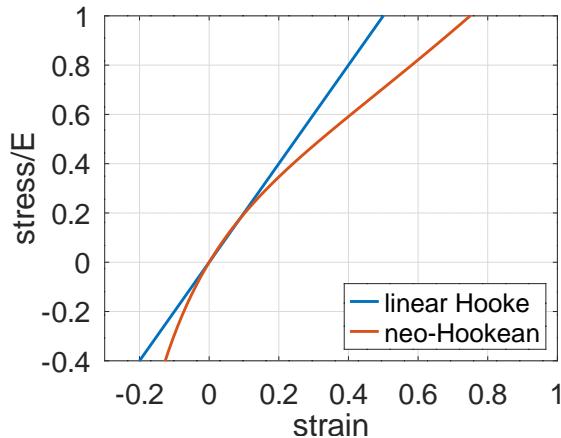


Figure 5.18: Stress-strain curve for Hooke's linear law and an incompressible neo-Hookean material under biaxial loading

**5-42 Example : From large strain energy to Hooke for uniaxial loading for compressible material**

not in class

In [Bowe10, §3.5.5] a Neo-Hookean energy density of the form

$$W = C_{10} \cdot (\bar{I}_1 - 3) + \frac{1}{D} (J - 1)^2 = C_{10} \cdot \left( \frac{I_1}{J^{2/3}} - 3 \right) + \frac{1}{D} (J - 1)^2 \quad (5.21)$$

is examined. The first expression should take shape changes into account, while the second term is related to volume changes.

To examine the energy in equation (5.21) for an uniaxial stretching a good approximations of the invariants as function of  $\varepsilon_{xx} = \frac{\partial u_1}{\partial x}$  is required. Assume a small deformation and use Hooke's law with a Poisson

ratio of  $0 \leq \nu \leq \frac{1}{2}$ . As consequence work with  $\varepsilon_{yy} = \varepsilon_{zz} = -\nu \varepsilon_{xx}$  and compute

$$\begin{aligned} J &= \lambda_1 \lambda_2 \lambda_3 = (1 + \varepsilon_{xx}) \cdot (1 + \varepsilon_{yy}) \cdot (1 + \varepsilon_{zz}) = (1 + \varepsilon_{xx}) \cdot (1 - \nu \varepsilon_{xx})^2 \\ &= 1 + (1 - 2\nu) \varepsilon_{xx} + (-2\nu + \nu^2) \varepsilon_{xx}^2 + \nu^2 \varepsilon_{xx}^3 \\ (J-1)^2 &= \varepsilon_{xx}^2 ((1-2\nu) + \nu(-2+\nu) \varepsilon_{xx} + \nu^2 \varepsilon_{xx}^2)^2 \\ &= \varepsilon_{xx}^2 ((1-2\nu)^2 + 2(1-2\nu)\nu(-2+\nu) \varepsilon_{xx} + \\ &\quad + (\nu^2(-2+\nu)^2 + 2(1-2\nu)\nu^2) \varepsilon_{xx}^2 + \dots) \\ &= \varepsilon_{xx}^2 (1-2\nu)((1-2\nu) - 2\nu(2-\nu) \varepsilon_{xx}) + O(\varepsilon_{xx}^4) \\ I_1 &= (1 + \varepsilon_{xx})^2 + (1 + \varepsilon_{yy})^2 + (1 + \varepsilon_{zz})^2 \\ &= (1 + \varepsilon_{xx})^2 + (1 - \nu \varepsilon_{xx})^2 + (1 - \nu \varepsilon_{xx})^2 = 3 + (2 - 4\nu) \varepsilon_{xx} + (1 + 2\nu^2) \varepsilon_{xx}^2. \end{aligned}$$

Using *Mathematica* or *Maxima* (for the elementary, tedious operations) to find

$$\begin{aligned} J^{-2/3} &= 1 - \frac{2(1-2\nu)}{3} \varepsilon_{xx} + \frac{5-8\nu+14\nu^2}{9} \varepsilon_{xx}^2 - \\ &\quad - \frac{4(-10+15\nu-21\nu^2+35\nu^3)}{81} \varepsilon_{xx}^3 + O(\varepsilon_{xx}^4) \\ \bar{I}_1 = \frac{I_1}{J^{2/3}} &= 3 + \frac{4(1+\nu)^2}{3} \varepsilon_{xx}^2 - \frac{4(1+\nu)^2(7-11\nu)}{27} \varepsilon_{xx}^3 + O(\varepsilon_{xx}^4). \end{aligned}$$

Observe that the invariant  $I_1$  contains a contribution proportional to  $\varepsilon_{xx}$ , while  $\bar{I}_1$  does not. When minimizing the energy based on  $I_1$  this would lead to a nonzero stress  $\sigma_x \neq 0$ , even if no deformation is applied with  $\varepsilon_{xx} = 0$ . Thus one should not use the energy  $C_{10} \cdot (I_3 - 3)$  for compressible materials. Now examine

$$\begin{aligned} W &= C_{10} \cdot \left( \frac{I_1}{J^{3/2}} - 3 \right) + \frac{1}{D} (J-1)^2 \\ &= \left( C_{10} \frac{4(1+\nu)^2}{3} + \frac{(1-2\nu)^2}{D} \right) \varepsilon_{xx}^2 - \\ &\quad - \left( C_{10} \frac{4(1+\nu)^2(7-11\nu)}{27} + \frac{2(1-2\nu)\nu(2-\nu)}{D} \right) \varepsilon_{xx}^3 + O(\varepsilon_{xx}^4) \end{aligned}$$

and use

$$C_{10} = \frac{E}{4(1+\nu)} = \frac{1}{2} \mu \quad \text{and} \quad \frac{1}{D} = \frac{E}{6(1-2\nu)} = \frac{1}{2} K$$

(with shear modulus  $\mu$  and bulk modulus  $K$ , see Table 5.5 on page 360 in Example 5-35) and elementary algebra to conclude<sup>23</sup>

$$W = \left( C_{10} \frac{4(1+\nu)^2}{3} + \frac{(1-2\nu)^2}{D} \right) \varepsilon_{xx}^2 + O(\varepsilon_{xx}^3)$$

<sup>23</sup>Another approach is based on  $\lambda_2 = \lambda_3 = 1 + \varepsilon_{yy}$ .

$$\begin{aligned} W &= C_{10} \cdot \left( \frac{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}{(\lambda_1 \lambda_2 \lambda_3)^{2/3}} - 3 \right) + \frac{1}{D} (\lambda_1 \lambda_2 \lambda_3 - 1)^2 = C_{10} \cdot (\lambda_1^{4/3} \lambda_2^{-4/3} + 2 \lambda_1^{-2/3} \lambda_2^{2/3} - 3) + \frac{1}{D} (\lambda_1 \lambda_2^2 - 1)^2 \\ \frac{\partial W}{\partial \lambda_1} &= C_{10} \cdot \left( \frac{4}{3} \lambda_1^{1/3} \lambda_2^{-4/3} - \frac{4}{3} \lambda_1^{-5/3} \lambda_2^{2/3} \right) + \frac{2}{D} (\lambda_1 \lambda_2^2 - 1) \lambda_2^2 \\ \frac{\partial W}{\partial \varepsilon_{xx}} &= \frac{4}{3} C_{10} \cdot ((1 + \varepsilon_{xx})^{1/3} (1 + \varepsilon_{yy})^{-4/3} - (1 + \varepsilon_{xx})^{-5/3} (1 + \varepsilon_{yy})^{2/3}) + \frac{2}{D} ((1 + \varepsilon_{xx}) (1 + \varepsilon_{yy})^2 - 1) (1 + \varepsilon_{yy})^2 \\ &\approx \frac{4}{3} C_{10} \cdot ((1 + \frac{1}{3} \varepsilon_{xx}) (1 - \frac{4}{3} \varepsilon_{yy}) - (1 - \frac{5}{3} \varepsilon_{xx}) (1 + \frac{2}{3} \varepsilon_{yy})) + \frac{2}{D} (\varepsilon_{xx} + 2 \varepsilon_{yy}) (1 + 2 \varepsilon_{yy}) \\ &\approx \frac{4}{3} C_{10} \cdot (\frac{1}{3} \varepsilon_{xx} - \frac{4}{3} \varepsilon_{yy} + \frac{5}{3} \varepsilon_{xx} - \frac{2}{3} \varepsilon_{yy}) + \frac{2}{D} (\varepsilon_{xx} + 2 \varepsilon_{yy}) \\ &= \left( \frac{8}{3} C_{10} + \frac{2}{D} \right) \varepsilon_{xx} + \left( -\frac{6}{3} C_{10} + \frac{4}{D} \right) \varepsilon_{yy}. \end{aligned}$$

$$\begin{aligned}
&= \left( \frac{E}{4(1+\nu)} \frac{4(1+\nu)^2}{3} + \frac{E}{6(1-2\nu)} (1-2\nu)^2 \right) \varepsilon_{xx}^2 + O(\varepsilon_{xx}^3) \\
&= \left( \frac{E(1+\nu)}{3} + \frac{E}{6} (1-2\nu) \right) \varepsilon_{xx}^2 + O(\varepsilon_{xx}^3) = \frac{1}{2} E \varepsilon_{xx}^2 + O(\varepsilon_{xx}^3).
\end{aligned}$$

For small, uniaxial deformations the energy densities generated by Hooke's law and by (5.21) coincide.

To determine the stress-strain curve use

$$\begin{aligned}
\sigma_x = \frac{\partial W}{\partial \varepsilon_{xx}} &= 2 \left( C_{10} \frac{4(1+\nu)^2}{3} + \frac{(1-2\nu)^2}{D} \right) \varepsilon_{xx} + \\
&\quad + 3 \left( -C_{10} \frac{4(1+\nu)^2(7-11\nu)}{27} + \frac{2(1-2\nu)\nu(-2+\nu)}{D} \right) \varepsilon_{xx}^2 + O(\varepsilon_{xx}^3)
\end{aligned}$$

and with the above values for  $C_{10}$  and  $D$  for  $\nu = \frac{1}{2}$  this leads to

$$\sigma_x = E \varepsilon_{xx} - E \varepsilon_{xx}^2 + O(\varepsilon_{xx}^3).$$

The stress-strain curve is again shown in Figure 5.17 and identical to the approximative curve for the incompressible Neo-Hookean material.  $\diamond$

### 5-43 Example : The hydrostatic pressure situation

not in class

For the hydrostatic pressure situation work with  $\varepsilon_{xx} = \varepsilon_{yy} = \varepsilon_{zz}$  and thus

$$\begin{aligned}
\lambda &= \lambda_1 = \lambda_2 = \lambda_3 = \sqrt{(1+\varepsilon_{xx})^2} = 1 + \varepsilon_{xx} \\
I_1 &= \lambda_1^2 + \lambda_2^2 + \lambda_3^2 = 3(1+2\varepsilon_{xx} + \varepsilon_{xx}^2) \\
J &= \lambda_1 \lambda_2 \lambda_3 = (1+\varepsilon_{xx})^3
\end{aligned}$$

and this leads to the expressions used for the different models for the energy density.

$$\begin{aligned}
I_1 - 3 &= 6\varepsilon_{xx} + 3\varepsilon_{xx}^2 \\
\bar{I}_1 - 3 &= \frac{I_1}{J^{2/3}} - 3 = \frac{3(1+\varepsilon_{xx})^2}{(1+\varepsilon_{xx})^2} - 3 = 0 \\
(J-1)^2 &= (3\varepsilon_{xx} + 3\varepsilon_{xx}^2 + \varepsilon_{xx}^3)^2 = \varepsilon_{xx}^2 (3+3\varepsilon_{xx} + \varepsilon_{xx}^2)^2 \\
&= \varepsilon_{xx}^2 (9+18\varepsilon_{xx}+15\varepsilon_{xx}^2+6\varepsilon_{xx}^3+\varepsilon_{xx}^4)
\end{aligned}$$

The result  $\bar{I}_1 - 3 = 0$  shows that the invariant  $\bar{I}_1$  does not take volume changes into account, while  $I_1 - 3$  and  $(J-1)^2$  do. Now have a closer look at two models frequently used for the elastic energy density  $W$ .

- **Neo-Hookean**

$$\begin{aligned}
W &= C_{10} \cdot (I_1 - 3) = C_{10} 3 \varepsilon_{xx} (2 + \varepsilon_{xx}) \\
\sigma_x &= \frac{1}{3} \frac{\partial W}{\partial \varepsilon_{xx}} = C_{10} 2 (1 + \varepsilon_{xx})
\end{aligned}$$

This can not be a correct model, since there would be a force required to **not deform** the solid, i.e. without any external force the body would shrink. This does not make sense and is in contradiction

With  $\varepsilon_{yy} = -\nu \varepsilon_{xx}$  and the above expressions for  $C_{10}$  and  $\frac{1}{D}$  this leads to

$$\begin{aligned}
\frac{\partial W}{\partial \varepsilon_{xx}} &\approx \left( \frac{8}{3} C_{10} + \frac{2}{D} \right) \varepsilon_{xx} - \nu \left( -\frac{8}{3} C_{10} + \frac{4}{D} \right) \varepsilon_{xx} = \left( \frac{8+8\nu}{3} C_{10} + \frac{2-4\nu}{D} \right) \varepsilon_{xx} \\
&= \left( \frac{8+8\nu}{3} \frac{E}{4(1+\nu)} + \frac{(2-4\nu)E}{6(1-2\nu)} \right) \varepsilon_{xx} = \left( \frac{2E}{3} + \frac{E}{3} \right) \varepsilon_{xx} = E \varepsilon_{xx}.
\end{aligned}$$

This confirms Hooke's law for small strains.

to the commonly used “Neo–Hookean is for incompressible solids”. Obviously any type of incompressible material model will struggle with the hydrostatic situation. But the material is assumed to be incompressible, thus the three stretch factors  $\lambda_i$  are not independent. Use  $\lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1$  to conclude  $\lambda_3 = \frac{1}{\lambda_1 \cdot \lambda_2}$ , which contradicts  $\lambda_1 = \lambda_2 = \lambda_3$ .

- In [Bowe10, §3.5.5] find a model with the energy density given by

$$\begin{aligned} W &= C_{10} \cdot (\bar{I}_1 - 3) + \frac{1}{D} (J - 1)^2 = C_{10} \cdot \left( \frac{I_1}{J^{2/3}} - 3 \right) + \frac{1}{D} (J - 1)^2 \\ &= C_{10} 0 + \frac{1}{D} \varepsilon_{xx}^2 (9 + 18 \varepsilon_{xx} + 15 \varepsilon_{xx}^2 + 6 \varepsilon_{xx}^3 + \varepsilon_{xx}^4) \\ \sigma_x &= \frac{1}{3} \frac{\partial W}{\partial \varepsilon_{xx}} = \frac{1}{D} (6 \varepsilon_{xx} + 18 \varepsilon_{xx}^2 + 20 \varepsilon_{xx}^3 + 10 \varepsilon_{xx}^4 + 2 \varepsilon_{xx}^5). \end{aligned}$$

In Example 5–32 Hooke’s linear law was used to find

$$\begin{aligned} -p = \sigma_x &= \frac{E}{1 - 2\nu} \varepsilon_{xx} < 0 \\ W &= \frac{1}{2} \frac{1 - 2\nu}{E} 3 \sigma_x^2 = \frac{1}{2} \frac{E}{1 - 2\nu} 3 \varepsilon_{xx}^2. \end{aligned}$$

Comparing the two results for  $\sigma_x$  for small  $\varepsilon_{xx}$  leads to

$$\frac{6}{D} = \frac{E}{1 - 2\nu} \implies \frac{1}{D} = \frac{E}{6(1 - 2\nu)} = \frac{1}{2} K.$$

where  $K$  is the bulk modulus, see page 360. This is consistent with the results generated using uniaxial loading in Example 5–42.  $\diamond$

#### 5–44 Example : Pure shearing with a Neo–Hookean energy density

Examine a pure shearing situation, i.e. the a plane  $y = \text{const}$  is shifted in  $x$ -direction. This is achieved with a displacement  $\vec{u} = (u_1, u_2, u_3) = \frac{\partial u_1}{\partial y} (y, 0, 0)$ . The displacement gradient tensor is then given by

$$\mathbf{DU} = \begin{bmatrix} 0 & \frac{\partial u_1}{\partial y} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and thus the Cauchy–Green deformation tensor is

$$\mathbf{C} = \mathbb{I} + \mathbf{DU} + \mathbf{DU}^T + \mathbf{DU}^T \mathbf{DU} = \begin{bmatrix} 1 & \frac{\partial u_1}{\partial y} & 0 \\ \frac{\partial u_1}{\partial y} & 1 + (\frac{\partial u_1}{\partial y})^2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Example 5–33 (page 358) shows that this deformation minimizes the Hookean elastic energy density  $W$  for a given  $\varepsilon_{xy}$ , i.e. this deformation is a consequence of Bernoulli’s principle.

The principal stretches  $\lambda_i$  are the square roots of the eigenvalues of the tensor  $\mathbf{C}$  and thus with  $\frac{\partial u_1}{\partial y} = 2\varepsilon_{xy}$  find the invariants<sup>24</sup>

$$\begin{aligned} \lambda_1^2 + \lambda_2^2 + \lambda_3^2 &= \text{trace}(\mathbf{C}) = 3 + (\frac{\partial u_1}{\partial y})^2 = 3 + 4\varepsilon_{xy}^2 \\ \lambda_1^2 \cdot \lambda_2^2 \cdot \lambda_3^2 &= \det(\mathbf{C}) = 1 \end{aligned}$$

<sup>24</sup>The invariants could be computed using the eigenvalues  $\mu_{1,2}$ , but this is not as elegant. Use  $\alpha := (\frac{\partial u_1}{\partial y})^2 = 4\varepsilon_{xy}^2$ .

$$\begin{aligned} 0 &= (1 - \mu)(1 + (\frac{\partial u_1}{\partial y})^2 - \mu) - (\frac{\partial u_1}{\partial y})^2 = \mu^2 - \mu(2 + (\frac{\partial u_1}{\partial y})^2) + 1 - (\frac{\partial u_1}{\partial y})^2 = \mu^2 - \mu(2 + \alpha) + 1 - \alpha \\ \mu_{1,2} &= \frac{1}{2} \left( 2 + \alpha \pm \sqrt{(2 + \alpha)^2 - 4(1 - \alpha)} \right) = \frac{1}{2} \left( 2 + \alpha \pm \sqrt{4\alpha + \alpha^2 + 4\alpha} \right) \end{aligned}$$

This shows that the shearing deformation does not compress the material, i.e. the volume is preserved. The Neo-Hookean energy density is

$$W = C_{10} \left( \frac{I_1}{J^{2/3}} - 3 \right) + \frac{1}{D} (J - 1)^2 = C_{10} 4 \varepsilon_{xy}^2.$$

This leads to the shearing stress

$$\tau_{xy} = \frac{1}{2} \frac{\partial W}{\partial \varepsilon_{xy}} = 4 C_{10} \varepsilon_{xy}.$$

Comparing with the similar result based on Hooke's law in Example 5–33 ( $\tau_{xy} = \frac{E}{1+\nu} \varepsilon_{xy}$ ) leads to  $4 C_{10} = \frac{E}{1+\nu}$ . For incompressible materials with  $\nu = \frac{1}{2}$  find  $C_{10} = \frac{E}{4(1+\frac{1}{2})} = \frac{1}{6} E$ . This is consistent with the previous examples. These results coincide with the results for an incompressible Neo-Hookean energy density  $W = C_{10} (I_1 - 3)$ , since  $J = \det(\mathbf{C}) = 1$ .

In Example 5–42 a uniaxial loading leads to the stress–strain curve

$$\sigma_x = \frac{\partial W}{\partial \varepsilon_{xx}} \approx 2 \left( C_{10} \frac{4(1+\nu)^2}{3} + \frac{(1-2\nu)^2}{D} \right) \varepsilon_{xx}$$

and the hydrostatic situation in Example 5–43 shows

$$\sigma_x = \frac{1}{3} \frac{\partial W}{\partial \varepsilon_{xx}} \approx \frac{6}{D} \varepsilon_{xx}$$

and the above shearing situation leads to

$$\tau_{xy} = \frac{1}{2} \frac{\partial W}{\partial \varepsilon_{xy}} \approx 4 C_{10} \varepsilon_{xy}$$

Thus the combination of the three results allows to determine  $C_{10}$ ,  $\frac{1}{D}$  and Poisson's ratio  $\nu$ .  $\diamond$

### 5–45 Example : Neo-Hookean energy density for compressible materials, uniaxial loading

not in class

In [BoneWood08, §6.4] (or [Shab08, p. 146]) find an energy density for Neo-Hookean, compressible materials, given by

$$W = \frac{\mu}{2} (I_1 - 3) - \mu \ln J + \frac{\lambda}{2} (\ln J)^2 \quad (5.22)$$

using two material constants (see Table 5.5 on page 360)

$$\text{shear modulus } \mu = \frac{E}{2(1+\nu)} \quad \text{and} \quad \text{Lamé parameter } \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}.$$

Using the FEM software COMSOL these two constants have to be given when using the Neo-Hookean material law. Now verify that this is consistent with Hooke's law for small deformations. For small, uniaxial deformations compute with  $\lambda_1 = 1 + \varepsilon_{xx}$  and  $\lambda_2 = \lambda_3 = 1 - \nu \varepsilon_{xx}$ , leading to

$$\begin{aligned} I_1 - 3 &= \lambda_1^2 + \lambda_2^2 + \lambda_3^2 - 3 = 2(1-2\nu)\varepsilon_{xx} + (1+2\nu^2)\varepsilon_{xx}^2 \\ J &= \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = (1+\varepsilon_{xx})(1-\nu\varepsilon_{xx})^2 = 1 + (1-2\nu)\varepsilon_{xx} + (-2\nu+\nu^2)\varepsilon_{xx}^2 + \nu^2\varepsilon_{xx}^3 \\ \ln x &\approx x - \frac{1}{2}x^2 \quad \text{for } |x| \ll 1 \quad \text{Taylor approximation} \\ \ln J &\approx +(1-2\nu)\varepsilon_{xx} + (-2\nu+\nu^2)\varepsilon_{xx}^2 - \frac{1}{2}(1-2\nu)^2\varepsilon_{xx}^2 = +(1-2\nu)\varepsilon_{xx} - \frac{1}{2}(1+2\nu^2)\varepsilon_{xx}^2 \\ (\ln J)^2 &\approx +(1-2\nu)^2\varepsilon_{xx}^2. \\ &= 1 + \frac{\alpha}{2} \pm \frac{1}{2}\sqrt{8\alpha + \alpha^2} = 1 + 2\varepsilon_{xy}^2 \pm \sqrt{16\varepsilon_{xy}^2 + 16\varepsilon_{xy}^4} = 1 + 2\varepsilon_{xy}^2 \pm 2\sqrt{\varepsilon_{xy}^2 + \varepsilon_{xy}^4} \end{aligned}$$

The principal stretches are  $\lambda_{1,2} = \sqrt{\mu_{1,2}}$  and  $\lambda_3 = 1$ , leading to the invariants

$$\begin{aligned} I_1 &= \lambda_1^2 + \lambda_2^2 + \lambda_3^2 = 1 + 2\varepsilon_{xy}^2 + 2\sqrt{\varepsilon_{xy}^2 + \varepsilon_{xy}^4} + 1 + 2\varepsilon_{xy}^2 - 2\sqrt{\varepsilon_{xy}^2 + \varepsilon_{xy}^4} + 1 = 3 + 4\varepsilon_{xy}^2 \\ J &= \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = (1+2\varepsilon_{xy}^2)^2 - 4(\varepsilon_{xy}^2 + \varepsilon_{xy}^4) = 1. \end{aligned}$$

With these expressions use the energy density  $W$  in (5.22) and conclude

$$\begin{aligned}
 W &= \frac{\mu}{2} (I_1 - 3) - \mu \ln J + \frac{\lambda}{2} (\ln J)^2 \\
 &\approx \frac{\mu}{2} 2(1 - 2\nu) \varepsilon_{xx} + \frac{\mu}{2} (1 + 2\nu^2) \varepsilon_{xx}^2 - \mu \left( (1 - 2\nu) \varepsilon_{xx} - \frac{1}{2} (1 + 2\nu^2) \varepsilon_{xx}^2 \right) + \frac{\lambda}{2} (1 - 2\nu)^2 \varepsilon_{xx}^2 \\
 &= +\mu (1 + 2\nu^2) \varepsilon_{xx}^2 + \frac{\lambda}{2} (1 - 2\nu)^2 \varepsilon_{xx}^2 = +\frac{E}{2(1+\nu)} (1 + 2\nu^2) \varepsilon_{xx}^2 + \frac{\nu E (1 - 2\nu)}{2(1+\nu)} \varepsilon_{xx}^2 \\
 &= E \left( \frac{(1 + 2\nu^2) + \nu - 2\nu^2}{2(1+\nu)} \right) \varepsilon_{xx}^2 = \frac{E}{2} \varepsilon_{xx}^2.
 \end{aligned}$$

Thus the Neo-Hookean energy expression (5.22) is consistent with the usual energy density based on Hooke's law for linear materials.

To obtain the stress-strain curve in Figure 5.19 determine  $\sigma_x = \frac{\partial W}{\partial \varepsilon_{xx}}$ . Using

$$\begin{aligned}
 \frac{\partial (I_1 - 3)}{\partial \varepsilon_{xx}} &= 2(1 - 2\nu) + 2(1 + 2\nu^2) \varepsilon_{xx} \\
 \frac{\partial J}{\partial \varepsilon_{xx}} &= (1 - 2\nu) + 2(-2\nu + \nu^2) \varepsilon_{xx} + 3\nu^2 \varepsilon_{xx}^2 \\
 \frac{\partial \ln J}{\partial \varepsilon_{xx}} &= \frac{1}{J} \frac{\partial J}{\partial \varepsilon_{xx}} = \frac{1}{J} ((1 - 2\nu) + 2(-2\nu + \nu^2) \varepsilon_{xx} + 3\nu^2 \varepsilon_{xx}^2) \\
 \frac{\partial (\ln J)^2}{\partial \varepsilon_{xx}} &= \frac{2 \ln J}{J} \frac{\partial J}{\partial \varepsilon_{xx}} = \frac{2 \ln J}{J} ((1 - 2\nu) + 2(-2\nu + \nu^2) \varepsilon_{xx} + 3\nu^2 \varepsilon_{xx}^2)
 \end{aligned}$$

obtain

$$\begin{aligned}
 \sigma_x &= \frac{\partial W}{\partial \varepsilon_{xx}} = \frac{\partial}{\partial \varepsilon_{xx}} \left( \frac{\mu}{2} (I_1 - 3 - 2 \ln J) + \frac{\lambda}{2} (\ln J)^2 \right) \\
 &= \mu \left( (1 - 2\nu) + (1 + 2\nu^2) \varepsilon_{xx} - \frac{1}{J} ((1 - 2\nu) + 2(-2\nu + \nu^2) \varepsilon_{xx} + 3\nu^2 \varepsilon_{xx}^2) \right) + \\
 &\quad + \frac{\lambda \ln J}{J} ((1 - 2\nu) + 2(-2\nu + \nu^2) \varepsilon_{xx} + 3\nu^2 \varepsilon_{xx}^2).
 \end{aligned}$$

For the value  $\nu = 0.3$  Figure 5.19 shows the resulting stress-strain curves for the Neo-Hookean and Hooke's law.  $\diamond$

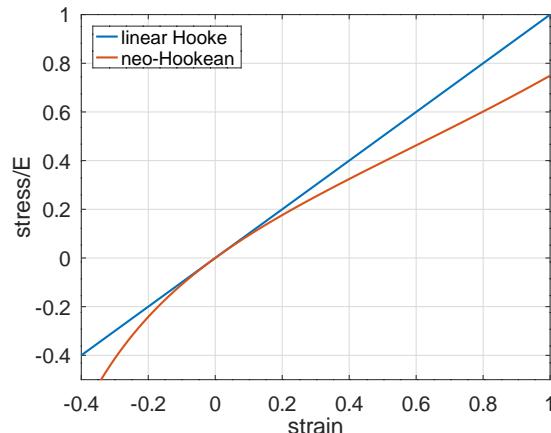


Figure 5.19: Stress-strain curve for Hooke's linear law and a compressible Neo-Hookean material with  $\nu = 0.3$  under uniaxial load

**5–46 Example : Neo-Hookean energy density for compressible materials, hydrostatic loading**

For hydrostatic loading use  $\lambda_1 = \lambda_2 = \lambda_3 = 1 + \varepsilon_{xx}$ . This leads to

$$\begin{aligned} I_1 - 3 &= \lambda_1^2 + \lambda_2^2 + \lambda_3^2 - 3 = 3(1 + \varepsilon_{xx})^2 - 3 = 6\varepsilon_{xx} + 3\varepsilon_{xx}^2 \\ J &= \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = (1 + \varepsilon_{xx})^3 \\ \ln J &= 3 \ln(1 + \varepsilon_{xx}) \approx 3(\varepsilon_{xx} - \frac{1}{2}\varepsilon_{xx}^2) \\ (\ln J)^2 &\approx 9\varepsilon_{xx}^2 (1 - \frac{1}{2}\varepsilon_{xx})^2 \approx 9\varepsilon_{xx}^2 (1 - \varepsilon_{xx}). \end{aligned}$$

Using the energy density find

$$\begin{aligned} W &= \frac{\mu}{2}(I_1 - 3) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2 \\ &\approx \frac{\mu}{2}(6\varepsilon_{xx} + 3\varepsilon_{xx}^2) - \mu 3(\varepsilon_{xx} - \frac{1}{2}\varepsilon_{xx}^2) + \frac{\lambda}{2}9\varepsilon_{xx}^2(1 - \varepsilon_{xx}) \\ &\approx (3\mu + \frac{9}{2}\lambda)\varepsilon_{xx}^2 = \left(\frac{3E}{2(1+\nu)} + \frac{9\nu E}{2(1+\nu)(1-2\nu)}\right)\varepsilon_{xx}^2 \\ &= \frac{3E(1-2\nu) + 9\nu E}{2(1+\nu)(1-2\nu)}\varepsilon_{xx}^2 = \frac{3E}{2(1-2\nu)}\varepsilon_{xx}^2. \end{aligned}$$

This coincides with the energy density determined by Hooke's linear law in Example 5–32 on page 357. ◇

**5–47 Example : Neo-Hookean energy density for compressible materials, pure shearing**

Based on Example 5–44 use

$$\begin{aligned} I_1 &= \lambda_1^2 + \lambda_2^2 + \lambda_3^2 = \text{trace}(\mathbf{C}) = 3 + (\frac{\partial u_1}{\partial y})^2 = 3 + 4\varepsilon_{xy}^2 \\ J &= \lambda_1^2 \cdot \lambda_2^2 \cdot \lambda_3^2 = \det(\mathbf{C}) = 1 = J \\ W &= \frac{\mu}{2}(I_1 - 3) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2 = \frac{\mu}{2}4\varepsilon_{xy}^2. \end{aligned}$$

This leads to the shearing stress

$$\tau_{xy} = \frac{1}{2} \frac{\partial W}{\partial \varepsilon_{xy}} = \mu 4\varepsilon_{xy} = \frac{E}{(1+\nu)}\varepsilon_{xy}.$$

For small deformations this is consistent with Hooke's law. ◇

**5.7.3 Ogden and Mooney–Rivlin Energy Density Models**

not in class

**Ogden energy density model for incompressible materials**

For rubber it is common to use Ogden's energy density to describe the elastic behavior. The general formula is given by

$$W = \sum_{i=1}^N \frac{\mu_i}{\alpha_i} (\lambda_1^{\alpha_i} + \lambda_2^{\alpha_i} + \lambda_3^{\alpha_i} - 3),$$

where  $\lambda_i$  are the principal stretches, i.e. the square roots of the eigenvalues of the Cauchy–Green deformation tensor. Since in the above expression all eigenvalues are raised to the same power, the expression is invariant under coordinate rotations.

The simplest case for Ogden's energy density is for  $N = 1$ .

$$W = \frac{\mu}{\alpha} (\lambda_1^\alpha + \lambda_2^\alpha + \lambda_3^\alpha - 3) \quad \text{subject to} \quad J = \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1 \quad (5.23)$$

- Considering  $\alpha = 2$  we find

$$W = \frac{\mu}{2} (\lambda_1^2 + \lambda_2^2 + \lambda_3^2 - 3) = \frac{\mu}{2} (I_1 - 3) = C_{10} \cdot (I_1 - 3)$$

i.e. the Neo-Hookean energy density (5.20) is a special case of Ogden's law.

- With  $N = 2$ ,  $\alpha_1 = 2$ ,  $\alpha_2 = -2$  and  $\lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1$  find

$$\begin{aligned} W &= \frac{\mu_1}{2} (\lambda_1^2 + \lambda_2^2 + \lambda_3^2 - 3) + \frac{\mu_2}{2} (\lambda_1^{-2} + \lambda_2^{-2} + \lambda_3^{-2} - 3) \\ &= \frac{\mu_1}{2} (\lambda_1^2 + \lambda_2^2 + \lambda_3^2 - 3) + \frac{\mu_2}{2} (\lambda_2^2 \lambda_3^2 + \lambda_1^2 \lambda_3^2 + \lambda_1^2 \lambda_2^2 - 3) \\ &= C_{10} \cdot (I_1 - 3) + C_{01} \cdot (I_2 - 3) \end{aligned}$$

which is the energy density for a Mooney-Rivlin material law. For the incompressible case one might use

$$W = C_{10} \cdot (\bar{I}_1 - 3) + C_{01} \cdot (\bar{I}_2 - 3) = C_{10} \cdot (I_1 - 3) + C_{01} \cdot (I_2 - 3),$$

where

$$\bar{I}_2 = \frac{I_2}{J^{4/3}} = \frac{\lambda_2^2 \lambda_3^2 + \lambda_1^2 \lambda_3^2 + \lambda_1^2 \lambda_2^2}{(\lambda_1 \lambda_2 \lambda_3)^{4/3}} = \lambda_1^{-4/3} \lambda_2^{2/3} \lambda_3^{2/3} + \lambda_1^{2/3} \lambda_2^{-4/3} \lambda_3^{2/3} + \lambda_1^{2/3} \lambda_2^{2/3} \lambda_3^{-4/3}.$$

- With  $C_{01} = 0$  the Mooney-Rivlin energy density is equal to the Neo-Hookean energy density in Example 5-40.
- Observe that minimizing the total energy, using the integral of the energy density  $W$ , does not respect the incompressibility constraint  $J = \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1$ . This has to be taken care of in the design of the algorithm.
- Using  $\bar{\lambda}_i = \lambda_i / J^{1/3}$  also find

$$W = \frac{\mu}{\alpha} (\bar{\lambda}_1^\alpha + \bar{\lambda}_2^\alpha + \bar{\lambda}_3^\alpha - 3) \quad (5.24)$$

as definition for Ogden energies. The FEM code ABAQUS is using this setup. Since  $\bar{\lambda}_1 \cdot \bar{\lambda}_2 \cdot \bar{\lambda}_3 = 1$  it seems that the incompressibility constraint is built in. In fact (5.23) and (5.24) are equivalent in the case of incompressible materials.

#### 5-48 Example : Mooney-Rivlin energy density for incompressible materials, uniaxial loading

For the Mooney-Rivlin energy density work with

$$\begin{aligned} W &= C_{10} \cdot (\bar{I}_1 - 3) + C_{01} \cdot (\bar{I}_2 - 3) \\ &= C_{10} \cdot (\bar{\lambda}_1^2 + \bar{\lambda}_2^2 + \bar{\lambda}_3^2 - 3) + C_{01} \cdot (\bar{\lambda}_2^2 \bar{\lambda}_3^2 + \bar{\lambda}_1^2 \bar{\lambda}_3^2 + \bar{\lambda}_1^2 \bar{\lambda}_2^2 - 3) \\ &= C_{10} \cdot (\lambda_1^{4/3} \lambda_2^{-2/3} \lambda_3^{-2/3} + \lambda_1^{-2/3} \lambda_2^{4/3} \lambda_3^{-2/3} + \lambda_1^{-2/3} \lambda_2^{-2/3} \lambda_3^{4/3} - 3) + \\ &\quad + C_{01} \cdot (\lambda_1^{-4/3} \lambda_2^{2/3} \lambda_3^{2/3} + \lambda_1^{2/3} \lambda_2^{-4/3} \lambda_3^{2/3} + \lambda_1^{2/3} \lambda_2^{2/3} \lambda_3^{-4/3} - 3). \end{aligned}$$

In the case of uniaxial, incompressible loading we use  $\lambda_1 = 1 + \varepsilon_{xx}$  and  $\lambda_2 = \lambda_3 = \lambda_1^{-1/2}$ , leading to

$$\begin{aligned} W &= C_{10} \cdot (\lambda_1^{4/3} \lambda_1^{2/3} + 2 \lambda_1^{-2/3} \lambda_1^{-1/3} - 3) + C_{01} \cdot (\lambda_1^{-4/3} \lambda_2^{4/3} + 2 \lambda_1^{2/3} \lambda_2^{-2/3} - 3) \\ &= C_{10} \cdot (\lambda_1^2 + 2 \lambda_1^{-1} - 3) + C_{01} \cdot (\lambda_1^{-2} + 2 \lambda_1 - 3) \\ &= C_{10} \cdot ((1 + \varepsilon_{xx})^2 + 2(1 + \varepsilon_{xx})^{-1} - 3) + C_{01} \cdot ((1 + \varepsilon_{xx})^{-2} + 2(1 + \varepsilon_{xx}) - 3) \\ &\approx C_{10} \cdot (2 \varepsilon_{xx} + \varepsilon_{xx}^2 - 2 \varepsilon_{xx} + 2 \varepsilon_{xx}^2) + C_{01} \cdot (-2 \varepsilon_{xx} + 3 \varepsilon_{xx}^2 + 2 \varepsilon_{xx}) \\ &= C_{10} 3 \varepsilon_{xx}^2 + C_{01} 3 \varepsilon_{xx}^2 = 3(C_{10} + C_{01}) \varepsilon_{xx}^2 = \frac{1}{2} E \varepsilon_{xx}^2. \end{aligned}$$

As a consequence obtain for small strains

$$C_{10} + C_{01} = \frac{1}{6} E.$$

To determine the stress in  $x$ -direction examine

$$\begin{aligned}\sigma_x = \frac{\partial W}{\partial \varepsilon_{xx}} &= C_{10} \cdot (2(1 + \varepsilon_{xx}) - 2(1 + \varepsilon_{xx})^{-2}) + C_{01} \cdot (-2(1 + \varepsilon_{xx})^{-3} + 2) \\ &\approx C_{10} \cdot (2\varepsilon_{xx} + 4\varepsilon_{xx}^2 - 6\varepsilon_{xx}^2 + 8\varepsilon_{xx}^3 - 10\varepsilon_{xx}^4) + \\ &\quad + C_{01} \cdot (6\varepsilon_{xx} - 12\varepsilon_{xx}^2 + 20\varepsilon_{xx}^3 - 30\varepsilon_{xx}^4) \\ &= 6(C_{10} + C_{01})\varepsilon_{xx} - (6C_{10} + 12C_{01})\varepsilon_{xx}^2 + \\ &\quad + (8C_{10} + 20C_{01})\varepsilon_{xx}^3 - (10C_{10} + 30C_{01})\varepsilon_{xx}^4.\end{aligned}$$

Find the resulting stress-strain curves in Figure 5.20.  $\diamond$

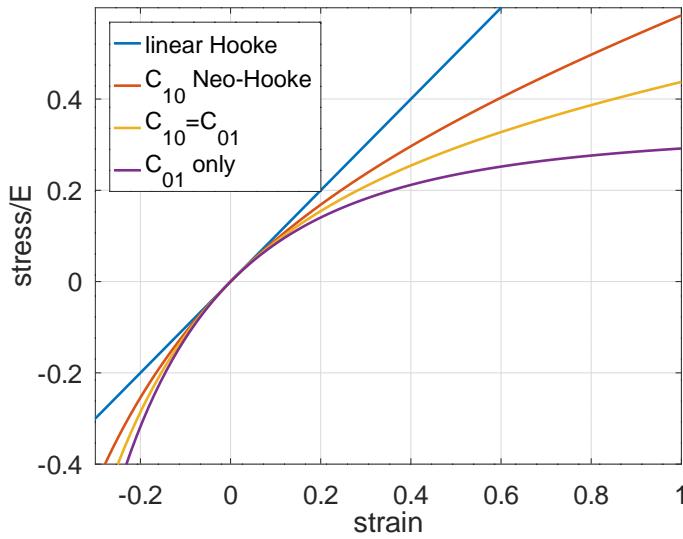


Figure 5.20: Stress-strain curve for Hooke's linear law and three cases of incompressible Mooney–Rivlin with uniaxial loading: (1):  $C_{10} = \frac{E}{6}$ ,  $C_{01} = 0$ , i.e. Neo–Hooke, (2):  $C_{10} = C_{01} = \frac{E}{12}$ , (3):  $C_{10} = 0$ ,  $C_{01} = \frac{E}{6}$ .

#### 5-49 Example : Mooney–Rivlin energy density for compressible materials, hydrostatic and uniaxial loading

For the compressible Mooney–Rivlin energy density use

$$\begin{aligned}W &= C_{10} \cdot (\bar{I}_1 - 3) + C_{01} \cdot (\bar{I}_2 - 3) + \frac{1}{D} (J - 1)^2 \\ &= C_{10} \cdot (\bar{\lambda}_1^2 + \bar{\lambda}_2^2 + \bar{\lambda}_3^2 - 3) + C_{01} \cdot (\bar{\lambda}_2^2 \bar{\lambda}_3^2 + \bar{\lambda}_1^2 \bar{\lambda}_3^2 + \bar{\lambda}_1^2 \bar{\lambda}_2^2 - 3) + \frac{1}{D} (\lambda_1 \lambda_2 \lambda_3 - 1)^2 \\ &= C_{10} \cdot (\lambda_1^{4/3} \lambda_2^{-2/3} \lambda_3^{-2/3} + \lambda_1^{-2/3} \lambda_2^{4/3} \lambda_3^{-2/3} + \lambda_1^{-2/3} \lambda_2^{-2/3} \lambda_3^{4/3} - 3) + \\ &\quad + C_{01} \cdot (\lambda_1^{-4/3} \lambda_2^{2/3} \lambda_3^{2/3} + \lambda_1^{2/3} \lambda_2^{-4/3} \lambda_3^{2/3} + \lambda_1^{2/3} \lambda_2^{2/3} \lambda_3^{-4/3} - 3) + \frac{1}{D} (\lambda_1 \lambda_2 \lambda_3 - 1)^2.\end{aligned}$$

- For the case of hydrostatic loading use  $\bar{\lambda}_i = \lambda_i = 1 + \varepsilon_{xx}$  and thus  $\bar{I}_1 = \bar{I}_2 = 3$  and minimize  $W(\varepsilon_{xx}) = \frac{1}{D} (J - 1)^2$ . The setup is identical to the compressible Neo–Hookean case in Example 5–43. This leads to

$$W = \frac{1}{D} (J - 1)^2 = +\frac{1}{D} \varepsilon_{xx}^2 (9 + 18\varepsilon_{xx} + 15\varepsilon_{xx}^2 + 6\varepsilon_{xx}^3 + \varepsilon_{xx}^4)$$

$$\sigma_x = \frac{1}{3} \frac{\partial W}{\partial \varepsilon_{xx}} = \frac{1}{D} (6 \varepsilon_{xx} + 18 \varepsilon_{xx}^2 + O(\varepsilon_{xx}^3))$$

and

$$\frac{6}{D} = \frac{E}{1 - 2\nu} \implies \frac{1}{D} = \frac{E}{6(1 - 2\nu)} = \frac{1}{2} K.$$

- In the case of uniaxial, compressible loading use  $\lambda_1 = 1 + \varepsilon_{xx}$  and  $\lambda_2 = \lambda_3 = 1 + \varepsilon_{yy}$ , leading to

$$\begin{aligned} W &= C_{10} \cdot (\lambda_1^{4/3} \lambda_2^{-4/3} + 2 \lambda_1^{-2/3} \lambda_2^{+2/3} - 3) + C_{01} \cdot (\lambda_1^{-4/3} \lambda_2^{4/3} + 2 \lambda_1^{2/3} \lambda_2^{-2/3} - 3) + \frac{1}{D} (\lambda_1 \lambda_2^2 - 1)^2 \\ &= C_{10} \cdot ((1 + \varepsilon_{xx})^{4/3} (1 + \varepsilon_{yy})^{-4/3} + 2 (1 + \varepsilon_{xx})^{-2/3} (1 + \varepsilon_{yy})^{2/3} - 3) + \\ &\quad + C_{01} \cdot ((1 + \varepsilon_{xx})^{-4/3} (1 + \varepsilon_{yy})^{4/3} + 2 (1 + \varepsilon_{xx})^{2/3} (1 + \varepsilon_{yy})^{-2/3} - 3) + \\ &\quad + \frac{1}{D} ((1 + \varepsilon_{xx}) (1 + \varepsilon_{yy})^2 - 1)^2. \end{aligned}$$

For an uniaxial loading the value of  $\varepsilon_{yy}$  is determined as minimizer of  $W(\varepsilon_{xx}, \varepsilon_{yy})$  with respect to  $\varepsilon_{yy}$ .

$$\begin{aligned} 0 = \frac{\partial W}{\partial \varepsilon_{yy}} &= C_{10} \cdot \left( \frac{-4}{3} (1 + \varepsilon_{xx})^{4/3} (1 + \varepsilon_{yy})^{-7/3} + \frac{4}{3} (1 + \varepsilon_{xx})^{-2/3} (1 + \varepsilon_{yy})^{-1/3} \right) + \\ &\quad + C_{01} \cdot \left( \frac{4}{3} (1 + \varepsilon_{xx})^{-4/3} (1 + \varepsilon_{yy})^{1/3} - \frac{4}{3} (1 + \varepsilon_{xx})^{2/3} (1 + \varepsilon_{yy})^{-5/3} \right) + \\ &\quad + \frac{4}{D} ((1 + \varepsilon_{xx}) (1 + \varepsilon_{yy})^2 - 1) (1 + \varepsilon_{yy}) \\ &\approx \frac{4}{3} C_{10} \cdot \left( -1 - \frac{4}{3} \varepsilon_{xx} + \frac{7}{3} \varepsilon_{yy} + 1 - \frac{2}{3} \varepsilon_{xx} - \frac{1}{3} \varepsilon_{yy} \right) + \\ &\quad - \frac{4}{3} C_{01} \cdot \left( 1 - \frac{4}{3} \varepsilon_{xx} + \frac{1}{3} \varepsilon_{yy} - 1 - \frac{2}{3} \varepsilon_{xx} + \frac{5}{3} \varepsilon_{yy} \right) + \frac{4}{D} (\varepsilon_{xx} + 2 \varepsilon_{yy}) \\ &= \frac{8}{3} C_{10} \cdot (-\varepsilon_{xx} + \varepsilon_{yy}) + \frac{8}{3} C_{01} \cdot (-\varepsilon_{xx} + \varepsilon_{yy}) + \frac{4}{D} (\varepsilon_{xx} + 2 \varepsilon_{yy}) \\ &= \left( \frac{4}{D} - \frac{8}{3} (C_{10} + C_{01}) \right) \varepsilon_{xx} + \left( 2 \frac{4}{D} + \frac{8}{3} (C_{10} + C_{01}) \right) \varepsilon_{yy} \\ \varepsilon_{yy} &= -\frac{\frac{4}{D} - \frac{8}{3} (C_{10} + C_{01})}{2 \frac{4}{D} + \frac{8}{3} (C_{10} + C_{01})} \varepsilon_{xx} = -\frac{3 - 2D(C_{10} + C_{01})}{6 + 2D(C_{10} + C_{01})} \varepsilon_{xx} \\ &= -\left( \frac{1}{2} - \frac{3D(C_{10} + C_{01})}{6 + 2D(C_{10} + C_{01})} \right) \varepsilon_{xx} = -\nu \varepsilon_{xx} \\ &= -\left( \frac{1}{2} - \frac{D}{2} (C_{10} + C_{01}) + O(D^2) \right) \varepsilon_{xx} = -\left( \frac{1}{2} - D \frac{E}{12} + O(D^2) \right) \varepsilon_{xx} \end{aligned}$$

For the incompressible case  $D = 0$  obtain  $\nu = \frac{1}{2}$  again.

For small strains use the energy density  $W$  and with the help of *Mathematica* generate a Taylor approximation for  $W$  with respect to  $\varepsilon_{xx}$ , using Poisson's law in the form  $\varepsilon_{yy} = -\frac{1}{2}(1 - D(C_{10} + C_{01}))\varepsilon_{xx}$ .

$$\begin{aligned} W(\varepsilon_{xx}) &\approx \frac{1}{3} (C_{10} + C_{01}) (9 - 3(C_{10} + C_{01})D + (C_{10} + C_{01})^2 D^2) \varepsilon_{xx}^2 + \\ &\quad + \left( -2(C_{10} + 2C_{01}) + \frac{1}{2} (3C_{10} + C_{01})(C_{10} + C_{01})D + \right. \\ &\quad \left. + \frac{1}{3} (C_{10} + C_{01})^2 (5C_{10} + 3C_{01})D^2 - \frac{1}{54} (C_{10} + C_{01})^3 (11C_{10} + 7C_{01})D^2 \right) \varepsilon_{xx}^3 \end{aligned}$$

This leads to the relation to Young's modulus

$$\begin{aligned} E &= \frac{2}{3} (C_{10} + C_{01}) (9 - 3(C_{10} + C_{01})D + (C_{10} + C_{01})^2 D^2)) \\ &= 6(C_{10} + C_{01}) - 2(C_{10} + C_{01})^2 D + \frac{2}{3}(C_{10} + C_{01})^3 D^2). \end{aligned} \quad (5.25)$$

The normal stress  $\sigma_x$  is given by

$$\sigma_x = \frac{\partial W}{\partial \varepsilon_{xx}} \approx \frac{2}{3} (C_{10} + C_{01}) (9 - 3(C_{10} + C_{01})D + (C_{10} + C_{01})^2 D^2) \varepsilon_{xx}.$$

For the incompressible case  $D = 0$  this simplifies to

$$\begin{aligned} W(\varepsilon_{xx}) &\approx 3(C_{10} + C_{01}) \varepsilon_{xx}^2 - 2(C_{10} + 2C_{01}) \varepsilon_{xx}^3 \\ \sigma_x &= \frac{\partial W}{\partial \varepsilon_{xx}} \approx 6(C_{10} + C_{01}) \varepsilon_{xx}. \end{aligned}$$

This is consistent with the property  $6(C_{10} + C_{01}) = E$  with the uniaxial loading of incompressible Mooney–Rivlin material in Example 5–48.

◇

### 5–50 Example : Ogden energy density for incompressible materials, uniaxial loading

If the material is assumed to be incompressible use the energy density in equation (5.23) and  $J = \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1$  to conclude  $\lambda_2 = \lambda_3 = \lambda_1^{-1/2}$ . With  $\lambda_1 = 1 + \varepsilon_{xx}$  this leads to (for small values of  $\varepsilon_{xx}$ )

$$\begin{aligned} W &= \frac{\mu}{\alpha} (\lambda_1^\alpha + \lambda_2^\alpha + \lambda_3^\alpha - 3) \quad \text{subject to} \quad J = \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1 \\ W &= \frac{\mu}{\alpha} (\lambda_1^\alpha + 2\lambda_1^{-\alpha/2} - 3) \\ &= \frac{\mu}{\alpha} ((1 + \varepsilon_{xx})^\alpha + 2(1 + \varepsilon_{xx})^{-\alpha/2} - 3) \quad \text{use a Taylor approximation} \\ &\approx \frac{\mu}{\alpha} (1 + \alpha \varepsilon_{xx} + \frac{\alpha}{2}(\alpha - 1) \varepsilon_{xx}^2 + 2 - \alpha \varepsilon_{xx} + \frac{\alpha}{2}(\frac{\alpha}{2} + 1) \varepsilon_{xx}^2 - 3) \\ &= \frac{\mu}{2\alpha} (\alpha^2 - \alpha + \frac{\alpha^2}{2} + \alpha) \varepsilon_{xx}^2 = \frac{3\mu\alpha}{4} \varepsilon_{xx}^2. \end{aligned}$$

Comparing this the the energy density based on Hooke's linear law  $W = \frac{1}{2} E \varepsilon_{xx}^2$  conclude

$$E = \frac{3}{2} \mu \alpha \quad \text{or} \quad \mu = \frac{2E}{3\alpha}.$$

In the case of  $\alpha = 2$  (Neo–Hookean) find  $\mu = \frac{1}{3} E$ , which is consistent with  $C_{10} = \frac{1}{6} E$ .

