Project #4

assigned: Thursday, November 9th, 2017
due: Thursday, November 23rd, 2017, 17:00
please hand in to one of the TAs in LEE N203

Review: Simplicial Elements

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

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

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

$$\begin{pmatrix} N^a_{e,x_1} \\ \ldots \\ N^a_{e,x_d} \end{pmatrix} = \boldsymbol{J}^{-1} \begin{pmatrix} N^a_{e,r_1} \\ \ldots \\ N^a_{e,r_d} \end{pmatrix} \qquad \text{or} \qquad \nabla_{\boldsymbol{x}} N_e^a = \boldsymbol{J}^{-1} \nabla_{\boldsymbol{\xi}} N_e^a.$$

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

$$I_e \approx \sum_{k=0}^{n_{QP}-1} W\left(\nabla \boldsymbol{u}^h(\boldsymbol{\xi}_k)\right) w_k \qquad \text{with} \qquad w_k = W_k J(\boldsymbol{\xi}_k)\, t_e$$

Nodal forces are obtained as (using the shape function derivatives from above)

$$F^a_{\mathsf{int},i} \approx \sum_{k=0}^{n_{QP}-1} \sigma_{ij} \left(\nabla \boldsymbol{u}^h(\boldsymbol{\xi}_k)\right) N^a_{,j}(\boldsymbol{\xi}_k)\, w_k.$$

Finally, the element stiffness matrix components are

$$T^{ab}_{in} \approx \sum_{k=0}^{n_{QP}-1} \mathbb{C}_{ijnl} \left(\nabla \boldsymbol{u}^h(\boldsymbol{\xi}_k)\right) N^a_{,j}(\boldsymbol{\xi}_k) N^b_{,l}(\boldsymbol{\xi}_k)\, w_k.$$

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). When needed, the deformation gradient, $\boldsymbol{F} = \boldsymbol{I} + \nabla_{\boldsymbol{x}} \boldsymbol{u}$, and strain tensor, $\boldsymbol{\varepsilon} = \mathrm{sym}(\nabla_{\boldsymbol{x}} \boldsymbol{u})$, can be obtained directly from the displacement gradient (again using the above shape functions)

$$\nabla_{\boldsymbol{x}} \boldsymbol{u}(\boldsymbol{\xi}) = \sum_{a=1}^{n} \boldsymbol{u}^a \otimes \nabla_{\boldsymbol{x}} N_e^a(\boldsymbol{\xi}).$$

In linearized kinematics, the equations are analogous with σ_{ij} being replaced by P_{iJ}.

(Coding) Problem 1: d-dimensional simplicial element type (35 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 $\{N_e^1(\boldsymbol{\xi}), \ldots, N_e^n(\boldsymbol{\xi})\}$.
- a method `computeShapeFunctionDerivatives` which for any point $\boldsymbol{\xi}$ inside the element (in its *reference* coordinate system) returns $\{\nabla_{\boldsymbol{\xi}} N_e^1(\boldsymbol{\xi}), \ldots, \nabla_{\boldsymbol{\xi}} N_e^n(\boldsymbol{\xi})\}$.

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 `SpatialDimension` (so it will work in 1D, 2D and 3D).

(Coding) Problem 2: d-dimensional simplicial element (60 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 projects):

- a *constructor*, which receives the element *nodes*, some element *properties* (such as thickness t), 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 and w_k for each quadrature point.

  Hint: We provide a *quadrature rule object* that has pre-computed quadrature point locations $\boldsymbol{\xi}_k$ and associated weights W_k. Also, even though shape functions are not required in the following (only their derivatives), we compute them here for future projects (they will be needed in dynamics later).
- a method `computeDispGradsAtGaussPoints`, which receives the the nodal displacements $\{\boldsymbol{u}_e^1, \ldots, \boldsymbol{u}_e^n\}$ and computes $\nabla_{\boldsymbol{x}} \boldsymbol{u}(\boldsymbol{\xi}_k)$ at each quadrature point $\boldsymbol{\xi}_k$.
- a method `computeEnergy`, which receives the nodal displacements $\{\boldsymbol{u}_e^1, \ldots, \boldsymbol{u}_e^n\}$ and computes the element energy I_e.
- a method `computeForces`, which receives the nodal displacements $\{\boldsymbol{u}_e^1, \ldots, \boldsymbol{u}_e^n\}$ and computes the nodal forces $\{\boldsymbol{F}_{\text{int}}^1, \ldots, \boldsymbol{F}_{\text{int}}^n\}$.
- a method `computeStiffnessMatrix`, which receives the nodal displacements $\{\boldsymbol{u}_e^1, \ldots, \boldsymbol{u}_e^n\}$ and computes the element stiffness matrix $\boldsymbol{T}$ in our common notation (using the same conversion utilities from project #4).
- methods `computeStressesAtGaussPoints` and `computeStrainsAtGaussPoints` for postprocessing.
- a few more convenience methods that we have already completed for you.

(Coding) Problem 3: element test (5 points).

Show that your element is implemented correctly by using the provided utility function `testElementDerivatives` which works completely analogously to the material model test utility that you wrote for Project #2 (i.e., this function checks if the forces are indeed the derivative of the energy, etc.). Test your element both in 2D and in 3D using the Neo-Hookean material model (the version we provide works both in 2D and 3D).

total: 100 points