COMPUTATIONAL SOLID MECHANICS

HYPERELASTICITY

BY

JOHN HANNA

MSc. in Computational Mechanics
Universitat Politècnica de Catalunya

7^{th} *June*, 2019

Contents

List of Figures

1 General Introduction

Linear elasticity doesn't accurately describe the observed material behaviour under large deformation. Hyperelasticity provides a way to model materials that can't be accurately described using linear elasticity. An example of these materials is rubber or many biological tissues. Hyperelastic models account for both material nonlinearities and large shape changes. In general, the constitutive relation is obtained from a strain energy density function that provides a nonlinear relationship between stress and strain. In this report, some material models were developed and tested using simple examples. Buckling phenomenon is also studied and treated numerically using the line search algorithm along with Newton-Raphson.

2 Kirchhoff Saint-Venant material model

The simplest hyperelastic material model, which is just an extension of linear elasticity, is the Kirchhoff Saint-Venant material model. This model is implemented in the MATLAB routine to test the results obtained by hand calculations. For that a slender beam in compressed beyond $\lambda = 0.577$, which is the critical stretch ration beyond which the model fails. In order to implement the model, the strain energy, second Piola-Kirchhoff stress and the tangent modulus are calculated. The consistency test was done to check for the correctness of the implementation and the test was passed.

$$W = \frac{\lambda}{2}(tr\,E)^2 + \mu\, tr(E^2)$$

$$S = \frac{\partial W}{\partial E} = \lambda tr\,E\,I + 2\mu\,E$$

$$\mathbb{C} = \lambda\mathbb{I} + (2\mu)\hat{\mathbb{I}}$$

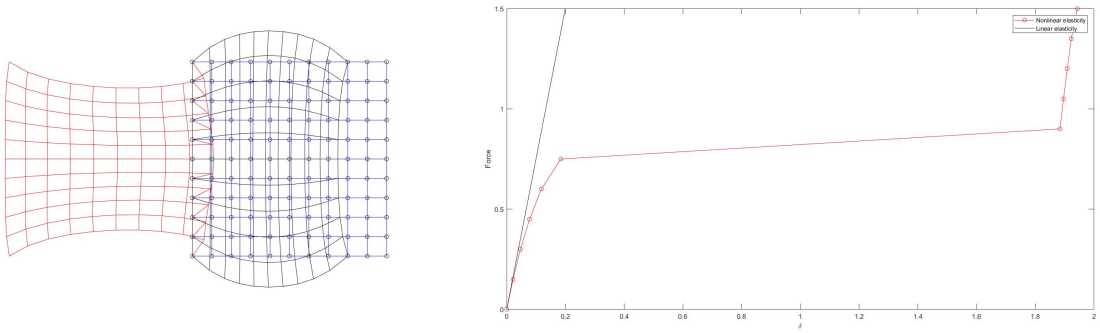


Figure 1: Deformed shape and force vs displacement curve for KSV model

The above results show the compression beyond the critical limit. As shown from the force-displacement curve, the material compressed normally in the beginning. After the limit was reached, instabilities started to take place and a flip of the material is obtained as expected.

3 Line-Search and Newton-Raphson Implementation

A structure under compression can suffer sudden deflection which is called buckling. It is an instability that leads to the failure of the structure. This may occur even though the stresses induced in the structure is not enough to cause failure. In a mathematical sense, buckling is a bifurcation of the solution of the equilibrium equation.

Numerically, Newton-Raphson algorithm might fail due to the convergence to saddle points instead of a point of minimum energy. Another method that can be used rather than Newton-Raphson is line search. The approach is basically finding a decent direction along which the objective funcion (energy) shall be reduced and then, define how far to go along that direction. The convergence of this method is guaranteed but its not used because it's slow. An incorporation of both methods can be beneficial where the step is calculated using Newton-Raphson and the line search algorithm decides the direction.

A buckling example is tested using both methods and the results are then compared. The example is an arch subjected to a dead load at the center. The material model used is the hyperelastic Neo-Hookean model. It is expected that the arch starts deforming till a sudden point where buckling takes place.

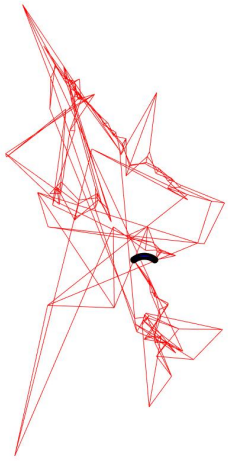


Figure 2: Deformed shape for arch using Newton-Raphson

The Newton-Raphson algorithm fails to capture the buckling behaviour of the arch. Before buckling, the algorithm converged with very small iteration numbers which is typical of Newton's method. However, once buckling is reached Newton's method couldn't minimize the energy function and the solution gets destroyed as can be seen in the above figure.

