Computational Methods for PDEs Summer School 2019

August 3–5 2019

Introduction

Goals:

- Numerical solutions of PDEs:
  overview, some analysis, solution approaches
- introduction to the deal.II Finite Element Library

Plan:

- each module contains slides, live demos, and exercises
- Remember: ask questions and work together (groups of 2)

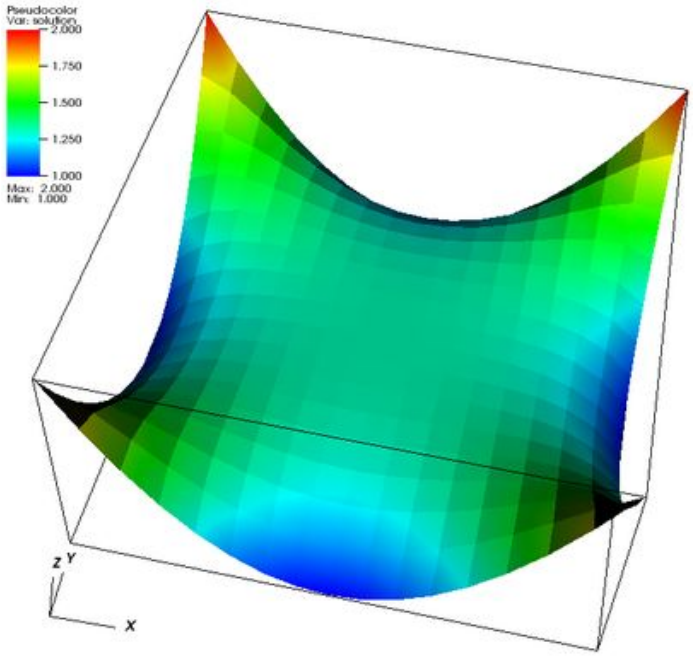
Resources

- Program: https://dealii.org/pdeschool2019
- Manual, tutorials: https://dealii.org/current/doxygen/deal.II/index.html
- repository for this program with slides, exercises, etc.: https://github.com/tjhei/pdeschool
- after opening the virtual machine and setting it up, you will have the repo above in pdeschool2019/ in your home directory
- people: peers, lecturers

What are PDEs?

- Partial Differential Equations (PDEs)
- describe physical phenomena, often based on energy balance or conservation laws
- Equations holding pointwise in a domain $\Omega \subset \mathbb{R}^d$
- optionally depending on time
- Equations contain partial derivatives
- Solutions are functions
- Example:
- find $u : \Omega \to \mathbb{R}$ with

$$
\begin{aligned}
-\Delta u &= f && \text{in } \Omega, \\
u &= 0 && \text{on } \partial\Omega.
\end{aligned}
$$



Examples for PDEs

- stationary:
  - Laplace / Poisson equation
  - (linear) elasticity
  - Stokes
  - Darcy
- Instationary (time dependent):
  - heat equation
  - convection-diffusion-reaction equation
  - Navier-Stokes equation
  - wave equation
  - …
- Multiphysics / coupled systems

Numerical Solutions

- what?
  - find approximations to the solutions using a computer
- why?
  - hard or impossible to find analytical solutions, especially on complicated domains
- how?
  - we will be focussing on the Finite Element Method

Important in Numerical Analysis

Important questions about solutions:

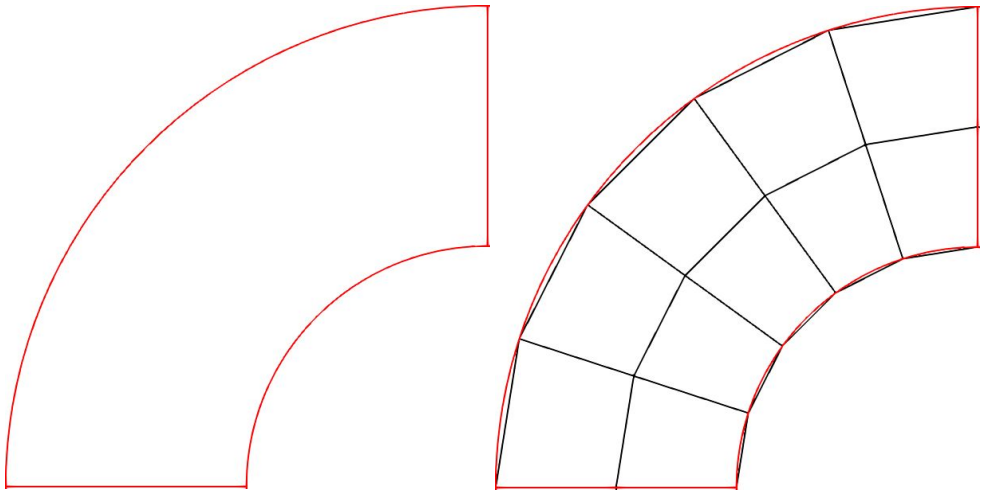
1. Existence
2. Uniqueness
3. Continuous dependency on data
4. Regularity of solutions

   1. & 2. & 3. ? => wellposedness

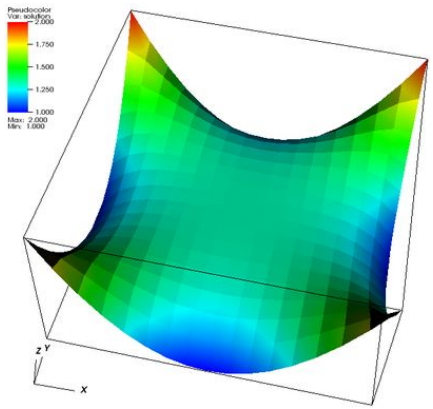
- Ask these questions before trying to solve a PDE!
- Important consequences for numerical method!

Summary of Finite Element Method

- Discretize domain using a mesh of triangles, quadrilaterals, etc..
- Define a discrete space to represent solutions using the mesh
- Implement a weak form of the PDE and turn it into a linear system
- Postprocess solutions (graphical output, compute statistics and errors, etc.)



