Numerical Solution of PDEs Using the Finite Element Method

May 15 – 19 2017

Martin Kronbichler
(kronbichler@lnm.mw.tum.de)
Luca Heltai (luca.heltai@sissa.it)

Motivations

Many of the big problems in scientific computing are described by partial differential equations (PDEs):

- Structural statics and dynamics
  – Bridges, roads, cars, …
- Fluid dynamics
  – Ships, pipe networks, …
- Aerodynamics
  – Cars, airplanes, rockets, …
- Plasma dynamics
  – Astrophysics, fusion energy
- But also in many other fields: Biology, finance, epidemiology, ...

Numerics for PDEs

There are 3 standard tools for the numerical solution of PDEs:

- Finite element method (FEM)
- Finite volume method (FVM)
- Finite difference method (FDM)

Common features:

- Split the domain into small volumes (cells)
- Define balance relations on each cell
- Obtain and solve very large (non-)linear systems

Problems:

- Every code has to implement these steps
- There is only so much time in a day
- There is only so much expertise anyone can have

Ideal Characteristics

Examples of what we would like to have:

- Adaptive meshes
- Realistic, complex geometries

- Quadratic or even higher order elements

- Multigrid solvers
- Scalability to 1000s of processors
- Efficient use of current hardware

- Graphical output suitable for high quality rendering

Q: How can we make all of this happen in a single code?

Main Question!

Q: How can we make all of this happen in a single code?

Not a question of feasibility but of how we develop software:

- Is every student developing their own software?
- Or are we re-using what others have done?

- Do we insist on implementing everything from scratch?
- Or do we build our software on existing libraries?

There has been a major shift on how we approach the second question in scientific computing over the past 10-15 years!

MHPC
Master in High Performance Computing

Main Answer…

The secret to good scientific software is (re)using existing libraries!

SISSA

ICTP

Existing Software

There is excellent software for almost every purpose!

Basic linear algebra (dense vectors, matrices):

- BLAS
- LAPACK

Parallel linear algebra (vectors, sparse matrices, solvers):

- PETSc
- Trilinos

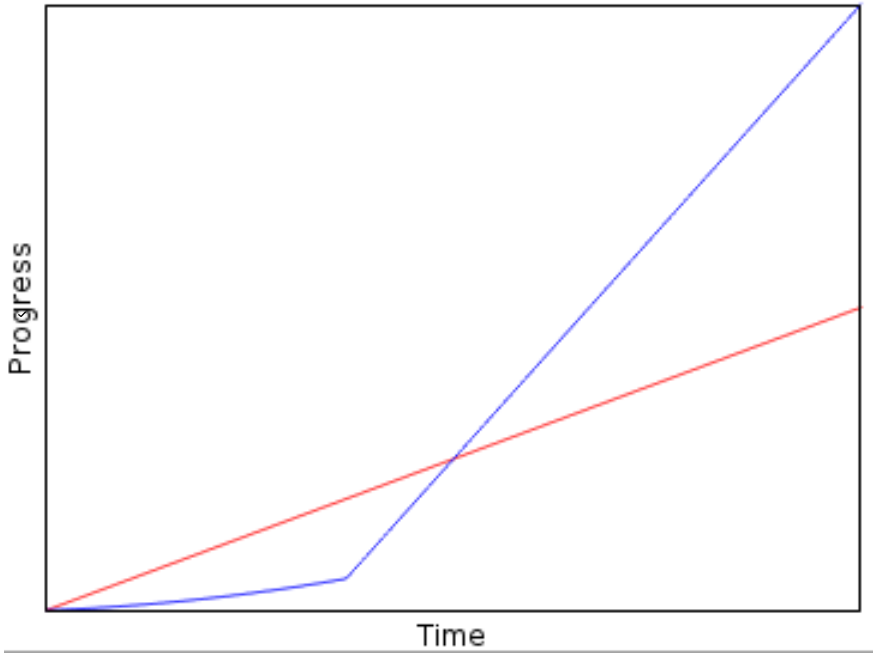
Meshes, finite elements, etc:

- deal.II – the topic of this class
- …

Visualization, dealing with parameter files, ...

