Homework Set #5

assigned: Friday, November 11th, 2016
due: Wednesday, November 23rd, 2016, 3 pm
drop boxes outside 374 Firestone

Review: Simplicial Elements

We showed in class that a simplicial element in d dimensions has $n = d + 1$ nodes and

$$N^1(r_1, \dots, r_d) = r_1, \qquad N^2(r_1, \dots, r_d) = r_2, \qquad \dots \qquad N^n(r_1, \dots, r_d) = 1 - \sum_i^d r_i,$$

where $\boldsymbol{\xi} = \{r_1, \dots, r_d\}$ denote the d *barycentric coordinates*. Then, the Jacobian $\boldsymbol{J}$ has components

$$J_{ij} = \frac{\partial X_j}{\partial r_i} = \sum_{a=1}^{n} X_j^a \frac{\partial N^a}{\partial r_i} \qquad \text{and} \qquad J = \det \boldsymbol{J}.$$

The shape function derivatives in the *physical coordinate system* thus follow as

$$\begin{pmatrix} N^a_{,X_1} \\ \dots \\ N^a_{,X_d} \end{pmatrix} = \boldsymbol{J}^{-1} \begin{pmatrix} N^a_{,r_1} \\ \dots \\ N^a_{,r_d} \end{pmatrix} \qquad \text{or} \qquad \nabla_{\boldsymbol{X}} N^a = \boldsymbol{J}^{-1} \nabla_{\boldsymbol{\xi}} N^a.$$

Using numerical quadrature with weights w_k and points $\boldsymbol{\xi}_k$ (in the reference system), the element energy is then

$$I_e \approx \sum_{k=1}^{n_{QP}} w_k \, W\left(\boldsymbol{F}(\boldsymbol{\xi}_k)\right) J(\boldsymbol{\xi}_k) \, t_e.$$

Nodal forces were obtained as

$$F^a_{\text{int},i} \approx \sum_{k=1}^{n_{QP}} w_k \, P_{iJ}\left(\boldsymbol{F}(\boldsymbol{\xi}_k)\right) N^a_{,J}(\boldsymbol{\xi}_k) J(\boldsymbol{\xi}_k) \, t_e \qquad \text{and} \qquad (\nabla_{\boldsymbol{X}} N^a)_J = (\boldsymbol{J}^{-1} \nabla_{\boldsymbol{\xi}} N^a)_J.$$

Finally, the element stiffness matrix components were derived as

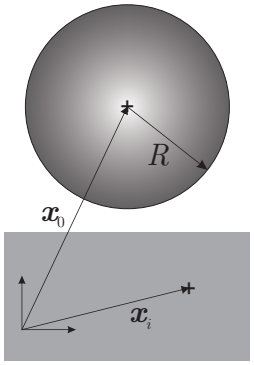
$$T^{ab}_{il} \approx \sum_{k=1}^{n_{QP}} w_k \, \mathbb{C}_{iJlL}\left(\boldsymbol{F}(\boldsymbol{\xi}_k)\right) N^a_{,J}(\boldsymbol{\xi}_k) N^b_{,L}(\boldsymbol{\xi}_k) J(\boldsymbol{\xi}_k) \, t_e.$$

Here, t_e is an element constant (e.g., the cross-sectional area A in 1D, the thickness t in 2D, and simply 1 in 3D). The deformation gradient, $\boldsymbol{F} = \boldsymbol{I} + \nabla_{\boldsymbol{X}} \boldsymbol{u}$, and strain tensor, $\boldsymbol{\varepsilon} = \text{sym}(\nabla_{\boldsymbol{X}} \boldsymbol{u})$, can be obtained directly from $\nabla_{\boldsymbol{X}} \boldsymbol{u}$. Also, recall that

$$\nabla_{\boldsymbol{X}} \boldsymbol{u}(\boldsymbol{\xi}) = \sum_{a=1}^{n} \boldsymbol{u}^a_e \otimes \nabla_{\boldsymbol{X}} N^a(\boldsymbol{\xi}).$$

In linearized kinematics, the equations are analogous with P_{iJ} becoming the Cauchy stress tensor.

