Assignment 3

COMPUTATIONAL SOLID MECHANICS

Marcos Boniquet Aparicio

1. *Kirchoff Saint-Venant material model*

Isotropic linear elasticity can be derived from balance of linear momentum, the linearized strain displacement relation $\varepsilon=½(\nabla u+\nabla u^T)$ and the stored elastic energy function

$$W(\varepsilon)=\lambda/2(tr\varepsilon)^2+\mu tr(\varepsilon^2)$$

1)Check that the stress tensor obtained from $\sigma=\partial W/\partial\varepsilon$ agrees with the usual linear elasticity expression

Since the linearization of the Green-Lagrange strain tensor $E=½(C-Id)$ is the small strain tensor it is natural to extend isotropic elasticity to nonlinear elasticity as

$$W(\mathbf{E})=\lambda/2(tr\mathbf{E})^2+\mu tr(\mathbf{E}^2)$$

This hyperelastic model is called Kirchhoff Saint-Venant material model.

2)According to the definition we gave in class about isotropy in nonlinear elasticity, is this model isotropic?

3) Derive the second Piola-Kirchhoff stress ***S***

Given that **S =2*dW/dC** and $W(\mathbf{C})=1/2\lambda_o(lnJ)^2-\mu_0 lnJ+1/2\mu_0(trace\mathbf{C}-3)$, *we use invariants and derive to obtain 2PK.*

$I_1(\mathbf{C})=trace\mathbf{C}$
$I_2(\mathbf{C})=½[(trace\mathbf{C})^2-trace\mathbf{C}^2]$
$I_3(\mathbf{C})=det\mathbf{C}=J^2$

$dI_1(\mathbf{C})/dC=Id$
$dI_2(\mathbf{C})/dC=I_1(\mathbf{C})Id-\mathbf{C}^T$
$dI_3(\mathbf{C})/dC=det\mathbf{C}=I_3(\mathbf{C})\mathbf{C}^{-T}$

thus
$d\ 1/2\lambda_o(lnJ)^2=lnJ\lambda_o\mathbf{C}^{-1}$
$d\ (\mu_0 lnJ/d\mathbf{C})=\mathbf{C}^{-1}$
$d\ (1/2\mu_0(trace\mathbf{C}-3))/d\mathbf{C}=1/2\mu_0 Id$ *(applying first invariant derivative)*

S=

4) For a uniform deformation of a rod aligned with the X axis ($x=\Lambda X$ $y=Y$, $z=Z$ where $\Lambda>0$ is the stretch ratio along the X direction) derive the relation between the nominal normal stress P (the xX component of the first Piola-Kirchhoff stress) and the stretch ratio Λ, $P(\Lambda)$, and plot it.

5) Is this relation $P(\Lambda)$ monotonic? If not, derive the critical stretch Λ_{crit} at which the model fails with zero stiffness. Does this critical stretch depend on the elastic constants? Show the material does not satisfy the growth conditions

$$W(\mathbf{E}) \rightarrow +\infty \text{ when } J \rightarrow 0^+$$

Discuss your answers.

6) Consider now the modified Kirchoff Saint-Venant material model:

$$W(\mathbf{E})=\lambda/2(\ln J)^2+\mu tr(\mathbf{E}^2)$$

Does this model circumvent the drawbacks of the previous model?

7) Implement the material model in Eq(1) in the MATLAB code. Perform the consistency test to check your implementation. Try to demonstrate the material instabilities of this model with a numerical example.

2. Implementation of line search

Implement a line-search algorithm to be used in combination with Newton's method. For this, I suggest you resort to Matlab's function ***fmindbd****, which performs 1D nonlinear minimization with bounds. You need to define Ener_1D that evaluates the energy along the line that passes through x in the direction of p (the descent search direction). The function Linesearch may include lines like the ones suggested next:*

```
t=1
opts=optimset('TolX',options.TolX,'MaxIter',options.n_iter_max_LS);
t=fminbnd(@(t) Ener_1D(t,x_short,p),0,2,opts);
x_short=x_short+t*p;
```

Test the code *with the examples where you expect buckling (the compression of the beams, or the deflection of the arch), and compare the results with and without linear search.*

3. Implementation of a material model

The code you are given implements a plane-strain finite element method for finite deformation elasticity. A compressible Neo-Hookean material is already in place (modeling a slightly porous rubber for instance), whose strain energy density (or hyper elastic potential) is:

$$W(\mathbf{C})=1/2\lambda_o(\ln J)^2-\mu_0\ln J+1/2\mu_0(\text{trace}\mathbf{C}-3)$$

This constitutive model is isotropic. Note that, since we are considering plane strain, we can use 2x2 reduced Cauchy-Green deformation tensor and replace trace C-3 by trace C-2 in the above equation.

We want to consider now an anisotropic material, more specifically, a transversely isotropic material. We consider a material constitutive law for a rubber reinforced by fibers, all aligned in the same direction in such a way that perpendicular to the fibers, the material remains isotropic. The orientation of the fibers is given in the reference configuration by a unit vector $\mathbf{N}^{fib}$*. Such a model depends on the principal invariants of* $\mathbf{C}$*, and additionally by the fourth invariant*

$$I_4(\mathbf{C})=\mathbf{N}^{fib}.\mathbf{C}.\mathbf{N}^{fib}$$

More specifically,

$$W(\mathbf{C})=\mu_0(\text{trace}\mathbf{C}-3)-\mu_0\ln J+kG(J)+c_0\{\exp[c_1(I_4(\mathbf{C})^{1/2}-1)^4]-1]\}$$

where μ_0, k ,c_0, *and* c_1 *are material parameters, and G(J) provides the volumetric response of the material. We consider*

$$G(J)= ¼(J^2-1-2\ln J)$$

The last term in the strain energy function specifies the contribution to the deformation energy of the fibers, and as typical in biological fibers, with this model these become stiffer the more deformed they are.

a) *Implement the material into the code. The code is prepared for this model (material=2), including the definition of the material parameters in preprocessing.m*

b) *Check the correctness (consistency test) of your implementation by running the script Ceck_Derivatives.m with material =2. This script checks the gradient of the energy (out-of-balance forces) and the Hessian of the energy (tangent stiffness matrix) by numerical differentiation. Check also that when solving a mechanical problem with this mode, Newton's method converges quadratically.*

c) *Solve example=0, a dead load applies on an elastic block in tension, with a few representative orientations of the fibers Consider* $\theta=\pi/4$; $\theta=\pi/6$ *and* $\theta=\pi/2$ *(fibers perpendicular to the loading direction), where,*

$$\mathbf{N}^{fib}=[\cos\theta,\ \sin\ \theta]^T$$

Explain the results forma mechanical viewpoint.

