Arbitrary order virtual element method for linear elastostatics

Elias Pescialli

Advanced Programming for Scientific Computing
Prof. Luca Formaggia, Matteo Caldana
Supervisor: Prof. Paola F. Antonietti

A.Y. 2022/2023

POLITECNICO
MILANO 1863

Contents

Outline

Outline

Virtual element method (VEM)

- numerical pde solution method as ”ultimate evolution of the mimetic finite differences approach”
- application to 3D linear elastostatics, through a mixed variational formulation based on three-field Hu-Washizu functional

Algorithms and implementation

- adopted algorithms for geometry handling, integration over polytopes embedded in $\mathbb{R}^1$, $\mathbb{R}^2$ and $\mathbb{R}^3$ and symbolic calculus for monomials
- c++ implementation, from mesh to visualization

Numerical tests

- h-refinement
- p-refinement

Future developments

Virtual element method

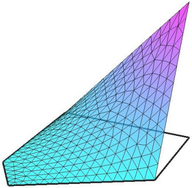
Virtual element method

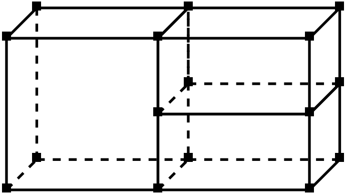
Features

1. addition to usual FE spaces of suitable **non-polynomial** functions as in generalized/extended FEM
2. **avoid explicit computation of shape functions**, hence the name **virtual**.

Highlights

1. supports general, possibly **non-convex elements** even with curved boundaries, handles **hanging nodes**
2. can guarantee **high-order continuity** requirements between elements
3. can be integrated in standard FEM environment





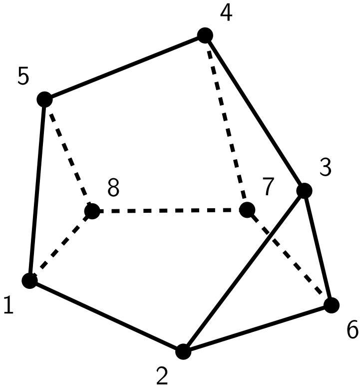
Degrees of freedom

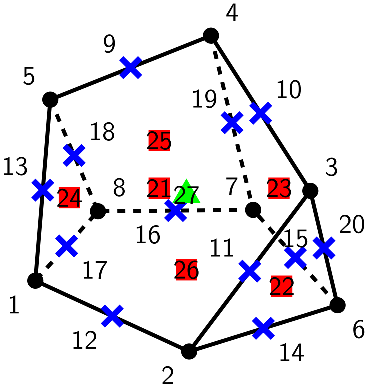
Four types of DOFs:

1. **vertex-type**
2. **edge-type**
3. **face-type**
4. **polyhedron-type**

(1–2) pointwise values $\varphi|_V$

(3–4) internal moments $\int_\Omega \varphi m \, d\Omega$





Projection operator

To construct the **local stiffness matrix** and **equivalent nodal loads** we exploit:

1. the **shape functions are polynomials on the skeleton** of the element (its edges), uniquely determined by pointwise DOFs
2. **internal moments** on the faces and in the interior of the element.

When no information is directly available, a **projection operator** is introduced from the virtual space $V_k(\Omega)$ into the polynomial space $\mathcal{P}_k(\Omega)$.

$$\Pi^{\nabla}_{\Omega,k} : V_k(\Omega) \to \mathcal{P}_k(\Omega)$$

$$\int_{\Omega} \nabla p_k \cdot \nabla \left(\Pi^{\nabla}_{\Omega,k} v - v \right) \, d\Omega = 0 \qquad \forall p_k \in \mathcal{P}_k(\Omega), \quad v \in V_k(\Omega)$$

This operation requires the introduction of a **stabilization part** to account for the non-polynomial functions filtered out by the projection.

