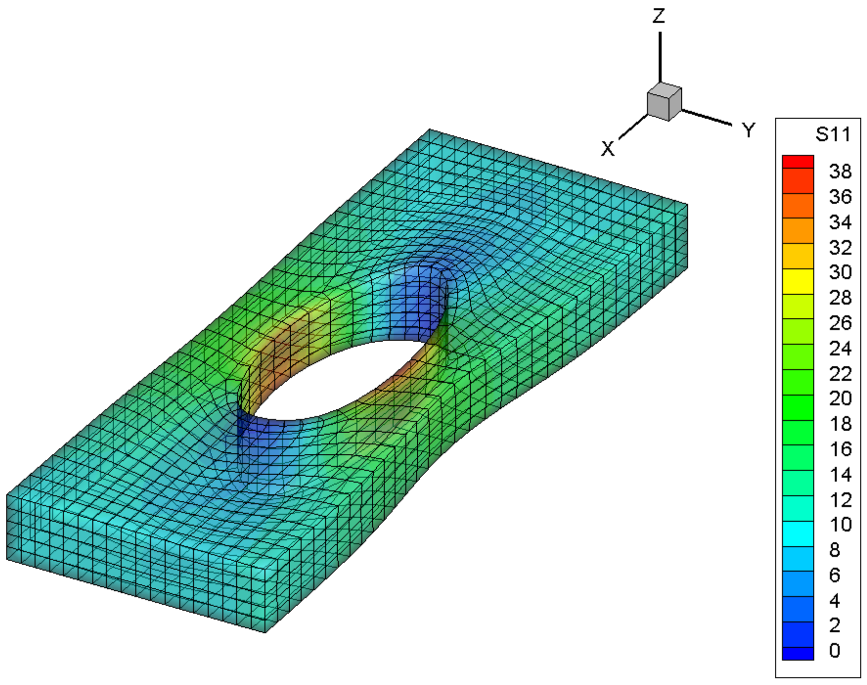
EN234: Computational methods in Structural and Solid Mechanics

Homework 7: Large deformation elasticity
Due Wed Nov4, 2015

School of Engineering
Brown University



HEALTH WARNING: Unfortunately FEA and solid mechanics use **B** to denote two different quantities – the Left Cauchy Green deformation tensor, and the **B** matrix that maps displacements to strains. We use the standard notation in this homework, so be careful to distinguish between the two!

In this homework you will extend EN234FEA to solve static boundary value problems for hyperelastic materials. As a simple nonlinear elastic material we will consider the neo-Hookean material, with stress-strain relation

$$\sigma_{ij} = \frac{\mu_1}{J^{5/3}}\left(B_{ij} - \frac{1}{3}B_{kk}\delta_{ij} \right) + K_1\left(J-1\right)\delta_{ij}$$

where σ_{ij} denotes the Cauchy stress, J=det(**F**), and $B_{ij} = F_{ik}F_{jk}$. This model is intended to be used with $K_1 >> \mu_1$.

As last week you can choose whether to do a straightforward version of this problem, or a more challenging (but more useful) implementation.

1. Simple problem – basic hyperelasticity

A straightforward implementation of finite strain elasticity was discussed in class. Recall that the equilibrium equation reduces to a set of nonlinear equations for the displacements of the form

$$R_i^a - F_i^a = 0 \qquad F_i^a = \int_{V_0} \rho_0 b_i N^a dV_0 + \int_{\partial_2 V_0} t_i^* N^a dA_0 \qquad R_i^a = \int_{V_0} \frac{\partial N^a}{\partial y_j} \tau_{ij}[F_{kl}] dV_0$$

where τ_{ij} is the Kirchhoff stress, and

$$\frac{\partial N^a}{\partial y_j} = \frac{\partial N^a}{\partial x_k} F_{kj}^{-1}$$

are the spatial shape function derivatives.

The nonlinear equations must be solved using Newton-Raphson iteration: as always this involves repeatedly computing a correction to the approximation to the displacement field by solving a set of linear equations

$$K_{aibk} dw_k^b = -R_i^a + F_i^a$$

The following expression was derived in class for the tangent stiffness matrix

$$K_{aibk}^{el} = \int_{V_0} \frac{\partial N^a}{\partial y_j} \frac{\partial \tau_{ij}}{\partial B_{ln}} \frac{\partial B_{ln}}{\partial F_{ks}} F_{rs} \frac{\partial N^b}{\partial y_r} dV_0 + \int_{V_0} \frac{\partial N^a}{\partial y_k} \tau_{ij}[\bar{F}_{kl}] \frac{\partial N^b}{\partial y_j} dV_0$$

Here,

$$\frac{\partial B_{ln}}{\partial F_{ks}} F_{rs} = \left(\delta_{nk} B_{lr} + \delta_{lk} B_{nr} \right)$$

$$\frac{\partial \tau_{ij}}{\partial B_{kn}} = \frac{\mu_1}{J^{2/3}} \left(\frac{\delta_{ik}\delta_{jn} + \delta_{jk}\delta_{in}}{2} - \frac{1}{3}\delta_{kn}\delta_{ij} \right) - \frac{1}{3}\frac{\mu_1}{J^{2/3}} \left(B_{ij} - \frac{1}{3} B_{nn}\delta_{ij} \right) B_{kn}^{-1} B_{nl} + K_1 (J^2 - J/2) \delta_{ij} B_{kn}^{-1} B_{nl}$$