Existing Software

Progress over time:

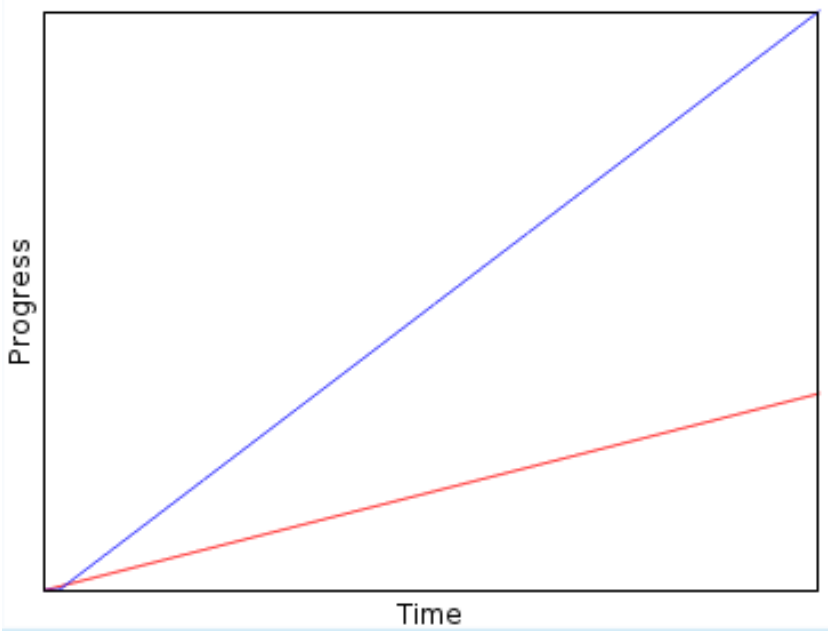


Red: Do it yourself. Blue: Use existing software.

Question: Where is the cross-over point?

Existing Software

Progress over time, the real picture:



Red: Do it yourself. Blue: Use existing software.

Answer: Cross-over is after 2–4 weeks! A PhD takes 3–4 years.

Existing Software

Experience:

It is realistic for a student developing numerical methods to have a code at the end of a PhD time that:

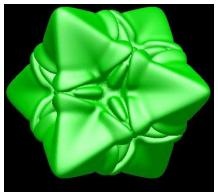
- Works in 2d and 3d
- On complex geometries
- Uses higher order finite element methods
- Uses multigrid solvers or preconditioners
- Solves a nonlinear, time dependent problem

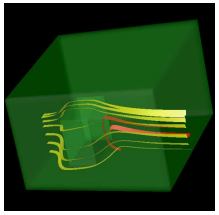
Doing this from scratch would take 10+ years.

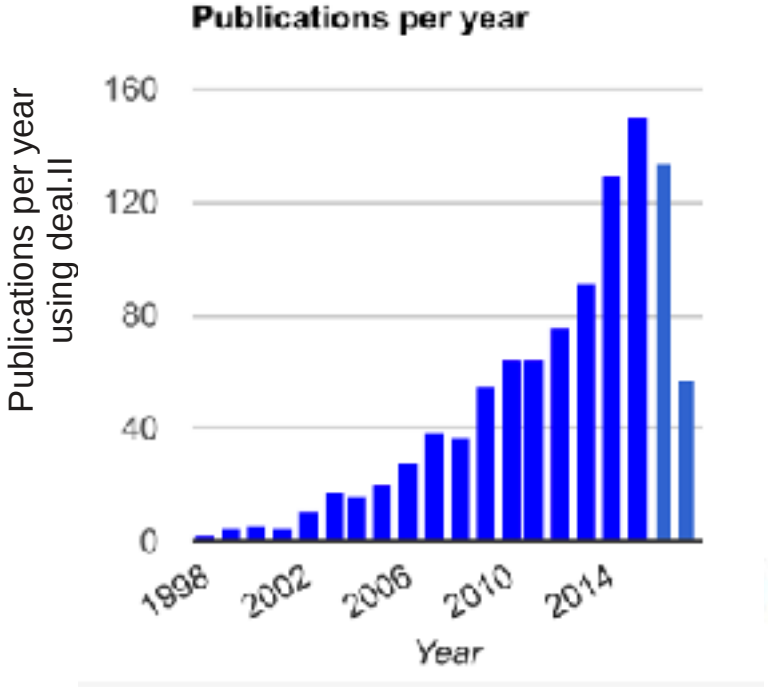
Why Deal.II?