-ASSIGNMENT-

1.Kirchhoff Saint-Venant material model

1) Starting from $W(\varepsilon)=\lambda/2(tr\varepsilon)^2+\mu tr(\varepsilon^2)$

$dW/d\varepsilon=\lambda tr\varepsilon*Id+\mu*d(tr(\varepsilon^2))/d\varepsilon$

from derivative of second invariant, is induced that $d(tr(\varepsilon^2)/\varepsilon=2\varepsilon^T$

ledding to the expression:

$u=dW/d\varepsilon=\lambda tr\varepsilon*Id+2\mu\varepsilon^T$, usual expression for linear elasticity

2)Any isotropic strain energy function can be written in terms of the principal invariants

This is an Isotropic case, given that invariants led to the calculation of linear elasticity displacement

3) Starting from $W(E)=\lambda/2(tr(½(C-Id))^2+\mu tr(½(C-Id)^2)$

Applying sum of traces is traces of sum

$W(E)=\lambda/2(½trC-trC*Id)^2+\mu/2(trC-3)$
$W(E)=\lambda/2[¼(trC)^2-3trC+9]+\mu/2(trC-3)$

And now we proceed derivation:

$S=\lambda/2[¼*2*trC*I-3I]+\mu I/2$
$S=\lambda I/2[trC/2+\mu/2-3]$

2. Implementation of line search

Two Line Search unconstrained optimizations are added as required. The first one is **MATLAB line search**, using minimizer function fminbnd of Matlab The second one is **backtracking** line search method. Both methods are used in combination we current Newton-Raphson's method.

These options can be chosen within following parameters:

To choose Line search (combinated with Newton):

options.linesearch=1

Choose type of line search:

options.type_LS=1,2   1: Backtracking, 2: Matlab

Two changes are done in the code delivered to students. Changes to function *Equilibrate*, and addition of new function called *linesearch*.

The complete new code is added in appendix.

<u>Reminder on Newton-Raphson's method:</u>

x is taken from main function, we is the displacement (x1,y1,x2,y2…) multiplied by the load of the current iteration divided by load on last iteration (coefficient similar to 1).

dx = -Hess_E\grad_E; is calculated via Energy on four Gauss points of each element, and thus the new positions of the node, which are x'=x+dx.(dx can be multiplied by scalar relaxation parameter as an option.).The second Piola Kirchhoff (S) and Isotropic strain energy (W) are taken from chosen NeoHookean.

This happens until errors are low enough:

```
err_x=norm(dx)/norm(x_short);
err_f=norm(grad_E);
```

Notice that both are calculated because gradient could be low but displacement in very low, or vice versa. So it's small residual and fa from root vs small steps but large residual.
Both must be considered at the same time. Of course there's a limit of iterations established.

So with a margin of confidence, the equilibrium for the energy state is achieved for x given by Equilibrium function.

Line Search

Matlab Line Search

In order to compute Line Search, a new function is coded:

```
function [t]= linesearch(dx,grad_E,x_short,options)
p=dx;
opts=optimset('TolX',options.TolX,'MaxIter',options.n_iter_max_LS);
t=fminbnd(@(t)Ener_1D(t,x_short,p),0,2,opts)%ha encontrado t que minimiza energia para x y gradx
end
```

This function uses fminbnd, a function which finds minimum between two values (not root).

Ener_1D was already coded, which search for energy of a particular array of displacements (dicarding constrained ones) ener = Ener_short(x+t*p,1).

So fminbnd finds “t” that makes minimum the energy (scalar) between t=0 and t=2. Here x_short being an array is not an issue , given that the function to minimize is scalar.

to find t value, the direction could be the gradient of the energy, which is the steepest descent method) x’=x+**pt. However, used in combination with Newton-Raphon’s method**, the direction is **dx** is taken, this is the displacement array just calculated.

x_short=x_short+t*dx*options.*relax_par* (optional relaxation par)

After, it is just about calculating the Eigenvalues on Hessian and warning if they are negative (which means is a maximum, so un unstable status).

Backtracking

This whole option is code-written in *Equilibrate* function.

```
 if (options.linesearch==1)&(options.type_LS==1) %backtracking LS
        options.n_iter_max=options.n_iter_max_LS;
        x=x_short;
        if  Ener_short(x+t*dx,1)> (Ener_short(x,1)+options.alfa*grad_E.*dx)
            t1=-options.beta*t; %backtrack
        else
            t1=options.beta*t;
        end
        p=dx;
        x_short=x+t1*p*options.relax_par;
        t=t1;
     end
```

This code searchs for the energy combining with Newton-Raphson’s already achieved dx.

Initializing t=1,

and as a reminder,

options.alfa=0.3
options.beta = .8

So it compares whether if the path we take is the correct and changes sense of direction if required.

Again relaxation parameter is optional (normally would be 1).

*x_short=x+t1*p*options.relax_par;*

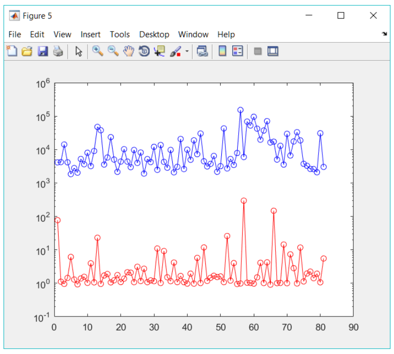
Once computed, same procedure as default is computed.

