Local error estimation and adaptive refinement

Denis Davydov (denis.davydov@fau.de)
Luca Heltai (luca.heltai@sissa.it)

25 February - 1 March 2019

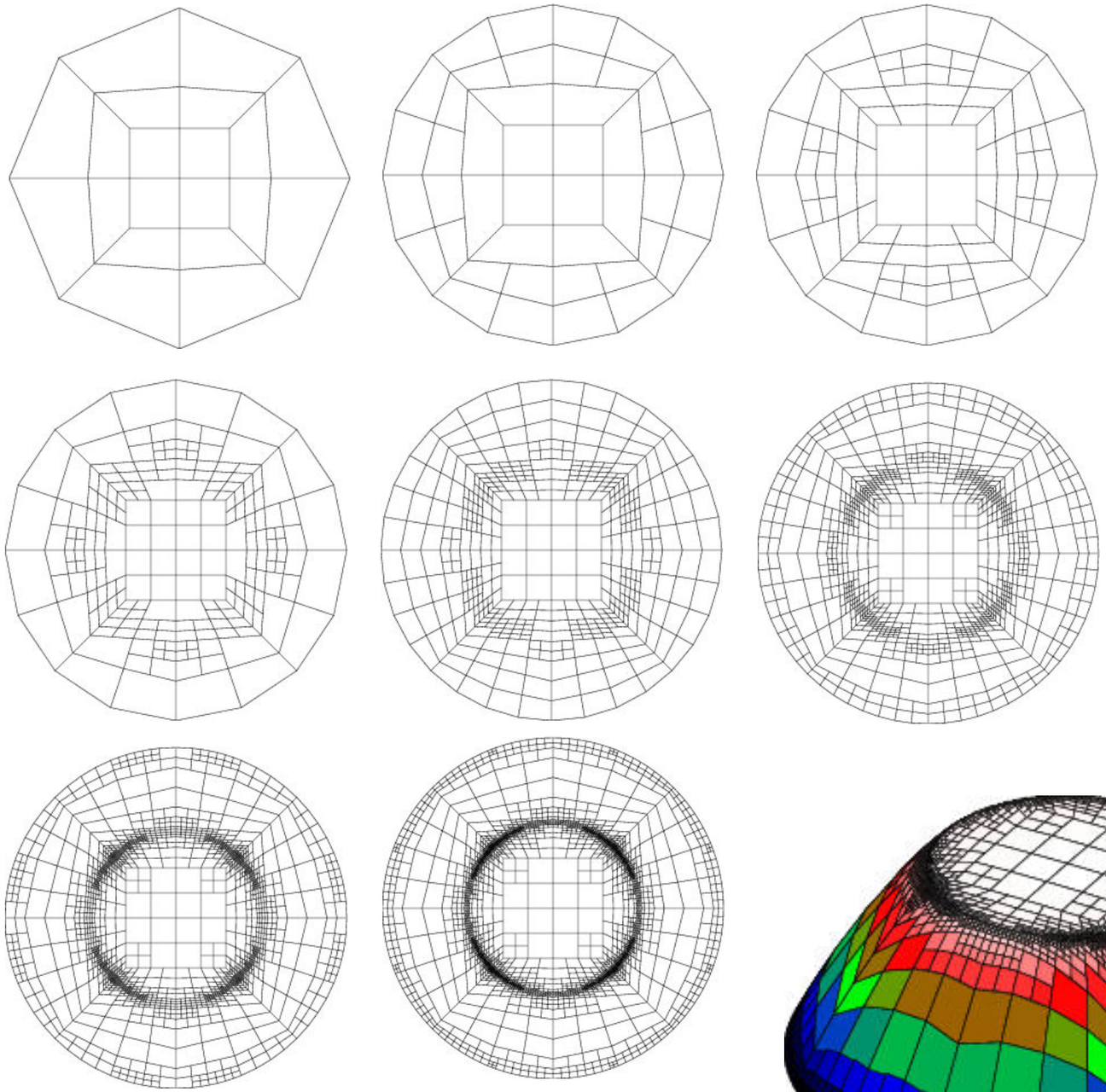
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Aims for this module

- Brief introduction to error estimation
- Implement adaptive mesh refinement
  - Hanging nodes
  - Error-based refinement marking
- Learn about the ConstraintMatrix

Adaptive mesh refinement

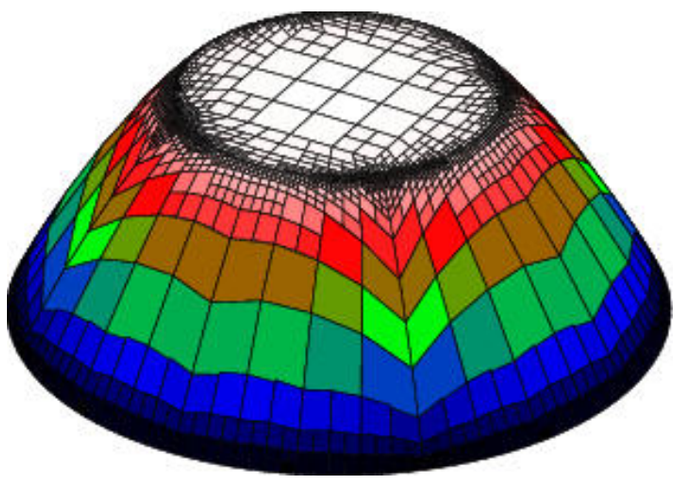
SOLVE — ESTIMATE — MARK — REFINE



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

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

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



Adaptive mesh refinement

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

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

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

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

thus 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 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-priori error estimation

$$-\Delta u = f \qquad u|_{\partial\Omega} = 0$$

Find $u \in \mathcal{V}$ such that:

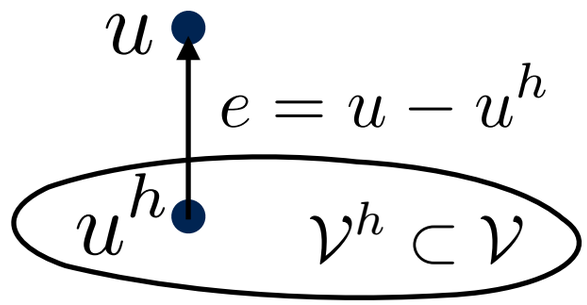
$$(\nabla u, \nabla v) = (f, v) \quad (*) \qquad \forall v \in \mathcal{V}$$

Find $u^h \in \mathcal{V}^h \subset \mathcal{V}$ such that:

$$(\nabla u^h, \nabla v^h) = (f, v^h) \quad (**) \qquad \forall v^h \in \mathcal{V}^h \subset \mathcal{V}$$

By taking $v \in \mathcal{V}^h$ in (*) and subtracting (**) we get "Galerkin orthogonality":

$$\left(\nabla \left[u - u^h\right], \nabla v^h\right) = 0 \qquad \forall v^h \in \mathcal{V}^h \subset \mathcal{V}$$



"orthogonality" because the bilinear form defines a scalar/inner product

Next, consider the "energy norm error":

$$\begin{aligned} ||\nabla \left[u - u^h\right]||^2 &\equiv \left(\nabla \left[u - u^h\right], \nabla \left[u - u^h\right]\right) \\ &= \left(\nabla \left[u - u^h\right], \nabla \left[u - u^h\right]\right) + \underbrace{\left(\nabla \left[u - u^h\right], \nabla v^h\right)}_{=0} \\ &= \left(\nabla \left[u - u^h\right], \nabla \left[u - u^h + v^h\right]\right) \quad \forall v^h \in \mathcal{V}^h \end{aligned}$$

Let $v^h = u^h - I^h u$ ... interpolant of the exact solution to the FE space (coincides with exact solution at nodes)

a-priori error estimation

…it then follows that

$$||\nabla \left[u - u^h\right]||^2 = (\nabla \left[u - u^h\right], \nabla \left[u - I^h u\right])$$

Recall the **Cauchy-Schwarz** inequality

$$(f, g) \leq ||f||\,||g|| \quad \forall f, g \in L_2$$

(similar to trigonometry and the scalar product between two vectors)

and therefore

$$||\nabla \left[u - u^h\right]||^2 \leq ||\nabla \left[u - u^h\right]||\,||\nabla \left[u - I^h u\right]||$$

$$||\nabla \left[u - u^h\right]|| \leq ||\nabla \left[u - I^h u\right]||$$

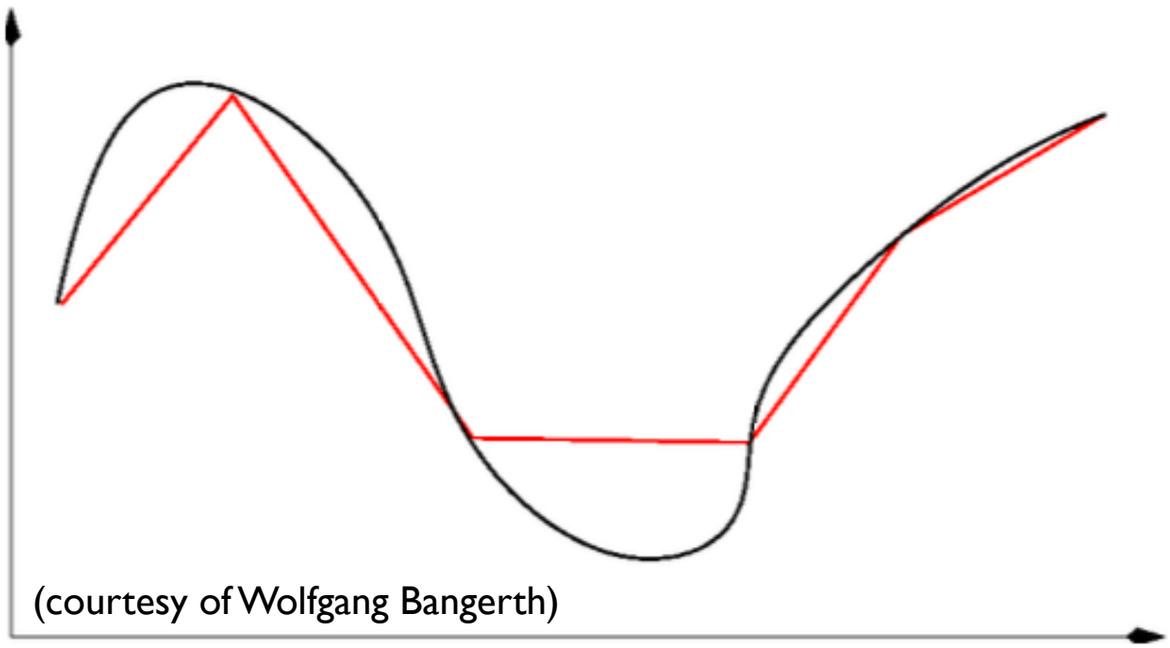
Interpretation: the FE error is no larger than the interpolation error. That is, it's closer (in energy norm) to the actual solution as compared to its interpolate onto FE space. Often called the "best approximation property".

…but we can't use it yet as we don't have exact solution! We need to have more knowledge about interpolant.

a-priori error estimation

Properties of the interpolant:

$$||\nabla \left[u - I^h u\right]||^2 = \int_\Omega |\nabla \left[u - I^h u\right]|^2 = \sum_K \int_K |\nabla \left[u - I^h u\right]|^2$$



Black: $u \in \mathcal{V}$

Red: $I^h u \in \mathcal{V}^h$

Intuitive observation: error is large where the second derivative of the exact solution is large.

