Master in High Performance Computing

Numerical Solution of PDEs using the Finite Element Method

The devil is in the details: boundary conditions and constraints

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)

Poisson problem revisited

Homogeneous Dirichlet case, constant coefficient equal to 1:

$$
\begin{aligned}
-\Delta u &= f && \text{in } \Omega \\
u &= 0 && \text{on } \partial\Omega
\end{aligned}
$$

$$\gamma_\Gamma : H^1(\Omega) \mapsto H^{\frac{1}{2}}(\Gamma) \quad \text{Trace operator}$$

$$V := H^1_0(\Omega) := \{v \,|\, v \in L^2(\Omega), \nabla v \in L^2(\Omega), \gamma_{\partial\Omega} v = 0\}$$

Weak form: given $f \in V^*$, find $u \in V$ such that

$$(\nabla u, \nabla v) = (f, v) \qquad \forall v \in V$$

Poisson problem revisited

Non-homogeneous Dirichlet case, constant coefficient equal to 1:

$$-\Delta u = f \qquad \text{in } \Omega$$
$$u = g \qquad \text{on } \partial\Omega$$

$$V_0 := H^1_0(\Omega) := \{v \mid v \in L^2(\Omega), \nabla v \in L^2(\Omega), \gamma_{\partial\Omega} v = 0\}$$

$V_g := V_0 + u_D$ Where $\gamma_{\partial\Omega} u_D = g$

Weak form: given $f \in V^*$, find $u \in V_g$ such that

$$(\nabla u, \nabla v) = (f, v) \qquad \forall v \in V_0$$

mathLab innovating with mathematics

ICTP The Abdus Salam International Centre for Theoretical Physics

SISSA

Poisson problem revisited

Mixed boundary conditions, non-constant coefficients

$$
\begin{aligned}
-\nabla \cdot (a \nabla u) &= f && \text{in } \Omega \\
u &= g_D && \text{on } \Gamma_D \\
n \cdot (a \nabla u) &= g_N && \text{on } \Gamma_N
\end{aligned}
$$

$$V_{0,\Gamma_D} := H^1_0(\Omega) := \{ v \,|\, v \in L^2(\Omega), \nabla v \in L^2(\Omega), \gamma_{\Gamma_D} v = 0 \}$$

$$V_{g_D,\Gamma_D} := V_{0,\Gamma_D} + u_D$$ Where $\gamma_{\Gamma_D} u_D = g_D$

Weak form: given $f \in V^*_{0,\Gamma_D}$, find $u \in V_{g_D,\Gamma_D}$ such that

$$(a \nabla u, \nabla v) = (f, v) + \int_{\Gamma_N} g_N v \qquad \forall v \in V_{0,\Gamma_D}$$

Trial spaces VS test spaces

$$V_{0,\Gamma_D} := H^1_0(\Omega) := \{v \,|\, v \in L^2(\Omega), \nabla v \in L^2(\Omega), \gamma_{\Gamma_D} v = 0\}$$

$V_{g_D,\Gamma_D} := V_{0,\Gamma_D} + u_D$ Where $\gamma_{\Gamma_D} u_D = g_D$

Weak form: given $f \in V^*_{0,\Gamma_D}$, find $u \in V_{g_D,\Gamma_D}$ such that

$$(a\nabla u, \nabla v) = (f, v) + \int_{\Gamma_N} g_N v \qquad \forall v \in V_{0,\Gamma_D}$$

CANNOT apply Lax-Milgram: $V_{0,\Gamma_D} \neq V_{g_D,\Gamma_D}$

Trial spaces VS test spaces

$$V_{0,\Gamma_D} := H^1_0(\Omega) := \{v \,|\, v \in L^2(\Omega), \nabla v \in L^2(\Omega), \gamma_{\Gamma_D} v = 0\}$$

$V_{g_D,\Gamma_D} := V_{0,\Gamma_D} + u_D$ Where $\gamma_{\Gamma_D} u_D = g_D$

Weak form: given $f \in V^*_{0,\Gamma_D}$, find $u_0 \in V_{0,\Gamma_D}$ such that

$$(a\nabla u_0, \nabla v) = (f, v) + \int_{\Gamma_N} (g_N - n \cdot (a \nabla u_D))v - (a \nabla u_D, \nabla v) \qquad \forall v \in V_{0,\Gamma_D}$$

Write $u = u_0 + u_D$ (now we can apply Lax-Milgram)

where u_D is arbitrary, and such that $\gamma_{\Gamma_D} u_D = g_D$

How to implement V_{g_D,Γ_D}, V_{0,Γ_D}?

- Option 1 (**not implemented in deal.II**):
  encode in DoFHandler (n_dofs of $H^1_{0,\Gamma_D}(\Omega)$ < n_dofs of $H^1(\Omega)$)
  and in basis functions (i.e., $\gamma_{\Gamma_D} v_i = 0 \quad \forall v_i \in V_h$)

- Option 2 (Penalty methods, Lagrange multipliers):
  impose boundary conditions weakly

- Option 3 (Algebraic approach: strong imposition):
  post-process Linear systems, solution vectors, and rhs vectors to **set to** g_D degrees of freedom with support points on Γ_D

Algebraic approach

- Main idea: assemble matrix $\tilde{A}_{ij} := (a\nabla v_j, \nabla v_i)$

  and right-hand-side $\tilde{F}_i := (f, v_i) + \int_{\Gamma_N} g_N v_i$

- split dofs $u = \begin{pmatrix} u_{\Omega\cup\Gamma_N} \equiv u_O \\ u_C \end{pmatrix}$ $\tilde{F} = \begin{pmatrix} F_O \\ F_C \end{pmatrix}$

- and matrix $\tilde{A} = \begin{pmatrix} A_{OO} & A_{OC} \\ A_{CO} & A_{CC} \end{pmatrix}$

- where “C” stands for “constrained”

Mimic continuous approach

- compute g_D, using VectorTools::interpolate_boundary_values
- eliminate row "C" from $\tilde{A}$, and set rhs $\tilde{F}_C \mapsto g_D$: $$\begin{pmatrix} A_{OO} & A_{OC} \\ 0 & I_{CC} \end{pmatrix} \begin{pmatrix} u_O \\ u_D \end{pmatrix} = \begin{pmatrix} \tilde{F}_O \\ g_D \end{pmatrix}$$
- "move" A_{OC} to rhs to restore symmetry in matrix: $$\begin{pmatrix} A_{OO} & 0 \\ 0 & I_{CC} \end{pmatrix} \begin{pmatrix} u_O \\ u_D \end{pmatrix} = \begin{pmatrix} \tilde{F}_O - A_{OC} g_D \\ g_D \end{pmatrix}$$
- rescale I_{CC} for conditioning: $$\begin{pmatrix} A_{OO} & 0 \\ 0 & \alpha I_{CC} \end{pmatrix} \begin{pmatrix} u_O \\ u_D \end{pmatrix} = \begin{pmatrix} \tilde{F}_O - A_{OC} g_D \\ \alpha g_D \end{pmatrix}$$

