Finite element methods in scientific computing

Wolfgang Bangerth, Colorado State University

Lecture 1:

Course overview.

Why consider software libraries?

Course overview

The topic of this class:

Learn how to solve partial differential equations on computers! *

* Using the finite element method.

Course overview

The numerical solution of partial differential equations is an immensely practical field!

It requires us to know about:

- Partial differential equations
- Methods for discretizations, solvers, preconditioners
- Programming
- Adequate tools

This course will cover all of this to some degree!

Course overview

Goals of this class:

- Learn about widely used methods in scientific computing
- Learn about deal.II – a software library that supports finite element computations in scientific computing
- Learn to use deal.II in practice
- Learn about modern software development tools
- Learn about software development practice

Course overview

Not goals of this class:

- Learn about the finite element method [1]

- Learn how to program in C++ [2]

[1] S. Brenner & R. Scott: The mathematical theory of finite element methods.
D. Braess: Finite elements

[2] Stroustrup: The C++ programming language

Partial differential equations

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, ...

On why to use existing software

There are times to write software ourselves:

- When developing new computational methods
- When solving non-standard problems

In such cases, we could:

1) Start from scratch, write everything ourselves
2) Build something from existing components
3) Adapt existing codes written for similar applications

But: Option 1) is difficult/time consuming/expensive.
Let us explore the other options!

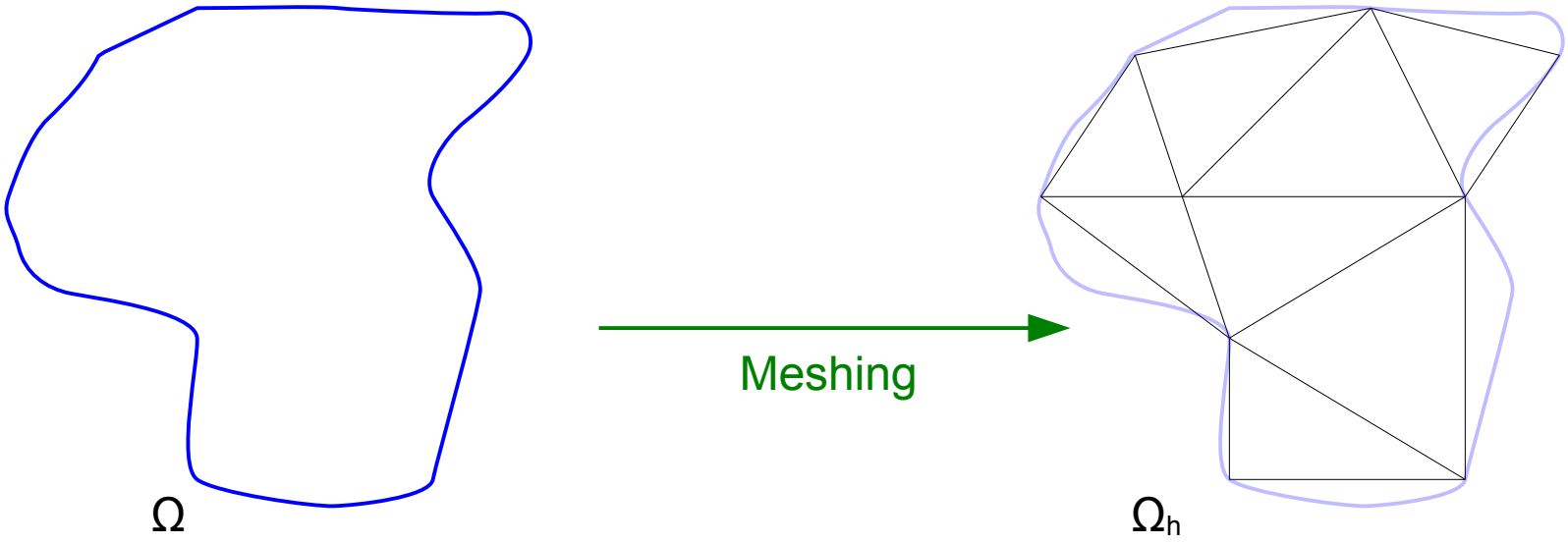
Numerics for PDEs

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)