$$-\triangle u = f \quad AU = F$$

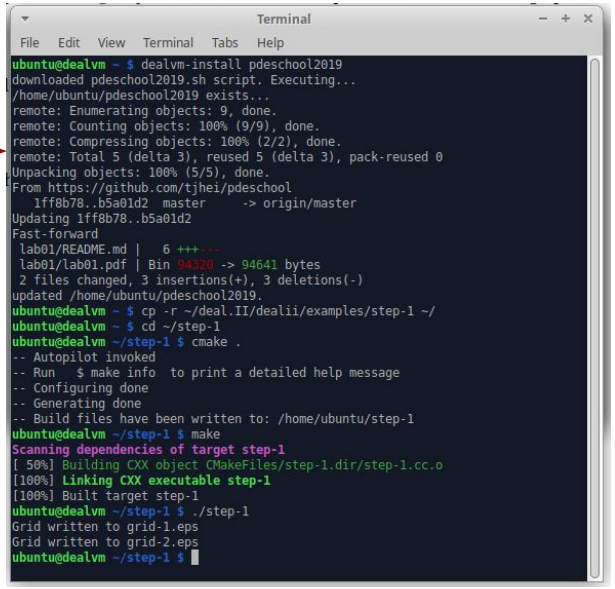


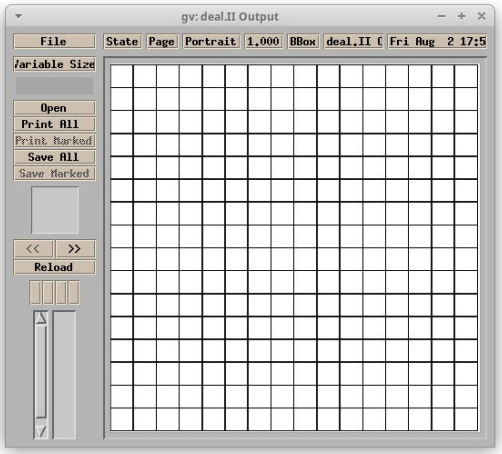
1-slide participant introductions

Using the Virtual Machine

Exercise time!

1. Start VM
2. Open terminal, type “dealvm-install pdeschool2019”
3. Follow “2. The Virtual machine” in the pdf you find at: pdeschool2019/lab01/lab01.pdf
4. Type “gv grid-1.eps”





Resources:
https://github.com/tjhei/pdeschool/tree/master/lab01

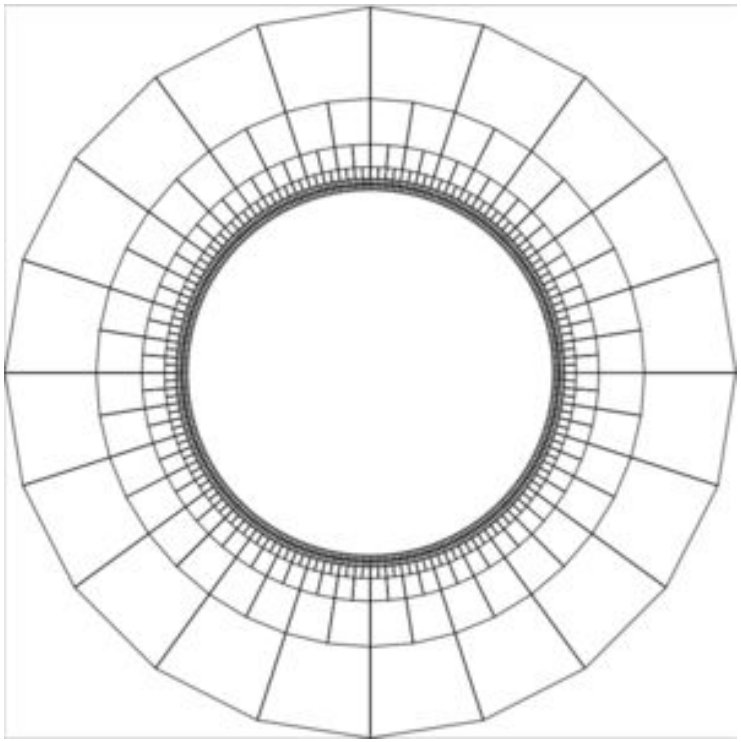
BREAK

Let us know if you have any problems/questions!

LAB 01

- based on deal.II step-1
- key points:
  - your first deal.II program!
  - including deal.II headers
  - creating a triangulation
  - mesh topology
  - traversing a triangulation and manipulating it
  - visualizing meshes

Exercise time!



1.1 Stationary PDEs

We will now derive the Poisson equation from the heat equation

Derivation of the Heat Equation

Conservation of heat energy can be summarized as:

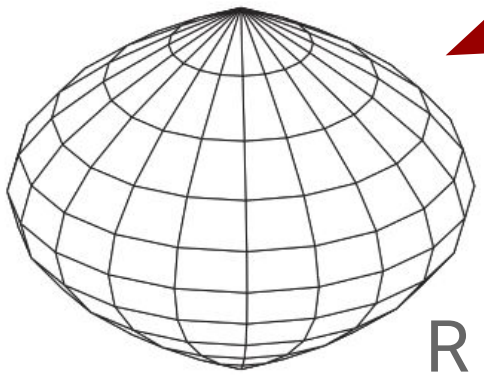
Rate of change of heat energy	=	heat energy flowing across the boundaries per unit time	+	heat energy generated inside per unit time

(following Haberman (2004) Applied partial differential equations with Fourier series and boundary value problems, pp. 20-24)

Derivation of the Heat Equation

Heat energy within an arbitrary subregion R:

$$\iiint_R c\rho u \, dV,$$



R

specific heat capacity

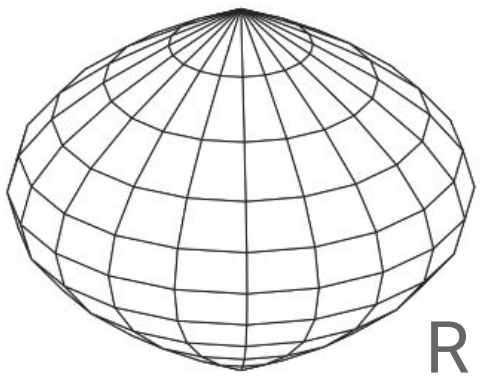
density

temperature

Derivation of the Heat Equation

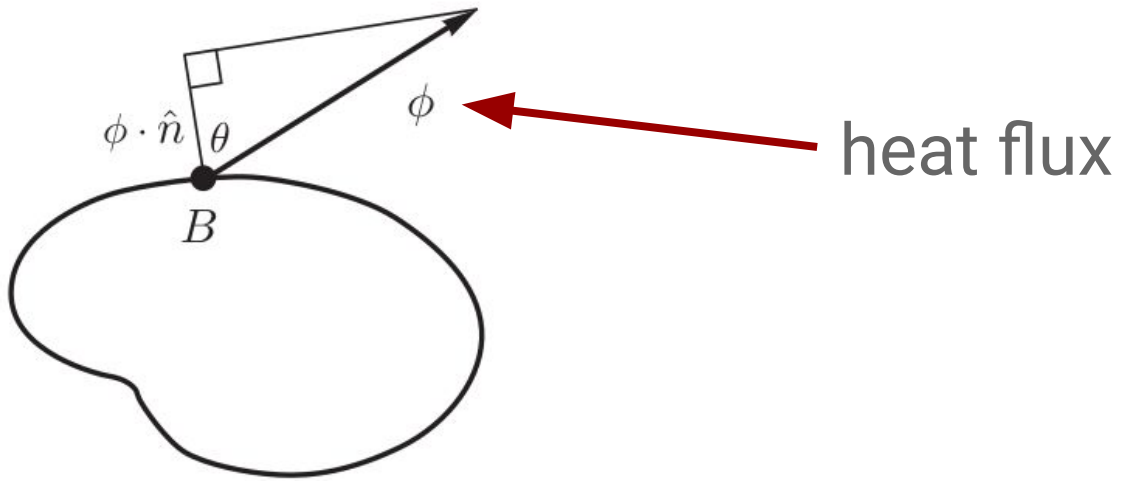
Heat energy within an arbitrary subregion R:

$$\iiint_R c\rho u \, dV,$$



Outward normal component of the heat flux vector:

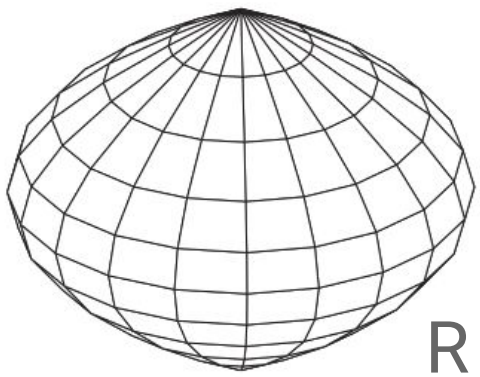
$$\boldsymbol{\phi} \cdot \hat{\boldsymbol{n}}$$



Derivation of the Heat Equation

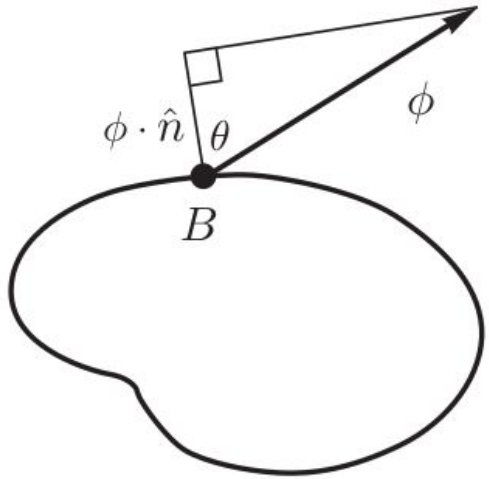
Heat energy within an arbitrary subregion R:

$$\iiint_R c\rho u \, dV,$$



Outward normal component of the heat flux vector:

$$\boldsymbol{\phi} \cdot \hat{\boldsymbol{n}}$$



Rate of heat energy generated per unit volume:

$$Q$$

Derivation of the Heat Equation

Conservation of heat energy can be summarized as:

Rate of change of heat energy = heat energy flowing across the boundaries per unit time + heat energy generated inside per unit time

$$\frac{d}{dt}\iiint_R c\rho u \, dV = -\oiint \boldsymbol{\phi} \cdot \hat{\boldsymbol{n}} \, dS + \iiint_R Q \, dV$$

Derivation of the Heat Equation

$$\frac{d}{dt}\iiint_R c\rho u \, dV = -\oiint \boldsymbol{\phi} \cdot \hat{\boldsymbol{n}} \, dS + \iiint_R Q \, dV$$

Use divergence theorem:

$$\frac{d}{dt}\iiint_R c\rho u \, dV = -\iiint_R \nabla \cdot \boldsymbol{\phi} \, dV + \iiint_R Q \, dV.$$

Use that all expressions are are volume integrals over the same volume:

$$\iiint_R \left[c\rho \frac{\partial u}{\partial t} + \nabla \cdot \boldsymbol{\phi} - Q \right] \, dV = 0.$$

Derivation of the Heat Equation

$$\frac{d}{dt}\iiint_R c\rho u \, dV = -\oiint \boldsymbol{\phi} \cdot \hat{\boldsymbol{n}} \, dS + \iiint_R Q \, dV$$

Use divergence theorem:

$$\frac{d}{dt}\iiint_R c\rho u \, dV = -\iiint_R \nabla \cdot \boldsymbol{\phi} \, dV + \iiint_R Q \, dV.$$

Use that all expressions are are volume integrals over the same volume:

(for all regions R)

$$c\rho \frac{\partial u}{\partial t} = -\nabla \cdot \boldsymbol{\phi} + Q.$$

Derivation of the Heat Equation

$$c\rho\frac{\partial u}{\partial t} = -\nabla \cdot \boldsymbol{\phi} + Q.$$

Using Fourier's law of heat conduction: $\boldsymbol{\phi} = -K_0\nabla u$,
(the heat flux is proportional to the gradient of the temperature)

$$c\rho\frac{\partial u}{\partial t} = \nabla \cdot (K_0\nabla u) + Q.$$

Without any heat sources (*Q=0*) and constant coefficients:

$$\frac{\partial u}{\partial t} = k\nabla^2 u.$$

Heat equation

thermal conductivity

Derivation of the Laplace Equation

$$c\rho\frac{\partial u}{\partial t} = \nabla \cdot (K_0 \nabla u) + Q.$$

In steady state (and for constant coefficients),
the equilibrium temperature distribution is:

$$\boxed{\nabla^2 u = -\frac{Q}{K_0},}$$

Poisson's equation

Laplace/Poisson Equation

$$-\triangle u = f, \quad u = 0 \text{ on } \partial\Omega$$

- used in:
  - displacement of membrane under forces (gravity)
  - heat distribution in equilibrium
  - distribution of electrical charge
  - etc..

1.2 The Variational Form

- The strong form:

$$-\triangle u = f, \quad u = 0 \text{ on } \partial\Omega$$

implies $u \in C^2(\Omega)$

- restrictive with respect to domain and *f*
- How to relax this? “Variational” or “weak” form

Towards the variational form

- multiply with arbitrary test function *v* from left, integrate over the domain:

$$\int_\Omega v(-\triangle u)\mathrm{d}x = \int_\Omega v f \mathrm{d}x$$

- use Green's formula:

$$\int_\Omega \nabla v \cdot \nabla u \mathrm{d}x - \int_{\partial\Omega} v(\nabla u \cdot n)\mathrm{d}s = \int_\Omega v f \mathrm{d}x$$

- Variational form:

$$\text{find } u \in V : (\nabla v, \nabla u) = (v, f) \quad \forall v \in V$$

Variational form

$$\text{find } u \in V : (\nabla v, \nabla u) = (v, f) \quad \forall v \in V$$

- Weaker requirements (“weak form”)
- We need $\int_\Omega v^2 \mathrm{d}x < \infty$ $\int_\Omega \nabla u \cdot \nabla u \mathrm{d}x < \infty$
- “square-integrable functions”

$$L^2(\Omega) = \{v \mid (v, v) = \int_\Omega v^2 \mathrm{d}x < \infty\}$$

  form a Banach space with norm $\|v\| := (v, v)^{1/2}$
- Sobolev space + boundary conditions:

$$V \subset H^1(\Omega) = \{v \in L^2(\Omega) : \nabla u \in L^2(\Omega)\}$$

- Note: derivatives meant in a weak sense

$$\|v\|_1^2 := \|v\|^2 + \|\nabla v\|^2$$

Boundary conditions

- Needed for unique solvability!
- easiest for theory: homogeneous Dirichlet $u = 0$
- Dirichlet: $u = g$
- Neumann: $\nabla u \cdot n = g$
- Robin: $a\nabla u \cdot n + bu = g$
- With Dirichlet we will use:

$$V = H_0^1(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}$$

- Note: accurate definition of $v = 0$ on $\partial\Omega$ is non-trivial

Finite vs infinite dimensional spaces

- need to represent as finite dimensional space with N basis functions (“conforming”):

$$V_h = \{\phi_i : \Omega \to \mathbb{R}\} \subset V$$

- Represent $u_h \in V_h$ using coefficients $u_1, \ldots, u_N$ as

$$u_h(x) = \sum_{i=1}^{N} u_i \phi_i(x)$$

- choice of V_h gives different methods: FEM, spectral, ...