Mixing Newton-Rapshon with line search provides excellent results. The buckling behaviour was perfectly captured with a clear jump that can be seen in the force-deflection curve shown above. Thus, it should be concluded that Newton-Raphson with line search is an efficient algorithm to minimize the energy even in phenomenon that is hard to capture such as buckling.

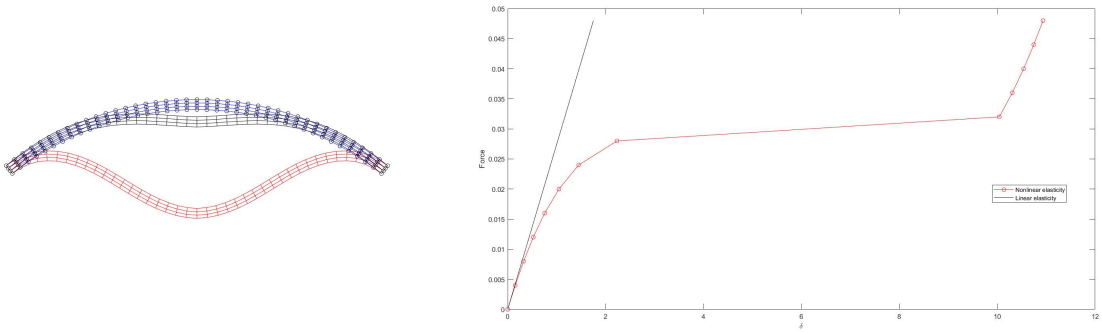


Figure 3: Deformed shape and force vs displacement curve for arch using line search Newton-Raphson

4 Transverse Isotropic Material Model

A new material model is implemented and tested. The model is transverse isotropic used for rubber reinforced by fibers. The fibers are all aligned in the same direction in such a way that perpendicular to the fibers, the material remains isotropic. The strain energy density function of the model is given as:

$$W = \frac{1}{2}\mu_0(tr\,C-3) - \mu_0\, ln\, J + \kappa G(J) + c_0\,(exp[c_1[\sqrt{I_4}-1]^4]-1)$$

where G is equal to

$$G = \frac{1}{4}(J^2 - 1 - 2ln\, J)$$

To implement the model, the stress as well as the tangent modulus need to be defined.

$$S = 2\frac{\partial W}{\partial C} = \mu I - \mu C^{-1} + \kappa(\frac{1}{2}J^2C^{-1} - 2C^{-1}) + \frac{4c_0c_1(exp[c_1[\sqrt{I_4}-1]^4]-1)[\sqrt{I_4}-1]^3N}{\sqrt{I_4}}$$

where $N = n \otimes\ n$, and n is the unit vector defining the direction of the fibers.

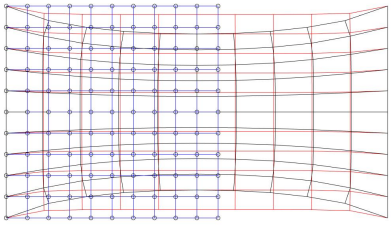
$$\mathbb{C}_{ijkl} = 2\frac{\partial S}{\partial C} = \mu(C^{-1}_{ij}C^{-1}_{kl} + C^{-1}_{ij}C^{-1}_{kl})$$

$$+\kappa(J^2(C^{-1}_{ij}C^{-1}_{kl}) - 0.5J^2(C^{-1}_{ij}C^{-1}_{kl} + C^{-1}_{ij}C^{-1}_{kl}) + 0.5(C^{-1}_{ij}C^{-1}_{kl} + C^{-1}_{ij}C^{-1}_{kl}))$$

$$+\ constant\ N_{ij}N_{kl}$$

where the constant is defined in the MATLAB code in the appendix

The model is tested on a square fixed from one side and under tension from the other side. The fibers are aligned in different ways and compared. The consistency test was done to check for the correctness of the implementation and the test was passed.



