Computational Solid Mechanics Assignment 1

Oriol Call Piñol

Master on Numerical Methods in Engineering

Table of Contents

Part I (rate independent models):

I.I Integration algorithms (rate independent and plane strain case):

For this first part, it is important to introduce how the code works for the model which is already implemented. We will have to take a look at the first function we are going to modify, which is “Modelos_de_dano1”. In this function we will have to add the parameters needed to compute the 2 methods the problem statement asks for: The “only-tension” damage model and “non-symmetric” damage model.

When the application of “rtrial” provided by the notes on all cases and having defined the tensor strain in a forward time step, we continue by modifying the function “dibujar_criterio_dano1”. This so called function will allow us to plot the solution of these problems after being computed. In this part, the only thing we need to modify to be able to draw the solution is adding the conditions in which the “only-tension” model works with stress, such, being 0 when stress should be negative.

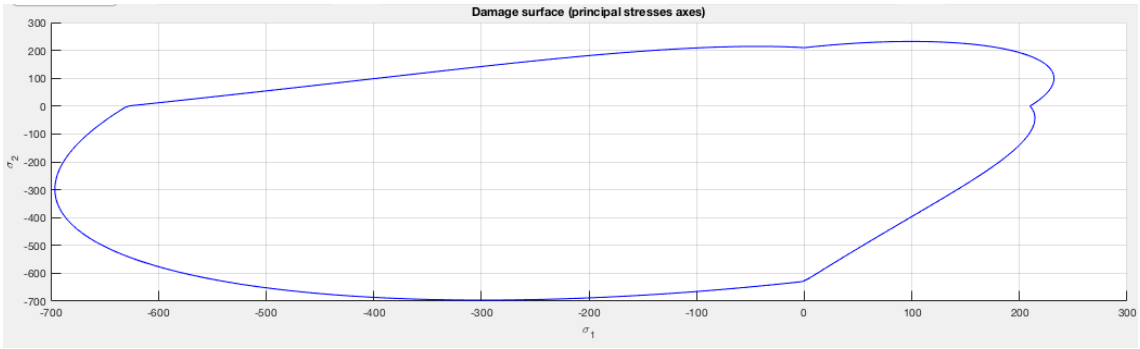


Figure 1 Non-symmetric damage model

For the non-symmetric case we get opposite results to the symmetric case, we can see that the first quadrant of pi is defined the same way as the symmetric case but the solution does

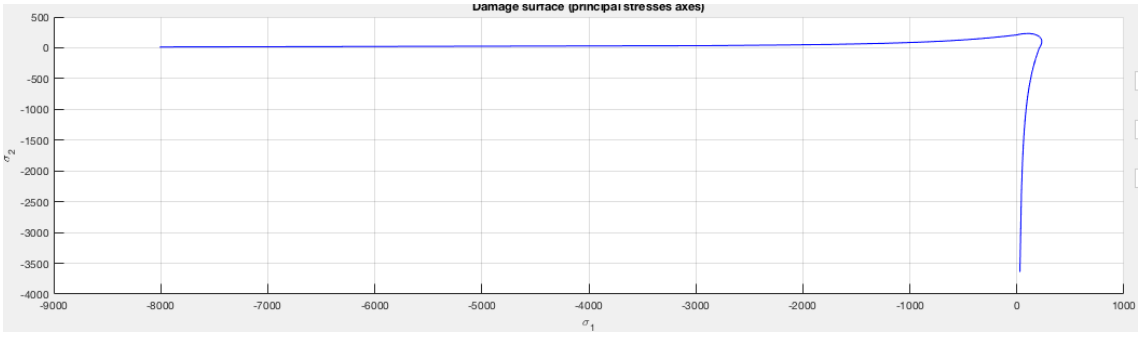


Figure 2 Only-tension Damage model

This figure shows the result we get when we apply only tension. As we can see, there is no definition of the damage surface on the compression part, as we are only applying tension.

I.II Implementation of linear/exponential (hardening/softening):

For this second part of the assignment, the request is to implement the linear and exponential hardening and softening for those 3 different models. To do so, we will need to modify the so called "rmap_dano1" function from our provided sources.

We will define "H" as the hardening coefficient in the linear case as a constant variable introduced to the function by an input, however, in the exponential case, "H" comes from the following formula from the slides:

$$q_\infty = r_0 + (r_0 - q_0)$$

We have to mention that the internal variable r is the one defined by the slides in the non-viscous case and "q" which is the hardening variable will be equal to:

$$q_n = q_{n+1} - H(r_{n+1} - r_n)$$

Hardening and softening won't happen during the unloading. When an elastic loading or unloading occurs, we apply to the hardening and internal variables the value from the previous step computed.

I.III Implementation of the previous models with some different stress cases

I.III.a Case 1

For the next part we will take into account that the load path consists in 3 steps. The first one is loading, followed by unloading and finishing by another loading. This will enable us to see how the hardening affects the different cases we are going to compute.

1.

$\Delta\bar{\sigma}_1^{(1)} = \alpha \quad ; \quad \Delta\bar{\sigma}_2^{(1)} = 0$ (uniaxial tensile loading)

$\Delta\bar{\sigma}_1^{(2)} = -\beta \quad ; \quad \Delta\bar{\sigma}_2^{(2)} = 0$ (uniaxial tensile unloading/compressive loading)

$\Delta\bar{\sigma}_1^{(3)} = \gamma \; ; \quad \Delta\bar{\sigma}_2^{(3)} = 0$ (uniaxial compressive unloading/ tensile loading)

With α = 500, β = 550, ϒ = 700

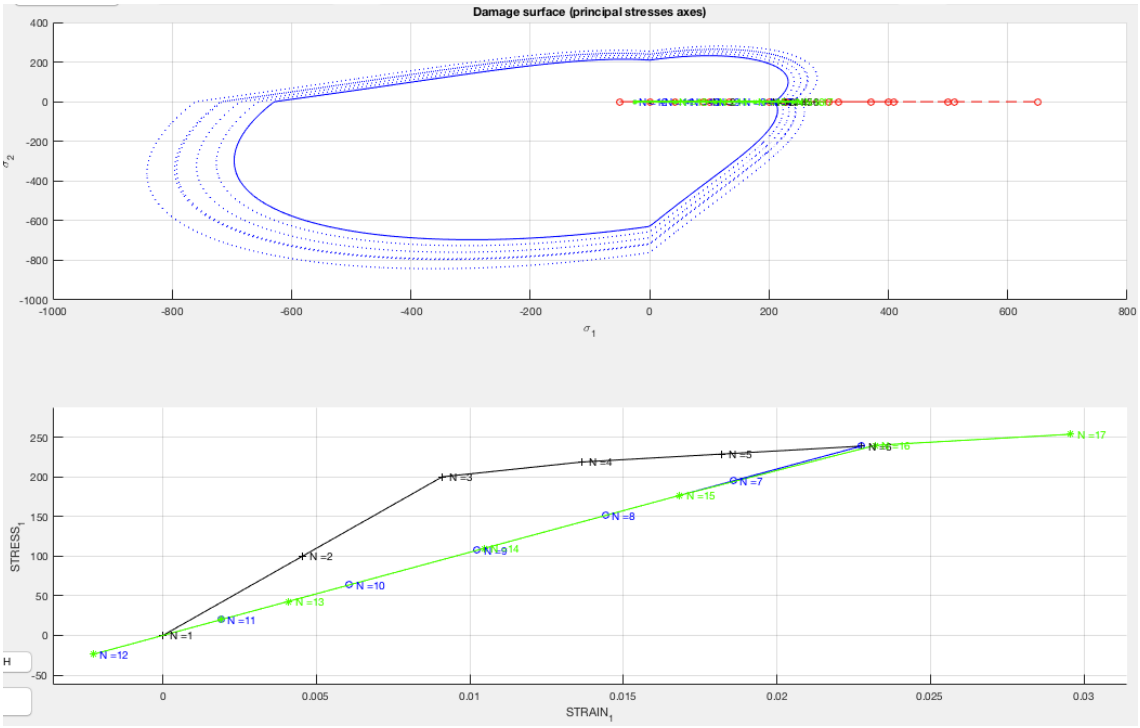


Ilustración 3 Load path for non-symmetric case 1. Inviscid, H=0.2.

