# Finite element methods in scientific computing

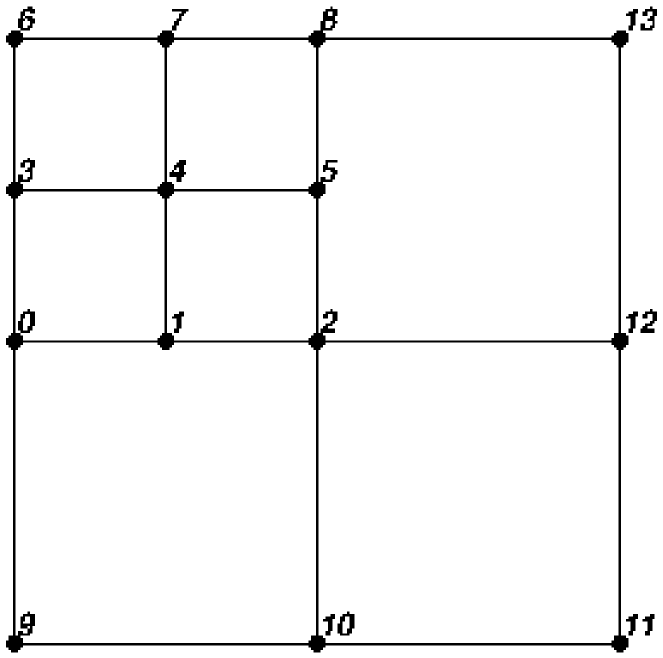
Wolfgang Bangerth, Colorado State University

# Lecture 16:

# Hanging nodes and other constraints

# Why constraints?

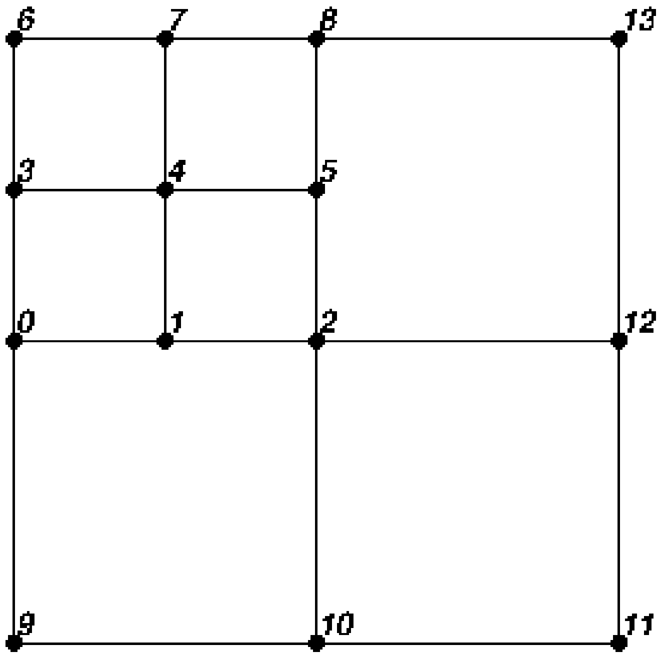
**Consider this mesh, Q1 elements, and DoFs as enumerated:**

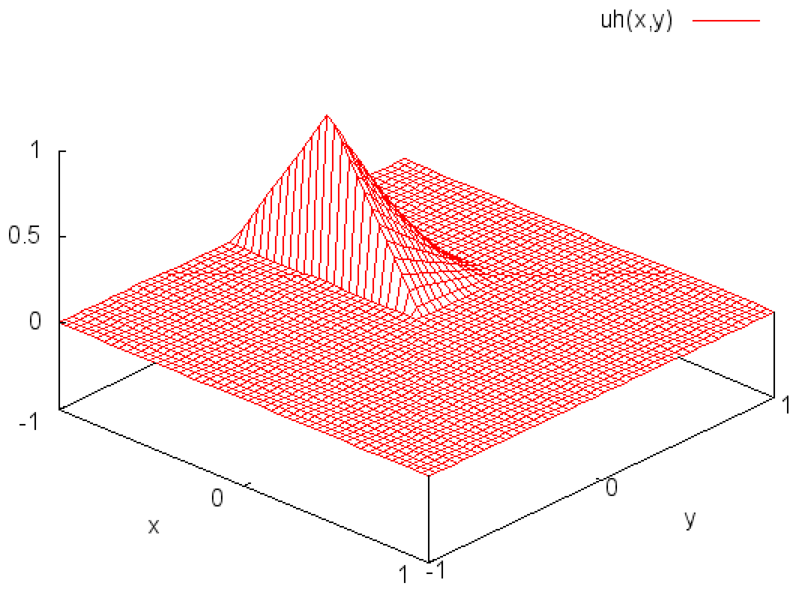



The corresponding space has dimension 14.