1.3 Existence, Convergence, Error estimates

- general theory for elliptic PDEs
- gives existence, uniqueness, and error estimates
- very little assumptions on V_h for now

Hilbert space setting

- Hilbert space V with inner product (,) and norm $\|\cdot\|$
- Variational problem $a(v,u) = f(v) \forall v \in V$

- bilinear form $a : V \times V \to \mathbb{R}$
  - continuous: $a(v,w) \leq \alpha \|v\| \|w\|$
  - coercive: $a(v,v) \geq \kappa \|v\|^2, \kappa > 0$

- linear form $f : V \to \mathbb{R}$
  - continuous $f(v) \leq \gamma \|v\|$

Lax-Milgram

Under the assumptions from above, the variational problem $a(v,u) = f(v) \forall v \in V$ has a unique solution $u \in V$ with

$$\|u\| \leq \frac{1}{\kappa} \|f\|_{V^*}$$

This works for $V = H_0^1(\Omega)$ and V_h for Laplace and other elliptic problems.

Cea's Lemma

Best approximation of the error:

$$\|u_h - u\| \leq \frac{\alpha}{\kappa} \inf_{\phi \in V_h} \|u - \phi\|$$

With a discretization that converges

$$\inf_{\phi \in V_h} \|u - \phi\| \to 0 \ (\text{for } h \to 0)$$

we have that our approximation converges as well!

1.4 Finite Element Method

- how to define the discrete space?
- how to implement it?

Implementing the finite element method

Brief re-hash of the FEM, using the Poisson equation:

We start with the strong form:

$$\begin{aligned} -\Delta u &= f \quad && \text{in } \Omega \\ u &= 0 && \text{on } \partial\Omega \end{aligned}$$

Implementing the finite element method

Brief re-hash of the FEM, using the Poisson equation:

We start with the strong form:

$$\begin{aligned} -\Delta u &= f \quad && \text{in } \Omega \\ u &= 0 \quad && \text{on } \partial\Omega \end{aligned}$$

...and derive the weak form by multiplying the equation by a test function φ from the left and integrating over the domain:

$$-\int_\Omega \varphi \Delta u = \int_\Omega \varphi f.$$

Implementing the finite element method

Brief re-hash of the FEM, using the Poisson equation:

We start with the strong form:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

...and derive the weak form by multiplying the equation by a test function φ from the left and integrating over the domain:

$$-\int_\Omega \varphi \Delta u = \int_\Omega \varphi f.$$

Integrate by parts:

$$\int_\Omega \nabla\varphi \cdot \nabla u - \int_{\partial\Omega} \varphi \mathbf{n} \cdot \nabla u = \int_\Omega \varphi f.$$

Implementing the finite element method

$$\int_\Omega \nabla\varphi \cdot \nabla u - \int_{\partial\Omega} \varphi \mathbf{n} \cdot \nabla u = \int_\Omega \varphi f.$$

The test function φ has to satisfy the same boundary conditions, so on the boundary *φ=0*:

Consequently the weak form reads:

$$(\nabla \varphi, \nabla u) = (\varphi, f) \quad \forall \varphi$$

where we have used the notation $(a, b) = \int_\Omega a\, b.$

The solution of this is a function *u(x)* from an infinite-dimensional function space.

Implementing the finite element method

Since computers can't handle objects with infinitely many coefficients, we seek a finite dimensional function of the form

$$u_h(x) = \sum_{j=1}^{N} U_j \varphi_j(x)$$

approximation

unknown expansion coefficients ("degrees of freedom")

finite element shape functions

Implementing the finite element method

Since computers can't handle objects with infinitely many coefficients, we seek a finite dimensional function of the form

$$u_h(x) = \sum_{j=1}^{N} U_j \varphi_j(x)$$

Go from the weak form $(\nabla \varphi, \nabla u) = (\varphi, f) \quad \forall \varphi$ to the weak form of the discrete problem:

$$(\nabla \varphi_i, \nabla u_h) = (\varphi_i, f) \quad \forall i = 1 \ldots N$$

by testing with N basis functions. If the basis functions are linearly independent, this yields N equations for N coefficients. This is called the Galerkin method.

Implementing the finite element method

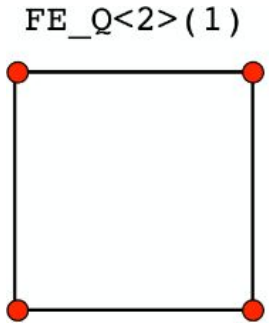
Practical question 1: How to define the basis functions?

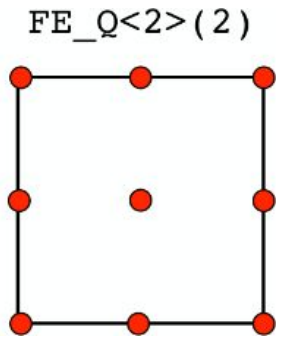
Answer: In the finite element method, this is done using the following concepts:

- Subdivision of the domain into a mesh (Lab 01)
- Definition of basis functions on the reference cell
- Each cell of the mesh is a mapping of the reference cell
- Each shape function corresponds to a degree of freedom on the global mesh