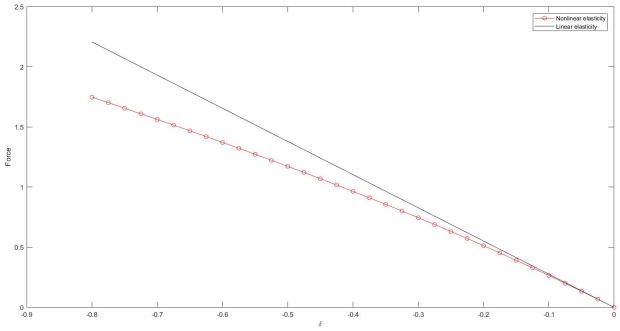
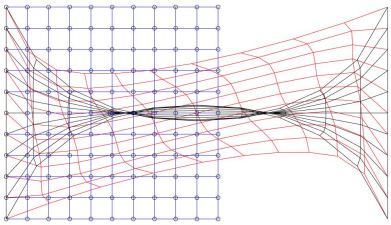


Figure 4: Deformed shape and force vs displacement curve for nonisotopic model ($\theta = 90$)



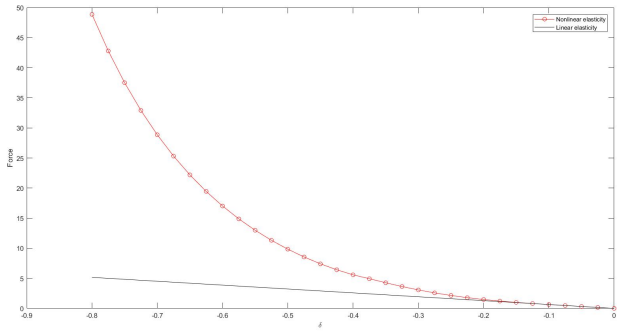


Figure 5: Deformed shape and force vs displacement curve for nonisotopic model ($\theta = 45$)

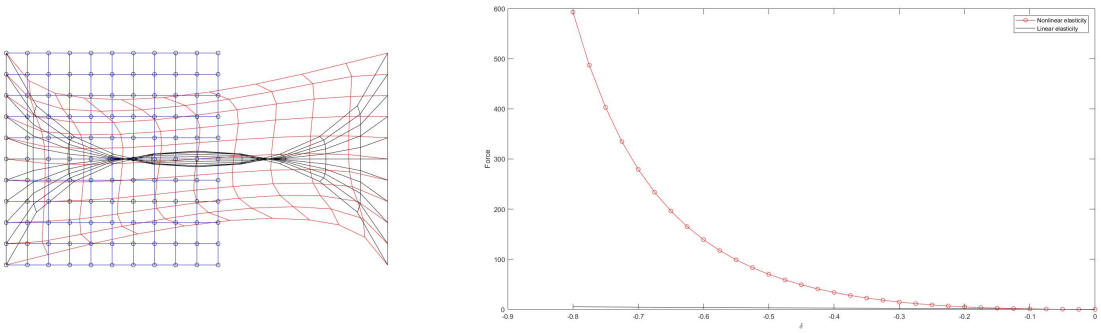


Figure 6: Deformed shape and force vs displacement curve for nonisotopic model ($\theta = 30$)

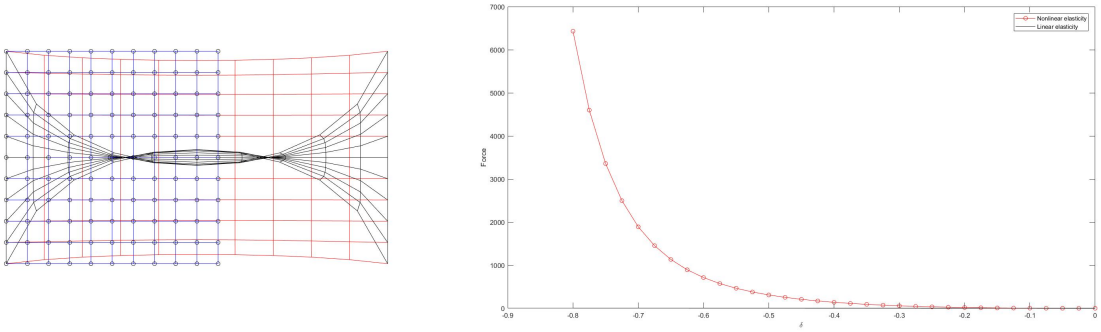


Figure 7: Deformed shape and force vs displacement curve for nonisotopic model ($\theta = 0$)

As can be noticed from the obtained results, the response depends greatly on the angle of the fibers. At $\theta = 0$, the fibers are aligned with the force which is very stiff resulting in a very steep exponential response. This steepness is reduced as θ increases, which can be seen in the results of $\theta = \frac{\pi}{6}, \frac{\pi}{4}$. At $\theta = \frac{\pi}{2}$, the fibers are perpendicular to the load direction, therefore, the response show that the material is soft if loaded in that direction.

