19/02/2016

Elasto-plastic analysis of a slit tube using total deformation theory

Computer assignment 2 – FEM Structure

Chalmers University of Technology
LACERDA ANTONIO – BROUSSEAU PAUL

INDEX

Laderda Antonio
Brousseau Paul

Problem description

During this second computer assignment, a 2D cross section of a slit tube is consider as illustrated in figure 1. The problem is symmetric so we will only analyse half of the tube and apply a vertical displacement -Δ/2, as shown in figure 2 in order to study the global force required to open the tube by separating the slit surfaces uniformly a distance Δ. Obviously, it will imply displacements, strains and stresses distributions on each part of the tube.

These three distributions are studied first by using a linear model (purely elastic case) and then a non-linear elasto-plastic one. For this last cases, iterative solution techniques will be used and compared: Newton method, modified Newton method and the BFGS method.

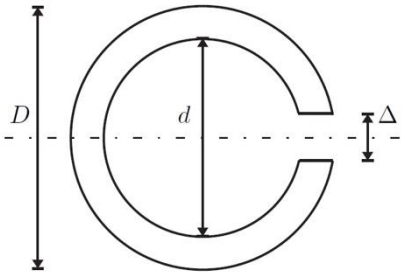

Figure 1 : The 2D cross section of a slit tube to be studied in the project. The opening of the cross section is denoted Δ.

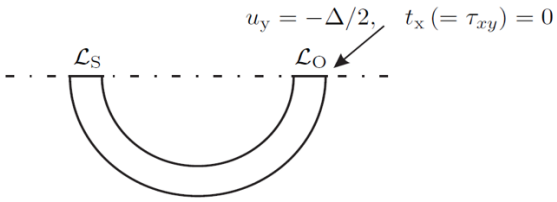

Figure 2 : Analysed part of the tube, due to symmetry, Δ as prescribed constant to choose during the analysis. On Ls, vertical displacement equals zero, and horizontal displacement of one point is zero too to block x-axis movement.

Tube parameters:

D = 0.5m
d = 0.3m
thickness = 0.5m
Plain stress conditions

Material parameters (Henky-type elasto-placticity):

E=210 Gpa
v=0.3
σ_y =220MPa

Laderda Antonio
Brousseau Paul

Analysis

Calfem will be used such as in CA1 – The Hoover dam. Thus we need to mesh the tube in order to used Calfem. "Ringmesh" program is given so that we don't have to generate our own mesh. Thus 3-node triangular elements will be used. "Ringmesh" gives us access to the connectivity matrix (Edof), the x and y coordinate of the nodes (Ex and Ey) and the degrees of freedom of boundaries' (Bi) and corners' (Pi) nodes (i = 1,2,3,4).

Purely linear elastic model analysis

For this fist analysis, only the linear model is considered. That is why the material matrix **D**, which is computed with Hooke's law remains constant.
In order to compute the global stiffness matrix **K**, we need to set the boundary conditions (figure 2):

- Define Δ and apply the displacement -Δ/2 on L_0 with the help of B1 degrees of freedom (dofs).
- Set to the desired value.
- The ring should not move on the y-axis, thus B3 dofs are set to zero.
- One dof on L_S (P_4 in our case) is locked to prevent movement on x-axis.

Displacement

Thus we can now use the *plante* function to compute **Ke** and **fe**, the element stiffness matrix and then assemble the values for each element, regrouping all the **Ke** into the **K** matrix and **fe** into the **F** matrix.

In order to obtain the displacement a, we need to obtain K_{FF} and the K_{FC} components of the K matrix. In order to do that we write, since we already have spilt the degree of freedom in fdof (free) and cdof (constraint) so:

$$K_{FF} = K(fdof, fdof)$$
$$K_{FC} = K(fdof, cdof)$$
$$K_{CC} = K(cdof, cdof)$$
$$K_{CF} = K(cdof, fdof)$$

We can now compute a because: $K_{FF}a_F = f_F - K_{FC}a_C$. The latter can be deduced by using the FE – form (4. 54(Larsson, 2010)) and stating the following:

$$\int_A \underline{\underline{B}}^T \underline{\sigma} t dA = \int_A \underline{\underline{N}}^T \underline{b} t dA + \int_{L_g} \underline{\underline{N}}^T \underline{t} t dL + \int_{L_h} \underline{\underline{N}}^T \underline{h} t dL$$

Where $\int_A \underline{\underline{B}}^T \underline{\sigma} t dA = \underline{f_{int}}$; $\int_A \underline{\underline{N}}^T \underline{b} t dA = \underline{f_e}$; $\int_{L_g} \underline{\underline{N}}^T \underline{t} t dL + \int_{L_h} \underline{\underline{N}}^T \underline{h} t dL = \underline{f_b}$ and

$$\int_{L_g} \underline{\underline{N}}^T \underline{t} t dL = reactions$$

Therefore, as we have linear elasticity, i.e. $\underline{\sigma} = \underline{\underline{D}}\,\underline{\varepsilon}$

$$\underline{f_{int}} = \int_A \underline{\underline{B^T}}\,\underline{\sigma} t dA = \left(\int_A \underline{\underline{B^T}}\,\underline{\sigma} t dA\right)\underline{a} = \underline{\underline{K}}\,\underline{a} = \underline{f_e} + \underline{f_b} = \underline{f}$$

We will, thus, obtain the formula stated in the beginning:

$$\underline{\underline{K_{FF}}}\,\underline{a_F} = \underline{f_F} - \underline{\underline{K_{FC}}}\,\underline{a_C} \leftrightarrow \underline{a_F} = \underline{\underline{K_{FF}}}^{-1}(\underline{f_F} - \underline{\underline{K_{FC}}}\,\underline{a_C})$$

In the formula above, the only unknown is $\underline{a_F}$.

Finally, the displacement can be plot:

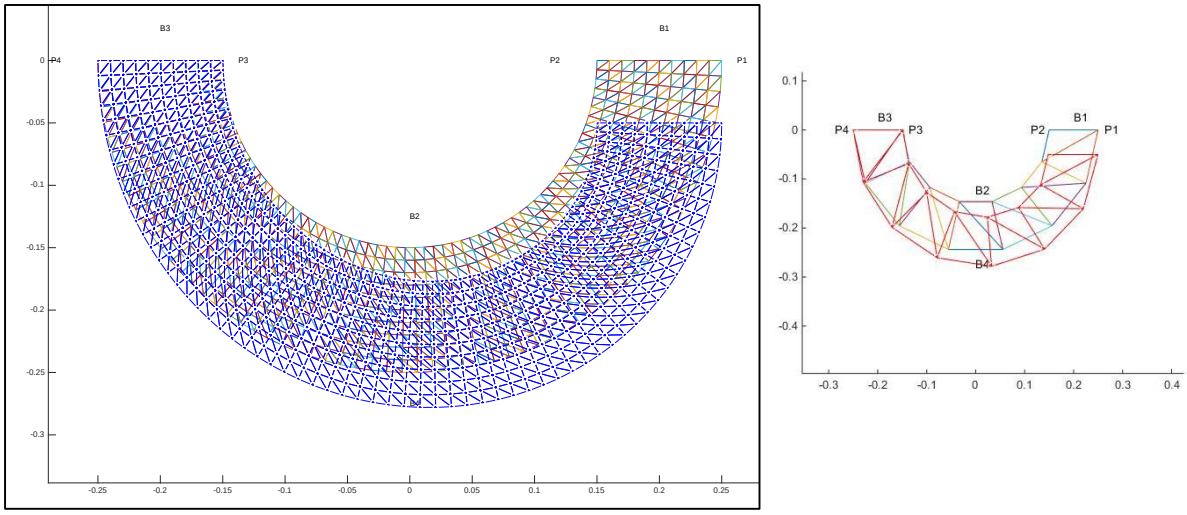

Figure 3 : In the left deformed mesh in blue and undeformed mesh; Δ = 0,001; hr = 0,01; hphi = 0,01. On the right the deformed mesh in red for a Δ = 0,1; hr = 0,01; hphi = 0,1.

Even though we are simulating using a purely linear elastic model, this result is reasonable, as we are most likely still in the elastic domain. This can be based on the results obtained later on, where we will have that for a sigma much higher than the yield stress the deformation delta is$\sim 0{,}0034$, which is 3 times higher than the delta used here.

Constraints

The effective (Mises) stress distribution also has to be calculated. The function *plants* is used in a loop in order to calculate the element stress matrix **es**. This function need the element displacement vector **ed** computed with the function *extract* applied to the displacement **a** and to the **Edof** matrix, i.e. the connectivity matrix. We are, thus, able to calculate the stress matrix **S** for each element and therefore, the effective stress distribution according to these formulas (We will use the one to the right):

$$\sigma_{eff} = \sqrt{\frac{3}{2}}.\sqrt{\sigma^T II_{c,dev}\sigma} = \sqrt{\frac{3}{2}}.\|S_{dev}\| \ (4.115\ and\ 4.122\ (\text{Larsson}, 2010))$$

Laderda Antonio
Brousseau Paul

Where $\sigma = [\, \sigma_{xx}\ \sigma_{yy}\ \sigma_{zz}\ \sigma_{xy}\ 0\ \ 0\,]^T$

$$II_{c,dev} = \begin{bmatrix} 2/3 & -1/3 & -1/3 & 0 & 0 & 0 \\ -1/3 & 2/3 & -1/3 & 0 & 0 & 0 \\ -1/3 & -1/3 & 2/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{bmatrix}.$$

$$S_{dev} = \ S - \ \sigma_m I$$

Therefore, we are able to plot the effective stress distribution for one cross section of the slit tube. In order to verify the accuracy of the result we will use different size of element by changing values of h_r and h_{phi}.