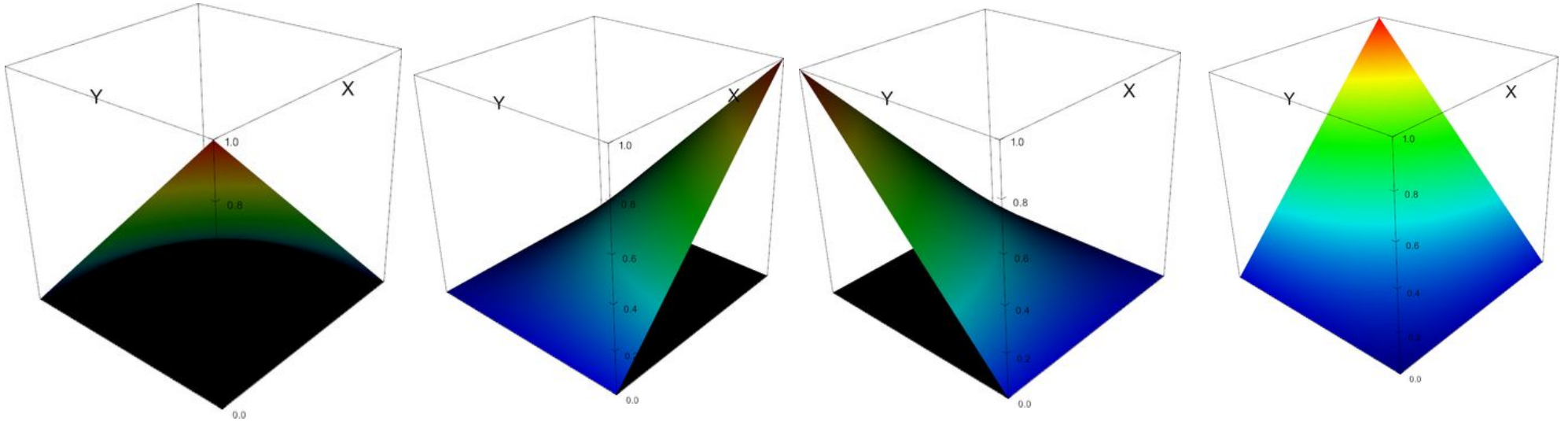
Basis function on the reference cell

Linear and quadratic elements and support points:





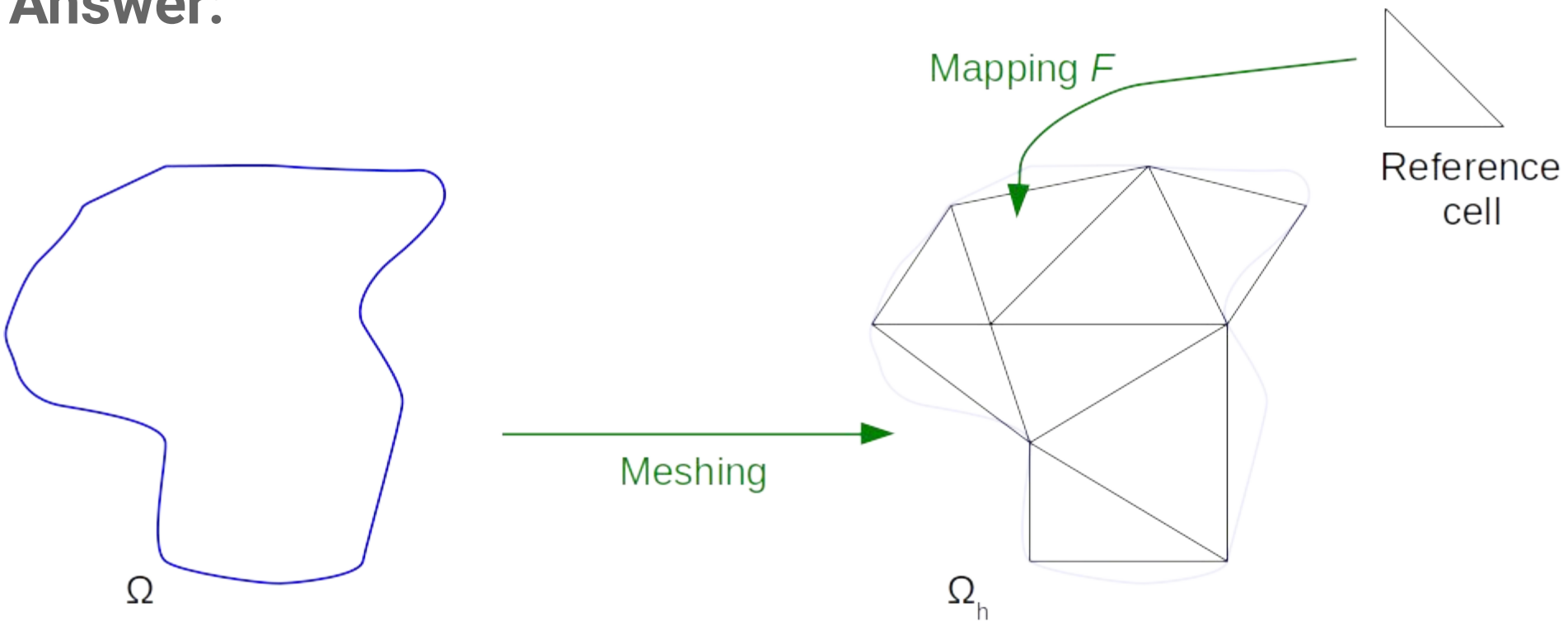
Basis functions for linear space:



Mesh cell: mapping of the reference cell

Practical question 1: How to define the basis functions?

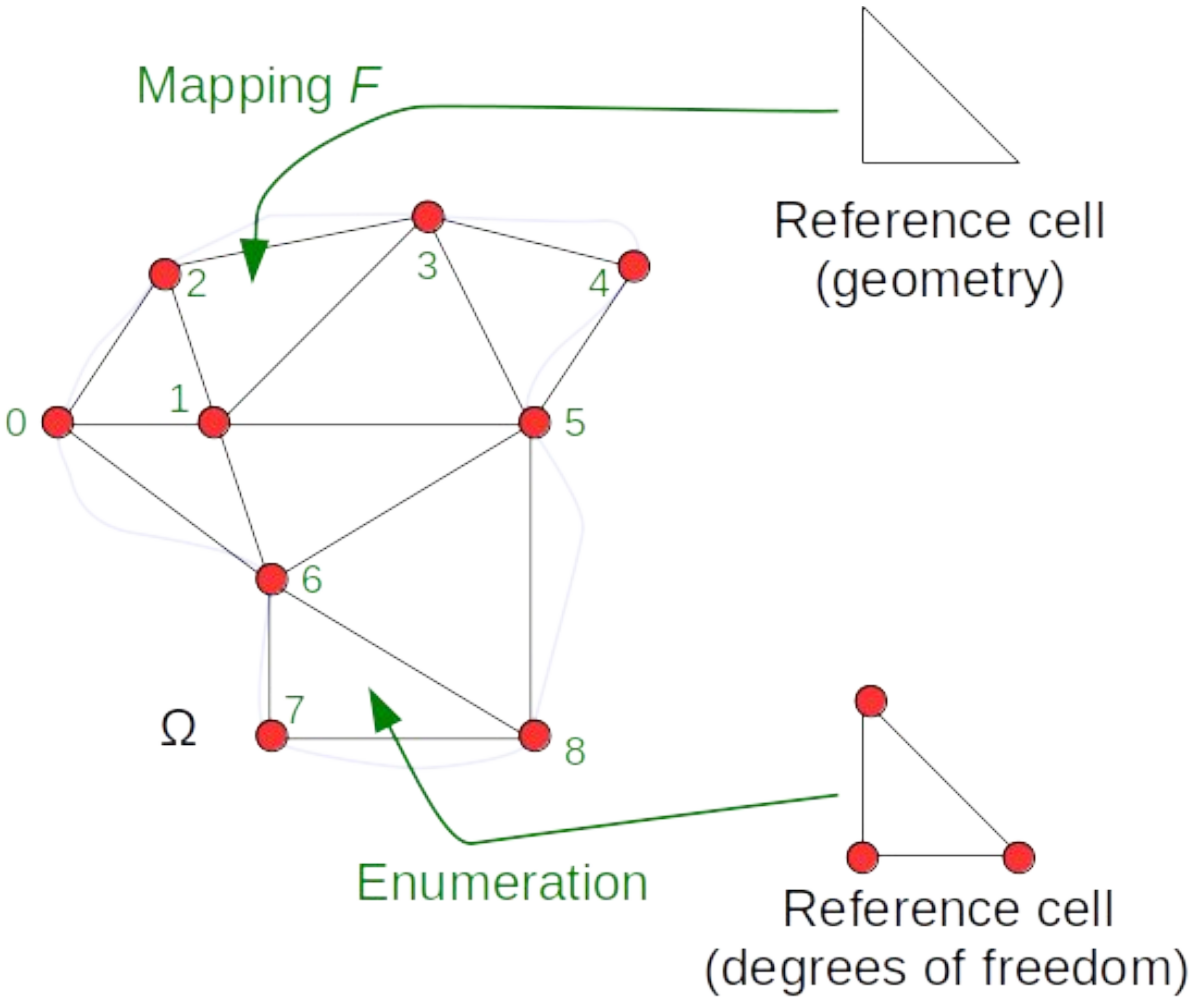
Answer:



Enumerating local DoFs globally

Practical question 1: How to define the basis functions?

Answer:



Implementing the finite element method

Practical question 1: How to define the basis functions?

Answer: In the finite element method, this is done using the following concepts:

- Subdivision of the domain into a mesh (Lab 01)
- Definition of basis functions on the reference cell
- Each cell of the mesh is a mapping of the reference cell
- Each shape function corresponds to a degree of freedom on the global mesh

Concepts in red will correspond to things we need to implement in software, explicitly or implicitly.

Implementing the finite element method

Given the definition $u_h = \sum_{j=1}^{N} U_j \varphi_j(x)$, we can expand the bilinear form

$$(\nabla \varphi_i, \nabla u_h) = (\varphi_i, f) \qquad \forall i = 1 \ldots N$$

to obtain:

$$\sum_{j=1}^{N} (\nabla \varphi_i, \nabla \varphi_j) U_j = (\varphi_i, f) \qquad \forall i = 1 \ldots N$$

This is a linear system

$$A U = F$$

with the matrix and right-hand side

$$A_{ij} = (\nabla \varphi_i, \nabla \varphi_j) \qquad F_i = (\varphi_i, f)$$

Implementing the finite element method

Practical question 2: How to compute

$$A_{ij} = (\nabla \varphi_i, \nabla \varphi_j) \qquad F_i = (\varphi_i, f)$$

First, split the integral over Ω into integrals over all cells:

$$\begin{aligned} A_{ij} &= (\nabla \varphi_i, \nabla \varphi_j) \\ &= \sum_K \int_K \nabla \varphi_i(x) \cdot \nabla \varphi_j(x) \end{aligned}$$

sum over all cells

Implementing the finite element method

Practical question 2: How to compute

$$A_{ij} = (\nabla \varphi_i, \nabla \varphi_j) \qquad F_i = (\varphi_i, f)$$

Then, map back to the reference cell...

$$\begin{aligned} A_{ij} &= (\nabla \varphi_i, \nabla \varphi_j) \\ &= \sum_K \int_K \nabla \varphi_i(x) \cdot \nabla \varphi_j(x) \\ &= \sum_K \int_{\hat{K}} J_K^{-1}(\hat{x}) \hat{\nabla} \hat{\varphi}_i(\hat{x}) \cdot J_K^{-1}(\hat{x}) \hat{\nabla} \hat{\varphi}_j(\hat{x}) \, |\det J_K(\hat{x})| \end{aligned}$$

mapping

shape function on reference cell

Need to compute the integrals!

Quadrature

Approximate each cell's contribution by quadrature (we replace the integrals by a weighted sum over a set of points on each cell):

$$A_{ij}^K = \int_K \nabla\varphi_i \cdot \nabla\varphi_j \approx \sum_q \nabla\varphi_i(\mathbf{x}_q^K) \cdot \nabla\varphi_j(\mathbf{x}_q^K) w_q^K,$$

qth quadrature point on cell K

qth quadrature weight

- locations and weights on the reference cell are described by a quadrature formula (i.e. a set of points and weights)

Quadrature

Map back to the reference cell...

$$
\begin{aligned}
A_{ij} &= (\nabla \varphi_i, \nabla \varphi_j) \\
&= \sum_K \int_K \nabla \varphi_i(x) \cdot \nabla \varphi_j(x) \\
&= \sum_K \int_{\hat{K}} J_K^{-1}(\hat{x}) \hat{\nabla} \hat{\varphi}_i(\hat{x}) \cdot J_K^{-1}(\hat{x}) \hat{\nabla} \hat{\varphi}_j(\hat{x}) \left|\det J_K(\hat{x})\right|
\end{aligned}
$$

...and quadrature:

$$
A_{ij} \approx \sum_K \sum_{q=1}^{Q} \underbrace{J_K^{-1}(\hat{x}_q) \hat{\nabla} \hat{\varphi}_i(\hat{x}_q)}_{=\nabla \varphi_i(x_q)} \cdot \underbrace{J_K^{-1}(\hat{x}_q) \hat{\nabla} \hat{\varphi}_j(\hat{x}_q)}_{=\nabla \varphi_j(x_q)} \underbrace{\left|\det J_K(\hat{x}_q)\right| \; w_q}_{=:\mathrm{JxW}}
$$

Similarly for the right hand side F.