Figure shows the load path for non-symmetric case with positive hardening (0.2) and we can see that the first slope un black represents the first tension applied, then the blue which goes backwards and then the last one in green which is tension again shows that the value increases after crossing the damage surface.

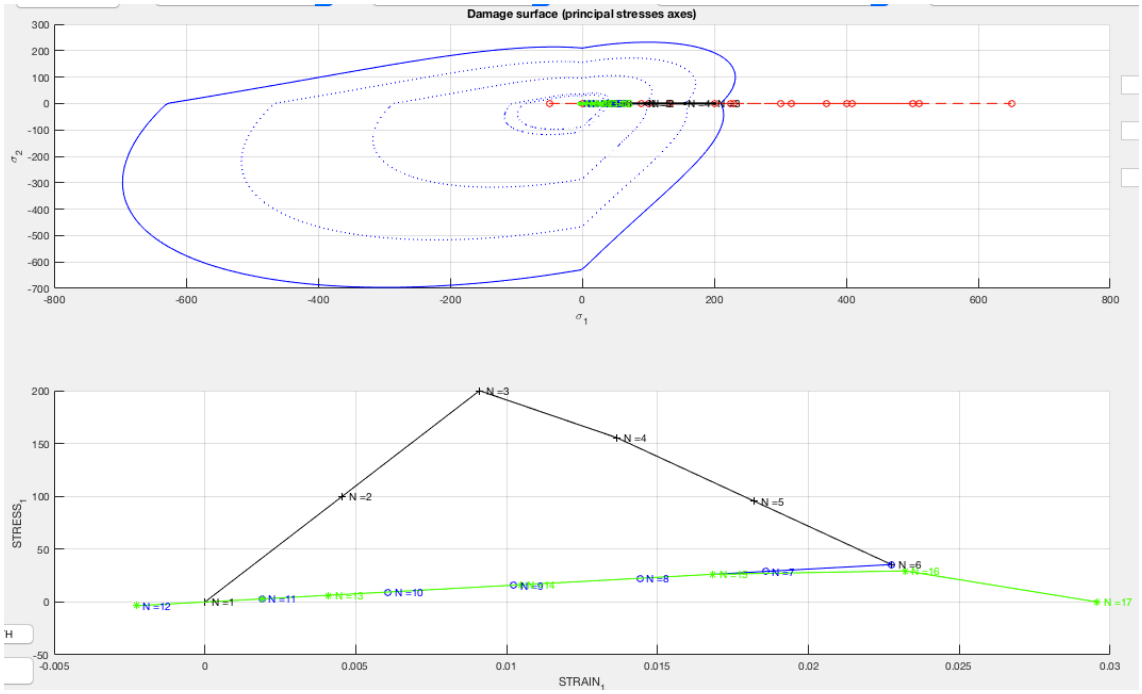


Ilustración 4 Load path for non-symmetric case 1. Inviscid, H=-0.6

This figure shows the softening and load path for non-symmetric case. As we can see, after the unloading, when loading again, the slope decreases due to that negative hardening coefficient (-0.6).

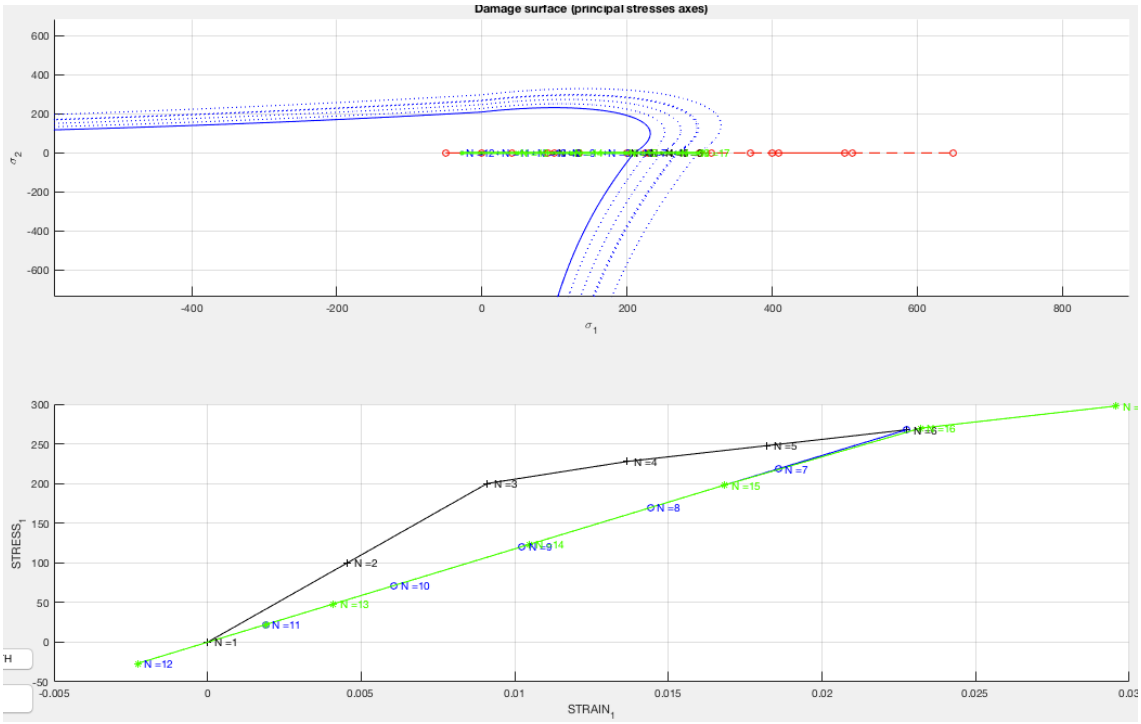


Figure 5 Load path for only-tension case 1. Inviscid, H=0.2

In figure 5 we can see that the results for the only-tension with positive hardening is very similar to the results we got in the non-symmetric case. The last slope is slightly stronger than the first one. Again, this is due to the hardening coefficient used.