MatrixTools::apply_boundary_values $$\tilde{A} \mapsto \begin{pmatrix} A_{OO} & 0 \\ 0 & \alpha I_{CC} \end{pmatrix} \qquad u \mapsto \begin{pmatrix} u_O \\ u_D \end{pmatrix} \qquad \tilde{F} \mapsto \begin{pmatrix} \tilde{F}_O - A_{OC} g_D \\ \alpha g_D \end{pmatrix}$$

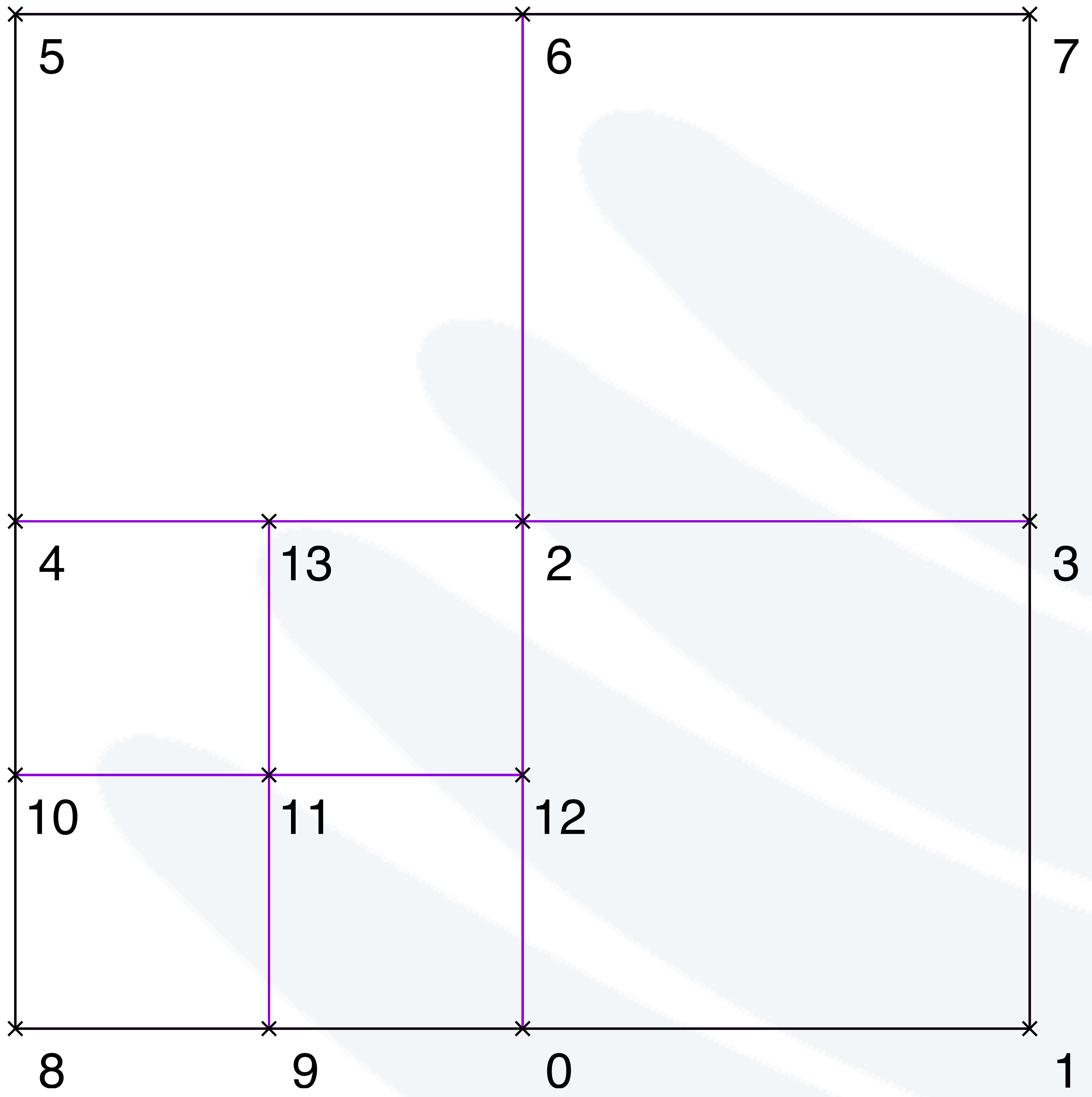
Special case of AffineConstraints

- General case: constrained dofs are a subset of all dofs $\mathcal{N}_C \subset \mathcal{N}$
- AffineConstraints: $x_i = \sum_{j \in \mathcal{N} \setminus \mathcal{N}_C} C_{ij} x_j + b_i \qquad \forall i \in \mathcal{N}_C$
- Algebraic solution can be performed efficiently as a three-step process:
  - Condense
  - Solve
  - Distribute (only needed if $C \neq 0$)

Condense-Solve-Distribute

- Given, $\tilde{A} = \begin{pmatrix} A_{OO} & A_{OC} \\ A_{CO} & A_{CC} \end{pmatrix}$, $\tilde{F} = \begin{pmatrix} F_O \\ F_C \end{pmatrix}$, and constraints $u_C = Cu_O + b$
- Take constraints into accounts in "O": $A_{OO}u_O + A_{OC}u_C = (A_{OO} + A_{OC}C)u_O + A_{OC}b = F_O$
- Ignore rows "C" in matrix and rhs and solve $Au = F$ where
  - $\tilde{A} = \begin{pmatrix} A_{OO} & A_{OC} \\ A_{CO} & A_{CC} \end{pmatrix} \mapsto A = \begin{pmatrix} A_{OO} + A_{OC}C & 0 \\ 0 & \alpha I_{CC} \end{pmatrix}$
  - $\tilde{F} = \begin{pmatrix} F_O \\ F_C \end{pmatrix} \mapsto F = \begin{pmatrix} F_O - A_{OC}b \\ \alpha b \end{pmatrix}$
- Distribute constraints: $u = \begin{pmatrix} u_O \\ b \end{pmatrix} \mapsto u = \begin{pmatrix} u_O \\ Cu_O + b \end{pmatrix}$