deal.II is probably the largest FEM library:

- Presently ~600,000 lines of C++ code
- 10,000+ pages of documentation
- ~45 tutorial programs
- Fairly widely distributed:
  20,000+ downloads in 2012
- At least 65+ publications in 2012,
  996 - overall, that use it
- Used in teaching at a number
  of universities
- 2007 Wilkinson prize.



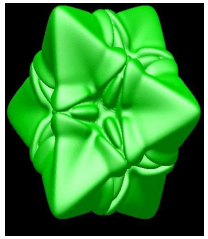




Why Deal.II?

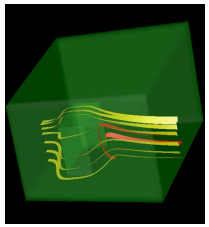
Linear algebra in deal.II:

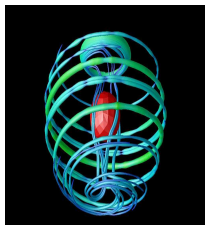
- Has its own sub-library for dense + sparse linear algebra
- Interfaces to PETSC, Trilinos, UMFPACK



Pre- and postprocessing:

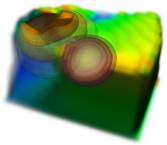
- Can read most mesh formats
- Can write almost any visualization file format





Parallelization:

- Uses threads and tasks on multicore machines
- Uses MPI, up to 10,000s of processors



MHPC
Master in High Performance Computing

Why Deal.II?

Professional-level software management:

- Globally accessible repository
- Mailing lists with significant volume
  – for user questions
  – for developer discussions
- ~2,700 tests run after every change
- Multi-platform build systems
  – Linux/Unix
  – Mac OS X
  – Windows
- Web sites tracking changes, tests, builds, ...

SISSA ICTP

Prerequisites

Operating system:

- Linux or other Unix
- Mac OS X
- Windows

C++ Compiler:

- GNU Compiler Collection (GCC)
- Intel C++
- Clang++

Other software: Perl, CMake, viz software, IDE

Disk space: 1–2GB

Obtaining deal.II

Development sources

Fork our repository on github:

https://github.com/dealii/dealii.git

Building

Detailed instructions:

http://www.dealii.org/developer/readme.html

Basic instructions:

- Create build directory:
  cd deal.II ; mkdir build ; cd build
- Determine properties of the system and compilers:
  cmake -DCMAKE_INSTALL_PREFIX=/a/b/c ..
- Compile the entire library (~1 hour):
  make ; make install
- Verify that everything works:
  cd examples/step-1
  cmake -DDEAL_II_DIR=/a/b/c . ; make run

Building

There are many flags other than

- the installation directory
- the source directory

that can be given to *cmake*. For example:

- Paths to PETSc, Trilinos and other libraries
- Whether to build documentation locally
- Whether to disable or enable multithreading

Detailed instructions:

http://www.dealii.org/developer/readme.html

Basis of FEM!

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

We start with the strong form:

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

MHPC
Master in High Performance Computing

Basis of FEM!

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

We start with the strong form:

$$-\Delta u = f$$

...and transform this into the weak form by multiplying *from the left* with a test function:

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

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

Basis of FEM!