5 Slender Beam Buckling

In this section, buckling of a slender beam under compression is being considered. The beam is positioned on an elastic floor which is modelled as elastic springs. The force is applied as imposed displacement compressing the beam till buckling starts. The elastic springs are added through modifying the energy, the energy gradient and the Hessian so that vertical force is added of the value (ky). The material used is the Neo-Hookean hyperelastic model. The consistency test was done to check for the correctness of the implementation and the test was passed.

Figure 8: Deformed shape for Slender beam ($k = 0.2,\ 0.001$)

Figure 9: Deformed shape for Slender beam ($k = 0.05,\ 0.1$)

As can be seen from the obtained results, the response and the obtained mode is highly dependant on the spring stiffness. Very small stiffness will lead to the 1st buckling mode (k=0.001). As the stiffness increases, higher modes are obtained. However, after certain value, buckling doesn't take place since the high stiffness springs don't allow negative displacements. These results are similar to a study by Casper H. L. Beentjes(1), where he obtained the first and second modes.

6 Appendix (MATLAB codes)

6.1 Kirchhoff Saint-Venant material model

```
function [W,S,CC] = transv_isotr_3(C,mod1)

trc = C(1,1)+C(1,2);

J = sqrt(C(1,1)*C(1,2)-C(1,3)*C(1,3));
G = (J^2-1-2*log(J))/4;
C_John = [C(1,1), C(1,3);C(1,3),C(1,2)];
inv_c = inv(C_John);
inv_voi = [inv_c(1,1) inv_c(2,2) inv_c(1,2)];

I4 = mod1.N_fib'*(C_John*mod1.N_fib);
expo = exp(mod1.c1*(sqrt(I4)-1)^4);

N = mod1.N_fib*mod1.N_fib';
N_voi = [N(1,1) N(2,2) N(1,2)];

W = (mod1.mu*(trc-2)/2) - (mod1.mu*log(J)) + (mod1.kappa*G) +
    mod1.c0*(expo-1);

s1 = mod1.mu*[1 1 0];
s2 = -mod1.mu*inv_voi;
s3 = mod1.kappa*((1/2)*(J^2)*inv_voi - (1/2)*inv_voi);
s4 = mod1.c0*expo*4*mod1.c1*((sqrt(I4)-1)^3)*(1/sqrt(I4))*N_voi
    ;

S = [];
CC=zeros(3);

S = s1+s2+s3+s4;

const = 8*mod1.c0*mod1.c1* ((((sqrt(I4)-1)^6)/I4)*2*expo*mod1.
    c1 + expo*1.5*((sqrt(I4)-1)^2)/I4 - 0.5*expo*((sqrt(I4)-1)
    ^3)/(sqrt(I4^3)));

CC(1,1) = mod1.mu*(inv_c(1,1)*inv_c(1,1)+inv_c(1,1)*inv_c(1,1))
    ....
     +mod1.kappa*((J^2)*(inv_c(1,1)*inv_c(1,1))-(J^2)*0.5*(inv_c
        (1,1)*inv_c(1,1)+inv_c(1,1)*inv_c(1,1))+0.5*(inv_c(1,1)*
        inv_c(1,1)+inv_c(1,1)*inv_c(1,1))) ....
     +const*N(1,1)*N(1,1);

CC(1,2) = mod1.mu*(inv_c(1,2)*inv_c(1,2)+inv_c(1,2)*inv_c(1,2))
    ....
     +mod1.kappa*((J^2)*(inv_c(1,1)*inv_c(2,2))-(J^2)*0.5*(inv_c
        (1,2)*inv_c(1,2)+inv_c(1,2)*inv_c(1,2))+0.5*(inv_c(1,2)*
```

```
        inv_c(1,2)+inv_c(1,2)*inv_c(1,2))) ....
37      +const*N(1,1)*N(2,2);
38
39  CC(1,3) = mod1.mu*(inv_c(1,1)*inv_c(1,2)+inv_c(1,2)*inv_c(1,1))
        ....
40      +mod1.kappa*((J^2)*(inv_c(1,1)*inv_c(1,2))-(J^2)*0.5*(inv_c
            (1,1)*inv_c(1,2)+inv_c(1,2)*inv_c(1,1))+0.5*(inv_c(1,1)*
            inv_c(1,2)+inv_c(1,2)*inv_c(1,1))) ....
41      +const*N(1,1)*N(1,2);
42
43  CC(2,2) = mod1.mu*(inv_c(2,2)*inv_c(2,2)+inv_c(2,2)*inv_c(2,2))
        ....
44      +mod1.kappa*((J^2)*(inv_c(2,2)*inv_c(2,2))-(J^2)*0.5*(inv_c
            (2,2)*inv_c(2,2)+inv_c(2,2)*inv_c(2,2))+0.5*(inv_c(2,2)*
            inv_c(2,2)+inv_c(2,2)*inv_c(2,2))) ....
45      +const*N(2,2)*N(2,2);
46
47  CC(2,3) = mod1.mu*(inv_c(2,1)*inv_c(2,2)+inv_c(2,2)*inv_c(2,1))
        ....
48      +mod1.kappa*((J^2)*(inv_c(2,2)*inv_c(1,2))-(J^2)*0.5*(inv_c
            (2,1)*inv_c(2,2)+inv_c(2,2)*inv_c(2,1))+0.5*(inv_c(2,1)*
            inv_c(2,2)+inv_c(2,2)*inv_c(2,1))) ....
49      +const*N(2,2)*N(1,2);
50
51  CC(3,3) = mod1.mu*(inv_c(1,1)*inv_c(2,2)+inv_c(1,2)*inv_c(2,1))
        ....
52      +mod1.kappa*((J^2)*(inv_c(1,2)*inv_c(1,2))-(J^2)*0.5*(inv_c
            (1,1)*inv_c(2,2)+inv_c(1,2)*inv_c(2,1))+0.5*(inv_c(1,1)*
            inv_c(2,2)+inv_c(1,2)*inv_c(2,1))) ....
53      +const*N(1,2)*N(1,2);
54
55  CC(2,1) = CC(1,2);
56
57  CC(3,1) = CC(1,3);
58
59  CC(3,2) = CC(2,3);
60
61
62  end
```