The relevant products can be expanded as matrix operations

$$\mathbf{R} = \int_{V_0} \mathbf{B}^T \boldsymbol{\sigma} dV_0$$

where $\boldsymbol{\sigma}$ now stores the Kirchoff stress, and the matrix $\mathbf{B}$ is the usual one, except (eg) that $\frac{\partial N^b}{\partial x_i}$ has been replaced by $\frac{\partial N^b}{\partial y_i} = \frac{\partial N^b}{\partial x_k} \bar{F}_{ki}^{-1}$.

$$\mathbf{K} = \int_{V_0} \mathbf{B}^T \mathbf{D} \mathbf{G} \mathbf{B}^* dV_0 - \int_{V_0} \boldsymbol{\Sigma} dV_0$$

Here, **B** is the usual matrix (but with derivatives wrt **y**); the Tangent Stiffness **D** represents

$$\frac{\partial \tau_{ij}}{\partial B_{kl}} = \frac{\mu_1}{J^{2/3}} \left(\frac{\delta_{ik}\delta_{jl} + \delta_{jk}\delta_{il}}{2} - \frac{1}{3}\delta_{kl}\delta_{ij} \right) - \frac{1}{3}\frac{\mu_1}{J^{2/3}} \left(B_{ij} - \frac{1}{3} B_{nn}\delta_{ij} \right) B_{kl}^{-1} + K_1 (J^2 - \frac{1}{2} J) \delta_{ij} B_{kl}^{-1}$$

Here $B_{ij} = F_{ik} F_{jk}$. We can store the identity matrix, B_{ij}, B_{ij}^{-1} as 1-D vectors, eg

$\underline{I} = [1,1,1,0,0,0] \qquad \underline{B} = [B_{11}, B_{22}, B_{33}, B_{12}, B_{13}, B_{23}]$, in which case

$$\mathbf{D} = \frac{\mu_1}{J^{2/3}} \begin{bmatrix} 1 & & & & & 0 \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1/2 & & \\ & & & & 1/2 & \\ 0 & & & & & 1/2 \end{bmatrix} + \frac{\mu_1}{3J^{2/3}} \left(\frac{B_{nn}}{3} \underline{I} \otimes \underline{B}^{-1} - \underline{I} \otimes \underline{I} - \underline{B} \otimes \underline{B}^{-1} \right) + K_1 J(J - 1/2) \underline{I} \otimes \underline{B}^{-1}$$

The matrix **G** represents $\frac{\partial B_{ln}}{\partial F_{ps}} F_{rs} = \left(\delta_{np} B_{lr} + \delta_{lp} B_{nr} \right)$. This is symmetric in *ln*, but not in *pr* and so must be stored as a 6x9 matrix

$$\mathbf{G} = \begin{bmatrix} 2B_{11} & 0 & 0 & 2B_{12} & 0 & 2B_{13} & 0 & 0 & 0 \\ 0 & 2B_{22} & 0 & 0 & 2B_{12} & 0 & 0 & 2B_{23} & 0 \\ 0 & 0 & 2B_{33} & 0 & 0 & 0 & 2B_{13} & 0 & 2B_{23} \\ 2B_{12} & 2B_{12} & 0 & 2B_{22} & 2B_{11} & 2B_{23} & 0 & 2B_{13} & 0 \\ 2B_{13} & 0 & 2B_{13} & 2B_{23} & 0 & 2B_{33} & 2B_{11} & 0 & 2B_{12} \\ 0 & 2B_{23} & 2B_{23} & 0 & 2B_{13} & 0 & 2B_{12} & 2B_{33} & 2B_{22} \end{bmatrix}$$

I couldn’t think of an elegant way to generate this matrix and just typed it in term-by-term. If you can think of a better way please send me the code! Be careful not to make any typos…

The matrix $\mathbf{B}^*$ maps nodal velocities to the velocity gradient (stored as a vector) and is a 9x3*n_nodes matrix:

$$\mathbf{B}^* = \begin{bmatrix} \frac{\partial N^1}{\partial y_1} & 0 & 0 & \frac{\partial N^2}{\partial y_1} & 0 & 0 & \\ 0 & \frac{\partial N^1}{\partial y_2} & 0 & 0 & \frac{\partial N^2}{\partial y_2} & 0 & \\ 0 & 0 & \frac{\partial N^1}{\partial y_3} & 0 & 0 & \frac{\partial N^2}{\partial y_3} & \\ \frac{\partial N^1}{\partial y_2} & 0 & 0 & \frac{\partial N^2}{\partial y_2} & 0 & 0 & \\ 0 & \frac{\partial N^1}{\partial y_1} & 0 & 0 & \frac{\partial N^2}{\partial y_1} & 0 & \ldots\ldots \\ \frac{\partial N^1}{\partial y_3} & 0 & 0 & \frac{\partial N^2}{\partial y_3} & & & \\ 0 & 0 & \frac{\partial N^1}{\partial y_1} & 0 & 0 & \frac{\partial N^2}{\partial y_1} & \\ 0 & \frac{\partial N^1}{\partial y_3} & 0 & & & & \\ 0 & 0 & \frac{\partial N^1}{\partial y_2} & & & & \end{bmatrix}$$

The matrix $\mathbf{\Sigma}$ is a 3*n_nodesx3*n_nodes matrix, which represents the product

$$\Sigma_{aibk} = P_{ak} S_{bi} \qquad P_{ak} = \frac{\partial N^a}{\partial y_k} \qquad S_{bi} = \tau_{ij}[\bar{F}_{kl}] \frac{\partial N^b}{\partial y_j}$$

This is a rather tricky matrix to assemble. Of course, you can do it using brute-force loops. If you want to use the faster built-in Fortran matrix operations, you can use the following approach.