To examine the stress as function of strain use

$$\begin{aligned} \sigma_x &= \frac{\partial W}{\partial \varepsilon_{xx}} = \frac{\partial}{\partial \varepsilon_{xx}} \frac{\mu}{\alpha} ((1 + \varepsilon_{xx})^\alpha + 2(1 + \varepsilon_{xx})^{-\alpha/2} - 3) \\ &= \frac{\mu}{\alpha} (\alpha(1 + \varepsilon_{xx})^{\alpha-1} - \alpha(1 + \varepsilon_{xx})^{-\alpha/2-1}) \\ &= \mu((1 + \varepsilon_{xx})^{\alpha-1} - (1 + \varepsilon_{xx})^{-\alpha/2-1}). \end{aligned}$$

Find the stress-strain curve for  $\alpha = 1.5$ ,  $\alpha = 2$  and  $\alpha = 3$  in Figure 5.21, together with the linear law by Hooke. ◇

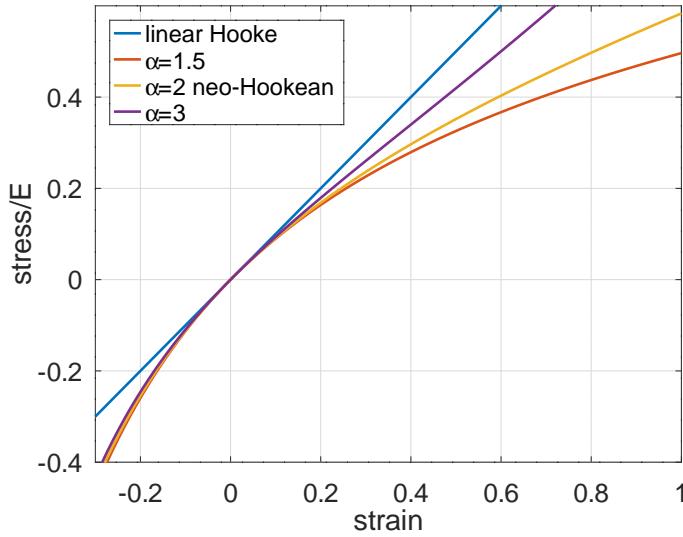


Figure 5.21: Stress–strain curve for Hooke’s linear law and an incompressible Ogden material under uniaxial load, for different values of  $\alpha$

An attempt to apply the same procedure to a hydrostatic situation has to fail for incompressible materials.

$$\begin{aligned} W &= 3 \frac{\mu}{\alpha} ((1 + \varepsilon_{xx})^\alpha - 1) \quad \text{using} \quad 1 + \varepsilon_{xx} = \lambda_1 = \lambda_2 = \lambda_3 \\ \sigma_x &= \frac{1}{3} \frac{\partial W}{\partial \varepsilon_{xx}} = \frac{\mu}{\alpha} \alpha (1 + \varepsilon_{xx})^{\alpha-1} = \mu (1 + \varepsilon_{xx})^{\alpha-1} \approx \mu (1 + (\alpha - 1) \varepsilon_{xx}). \end{aligned}$$

Thus we would have a resulting force, without applying a stretch!

### Ogden energy density model for compressible materials

Ogden’s energy density can be modified to take compressibility into account. The general form is

$$W = \sum_{i=1}^N \frac{\mu_i}{\alpha_i} \left( \bar{\lambda}_1^{\alpha_i} + \bar{\lambda}_2^{\alpha_i} + \bar{\lambda}_3^{\alpha_i} - 3 \right) + \sum_{i=1}^N \frac{1}{D_i} (\lambda_1 \cdot \lambda_2 \cdot \lambda_3 - 1)^{2i}.$$

The simplest case  $N = 1$  leads to

$$W = \frac{\mu}{\alpha} (\bar{\lambda}_1^\alpha + \bar{\lambda}_2^\alpha + \bar{\lambda}_3^\alpha - 3) + \frac{1}{D} (\lambda_1 \cdot \lambda_2 \cdot \lambda_3 - 1)^2. \quad (5.26)$$

- In the special case of  $\alpha = 2$  find

$$W = \frac{\mu}{2} (\bar{\lambda}_1^2 + \bar{\lambda}_2^2 + \bar{\lambda}_3^2 - 3) + \frac{1}{D} (\lambda_1 \cdot \lambda_2 \cdot \lambda_3 - 1)^2 = \frac{\mu}{2} (\bar{I}_1 - 3) + \frac{1}{D} (J - 1)^2$$

i.e. the Neo–Hookean energy density (5.20) is a special case of Ogden’s law.

- Observe that minimizing the total energy, using the integral of the energy density  $W$ , takes the compressibility into account by considering  $(J - 1)^2$ .

**5–51 Example : Ogden energy density for compressible materials, hydrostatic loading**

For hydrostatic loading use  $\lambda_1 = \lambda_2 = \lambda_3 = 1 + \varepsilon_{xx}$ , leading to  $\bar{\lambda}_i = 1$  and thus the first contribution in Ogden's energy density equals zero. The values of the material parameters  $\mu$  and  $\alpha$  have no influence on the result. This leads to

$$\begin{aligned} W &= 0 + \frac{1}{D} ((1 + \varepsilon_{xx})^3 - 1)^2 = \frac{1}{D} (3\varepsilon_{xx} + 3\varepsilon_{xx}^2 + \varepsilon_{xx}^3)^2 \approx \frac{9}{D} \varepsilon_{xx}^2 \\ \frac{\partial W}{\partial \varepsilon_{xx}} &= \frac{2}{D} ((1 + \varepsilon_{xx})^3 - 1) (3 + 6\varepsilon_{xx} + 3\varepsilon_{xx}^2) \\ &= \frac{6}{D} ((1 + \varepsilon_{xx})^3 - 1) (1 + \varepsilon_{xx})^2 \approx \frac{18}{D} \varepsilon_{xx}. \end{aligned}$$

Comparing with the computations using Hooke's law in Example 5–32 (page 357) conclude

$$\frac{1}{2} \frac{E}{1 - 2\nu} = \frac{3}{D} \quad \text{or} \quad D = \frac{6(1 - 2\nu)}{E}.$$

The stress–strain curve for Hooke's and Ogden's material are given by

$$\begin{aligned} \sigma_{\text{Hooke}} &= \frac{E}{1 - 2\nu} \varepsilon_{xx} \\ \sigma_{\text{Ogden}} &= \frac{1}{3} \frac{\partial W}{\partial \varepsilon_{xx}} = \frac{2}{D} ((1 + \varepsilon_{xx})^3 - 1) (1 + \varepsilon_{xx})^2 \\ &= \frac{E}{3(1 - 2\nu)} (3\varepsilon_{xx} + 3\varepsilon_{xx}^2 + \varepsilon_{xx}^3) (1 + \varepsilon_{xx})^2 \\ &= \frac{E}{1 - 2\nu} \varepsilon_{xx} \left(1 + \varepsilon_{xx} + \frac{1}{3} \varepsilon_{xx}^2\right) (1 + \varepsilon_{xx})^2 \\ &= \frac{E}{1 - 2\nu} \varepsilon_{xx} \left(1 + 3\varepsilon_{xx} + \frac{7}{3} \varepsilon_{xx}^2 + O(\varepsilon_{xx}^3)\right). \end{aligned}$$

Find the result in Figure 5.22, the computations performed with a Poisson ratio  $\nu = 0.3$ . ◊

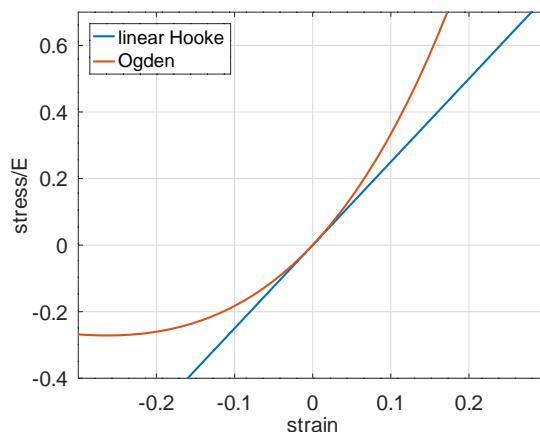


Figure 5.22: Stress–strain curve for Hooke's linear law and a compressible Ogden material under hydrostatic load with  $\nu = 0.3$ .

**5–52 Example : Ogden energy density for compressible materials, uniaxial loading**

For uniaxial loading we  $\lambda_1 = 1 + \varepsilon_{xx}$  and  $\lambda_2 = \lambda_3 = 1 + \varepsilon_{yy}$ , leading to

$$J = \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = (1 + \varepsilon_{xx})(1 + \varepsilon_{yy})^2$$

and thus

$$\bar{\lambda}_1 = \frac{\lambda_1}{J^{1/3}} = \frac{1 + \varepsilon_{xx}}{(1 + \varepsilon_{xx})^{1/3} (1 + \varepsilon_{yy})^{2/3}}, \quad \bar{\lambda}_2 = \bar{\lambda}_3 = \frac{1 + \varepsilon_{yy}}{(1 + \varepsilon_{xx})^{1/3} (1 + \varepsilon_{yy})^{2/3}}.$$

The value of  $\lambda_2$  is determined by minimizing  $W(\lambda_1, \lambda_2)$  with respect to  $\lambda_2$ .

$$\begin{aligned} W(\lambda_1, \lambda_2) &= \frac{\mu}{\alpha} (\bar{\lambda}_1^\alpha + \bar{\lambda}_2^\alpha + \bar{\lambda}_3^\alpha - 3) + \frac{1}{D} (\lambda_1 \cdot \lambda_2 \cdot \lambda_3 - 1)^2 \\ &= \frac{\mu}{\alpha} \left( \frac{\lambda_1^\alpha}{\lambda_1^{\alpha/3} \lambda_2^{2\alpha/3}} + 2 \frac{\lambda_2^\alpha}{\lambda_1^{\alpha/3} \lambda_2^{2\alpha/3}} - 3 \right) + \frac{1}{D} (\lambda_1 \cdot \lambda_2^2 - 1)^2 \\ &= \frac{\mu}{\alpha} \left( \lambda_1^{2\alpha/3} \lambda_2^{-2\alpha/3} + 2 \lambda_1^{-\alpha/3} \lambda_2^{\alpha/3} - 3 \right) + \frac{1}{D} (\lambda_1 \cdot \lambda_2^2 - 1)^2 \\ \frac{\partial}{\partial \lambda_2} W(\lambda_1, \lambda_2) &= \frac{\mu}{\alpha} \left( -\frac{2\alpha}{3} \lambda_1^{2\alpha/3} \lambda_2^{-2\alpha/3-1} + \frac{2\alpha}{3} \lambda_1^{-\alpha/3} \lambda_2^{\alpha/3-1} \right) + \frac{4}{D} (\lambda_1 \cdot \lambda_2^2 - 1) \lambda_1 \lambda_2 \\ &= \frac{2\mu}{3} \left( -\lambda_1^{2\alpha/3} \lambda_2^{-2\alpha/3-1} + \lambda_1^{-\alpha/3} \lambda_2^{\alpha/3-1} \right) + \frac{4}{D} (\lambda_1 \cdot \lambda_2^2 - 1) \lambda_1 \lambda_2 = 0 \end{aligned}$$

This equation has to be solved for  $\lambda_2 = 1 + \varepsilon_{yy}$  as function of  $\lambda_1 = 1 + \varepsilon_{xx}$ . Using Taylor approximations for  $\varepsilon_{xx} \approx 0$  and  $\varepsilon_{yy} \approx 0$  find

$$\begin{aligned} 0 = \frac{\partial}{\partial \varepsilon_{yy}} W(\varepsilon_{xx}, \varepsilon_{yy}) &= \frac{2\mu}{3} \left( -(1 + \varepsilon_{xx})^{2\alpha/3} (1 + \varepsilon_{yy})^{-2\alpha/3-1} + (1 + \varepsilon_{xx})^{-\alpha/3} (1 + \varepsilon_{yy})^{\alpha/3-1} \right) + \\ &\quad + \frac{4}{D} ((1 + \varepsilon_{xx}) \cdot (1 + \varepsilon_{yy})^2 - 1) (1 + \varepsilon_{xx}) (1 + \varepsilon_{yy}) \\ &\approx \frac{2\mu}{3} \left( -(1 + \frac{2\alpha}{3} \varepsilon_{xx}) (1 + \frac{-2\alpha-3}{3} \varepsilon_{yy}) + (1 - \frac{\alpha}{3} \varepsilon_{xx}) (1 + \frac{\alpha-3}{3} \varepsilon_{yy}) \right) + \\ &\quad + \frac{4}{D} (\varepsilon_{xx} + 2\varepsilon_{yy}) (1 + \varepsilon_{xx}) (1 + \varepsilon_{yy}) \\ &\approx \frac{2\mu}{3} \left( -\frac{2\alpha}{3} \varepsilon_{xx} + \frac{2\alpha+3}{3} \varepsilon_{yy} - \frac{\alpha}{3} \varepsilon_{xx} + \frac{\alpha-3}{3} \varepsilon_{yy} \right) + \frac{4}{D} (\varepsilon_{xx} + 2\varepsilon_{yy}) \\ &= \left( -\frac{2\mu\alpha}{3} + \frac{4}{D} \right) \varepsilon_{xx} + \left( \frac{2\mu\alpha}{3} + \frac{8}{D} \right) \varepsilon_{yy} \\ &= \frac{-2\mu\alpha D + 12}{3D} \varepsilon_{xx} + \frac{2\mu\alpha D + 24}{3D} \varepsilon_{yy} \\ \varepsilon_{yy} &= -\frac{12 - 2\mu\alpha D}{24 + 2\mu\alpha D} \varepsilon_{xx} = -\frac{6 - \mu\alpha D}{12 + \mu\alpha D} \varepsilon_{xx}. \end{aligned}$$

For small, uniaxial deformation this leads to Poisson ratio

$$\nu = \frac{-\varepsilon_{yy}}{\varepsilon_{xx}} = \frac{6 - \mu\alpha D}{12 + \mu\alpha D} = \frac{1}{2} - \frac{3\mu\alpha D}{2(12 + \mu\alpha D)}.$$

For  $0 < D \ll 1$  find  $\nu \approx \frac{1}{2}$ , as expected for an almost incompressible material.

To determine the stress-strain curve examine the dependence of  $W$  on  $\lambda_1$ , resp.  $\varepsilon_{xx}$ . Since the energy density  $W$  is minimized with respect to  $\lambda_2$  use

$$\frac{d W(\lambda_1, \lambda_2)}{d \lambda_1} = \frac{\partial W(\lambda_1, \lambda_2)}{\partial \lambda_1} + \frac{\partial W(\lambda_1, \lambda_2)}{\partial \lambda_2} \frac{\partial \lambda_2}{\partial \lambda_1} = \frac{\partial W(\lambda_1, \lambda_2)}{\partial \lambda_1} + 0.$$

For small strains determine

$$\frac{\partial}{\partial \lambda_1} W(\lambda_1, \lambda_2) = \frac{\mu}{\alpha} \left( +\frac{2\alpha}{3} \lambda_1^{2\alpha/3-1} \lambda_2^{-2\alpha/3} - \frac{2\alpha}{3} \lambda_1^{-\alpha/3-1} \lambda_2^{\alpha/3} \right) + \frac{2}{D} (\lambda_1 \cdot \lambda_2^2 - 1) \lambda_2^2$$

$$\begin{aligned}
\frac{\partial}{\partial \varepsilon_{xx}} W(\varepsilon_{xx}, \varepsilon_{yy}) &= \frac{2\mu}{3} \left( (1 + \varepsilon_{xx})^{2\alpha/3-1} (1 + \varepsilon_{yy})^{-2\alpha/3} - (1 + \varepsilon_{xx})^{-\alpha/3-1} (1 + \varepsilon_{yy})^{\alpha/3} \right) + \\
&\quad + \frac{2}{D} ((1 + \varepsilon_{xx}) \cdot (1 + \varepsilon_{yy})^2 - 1) (1 + \varepsilon_{yy})^2 \\
&\approx \frac{2\mu}{3} \left( (1 + \frac{2\alpha-3}{3}\varepsilon_{xx}) (1 - \frac{2\alpha}{3}\varepsilon_{yy}) - (1 - \frac{\alpha+3}{3}\varepsilon_{xx}) (1 + \frac{\alpha}{3}\varepsilon_{yy}) \right) + \\
&\quad + \frac{2}{D} (\varepsilon_{xx} + 2\varepsilon_{yy}) (1 + 2\varepsilon_{yy}) \\
&\approx \frac{2\mu}{3} \left( \frac{2\alpha-3}{3}\varepsilon_{xx} - \frac{2\alpha}{3}\varepsilon_{yy} + \frac{\alpha+3}{3}\varepsilon_{xx} - \frac{\alpha}{3}\varepsilon_{yy} \right) + \frac{2}{D} (\varepsilon_{xx} + 2\varepsilon_{yy}) \\
&= \left( \frac{2\mu\alpha}{3} + \frac{2}{D} \right) \varepsilon_{xx} + \left( \frac{-2\mu\alpha}{3} + \frac{4}{D} \right) \varepsilon_{yy} \\
&= \frac{2\mu\alpha D + 6}{3D} \varepsilon_{xx} - \frac{2\mu\alpha D - 12}{3D} \varepsilon_{yy}.
\end{aligned}$$

Using the above Poisson ratio this leads to

$$\begin{aligned}
\frac{\partial}{\partial \varepsilon_{xx}} W(\varepsilon_{xx}, \varepsilon_{yy}) &\approx \frac{2\mu\alpha D + 6}{3D} \varepsilon_{xx} + \frac{2\mu\alpha D - 12}{3D} \frac{6 - \mu\alpha D}{12 + \mu\alpha D} \varepsilon_{xx} \\
&= \left( \frac{2(3 + \mu\alpha D)(12 + \mu\alpha D)}{3D(12 + \mu\alpha D)} - \frac{2(6 - \mu\alpha D)^2}{3D(12 + \mu\alpha D)} \right) \varepsilon_{xx} \\
&= \frac{2(3 + \mu\alpha D)(12 + \mu\alpha D) - 2(6 - \mu\alpha D)^2}{3D(12 + \mu\alpha D)} \varepsilon_{xx} \\
&= \frac{(30 + 24)\mu\alpha D}{3D(12 + \mu\alpha D)} \varepsilon_{xx} = \frac{18\mu\alpha}{12 + \mu\alpha D} \varepsilon_{xx}.
\end{aligned}$$

As a consequence we find for small strains  $\varepsilon_{xx}$

$$E \approx \frac{18\mu\alpha}{12 + \mu\alpha D}.$$

In case of an incompressible material this simplifies to  $E = \frac{3}{2}\mu\alpha$ , which is consistent with Example 5–50.  $\diamond$

### 5.7.4 References used in the Above Section

For the above examples I used a few references:

- On Wikipedia find a very informative page on Neo–Hookean material laws at [https://en.wikipedia.org/wiki/Neo-Hookean\\_solid](https://en.wikipedia.org/wiki/Neo-Hookean_solid).
- The book by G. Holzapfel [Holz00, §6.5] for a few formulations of strain based energies.
- In [Bowe10, §3.5.5] and the corresponding web page [solidmechanics.org](http://solidmechanics.org) a nonlinear energy density is given by

$$W = C_{10} (\bar{I}_1 - 3) + \frac{1}{D} (J - 1)^2 = C_{10} \left( \frac{I_1}{J^{2/3}} - 3 \right) + \frac{1}{D} (J - 1)^2$$

with

$$C_{10} = \frac{E}{4(1+\nu)} \quad \text{and} \quad \frac{1}{D} = \frac{E}{6(1-2\nu)}.$$

For  $\nu = 0.5$  this leads to  $C_{10} = \frac{E}{6}$  and  $D = 0$ .

- In [Ogde13, p 221] a few models are examined.

$$\begin{aligned} W &= C_{10} (I_1 - 3) + C_{01} (I_2 - 3) && \text{Mooney–Rivlin} \\ W &= C_{10} (I_1 - 3) && \text{Neo–Hookean} \end{aligned}$$

In [Ogde13, §4.4, p. 222] the two Neo–Hookean energies for compressible and incompressible materials are discussed.

$$W = C_{10} (I_1 - 3) \quad \text{and} \quad W = \frac{\mu}{2} (I_1 - 3 - 2 \ln J) + \frac{\lambda}{2} (\ln J)^2$$

- [Hack15, (4.6), p.20] Mooney–Rivlin

$$W = C_{10} (I_1 - 3) + C_{01} (I_2 - 3).$$

For small strains 2( $C_{10} + C_{01}$ ) equals the shear modulus  $G = \frac{E}{2(1+\nu)}$  and thus  $C_{10} + C_{01} = \frac{1}{6} E$ .

- [Hack15, (4.7)] Neo–Hookean

$$W = C_{10} (I_1 - 3).$$

For small strains 2 $C_{10}$  is equal to the shear modulus  $G = \frac{E}{2(1+\nu)}$  and thus  $C_{10} = \frac{1}{6} E$ .

- [Hack15, p. 21] decoupling deviatoric and volumetric response.

$$W = C_{10} \left( \frac{I_1}{J^{2/3}} - 3 \right) + C_{01} \left( \frac{I_2}{J^{2/3}} - 3 \right) + D (J - 1)^2$$

There might be a typo in [Hack15, (4.11a)], since it says  $\bar{I}_1 = \frac{I_1}{J^{1/3}}$  instead of  $\bar{I}_1 = \frac{I_1}{J^{2/3}}$ . But

$$\begin{aligned} \bar{\lambda}_i &= \frac{\lambda_i}{J^{1/3}} \\ \bar{I}_1 &= \frac{I_1}{J^{2/3}} = \frac{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}{J^{2/3}} = \bar{\lambda}_1^2 + \bar{\lambda}_2^2 + \bar{\lambda}_3^2 \end{aligned}$$

- Use [Oden71, p. 315] for graphs and p.222ff for Neo–Hookean.
- The COMSOL manual `StructuralMechanicsModuleUsersGuide.pdf` for different material models, using the above invariants.
- The ABAQUS theory manual contains useful information, e.g. Section 4.6.1 on hyperelastic material behavior

## 5.8 Plane Strain

### 5.8.1 Description of Plane Strain and Plane Stress

If a three dimensional problem can be reduced to two dimensions, then the computational effort can be reduced considerably and the visualization is simplified. For 3D elasticity problems we have to simplify the situation such that only two independent variables  $x$  and  $y$  come into play. There are two important setups leading to this situation: plane strain and plane stress. In both cases a solid with a constant cross section  $\Omega$  (parallel to the  $xy$  plane) is considered and horizontal forces are applied to the solid. If the solid is long (in the  $z$  direction) and fixed in  $z$  direction on both ends we have the situation of plane strain, i.e. no deformations in  $z$  direction. There might be forces in  $z$  direction. If the solid is thin and we have no forces in  $z$  direction we have a plane stress situation. In a concrete situation the user has to decide if one of the two simplifications is applicable. The facts are listed and illustrated in Table 5.8 and Figure 5.23.

	plane strain	plane stress
assumptions	strains in $xy$ plane only $\varepsilon_{zz} = \varepsilon_{xz} = \varepsilon_{yz} = 0$	stress in $xy$ plane only $\sigma_z = \tau_{xz} = \tau_{yz} = 0$
free expressions	$\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy}$	$\sigma_x, \sigma_y, \tau_{xy}$
consequences	$\tau_{xz} = \tau_{yz} = 0$ $\sigma_z = \frac{E \nu}{(1+\nu)(1-2\nu)} (\varepsilon_{xx} + \varepsilon_{yy})$	$\varepsilon_{xz} = \varepsilon_{yz} = 0$ $\varepsilon_{zz} = \frac{-\nu}{E} (\sigma_x + \sigma_y)$

Table 5.8: Plane strain and plane stress situation

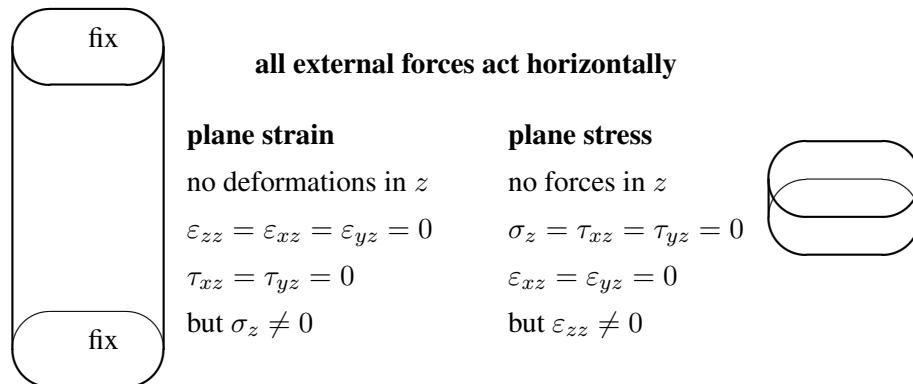


Figure 5.23: Plane strain and plane stress situation

Consider a situation where the  $z$  component of the displacement vector is a constant

$$u_3 \text{ independent on } x, y \text{ and } z$$

and

$$u_1 = u_1(x, y), \quad u_2 = u_2(x, y), \quad \text{independent on } z.$$

This leads to vanishing strains in  $z$  direction

$$\varepsilon_{zz} = \varepsilon_{xz} = \varepsilon_{yz} = 0$$

and thus this is called a **plane strain** situation. It can be realized by a long solid in the direction of the  $z$  axis with constant cross section and a force distribution parallel to the  $xy$  plane, independent on  $z$ . The two ends are to be fixed in  $z$  direction. Due to Saint-Venants's principle (see e.g. [Sout73, §5.6]) the boundary effects at the two far ends can safely be ignored. Another example is the expansion of a blood vessel, embedded in body tissue. The pulsating blood pressure will stretch the walls of the vessel, but there is no movement of the wall in the direction of the blood flow.

Hooke's law in the form (5.14) (page 348) implies

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu \\ \nu & 1-\nu & \nu \\ \nu & \nu & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ 0 \end{pmatrix} \quad \text{and} \quad \begin{aligned} \tau_{xy} &= \frac{E}{(1+\nu)} \varepsilon_{xy} \\ \tau_{xz} &= \frac{E}{(1+\nu)} 0 \\ \tau_{yz} &= \frac{E}{(1+\nu)} 0 \end{aligned}$$

or equivalently

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 1-2\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}. \quad (5.27)$$

$$\sigma_z = \frac{E\nu(\varepsilon_{xx} + \varepsilon_{yy})}{(1+\nu)(1-2\nu)}, \quad \tau_{xz} = \tau_{yz} = 0$$

The energy density can be found by equation (5.16) as

$$W = \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2(1-2\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle. \quad (5.28)$$

As unknown functions examine the two components of the displacement vector  $\vec{u} = (u_1, u_2)^T$ , as functions of  $x$  and  $y$ . The components of the strain can be computed as derivatives of  $\vec{u}$ . Thus if  $\vec{u}$  is known, all other expressions can be computed.

If the volume and surface forces are parallel to the  $xy$  plane and independent on  $z$ , then the corresponding energy contributions<sup>25</sup> can be written as integrals over the domain  $\Omega \subset \mathbb{R}^2$ , resp. the boundary  $\partial\Omega$ . Obtain the total energy as a **functional** of the yet unknown function  $\vec{u}$ .

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} & (5.29) \\ &= \iint_{\Omega} \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2(1-2\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy - \\ &\quad - \iint_{\Omega} \vec{f} \cdot \vec{u} dx dy - \oint_{\partial\Omega} \vec{g} \cdot \vec{u} ds \end{aligned}$$

As in many other situations use again the Bernoulli principle to find the solution of the plane strain elasticity problem.

<sup>25</sup>Observe that we quietly switch from a domain in  $\Omega \times [0, H] \subset \mathbb{R}^3$  to the planar domain  $\Omega \subset \mathbb{R}^2$ . The ‘energy’  $U$  actually denotes the ‘energy divided by height  $H$ ’.

**5–53 Example :** Consider a horizontal, rectangular plate, trapped in  $z$  direction between two very hard surfaces. Then compress the plate in  $x$  direction ( $\varepsilon_{xx} < 0$ ) by a force  $F$  applied to its right edge. Assume that the normal strain  $\varepsilon_{xx}$  in  $x$ -direction is known and then determine the other expressions. Since it is a plane strain situation use  $u_3 = \varepsilon_{zz} = \varepsilon_{xz} = \varepsilon_{yz} = 0$ . The similar plane stress problem will be examined in Example 5–54. Assume that all strains are constant. Now  $\varepsilon_{yy}$  and  $\varepsilon_{xy}$  can be determined by minimizing the energy density. From equation (5.28) obtain

$$W = \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} ((1-\nu)\varepsilon_{xx}^2 + 2\nu\varepsilon_{xx}\varepsilon_{yy} + (1-\nu)\varepsilon_{yy}^2 + 2(1-2\nu)\varepsilon_{xy}^2).$$

As a necessary condition for a minimum the partial derivatives with respect to  $\varepsilon_{yy}$  and  $\varepsilon_{xy}$  have to vanish. This leads to

$$+2\nu\varepsilon_{xx} + 2(1-\nu)\varepsilon_{yy} = 0 \quad \text{and} \quad \varepsilon_{xy} = 0.$$

This leads to a modified Poisson's ratio  $\nu^*$  for the plane strain situation.

$$\varepsilon_{yy} = -\frac{\nu}{1-\nu}\varepsilon_{xx} = -\nu^*\varepsilon_{xx}.$$

Since  $\nu > 0$  we conclude  $\nu^* > \nu$ , i.e. the plate will expand ( $\varepsilon_{yy} > 0$ ) more in  $y$  direction than a free plate. This is caused by the plate not being allowed to expand in  $z$  direction, i.e.  $\varepsilon_{zz} = 0$ .

The energy density in this situation is given by

$$\begin{aligned} W &= \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left( (1-\nu)\varepsilon_{xx}^2 - 2\frac{\nu^2}{1-\nu}\varepsilon_{xx}^2 + \frac{\nu^2(1-\nu)}{(1-\nu)^2}\varepsilon_{xx}^2 \right) \\ &= \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left( \frac{1-2\nu+\nu^2}{1-\nu} - \frac{2\nu^2}{1-\nu} + \frac{\nu^2}{1-\nu} \right) \varepsilon_{xx}^2 \\ &= \frac{1}{2} \frac{E}{1-\nu^2} \varepsilon_{xx}^2. \end{aligned}$$

By comparing this with the situation of a simple stretched shaft (Example 5–31, page 355) find a modified modulus of elasticity

$$E^* = \frac{1}{1-\nu^2} E$$

and the pressure required to compress the plate is given by using  $\sigma_z = \frac{\partial W}{\partial \varepsilon_{xx}}$

$$\frac{F}{A} = E^* \frac{\Delta L}{L} = E^* \varepsilon_{xx}.$$

The fixation of the plate in  $z$  direction (plane strain) prevents the plate from showing the Poisson contraction (resp. expansion), when compressed in  $x$  direction. Thus more force is required to compress it in  $x$  direction. This information is given by  $E^* = \frac{1}{1-\nu^2} E > E$ .  $\diamond$

Similarly modified constants  $\nu^* = \frac{\nu}{1-\nu} \geq \nu$  and  $E^*$  are used in [Sout73, p. 87] to formulate the partial differential equations governing this situation. It is a matter of elementary algebra to verify that

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 1-2\nu \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}$$

<sup>26</sup>One may also use Hooke's law for a plane strain setup to conclude

$$\begin{aligned} \sigma_x &= \frac{E}{(1+\nu)(1-2\nu)} ((1-\nu)\varepsilon_{xx} + \nu\varepsilon_{yy}) = \frac{E}{(1+\nu)(1-2\nu)} \left( (1-\nu) - \frac{\nu^2}{1-\nu} \right) \varepsilon_{xx} \\ &= \frac{E}{(1+\nu)(1-2\nu)} \frac{(1-\nu)^2 - \nu^2}{1-\nu} \varepsilon_{xx} = \frac{E}{1-\nu^2} \varepsilon_{xx}. \end{aligned}$$

$$= \frac{E^*}{1 - (\nu^*)^2} \begin{bmatrix} 1 & \nu^* & 0 \\ \nu^* & 1 & 0 \\ 0 & 0 & 1 - \nu^* \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}.$$

This form of Hooke's law for a plane strain situation coincides with the plane stress situation in equation (5.31) on page 394, but using  $E^*$  and  $\nu^*$  instead of the usual  $E$  and  $\nu$ .

### 5.8.2 From the Minimization Formulation to a System of PDE's

Using the Bernoulli principle the displacement vector  $\vec{u}$  has to minimize to total energy of the system, given by

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \\ &= \iint_{\Omega} \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2(1-2\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy - \\ &\quad - \iint_{\Omega} \vec{f} \cdot \vec{u} dx dy - \oint_{\partial\Omega} \vec{g} \cdot \vec{u} ds. \end{aligned}$$

This can be used to derive a system of partial differential equations that are solved by the actual displacement function. Use the abbreviation

$$k = \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)}.$$

to find the main expression for the elastic energy given by

$$\begin{aligned} U_{elast} &= \iint_{\Omega} \left\langle \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, k \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2(1-2\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy \\ &= \iint_{\Omega} \left\langle \begin{pmatrix} \frac{\partial u_1}{\partial x} \\ \frac{\partial u_2}{\partial y} \\ \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \end{pmatrix}, k \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2(1-2\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy \\ &= \iint_{\Omega} \left\langle \begin{pmatrix} \frac{\partial u_1}{\partial x} \\ \frac{\partial u_1}{\partial y} \end{pmatrix}, k \begin{bmatrix} 1-\nu & \nu & 0 \\ 0 & 0 & 1-2\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy \\ &\quad + \iint_{\Omega} \left\langle \begin{pmatrix} \frac{\partial u_2}{\partial x} \\ \frac{\partial u_2}{\partial y} \end{pmatrix}, k \begin{bmatrix} 0 & 0 & 1-2\nu \\ \nu & 1-\nu & 0 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy. \end{aligned}$$

Using the divergence theorem (Section 5.2.3, page 312) on the two integrals find

$$\begin{aligned} U_{elast} &= - \iint_{\Omega} u_1 \operatorname{div} \left( k \begin{bmatrix} 1-\nu & \nu & 0 \\ 0 & 0 & 1-2\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right) dx dy \\ &\quad + \oint_{\partial\Omega} u_1 \langle \vec{n}, k \begin{bmatrix} 1-\nu & \nu & 0 \\ 0 & 0 & 1-2\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle ds \end{aligned}$$

$$\begin{aligned}
& - \iint_{\Omega} u_2 \operatorname{div} \left( k \begin{bmatrix} 0 & 0 & 1-2\nu \\ \nu & 1-\nu & 0 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right) dx dy \\
& + \oint_{\partial\Omega} u_2 \langle \vec{n}, k \begin{bmatrix} 0 & 0 & 1-2\nu \\ \nu & 1-\nu & 0 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle ds.
\end{aligned}$$

Using a calculus of variations argument with perturbations of  $\phi_1$  of  $u_1$  vanishing on the boundary conclude<sup>27</sup>

$$\begin{aligned}
\operatorname{div} \left( \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ 0 & 0 & 1-2\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right) &= -f_1 \\
\operatorname{div} \left( \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} (1-\nu) \frac{\partial u_1}{\partial x} + \nu \frac{\partial u_2}{\partial y} \\ \frac{1-2\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \end{pmatrix} \right) &= -f_1
\end{aligned}$$

and similarly, using perturbations of  $u_2$ , leads to

$$\operatorname{div} \left( \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} \frac{1-2\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \nu \frac{\partial u_1}{\partial x} + (1-\nu) \frac{\partial u_2}{\partial y} \end{pmatrix} \right) = -f_2.$$

This is a system of second order partial differential equations (PDE) for the unknown displacement vector function  $\vec{u}$ . If the coefficients  $E$  and  $\nu$  are constant one can juggle with these equations and arrive at different formulations. The first equation may be rewritten in the form

$$\begin{aligned}
\frac{E}{2(1+\nu)} \left( \frac{2(1-\nu)}{1-2\nu} \frac{\partial^2 u_1}{\partial x^2} + \frac{2\nu}{1-2\nu} \frac{\partial^2 u_2}{\partial y \partial x} + \frac{\partial^2 u_1}{\partial y^2} + \frac{\partial^2 u_2}{\partial x \partial y} \right) &= -f_1 \\
\frac{E}{2(1+\nu)} \left( \frac{1+(1-2\nu)}{1-2\nu} \frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} + \frac{1}{1-2\nu} \frac{\partial^2 u_2}{\partial y \partial x} \right) &= -f_1 \\
\frac{E}{2(1+\nu)} \left( \frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} + \frac{1}{1-2\nu} \frac{\partial}{\partial x} \left( \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} \right) \right) &= -f_1.
\end{aligned}$$

By rewriting the second differential equation in a similar fashion we arrive at a formulation given in [Sout73, p. 87].

$$\begin{aligned}
\frac{E}{2(1+\nu)} \left( \Delta u_1 + \frac{1}{1-2\nu} \frac{\partial}{\partial x} \left( \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} \right) \right) &= -f_1 \\
\frac{E}{2(1+\nu)} \left( \Delta u_2 + \frac{1}{1-2\nu} \frac{\partial}{\partial y} \left( \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} \right) \right) &= -f_2
\end{aligned}$$

---

<sup>27</sup>There is a minor gap in the argument: we only take variations of  $u_1$  into account while the resulting variations on  $\varepsilon_{xx}$ ,  $\varepsilon_{yy}$  and  $\varepsilon_{xy}$  are ignored. Thus use

$$\langle u + \phi, \mathbf{A}u - f \rangle \text{ minimal} \implies \mathbf{A}u - f = 0.$$

The preceding calculation examines an expression  $\langle u, \mathbf{A}u \rangle$  for an accordingly defined scalar product. For a symmetric matrix  $\mathbf{A}$  and a perturbation  $u + \phi$  of  $u$  we should actually examine

$$\begin{aligned}
\langle u + \phi, \mathbf{A}(u + \phi) - f \rangle &= \langle u, \mathbf{A}u - f \rangle + \langle \phi, \mathbf{A}u - f \rangle + \langle u, \mathbf{A}\phi \rangle + \langle \phi, \mathbf{A}\phi \rangle \\
&\approx \langle u, \mathbf{A}u - f \rangle + \langle \phi, 2\mathbf{A}u - f \rangle.
\end{aligned}$$

If this expression is minimized at  $u$  then we conclude  $2\mathbf{A}u - f = 0$ . The only difference to the first approach is a factor of 2, which is taken into account for the resulting differential equations in the main text.

With the usual definitions of the operators  $\vec{\nabla}$  and  $\Delta$  this can be written in the dense form

$$\frac{E}{2(1+\nu)} \left( \Delta \vec{u} + \frac{1}{1-2\nu} \vec{\nabla} (\vec{\nabla} \cdot \vec{u}) \right) = -\vec{f}. \quad (5.30)$$

This author is convinced however that the above formulation as a system of PDE's is considerably less efficient than the formulation as a minimization problem of the energy given by equation (5.29).

### 5.8.3 Boundary Conditions

There are different types of useful boundary conditions. Only the most important situations are presented in these notes.

#### Prescribed displacement

If on a section  $\Gamma_1$  of the boundary  $\partial\Omega$  the displacement vector  $\vec{u}$  is known, use this as a boundary condition on the section  $\Gamma_1$ . Thus find Dirichlet conditions on this section of the boundary.

#### Given boundary forces, no constraints

If on a section  $\Gamma_2$  of the boundary  $\partial\Omega$  the displacement  $\vec{u}$  is free, then use calculus of variations again and examine all contributions of the integral over the boundary section  $\Gamma_2$  in the total energy  $U_{elast} + U_{Vol} + U_{Surf}$ .

$$\begin{aligned} \int_{\Gamma_2} \dots ds &= \int_{\Gamma_2} u_1 \langle \vec{n}, k \begin{bmatrix} 1-\nu & \nu & 0 \\ 0 & 0 & 1-2\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle ds \\ &\quad + \int_{\Gamma_2} u_2 \langle \vec{n}, k \begin{bmatrix} 0 & 0 & 1-2\nu \\ \nu & 1-\nu & 0 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle ds - \int_{\Gamma_2} \vec{g} \cdot \vec{u} ds \\ &= \int_{\Gamma_2} u_1 \langle \vec{n}, k \cdot \begin{pmatrix} (1-\nu)\varepsilon_{xx} + \nu\varepsilon_{yy} \\ (1-2\nu)\varepsilon_{xy} \end{pmatrix} \rangle ds - \int_{\Gamma_2} g_1 u_1 ds \\ &\quad + \int_{\Gamma_2} u_2 \langle \vec{n}, k \cdot \begin{pmatrix} (1-2\nu)\varepsilon_{xy} \\ \nu\varepsilon_{xx} + (1-\nu)\varepsilon_{yy} \end{pmatrix} \rangle ds - \int_{\Gamma_2} g_2 u_2 ds \end{aligned}$$

Using the perturbations  $\vec{u} \rightarrow \vec{u} + \vec{\phi}$  this leads to two boundary conditions.

$$\begin{aligned} \frac{E}{(1+\nu)(1-2\nu)} \langle \vec{n}, \begin{pmatrix} (1-\nu)\varepsilon_{xx} + \nu\varepsilon_{yy} \\ (1-2\nu)\varepsilon_{xy} \end{pmatrix} \rangle &= g_1 \\ \frac{E}{(1+\nu)(1-2\nu)} \langle \vec{n}, \begin{pmatrix} (1-2\nu)\varepsilon_{xy} \\ \nu\varepsilon_{xx} + (1-\nu)\varepsilon_{yy} \end{pmatrix} \rangle &= g_2 \end{aligned}$$

Using Hooke's law (equation (5.14), page 348) these conditions are expressed in terms of stresses. This leads to

$$\begin{aligned} \langle \vec{n}, \begin{pmatrix} \sigma_x \\ \tau_{xy} \end{pmatrix} \rangle &= n_x \sigma_x + n_y \tau_{xy} = g_1 \\ \langle \vec{n}, \begin{pmatrix} \tau_{xy} \\ \sigma_y \end{pmatrix} \rangle &= n_y \sigma_y + n_x \tau_{xy} = g_2. \end{aligned}$$

This allows a verification of the equations by comparing with (5.9) (page 336), i.e. find

$$\begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} n_x \\ n_y \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}.$$

At the surface the stresses have to coincide with the externally applied stresses. The above boundary conditions can be written in terms of the unknown displacement vector  $\vec{u}$ , leading to

$$\begin{aligned} \frac{E}{(1+\nu)(1-2\nu)} \left( n_x \left( (1-\nu) \frac{\partial u_1}{\partial x} + \nu \frac{\partial u_2}{\partial y} \right) + n_y \frac{1-2\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \right) &= g_1 \\ \frac{E}{(1+\nu)(1-2\nu)} \left( n_y \left( (1-\nu) \frac{\partial u_2}{\partial y} + \nu \frac{\partial u_1}{\partial x} \right) + n_x \frac{1-2\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \right) &= g_2. \end{aligned}$$

## 5.9 Plane Stress

Consider the situation of a thin (thickness  $h$ ) plate in the plane  $\Omega \subset \mathbb{R}^2$ . There are no external stresses on the top and bottom surface and no vertical forces within the plate. Thus assume that  $\sigma_z = 0$  within the plate and  $\tau_{xz} = \tau_{yz} = 0$ , i.e. all stress components in  $z$  direction vanish. Thus this is called a **plane stress** situation.

$$\sigma_z = \tau_{xz} = \tau_{yz} = 0$$

Hooke's law in the form (5.12) (page 348) implies

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu & & & \\ -\nu & 1 & -\nu & & & 0 \\ -\nu & -\nu & 1 & & & \\ & & & 1+\nu & 0 & 0 \\ 0 & & & 0 & 1+\nu & 0 \\ & & & 0 & 0 & 1+\nu \end{bmatrix} \begin{pmatrix} \sigma_x \\ \sigma_y \\ 0 \\ \tau_{xy} \\ 0 \\ 0 \end{pmatrix}.$$

or by eliminating vanishing terms

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & 0 \\ -\nu & 1 & 0 \\ 0 & 0 & 1+\nu \end{bmatrix} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} \quad \text{and} \quad \begin{aligned} \varepsilon_{zz} &= \frac{-\nu}{E} (\sigma_x + \sigma_y) \\ \varepsilon_{xz} &= 0 \\ \varepsilon_{yz} &= 0 \end{aligned}.$$

This matrix can be inverted, leading to

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 1-\nu \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \quad \text{and} \quad \begin{aligned} \varepsilon_{zz} &= \frac{-\nu}{1-\nu} (\varepsilon_{xx} + \varepsilon_{yy}) \\ \varepsilon_{xz} &= 0 \\ \varepsilon_{yz} &= 0 \end{aligned}. \quad (5.31)$$

The energy density is given by equation (5.16) or (5.17).

$$\begin{aligned} W &= \frac{1}{2} \langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \end{pmatrix} \rangle + \langle \begin{pmatrix} \tau_{xy} \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{pmatrix} \rangle = \frac{1}{2} \langle \begin{pmatrix} \sigma_x \\ \sigma_y \\ 2\tau_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle \\ &= \frac{E}{2(1-\nu^2)} \langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2(1-\nu) \end{bmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle \\ &= \frac{E}{2(1-\nu^2)} (\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\nu\varepsilon_{xx}\varepsilon_{yy} + 2(1-\nu)\varepsilon_{xy}^2) \end{aligned}$$

Now express the total energy of a plane stress problem as function of the strains.

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \\ &= \iint_{\Omega} \frac{1}{2} \frac{E}{(1-\nu^2)} \left\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy - \\ &\quad - \iint_{\Omega} \vec{f} \cdot \vec{u} dx dy - \oint_{\partial\Omega} \vec{g} \cdot \vec{u} ds \end{aligned} \quad (5.32)$$

As in many other situations use again the Bernoulli principle to find the solution of the plane stress elasticity problem.

### 5.9.1 From the Plane Stress Matrix to the Full Stress Matrix

For a plane stress problem the reduced stress matrix is a  $2 \times 2$  matrix, while the full stress matrix has to be  $3 \times 3$ .

$$\text{plane stress} \longrightarrow \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix}, \quad \text{3D} \longrightarrow \begin{bmatrix} \sigma_x & \tau_{xy} & 0 \\ \tau_{xy} & \sigma_y & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

To compute the principal stresses  $\sigma_i$  determine the eigenvalues of this matrix, i.e. solve

$$\begin{aligned} 0 &= \det \begin{bmatrix} \sigma_x - \lambda & \tau_{xy} & 0 \\ \tau_{xy} & \sigma_y - \lambda & 0 \\ 0 & 0 & -\lambda \end{bmatrix} = -\lambda \det \begin{bmatrix} \sigma_x - \lambda & \tau_{xy} \\ \tau_{xy} & \sigma_y - \lambda \end{bmatrix} \\ &= -\lambda ((\sigma_x - \lambda)(\sigma_y - \lambda) - \tau_{xy}^2) = -\lambda (\lambda^2 - \lambda(\sigma_x + \sigma_y) + \sigma_x \sigma_y - \tau_{xy}^2). \end{aligned}$$

This leads to

$$\begin{aligned} \sigma_{1,2} &= \frac{\sigma_x + \sigma_y}{2} \pm \frac{1}{2} \sqrt{(\sigma_x + \sigma_y)^2 - 4(\sigma_x \sigma_y - \tau_{xy}^2)} = \frac{\sigma_x + \sigma_y}{2} \pm \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2} \\ \sigma_3 &= 0. \end{aligned}$$

The above principal stresses may be used to determine the von Mises and the Tresca stress. The solution of the quadratic equation can be visualized using Mohr's circle, see Result 5-23 on page 340.

**5-54 Example :** Consider a horizontal, rectangular plate, compressed in  $x$  direction by a force  $F$  applied to its right edge. Assume that the normal strain  $\varepsilon_{xx} < 0$  in  $x$ -direction is known and then determine the other expressions. Use a plane stress model, i.e.  $\sigma_z = \tau_{xz} = \tau_{yz} = 0$ . The similar plane strain problem was examined in Example 5-53. Assume that all strains are constant. Now  $\varepsilon_{yy}$  and  $\varepsilon_{xy}$  can be determined by minimizing the energy density. For the energy density obtain

$$W = \frac{E}{2(1-\nu^2)} (\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\nu\varepsilon_{xx}\varepsilon_{yy} + 2(1-\nu)\varepsilon_{xy}^2).$$

As a necessary condition for a minimum the partial derivatives with respect to  $\varepsilon_{yy}$  and  $\varepsilon_{xy}$  have to vanish. This leads to

$$+2\varepsilon_{yy} + 2\nu\varepsilon_{xx} = 0 \quad \text{and} \quad \varepsilon_{xy} = 0.$$

This leads to the standard Poisson's ratio  $\nu$  for the plane strain situation, i.e.  $\varepsilon_{yy} = -\nu\varepsilon_{xx}$ .