Hanging nodes

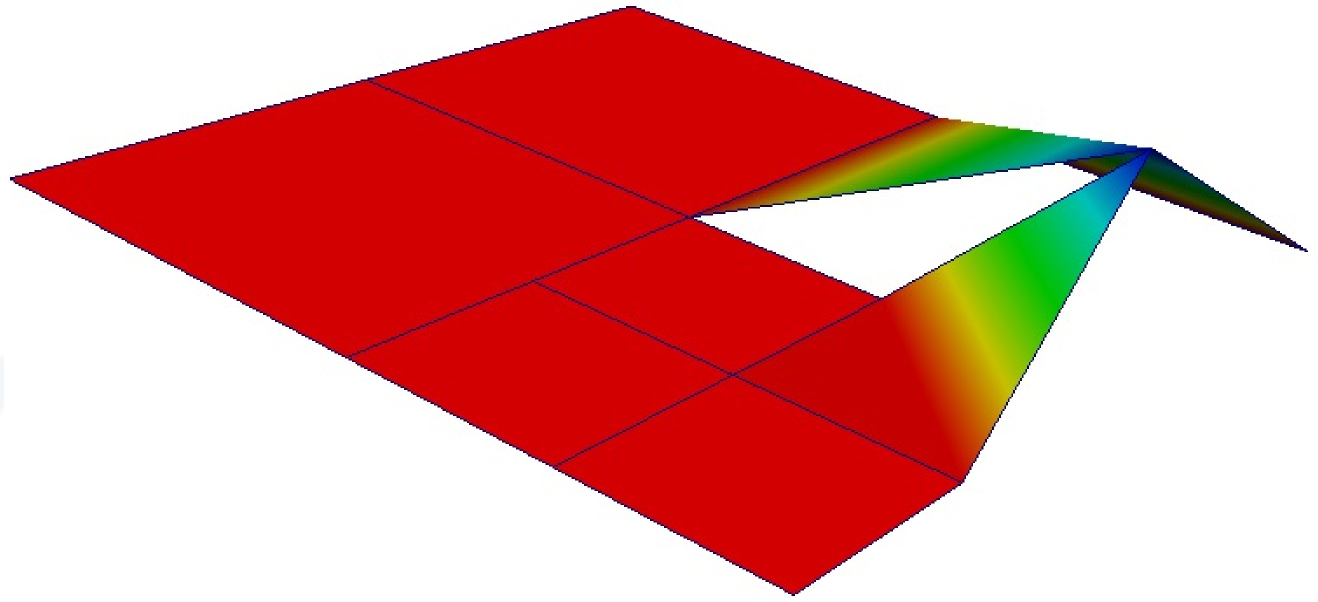


Discontinuous FE space!

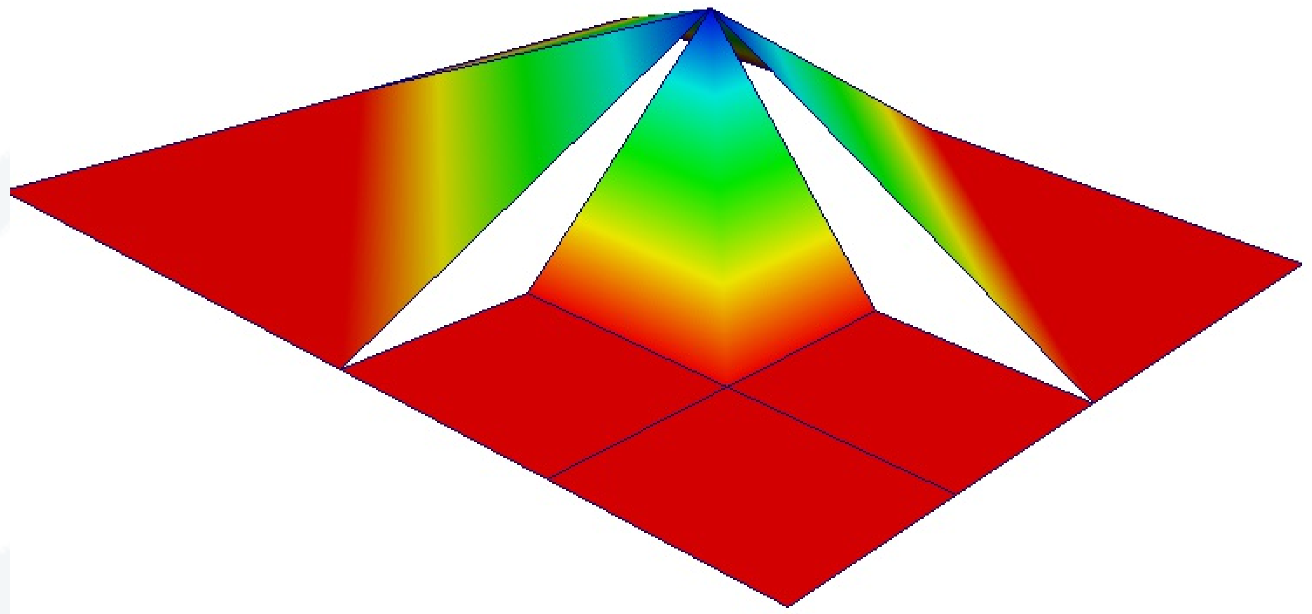
Not a subspace of H^1

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

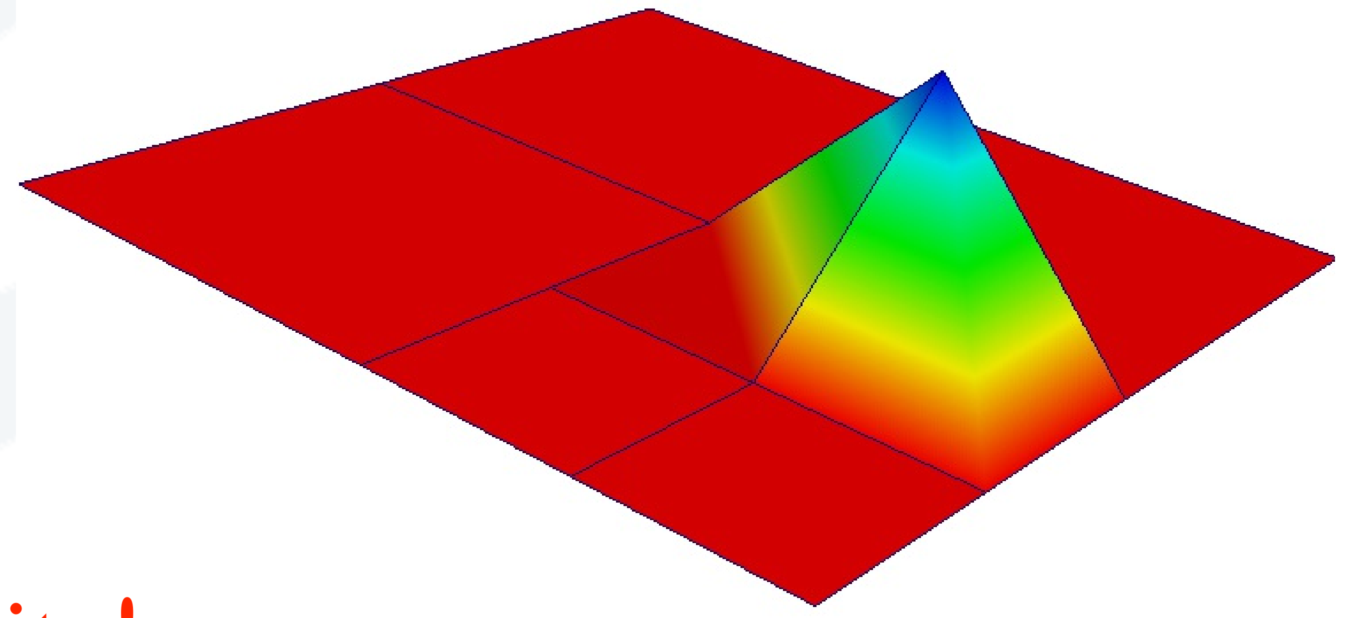
$N_0(\mathbf{x})$:



$N_2(\mathbf{x})$:



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

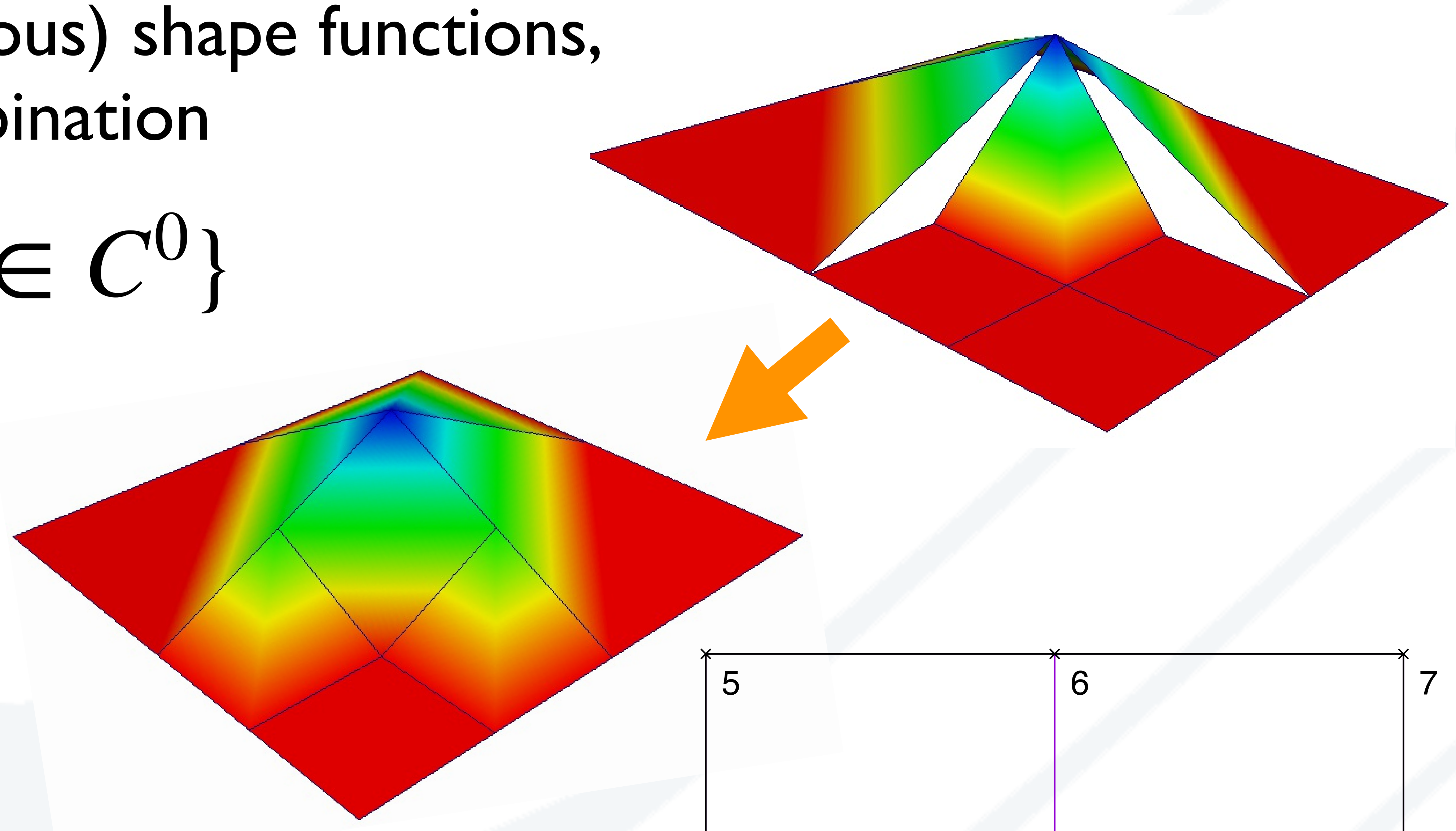


Solution: introduce constraints to require continuity!

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.

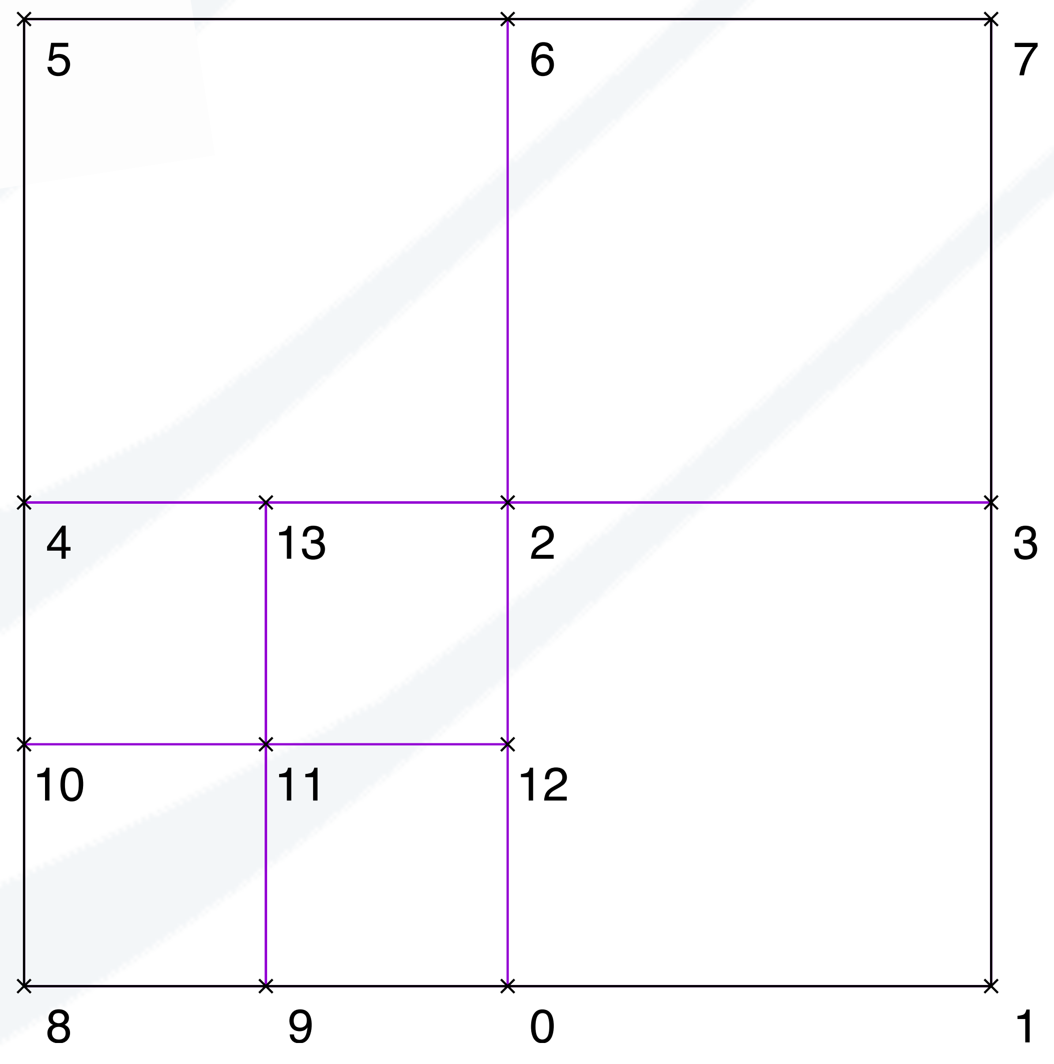
We can make the function continuous by making it continuous at vertices 12 and 13:

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

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

define a subset of all DoFs to be constrained

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



The general form:

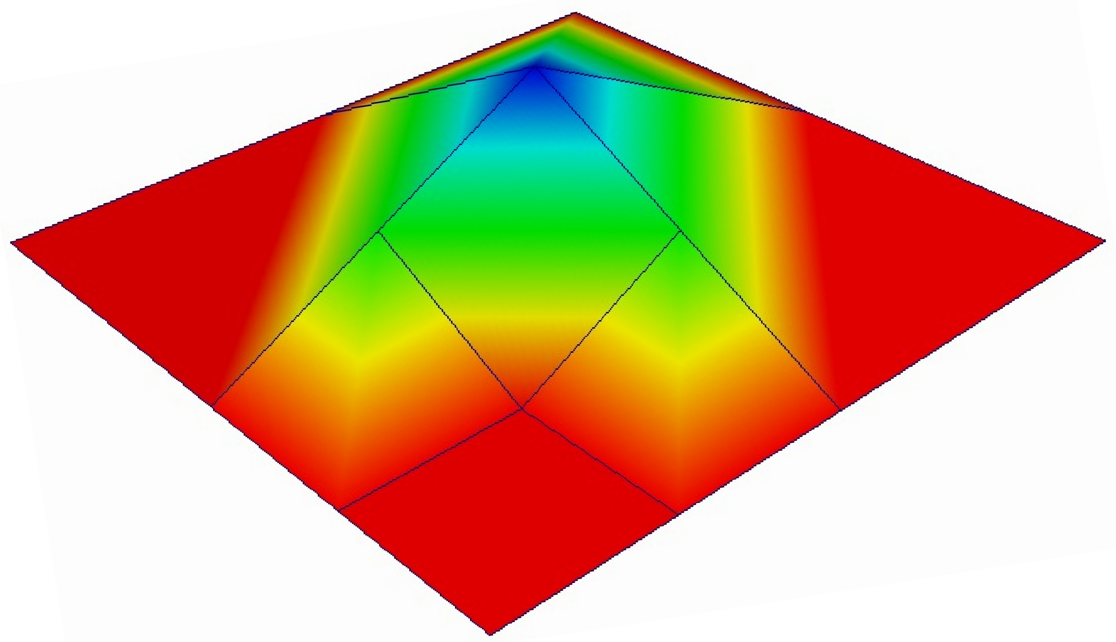
$$u_i = \sum_{j\in\mathcal{N}} c_{ij}\, u_j + b_i \quad \forall i \in \mathcal{N}_C$$

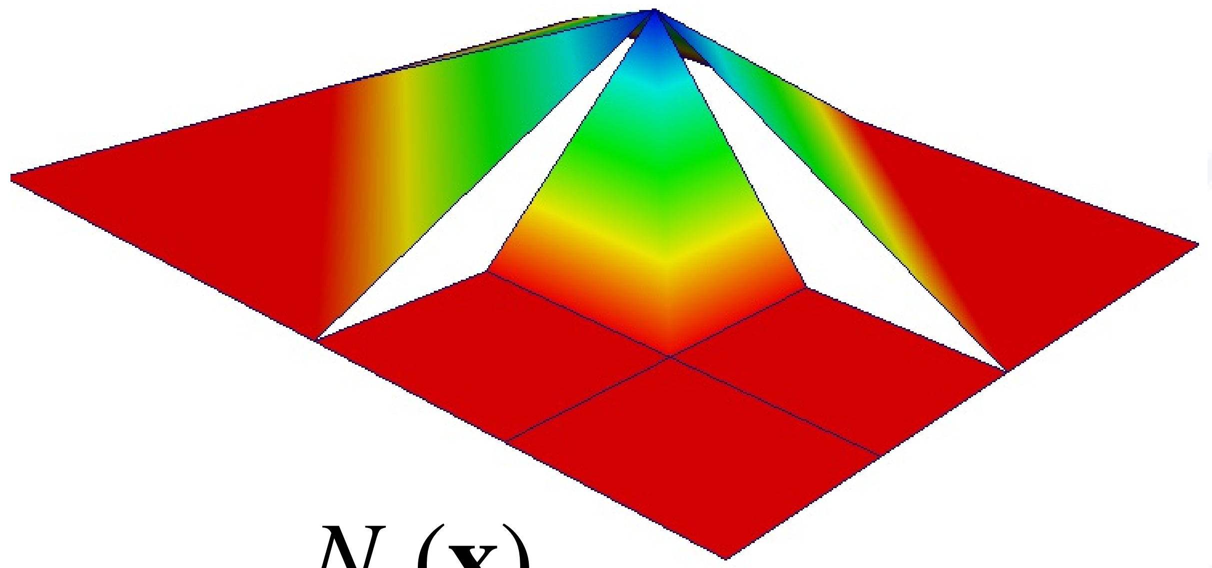
similar constraints arise from boundary conditions (normal/ tangential component) or hp-adaptive FE

