Local adaptive refinement

Jean-Paul Pelteret (jean-paul.pelteret@fau.de)
Luca Heltai (luca.heltai@sissa.it)

20 March 2018

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Aims for this module

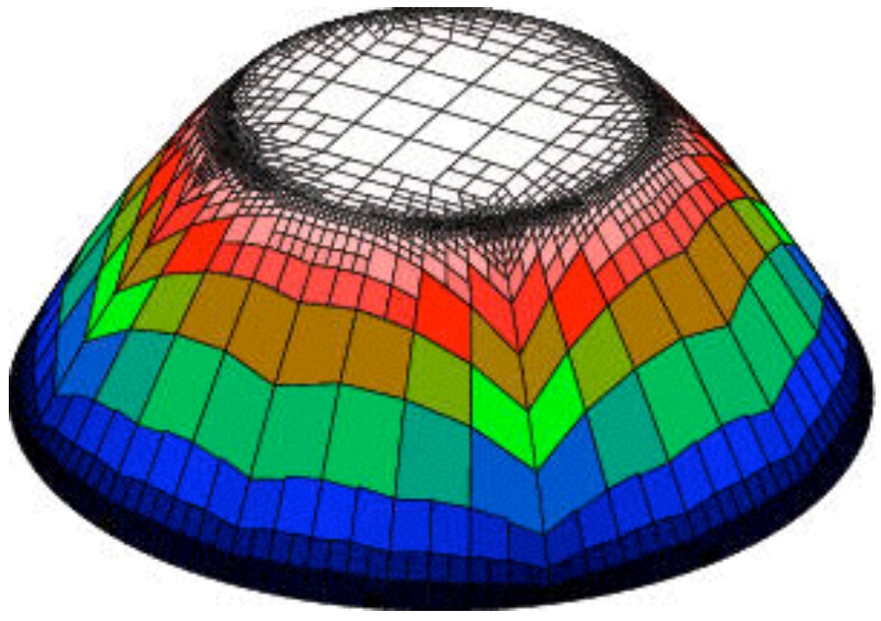
- Implement adaptive mesh refinement
  - Hanging nodes
  - Error-based refinement marking
- Learn about the ConstraintMatrix

Reference material

- Tutorials
  - https://dealii.org/8.5.1/doxygen/deal.II/step_6.html
  - http://www.math.colostate.edu/~bangerth/videos.676.15.html
  - http://www.math.colostate.edu/~bangerth/videos.676.16.html
  - http://www.math.colostate.edu/~bangerth/videos.676.17.html
  - http://www.math.colostate.edu/~bangerth/videos.676.17.25.html
  - http://www.math.colostate.edu/~bangerth/videos.676.17.5.html
  - http://www.math.colostate.edu/~bangerth/videos.676.17.75.html
- Documentation
  - https://dealii.org/8.5.1/doxygen/deal.II/group__constraints.html
  - https://dealii.org/8.5.1/doxygen/deal.II/group__grid.html
  - https://dealii.org/8.5.1/doxygen/deal.II/namespaceGridRefinement.html
  - https://dealii.org/8.5.1/doxygen/deal.II/namespaceDerivativeApproximation.html

Adaptive mesh refinement

- Typical steps to perform adaptivity
  - Solve (non-)linear system
  - Estimate error
  - Mark cells
  - Refine/coarsen
  - Interpolate original solution to new mesh