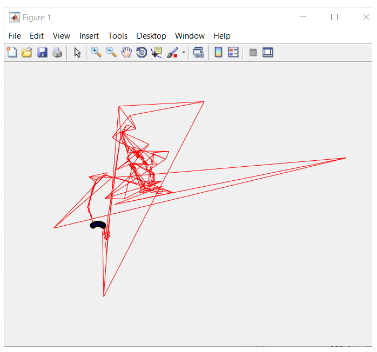
COMPARISON ON EXAMPLES

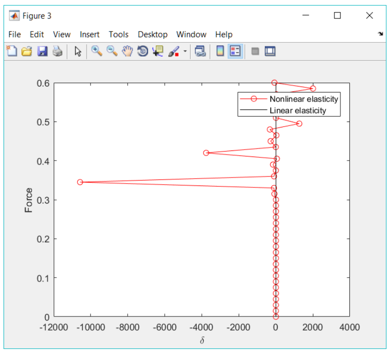
With relax_par=1 in all 3 cases and material 1

Example 5 (TWHO DEAD LOADS AT BOTH SIDES OF ARCH)

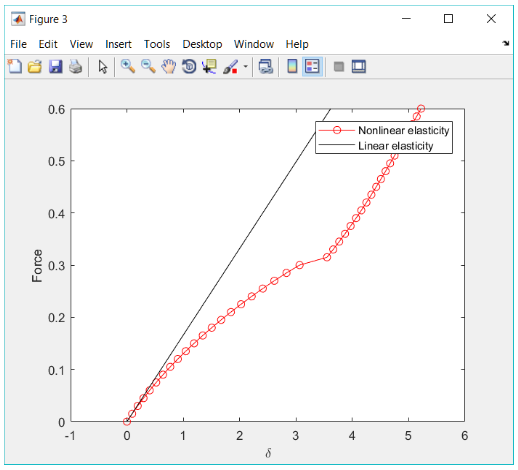
W/O LINE SEARCH



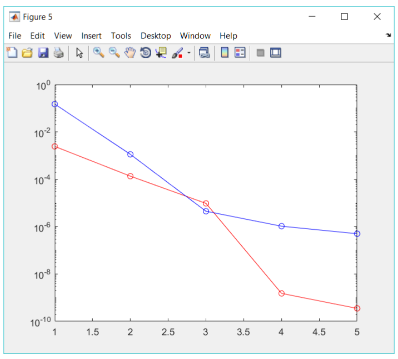


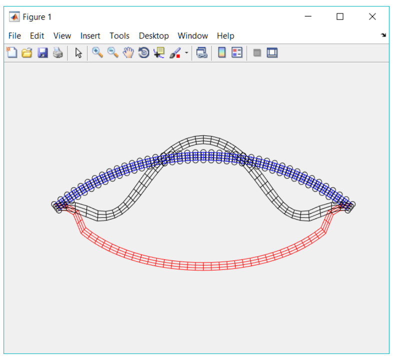


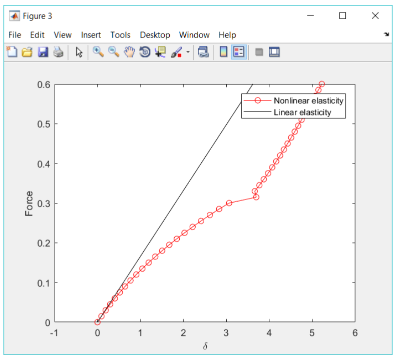
With relaxation parameter =0,5 we achieve converge. Buckling at load iteration 19:



WITH MATLAB LINE SEARCH COMBINED TO N-R

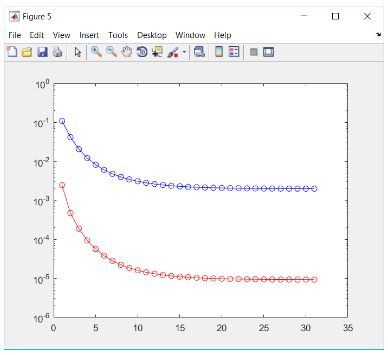


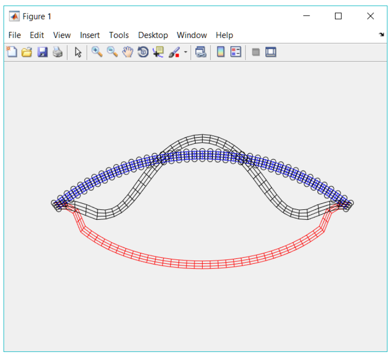


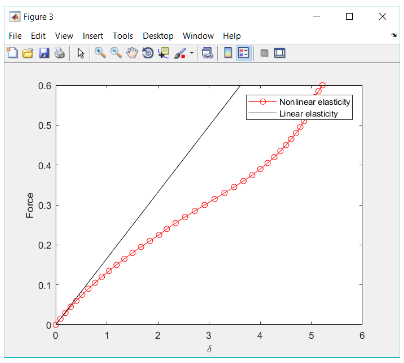


Buclink at load iteration 23. No need of relax. parameter.

WITH MATLAB LINE SEARCH BACKTRACKING COMBINED TO N-R



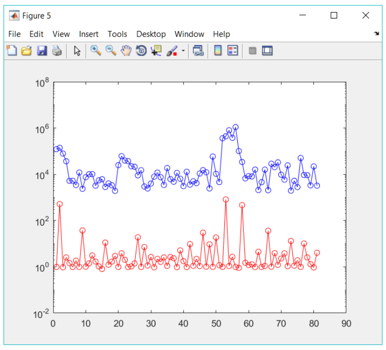


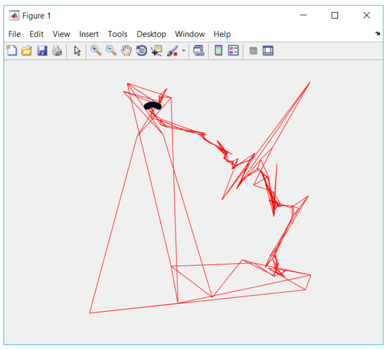


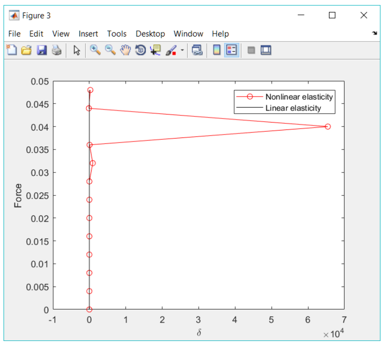
Buckling not found. Much more displacements iterations.