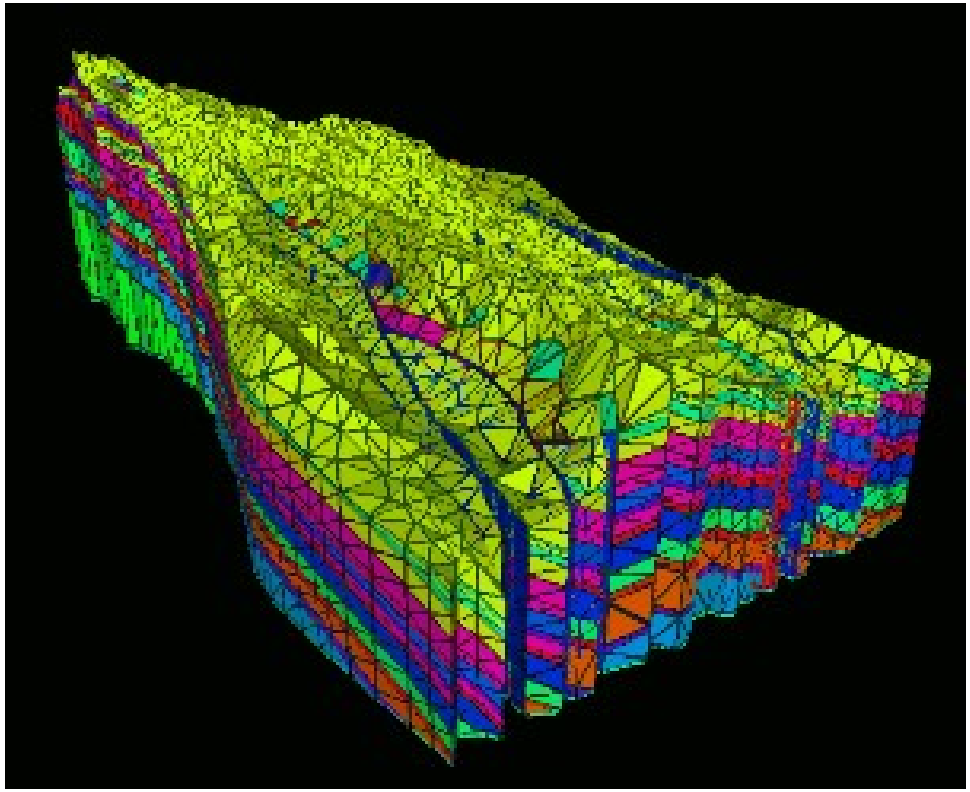
Numerics for PDEs

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)



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

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

Numerics for PDEs

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

In addition:

- We don't just want a simple algorithm
- We want state-of-the-art methods for everything!

Numerics for PDEs

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?

How we develop software

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 on existing libraries?

How we develop software

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 on existing libraries?

There has been a major shift on how we approach this question in scientific computing over the past 20 years!

How we develop software

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

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

Arguments against using other people's packages:

I would need to learn a new piece of software, how it works, its conventions. I would have to find my way around its documentation. Etc.

I think I'll be faster writing the code I want myself!

Existing software

Arguments against using other people's packages:

> I would need to learn a new piece of software, how it works, its conventions. I would have to find my way around its documentation. Etc.
>
> I think I'll be faster writing the code I want myself!

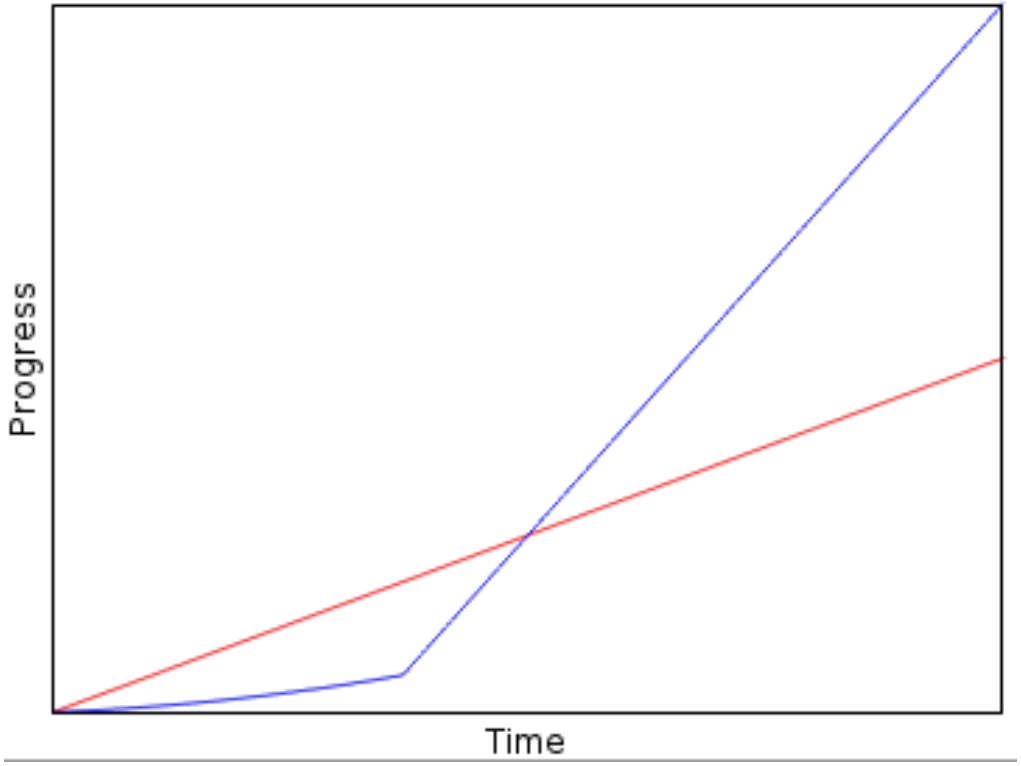
Answers:

- The first part is true.
- The second is not!