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

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

To determine the *N* coefficients, test with the *N* basis functions:

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

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

Basis of FEM!

Practical question 1: How to define the basis functions?

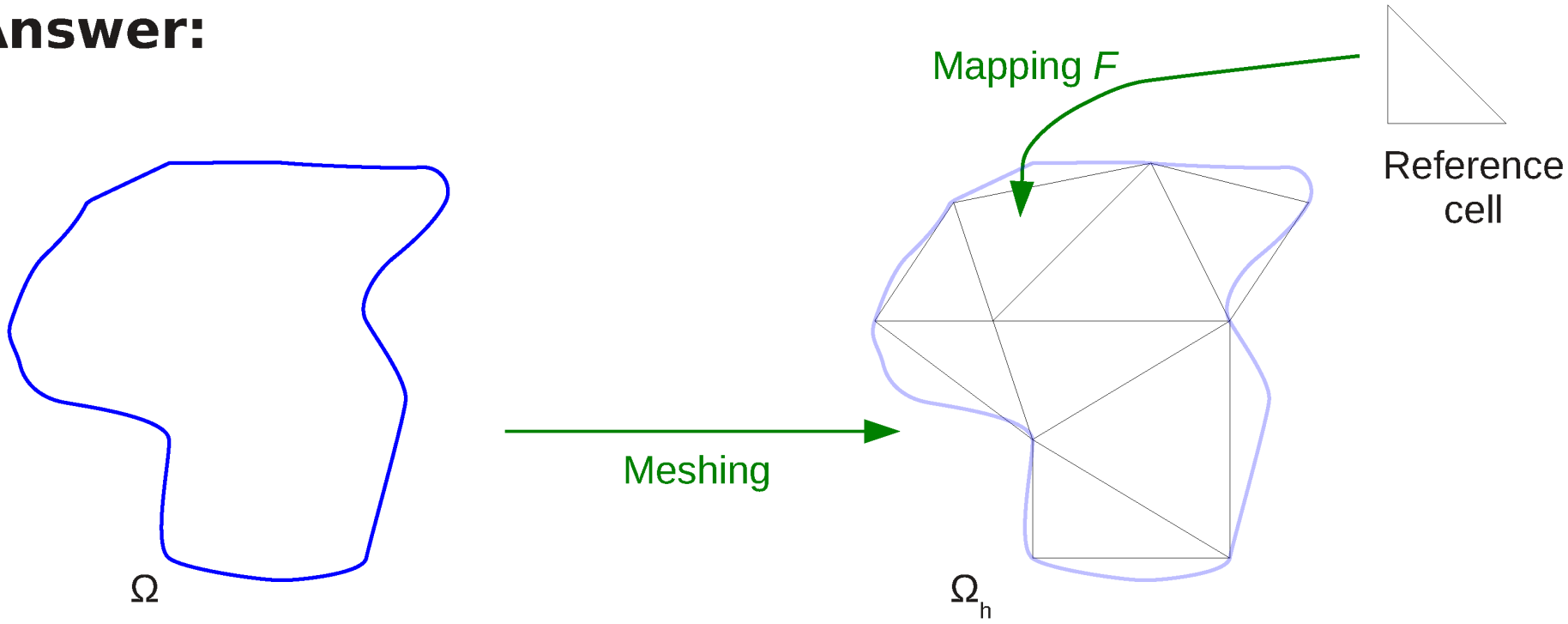
Answer: In the finite element method, this is done using the following concepts:

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

Basis of FEM!

Practical question 1: How to define the basis functions?