Example 4 DEAD LOAD AT CENTER OF ARCH

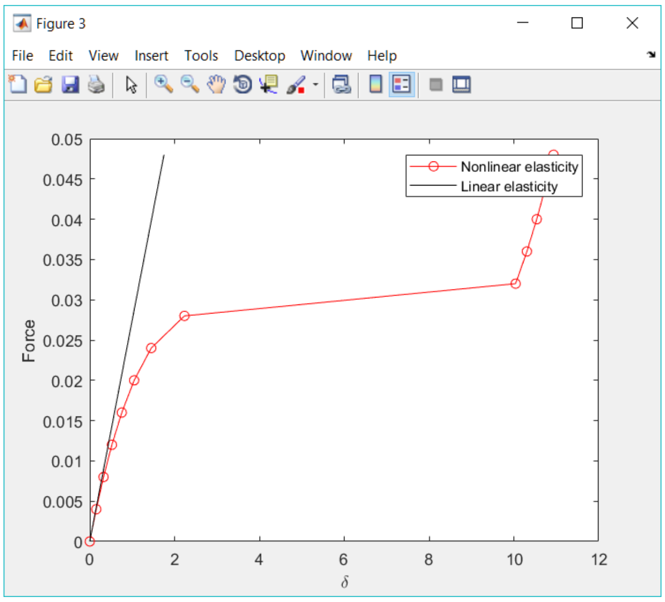
W/O LINE SEARCH



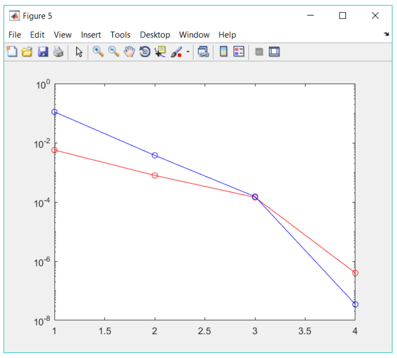


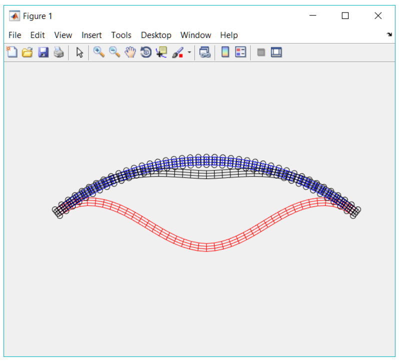


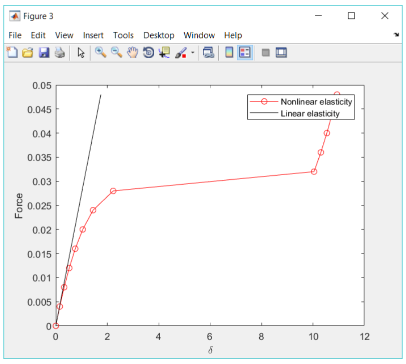
With relaxation parameter =0,5 we achieve converge. Buckling at eighth load iteration:



WITH MATLAB LINE SEARCH COMBINED TO N-R

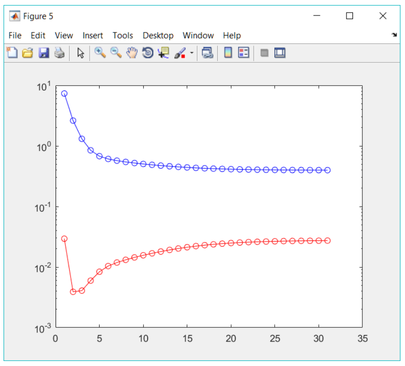


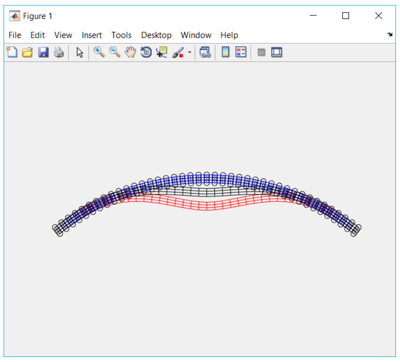


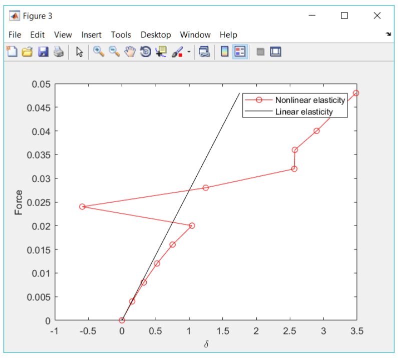


Bucking at load iteration nr 7.

WITH MATLAB LINE SEARCH BACKTRACKING COMBINED TO N-R



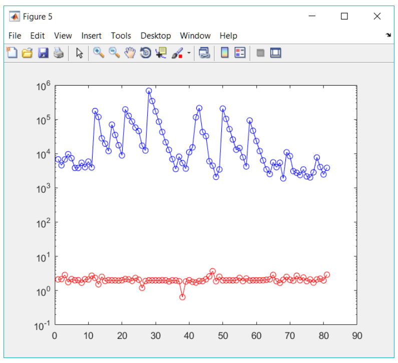


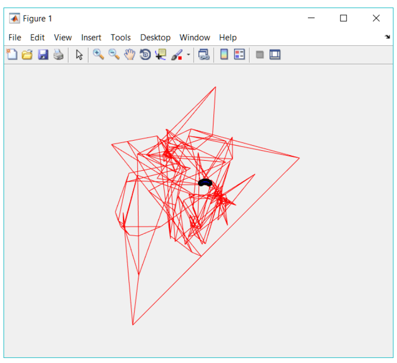


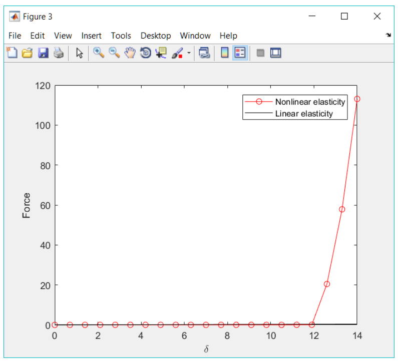
Much more displacements iterations. A negative displacement is achieved in the middle within iterations 4-5.