- Note that

$$P_{ak} S_{bi} = \begin{bmatrix} P_{11}S_{11} & P_{12}S_{11} & P_{13}S_{11} & P_{11}S_{21} & P_{12}S_{21} \\ P_{11}S_{12} & & & & \\ P_{11}S_{13} & & & & \\ P_{21}S_{11} & & & & \\ P_{21}S_{12} & & & & \end{bmatrix}$$

$$= \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{11} & P_{12} \\ P_{11} & & & & \\ P_{11} & & & & \\ P_{21} & & & & \\ P_{21} & & & & \end{bmatrix} @ \begin{bmatrix} S_{11} & S_{11} & S_{11} & S_{21} & S_{21} \\ S_{12} & & & & \\ S_{13} & & & & \\ S_{11} & & & & \\ S_{12} & & & & \end{bmatrix}$$

Where @ denotes the elemental product of the two matrices. Note that blocks of these matrices can be created by a sequence of spread and reshape operations.

$$\begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \end{bmatrix} \rightarrow \begin{bmatrix} P_{11} & P_{11} & P_{11} & \cdots & \\ P_{12} & P_{12} & P_{12} & & \\ P_{13} & P_{13} & P_{13} & & \end{bmatrix} \rightarrow \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{11} \cdots \end{bmatrix} \rightarrow \begin{bmatrix} P_{11} & P_{12} & P_{13} & \cdots & \\ P_{11} & P_{12} & P_{13} & & \\ P_{11} & P_{12} & P_{13} & & \end{bmatrix}$$

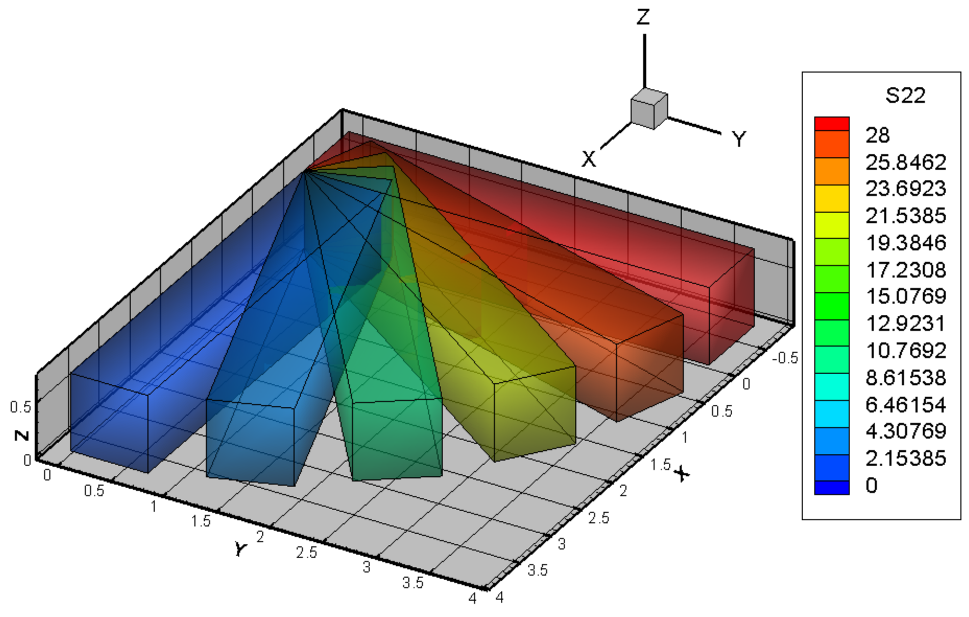
The following lines therefore construct $\mathbf{\Sigma}$ (there are probably other ways as well)

```
S = reshape(matmul(transpose(B),stress),(/3,length_dof_array/3/))
 do i = 1,n_nodes
   Pvec = reshape(spread(transpose(dNdx(i:i,1:3)),dim=2,ncopies=n_nodes),(/3*n_nodes/))
   Pmat(3*i-2:3*i,1:3*n_nodes) = spread(Pvec,dim=1,ncopies=3)
   Svec = reshape(spread(S(1:3,i:i),dim=2,ncopies=n_nodes),(/3*n_nodes/))
   Smat(3*i-2:3*i,1:3*n_nodes) = spread(Svec,dim=1,ncopies=3)
 end do
Sigma = Pmat*transpose(Smat)
```

Run the following tests to check your code:

(1) As always, use CHECK STIFFNESS to check that your element stiffness is consistent with the element force vector.

(2) Run a simple test in which you stretch a hyperelastic bar, with a coarse mesh. A sample input file is provided in a file called Hyperelastic_bar_stretch.in. You will probably need to edit the file to change the integer number used to identify the element, to make it consistent with your code.



