Master in High Performance Computing

Numerical Solution of PDEs using the Finite Element Method

A posteriori error estimates and adaptive meshes

Luca Heltai <luca.heltai@sissa.it>

International School for Advanced Studies (www.sissa.it)
Mathematical Analysis, Modeling, and Applications (math.sissa.it)
Master in High Performance Computing (www.mhpc.it)
SISSA mathLab (mathlab.sissa.it)

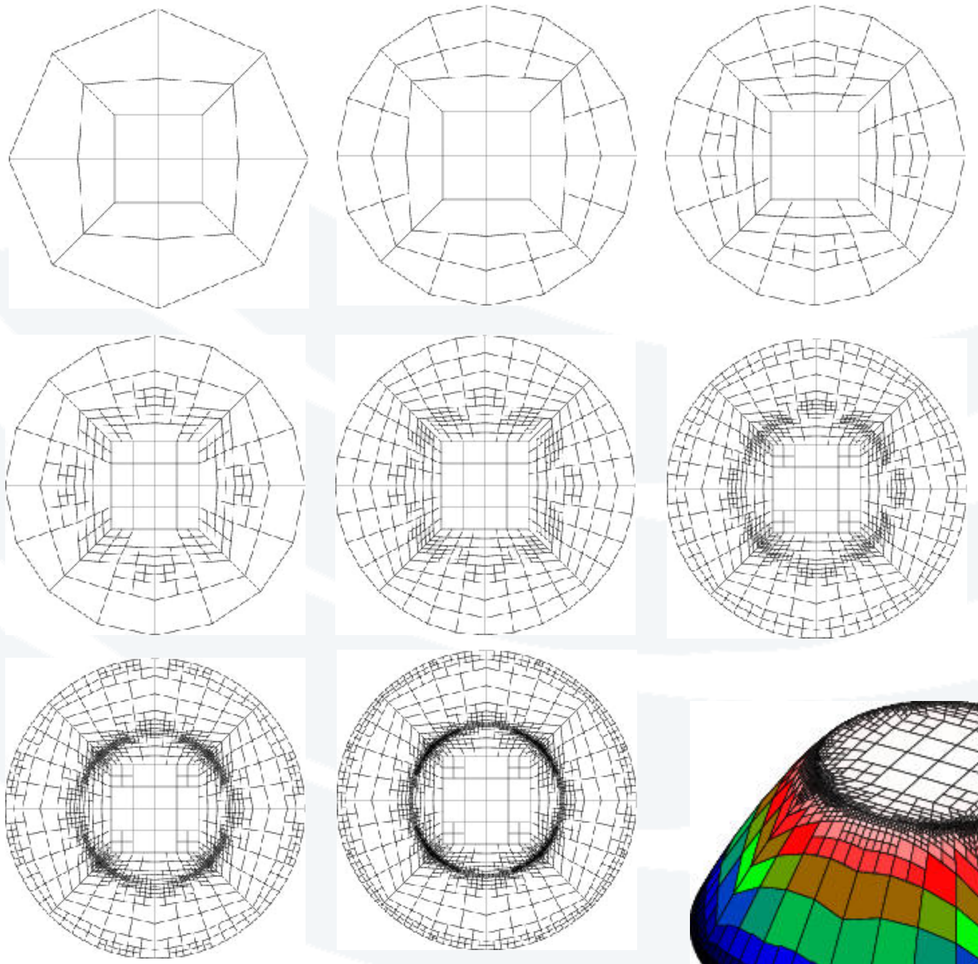
sissa mathLab innovating with mathematics

The Abdus Salam International Centre for Theoretical Physics

SISSA

Adaptive mesh refinement

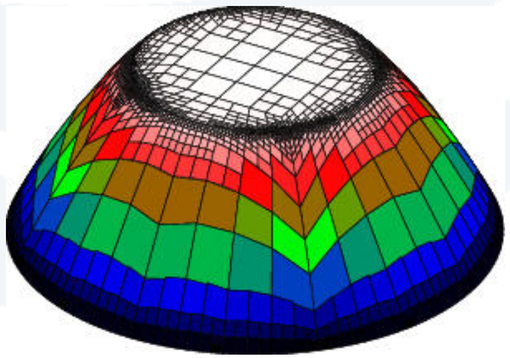
SOLVE — ESTIMATE — MARK — REFINE



$$\nabla \cdot a(\boldsymbol{x}) \nabla u(\boldsymbol{x}) = 1 \quad \text{in } \Omega$$
$$u = 0 \quad \text{on } \partial\Omega$$

$$a(\boldsymbol{x}) = \begin{cases} 20 & \text{if } |\boldsymbol{x}| < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

Need an error indicator η_K on each cell without knowing the exact solution.