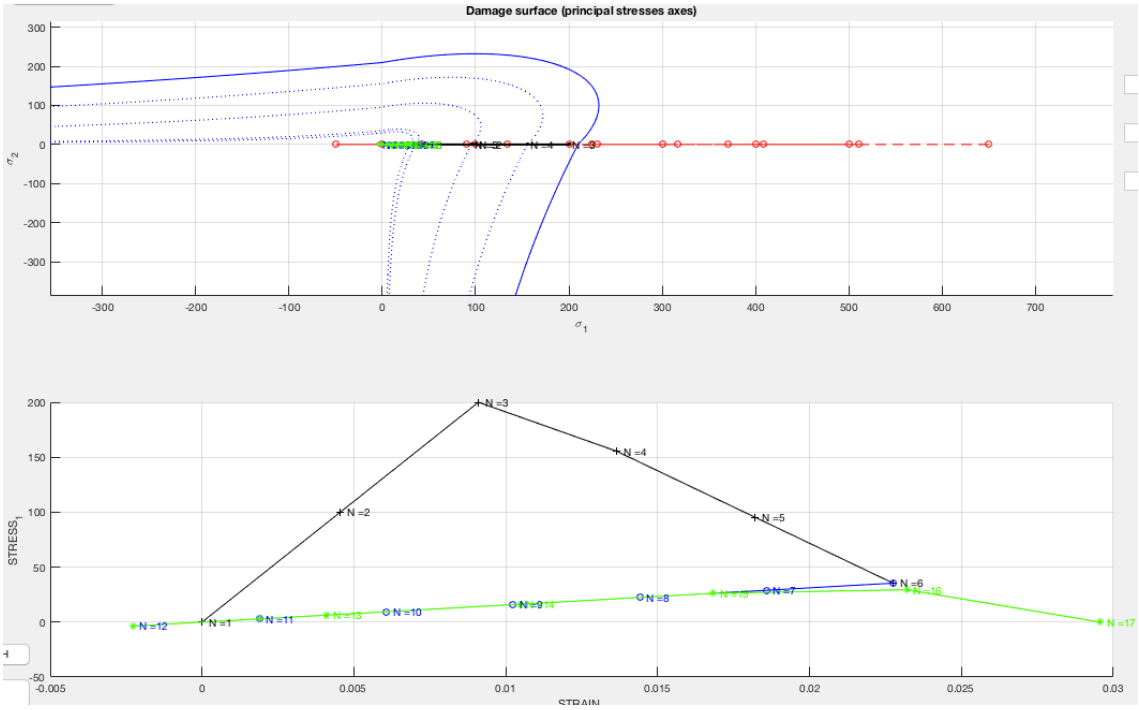


Figure 6 Load path for only-tension case 1. Inviscid, H= -0.6.

The figure 5 shows the only-tension case with negative hardening (-0.6). This results are practically the same as the non-symmetric case for the same exact reasons.

I.III.a Case 2

2.

$$\Delta\bar{\sigma}_1^{(1)} = \alpha \quad ; \quad \Delta\bar{\sigma}_2^{(1)} = 0 \text{ (uniaxial tensile loading)}$$

$$\Delta\bar{\sigma}_1^{(2)} = -\beta \quad ; \quad \Delta\bar{\sigma}_2^{(2)} = -\beta \text{ (biaxial tensile unloading/compressive loading)}$$

$$\Delta\bar{\sigma}_1^{(3)} = \gamma \; ; \quad \Delta\bar{\sigma}_2^{(3)} = \gamma \text{ (biaxial compressive unloading/tensile loading)}$$

With α = 500, β = 800, ϒ = 400

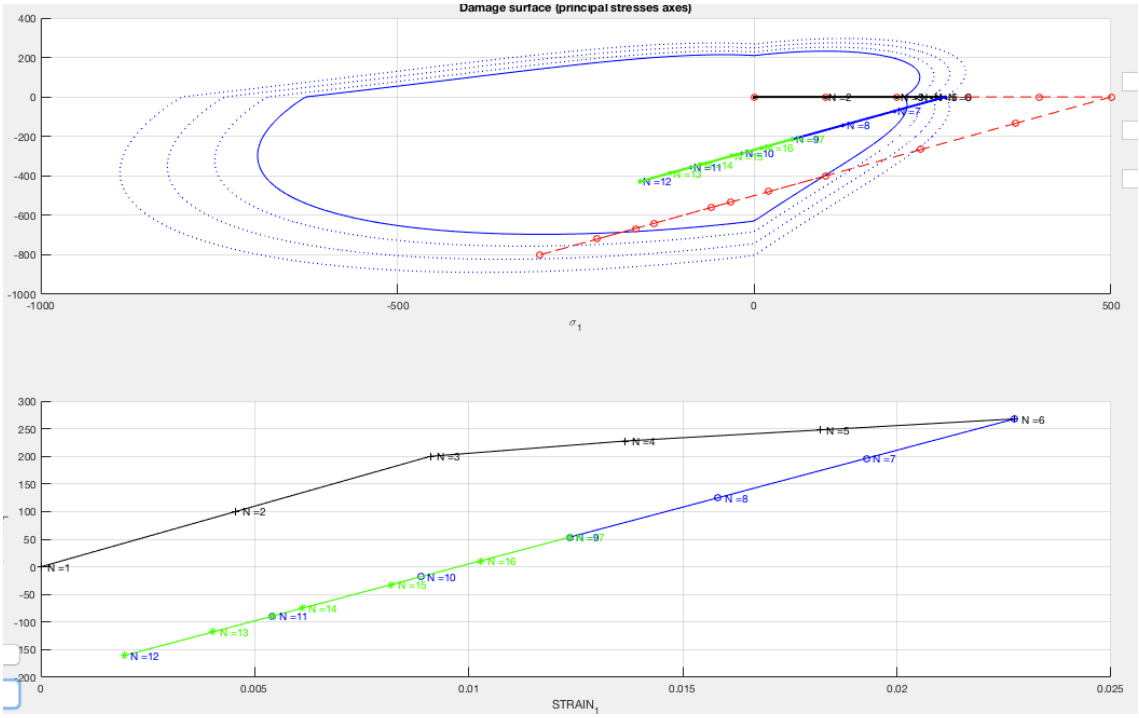


Figure 7 Load path for non-symmetric case 2. Inviscid, H= 0.2

In this figure we can appreciate the non-symmetric case with a positive hardening. Starting from the origin, the black slope changes its path when it collides with the damage surface and continues ascending, the next part is a biaxial unloading or compression, we can see that it traspasses the damage surface as well without any perturbation on this path. At the end, in green, we have another biaxial unloading which doesn't go through the surface.