The "Bramble-Hilbert Lemma" provides the following bound for **piecewise linear** elements:

$$||\nabla \left[u - I^h u\right]||_K^2 \equiv \int_K |\nabla \left[u - I^h u\right]|^2 \leq C^2 h_K^2 ||\nabla^2 u||_K^2$$

Consequently

$$||\nabla \left[u - I^h u\right]||_\Omega^2 \leq C^2 \sum_K h_K^2 ||\nabla^2 u||_K^2$$

from which it follows

(this is a-priori error estimator)

derivation: FE error bound by interpolation error which is bound by terms with mesh size and second derivatives

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

p.s. extension to degree p:

$$||\nabla \left[u - I^h u\right]||_\Omega^2 \leq C \sum_K h_K^{2p} ||\nabla^{p+1} u||_K^2 \equiv C \sum_K h_K^{2p} |u|_{H^{p+1}(K)}^2$$

a-priori error estimation

$$||u^h - u||^2_{H^1(\Omega)} := ||e||^2_{H^1(\Omega)} \leq C^2 \sum_K h_K^2 |u|^2_{H^2(K)} =: \sum_K e_K^2$$

- In practice not too useful as we rarely have any knowledge about the exact solution u.
- But it proves that the FE solution will converge as we can always make the mesh size smaller and smaller

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

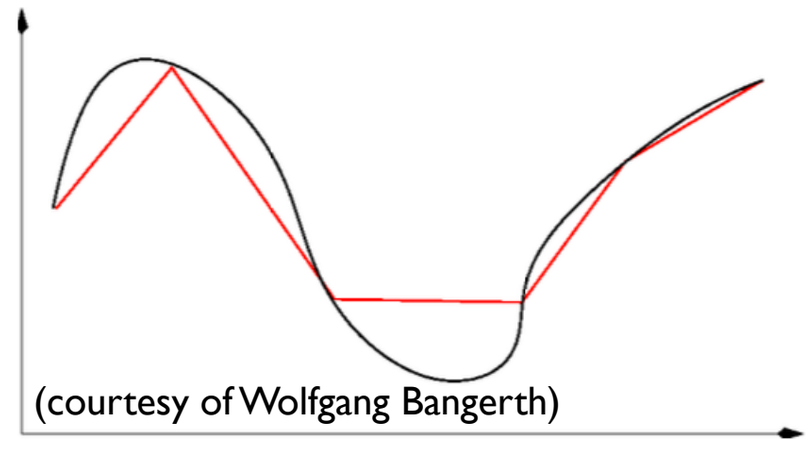
a better idea (in 1D) to approximate second derivatives at interface i:

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

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

can generalize to:

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



a-posteriori error estimation

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

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

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

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

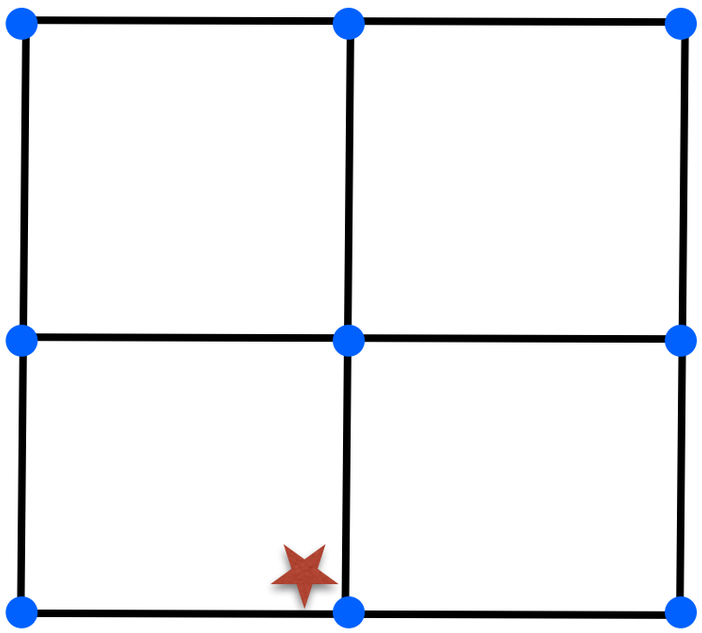
Note 1:

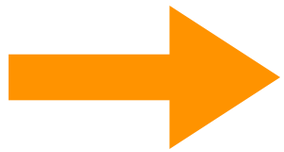
"**estimator**" is always a proven upper bound of error (*), whereas "**indicator**" is our best guess of error per cell which may not be an upper bound in the sense (*), but may still work well for considered equations and/or FE space.

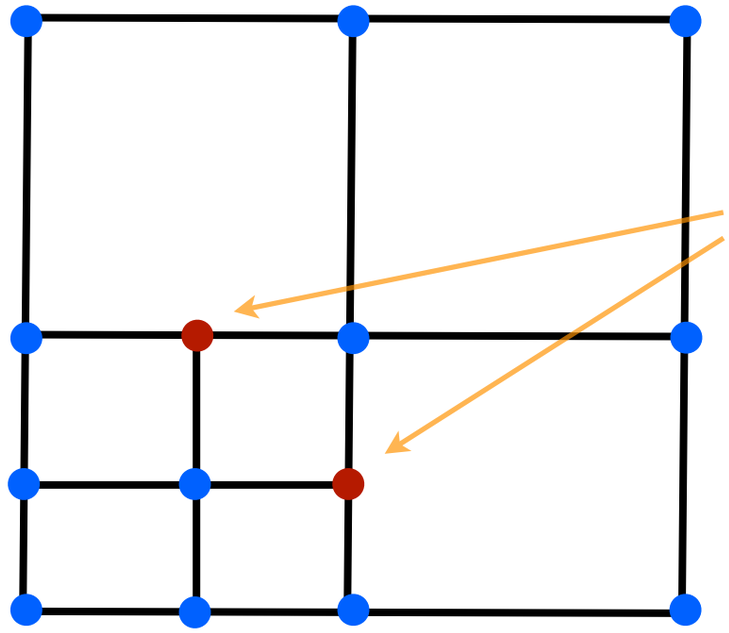
Adaptive mesh refinement

- 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 (quad/hex)