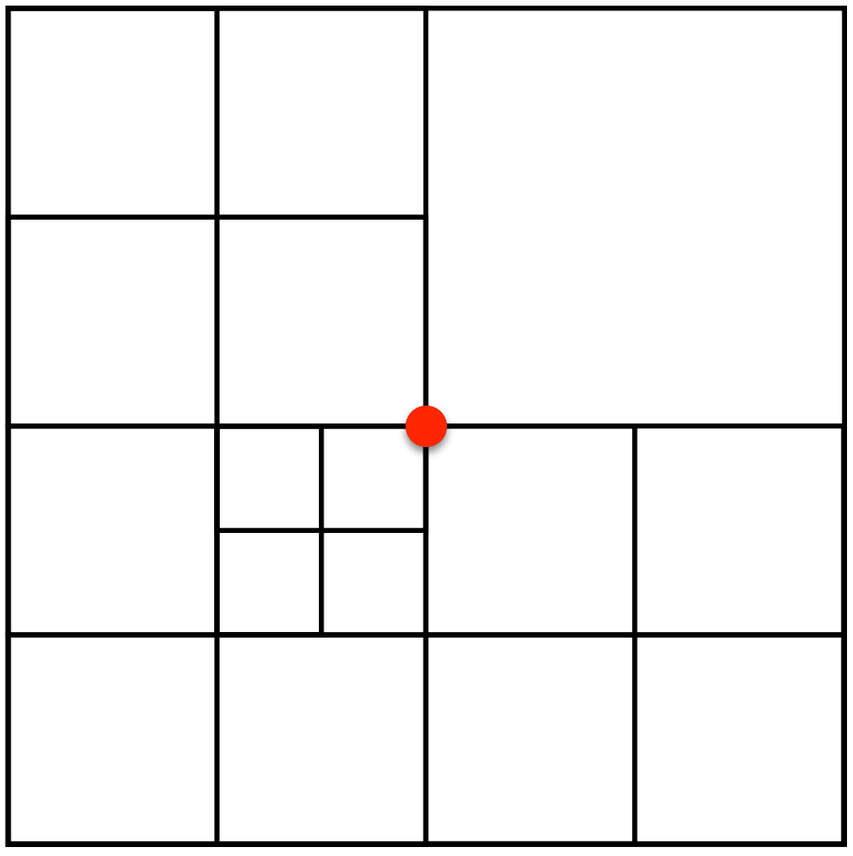


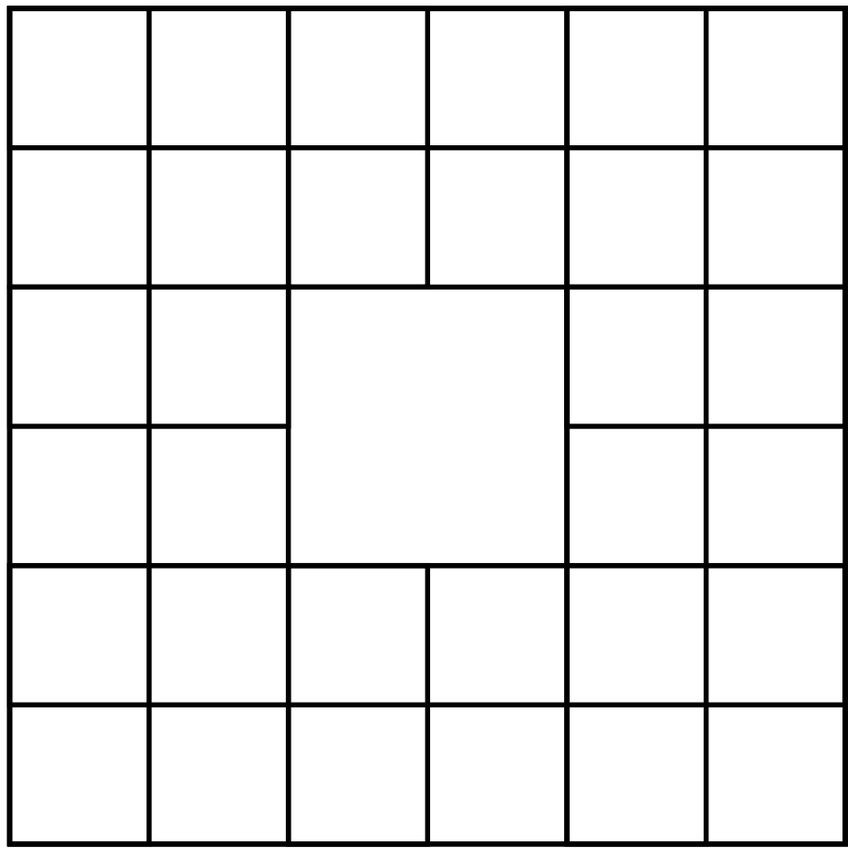
Adaptive mesh refinement

- 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)