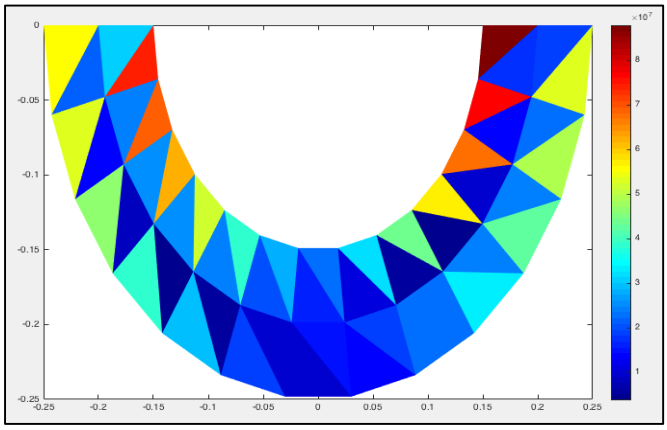

Figure 5 : Stress distribution; hr = 0, 05; hphi = 0, 05

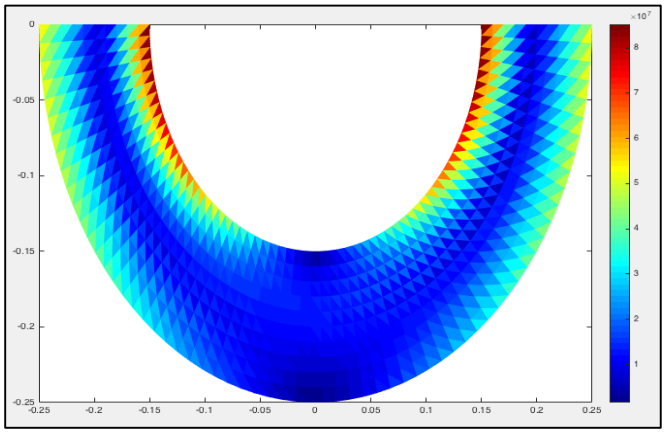

Figure 4 : Stress distribution; hr = 0, 01; hphi = 0, 01

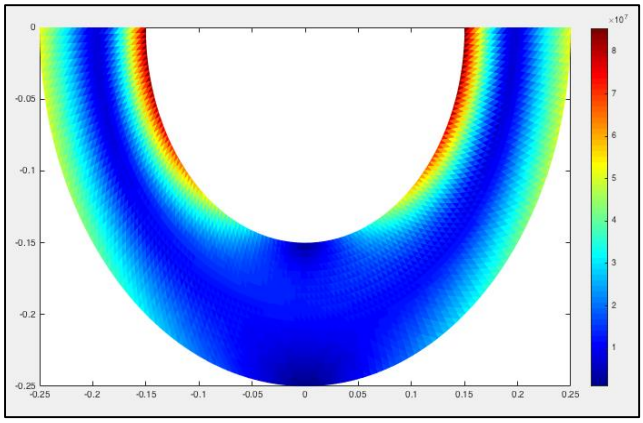

Figure 6 : Stress distribution; hr = 0,005; hphi = 0,005

We can notice that no matter how many elements we have, the maximal constraint is between $8*10^7$ and $9*10^7$ (figure 4,5 and 6).
Obviously, the smaller the elements, the more accurate and the longer the calculation. Thus, a compromise has to be done to get the best results with an acceptable calculation time.

Laderda Antonio
Brousseau Paul

Choice of Δ

In order to be coherent for the further analysis, we need to choose a displacement Δ and keep it constant during all the analysis. We choose the Δ which gives the maximum stress well above the yield limit of the material: $\bar{\Delta} = \ 0{,}006$.

Vertical reaction force for the linear elasticity model

We want now to compute the vertical reaction force on L_0: $V = \int_{Lo} t_y dL$. It can be found using the result Q = K*A with **K** the stiffness matrix and **a** the displacement (note that we also might obtain this result by using the function *solveq* and *extract* of calfem). **Q** is therefore the reaction force vector. **Q** has a value for each dof so that we need to calculate and to sum the reaction force only on the dof of L_0, that is to say B1 in the mesh. Thus:

$$\mathrm{V} \ = \ \mathrm{sum}(\mathrm{Q}(\mathrm{B1}(:,2)))$$

We can now plot V as a function of Δ/2. We need for that purpose to generate a loop we compute for each step the value of Δ/2, the new boundary conditions for this delta, and the corresponding reaction force vector Q. Finally, we are able to plot V:

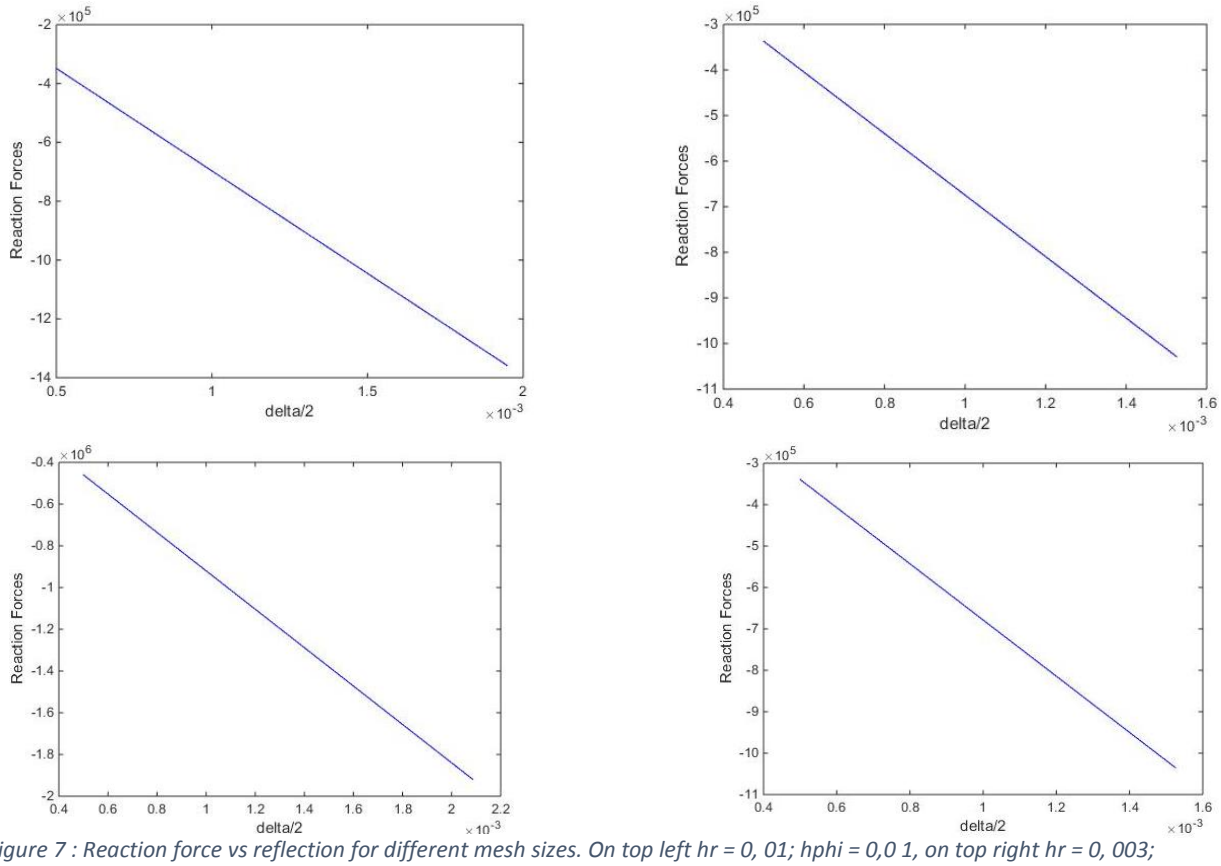

Figure 7 : Reaction force vs reflection for different mesh sizes. On top left hr = 0, 01; hphi = 0,0 1, on top right hr = 0, 003; hphi = 0, 003 , on bottom left hr = 0, 02; hphi = 0, 05 and on bottom rigth hr = 0, 005; hphi = 0, 005.

The plot obtained is a linear curve. This is not surprising at all because we assumed a purely linear elastic model. Finally, the absolute value of V increases with the deflection. This is the result expected, because more deflection means more stresses in the tube, which means a higher reaction on Lo. We can also observe that for a fixed value of Delta/2, the corresponding Reaction Force reduces as we decrease the size of each element of the mesh.

Laderda Antonio
Brousseau Paul

Henky plasticity model

The Henky plasticity model is an elasto-plastic model, which means that the material matrix D, the Von Mises effective stress (more generally the stress) and the shear modulus depend on the strain. At the end, we should obtain the convergence of the out-of-balance force vector g, which will be defined later in this report, to either 0 or smaller than the Tolerance (10^{-6} for our program). To do so, we will be using three different iteration methods.

First, we set $\frac{\Delta}{2} = 3\,mm.$

Thus, a for loop is generated in order to simulate the opening of Δ/2 step by step. All the calculations (boundary conditions, Henky plasticity implementation, iterations) are made for each step. Therefore, the final displacement Δ/2 is reached by addition of all of the small displacements. This method is called load stepping and it is used to improve the convergence of the iteration method e.g. Newton method. Without this, the initial guess for the displacements[1] is most likely not to be good enough to allow for convergence and the iteration method is probably going to diverge.

In order to apply the iterative methods and the Henky plasticity implementation a while loop is generated. It will calculate the corresponding out of the balance force vector **g** (which is, for our case, equal $\underline{f_{int}}$) and, if the norm(**g**) doesn't respects the Tolerance, will update the displacement vector to a more suitable assumption. This process will repeat until the norm(**g**) respects the desired Tolerance value. This process is detailed bellow.

We will initially calculate the strain[2] vector using the displacement vector **a** with the plants method. Making it possible to compute the effective Von Mises strain because:

$$\varepsilon_e^{vM} = \sqrt{\frac{2}{3}} \cdot \sqrt{\varepsilon^T II_{s,dev} \varepsilon}$$

With $\varepsilon = [\,\varepsilon_{xx}\ \varepsilon_{yy}\ \varepsilon_{zz}\ \varepsilon_{xy}\ 0\ \ 0\,]^T$

$$II_{s,dev} = \begin{bmatrix} 2/3 & -1/3 & -1/3 & 0 & 0 & 0 \\ -1/3 & 2/3 & -1/3 & 0 & 0 & 0 \\ -1/3 & -1/3 & 2/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{bmatrix}$$

Now, Gs and the material matrix D is able to be computed, which depend on ε_e^{vM}. They follow the conditions bellow:

[1] We only know the constrained values, the ones corresponding to the free dofs are unknowns. Since we need to know the strain to calculate the stress, we need to make an initial guess for the displacement vector in order to start the calculation.