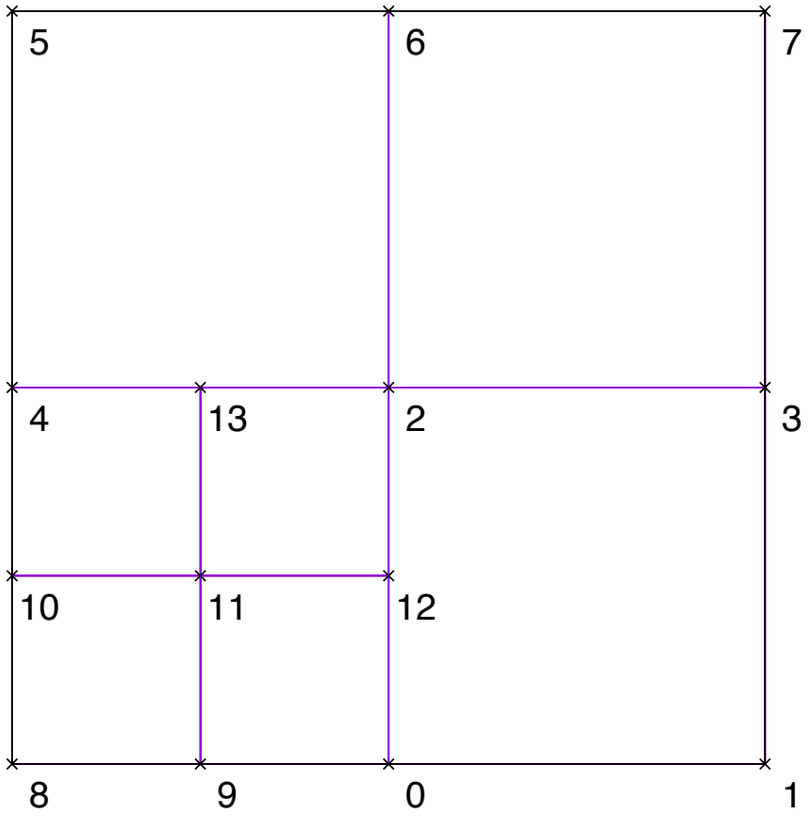




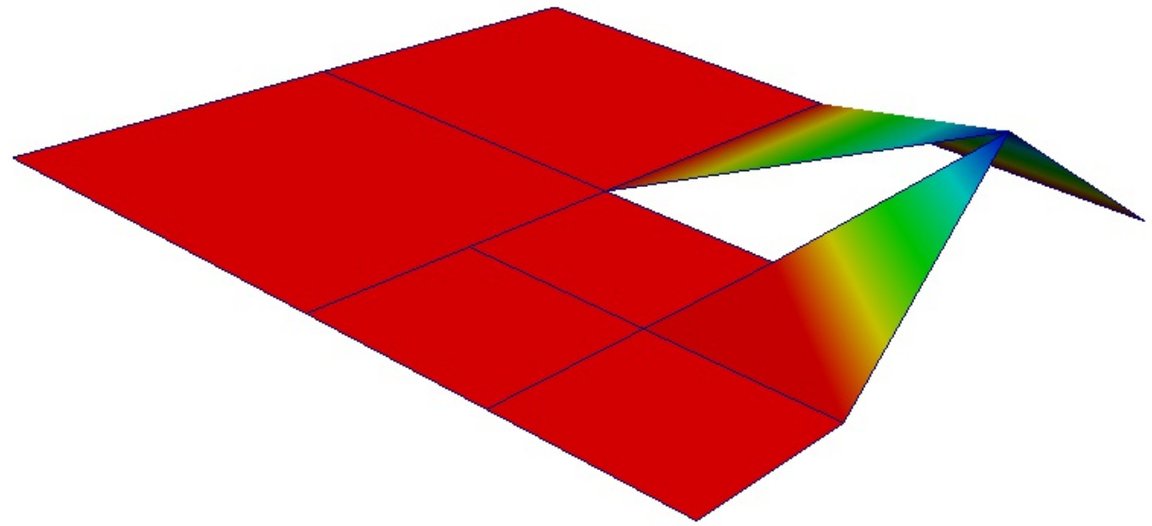
no easy options to keep the children of the top left/bottom right cell as quadrilaterals!