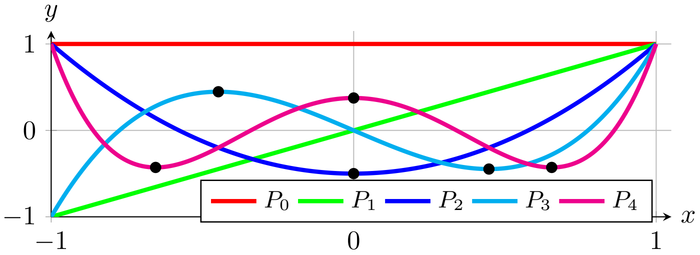
$$\implies \text{local stiffness matrix} = \text{consistent} + \text{stabilizing part}$$

$$\boldsymbol{K}_e = \boldsymbol{K}_e^c + \boldsymbol{K}_e^s$$

Algorithms

Gauss-Lobatto quadrature rule

Pointwise DOFs on each edge of k^{th}-order VEM correspond to **Gauss-Lobatto $(k+1)$-point rule**. The internal $k-1$ GL points over $[-1,1]$ correspond to the stationary points of the k^{th} Legendre polynomial.



Each point X_i and the corresponding weight W_i of the **internal $k-1$ points of GL $(k+1)$-point** rule can be found by solving

$$P_k'(x) = \frac{kxP_k(x) - kP_{k-1}(x)}{x^2-1} = 0 \qquad W_i = \frac{2}{k(k-1)\left[P_{k-1}(X_i)\right]^2}$$

in the intervals generated by the **GL k-point** rule.

Outward normal unit vector and area of polygons

Newell's algorithm allows to compute the outward normal vector $\boldsymbol{n}$ for a general polygon and its area with the coordinates of its vertices $\boldsymbol{V}$ through

$$\tilde{\boldsymbol{n}} = \frac{1}{2} \sum_{i=1}^{N_V} (\boldsymbol{V}_i \times \boldsymbol{V}_{i+1})$$

Normalizing $\tilde{\boldsymbol{n}}$ yields to $\boldsymbol{n}$ while the magnitude $\|\tilde{\boldsymbol{n}}\|$ is the area.

Integration of monomials over polytopic domains

Quadrature-free integration scheme from Antonietti, Houston, Pennesi.

Algorithm Integration of a monomial over a polytopic domain

$\mathcal{I}(N, \mathcal{E}, k_1, \dots, k_d) = \int_{\mathcal{E}} x_1^{k_1} \dots x_d^{k_d} \, d\sigma_N(x_1, \dots, x_d)$

1: **if** $N = 0 \quad \big(\mathcal{E} = (v_1, \dots, v_d) \in \mathbb{R}^d$ is a point$\big)$ **then**

2: **return** $\mathcal{I}(N, \mathcal{E}, k_1, \dots, k_d) = v_1^{k_1} \dots v_d^{k_d}$

3: **else if** $1 \leq N \leq d - 1 \quad$ ($\mathcal{E}$ point if $d = 1$ or edge if $d = 2$ or face if $d = 3$) **then**

4: **return** $\mathcal{I}(N, \mathcal{E}, k_1, \dots, k_d) = \frac{1}{N + \sum_{n=1}^{d} k_n} \big(\sum_{i=1}^{m} d_i \mathcal{I}(N-1, \mathcal{E}_i, k_1, \dots, k_d) +$
$+ x_{0,1} k_1 \mathcal{I}(N, \mathcal{E}, k_1 - 1, \dots, k_d) +$
$+ \dots +$
$+ x_{0,d} k_d \mathcal{I}(N, \mathcal{E}, k_1, \dots, k_d - 1)\big)$

5: **else if** $N = d \quad$ ($\mathcal{E}$ interval if $d = 1$ or polygon if $d = 2$ or polyhedron if $d = 3$) **then**

6: **return** $\mathcal{I}(N, \mathcal{E}, k_1, \dots, k_d) = \frac{1}{N + \sum_{n=1}^{d} k_n} \big(\sum_{i=1}^{m} b_i \mathcal{I}(N-1, \mathcal{E}_i, k_1, \dots, k_d)\big)$

7: **end if**

c++ implementation

General outlook

The main class `Problem` reads the parameters, including the mesh. The code is divided into **7 macro-components**:

- **parameters**, handles the parameters and parsing through **GetPot**
- **geometrical entities**, defines objects `Point`, ..., `Mesh`, reads from **Gmsh**
- **monomials and polynomials**, suite for **symbolic calculus**
- **integration**, Gauss-Lobatto rules, quadrature-free, Gaussian rules
- **virtual space**, **virtual DOFs** and **projections**
- **solver**, assembly with **OpenMP**, boundary conditions, solution
- **visualization of results**, exports for **Paraview**

```cpp
class Problem
{
public:
    // Constructor
    Problem(const char *parametersFilename = "parameters.dat", bool
    printError = false);
};
```

Geometrical entities

Five classes `Point`, `Edge`, `Polygon`, `Polyhedron` and `Mesh` define the corresponding objects through **template programming**.