[2] We will use here the Voigt notation

Laderda Antonio
Brousseau Paul

$$G_S = \begin{cases} G & if \quad \varepsilon_e^{vM} \leq \frac{\sigma_Y}{3G} \\ \frac{\sigma_Y}{3\varepsilon_e^{vM}} & else \end{cases}$$

$$D_t = \begin{cases} 2GII_{s,dev} + K_{(modulus)}\delta\delta^T & if \quad \varepsilon_e^{vM} < \frac{\sigma_Y}{3G} \\ \frac{2\sigma_Y}{3\varepsilon_e^{vM}} II_{s,dev} - \varepsilon_{dev}\frac{4\sigma_Y}{9(\varepsilon_e^{vM})^3}\varepsilon_{dev}^T + K\delta\delta^T & else \end{cases}$$

With $\delta = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ and $\varepsilon_{dev} = II_{s,dev}.\varepsilon$

And then we are able to determine $D_S(\varepsilon) = 2G^*(\varepsilon_e^{vM})II_{s,dev} + K\delta\delta^T$

The $D_S(\varepsilon)$ will then be used to deduce the secant matrix $\boldsymbol{K_S}$ and $D_t(\varepsilon)$ will be used to deduce the tangent matrix $\boldsymbol{K_T}$ **.** They will both be calculated using the plate function. The tangent matrix will be used later on to obtain the Jacobian and the secant will be used here to calculate the internal forces f_{int}.

The stress and the effective Von Mises stress σ_e^{vM} are calculated as follows:

$$\sigma = D_S(\varepsilon)$$

$$\sigma_e^{vM} = \left(\frac{3}{2}\right)^{\frac{1}{2}} \left(\sigma^T II_{c,dev}\sigma\right)^{1/2}$$

Where $II_{s,dev} = \begin{bmatrix} 2/3 & -1/3 & -1/3 & 0 & 0 & 0 \\ -1/3 & 2/3 & -1/3 & 0 & 0 & 0 \\ -1/3 & -1/3 & 2/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$

Finally, we will calculate f_{int} : $f_{int} = K_s a$

Therefore, we are able to obtain the out of the balance force vector **g :** $g(free) = f_{int}(free)$

As stated before, this will be calculated for every iteration until norm(**g**) is smaller than the tolerance. After norm(**g**) is calculated and in the case that it does not respect the tolerance, we will use the methods explained bellow to update the guess for the displacement.

Computation of the different iteration methods

In order to solve the problem, three different iteration methods will be used in the for loop generated to open the tube step by step: Newton, Modified Newton and BFGS. To do so, we need to consider a small displacement, since, as explained above, if the initial guess is too far from the reality, there will be no convergence. Therefore, at each step it will be increased until Δ/2. Furthermore, only the free displacement **af** has to be guessed.

Laderda Antonio
Brousseau Paul

Finally, the goal of the three method is to make the function norm(**g**) converged.

Newton method

In order to use Newton method, an initial guess for the displacement **a** is needed. We will be using zero for the free displacement and the boundary conditions for the constrained values (will remain constant until the end). The Jacobian matrix corresponds to the stiffness matrix **Kt** for the free degrees of freedom calculated in the previous part. At each iteration we have:

$$\begin{cases} J^{(k)} = K_{T,FF}^{(k)} \\ g^{(k)} = f_{int,F} \\ \Delta a^{(k)} = -\dfrac{g^{(k)}}{J^{(k)}} \\ a_F^{(k+1)} = a_F^{(k)} + \Delta a^{(k)} \end{cases}$$

We obtain a quadratic convergence, i.e. the convergence rate is two.

$$|\mathrm{x} - \mathrm{x(k+1)}| \leq \mathrm{C}|\mathrm{x} - \mathrm{x(k)}|^2$$

Moreover, the convergence can be calculated as: $q = \frac{\ln\left(g(x^{k+1})\right)}{\ln(g(x^k))}$ where **q** is the convergence rate.

Modified Newton method

In modified Newton's method, the Jacobian matrix is calculated once at the first iteration and then maintained constant, **J** is fixed:

$$\begin{cases} J^{(k)} = J^{(0)} = K_{T,FF}^{(0)} \\ g^{(k)} = f_{int,F} \\ \Delta a^{(k)} = -\dfrac{g^{(k)}}{J^{(k)}} \\ a_F^{(k+1)} = a_F^{(k)} + \Delta a^{(k)} \end{cases}$$

The modified newton does not yield quadratic convergence.

For our specific case, we were not able to find convergence if we used the Jacobian for the first iteration, therefore we need to use the Jacobian calculated in the second iteration. This is most likely due to a bad guess and an error in the initialization, nevertheless we were not able to find it.[3]

[3] The tutor told us that using the Jacobian calculated during the second iteration was accepted.

Laderda Antonio
Brousseau Paul

BFGS method

In BFGS's method, the first iteration Jacobian is determined like in Newton's method:

$$\begin{cases} J^{(0)} = K_{T,FF}^{(0)} \\ g^{(0)} = f_{int,F} \\ \Delta a^{(0)} = -\dfrac{g^{(0)}}{J^{(0)}} \\ a_F^{(1)} = a_F^{(0)} + \Delta a^{(0)} \end{cases}$$

Then, the new Jacobian is estimated using the one of the last iteration:

$$\begin{cases} \bar{c} = f_{int,F} - g^{(k-1)} \\ \bar{b} = J^{(k-1)}\Delta a^{(k-1)} \\ \alpha = \dfrac{1}{(f_{int,F} - g^{(k-1)})^T \Delta a^{(k-1)}} \\ \beta = \dfrac{-1}{\Delta a^{(k-1)T} J^{(k-1)T} \Delta a^{(k-1)}} \\ J^{(k)} = J^{(k-1)} + \alpha c c^T + \beta b b^T \\ g^{(k)} = f_{int,F} \\ \Delta a^{(k)} = -\dfrac{g^{(k)}}{J^{(k)}} \\ a_F^{(k+1)} = a_F^{(k)} + \Delta a^{(k)} \end{cases}$$

The convergence for the BFGS method is superlinear, i.e. the convergence behaves like for a linear convergence but, the constant C reduces as the approximation gets closer to the exact solution. This can be seen as the transition from linear to a quadratic convergence.

$$|\mathrm{x} - \mathrm{x}(\mathrm{k}+1)| \leq \mathrm{C}|\mathrm{x} - \mathrm{x}(\mathrm{k})|$$

Optimization methods

In our program we used two methods to improve the convergence of the iteration methods. The first is the load stepping, which has already been explained. The second is a modification in how we increment the initial guess after each load step. If instead of merely using the guess obtained for the previous step, we use the following formula:

$$\underline{a_{F,i+2}^0} = \underline{a_{F,i+1}^n} + \underline{a_{F,i+1}^n} - \underline{a_{F,i}^n}$$

Where: $a_{F,i+1}^n$ is the approximation obtained for the previous step.
$a_{F,i}^n$ Is the one for the step before the previous step.

This will allow us to make a much better guess.

Laderda Antonio
Brousseau Paul

Load deflection curve

In order to calculate the vertical reaction on L_0 we need to determine all of the internal force vector f_{int}^{e} on all the element of L_0. For this we will do the following.

$$f_{int}^{e} = KS.a$$

And

$$V = sum(f_{int}^{e}((\mathrm{B1}(:,2))))$$

We will plot the V in function of the displacement.

Results

Stress Field

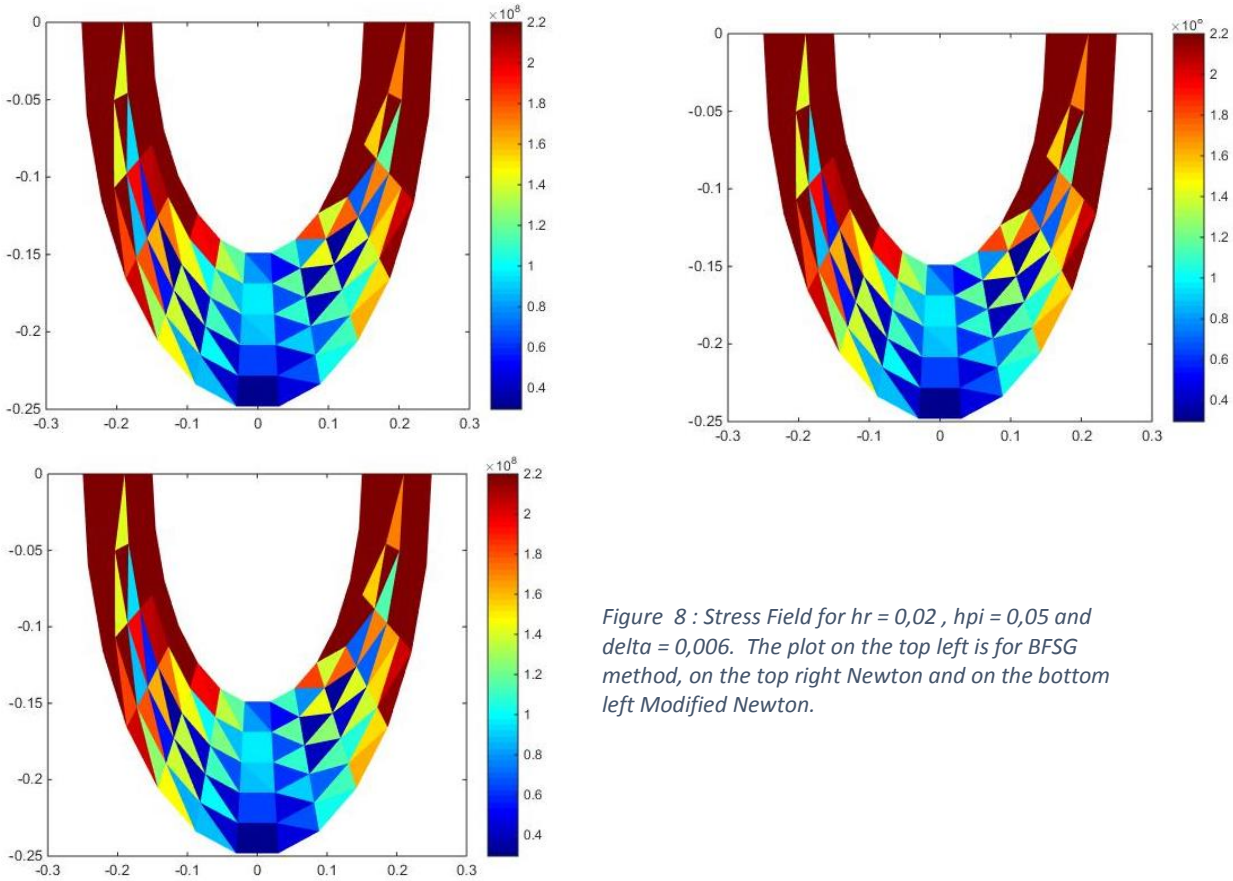

Figure 8 : Stress Field for hr = 0,02 , hpi = 0,05 and delta = 0,006. The plot on the top left is for BFSG method, on the top right Newton and on the bottom left Modified Newton.