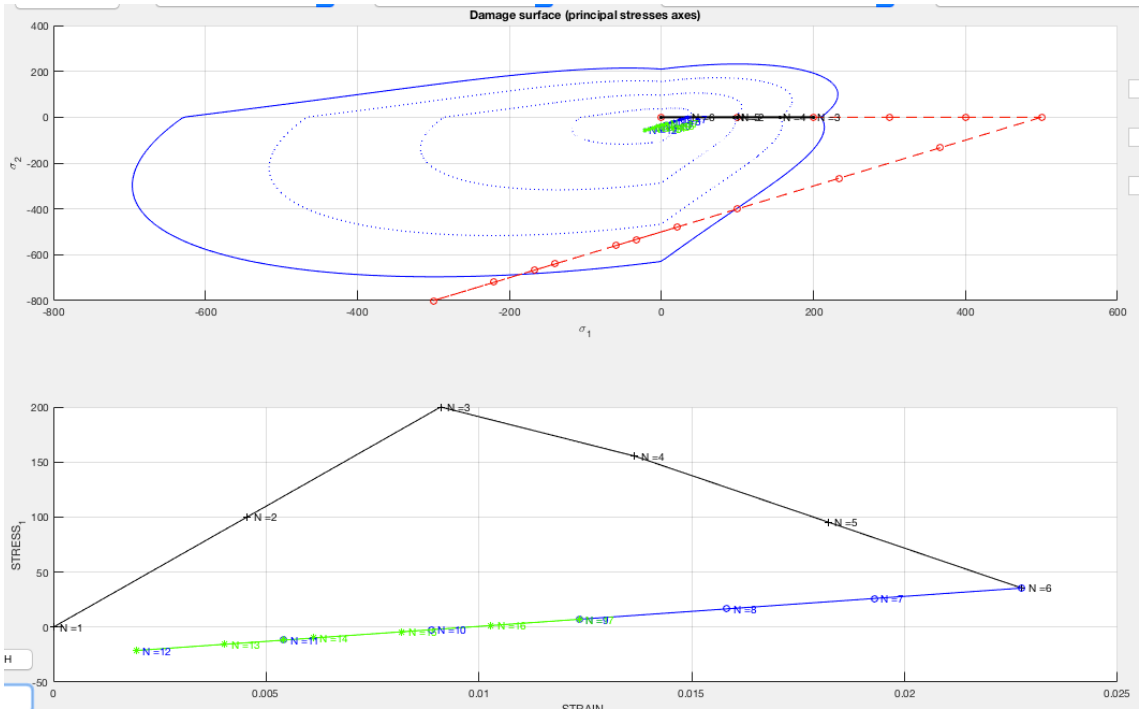


Figure 8 Load path for non-symmetric case 2. Inviscid, H= -0.6

The figure above shows the same example with a negative hardening coefficient, as we can see, the first tension applied, after crossing the damage Surface, goes far below compared to the positive hardening case, which means that in the next curve, the yield stress will start to decrease.

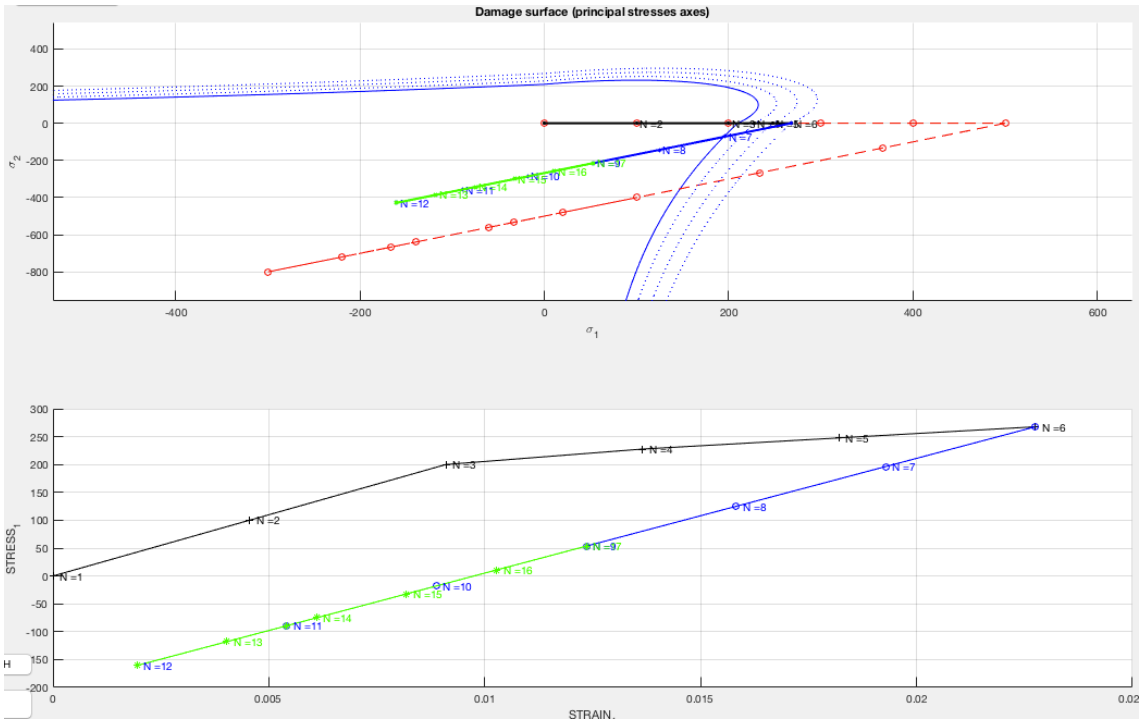


Figure 9 Load path for only-tension case 2. Inviscid, H= 0.2

As we can see in the figure below, the results for only-tension and positive hardening show the same behavior than the non-symmetric case for the same reasons.

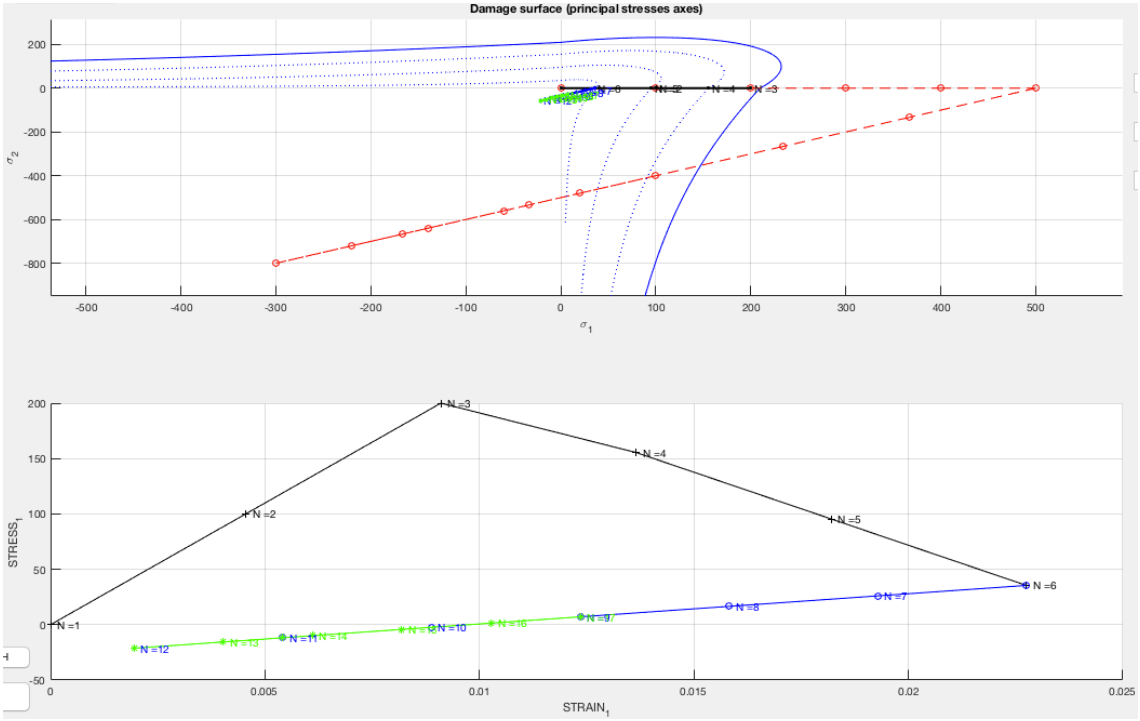


Figure 10 Load path for only-tension case 2. Inviscid, H= -0.6

The same explanation goes for this case, the negative hardening coefficient makes the system behave just as the non-symmetric case.