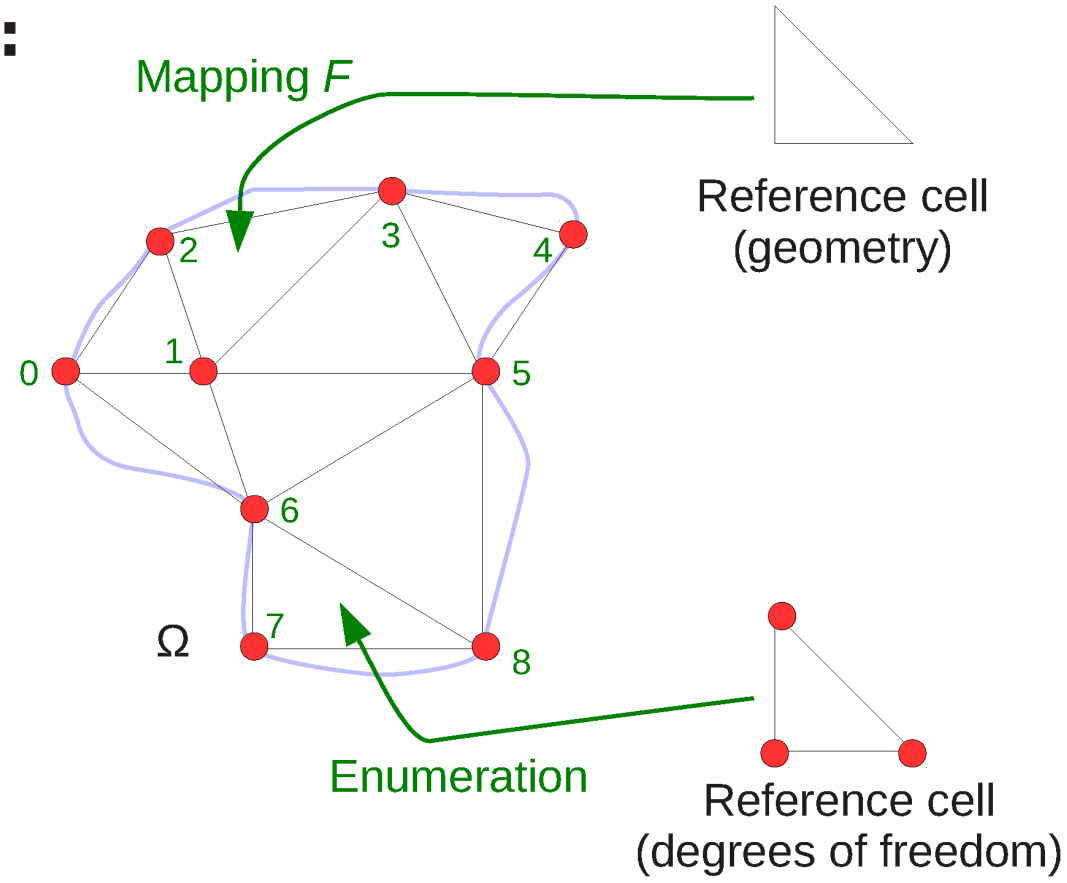
Answer:



Basis of FEM!

Practical question 1: How to define the basis functions?

Answer:



Basis of FEM!

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
- Each cell of the mesh is a mapping of the reference cell
- Definition of basis functions on 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.

Basis of FEM!

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

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

to obtain:

$$\sum_{j=1}^{N} (\nabla \phi_i, \nabla \phi_j) U_j = (\phi_i, f) \quad \forall i = 1 \ldots N$$

This is a linear system

$$A U = F$$

with

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

Basis of FEM!

Practical question 2: How to compute

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

Answer: By mapping back to the reference cell...

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

...and quadrature:

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

Similarly for the right hand side *F*.

Basis of FEM!

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

Basis of FEM!

Practical question 4: How to solve the linear system

$$AU = F$$

Answers: In practical computations, we need a variety of

- Direct solvers
- Iterative solvers
- Parallel solvers

Basis of FEM!