Initially, we can observe that the plot is the same for all three methods, which confirm that our program work as expected for medium meshes. Moreover, we can see that it corresponds to a plastic deformation, as the maximum stress is $220\ MPa$ in the B1 and B3 borders. We can also notice that the stress is lower in the middle of the tube, which is also expected.

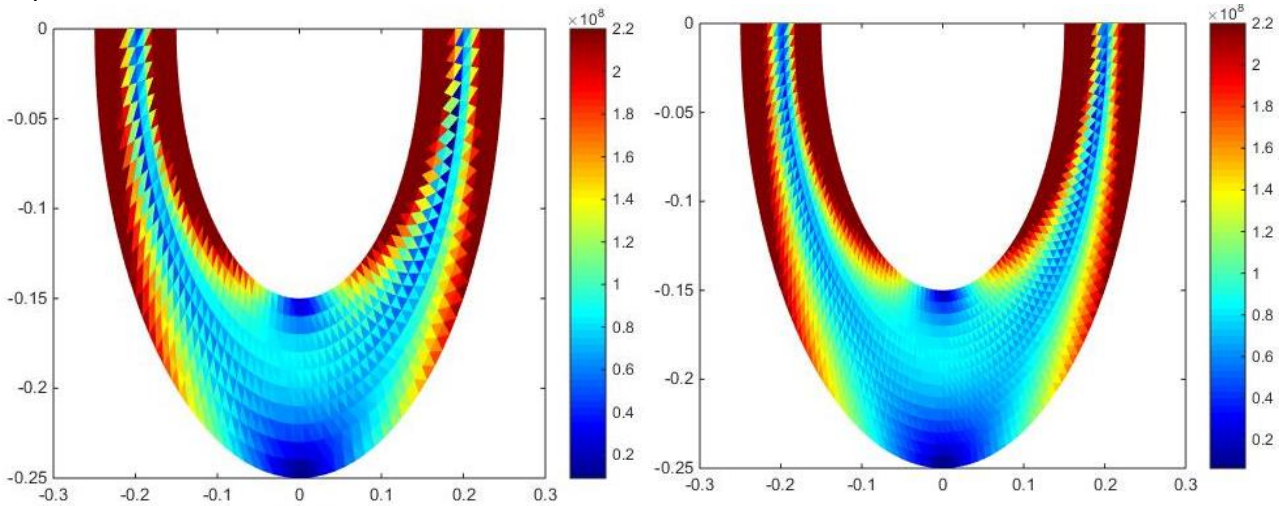

Figure 9 : Stress Field for hr = 0, 01, hpi = 0, 01 and delta = 0,006 to the left and Force for hr = 0, 007, hpi = 0, 007 and delta = 0,006 to the right.

We can see here that the maximum stress is, if compared to the medium size mesh, more localized at the surface. This is expected as we used in this program the Henky elasto-plasticity method, which considers the stress to be dependent on the strain. Since the strain is bigger at the surface, this means that the stress will be also higher at the surface. By the same logic, the stresses should reduce as we move into the material, which is precisely what we see in this image. This also explains why in the surface at the bottom we have nearly zero as the stress.

Vertical Reaction Force

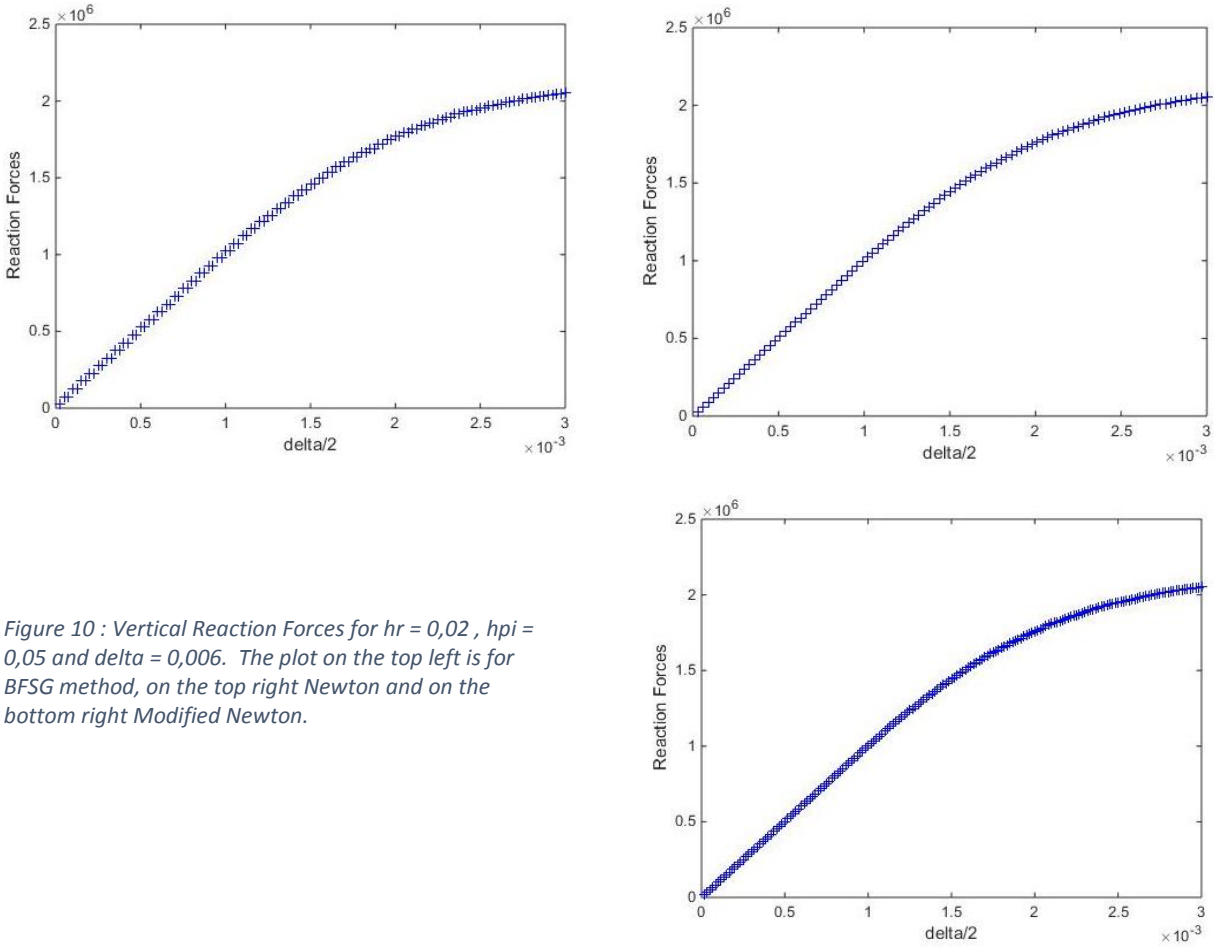

Figure 10 : Vertical Reaction Forces for hr = 0,02 , hpi = 0,05 and delta = 0,006. The plot on the top left is for BFSG method, on the top right Newton and on the bottom right Modified Newton.

As we can see, just as for the Stress field plot, the Vertical reaction force graphics are the same for all three methods. This helps us to confirm that it is correct. Moreover, as we can see, for the zone where we have elasticity, i.e. the curve is linear, it corresponds to the one obtained in question 1 / 2 for the same mesh size. After this domain, we can clearly see that we enter in the plasticity zone.

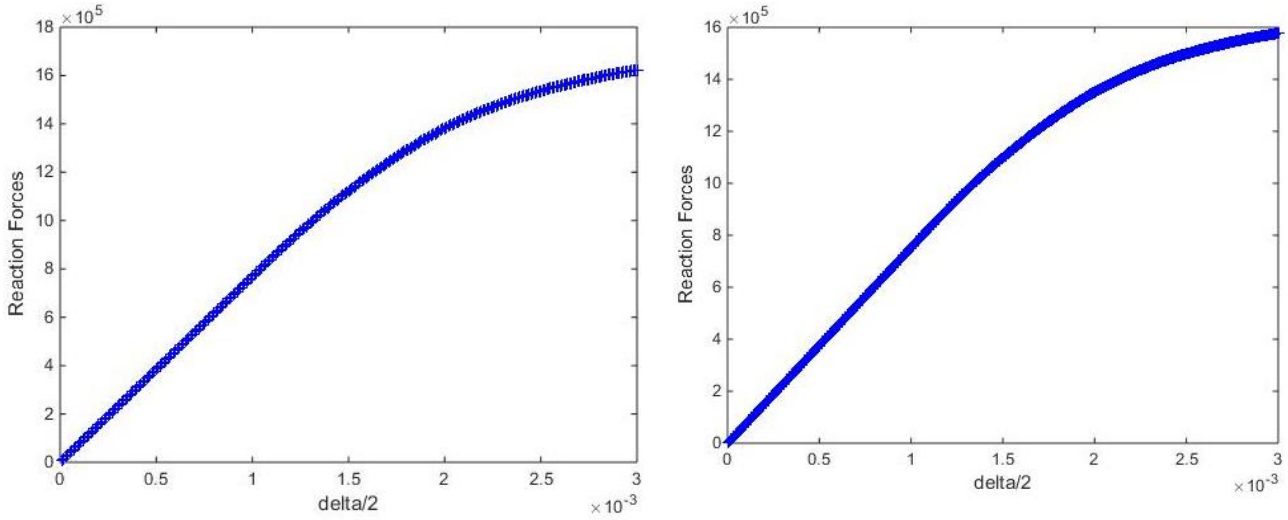

Figure 11 : Vertical Reaction Force for hr = 0, 01, hpi = 0, 01 and delta = 0,006 to the left and Force for hr = 0, 007, hpi = 0, 007 and delta = 0,006 to the right.