Theory: Indenter



To apply external forces, let us use a trick and implement external like internal forces, i.e., via element classes. For example, a spherical indenter with radius R whose center is at a known point $\boldsymbol{x}_0$ can act on a node at *deformed* location $\boldsymbol{x}_i$ via the potential energy

$$I_e^{\text{ext}} = C\,(R - \|\boldsymbol{x}_i - \boldsymbol{x}_0\|)^3\, H(R - \|\boldsymbol{x}_i - \boldsymbol{x}_0\|),$$

where $C > 0$ is a force constant, and $H(\cdot)$ the Heavyside step function. The nodal force and stiffness matrix thus follow as

$$\boldsymbol{F}^a = \partial I_e/\partial \boldsymbol{u}^a, \qquad K_{ij}^{ab} = \partial F_i^a/\partial u_j^b.$$

As we do not know which nodes are in contact with the indenter, we can simply add the above potential to *all* nodes (it only becomes effective if the node is inside the indenter radius due to the Heaviside function).

(Coding) Problem 1: d-dimensional simplicial element type (15 points).

Let us introduce a class `ElementType::Simplex`, which has the following:

- defines element specifics such as `NumberOfNodes`, `SpatialDimensions`, `Point` , etc.
- a method `computeShapeFunctions`which for any point $\boldsymbol{\xi}$ inside the element (in its *reference* coordinate system) returns $\{N^1(\boldsymbol{\xi}), \dots, N^n(\boldsymbol{\xi})\}$.
- a method `computeShapeFunctionDerivatives`which for any point $\boldsymbol{\xi}$ inside the element (in its *reference* coordinate system) returns $\{\nabla_{\boldsymbol{\xi}} N^1(\boldsymbol{\xi}), \dots, \nabla_{\boldsymbol{\xi}} N^n(\boldsymbol{\xi})\}$.

Let us make the `ElementType::Simplex` class as general as possible, so that it works in arbitrary dimensions, by templating the class based on the `Dimension`.

(Coding) Problem 2: d-dimensional simplicial element (35 points).

Let us write a class `Elements::IsoparametricElement`, which uses the above `ElementType` to compute all quantities needed for an element (using the same conventions as in previous homework sets):

- a *constructor*, which receives the element *nodes*, some element *properties*, an *element type*, a *quadrature rule*, and a *material model*. The constructor should use all of those to compute and store the shape function derivatives at the quadrature points.

  Hint: Please have a look at the source file to see how to use the *quadrature rule object*. It simply provides a list of quadrature point locations $\boldsymbol{\xi}_k$ and associated weights w_k.
- a method `computeDispGradsAtGaussPoints`, which computes $\nabla \boldsymbol{u}$ at each quadrature point.
- a method `computeEnergy`, which receives the nodal displacements $\{\boldsymbol{u}_e^1, \dots, \boldsymbol{u}_e^n\}$ and computes the element energy I_e.
- a method `computeForces`, which receives the nodal displacements $\{\boldsymbol{u}_e^1, \dots, \boldsymbol{u}_e^n\}$ and computes the nodal forces $\{\boldsymbol{F}_{\text{int}}^1, \dots, \boldsymbol{F}_{\text{int}}^n\}$.

- a method `computeStiffnessMatrix`, which receives the nodal displacements $\{\boldsymbol{u}_e^1, \dots, \boldsymbol{u}_e^n\}$ and computes the element stiffness matrix $\boldsymbol{T}$ in our common notation (same as homework set #4).
- methods `computeStressesAtGaussPoints` and `computeStrainsAtGaussPoints` for postprocessing.
- a few more convenience methods that we have already completed for you.

Test your element by using `testElementDerivatives` as before.

(Coding) Problem 3: d-dimensional external force indenter (15 points).

Let us implement a spherical/circular indenter to perform indentation simulations, as explained above. We implement the class `ExternalForce::Indenter` in close analogy to the above element class. In fact, it also needs methods `computeEnergy`, `computeForces` and `computeStiffnessMatrix` .

Test your element by using `testElementDerivatives`, as shown in `Main`.

(Coding) Problem 4: boundary value problems (35 points).

Let us use the above classes to solve two boundary value problems (BVPs), one in 2D and one in 3D. The code provides convenience functions to construct rectangular block meshes of triangles or tetrahedra with arbitrary side lengths and numbers of nodes per edge (*please be considerate when choosing element sizes and do not break* `Davinci`!). For both BVPs, we will re-use the NewtonRaphson solver from homework set #4.

In **2D**, let us use the d-dimensional *linear elastic material model* implemented by some of you on set #2. In **3D**, let us use your implementation of the Neo-Hookean solid from set #3. Our versions for both classes are available in case you prefer to not use your own implementation.

(a) Let us simulate a 2D rectangular block made of a linear elastic material that is indented from the top by a circular indenter in small incremental steps, while the bottom edge is fixed rigidly.

(b) Let us simulate a 3D cuboid made of a Neo-Hookean material that is indented from the top by a spherical indenter in small incremental steps, while the bottom edge is fixed rigidly.

total: 100 points