Solution: keep hanging nodes and “deal” with them in another way

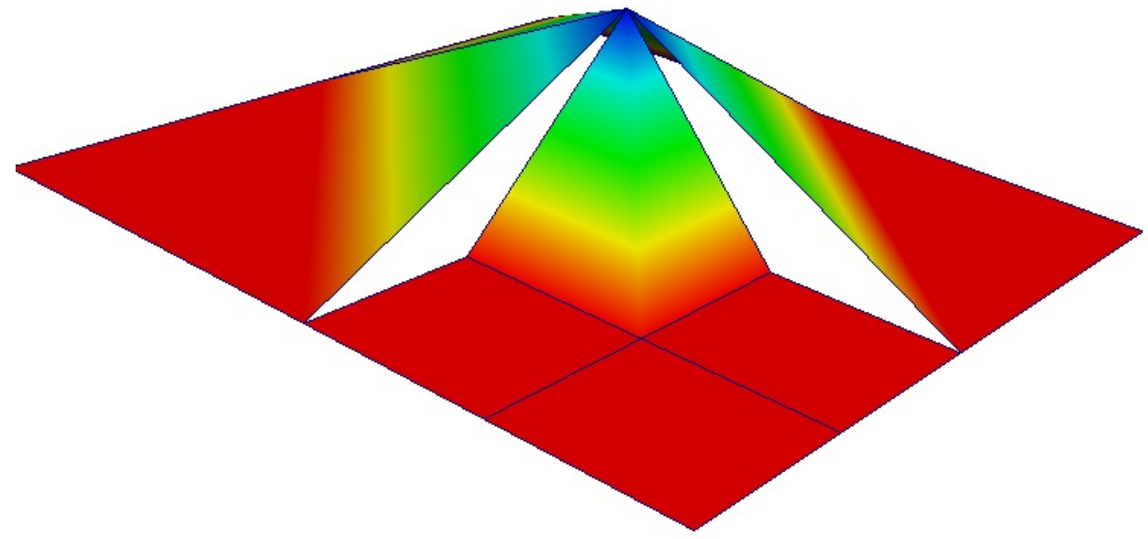
Hanging nodes



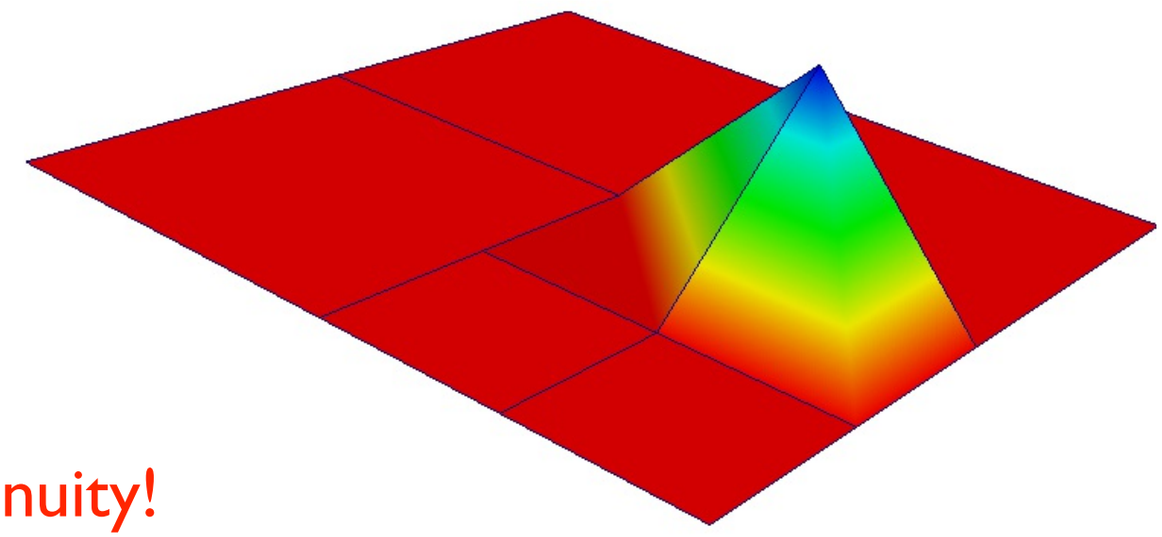
$N_0(\mathbf{x})$:



$N_2(\mathbf{x})$:



$N_{12}(\mathbf{x})$:



Discontinuous FE space!

Not a subspace of H^1

Bilinear forms would require special treatment as gradients are not defined everywhere

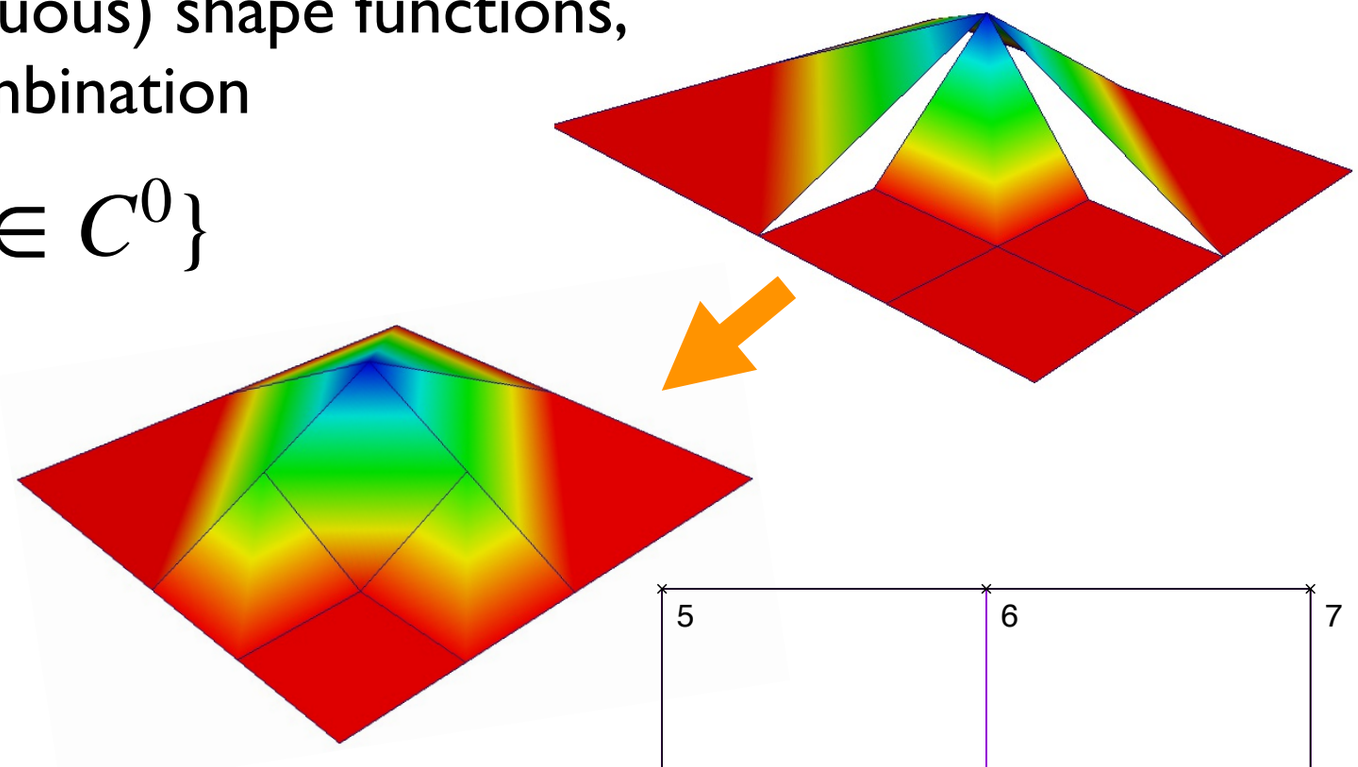
$$a(\phi_i, \phi_j) = \int_\Omega \nabla N_i(\mathbf{x}) \cdot \nabla N_j(\mathbf{x}) \mathrm{dv}$$

Solution: introduce constraints to require continuity!