As we saw for questions one and two, the Reaction Forces reduce as we reduce the element size in the mesh. Nevertheless, the elastic behaviour is in the same domain of delta/2 for the plot with the medium size mesh. Moreover, the forces in the elastic domain are the same as the ones found for the questions one and two. As we can see, the Reaction Force curve will converge until it reaches approximately[4] the shape and values found for a mesh corresponding to hr = 0, 01, hpi = 0, 01. We can assume the latter as being a mesh sufficiently fine which would allow us to obtain good results, i.e. find the proper reaction force in case of a real life application.

Convergence

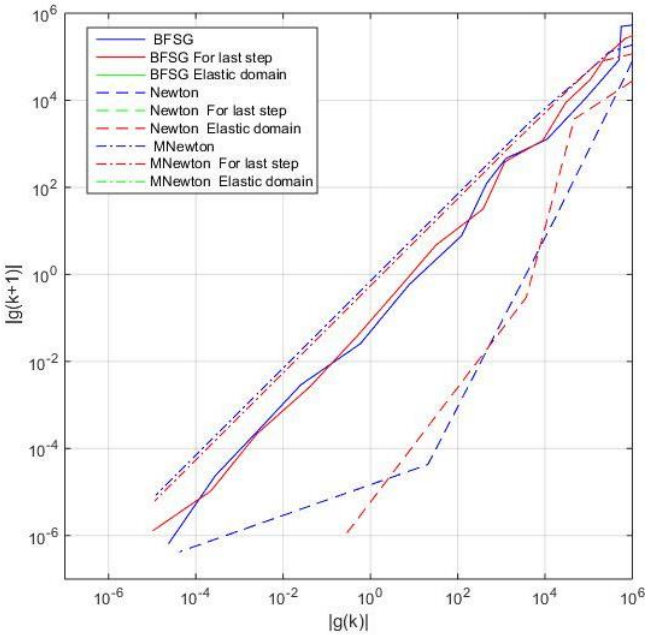

Figure 12 : Convergence plot for hr = 0, 02, hpi = 0, 05 and delta = 0,006. The blue curve is the load step which needed the highest number of iterations.

[4] If we look at the values obtained at question one and two at the linear domain, we see that the reaction forces corresponding to delta/2 = 1 for both hr = 0, 003, hpi = 0, 003 and hr = 0, 005, hpi = 0, 005 are extremely similar to the ones obtained now with hr = 0, 01, hpi = 0, 01.

As we can see, the curve corresponding to the Modified Newton method is linear (q = 1.03), as expected. The one for the Newton method converge faster than the Modified Newton and, in the zone between $10^2 - 10^6$ $q = 1{,}9$ (blue curve) which corresponds to a quadratic convergence. For the BFGS we can see that the convergence changes with every interaction, which is expected as C will not be constant. Moreover we can see that its convergence rate is between the Newton and Modified Newton methods convergence rate, as expected.

If we know focus on how much time it took to convergence, it took 110.238s to converge using the Modified Newton (with 100 load steps), 29.139s using the BFGS (with 120 steps) and 27.703s for the Newton (with 180 steps). This shows us that the Modified Newton is the slowest method, which is acceptable, as is has the slowest rate of convergence (linear). The BFGS just took a slightly longer time than the Newton, which is expected as the convergence rate for Newton faster than for BFGS. Although that difference is small, we used a much higher number of load steps for the Newton method, which should explain why the difference between both methods wasn't bigger.

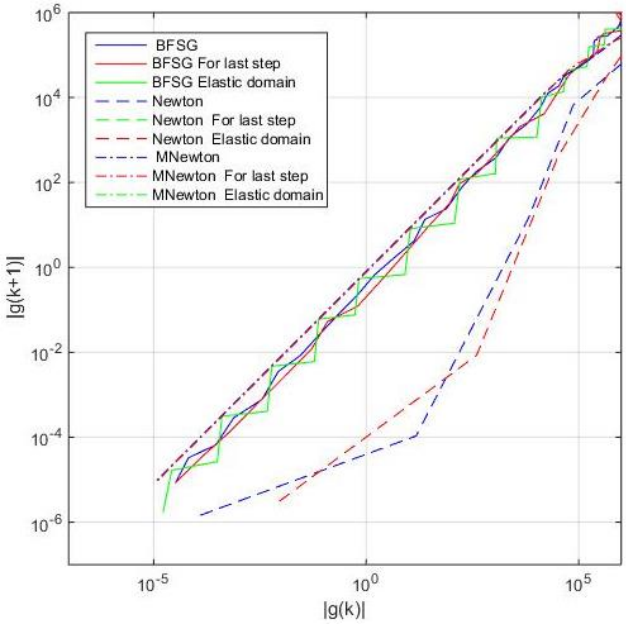

Figure 13 : Convergence plot for hr = 0, 01, hpi = 0, 01 and delta = 0,006.

As we can see, the behaviour is similar to what we saw in the plot for the medium size mesh. The value q calculated for the Modified Newton is $q = 1.00194$ and for the Newton $q = 1.9273$ (blue curve - in the domain 10^1 to 10^5). The BFGS curve for the elastic domain (green) is acceptable since we can see that the q is near one[5] and the steps represent the variation of the value of C. Moreover, the values obtained for q for the Newton and Modified Newton shows that the finer the mesh, better the result will be, i.e. will respect the respectively quadratic and linear convergence rates.

When look at how much time the program spent in each method, we obtain:

- 5502 s for the Modified Newton
- 1021.574 s for the BFGS
- 610.937 s for the Newton

[5] If we approximate this curve to a first degree polynomial equation, the coefficient A, as in y = Ax + C, would be one and C changes with each iteration.

Laderda Antonio
Brousseau Paul

This is in accordance with what we found for a medium mesh and for what we expected to obtain.

Laderda Antonio
Brousseau Paul

Bibliography

Larsson, F. (2010). Non-Linear finite element analysis - VSM014. Göteborg, Sweden.

Laderda Antonio
Brousseau Paul

Appendix

Question 1 and 2

```
close all
clear all
clc


% Defining constants
% Ri    - Inner radius of the ring
% Ro    - Outer radius of the ring
% elemtype - Element type    "tria3"  - 3-node triangular element
%                            "quad4" - 4-node quadrilateral element
% hr    - Approximate element size in radial direction
% hphi  - Approximate element size in circumferencial direction

Ri  = 0.15;
Ro = 0.25 ;
elemtype = 'tria3';
hr = 0.1;
hphi = 0.1;
E = 210e9 ;
ndelta = 100;
maxdelta = 0.008;
mindelta = 0.001;
poisson = 0.3;
sigyield = 220e6;
t = 0.5;


% Here we plot the mesh

[Edof,Ex,Ey,B1,B2,B3,B4,P1,P2,P3,P4]=ringmesh(Ri,Ro,elemtype,hr,hphi);

% number of elements

nelement = size (Ex,1);

% type 1   - axysimmetric

D = hooke(2,E,poisson);


% Calculating the boundary Conditions
% The boundary condiions for B1 - moves - we consider here y not moving,
% symmetry, thats why we take only the values for y degrees of freedom
% now we do the same thing for the other side
% The boundary condiions for B3 - doesn't move - y doesnt move
% we gotta also say that the x doesn't moves, meaning
% calculate K for all elements


deltavect = linspace(mindelta,maxdelta,ndelta); % Displacement vector

% We do all that until we have max stress above yield stress
```

```
gamma = 1;

sigmax = 0;

% Vertical reaction

V = zeros (ndelta,1);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while (sigyield + 100e6) > sigmax


    delta = deltavect (gamma);

    % find the dofs for the constrained noeuds

    cdofs = [B1(:,2);B3(:,2);P4(1,1)];

    % set the bc's

    bc = zeros(length(cdofs),2);
    bc(:,1) = cdofs;
    bc(1:size(B1,1),2) = -delta/2 ;

    % find the free dofs

    ndofs = max(max(Edof));

    fdofs = 1:ndofs;

    fdofs(cdofs) = [];

    % We remove values existing in dofs that match the ones existing in cdofs

    K = zeros (ndofs,ndofs);
    F = zeros (ndofs,1);

    % we will here do the partitioning of K

    for n = 1:nelement

        % the vector with all dofs associated with elements

        elemdofs = Edof(n,2:end);

        ep = [1,t];
        eq = [0;0];

        % retrieving K and Fe associated with each element

        [ke,fe]=plante(Ex(n,:),Ey(n,:),ep,D,eq);

        % Assembling everything

        K(elemdofs,elemdofs) = K(elemdofs,elemdofs) + ke;
```

```
        F(elemdofs) = F(elemdofs) + fe;


    end


    % Here we recover Kff

    Kff = K(fdofs,fdofs);

    % Here we recover Kfc

    Kfc = K(fdofs,cdofs);

    % Here we recover kcf

    Kcf = K(cdofs,fdofs);

    % here we define Kcc

    Kcc = K(cdofs,cdofs);

    A = zeros(ndofs,1);

    A(B1(:,2)) = -delta/2;

    % Here we calculate the displacement for the free node

    A(fdofs) = (Kff)\(F(fdofs) - Kfc*bc(:,2));
    Q=K*A;
    Ed = extract (Edof,A);


    % here we calculate the element stress  matrix and element strain matrix

    sigVM = zeros(nelement,1);
    sigmaxvect = zeros(nelement,1);

    for l = 1:nelement

        % the vector with all dofs associated with elements

        elemdofs = Edof(l,2:end);
        ep = [1,t];
        eq = [0;0];

        % retrieving es and et associated to each eple

        [es,et]=plants(Ex(l,:),Ey(l,:),ep,D,Ed(l,:));

        % our stress matrix will be

        S = [es(:,1) es(:,4) 0;
            es(:,4) es(:,2) 0;
            0 0 es(:,3)];

        sigmaxvect(l) = max(max(S));
```

```
        % here we define the mean stress
        meanstress = (1/3)*trace(S);

        % we calculate the Deviatoric stress

        sigdev = S - meanstress*eye(3,3);

        % we define the effective stress - Von Mises

        sigVM(Edof(l,1)) = sqrt(3/2)*norm(sigdev);

        ny = [0;1;0];




    end

    % we calculate here the vertical reaction

    ty = Q(B1(:,2));
    % to get vertical reaction force
    V(gamma) = sum(ty);
    sigmax = max(sigmaxvect);


    gamma = gamma + 1;


end

% Removing if zero

V(V==0)=[];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plotting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(1)


plotpar = [1 4 1];
eldisp2(Ex,Ey,Ed,plotpar,100)



figure (2)


fill (Ex',Ey',sigVM');
shading flat
colorbar
colormap(jet);

figure (3)
```

```
plot(deltavect(1:size(V,1))/2,V(:,1),'-b');


xlabel('delta/2');
ylabel('Reaction Forces');
```

