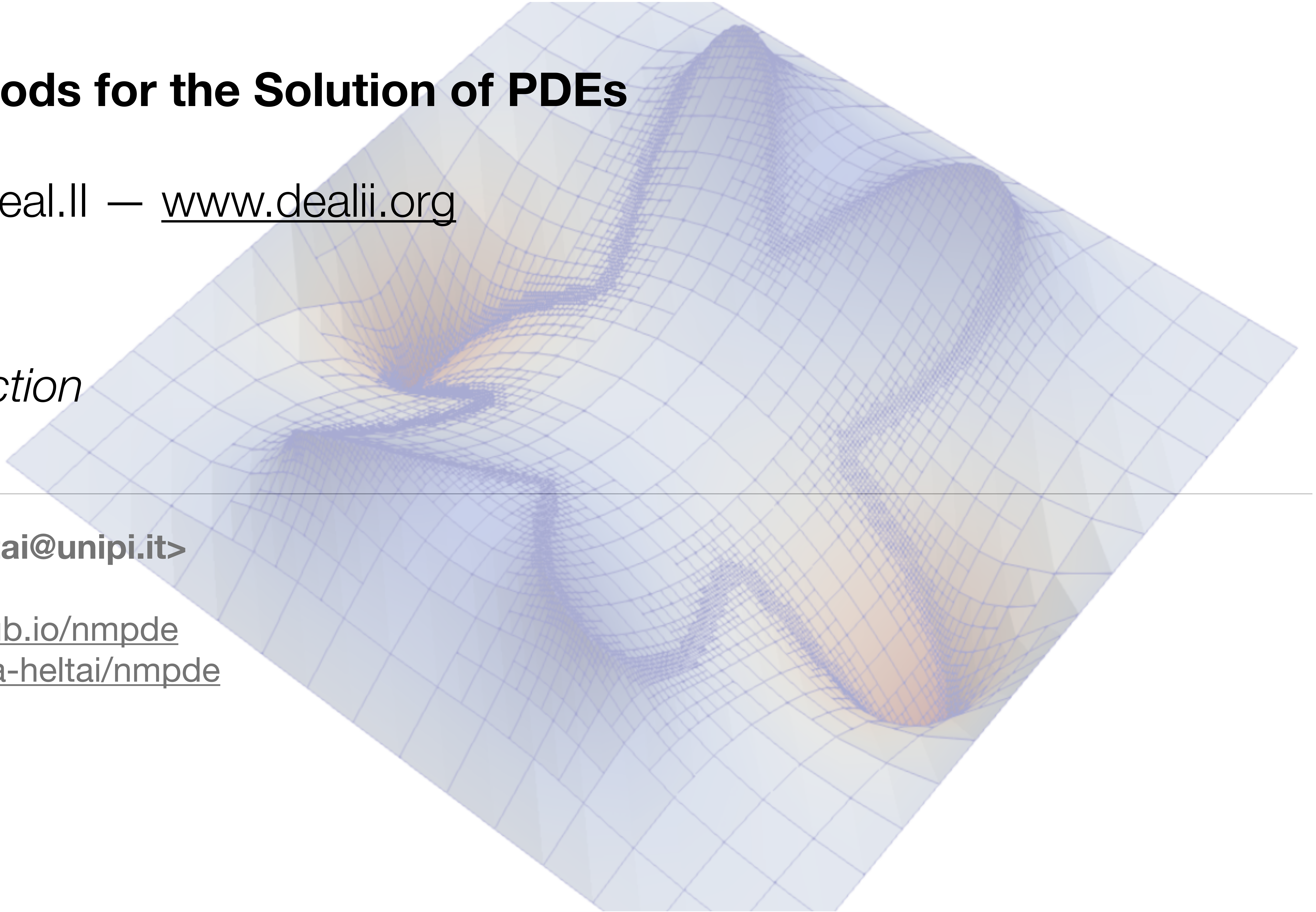
Numerical Methods for the Solution of PDEs

Laboratory with deal.II — www.dealii.org

LAB 1 — Introduction



Luca Heltai <luca.heltai@unipi.it>

https://luca-heltai.github.io/nmpde
https://github.com/luca-heltai/nmpde

Main arguments

- Serial scalar poisson solver, in various flavours
- Convergence tests
- Local adaptivity
- Vector and Mixed problems
- block preconditioning

Tools, Techniques, Best Practices

- What you will learn:
  - Advanced Finite Element theory
  - How to use a modern C++ IDE, to build and debug your codes
  - How to use a large FEM library to solve complex PDE problems
  - A touch of Best Practices:
    - code documenting (Doxygen)
    - version control (Git)
    - code testing (google tests, docker images, GithHub actions)

Outcome of the course

- You will produce your own FEM application based on deal.II which:
  - Solves a PDE of interest to you, on adaptively refined grids, arbitrary order
  - Uses modern version control tools (on GitHub)
  - *Bonus*: it is tested automatically (through GitHub actions) every time you push a commit, or open a pull request
  - *Bonus*: it is documented using Doxygen, and its web page is updated and deployed automatically every time you merge to master a new branch