(3) Run a simulation in which the bar is first stretched, then rotated, as shown in the figure (this simulation would be impossible with linear elasticity). A user subroutine has been provided with EN234FEA that applies this sequence of deformations, along with the input file called hyperelastic_stretch_rotate (again, you will probably need to edit the file a bit). You should find that s22 at the end of the deformation is equal to S11 at the end of the stretch (obviously!). You could modify the field projection routine to plot principal stresses (which should be constant). The element_utilities module includes a function that calculates principal stresses given a 3D stress vector. Also, the stiffness matrix should still be correct at the end of the deformation.

(4) If steps 1-3 all work, you can try a more complicated problem – there is a sample input file to run the dreaded hole-in-a-plate simulation but with large elastic strains, but you can create your own mesh with a more interesting problem too. The holeplate example will take a few minutes to run (and you have to be a little careful with the choice of material properties to avoid locking).

As a solution to this homework please submit:

(1) A short summary of the tests you ran to demonstrate that your code works correctly

(2) Please push your updated code to GitHub.

2. More challenging problem – a B-bar element for hyperelasticity

Most hyperelastic materials have a large bulk modulus compared to their shear modulus (and are often idealized as incompressible materials). As we have seen, it is important to design elements carefully for this sort of material. The simple implementation described in class (and in problem 1) will suffer from volumetric locking. With this in mind, you will implement a finite strain version of the B-bar method.

As in the small strain B-bar method the basic idea is to interpolate the volumetric and deviatoric strains separately. In finite strain problems we use the deformation gradient as the basic measure of deformation rather than the strain; as you probably know, the Jacobian of the deformation gradient quantifies volume changes. Accordingly, the B-bar method for finite strains uses a modified measure of the Jacobian of the deformation gradient. To this end:

- We define the volume averaged Jacobian of the deformation gradient

$$\eta = \frac{1}{V_{el}} \int_{V_{el}} \det(\mathbf{F}) dV$$

  Here, the integral is taken over the volume of the element in the reference configuration.
- The deformation gradient is replaced by an approximation $\bar{F}_{ij} = F_{ij} \left(\eta / J\right)^{1/n}$, where *n=2* for a 2D plane strain problem and *n=3* for a 3D problem, while *J*=det(**F**).
- The virtual work equation is expressed in terms of this modified deformation gradient as

$$\int_{V_0} \tau_{ij}[\bar{F}_{kl}] \delta \bar{L}_{ij} dV_0 - \int_{V_0} \rho_0 b_i \delta v_i dV_0 - \int_{\partial_2 V_0} t_i^* \delta v_i \eta dA_0 = 0 \qquad (1)$$

  where $\tau_{ij} = J\sigma_{ij}$ is the Kirchhoff stress,

$$\delta \bar{L}_{ij} = \delta \dot{\bar{F}}_{ik} \bar{F}_{kj}^{-1} = \delta L_{ij} + \frac{\delta_{ij}}{n} \left(\frac{\delta \dot{\eta}}{\eta} - \delta L_{kk} \right)$$

  and

$$\delta \dot{\eta} = \frac{1}{V_{el}} \int_{V_{el}} J F_{ji}^{-1} \delta \dot{F}_{ij} dV = \frac{1}{V_{el}} \int_{V_{el}} J \delta L_{kk} dV$$

- We now introduce the usual element interpolation functions and calculate the deformation gradient and velocity gradient as

$$F_{ij} = \delta_{ij} + \frac{\partial u_i}{\partial x_j} = \delta_{ij} + \sum_{a=1}^{n} \frac{\partial N^a}{\partial x_j} u_i^a$$

$$\delta L_{ij} = \frac{\partial \delta v_i}{\partial y_j} = \sum_{a=1}^{n} \frac{\partial N^a}{\partial y_j} \delta v_i^a \qquad \qquad \frac{\partial N^a}{\partial y_j} = \frac{\partial N^a}{\partial x_k} F_{kj}^{-1}$$

- As in the small-strain B-bar method we introduce the volume averaged shape function derivatives

$$\overline{\frac{\partial N^a}{\partial y_i}} = \frac{1}{\eta V_{el}} \int_{V_{el}} J \frac{\partial N^a}{\partial y_i} dV$$

- With these identities the discrete equilibrium equation can be expressed as

$$R_i^a - F_i^a = 0$$

$$F_i^a = \int_{V_0} \rho_0 b_i N^a dV_0 + \int_{\partial_2 V_0} t_i^* N^a \hat{\eta} dA_0$$

$$R_i^a = \int_{V_0} \tau_{mj}[\bar{F}_{kl}] \left[\delta_{im} \frac{\partial N^a}{\partial y_j} + \frac{\delta_{mj}}{n} \left(\overline{\frac{\partial N^a}{\partial y_i}} - \frac{\partial N^a}{\partial y_i} \right) \right] dV_0$$

Notice that this result is essentially identical to equation (1) of Homework 5, except that the shape function derivatives are multiplied by the inverse of the deformation gradient, and we are using the Kirchhoff stress in the finite strain version.