Practical question 5: What to do with the solution of the linear system

$$AU = F$$

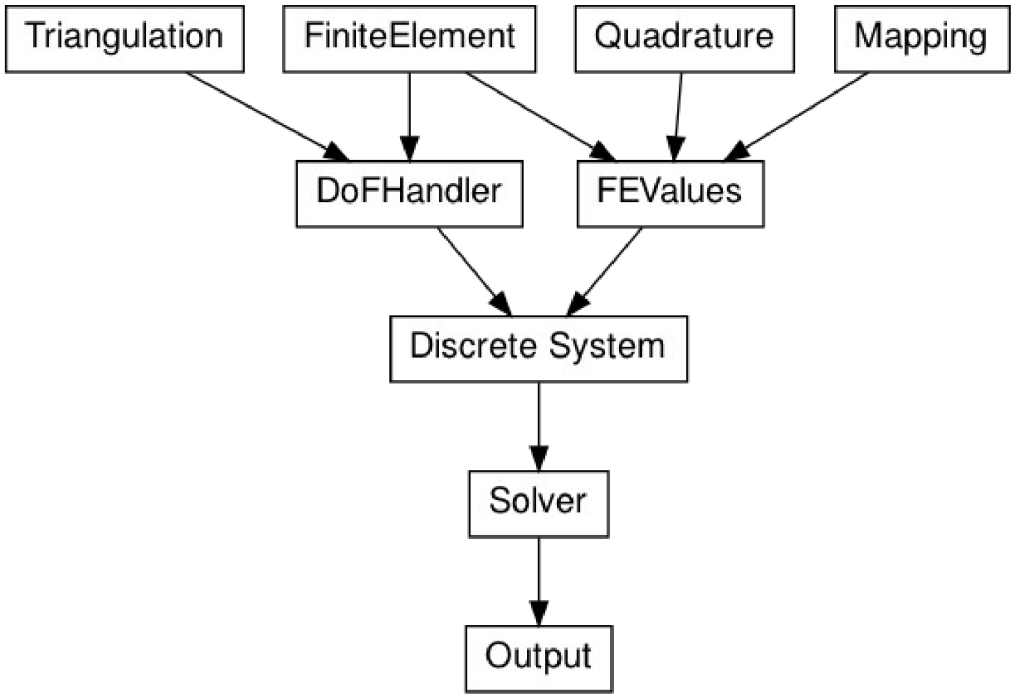
Answers: The goal is not to solve the linear system, but to do something with its solution:

- Visualize
- Evaluate for quantities of interest
- Estimate the error

These steps are often called *postprocessing the solution*.

Basis of FEM!

Together, the concepts we have identified lead to the following components that all appear (explicitly or implicitly) in finite element codes:



Start with Tutorial: step-1

After reading, play with the program:

cd examples/step-1
cmake -DDEAL_II_DIR=/path/to/deal.II .
make run

This will run the program and generate output files:

ls -l
okular grid-2.eps

Next step: Play by following the suggestions in the results section. This is the best way to learn!

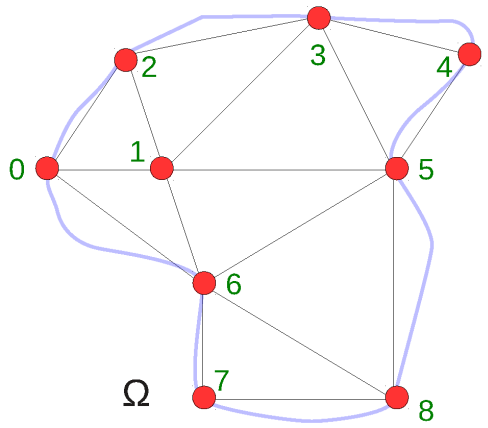
Start with Tutorial: step-2

Step-2 shows:

- How degrees of freedom are defined with finite elements
- The *DoFHandler* class
- How DoFs are connected by bilinear forms
- Sparsity patterns of matrices
- How to visualize a sparsity pattern

Start with Tutorial: step-2

Example: Consider this mesh and bilinear form:



$$A_{ij} = (\nabla \phi_i, \nabla \phi_j) = \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \, \mathrm{d}x$$