Condensed shape functions

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

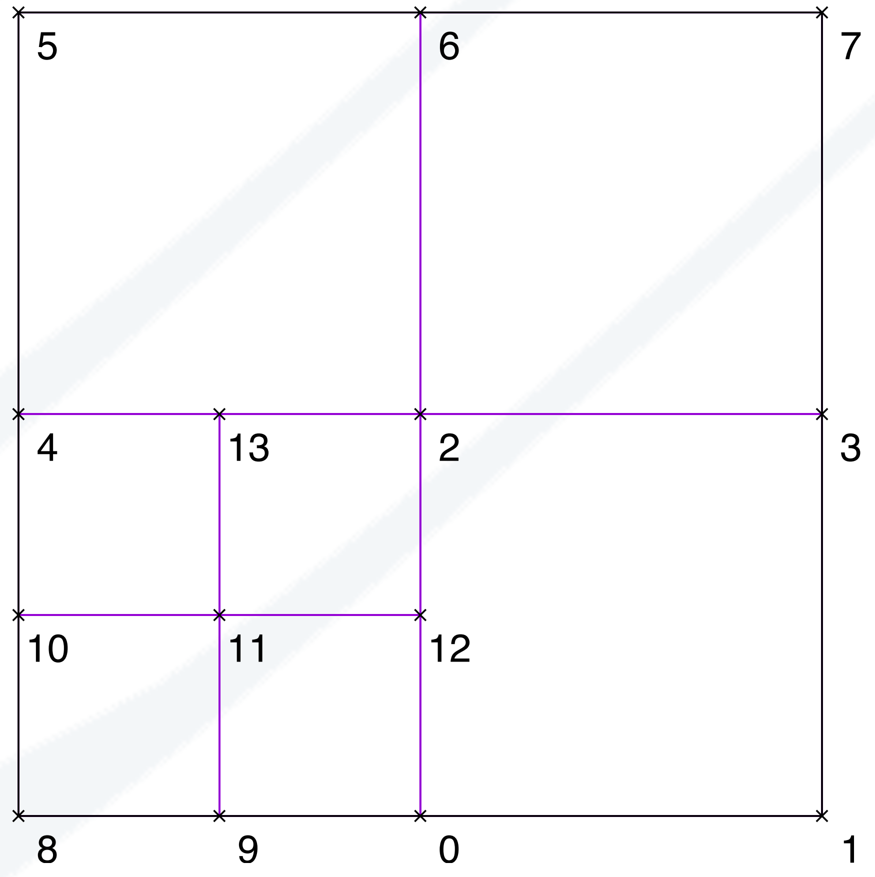
$$\widetilde{N}_2 := N_2 + \frac{1}{2}N_{13} + \frac{1}{2}N_{12}$$





$N_2(\mathbf{x})$

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



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

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

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

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

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

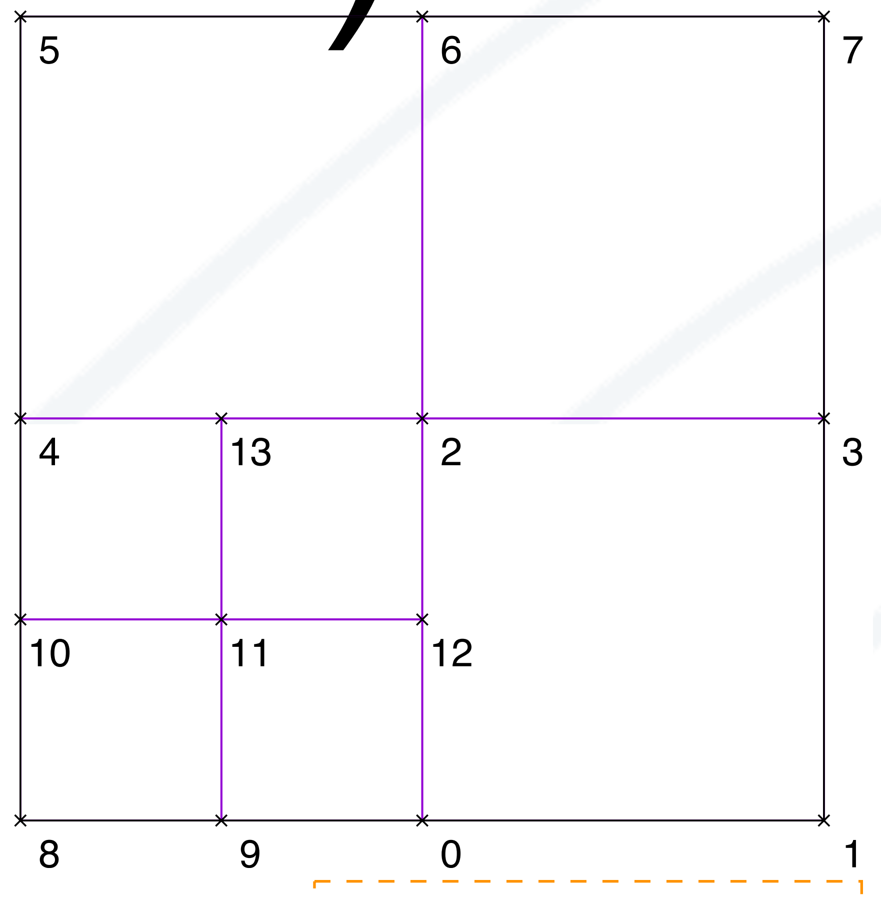
Step 3: Solve $\mathbf{K} \cdot \mathbf{u} = \mathbf{F}$

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 1 (example):

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




```
============================================
Number of active cells: 7
Number of degrees of freedom: 14
=============== constraints ===============
    12 0:  0.5
    12 2:  0.5
    13 2:  0.5
    13 4:  0.5
============== un-condensed ===============
================= matrix ==================
 1.333e+00 -1.667e-01 -1.667e-01 -3.333e-01  0.000e+00                                             -1.667e-01            -3.333e-01 -1.667e-01
-1.667e-01  6.667e-01 -3.333e-01 -1.667e-01
-1.667e-01 -3.333e-01  2.667e+00 -3.333e-01 -1.667e-01 -3.333e-01 -3.333e-01 -3.333e-01             0.000e+00  0.000e+00 -3.333e-01 -1.667e-01 -1.667e-01
-3.333e-01 -1.667e-01 -3.333e-01  1.333e+00                       -3.333e-01 -1.667e-01
 0.000e+00            -1.667e-01             1.333e+00 -1.667e-01 -3.333e-01                                  -1.667e-01 -3.333e-01            -1.667e-01
                      -3.333e-01            -1.667e-01  6.667e-01 -1.667e-01
                      -3.333e-01 -3.333e-01 -3.333e-01 -1.667e-01  1.333e+00 -1.667e-01
                      -3.333e-01 -1.667e-01                       -1.667e-01  6.667e-01
                                                                                         6.667e-01 -1.667e-01 -1.667e-01 -3.333e-01
-1.667e-01             0.000e+00                                                        -1.667e-01  1.333e+00 -3.333e-01 -3.333e-01 -3.333e-01
                       0.000e+00            -1.667e-01                                  -1.667e-01 -3.333e-01  1.333e+00 -3.333e-01            -3.333e-01
-3.333e-01            -3.333e-01            -3.333e-01                                  -3.333e-01 -3.333e-01 -3.333e-01  2.667e+00 -3.333e-01 -3.333e-01
-1.667e-01            -1.667e-01                                                                   -3.333e-01            -3.333e-01  1.333e+00 -3.333e-01
                      -1.667e-01            -1.667e-01                                                        -3.333e-01 -3.333e-01 -3.333e-01  1.333e+00
```

```
================ condensed ================
================= matrix ==================
 1.500e+00 -1.667e-01 -8.333e-02 -3.333e-01 -8.333e-02                                             -3.333e-01            -5.000e-01  0.000e+00
-1.667e-01  6.667e-01 -3.333e-01 -1.667e-01
-8.333e-02 -3.333e-01  2.833e+00 -3.333e-01 -8.333e-02 -3.333e-01 -3.333e-01 -3.333e-01            -1.667e-01 -1.667e-01 -6.667e-01  0.000e+00  0.000e+00
-3.333e-01 -1.667e-01 -3.333e-01  1.333e+00                       -3.333e-01 -1.667e-01
-8.333e-02            -8.333e-02             1.500e+00 -1.667e-01 -3.333e-01                                  -3.333e-01 -5.000e-01             0.000e+00
                      -3.333e-01            -1.667e-01  6.667e-01 -1.667e-01
                      -3.333e-01 -3.333e-01 -3.333e-01 -1.667e-01  1.333e+00 -1.667e-01
                      -3.333e-01 -1.667e-01                       -1.667e-01  6.667e-01
                                                                                         6.667e-01 -1.667e-01 -1.667e-01 -3.333e-01
-3.333e-01            -1.667e-01                                                        -1.667e-01  1.333e+00 -3.333e-01 -3.333e-01  0.000e+00
                      -1.667e-01            -3.333e-01                                  -1.667e-01 -3.333e-01  1.333e+00 -3.333e-01             0.000e+00
-5.000e-01            -6.667e-01            -5.000e-01                                  -3.333e-01 -3.333e-01 -3.333e-01  2.667e+00  0.000e+00  0.000e+00
 0.000e+00             0.000e+00                                                                    0.000e+00             0.000e+00  1.333e+00  0.000e+00
                       0.000e+00             0.000e+00                                                         0.000e+00  0.000e+00  0.000e+00  1.333e+00
```