Example 44 DEAD LOAD AT CENTER OF ARCH

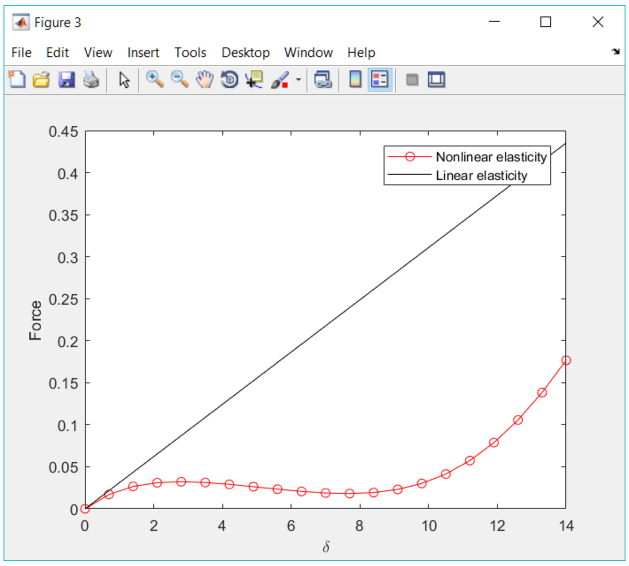
W/O LINE SEARCH



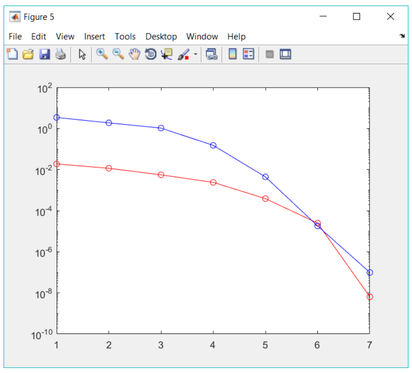


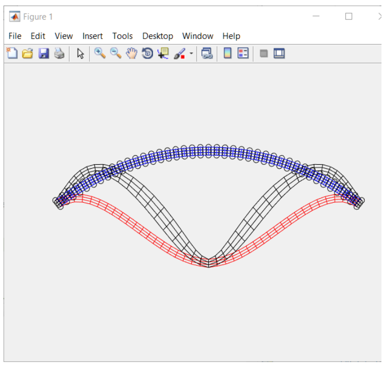


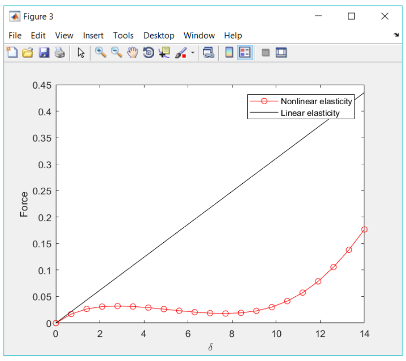
Not until relaxation parameter is about 0,4 converge is achieved. For higher values it diverges. Following, case for relax_par=0.4:



WITH MATLAB LINE SEARCH COMBINED TO N-R







WITH MATLAB LINE SEARCH BACKTRACKING COMBINED TO N-R
It diverges whatever relax. parameter is configured.

In general , we achieved convergence for line search methods for all cases(with last exception), either with backtracking or not. This is improves the Newton-Raphson's methodology.

Also noticeable that iterations for backtracking met**hod where much more in all cases.**

3.IMPLEMENTATION OF A MATERIAL MODEL

a) There's only need to establish the model/functions for the transversal isotropic material. The difficulty here is about derivating the PK2 (S) and tangent elasticity tensor ℂ with the correct Vogt notation.

Preprocessing has already the values for this material (case = 2), which are

```
mod1.mu=1;
mod1.c0=80;
mod1.c1=5;
mod1.kappa=100;
theta=pi/6;
mod1.N_fib=[cos(theta);sin(theta)];
```

The function energy will call transversal isotropic functions instead of Neohookian if material is number 2, given that preprocessing function sets potencial to 2.

The functions are analog to the Neohookian ones, with no dependency in λ.

```
[W]=transv_isotr_1(C,c0,c1,k,mu,fib)
[W,S]=transv_isotr_2(C,c0,c1,k,mu,fib)
[W,S,CC]=transv_isotr_3(C,c0,c1,k,mu,fib)
```

Given the following Isotropic strain energy:

$W(\mathbf{C})=\mu_0(\mathrm{trace}\mathbf{C}-3)-\mu_0 \ln J+kG(J)+c_0\{\exp[c_1(I_4(\mathbf{C})^{1/2}-1)^4]-1]\}$
$G(J)= \tfrac{1}{4}(J^2-1-2\ln J)$
$I_4(\mathbf{C})=\mathbf{N}^{fib}.\mathbf{C}.\mathbf{N}^{fib}$
$\mathbf{N}^{fib}=[\cos\theta, \sin\ \theta]^T$

we calculate first S for isotr_2 and istro_3, which is a vector that energy requires in order to compute gradient.

$\mu_0(\mathrm{trace}\mathbf{C}-3)\text{-->}\mu_0(\mathbf{C^{-1}}\text{-}\mathbf{Id})$
$kG(J)\text{-->}k/4*\det C*\mathbf{C^{-1}}\text{-}\mathbf{2C^{-1}}$

regrouping this

$\mathbf{S}=\mu_0(\mathbf{C^{-1}}\text{-}\mathbf{Id})+k/4*\det C*\mathbf{C^{-1}}\text{-}\mathbf{2C^{-1}}$

Same procedure to calculate tangent elasticity tensor:

$\mathbf{CC}=(\mu-2)*(d\mathbf{C}^{-1}/dC)+k*J2/4*(d\mathbf{C}^{-1}/d\mathbf{C})=(\mu_0-2+kJ^2/4)*(d\mathbf{C}^{-1}/d\mathbf{C})$