- As usual, we must solve this equation using Newton-Raphson iteration. The correction to the approximation to the solution is computed using

$$K_{aibk} dw_k^b = -R_i^a + F_i^a$$

where the stiffness follow from linearizing the virtual work equation. The element stiffness is

$$K_{aibk}^{el} = \int_{V_0} \frac{\partial \tau_{mj}[\bar{F}_{kl}]}{\partial B_{ln}} \frac{\partial B_{ln}}{\partial \bar{F}_{ps}} \bar{F}_{rs} \bar{F}_{qr}^{-1} \frac{\partial \bar{F}_{pq}}{\partial u_k^b} \left(\delta_{im} \frac{\partial N^a}{\partial y_j} + \frac{\delta_{mj}}{n} \left(\overline{\frac{\partial N^a}{\partial y_i}} - \frac{\partial N^a}{\partial y_i} \right) \right) dV_0$$

$$+ \int_{V_0} \tau_{mj}[\bar{F}_{kl}] \frac{\partial}{\partial u_k^b} \left(\delta_{im} \frac{\partial N^a}{\partial y_j} + \frac{\delta_{mj}}{n} \left(\overline{\frac{\partial N^a}{\partial y_i}} - \frac{\partial N^a}{\partial y_i} \right) \right) dV_0$$

where the term $\bar{F}_{rs} \bar{F}_{qr}^{-1} = \delta_{qs}$ has been introduced for convenience. Simplifying this result is a tedious but straightforward exercise. Note that

$$\frac{\partial \bar{F}_{pq}}{\partial u_k^b} \bar{F}_{qr}^{-1} = \left(\frac{\partial N^b}{\partial y_r} \delta_{pk} + \frac{\delta_{pr}}{n} \left(\overline{\frac{\partial N^b}{\partial y_k}} - \frac{\partial N^b}{\partial y_k} \right) \right)$$

While

$$\frac{\partial}{\partial u_k^b} \left(\delta_{im} \frac{\partial N^a}{\partial y_j} + \frac{\delta_{mj}}{n} \left(\overline{\frac{\partial N^a}{\partial y_i}} - \frac{\partial N^a}{\partial y_i} \right) \right) = -\delta_{im} \frac{\partial N^a}{\partial y_k} \frac{\partial N^b}{\partial y_j} + \frac{\delta_{mj}}{n} \left(P_{aibk} + \frac{\partial N^a}{\partial y_k} \frac{\partial N^b}{\partial y_i} \right)$$

where

$$P_{aibk} = \frac{1}{\eta V_{el}} \int_{V_{el}} \left(J \frac{\partial N^b}{\partial y_k} \frac{\partial N^a}{\partial y_i} - J \frac{\partial N^a}{\partial y_k} \frac{\partial N^b}{\partial y_i} \right) dV - \overline{\frac{\partial N^a}{\partial y_k}} \, \overline{\frac{\partial N^b}{\partial y_i}}$$

We now need to find a way to reduce this to a set of compact matrix operations.
The residual vector has the usual form

$$\mathbf{R} = \int_{V_0} \bar{\mathbf{B}}^T \boldsymbol{\sigma} dV_0$$

where $\boldsymbol{\sigma}$ now stores the Kirchoff stress, and the matrix $\overline{\mathbf{B}}$ is identical to the one you used in Homework 5, except (eg) that $\dfrac{\partial N^b}{\partial x_i}$ has been replaced by $\dfrac{\partial N^b}{\partial y_i} = \dfrac{\partial N^b}{\partial x_k}\overline{F}_{ki}^{-1}$. You can cut and paste the code you used in HW5 and add this small modification.

The stiffness can be expressed as a series of matrix products

$$\mathbf{K} = \int_{V_0} \overline{\mathbf{B}}^T \mathbf{D} \mathbf{G} \mathbf{B}^* dV_0 \;\; + \int_{V_0} -\boldsymbol{\Sigma} + \frac{\tau_{nn}}{n}\left(\mathbf{P}+\mathbf{Q}\right) dV_0$$

The matrices in this expression are defined (and can be computed) as follows:

(1) The Tangent Stiffness $\mathbf{D}$ represents

$$\frac{\partial \tau_{ij}}{\partial B_{kl}} = \frac{\mu_1}{J^{2/3}}\left(\frac{\delta_{ik}\delta_{jl}+\delta_{jk}\delta_{il}}{2} - \frac{1}{3}\delta_{kl}\delta_{ij}\right) - \frac{1}{3}\frac{\mu_1}{J^{2/3}}\left(B_{ij} - \frac{1}{3}B_{nn}\delta_{ij}\right)B_{kl}^{-1} + K_1(J^2 - \frac{1}{2}J)\delta_{ij}B_{kl}^{-1}$$

Here $B_{ij} = \overline{F}_{ik}\overline{F}_{jk}$. We can store the identity matrix, B_{ij}, B_{ij}^{-1} as 1-D vectors, eg

$\underline{I} = [1,1,1,0,0,0]$ $\quad\quad$ $\underline{B} = [B_{11}, B_{22}, B_{33}, B_{12}, B_{13}, B_{23}]$, in which case

$$\mathbf{D} = \frac{\mu_1}{J^{2/3}}\begin{bmatrix} 1 & & & & & 0 \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1/2 & & \\ & & & & 1/2 & \\ 0 & & & & & 1/2 \end{bmatrix} + \frac{\mu_1}{3J^{2/3}}\left(\frac{B_{nn}}{3}\underline{I}\otimes\underline{B}^{-1} - \underline{I}\otimes\underline{I} - \underline{B}\otimes\underline{B}^{-1}\right) + K_1 J(J-1/2)\underline{I}\otimes\underline{B}^{-1}$$

Recall that in Fortran90 the outer product of two 6 dimensional vectors $\mathbf{a}\otimes\mathbf{b}$ can be calculated using the following code `aouterproductb = spread(a,dim=2,ncopies=6)*spread(b,dim=1,ncopies=6)`