The energy density is given by

$$\begin{aligned} W &= \frac{1}{2} \frac{E}{1-\nu^2} (\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\nu\varepsilon_{xx}\varepsilon_{yy} + 2(1-\nu)\varepsilon_{xy}^2) \\ &= \frac{1}{2} \frac{E}{1-\nu^2} (\varepsilon_{xx}^2 + \nu^2\varepsilon_{xx}^2 - 2\nu^2\varepsilon_{xx}^2 + 0) \\ &= \frac{1}{2} \frac{E}{1-\nu^2} (1 + \nu^2 - 2\nu^2) \varepsilon_{xx}^2 = \frac{1}{2} \frac{E}{1-\nu^2} (1 - \nu^2) \varepsilon_{xx}^2 \\ &= \frac{1}{2} E \varepsilon_{xx}^2. \end{aligned}$$

By comparing this situation with the situation of a simple stretched shaft (Example 5–31, page 355) find the standard modulus of elasticity  $E$ .  $\diamond$

### 5.9.2 From the Minimization Formulation to a System of PDE's

The energy density can be found by equation (5.16) as

$$W = \frac{1}{2} \frac{E}{1-\nu^2} \left\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle. \quad (5.33)$$

This equation is very similar to the expression for a plane strain situation in equation (5.29) (page 389). The only difference is in the coefficients. As a starting point for a finite element solution of a plane stress problem use the Bernoulli principle and minimize the energy functional

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \\ &= \iint_{\Omega} \frac{1}{2} \frac{E}{1-\nu^2} \left\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy - \\ &\quad - \iint_{\Omega} \vec{f} \cdot \vec{u} dx dy - \oint_{\partial\Omega} \vec{g} \cdot \vec{u} ds. \end{aligned} \quad (5.34)$$

Using the divergence theorem rewrite the elastic energy as

$$\begin{aligned} U_{elast} &= \frac{1}{2} \iint_{\Omega} \frac{E}{1-\nu^2} \left\langle \begin{pmatrix} \frac{\partial u_1}{\partial x} \\ \frac{\partial u_2}{\partial y} \\ \frac{1}{2}(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) \end{pmatrix}, \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy \\ &= \frac{E}{2(1-\nu^2)} \iint_{\Omega} \left\langle \begin{pmatrix} \frac{\partial u_1}{\partial x} \\ \frac{\partial u_1}{\partial y} \end{pmatrix}, \begin{bmatrix} 1 & \nu & 0 \\ 0 & 0 & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy \\ &\quad + \frac{E}{2(1-\nu^2)} \iint_{\Omega} \left\langle \begin{pmatrix} \frac{\partial u_2}{\partial x} \\ \frac{\partial u_2}{\partial y} \end{pmatrix}, \begin{bmatrix} 0 & 0 & 1-\nu \\ \nu & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy \\ &= -\frac{E}{2(1-\nu^2)} \iint_{\Omega} u_1 \operatorname{div} \left( \begin{bmatrix} 1 & \nu & 0 \\ 0 & 0 & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right) dx dy \end{aligned}$$

$$\begin{aligned}
& + \frac{E}{2(1-\nu^2)} \oint_{\partial\Omega} u_1 \langle \vec{n}, \begin{bmatrix} 1 & \nu & 0 \\ 0 & 0 & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle ds \\
& - \frac{E}{2(1-\nu^2)} \iint_{\Omega} u_2 \operatorname{div} \left( \begin{bmatrix} 0 & 0 & 1-\nu \\ \nu & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right) dx dy \\
& + \frac{E}{2(1-\nu^2)} \oint_{\partial\Omega} u_2 \langle \vec{n}, \begin{bmatrix} 0 & 0 & 1-\nu \\ \nu & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle ds.
\end{aligned}$$

Reconsidering the calculations for the plane strain situation and make a few minor changes to adapt the results to the above plane stress situation to arrive at the system of partial differential equations.

$$\operatorname{div} \left( \frac{E}{1-\nu^2} \begin{pmatrix} \frac{\partial u_1}{\partial x} + \nu \frac{\partial u_2}{\partial y} \\ \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \end{pmatrix} \right) = -f_1 \quad (5.35)$$

$$\operatorname{div} \left( \frac{E}{1-\nu^2} \begin{pmatrix} \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \nu \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} \end{pmatrix} \right) = -f_2 \quad (5.36)$$

If  $E$  and  $\nu$  are constant, i.e. a homogeneous material, we can use elementary, tedious operations to find

$$\begin{aligned}
\frac{E}{2(1+\nu)} \left( \frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} + \frac{1+\nu}{1-\nu} \frac{\partial}{\partial x} \left( \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} \right) \right) &= -f_1 \\
\frac{E}{2(1+\nu)} \left( \frac{\partial^2 u_2}{\partial x^2} + \frac{\partial^2 u_2}{\partial y^2} + \frac{1+\nu}{1-\nu} \frac{\partial}{\partial y} \left( \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} \right) \right) &= -f_2
\end{aligned}$$

or with a shorter notation

$$\frac{E}{2(1+\nu)} \left( \Delta \vec{u} + \frac{1+\nu}{1-\nu} \vec{\nabla} (\vec{\nabla} \vec{u}) \right) = -\vec{f}. \quad (5.37)$$

This has a structure similar to the equations (5.30) for the plane strain situation. With

$$\nu^* = \frac{\nu}{1-\nu} \quad \text{and} \quad E^* = \frac{E}{1-\nu^2}$$

then find

$$\frac{1+\nu^*}{1-\nu^*} = \frac{1+\frac{\nu}{1-\nu}}{1-\frac{\nu}{1-\nu}} = \frac{1}{1-2\nu} \quad \text{and} \quad \frac{E^*}{1+\nu^*} = \frac{\frac{E}{1-\nu^2}}{1+\frac{\nu}{1-\nu}} = \frac{\frac{E}{1-\nu^2}}{1-\nu+\nu} = \frac{E}{1+\nu}.$$

Thus the plane strain equations (5.30) take the form

$$\frac{E^*}{2(1+\nu^*)} \left( \Delta \vec{u} + \frac{1+\nu^*}{1-\nu^*} \vec{\nabla} (\vec{\nabla} \vec{u}) \right) = -\vec{f}$$

and are very similar to the plane stress equations (5.37).

### 5.9.3 Boundary Conditions

Again only two types of boundary conditions are presented:

- On a section  $\Gamma_1$  of the boundary we assume that the displacement vector  $\vec{u}$  is known and thus we find Dirichlet boundary conditions.

- On the section  $\Gamma_2$  the displacement  $\vec{u}$  is not submitted to constraints, but we apply an external force  $\vec{g}$ . Again we use a calculus of variations argument to find the resulting boundary conditions.

The contributions of the integral over the boundary section  $\Gamma_2$  in the total energy  $U_{elast} + U_{Vol} + U_{Surf}$  are given by

$$\begin{aligned} \int_{\Gamma_2} \dots ds &= \frac{E}{2(1-\nu^2)} \int_{\Gamma_2} u_1 \langle \vec{n}, \begin{bmatrix} 1 & \nu & 0 \\ 0 & 0 & 1-\nu \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle ds \\ &\quad + \frac{E}{2(1-\nu^2)} \int_{\Gamma_2} u_2 \langle \vec{n}, \begin{bmatrix} 0 & 0 & 1-\nu \\ \nu & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle ds - \int_{\Gamma_2} \vec{g} \cdot \vec{u} ds \\ &= \frac{E}{2(1-\nu^2)} \int_{\Gamma_2} u_1 \langle \vec{n}, \begin{pmatrix} \varepsilon_{xx} + \nu \varepsilon_{yy} \\ (1-\nu) \varepsilon_{xy} \end{pmatrix} \rangle ds - \int_{\Gamma_2} g_1 u_1 ds \\ &\quad + \frac{E}{2(1-\nu^2)} \int_{\Gamma_2} u_2 \langle \vec{n}, \begin{pmatrix} (1-\nu) \varepsilon_{xy} \\ \nu \varepsilon_{xx} + \varepsilon_{yy} \end{pmatrix} \rangle ds - \int_{\Gamma_2} g_2 u_2 ds. \end{aligned}$$

This leads to two boundary conditions.

$$\begin{aligned} \frac{E}{1-\nu^2} \langle \vec{n}, \begin{pmatrix} \varepsilon_{xx} + \nu \varepsilon_{yy} \\ (1-\nu) \varepsilon_{xy} \end{pmatrix} \rangle &= g_1 \\ \frac{E}{1-\nu^2} \langle \vec{n}, \begin{pmatrix} (1-\nu) \varepsilon_{xy} \\ \nu \varepsilon_{xx} + \varepsilon_{yy} \end{pmatrix} \rangle &= g_2. \end{aligned}$$

Using Hooke's law this can be expressed in terms of stresses

$$\langle \vec{n}, \begin{pmatrix} \sigma_x \\ \tau_{xy} \end{pmatrix} \rangle = g_1 \quad \text{and} \quad \langle \vec{n}, \begin{pmatrix} \tau_{xy} \\ \sigma_y \end{pmatrix} \rangle = g_2$$

or with a matrix notation

$$\begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \begin{pmatrix} n_x \\ n_y \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}.$$

The above boundary conditions can be written in terms of the unknown displacement vector  $\vec{u}$ , leading to

$$\begin{aligned} \frac{E}{1-\nu^2} \left( n_x \left( \frac{\partial u_1}{\partial x} + \nu \frac{\partial u_2}{\partial y} \right) + n_y \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \right) &= g_1 \\ \frac{E}{1-\nu^2} \left( n_y \left( \frac{\partial u_2}{\partial y} + \nu \frac{\partial u_1}{\partial x} \right) + n_x \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \right) &= g_2. \end{aligned}$$

These equations are a mathematical model for the correct situation, but certainly not in a very readable form.

#### 5.9.4 Deriving the Differential Equations using the Euler–Lagrange Equation

In this section the above partial differential equations are regenerated, using the Euler–Lagrange equations from Result 5–11 on page 319. For this use the energy density  $W$  for a plane stress problem

$$W = \frac{1}{2} \frac{E}{1-\nu^2} \langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \rangle$$

$$\begin{aligned}
&= \frac{1}{2} \frac{E}{1-\nu^2} (\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + 2\nu \varepsilon_{xx} \varepsilon_{yy} + 2(1-\nu) \varepsilon_{ex}^2) \\
&= \frac{1}{2} \frac{E}{1-\nu^2} \left( \left( \frac{\partial u_1}{\partial x} \right)^2 + \left( \frac{\partial u_2}{\partial y} \right)^2 + 2\nu \left( \frac{\partial u_1}{\partial x} \right) \left( \frac{\partial u_2}{\partial y} \right) + \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right)^2 \right).
\end{aligned}$$

Thus the total energy in the system is given by

$$\begin{aligned}
U &= U(u_1, u_2) = U_{elast} + U_{Vol} + U_{Surf} \\
&= \iint_{\Omega} \frac{1}{2} \frac{E}{1-\nu^2} \left( \left( \frac{\partial u_1}{\partial x} \right)^2 + \left( \frac{\partial u_2}{\partial y} \right)^2 + 2\nu \frac{\partial u_1}{\partial x} \frac{\partial u_2}{\partial y} + \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right)^2 \right) - \\
&\quad - \iint_{\Omega} f_1 \cdot u_1 + f_2 \cdot u_2 \, dx \, dy - \int_{\Gamma_2} g_1 \cdot u_1 + g_2 \cdot u_2 \, ds.
\end{aligned}$$

This leads to a functional of the form (5.6), see page 318. It is a quadratic functional with the expression to be integrated given by

$$\begin{aligned}
F &= F(u_1, u_2, \nabla u_1, \nabla u_2) \\
&= \frac{1}{2} \frac{E}{1-\nu^2} \left( \left( \frac{\partial u_1}{\partial x} \right)^2 + \left( \frac{\partial u_2}{\partial y} \right)^2 + 2\nu \frac{\partial u_1}{\partial x} \frac{\partial u_2}{\partial y} + \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right)^2 \right) - \\
&\quad - f_1 \cdot u_1 - f_2 \cdot u_2.
\end{aligned}$$

To use the Euler–Lagrange equations (5.7) and (5.8)

$$\begin{aligned}
\operatorname{div}(F_{\nabla u_1}(u_1, u_2, \nabla u_1, \nabla u_2)) &= F_{u_1}(u_1, u_2, \nabla u_1, \nabla u_2), \\
\operatorname{div}(F_{\nabla u_2}(u_1, u_2, \nabla u_1, \nabla u_2)) &= F_{u_2}(u_1, u_2, \nabla u_1, \nabla u_2),
\end{aligned}$$

the partial derivatives of the expression  $F(u_1, u_2, \nabla u_1, \nabla u_2)$  are required.<sup>28</sup>

$$\begin{aligned}
F_{u_1} &= -f_1 \quad \text{and} \quad F_{u_2} = -f_2 \\
F_{\nabla u_1} &= \frac{1}{2} \frac{E}{1-\nu^2} \left( \begin{array}{c} 2 \frac{\partial u_1}{\partial x} + 2\nu \frac{\partial u_2}{\partial y} \\ (1-\nu) \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \end{array} \right) = \frac{E}{1-\nu^2} \left( \begin{array}{c} \frac{\partial u_1}{\partial x} + \nu \frac{\partial u_2}{\partial y} \\ \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \end{array} \right) \\
F_{\nabla u_2} &= \frac{1}{2} \frac{E}{1-\nu^2} \left( \begin{array}{c} (1-\nu) \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ 2 \frac{\partial u_2}{\partial y} + 2\nu \frac{\partial u_1}{\partial x} \end{array} \right) = \frac{E}{1-\nu^2} \left( \begin{array}{c} \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \frac{\partial u_2}{\partial y} + \nu \frac{\partial u_1}{\partial x} \end{array} \right)
\end{aligned}$$

Now the Euler–Lagrange equations lead to

$$\begin{aligned}
\operatorname{div} \left( \frac{E}{1-\nu^2} \left( \begin{array}{c} \frac{\partial u_1}{\partial x} + \nu \frac{\partial u_2}{\partial y} \\ \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \end{array} \right) \right) &= -f_1 \\
\operatorname{div} \left( \frac{E}{1-\nu^2} \left( \begin{array}{c} \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \frac{\partial u_2}{\partial y} + \nu \frac{\partial u_1}{\partial x} \end{array} \right) \right) &= -f_2.
\end{aligned}$$

This is identical to the PDEs (5.35) and (5.36).

On the boundary  $\Gamma_2$  the natural boundary conditions

$$\langle F_{\nabla u_1}(u_1, u_2, \nabla u_1, \nabla u_2), \vec{n} \rangle = g_1 \quad \text{and} \quad \langle F_{\nabla u_2}(u_1, u_2, \nabla u_1, \nabla u_2), \vec{n} \rangle = g_2$$

---

<sup>28</sup>Use the notation  $F_{\nabla u} = (\frac{\partial}{\partial u_x} F, \frac{\partial}{\partial u_y} F)^T$ .

lead to

$$\begin{aligned} \frac{E}{1-\nu^2} \langle \vec{n}, \left( \begin{array}{c} \frac{\partial u_1}{\partial x} + \nu \frac{\partial u_2}{\partial y} \\ \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \end{array} \right) \rangle &= g_1 \\ \frac{E}{1-\nu^2} \langle \vec{n}, \left( \begin{array}{c} \frac{1-\nu}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right) \\ \frac{\partial u_2}{\partial y} + \nu \frac{\partial u_1}{\partial x} \end{array} \right) \rangle &= g_2. \end{aligned}$$

Using Hooke's law for the plane stress situation (5.31) it can be simplified to

$$\langle \vec{n}, \left( \begin{array}{c} \sigma_x \\ \tau_{xy} \end{array} \right) \rangle = g_1 \quad \text{and} \quad \langle \vec{n}, \left( \begin{array}{c} \tau_{xy} \\ \sigma_y \end{array} \right) \rangle = g_2$$

and this is consistent with with (5.9) (page 336), i.e.

$$\left[ \begin{array}{cc} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{array} \right] \vec{n} = \left( \begin{array}{c} g_1 \\ g_2 \end{array} \right).$$

## Bibliography

- [Aris62] R. Aris. *Vectors, Tensors and the Basic Equations of Fluid Mechanics*. Prentice Hall, 1962. Republished by Dover.
- [BoneWood08] J. Bonet and R. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, 2008.
- [BoriTara79] A. I. Borisenko and I. E. Tarapov. *Vector and Tensor Analysis with Applications*. Dover, 1979. first published in 1966 by Prentice-Hall.
- [Bowe10] A. F. Bower. *Applied Mechanics of Solids*. CRC Press, 2010. web site at solidmechanics.org.
- [ChouPaga67] P. C. Chou and N. J. Pagano. *Elasticity, Tensors, dyadic and engineering Approaches*. D Van Nostrand Company, 1967. Republished by Dover 1992.
- [GhabPeckWu17] J. Ghaboussi, D. Pecknold, and X. Wu. *Nonlinear Computational Solid Mechanics*. CRC Press, 2017.
- [Gree77] D. T. Greenwood. *Classical Dynamics*. Prentice Hall, 1977. Republished by Dover 1997.
- [Hack15] R. Hackett. *Hyperelasticity Primer*. Springer International Publishing, 2015.
- [Hear97] E. J. Hearns. *Mechanics of Materials 1*. Butterworth-Heinemann, third edition, 1997.
- [HenrWann17] P. Henry and G. Wanner. Johann Bernoulli and the cycliod: A theorem for posteriority. *Elemente der Mathematik*, 72(4):137–163, 2017.
- [Holz00] G. A. Holzapfel. *Nonlinear Solid Mechanics, a Continuum Approach for Engineering*. John Wiley & Sons, 2000.
- [Oden71] J. Oden. *Finite Elements of Nonlinear Continua*. Advanced engineering series. McGraw-Hill, 1971. Republished by Dover, 2006.
- [Ogde13] R. Ogden. *Non-Linear Elastic Deformations*. Dover Civil and Mechanical Engineering. Dover Publications, 2013.

- [OttoPete92] N. S. Ottosen and H. Petersson. *Introduction to the Finite Element Method*. Prentice Hall, 1992.
- [Prze68] J. Przemieniecki. *Theory of Matrix Structural Analysis*. McGraw–Hill, 1968. Republished by Dover in 1985.
- [Redd84] J. N. Reddy. *An Introduction to the Finite Element Analysis*. McGraw–Hill, 1984.
- [Redd13] J. N. Reddy. *An Introduction to Continuum Mechanics*. Cambridge University Press, 2nd edition, 2013.
- [Redd15] J. N. Reddy. *An Introduction to Nonlinear Finite Element Analysis*. Oxford University Press, 2nd edition, 2015.
- [Sege77] L. A. Segel. *Mathematics Applied to Continuum Mechanics*. MacMillan Publishing Company, New York, 1977. republished by Dover 1987.
- [Shab08] A. A. Shabana. *Computational Continuum Mechanics*. Cambridge University Press, 2008.
- [Sout73] R. W. Soutas-Little. *Elasticity*. Prentice–Hall, 1973.
- [VarFEM] A. Stahel. Calculus of Variations and Finite Elements. Lecture Notes used at HTA Biel, 2000.
- [Wein74] R. Weinstock. *Calculus of Variations*. Dover, New York, 1974.

# Chapter 6

## Finite Element Methods

### 6.1 From Minimization to the Finite Element Method

Let  $\Omega \subset \mathbb{R}^2$  be a bounded domain with a smooth boundary  $\Gamma$ , consisting of two disjoint parts  $\Gamma_1$  and  $\Gamma_2$ . In Section 5.2.4 the minimizer of a functional  $F(u)$

$$F(u) = \iint_{\Omega} \frac{1}{2} a (\nabla u)^2 + \frac{1}{2} b u^2 + f \cdot u \, dA - \int_{\Gamma_2} g_2 u \, ds$$

was examined, with the boundary condition  $u(x, y) = g_1(x, y)$  for  $(x, y) \in \Gamma_1$ . At the minimal function  $u$  the derivative in the direction of the function  $\phi$  has to vanish. Thus the minimal solution has to satisfy

$$0 = \iint_{\Omega} (-\nabla(a \nabla u) + b u + f) \cdot \phi \, dA + \int_{\Gamma_2} (a \vec{n} \cdot \nabla u - g_2) \phi \, ds \quad (6.1)$$

for all test function  $\phi$  vanishing on the boundary  $\Gamma_1$ . Using Green's formula (integration by parts) this leads to

$$0 = \iint_{\Omega} a \nabla u \cdot \nabla \phi + (b u + f) \cdot \phi \, dA - \int_{\Gamma_2} g_2 \phi \, ds. \quad (6.2)$$

A function  $u$  satisfying this condition is called a **weak solution**. Using the fundamental lemma of the calculus of variations conclude that the function  $u$  is a solution of the boundary value problem

$$\begin{aligned} -\nabla \cdot (a \nabla u) + b u &= -f && \text{for } (x, y) \in \Omega \\ u &= g_1 && \text{for } (x, y) \in \Gamma_1 \\ a \vec{n} \cdot \nabla u &= g_2 && \text{for } (x, y) \in \Gamma_2. \end{aligned}$$

Thus we have the following chain of results.

$$\boxed{u \text{ minimizer of } F(u)} \longrightarrow \boxed{u \text{ weak solution}} \longrightarrow \boxed{u \text{ classical solution}}$$

For weak solutions or the minimization formulation use a numerical integration to discretize the formulation. This will lead directly to the **Finite Element Method** (FEM). The path to follow is illustrated in Figure 6.1. The branch on the left can only be used for self-adjoint problems and the resulting matrix  $\mathbf{A}$  will be symmetric. The branch on the right is applicable to more general problems, leading to possibly non-symmetric matrices.

In this chapter the order of presentation is as follows:

- Develop the algorithm for piecewise linear FEM.

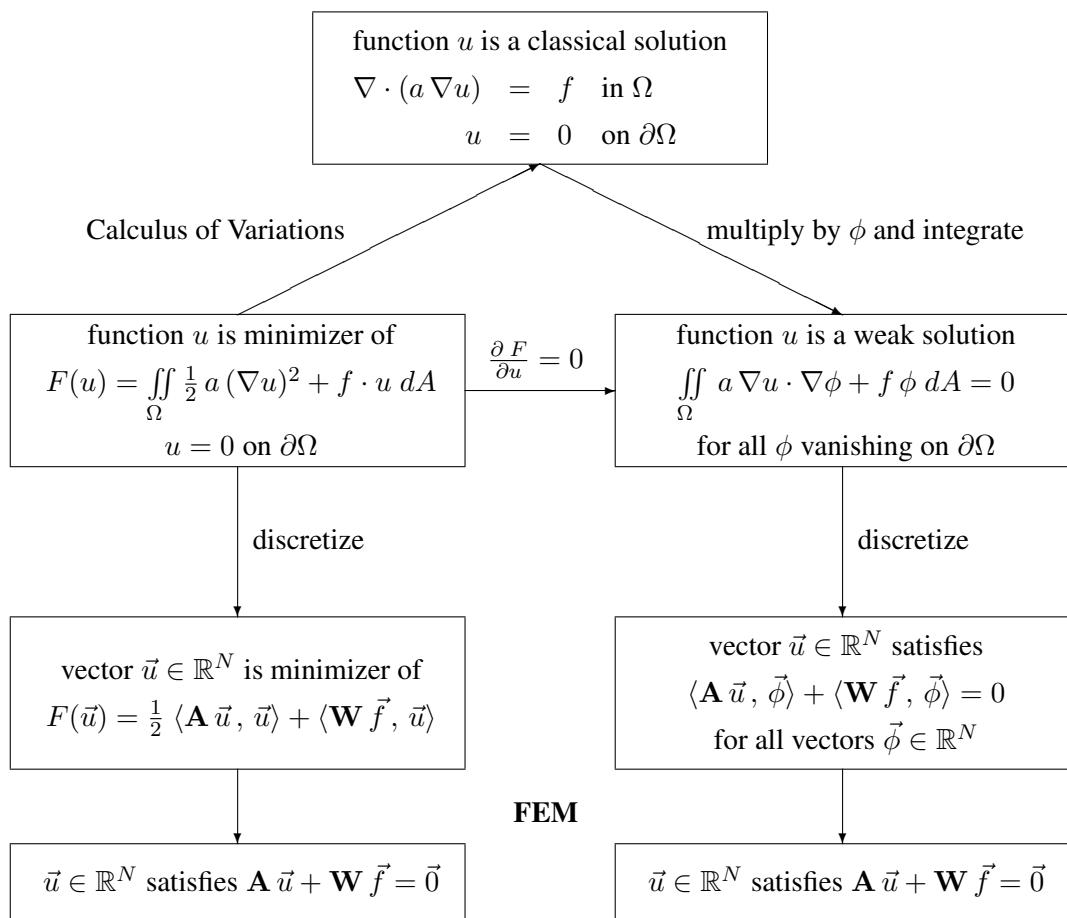


Figure 6.1: Classical and weak solutions, minimizers and FEM

- Examine weak solutions of second order boundary value problems.
- Present the theoretical results for self-adjoint BVP as minimization problems. State the required approximation results.
- Examine the Galerkin formulation for second order BVP.
- Develop algorithms for triangular second order elements.
- Compare first and second order triangular elements.
- Give a very brief introduction to quadrilateral elements.
- To finish this chapter apply the FEM to a tumor growth problem.

## 6.2 Piecewise Linear Finite Elements

Start by developing an algorithm for a FEM solution of the model boundary value problem

$$\begin{aligned} \nabla \cdot (a \nabla u) - b u &= f && \text{for } (x, y) \in \Omega \\ u &= 0 && \text{for } (x, y) \in \Gamma = \partial\Omega \end{aligned} .$$

Thus minimize the functional

$$F(u) = \iint_{\Omega} \frac{1}{2} a (\nabla u)^2 + \frac{1}{2} b \cdot u^2 + f \cdot u \, dA \quad (6.3)$$

amongst all functions  $u$  vanishing on the boundary  $\Gamma = \partial\Omega$ .

This simple problem is used to explain the basic algorithms and can be implemented with MATLAB or Octave<sup>1</sup>. The purpose of the code is to illustrate the necessary degree of complexity.

### 6.2.1 Discretization, Approximation and Assembly of the Global Stiffness Matrix

In the above functional  $F(u)$  integrals over the domain  $\Omega \subset \mathbb{R}^2$  have to be computed. To discretize this integration use a triangulation of the domain, using grid points  $(x_i, y_i) \in \Omega$ ,  $1 \leq i \leq n$ . The grid points are called **nodes** of the FEM approach and each node leads to one degree of freedom, i.e. one unknown function value and one equation to be solved. On each triangle  $T_k$  replace the function  $u$  by a polynomial of degree 1, i.e. a linear function. These polynomials are completely determined by their values at the three corners of the triangle. Integrals over the complete domain  $\Omega$  are split up into integrals over each triangle and then a summation, i.e.

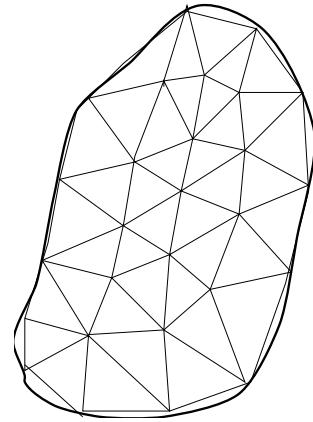
<sup>1</sup>The codes presented are simplifications taken from the package FEMoctave by this author and are intended for instructional purposes only.

$$\iint_{\Omega} \dots dA \approx \sum_k \iint_{T_k} \dots dA.$$

The gradient of  $u$  is replaced by the gradient of the piecewise polynomials. Since the partial derivatives of a linear function are constants the gradient is constant on each triangle. Each contribution is written in the form

$$\iint_{T_k} \dots dA \approx \frac{1}{2} \langle \mathbf{A}_k \vec{u}_k, \vec{u}_k \rangle + \langle \mathbf{W}_k \vec{f}_k, \vec{u}_k \rangle$$

where the  $3 \times 3$  matrix  $\mathbf{A}_k$  is the **element stiffness matrix** and  $\mathbf{W}_k$  is a weight matrix, most often  $\mathbf{W}_k$  is a diagonal matrix.



If the above process is carried out correctly the functional  $F$  is replaced by a discrete approximation

$$F(u) \approx \frac{1}{2} \langle \vec{u}, \mathbf{A} \vec{u} \rangle + \langle \mathbf{W} \vec{f}, \vec{u} \rangle$$

with a symmetric, positive definite matrix  $\mathbf{A}$ . This expression is minimized by the solution of the linear system of equations

$$\mathbf{A} \vec{u} = -\mathbf{W} \vec{f}.$$

It is one of the most important advantages of the Finite Element Method that it can be applied on irregularly shaped domains. For rectangular domains  $\Omega$  the finite difference methods could be used to solve the BVP in his section. Applying the finite differences to non-rectangular domains can be very challenging.

For one possible construction of finite elements the value of the unknown function at each of the nodes is one degree of freedom. Thus for each triangle we have exactly 3 degrees of freedom and the total number  $N$  of (interior) nodes corresponds to the number of unknowns. The element stiffness matrices  $\mathbf{A}_T$  will be of size  $3 \times 3$  and the global stiffness matrix  $\mathbf{A}$  is a  $N \times N$  matrix.

Thus a rather general FEM algorithm is described by

- Decompose the domain  $\Omega \subset \mathbb{R}^2$  in triangles and determine the degrees of freedom.
- Create the  $N \times N$  matrix  $\mathbf{A}$ , originally filled with zeros, and the vector  $\vec{f} \in \mathbb{R}^N$ .
- For each triangle  $T$ :
  - Compute the element stiffness matrix  $\mathbf{A}_T$  and the vector  $\mathbf{W}_T \vec{f}_T$ . Use equation (6.3) and a numerical integration scheme.
  - Add matrix and vector to the global structure.
- Solve the global system  $\mathbf{A} \vec{u} + \mathbf{W} \vec{f} = \vec{0}$  for the vector of unknown values in  $\vec{u}$ .
- Visualize the solution and make the correct conclusion for your application.

The actual computation of an element stiffness matrix will be examined carefully in the subsequent sections. It is the most important building block of any FEM approach.

## 6.2.2 Integration over one Triangle

If a triangle  $T$  is given by its three corners at  $\vec{x}_1, \vec{x}_2$  and  $\vec{x}_3 \in \mathbb{R}^2$ , then its area  $A$  can be computed by a cross product<sup>2</sup>

$$A = \frac{1}{2} \|(\vec{x}_2 - \vec{x}_1) \times (\vec{x}_3 - \vec{x}_1)\| = \frac{1}{2} |(x_2 - x_1) \cdot (y_3 - y_1) - (y_2 - y_1) \cdot (x_3 - x_1)|.$$

If the values of a general function  $f$  are given at the three corners of the triangle by  $f_1, f_2$  and  $f_3$  replace the exact function by a linearly interpolated function and find an approximate integral by

$$\iint_T f \, dA \approx A \cdot \frac{f_1 + f_2 + f_3}{3}.$$

Observe that there is a systematic integration error due to replacing the true function by an approximate, linear function.

This leads to the integrals

$$\begin{aligned} \iint_T f \cdot u \, dA &\approx \frac{A}{3} (f_1 u_1 + f_2 u_2 + f_3 u_3) \\ \iint_T \frac{1}{2} b \cdot u^2 \, dA &\approx \frac{1}{2} \frac{A}{3} (b_1 u_1^2 + b_2 u_2^2 + b_3 u_3^2). \end{aligned}$$

Using a vector and matrix notation find

$$\iint_T \frac{1}{2} b \cdot u^2 + f \cdot u \, dA \approx \frac{1}{2} \frac{A}{3} \left\langle \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & b_3 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right\rangle + \frac{A}{3} \left\langle \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}, \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right\rangle.$$

This is one of the contributions in equation (6.3).

## 6.2.3 Integration of $\nabla u \cdot \nabla u$ over one Triangle

To examine the other contribution we first need to compute the gradient of the function  $u$ . If the true function is replaced by a linear interpolation on the triangle, then the gradient is constant on this triangle and can be determined with the help of a normal vector of the plane passing through the three points

$$\begin{pmatrix} x_1 \\ y_1 \\ u_1 \end{pmatrix}, \quad \begin{pmatrix} x_2 \\ y_2 \\ u_2 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} x_3 \\ y_3 \\ u_3 \end{pmatrix}.$$

The situation of one triangle in the  $xy$  plane and the corresponding triangle in the  $(xyu)$ -space is shown in Figure 6.2. A normal vector  $\vec{n}$  is given by the vector product  $\vec{n} = \vec{a} \times \vec{b}$ .

$$\begin{aligned} \vec{n} &= \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ u_2 - u_1 \end{pmatrix} \times \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \\ u_3 - u_1 \end{pmatrix} = \begin{pmatrix} +(y_2 - y_1) \cdot (u_3 - u_1) - (u_2 - u_1) \cdot (y_3 - y_1) \\ -(x_2 - x_1) \cdot (u_3 - u_1) + (u_2 - u_1) \cdot (x_3 - x_1) \\ +(x_2 - x_1) \cdot (y_3 - y_1) - (y_2 - y_1) \cdot (x_3 - x_1) \end{pmatrix} \\ &= \lambda \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ -1 \end{pmatrix} \quad \text{with} \quad \lambda = -2A. \end{aligned}$$

<sup>2</sup>Quietly extend the vector  $\vec{x} = (x, y) \in \mathbb{R}^2$  to a vector  $\vec{x} = (x, y, 0) \in \mathbb{R}^3$ .

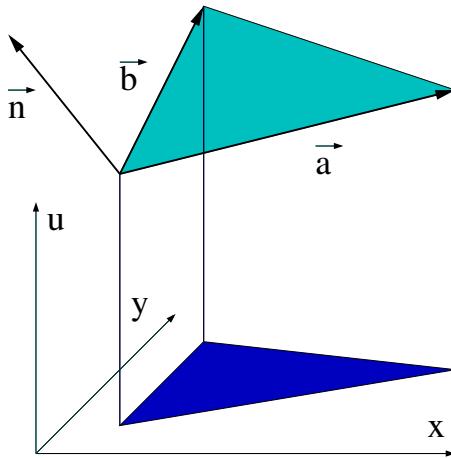


Figure 6.2: One triangle in space (green) and projected to plane (blue)

The third component of this vector equals twice the oriented<sup>3</sup> area  $A$  of the triangle. To obtain the gradient in the first two components, the vector has to be normalized, such that the third component is equals  $-1$ .

$$\nabla u = \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix} = \frac{-1}{2A} \begin{pmatrix} +(y_2 - y_1) \cdot (u_3 - u_1) - (u_2 - u_1) \cdot (y_3 - y_1) \\ -(x_2 - x_1) \cdot (u_3 - u_1) + (u_2 - u_1) \cdot (x_3 - x_1) \end{pmatrix}$$

This formula can be written as

$$\nabla u = \frac{-1}{2A} \begin{bmatrix} (y_3 - y_2) & (y_1 - y_3) & (y_2 - y_1) \\ (x_2 - x_3) & (x_3 - x_1) & (x_1 - x_2) \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \frac{-1}{2A} \mathbf{M} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \quad (6.4)$$

and leads to

$$\langle \nabla u, \nabla u \rangle = \frac{1}{4A^2} \langle \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle = \frac{1}{4A^2} \langle \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \mathbf{M}^T \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle.$$

Thus conclude

$$\iint_T \frac{1}{2} a (\nabla u)^2 dA \approx A \frac{a_1 + a_2 + a_3}{3 \cdot 2} \langle \nabla u, \nabla u \rangle = \frac{a_1 + a_2 + a_3}{2 \cdot 3 \cdot 4A} \langle \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \mathbf{M}^T \mathbf{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle.$$

It is an exercise to verify that the matrix  $\mathbf{M}^T \mathbf{M}$  is symmetric and positive semidefinite. The expression vanishes if and only if  $u_1 = u_2 = u_3$ . This corresponds to a horizontal plane in Figure 6.2.

## 6.2.4 The Element Stiffness Matrix

Collecting the above results find

$$\iint_T \frac{1}{2} a (\nabla u)^2 + \frac{1}{2} b u^2 + f \cdot u dA \approx \frac{1}{2} \langle \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \mathbf{A}_T \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \rangle + \langle \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \mathbf{W}_T \vec{f}_T \rangle,$$

<sup>3</sup>We quietly assumed that the third component of  $\vec{n}$  is positive. As we use only the square of the gradient the influence of this ignorance will disappear.

where

$$\mathbf{A}_T = \frac{a_1 + a_2 + a_3}{12A} \mathbf{M}^T \mathbf{M} + \frac{A}{3} \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & b_3 \end{bmatrix} \quad (6.5)$$

$$\mathbf{W}_T \vec{\mathbf{f}}_T = \frac{A}{3} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}. \quad (6.6)$$

If  $b_i \geq 0$  then the element stiffness matrix  $\mathbf{A}_T$  is positive semidefinite.

The above can readily be implemented in Octave.

#### ElementContribution.m

```
function [elMat,elVec] = ElementContribution(corners,aFunc,bFunc,fFunc)

elMat = zeros(3); elVec = zeros(3,1);
if is_scalar(aFunc) aV = aFunc*ones(3,1);else aV=feval(aFunc,corners);endif
if is_scalar(bFunc) bV = bFunc*ones(3,1);else bV=feval(bFunc,corners);endif
if is_scalar(fFunc) fV = fFunc*ones(3,1);else fV=feval(fFunc,corners);endif
area = ((corners(2,1)-corners(1,1))*(corners(3,2)-corners(1,2))-...
          (corners(2,2)-corners(1,2))*(corners(3,1)-corners(1,1)) )/2;
M = [corners(3,2)-corners(2,2),corners(1,2)-corners(3,2),corners(2,2)-corners(1,2);
      corners(3,1)-corners(2,1),corners(1,1)-corners(3,1),corners(2,1)-corners(1,1)];

elMat = sum(aV)/(12*area)*M'*M + area/3*diag(bV);
elVec = area/3*fV;
end%function
```

### 6.2.5 Triangularization of the Domain $\Omega \subset \mathbb{R}^2$

One of the first, and important, task for FEM is the generation of a mesh. For the current setup decompose a domain  $\Omega \subset \mathbb{R}^2$  into triangles. Any domain limited by straight edges can be decomposed into triangles. This is (almost) always performed by suitable software.

As example consider the code `triangle` from [[www:triangle](#)], which can be called by the FEMoctave package within Octave. Apply it to a domain with corners at  $(1, 0)$ ,  $(2, 0)$ ,  $(2, 2)$  and  $(0, 1)$  and generate a mesh visible in Figure 6.3. The typical value of the area of one triangle is 0.1 .

#### Octave

```
ProbName = 'TestLinear';
xy = [0,0,-1;2,0,-1;2,2,-1;0,1,-1];
Mesh = CreateMeshTriangle(ProbName,xy ,0.1)
figure(1); FEMtrimesh(Mesh.elem,Mesh.nodes(:,1),Mesh.nodes(:,2))
xlabel('x'); ylabel('y');
```

The generated mesh consists of 33 nodes, forming 48 triangles. On the boundary the values of the function  $u$  are given by the known function  $g(x, y)$  and thus not all nodes are degrees of freedom for the FEM problem. As a result find 17 interior points in this mesh and thus the resulting system of equations will have 17 equations and unknowns.

### 6.2.6 Assembly of the System of Linear Equations

With the above results one can now assemble the global stiffness matrix, i.e. generate the system of linear equations to be solved.

The structure `Mesh` contains multiple fields, amongst them:

- The variable `nodes` contains the coordinates of the numbered nodes and the information whether the node is on the boundary or in the interior of the domain.
- The variable `elem` contains a list of all the triangles and the node numbers of the corners.
- The variable `edges` contains a list of all the boundary edges of the discretized domain.

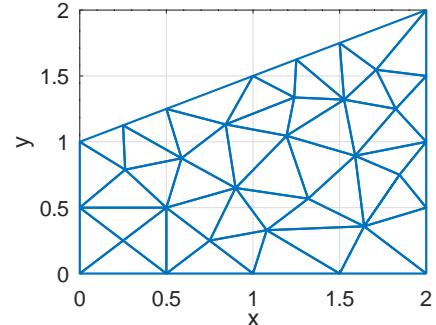


Figure 6.3: A small mesh of a simple domain in  $\mathbb{R}^2$

For each triangle we find the element stiffness matrix  $\mathbf{A}_T$ , which will contribute to the global stiffness matrix  $\mathbf{A}$ . As an example consider Figure 6.4 with 3 nodes for each triangle. The entries of  $\mathbf{A}_T$  have to be added to the previous entries in the global matrix  $\mathbf{A}$  and accordingly the entries of  $\vec{b}_k$  have to be added to the global vector  $\vec{f}$ .

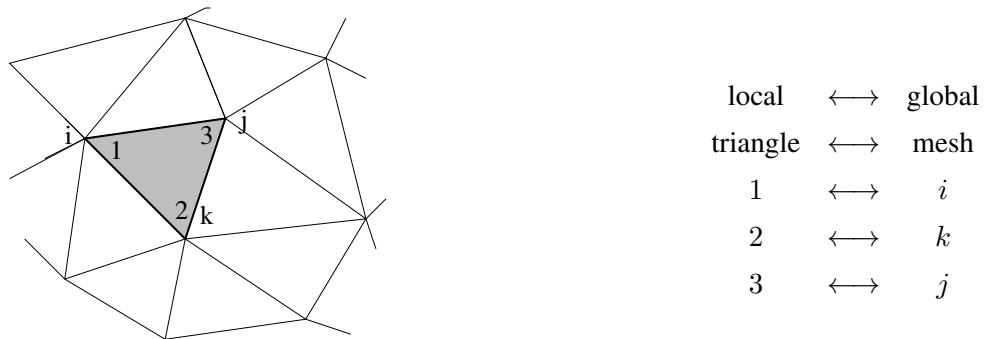


Figure 6.4: Local and global numbering of nodes

$$\mathbf{A}_T = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \rightarrow \quad \mathbf{A} = \mathbf{A} + \begin{array}{c} \text{row } i \\ \text{row } j \\ \text{row } k \end{array} \begin{bmatrix} \text{col } i & \text{col } j & \text{col } k \\ \vdots & \vdots & \vdots \\ \cdots & a_{11} & \cdots & a_{13} & \cdots & a_{12} & \cdots \\ & \vdots & \ddots & \vdots & & \vdots & \\ \cdots & a_{31} & \cdots & a_{33} & \cdots & a_{32} & \cdots \\ & \vdots & & \vdots & \ddots & \vdots & \\ \cdots & a_{21} & \cdots & a_{23} & \cdots & a_{22} & \cdots \\ & \vdots & & \vdots & & \vdots & \ddots \end{bmatrix}$$

The above construction allows to verify that symmetry of the element stiffness matrices  $\mathbf{A}_T$  carries over to the global matrix  $\mathbf{A}$ . If all element stiffness matrices are positive definite the global stiffness matrix will be positive definite. Once the matrix and the right hand side vector are generated, a linear system of equations has to be solved. Thus results from chapter 2 have to be used. This assembling of the global stiffness matrix can be implemented in (almost) any programming language. As an example you might have a look at the Octave package FEMoctave on GitHub<sup>4</sup>.

<sup>4</sup>Use [github.com/AndreasStahel/FEMoctave](https://github.com/AndreasStahel/FEMoctave) or the documentation [web.math1.bfh.science/FEMoctave/FEMdoc.pdf](http://web.math1.bfh.science/FEMoctave/FEMdoc.pdf).

not in class

### 6.2.7 The Algorithm of Cuthill and McKee to Reduce Bandwidth

The numbering of the nodes of a mesh created on a given domain will determine the bandwidth of the resulting stiffness matrix  $\mathbf{A}$  for the given differential equation to be solved by the FEM<sup>5</sup>. For linear elements on triangles each (interior) node leads to one degree of freedom, the value of the function at this node. Find  $a_{i,j} \neq 0$  if the nodes with number  $i$  and  $j$  share a common triangle. In view of the result in Section 2.6.4 one should aim for a numbering leading to a small bandwidth. One possible (and rather efficient) algorithm is known as the Cuthill–McKee<sup>6</sup> algorithm.

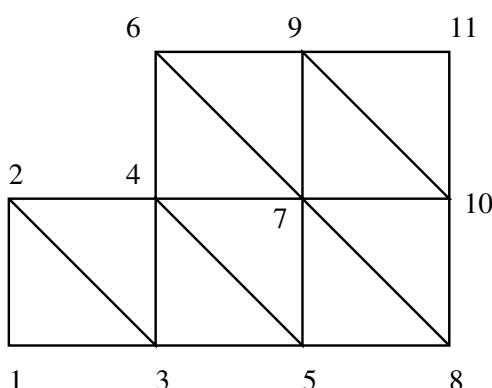
```

choose a starting node and give it the number 1
while there are unnumbered nodes
    pick the next node
    find all its neighbors not yet numbered
    sort them, using the the nodes with fewer of connections
        to unnumbered nodes first
    give them the next free numbers
endwhile

```

Table 6.1: Algorithm of Cuthill–McKee

There are different criteria on how to choose an optimal first node. Tests show that nodes with few neighbors are often good starting nodes. Thus one may choose nodes with the minimal number of neighbors. Also good candidates are nodes at extreme points of the discretized domain. A more detailed description of the Cuthill–McKee algorithm and how to choose starting points is given in [LascTheo87].



	1	2	3	4	5	6	7	8	9	10	11
1	*	*	*								
2	*	*	*	*							
3	*	*	*	*	*						
4	*	*	*	*	*	*	*				
5		*	*	*		*	*				
6			*	*	*	*	*				
7			*	*	*	*	*	*	*		
8				*	*	*				*	
9					*	*		*	*	*	
10						*	*	*	*	*	
11							*	*	*		

Figure 6.5: Numbering of a simple mesh by Cuthill–McKee

The algorithm is illustrated by numbering the simple mesh in Figure 6.5. On the right the structure of the nonzero elements in the resulting stiffness matrix is shown. The band structure is clearly recognizable.

- The first node is chosen, since it has only two neighbors and is at one end of the domain.
- Node 1 has two neighbors, number 2 is given to the node above, since it has only one free neighbor. The node on the right (two free neighbors) of 1 will be number 3 .

<sup>5</sup>When using iterative solvers with sparse matrices, the reduction of bandwidth is irrelevant. Since many (newer) direct solvers internally renumber equations and variable the importance of the Cuthill–McKee algorithm has clearly diminished.

<sup>6</sup>Named after Elizabeth Cuthill and James McKee.

- Node 2 has only one free node with number 4 .
- Node 3 now has also only one free node left, number 5 .
- Of the two free neighbors of node 4, the one above has fewer free nodes and thus will receive number 6. The node on the right will be number 7 .
- The only free neighbor of node 5 will now receive number 8 .
- The only free neighbor of node 6 will now receive number 9 .
- The only free neighbor of node 7 will now receive number 10 .
- The last node will be number 11 .

As an example examine a BVP on the domain shown in Figure 6.6, where the mesh was generated by the program `triangle` (see [[www:triangle](#)]). The mesh has 518 nodes and the original numbering leads to a semi-bandwidth of 515, i.e. no band structure. Nonetheless we have a **sparse** matrix, since only 3368 entries are nonzero (i.e. 1.25%). The nonzero elements in the matrix  $\mathbf{A}$  are shown in Figure 6.7, before and after applying the Cuthill–McKee algorithm. The new semi-bandwidth is 28. If finer meshes (more nodes) are used, then the improvements due to a good renumbering of the nodes will be even larger.

Within the band only 21% of the entries are not zero, i.e. we still have a certain sparsity within the band. The algorithm of Cholesky can not take advantage of this sparsity within the band, but iterative methods can, as examined in Section 2.7.

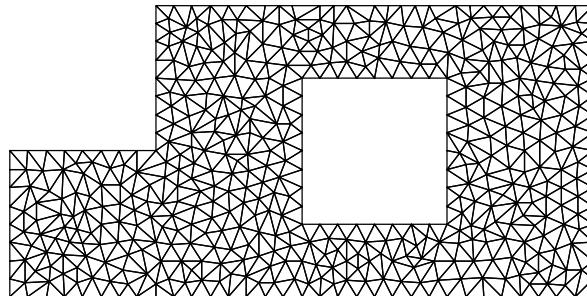


Figure 6.6: Mesh generated by `triangle`

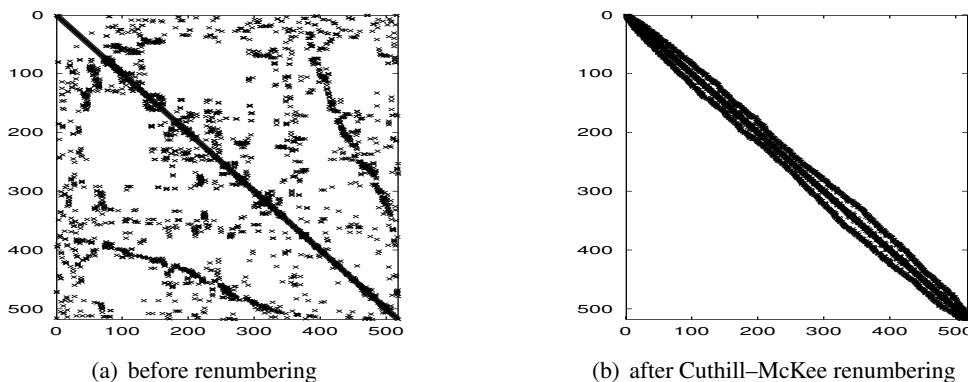


Figure 6.7: Structure of the nonzero entries in a stiffness matrix

### 6.2.8 A First Solution by the FEM

For a numerical example examine the domain  $\Omega \subset \mathbb{R}^2$  given in Figure 6.3 and solve the boundary value problem

$$\begin{aligned}-\nabla \cdot (\nabla u) &= 5 && \text{for } (x, y) \in \Omega \\ u &= 0 && \text{for } (x, y) \in \Gamma = \partial\Omega\end{aligned}$$

Apply the algorithms from the preceding sections to find an approximate solution. The code below generates a mesh on a non-rectangular domain. The triangles have an area of approximately 0.1. The resulting mesh consists of 48 triangles based on 33 nodes. Since 16 nodes are on the boundary the resulting system of linear equations is of size  $17 \times 17$ .

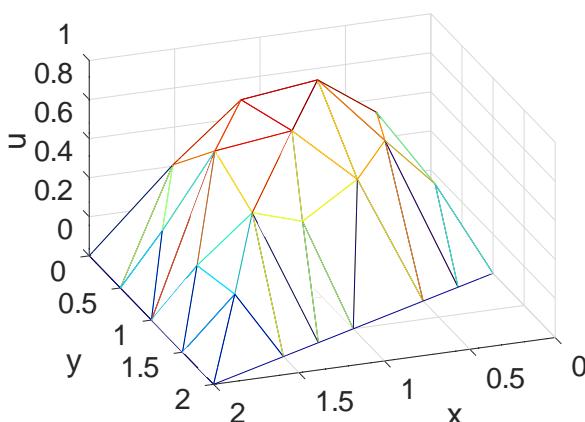
[run demo](#)

#### TestLinear.m

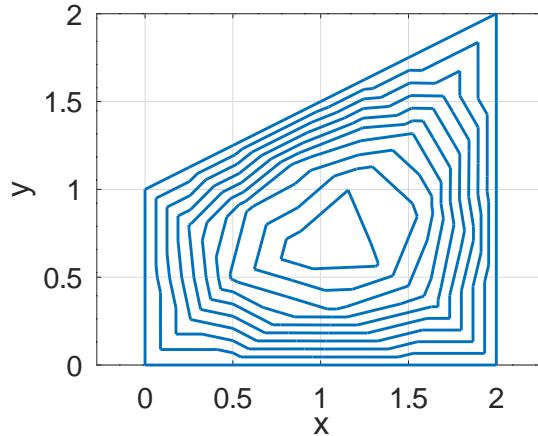
```
Mesh = CreateMeshTriangle('TestLinear', [0,0,-1;2,0,-1;2,2,-1;0,1,-1], 0.1);
figure(1); FEMtrimesh(Mesh.elem, Mesh.nodes(:,1), Mesh.nodes(:,2))
xlabel('x'); ylabel('y');

u = BVP2D(Mesh, 1, 0, 0, 0, 5, 0, 0, 0);
figure(2); FEMtrimesh(Mesh.elem, Mesh.nodes(:,1), Mesh.nodes(:,2), u)
xlabel('x'); ylabel('y'); zlabel('u'); view([160, 35])

figure(3); tricontour(Mesh.elem, Mesh.nodes(:,1), Mesh.nodes(:,2), u, linspace(0, 1, 11)+eps)
xlabel('x'); ylabel('y'); axis equal
```



(a) surface of a solution



(b) contour levels of a solution

Figure 6.8: A first FEM solution

The results in Figure 6.8 are obviously far from optimal.

- The level curves are very ragged. This can be improved by generating a finer mesh and consequently a larger system of equations to be solved. A test run with a typical area of 0.001 (diameter divided by 10) leads to 2470 nodes, forming 4750 triangles. The resulting matrix  $\mathbf{A}$  has size  $2282 \times 2282$ . The level curves are very smooth.
- There is no control over the approximation error yet. The corresponding results will be examined in Section 6.4, starting on page 418.
- It is not clear whether the approximation by piecewise linear functions was a good choice. This will be clarified in Section 6.4 and a more efficient approach, using second order elements, will be examined in Section 6.5.

- The numerical integration was done with a rather simplistic idea. A better approach will be presented in Section 6.5.
- It is not obvious how to adapt the above approach to more general problems. This will be examined in following sections.

### 6–1 Example : A finite difference stencil as special case of the FEM

Examine the boundary value problem

$$\begin{aligned} \frac{\partial^2}{\partial x^2} u(x, y) + \frac{\partial^2}{\partial y^2} u(x, y) &= f(x, y) && \text{for } (x, y) \in \Omega = (0, a) \times (0, b) \\ u(x, y) &= 0 && \text{for } (x, y) \text{ on boundary } \partial\Omega \end{aligned}$$

This corresponds to minimizing the functional

$$F(u) = \int_{\Omega} \frac{1}{2} \left( \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial u}{\partial y} \right)^2 \right) + u \cdot f \, dA.$$

Use the domain  $\Omega$  in Figure 6.9 with a uniform, rectangular mesh, which has  $nx$  interior nodes in  $x$  direction and  $ny$  interior nodes in  $y$  direction. In the shown example  $nx = 18$  and  $ny = 5$ . The step sizes are given by  $hx = \frac{a}{nx+1}$  and  $hy = \frac{b}{ny+1}$ .