Highlights:

- **N-dimensional points** with wide range of methods implemented (dot and cross product, transform coordinates,...)
- **automatic numbering** of IDs
- edges stored as **half-edges**
- polygons ordered sequence of half-edges, methods to **check the consistency** of a polygon
- every object contains **references** to the lower-dimensional entity: an edge exists only if the two points already exist
- a `Mesh` object constructed reading a .geo file, easily generated with **Gmsh**

Monomials and polynomials

Monomials are extensively manipulated in the VEM $\implies$ **symbolic calculus tool**.

Templated base class `Monomial`

- supports monomials embedded in **N-dimensional spaces**
- overloaded `operator*`, method to compute the **derivative** and **evaluate** a monomial in a point
- two **derived classes** `Monomial2D` and `Monomial3D` to specialize the general monomial and handle a vector of **ordered monomials**, their **gradients** and their **Laplacians**.

Templated base class `Polynomial`

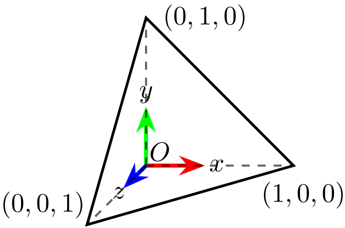
- map of `Monomials`
- derived class `LinearTrinomialPower` handles a polynomial generated by the n^{th} power of a trinomial of the kind $ax + by + c$, useful to pass from polyhedron to face reference frame.

```
template <unsigned int Dimension>
class Monomial
{private:
    std::vector<unsigned int> exponents;
    real coefficient;};
```

Integration

Three **namespaces**:

1. **GaussLobatto**
   - computes points and weights of the GL rule over $[-1, 1]$ storing them in a **cache**
   - exploits `toms748_solve` from `boost`
   - maps the abscissas and weights for an **edge**
2. **IntegrationMonomial**
   - defines the templated function `integrateMonomial`, which allows to integrate a `Monomial2D` or `Monomial3D` over a 2D or 3D domain
   - accepts parameters to perform **change of coordinates**
   - methods to compute **centroid** and **volume**
3. **Gauss**
   - points and weights of **Gauss quadrature** over the standard tetrahedron
   - **subtetrahedralization** for star-shaped polyhedra w.r.t. centroid, all convex faces



Virtual space

Two parts:

1. **virtual degrees of freedom**
   - **Polymorphism**: base virtual class `VirtualDof`, four derived classes `VertexDof`, `EdgeDof`, `FaceDof`, `PolyhedronDof`
   - class `VirtualDofsCollection` gathers all DOFs through **shared pointers**
   - class `LocalVirtualDof` maps global to local DOFs and vice-versa
2. **virtual projections**
   - computes **face projections**
   - computes **polyhedra projections**, local matrices
   - relies on **Eigen**, exploits `SparseMatrix` and upper part storage

```
class VirtualFaceProjections
{private:
    map<size_t, vector<Polynomial<2>>> faceProjections;};
class VirtualPolyhedronProjections
{private:
    map<size_t, SparseMatrix<real>> polyhedronProjections; // C
    map<size_t, SparseMatrix<real>> elastic_matrices; // E, upper part
    map<size_t, SparseMatrix<real>> deformation_RBM_matrices; // Tdr
    map<size_t, VectorXd> forcingProjections; // F
};
```

Solver

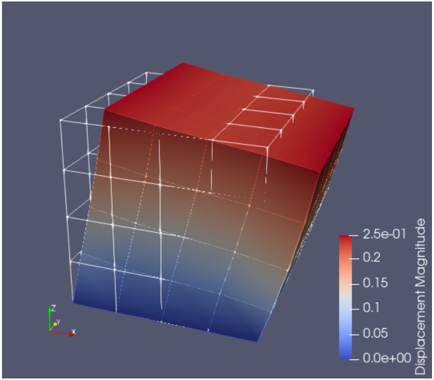
Handles **assembly** of local matrices into global system, exploits parallelization through **OpenMP**, enforces **boundary conditions** and **solves** the linear algebraic system.