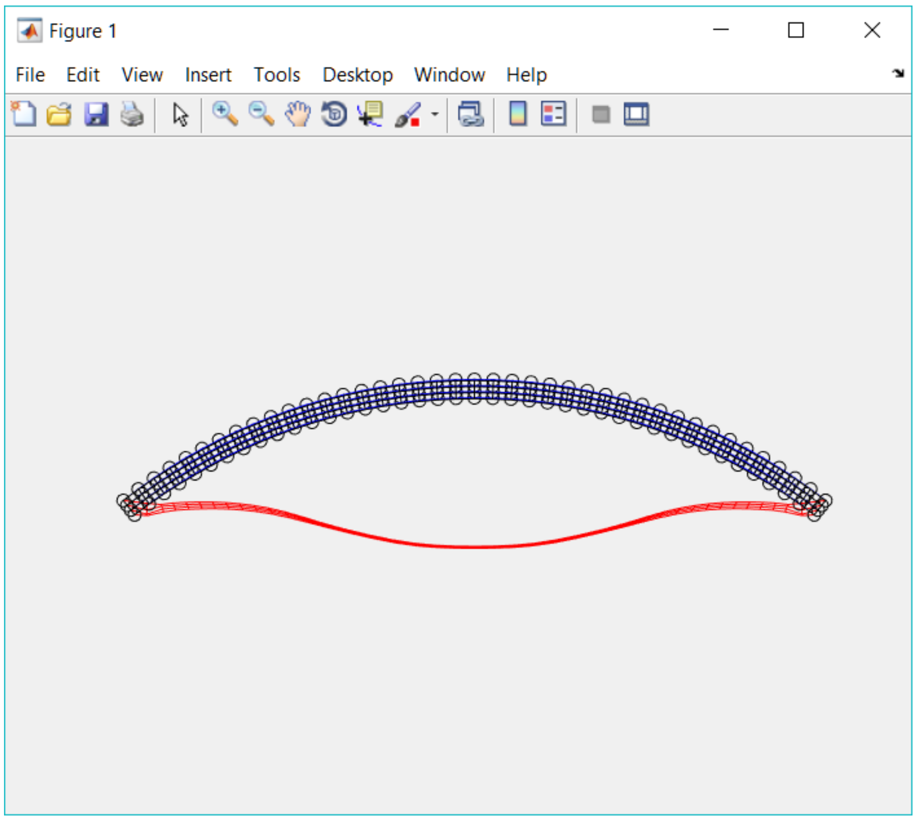
Following, how transversal isotropic has been coded:

```
function [W,S,CC]=transv_isotr_3(C,c0,c1,k,mu,fib)

J2=C(1)*C(2)-C(3)*C(3);  %es J^2=detC
J=sqrt(J2);
logJ=log(J);
G=(1/4)*(J^2-1-2*logJ);
i4=C(3)*fib(1)*fib(2);

W = 1/2*mu*(C(1)+C(2)-2) - mu*logJ +k*G+c0*exp(c1*((sqrt(i4)-1)^4))-1;
C_inv=[C(2) C(1) -C(3)]/J2;
S= mu*(C_inv-[1,1,0])+(k/4)*J2*C_inv-2*C_inv
CC=zeros(3);
const=(mu-2+J2*k/4);
CC(1,1)=const*(C_inv(1)*C_inv(1)+C_inv(1)*C_inv(1));
CC(1,2)=const*(C_inv(3)*C_inv(3)+C_inv(3)*C_inv(3));
CC(1,3)=const*(C_inv(1)*C_inv(3)+C_inv(1)*C_inv(3));
CC(2,2)=const*(C_inv(2)*C_inv(2)+C_inv(2)*C_inv(2));
CC(2,3)=const*(C_inv(2)*C_inv(3)+C_inv(2)*C_inv(3));
CC(3,3)=const*(C_inv(1)*C_inv(2)+C_inv(3)*C_inv(3));
CC(2,1)=CC(1,2);
CC(3,1)=CC(1,3);
CC(3,2)=CC(2,3);
```

When tested with Matlab line searching, a relaxation parameter (0,4) is required to ensure the convergence for example 4



.

What can be induced by the results is that a ***major buckling*** *occurs with the first load iteration. After that it slowly behaves as a linear F-d.*

b) Checking the **consistency** through *Check_Derivatives* function. **Gradient and the Hessian** of the energy ae computed given a very small perturbation of **h**=$1*10^{-9}$ and two different deformation fields for x and y displacements around **x0** (preprocessing), which are non homogeneous and arbitrary.
As an example are settled exponential /sinusoidal as x and y displacements respectively.

For each degree of freedom, are calculated as said before, and thus comparison between gradient (and Hessian) and slope it's known how sensitive is current state and awares us about mistakes or needs of inclusion of relaxation parameters.

$$|(\mathbf{E}^h\text{-}\mathbf{E})/h - \nabla \boldsymbol{E}) \;/\; \nabla \boldsymbol{E}| > 1*10^{-3}$$

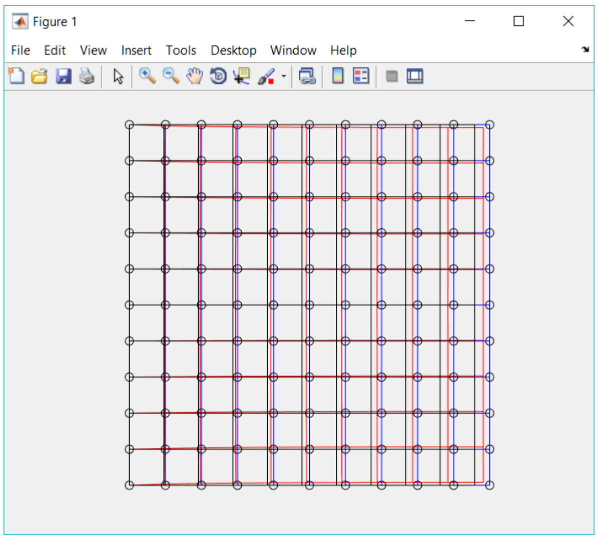
$$||(\nabla \boldsymbol{E}^h - \nabla \boldsymbol{E})/h - \boldsymbol{H}|| \;/\; ||\boldsymbol{H}|| > 1*10^{-3}$$

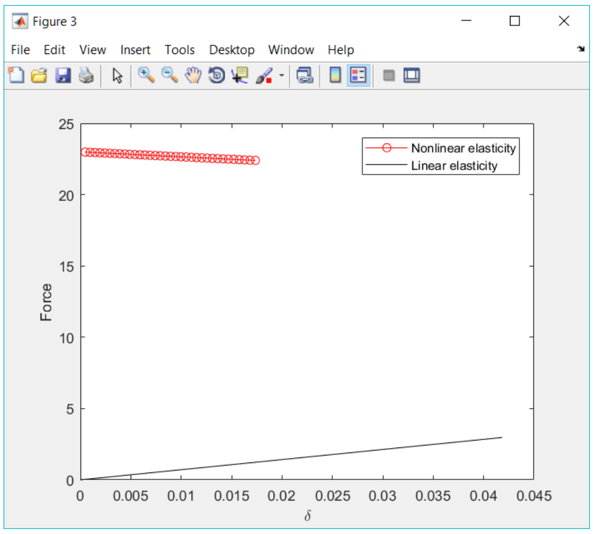
In the case analyzed (example 4 , trans. isotropic) are both 3 orders of magnitude smaller than warning activations ($1*10^{-3}$). So model is consistent.

c) Solving the example =0, **dead load** on elastic block in tension, for different angles of θ, we obtain different solutions that can be explained through a mechanical viewpoint.