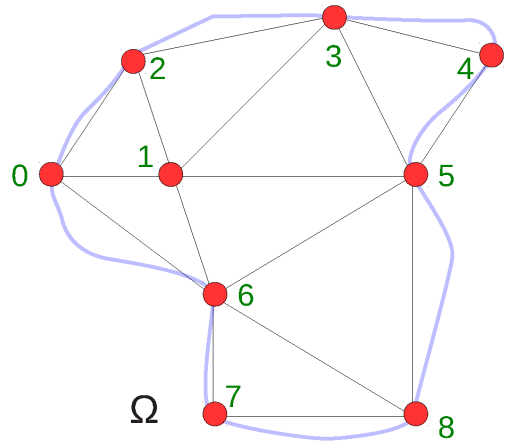
Note: In general we have that

- $A_{00} \neq 0, A_{01} \neq 0, A_{02} \neq 0, A_{06} \neq 0$
- $A_{03} = A_{04} = A_{05} = A_{07} = A_{08} = 0$

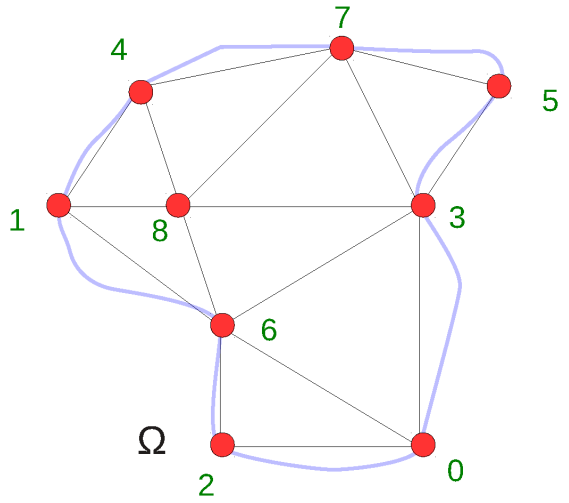
The bigger the mesh, the more zeros there are per row!

Start with Tutorial: step-2

Renumbering: The order of enumerating degrees of freedom is arbitrary



vs.



Notes:

- Resulting matrices are just permutations of each other
- Both sparse, but some algorithms care

Start with Tutorial: step-3

Step-3 shows:

- How to set up a linear system
- How to assemble the linear system from the bilinear form:
  - The loop over all cells
  - The *FEValues* class
- Solving linear systems
- Visualizing the solution

Start with Tutorial: step-3

Recall:

- For the Laplace equation, the bilinear form is written as a sum over all cells:

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

- But on each cell, only few shape functions are nonzero!
- For Q_1, only $16=4^2$ matrix entries are nonzero per cell
- Only compute this (dense) sub-matrix, then “distribute” it to the global A
- Similar for the right hand side vector.

Start with Tutorial: step-3

Recall:

- We use quadrature

$$
\begin{aligned}
A_{ij}^{K} &= \int_{K} \nabla \hat{\phi}_i(x) \cdot \nabla \hat{\phi}_j \, dx \\
&\approx \sum_{q=1}^{Q} J_K^{-1}(\hat{x}_q) \hat{\nabla} \hat{\phi}_i(\hat{x}_q) \cdot J_K^{-1}(\hat{x}_q) \hat{\nabla} \hat{\phi}_j(\hat{x}_q) \underbrace{|\det J(\hat{x}_q)| \; w_q}_{=:\mathrm{JxW}}
\end{aligned}
$$

- We really only have to evaluate shape functions, Jacobians, etc., at quadrature points – not as functions
- All evaluations happen on the reference cells

MHPC
Master in High Performance Computing

Now … Exercise Time!

SISSA
Scuola Internazionale Superiore di Studi Avanzati

CLEMSON UNIVERSITY

CIG COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

ICTP The Abdus Salam International Centre for Theoretical Physics