I.III.a Case 3

3.

$$\Delta\bar{\sigma}_1^{(1)} = \alpha \;;\; \Delta\bar{\sigma}_2^{(1)} = \alpha \text{ (biaxial tensile loading)}$$

$$\Delta\bar{\sigma}_1^{(2)} = -\beta \;;\; \Delta\bar{\sigma}_2^{(2)} = -\beta \text{ (biaxial tensile unloading/compressive loading)}$$

$$\Delta\bar{\sigma}_1^{(3)} = \gamma \;;\; \Delta\bar{\sigma}_2^{(3)} = \gamma \text{ (biaxial compressive unloading/tensile loading)}$$

With $\alpha = 500$, $\beta = 800$, $\Upsilon = 400$

In this case we are going to apply in a first instance a biaxial tensile loading followed by biaxial compressive loading, just as the previous case and concluding with another biaxial tensile loading.

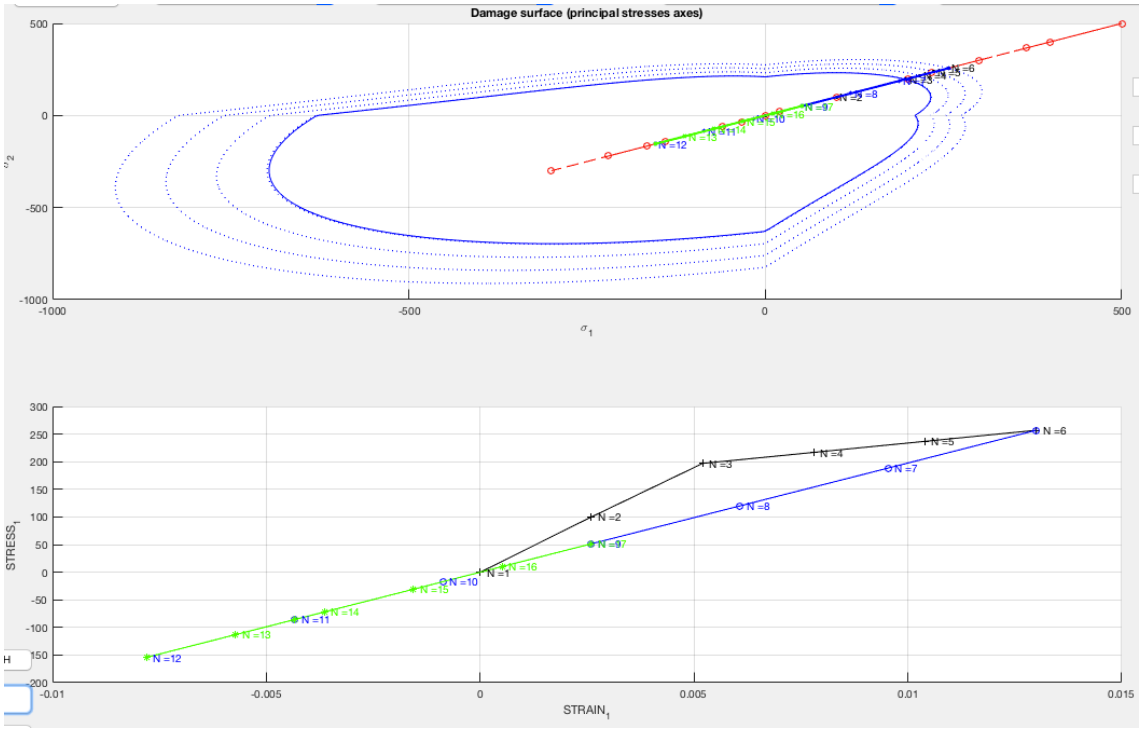


Figure 11 Load path for non-symmetric case 3. Inviscid, H= 0.2

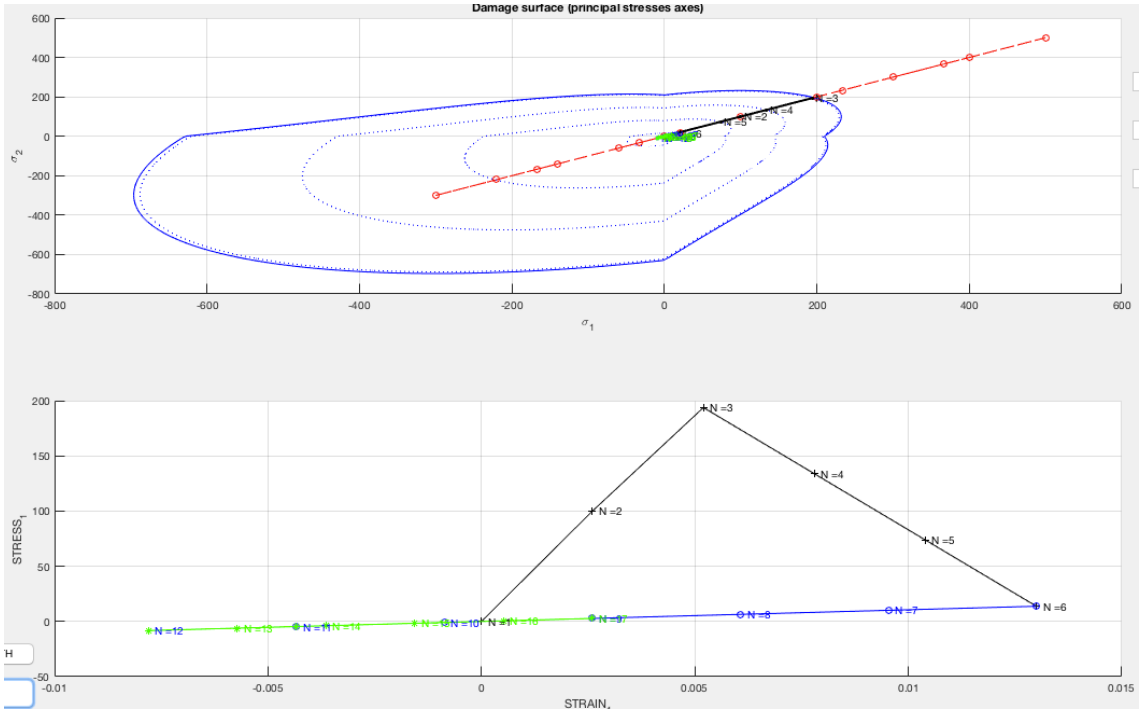


Figure 12 Load path for non-symmetric case 3. Inviscid, H= -0.6

In the previous figures we can see the different behaviours for positive and negative hardening in the non-symmetric case for the last loading and unloading instance. The material remains

elastic as far as it stays in the elastic domain. During the second and third part of the process, both biaxial unloading we can see that the material behaves as expected following the same path.