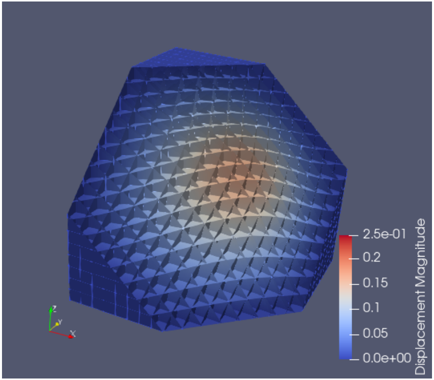
- class `Solver` exploits `Eigen::ConjugateGradient`, easily adaptable for **other problems**, e.g. extend to elastodynamics, iterative solution for nonlinear constitutive laws
- derived class `SolverVEM` assembles and enforces homogeneous Dirichlet BC, computes the **strain L^2-norm of the error**, adopted for convergence tests

$$\|e_{\boldsymbol{\varepsilon}}\|_{L^2} = \sqrt{\sum_{P \in \mathcal{P}} \int_P \|\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^h\|^2 \, d\Omega}$$

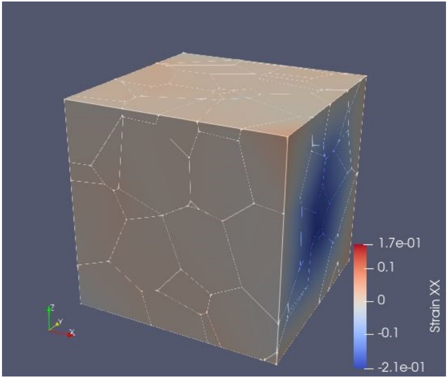
Visualization of results - 1

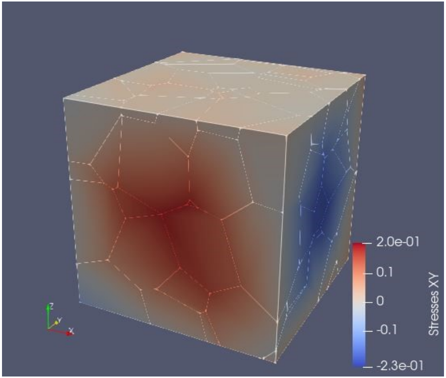
Displacements, **strains** and **stresses** in nodal values can be exported in `.vtk` format, to be read with, e.g., **Paraview**. Visualization of **deformed** and **undeformed** configuration is possible with the filter **Warp By Vector** and the interior can be viewed with **Slice** or **Cut**.





Visualization of results - 2





Numerical tests

Data of the problem

$$\begin{cases} -\nabla \cdot [\boldsymbol{D}\boldsymbol{\varepsilon}(\boldsymbol{u})] = \boldsymbol{f} & \text{in } \Omega = (0,1)^3 \\ \boldsymbol{u} = \boldsymbol{0} & \text{on } \partial\Omega \end{cases}$$

$$\boldsymbol{\varepsilon}(\boldsymbol{u}) = \frac{\nabla \boldsymbol{u} + \nabla^{\mathrm{T}} \boldsymbol{u}}{2} \qquad D_{ijkl} = \lambda \delta_{ij}\delta_{kl} + \mu \left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{kj}\right)$$

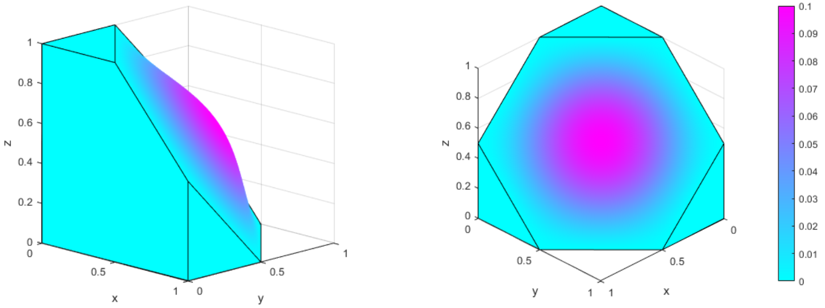
$$\boldsymbol{f}(x,y,z) = C \begin{Bmatrix} -\pi^2 \left[(\lambda+\mu)\cos(\pi x)\sin(\pi y + \pi z) - (\lambda + 4\mu)\sin(\pi x)\sin(\pi y)\sin(\pi z)\right] \\ -\pi^2 \left[(\lambda+\mu)\cos(\pi y)\sin(\pi x + \pi z) - (\lambda + 4\mu)\sin(\pi x)\sin(\pi y)\sin(\pi z)\right] \\ -\pi^2 \left[(\lambda+\mu)\cos(\pi z)\sin(\pi x + \pi y) - (\lambda + 4\mu)\sin(\pi x)\sin(\pi y)\sin(\pi z)\right] \end{Bmatrix}$$