Implementing the finite element method

Practical question 3: How to store the matrix and vectors of the linear system

$$AU = F$$

Answers:

- A is sparse, so store it in compressed row format
- U,F are just vectors, store them as arrays
- Implement efficient algorithms on them, e.g. matrix-vector products, preconditioners, etc.
- For large-scale computations, data structures and algorithms must be parallel

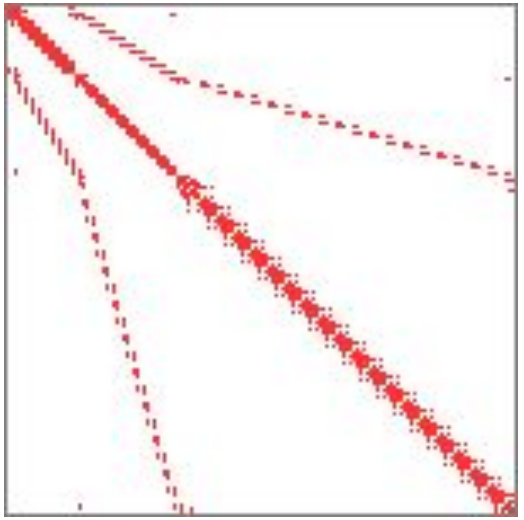
1.4 towards LAB 02: distributing DoFs

- what are the potential non-zero entries in the matrix A?

$$A_{ij} = (\phi_i, \phi_j) \neq 0$$

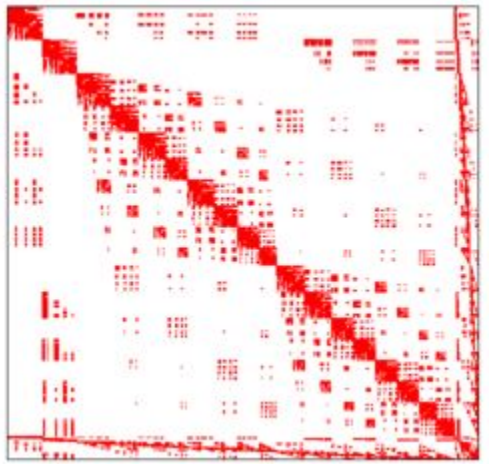
- answer: joint support between two basis functions on the same cell
- steps:

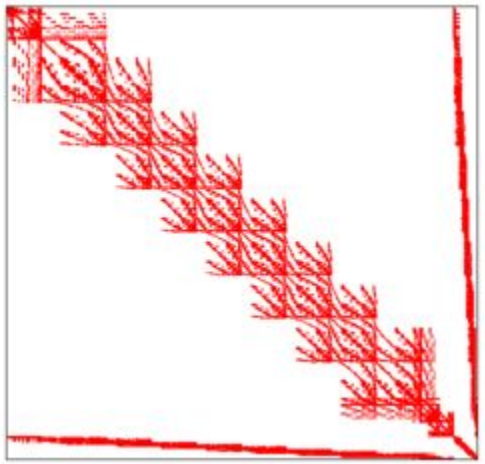
1. distribute unknowns (DoFHandler)
2. compute “sparsity pattern” (non-zero entries in the matrix)
3. visualize it



Renumbering of DoFs

- permutation of rows/columns
- same solution of linear system
- why?
  - reduce bandwidth
  - collect components of PDE
  - create block structure for preconditioning
  - parallel partitioning

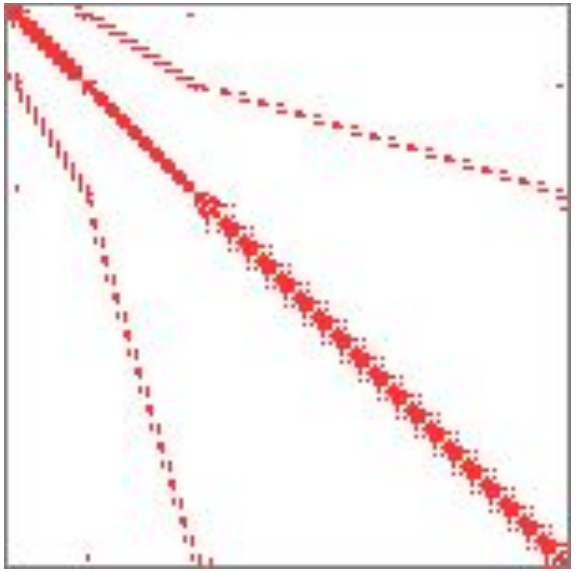




LAB 02

- based on step-2
- see lab02.pdf
- key points:
  - choosing a Finite Element
  - distributing degrees-of-freedom on a mesh
  - renumbering degrees of freedom
  - visualizing sparsity patterns

Exercise time!



1.4 towards LAB 03: solving Laplace eqn

To solve the Laplace equation we need to:

1. create storage for the matrix
2. assemble the linear system
3. solve the linear system
4. generating graphical output

How to efficiently compute A_{ij} ?

$$A_{ij} = (\nabla\phi_i, \nabla\phi_j)$$

$$A_{ij} \approx \sum_K \sum_q J_K^{-1}(x_q)\nabla\phi_i(x_q) \cdot J_K^{-1}(x_q)\nabla\phi_j(x_q) \cdot |detJ(x_q)|w_q$$

in pseudo-code:

```
for i=0,...,N-1:
 for j=0,...,N-1:
  for all K:
    A_ij += \sum_q grad_phi(i,q) grad_phi(j,q) JxW(q)
```

But most of these contribution are zero. So we switch the order of the loops to get

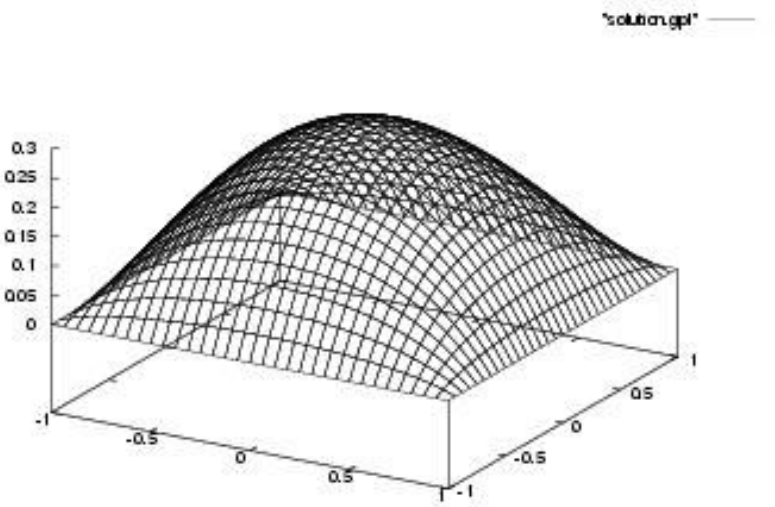
```
for all K:
 for i = 0,...,N-1:
  for j = 0,...,N-1:
    A_ij += \sum_q grad_phi(i,q) grad_phi(j,q) JxW(q)
```

which I can simplify to only look at non-zero basis functions:

```
for all K:
 a = 0
 for alpha = 0,...,n_local_dofs:
  for beta = 0,...,n_local_dofs:
   for q:
    a_{alpha,beta} += grad_phi(alpha,q) grad_phi(beta,q) JxW(q)
 A_ij += a
```

LAB 03

- see lab03.pdf, based on step-3 in deal.II
- key points:
  - assembling and solving our first PDE
  - applying boundary conditions
  - implementing right-hand sides
  - outputting graphical results



Exercise time!

BREAK

Let us know if you have any problems/questions!