# Why constraints?

**Now consider a solution vector *U=(0,1,0,0,...)* and the function $u_h$ associated with it:**




**Note:** This function is not continuous!


*Wolfgang Bangerth*

# Why constraints?

**If our function space $V_h$ has discontinuous functions:**

- It is *not a subspace* of the solution space $H^1$
- A bilinear form such as

$$a(u_h, \varphi_h) = \int_\Omega \nabla u_h \cdot \nabla \varphi_h \, dx$$

  no longer makes immediate sense

**Approach:**

For spaces such as $Q_1$, we really need to *require* continuity! We do so through *constraints*.

# Why constraints?

**Defining $V_h$ via constraints:**

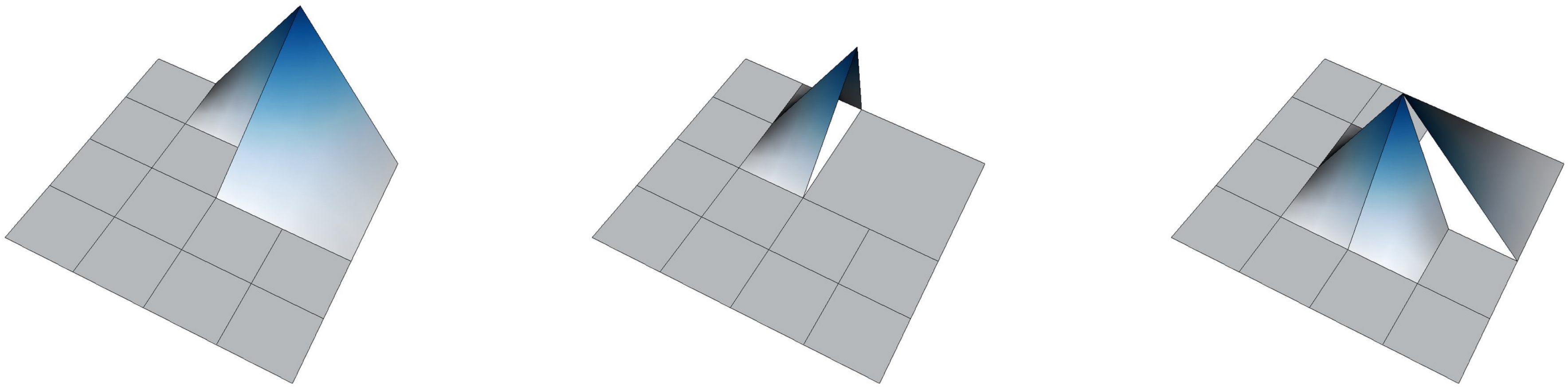
- Shape functions are defined on each cell as usual
- Functions in $V_h$ are linear combinations of shape functions
- Functions in $V_h$ are globally continuous

In other words:

$$V_h = \left\{ v_h(x) = \sum_i V_i \varphi_i(x) : \; v_h(x) \text{ is continuous in } \Omega \right\}$$

# Why constraints?

**How do the shape functions look like:**



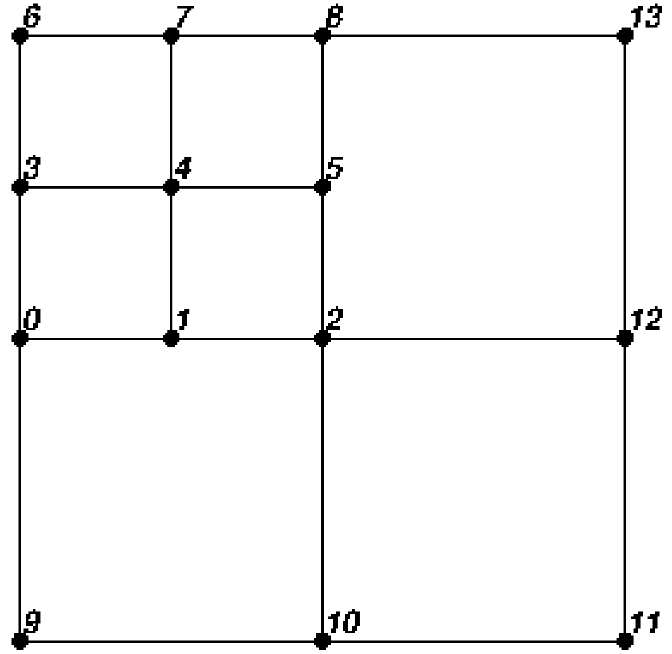
**Note:** Not all of these functions are in $V_h$.

# Which constraints?

**Remember that we define $V_h$ via constraints as:**

$$V_h = \left\{ v_h(x) = \sum_i V_i \varphi_i(x):\ v_h(x) \text{ is continuous in } \Omega \right\}$$

The only possible discontinuities are along edges 0–1–2 and 2–5–8.