From fibers along loading direction θ=0, to perpendicular to the load, are computed
θ=0,π/6,π/4 ,π/2
A bigger right-center displacement is produced in between **0** and **π/2.** Both cases present similar displacements and **minimum of all these examples.**
Also noticeable that when fibers are diagonal (**π/6,π/4)** it appears an **asymmetry in vertical displacement.**

θ=0



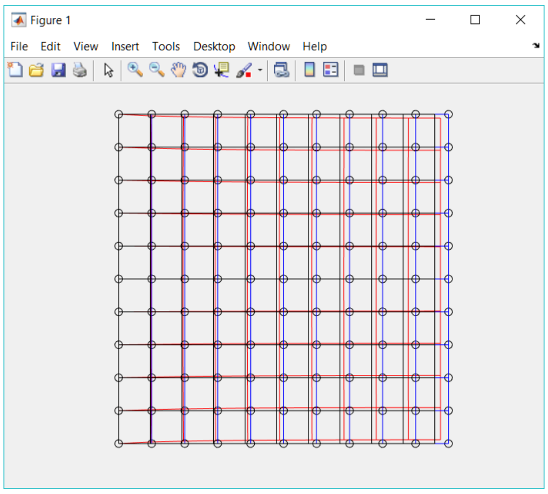


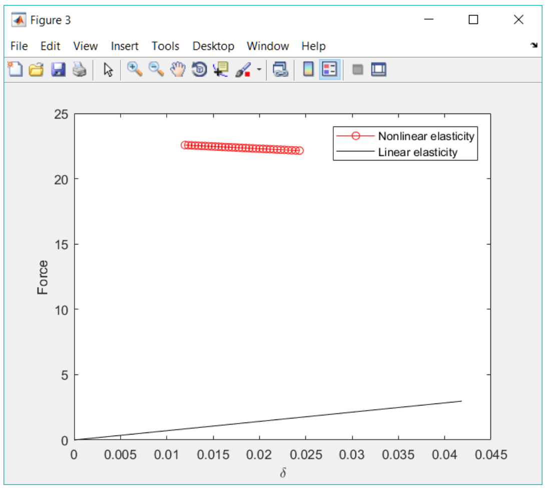
Assignment 3

COMPUTATIONAL SOLID MECHANICS

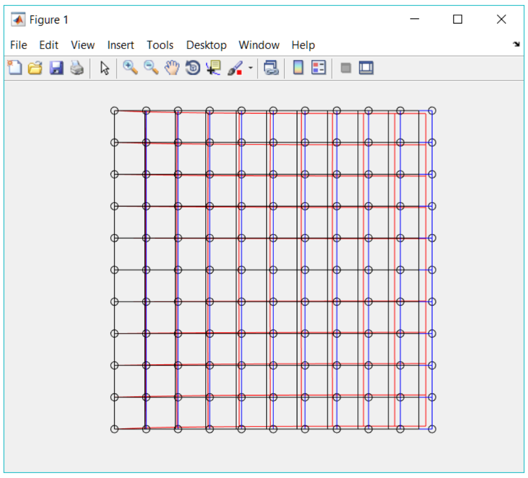
Marcos Boniquet Aparicio

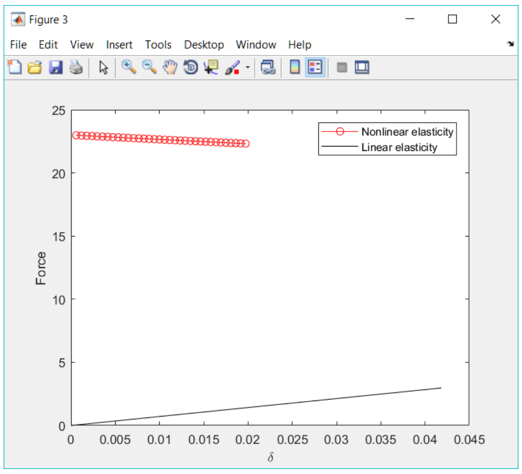
θ=π/6



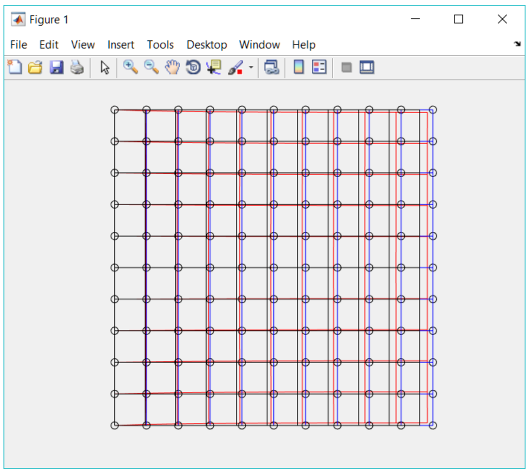


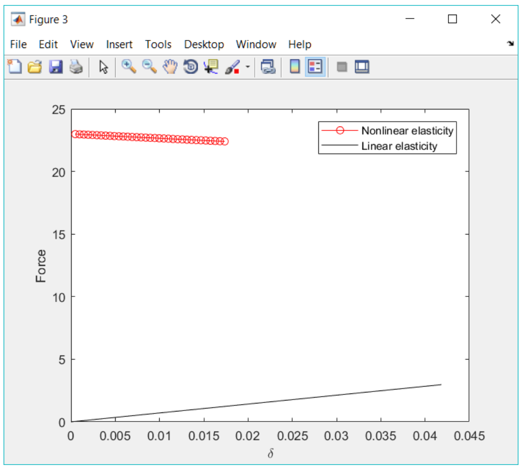
θ=π/4





θ=π/2





```
function [x_equil,iflag,iter,Ener] = Equilibrate(x,options)
global mod1 mesh1 load1 el1 undeformed1

err_plot=[];
err_plot1=[];
[x_short] = short(x);

switch options.method
  case 0,   %vanilla Newton-Raphson with line search as option
    iter=0;
    err_x=100;
    err_f=100;
    [Ener,grad_E,Hess_E] = Ener_short(x_short,3);


    t=1;   %initializing for backtracking
    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 options.linesearch==0
           x_short=x_short+dx*options.relax_par;
       end


      if (options.linesearch==1)&(options.type_LS==2) %matlab LS
          options.n_iter_max=options.n_iter_max_LS;
          [t]=linesearch(dx,grad_E,x_short,options);
          x_short=x_short+t*dx*options.relax_par;
      end

      if (options.linesearch==1)&(options.type_LS==1) %backtracking LS
         options.n_iter_max=options.n_iter_max_LS;
         x=x_short;
         if  Ener_short(x+t*dx,1)>
 (Ener_short(x,1)+options.alfa*grad_E.*dx)
             t1=-options.beta*t;
         else %backtrack
             t1=options.beta*t;
         end
         p=dx;
         x_short=x+t1*p*options.relax_par;
         t=t1;
      end

     [Ener,grad_E,Hess_E] = Ener_short(x_short,3);
      err_x=norm(dx)/norm(x_short);
      err_f=norm(grad_E);
      err_plot=[err_plot err_x];
      err_plot1=[err_plot1 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')
           D=sort(D);
           disp(D(1:6))
      end
    end
  otherwise,
    error('This option does not exist');
end

[x_equil] = long(x_short);
if (iter<=options.n_iter_max)
  if options.info>0
    fprintf('The equilibration was successful in %i iterations
 \n',iter)
  end
  iflag=1;
else
  if options.info>0