Published with MATLAB® R2014b

Question 3, 4 and 5

```
close all

% We will calculate using the three methods and plot the Convergence Curve

TOL = 10e-6;

[ test1xlastBFSG,test1ylastBFSG,test1xBFSG,test1yBFSG,test1xelasticBFSG,test1yelasticBFSG ] =
HP_BFGS( );

[ test1xlastN,test1ylastN,test1xN,test1yN,test1xelasticN,test1yelasticN ] = HP_Newton( );

[ test1xlastMN,test1ylastMN,test1xMN,test1yMN,test1xelasticMN,test1yelasticMN ] =
HP_Modified_Newton(  );


figure (11)


% We will plot using log scale

loglog(test1xBFSG,test1yBFSG,'-b');


% We will force both axis to be proportional, which is not achieveable by
% using equal axis when ploting with loglog

xLimits = [TOL*10e-3 10e5];
yLimits = [TOL*10e-3 10e5];
axes_equal_loglog (gca,xLimits,yLimits)

xlabel('|g(k)|');
ylabel('|g(k+1)|');

hold on

loglog(test1xlastBFSG,test1ylastBFSG,'-r');
loglog(test1xelasticBFSG,test1yelasticBFSG,'-g');

loglog(test1xN,test1yN,'--b');
loglog(test1xelasticN,test1yelasticN,'--g');
loglog(test1xlastN,test1ylastN,'--r');

loglog(test1xMN,test1yMN,'-.b');
loglog(test1xlastMN,test1ylastMN,'-.r');
loglog(test1xelasticMN,test1yelasticMN,'-.g');
```

```
grid on

legend(' BFSG  ','BFSG For last step  ','BFSG Elastic domain',...
    'Newton ','Newton  For last step ','Newton  Elastic domain',...
    ' MNewton ','MNewton  For last step ','MNewton  Elastic domain');

hold off
```

```
function [ test1xlast,test1ylast,test1x,test1y,test1xelastic,test1yelastic ] = HP_BFGS( )
```

```
% Defining constants
% Ri    - Inner radius of the ring
% Ro    - Outer radius of the ring
% elemtype - Element type    "tria3"  - 3-node triangular element
%                            "quad4" - 4-node quadrilateral element
% hr    - Approximate element size in radial direction
% hphi  - Approximate element size in circumferencial direction

Ri  = 0.15;
Ro = 0.25 ;
elemtype = 'tria3';
hr = 0.1;
hphi = 0.1;
E = 210e9 ;
delta = 0.006;
poisson = 0.3;
sigyield = 220e6;
t = 0.5;


nsteps = 120; % number of steps

G = E/(2*(1+poisson));
K = E/(3*(1-2*poisson));

% Here we plot the mesh

[Edof,Ex,Ey,B1,~,B3,~,~,~,~,P4]=ringmesh(Ri,Ro,elemtype,hr,hphi);

% number of elements

nelement = size (Ex,1);


ndofs = max(max(Edof));

AoF = zeros (ndofs,1); % Initial guess for a displacement

TOL =10e-6; % Tolerance


% Dummy D

D = ones (4);

%  find the dofs for the constrained noeuds
```

```
cdofs = [B1(:,2);B3(:,2);P4(1,1)];

%  find the free dofs
fdofs = 1:ndofs;

fdofs(cdofs) = [];

% Identity matrix in voigt notation

Ivoigt = [1;
    1;
    1;
    0;
    0;
    0];


% 3x3 Identity matrix - voigt notation

Eyevoigtdev = [2/3 -1/3 -1/3 0 0 0;
    -1/3 2/3 -1/3 0 0 0;
    -1/3 -1/3 2/3 0 0 0;
    0 0 0 1/2 0 0;
    0 0 0 0 1/2 0;
    0 0 0 0 0 1/2;];

EyevoigtdevC = [2/3 -1/3 -1/3 0 0 0;
    -1/3 2/3 -1/3 0 0 0;
    -1/3 -1/3 2/3 0 0 0;
    0 0 0 2 0 0;
    0 0 0 0 2 0;
    0 0 0 0 0 2];




KS = spalloc(ndofs,ndofs,10*15*ndofs);
KTJ = spalloc(ndofs,ndofs,10*15*ndofs);

AoF_1 = zeros (ndofs,1);

% The reaction forces

V = zeros(nsteps,1);

% TO save the values of g
% I won't... probably, make plus than 500 iterations per step...
gmatrix = zeros(500,nsteps);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation - BFGS Method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
for step = 1:nsteps

    i = 1;

    g = ones (size(fdofs,2),1) ;
    g_1 = zeros (size(fdofs,2),1) ; % for the BFGS method

    % Input the boundary conditions

    AoF(B1(:,2),1) = -(delta/nsteps*step)/2;

    AoF(B3(:,2),1) = 0;

    AoF(P4(1,1),1) = 0;


    if step > 1

        AoF = AoF + (AoF - AoF_1);

    end

    KTJ = spalloc(ndofs,ndofs,10*15*ndofs);

    while norm(g) > TOL

        if i == 1
            g = zeros (size(fdofs,2),1) ;
        end

        KS = spalloc(ndofs,ndofs,10*15*ndofs);

        EdF = extract(Edof,AoF);

        sigVM = zeros(nelement,1);

        for l = 1:nelement
            % the vector with all dofs associated with elements

            elemdofs = Edof(l,2:end);

            ep = [2,t];

            eq = [0;0];

            % retrieving es and et associated to each eple

            [~,et]=plants(Ex(l,:),Ey(l,:),ep,D,EdF(l,:));

            % Deviatoric strain - We use et as it has to be in voigt notation
            % Change et to six dim,

            etvoigt = [et(:,1);
                et(:,2);
                et(:,3);
                et(:,4);
                0;
                0];
```

```
            Etadeviatoric = Eyevoigtdev*etvoigt ;

            % the Von mises eta

            Etamises = sqrt(2/3)*sqrt(etvoigt'*Eyevoigtdev*etvoigt);

            % We will now define 1- G we will be using
            %                    2- Continuum tangent stifness Dt


            etacondition = sigyield/(3*G);

            if etacondition < Etamises
                Gstar =  sigyield/(3*Etamises);

                % The Dc

                Dt = 2*Gstar*Eyevoigtdev - Etadeviatoric*...
                    (4*sigyield/(9*(Etamises^3)))*Etadeviatoric'...
                    + K*(Ivoigt*Ivoigt');
            else
                Gstar = G;

                % The Dc

                Dt = 2*Gstar*Eyevoigtdev + K*(Ivoigt*Ivoigt');
            end

            % We will define here the Ds

            Ds = 2*Gstar*Eyevoigtdev + K*(Ivoigt*Ivoigt');

            % We will now compute sigma

            sigma = Ds*etvoigt;

            sigVM(Edof(l,1)) = sqrt(3/2)*sqrt(sigma'*EyevoigtdevC*sigma);

            % We will now calculate Ks

            [Ks,~]=plante(Ex(l,:),Ey(l,:),ep,Ds,eq);

            KS(elemdofs,elemdofs) = KS(elemdofs,elemdofs) + Ks;

            % we now compute the Jacobian

            switch i
                case 1

                    [KtJ,~]=plante(Ex(l,:),Ey(l,:),ep,Dt,eq);
                    KTJ(elemdofs,elemdofs) = KTJ(elemdofs,elemdofs) + KtJ;
                otherwise

            end

        end
```

```
        % solving the g(af) and we assume Ff to be zero

        Fint = KS*AoF;

        g_1 = g;
        g = Fint(fdofs);
        gmatrix(i,step) = norm(g);



        if norm(g) < TOL
            % To Calculate the reaction forces
            Q = KS*AoF;

            V(step) = -sum(Q(B1(:,2)));

            % if we want to know where we stopped to add values to gmatrix
            % ( since the converging value migh be so small that we can
            % count on properly differentiating it from 0)
            % by using the sum we can be sure to know that there is no
            % conflict and that it will be the maximum value, i.e. we won't
            % by an accident choose a value that is
            % already there

            gmatrix (i+1,step) = sum(gmatrix(1:i,step));
            break

        end

        %In order to update the Jacobian for the BFGS

        if i>1

            deltag = g - g_1;

            deltax = -KTJ(fdofs,fdofs)\g_1;

            KTJ(fdofs,fdofs) = J_1 + (deltag*deltag')/(deltag'*deltax) - ...
                ((J_1*deltax)*(J_1*deltax)')/(deltax'*J_1*deltax);



        end

        J_1 = KTJ(fdofs,fdofs);

        AoF(fdofs) = AoF(fdofs) - KTJ(fdofs,fdofs)\g;

        norm(g)

        i = i+ 1

    end

    AoF_1 = AoF;

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOTING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure (2)


fill (Ex',Ey',sigVM');
shading flat
colorbar
colormap(jet);

figure(3)

plot(linspace((delta/(2*nsteps)),delta/2,nsteps),V(:,1),'b+');


xlabel('delta/2');
ylabel('Reaction Forces');

figure (4)

% we define the x and y axis, y being g(k+1) and x being g(k)


% We find the one who has the biggest amount of points

posMAXtest1x = 0;
test1x = 0;
stepmax = 0;
for gamma = 1:nsteps
    test = gmatrix (:,gamma);
    posMAXx =find(test == max(test));

    if posMAXx > posMAXtest1x
        posMAXtest1x = posMAXx;
        test1x = test;
        stepmax = gamma;
    end

end



% we use the previouus vector to define the vector on the y axis
test1y = test1x(2:(posMAXtest1x-1));

% we remove the unwanted values from test1x
test1x((posMAXtest1x-1):end) = [];

% we will plot for the last step
test1xelastic = gmatrix (:,1);

% we find the number of iterations that were used
posMAXtest1xelastic =find(test1xelastic == max(test1xelastic));

% we use the previouus vector to define the vector on the y axis
```

```
test1yelastic = test1xelastic(2:(posMAXtest1xelastic-1));

% we remove the unwanted values from test1x
test1xelastic((posMAXtest1xelastic-1):end) = [];

% we will plot for the last step
test1xlast = gmatrix (:,nsteps);

% we find the number of iterations that were used