Hanging nodes

Use standard (possibly globally discontinuous) shape functions, but require continuity of their linear combination

$$\mathcal{S}^h = \{u^h = \sum_i u_i N_i(\mathbf{x}) : u^h(\mathbf{x}) \in C^0\}$$



Note, that we encounter discontinuities along edges 0-12-2 and 2-13-4.

We can make the function continuous by making it continuous at vertices 12 and 13:

$$u_{12} = \frac{1}{2}u_0 + \frac{1}{2}u_2$$

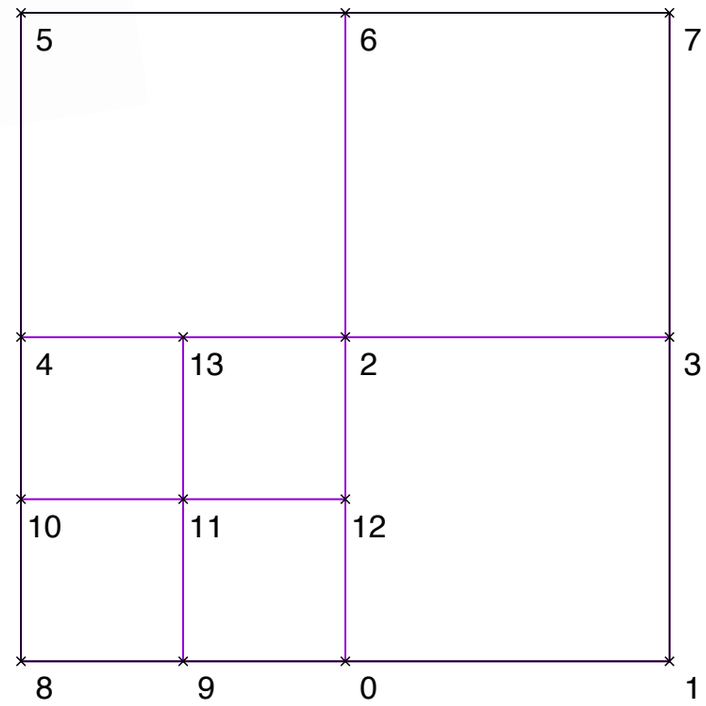
$$u_{13} = \frac{1}{2}u_2 + \frac{1}{2}u_4$$

define a subset of all DoFs to be constrained

$$\mathcal{N}_C \subset \mathcal{N}$$

The general form:

$$u_i = \sum_{j \in \mathcal{N}} c_{ij} u_j + b_i \quad \forall i \in \mathcal{N}_C$$

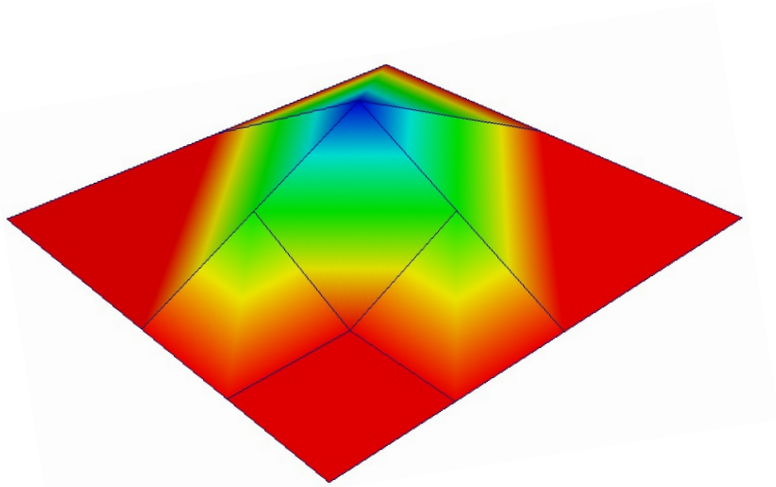


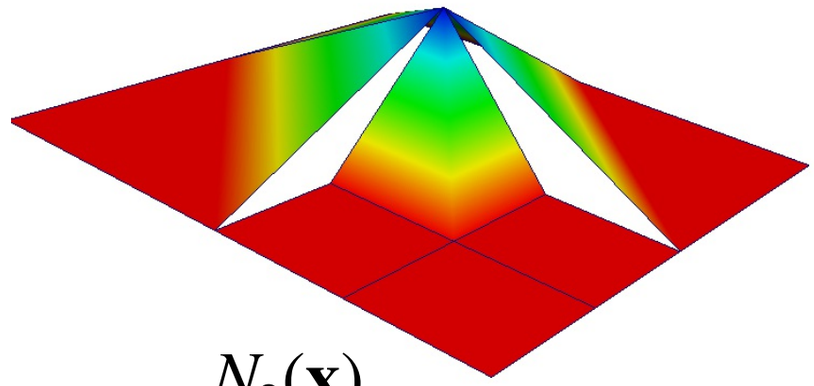
similar constraints arise from boundary conditions (normal/ tangential component) or hp-adaptive FE

Condensed shape functions

The alternative viewpoint is to construct a set of conforming shape functions:

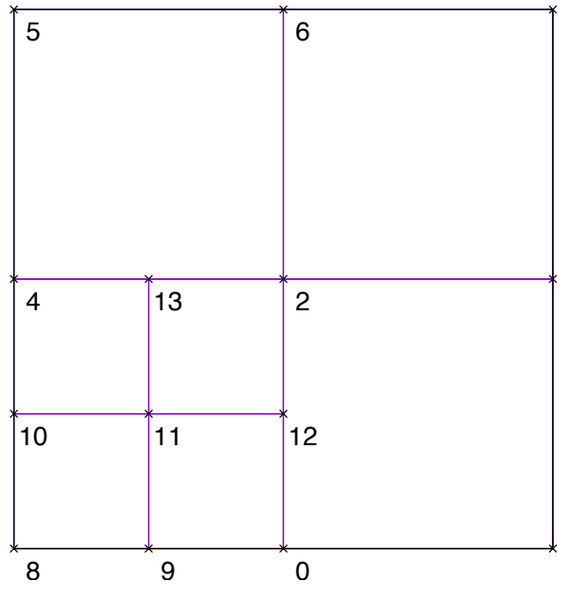
$$\widetilde{N}_2 := N_2 + \frac{1}{2}N_{13} + \frac{1}{2}N_{12}$$