1.4 FEM error estimates

- what do we know about errors and convergence of the error when you keep refining the mesh?
- does it help to use finite elements of higher order?

a priori estimates

Cea's lemma: $$\|u_h - u\| \leq \frac{\alpha}{\kappa} \inf_{\phi \in V_h} \|u - \phi\|$$

For finite element space of degree k and and certain assumptions on the geometry (shape-regular, conforming, convex), and $u \in H^{k+1}(\Omega)$:

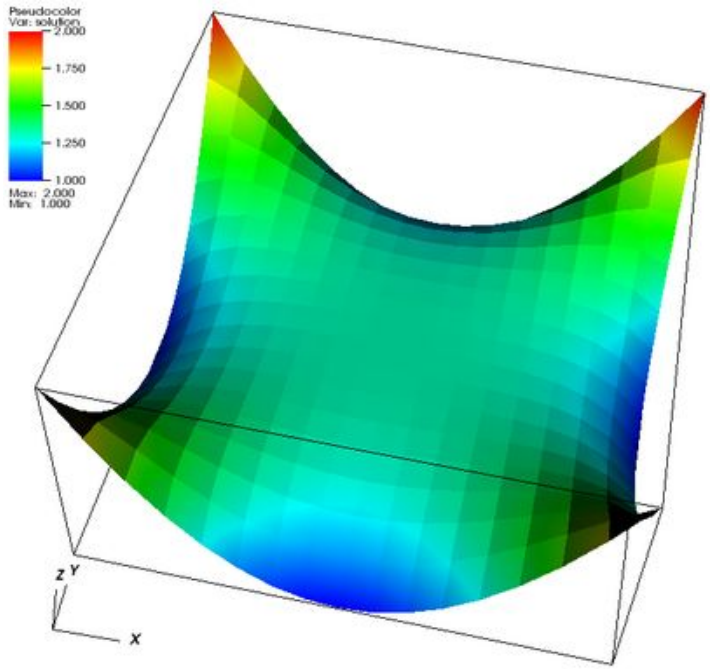
$$\|u - u_h\|_1 \leq Ch^k |u|_{k+1}$$

$$\|u - u_h\| \leq Ch^{k+1} |u|_{k+1}$$

So, for smooth problems and FE degree k, L2 norms converge with order k+1, and H1 norm with order k.

1.5 towards LAB 04

- key points:
  - Modified step-4 to check correctness
  - Using the method of manufactured solutions
  - Computing L2 and H1 errors and check convergence orders



Manufactured Solutions

- think of any function u(x), the **exact solution**
- compute right-hand side to fulfill the PDE
- apply boundary conditions of exact solution
- solve the PDE numerically
- finally: measure the error between the computed and exact solution

Computing Errors

- Important for code verification!
- See step-7 for details
- We set up the problem with analytical solution and implement it as a Function<dim>
- Quantities or interest:
  - Break it down as one operation per cell and the “summation” (local and global error)
  - Need quadrature to compute integrals

Computing Errors

Code:

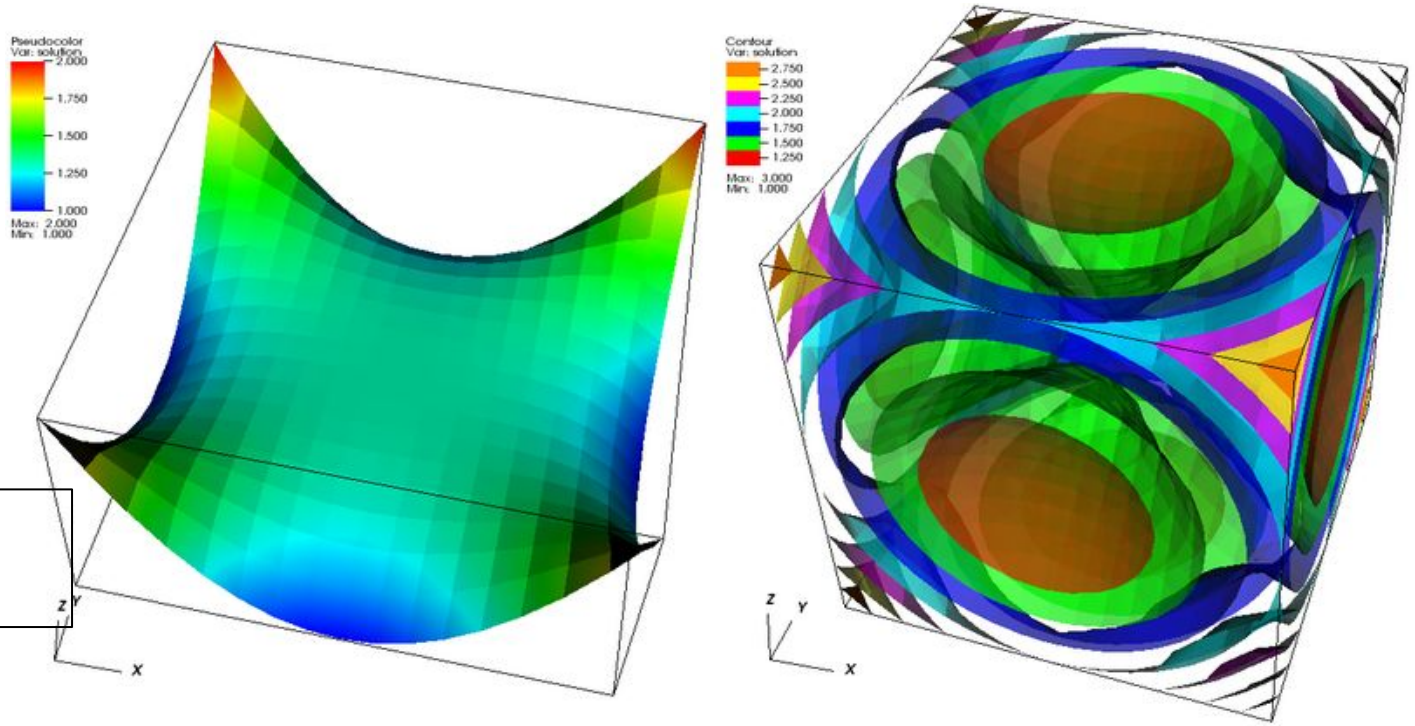
```
Vector<float> difference_per_cell (triangulation.n_active_cells());

VectorTools::integrate_difference (dof_handler,
                                   solution,                  // solution vector
                                   Solution<dim>(),           // reference solution
                                   difference_per_cell,
                                   QGauss<dim>(3),            // quadrature
                                   VectorTools::L2_norm); // local norm

 const double L2_error = VectorTools::compute_global_error(triangulation,
                                                           difference_per_cell,
                                                           VectorTools::L2_norm);
```

norms:

mean, L1_norm, L2_norm,Linfty_norm, H1_seminorm, H1_norm, ...

LAB 04

- key points:
  - Modified step-4 to check correctness
  - Using the method of manufactured solutions
  - Computing L2 and H1 errors and check convergence orders



Exercise time!