Prerequisites

- Theory:
  - Some knowledge of Sobolev Spaces
    - Linear operators, Banach and Hilbert spaces, duality, etc.
  - One elementary course on Numerical Analysis
    - Quadrature, interpolation, Taylor expansions, etc.
- Practice (for the first few lectures):
  - a machine with Visual Studio Code installed
    (c++11 is required, c++17 desired)
  - A GitHub account

More Info

- Course pages:
  - Course main page, with schedule and up to date information

    https://luca-heltai.github.io/nmpde/intro.html
  - Course slides, notes, materials, and codes:

    https://github.com/luca-heltai/nmpde
  - Email:
    Luca Heltai <luca.heltai@unipi.it>

Why deal.II (or any other Finite Element library)

- The numerical solution of partial differential equations is an immensely vast 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!**

Numerics of 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 of 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

In addition:

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

Numerics of 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?

The hard reality

- Most research software today:
  - **Written by graduate students**
    - without a good overview of existing software
    - with little software experience
    - with little incentive to write high quality code
  - **Maintained by postdocs**
    - with little time
    - who need to consider the software primarily as a tool to publish papers
  - **Advised by faculty**
    - with no time
    - oftentimes also with little software experience

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 our software 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 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!

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, ...

Our experience

It is realistic for a student developing numerical methods using external libraries 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.

Common arguments…

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!

Common arguments…

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:

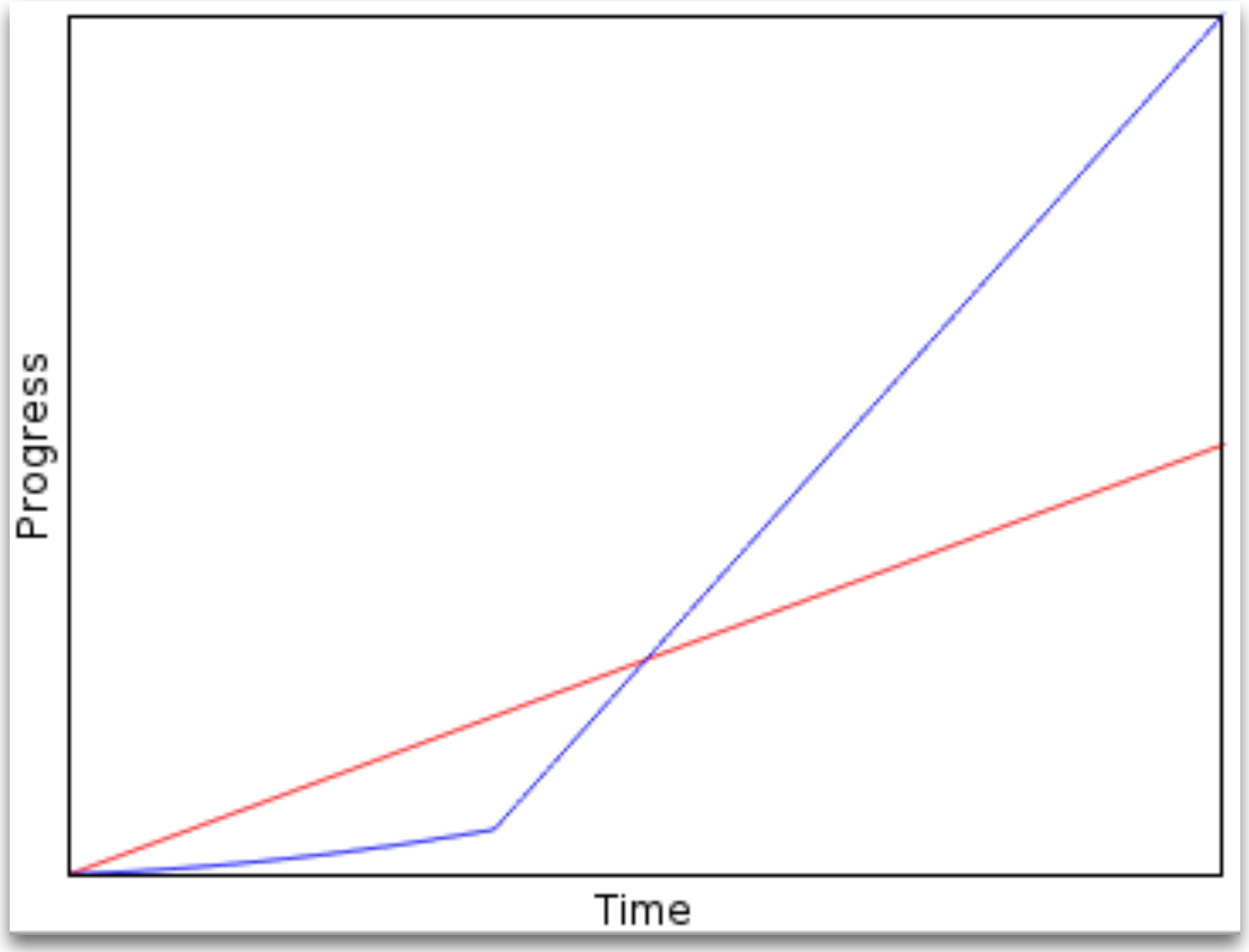
The first part is true.