$N_2(\mathbf{x})$

$$\mathcal{S}^h = \{u^h = \sum_{i \in \mathcal{N}/\mathcal{N}_c} u_i \widetilde{N}_i(\mathbf{x})\}$$



$$[\boldsymbol{K}]_{ij} = \begin{cases} a(\widetilde{N}_i, \widetilde{N}_j) & \text{if } i \in \mathcal{N} \setminus \mathcal{N}_c \text{ and } j \in \mathcal{N} \setminus \mathcal{N}_c \\ 1 & \text{if } i \equiv j \text{ and } j \in \mathcal{N}_c \\ 0 & \text{otherwise} \end{cases}$$

$$[\boldsymbol{F}]_i = \begin{cases} (f, \widetilde{N}_i) & \text{if } i \in \mathcal{N} \setminus \mathcal{N}_c \\ 0 & \text{otherwise} \end{cases}$$

The beauty of the approach is that we can assemble local matrix and RHS as usual and then obtain condensed forms in a separate step, i.e

$$\forall i \in \mathcal{N}\setminus\mathcal{N}_c: \quad \left[\mathbf{F}\right]_i = (f, \widetilde{N}_i) = (f, N_i + \sum_{j \in \mathcal{N}_c} c_{ji} N_j) = (f, N_i) + \sum_{j \in \mathcal{N}_c} c_{ji}(f, N_j) = \left[\widetilde{\mathbf{F}}\right]_i + \sum_{j \in \mathcal{N}_c} c_{ji} \left[\widetilde{\mathbf{F}}\right]_j$$

Algebraic form

$$\boldsymbol{x} =: \begin{bmatrix} \boldsymbol{x}_u \\ \boldsymbol{x}_m \\ \boldsymbol{x}_c \end{bmatrix}$$

not involved in constrains

master dofs in constrains

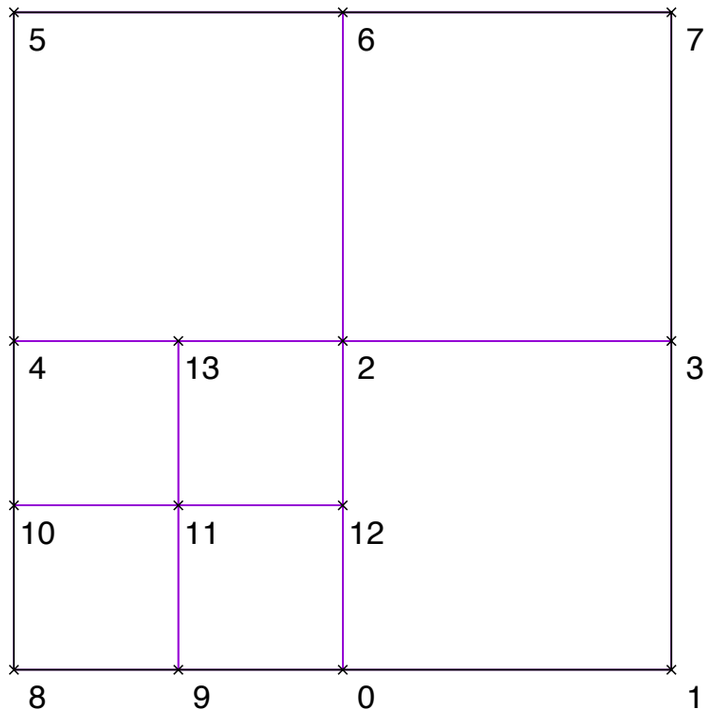
constrained dofs

$$\begin{bmatrix} u_{12} \\ u_{13} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} u_0 \\ u_2 \\ u_4 \end{bmatrix}$$

$$\boldsymbol{x}_c = \boldsymbol{C}\boldsymbol{x}_m$$

$$\boldsymbol{\Gamma} := \begin{bmatrix} \boldsymbol{I}_u & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_m \\ \boldsymbol{0} & \boldsymbol{C} \end{bmatrix}$$

transformation matrix



The condensed matrix and vector can be written as

$$\boldsymbol{K} = \boldsymbol{\Gamma}^T \widetilde{\boldsymbol{K}} \boldsymbol{\Gamma} \qquad \boldsymbol{F} = \boldsymbol{\Gamma}^T \widetilde{\boldsymbol{F}}$$

To look closer at what condensation does, write the original matrix/vector in block form:

$$\widetilde{\boldsymbol{K}} =: \begin{bmatrix} \boldsymbol{K}_{uu} & \boldsymbol{K}_{um} & \boldsymbol{K}_{uc} \\ \boldsymbol{K}_{mu} & \boldsymbol{K}_{mm} & \boldsymbol{K}_{mc} \\ \boldsymbol{K}_{cu} & \boldsymbol{K}_{cm} & \boldsymbol{K}_{cc} \end{bmatrix} \qquad \widetilde{\boldsymbol{F}} =: \begin{bmatrix} \boldsymbol{F}_u \\ \boldsymbol{F}_m \\ \boldsymbol{F}_c \end{bmatrix} \qquad \boldsymbol{F} =: \begin{bmatrix} \boldsymbol{F}_u \\ \boldsymbol{F}_m + \boldsymbol{C}^T \boldsymbol{F}_c \end{bmatrix}$$

$$\boldsymbol{K} =: \begin{bmatrix} \boldsymbol{K}_{uu} & \boldsymbol{K}_{um} + \boldsymbol{K}_{uc}\boldsymbol{C} \\ \boldsymbol{K}_{mu} + \boldsymbol{C}^T \boldsymbol{K}_{cu} & \boldsymbol{K}_{mm} + \boldsymbol{C}^T \boldsymbol{K}_{cm} + \boldsymbol{K}_{mc}\boldsymbol{C} + \boldsymbol{C}^T \boldsymbol{K}_{cc}\boldsymbol{C} \end{bmatrix}$$