Adaptive mesh refinement

$$\|u\|^2_{H^2(K)} := \int_K u^2 + |\nabla u|^2 + |\nabla^2 u|^2$$

$$|u|^2_{H^2(K)} := \int_K |\nabla^2 u|^2$$

Error estimate for Q1/P1 elements applied to Laplace problem:

$$\|u - u^h\|_{H^1} \equiv \|e\|_{H^1} \leq C\, h_{max}\, \|u\|_{H^2}$$

this error depends on the largest element size and the global norm of the solution.
To reduce error (increase accuracy) one can refine the mesh size.

more precisely...

$$\|e\|^2_{H^1} \leq C^2 \sum_K h_K^2\, |u|^2_{H^2(K)}$$

Thus one needs to make mesh finer where the local H^2 semi-norm is large.

But apart from some special cases we don't know the exact solution u !

Thus we need to create meshes iteratively (adaptively).

Optimal strategy is to equilibrate the error $e_K := C\, h_K\, |u|_{H^2(K)}$

That is, we want to choose $h_K \sim \dfrac{1}{|u|_{H^2(K)}}$

a-posteriori error estimation

$$||e||^2_{H^1(\Omega)} \leq C \sum_K e_K^2$$

$$e_K = h_K ||\nabla^2 u||_K$$

cell-wise error indicators

(wrong) idea:

$$e_K \approx h_K ||\nabla^2 u^h||_K$$

will not work as linear elements have zero second derivates within the element and first derivatives have jumps on the interfaces

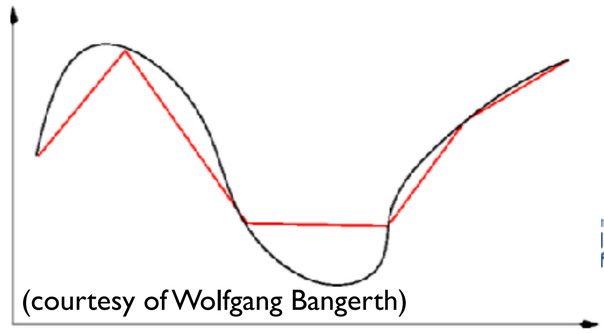
a better idea (in 1D) to approximate second derivatives at interface i:

$$\nabla^2 u \approx \frac{\nabla u^h(x^+) - \nabla u^h(x^-)}{h} =: \frac{[\![\nabla u^h]\!]_i}{h}$$

use jump in gradient as an indicator of the second derivative at vertices

can generalize to:

$$||\nabla^2 u||^2_K \approx \sum_{i \in \partial K} \frac{[\![\nabla u^h]\!]_i^2}{h}$$



(courtesy of Wolfgang Bangerth)

a-posteriori error estimation

As a result, the simplest and most widely used Kelly error **indicator** in 2D/3D follows:

$$e_K^2 = h_K^2 ||\nabla^2 u||_K^2 \approx h_K \int_{\partial K} |[\![\boldsymbol{\nabla} u \cdot \boldsymbol{n}]\!]|^2 \mathrm{ds} =: \eta_K^2$$