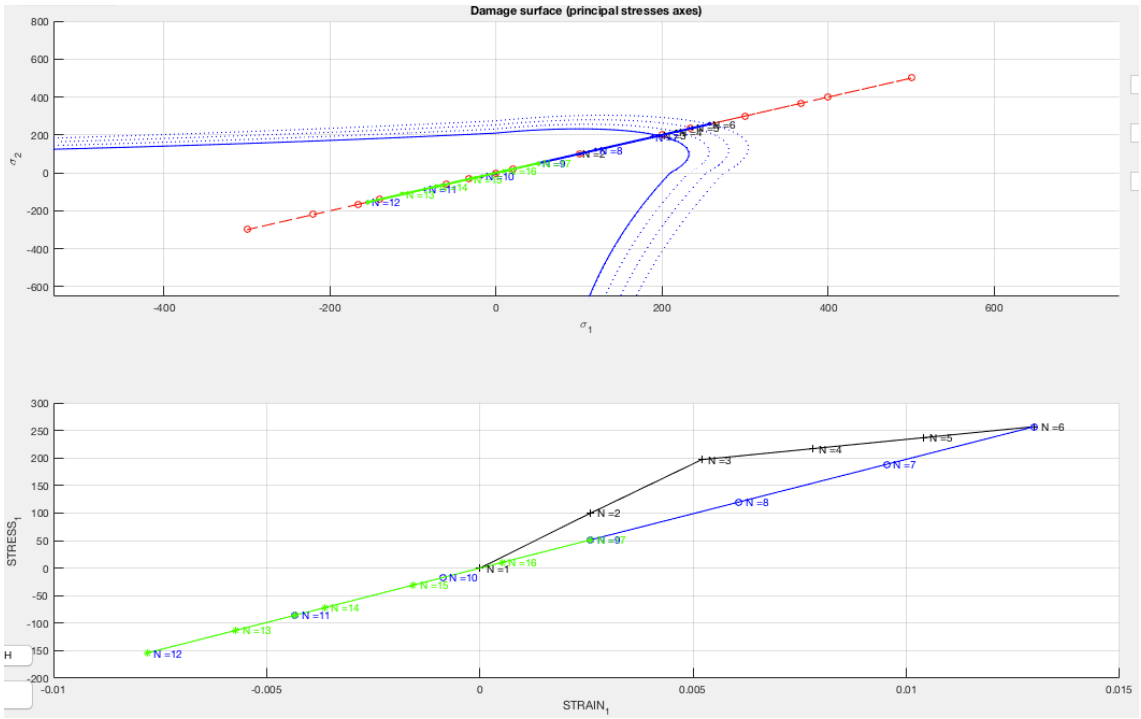


Figure 13 Load path for only-tension case 2. Inviscid, H= 0.2

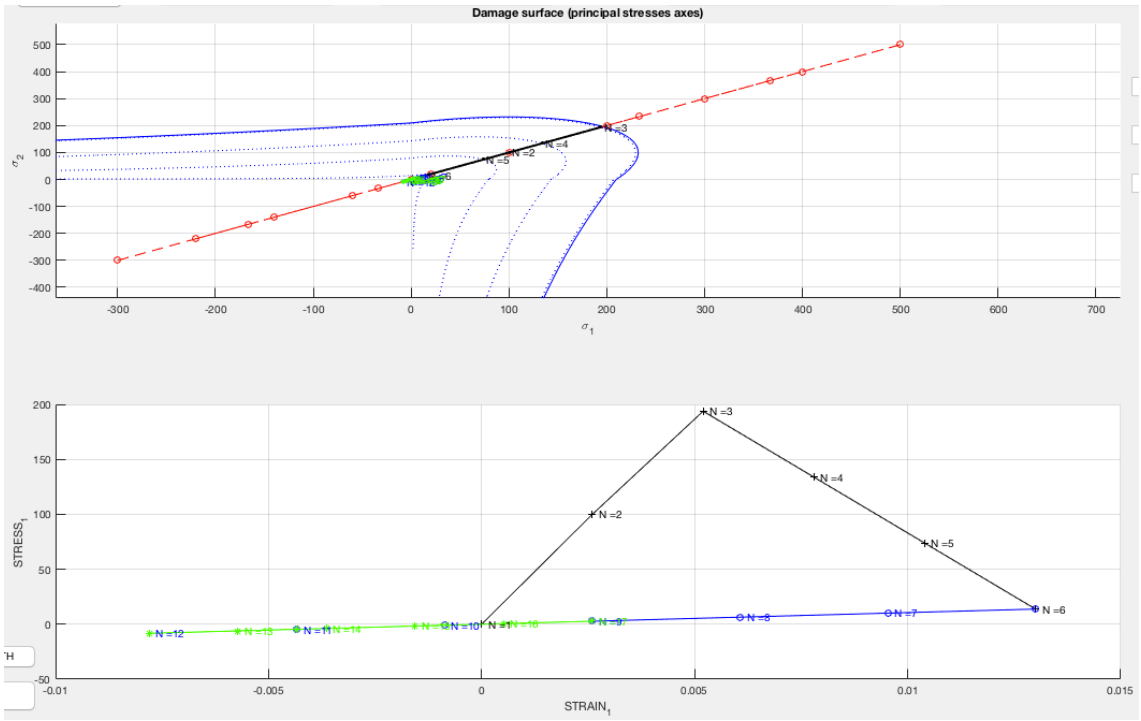
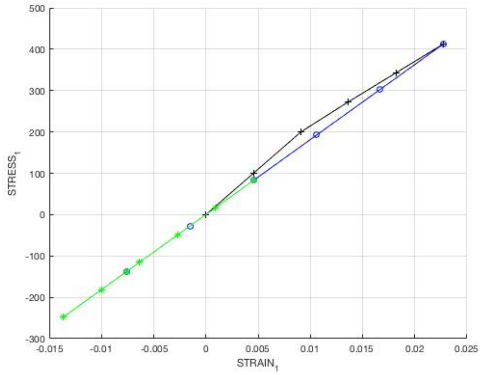


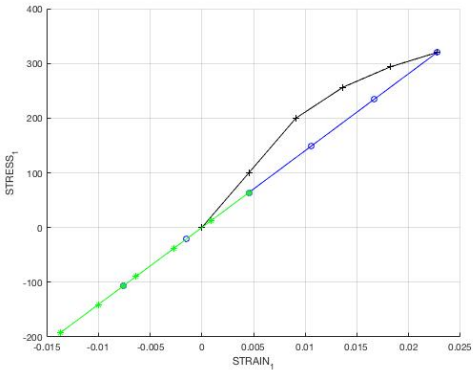
Figure 14 Load path for only-tension case 2. Inviscid, H= -0.6

The results for the only-tension scheme are the exact same as for the non-symmetric, thus, the explanation is the same.

I.IV Linear/exponential

We can show the difference of linear and exponential loading/unloading by taking as an example one of the previous cases and increasing slightly the value of the hardening coefficient. We took H = 0.9 and the results shows that, the linear loading takes a straight path just after crossing the damage surface, but the exponential case clearly decreases gradually such as:

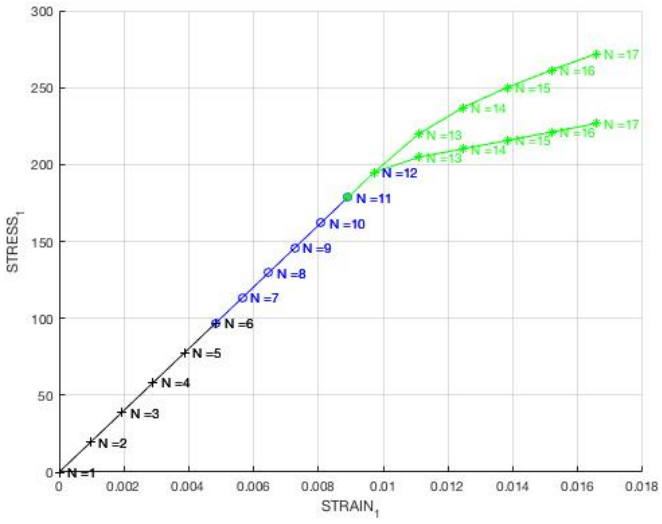




Part II: Rate dependent models

Part II.I Viscous/non Viscous

For this second part, the viscous model has to be coded. We will have to modify the functions “modelos_de_dano” and “damage_main” so we can include the strain tensor at n and n+1 time steps. After having included the strain tensor, we need to make sure that the code reads if we want inviscid or viscous solution, we will implement a simple condition for each case.