posMAXtest1xlast =find(test1xlast == max(test1xlast));

% we use the previouus vector to define the vector on the y axis
test1ylast = test1xlast(2:(posMAXtest1xlast-1));

% we remove the unwanted values from test1x
test1xlast((posMAXtest1xlast-1):end) = [];

% Setting the axis as equals - gca is the current axis handle



loglog(test1x,test1y,'-g');

xLimits = [TOL*10e-3 1];
yLimits = [TOL*10e-3 1];
axes_equal_loglog (gca,xLimits,yLimits)

xlabel('|g(k)|');
ylabel('|g(k+1)|');

hold on

% Setting the axis as equals - gca is the current axis handle

loglog(test1xlast,test1ylast,'-s');
loglog(test1xelastic,test1yelastic,'-r');

grid on
legend(strcat('For step = ',num2str(stepmax)),strcat(' For last step = ',...
    num2str(nsteps)),'Elastic domain');

hold off
```

```
end
```

```
function [ test1xlast,test1ylast,test1x,test1y,test1xelastic,test1yelastic ] =
HP_Modified_Newton(  )
```

```
% Defining constants
% Ri    - Inner radius of the ring
% Ro    - Outer radius of the ring
% elemtype - Element type    "tria3"  - 3-node triangular element
%                            "quad4" - 4-node quadrilateral element
% hr    - Approximate element size in radial direction
% hphi  - Approximate element size in circumferencial direction
```

```
Ri  = 0.15;
Ro = 0.25 ;
elemtype = 'tria3';
hr = 0.1;
hphi = 0.1;
E = 210e9 ;
delta = 0.006;
poisson = 0.3;
sigyield = 220e6;
t = 0.5;

nsteps = 300; % number of load steps

G = E/(2*(1+poisson));
K = E/(3*(1-2*poisson));

% Here we plot the mesh

[Edof,Ex,Ey,B1,~,B3,~,~,~,~,P4]=ringmesh(Ri,Ro,elemtype,hr,hphi);

% number of elements

nelement = size (Ex,1);

% number of degrees of freedom

ndofs = max(max(Edof));

AoF = zeros (ndofs,1); % Initial guess for a displacement

TOL = 10e-6; % Tolerance


% Dummy D

D = ones (4);

%  find the dofs for the constrained noeuds

cdofs = [B1(:,2);B3(:,2);P4(1,1)];

%  find the free dofs

fdofs = 1:ndofs;

fdofs(cdofs) = [];

% Identity matrix in voigt notation

Ivoigt = [1;
    1;
    1;
    0;
    0;
    0];


% 3x3 Identity matrix - voigt notation
```

```
Eyevoigtdev = [2/3 -1/3 -1/3 0 0 0;
    -1/3 2/3 -1/3 0 0 0;
    -1/3 -1/3 2/3 0 0 0;
    0 0 0 1/2 0 0;
    0 0 0 0 1/2 0;
    0 0 0 0 0 1/2;];

EyevoigtdevC = [2/3 -1/3 -1/3 0 0 0;
    -1/3 2/3 -1/3 0 0 0;
    -1/3 -1/3 2/3 0 0 0;
    0 0 0 2 0 0;
    0 0 0 0 2 0;
    0 0 0 0 0 2];



KS = zeros(ndofs,ndofs);
KTJ = zeros(ndofs,ndofs);

% The reaction forces

V = zeros(nsteps,1);

% TO save the values of g
% I won't... probably, make plus than 500 iterations per step...
gmatrix = zeros(500,nsteps);


AoF_1 = zeros (ndofs,1);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation - Modified Newton Method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for step = 1:nsteps

    i = 1;

    g = ones (size(fdofs,1),1) ;

    % Input the boundary conditions

    AoF(B1(:,2),1) = -(delta/nsteps*step)/2;

    AoF(B3(:,2),1) = 0;

    AoF(P4(1,1),1) = 0;

    KTJ = zeros(ndofs,ndofs);

    % To obtain a better guess

    if step > 1

        AoF(fdofs) = 2*AoF(fdofs)  - AoF_1(fdofs);
```

```
end

AoF_1 = AoF;


while norm(g) > TOL



    KS = zeros(ndofs,ndofs);

    EdF = extract(Edof,AoF);

    sigVM = zeros(nelement,1);

    for l = 1:nelement

        % the vector with all dofs associated with elements
        elemdofs = Edof(l,2:end);

        ep = [2,t];

        eq = [0;0];

        % retrieving es and et associated to each eple

        [~,et]=plants(Ex(l,:),Ey(l,:),ep,D,EdF(l,:));

        % Deviatoric strain - We use et as it has to be in voigt notation
        % Change et to six dim,

        etvoigt = [et(:,1);
            et(:,2);
            et(:,3);
            et(:,4);
            0;
            0];

        Etadeviatoric = Eyevoigtdev*etvoigt ;

        % the Von mises

        Etamises = sqrt(2/3)*sqrt(etvoigt'*Eyevoigtdev*etvoigt);

        % We will now define 1- G we will be using
        %                    2- Continuum tangent stifness Dt


        etacondition = sigyield/(3*G);

        if etacondition < Etamises
            Gstar =  sigyield/(3*Etamises);

            % The Dc

            Dt = 2*Gstar*Eyevoigtdev - Etadeviatoric*...
                (4*sigyield/(9*(Etamises^3)))*Etadeviatoric'...
```

```
                + K*(Ivoigt*Ivoigt');
        else
            Gstar = G;

            % The Dc

            Dt = 2*Gstar*Eyevoigtdev + K*(Ivoigt*Ivoigt');
        end

        % We will define here the Ds

        Ds = 2*Gstar*Eyevoigtdev + K*(Ivoigt*Ivoigt');

        % We will now calculate Ks

        [Ks,~]=plante(Ex(l,:),Ey(l,:),ep,Ds,eq);

        KS(elemdofs,elemdofs) = KS(elemdofs,elemdofs) + Ks;

        % We will now compute sigma

        sigma = Ds*etvoigt;

        sigVM(Edof(l,1)) = sqrt(3/2)*sqrt(sigma'*EyevoigtdevC*sigma);

        % we now compute the Jacobian

        switch i
            case 1
                [KtJ,~]=plante(Ex(l,:),Ey(l,:),ep,Dt,eq);
                KTJ(elemdofs,elemdofs) = KTJ(elemdofs,elemdofs) + KtJ;
                % there is a initialization problem, the first iteration is
                % not enough because of it. It convernges if we calculate KTJ up to
                % the second iteration
            case 2
                [KtJ,~]=plante(Ex(l,:),Ey(l,:),ep,Dt,eq);
                KTJ(elemdofs,elemdofs) = KTJ(elemdofs,elemdofs) + KtJ;
            otherwise
        end


    end

    % solving the g(af) and we assume Ff to be zero

    Fint = KS*AoF;


    g = Fint(fdofs);
    gmatrix(i,step) = norm(g);

    %In order to update the Jacobian for the BFGS


    if norm(g) < TOL

        Q = KS*AoF;
```

```
            V(step) = -sum(Q(B1(:,2)));

            % if we want to know where we stopped to add values to gmatrix
            %( since the converging value migh be so small that we can
            % count on properly differentiating it from 0)
            % by using the sum we can be sure to know that there is no
            % conflict and that it will be the maximum value, i.e. we won't
            % by an accident choose a value that is
            % already there

            gmatrix (i+1,step) = sum(gmatrix(1:i,step));
            break

        end

        AoF(fdofs) = AoF(fdofs) - KTJ(fdofs,fdofs)\g;

        norm(g)

        i = i+ 1


    end


end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOTING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure (8)


fill (Ex',Ey',sigVM');
shading flat
colorbar
colormap(jet);

figure(9)

plot(linspace((delta/(2*nsteps)),delta/2,nsteps),V(:,1),'b+');


xlabel('delta/2');
ylabel('Reaction Forces');

figure (10)

% we define the x and y axis, y being g(k+1) and x being g(k)


% We find the one who has the biggest amount of points

posMAXtest1x = 0;
test1x = 0;
stepmax = 0;
for gamma = 1:nsteps
```

```
    test = gmatrix (:,gamma);
    posMAXx =find(test == max(test));

    if posMAXx > posMAXtest1x
        posMAXtest1x = posMAXx;
        test1x = test;
        stepmax = gamma;
    end

end



% we use the previouus vector to define the vector on the y axis
test1y = test1x(2:(posMAXtest1x-1));

% we remove the unwanted values from test1x
test1x((posMAXtest1x-1):end) = [];

% we will plot for the last step
test1xelastic = gmatrix (:,1);

% we find the number of iterations that were used
posMAXtest1xelastic =find(test1xelastic == max(test1xelastic));

% we use the previouus vector to define the vector on the y axis
test1yelastic = test1xelastic(2:(posMAXtest1xelastic-1));

% we remove the unwanted values from test1x
test1xelastic((posMAXtest1xelastic-1):end) = [];

% we will plot for the last step
test1xlast = gmatrix (:,nsteps);

% we find the number of iterations that were used
posMAXtest1xlast =find(test1xlast == max(test1xlast));

% we use the previouus vector to define the vector on the y axis
test1ylast = test1xlast(2:(posMAXtest1xlast-1));

% we remove the unwanted values from test1x
test1xlast((posMAXtest1xlast-1):end) = [];

% Setting the axis as equals - gca is the current axis handle



loglog(test1x,test1y,'-g');

xLimits = [TOL*10e-3 1];
yLimits = [TOL*10e-3 1];
axes_equal_loglog (gca,xLimits,yLimits)

xlabel('|g(k)|');
ylabel('|g(k+1)|');

hold on
```

```
% Setting the axis as equals - gca is the current axis handle

loglog(test1xlast,test1ylast,'-s');
loglog(test1xelastic,test1yelastic,'-r');

grid on
legend(strcat('For step = ',num2str(stepmax)),strcat(' For last step = ',...
    num2str(nsteps)),'Elastic domain');

hold off

end

function [ test1xlast,test1ylast,test1x,test1y,test1xelastic,test1yelastic ] = HP_Newton( )