Adaptive mesh refinement

- Note: `Triangulation::MeshSmoothing`

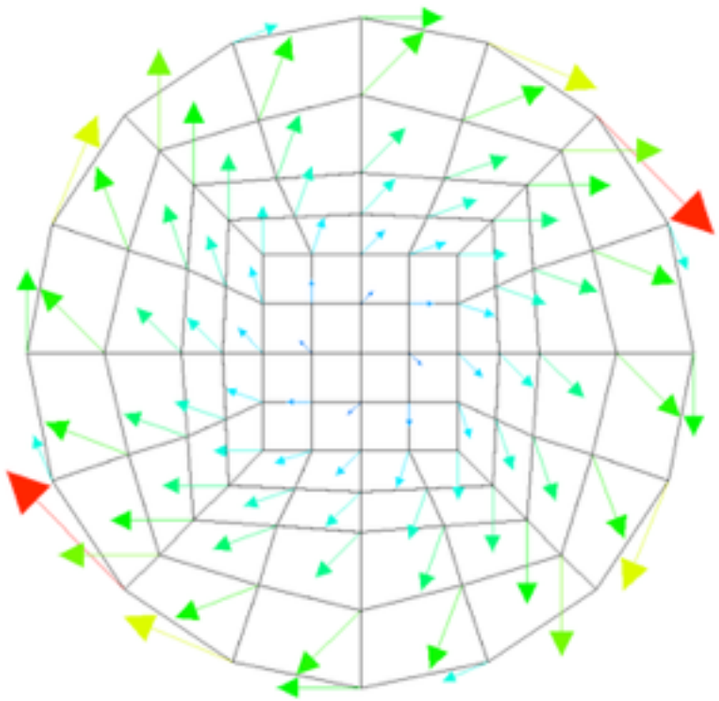




Applying constraints: the ConstraintMatrix class

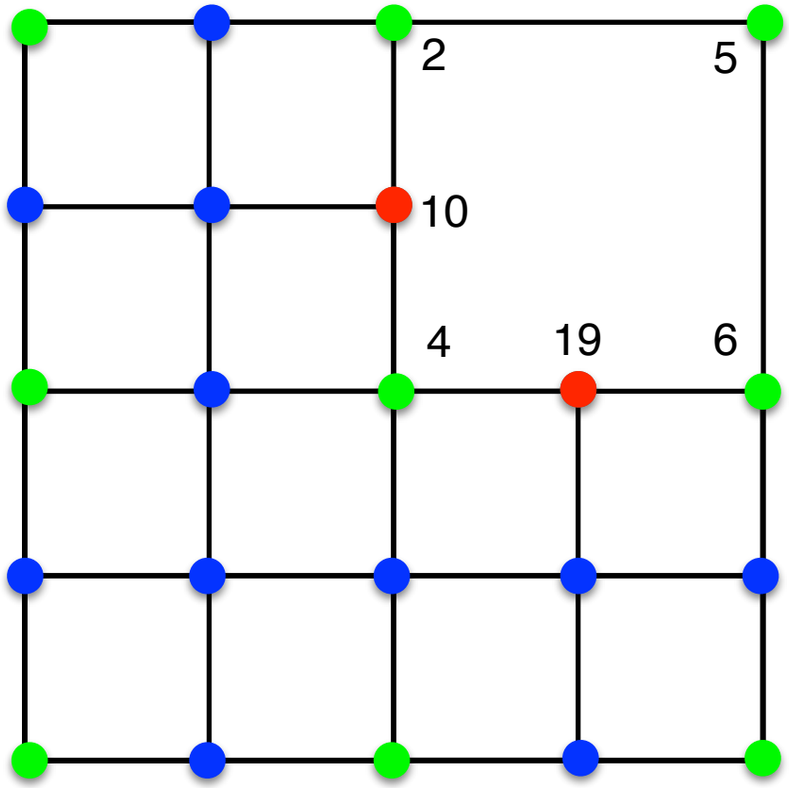
- This class is used for
  - Hanging nodes
  - Dirichlet and periodic constraints
  - Other constraints
- Linear constraints of the the form