We can see the results using viscous and non viscous solutions in the symmetric case such:

The lower curve shows the inviscid case; we can see that the slope is notably lower than the viscous case after crossing the damage surface. Both behave the same way in the elastic domain, however, we can see through the

theory, that this is explained because the viscous case has one more term in the formula, which explains why it escalates further.

Part II.2 Correctness of the implementation

For this last part of the assignment, we need to keep some parameters constants such as: Poisson ration (we are going to take the given one; 0.3) and a constant hardening coefficient, which will be 0.4. We will also be recycling the loading path used in the previous exercice which is a simple uniaxial tension compression:

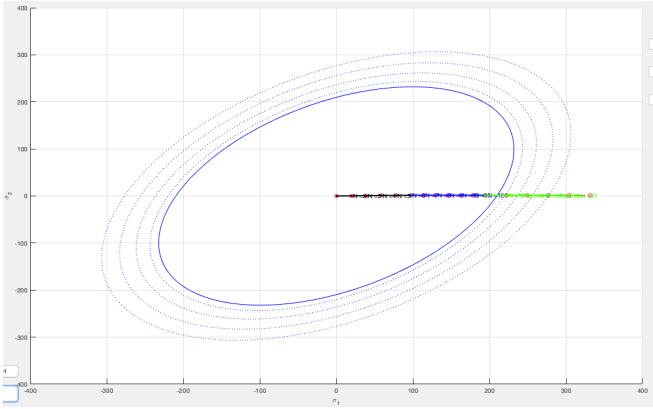


Figure 16 Load path used for Viscous example

By changing the value of he viscosity from 0.01-0.1-0.3-0.5-0.9 we get this results:

The steepest line is the one corresponding to the most higher value of eta (0.9). We must also say that we get an oscillating curve for eta = 0.01. The solutions within are the ones listed above. This solutions make sense since we know that the stress rate will vary increasing in the case the viscous coefficient increases and decreasing the other way round.

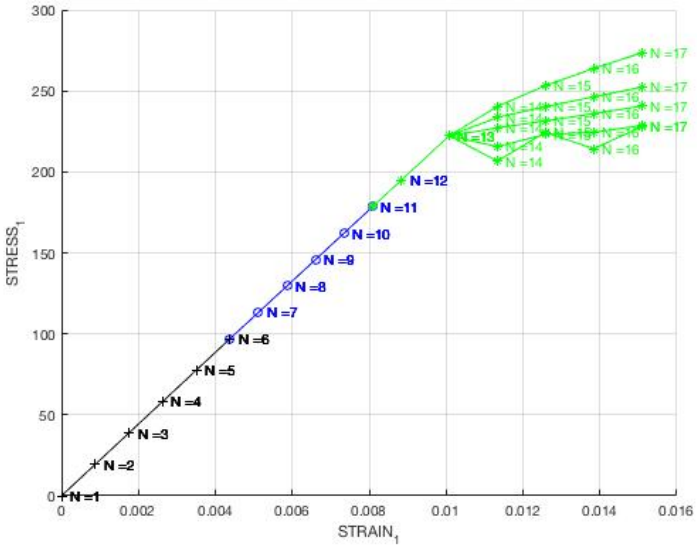


Figure 17 Different Viscous values example

In the next figure we can see how it affects the strain rate in our solutions. The steepest slope shows the lowest strain rate. To make it vary, we will need to modify our Time Int. variable.

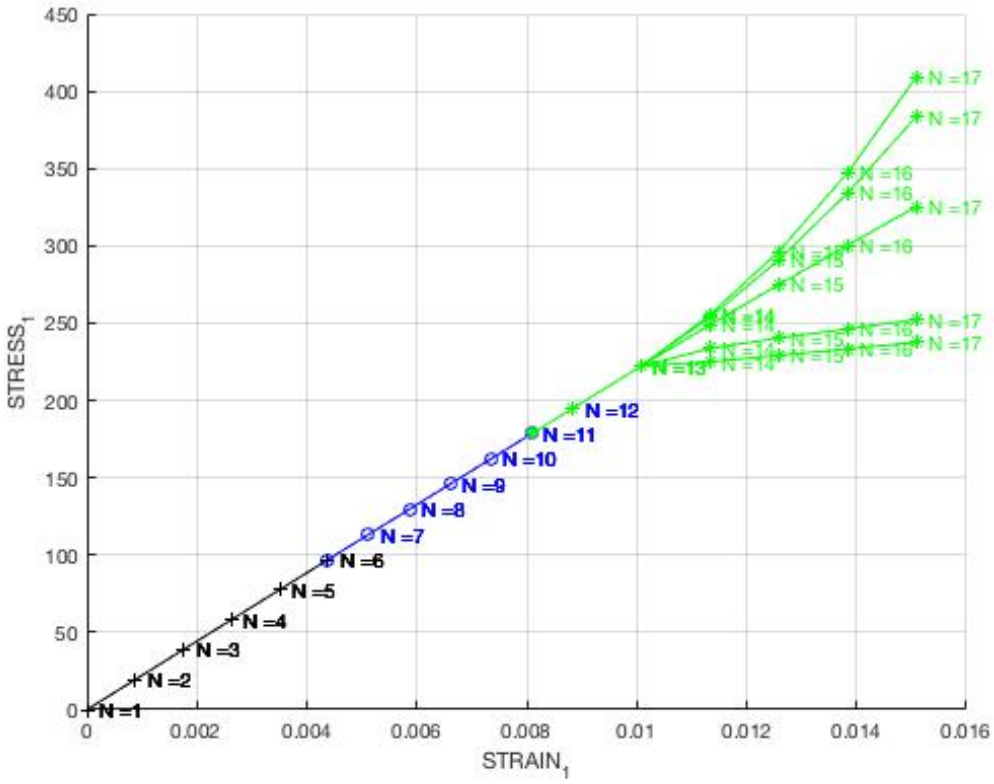


Figure 18 Time int. variation example

From up to down: 0.1 – 0.5 – 2 – 5 – 10.