% Defining constants
% Ri    - Inner radius of the ring
% Ro    - Outer radius of the ring
% elemtype - Element type    "tria3"  - 3-node triangular element
%                            "quad4" - 4-node quadrilateral element
% hr    - Approximate element size in radial direction
% hphi  - Approximate element size in circumferencial direction

Ri  = 0.15;
Ro = 0.25 ;
elemtype = 'tria3';
hr = 0.1;
hphi = 0.1;
E = 210e9 ;
delta = 0.006;
poisson = 0.3;
sigyield = 220e6;
t = 0.5;

nsteps =350; % number of steps

G = E/(2*(1+poisson));
K = E/(3*(1-2*poisson));

% Here we plot the mesh
[Edof,Ex,Ey,B1,~,B3,~,~,~,~,P4]=ringmesh(Ri,Ro,elemtype,hr,hphi);

% number of elements

nelement = size (Ex,1);


ndofs = max(max(Edof));

AoF = zeros (ndofs,1); % Initial guess for a displacement

TOL = 10e-6; % Tolerance


% Dummy D

D = ones (4);

%  find the dofs for the constrained noeuds
```

```
cdofs = [B1(:,2);B3(:,2);P4(1,1)];

%  find the free dofs
fdofs = 1:ndofs;

fdofs(cdofs) = [];

% Identity matrix in voigt notation

Ivoigt = [1;
    1;
    1;
    0;
    0;
    0];


% 3x3 Identity matrix - voigt notation

Eyevoigtdev = [2/3 -1/3 -1/3 0 0 0;
    -1/3 2/3 -1/3 0 0 0;
    -1/3 -1/3 2/3 0 0 0;
    0 0 0 1/2 0 0;
    0 0 0 0 1/2 0;
    0 0 0 0 0 1/2];

EyevoigtdevC = [2/3 -1/3 -1/3 0 0 0;
    -1/3 2/3 -1/3 0 0 0;
    -1/3 -1/3 2/3 0 0 0;
    0 0 0 2 0 0;
    0 0 0 0 2 0;
    0 0 0 0 0 2];


KS = spalloc(ndofs,ndofs,10*15*ndofs);
KTJ = spalloc(ndofs,ndofs,10*15*ndofs);

% The reaction forces

V = zeros(nsteps,1);

% TO save the values of g
% I won't... probably, make plus than 500 iterations per step...
gmatrix = zeros(500,nsteps);


AoF_1 = zeros (ndofs,1);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation - Newton Method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for step = 1:nsteps

    i = 1;
```

```
g = ones (size(fdofs,1),1) ;

% Input the boundary conditions

AoF(B1(:,2),1) = -(delta/nsteps*step)/2;

AoF(B3(:,2),1) = 0;

AoF(P4(1,1),1) = 0;

% To obtain a better guess

if step > 1

    AoF(fdofs) = 2*AoF(fdofs)  - AoF_1(fdofs);

end

AoF_1 = AoF;

while norm(g) > TOL


    KS = spalloc(ndofs,ndofs,10*15*ndofs);
    KTJ = spalloc(ndofs,ndofs,10*15*ndofs);
    EdF = extract(Edof,AoF);

    sigVM = zeros(nelement,1);

    for l = 1:nelement
        % the vector with all dofs associated with elements

        elemdofs = Edof(l,2:end);

        ep = [2,t];

        eq = [0;0];

        % retrieving es and et associated to each eple

        [~,et]=plants(Ex(l,:),Ey(l,:),ep,D,EdF(l,:));

        % Deviatoric strain - We use et as it has to be in voigt notation
        % Change et to six dim,

        etvoigt = [et(:,1);
            et(:,2);
            et(:,3);
            et(:,4);
            0;
            0];


        Etadeviatoric = Eyevoigtdev*etvoigt ;

        % the Von mises eta
```

```
            Etamises = sqrt(2/3)*sqrt(etvoigt'*Eyevoigtdev*etvoigt);

            % We will now define 1- G we will be using
            %                    2- Continuum tangent stifness Dt


            etacondition = sigyield/(3*G);

            if etacondition < Etamises
                Gstar =  sigyield/(3*Etamises);

                % The Dc

                Dt = 2*Gstar*Eyevoigtdev - Etadeviatoric*Etadeviatoric'...
                    *(4*sigyield/(9*(Etamises^3))) ...
                    + K*(Ivoigt*Ivoigt');
            else
                Gstar = G;

                % The Dc

                Dt = 2*Gstar*Eyevoigtdev + K*(Ivoigt*Ivoigt');
            end

            % We will define here the Ds

            Ds = 2*Gstar*Eyevoigtdev + K*(Ivoigt*Ivoigt');

            % We will now compute sigma

            sigma = Ds*etvoigt;


            sigVM(Edof(l,1)) = sqrt(3/2)*sqrt(sigma'*EyevoigtdevC*sigma);

            % We will now calculate Ks

            [Ks,~]=plante(Ex(l,:),Ey(l,:),ep,Ds,eq);

            KS(elemdofs,elemdofs) = KS(elemdofs,elemdofs) + Ks;

            % we now compute the Jacobian


            [KtJ,~]=plante(Ex(l,:),Ey(l,:),ep,Dt,eq);
            KTJ(elemdofs,elemdofs) = KTJ(elemdofs,elemdofs) + KtJ;

        end

        % solving the g(af) and we assume Ff to be zero

        Fint = KS*AoF;

        % Saving the values in order to compare them afterwards

        g = Fint(fdofs);
        gmatrix(i,step) = norm(g);
```

```
        if norm(g) < TOL

            % To Calculate the reaction forces

            Q = KS*AoF;

            V(step) = -sum(Q(B1(:,2)));

            % if we want to know where we stopped to add values to gmatrix
            % ( since the converging value migh be so small that we can
            % count on properly differentiating it from 0)
            % by using the sum we can be sure to know that there is no
            % conflict and that it will be the maximum value, i.e. we won't
            % by an accident choose a value that is
            % already there

            gmatrix (i+1,step) = sum(gmatrix(1:i,step));
            break

        end



        AoF(fdofs) = AoF(fdofs) - KTJ(fdofs,fdofs)\g;

        norm(g)

        i = i+ 1


    end


end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOTING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure (5)


fill (Ex',Ey',sigVM');
shading flat
colorbar
colormap(jet);

figure(6)

plot(linspace((delta/(2*nsteps)),delta/2,nsteps),V(:,1),'b+');


xlabel('delta/2');
ylabel('Reaction Forces');

figure (7)
```

```
% we define the x and y axis, y being g(k+1) and x being g(k)


% We find the one who has the biggest amount of points

posMAXtest1x = 0;
test1x = 0;
stepmax = 0;
for gamma = 1:nsteps
    test = gmatrix (:,gamma);
    posMAXx =find(test == max(test));

    if posMAXx > posMAXtest1x
        posMAXtest1x = posMAXx;
        test1x = test;
        stepmax = gamma;
    end

end



% we use the previouus vector to define the vector on the y axis
test1y = test1x(2:(posMAXtest1x-1));

% we remove the unwanted values from test1x
test1x((posMAXtest1x-1):end) = [];

% we will plot for the last step
test1xelastic = gmatrix (:,1);

% we find the number of iterations that were used
posMAXtest1xelastic =find(test1xelastic == max(test1xelastic));

% we use the previouus vector to define the vector on the y axis
test1yelastic = test1xelastic(2:(posMAXtest1xelastic-1));

% we remove the unwanted values from test1x
test1xelastic((posMAXtest1xelastic-1):end) = [];

% we will plot for the last step
test1xlast = gmatrix (:,nsteps);

% we find the number of iterations that were used
posMAXtest1xlast =find(test1xlast == max(test1xlast));

% we use the previouus vector to define the vector on the y axis
test1ylast = test1xlast(2:(posMAXtest1xlast-1));

% we remove the unwanted values from test1x
test1xlast((posMAXtest1xlast-1):end) = [];

% Setting the axis as equals - gca is the current axis handle



loglog(test1x,test1y,'-g');
```

```
xLimits = [TOL*10e-3 1];
yLimits = [TOL*10e-3 1];
axes_equal_loglog (gca,xLimits,yLimits)

xlabel('|g(k)|');
ylabel('|g(k+1)|');

hold on

% Setting the axis as equals - gca is the current axis handle

loglog(test1xlast,test1ylast,'-s');
loglog(test1xelastic,test1yelastic,'-r');

grid on
legend(strcat('For step = ',num2str(stepmax)),strcat(' For last step = ',...
    num2str(nsteps)),'Elastic domain');

hold off

msg = 'If you are not able to see the plot, remove the xLimits and yLimits from
axes_equal_loglog '

end
```

```
function axes_equal_loglog ( hAxes,xLimits,yLimits )
%This Function replaces axes equal for loglog plot as the former doesn't
%present acceptaveble results when used with the latter.
%If you don't want to set the axis limits, input only hAxis.
%Example : axes_equal_loglog (gca) where gca the current axis handle

% In case I don't want to limit the axis
if (nargin < 2) || isempty(xLimits)
    xLimits = get(hAxes,'XLim');
    msg = 'xLimits may be empty';
    warning(msg);
end
if (nargin < 3) || isempty(yLimits)
    yLimits = get(hAxes,'YLim');
    msg = 'yLimits may be empty ';
    warning(msg);
end

% We need to consider both number and sizes of decades in the axis range
% Therefore, we calculate the average size for the decade of each axis and
% then using the ration of that average for both axis.

% i.e. diff(xLimits) total size of the x axis and diff(log10(xLimits) the
% number of decades

averagey = diff(yLimits)/diff(log10(yLimits));
averagex = diff(xLimits)/diff(log10(xLimits));

set(hAxes,'XLim',xLimits,'YLim',yLimits,'DataAspectRatio',[1 averagey/averagex 1]);


end
```

Laderda Antonio
Brousseau Paul

Published with MATLAB® R2014b