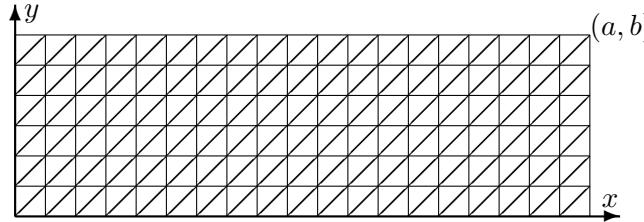


Figure 6.9: A simple rectangular mesh

In the mesh in Figure 6.9 there are two types of triangles, shown in Figure 6.10 and thus one can compute all element contributions with the help of those two standard triangles.

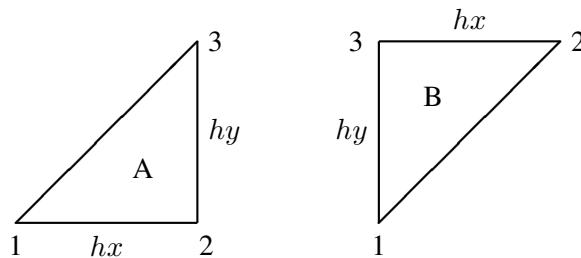


Figure 6.10: The two types of triangles in a rectangular mesh

If the values  $u_i$  of the function at the three corners of a type A triangle are known, then compute the gradient of the linearly interpolated function by

$$\nabla u = \frac{-1}{2 \text{ area}} \mathbf{M} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \frac{-1}{2 \text{ area}} \begin{bmatrix} (y_3 - y_2) & (y_1 - y_3) & (y_2 - y_1) \\ (x_2 - x_3) & (x_3 - x_1) & (x_1 - x_2) \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

$$= \frac{-1}{hx \cdot hy} \begin{bmatrix} hy & -hy & 0 \\ 0 & hx & -hx \end{bmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

and thus

$$\mathbf{M}^T \cdot \mathbf{M} = \begin{bmatrix} hy & 0 \\ -hy & hx \\ 0 & -hx \end{bmatrix} \cdot \begin{bmatrix} hy & -hy & 0 \\ 0 & hx & -hx \end{bmatrix} = \begin{bmatrix} hy^2 & -hy^2 & 0 \\ -hy^2 & hy^2 + hx^2 & -hx^2 \\ 0 & -hx^2 & hx^2 \end{bmatrix}.$$

Observe that the zeros in the off-diagonal corners are based on the facts  $y_1 = y_2$  and  $x_2 = x_3$ . Equation (6.5) leads to the element stiffness matrix

$$\mathbf{A}_A = \frac{a_1 + a_2 + a_3}{12 \text{ area}} \mathbf{M}^T \cdot \mathbf{M} = \frac{1}{2 hx hy} \begin{bmatrix} hy^2 & -hy^2 & 0 \\ -hy^2 & hy^2 + hx^2 & -hx^2 \\ 0 & -hx^2 & hx^2 \end{bmatrix}$$

and the vector

$$\vec{b}_A = \frac{A}{3} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \frac{hx hy}{6} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}.$$

Similar calculations lead to the element stiffness matrix for triangles of type B

$$\mathbf{A}_B = \frac{1}{2 hx hy} \begin{bmatrix} hx^2 & 0 & -hx^2 \\ 0 & hy^2 & -hy^2 \\ -hx^2 & -hy^2 & hy^2 + hx^2 \end{bmatrix}$$

and  $\vec{b}_B = \vec{b}_A$ .

Now construct the system of linear equations for the boundary value problem on the mesh in Figure 6.9. For this consider the mesh point in column  $i$  and row  $j$  (starting at the bottom) in the mesh and denote the value of the solution at this point by  $u_{i,j}$ . In Figure 6.9 observe that  $u_{i,j}$  is directly connected to 6 neighboring points and 6 triangles are used to built the connections. This is visualized in Figure 6.11, the left part of that figure represents the **stencil** at this point in the mesh.

Start by examining the contributions to  $\int_{\Omega} u \cdot f \, dA$  involving the coefficients  $u_{i,j}$ . As the coefficients in  $\vec{b}_A = \vec{b}_B$  are all constant, obtain six contributions of the size  $\frac{hx hy}{6} f_{i,j}$  leading to a total of  $hx hy f_{i,j}$

$$f_{i,j} \rightarrow 6 \left( \frac{hx hy}{6} \right) = hx hy.$$

With similar arguments examine the contributions to  $\frac{1}{2} \int_{\Omega} \nabla u \cdot \nabla u \, dA$ . Use the element matrices  $\mathbf{A}_A$ ,  $\mathbf{A}_B$  and Figure 6.11 to verify the contributions below.

$$\begin{aligned} u_{i,j} &\rightarrow \frac{1}{2 hx hy} (hy^2 + hx^2 + (hx^2 + hy^2) + hy^2 + hx^2 + (hx^2 + hy^2)) = \frac{2 (hx^2 + hy^2)}{hx hy} \\ u_{i+1,j} &\rightarrow \frac{1}{2 hx hy} (-hx^2 + 0 + 0 + 0 + 0 - hx^2) = \frac{-hx^2}{hx hy} \\ u_{i-1,j} &\rightarrow \frac{1}{2 hx hy} (0 + 0 - hx^2 - hx^2 + 0 + 0) = \frac{-hx^2}{hx hy} \end{aligned}$$

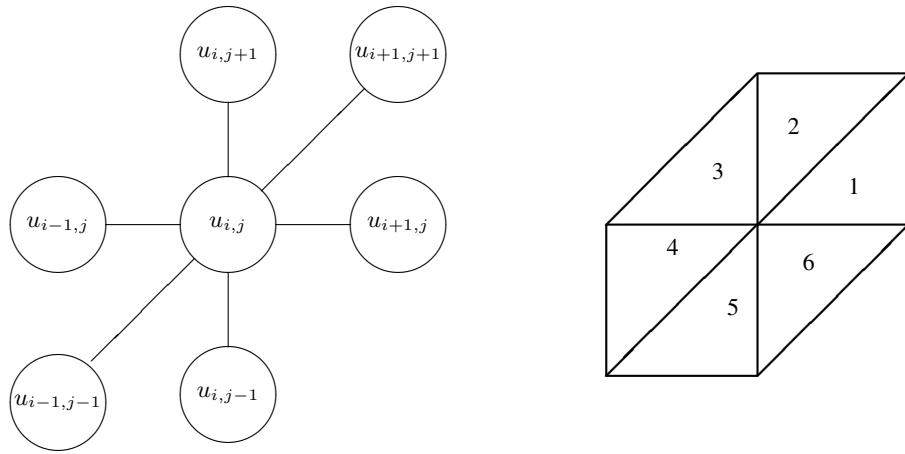


Figure 6.11: FEM stencil and neighboring triangles of a mesh point

$$\begin{aligned}
 u_{i,j+1} &\rightarrow \frac{1}{2hxhy} (0 - hy^2 - hy^2 + 0 + 0 + 0) = \frac{-hy^2}{hxhy} \\
 u_{i,j-1} &\rightarrow \frac{1}{2hxhy} (0 + 0 + 0 + 0 - hy^2 - hy^2) = \frac{-hy^2}{hxhy} \\
 u_{i+1,j+1} &\rightarrow \frac{1}{2hxhy} (0 + 0 + 0 + 0 + 0 + 0) = 0 \\
 u_{i-1,j-1} &\rightarrow \frac{1}{2hxhy} (0 + 0 + 0 + 0 + 0 + 0) = 0
 \end{aligned}$$

Observe that the two diagonal connections in the stencil lead to zero contributions. This is correct for the rectangular mesh. Thus the resulting equation for the degree of freedom  $u_{i,j}$  is given by

$$u_{i,j} \frac{2(hx^2 + hy^2)}{hxhy} - u_{i+1,j} \frac{hx^2}{hxhy} - u_{i-1,j} \frac{hx^2}{hxhy} - u_{i,j+1} \frac{hy^2}{hxhy} - u_{i,j-1} \frac{hy^2}{hxhy} + f_{i,j} hxhy = 0$$

or by rearranging

$$\frac{-u_{i+1,j} + 2u_{i,j} - u_{i-1,j}}{hx^2} + \frac{-u_{i,j+1} + 2u_{i,j} - u_{i,j-1}}{hy^2} = -f_{i,j}. \quad (6.7)$$

For the special case  $hx = hy$  obtain

$$\frac{1}{hx^2} (4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}) = -f_{i,j}$$

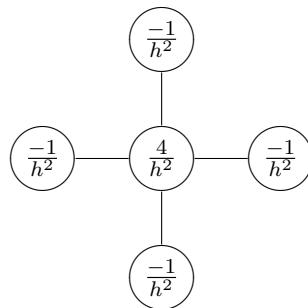
This is the **finite difference** approximation to the differential expression  $-\Delta u = -(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$  and leads to the well known finite difference stencil in Figure 6.12, used to solve boundary value problems in two variables, see Section 4.4.2.

◇

### 6.2.9 Contributions to the Approximation Error

The algorithm in the previous section constructs an approximation of the exact solution of the boundary value problem. It is important to identify possible sources of errors of the approximate solution:

1. The true solution is approximated by a piecewise linear function on each triangle.

Figure 6.12: Finite difference stencil for  $-u_{xx} - u_{yy}$  for  $h = hx = hy$ 

2. On each triangle we have to use a **numerical integration** to determine the contributions.
3. Not all domains  $\Omega \subset \mathbb{R}^2$  can be decomposed into triangles.

In the following sections we will examine the first two contributions, and methods to minimize the errors.

- Using quadratic polynomials on each triangle will (usually) lead to smaller errors.
- Using the brilliant integration methods by Gauss, the influence of the integration error will be very small.

## 6.3 Classical and Weak Solutions

not in class

In this section some of the more mathematical notations are presented. They are used to formulate the essential convergence results for FEM.

### 6.3.1 Weak Solutions of a System of Linear Equations

As an introduction to the concept of weak solutions examine systems of linear equations, i.e. for a matrix  $\mathbf{A}$  and a vector  $\vec{f}$  search for solutions  $\vec{u}$  of  $\mathbf{A} \vec{u} = \vec{f}$ . The same idea will be applied to boundary value problems later in this section.

A vector  $\vec{x} \in \mathbb{R}^N$  vanishes if and only if it is orthogonal to all vectors  $\vec{\phi} \in \mathbb{R}^N$  or orthogonal to all vectors  $\vec{\phi}_i \in \mathbb{R}^N$  ( $i = 1, 2, \dots, N$ ) which form a basis of  $\mathbb{R}^N$ .

$$\vec{x} = \vec{0} \iff \langle \vec{x}, \vec{\phi} \rangle = 0 \quad \text{for all } \vec{\phi} \in \mathbb{R}^N \iff \langle \vec{x}, \vec{\phi}_i \rangle = 0 \quad \text{for } i = 1, 2, \dots, N$$

This obvious fact also applies to Hilbert spaces, which will be functions space for our boundary value problems to be examined.

Let  $H$  be a Hilbert space with a finite dimensional subspace  $H_h$ . Let  $\vec{\phi}_i$  for  $i = 1, 2, 3, \dots, N$  be a basis of  $H_h$  and thus any vector  $\vec{x} \in H_h$  can be written in the form

$$\vec{x} = \sum_{j=1}^N x_j \vec{\phi}_j.$$

With the above examine a weak solution of a system of linear equations and find the following.

$$\begin{aligned} \mathbf{A} \vec{u} = \vec{f} \in H_h &\iff \mathbf{A} \vec{u} - \vec{f} = 0 \\ &\iff \langle \mathbf{A} \vec{u} - \vec{f}, \vec{\phi} \rangle = 0 \quad \text{for all } \vec{\phi} \in H_h \\ &\iff \langle \mathbf{A} \vec{u} - \vec{f}, \vec{\phi}_j \rangle = 0 \quad \text{for all } j = 1, 2, \dots, N \end{aligned}$$

### Taking inhomogenous boundary conditions into account

For the boundary value problems the partial differential equation in the domain has to be solved together with boundary conditions. Thus we have to be able to take those into account for the correct definition of a weak solution.

Let  $H_0 = \{v \in H \mid \mathbf{B}_1 v = 0\}$  be a linear subspace of  $H$  with basis  $\varphi_j$ . Examine the system of equations

$$\begin{aligned}\mathbf{A} u &= f \\ \mathbf{B}_1 u &= g_1 \\ \mathbf{B}_2 u &= g_2.\end{aligned}$$

If  $u_1 \in H$  satisfies  $\mathbf{B}_1 u_1 = g_1$ , then write  $u = u_1 + v$  and arrive at the modified problem

$$\begin{aligned}\mathbf{A} v &= f - \mathbf{A} u_1 \\ \mathbf{B}_1 v &= 0 \\ \mathbf{B}_2 v &= g_2 - \mathbf{B}_2 u_1.\end{aligned}$$

Thus examine weak solutions of linear systems of equations additional conditions.

$$\mathbf{A} v = f - \mathbf{A} u_1 \iff \langle \mathbf{A} v - f + \mathbf{A} u_1, \phi \rangle = 0 \quad \text{for all } \phi \text{ with } \mathbf{B}_1 \phi = 0$$

### 6.3.2 Classical Solutions and Weak Solutions of Differential Equations

Examine boundary value problems of the form (6.8) and define classical and weak solutions to this problems. The main point of this section is to motivate<sup>7</sup> that being a classical is equivalent to being a weak solution.

$$\begin{aligned}-\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u &= f && \text{for } (x, y) \in \Omega \\ u &= g_1 && \text{for } (x, y) \in \Gamma_1 \\ \vec{n} \cdot (a \nabla u + u \vec{b}) &= g_2 + g_3 u && \text{for } (x, y) \in \Gamma_2\end{aligned}\tag{6.8}$$

The functions  $a, b, f$  and  $g_i$  are known functions and we have to determine the solution  $u$ , all depending on the independent variables  $(x, y) \in \Omega \subset \mathbb{R}^2$ . The vector  $\vec{n}$  is the **outer unit normal vector**. The expression

$$\vec{n} \cdot \nabla u = n_1 \frac{\partial u}{\partial x} + n_2 \frac{\partial u}{\partial y} = \frac{\partial u}{\partial \vec{n}}$$

equals the directional derivative of the function  $u$  in the direction of the outer normal  $\vec{n}$ .

Consider a smooth test-function  $\phi$  vanishing on the Dirichlet boundary  $\Gamma_1$  and a classical solution  $u$  of the boundary value problem (6.8). Then multiply the differential equation in (6.8) by  $\phi$  and integrate over the domain  $\Omega$ .

$$\begin{aligned}0 &= -\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u - f \\ 0 &= \iint_{\Omega} \phi \left( -\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u - f \right) dA \\ &= \iint_{\Omega} \nabla \phi \cdot (a \nabla u + u \vec{b}) + \phi (b_0 u - f) dA - \int_{\Gamma} \phi (a \nabla u + u \vec{b}) \cdot \vec{n} ds \\ &= \iint_{\Omega} \nabla \phi \cdot (a \nabla u + u \vec{b}) + \phi (b_0 u - f) dA - \int_{\Gamma_2} \phi (g_2 + g_3 u) ds\end{aligned}$$

<sup>7</sup>We knowingly ignore regularity problems, i.e. we assume that all expressions and solutions are smooth enough. These problems are carefully examined in books and classes on PDEs.

**6–2 Definition :** A function  $u$  satisfying the above equation for all smooth test functions  $\phi$  vanishing on  $\Gamma_1$  is said to be an **weak solution** of the BVP (6.8).

The above computation shows that classical solutions have to be weak solutions.

If  $u$  is a weak solution, then for all smooth test functions  $\phi$  conclude

$$\begin{aligned} 0 &= \iint_{\Omega} \nabla \phi \cdot (a \nabla u + u \vec{b}) + \phi (b_0 u - f) \, dA - \int_{\Gamma_2} \phi (g_2 + g_3 u) \, ds \\ &= \iint_{\Omega} \phi \left( -\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u - f \right) \, dA + \int_{\Gamma} \phi \left( a \nabla u + u \vec{b} \right) \cdot \vec{n} - \phi (g_2 + g_3 u) \, ds. \end{aligned}$$

In particular find that for all functions  $\phi$  vanishing on all of the boundary  $\Gamma$  we have

$$0 = \iint_{\Omega} \phi \left( -\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u - f \right)$$

and the fundamental lemma of the calculus of variations implies

$$0 = -\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u - f$$

i.e. we have a solution of the differential equation. This in turn now leads to

$$0 = + \int_{\Gamma} \phi \left( a \nabla u + u \vec{b} \right) \cdot \vec{n} - \phi (g_2 + g_3 u) \, ds$$

for all smooth function  $\phi$  on the boundary  $\Gamma_2$  and thus we also recover the boundary condition in (6.8)

$$0 = \left( a \nabla u + u \vec{b} \right) \cdot \vec{n} - (g_2 + g_3 u) .$$

Thus we have equivalence of weak and classical solution (ignoring smoothness problems).

If there were an additional term  $\vec{c} \cdot \nabla u$  in the PDE (6.8) we would have to consider an additional term in the above computations.

$$\begin{aligned} \iint_{\Omega} \phi \vec{c} \cdot \nabla u \, dA &= \iint_{\Omega} \nabla(\phi u \vec{c}) - u \nabla(\phi \vec{c}) \, dA \\ &= - \int_{\Gamma} \phi u \vec{c} \cdot \vec{n} \, ds - \iint_{\Omega} u \nabla \phi \cdot \vec{c} + u \phi \nabla \vec{c} \, dA \end{aligned}$$

## 6.4 Energy Norms and Error Estimates

To illustrate the theoretical background examine the boundary value problem

$$\begin{aligned} -\nabla \cdot (a \nabla u) + b_0 u &= f && \text{for } (x, y) \in \Omega \\ u &= g_1 && \text{for } (x, y) \in \Gamma_1 \\ \vec{n} \cdot (a \nabla u) &= g_2 && \text{for } (x, y) \in \Gamma_2 \end{aligned} \quad (6.9)$$

Solving this problem is equivalent to minimize the functional

$$F(u) = \iint_{\Omega} \frac{1}{2} a (\nabla u)^2 + \frac{1}{2} b_0 u^2 - f u \, dA - \int_{\Gamma_2} g_2 u \, ds .$$

In general the exact  $u_e$  solution can not be found and we have to settle for an approximate solution  $u_h$ , where the parameter  $h$  corresponds to the typical size (e.g. diameter) of the elements used for the approximation. Obviously we hope for the solution  $u_h$  to converge to the exact solution  $u_e$  as  $h$  approaches 0. It is the goal of this section to show under what circumstances this is in fact the case and also to determine the rate of convergence. The methods and ideas used can also be applied to partial differential equations with multiple variables.

### 6.4.1 Basic Assumptions and Regularity Results

For the results of this section to be correct we need assumptions on the functions  $a$ ,  $b$  and  $g_i$ , such that the solution of the boundary value problem is well behaved. Throughout the section we assume:

- $a$ ,  $b_0$  and  $g_i$  are continuous, bounded functions.
- There is a positive number  $\alpha_0$  such that  $0 < \alpha_0 \leq a \leq \alpha_1$  and  $0 \leq b_0 \leq \beta_1$  for all points in the domain  $\Omega \subset \mathbb{R}^2$ .
- The quadratic functional  $F(u)$  is **strictly positive definite**. This condition is satisfied if
  - either on a nonempty section  $\Gamma_1$  of the boundary the Dirichlet boundary condition is imposed
  - or the function  $b_0$  is strictly positive on a subdomain.

There are other combinations of conditions to arrive at a strictly positive functional  $F$ , but the above two are easiest to verify.

With the above assumptions we know that the BVP (6.9) has exactly one solution. The proof of this result is left to mathematicians. As a rule of thumb use that the solution  $u$  is  $(k+2)$ -times differentiable if  $f$  is  $k$ -times differentiable.

This mathematical result tells us that there is a unique solution  $u_e$  of the boundary value problem, but it does not give the solution. Use the finite element method to find numerical approximations  $u_h$  to this exact solution  $u_e$ .

### 6.4.2 Function Spaces, Norms and Continuous Functionals

In view of the above definition of a weak solution define for functions  $u$  and  $v$

$$\begin{aligned}\langle u, v \rangle &:= \iint_{\Omega} u v \, dA \\ A(u, v) &:= \iint_{\Omega} a \nabla u \cdot \nabla v + b_0 u v \, dA = \langle a \nabla u, \nabla v \rangle + \langle b_0 u, v \rangle \\ \langle u, v \rangle_{\Gamma_2} &:= \int_{\Gamma_2} u v \, ds.\end{aligned}$$

Basic properties of the integral imply that the bilinear form  $A$  is symmetric and linear with respect to each argument, i.e. for  $\lambda_i \in \mathbb{R}$  use

$$\begin{aligned}A(u, v) &= A(v, u) \\ A(\lambda_1 u_1 + \lambda_2 u_2, v) &= \lambda_1 A(u_1, v) + \lambda_2 A(u_2, v) \\ A(u, \lambda_1 v_1 + \lambda_2 v_2) &= \lambda_1 A(u, v_1) + \lambda_2 A(u, v_2).\end{aligned}$$

The function  $u$  is a **weak solution** of (6.9) iff

$$A(u, \phi) = \langle f, \phi \rangle + \langle g_2, \phi \rangle_{\Gamma_2} \quad \text{for all functions } \phi \quad .$$

We can also search for a minimum of the functional

$$F(u) = \frac{1}{2} A(u, u) - \langle f, u \rangle - \langle g_2, u \rangle_{\Gamma_2}.$$

The only new aspect is notation. For the subsequent observations it is convenient to introduce two spaces of functions.

**6–3 Definition :** Let  $u$  be a piecewise differentiable function defined on the domain  $\Omega \subset \mathbb{R}^2$ . Then  $L_2$  and  $V$  denote two sets of functions, both spaces equipped with a norm<sup>8</sup>

$$\begin{aligned} L_2 &:= \{u : \Omega \rightarrow \mathbb{R} \mid u \text{ is square integrable}\} \\ \|u\|_2^2 &:= \langle u, u \rangle = \iint_{\Omega} u^2 \, dA. \end{aligned}$$

For the function  $u$  to be in the smaller subspace  $V$  we require the function  $u$  and its derivatives  $\nabla u$  to be square integrable and  $u$  has to satisfy the Dirichlet boundary condition (if there are any imposed). The norm in this space is given by

$$\|u\|_V^2 := \|\nabla u\|_2^2 + \|u\|_2^2 = \iint_{\Omega} |\nabla u|^2 + u^2 \, dA = \iint_{\Omega} \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial u}{\partial y} \right)^2 + u^2 \, dA.$$

$L_2$  and  $V$  are vector spaces and  $\langle \cdot, \cdot \rangle$  is a scalar product on  $L_2$ .  $V$  is called a Sobolev space. Obviously we have

$$V \subset L_2 \quad \text{and} \quad \|u\|_2 \leq \|u\|_V.$$

Since the ‘energy’ to be minimized

$$F(u) = \frac{1}{2} A(u, u) = \frac{1}{2} \iint_{\Omega} a (\nabla u)^2 + b u^2 \, dA$$

is closely related to  $\|u\|_V^2$  this norm is often called an **energy norm**.

If  $u, v \in V$  the expression  $A(u, v)$  can be computed and find

$$\begin{aligned} |A(u, v)| &= \left| \iint_{\Omega} a \nabla u \cdot \nabla v + b_0 u v \, dA \right| \leq \iint_{\Omega} |a| |\nabla u| |\nabla v| + |b_0| |u| |v| \, dA \\ &\leq \alpha_1 \iint_{\Omega} |\nabla u| |\nabla v| \, dA + \beta_1 \iint_{\Omega} |u| |v| \, dA \leq \alpha_1 \|\nabla u\|_2 \|\nabla v\|_2 + \beta_1 \|u\|_2 \|v\|_2 \\ &\leq (\alpha_1 + \beta_1) \|u\|_V \|v\|_V. \end{aligned}$$

Assuming  $0 < \beta_0 \leq b_0(x)$  for all  $x$  leads to

$$\begin{aligned} A(u, u) &= \iint_{\Omega} a \nabla u \cdot \nabla u + b_0 u u \, dA = \iint_{\Omega} a |\nabla u|^2 + b_0 |u|^2 \, dA \\ &\geq \iint_{\Omega} \alpha_0 |\nabla u|^2 + \beta_0 |u|^2 \, dA \geq \min\{\alpha_0, \beta_0\} \iint_{\Omega} |\nabla u|^2 + |u|^2 \, dA = \gamma_0 \|u\|_V^2. \end{aligned}$$

---

<sup>8</sup>A mathematically correct introduction of these function spaces is well beyond the scope of these notes. The tools of Lebesgue integration and completion of spaces is not available. As a consequence we ignore most of the mathematical problems.

It can be shown<sup>9</sup> that the above inequality is correct as long as the assumptions in Section 6.4.1 are satisfied. Thus find

$$\gamma_0 \|u\|_V^2 \leq A(u, u) \leq (\alpha_1 + \beta_1) \|u\|_V^2 \quad \text{for all } u \in V. \quad (6.10)$$

This inequality is the starting point for most theoretical results on boundary value problems of the type examined in these notes. The bilinear form  $A(\cdot, \cdot)$  is called coercive or elliptic. For the purposes of these notes it is sufficient to realize that the expression  $A(u, u)$  corresponds to the squared integral of the function  $u$  and its partial derivatives of order 1.

### 6.4.3 Convergence of the Finite Dimensional Approximations

The space  $V$  contains all piecewise differentiable, continuous functions and thus  $V$  is not a finite dimensional vectors space. For a fixed parameter  $h > 0$  we choose a discretisation of the bounded domain  $\Omega \subset \mathbb{R}^2$  in finite many triangles of typical diameter  $h$ . Then we consider only continuous functions that are polynomials of a given degree (e.g. 1 or 2) on each of the triangles, i.e. a piecewise linear or quadratic function. This leads to a finite dimensional subspace  $V_h$ , i.e. finitely many degrees of freedom.

$$V_h \subset V \quad \text{finite dimensional subspace}$$

The functions in the finite dimensional subspace  $V_h$  have to be piecewise differentiable and everywhere continuous. This condition is necessary, since we try to minimize a functional involving first order partial derivatives. This property is called **conforming elements**. Instead of searching for a minimum on all of  $V$  we now only consider functions in  $V_h \subset V$  to find the minimizer of the functional. This is illustrated in Table 6.2. We hope that the minimum  $u_h \in V_h$  will be close to the exact solution  $u_e \in V$ . The main goal of this section is to show that this is in fact the case. The ideas of proofs are adapted from [John87, p.54] and [Davi80, §7] and can also be used in more general situations, e.g. for differential equations with more independent variables. To simplify the proof of the abstract error estimate we use two lemmas.

	exact problem	approximate problem
functional to minimize amongst functions	$F(u) = \frac{1}{2} A(u, u) - \langle f, u \rangle - \langle g_2, u \rangle_{\Gamma_2}$ $u \in V$ (infinite dimensional)	$F(u_h) = \frac{1}{2} A(u_h, u_h) - \langle f, u_h \rangle - \langle g_2, u_h \rangle_{\Gamma_2}$ $u_h \in V_h$ (finite dimensional)
necessary condition for minimum	$A(u, \phi) - \langle f, \phi \rangle - \langle g_2, \phi \rangle_{\Gamma_2} = 0$ for all $\phi \in V$	$A(u_h, \phi_h) - \langle f, \phi_h \rangle - \langle g_2, \phi_h \rangle_{\Gamma_2} = 0$ for all $\phi_h \in V_h$
main goal	$u_h \rightarrow u$ as $h \rightarrow 0$	

Table 6.2: Minimization of exact and approximate problem

---

<sup>9</sup>The correct mathematical result to be used is Poincaré's inequality. There exists a constant  $C$  (depending on  $\Omega$  only) such that for any smooth function  $u$  vanishing on the boundary we have

$$\iint_{\Omega} u^2 \, dA \leq C \iint_{\Omega} |\nabla u|^2 \, dA.$$

This inequality replaces the condition  $0 < \beta_0 \leq b$ . Intuitively the inequality shows that the values of the function are controlled by the values of the derivatives. For elasticity problems Korn's inequality will play the same role.

**6-4 Lemma :** If  $u_h$  is a minimizer of the functional  $F$  on  $V_h$ , i.e.

$$F(u_h) = \frac{1}{2} A(u_h, u_h) - \langle f, u_h \rangle - \langle g_2, u_h \rangle_{\Gamma_2} \leq \frac{1}{2} A(v_h, v_h) - \langle f, v_h \rangle - \langle g_2, v_h \rangle_{\Gamma_2} \quad \text{for all } v_h \in V_h$$

then

$$A(u_h, \phi_h) - \langle f, \phi_h \rangle - \langle g_2, \phi_h \rangle_{\Gamma_2} = 0 \quad \text{for all } \phi_h \in V_h.$$

◇

**Proof :** Use Figure 6.13 to visualize the proof. We examine the function along one straight line, i.e. the derivative of

$$\begin{aligned} g(t) &= F(u_h + t \phi_h) = \frac{1}{2} A(u_h + t \phi_h, u_h + t \phi_h) - \langle f, u_h + t \phi_h \rangle - \langle g_2, u_h + t \phi_h \rangle_{\Gamma_2} \\ &= \frac{1}{2} A(u_h, u_h) + \langle g, u_h \rangle + t (A(u_h, \phi) - \langle f, \phi \rangle - \langle g_2, \phi \rangle_{\Gamma_2}) + t^2 \frac{1}{2} A(\phi_h, \phi_h) \end{aligned}$$

has to vanish at  $t = 0$  for all  $\phi_h \in V_h$ . Since the above expression is of the form  $g(t) = c_0 + c_1 t + c_2 t^2$  and  $\frac{d}{dt} g(0) = c_1$  we find

$$A(u_h, \phi) - \langle f, \phi \rangle - \langle g_2, \phi \rangle_{\Gamma_2} = 0.$$

This implies the desired result. □

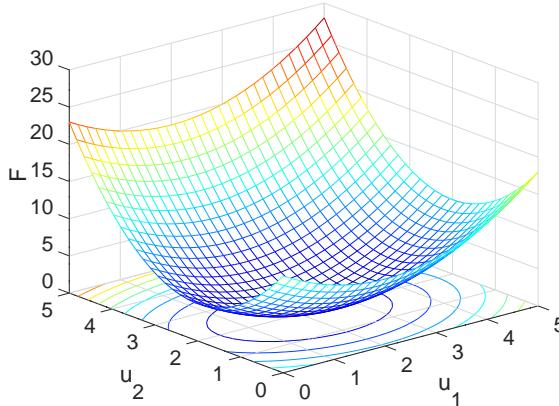


Figure 6.13: A function to be minimized

**6-5 Lemma :** Let  $u_e \in V$  be the minimizer of the functional  $F$  on all of  $V$  and let  $u_h \in V_h$  be the minimizer of

$$F(\psi_h) = \frac{1}{2} A(\psi_h, \psi_h) - \langle f, \psi_h \rangle - \langle g_2, \psi_h \rangle_{\Gamma_2}$$

amongst all  $\psi_h \in V_h$ . This implies that  $u_h \in V_h$  is also the minimizer of

$$G(\psi_h) = A(u_e - \psi_h, u_e - \psi_h)$$

amongst all  $\psi_h \in V_h$ . □

**Proof :** If  $u_h \in V_h$  minimizes  $F$  in  $V_h$  and  $u_e \in V$  minimizes  $F$  in  $V$  then the previous lemma implies

$$\begin{aligned} A(u_e, \phi_h) &= \langle f, \phi_h \rangle + \langle g_2, \phi_h \rangle_{\Gamma_2} \quad \text{for all } \phi_h \in V_h \\ A(u_h, \phi_h) &= \langle f, \phi_h \rangle + \langle g_2, \phi_h \rangle_{\Gamma_2} \quad \text{for all } \phi_h \in V_h \end{aligned}$$

and thus  $A(u_e - u_h, \phi_h) = 0$ . This leads to

$$\begin{aligned} G(u_h + \phi_h) &= A(u_e - u_h - \phi_h, u_e - u_h - \phi_h) \\ &= A(u_e - u_h, u_0 - u_h) - 2 A(u_e - u_h, \phi_h) + A(\phi_h, \phi_h) \\ &= A(u_e - u_h, u_e - u_h) + A(\phi_h, \phi_h) \\ &\geq A(u_e - u_h, u_e - u_h). \end{aligned}$$

Equality occurs only if  $\phi_h = 0$ . Thus  $\phi_h = 0 \in V_h$  is the unique minimizer of the above function and the result is established.  $\square$

**6–6 Theorem :** (Abstract error estimate, Lemma of Céa)

If  $u_e$  is the minimizer of the functional

$$F(u) = \frac{1}{2} A(u, u) - \langle f, u \rangle - \langle g_2, u \rangle_{\Gamma_2}$$

amongst all  $u \in V$  and  $u_h \in V_h$  is the minimizer of  $F$  amongst all  $u_h$  in the subspace  $V_h \subset V$ , then the distance of  $u_e$  and  $u_h$  (in the  $V$ -norm) can be estimated. There exists a positive constant  $k$  such that

$$\|u_e - u_h\|_V \leq k \min_{\psi_h \in V_h} \|u_e - \psi_h\|_V.$$

The constant  $k$  is independent on  $h$ .  $\diamond$

It assumes that the integrations are carried out without error. Since we will use a Gauss integration this is not far from the truth.

As a consequence of Céa's Lemma we have to be able to approximate an the exact solution  $u_e \in V$  by approximate function  $\psi_h \in V_h$  and the error of the finite element solution  $u_h \in V_h$  is smaller than the approximation error, except for the factor  $k$ . Thus the Lemma of Céa reduces the question of estimating the error of the approximate solution to a question of estimating the approximation error for a given function (the exact solution) in the energy norm. Standard interpolation results allow to estimate the error of the approximation, assuming some regularity on the exact solution  $u$ .

**Proof :** Use the inequality (6.10) and the above lemma to conclude that

$$\begin{aligned} \gamma_0 \|u_e - u_h\|_V^2 &\leq A(u_e - u_h, u_0 - u_h) \leq A(u_e - u_h - \phi_h, u_e - u_h - \phi_h) \\ &\leq (\alpha_1 + \beta_1) \|u_e - u_h - \phi_h\|_V^2 \quad \text{for all } \phi_h \in V_h \end{aligned}$$

and thus

$$\|u_e - u_h\|_V \leq \sqrt{\frac{\alpha_1 + \beta_1}{\gamma_0}} \|u_e - u_h - \phi_h\|_V \quad \text{for all } \phi_h \in V_h.$$

As  $\phi_h \in V_h$  is arbitrary find the claimed result.  $\square$

**6–7 Observation :** Connecting FEM and finite difference method

When working with the method of finite differences in Chapter 4 the Lax Equivalence Theorem 4–2 (page 261) is used to conclude that consistency and stability imply convergence. There is a similar path for the Finite Element Method:

- The concept of consistency of a finite difference approximation is replaced by approximation results, based on piecewise interpolation. By general approximation methods one shows that “any” function can be approximated by piecewise linear functions (Result 6–8) or piecewise quadratic functions (Result 6–10). These results lead to the order of convergence, i.e. the approximation error is proportional to  $h^p$  for some order  $p$ , where  $h$  is the typical size of an element.

- The concept of stability of a finite difference approach is replaced by the coercive bilinear form  $A(u, u)$ , e.g. inequality (6.10). These rather mathematical results might be difficult to prove. For second order boundary value problems of the type (6.9) Poincaré's inequality has to be used. For elasticity problems the tool to be used is Korn's inequality.
- The abstract error estimate in Theorem 6–6 then implies that the FEM approximation  $u_h$  converges to the true solution  $u_0$ , in some very specific sense. Thus Theorem 6–6 replaces the Lax Equivalence Theorem 4–2.

Example 6–1 (page 413) illustrates that in special cases a finite element approximation is equivalent to a finite difference approximation.  $\diamond$

#### 6.4.4 Approximation by Piecewise Linear Interpolation

Let  $\Omega \subset \mathbb{R}^2$  be a bounded polygonal domain, divided into triangles of typical diameter  $h$ . No corner of a triangle can be placed in the middle of a triangle side. In addition we require a **minimal angle condition**: no angle is smaller than a given minimal angle. Equivalently require that the ratio of the triangle diameter  $h$  and the radius of the inscribed circle be smaller than a given fixed value. This is often mentioned as the **quality of the mesh**. A typical triangulation is shown in Figure 6.3 on page 409.

Then compute the value of the function  $u(x, y)$  at each corner of the triangles. Within each triangle the function is replaced by a linear function. Thus an interpolating function  $\Pi_h u$  is constructed. The operator  $\Pi_h$  can be considered a **projection operator** of  $V$  onto the finite dimensional subspace  $V_h$ .

$$\Pi_h : V \longrightarrow V_h \quad , \quad u \mapsto \Pi_h u$$

For two neighboring triangles the interpolated functions will coincide along the common edge, since the linear functions coincide at the two corners. Thus the interpolated function is continuous on the domain and we have **conforming elements**. The interpolated function  $\Pi_h u$  and the original function  $u$  coincide if  $u$  happens to be a linear function. By considering a Taylor expansion one can verify<sup>10</sup> that the typical approximation error of the function is of the order  $ch^2$ , where the constant  $c$  depends on higher order derivatives of  $u$ . The error of the gradient is of order  $h$ .

##### 6–8 Result : (Piecewise linear interpolation)

Assume that a function  $u$  is at least twice differentiable on the domain  $\Omega$  and use the **piecewise linear interpolation**  $\Pi_h u$ . Approximation theory implies that there is a constant  $M$  (depending on second order derivatives of  $u$ ), such that

$$\begin{aligned} |u(\vec{x}) - \Pi_h u(\vec{x})| &\leq M h^2 \quad \text{for all } \vec{x} \in \Omega \\ |\nabla u(\vec{x}) - \nabla(\Pi_h u(\vec{x}))| &\leq M h \quad \text{for all } \vec{x} \in \Omega. \end{aligned}$$

Thus an integration implies that there is a constant  $c$  such that

$$\|u - \Pi_h u\|_V \leq c h \|u\|_2 \quad \text{and} \quad \|u - \Pi_h u\|_2 \leq c h^2 \|u\|_2$$

where

$$\|u\|_2^2 = \iint_{\Omega} \left| \frac{\partial^2 u}{\partial x^2} \right|^2 + \left| \frac{\partial^2 u}{\partial x \partial y} \right|^2 + \left| \frac{\partial^2 u}{\partial y^2} \right|^2 dA.$$

The constant  $c$  does not depend on  $h$  and the function  $u$ , as long as a minimal angle condition is satisfied.  $\diamond$

<sup>10</sup>Use the fact that the quadratic terms in the Taylor expansion lead to an approximation error. For an error vanishing at the nodes at  $x = 0$  and  $h$  we use a function  $f(x) = a \cdot x \cdot (h-x)$  with derivatives  $f'(x) = a(h-2x)$  and  $f''(x) = -2a$ . Since the maximal value of  $a \cdot h^2/4$  is attained at  $h/2$  we find  $|f(x)| \leq \frac{ah^2}{4} = \frac{h^2}{8} \max |f''|$  and  $|f'(x)| \leq \frac{ah}{2} \max |f''|$  for all  $0 \leq x \leq h$ .

An exact statement and proof of this result is given in [Brae02, §II.6]. The result is based on the fundamental Bramble–Hilbert–Lemma.

Now we have all the ingredients to state and proof the basic convergence results for finite element solutions to boundary value problems in two variables. The exact solution  $u_e \in V$  to be approximated is the minimizer of the functional

$$F(u) = \iint_{\Omega} \frac{1}{2} a(\nabla u)^2 + \frac{1}{2} b_0 u^2 - f u \, dA - \int_{\Gamma_2} g_2 u \, ds$$

On a smooth domain  $\Omega \subset \mathbb{R}^2$  the exact solution  $u_e$  is smooth (often differentiable) if  $a$ ,  $b_0$ ,  $f$  and  $g_2$  are smooth. Instead of searching on the space  $V$  we restrict the search on the finite dimensional subspace  $V_h$  and arrive at the approximate minimizer  $u_h$ . Thus the error function  $e = u_h - u_e$  has to be as small as possible for the approximation to be of a good quality. In fact we hope for a convergence

$$u_h \rightarrow u_e \quad \text{as} \quad h \rightarrow 0$$

in some sense to be specified.

**6–9 Theorem :** Examine the boundary value problem (6.9) where the conditions are such that the unique solutions  $u$  and all its partial derivative up to order 2 are square integrable over the domain  $\Omega$ . If the subspace  $V_h$  is generated by the piecewise linear interpolation operator  $\Pi_h$  then we find

$$\|u_h - u_e\|_V \leq C h \quad \text{and} \quad \|u_h - u_e\|_2 \leq C_1 h^2$$

for some constants  $C$  and  $C_1$  independent on  $h$ .

We may say that

- $u_h$  converges to  $u_e$  with an error proportional to  $h^2$  as  $h \rightarrow 0$ .
- $\nabla u_h$  converges to  $\nabla u_e$  with an error proportional to  $h$  as  $h \rightarrow 0$ .



Observe that the above estimates are **not** point-wise estimates. It is the integrals of the solution and its derivatives that are controlled.

**Proof :** The interpolation result 6–8 and the abstract error estimate 6–6 imply immediately

not in class

$$\|u_h - u_e\|_V \leq k \min_{\phi_h \in V_h} \|\phi_h - u_e\|_V \leq k \|\Pi_h u_e - u_e\|_V \leq k c h .$$

This is the first of the desired estimates. It shows that the error of the function and its first order partial derivatives are approximately proportional to  $h$ . The second estimate states that the error of the function is proportional to  $h^2$ . This second estimate requires considerably more work. The method of proof is known as **Nitsche trick** and is due to Joachim Nitsche and Jean-Pierre Aubin. A good presentation is given in [StraFix73, §3.4] or [KnabAnge00, Satz 3.37]. For sake of completeness a similar presentation is shown below.

Use the notation  $e = u_h - u_e$  and equation (6.10) to conclude

$$A(e, e) \leq (\alpha_1 + \beta_1) \|e\|_V^2 \leq (\alpha_1 + \beta_1) k^2 c^2 h^2 .$$

Let  $w \in V$  be the minimizer of the functional

$$\frac{1}{2} A(w, w) + \langle e, w \rangle .$$

Thus  $w$  is a solution of the boundary value problem

$$\begin{aligned} -\nabla \cdot (a \nabla w) + b_0 w &= e && \text{for } (x, y) \in \Omega \\ w &= 0 && \text{for } (x, y) \in \Gamma_1 \\ \vec{n} \cdot (a \nabla w) &= 0 && \text{for } (x, y) \in \Gamma_2 \end{aligned}$$

Regularity theory now implies that the second order derivatives of  $w$  are bounded by the values of  $e$  (in the  $L_2$  sense) or more precisely

$$|w|_2 \leq c \|e\|_2 = c \|u_h - u_e\|_2.$$

The interpolation result 6–8 leads to

$$\|w - \Pi_h w\|_V \leq c_1 h |w|_2 \leq c_2 h \|u_h - u_e\|_2$$

Using Theorem 6–6 conclude

$$A(w, w) \leq A(w - \Pi_h w, w - \Pi_h w) \leq (\alpha_1 + \beta_1) \|w - \Pi_h w\|_V^2 \leq (\alpha_1 + \beta_1) c_2^2 h^2 \|u_h - u_e\|_2^2.$$

Since  $w$  is a minimizer of the functional find

$$A(w, \psi) + \langle e, \psi \rangle = 0 \quad \text{for all } \psi \in V.$$

By choosing  $\psi = e$  arrive at

$$-A(w, e) = \langle e, e \rangle = \|e\|_2^2 = \iint_{\Omega} |u_h - u_e|^2 dA.$$

Now use the Cauchy–Schwartz inequality to conclude that

$$\begin{aligned} \|e\|_2^2 &= \iint_{\Omega} |u_h - u_e|^2 dA = |A(w, e)| = \iint_{\Omega} a \nabla w \cdot \nabla e + b_0 w e dA \\ &\leq (A(w, w))^{1/2} \cdot (A(e, e))^{1/2} \leq \sqrt{\alpha_1 + \beta_1} c_2 h \|u_h - u_e\|_2 \cdot \sqrt{\alpha_1 + \beta_1} k c h. \end{aligned}$$

A division by  $\|e\|_2 = \|u_h - u_e\|_2$  leads to

$$\|u_h - u_e\|_2 \leq C_1 h^2.$$

This is the claimed second convergence estimate. □

#### 6.4.5 Approximation by Piecewise Quadratic Interpolation

We start with a triangulation similar to the piecewise linear interpolation, but we use a piecewise quadratic interpolation on each triangle, i.e. we examine an interpolating function in Figure 6.14. The six coefficient  $c_k$  can be used to assure that the interpolated function coincides with the given function on the corners and the mid points of triangle, as show in Figure 6.14. Along each edge we find a quadratic function, uniquely determined by the values at the three points. Thus the interpolating functions from neighboring triangles will coincide on all points on the common edge. Thus we find again **conforming elements**.

Thus we arrive again at an **projection operator**  $\Pi_h$  from  $V$  onto the finite dimensional subspace  $V_h$ . We have

$$\Pi_h : V \longrightarrow V_h \quad , \quad u \mapsto \Pi_h u.$$

The resulting functions  $\Pi_h u$  are continuous on the domain and on each triangle we have a quadratic function. The interpolated function  $\Pi_h u$  and the original function  $u$  coincide if  $u$  happens to be a quadratic function.

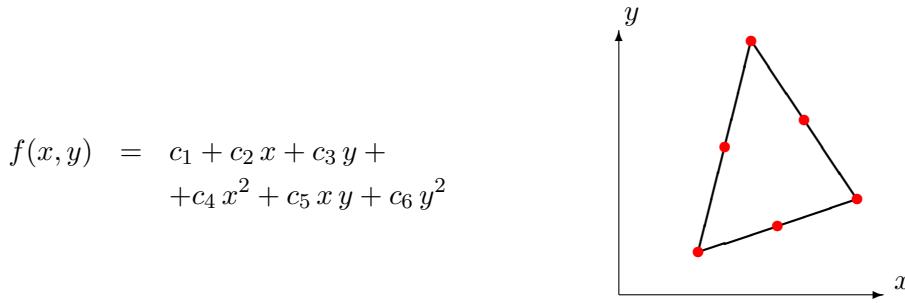


Figure 6.14: Quadratic interpolation on a triangle

By considering a Taylor expansion one can verify<sup>11</sup> that the typical approximation error of the function is of the order  $c h^3$  where the constant  $c$  depends on higher order derivatives of  $u$ . The error of the gradient is of order  $h^2$ .

#### 6–10 Result : (Piecewise quadratic interpolation)

Assume that a function  $u$  is at least three times differentiable on the domain  $\Omega$  and use the **piecewise quadratic interpolation**  $\Pi_h u$ . Approximation theory implies that there is a constant  $M$  (depending on third order derivatives of  $u$ ), such that

$$\begin{aligned} |u(\vec{x}) - \Pi_h u(\vec{x})| &\leq M h^3 \quad \text{for all } \vec{x} \in \Omega \\ |\nabla u(\vec{x}) - \nabla(\Pi_h u(\vec{x}))| &\leq M h^2 \quad \text{for all } \vec{x} \in \Omega. \end{aligned}$$

Thus an integration implies that there is a constant  $c$  such that

$$\|u - \Pi_h u\|_V \leq c h^2 |u|_3 \quad \text{and} \quad \|u - \Pi_h u\|_2 \leq c h^3 |u|_3$$

where  $|u|_3^2$  is the sum of all squared and integrated partial derivatives of order 3. The constant  $c$  does not depend on  $h$  and the function  $u$ , as long as a minimal angle condition is satisfied.  $\diamond$

An exact statement and proof of this result is given in [Brae02, §II.6]. The result is based on the fundamental Bramble–Hilbert–Lemma. Based on this interpolation estimate we can again formulate the basic convergence result for a finite element approximation using piecewise quadratic approximations.

**6–11 Theorem :** Examine the boundary value problem (6.9) where the conditions are such that the unique solutions  $u$  and all its partial derivative up to order 3 are square integrable over the domain  $\Omega$ . If the subspace  $V_h$  is generated by the piecewise quadratic interpolation operator  $\Pi_h$  then we find

$$\|u_h - u_e\|_V \leq C h^2 \quad \text{and} \quad \|u_h - u_e\|_2 \leq C_1 h^3$$

for some constants  $C$  and  $C_1$  independent on  $h$ .

We may say that

- $u_h$  converges to  $u_e$  with an error proportional to  $h^3$  as  $h \rightarrow 0$ .
- $\nabla u_h$  converges to  $\nabla u_e$  with an error proportional to  $h^2$  as  $h \rightarrow 0$ .

$\diamond$

<sup>11</sup>Use the fact the cubic terms in the Taylor expansion lead to an approximation error. For an error vanishing at the nodes at  $x = 0$  and  $\pm h$  we use a function  $f(x) = a \cdot x \cdot (h^2 - x^2)$  with derivatives  $f'(x) = a(h^2 - 3x^2)$ ,  $f''(x) = -a6x$  and  $f'''(x) = -a6$ . The maximal value  $\frac{a2h^3}{3\sqrt{3}}$  of the function is attained at  $\pm h/\sqrt{3}$  we find  $|f(x)| \leq c h^3 \max |f'''|$  and  $|f'(x)| \leq c h^2 \max |f''|$  for all  $-h \leq x \leq h$ .

**Proof:** The interpolation result 6–10 and the abstract error estimate 6–6 imply immediately

$$\|u_h - u_e\|_V \leq k \min_{\phi_h \in V_h} \|\psi_h - u_e\|_V \leq k \|\Pi_h u_e - u_e\|_V \leq k c h^2.$$

which is already the first of the desired estimates. The second estimate has to be verified with the Aubin–Nitsche method, as in the proof of Theorem 6–9.  $\square$

Observe that the convergence with quadratic interpolation (Theorem 6–11) is improved by a factor of  $h$  compared to the linear interpolation, i.e. Theorem 6–9. Thus one might be tempted to increase the order of the approximating polynomials further and further. But there are also reasons that speak against such a process:

- Carrying out the interpolation will get more and more difficult. In particular the continuity across the edges of the triangles is not easily obtained. It is more difficult to construct higher order **conforming elements**.
- For higher order approximations to be effective we need bounds on higher order derivatives of the exact solution  $u_e$ . This might be difficult or impossible to achieve. If the domain is a polygon, there will be corners and smoothness of the solution is far from obvious. Some of the coefficient functions in the BVP (6.9) might not be smooth, e.g. by two different materials used in the domain. Thus we might not benefit from a higher order convergence with higher order elements.

In the interior of the domain  $\Omega$  smoothness of the exact solution  $u_e$  is often true and with higher order approximations we get a faster convergence. Thus piecewise approximations of orders 1, 2 and 3 are regularly used. In the next section a detailed construction of second order elements is presented.

Presentations rather similar to the above can be found in many books on FEM. As example consult [Brae02] for a proof of energy estimates and also for error estimates in the  $L_\infty$  norm, i.e. point wise estimates. In [AxelBark84] find Céa's lemma and regularity results for non-symmetric problems.