$$\lambda = 1 \qquad \mu = 1 \qquad C = 0.1$$

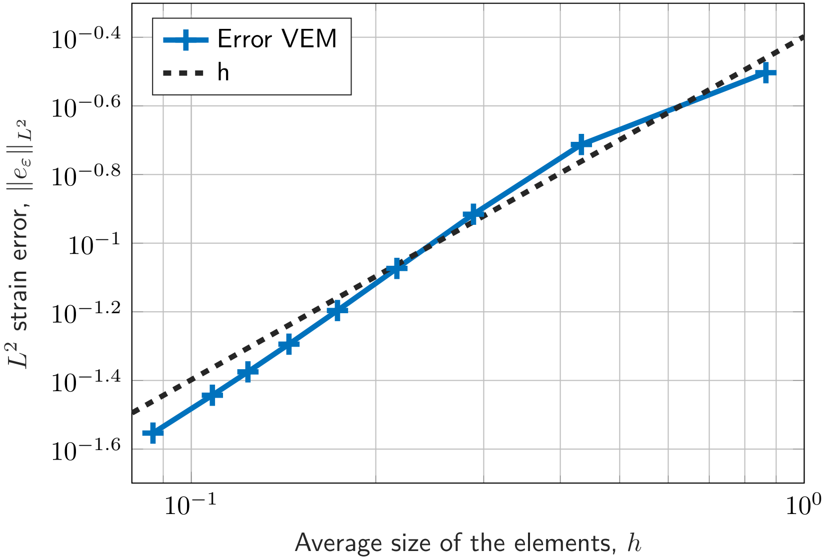
Units of measurements are not specified, but they are taken in a consistent way (e.g. N/mm^3 for body forces; N/mm^2 for surface tractions, stresses, Young's modulus and Lamé constants; mm for lengths)

Exact solution

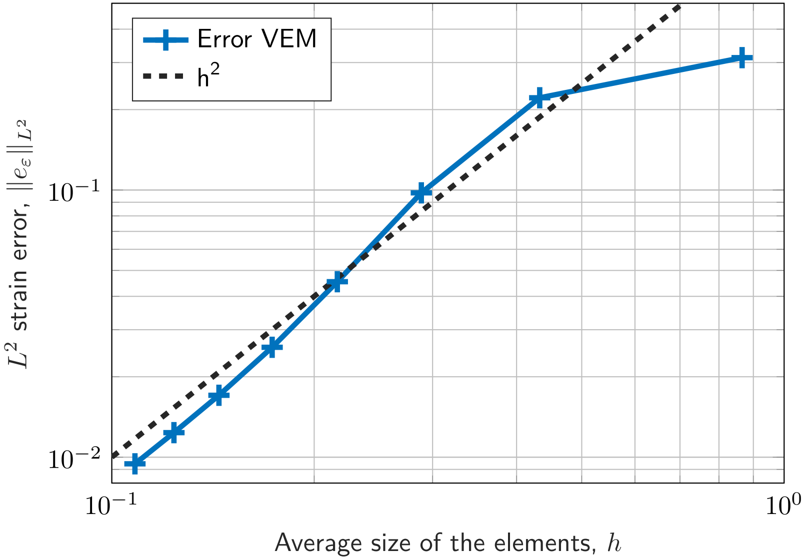
$$\boldsymbol{u}_{ex}(x,y,z) = C\sin(\pi x)\sin(\pi y)\sin(\pi z)\begin{Bmatrix}1\\1\\1\end{Bmatrix}$$