- You get to use a lot of functionality you could never in a lifetime implement yourself.
- Think of how we use Matlab today!

Existing software

Progress over time:

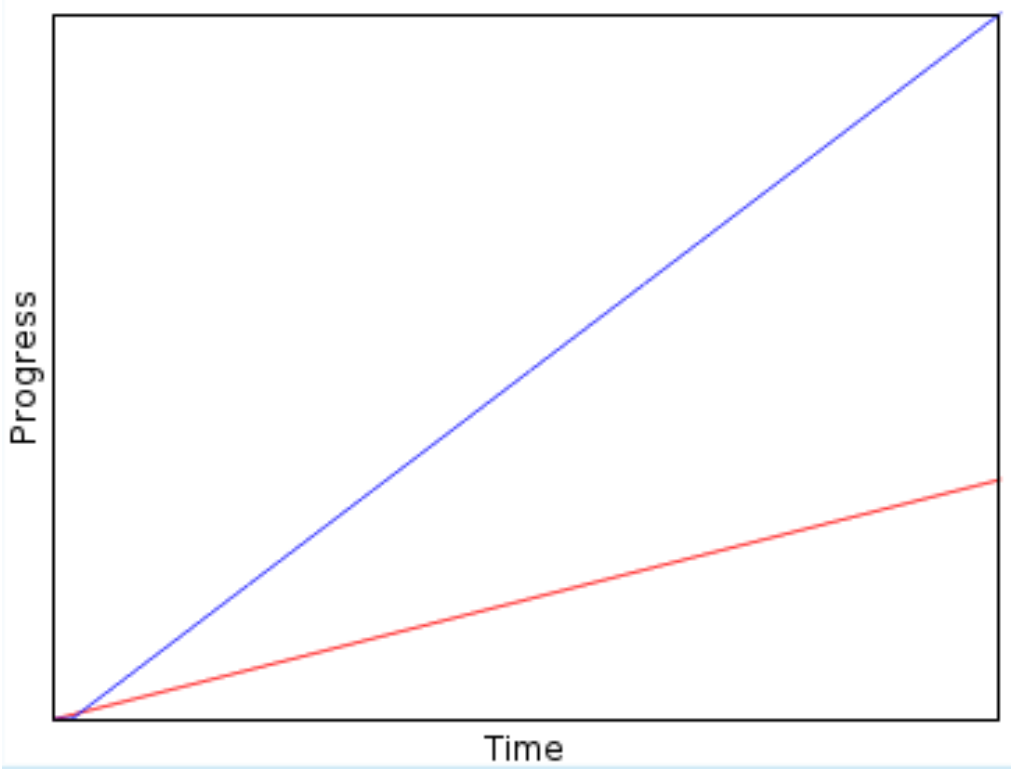


Red: Do it yourself. Blue: Use existing software.

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.

Existing software

Arguments against using other people's packages:

I want my students to actually understand the methods they are using.
So I let them code things from scratch!

Existing software

Arguments against using other people's packages:

> I want my students to actually understand the methods they are using.
> So I let them code things from scratch!

Answers:

- Yes, there is value to that.
- But: if you know quadrature in 2d, why implement it again in 3d?

- So let them write a toy code and throw it away after 3 months and do it right based on existing software.

Existing software

Arguments against using other people's packages:

How do I know that that software I'm supposed to use doesn't have bugs? How can I *trust* other people's software?

With my own software, at least I know that I don't have bugs!

Existing software

Arguments against using other people's packages:

> How do I know that that software I'm supposed to use doesn't have bugs? How can I *trust* other people's software?
>
> With my own software, at least I know that I don't have bugs!

Answer 1:

- You can't be serious to think that your own software has no bugs!

Existing software

Arguments against using other people's packages:

> How do I know that that software I'm supposed to use doesn't have bugs? How can I *trust* other people's software?
>
> With my own software, at least I know that I don't have bugs!

Answer 2:

- The packages I will talk about are developed by professionals with decades of experience
- They have extensive test suites
- For example, deal.II runs 13,000+ tests after every single change

Conclusions

- When having to implement software, *re-use what others have done already!*
- There are many high-quality, open source software libraries for every purpose in scientific computing
- Use them:
  – You will be far more productive
  – You will be able to use state-of-the-art methods
  – You will have far fewer bugs in your code
- **If you are a graduate student:** Use them because you will be able to impress your adviser with quick results!

Finite element methods in scientific computing

Wolfgang Bangerth, Colorado State University