## 6.5 Construction of Triangular Second Order Elements

In this section we construct a second order FEM approximation to the solution of the BVP shown as equation (6.8) on page 417.

$$\begin{aligned} -\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u &= f && \text{for } (x, y) \in \Omega \\ u &= g_1 && \text{for } (x, y) \in \Gamma_1 \\ \vec{n} \cdot (a \nabla u + u \vec{b}) &= g_2 + g_3 u && \text{for } (x, y) \in \Gamma_2 \end{aligned}$$

The function  $u$  is a weak solution of the above BVP if it satisfies the boundary condition  $u = g_1$  on  $\Gamma_1$  and for all test functions  $\phi$  (vanishing in  $\Gamma_1$ ) we have the integral condition

$$0 = \iint_{\Omega} \nabla \phi \cdot (a \nabla u + u \vec{b}) + \phi (b_0 u - f) \, dA - \int_{\Gamma_2} \phi (g_2 + g_3 u) \, ds.$$

The domain  $\Omega \subset \mathbb{R}^2$  is triangulated and the values of the function  $u$  at the corners and the midpoints of the edges of the triangles are considered as degrees of freedom of the system. This leads to a vector  $\vec{u}$  to be determined. We have to find a discretized version of the integrals in the above functions and determine the **global stiffness matrix  $A$**  such that the above integral condition translates to

$$0 = \langle \mathbf{A} \vec{u}, \vec{\phi} \rangle + \langle \mathbf{W} \vec{f}, \vec{\phi} \rangle.$$

Then the discretized solution is given as solution of the linear systems of equations

$$\mathbf{A} \vec{u} + \mathbf{W} \vec{f} = \vec{0}.$$

All computations should be formulated with matrices, such that an implementation with *Octave/MATLAB* will be easy.

The order of presentation is as follows:

- Integration over a general triangle, using Gauss integration.
- Examine the basis functions for a second order element on the standard triangle.
- Integrate a function using the values at the corners and midpoints.
- Show all the integrals to be computed.
- Integration of  $f \phi$ .
- Integration of  $b_0 u \phi$ .
- Transformation of the gradient from standard triangle to general triangle.
- Integration of  $u \vec{b} \nabla \phi$ .
- Integration of  $a \nabla u \nabla \phi$ .

### 6.5.1 Integration over a General Triangle

#### Transformation of coordinates

All of the necessary integrals for the FEM method are integrals over general triangles  $E$ . These can be written as images of a standard triangle in a  $(\xi, \nu)$ -plane, according to Figure 6.15. The transformation is given by the linear (resp. affine) mapping

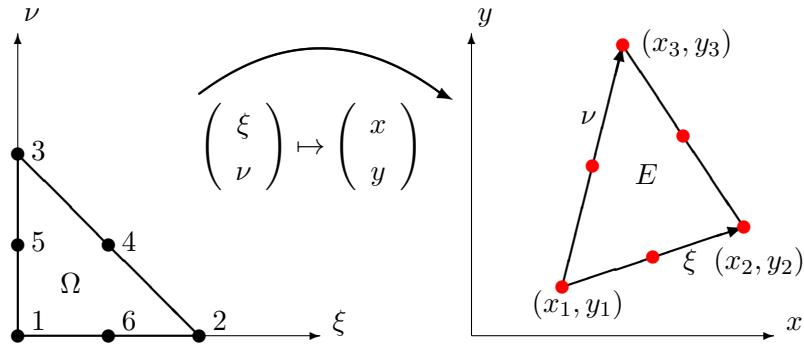


Figure 6.15: Transformation of standard triangle to general triangle

$$\begin{aligned} \begin{pmatrix} x \\ y \end{pmatrix} &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \xi \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \nu \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix} \\ &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix} \end{aligned}$$

where

$$\mathbf{T} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} = \frac{\partial(x, y)}{\partial(\xi, \nu)}.$$

If the coordinates  $(x, y)$  are given find the values of  $(\xi, \nu)$  with the help of

$$\begin{pmatrix} \xi \\ \nu \end{pmatrix} = \mathbf{T}^{-1} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}.$$

### Integration over the standard triangle and Gauss integration

If a function  $f(x, y)$  is to be integrated over the triangle  $E$  use the transformation identity

$$\iint_E f \, dA = \iint_{\Omega} f(\vec{x}(\xi, \nu)) \left| \det \left( \frac{\partial(x, y)}{\partial(\xi, \nu)} \right) \right| \, d\xi \, d\nu = |\det \mathbf{T}| \int_0^1 \left( \int_0^{\nu} f(\vec{x}(\xi, \nu)) \, d\xi \right) d\nu. \quad (6.11)$$

The Jaccobi determinant is given by

$$\left| \det \left( \frac{\partial(x, y)}{\partial(u, v)} \right) \right| = |\det \mathbf{T}| = |(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)|.$$

Since the area of the standard triangle  $\Omega$  is  $\frac{1}{2}$  find the area of  $E$  is given by  $\frac{1}{2} |\det \mathbf{T}|$ . For a numerical integration over the standard triangle  $\Omega$  choose some integration points  $\vec{g}_j \in \Omega$  and the corresponding weights  $w_j$  for  $j = 1, 2, \dots, m$  and then work with

$$\iint_{\Omega} f(\vec{\xi}) \, dA \approx \sum_{j=1}^m w_j f(\vec{g}_j). \quad (6.12)$$

The integration points and weights have to be chosen, such that the integration error is as small as possible. There are many integration schemes available. One of the early papers is [Cowp73] and a list is shown in [Hugh87, Table 3.1.1, p. 173] and a short list in [TongRoss08]<sup>12</sup>, [Gmur00, Tableau D3, page 233], [Zien13, Table 6.3, p. 181] or [Li21, Table 5.5, page 199].

As a concrete and useful example use the points  $g_1 = \frac{1}{2}(\lambda_1, \lambda_1)$  and  $g_4 = \frac{1}{2}(\lambda_2, \lambda_2)$  along the diagonal  $\xi = \nu$ . Similarly use two more points along each connecting straight line from a corner of the triangle to the midpoint of the opposing edge. This leads to a total of 6 integration points where groups of 3 have the same weight, i.e.  $w_1 = w_2 = w_3$  and  $w_4 = w_5 = w_6$ . Finally add the midpoint with weight  $w_7$ . This is illustrated in Figure 6.16. The nodes are shown in red and the integration points in blue. This choice satisfies two essential conditions:

- If a sample point is used in a Gauss integration, then all other points obtainable by permuting the three corners of the triangle must appear and with identical weight.
- All sample points must be inside the triangle (or on the triangle boundary) and all weights must be positive.

Then one arrives at a  $7 \times 2$  matrix  $\mathbf{G}$  (see equation (6.13)) containing in each row the coordinates of one integration point  $\vec{g}_j$  and a vector  $\vec{w}$  with the corresponding integration weights.

<sup>12</sup>The results are known as Hammer's formula. There is a typing error in [TongRoss08, Table 6.2, page 190]. For the integration scheme using four points the coefficient for the central point is negative, i.e.  $-0.5625$ . Thus the scheme should not be used for FEM codes, since there is a danger that the stiffness matrix will not be positive definite. For integration purposes the scheme does just fine.

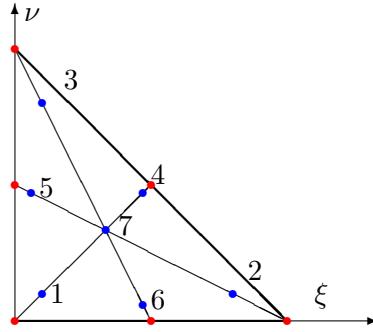


Figure 6.16: Gauss integration of order 5 on the standard triangle, using 7 integration points

To determine the optimal values setup and solve a system of five nonlinear equations for the unknowns  $\lambda_1, \lambda_2, w_1, w_4$  and  $w_7$ . The integration is approximated by

$$\iint_{\Omega} f(\vec{x}_i) dA \approx w_1 \left( f(\vec{\xi}_1) + f(\vec{\xi}_2) + f(\vec{\xi}_3) \right) + w_4 \left( f(\vec{\xi}_4) + f(\vec{\xi}_5) + f(\vec{\xi}_6) \right) + w_7 f(\vec{\xi}_7)$$

We require that  $\xi^k$  for  $0 \leq k \leq 5$  be integrated exactly. This generates five equations to be solved<sup>13</sup>. Due to the symmetric arrangement of the integration points this implies that all polynomial up to degree 5 are integrated exactly. The equations to be solved are generated by elementary computations.

$$\begin{aligned} \iint_{\Omega} 1 dA &= \frac{1}{2} = 3w_1 + 3w_4 + w_7 \\ \iint_{\Omega} \xi dA &= \frac{1}{6} = w_1 \left( 2 \frac{\lambda_1}{2} + 1 - \lambda_1 \right) + w_4 \left( 2 \frac{\lambda_2}{2} + 1 - \lambda_2 \right) + w_7 \frac{1}{3} \\ &= w_1 1 + w_4 1 + w_7 \frac{1}{3} \quad \text{equivalent to the above condition} \\ \iint_{\Omega} \xi^2 dA &= \frac{1}{12} = w_1 \left( 1 - 2\lambda_1 + \frac{3}{2}\lambda_1^2 \right) + w_4 \left( 1 - 2\lambda_2 + \frac{3}{2}\lambda_2^2 \right) + w_7 \frac{1}{9} \\ \iint_{\Omega} \xi^3 dA &= \frac{1}{20} = w_1 \left( 1 - 3\lambda_1 + 3\lambda_1^2 - \frac{3}{4}\lambda_1^3 \right) + w_4 \left( 1 - 3\lambda_2 + 3\lambda_2^2 - \frac{3}{4}\lambda_2^3 \right) + w_7 \frac{1}{27} \\ \iint_{\Omega} \xi^4 dA &= \frac{1}{30} = w_1 \left( 1 - 4\lambda_1 + 6\lambda_1^2 - 4\lambda_1^3 + \frac{9}{8}\lambda_1^4 \right) + w_4 \left( 1 - 4\lambda_2 + 6\lambda_2^2 - 4\lambda_2^3 + \frac{9}{8}\lambda_2^4 \right) + w_7 \frac{1}{81} \\ \iint_{\Omega} \xi^5 dA &= \frac{1}{42} = w_1 \left( 1 - 5\lambda_1 + 10\lambda_1^2 - 10\lambda_1^3 + 5\lambda_1^4 - \frac{15}{16}\lambda_1^5 \right) \\ &\quad + w_4 \left( 1 - 5\lambda_2 + 10\lambda_2^2 - 10\lambda_2^3 + 5\lambda_2^4 - \frac{15}{16}\lambda_2^5 \right) + \frac{1}{241} w_7 \end{aligned}$$

<sup>13</sup>As example we consider the case  $f(\xi) = \xi^2$  with some more details.

$$\begin{aligned} \iint_{\Omega} \xi^2 dA &= \int_0^1 \int_0^{1-\xi} \xi^2 d\nu d\xi = \int_0^1 (1-\xi) \xi^2 d\xi = \left( \frac{\xi^3}{3} - \frac{\xi^4}{4} \right) \Big|_{\xi=0}^1 = \frac{1}{12} \\ \sum_{j=1}^7 w_j f(\vec{g}_j) &= w_1 \left( \left( \frac{\lambda_1}{2} \right)^2 + (1-\lambda_1)^2 + \left( \frac{\lambda_1}{2} \right)^2 \right) + w_4 \left( \left( \frac{\lambda_2}{2} \right)^2 + (1-\lambda_2)^2 + \left( \frac{\lambda_2}{2} \right)^2 \right) + w_7 \left( \frac{1}{3} \right)^2 \\ &= w_1 \left( 1 - 2\lambda_1 + \frac{3}{2}\lambda_1^2 \right) + w_4 \left( 1 - 2\lambda_2 + \frac{3}{2}\lambda_2^2 \right) + w_7 \frac{1}{9} \end{aligned}$$

The above leads to a system of five nonlinear equations for the five unknowns  $\lambda_1$ ,  $\lambda_2$ ,  $w_1$ ,  $w_4$  and  $w_7$ . This can be solved by Octave/MATLAB or Mathematica<sup>14</sup>. There are multiple solutions possible, but you need a solution satisfying the following key properties:

- Pick a solution with  $0 < \lambda_1 < \lambda_2 < 1$ . This corresponds to the desirable result that all Gauss points are inside the triangle.
- Pick a solution with positive weights  $w_1$ ,  $w_4$  and  $w_7$ . This guarantees that the integral of a positive function is positive.

The equation resulting from the integral of  $\xi$  over the triangle is identical to the equation generated by the integral of 1 and thus not taken into account. Due to the symmetry of the Gauss points and the weights one can verify that all polynomials up to degree 5 are integrated exactly, e.g.  $\nu$ ,  $\nu^5$ ,  $\xi^2 \nu^3$ , ...

The solution is given by

$$\mathbf{G} = \begin{bmatrix} \lambda_1/2 & \lambda_1/2 \\ 1 - \lambda_1 & \lambda_1/2 \\ \lambda_1/2 & 1 - \lambda_1 \\ \lambda_2/2 & \lambda_2/2 \\ 1 - \lambda_2 & \lambda_2/2 \\ \lambda_2/2 & 1 - \lambda_2 \\ 1/3 & 1/3 \end{bmatrix} \approx \begin{bmatrix} 0.101287 & 0.101287 \\ 0.797427 & 0.101287 \\ 0.101287 & 0.797427 \\ 0.470142 & 0.470142 \\ 0.059716 & 0.470142 \\ 0.470142 & 0.059716 \\ 0.333333 & 0.333333 \end{bmatrix}, \quad \vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \end{pmatrix} \approx \begin{pmatrix} 0.0629696 \\ 0.0629696 \\ 0.0629696 \\ 0.0661971 \\ 0.0661971 \\ 0.0661971 \\ 0.1125000 \end{pmatrix}. \quad (6.13)$$

Using the transformation results in this section compute the coordinates  $\mathbf{X}_G$  for the Gauss integration points in a general triangle by

$$\mathbf{X}_G = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \mathbf{G}^T = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \mathbf{G}^T.$$

This approximate integration yields the exact results for polynomials up to degree 5. Thus for one triangle with diameter  $h$  and an area of the order  $h^2$  the integration error for smooth functions is of the order  $h^6 \cdot h^2 = h^8$ . When dividing a large domain in sub-triangles of size  $h$  this leads to a total integration error of the order  $h^6$ . For most problems this error will be considerably smaller than the approximation error of the FEM method and one can ignore this error contribution and we will from now on assume that the integrations yield exact results.

The above can be implemented in Octave or MATLAB.

### IntegrationTriangle.m

```
function res = IntegrationTriangle(corners, func)

la1 = (12 - 2*sqrt(15))/21;    la2 = (12 + 2*sqrt(15))/21;
w1  = (155 - sqrt(15))/2400;   w2 = (155 + sqrt(15))/2400;
w3  = 0.1125;                  w = [w1,w1,w1,w2,w2,w3];

G = [la1/2 la1/2; 1-la1,la1/2; la1/2, 1-la1;
      la2/2 la2/2; 1-la2,la2/2; la2/2, 1-la2; 1/3 1/3];

T = [corners(2,:) - corners(1,:); corners(3,:) - corners(1,:)];
```

<sup>14</sup>The exact values are  $\lambda_1 = (12 - 2\sqrt{15})/21$ ,  $\lambda_2 = (12 + 2\sqrt{15})/21$ ,  $w_1 = (155 - \sqrt{15})/2400$ ,  $w_4 = (155 + \sqrt{15})/2400$  and  $w_7 = 9/80$ .

```

if ischar(func)
    P = G*T; P(:,1) = P(:,1)+corners(1,1); P(:,2) = P(:,2)+corners(1,2);
    val = feval(func,P);
else
    val = func(:);
endif

res = w*val*abs(det(T));
end%function

```

and the above function can be tested by integrating the function  $x + y^2$  over a triangle.

### Octave

```

corners = [0 0; 1 0; 0 1];

function res = intFunc(xy)
    x = xy(:,1); y = xy(:,2);
    res = x+y.^2;
end%function

IntegrationTriangle(corners,'intFunc')

```

#### Observations:

- To save computation time some FEM algorithms use a simplified numerical integration. If the functions to be examined are close to constants over each triangle, then the error is acceptable.
- It is important to observe that the functions and the solutions are only evaluated at the integration points. This may lead to surprising (and wrong) results. Keywords: hourgassing and shear locking.
- The material properties are usually given by coefficient functions, e.g. for Young's modulus  $E$  and Poisson's ratio  $\nu$ . Thus these properties are evaluated at the Gauss points, but not at the nodes. This can lead to surprising extrapolation effects, e.g. a material constraint might not be satisfied at the nodes, but only at the integration points.

## 6.5.2 The Basis Functions for a Second Order Element

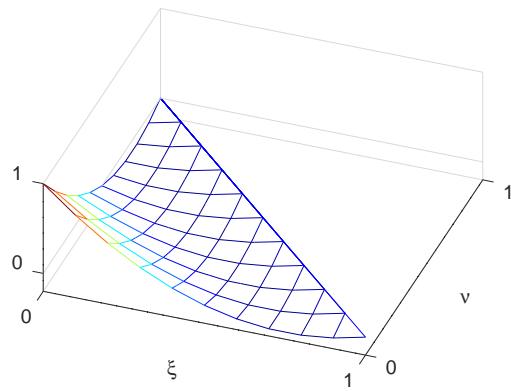
There are 6 linearly independent polynomials of degree 2 or less, namely 1,  $x$ ,  $y$ ,  $x^2$ ,  $y^2$  and  $x \cdot y$ . Examine the standard triangle  $\Omega$  in Figure 6.15 with the values of a function  $f(\xi, \nu)$  at the corners and at the midpoints of the edges. Use the numbering as shown in Figure 6.15. Now construct polynomials  $\phi_i(\xi, \nu)$  of degree 2, such that

$$\Phi_i(\xi_j, \nu_j) = \delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases},$$

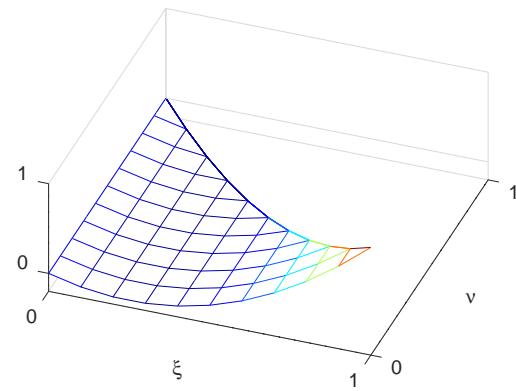
i.e. each basis function is equal to 1 at one of the nodes and vanishes on all other nodes. These basis polynomials are given by

$$\begin{aligned}
\Phi_1(\xi, \nu) &= (1 - \xi - \nu)(1 - 2\xi - 2\nu) \\
\Phi_2(\xi, \nu) &= \xi(2\xi - 1) \\
\Phi_3(\xi, \nu) &= \nu(2\nu - 1) \\
\Phi_4(\xi, \nu) &= 4\xi\nu \\
\Phi_5(\xi, \nu) &= 4\nu(1 - \xi - \nu) \\
\Phi_6(\xi, \nu) &= 4\xi(1 - \xi - \nu)
\end{aligned}$$

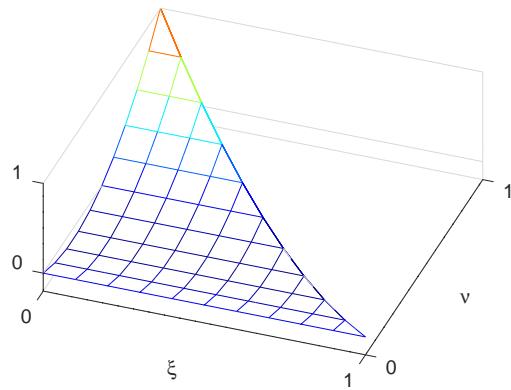
and find their graphs in Figure 6.17. Any quadratic polynomial  $f$  on the standard triangle  $\Omega$  can be written



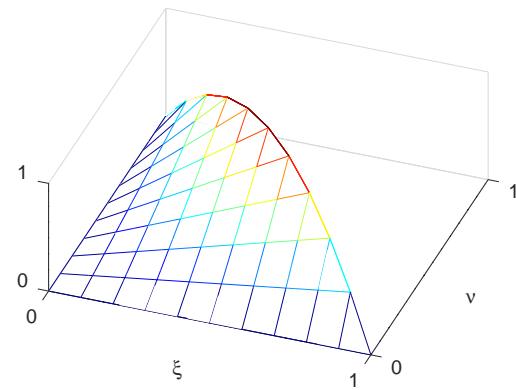
$$(a) \Phi_1(\xi, \nu) = (1 - \xi - \nu)(1 - 2\xi - 2\nu)$$



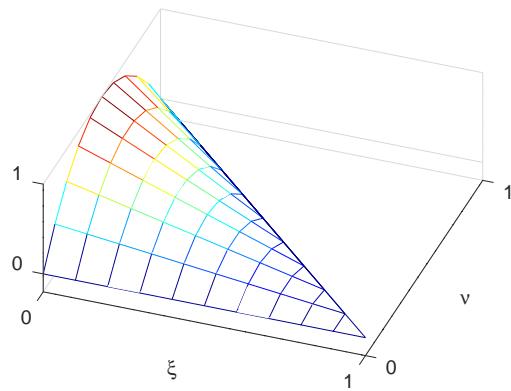
$$(b) \Phi_2(\xi, \nu) = \xi(2\xi - 1)$$



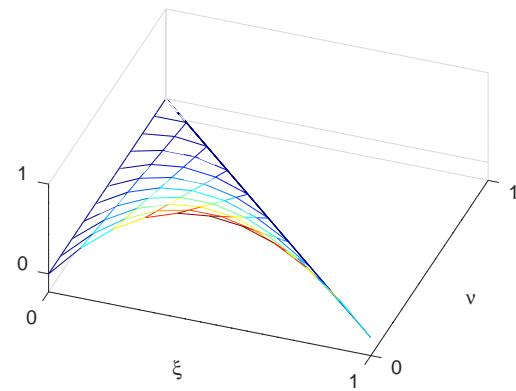
$$(c) \Phi_3(\xi, \nu) = \nu(2\nu - 1)$$



$$(d) \Phi_4(\xi, \nu) = 4\xi\nu$$



$$(e) \Phi_5(\xi, \nu) = 4\nu(1 - \xi - \nu)$$



$$(f) \Phi_6(\xi, \nu) = 4\xi(1 - \xi - \nu)$$

Figure 6.17: Basis functions for second order triangular elements

as linear combination of the basis functions by using

$$f(\xi, \nu) = \sum_{i=1}^6 f(x_i, y_i) \Phi_i(\xi, \nu) = \sum_{i=1}^6 f_i \Phi_i(\xi, \nu).$$

### 6.5.3 Integration of Functions Given at the Nodes

#### Integration of quadratic functions

Compute the values of the basis functions  $\Phi_i(\xi, \nu)$  at the Gauss points  $\vec{g}_j$  by  $m_{j,i} = \Phi_i(\vec{g}_j)$  to find

$$f(\vec{g}_j) = \sum_{i=1}^6 f_i \Phi_i(\vec{g}_j) = \sum_{i=1}^6 m_{j,i} f_i$$

or using a matrix notation

$$\begin{pmatrix} f(\vec{g}_1) \\ f(\vec{g}_2) \\ \vdots \\ f(\vec{g}_7) \end{pmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,6} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,6} \\ \vdots & \vdots & \ddots & \vdots \\ m_{7,1} & m_{7,2} & \cdots & m_{7,6} \end{bmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_6 \end{pmatrix} = \mathbf{M} \cdot \vec{f}.$$

The integration formula (6.12) now leads to

$$\iint_{\Omega} f(\xi, \nu) dA \approx \sum_{j=1}^7 w_j f(\vec{g}_j) = \langle \vec{w}, \mathbf{M} \cdot \vec{f} \rangle.$$

If the numbers  $f_i$  represent the values of a quadratic function  $f(x, y)$  at the corners and midpoints of a general triangle then use the above and the transformation rule to conclude

$$\iint_E f(\vec{x}) dA = |\det \mathbf{T}| \langle \vec{w}, \mathbf{M} \cdot \vec{f} \rangle = |\det \mathbf{T}| \langle \mathbf{M}^T \cdot \vec{w}, \vec{f} \rangle. \quad (6.14)$$

The interpolation matrix  $\mathbf{M}$

$$\mathbf{M} \approx \begin{bmatrix} +0.4743526 & -0.0807686 & -0.0807686 & 0.0410358 & 0.3230744 & 0.3230744 \\ -0.0807686 & +0.4743526 & -0.0807686 & 0.3230744 & 0.0410358 & 0.3230744 \\ -0.0807686 & -0.0807686 & +0.4743526 & 0.3230744 & 0.3230744 & 0.0410358 \\ -0.0525839 & -0.0280749 & -0.0280749 & 0.8841342 & 0.1122998 & 0.1122998 \\ -0.0280749 & -0.0525839 & -0.0280749 & 0.1122998 & 0.8841342 & 0.1122998 \\ -0.0280749 & -0.0280749 & -0.0525839 & 0.1122998 & 0.1122998 & 0.8841342 \\ -0.1111111 & -0.1111111 & -0.1111111 & 0.4444444 & 0.4444444 & 0.4444444 \end{bmatrix}$$

and the weight vector  $\vec{w}$  do not depend on the triangle  $E$ , but only on the standard elements and the choice of integration method. Thus for a new triangle  $E$  only the determinant of  $\mathbf{T}$  has to be computed. Since

$$\mathbf{M}^T \cdot \vec{w} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1/6 \\ 1/6 \\ 1/6 \end{pmatrix}$$

the integration of quadratic functions by (6.14) is rather easy to do: add up the values of the function at the three mid-points of the edges, then divide the result by 6 and multiply by  $|\det \mathbf{T}|$ .

### Integration of a product of quadratic functions

Let  $f$  and  $g$  be quadratic functions given at the nodal points in the general triangle  $E$ . Then the integral of the product can be computed by

$$\iint_E f(\vec{x}) g(\vec{x}) dA \approx |\det \mathbf{T}| \langle \mathbf{M} \cdot \vec{f}, \text{diag}(\vec{w}) \cdot \mathbf{M} \cdot \vec{g} \rangle = |\det \mathbf{T}| \langle \mathbf{M}^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M} \cdot \vec{f}, \vec{g} \rangle$$

where

$$\mathbf{M}^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M} = \frac{1}{360} \begin{bmatrix} 6 & -1 & -1 & -4 & 0 & 0 \\ -1 & 6 & -1 & 0 & -4 & 0 \\ -1 & -1 & 6 & 0 & 0 & -4 \\ -4 & 0 & 0 & 32 & 16 & 16 \\ 0 & -4 & 0 & 16 & 32 & 16 \\ 0 & 0 & -4 & 16 & 16 & 32 \end{bmatrix}.$$

This result may be confirmed with the help of a program capable of symbolic computations.

### 6.5.4 Integrals to be Computed

To examine weak solution of boundary value problems the following integrals have to be computed.

$$\iint_{\Omega} \nabla \phi \cdot (a \nabla u + u \vec{b}) + \phi (b_0 u - f) dA - \oint_{\Gamma_2} \phi (g_2 + g_3 u) ds = 0.$$

The unknown function  $u$  and the test function  $\varphi$  will be given at the nodes. Now develop numerical integration formulas for the above expressions. We have to aim for expression of the form

$$\langle \mathbf{A} \cdot \vec{u}, \vec{\phi} \rangle \quad \text{and} \quad \langle \vec{b}, \vec{\phi} \rangle.$$

### 6.5.5 The Octave code ElementContribution.m

To start examine integrals over one triangle only and seek an algorithm implemented in *Octave* to compute the above integrals for a given triangle  $E$  and functions  $a$ ,  $b$  and  $f$ . As a starting point use the codes for the Gauss points and weights. Observe that the code below is far from complete.

#### ElementContribution.m

```
function [elMat,elVec] = ElementContribution(corners,aFunc,bFunc,fFunc)
% [...] = ElementContribution(...)
%   compute the element stiffness matrix of one element
%
% [elMat,elVec] = ElementContribution(corners,aFunc,bFunc,fFunc)
%   corners  coordinates of the three corners forming the triangular element
%   aFunc bFunc fFunc  function files or scalars for the coefficient functions
%   elMat element stiffness matrix
%   elVec vector contribution to the RHS of the linear equation

% define the Gauss points and integration weights
11 = 0.20257301464691267760; 12 = 0.94028412821023017954;
w1 = 0.062969590272413576298; w2 = 0.066197076394253090369;
w3 = 0.1125; w = [w1,w1,w1,w2,w2,w3]';
```

```

G = [11/2 11/2; 1-11,11/2; 11/2, 1-11;
      12/2 12/2; 1-12,12/2; 12/2, 1-12; 1/3 1/3];

T = [corners(2,:)-corners(1,:); corners(3,:)-corners(1,:)]; detT = abs(det(T));
P = G*T;
P(:,1) = P(:,1)+corners(1,1); P(:,2) = P(:,2)+corners(1,2);

InterpolationMatrix = [...give the numerical values...];

```

## 6.5.6 Integration of $f \phi$ over one Triangle

### General situation

First evaluate the coefficient function  $f(\vec{x})$  at the Gauss integration points, leading to a vector  $\vec{f}$ . Use the computations from the above section to conclude that

$$\iint_E f \phi \, dA \approx |\det \mathbf{T}| \langle \mathbf{M}^T \cdot \text{diag}(\vec{w}) \cdot \vec{f}, \vec{\phi} \rangle = \langle \mathbf{W}_E \vec{f}, \vec{\phi} \rangle.$$

The contribution to the element vector is

$$\mathbf{W}_E \vec{f} = |\det \mathbf{T}| \mathbf{M}^T \cdot \text{diag}(\vec{w}) \cdot \vec{f} = |\det \mathbf{T}| \mathbf{M}^T \cdot \begin{pmatrix} w_1 f(\vec{g}_1) \\ w_1 f(\vec{g}_2) \\ w_1 f(\vec{g}_3) \\ w_2 f(\vec{g}_4) \\ w_2 f(\vec{g}_5) \\ w_2 f(\vec{g}_6) \\ w_3 f(\vec{g}_7) \end{pmatrix}.$$

This is implemented by the code lines below.

### ElementContribution.m

```

if ischar(fFunc) val = feval(fFunc,P); else val = fFunc(:); end%if
elVec = detT*InterpolationMatrix'*(w.*val);

```

### Simplification if $f$ is constant

Now the contribution to the element vector is

$$f |\det \mathbf{T}| \mathbf{M}^T \cdot \vec{w} = f |\det \mathbf{T}| \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1/6 \\ 1/6 \\ 1/6 \end{pmatrix}$$

and thus the effect of  $\iint_E f \phi \, dA$  is very easy to implement.

### 6.5.7 Integration of $b_0 u \phi$ over one Triangle

#### General situation

First evaluate the coefficient function  $b_0(\vec{x})$  at the Gauss integration points, leading to a vector  $\vec{b}_0$ . Then we use the computations from the above section to conclude that

$$\iint_E b_0 u \phi dA \approx |\det \mathbf{T}| \langle \mathbf{M}^T \cdot \text{diag}(\vec{w}) \cdot \text{diag}(\vec{b}_0) \cdot \mathbf{M} \cdot \vec{u}, \vec{\phi} \rangle = \langle \mathbf{B}_0 \vec{u}, \vec{\phi} \rangle.$$

This is implemented by the code lines below.

#### ElementContribution.m

```
if ischar(bFunc) val = feval(bFunc,P); else val = bFunc(:); end%if
elMat = detT*InterpolationMatrix'*diag(w.*val)*InterpolationMatrix;
```

#### Simplification if $b_0$ is constant

Now the contribution to the element stiffness matrix is

$$b_0 |\det \mathbf{T}| \mathbf{M}^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M} = \frac{b_0 |\det \mathbf{T}|}{360} \begin{bmatrix} 6 & -1 & -1 & -4 & 0 & 0 \\ -1 & 6 & -1 & 0 & -4 & 0 \\ -1 & -1 & 6 & 0 & 0 & -4 \\ -4 & 0 & 0 & 32 & 16 & 16 \\ 0 & -4 & 0 & 16 & 32 & 16 \\ 0 & 0 & -4 & 16 & 16 & 32 \end{bmatrix}.$$

### 6.5.8 Transformation of the Gradient to the Standard Triangle

The still to be examined integral expression contain components of the gradients  $\nabla u$  and  $\nabla \phi$ . First examine how the gradient behave under the transformation to the standard triangle, only then use the above integration methods.

According to section 6.5.1 the coordinates  $(\xi, \nu)$  of the standard triangle are connected to the global coordinates  $(x, y)$  by

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \begin{pmatrix} \xi \\ \nu \end{pmatrix}$$

or equivalently

$$\begin{pmatrix} \xi \\ \nu \end{pmatrix} = \mathbf{T}^{-1} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}.$$

If a function  $f(x, y)$  is given on the general triangle  $E$  we can pull it back to the standard triangle by

$$g(\xi, \nu) = f(x(\xi, \nu), y(\xi, \nu))$$

and then compute the gradient of  $g$  with respect to its independent variables  $\xi$  and  $\nu$ . The result will depend on the partial derivatives of  $f$  with respect to  $x$  and  $y$ . The standard chain rule implies

$$\frac{\partial}{\partial \xi} g(\xi, \nu) = \frac{\partial}{\partial \xi} f(x(\xi, \nu), y(\xi, \nu)) = \frac{\partial f(x, y)}{\partial x} \frac{\partial x}{\partial \xi} + \frac{\partial f(x, y)}{\partial y} \frac{\partial y}{\partial \xi}$$

$$\begin{aligned}
&= \frac{\partial f(x, y)}{\partial x} (x_2 - x_1) + \frac{\partial f(x, y)}{\partial y} (y_2 - y_1) \\
\frac{\partial}{\partial \nu} g(\xi, \nu) &= \frac{\partial}{\partial \nu} f(x(\xi, \nu), y(\xi, \nu)) = \frac{\partial f(x, y)}{\partial x} \frac{\partial x}{\partial \nu} + \frac{\partial f(x, y)}{\partial y} \frac{\partial y}{\partial \nu} \\
&= \frac{\partial f(x, y)}{\partial x} (x_3 - x_1) + \frac{\partial f(x, y)}{\partial y} (y_3 - y_1).
\end{aligned}$$

This can be written with the help of matrices as

$$\begin{pmatrix} \frac{\partial g}{\partial \xi} \\ \frac{\partial g}{\partial \nu} \end{pmatrix} = \begin{bmatrix} (x_2 - x_1) & (y_2 - y_1) \\ (x_3 - x_1) & (y_3 - y_1) \end{bmatrix} \cdot \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \mathbf{T}^T \cdot \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

or equivalently

$$\left( \frac{\partial g}{\partial \xi}, \frac{\partial g}{\partial \nu} \right) = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \cdot \mathbf{T} \quad .$$

This implies

$$\left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = \left( \frac{\partial g}{\partial \xi}, \frac{\partial g}{\partial \nu} \right) \cdot \mathbf{T}^{-1} = \frac{1}{\det \mathbf{T}} \left( \frac{\partial g}{\partial \xi}, \frac{\partial g}{\partial \nu} \right) \cdot \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix} .$$

Let  $\varphi$  be a function on the standard triangle  $\Omega$ , given as a linear combination of the basis functions, i.e.

$$\varphi(\xi, \nu) = \sum_{i=1}^6 \varphi_i \Phi_i(\xi, \nu) ,$$

where

$$\vec{\Phi}(\xi, \nu) = \begin{pmatrix} \Phi_1(\xi, \nu) \\ \Phi_2(\xi, \nu) \\ \Phi_3(\xi, \nu) \\ \Phi_4(\xi, \nu) \\ \Phi_5(\xi, \nu) \\ \Phi_6(\xi, \nu) \end{pmatrix} = \begin{pmatrix} (1 - \xi - \nu)(1 - 2\xi - 2\nu) \\ \xi(2\xi - 1) \\ \nu(2\nu - 1) \\ 4\xi\nu \\ 4\nu(1 - \xi - \nu) \\ 4\xi(1 - \xi - \nu) \end{pmatrix} .$$

Then its gradient with respect to  $\xi$  and  $\nu$  can be determined with the help of

$$\text{grad } \vec{\Phi} = \begin{bmatrix} -3 + 4\xi + 4\nu & -3 + 4\xi + 4\nu \\ 4\xi - 1 & 0 \\ 0 & 4\nu - 1 \\ 4\nu & 4\xi \\ -4\nu & 4 - 4\xi - 8\nu \\ 4 - 8\xi - 4\nu & -4\xi \end{bmatrix} = \begin{bmatrix} \vec{\Phi}_\xi(\xi, \nu) & \vec{\Phi}_\nu(\xi, \nu) \end{bmatrix} .$$

Thus conclude

$$\left( \frac{\partial \varphi}{\partial \xi}, \frac{\partial \varphi}{\partial \nu} \right) = (\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6) \cdot \begin{bmatrix} \vec{\Phi}_\xi(\xi, \nu) & \vec{\Phi}_\nu(\xi, \nu) \end{bmatrix} = \vec{\varphi}^T \cdot \begin{bmatrix} \vec{\Phi}_\xi(\xi, \nu) & \vec{\Phi}_\nu(\xi, \nu) \end{bmatrix} .$$

If the function  $\varphi(x, y)$  is given on the general triangle as linear combination of the basis-functions on  $E$  find

$$\varphi(x, y) = \sum_{i=1}^6 \varphi_i \Phi_i(\xi(x, y), \nu(x, y)) .$$

Now combine the results in this section to find

$$\left( \frac{\partial \varphi}{\partial x}, \frac{\partial \varphi}{\partial y} \right) = \left( \frac{\partial \varphi}{\partial \xi}, \frac{\partial \varphi}{\partial \nu} \right) \cdot \mathbf{T}^{-1} = \vec{\varphi}^T \cdot \begin{bmatrix} \vec{\Phi}_\xi & \vec{\Phi}_\nu \end{bmatrix} \cdot \mathbf{T}^{-1}$$

or by transposition

$$\begin{pmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \end{pmatrix} = (\mathbf{T}^{-1})^T \cdot \begin{bmatrix} \vec{\Phi}_\xi^T \\ \vec{\Phi}_\nu^T \end{bmatrix} \cdot \vec{\varphi} = \frac{1}{\det(\mathbf{T})} \begin{bmatrix} y_3 - y_1 & -y_2 + y_1 \\ -x_3 + x_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{bmatrix} \vec{\Phi}_\xi^T \\ \vec{\Phi}_\nu^T \end{bmatrix} \cdot \vec{\varphi}$$

and the same identities can be spelled out for the two components independently

$$\begin{aligned} \frac{\partial \varphi}{\partial x} &= \frac{1}{\det(\mathbf{T})} \left[ (+y_3 - y_1) \vec{\Phi}_\xi^T + (-y_2 + y_1) \vec{\Phi}_\nu^T \right] \cdot \vec{\varphi}, \\ \frac{\partial \varphi}{\partial y} &= \frac{1}{\det(\mathbf{T})} \left[ (-x_3 + x_1) \vec{\Phi}_\xi^T + (+x_2 - x_1) \vec{\Phi}_\nu^T \right] \cdot \vec{\varphi}. \end{aligned}$$

For the numerical integration use the values of the gradients at the Gauss integration points  $\vec{g}_j = (\xi_j, \nu_j)$ . We already found that the values of the function  $\varphi$  at the Gauss points can be computed with the help of the interpolation matrix  $\mathbf{M}$  by

$$\begin{pmatrix} \varphi(\vec{g}_1) \\ \varphi(\vec{g}_2) \\ \vdots \\ \varphi(\vec{g}_7) \end{pmatrix} = \mathbf{M} \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_6 \end{pmatrix}.$$

Similarly define the interpolation matrices for the partial derivatives. Using

$$\begin{aligned} \mathbf{M}_\xi &= \begin{bmatrix} -3 + 4\xi_1 + 4\nu_1 & 4\xi_1 - 1 & 0 & 4\nu_1 & -4\nu_1 & 4 - 8\xi_1 - 4\nu_1 \\ -3 + 4\xi_2 + 4\nu_2 & 4\xi_2 - 1 & 0 & 4\nu_2 & -4\nu_2 & 4 - 8\xi_2 - 4\nu_2 \\ \vdots & & & & & \vdots \\ -3 + 4\xi_m + 4\nu_m & 4\xi_m - 1 & 0 & 4\nu_m & -4\nu_m & 4 - 8\xi_m - 4\nu_m \end{bmatrix} \\ &\approx \begin{bmatrix} -2.18971 & -0.59485 & 0.00000 & 0.40515 & -0.40515 & 2.78456 \\ 0.59485 & 2.18971 & 0.00000 & 0.40515 & -0.40515 & -2.78456 \\ 0.59485 & -0.59485 & 0.00000 & 3.18971 & -3.18971 & 0.00000 \\ 0.76114 & 0.88057 & 0.00000 & 1.88057 & -1.88057 & -1.64170 \\ -0.88057 & -0.76114 & 0.00000 & 1.88057 & -1.88057 & 1.64170 \\ -0.88057 & 0.88057 & 0.00000 & 0.23886 & -0.23886 & 0.00000 \\ -0.33333 & 0.33333 & 0.00000 & 1.33333 & -1.33333 & 0.00000 \end{bmatrix} \end{aligned}$$

find

$$\begin{pmatrix} \varphi_\xi(\vec{g}_1) \\ \varphi_\xi(\vec{g}_2) \\ \vdots \\ \varphi_\xi(\vec{g}_7) \end{pmatrix} = \mathbf{M}_\xi \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_6 \end{pmatrix}.$$

Similarly write

$$\mathbf{M}_\nu = \begin{bmatrix} -3 + 4\xi_1 + 4\nu_1 & 0 & 4\nu_1 - 1 & 4\xi_1 & 4 - 4\xi_1 - 8\nu_1 & -4\xi_1 \\ -3 + 4\xi_2 + 4\nu_2 & 0 & 4\nu_2 - 1 & 4\xi_2 & 4 - 4\xi_2 - 8\nu_2 & -4\xi_2 \\ \vdots & & & & & \vdots \\ -3 + 4\xi_m + 4\nu_m & 0 & 4\nu_m - 1 & 4\xi_m & 4 - 4\xi_m - 8\nu_m & -4\xi_m \end{bmatrix}$$

$$\approx \begin{bmatrix} -2.18971 & 0.00000 & -0.59485 & 0.40515 & 2.78456 & -0.40515 \\ 0.59485 & 0.00000 & -0.59485 & 3.18971 & 0.00000 & -3.18971 \\ 0.59485 & 0.00000 & 2.18971 & 0.40515 & -2.78456 & -0.40515 \\ 0.76114 & 0.00000 & 0.88057 & 1.88057 & -1.64170 & -1.88057 \\ -0.88057 & 0.00000 & 0.88057 & 0.23886 & 0.00000 & -0.23886 \\ -0.88057 & 0.00000 & -0.76114 & 1.88057 & 1.64170 & -1.88057 \\ -0.33333 & 0.00000 & 0.33333 & 1.33333 & 0.00000 & -1.33333 \end{bmatrix}$$

and

$$\begin{pmatrix} \varphi_\nu(\vec{g}_1) \\ \varphi_\nu(\vec{g}_2) \\ \vdots \\ \varphi_\nu(\vec{g}_7) \end{pmatrix} = \mathbf{M}_\nu \cdot \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_6 \end{pmatrix}.$$

Combining the above two computations use the notation

$$\vec{x}_i = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{T} \cdot \begin{pmatrix} \xi_i \\ \nu_i \end{pmatrix} \quad \text{for } i = 1, 2, 3, \dots, 7$$

and find for the first component  $\varphi_x = \frac{\partial \varphi}{\partial x}$  of the gradient at the Gauss points

$$\begin{pmatrix} \varphi_x(\vec{x}_1) \\ \varphi_x(\vec{x}_2) \\ \vdots \\ \varphi_x(\vec{x}_7) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[ (+y_3 - y_1) \mathbf{M}_\xi^T + (-y_2 + y_1) \mathbf{M}_\nu^T \right] \cdot \vec{\phi}$$

and for the second component of the gradient

$$\begin{pmatrix} \varphi_y(\vec{x}_1) \\ \varphi_y(\vec{x}_2) \\ \vdots \\ \varphi_y(\vec{x}_7) \end{pmatrix} = \frac{1}{\det(\mathbf{T})} \left[ (-x_3 + x_1) \mathbf{M}_\xi^T + (+x_2 - x_1) \mathbf{M}_\nu^T \right] \cdot \vec{\phi}.$$

The above results for  $\mathbf{M}_\xi$  and  $\mathbf{M}_\nu$  can be implemented in MATLAB/Octave and then used to compute the element stiffness matrix.