(2) The matrix $\mathbf{G}$ represents $\dfrac{\partial B_{ln}}{\partial F_{ps}} F_{rs} = \left(\delta_{np}B_{lr} + \delta_{lp}B_{nr}\right)$. This is symmetric in *ln*, but not in *pr* and so must be stored as a 6x9 matrix

$$\mathbf{G} = \begin{bmatrix} 2B_{11} & 0 & 0 & 2B_{12} & 0 & 2B_{13} & 0 & 0 & 0 \\ 0 & 2B_{22} & 0 & 0 & 2B_{12} & 0 & 0 & 2B_{23} & 0 \\ 0 & 0 & 2B_{33} & 0 & 0 & 0 & 2B_{13} & 0 & 2B_{23} \\ 2B_{12} & 2B_{12} & 0 & 2B_{22} & 2B_{11} & 2B_{23} & 0 & 2B_{13} & 0 \\ 2B_{13} & 0 & 2B_{13} & 2B_{23} & 0 & 2B_{33} & 2B_{11} & 0 & 2B_{12} \\ 0 & 2B_{23} & 2B_{23} & 0 & 2B_{13} & 0 & 2B_{12} & 2B_{33} & 2B_{22} \end{bmatrix}$$

I couldn't think of an elegant way to generate this matrix and just typed it in term-by-term. If you can think of a better way please send me the code! Be careful not to make any typos…

(4) The matrix $\mathbf{B}^*$ maps nodal velocities to the velocity gradient (stored as a vector)

$$\mathbf{B}^{*}=\mathbf{B}^{*}=\begin{bmatrix}
\frac{\partial N^1}{\partial y_1} & 0 & 0 & \frac{\partial N^2}{\partial y_1} & 0 & 0 & \\
0 & \frac{\partial N^1}{\partial y_2} & 0 & 0 & \frac{\partial N^2}{\partial y_2} & 0 & \\
0 & 0 & \frac{\partial N^1}{\partial y_3} & 0 & 0 & \frac{\partial N^2}{\partial y_3} & \\
\frac{\partial N^1}{\partial y_2} & 0 & 0 & \frac{\partial N^2}{\partial y_2} & 0 & 0 & \\
0 & \frac{\partial N^1}{\partial y_1} & 0 & 0 & \frac{\partial N^2}{\partial y_1} & 0 & \ldots\ldots \\
\frac{\partial N^1}{\partial y_3} & 0 & 0 & \frac{\partial N^2}{\partial y_3} & & & \\
0 & 0 & \frac{\partial N^1}{\partial y_1} & 0 & 0 & \frac{\partial N^2}{\partial y_1} & \\
0 & \frac{\partial N^1}{\partial y_3} & 0 & & & & \\
0 & 0 & \frac{\partial N^1}{\partial y_2} & & & &
\end{bmatrix}+\frac{1}{3}\begin{bmatrix}
\left(\overline{\frac{\partial N^1}{\partial y_1}}-\frac{\partial N^1}{\partial y_1}\right) & \left(\overline{\frac{\partial N^1}{\partial y_2}}-\frac{\partial N^1}{\partial y_2}\right) & \left(\overline{\frac{\partial N^1}{\partial y_3}}-\frac{\partial N^1}{\partial y_3}\right) & \left(\overline{\frac{\partial N^2}{\partial y_1}}-\frac{\partial N^2}{\partial y_1}\right) & \cdots \\
\left(\overline{\frac{\partial N^1}{\partial y_1}}-\frac{\partial N^1}{\partial y_1}\right) & \left(\overline{\frac{\partial N^1}{\partial y_2}}-\frac{\partial N^1}{\partial y_2}\right) & \left(\overline{\frac{\partial N^1}{\partial y_3}}-\frac{\partial N^1}{\partial y_3}\right) & & \\
\left(\overline{\frac{\partial N^1}{\partial y_1}}-\frac{\partial N^1}{\partial y_1}\right) & \left(\overline{\frac{\partial N^1}{\partial y_2}}-\frac{\partial N^1}{\partial y_2}\right) & & & \\
0 & 0 & 0 & & \\
 & & & \ddots & \\
 & & & &
\end{bmatrix}$$

We next examine the terms in the geometric stiffness: $\int\limits_{V_0} -\boldsymbol{\Sigma}+\frac{\tau_{nn}}{n}\left(\mathbf{P}+\mathbf{Q}\right)dV_0$.

(1) The matrix **P** stores $P_{aibk}=\frac{1}{\eta V_{el}}\int\limits_{V_{el}}\left(J\frac{\partial N^b}{\partial y_k}\frac{\partial N^a}{\partial y_i}-J\frac{\partial N^a}{\partial y_k}\frac{\partial N^b}{\partial y_i}\right)dV-\overline{\frac{\partial N^a}{\partial y_i}}\,\overline{\frac{\partial N^b}{\partial y_k}}$. **P** is constant in each element, and for a 3D problem with n nodes it is a 3n x 3n matrix. It can be calculated at the same time as $\overline{\frac{\partial N^a}{\partial y_i}}$. To evaluate the various products of the shape function derivatives, note that:

- Matrices of the form $\frac{\partial N^a}{\partial y_i}\frac{\partial N^b}{\partial y_k}$ can be assembled as the outer product of two vectors

$$\mathbf{n}_V=\begin{bmatrix}\frac{\partial N^1}{\partial y_1} & \frac{\partial N^1}{\partial y_2} & \frac{\partial N^1}{\partial y_3} & \frac{\partial N^2}{\partial y_1} & \frac{\partial N^2}{\partial y_2} & \frac{\partial N^2}{\partial y_3} & \cdots\end{bmatrix}$$

  They can therefore be constructed with the following two lines of code (which assumes a 3D problem):

```
dNdxvec(1:3*n_nodes) = reshape(transpose(dNdx(1:n_nodes,1:3)),(/3*n_nodes/))
result=spread(dNdxvec,dim=2,ncopies=3*n_nodes)*spread(dNdxvec,dim=1,ncopies=3*n_nodes)
```

where `dNdxvec` must be declared as a 3*n_nodes long 1-D vector.