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.

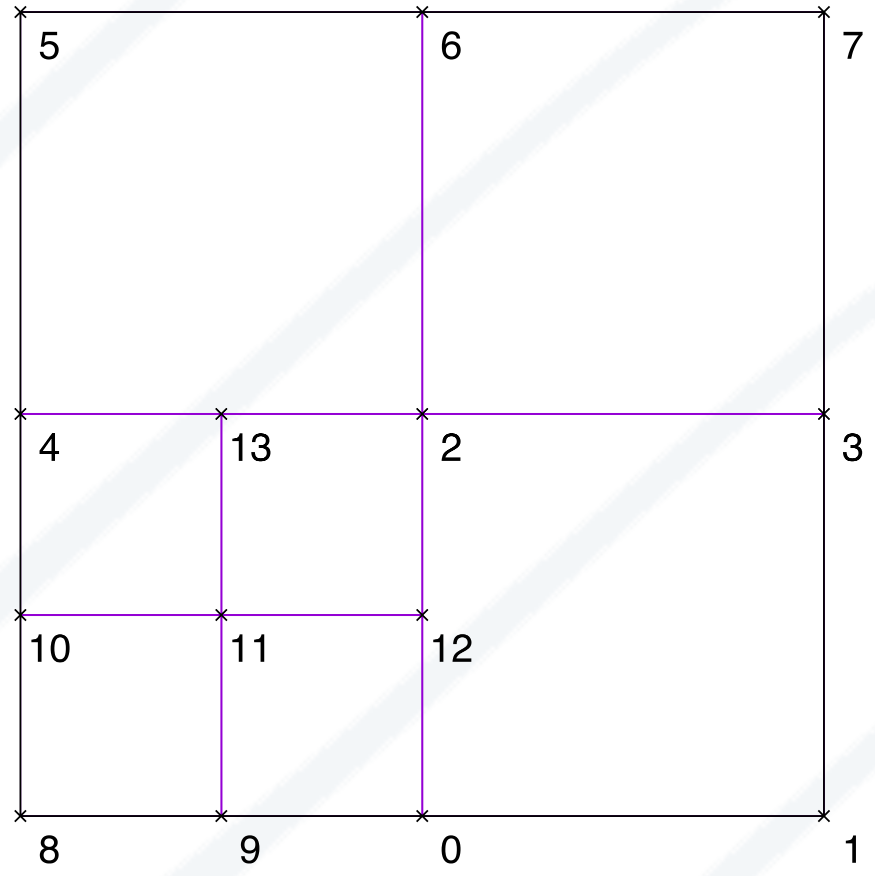
Step 2: Apply constraints during assembly operation (local-to-global) $\mathbf{K}_K, \mathbf{F}_K$

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.

Approach 2 (example):