### 6.5.9 Integration of $u \vec{b} \nabla \phi$ over one Triangle

The vector function  $\vec{b}(\vec{x})$  has to be evaluated at the Gauss integration points  $\vec{g}_j$ . Then the integration of

$$\text{Cont} = \iint_E u \vec{b} \nabla \phi \, dA = \iint_E u b_1 \frac{\partial \phi}{\partial x} \, dA + \iint_E u b_2 \frac{\partial \phi}{\partial y} \, dA$$

is replaced by a weighted summation. Introduce the vectors

$$\overrightarrow{wb}_1 = \begin{pmatrix} w_1 b_1(\vec{g}_1) \\ w_2 b_1(\vec{g}_2) \\ \vdots \\ w_7 b_1(\vec{g}_7) \end{pmatrix} \quad \text{and} \quad \overrightarrow{wb}_2 = \begin{pmatrix} w_1 b_2(\vec{g}_1) \\ w_2 b_2(\vec{g}_2) \\ \vdots \\ w_7 b_2(\vec{g}_7) \end{pmatrix}.$$

Using the results of the previous sections find

$$\begin{aligned} \iint_E u b_1 \frac{\partial \phi}{\partial x} \, dA &= \frac{1}{\det \mathbf{T}} \iint_E u b_1 ((y_3 - y_1) \phi_\xi + (-y_2 + y_1) \phi_\nu) \, dA \\ &\approx \frac{|\det \mathbf{T}|}{\det \mathbf{T}} \langle \text{diag}(\overrightarrow{wb}_1) \cdot \mathbf{M} \cdot \vec{u}, (y_3 - y_1) \mathbf{M}_\xi \cdot \vec{\phi} + (-y_2 + y_1) \mathbf{M}_\nu \cdot \vec{\phi} \rangle \\ &= \frac{|\det \mathbf{T}|}{\det \mathbf{T}} \langle ((y_3 - y_1) \mathbf{M}_\xi^T + (-y_2 + y_1) \mathbf{M}_\nu^T) \cdot \text{diag}(\overrightarrow{wb}_1) \cdot \mathbf{M} \cdot \vec{u}, \vec{\phi} \rangle \\ &= \langle \mathbf{B}_1 \vec{u}, \vec{\phi} \rangle \end{aligned}$$

and similarly

$$\begin{aligned} \iint_E u b_2 \frac{\partial \phi}{\partial y} \, dA &= \frac{1}{\det \mathbf{T}} \iint_E u b_2 ((-x_3 + x_1) \phi_\xi + (x_2 - x_1) \phi_\nu) \, dA \\ &\approx \frac{|\det \mathbf{T}|}{\det \mathbf{T}} \langle \text{diag}(\overrightarrow{wb}_2) \cdot \mathbf{M} \cdot \vec{u}, (-x_3 + x_1) \mathbf{M}_\xi \cdot \vec{\phi} + (x_2 - x_1) \mathbf{M}_\nu \cdot \vec{\phi} \rangle \\ &= \frac{|\det \mathbf{T}|}{\det \mathbf{T}} \langle ((-x_3 + x_1) \mathbf{M}_\xi^T + (x_2 - x_1) \mathbf{M}_\nu^T) \cdot \text{diag}(\overrightarrow{wb}_2) \cdot \mathbf{M} \cdot \vec{u}, \vec{\phi} \rangle \\ &= \langle \mathbf{B}_2 \vec{u}, \vec{\phi} \rangle \end{aligned}$$

For computations use the above formula. A slightly more compact notation is given by

$$\begin{aligned} \text{Cont} &= \iint_E u \vec{b} \nabla \phi \, dA \\ &\approx \frac{|\det \mathbf{T}|}{\det \mathbf{T}} \langle [\mathbf{M}_\xi^T, \mathbf{M}_\nu^T] \cdot \begin{bmatrix} (y_3 - y_1) \text{diag}(\overrightarrow{wb}_1) + (-x_3 + x_1) \text{diag}(\overrightarrow{wb}_2) \\ (-y_2 + y_1) \text{diag}(\overrightarrow{wb}_1) + (x_2 - x_1) \text{diag}(\overrightarrow{wb}_2) \end{bmatrix} \cdot \mathbf{M} \cdot \vec{u}, \vec{\phi} \rangle \\ &= \frac{|\det \mathbf{T}|}{\det \mathbf{T}} \langle [\mathbf{M}_\xi^T, \mathbf{M}_\nu^T] \cdot \begin{bmatrix} (y_3 - y_1) & (-x_3 + x_1) \\ (-y_2 + y_1) & (x_2 - x_1) \end{bmatrix} \cdot \begin{bmatrix} \text{diag}(\overrightarrow{wb}_1) \\ \text{diag}(\overrightarrow{wb}_2) \end{bmatrix} \cdot \mathbf{M} \cdot \vec{u}, \vec{\phi} \rangle \\ &= |\det \mathbf{T}| \langle [\mathbf{M}_\xi^T, \mathbf{M}_\nu^T] \cdot \mathbf{T}^{-1} \cdot \begin{bmatrix} \text{diag}(\overrightarrow{wb}_1) \\ \text{diag}(\overrightarrow{wb}_2) \end{bmatrix} \cdot \mathbf{M} \cdot \vec{u}, \vec{\phi} \rangle. \end{aligned}$$

In the special case of  $E = \Omega$  use  $x_2 - x_1 = y_3 - y_1 = 1$ ,  $x_3 - x_1 = y_3 - y_1 = 0$  and thus  $\mathbf{T} = \mathbb{I}$  and  $\det \mathbf{T} = 1$ . For a constant vector  $\vec{b}$  the above simplifies to

$$\begin{aligned}\iint_{\Omega} u b_1 \frac{\partial \phi}{\partial x} dA &= b_1 \langle \mathbf{M}_{\xi}^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M} \cdot \vec{u}, \vec{\phi} \rangle = \langle \mathbf{B}_1 \vec{u}, \vec{\phi} \rangle \\ \iint_{\Omega} u b_2 \frac{\partial \phi}{\partial y} dA &= b_2 \langle \mathbf{M}_{\nu}^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M} \cdot \vec{u}, \vec{\phi} \rangle = \langle \mathbf{B}_2 \vec{u}, \vec{\phi} \rangle.\end{aligned}$$

### 6.5.10 Integration of $a \nabla u \cdot \nabla \phi$ over one Triangle

The function  $a \nabla u \cdot \nabla \phi = a (\frac{\partial u}{\partial x} \frac{\partial \phi}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial \phi}{\partial y})$  has to be evaluated at the Gauss integration points  $\vec{g}_j$ , then multiplied by the Gauss weights  $w_i$  and added up. Introduce the vector

$$\overrightarrow{wa} = \begin{pmatrix} w_1 a(\vec{x}(\vec{g}_1)) \\ w_2 a(\vec{x}(\vec{g}_2)) \\ \vdots \\ w_7 a(\vec{x}(\vec{g}_7)) \end{pmatrix}.$$

For the integration over the general triangle  $E$  use the transformation formula (6.11) and obtain

$$\begin{aligned}\iint_E a \frac{\partial u(\vec{x})}{\partial x} \frac{\partial \phi(\vec{x})}{\partial x} dA &= |\det \mathbf{T}| \iint_{\Omega} a(\vec{x}(\xi, \nu)) \frac{\partial u(\vec{x}(\xi, \nu))}{\partial x} \frac{\partial \phi(\vec{x}(\xi, \nu))}{\partial x} d\xi d\nu \\ &\approx \frac{|\det \mathbf{T}|}{(\det \mathbf{T})^2} \langle \mathbf{A}_x \cdot \vec{u}, \vec{\phi} \rangle = \frac{1}{|\det \mathbf{T}|} \langle \mathbf{A}_x \cdot \vec{u}, \vec{\phi} \rangle\end{aligned}$$

where

$$\begin{aligned}\mathbf{A}_x &= \left[ (+y_3 - y_1) \mathbf{M}_{\xi} + (-y_2 + y_1) \mathbf{M}_{\nu} \right]^T \cdot \text{diag}(\overrightarrow{wa}) \cdot \left[ (+y_3 - y_1) \mathbf{M}_{\xi} + (-y_2 + y_1) \mathbf{M}_{\nu} \right] \\ &= \left[ (+y_3 - y_1) \mathbf{M}_{\xi}^T + (-y_2 + y_1) \mathbf{M}_{\nu}^T \right] \cdot \text{diag}(\overrightarrow{wa}) \cdot \left[ (+y_3 - y_1) \mathbf{M}_{\xi} + (-y_2 + y_1) \mathbf{M}_{\nu} \right] \\ &= (+y_3 - y_1)^2 \mathbf{M}_{\xi}^T \cdot \text{diag}(\overrightarrow{wa}) \cdot \mathbf{M}_{\xi} + (-y_2 + y_1)^2 \mathbf{M}_{\nu}^T \cdot \text{diag}(\overrightarrow{wa}) \cdot \mathbf{M}_{\nu} \\ &\quad + (+y_3 - y_1)(-y_2 + y_1) \left( \mathbf{M}_{\xi}^T \cdot \text{diag}(\overrightarrow{wa}) \cdot \mathbf{M}_{\nu} + \mathbf{M}_{\nu}^T \cdot \text{diag}(\overrightarrow{wa}) \cdot \mathbf{M}_{\xi} \right).\end{aligned}$$

For a constant coefficient  $a$  more of the above expressions can be computed explicitly and simplified.

$$\begin{aligned}\mathbf{M}_{\xi}^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M}_{\xi} &= \frac{1}{6} \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & -4 \\ 1 & 3 & 0 & 0 & 0 & -4 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & -8 & 0 \\ 0 & 0 & 0 & -8 & 8 & 0 \\ -4 & -4 & 0 & 0 & 0 & 8 \end{bmatrix} \\ \mathbf{M}_{\nu}^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M}_{\nu} &= \frac{1}{6} \begin{bmatrix} 3 & 0 & 1 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 0 & -4 & 0 \\ 0 & 0 & 0 & 8 & 0 & -8 \\ -4 & 0 & -4 & 0 & 8 & 0 \\ 0 & 0 & 0 & -8 & 0 & 8 \end{bmatrix}\end{aligned}$$

$$\mathbf{M}_\xi^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M}_\nu + \mathbf{M}_\nu^T \cdot \text{diag}(\vec{w}) \cdot \mathbf{M}_\xi = \frac{1}{6} \begin{bmatrix} 6 & 1 & 1 & 0 & -4 & -4 \\ 1 & 0 & -1 & 4 & 0 & -4 \\ 1 & -1 & 0 & 4 & -4 & 0 \\ 0 & 4 & 4 & 8 & -8 & -8 \\ -4 & 0 & -4 & -8 & 8 & 8 \\ -4 & -4 & 0 & -8 & 8 & 8 \end{bmatrix}$$

Similarly determine

$$\begin{aligned} \iint_E a \frac{\partial u(\vec{x})}{\partial y} \frac{\partial \phi(\vec{x})}{\partial y} dA &= |\det \mathbf{T}| \iint_\Omega a(\vec{x}(\xi, \nu)) \frac{\partial u(\vec{x}(\xi, \nu))}{\partial y} \frac{\partial \phi(\vec{x}(\xi, \nu))}{\partial y} d\xi d\nu \\ &\approx \frac{|\det \mathbf{T}|}{(\det \mathbf{T})^2} \langle \mathbf{A}_y \cdot \vec{u}, \vec{\phi} \rangle = \frac{1}{|\det \mathbf{T}|} \langle \mathbf{A}_y \cdot \vec{u}, \vec{\phi} \rangle \end{aligned}$$

where

$$\mathbf{A}_y = \left[ (-x_3 + x_1) \mathbf{M}_\xi + (+x_2 - x_1) \mathbf{M}_\nu \right]^T \cdot \text{diag}(\vec{w}) \cdot \left[ (-x_3 + x_1) \mathbf{M}_\xi + (+x_2 - x_1) \mathbf{M}_\nu \right].$$

Now put all the above computations into one single formula, leading to

$$\iint_E a \nabla u \cdot \nabla \phi dA \approx \frac{1}{|\det \mathbf{T}|} \langle (\mathbf{A}_x + \mathbf{A}_y) \cdot \vec{u}, \vec{\phi} \rangle.$$

This is implemented by the code lines below

---

#### ElementContribution.m

---

```
if ischar(aFunc) val = feval(aFunc,P); else val = aFunc(:); end%if

tt = T(2,2)*Mxi-T(1,2)*Mnu; Ax = tt'*diag(w.*val)*tt;
tt = -T(2,1)*Mxi+T(1,1)*Mnu; Ay = tt'*diag(w.*val)*tt;

elMat = elMat + (Ax+Ay)/detT;
```

---

and with this segment the code for the function `ElementContribution()` is complete. The element stiffness matrix and the element vector can now be computed by

---

#### Octave

---

```
corners = [0 0; 1 0; 0 1];

function res = fFunc(xy)
x = xy(:,1); y = xy(:,2);
res = 1+x.^2;
end%function

function res = bFunc(xy)
x = xy(:,1); y = xy(:,2);
res = 0*x;
end%function

[elMat,elVec] = ElementContribution(corners,'fFunc','bFunc','fFunc')
```

---

### 6.5.11 Construction of the Element Stiffness Matrix

With the above algorithms and resulting codes the element stiffness matrices and vectors can be computed. For each element  $E$  (a triangle) use the approximated integral

$$\iint_E \nabla \phi \cdot (a \nabla u + u \vec{b}) + \phi (b_0 u - f) \, dA \approx \langle \mathbf{A}_E \vec{u}_E, \vec{\phi}_E \rangle - \langle \mathbf{W}_E \vec{f}_E, \vec{\phi}_E \rangle$$

with the element stiffness matrix  $\mathbf{A}_E = \mathbf{A}_x + \mathbf{A}_y + \mathbf{B}_0 + \mathbf{B}_1 + \mathbf{B}_2$ . Then use ideas similar to Section 6.2.6 (page 408) to assemble the results to obtain the global stiffness matrix  $\mathbf{A}$  and the resulting system of linear equations to be solved is given by

$$\mathbf{A} \vec{u} = \mathbf{W} \vec{f}.$$

The boundary contributions in

$$\iint_{\Omega} \nabla \phi \cdot (a \nabla u + u \vec{b}) + \phi (b_0 u - f) \, dA - \oint_{\Gamma_2} \phi (g_2 + g_3 u) \, ds = 0$$

have to be taken into account by very similar procedures. If all goes well the vector  $\vec{u} \in \mathbb{R}^N$  is an approximation of the solution  $u(x, y)$  of the boundary value problem

$$\begin{aligned} -\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u &= f && \text{for } (x, y) \in \Omega \\ u &= g_1 && \text{for } (x, y) \in \Gamma_1 \\ \vec{n} \cdot (a \nabla u + u \vec{b}) &= g_2 + g_3 u && \text{for } (x, y) \in \Gamma_2 \end{aligned}$$

## 6.6 Comparing First and Second Order Triangular Elements

### 6.6.1 Observe the Convergence of the Error as $h \rightarrow 0$

Consider the unit square  $\Omega = [0, 1] \times [0, 1]$ . Verify that the function  $u_e(x, y) = \sin(x) \cdot \sin(y)$  is solution of

$$\begin{aligned} -\nabla \cdot \nabla u &= -2 \sin(x) \cdot \sin(y) && \text{for } 0 \leq x, y \leq 1 \\ \frac{\partial u(x, 1)}{\partial y} &= -\sin(x) \cdot \cos(1) && \text{for } 0 \leq x \leq 1 \text{ and } y = 1 \\ u(x, y) &= u_e(x, y) && \text{on the other sections of the boundary} \end{aligned}$$

Let  $h > 0$  be the typical length of a side of a triangle. For first order elements  $\frac{1}{2}h$  is used, such that the computational effort is comparable to second order elements, i.e. the same number of degrees of freedom. Nonuniform meshes are used, to avoid the effect of superconvergence. By choosing different values of  $h$  one should observe smaller errors for smaller values of  $h$ . The error is measured by computing the  $L_2$  norms of the difference of the exact and approximate solutions, for the values of the functions and its partial derivative with respect to  $y$ . These are the expressions used in the general convergence estimates, based on the abstract error estimate in Theorem 6–6. A double logarithmic plot leads to Figure 6.18.

- For linear elements:
  - The slope of the curve for the absolute values of  $u(x, y) - u_e(x, y)$  is approximately 2 and thus conclude that the error is proportional to  $h^2$ .
  - The slope of the curve for the absolute values of  $\frac{\partial}{\partial y} (u(x, y) - u_e(x, y))$  is approximately 1 and thus conclude that the error of the gradient is proportional to  $h$ .
- For quadratic elements:

- The slope of the curve for the absolute values of  $u(x, y) - u_e(x, y)$  is approximately 3 and thus conclude that the error is proportional to  $h^3$ .
- The slope of the curve for the absolute values of  $\frac{\partial}{\partial y} (u(x, y) - u_e(x, y))$  is approximately 2 and thus conclude that the error of the gradient is proportional to  $h^2$ .

These observations confirm the theoretical error estimates in Theorem 6–9 (page 425) and Theorem 6–11 (page 427). It is rather obvious from Figure 6.18 that second order elements generate more accurate solutions for a comparable computational effort.

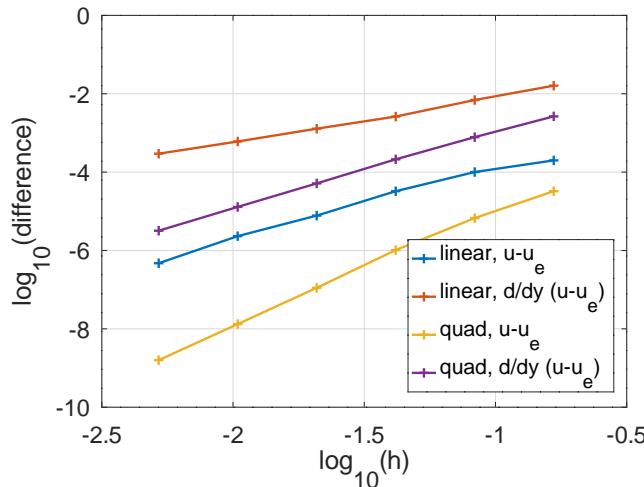


Figure 6.18: Convergence results for linear and quadratic elements

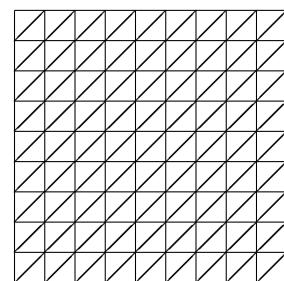
### 6.6.2 Estimate the Number of Nodes and Triangles and the Effect on the Sparse Matrix

Let  $\Omega \subset \mathbb{R}^2$  be a domain with a triangular mesh with many triangles. Examine the typical mesh, shown below, and consider only triangles and nodes inside the mesh, as the contributions by the borders are considerably smaller for large meshes.

We search a connection between

$$N = \text{number of nodes and } T = \text{number of triangles.}$$

- each triangle has three corners
- each (internal) corner is touched by 6 triangles
- each triangle has 3 midpoints of edges and each of the midpoints is shared by 2 triangles
- For first order elements the nodes are the corners of the triangles.



$$N \approx \frac{1}{6} T 3 = \frac{1}{2} T$$

Thus the number  $N$  of nodes is approximately half the number  $T$  of triangles.

- For second order elements the nodes are the corners of the triangles and the midpoints of the edges. Each midpoint is shared by two triangles.

$$N \approx \frac{1}{2} T + \frac{3}{2} T = 2T$$

Thus the number  $N$  of nodes is approximately twice the number  $T$  of triangles.

The above implies that the number of degrees of freedom to solve a problem with second order elements with a typical diameter  $h$  of the triangles is approximately equal to using linear element on triangles with diameter  $h/2$ .

The above estimates also allow to estimate how many entries in the sparse matrix resulting from an FEM algorithm will be different from zero.

- For linear elements each node typically touches 6 triangles and each of the involved corners is shared by two triangles. Thus there might be  $6 + 1 = 7$  nonzero entries in each row of the matrix.
- For second order triangles we have to distinguish between corners and midpoints.
  - Each corner touches typically six triangles and thus expect up to  $6 \times 3 + 1 = 19$  nonzero entries in the corresponding row of the matrix.
  - Each midpoint touches two triangles and two of the corner points are shared. Thus expect up to  $2 + 2 \times 3 + 1 = 9$  nonzero entries in the corresponding row of the matrix.

The midpoints outnumber the corners by a factor of three. Thus expect an average of  $\frac{3 \cdot 9 + 19}{4} = 11.5$  nonzero entries in each row of the matrix.

This points to about a factor of  $\frac{11.5}{7} \approx 1.6$  more nonzero entries in the matrix for quadratic elements for the same number of degrees of freedom. This implies that the computational effort is larger, the size of the effect depends on the linear solver used.

### 6.6.3 Behavior of a FEM Solution within Triangular Elements

To examine the behavior of a solution  $u(x, y)$  within each of the triangular elements use the boundary value problem

$$\begin{aligned} -\Delta u &= -\exp(y) && \text{for } (x, y) \in \Omega \\ u(x, y) &= +\exp(y) && \text{for } (x, y) \in \Gamma = \partial\Omega \end{aligned}$$

on the domain  $\Omega$  displayed in Figure 6.19(a). The exact solution is given by  $u(x, y) = \exp(y)$ , shown in Figure 6.19(b). The problem is solved twice, using different elements:

1. using 32 triangular elements of order 1.
2. using 8 triangular elements of order 2.

The degrees of freedom and nodes used coincide for the two approaches, i.e four triangles in Figure 6.19(a) for the linear elements from one of the eight triangles for the quadratic elements.

Figure 6.20(a) shows the difference of the computed solution with first order elements to the exact solution. Within each of the 32 elements the difference is not too far from a quadratic function. Figure 6.20(b) shows the values of the partial derivative  $\frac{\partial u}{\partial y}$ . It is clearly visible that the gradient is constant within each triangle, and not continuous across element borders.

Figure 6.21(a) shows the difference of the computed solution with second order elements to the exact solution. The error is considerably smaller than for linear elements, using identical degrees of freedom. Within each of the 8 elements the difference does not show a simple structure. Figure 6.21(b) shows the

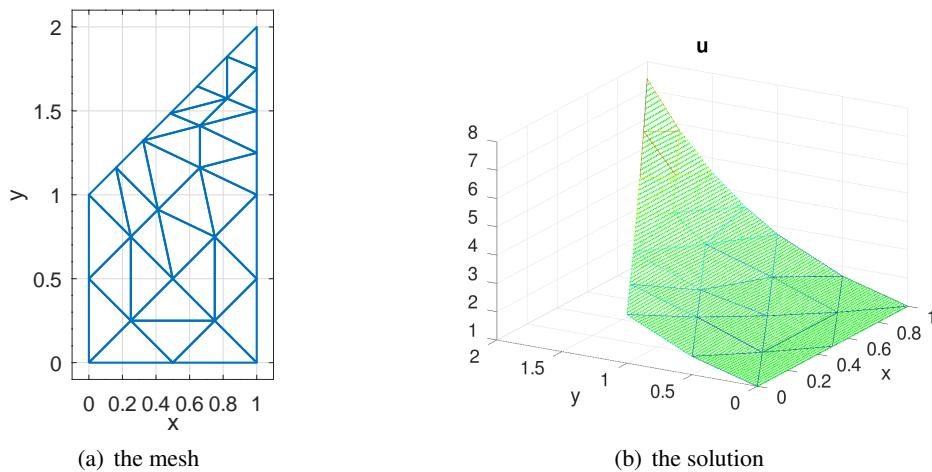
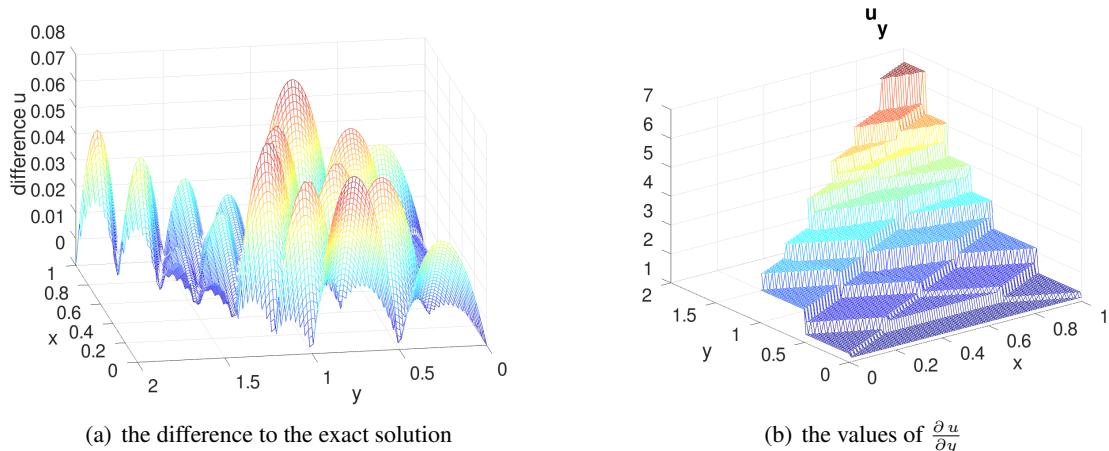
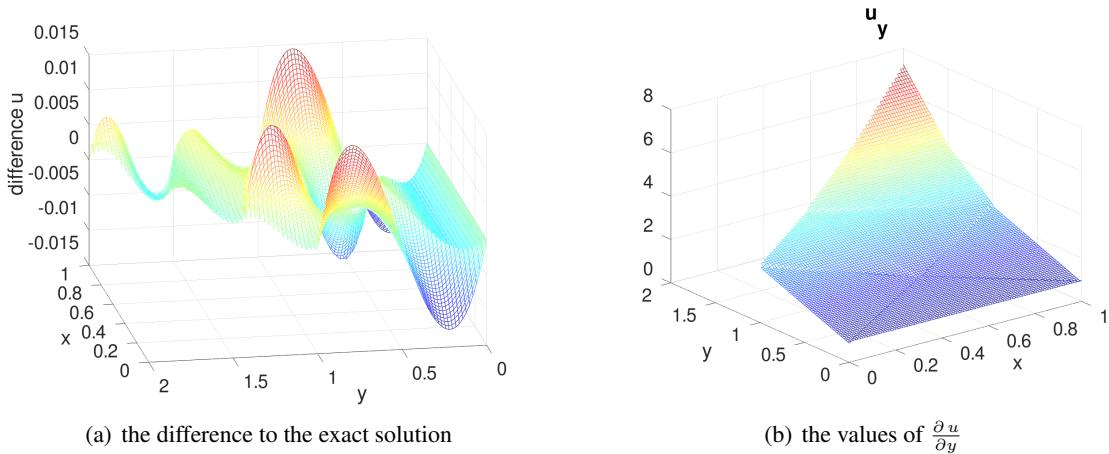


Figure 6.19: The mesh and the solution for a BVP

Figure 6.20: Difference to the exact solution and values of  $\frac{\partial u}{\partial y}$ , using a first order meshFigure 6.21: Difference to the exact solution and values of  $\frac{\partial u}{\partial y}$ , using a second order mesh

values of the partial derivative  $\frac{\partial u}{\partial y}$ . It is clearly visible that the gradient is not constant within the triangles. By a careful inspection one has to accept that the gradient is not continuous across element borders, but the jumps are considerably smaller than for linear elements.

Figure 6.22 shows the errors for the partial derivative  $\frac{\partial u}{\partial y}$  and confirms this observation. For first order elements (Figure 6.22(a)) the gradient is constant within each triangle and thus the maximal error on the triangles is proportional to the size of the triangles. This confirms the convergence of order 1 (i.e.  $\approx ch^1$ ) for the gradients with linear elements. The error for quadratic elements is considerably smaller, for a comparable computational effort.

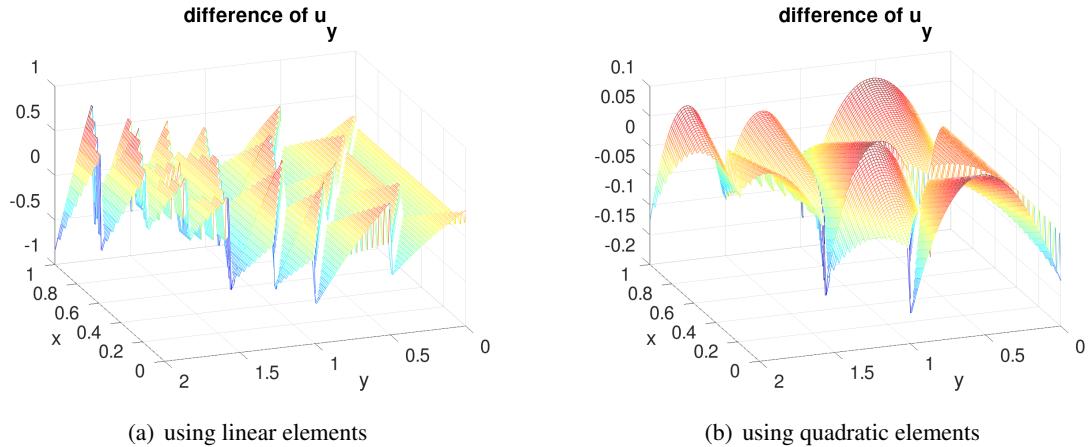


Figure 6.22: Difference of the approximate values of  $\frac{\partial u}{\partial y}$  to the exact values

#### 6.6.4 Remaining Pieces for a Complete FEM Algorithm, the FEMoctave Package

With the above algorithms and codes we can construct the element stiffness matrix and the vector contribution for a triangular element with second order polynomials as basis functions. Thus we can take advantage of the convergence result in Theorem 6–11 on page 427. The missing parts for a complete algorithm are:

- Examine the integrals over the boundary edges in a similar fashion. This poses no major technical problem.
- Assemble the global stiffness matrix and vector, similar to the method in Section 6.2.6, page 408.
- Solve the resulting system of linear systems, using either a direct method or an iterative approach. Use the methods from Chapter 2.
- Visualize the results.

In 2020 I wrote a FEM code in *Octave*, implementing all of the above. Find the documentation with description of the algorithms and sample codes at [web.sha1.bfh.science/FEMoctave/FEMdoc.pdf](https://web.sha1.bfh.science/FEMoctave/FEMdoc.pdf). The package is hosted on GitHub at [github.com/AndreasStahel/FEMoctave](https://github.com/AndreasStahel/FEMoctave). Within *Octave* use `pkg install https://github.com/AndreasStahel/FEMoctave/archive/v2.0.9.tar.gz` to download and install the package and then `pkg load femoctave` to load the package. For Linux systems the complete package is on my web page at [web.sha1.bfh.science/FEMoctave2.0.9.tgz](https://web.sha1.bfh.science/FEMoctave2.0.9.tgz). The source code, demos and examples for FEMoctave is also available in the directory [web.sha1.bfh.science/FEMoctave](https://web.sha1.bfh.science/FEMoctave).

## 6.7 Applying the FEM to Other Types of Problems, e.g. Plane Elasticity

In the previous section approximate solutions of the boundary value problem

$$\begin{aligned} -\nabla \cdot (a \nabla u + u \vec{b}) + b_0 u &= -f && \text{for } (x, y) \in \Omega \\ u &= g_1 && \text{for } (x, y) \in \Gamma_1 \\ \vec{n} \cdot (a \nabla u + u \vec{b}) &= g_2 + g_3 u && \text{for } (x, y) \in \Gamma_2 \end{aligned}$$

are generated, either as weak solutions or, if  $\vec{b} = \vec{0}$ , as minimizers of the functional

$$F(u) = \iint_{\Omega} \frac{1}{2} a (\nabla u)^2 + \frac{1}{2} b_0 u^2 + f \cdot u \, dA$$

among all functions  $u$  with  $u = g_1$  on the boundary section  $\Gamma_1$ . By examining Table 5.2 on page 316 verify that the above setup covers a wide variety of applications. With a standard finite difference approximation of the time derivative many dynamic problems can be solved too.

According to equation (5.29) on page 389 a plane strain problem can be examined as minimizer of the functional

$$\begin{aligned} U(\vec{u}) &= \iint_{\Omega} \frac{1}{2} \frac{E}{(1+\nu)(1-2\nu)} \left\langle \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 2(1-2\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy - \\ &\quad - \iint_{\Omega} \vec{f} \cdot \vec{u} \, dx \, dy - \oint_{\partial\Omega} \vec{g} \cdot \vec{u} \, ds \end{aligned}$$

where the strain tensor  $\varepsilon$  depends on the displacements  $\vec{u}$  through

$$\varepsilon_{xx} = \frac{\partial u_1}{\partial x}, \quad \varepsilon_{yy} = \frac{\partial u_2}{\partial y} \quad \text{and} \quad \varepsilon_{yx} = \frac{1}{2} \left( \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \right).$$

For a plane stress problem the total energy is given by (5.32) on page 395

$$\begin{aligned} U(\vec{u}) &= U_{elast} + U_{Vol} + U_{Surf} \\ &= \iint_{\Omega} \frac{1}{2} \frac{E}{(1-\nu^2)} \left\langle \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 2(1-\nu) \end{bmatrix} \cdot \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix}, \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{pmatrix} \right\rangle dx dy - \\ &\quad - \iint_{\Omega} \vec{f} \cdot \vec{u} \, dx \, dy - \oint_{\partial\Omega} \vec{g} \cdot \vec{u} \, ds. \end{aligned}$$

Thus all contributions to the above elastic energy functionals are of the same type as the integrals in the previous sections. Thus identical techniques are used to develop FEM code for elasticity problems. The code in FEMoctave can solve plane stress and plane strain problems, and in addition axially symmetric problems. The convergence results in Section 6.4 also apply. The role of Poincaré's inequality is taken over by Korn's inequality.

## 6.8 Using Quadrilateral Elements

The presentation is mostly based on [TongRoss08]. Find additional information in [Hugh87, §3.2].

not in class

### 6.8.1 First Order Quadrilateral Elements

A quadrilateral domain in the  $(x, y)$ -plane and its standard square in the  $(\xi, \nu)$  plane with  $-1 \leq \xi, \nu \leq +1$  is shown in Figure 6.23. Assume that all interior angles of this quadrilateral are smaller than  $180^\circ$ . The transformation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \vec{F}(\xi, \nu) = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \frac{\xi+1}{2} \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \frac{\nu+1}{2} \begin{pmatrix} x_4 - x_1 \\ y_4 - y_1 \end{pmatrix} + \frac{(\xi+1)(\nu+1)}{4} \begin{pmatrix} x_3 - x_2 - x_4 + x_1 \\ y_3 - y_2 - y_4 + y_1 \end{pmatrix} \quad (6.15)$$

is a bilinear mapping, i.e. it is linear (resp. affine) with respect to  $\xi$  and  $\nu$  separately. If the quadrilateral is a parallelogram, then the contribution with the factor  $\frac{1}{4}(\xi+1)(\nu+1)$  vanishes and we have the same transformation formula as for triangles. The above transformation is built with the corner at  $(x_1, y_1)$  as starting point and the vectors in directions of  $(x_2, y_2)$  and  $(x_4, y_4)$ . One can also build on the central point  $\vec{c}$  and the directional vectors  $\vec{d}_\xi$  and  $\vec{d}_\nu$ , given by

$$\vec{c} = \frac{1}{4} \sum_{i=1}^4 \begin{pmatrix} x_i \\ y_i \end{pmatrix}, \quad \vec{d}_\xi = \frac{1}{2} \begin{pmatrix} x_2 + x_3 \\ y_2 + y_3 \end{pmatrix} - \vec{c}, \quad \vec{d}_\nu = \frac{1}{2} \begin{pmatrix} x_3 + x_4 \\ y_3 + y_4 \end{pmatrix} - \vec{c} .$$

If the quadrilateral is a parallelogram then the midpoints of the opposite corners coincide, thus the vector

$$\vec{d}_{\xi\nu} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - \begin{pmatrix} x_4 \\ y_4 \end{pmatrix}$$

vanishes for parallelograms. The vector  $\vec{d}_{\xi\nu}$  indicates the deviation from a parallelogram and equation (6.15) is identical to

$$\begin{aligned} \vec{F}(\xi, \nu) &= \vec{c} + \xi \vec{d}_\xi + \nu \vec{d}_\nu + \frac{\xi \nu}{4} \vec{d}_{\xi\nu} \\ &= \vec{c} + \frac{\xi}{4} \begin{pmatrix} -x_1 + x_2 + x_3 - x_4 \\ -y_1 + y_2 + y_3 - y_4 \end{pmatrix} + \frac{\nu}{4} \begin{pmatrix} -x_1 - x_2 + x_3 + x_4 \\ -y_1 - y_2 + y_3 + y_4 \end{pmatrix} + \\ &\quad + \frac{\xi \nu}{4} \begin{pmatrix} +x_1 - x_2 + x_3 - x_4 \\ +y_1 - y_2 + y_3 - y_4 \end{pmatrix} \end{aligned} \quad (6.16)$$

Observe that

$$\vec{F}(-1, -1) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \vec{F}(+1, -1) = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}, \quad \vec{F}(-1, +1) = \begin{pmatrix} x_4 \\ y_4 \end{pmatrix}, \quad \vec{F}(+1, +1) = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix}$$

and the Jacobi matrix  $\mathbf{T}(\xi, \nu)$ , given by

$$\begin{aligned} \mathbf{T}(\xi, \nu) &= \begin{bmatrix} \frac{\partial F_1}{\partial \xi} & \frac{\partial F_1}{\partial \nu} \\ \frac{\partial F_2}{\partial \xi} & \frac{\partial F_2}{\partial \nu} \end{bmatrix} = \begin{bmatrix} \frac{x_2 - x_1}{2} & \frac{x_4 - x_1}{2} \\ \frac{y_2 - y_1}{2} & \frac{y_4 - y_1}{2} \end{bmatrix} + \begin{bmatrix} \frac{(\nu+1)(x_3 - x_2 - x_4 + x_1)}{4} & \frac{(\xi+1)(x_3 - x_2 - x_4 + x_1)}{4} \\ \frac{(\nu+1)(y_3 - y_2 - y_4 + y_1)}{4} & \frac{(\xi+1)(y_3 - y_2 - y_4 + y_1)}{4} \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} x_2 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_4 - y_1 \end{bmatrix} + \frac{1}{4} \begin{bmatrix} x_3 - x_2 - x_4 + x_1 \\ y_3 - y_2 - y_4 + y_1 \end{bmatrix} \cdot \begin{bmatrix} \nu + 1 & \xi + 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{-x_1 + x_2 + x_3 - x_4}{4} & \frac{-x_1 - x_2 + x_3 + x_4}{4} \\ \frac{-y_1 + y_2 + y_3 - y_4}{4} & \frac{-y_1 - y_2 + y_3 + y_4}{4} \end{bmatrix} + \begin{bmatrix} \frac{+x_1 - x_2 + x_3 - x_4}{4} \\ \frac{+y_1 - y_2 + y_3 - y_4}{4} \end{bmatrix} \cdot \begin{bmatrix} \nu & \xi \end{bmatrix}, \end{aligned}$$

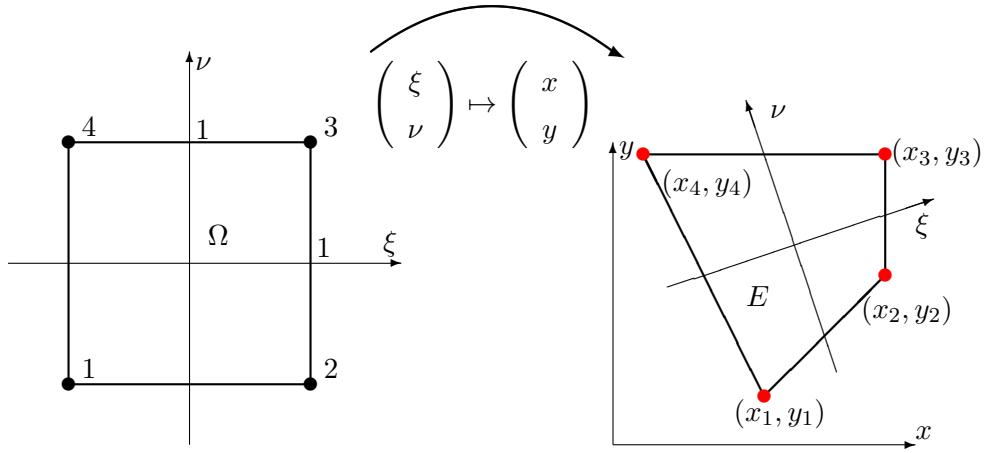


Figure 6.23: The transformation of a linear quadrilateral element and the four nodes

is not constant, which is the case for triangular elements. This has to be taken into account when integrating over the general quadrilateral  $E$  by using the integration formula (6.11) on page 430.

Along each of the edges of the standard square  $\Omega \subset \mathbb{R}^2$  one of  $\xi$  or  $\nu$  is constant and thus the transformation  $\vec{F}$  leads to straight lines as borders of the domain  $E = \vec{F}(\Omega) \in \mathbb{R}^2$ . Using the functions 1,  $\xi$ ,  $\nu$  and  $\xi\nu$  construct the four basis functions with the key property

$$\Phi_i(\xi_j, \nu_j) = \delta_{i,j} = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{for } i \neq j \end{cases} .$$

They are given by the bilinear functions<sup>15</sup>

$$\begin{aligned} \Phi_1(\xi, \nu) &= \frac{1}{4}(1 - \xi)(1 - \nu), & \Phi_2(\xi, \nu) &= \frac{1}{4}(1 + \xi)(1 - \nu), \\ \Phi_3(\xi, \nu) &= \frac{1}{4}(1 + \xi)(1 + \nu), & \Phi_4(\xi, \nu) &= \frac{1}{4}(1 - \xi)(1 + \nu) \end{aligned}$$

or shorter

$$\Phi_i(\xi, \nu) = \frac{1}{4}(1 + \xi_i \xi)(1 + \nu_i \nu) \quad \text{for } i = 1, 2, 3, 4,$$

where  $\xi_i, \nu_i = \pm 1$  are the coordinates of the corners, e.g.  $\xi_1 = \nu_1 = -1$  or  $\xi_3 = \nu_3 = +1$ .

- It is a matter of tedious algebra to verify that for any bilinear function  $\vec{F}(\xi, \nu)$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \vec{F}(\xi, \nu) = \sum_{i=1}^4 \begin{pmatrix} x_i \\ y_i \end{pmatrix} \Phi_i(\xi, \nu).$$

- Any bilinear function  $f(\xi, \nu) = c_1 + c_2 \xi + c_3 \nu + c_4 \xi \nu$  can be written as a linear combination of the  $\Phi_i(\xi, \nu)$ , i.e.

$$f(\xi, \nu) = \sum_{i=1}^4 f(\xi_i, \nu_i) \Phi_i(\xi, \nu) = \sum_{i=1}^4 f_i \Phi_i(\xi, \nu).$$

<sup>15</sup>The function is linear with respect to each of the arguments  $\xi$  and  $\nu$ , but not linear overall, caused by the contribution  $\xi \nu$ .

This is a bilinear interpolation on the standard square  $\Omega$ . To verify this examine<sup>16</sup> the linear system

$$\frac{1}{4} \begin{bmatrix} +1 & +1 & +1 & +1 \\ -1 & +1 & +1 & -1 \\ -1 & -1 & +1 & +1 \\ +1 & -1 & +1 & -1 \end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}$$

and observe that the matrix is invertible. Thus given the values of  $c_i$  we can solve uniquely for  $f_i$ . Let  $\mathbf{M}$  be the above matrix, then  $\mathbf{M} \cdot \mathbf{M}^T = \frac{1}{4} \mathbb{I}$ , i.e.  $2 \mathbf{M}$  is a unitary matrix. This leads to  $\mathbf{M}^{-1} = 4 \mathbf{M}^T$ .

- With these shape functions implement an interpolation on the general quadrilateral  $E$  in Figure 6.23 by using the values of the function at the nodes. This leads to the values when pulled back to the standard square  $\Omega$ .
- Along each of the edges these basis functions are linear functions, since one of the variables  $\xi$  or  $\nu$  is constant. This leads to conforming elements, i.e. the values of the functions will be continuous across element boundaries. Observe that these functions are not linear when displayed on a general quadrilateral, see Figure 6.24. There is no easy formula for this bilinear interpolation on a general quadrilateral.

Based on this the general approximation results are applicable and similar to Theorem 6–9 obtain for an FEM algorithm based in this interpolation the error estimates

$$\|u_h - u_0\|_V \leq C h \quad \text{and} \quad \|u_h - u_0\|_2 \leq C_1 h^2$$

for some constants  $C$  and  $C_1$  independent on  $h$ . A possible formulation is

- $u_h$  converges to  $u_0$  with an error proportional to  $h^2$  as  $h \rightarrow 0$ .
- $\nabla u_h$  converges to  $\nabla u_0$  with an error proportional to  $h$  as  $h \rightarrow 0$ .

To evaluate integrals over the domain  $E \subset \mathbb{R}^2$  use the transformation rule (6.11), i.e.

$$\iint_E f \, dA = \iint_{\Omega} f(\vec{x}(\xi, \nu)) \left| \det \frac{\partial(x, y)}{\partial(\xi, \nu)} \right| \, d\xi \, d\nu = \int_{-1}^{+1} \left( \int_{-1}^{+1} f(\vec{x}(\xi, \nu)) |\det \mathbf{T}(\xi, \nu)| \, d\xi \right) d\nu.$$

Observe that the term  $|\det \mathbf{T}(\xi, \nu)|$  is not constant. The idea of Gauss integration can be applied to squares, using the 1D integrals<sup>17</sup>

$$\int_{-1}^{+1} f(t) \, dt \approx \sum_j w_j f(t_j) \quad \text{general formula}$$

---

<sup>16</sup>Compare the coefficients of  $1, \xi, \nu$  and  $\xi\nu$ .

$$\begin{aligned} 4 \sum_{i=1}^4 f_i \Phi_i(\xi, \nu) &= f_1(1-\xi)(1-\nu) + f_2(1+\xi)(1-\nu) + f_3(1+\xi)(1+\nu) + f_4(1-\xi)(1+\nu) \\ &= (f_1 + f_2 + f_3 + f_4) + (-f_1 + f_2 + f_3 - f_4)\xi + (-f_1 - f_2 + f_3 + f_4)\nu + (+f_1 - f_2 + f_3 - f_4)\xi\nu \end{aligned}$$

<sup>17</sup>To derive the first formula integrate  $1, t$  and  $t^2$ .

$$\begin{aligned} \int_{-1}^{+1} f(t) \, dt &= w_1 f(-\xi) + w_1 f(+\xi) \\ \int_{-1}^{+1} 1 \, dt &= w_1 1 + w_1 1 \implies w_1 = 1 \\ \int_{-1}^{+1} t \, dt &= -w_1 \xi + w_1 \xi = 0 \\ \int_{-1}^{+1} t^2 \, dt &= \frac{2}{3} = +w_1 \xi^2 + w_1 \xi^2 \implies \xi = \sqrt{1/3} \\ \int_{-1}^{+1} t^3 \, dt &= 0 = -w_1 \xi^3 + w_1 \xi^3 = 0 \end{aligned}$$

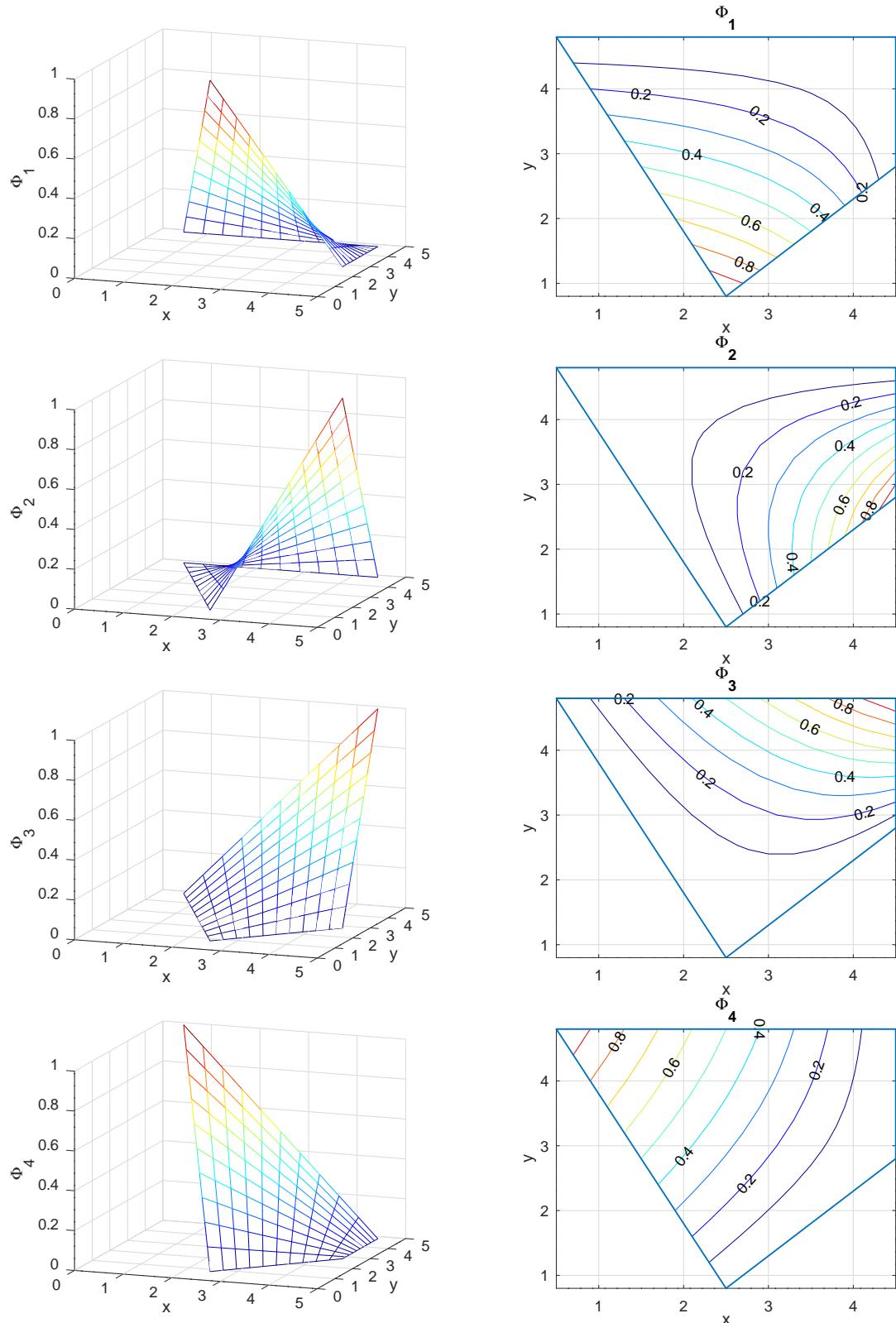


Figure 6.24: Bilinear shape functions on a 4 node quadrilateral element

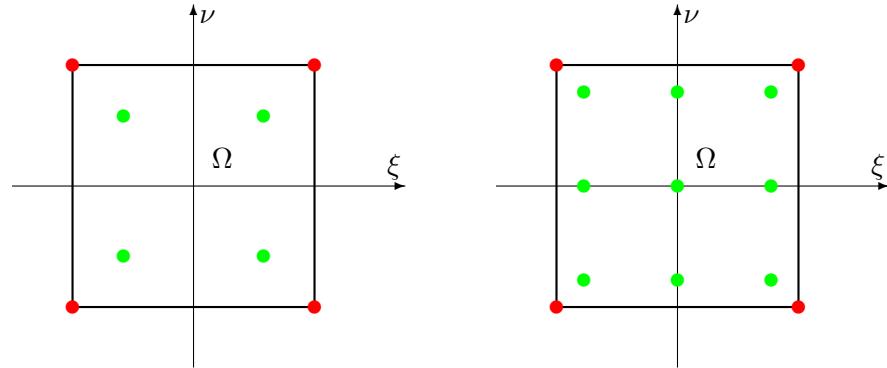


Figure 6.25: Gauss integration points on the standard square, using  $2^2 = 4$  or  $3^2 = 9$  integration points

$$\int_{-1}^{+1} f(t) dt \approx 1 f\left(\frac{-1}{\sqrt{3}}\right) + 1 f\left(\frac{+1}{\sqrt{3}}\right) \quad \text{2 point Gauss integration}$$

$$\int_{-1}^{+1} f(t) dt \approx \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(+\sqrt{\frac{3}{5}}\right) \quad \text{3 point Gauss integration}$$

For the standard square  $\Omega = [-1, +1] \times [-1, +1] \subset \mathbb{R}^2$  use

$$\iint_{\Omega} f(\xi, \nu) d\xi d\nu = \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \nu) d\xi d\nu \approx \int_{-1}^{+1} \sum_j w_j f(\xi_j, \nu) d\nu \approx \sum_i \sum_j w_j w_i f(\xi_j, \nu_i).$$

As an example for the result based on the 2 point Gauss formula use  $\alpha = \sqrt{1/3}$  and find

$$\iint_{\Omega} f dA \approx 1 \cdot f(-\alpha, -\alpha) + 1 \cdot f(+\alpha, -\alpha) + 1 \cdot f(+\alpha, +\alpha) + 1 \cdot f(-\alpha, +\alpha).$$

This leads to the 2D Gauss integration points shown on the left in Figure 6.25. If the 2D integration is based on the 3 point formula for the integration on  $[-1, +1]$  the 2D situation is shown on the right in Figure 6.25. There are more integration schemes of the same type, e.g. see [TongRoss08, §6.5.3], [Hugh87, §3.8] or [Li21, Tables 3.3, 5.4]. With this we have all the tools to apply the ideas from Section 6.5 to construct the matrices and vectors required for an FEM algorithm, based on first order quadrilateral elements with four nodes.

Thus  $t^4$  is not integrated exactly and the error is proportional to  $h^4$ . To derive the second formula use

$$\begin{aligned} \int_{-1}^{+1} f(t) dt &= w_1 f(-\xi) + w_0 f(0) + w_1 f(+\xi) \\ \int_{-1}^{+1} 1 dt &= w_1 1 + w_0 1 + w_1 1 \\ \int_{-1}^{+1} t dt &= -w_1 \xi + w_0 0 + w_1 \xi = 0 \\ \int_{-1}^{+1} t^2 dt &= \frac{2}{3} = +w_1 \xi^2 + w_1 \xi^2 \\ \int_{-1}^{+1} t^3 dt &= 0 = -w_1 \xi^3 + w_1 \xi^3 = 0 \\ \int_{-1}^{+1} t^4 dt &= \frac{2}{5} = +w_1 \xi^4 + w_1 \xi^4 \\ \int_{-1}^{+1} t^5 dt &= 0 = -w_1 \xi^5 + w_1 \xi^5 = 0 \end{aligned}$$

Thus  $t^6$  is not integrated exactly and the error is proportional to  $h^6$ . The system to be solved is

$$\left\{ \begin{array}{lcl} w_0 + 2w_1 & = & 2 \\ 2w_1 \xi^2 & = & \frac{2}{3} \\ 2w_1 \xi^4 & = & \frac{2}{5} \end{array} \right. \implies \xi^2 = \frac{3}{5}, w_1 = \frac{5}{9}, w_0 = \frac{8}{9} .$$

## 6.8.2 Second Order Quadrilateral Elements

On the above quadrilateral domain a second order element can be constructed, using the midpoints of the sides as additional nodes, see Figure 6.26. The transformation is identical to the four node element, i.e. equation (6.15) or (6.16). Using the eight functions  $1, \xi, \nu, \xi^2, \nu^2, \xi\nu, \xi^2\nu$  and  $\xi\nu^2$  construct the eight basis functions

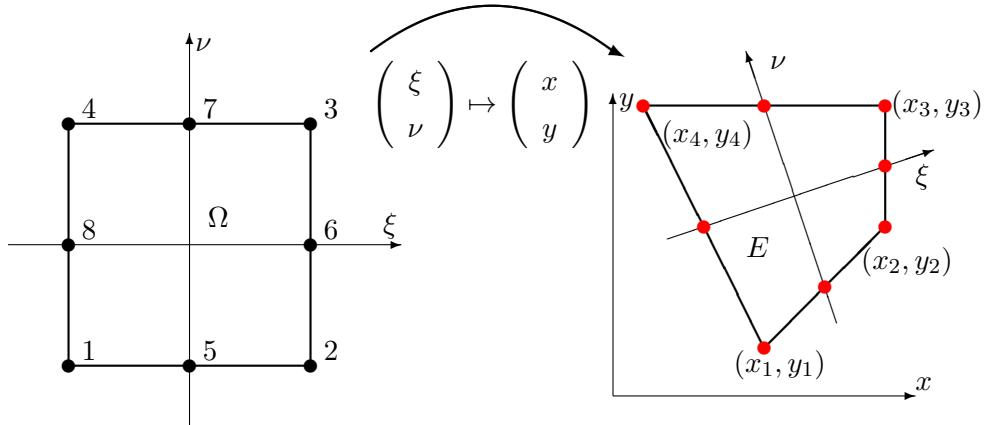


Figure 6.26: The transformation of a second order quadrilateral element and the eight nodes

$$\begin{aligned}\Phi_i(\xi, \nu) &= \frac{1}{4} (1 - \xi_i \xi) (1 - \nu_i \nu) (\xi_i \xi + \nu_i \nu - 1) \quad \text{for } i = 1, 2, 3, 4 \\ \Phi_i(\xi, \nu) &= \frac{1}{2} (1 - \xi^2) (1 + \nu_i \nu) \quad \text{for } i = 5, 7 \\ \Phi_i(\xi, \nu) &= \frac{1}{2} (1 + \xi_i \xi) (1 - \nu^2) \quad \text{for } i = 6, 8\end{aligned}$$

- The above shape functions are all of the form

$$f(\xi, \nu) = c_1 + c_2 \xi + c_3 \nu + c_4 \xi^2 + c_5 \nu^2 + c_6 \xi \nu + c_7 \xi^2 \nu + c_8 \xi \nu^2,$$

i.e. these are biquadratic<sup>18</sup> functions. Any function of this form can be written as a linear combination of the  $\Phi_i(\xi, \nu)$ , i.e.

$$f(\xi, \nu) = \sum_{i=1}^8 f(\xi_i, \nu_i) \Phi_i(\xi, \nu) = \sum_{i=1}^8 f_i \Phi_i(\xi, \nu)$$

- Along each of the edges these basis functions are quadratic functions, since one of the variables  $\xi$  or  $\nu$  is constant. This leads to conforming elements, i.e. the values of the functions will be continuous across element boundaries. Find the contour lines of the shape functions on a general quadrilateral in Figure 6.27. There is no easy formula for this biquadratic interpolation on a general quadrilateral.

Based on this the general approximation results are applicable and similar to Theorem 6–11 obtain for an FEM algorithm based in this interpolation the error estimates

$$\|u_h - u_0\|_V \leq C h^2 \quad \text{and} \quad \|u_h - u_0\|_2 \leq C_1 h^3$$

for some constants  $C$  and  $C_1$  independent on  $h$ . We may say that

- $u_h$  converges to  $u_0$  with an error proportional to  $h^3$  as  $h \rightarrow 0$ .

<sup>18</sup>The functions are quadratic with respect to each of the arguments  $\xi$  and  $\nu$ , but not quadratic overall, caused by the contributions  $\xi^2 \nu$  and  $\xi \nu^2$ .