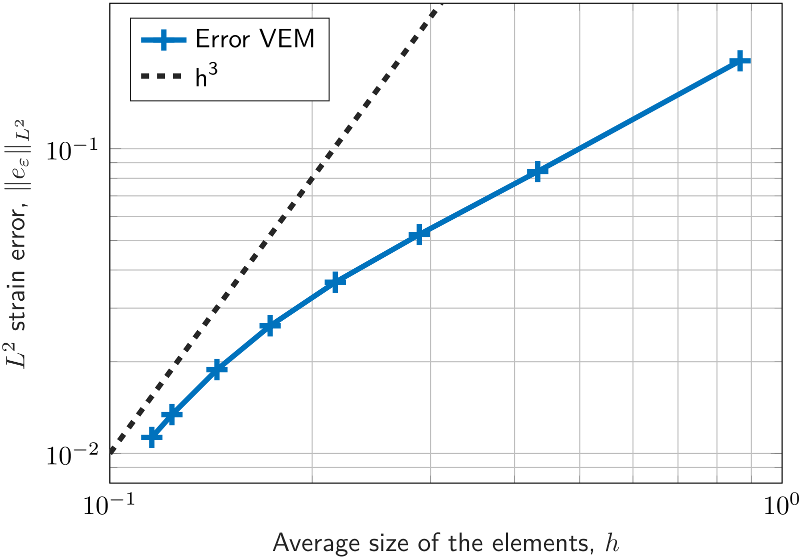
h-refinement test - $k = 1$



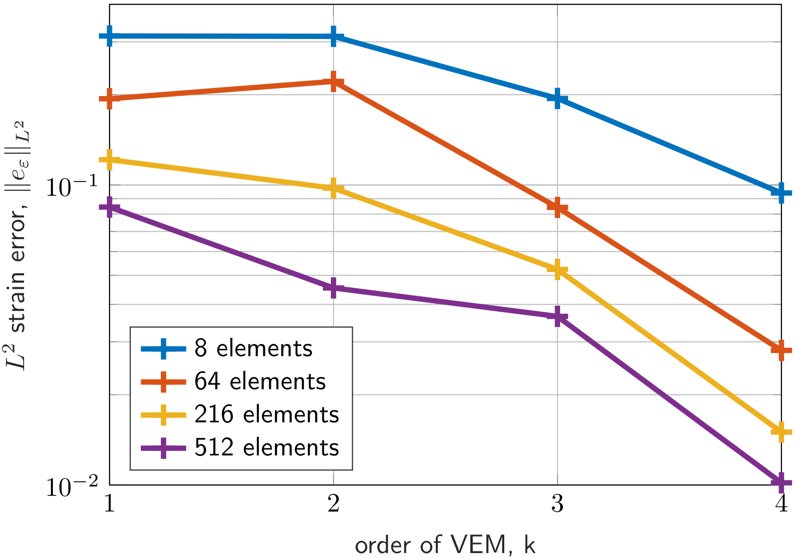
h-refinement test - $k = 2$



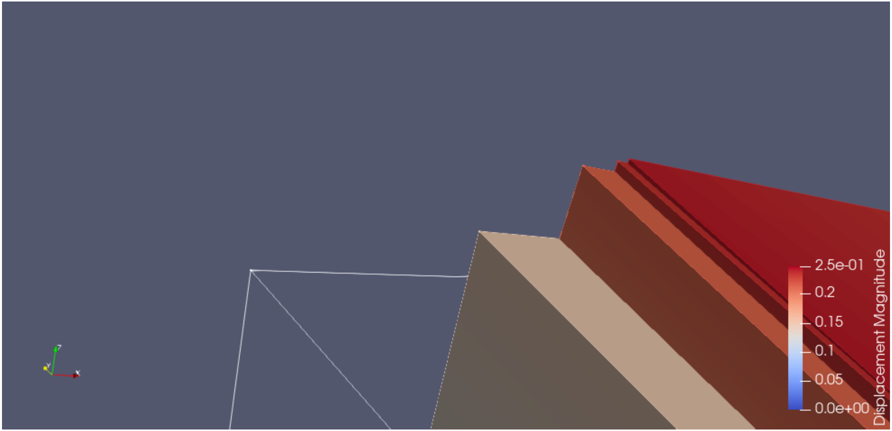
h-refinement test - $k = 3$



p-refinement



p-refinement - another problem visualized



Magnitude of the displacement vector field displayed on the deformed body through a 8-element, 1^{st} to 4^{th} order 8-element VEM, constant body force field $f_x = 0.2$, $\lambda = 1$, $\mu = 1$, homogeneous Dirichlet boundary conditions applied on the plane $z = 0$.

Future developments

Future developments

- extend to account for **dynamic effects** or **material nonlinearities**
- exploit modularity of the present code, adapt the solver to handle **iterative methods**
- possible improvement for **integration of general functions over general polyhedra** needed in rhs computation
- adopt graph partitioning techniques (e.g. **Metis**) to **agglomerate** elements in tetrahedral meshes generated in complex geometries to further test the effectiveness of the VEM.

Thank you!