The function is in fact continuous if it is continuous at vertices 1 and 5!

That is:

$$V_1 = \frac{1}{2} V_0 + \frac{1}{2} V_2, \qquad V_5 = \frac{1}{2} V_2 + \frac{1}{2} V_8$$

# Which constraints?

**As a general rule:**

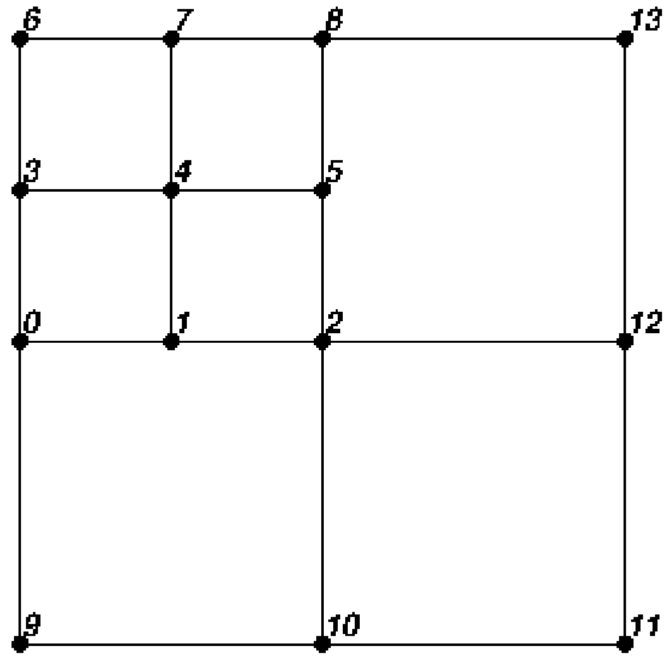
- When using hanging nodes, there is a subset $I$ of *[0,n_dofs)* that is constrained
- These constraints have the form

$$V_i = \sum_{j=0}^{\text{n\_dofs}} \alpha_{ij} V_j \quad \forall i \in I$$

where most of the alphas are zero

- Here, for example:

$$V_1 = \frac{1}{2} V_0 + \frac{1}{2} V_2, \qquad V_5 = \frac{1}{2} V_2 + \frac{1}{2} V_8$$




- We can write this as $CV = 0, \quad C \in \mathbb{R}^{\#\text{ constraints} \times \#\text{ dofs}}$

# Representation in deal.II

**In deal.II:**

- The constraints *CV=0* for hanging nodes are represented by the deal.II class *AffineConstraints*
- *AffineConstraints* objects are built by the function
  *DoFTools::make_hanging_node_constraints()*

**Note:** All of this works for *any* finite element, not just $Q_1$. Furthermore, it also works for the *hp*-refinement case (see step-27).

# Using constraints

**Premise:**

- The beauty of the FEM is that we do *exactly* the same thing on every cell
- Let us not destroy this property!
- That is: assembly on cells with hanging nodes should work exactly as on cells without.

**Note:** The mathematical and algorithmic details of dealing with constraints are complex (see Bangerth & Kayser-Herold, 2009). Therefore, let's discuss only the mechanics.

# Using constraints

Define $\tilde{V}_h = \left\{ v_h(x) = \sum_i V_i \varphi_i(x) \right\}$

$$V_h = \left\{ v_h(x) = \sum_i V_i \varphi_i(x) :\ v_h(x) \text{ is continuous in } \Omega \right\}$$
$$= \left\{ v_h(x) = \sum_i V_i \varphi_i(x) :\ CV = 0 \right\}$$

**Approach 1:**

- Step 1: Build matrix/rhs $\tilde{A}, \tilde{F}$ with all DoFs as if there were no hanging nodes.
- Step 2: Modify $\tilde{A}, \tilde{F}$ to get $A, F$ out of this ("*condense*")
- Step 3: Solve $AU = F$
- Step 4: Get all components of *U* ("*distribute*")

# Using constraints

Define $\tilde{V}_h = \left\{ v_h(x) = \sum_i V_i \varphi_i(x) \right\}$

$$V_h = \left\{ v_h(x) = \sum_i V_i \varphi_i(x) : \; v_h(x) \text{ is continuous in } \Omega \right\}$$
$$= \left\{ v_h(x) = \sum_i V_i \varphi_i(x) : \; CV = 0 \right\}$$

**Approach 2 (step-6):**

- Step 1: Build *local* matrix/rhs $\tilde{A}^K, \tilde{F}^K$ with all DoFs as if there were no hanging nodes.
- Step 2: Modify when copying local contributions into global matrices $A, F$ ("*copy_local_to_global*")
- Step 3: Solve $AU = F$
- Step 4: Get all components of *U* ("*distribute*")

# Finite element methods in scientific computing

Wolfgang Bangerth, Colorado State University