- Products of the form $A_{ak}B_{bi}$ appear in both **P** and **Q**, and are somewhat more cumbersome to assemble. Note that

$$A_{ak}B_{bi} = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{11} & A_{13}B_{11} & A_{11}B_{21} & A_{12}B_{21} \\ A_{11}B_{12} & & & & \\ A_{11}B_{13} & & & & \\ A_{21}B_{11} & & & & \\ A_{21}B_{12} & & & & \end{bmatrix}$$

$$= \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{11} & A_{12} \\ A_{11} & & & & \\ A_{11} & & & & \\ A_{21} & & & & \\ A_{21} & & & & \end{bmatrix} @ \begin{bmatrix} B_{11} & B_{11} & B_{11} & B_{21} & B_{21} \\ B_{12} & & & & \\ B_{13} & & & & \\ B_{11} & & & & \\ B_{12} & & & & \end{bmatrix}$$

Where @ denotes the elemental product of the two matrices. Note that blocks of these matrices can be created by a sequence of spread and reshape operations.

$$\begin{bmatrix} A_{11} \\ A_{12} \\ A_{13} \end{bmatrix} \rightarrow \begin{bmatrix} A_{11} & A_{11} & A_{11} & \cdots \\ A_{12} & A_{12} & A_{12} & \\ A_{13} & A_{13} & A_{13} & \end{bmatrix} \rightarrow \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{11} \cdots \end{bmatrix} \rightarrow \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots \\ A_{11} & A_{12} & A_{13} & \\ A_{11} & A_{12} & A_{13} & \end{bmatrix}$$

The matrix representing $\dfrac{\partial N^a}{\partial y_k}\dfrac{\partial N^b}{\partial y_i}$ can therefore be constructed as follows

```
do i = 1,n_nodes
  Pvec = reshape(spread(transpose(dNdx(i:i,1:3)),dim=2,ncopies=n_nodes),(/3*n_nodes/))
  Pmat(3*i-2:3*i,1:3*n_nodes) = spread(Pvec,dim=1,ncopies=3)
end do
result = Pmat*transpose(Pmat))
```

(2) The matrix **Q** is also a 3*n x* 3*n* matrix storing $\dfrac{\partial N^a}{\partial y_k}\dfrac{\partial N^b}{\partial y_i}$. This can be computed using the procedure outlined above.

(3) Finally, the matrix $\boldsymbol{\Sigma}$ represents the product $\dfrac{\partial N^a}{\partial y_k}\tau_{ij}[\bar{F}_{kl}]\dfrac{\partial N^b}{\partial y_j}$ and is also a 3*n* x 3*n* matrix (for a 3D problem). It can be assembled by first constructing the matrix $S_{bi} = \dfrac{\partial N^b}{\partial y_j}\tau_{ij}$ and then following the procedure used to create **P** and **Q**, i.e

```
 S = reshape(matmul(transpose(B),stress),(/3,length_dof_array/3/))
 do i = 1,n_nodes
   Pvec = reshape(spread(transpose(dNdx(i:i,1:3)),dim=2,ncopies=n_nodes),(/3*n_nodes/))
   Pmat(3*i-2:3*i,1:3*n_nodes) = spread(Pvec,dim=1,ncopies=3)
   Svec = reshape(spread(S(1:3,i:i),dim=2,ncopies=n_nodes),(/3*n_nodes/))
   Smat(3*i-2:3*i,1:3*n_nodes) = spread(Svec,dim=1,ncopies=3)
 end do
Sigma = Pmat*transpose(Smat)
```

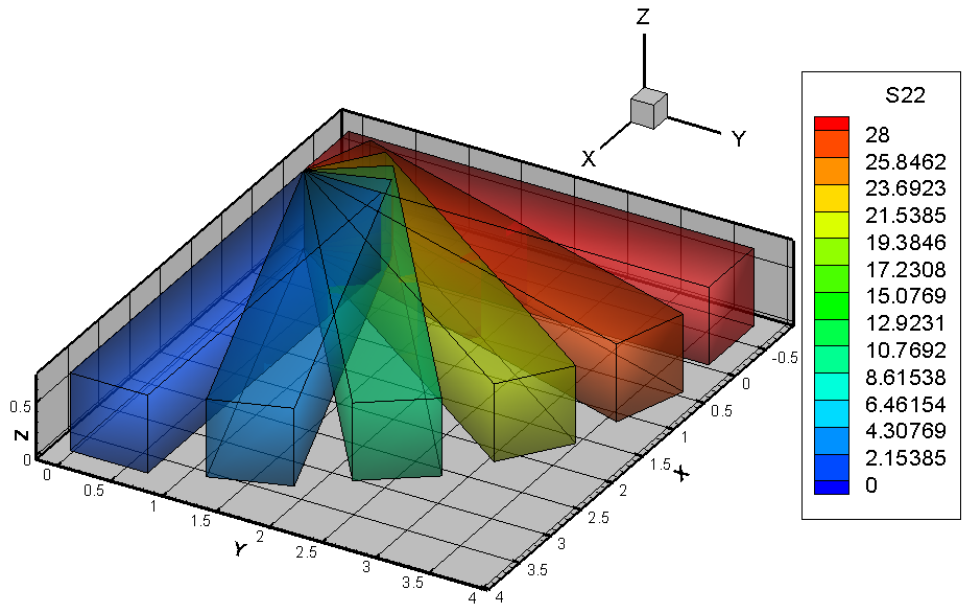
Here is a rough code template, which broadly follows the steps you used to construct your B-bar element:

1. Initialize any variables that are incremented in the loops below
2. Initialize integration points and weights
3. Loop over integration points
   a. Compute shape functions and derivatives; convert to $\frac{\partial N^b}{\partial x_k}$ in the usual way
   b. Compute the deformation gradient **F** (3x3 matrix) $F_{ij} = \delta_{ij} + \frac{\partial u_i}{\partial x_j} = \delta_{ij} + \sum_{a=1}^{n} \frac{\partial N^a}{\partial x_j} u_i^a$
   c. Compute inverse of the deformation gradient and J
   d. Convert the shape function derivatives to derivatives with respect to deformed coordinates $\frac{\partial N^b}{\partial y_i} = \frac{\partial N^b}{\partial x_k} F_{ki}^{-1}$
   e. Add contribution to $\partial \bar{N}^a / \partial y_i$ and $\bar{J} = \frac{1}{V_{el}} \int_{V_{el}} J dV$ from current integration point
   f. Add contribution to element volume from current integration point
   g. Add contribution to $\int_{V_{el}} \left(J \frac{\partial N^b}{\partial y_k} \frac{\partial N^a}{\partial y_i} - J \frac{\partial N^a}{\partial y_k} \frac{\partial N^b}{\partial y_i} \right) dV$ from current integration point
4. Divide the result of (g) above and $\partial \bar{N}^a / \partial y_i$ by element volume and $\bar{J}$; add the last term in the expression for **P**
5. Loop over integration points
   a. Compute shape functions and derivatives; convert to $\frac{\partial N^b}{\partial x_k}$ in the usual way
   b. Compute the deformation gradient **F** (3x3 matrix) $F_{ij} = \delta_{ij} + \frac{\partial u_i}{\partial x_j} = \delta_{ij} + \sum_{a=1}^{n} \frac{\partial N^a}{\partial x_j} u_i^a$
   c. Compute inverse of the deformation gradient and J
   d. Convert the shape function derivatives $\frac{\partial N^b}{\partial y_i} = \frac{\partial N^b}{\partial x_k} F_{ki}^{-1}$
   e. Compute $\bar{\mathbf{F}} = (\bar{J} / J)^{1/3} \mathbf{F}$
   f. Compute the Kirchhoff stress $\boldsymbol{\sigma}$ and material tangent stiffness **D**
   g. Assemble $\bar{\mathbf{B}}, \mathbf{B}^*, \mathbf{G}$ (note that you must use the modified deformation gradient $\bar{\mathbf{F}}$ in computing **G**).
   h. Compute $\boldsymbol{\Sigma}$ and **Q**.
   i. Add the contributions from the current integration point to the integrals

$$\mathbf{R} = \int_{V_0} \bar{\mathbf{B}}^T \boldsymbol{\sigma} dV_0 \qquad \mathbf{K} = \int_{V_0} \bar{\mathbf{B}}^T \mathbf{D} \mathbf{G} \mathbf{B}^* dV_0 + \int_{V_0} -\boldsymbol{\Sigma} + \frac{\tau_{nn}}{n} (\mathbf{P} + \mathbf{Q}) dV_0$$

Run the following tests to check your code:

(5) As always, use CHECK STIFFNESS to check that your element stiffness is consistent with the element force vector.
(6) Run a simple test in which you stretch a hyperelastic bar, with a coarse mesh. A sample input file is provided in a file called Hyperelastic_bar_stretch.in. You will probably need to edit the file to change the integer number used to identify the element, to make it consistent with your code.



(7) Run a simulation in which the bar is first stretched, then rotated, as shown in the figure (this simulation would be impossible with linear elasticity). A user subroutine has been provided with EN234FEA that applies this sequence of deformations, along with the input file called hyperelastic_stretch_rotate (again, you will probably need to edit the file a bit). You should find that s22 at the end of the deformation is equal to S11 at the end of the stretch (obviously!). You could modify the field projection routine to plot principal stresses (which should be constant). The element_utilities module includes a function that calculates principal stresses given a 3D stress vector. Also, the stiffness matrix should still be correct at the end of the deformation.
(8) If steps 1-3 all work, you can try a more complicated problem – there is a sample input file to run the dreaded hole-in-a-plate simulation but with large elastic strains, but you can create your own mesh with a more interesting problem too. The holeplate example will take a few minutes to run….

As a solution to this homework please submit:
(3) A short summary of the tests you ran to demonstrate that your code works correctly
(4) Please push your updated code to GitHub.