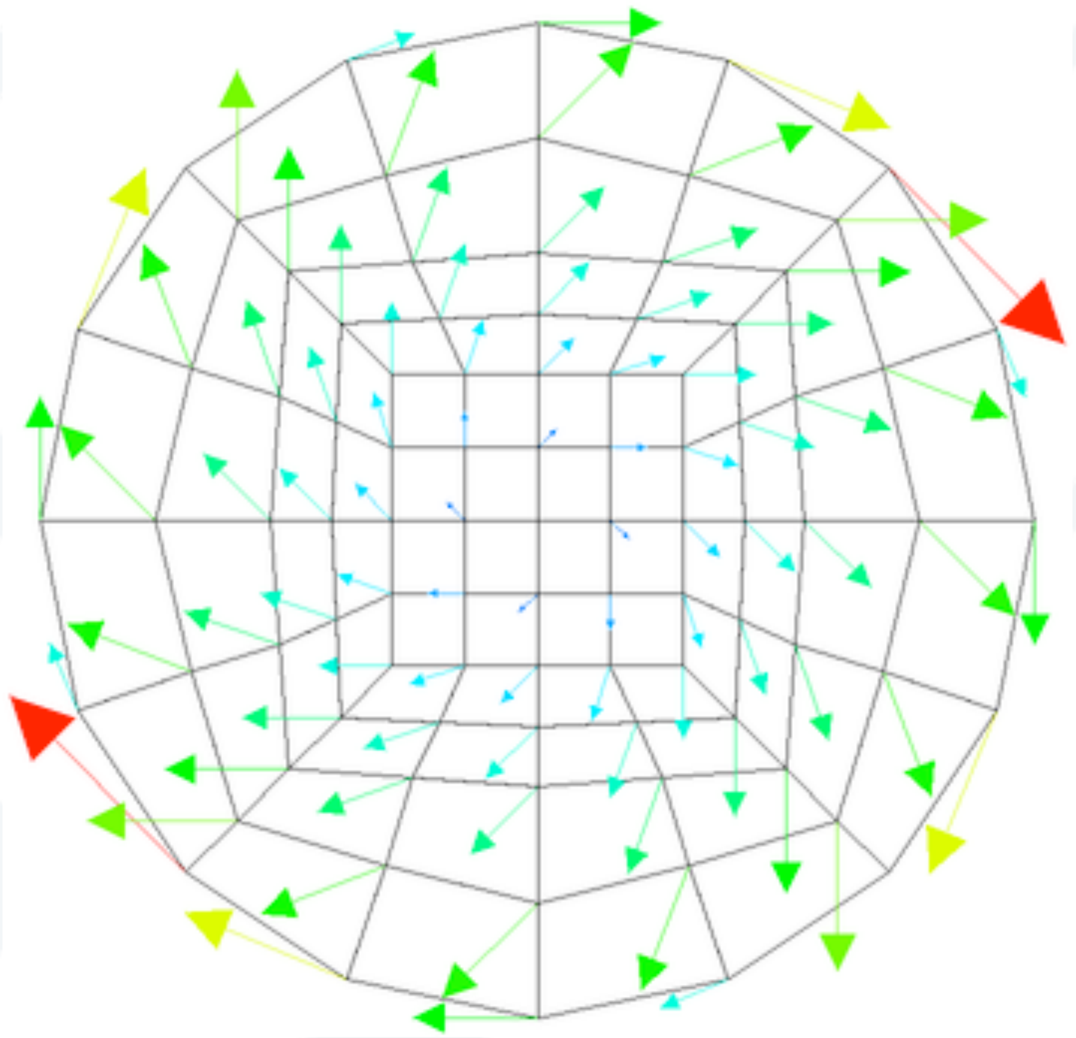
$$\begin{bmatrix} u_{12} \\ u_{13} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} u_0 \\ u_2 \\ u_4 \end{bmatrix}$$




```
===========================================
Number of active cells: 7
Number of degrees of freedom: 14
=============== constraints ===============
    12 0:  0.5
    12 2:  0.5
    13 2:  0.5
    13 4:  0.5
================ condensed ================
================= matrix ==================
 1.500e+00 -1.667e-01 -8.333e-02 -3.333e-01 -8.333e-02                                             -3.333e-01            -5.000e-01  0.000e+00
-1.667e-01  6.667e-01 -3.333e-01 -1.667e-01
-8.333e-02 -3.333e-01  2.833e+00 -3.333e-01 -8.333e-02 -3.333e-01 -3.333e-01 -3.333e-01            -1.667e-01 -1.667e-01 -6.667e-01  0.000e+00  0.000e+00
-3.333e-01 -1.667e-01 -3.333e-01  1.333e+00                       -3.333e-01 -1.667e-01
-8.333e-02            -8.333e-02             1.500e+00 -1.667e-01 -3.333e-01                                  -3.333e-01 -5.000e-01             0.000e+00
                      -3.333e-01            -1.667e-01  6.667e-01 -1.667e-01
                      -3.333e-01 -3.333e-01 -3.333e-01 -1.667e-01  1.333e+00 -1.667e-01
                      -3.333e-01 -1.667e-01                       -1.667e-01  6.667e-01
                                                                                         6.667e-01 -1.667e-01 -1.667e-01 -3.333e-01
-3.333e-01            -1.667e-01                                                        -1.667e-01  1.333e+00 -3.333e-01 -3.333e-01  0.000e+00
                      -1.667e-01            -3.333e-01                                  -1.667e-01 -3.333e-01  1.333e+00 -3.333e-01             0.000e+00
-5.000e-01            -6.667e-01            -5.000e-01                                  -3.333e-01 -3.333e-01 -3.333e-01  2.667e+00  0.000e+00  0.000e+00
 0.000e+00             0.000e+00                                                                   0.000e+00             0.000e+00  1.333e+00  0.000e+00
                       0.000e+00             0.000e+00                                                        0.000e+00  0.000e+00  0.000e+00  1.333e+00
```

Applying constraints: the AffineConstraints class

- This class is used for
  - Hanging nodes
  - Dirichlet and periodic constraints
  - Other constraints
- Linear constraints of the the form $u_C = Cu_O + b$



Applying constraints: the AffineConstraints 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)`
  - Sort, rearrange, optimise constraints
    `constraints.close()`

Applying constraints: the AffineConstraints class

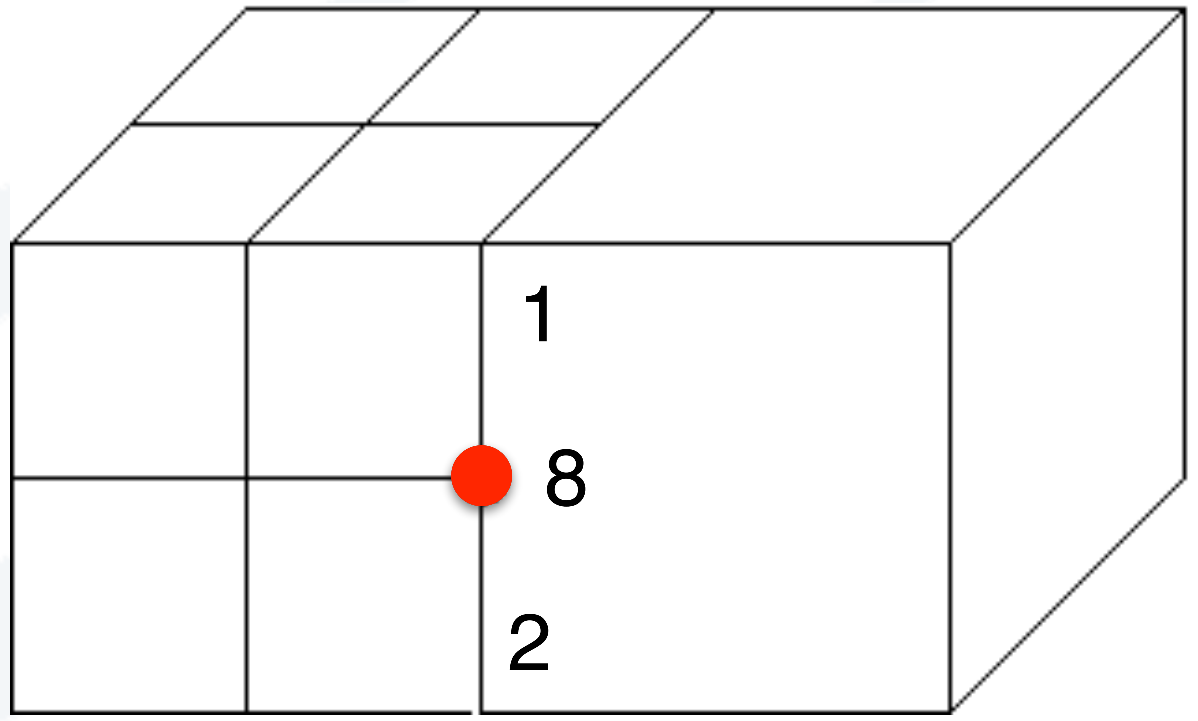
- Assembly
  - Assemble local matrix and vector as normal
  - Eliminate while transferring to global matrix:
```
constraints.distribute_local_to_global (
   cell_matrix, cell_rhs,
   local_dof_indices,
   system_matrix, system_rhs);
```
  - Solve and then set all constraint values correctly:
```
ConstraintMatrix::distribute(...)
```

Applying constraints: Conflicts

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

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

- Which is right? $u_8 = \bar{u}$ or
$$u_8 = \frac{1}{2}\left[u_1 + u_2\right]$$



- Careful on loops: $u_1 = u_2\ ;\ u_2 = u_3\ ;\ u_3 = u_1$