    fprintf('The equilibration was not reached in %i iterations
 \n',iter)
  end
  iflag=0;
end

%Output
if options.info>=2
  figure(5)
  hold off
  semilogy([1:iter],err_plot,'ro-',[1:iter],err_plot1,'bo-');
  %pause
end
```

```
Not enough input arguments.

Error in Equilibrate (line 6)
[x_short] = short(x);
```

Published with MATLAB® R2017a

```
function [t]= linesearch(dx,grad_E,x_short,options)
p=dx;
opts=optimset('TolX',options.TolX,'MaxIter',options.n_iter_max_LS);
t=fminbnd(@(t)Ener_1D(t,x_short,p),0,2,opts)%ha encontrado t que
 minimiza energia para x y gradx
end
```

```
Not enough input arguments.

Error in linesearch (line 2)
p=dx;
```

Published with MATLAB® R2017a

```
function [W]=transv_isotr_1(C,c0,c1,k,mu,fib)

J2=C(1)*C(2)-C(3)*C(3);  %es J^2=detC
J=sqrt(J2);
logJ=log(J);
G=(1/4)*(J^2-1-2*logJ);

i4=C(3)*fib(1)*fib(2);

W = 1/2*mu*(C(1)+C(2)-2) - mu*logJ +k*G+c0*(exp(c1*sqrt(i4)-1)^4-1);
```

```
Not enough input arguments.

Error in transv_isotr_1 (line 3)
J2=C(1)*C(2)-C(3)*C(3);  %es J^2=detC
```

Published with MATLAB® R2017a

```
function [W,S]=transv_isotr_2(C,c0,c1,k,mu,fib)
J2=C(1)*C(2)-C(3)*C(3);  %es J^2=detC
J=sqrt(J2);
logJ=log(J);
G=(1/4)*(J^2-1-2*logJ);

i4=C(3)*fib(1)*fib(2);

W = 1/2*mu*(C(1)+C(2)-2) - mu*logJ +k*G+c0*exp(c1*((sqrt(i4)-1)^4))-1;
S =[];

C_inv=[C(2) C(1) -C(3)]/J2;

S= mu*(C_inv-[1,1,0])+(k/4)*J2*C_inv-2*C_inv;

end
```

```
Not enough input arguments.

Error in transv_isotr_2 (line 2)
J2=C(1)*C(2)-C(3)*C(3);  %es J^2=detC
```

Published with MATLAB® R2017a

```
function [W,S,CC]=transv_isotr_3(C,c0,c1,k,mu,fib)

J2=C(1)*C(2)-C(3)*C(3);  %es J^2=detC
J=sqrt(J2);
logJ=log(J);
G=(1/4)*(J^2-1-2*logJ);
i4=C(3)*fib(1)*fib(2);

W = 1/2*mu*(C(1)+C(2)-2) - mu*logJ +k*G+c0*exp(c1*((sqrt(i4)-1)^4))-1;

C_inv=[C(2) C(1) -C(3)]/J2;

S= mu*(C_inv-[1,1,0])+(k/4)*J2*C_inv-2*C_inv  ;

CC=zeros(3);

const=(mu-2+J2*k/4);

%W ES SCALAR, S ES VECTOR, CC ES 3X3

CC(1,1)=const*(C_inv(1)*C_inv(1)+C_inv(1)*C_inv(1));
CC(1,2)=const*(C_inv(3)*C_inv(3)+C_inv(3)*C_inv(3));
CC(1,3)=const*(C_inv(1)*C_inv(3)+C_inv(1)*C_inv(3));
CC(2,2)=const*(C_inv(2)*C_inv(2)+C_inv(2)*C_inv(2));
CC(2,3)=const*(C_inv(2)*C_inv(3)+C_inv(2)*C_inv(3));
CC(3,3)=const*(C_inv(1)*C_inv(2)+C_inv(3)*C_inv(3));
CC(2,1)=CC(1,2);
CC(3,1)=CC(1,3);
CC(3,2)=CC(2,3);
```

```
Not enough input arguments.

Error in transv_isotr_3 (line 3)
J2=C(1)*C(2)-C(3)*C(3);  %es J^2=detC
```

Published with MATLAB® R2017a