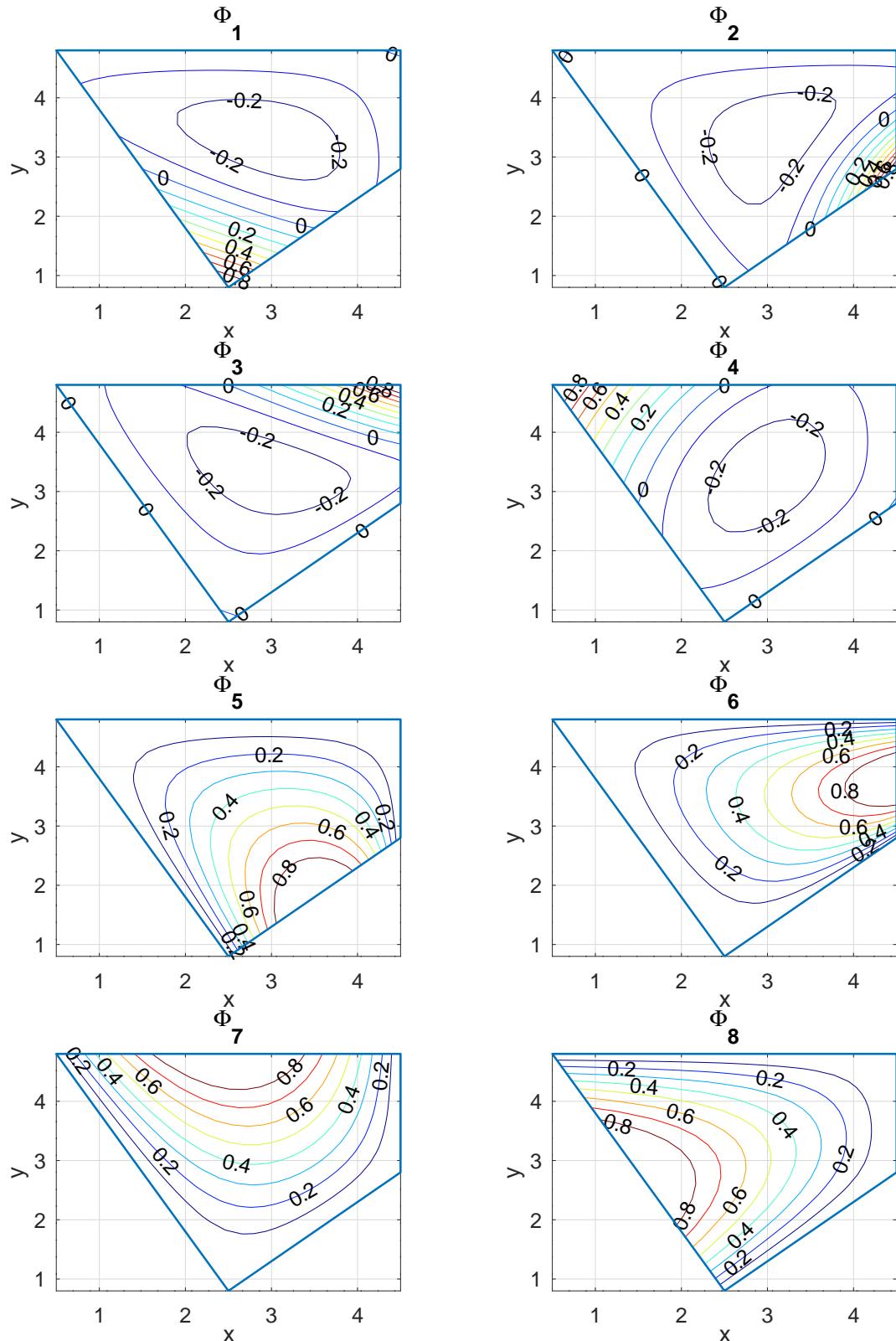


Figure 6.27: Contour levels of the shape functions on a 8 node quadrilateral element

- $\nabla u_h$  converges to  $\nabla u_0$  with an error proportional to  $h^2$  as  $h \rightarrow 0$ .

The numerical integration schemes for linear quadrilateral elements can be used for second order elements too. With this we have all the tools to apply the ideas from Section 6.5 to construct the matrices and vectors required for an FEM algorithm, based on second order quadrilateral elements with eight nodes.

The above is the basis to implement a FEM algorithm based on quadrilateral elements.

## 6.9 An Application of FEM to a Tumor Growth Model

### 6.9.1 Introduction

The goal is to examine the growth of tumor cells in healthy tissue, i.e. find a simple mathematical model to describe this situation.

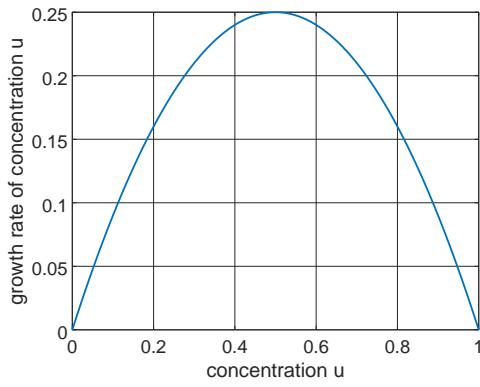
Let  $0 \leq u(t, \vec{x}) \leq 1$  describe the concentration of tumor cells, i.e.  $u = 0$ : no tumor and  $u = 1$  only tumor. There are two effects contribution to the tumor growth and spreading.

- growth: assume that the growth rate of the tumor is given by the ordinary differential equation

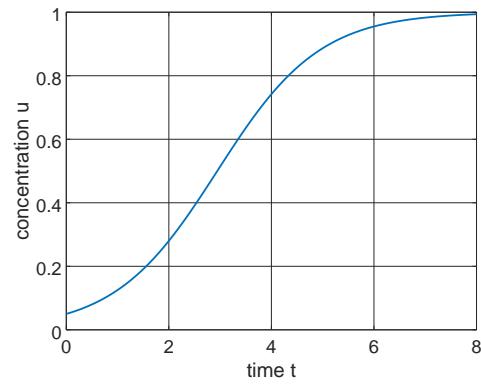
$$\dot{u}(t) = \alpha \cdot f(u(t)) = \alpha \cdot u(t) \cdot (1 - u(t))$$

where  $\alpha > 0$  is a parameter. This is called a logistic growth model. Find the graph of the function for  $\alpha = 1$  and the solution of the corresponding logistic differential equation  $\frac{d}{dt} u(t) = u(t) \cdot (1 - u(t))$  in Figure 6.28.

- diffusion: the tumor cells will also spread out, just like heat spreads in a medium. Thus describe this effect by a heat equation.



(a) the function for logistic growth



(b) solution of the logistic differential equation

Figure 6.28: The function leading to logistic growth and the solution of the differential equation

The above two effects lead to the partial differential equation

$$\frac{d}{dt} u(t, \vec{x}) = \Delta u(t, \vec{x}) + \alpha f(u(t, \vec{x})). \quad (6.17)$$

To get a first impression it is a good idea to assume radial symmetry, i.e. the function  $u(t, \vec{x})$  depends on the radius  $r = \|\vec{x}\|$  only. For this we express the Laplace operator  $\Delta u$  in spherical coordinates, i.e. for functions depending on the radius  $r$  only.

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \frac{1}{r^2} \frac{\partial}{\partial r} \left( r^2 \frac{\partial u}{\partial r} \right).$$

Now equation (6.17) takes the form

$$\frac{d}{dt} u(t, r) = \frac{1}{r^2} \frac{\partial}{\partial r} \left( r^2 \frac{\partial u(t, r)}{\partial r} \right) + \alpha f(u(t, r))$$

or after a multiplication by  $r^2$

$$r^2 \frac{d}{dt} u(t, r) = \frac{\partial}{\partial r} \left( r^2 \frac{\partial u(t, r)}{\partial r} \right) + r^2 \alpha f(u(t, r)). \quad (6.18)$$

This has to be supplemented with appropriate initial and boundary values. The goal is to examine the behavior of solutions of this partial differential equation, using finite elements.

### 6.9.2 The Finite Element Method Applied to 1D Problems

First examine steady state boundary value problems. For given functions  $a(r) > 0$ ,  $b(r)$  and  $f(r)$  we examine a boundary value problem of the form<sup>19</sup>

$$(a(r) u'(r))' + b(r) f(r) = 0 \quad (6.19)$$

with some boundary conditions. Multiplying (6.19) by a smooth test function  $\phi(r)$  and an integration by parts leads to

$$\begin{aligned} 0 &= \int_0^R \left( (a(r) u'(r))' + b(r) f(r) \right) \phi(r) dr \\ &= a(r) u'(r) \phi(r) \Big|_{r=0}^R + \int_0^R -a(r) u'(r) \phi'(r) + b(r) f(r) \phi(r) dr. \end{aligned} \quad (6.20)$$

Using FEM this equation will be discretized, leading to the stiffness matrix  $\mathbf{A}$  and the weight matrix  $\mathbf{M}$ , such that  $\langle \mathbf{A}\vec{u} - \mathbf{M}\vec{f}, \vec{\phi} \rangle = 0$  for all vectors  $\vec{\phi}$ . This then leads to the linear system  $\mathbf{A}\vec{u} = \mathbf{M}\vec{f}$  to be solved for the vector  $\vec{u}$ .

The section below will lead to a numerical implementation of the above idea. Then the developed MATLAB/Octave code will be tested with the help of a few example problems.

#### Interpolation and Gauss integration

To generate a finite element formulation first examine an interval  $r_i \leq r \leq r_{i+1}$ . For given coefficient functions  $a(r)$ ,  $b(r)$  and  $f(r)$  and the values of the functions  $u(r)$  and  $\phi(r)$  given at the three nodes at  $r = r_i$ ,  $\frac{r_i+r_{i+1}}{2}$  and  $r_{i+1}$  use a quadratic interpolation to construct the functions  $u(r)$  and  $\phi(r)$  on the interval. Then integrate

$$I_0 = \int_{r_i}^{r_{i+1}} b(r) f(r) \phi(r) dr \quad \text{and} \quad I_1 = \int_{r_i}^{r_{i+1}} a(r) u'(r) \phi'(r) dr.$$

To compute these integrals first examine the very efficient Gauss integration on a standard interval  $[-\frac{h}{2}, \frac{h}{2}]$  of length  $h$ .

- On the interval  $-\frac{h}{2} \leq x \leq +\frac{h}{2}$  the Gauss integration formula is given by

$$\int_{-h/2}^{h/2} u(x) dx \approx \frac{h}{18} \left( 5u(-\sqrt{\frac{3}{5}}\frac{h}{2}) + 8u(0) + 5u(+\sqrt{\frac{3}{5}}\frac{h}{2}) \right). \quad (6.21)$$

---

<sup>19</sup>At this stage using two functions  $f(r)$  and  $b(r)$  seem to be overkill, but this will turn out to be useful when solving the dynamic problem in Section 6.9.4.

- The three values of a function  $u(x)$  at  $u(-h/2) = u_-$ ,  $u(0) = u_0$  and  $u(h/2) = u_+$  determine a quadratic interpolating polynomial<sup>20</sup>

$$u(x) = u_0 + \frac{u_+ - u_-}{h} x + \frac{u_+ - 2u_0 + u_-}{h^2} 2x^2.$$

Use  $x = 0$  and  $x = \pm\sqrt{\frac{3}{5}}\frac{h}{2}$  to determine the values of  $u(x)$  at the Gauss points by

$$\begin{aligned} \begin{pmatrix} u(-\sqrt{\frac{3}{5}}\frac{h}{2}) \\ u(0) \\ u(+\sqrt{\frac{3}{5}}\frac{h}{2}) \end{pmatrix} &= \begin{bmatrix} \frac{3}{10} + \frac{\sqrt{\frac{3}{5}}}{2} & \frac{4}{10} & \frac{3}{10} - \frac{\sqrt{\frac{3}{5}}}{2} \\ 0 & 1 & 0 \\ \frac{3}{10} - \frac{\sqrt{\frac{3}{5}}}{2} & \frac{4}{10} & \frac{3}{10} + \frac{\sqrt{\frac{3}{5}}}{2} \end{bmatrix} \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix} \\ &= \frac{1}{10} \begin{bmatrix} 3 + \sqrt{15} & 4 & 3 - \sqrt{15} \\ 0 & 10 & 0 \\ 3 - \sqrt{15} & 4 & 3 + \sqrt{15} \end{bmatrix} \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix} = \mathbf{G}_0 \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix}. \end{aligned}$$

Use this Gaussian interpolation matrix to compute the values of the function at the Gauss integration points, using the values at the nodes.

- The above can be repeated to obtain the values of the derivative  $u'(x)$  at the Gauss points.

$$\begin{pmatrix} u'(-\sqrt{\frac{3}{5}}\frac{h}{2}) \\ u'(0) \\ u'(+\sqrt{\frac{3}{5}}\frac{h}{2}) \end{pmatrix} = \frac{1}{h} \begin{bmatrix} -1 - 2\sqrt{\frac{3}{5}} & +4\sqrt{\frac{3}{5}} & +1 - 2\sqrt{\frac{3}{5}} \\ -1 & 0 & +1 \\ -1 + 2\sqrt{\frac{3}{5}} & -4\sqrt{\frac{3}{5}} & +1 + 2\sqrt{\frac{3}{5}} \end{bmatrix} \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix} = \frac{1}{h} \mathbf{G}_1 \cdot \begin{pmatrix} u_- \\ u_0 \\ u_+ \end{pmatrix}.$$

- Define a weight matrix  $\mathbf{W}$  by

$$\mathbf{W} = \text{diag}\left(\left[\frac{5}{18}, \frac{8}{18}, \frac{5}{18}\right]\right) = \begin{bmatrix} \frac{5}{18} & 0 & 0 \\ 0 & \frac{8}{18} & 0 \\ 0 & 0 & \frac{5}{18} \end{bmatrix}$$

and then rewrite (6.21) in the form

$$\int_{-h/2}^{h/2} u(x) dx \approx \frac{h}{18} \left( 5u(-\sqrt{\frac{3}{5}}\frac{h}{2}) + 8u(0) + 5u(+\sqrt{\frac{3}{5}}\frac{h}{2}) \right) = h \sum_{i=1}^3 (\mathbf{W} \mathbf{G}_0 \vec{u})_i$$

where the vector  $\vec{u}$  contains the values of the function at  $\pm\frac{h}{2}$  and 0.

- To evaluate the function  $a(x)$  at the Gauss points use the notation

$$\mathbf{a} = \begin{bmatrix} a(-\sqrt{\frac{3}{5}}\frac{h}{2}) & 0 & 0 \\ 0 & a(0) & 0 \\ 0 & 0 & a(+\sqrt{\frac{3}{5}}\frac{h}{2}) \end{bmatrix}$$

and similarly for the function  $b(r)$ , leading to a diagonal matrix  $\mathbf{b}$ .

---

<sup>20</sup>To verify the formula use  $u(0) = u_0$  and for  $x = \pm h$

$$u(\pm\frac{h}{2}) = u_0 \pm \frac{u_+ - u_-}{h} \frac{h}{2} + \frac{u_+ - 2u_0 + u_-}{h^2} \frac{2h^2}{4} = u_0 (1 - 1) + u_+ (\pm\frac{1}{2} + \frac{1}{2}) - u_- (\pm\frac{1}{2} - \frac{1}{2}).$$

The above notation leads to the required integrals. With  $\Delta r_i = r_{i+1} - r_i$  obtain

$$\begin{aligned} I_0 &= \int_{r_i}^{r_{i+1}} b(r) f(r) \phi(r) dr \approx \Delta r_i \langle \mathbf{W} \mathbf{b} \mathbf{G}_0 \vec{f}, \mathbf{G}_0 \vec{\phi} \rangle = \Delta r_i \langle \mathbf{G}_0^T \mathbf{W} \mathbf{b} \mathbf{G}_0 \vec{f}, \vec{\phi} \rangle \\ I_1 &= \int_{r_i}^{r_{i+1}} a(r) u'(r) \phi'(r) dr \approx \frac{\Delta r_i}{(\Delta r_i)^2} \langle \mathbf{W} \mathbf{a} \mathbf{G}_1 \vec{u}, \mathbf{G}_1 \vec{\phi} \rangle = \frac{1}{\Delta r_i} \langle \mathbf{G}_1^T \mathbf{W} \mathbf{a} \mathbf{G}_1 \vec{u}, \vec{\phi} \rangle \end{aligned}$$

Now work on the interval  $[0, R]$ , discretized by  $0 = r_0 < r_1 < r_2 < \dots < r_{n-1} < r_n = R$ . Then examine the discrete version of the weak solution, thus integrals of the type

$$\begin{aligned} I &= \int_0^R (a(r) u'(r))' \phi(r) - b(r) f(r) \phi(r) dr \\ &= \sum_{i=0}^{n-1} \int_{r_i}^{r_{i+1}} (a(r) u'(r))' \phi(r) - b(r) f(r) \phi(r) dr \\ &\approx \sum_{i=0}^{n-1} \left( \frac{1}{\Delta r_i} \langle \mathbf{G}_1^T \mathbf{W} \mathbf{a}_i \mathbf{G}_1 \vec{u}_i, \vec{\phi}_i \rangle - \Delta r_i \langle \mathbf{G}_0^T \mathbf{W} \mathbf{b}_i \mathbf{G}_0 \vec{f}_i, \vec{\phi}_i \rangle \right) \\ &= \langle \mathbf{A} \vec{u} - \mathbf{M} \vec{f}, \vec{\phi} \rangle \quad \text{for all vectors } \vec{\phi}. \end{aligned}$$

The stiffness matrix  $\mathbf{A}$  and the weight matrix  $\mathbf{M}$  are both of size  $(2n + 1) \times (2n + 1)$ , but possible boundary conditions are not taken into account yet<sup>21</sup>. This has to be done with some care, since the differential equation (6.19) has a unique solution only if boundary conditions are specified.

### Simplified matrices for differential equations with constant coefficients

not in class

If the coefficients  $a$  and  $b$  are constant the above formulas can be simplified by using

$$\mathbf{G}_1^T \mathbf{W} \mathbf{G}_1 = \frac{1}{3} \begin{bmatrix} +7 & -8 & +1 \\ -8 & +16 & -8 \\ +1 & -8 & +7 \end{bmatrix}$$

---

<sup>21</sup>We also ignored contributions of the type  $c(r) u'(r)$  or  $h(r) u(r)$  in the differential equation. These contributions can be treated using exactly the same procedures, e.g.

$$\begin{aligned} \int_{r_i}^{r_{i+1}} c(r) u'(r) \phi(r) dr &\approx \frac{\Delta r_i}{\Delta r_i} \langle \mathbf{W} \mathbf{c} \mathbf{G}_1 \vec{u}, \mathbf{G}_0 \vec{\phi} \rangle = \langle \mathbf{G}_0^T \mathbf{W} \mathbf{c} \mathbf{G}_1 \vec{u}, \vec{\phi} \rangle \\ \int_{r_i}^{r_{i+1}} h(r) u(r) \phi(r) dr &\approx \Delta r_i \langle \mathbf{W} \mathbf{h} \mathbf{G}_0 \vec{u}, \mathbf{G}_0 \vec{\phi} \rangle = \Delta r_i \langle \mathbf{G}_0^T \mathbf{W} \mathbf{h} \mathbf{G}_0 \vec{u}, \vec{\phi} \rangle . \end{aligned}$$

Thus it is straightforward to adapt the algorithm and the codes to examine differential equations of the type

$$(a(r) u'(r))' + c(r) u'(r) + h(r) u(r) + b(r) f(r) = 0 .$$

and the integral  $\int_0^R a u'(r) \phi'(r) dr$  leads to a matrix

$$\frac{a}{3\Delta r} \begin{bmatrix} +7 & -8 & +1 \\ -8 & +16 & -8 \\ +1 & -8 & +14 & -8 & +1 \\ & -8 & +16 & -8 \\ +1 & -8 & +14 & -8 & +1 \\ & -8 & +16 & -8 \\ +1 & -8 & +14 & -8 & +1 \\ & \ddots & \ddots & \ddots & \ddots \\ & +1 & -8 & +14 & -8 & +1 \\ & & -8 & +16 & -8 \\ & +1 & -8 & +14 & -8 & +1 \\ & & \ddots & \ddots & \ddots & \ddots \\ & +1 & -8 & +14 & -8 & +1 \\ & & -8 & +16 & -8 \\ & +1 & -8 & +14 & -8 & +1 \\ & & +1 & -8 & 7 \end{bmatrix}. \quad (6.22)$$

Observe that this matrix has a band structure with 3 or 5 elements only arranged along the diagonal. This matrix is symmetric and for  $a > 0$  positive semi-definite. If we have Dirichlet boundary conditions the first (or last) row and column are removed, then the matrix will be strictly positive definite. This corresponds to the analytical observations

$$\int_0^R a |u'(r)|^2 dr \geq 0 \quad \text{and} \quad \int_0^R a |u'(r)|^2 dr = 0 \iff u(r) = \text{const.}$$

Some of the key properties are similar to our model matrix  $\mathbf{A}_n$  from Section 2.3.1.

Similarly using

$$\mathbf{G}_0^T \mathbf{W} \mathbf{G}_0 = \frac{1}{30} \begin{bmatrix} +4 & +2 & -1 \\ +2 & +16 & +2 \\ -1 & +2 & +4 \end{bmatrix}$$

the integral  $\int_0^R b f(r) \phi(r) dr$  leads to

$$\frac{b\Delta r}{30} \begin{bmatrix} +4 & +2 & -1 \\ +2 & +16 & +2 \\ -1 & +2 & +8 & +2 & -1 \\ & +2 & +16 & +2 \\ & -1 & +2 & +8 & +2 & -1 \\ & +2 & +16 & +2 \\ & -1 & +2 & +8 & +2 & -1 \\ & \ddots & \ddots & \ddots & \ddots & \ddots \\ & -1 & +2 & +8 & +2 & -1 \\ & +2 & +16 & +2 \\ & -1 & +2 & +4 \end{bmatrix}. \quad (6.23)$$

### Taking boundary conditions into account

The contribution in (6.20) by boundary terms is

$$a(r) u'(r) \phi(r) \Big|_{r=0}^R = a(R) u'(R) \phi(R) - a(0) u'(0) \phi(0).$$

This leads to different algorithms to take Dirichlet or Neumann conditions into account.

- If the value of  $u(R)$  is known, then  $\phi(R)$  needs to be zero and the contribution vanishes.
  - If  $u(R) = 0$  the last value in the vector  $\vec{u}$  is zero and we can safely remove this zero in  $\vec{u}$  and the last column in the matrix  $\mathbf{A}$ .
  - If  $u(R) \neq 0$  the last value in the vector  $\vec{u}$  equals  $u(R)$  and the last equation can be removed. But there is a contribution to the last few equations. Use the last column of  $\mathbf{A}$  to determine the contribution.
- If we have no constraint on  $u(R)$  the natural boundary condition is  $a(R) u'(R) = 0$ . We do not have to do anything to take this condition into account.
- For boundary conditions of the type  $a(R) u'(r) = c_1 + c_2 u(R)$  the correct type of contribution will have to be added.

This leads to the linear system

$$\mathbf{A} \vec{u} - \mathbf{M} \vec{f} = \vec{0} \quad \text{or} \quad \vec{u} = \mathbf{A}^{-1} \mathbf{M} \vec{f} = \mathbf{A} \setminus \mathbf{M} \vec{f}.$$

The resulting matrix  $\mathbf{A}$  is symmetric and has a band structure with semi-bandwidth 3, i.e. in each row there are up to 5 entries about the diagonal.

### The MATLAB/Octave code

The matrices  $\mathbf{A}$  and  $\mathbf{M}$  generated by the above algorithm depend on the interval and its division in finite elements and the two functions  $a(r)$  and  $b(r)$ . Thus we write code to construct those matrices. In addition the new vector with the nodes will be generated, i.e. the mid points of the sub-intervals are added.

#### GenerateFEM.m

```

function [A,M,xnew] = GenerateFEM(x,a,b)
% for documentation type "help GenerateFEM" or "demo GenerateFEM"
dx = diff(x(:));
n = length(x)-1;
xnew = [x(:)' ; x(:)' + [dx',0]/2]; xnew = xnew(1:end-1); xnew = xnew(:);
A = sparse(2*n+1,2*n+1); M = A;
%% interpolation matrix for the function values
s06 = sqrt(0.6);
G0 = [3/10+s06/2, 4/10, 3/10-s06/2;
       0,      1,      0;
       3/10-s06/2, 4/10, 3/10+s06/2];
%% interpolation matrix for the derivative values
G1 = [-1-2*s06, +4*s06, +1-2*s06;
       -1,      0,      1;
       -1+2*s06, -4*s06, +1+2*s06];
W = diag([5 8 5])/18;
for ind = 1:n
  x_elem = x(ind)+dx(ind)/2*(1+[-s06 0 s06]);
  M_elem = dx(ind)*G0'*W*diag(b(x_elem))*G0;
  A_elem = (G1'*W*diag(a(x_elem))*G1)/dx(ind);
  ra = 2*ind-1:2*ind+1;
  M(ra,ra) = M(ra,ra) + M_elem;
  A(ra,ra) = A(ra,ra) + A_elem;
end%for
end%function

```

### 6.9.3 A Few Examples, Static and Dynamic

#### A first example, as simple as possible

To solve the boundary value problem

$$-u''(r) = 1 \quad \text{on} \quad 0 \leq r \leq 1 \quad \text{with} \quad u(0) = u(1) = 0$$

with the exact solution  $u_{exact}(r) = \frac{1}{2}r(1-r)$  use the code below.

**Test1.m**

run demo

```
N = 10; % number of elements, then 2*N+1 nodes
x = linspace(0,1,N+1);
a = @ (x) 1*ones(size(x)); b = @ (x) 1*ones(size(x)); % solve -u''=1

[A,M,r] = GenerateFEM(x,a,b);
A = A(2:end-1,2:end-1); M = M(2:end-1,:); % Dirichlet BC on both ends
f = ones(size(r(:)));
u_exact = r(:).* (1-r(:))/2;
u = A \ (M*f);

figure(1); plot(r,[0;u;0],r,u_exact)
xlabel('r'); ylabel('u'); title('solution, exact and approximate')

figure(2); plot(r,[0;u;0]-u_exact)
xlabel('r'); ylabel('u');
title('difference of solutions, exact and approximate')
```

It turns out that the generated solution coincides with the exact solution. This is no real surprise, since the exact solution is a quadratic function, which we approximate by a piecewise quadratic function. For real problems this is very unlikely to occur. Thus this example is only useful to verify the algorithm and the coding.

#### A second example, to illustrate superconvergence

To solve the boundary value problem

$$-u''(r) = \cos(3r) \quad \text{on} \quad 0 \leq r \leq \frac{\pi}{2} \quad \text{with} \quad u'(0) = u\left(\frac{\pi}{2}\right) = 0$$

with the exact solution  $u_{exact}(r) = \frac{1}{9}\cos(3r)$  use the code `Test2.m` below. Find the solution in Figure 6.29(a).

**Test2.m**

run demo  
Test2Demo

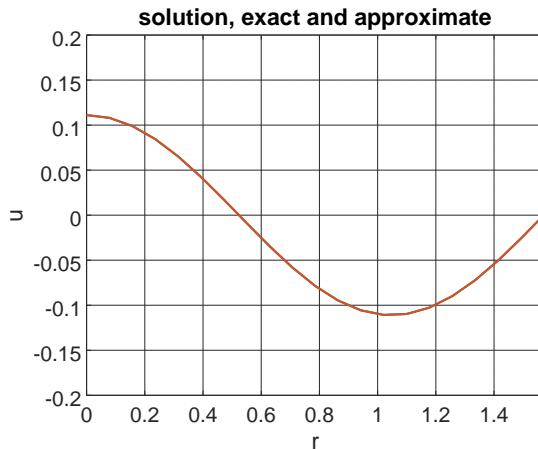
```
N = 2*10; % number of elements
x = linspace(0,pi/2,N+1);
a = @ (x) 1*ones(size(x)); b = @ (x) 1*ones(size(x)); % solve -u''= cos(3*x)
[A,M,r] = GenerateFEM(x,a,b);

A = A(1:end-1,1:end-1); % Neumann on the left and Dirichlet BC on the right
M = M(1:end-1,:);
f = cos(3*x); u_exact = 1/9*cos(3*x);
u = A \ (M*f);
figure(1); plot(r,[u;0],r,u_exact)
xlabel('r'); ylabel('u'); title('solution, exact and approximate')
figure(2); plot(r,[u;0]-u_exact)
xlabel('r'); ylabel('u');
title('difference of exact and approximate solution')
```

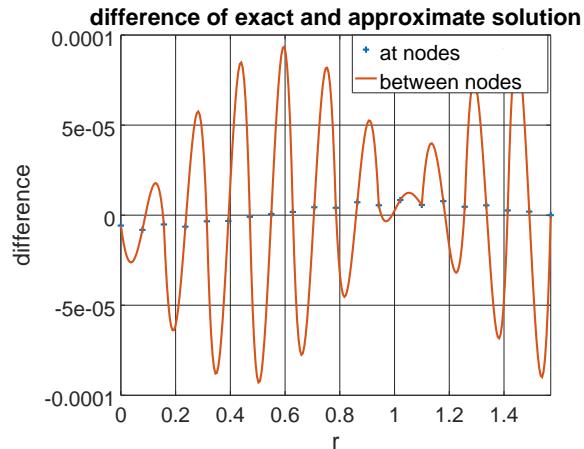
differences	$N = 10$	$N = 20$	ratio	order of convergence
at the nodes	$8.39 \cdot 10^{-6}$	$5.32 \cdot 10^{-7}$	$16 = 2^4$	4
at all points	$9.36 \cdot 10^{-5}$	$1.16 \cdot 10^{-5}$	$8 = 2^3$	3

Table 6.3: Maximal approximation error

By using different values  $N$  for the number of elements observe (see Table 6.3) an order of convergence at the nodes of approximately 4. This is better than to be expected by Theorem 6–11 (page 427). If the solution is reconstructed between the grid points by a piecewise quadratic interpolation one observes a cubic convergence. This is the expected result by the abstract error estimate for a piecewise quadratic approximation. The additional accuracy is caused by the effect of superconvergence, and we can **not** count on it to occur. Figure 6.29(b) shows the error at the nodes and also at points between the nodes<sup>22</sup>. For those we obtain the expected third order of convergence. A closer look at the difference of exact and approximate solution at  $r \approx 1.1$  also reveals that the approximate solution is not twice differentiable, since the slope of the curve has a severe jump. The piecewise quadratic interpolation can (and does) lead to non-continuous derivatives.



(a) the solutions



(b) difference of exact and approximate solution

Figure 6.29: Exact and approximate solution of the second test problem

### A third example, with variable coefficients

The function  $u_{exact}(r) = \exp(-r^2) - \exp(-R^2)$  solves the boundary value problem

$$-(r^2 u'(r))' = r^2 \cdot f(r) \quad \text{on } 0 \leq r \leq R \quad \text{with} \quad u'(0) = u(R) = 0$$

where  $f(r) = (6 - 4r) \exp(-r^2)$ . In this example the coefficient functions  $a(r) = b(r) = r^2$  are used. The effect of super-convergence can be observed for this example too.

#### Test3.m

```
N = 20; R = 3;
x = linspace(0,R,N);
a = @(x) x.^2; b = @(x) x.^2;
```

<sup>22</sup>This figure can not be generated by the codes in these lecture notes. The Octave command `pwquadinterp()` allows to apply the piecewise quadratic interpolation within the elements.

```
[A,M,r] = GenerateFEM(x,a,b);
A = A(1:end-1,1:end-1); % Dirichlet BC at the right end point
M = M(1:end-1,:);
r = r(:); f = (6-4*r.^2).*exp(-r.^2);
u_exact = exp(-r.^2)-exp(-R^2);
u = A\ (M*f);

figure(1); plot(r,[u;0],r,u_exact)
xlabel('r'); ylabel('u'); title('solution, exact and approximate')
legend('approximate','exact')

figure(2); plot(r,[u;0]-u_exact)
xlabel('r'); ylabel('u');
title('difference of exact and approximate solution')
```

### A fourth example, with a nonzero boundary condition

A nonzero Dirichlet condition requires a bit more care. To illustrate examine a boundary condition  $u(L) = u_n = gD$  at the right end of the interval and at the points  $x_i$  for  $1 \leq i \leq n - 1$  the values  $u_i$  of the solution. A discretization of  $u''(x) = f(x)$  might<sup>23</sup> lead to

$$\frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 & -1 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-3} \\ u_{n-2} \\ u_{n-1} \\ gD \end{pmatrix} = \mathbf{M} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-3} \\ f_{n-2} \\ f_{n-1} \end{pmatrix}$$

with a matrix of size  $(n - 1) \times n$ . The value  $u_n = gD$  is not an unknown, thus move the contribution from the last column of the matrix and change the above to

$$\frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-3} \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \mathbf{M} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-3} \\ f_{n-2} \\ f_{n-1} \end{pmatrix} + \frac{1}{h^2} \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ gD \end{pmatrix}$$

with the reduced matrix of size  $(n - 1) \times (n - 1)$ . A compact notation might be

$$\begin{bmatrix} \mathbf{A} & \vec{a}_n \end{bmatrix} \begin{pmatrix} \vec{u} \\ gD \end{pmatrix} = \mathbf{M} \vec{f} \quad \rightarrow \quad \mathbf{A} \vec{u} = \mathbf{M} \vec{f} - \vec{a}_n gD.$$

<sup>23</sup>With `GenerateFEM()` it will not be a tridiagonal matrix, but typically 5 entries in each row/column.

As example examine the function  $u_{exact}(x) = \frac{7}{L}x - x(L-x)$ , solving the boundary value problem

$$-u''(x) = -2 \quad \text{on } 0 \leq x \leq L \quad \text{with } u'(0) = 0 \quad \text{and } u(L) = 7.$$

---

**Test4.m**


---

```
N = 20; L = 3; gD = 7;
x = linspace(0,L,N+1);
a = @ (x) ones(size(x)); b = @ (x) ones(size(x));

[A,M,r] = GenerateFEM(x,a,b);
A = A(2:end,2:end); % zero Dirichlet BC at the left end point
M = M(2:end,:);

gDvec = full(A(1:end-1,end)) * gD; %% nonzero Dirichlet BC at right end point
A = A(1:end-1,1:end-1); M = M(1:end-1,:);
f = -2*ones(size(r));
u_exact = r*gD/L - r.* (L-r);
u = A \ (M*f-gDvec);

figure(1); plot(r,[0;u;gD],r,u_exact)
xlabel('r'); ylabel('u'); title('solution, exact and approximate')
legend('approximate','exact')

figure(2); plot(r,[0;u;gD]-u_exact)
xlabel('r'); ylabel('u');
title('difference of exact and approximate solution')
```

---

### A fifth example, a dynamic problem with nonzero boundary conditions

A similar notation can be used when solving a dynamic problem with nonzero Dirichlet conditions, i.e.

$$\begin{bmatrix} \mathbf{A} & \vec{a}_n \end{bmatrix} \begin{pmatrix} \vec{u}(t) \\ gD \end{pmatrix} + \mathbf{M}_t \frac{d}{dt} \vec{u}(t) = \mathbf{M}_f \vec{f} \quad \rightarrow \quad \mathbf{A} \vec{u}(t) + \mathbf{M}_t \frac{d}{dt} \vec{u}(t) = \mathbf{M}_f \vec{f} - \vec{a}_n gD.$$

- Using an implicit scheme with time step  $\Delta t$  leads to

$$\begin{aligned} \mathbf{A} \vec{u}_{i+1} + \mathbf{M}_t \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} &= \mathbf{M}_f \vec{f}_{i+1} - \vec{a}_n gD \\ (\Delta t \mathbf{A} + \mathbf{M}_t) \vec{u}_{i+1} &= +\mathbf{M}_t \vec{u}_i + \Delta t (\mathbf{M}_f \vec{f}_{i+1} - \vec{a}_n gD) \end{aligned}$$

- Using a Crank–Nicolson step of size  $\Delta t$  leads to

$$\begin{aligned} \frac{1}{2} \mathbf{A} (\vec{u}_{i+1} + \vec{u}_i) + \mathbf{M}_t \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} &= \frac{1}{2} \mathbf{M}_f (\vec{f}_{i+1} + \vec{f}_i) - \vec{a}_n gD \\ (2 \mathbf{M}_t + \Delta t \mathbf{A}) \vec{u}_{i+1} &= +(2 \mathbf{M}_t - \Delta t \mathbf{A}) \vec{u}_i + \Delta t (\mathbf{M}_f (\vec{f}_{i+1} + \vec{f}_i) - 2 \vec{a}_n gD) \end{aligned}$$

As example consider the heat distribution in a ball of radius  $R$  with initial temperature  $u_0(0, r) = 20$  and a boundary temperature of  $u(T, R) = 85$ . The corresponding initial boundary value problem is

$$\begin{aligned} r^2 \frac{\partial}{\partial t} u(t, r) &= \frac{\partial}{\partial r} (r^2 \frac{\partial u}{\partial r}) && \text{for } t > 0 \text{ and } 0 \leq r \leq R \\ u(t, R) &= 85 && \text{for } t > 0 \\ u(0, r) &= 20 && \text{for } 0 \leq r \leq R \end{aligned}$$

The discontinuous initial condition at  $r = R$  leads to numerical instabilities for the Crank–Nicolson algorithm and is thus replaced by a smooth approximation.

**Test5CN.m**

```

gD = 85; u0value = 20;
R = 2; T = 1;
x = linspace(0,R,101);
Nt = 100;
u0 = @ (x,R) u0value*ones(size(x)); %% 20
u0 = @ (x,R) u0value + exp(-20*(R-x)) * (gD-u0value); %% smoothed out
a = @ (x) x.^2; b = @ (x) x.^2;
[A,M,r] = GenerateFEM(x,a,b);
A = A(1:end-1,:); M = M(1:end-1,1:end-1); r = r(1:end-1);
an = A(:,end); A = A(:,1:end-1);
dt = T/Nt;
u = u0(r,R);
figure(2); plot([r;R], [u;gD])
    xlabel('radius r'); ylabel('initial temperature'); axis([0,R,0,90])
t = 0;
for ii = 1:Nt
    u = (M+dt/2*A)\((M-dt/2*A)*u - dt*an*gD);
    t = t+dt;
    figure(1); plot([r;R], [u;gD])
        xlabel('radius r'); ylabel('temperature'); axis([0,R,0,90])
        drawnow()
end

```

The implicit time stepper is less susceptible to the discontinuous initial value. This is related to the concept of A-stability and L-stability, shown in Section 3.4.3 (starting on page 195) and visualized in Figure 4.22 on page 289.

**Test5Implicit.m**

```

gD = 85; u0value = 20;
R = 2; T = 1;
x = linspace(0,R,21);
Nt = 100;
u0 = @ (x,R) u0value*ones(size(x)); %% 20
u0 = @ (x,R) u0value + exp(-20*(R-x)) * (gD-u0value); %% smoothed out
a = @ (x) x.^2;
b = @ (x) x.^2;
[A,M,r] = GenerateFEM(x,a,b);
A = A(1:end-1,:); M = M(1:end-1,1:end-1); r = r(1:end-1);
an = A(:,end); A = A(:,1:end-1);
dt = T/Nt;
u = u0(r,R);
figure(2); plot([r;R], [u;gD])
    xlabel('radius r'); ylabel('initial temperature'); axis([0,R,0,90])
t = 0;
for ii = 1:Nt
    u = (M+dt*A)\(M*u - dt*an*gD); %%%%
    t = t+dt;
    figure(1); plot([r;R], [u;gD])
        xlabel('radius r'); ylabel('temperature'); axis([0,R,0,90])
        drawnow()
end

```

## 6.9.4 Solving the Dynamic Tumor Growth Problem

The equation (6.18) to be examined is

$$(r^2 u'(t,r))' + r^2 \alpha f(u(t,r)) - r^2 \dot{u}(t,r) = 0 \quad \text{for } 0 < r < R \quad \text{and} \quad t > 0.$$

Using the radial symmetry conclude that  $\frac{\partial u(t,0)}{\partial r} = 0$  and thus use a no flux condition  $\frac{\partial u(t,R)}{\partial r} = 0$  as boundary condition for some large radius  $R$ . This differential equation has to be supplemented with an initial condition, e.g.

$$u(0, r) = 0.001 \cdot e^{-r^2/4}.$$

This represents a very small initial set of tumor cells located close to the origin at  $r \approx 0$ .

### Use FEM and a time discretization

Using the space discretization  $u(t, r) \rightarrow \vec{u}(t)$  and the FEM notation from the previous section (with  $a(r) = b(r) = r^2$ ) leads to<sup>24</sup>

$$\mathbf{A} \vec{u}(t) - \alpha \mathbf{M} \vec{f}(\vec{u}(t)) + \mathbf{M} \frac{d\vec{u}(t)}{dt} = \vec{0}$$

or by rearranging terms

$$\mathbf{M} \frac{d\vec{u}(t)}{dt} = -\mathbf{A} \vec{u}(t) + \alpha \mathbf{M} \vec{f}(\vec{u}(t)). \quad (6.24)$$

Because of the nonlinear function  $f(u) = u \cdot (1 - u)$  this is a nonlinear system of ordinary differential equations. Now use the finite difference method from Section 4.5, starting on page 272, to discretize the dynamic behavior. To start out use a Crank–Nicolson scheme for the time discretization, but for the nonlinear contribution use an explicit expression<sup>25</sup>. This will lead to systems of linear equations to be solved. With the time discretization  $\vec{u}_i = \vec{u}(i \Delta t)$  this leads to

$$\begin{aligned} \mathbf{M} \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} &= -\mathbf{A} \frac{\vec{u}_{i+1} + \vec{u}_i}{2} + \alpha \mathbf{M} \vec{f}(\vec{u}_i) \\ \mathbf{M} (\vec{u}_{i+1} - \vec{u}_i) &= -\frac{\Delta t}{2} \mathbf{A} (\vec{u}_{i+1} + \vec{u}_i) + \Delta t \alpha \mathbf{M} \vec{f}(\vec{u}_i) \\ (\mathbf{M} + \frac{\Delta t}{2} \mathbf{A}) \vec{u}_{i+1} &= (\mathbf{M} - \frac{\Delta t}{2} \mathbf{A}) \vec{u}_i + \Delta t \alpha \mathbf{M} \vec{f}(\vec{u}_i). \end{aligned}$$

Thus for each time step a system of linear equations has to be solved. The matrix  $\mathbf{M} + \frac{\Delta t}{2} \mathbf{A}$  does not change for the different time steps. This is a perfect situation to use MATLAB/Octave.

### Create an animation

The above is implemented in MATLAB/Octave and an animation<sup>26</sup> can be displayed on screen. Find the final state in Figure 6.30.

<pre>R = 50; % space interval [0,R] T = 6; % time interval [0,T] x = linspace(0,R,200); Nt = 200;  LogF = @(u) max(0,u.* (1-u)); al = 10; % scaling factor u0 = @(x,R) 0.001*exp(-x.^2/4); a = @(x) x.^2; b = @(x) x.^2; [A,M,r] = GenerateFEM(x,a,b); dt = T/Nt; u = u0(r,R); t = 0;  for ii = 1:Nt     u = (M+dt/2*A)\((M-dt/2*A)*u + dt*M*al*LogF(u)); % Crank-Nicolson</pre>	<a href="#">run demo</a>
--	--------------------------

<sup>24</sup>The static equation  $(a(r) u'(r))' + b(r) f(r) = 0$  leads to the linear system  $\mathbf{A} \vec{u} - \mathbf{M} \vec{f} = \vec{0}$ .

<sup>25</sup>This can be improved, see later in the notes.

<sup>26</sup>With the animation the code took 12.5 seconds to run, without the plots only 0.12 seconds. Thus most of the time is used to generate and display the plots.

```

t = t + dt;
figure(1); plot(r,u)
    xlabel('radius r'); ylabel('density u')
    axis([0 R -0.2 1.1]); text(0.7*R,0.7,sprintf('t = %5.3f',t))
drawnow();
end%for

```

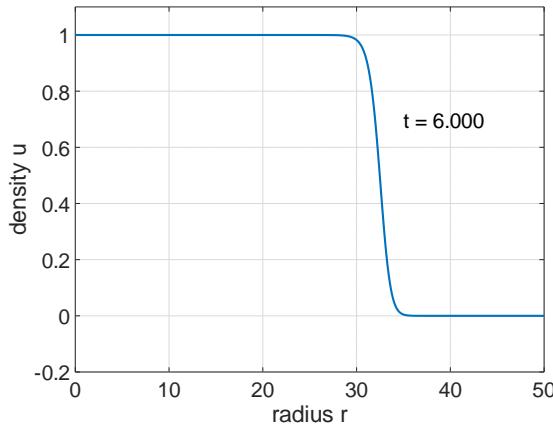


Figure 6.30: A snapshot of the solution

### Generate surfaces and contours

With a modification of the above code one can generate surfaces and contours for the concentration of the tumor cells for short (Figure 6.31) and long (Figure 6.32) time intervals.

---

#### LogisticModelContour.m

---

run demo  
twice

```

% short time interval
R = 35/3; % space interval [0,R]
T = 5/3; % time interval [0,T]
Nx = 100; x = linspace(0,R,Nx); Nt = 100;

% long time interval
%R = 100; % space interval [0,R]
%T = 15; % time interval [0,T]
%Nx = 400; x = linspace(0,R,Nx); Nt = 400;

u_all = zeros(2*Nx-1,Nt+1);
LogF = @(u) max(0,u.* (1-u)); al = 10; % scaling factor
u0 = @(x,R) 0.001*exp(-x.^2/4);
a = @(x) x.^2; b = @(x) x.^2;
[A,M,r] = GenerateFEM(x,a,b);

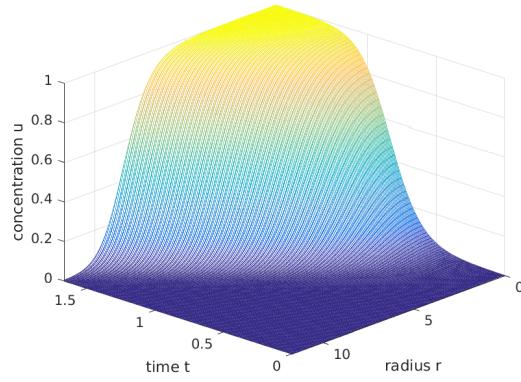
dt = T/Nt;
u = u0(r,R);
u_all(:,1) = u;
t = 0;
for ii = 1:Nt
    u = (M+dt/2*A)\((M-dt/2*A)*u + dt*M*al*LogF(u)); % Crank-Nicolson,
    t = t + dt;
    u_all(:,ii+1) = u;
end%for

```

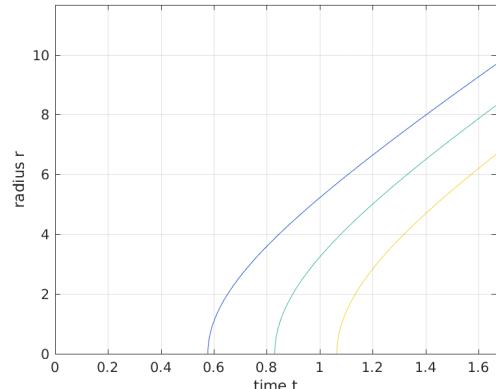
```

figure(2); mesh(0:dt:T,r,u_all)
    ylabel('radius r'); xlabel('time t'); zlabel('concentration u')
    axis([0,T,0,R,0,1])
figure(3); contour(0:dt:T,r,u_all,[0.1 0.5 0.9])
    caxis([0 1]); ylabel('radius r'); xlabel('time t')

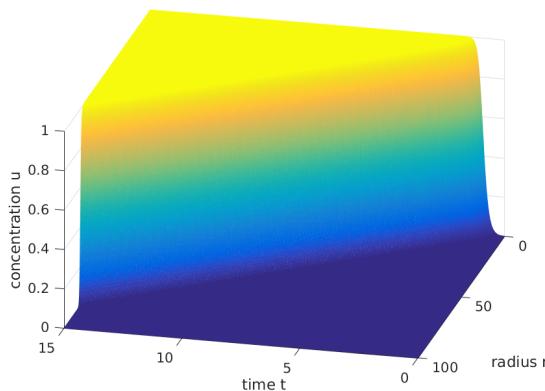
```



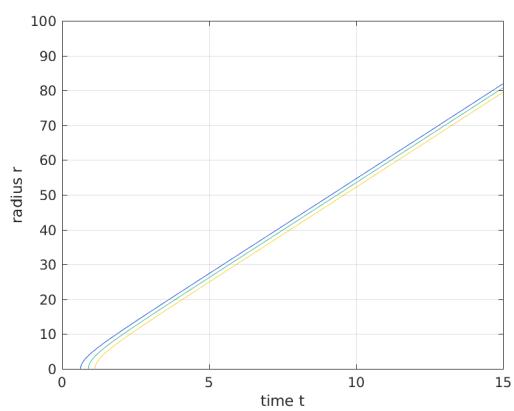
(a) surface



(b) contours at concentrations of 10%, 50% and 90%

Figure 6.31: Concentration  $u(t, r)$  as function of time  $t$  and radius  $r$  on a short time interval and contours

(a) surface



(b) contours at concentrations of 10%, 50% and 90%

Figure 6.32: Concentration  $u(t, r)$  as function of time  $t$  and radius  $r$  on a long time interval and contours

## Discussion

- In Figure 6.31 observe the small initial seed growing to a moving front where the concentration increases from 0 to 1 over a short distance.
- In Figure 6.32 it is obvious that this front moves with a constant speed, without changing width or shape.
- The above is a clear indication that for the moving front section the original equation

$$r^2 \dot{u}(t, r) = (r^2 u'(t, r))' + r^2 \alpha f(u(t, r))$$

$$\dot{u}(t, r) = u''(t, r) + \frac{2}{r} u'(t, r) + \alpha f(u(t, r))$$

can be replaced by Fisher's equation

$$\dot{u}(t, r) = D u''(t, r) + \alpha u(t, r) (1 - u(t, r)),$$

i.e. the contribution  $\frac{2}{r} u'(t, r)$  is dropped. The behavior of solutions of this equation is well studied and the literature is vast!

- If clinical data is available the real task will be to find good values for the parameters  $D$  and  $\alpha$  to match the observed behavior of the tumors.
- Instead of the function  $f(u) = u \cdot (1 - u)$  other functions may be used to describe the growth of the tumor.
- For small time steps the algorithm showed severe instability, caused by the nonlinear contribution  $f(u) = u \cdot (1 - u)$ . Using  $\max\{f(u), 0\}$  improved the situation slightly.
- The traveling speed of the front depended rather severely on the time step. This is not surprising as the contribution  $f(\vec{u}_i)$  is the driving force, and it is lagging behind by  $\frac{1}{2} \Delta t$  since Crank–Nicolson is formulated at time  $t = t_i + \frac{1}{2} \Delta t$ . One should use  $f(\frac{1}{2} (\vec{u}_i + \vec{u}_{i+1}))$  instead, but this leads to a nonlinear system of equations to be solved for each time step.

### Improvements for the nonlinear Crank–Nicolson step

In the above approach a Crank–Nicolson scheme is used for the linear part and an explicit scheme for the nonlinear part, i.e. solve

$$\mathbf{M} \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} = -\mathbf{A} \frac{\vec{u}_{i+1} + \vec{u}_i}{2} + \alpha \mathbf{M} \vec{f}(\vec{u}_i)$$

for  $\vec{u}_{i+1}$ . This approach is consistent of order 1 with respect to the time step, i.e. the error is expected to be proportional to  $\Delta t$ . A more consistent approximation should evaluate the nonlinear function at the midpoint too, i.e. at  $\frac{1}{2} (\vec{u}_{i+1} + \vec{u}_i)$ . Then the approximation error is expected to be proportional to  $(\Delta t)^2$ . Thus one should solve the nonlinear system

$$\mathbf{M} \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} = -\mathbf{A} \frac{\vec{u}_{i+1} + \vec{u}_i}{2} + \alpha \mathbf{M} \vec{f}\left(\frac{\vec{u}_{i+1} + \vec{u}_i}{2}\right)$$

for the unknown  $\vec{u}_{i+1}$ . There are (at least) two approaches for this.