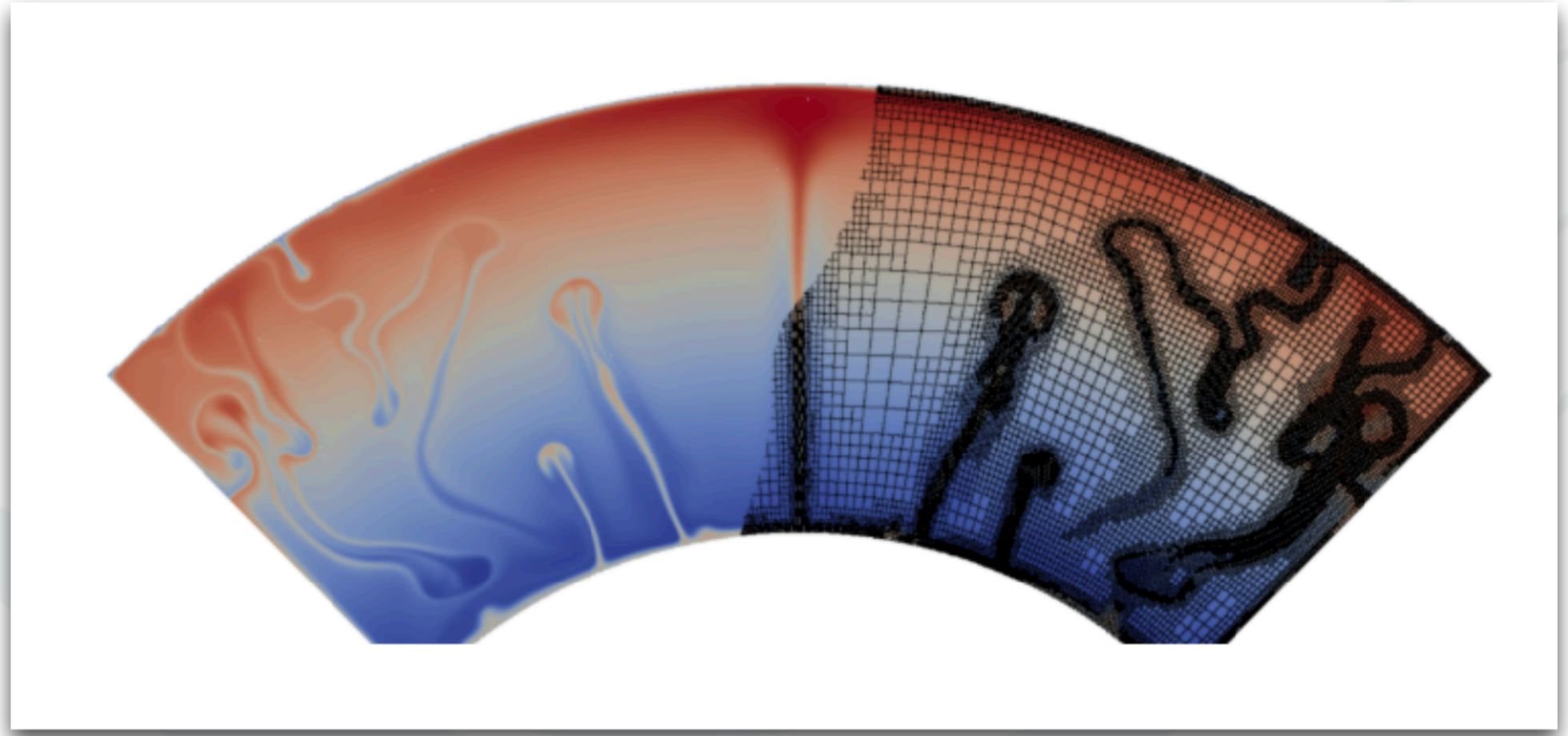
For the Laplace equation, **Kelly, de Gago, Zienkiewicz, Babushka (1983)** proved that

$$||\nabla \left[u - u^h\right]||^2 \leq C \sum_K \eta_K^2$$ a-posteriori error estimator (*)

Note 1:

"**estimator**" is always a proven upper bound of error (*), whereas "**indicator**" is our best guess of error per cell which may not be an upper bound in the sense (*), but may still work well for considered equations and/or FE space.

Adaptive mesh refinement



Refine where things are happening!

Basic AFEM algorithm

- SOLVE-ESTIMATE-MARK-REFINE
  - On the current mesh, solve the problem
  - Estimate the error per cell (Exact, Kelly, Residual, etc.)
  - Mark cells according to given criterion (estimator is greater than a tolerance, or fraction of cells with largest error, or …)
  - Refine the marked cells
- Repeat until tolerance met, or max number of cycles

deal.II classes

- Error estimate is problem dependent:
  - Approximate gradient jumps: `KellyErrorEstimator` class
  - Approximate local norm of gradient: `DerivativeApproximation` class
  - … or something else
- Cell marking strategy:
  - `GridRefinement::refine_and_coarsen_fixed_number(...)`
  - `GridRefinement::refine_and_coarsen_fixed_fraction(...)`
  - `GridRefinement::refine_and_coarsen_optimize(...)`
- Refine/coarsen grid: `triangulation.execute_coarsening_and_refinement ()`
- Transferring the solution: `SolutionTransfer` class (discussed later)