6.2 Line-Search and Newton-Raphson

```
    iter=0;
    err_x=100;
    err_f=100;
    [Ener,grad_E,Hess_E] = Ener_short(x_short,3);

    while (iter<=options.n_iter_max) & ...
        ( (err_x>options.tol_x) | ...
        (err_f>options.tol_f))
      iter=iter+1;
      dx = -Hess_E\grad_E;
      if dx'*grad_E <= 0
          p = dx;
      else
          p = -dx;
      end

      t = 1;

      opts=optimset('TolX',options.TolX,'MaxIter',options.
         n_iter_max_LS);

      t = fminbnd(@(t) Ener_1D(t,x_short,p,3),0,2,opts);

      x_short=x_short + t*p;

      [Ener,grad_E,Hess_E] = Ener_short(x_short,3);
      err_x=norm(t*p)/norm(x_short);
      err_f=norm(grad_E);
      err_plot=[err_plot err_x];
      err_plot1=[err_plot1 err_f];
      %fprintf('Iteration %i, errors %e %e \n', iter,err_x,
         err_f)
    end
    %Check positive definiteness
    if options.info==3
      [V,D] = eig(Hess_E);
      D=diag(D);
      if ((min(D))<=-1e-6*abs(max(D)))
          fprintf('Warning, the Hessian has a negative
             eigenvalue \n')
      end
    end


function Ener = Ener_1D(t,x_short,p,icode)

    global mod1 mesh1 load1 el1
```

```
45
46      x_short = x_short + t*p;
47
48      [x_long] = long(x_short);
49
50      [Ener,grad_E_l,Hess_E_l] = Energy(x_long,icode);
51  end
```

6.3 Transverse Isotropic model

```
1   function [W,S,CC]=Kirc_3(C,lambda,mu,icode)
2
3   E = 0.5*(C-[1 1 0]);
4   E2(1) = E(1)^2+E(3)^2;
5   E2(2) = E(3)^2+E(2)^2;
6   E2(3) = E(1)*E(3)+E(2)*E(3);
7
8   W = 1/2*lambda*(E(1)+E(2))^2 + mu*(E2(1)+E2(2)) ;
9
10  S = [];
11  CC=zeros(3);
12
13  S = lambda*(E(1)+E(2))*[1 1 0] + 2*mu*E;
14
15  CC(1,1)=lambda+2*mu;
16  CC(1,2)=lambda;
17  CC(1,3)=0;
18  CC(2,2)=lambda+2*mu;
19  CC(2,3)=0;
20  CC(3,3)=mu;
21  CC(2,1)=CC(1,2);
22  CC(3,1)=CC(1,3);
23  CC(3,2)=CC(2,3);
24  end
```

6.4 Slender beam buckling

```
1   posi = 2*(2:41);
2   k=0.2;
3   kmat=zeros(328);
4       for i = posi
5           kmat(i,i) = kmat(i,i)+k;
6       end
7
8       for i = posi
9           Hess_E(i,i) = Hess_E(i,i)+k;
10      end
11  Ener = Ener - load1.force'*x+ 0.5*(kmat*x)'*x;
12  grad_E = grad_E - load1.force + kmat*x ;
```