- Use a linear approximation for the nonlinear term. Based on the approximation<sup>27</sup>

$$f\left(\frac{\vec{u}_i + \vec{u}_{i+1}}{2}\right) = f(\vec{u}_i) + \frac{\vec{u}_{i+1} - \vec{u}_i}{2} \approx f(\vec{u}_i) + f'(\vec{u}_i) \cdot \frac{\vec{u}_{i+1} - \vec{u}_i}{2}$$

find the slightly modified system of equations for  $\vec{u}_{i+1}$ .

$$\begin{aligned} \mathbf{M} \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} &= -\mathbf{A} \frac{\vec{u}_{i+1} + \vec{u}_i}{2} + \alpha \mathbf{M} \vec{f}\left(\frac{\vec{u}_i + \vec{u}_{i+1}}{2}\right) \\ &\approx -\mathbf{A} \frac{\vec{u}_{i+1} + \vec{u}_i}{2} + \alpha \mathbf{M} \left( \vec{f}(\vec{u}_i) + f'(\vec{u}_i) \cdot \frac{\vec{u}_{i+1} - \vec{u}_i}{2} \right) \\ \left( \mathbf{M} + \frac{\Delta t}{2} \mathbf{A} - \frac{\alpha \Delta t}{2} \mathbf{M} \vec{f}'(\vec{u}_i) \right) \vec{u}_{i+1} &= \left( \mathbf{M} - \frac{\Delta t}{2} \mathbf{A} - \frac{\alpha \Delta t}{2} \mathbf{M} \vec{f}'(\vec{u}_i) \right) \vec{u}_i + \Delta t \alpha \mathbf{M} \vec{f}(\vec{u}_i). \end{aligned}$$

This can be implemented with MATLAB/Octave, leading to the code in `LogisticModel2.m`. The solutions look very similar to the ones from the previous section, but the traveling speed of the front

<sup>27</sup>Approximating  $\frac{1}{2} (f(\vec{u}_i) + f(\vec{u}_{i+1}))$  leads to the identical formula.

is different. The linear system to be solved is modified for each time step, thus the computational effort is higher. Since all matrices have a very narrow band structure the computational penalty is not very high. A more careful examination of the above approach reveals that this actually one step of Newton's method to solve the nonlinear system of equations.

- One might as well use Newton's algorithm to solve the nonlinear system of equations for  $\vec{u}_{i+1}$ . To apply the algorithm consider each time step as a nonlinear system of equations  $\mathbf{F}(\vec{u}_{i+1}) = \vec{0}$ . To approximate  $\mathbf{F}(\vec{u}_{i+1} + \vec{\Phi})$  differentiate with respect to  $\vec{u}_{i+1}$ .

$$\begin{aligned}\vec{0} &= \mathbf{F}(\vec{u}_{i+1}) = \mathbf{M} \frac{\vec{u}_{i+1} - \vec{u}_i}{\Delta t} + \mathbf{A} \frac{\vec{u}_{i+1} + \vec{u}_i}{2} - \alpha \mathbf{M} \vec{f} \left( \frac{\vec{u}_i + \vec{u}_{i+1}}{2} \right) \\ \mathbf{DF}(\vec{u}_{i+1}) \vec{\Phi} &= \frac{1}{\Delta t} \mathbf{M} \vec{\Phi} + \frac{1}{2} \mathbf{A} \vec{\Phi} - \frac{\alpha}{2} \mathbf{M} \vec{f} \left( \frac{\vec{u}_i + \vec{u}_{i+1}}{2} \right) \vec{\Phi} \\ \vec{0} &= \mathbf{F}(\vec{u}_{i+1}) + \mathbf{DF}(\vec{u}_{i+1}) \vec{\Phi} \quad \text{solve for } \vec{\Phi} \\ \vec{u}_{i+1} \longrightarrow \vec{u}_{i+1} + \vec{\Phi} &= \vec{u}_{i+1} - (\mathbf{DF}(\vec{u}_{i+1}))^{-1} \mathbf{F}(\vec{u}_{i+1})\end{aligned}$$

This can be implemented with MATLAB/Octave, leading to the code in `LogisticModel3.m`. The solutions look very similar to the ones from the previous section, and the traveling speed of the front is close to the speed observed in `LogisticModel2.m`.

- The different speeds observed for the above approaches should trigger the question: what is the correct speed. For this examine the results of four different runs.
  1. Use the explicit approximation in `LogisticModel1.m` at `Nt=200` time steps.
  2. Use the explicit approximation `LogisticModel1.m` with `Nt=3200` time steps.
  3. Use the linearized approximation in `LogisticModel2.m` with `Nt=200` time steps.
  4. Use the nonlinear approximation in `LogisticModel3.m` with `Nt=200` time steps.

Then examine the graphical results in Figure 6.33.

- The solutions by the linearized and fully nonlinear approaches differ very little. This is no surprise, since the linearized approach is just the first step of Newton's method, which is used for the nonlinear approach.
- The explicit solution with `Nt=200` leads to a clearly slower speed and the smaller time step with `Nt=3200` leads to a speed much closer to the one observed by the nonlinear approach. This clearly indicates that the observed speed depends on the time step, and for smaller time steps the front is moving with a speed move closer to the observed speed for the linearized and nonlinear approach.
- As a consequence one should use the linearized or nonlinear approach.

### LogisticModel2.m

```
R = 50; % space interval [0,R]
T = 6; % time interval [0,T]
x = linspace(0,R,200); Nt = 200;
LogF = @(u) u.*.(1-u); dLogF = @(u) 1-2*u;

al = 10; % scaling factor
u0 = @(x,R) 0.001*exp(-x.^2/4);
a = @(x) x.^2; b = @(x) x.^2;
[A,M,r] = GenerateFEM(x,a,b);
dt = T/Nt; u = u0(r,R); t = 0;

for ii = 1:Nt
```

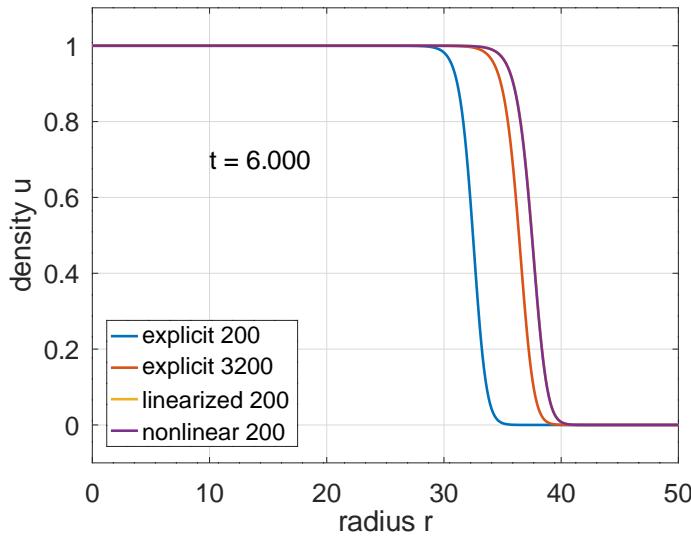


Figure 6.33: Graph of the solutions by four computations with different algorithms

```

dfu = dt*al/2*M*diag(dLogF(u));
u = (M+dt/2*A-dfu)\( (M-dt/2*A-dfu)*u + dt*M*al*LogF(u)); % Crank-Nicolson
t = t + dt;
figure(1); plot(r,u); xlabel('radius r'); ylabel('density u')
axis([0 R -0.2 1.1]); text(0.7*R, 0.7, sprintf('t = %5.3f', t))
drawnow();
end%for

```

**LogisticModel3.m**

```

% use true Newton
R = 50; % space interval [0,R]
T = 6; % time interval [0,T]
x = linspace(0,R,200); Nt = 200;
LogF = @(u) u.*.(1-u); dLogF = @(u) 1-2*u;

al = 10; % scaling factor
u0 = @(x,R) 0.001*exp(-x.^2/4);
a = @(x) x.^2; b = @(x) x.^2;
[A,M,r] = GenerateFEM(x,a,b);
dt = T/Nt; u = u0(r,R); t = 0;

for ii = 1:Nt
    up = u; % start for Newton iteration
    for iter = 1:3 % choose number of Newton steps
        F = 1/dt*M*(up-u) + 1/2*A*(u+up) - al*M*LogF((u+up)/2);
        DF = 1/dt*M + 1/2*A - 1/2*al*M*diag(dLogF((u+up)/2));
        phi = DF\F;
    % disp([norm(phi),norm(F)]) % trace the error
        up = up-phi;
    end%for
    u = up; t = t + dt; % update solution
    figure(1); plot(r,u); xlabel('radius r'); ylabel('density u')
    axis([0 R -0.2 1.1]); text(0.7*R, 0.7, sprintf('t = %5.3f', t))
    drawnow();
end%for

```

## Bibliography

- [AxelBark84] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Values Problems*. Academic Press, 1984.
- [Brae02] D. Braess. *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer, second edition, 2002.
- [Cowp73] G. R. Cowper. Gaussian quadrature formulas for triangles. *International Journal on Numerical Methods and Engineering*, 7:405–408, 1973.
- [Davi80] A. J. Davies. *The Finite Element Method: a First Approach*. Oxford University Press, 1980.
- [Gmür00] T. Gmür. *Méthode des éléments finis en mécanique des structures*. Mécanique (Lausanne). Presses polytechniques et universitaires romandes, 2000.
- [Hugh87] T. J. R. Hughes. *The Finite Element Method, Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, 1987. Reprinted by Dover.
- [John87] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, 1987. Republished by Dover.
- [KnabAnge00] P. Knabner and L. Angermann. *Numerik partieller Differentialgleichungen*. Springer Verlag, Berlin, 2000.
- [LascTheo87] P. Lascaux and R. Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur, Tome 2*. Masson, Paris, 1987.
- [Li21] G. Li. *Introduction to the Finite Element Method and Implementation with MATLAB*. Cambridge University Press, 2021.
- [www:triangle] J. R. Shewchuk. <https://www.cs.cmu.edu/~quake/triangle.html>.
- [StraFix73] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method*. Prentice-Hall, 1973.
- [TongRoss08] P. Tong and J. Rossettos. *Finite Element Method, Basic Technique and Implementation*. MIT, 1977. Republished by Dover in 2008.
- [Zienkiewicz13] O. Zienkiewicz, R. Taylor, and J. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 7 edition, 2013.

Find a longer literature list for FEM, with some comments, in Section 0.3.2 and Table 2, starting on page 3.

# Bibliography

- [AbraSteg] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, 1972.
- [Acto90] F. S. Acton. *Numerical Methods that Work; 1990 corrected edition*. Mathematical Association of America, Washington, 1990.
- [Agga20] C. Aggarwal. *Linear Algebra and Optimization for Machine Learning*. Springer, first edition, 2020.
- [AmreWihl14] M. Amrein and T. Wihler. An adaptive Newton-method based on a dynamical systems approach. *Communications in Nonlinear Science and Numerical Simulation*, 19(9):2958–2973, 2014.
- [Aris62] R. Aris. *Vectors, Tensors and the Basic Equations of Fluid Mechanics*. Prentice Hall, 1962. Republished by Dover.
- [AtkiHan09] K. Atkinson and W. Han. *Theoretical Numerical Analysis*. Number 39 in Texts in Applied Mathematics. Springer, 2009.
- [Axel94] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, 1994.
- [AxelBark84] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Values Problems*. Academic Press, 1984.
- [templates] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.
- [BoneWood08] J. Bonet and R. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, 2008.
- [BoriTara79] A. I. Borisenko and I. E. Tarapov. *Vector and Tensor Analysis with Applications*. Dover, 1979. first published in 1966 by Prentice-Hall.
- [Bowe10] A. F. Bower. *Applied Mechanics of Solids*. CRC Press, 2010. web site at solidmechanics.org.
- [Brae02] D. Braess. *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer, second edition, 2002.
- [Butc03] J. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, second edition, 2003.
- [Butc16] J. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, third edition, 2016.
- [ChouPaga67] P. C. Chou and N. J. Pagano. *Elasticity, Tensors, dyadic and engineering Approaches*. D Van Nostrand Company, 1967. Republished by Dover 1992.
- [Ciar02] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. SIAM, 2002.

- [Cowp73] G. R. Cowper. Gaussian quadrature formulas for triangles. *International Journal on Numerical Methods and Engineering*, 7:405–408, 1973.
- [TopTen] B. A. Cypr. The best of the 20th century: Editors name top 10 algorithms. *SIAM News*, 2000.
- [DahmReus07] W. Dahmen and A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. Springer, 2007.
- [Davi80] A. J. Davies. *The Finite Element Method: a First Approach*. Oxford University Press, 1980.
- [Deim84] K. Deimling. *Nonlinear Functional Analysis*. Springer Verlag, 1984.
- [DeisFaisOng20] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. pre-publication.
- [Demm97] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.
- [www:LinAlgFree] J. Dongarra. Freely available software for linear algebra on the web. <http://www.netlib.org/utk/people/JackDongarra/la-sw.html>.
- [DowdSeve98] K. Dowd and C. Severance. *High Performance Computing*. O'Reilly, 2nd edition, 1998.
- [DrapSmit98] N. Draper and H. Smith. *Applied Regression Analysis*. Wiley, third edition, 1998.
- [Farl82] S. J. Farlow. *Partial Differential Equations for Scientist and Engineers*. Dover, New York, 1982.
- [Froc16] J. Frochte. *Finite-Elemente-Methode: Eine praxisbezogene Einführung mit GNU Octave/MATLAB*. Hanser Fachbuchverlag, 2016. Octave/Matlab code available.
- [GhabPeckWu17] J. Ghaboussi, D. Pecknold, and X. Wu. *Nonlinear Computational Solid Mechanics*. CRC Press, 2017.
- [GhabWu16] J. Ghaboussi and X. Wu. *Numerical Methods in Computational Mechanics*. CRC Press, 2016.
- [Gmür00] T. Gmür. *Méthode des éléments finis en mécanique des structures*. Mécanique (Lausanne). Presses polytechniques et universitaires romandes, 2000.
- [Gold91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1), March 1991.
- [GoluVanLoan96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.
- [GoluVanLoan13] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2013.
- [Gree77] D. T. Greenwood. *Classical Dynamics*. Prentice Hall, 1977. Republished by Dover 1997.
- [Hack94] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, volume 95 of *Applied Mathematical Sciences*. Springer, first edition, 1994.
- [Hack16] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, volume 95 of *Applied Mathematical Sciences*. Springer, second edition, 2016.
- [Hack15] R. Hackett. *Hyperelasticity Primer*. Springer International Publishing, 2015.
- [HairNorsWann08] E. Hairer, S. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Non-stiff Problems*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, second edition, 1993. third printing 2008.

- [HairNorsWann96] E. Hairer, S. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Lecture Notes in Economic and Mathematical Systems. Springer, second edition, 1996.
- [Hear97] E. J. Hearns. *Mechanics of Materials 1*. Butterworth–Heinemann, third edition, 1997.
- [HenrWann17] P. Henry and G. Wanner. Johann Bernoulli and the cycloid: A theorem for posterity. *Elemente der Mathematik*, 72(4):137–163, 2017.
- [HeroArnd01] H. Herold and J. Arndt. *C-Programmierung unter Linux*. SuSE Press, 2001.
- [Holz00] G. A. Holzapfel. *Nonlinear Solid Mechanics, a Continuum Approach for Engineering*. John Wiley & Sons, 2000.
- [HornJohn90] R. Horn and C. Johnson. *Matrix Analysis*. Cambridge University Press, 1990.
- [Hugh87] T. J. R. Hughes. *The Finite Element Method, Linear Static and Dynamic Finite Element Analysis*. Prentice–Hall, 1987. Reprinted by Dover.
- [HuntLipsRose14] B. R. Hunt, R. L. Lipsman, and J. M. Rosenberg. *A Guide to MATLAB: For Beginners and Experienced Users*. Cambridge University Press, New York, NY, USA, third edition, 2014.
- [Intel90] Intel Corporation. *i486 Microprocessor Programmers Reference Manual*. McGraw-Hill, 1990.
- [IsaaKell66] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. John Wiley & Sons, 1966. Republished by Dover in 1994.
- [John87] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, 1987. Republished by Dover.
- [Kell92] H. B. Keller. *Numerical Methods for Two–Point Boundary Value Problems*. Dover, 1992.
- [KhanKhan18] O. Khanmohamadi and E. Khanmohammadi. Four fundamental spaces of numerical analysis. *Mathematics Magazin*, 91(4):243–253, 2018.
- [KnabAnge00] P. Knabner and L. Angermann. *Numerik partieller Differentialgleichungen*. Springer Verlag, Berlin, 2000.
- [Knor08] M. Knorrer. *Numerische Mathematik*. Carl Hanser Verlag, 2008.
- [Koko15] J. Koko. *Approximation numérique avec Matlab, Programmation vectorisée, équations aux dérivées partielles*. Ellipses, Paris, 2015.
- [LascTheo87] P. Lascaux and R. Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur, Tome 2*. Masson, Paris, 1987.
- [Li21] G. Li. *Introduction to the Finite Element Method and Implementation with MATLAB*. Cambridge University Press, 2021.
- [LiesTich05] J. Liesen and P. Tichý. Convergence analysis of Krylov subspace methods. *GAMM Mitt. Ges. Angew. Math. Mech.*, 27(2):153–173 (2005), 2004.
- [Linz79] P. Linz. *Theoretical Numerical Analysis*. John Wiley & Sons, 1979. Republished by Dover.
- [MeybVach91] K. Meyberg and P. Vachenauer. *Höhere Mathematik II*. Springer, Berlin, 1991.
- [MontRung03] D. Montgomery and G. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, third edition, 2003.

- [Oden71] J. Oden. *Finite Elements of Nonlinear Continua*. Advanced engineering series. McGraw-Hill, 1971. Republished by Dover, 2006.
- [DLMF15] N. I. of Standards. NIST Digital Library of Mathematical Functions, at <http://dlmf.nist.gov>, 2015.
- [Ogde13] R. Ogden. *Non-Linear Elastic Deformations*. Dover Civil and Mechanical Engineering. Dover Publications, 2013.
- [OttoPete92] N. S. Ottosen and H. Petersson. *Introduction to the Finite Element Method*. Prentice Hall, 1992.
- [Pres92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.
- [Prze68] J. Przemieniecki. *Theory of Matrix Structural Analysis*. McGraw–Hill, 1968. Republished by Dover in 1985.
- [RalsRabi78] A. Ralston and P. Rabinowitz. *A first Course in Numerical Analysis*. McGraw–Hill, second edition, 1978. republished by Dover in 2001.
- [RawlPantuDick98] J. Rawlings, S. Pantula, and D. Dickey. *Applied regression analysis*. Springer texts in statistics. Springer, New York, 2. ed edition, 1998.
- [Redd84] J. N. Reddy. *An Introduction to the Finite Element Analysis*. McGraw–Hill, 1984.
- [Redd13] J. N. Reddy. *An Introduction to Continuum Mechanics*. Cambridge University Press, 2nd edition, 2013.
- [Redd15] J. N. Reddy. *An Introduction to Nonlinear Finite Element Analysis*. Oxford University Press, 2nd edition, 2015.
- [Saad00] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS, second edition, 2000. available on the internet.
- [SchnWihl11] H. R. Schneebeli and T. Wihler. The Netwon-Raphson Method and Adaptive ODE Solvers. *Fractals*, 19(1):87–99, 2011.
- [Schw86] H. R. Schwarz. *Numerische Mathematik*. Teubner, Braunschweig, 1986.
- [Schw88] H. R. Schwarz. *Finite Element Method*. Academic Press, 1988.
- [Schw09] H. R. Schwarz. *Numerische Mathematik*. Teubner und Vieweg, 7. edition, 2009.
- [Sege77] L. A. Segel. *Mathematics Applied to Continuum Mechanics*. MacMillan Publishing Company, New York, 1977. republished by Dover 1987.
- [Shab08] A. A. Shabana. *Computational Continuum Mechanics*. Cambridge University Press, 2008.
- [ShamDym95] I. Shames and C. Dym. *Energy and Finite Element Methods in Structural Mechanics*. New Age International Publishers Limited, 1995.
- [ShamReic97] L. Shampine and M. W. Reichelt. The MATLAB ODE Suite. *SIAM Journal on Scientific Computing*, 18:1–22, 1997.
- [www:triangle] J. R. Shewchuk. <https://www.cs.cmu.edu/~quake/triangle.html>.
- [Shew94] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.

- [Smit84] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, Oxford, third edition, 1986.
- [Sout73] R. W. Soutas-Little. *Elasticity*. Prentice-Hall, 1973.
- [VarFEM] A. Stahel. Calculus of Variations and Finite Elements. Lecture Notes used at HTA Biel, 2000.
- [Stah00] A. Stahel. Calculus of Variations and Finite Elements. supporting notes, 2000.
- [Octave07] A. Stahel. Octave and Matlab for Engineering Applications. lecture notes, 2007.
- [Stah08] A. Stahel. Numerical Methods. lecture notes, BFH-TI, 2008.
- [Stah16] A. Stahel. Statistics with Matlab/Octave. supporting notes, BFH-TI, 2016.
- [StraFix73] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method*. Prentice-Hall, 1973.
- [Thom95] J. W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*, volume 22 of *Texts in Applied Mathematics*. Springer Verlag, New York, 1995.
- [TongRoss08] P. Tong and J. Rossettos. *Finite Element Method, Basic Technique and Implementation*. MIT, 1977. Republished by Dover in 2008.
- [Trim90] D. W. Trim. *Applied Partial Differential Equations*. PWS-Kent, 1990.
- [Wein74] R. Weinstock. *Calculus of Variations*. Dover, New York, 1974.
- [Wilk63] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall, 1963. Republished by Dover in 1994.
- [Wlok82] J. Wloka. *Partielle Differentialgleichungen*. Teubner, Stuttgart, 1982.
- [YounGreg72] D. M. Young and R. T. Gregory. *A Survey of Numerical Analysis, Volume 1*. Dover Publications, New York, 1972.
- [Zien13] O. Zienkiewicz, R. Taylor, and J. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 7 edition, 2013.
- [ZienMorg06] O. C. Zienkiewicz and K. Morgan. *Finite Elements and Approximation*. John Wiley & Sons, 1983. Republished by Dover in 2006.

# List of Figures

1	Structure of the topics examined in this class . . . . .	2
1.1	Temperature $T$ as function of the horizontal position . . . . .	12
1.2	Segment of a string . . . . .	13
1.3	Nonlinear stress strain relation . . . . .	18
1.4	Bending of a Beam . . . . .	18
1.5	A nonsmooth function $f$ and three regularized approximations $u$ . . . . .	22
2.1	Memory and cache access times on a Alpha 21164 system . . . . .	28
2.2	FLOPS for a 21164 microprocessor system, four implementations of one algorithm . . . . .	28
2.3	FLOPS for a few systems, four implementations of one algorithm . . . . .	29
2.4	CPU-cache structure for the Intel I7-920 (Nehalem) . . . . .	30
2.5	The discrete approximation of a continuous function . . . . .	32
2.6	A $4 \times 4$ grid on a square domain . . . . .	34
2.7	LR factorization, using elementary matrices . . . . .	44
2.8	The Cholesky decomposition for a banded matrix . . . . .	66
2.9	Cholesky steps for a banded matrix. The active area is marked . . . . .	67
2.10	The sparsity pattern of a band matrix and two Cholesky factorizations . . . . .	71
2.11	Graph of a function to be minimized and its level curves . . . . .	74
2.12	One step of a gradient iteration . . . . .	75
2.13	The gradient algorithm for a large condition number . . . . .	77
2.14	Ellipse and circle to illustrate conjugate directions . . . . .	78
2.15	One step of a conjugate gradient iteration . . . . .	79
2.16	Two steps of the gradient algorithm (blue) and the conjugate gradient algorithm (green) . . . . .	80
2.17	Number of flops for banded Cholesky, steepest descent and conjugate gradient algorithm . . . . .	84
2.18	The GMRES algorithm . . . . .	94
2.19	The GMRES(m) algorithm . . . . .	95
2.20	The Arnoldi algorithm . . . . .	97
2.21	Comparison of linear solvers . . . . .	99
3.1	Method of bisection to solve one equation . . . . .	105
3.2	Method of false position to solve one equation . . . . .	106
3.3	Secant method to solve one equation . . . . .	107
3.4	Newton's method to solve one equation . . . . .	107
3.5	Three functions that might cause problems for Newton's methods . . . . .	108
3.6	The contraction mapping principle . . . . .	112
3.7	Successive substitution to solve $\cos x = x$ . . . . .	113
3.8	Partial successive substitution to solve $3 + 3x = \exp(x)$ . . . . .	114
3.9	Graph of the function $y = x^2 - 1 - \cos(x)$ . . . . .	121
3.10	Definition and graph of the auxiliary function $h$ . . . . .	123
3.11	Graphs for stretching of a beam, with Poisson contraction . . . . .	125

3.12	Graph of a function $h = f(x, y)$ , with contour lines . . . . .	129
3.13	A linear mapping applied to a rectangle . . . . .	130
3.14	Level curve or surface of quadratic forms in $\mathbb{R}^2$ or $\mathbb{R}^3$ . . . . .	137
3.15	Level surface of a quadratic form in $\mathbb{R}^3$ with intersection and projection onto $\mathbb{R}^2$ . . . . .	141
3.16	A linear vector field with the eigenvectors . . . . .	146
3.17	Spirals as solutions of a system of three differential equations . . . . .	147
3.18	Image compression by SVD with different number $n$ of contributions . . . . .	151
3.19	Raw data and level curves for the likelihood function . . . . .	156
3.20	Raw data and the domain of confidence at level of significance $\alpha = 0.05$ and the PCA . . . . .	157
3.21	PCA demo for data in $\mathbb{R}^3$ . . . . .	159
3.22	Projection of data in the direction of the first principal component . . . . .	160
3.23	Scaled data and level curves for the likelihood function . . . . .	162
3.24	Trapezoidal integration . . . . .	163
3.25	The integral $\int_0^{\pi/2} \cos(x) dx$ by the trapezoidal rule . . . . .	165
3.26	The approximation errors of three integration algorithms as function of the stepsize . . . . .	173
3.27	Two graphs of functions for integration . . . . .	175
3.28	Domains in the plane $\mathbb{R}^2$ . . . . .	179
3.29	Vector field and three solutions for a logistic equation . . . . .	183
3.30	One solution and the vector field for the Volterra-Lotka problem . . . . .	184
3.31	Vector field and a solution for a spring-mass problem . . . . .	186
3.32	Vector field and solution of the ODE $\frac{d}{dt} x(t) = x(t)^2 - 2t$ . . . . .	187
3.33	One step of the Heun Method . . . . .	188
3.34	One step of the Runge-Kutta Method . . . . .	189
3.35	Code for Runge-Kutta with fixed step size . . . . .	194
3.36	Conditional stability of Euler's approximation to $\frac{d}{dt} y(t) = \lambda y(t)$ with $\lambda < 0$ . . . . .	196
3.37	Unconditional stability of the implicit approximation to $\frac{d}{dt} y(t) = \lambda y(t)$ with $\lambda < 0$ . . . . .	197
3.38	Domains of stability in $\mathbb{C}$ for a few algorithms . . . . .	199
3.39	Graphical results for a Heun based method and Runge-Kutta . . . . .	208
3.40	Solution of an ODE by <code>ode45()</code> at the computed times or at preselected times . . . . .	209
3.41	SIR model,with infection rate $b$ and recovery rate $k$ . . . . .	211
3.42	SIR model vector field with $b = \frac{1}{3}$ and $k = \frac{1}{10}$ . . . . .	212
3.43	Solution of a non-stiff ODE with different algorithms . . . . .	214
3.44	Solution of a stiff ODE with different algorithms . . . . .	216
3.45	Solution of a stiff ODE system with different algorithms . . . . .	217
3.46	Solution of a non-stiff ODE system with different algorithms . . . . .	219
3.47	Regression of a straight line . . . . .	220
3.48	An example for linear regression . . . . .	223
3.49	Regions of confidence for two similar regressions with identical data . . . . .	227
3.50	Linear regression for a parabola, with a small or a large noise contribution . . . . .	232
3.51	Region of confidence for the parameters $p_1$ , $p_2$ and $p_3$ . . . . .	234
3.52	Fitting a straight line to data close to a parabola . . . . .	236
3.53	Intensity of light as function of the angle $\alpha$ . . . . .	237
3.54	Result of a 3D linear regression . . . . .	238
3.55	Least square approximation of a damped oscillation . . . . .	241
3.56	Nonlinear least square approximation with <code>fsolve()</code> . . . . .	242
3.57	Data points and the optimal fit by a logistic function . . . . .	243
3.58	The intersection of the 95% confidence ellipsoid in $\mathbb{R}^4$ with 2D-planes . . . . .	246
3.59	The intersection of the 95% confidence ellipsoid in $\mathbb{R}^4$ with the $p_4 = \text{const}$ plane . . . . .	247
4.1	FD stencils for $y'(t)$ , forward, backward and centered approximations . . . . .	254
4.2	Finite difference approximations of derivatives . . . . .	255

4.3	Discretization and arithmetic error contributions . . . . .	256
4.4	Finite difference stencil for $-u_{xx} - u_{yy}$ if $h = h_x = h_y$ . . . . .	257
4.5	Finite difference stencil for $u_t - u_{xx}$ , explicit, forward . . . . .	257
4.6	Finite difference stencil for $u_t - u_{xx}$ , implicit, backward . . . . .	257
4.7	A finite difference approximation of an initial value problem . . . . .	258
4.8	Exact and approximate solution of a boundary value problem . . . . .	259
4.9	An approximation scheme for $-u''(x) = f(x)$ . . . . .	260
4.10	A general approximation scheme for boundary value problems . . . . .	260
4.11	Stretching of a beam, displacement and force . . . . .	266
4.12	Stretching of a beam with constant and variable cross section . . . . .	268
4.13	A finite difference grid for a steady state heat equation . . . . .	270
4.14	Solution of the steady state heat equation on a square . . . . .	271
4.15	A finite difference grid for a dynamic heat equation . . . . .	273
4.16	Explicit finite difference approximation . . . . .	275
4.17	Solution of 1-d heat equation, stable and unstable algorithms with $r \approx 0.5$ . . . . .	276
4.18	Implicit finite difference approximation . . . . .	278
4.19	Solution of 1-d heat equation, implicit scheme with small and large step sizes . . . . .	278
4.20	Crank–Nicolson finite difference approximation . . . . .	280
4.21	Solution of the dynamic heat equation on a square . . . . .	284
4.22	Stability functions $g(z)$ for four algorithms with $z = \lambda \Delta t$ . . . . .	289
4.23	Explicit finite difference approximation for the wave equation . . . . .	290
4.24	D'Alembert's solution of the wave equation . . . . .	294
4.25	Implicit finite difference approximation for the wave equation . . . . .	294
4.26	The nonlinear beam stretching problem, solved by successive substitution, with errors . . . . .	298
4.27	Bending of a beam, solved by Newton's method . . . . .	300
4.28	Bending of a beam with large force, solved as linear problem and by Newton's method . . . . .	301
4.29	Nonlinear beam problem for a large force, solved by a parameterized Newton's method . . . . .	302
5.1	Shortest connection between two points . . . . .	309
5.2	Pendulum with moving support . . . . .	323
5.3	Numerical solution for a pendulum with moving support . . . . .	325
5.4	Definition of the modulus of elasticity $E$ and the Poisson number $\nu$ . . . . .	326
5.5	Deformation of an elastic solid . . . . .	326
5.6	Definition of strain: rectangle before and after deformation . . . . .	327
5.7	Rotation of the coordinate system . . . . .	331
5.8	Definition of stress in a plane . . . . .	336
5.9	Normal and tangential stress in an arbitrary direction . . . . .	337
5.10	Components of stress in space . . . . .	338
5.11	Mohr's circle for the 2D and 3D situations . . . . .	341
5.12	How to determine the maximal principal stress, von Mises stress and Tresca stress . . . . .	343
5.13	Action of a linear mapping from $\mathbb{R}^2$ onto $\mathbb{R}^2$ . . . . .	347
5.14	Block to be deformed, used to determine the elastic energy density $W$ . . . . .	350
5.15	Situation for the basic version of Hooke's law . . . . .	355
5.16	Torsion of a tube . . . . .	361
5.17	Stress–strain curve for Hooke's linear law and an incompressible neo–Hookean material under uniaxial loading . . . . .	371
5.18	Stress–strain curve for Hooke's linear law and an incompressible neo–Hookean material under biaxial loading . . . . .	372
5.19	Stress–strain curve for Hooke's linear law and a compressible Neo–Hookean material with $\nu = 0.3$ under uniaxial load . . . . .	377
5.20	Stress-strain curve for Mooney–Rivlin under uniaxial loading . . . . .	380

5.21 Stress-strain curve for Hooke's linear law and an incompressible Ogden material under uniaxial load, for different values of $\alpha$	383
5.22 Stress-strain curve for Hooke's linear law and a compressible Ogden material under hydrostatic load with $\nu = 0.3$	384
5.23 Plane strain and plane stress situation	388
6.1 Classical and weak solutions, minimizers and FEM	403
6.2 One triangle in space (green) and projected to plane (blue)	407
6.3 A small mesh of a simple domain in $\mathbb{R}^2$	409
6.4 Local and global numbering of nodes	409
6.5 Numbering of a simple mesh by Cuthill-McKee	410
6.6 Mesh generated by <code>triangle</code>	411
6.7 Structure of the nonzero entries in a stiffness matrix	411
6.8 A first FEM solution	412
6.9 A simple rectangular mesh	413
6.10 The two types of triangles in a rectangular mesh	413
6.11 FEM stencil and neighboring triangles of a mesh point	415
6.12 Finite difference stencil for $-u_{xx} - u_{yy}$ for $h = hx = hy$	416
6.13 A function to be minimized	422
6.14 Quadratic interpolation on a triangle	427
6.15 Transformation of standard triangle to general triangle	429
6.16 Gauss integration of order 5 on the standard triangle, using 7 integration points	431
6.17 Basis functions for second order triangular elements	434
6.18 Convergence results for linear and quadratic elements	446
6.19 The mesh and the solution for a BVP	448
6.20 Difference to the exact solution and values of $\frac{\partial u}{\partial y}$ , using a first order mesh	448
6.21 Difference to the exact solution and values of $\frac{\partial u}{\partial y}$ , using a second order mesh	448
6.22 Difference of the approximate values of $\frac{\partial u}{\partial y}$ to the exact values	449
6.23 The transformation of a linear quadrilateral element and the four nodes	452
6.24 Bilinear shape functions on a 4 node quadrilateral element	454
6.25 Gauss integration points on the standard square, using $2^2 = 4$ or $3^2 = 9$ integration points	455
6.26 The transformation of a second order quadrilateral element and the eight nodes	456
6.27 Contour levels of the shape functions on a 8 node quadrilateral element	457
6.28 The function leading to logistic growth and the solution of the differential equation	458
6.29 Exact and approximate solution of the second test problem	465
6.30 A snapshot of the solution	470
6.31 Concentration $u(t, r)$ as function of time $t$ and radius $r$ on a short time interval and contours	471
6.32 Concentration $u(t, r)$ as function of time $t$ and radius $r$ on a long time interval and contours	471
6.33 Graph of the solutions by four computations with different algorithms	474

# List of Tables

1	Literature on Numerical Methods . . . . .	4
2	Literature on the Finite Element Method . . . . .	6
1.1	Some values of heat related constants . . . . .	8
1.2	Symbols and variables for heat conduction . . . . .	9
1.3	Symbols and variables for a vibrating membrane . . . . .	14
1.4	Variables used for the stretching of a beam . . . . .	16
1.5	Typical values for the elastic constants . . . . .	17
1.6	Variables used for a bending beam . . . . .	19
2.1	Binary representation of floating point numbers . . . . .	24
2.2	Normalized timing for different operations on an Intel I7 . . . . .	27
2.3	FLOPS for a few CPU architectures, using one core only . . . . .	28
2.4	Properties of the model matrices $\mathbf{A}_n$ , $\mathbf{A}_{nn}$ and $\mathbf{A}_{nnn}$ . . . . .	36
2.5	Memory requirements for the Cholesky algorithm for banded matrices . . . . .	67
2.6	Comparison of direct solvers for $\mathbf{A}_{nn}$ with $n = 200$ . . . . .	70
2.7	Gradient algorithm . . . . .	76
2.8	The conjugate gradient algorithm . . . . .	80
2.9	Comparison of algorithms for the model problems . . . . .	84
2.10	Time required to complete a given number of flops on a 100 MFLOPS or 100 GFLOPS CPU	84
2.11	Preconditioned conjugate gradient algorithms to solve $\mathbf{A} \vec{x} + \vec{b} = \vec{0}$ . . . . .	86
2.12	The condition numbers $\kappa$ using the incomplete Cholesky preconditioner $\text{IC}(0)$ . . . . .	89
2.13	Performance for MATLAB's <code>pcg()</code> with an incomplete Cholesky preconditioner . . . . .	90
2.14	Performance for Octave's <code>pcg()</code> with an incomplete Cholesky preconditioner . . . . .	91
2.15	Timing for solving a system with $10^6 = 1'000'000$ unknowns . . . . .	91
2.16	Performance of Octave's <code>pcg()</code> with an <code>ilu()</code> preconditioner . . . . .	92
2.17	Performance of MATLAB's <code>pcg()</code> with an <code>ilu()</code> preconditioner . . . . .	92
2.18	Iterative solvers in Octave/MATLAB . . . . .	99
2.19	Benchmark of different algorithms for linear systems, used with COMSOL Multiphysics .	100
2.20	Codes for chapter 2 . . . . .	100
3.1	Comparison of methods to solve one equation . . . . .	109
3.2	Performance of some basic algorithms to solve $x^2 - 2 = 0$ . . . . .	110
3.3	Compare partial substitution method and Newton's method . . . . .	127
3.4	Standard Gaussian PDF in $n$ dimensions . . . . .	153
3.5	Approximation errors of three integration algorithms . . . . .	172
3.6	Discretization errors for the methods of Euler, Heun and Runge–Kutta . . . . .	190
3.7	A comparison of Euler and Runge–Kutta . . . . .	191
3.8	Comparing integration and solving ODEs . . . . .	192
3.9	Comparison of a Heun based method and Runge–Kutta . . . . .	207
3.10	Data for a non–stiff ODE problem with different algorithms . . . . .	214

3.11	Data for a stiff ODE problem with different algorithms . . . . .	215
3.12	Data for a stiff ODE system with different algorithms . . . . .	217
3.13	Data for a stiff ODE system with different algorithms, using the Jacobian matrix . . . . .	218
3.14	Data for non-stiff ODE system with different algorithms . . . . .	219
3.15	Commands for linear regression . . . . .	228
3.16	Examples for linear and nonlinear regression . . . . .	239
3.17	Commands for nonlinear regression . . . . .	240
3.18	Estimated and exact values of the parameters . . . . .	242
3.19	Codes for chapter 3 . . . . .	250
4.1	Finite difference approximations . . . . .	255
4.2	Exact and approximate boundary value problem . . . . .	262
4.3	Comparison of finite difference schemes for the 1D heat equation . . . . .	281
4.4	Comparison of finite difference schemes for 2D dynamic heat equations . . . . .	283
4.5	Properties of the ODE solvers . . . . .	289
4.6	Codes for chapter 4 . . . . .	303
5.1	Examples of second order differential equations . . . . .	313
5.2	Some examples of Poisson's equation $-\nabla \cdot (a \nabla u) = f$ . . . . .	316
5.3	Normal and shear strains in space . . . . .	335
5.4	Description of normal and tangential stress in space . . . . .	338
5.5	Elastic moduli and their relations . . . . .	360
5.6	Different tensors in 2D . . . . .	367
5.7	Nonlinear material laws . . . . .	369
5.8	Plane strain and plane stress situation . . . . .	388
6.1	Algorithm of Cuthill–McKee . . . . .	410
6.2	Minimization of exact and approximate problem . . . . .	421
6.3	Maximal approximation error . . . . .	465

# Index

- $\xi^2$  distribution, 152
- A–stability, 195, 285
- AbsTol, 213
- adaptive step size, 205
- Alpha 21164 processor, 30
- amd, 70
- approximation, linear, 472
- Arnoldi iteration, 93, 96
- ATLAS, 31, 40
- Banach’s fixed point theorem, 111
- bandwidth, 66, 410
- basis function, 433, 452
- BDF2, 286
- beam
- bending, 18, 297, 301, 311
  - buckling, 20
  - stretching, 15, 264, 266, 267
- Bernoulli principle, 311, 354, 363, 389, 391, 395, 396
- BiCGSTAB, 92
- bilinear function, 452
- binary representation, 24
- bisection, 105
- BLAS, 40
- Bogacki–Shampine, 204
- boundary condition, 462
  - Dirichlet, 312, 393
  - natural, 308, 311, 312, 317
  - Neumann, 312
- bracketed, 105
- Bramble–Hilbert lemma, 425
- brittle, 342
- bulk modulus, 353, 357, 359, 375
- Butcher table, 200, 202
- Céa lemma, 423
- cache structure, 26
- calculus of variations, 304
- CGNR, 92
- characteristic polynomial, 132
- chi–square distribution, 152
- Cholesky, 56
- classical, 56
- incomplete, 87, 89
- modified, 56
- classical solution, 403, 416, 417
- coercive, 421
- condition number, 50
- cone of dependence, 293
- confidence interval, 225, 231, 239
- conforming element, 421, 424, 426, 428, 453, 456
- conjugate
  - direction, 78
  - gradient algorithm, 78
  - gradient normal residual, 92
- conjugate residual, 92
- connection, shortest, 309
- consistency, 258, 259, 261
- contraction mapping principle, 111
- convergence, 259, 260, 445
  - linear, 104
  - quadratic, 104
- correlation, 161, 162
- Courant–Fischer Minimax Theorem, 135
- covariance, 155, 157, 161, 233
- Crank–Nicolson, 202, 279, 281–283, 286, 469
- cumtrapz, 164
- curvefit\_stat, 247
- Cuthill–McKee, 410
- d’Alembert, 293
- diagonalization, 133, 134
- diagonally dominant, 62
  - strictly, 62
- direct method, 73
- discretization error, 190
- displacement gradient tensor, 347
- displacement vector, 326
- distortion energy, 344
- distribution
  - chi–square, 152
  - F, 225
  - Gaussian, 152
  - Student-t, 226, 231, 239
- divergence theorem, 312

domain of confidence, 245  
Dormand–Prince, 205  
drop tolerance, 87  
ductile, 342  
  
eig, 142  
eigenvalue, 130, 131  
    generalized, 131, 148, 281  
eigenvector, 130, 131  
eigs, 143  
element, second order, 428  
elliptic, 421  
energy  
    density, 353  
    density, elastic, 349  
    norm, 76, 418, 420  
    shape changing, 352  
    volume changing, 352  
error estimate, 418  
Euler  
    implicit, 202  
Euler buckling, 21  
Euler method, 186, 187  
Euler–Lagrange equation, 21, 304, 308–311, 314, 315, 317, 319–321, 323, 361, 363, 398  
expfit, 239  
explicit, 282, 285, 289  
extrapolation  
    Richardson, 173, 205  
  
F-distribution, 227, 245  
failure mode, elastic, 342  
false position method, 106  
FEMoctave, 449  
finite difference, 253, 415  
finite difference stencil, 256  
    explicit, 257  
    implicit, 257  
Finite Element Method, FEM, 402  
Fisher’s equation, 472  
floating point arithmetic, 23  
flop, 26  
FLOPS, 26  
flux of thermal energy, 8  
fminbnd, 128  
fminsearch, 129  
Fourier’s law, 8  
Frobenius norm, 49  
fsolve, 120, 239, 242, 248  
function space, 419  
functional, 304, 305, 418, 419  
    quadratic, 311, 314, 399  
fzero, 120  
  
Gauss integration, 169, 416, 423, 429–432, 453, 455, 459  
Gauss, algorithm, 38  
Gaussian probability distribution, 152  
GenerateFEM, 463  
geometric nonlinearity, 301  
Given’s rotation, 97  
GMRES, 93  
GMRES(m), 94  
gradient algorithm, 74, 75  
Green’s identity, 312, 314  
Green–Gauss theorem, 312  
  
Hamilton’s principle, 319  
heat capacity, 8  
heat equation, 8  
    2D, 11  
    dynamic, 10, 272, 277, 279, 282  
    steady state, 10, 270  
Hesse matrix, 347  
Hessenberg matrix, 97  
Heun, 187, 201  
Hooke’s law, 18, 325, 326, 348, 355  
hourgassing, 433  
  
IC(0), 87, 89  
ichol, 89  
ICT(), 87, 89  
IEEE-754, 23  
iff, 37  
implicit, 197, 257, 258, 277, 280, 282, 285, 293  
integral, 176  
integral2, 179  
integration  
    adaptive, 173  
    Gauss, 169  
    numerical, 163, 416  
    Simpson, 165  
    trapezoidal, 163  
Intel Haswell processor, 30  
Intel I7 processor, 30  
interpolation, 459  
    bilinear, 453  
    biquadratic, 456  
    linear, 406, 425  
    piecewise linear, 424  
    piecewise quadratic, 426, 427, 465  
    quadratic, 426  
invariant, 335, 340, 344, 351, 352, 368  
irreducible, 63

- irreducibly diagonally dominant, 63  
 iterative method, 73, 103  
 iterative solver, 72
- Jacobi determinant, 430  
 Jacobian, 201
- Korn's inequality, 421, 424, 450  
 Krylov subspace, 81
- L–stability, 195, 285  
 Lagrange function, 319  
 Lagrange multiplier, 135  
 Lamé's parameter, 359  
 Laplace operator, 12  
 Lax equivalence theorem, 261, 423  
 leasqr, 239, 241, 244  
 least square, 95, 220  
 lemma, fundamental, 306  
 Levenberg–Marquardt, 110, 119, 239  
 linear element, 404  
 LinearRegression, 228, 230, 236, 238  
 loading  
     biaxial, 359, 371, 372  
     hydrostatic, 369, 374, 378, 380, 384  
     uniaxial, 369, 370, 372, 376, 379, 380, 382, 384
- logistic  
     equation, 182, 458  
     function, 243
- LR factorization, 36–38  
 lscov, 230, 233  
 lsqcurvefit, 239, 248  
 lsqnonlin, 239, 243
- mapping  
     bilinear, 451  
     linear, 346, 429
- mass spring system, damped, 185  
 matrix  
     banded, 66  
     elementary, 43  
     factorization, 36  
     inverse, 68  
     norm, 45  
     orthogonal, 49, 134, 149  
     permutation, 53  
     positive definite, 60  
     positive semidefinite, 60  
     sparse, 72  
     symmetric, 133, 134  
     triangular, 36
- maximum likelihood, 154  
 membrane  
     steady state, 15, 315  
     vibrating, 14, 315
- mesh quality, 424  
 minimal angle condition, 424, 427  
 model matrix  $\mathbf{A}_n$ , 32  
 model matrix  $\mathbf{A}_{nn}$ , 34  
 modulus of elasticity, 16, 325, 356  
 Mohr's circle, 340, 395  
 Mooney–Rivlin, 379, 380  
 multi core architecture, 30
- Neo–Hookean, 369, 372, 374, 375  
     compressible, 376, 378  
     incompressible, 369, 372
- Newton's method, 107, 115, 297, 473  
     parameterized, 301  
     damped, 119  
     modified, 119
- Newton–Raphson, 107  
 Nitsche trick, 425  
 nlinfit, 239  
 nlparci, 239, 248  
 nonlin\_curvefit, 247  
 norm, 45  
     equivalent, 46  
     matrix, 46  
     vector, 45
- normal equation, 92, 220, 221  
 normal strain, 328
- octahedral shear stress, 340  
 ode15s, 214  
 ode23, 204, 213  
 ode23s, 214  
 ode45, 185, 205, 213  
 odeget, 211, 213  
 odeset, 211  
 Ogden, 369  
     compressible, 382, 384  
     incompressible, 384
- OPENBLAS, 31, 40  
 options, 211  
 order of convergence, 104  
 orthogonal matrix, 130, 134  
 orthonormal, 130
- partial successive substitution, 114, 296  
 PCA, 156, 157, 160  
 pca, 161  
 pcacov, 161  
 pendulum, 185, 193

- double, 321
- moving support, 323
- single, 320
- permutation matrix, 53
- Picard iteration, 114, 296
- pivoting, 52, 53
  - partial, 53
  - total, 53
- Plateau problem, 318
- Poincaré's inequality, 421, 424, 450
- Poisson's ratio, 17, 326, 356, 359
- preconditioning, 84
- pressure
  - hydrostatic, 357
- principal component analysis, 156, 157
- principal strain, 334
- principal stress, 339
- principle of least action, 319
- princomp, 161
- projection operator, 424, 426
- QR factorization, 95, 221
- quad, 177
- quad2d, 179
- quadv, 178
- quiver, 185
- Rayleigh quotient, 135
- reducible, 62, 63
- regress, 229, 232
- regression
  - linear, 95, 220, 221, 224
  - nonlinear, 239, 242, 243
  - straight line, 220
- regula falsi, 106
- RelTol, 213
- residual vector, 74
- rounding error, 23
- row reduction, 38
- Runge–Kutta, 287
  - classical, 188, 201
  - embedded, 203
  - explicit, 200
  - implicit, 201
  - order 2, 187
  - order 4, 188
- Saint–Venant's principle, 389
- secant method, 106
- selection tree, 72
- semibandwidth, 66
- separation of variables, 10
- shear locking, 433
- shear modulus, 353, 358, 359
- shear strain, 328
- sigmoid function, 243
- Simpson integration, 165
- singular value decomposition, 51, 149, 224
- SIR, 209
- sparse direct solver, 69
- sparse solver, 69
- stability, 195, 202, 259, 261, 285, 292
  - conditional, 258, 276
  - backward, 52
  - conditional, 196, 198, 291
  - domain, 200
  - function, 202
  - of Cholesky, 64
  - unconditional, 197, 199, 203, 258, 295, 296
- steepest descent, 74
- stencil, 256, 414, 415
- step size
  - adaptive, 205
- stiff ODE, 213
- stiffness matrix
  - element, 405, 407, 408
  - global, 405, 428
- strain, 16, 18, 327
  - invariant, 335, 340
  - plane, 388
  - principal, 333, 334, 359
  - tensor, 346, 364
- stress, 18, 336
  - engineering, 17, 125, 371
  - hydrostatic, 344, 361
  - matrix, 339
  - maximal principal, 343
  - normal, 336
  - plane, 388, 394
  - principal, 339, 359
  - shape changing, 344, 361
  - tangential, 336
  - tensor, 337, 346
  - Tresca, 340–342
  - true, 17, 125, 371
  - volume changing, 361
  - von Mises, 340, 342
  - yield, 343
- stretch
  - principal, 368, 378
- string
  - deformation, 13, 310
  - vibrating, 14

Students's t-distribution, 226  
successive substitution, 111  
superconvergence, 445, 465  
surface forces, 354  
svd, 51, 149, 160, 224  
  
templates, 85  
tensor, 344

- Cauchy–Green deformation, 365, 366
- deformation gradient, 364
- displacement gradient, 347, 364, 366
- Green strain, 365, 366, 368
- infinitesimal strain, 329, 366
- infinitesimal stress, 366

thermal conductivity, 8  
Tikhonov regularization, 21  
time discretization, 469  
tolerance, 213

- absolute, 213
- relative, 213

trapz, 164  
Tresca stress, 340, 343  
triangle, 408  
triangular matrix, 36  
triangularization, 408  
tube, torsion, 361  
tumor growth, 458  
  
unit roundoff, 24  
  
variance

- of parameters, 225

vector

- norm, 45
- outer unit normal, 315, 417

Volterra–Lotka, 183  
volume forces, 354  
von Mises stress, 340, 344  
  
wave equation, 289  
weak solution, 402, 416–419  
well conditioned problem, 51  
  
Young's modulus, 325, 359