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!

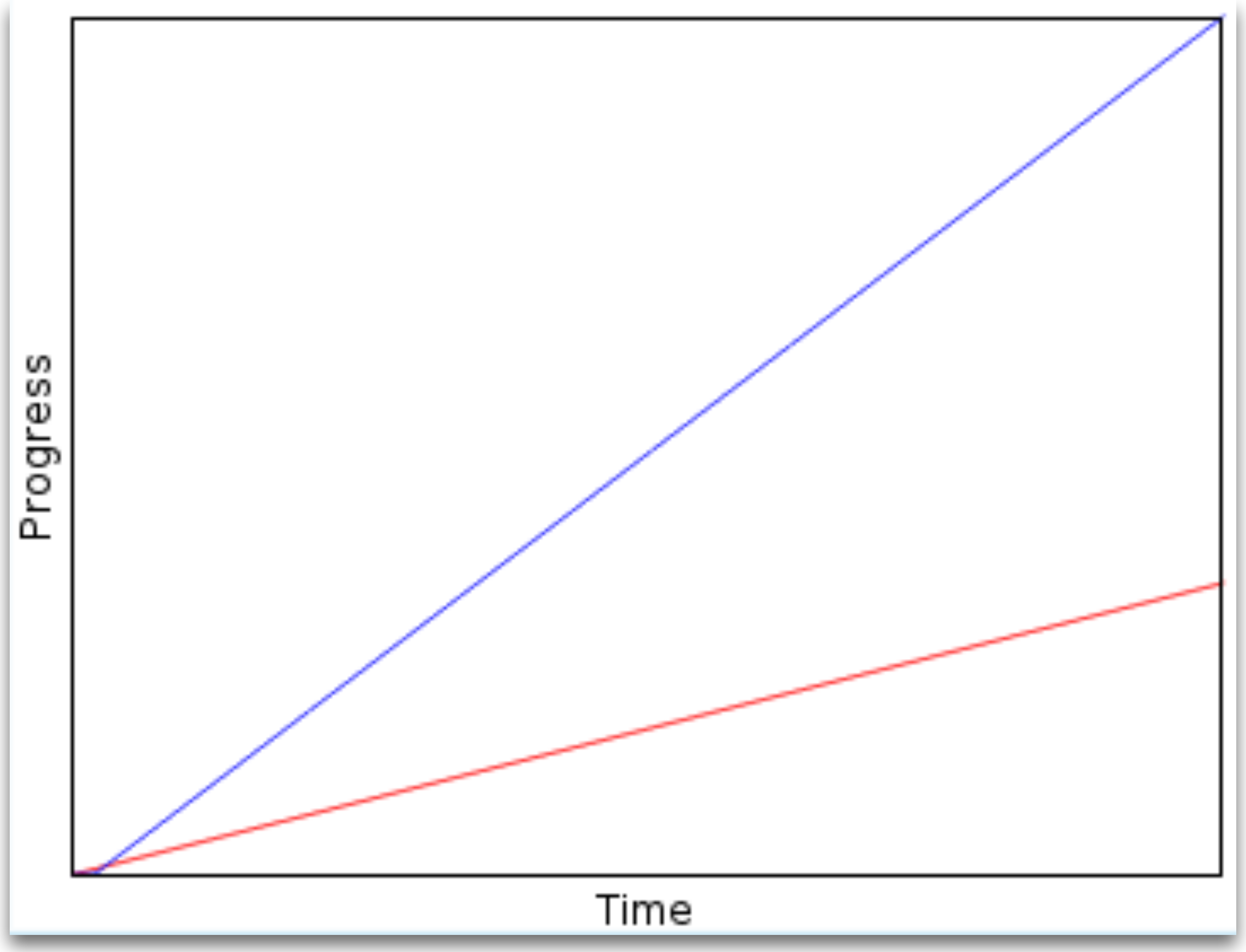
I'm faster!



Blue: use external libraries

Red: do it yourself

The real picture…



Blue: use external libraries

Red: do it yourself

Common Arguments…

Arguments against using other people's packages:

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

Common Arguments…

Arguments against using other people's packages:

I want my students to actually understand the methods they are doing. 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.

Common Arguments…

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!

Common Arguments…

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 a lot of experience

They have extensive testsuites

For example, deal.II runs 5,000+ tests **after every single change**

Bottomline:

When having to implement software for a particular problem, 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 advisor with quick results!

Roadmap for next lectures:

- Modern IDE (VSCode)
- Version control system (git)
- Cross platform build systems (cmake)
- Automatic formatting (clang-format)
- Test driven development (google test, deal.II testing framework)
- Inline documentation (doxygen)
- External visualisation tools (paraview)