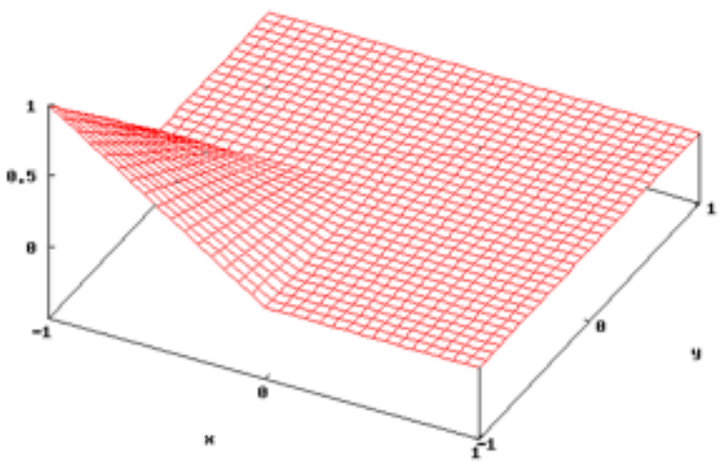
$$x_i = \sum_j \alpha_{ij} x_x + c_j$$

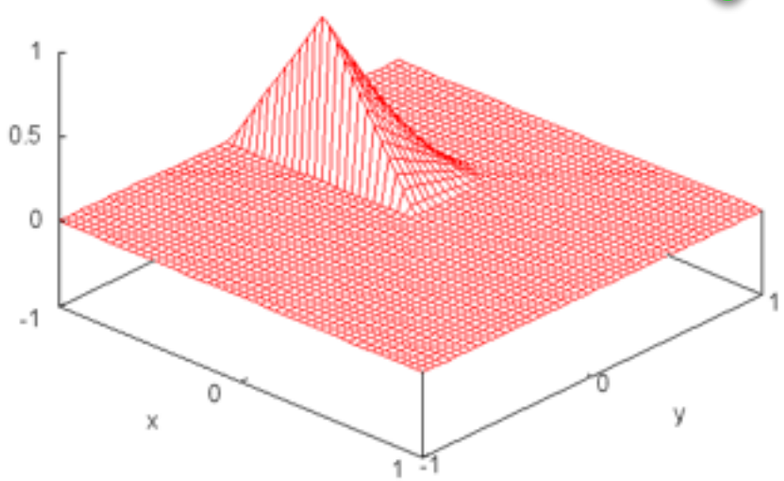


Applying constraints: Hanging node constraints

- Finite element function must be globally continuous
  - Red nodes must have values that are compatible with the adjacent green nodes
  - Function has no jump when traversing from the small cells to the large one at the top right







$$u_{10} = \frac{1}{2}\left[u_2 + u_4\right]$$

$$u_{19} = \frac{1}{2}\left[u_4 + u_6\right]$$

Applying constraints: the ConstraintMatrix class

- System setup
  - Hanging node constraints created using
    `DoFTools::make_hanging_node_constraints()`
  - Will also use for boundary values from now on:
    `VectorTools::interpolate_boundary_values(..., constraints);`
  - Need different SparsityPattern creator
    `DoFTools::make_sparsity_pattern (..., constraints, …)`
    - Can remove constraints from linear system
- Assembly
  - Assemble local matrix and vector as normal
  - Eliminate while transferring to global matrix:
    `constraints.distribute_local_to_global (cell_matrix, cell_rhs, local_dof_indices, system_matrix, system_rhs);`
  - Solve and then set all constraint values correctly:
    `ConstraintMatrix::distribute(...)`

Applying constraints: Conflicts

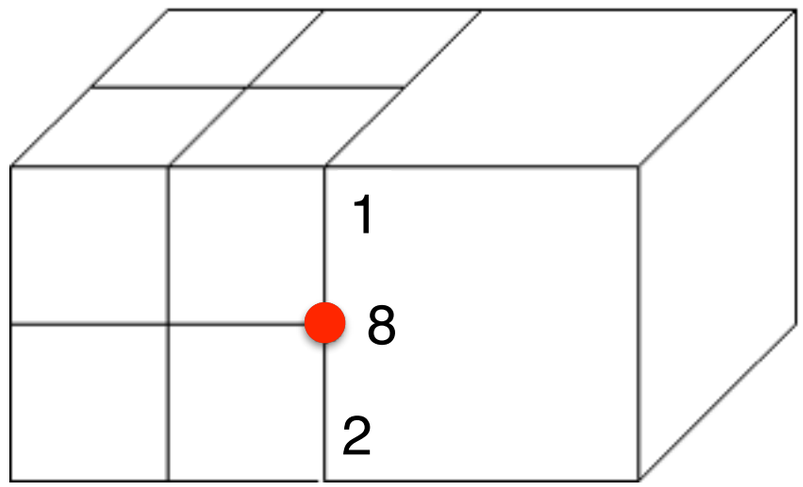
- When writing into a ConstraintMatrix, existing constraints are not overwritten.

- Can merge constraints together:
  `constraints.merge (other_constraints, MergeConflictBehavior::left_object_wins);`

- Which is right?

$$u_8 = \bar{u} \quad \text{or}$$

$$u_8 = \frac{1}{2}\left[u_1 + u_2\right]$$



- Beware of introducing constraint cycles

$$u_1 = u_2 \; ; \; u_2 = u_3 \; ; \; u_3 = u_1$$