Using constraints:

- The beauty of the FEM is that we do exactly the same thing on every cell
- In other words: assembly on cells with hanging nodes should work exactly as on cells without

Approach 1:

$$\widetilde{\mathcal{S}}^h = \{u^h = \sum_i u_i N_i(x)\}$$

this is not a continuous space, but we may still use it as an intermediate step for matrices!

$$\mathcal{S}^h = \{u^h = \sum_i u_i N_i(x) : u^h(x) \in C^0\}$$

Step 1: Build matrix/rhs $\widetilde{\mathbf{K}}, \widetilde{\mathbf{F}}$ with all DoFs as if there were no constraints.

Step 2: Modify $\widetilde{\mathbf{K}}, \widetilde{\mathbf{F}}$ to get $\mathbf{K}, \mathbf{F}$

i.e. eliminate the rows and columns of the matrix that correspond to constrained degrees of freedom

Step 3: Solve $\mathbf{K} \cdot \mathbf{u} = \mathbf{F}$

Step 4: Fill in the constrained components of $\mathbf{u}$ to use $\widetilde{\mathcal{S}}^h$ for evaluation of the field.

Disadvantages: (i) bottleneck for 3d or higher order/hp FEM; (ii) hard to implement in parallel where a process may not have access to all elements of the matrix; (iii) two matrices may have different sparsity pattern.

Approach 2:

$$\widetilde{\mathcal{S}}^h = \{u^h = \sum_i u_i N_i(x)\}$$

$$\mathcal{S}^h = \{u^h = \sum_i u_i N_i(x) : u^h(x) \in C^0\}$$

Step 1: Build local matrix/rhs $\widetilde{\mathbf{K}}_K$, $\widetilde{\mathbf{F}}_K$ with all DoFs as if there were no constraints.

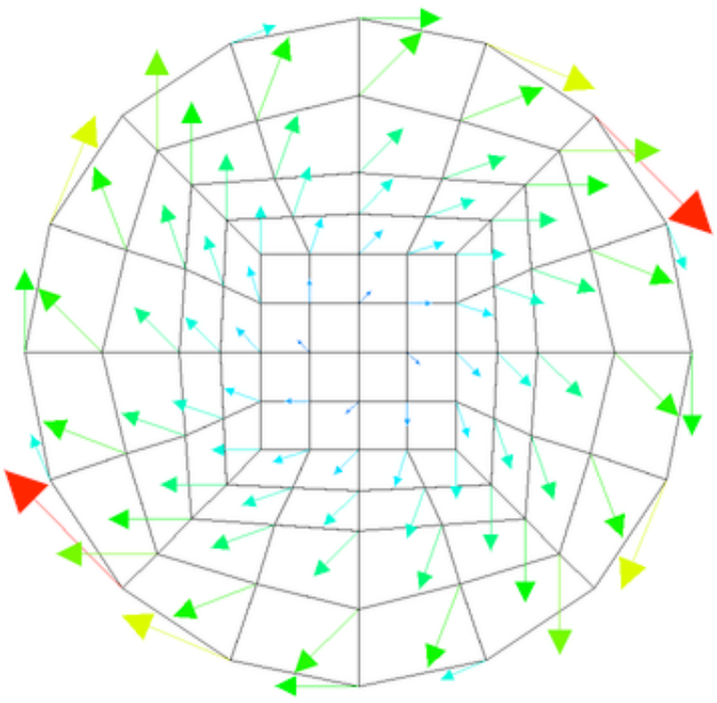
Step 2: Apply constraints during assembly operation (local-to-global) $\mathbf{K}_K, \mathbf{F}_K$

Step 3: Solve $\mathbf{K} \cdot \mathbf{u} = \mathbf{F}$

Step 4: Fill in the constrained components of $\mathbf{u}$ to use $\widetilde{\mathcal{S}}^h$ for evaluation of the field.

Applying constraints: the ConstraintMatrix class

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



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 `DoFTools::make_sparsity_pattern (..., constraints, / *keep_constrained_dofs = * / false)`
  - Sorted, rearrange, optimise constraints `constraints.close()`

Applying constraints: the ConstraintMatrix class

- 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);
```

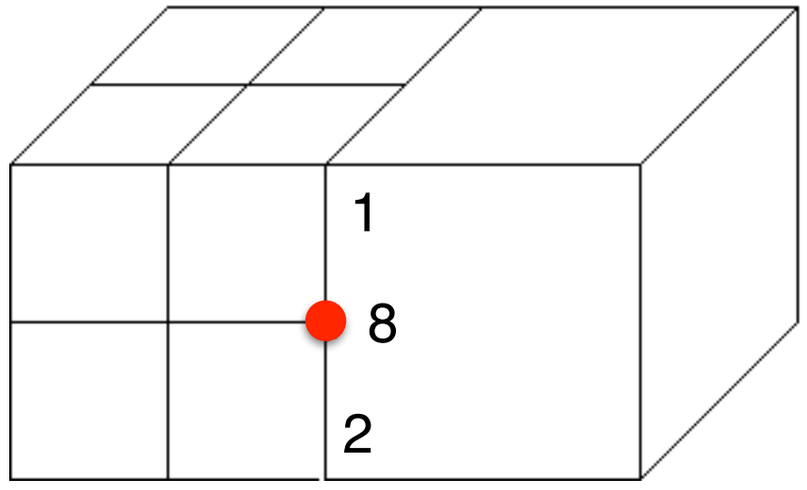
  - Solve and then set all constraint values correctly: `ConstraintMatrix::distribute(...)`

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

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

Adaptive mesh refinement

- Note: `Triangulation::MeshSmoothing`