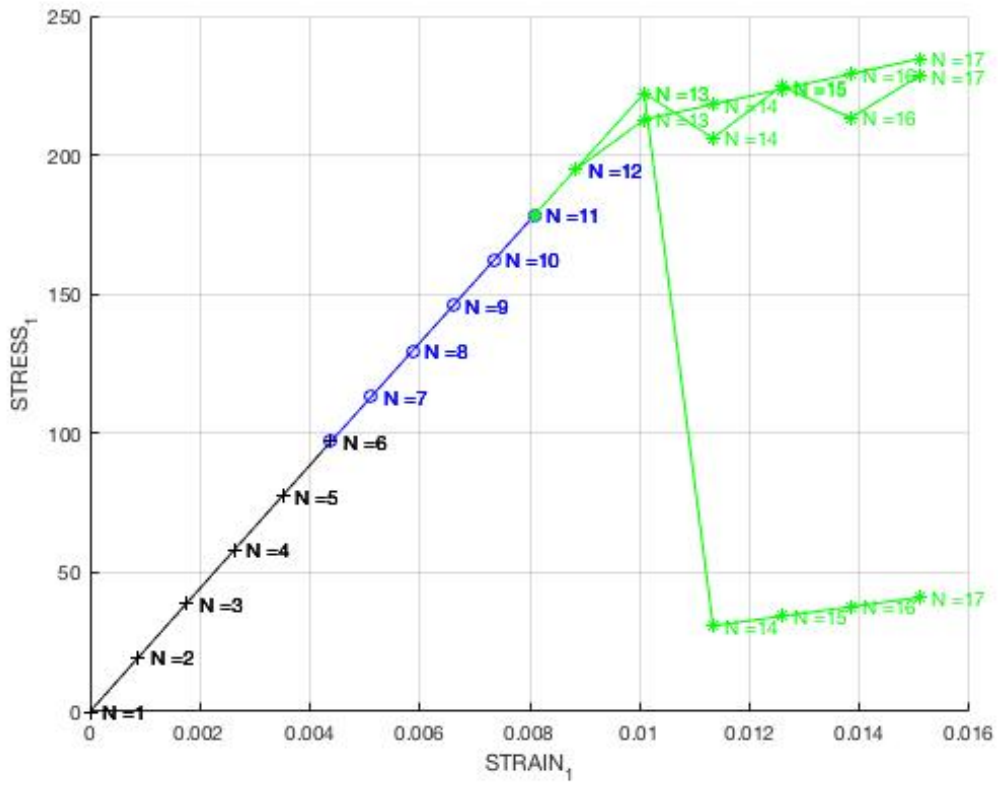


Figure 19 Example for some different alpha values

As we can see in the figure above, modifying the value of alpha, makes big differences when it comes to compute our solution. The first and steepest slope corresponds to the value of alpha = 1, which corresponds to the implicit and always stable method Backward Euler. If we take a look at the one which oscillates in between, corresponds to alpha = 0.5, Crank Nicholson method, and finally, we've got the explicit Forward Euler for alpha = 0 which is the one left.

Conclusion

In this assignment we had the chance to introduce ourselves in the damage model for computational solid mechanics. In a first part, we had to integrate two different cases apart from the already implemented symmetric tension/compression and see if they performed as expected. In a second instance we had to check for those methods if they were having logical solutions for the proposed problems. The last part was to introduce the viscosity parameter into the equation and see how it affected the whole system. As an added part we were supposed to make vary the different parameters given such as time integration and alpha and explain their role.

Annexes

Annex 1: Modelos_de_dano.m

```
function [rtrial,taw_n1] = Modelos_de_dano1
(MDtype,ce,eps_n1,n,eps_n,viscpr,ALPHA)
%*****************************************************************
*****************
%*          Defining damage criterion surface
%*
%*
%*
%*
%*                            MDtype=  1      : SYMMETRIC
%*
%*                            MDtype=  2      : ONLY TENSION
%*
%*                            MDtype=  3      : NON-SYMMETRIC
%*
%*
%*
%*
%*
%* OUTPUT:
%*
%*                            rtrial
%*
%*****************************************************************
*****************



%*****************************************************************
*****************
if (MDtype==1)      %* Symmetric
    taw_n = sqrt(eps_n*ce*eps_n') ;
    taw_n1 = sqrt(eps_n1*ce*eps_n1') ;

    if viscpr == 0
        rtrial= sqrt(eps_n1*ce*eps_n1') ;

    elseif viscpr == 1

        rtrial = (1-ALPHA)*taw_n + ALPHA*taw_n1;
    end

elseif (MDtype==2)  %* Only tension

    sigma_e_n = eps_n*ce;
    sigma_e_n1 = ce*eps_n1';

    sigma_e_plus_n(:) = sigma_e_n(:).*(sigma_e_n(:)>0);
    sigma_e_plus_n1(:) = sigma_e_n1(:).*(sigma_e_n1(:)>0);
```

```
    taw_n = sqrt(sigma_e_plus_n*eps_n') ;
    taw_n1 = sqrt(sigma_e_plus_n1*eps_n1') ;

    if viscpr == 0
        rtrial= sqrt(sigma_e_plus_n1*eps_n1');
    elseif viscpr == 1
        rtrial = (1-ALPHA)*taw_n + ALPHA*taw_n1;
    end

elseif (MDtype==3)  %*Non-symmetric

    sigma_e_n = eps_n*ce;
    sigma_e_n1 = eps_n1*ce;

    sigma_e_plus_n(:) = sigma_e_n(:).*(sigma_e_n(:)>0);
    sigma_e_plus_n1(:) = sigma_e_n1(:).*(sigma_e_n1(:)>0);

    theta_n =
(sigma_e_plus_n(1)+sigma_e_plus_n(2))/(abs(sigma_e_n(1))+abs(sigma_e_n
(2)));
    theta_n1 =
(sigma_e_plus_n1(1)+sigma_e_plus_n1(2))/(abs(sigma_e_n1(1))+abs(sigma_
e_n1(2)));

    taw_n= (theta_n+(1-theta_n)/n)*sqrt(eps_n*ce*eps_n');
    taw_n1= (theta_n1+(1-theta_n1)/n)*sqrt(eps_n1*ce*eps_n1');

    if viscpr == 0
        rtrial= (theta_n1+(1-theta_n1)/n)*sqrt(eps_n1*ce*eps_n1');
    elseif viscpr == 1
        rtrial = (1-ALPHA)*taw_n + ALPHA*taw_n1;
    end

end
%*****************************************************************
*****************
return
```

Annex 2: damage_main

```
function [sigma_n1,hvar_n1,aux_var] = rmap_dano1
(eps_n1,hvar_n,Eprop,ce,MDtype,n,eps_n,delta_t)

%*****************************************************************
*****************
%*                                          *
%*           Integration Algorithm for a isotropic damage model
%*
%*
*
%*            [sigma_n1,hvar_n1,aux_var] = rmap_dano1
(eps_n1,hvar_n,Eprop,ce)        *
%*
*
%* INPUTS              eps_n1(4)   strain (almansi)    step n+1
*
%*                                 vector R4    (exx eyy exy ezz)
```

```
*
%*                      hvar_n(6)   internal variables , step n
*
%*                                  hvar_n(1:4) (empty)
*
%*                                  hvar_n(5) = r  ; hvar_n(6)=q
*
%*                      Eprop(:)    Material parameters
*
%*
%*                      ce(4,4)     Constitutive elastic tensor
*
%*
*
%* OUTPUTS:             sigma_n1(4) Cauchy stress  , step n+1
*
%*                      hvar_n(6)   Internal variables , step n+1
*
%*                      aux_var(3)  Auxiliar variables for computing
const. tangent tensor  *
%*****************************************************************
******************


hvar_n1 = hvar_n;       %DÈclaration des variables
r_n     = hvar_n(5);
q_n     = hvar_n(6);
E       = Eprop(1);
nu      = Eprop(2);
H       = Eprop(3);
sigma_u = Eprop(4);
hard_type = Eprop(5) ;
viscpr = Eprop(6) ;
eta = Eprop(7);
ALPHA = Eprop(8);
%*****************************************************************
***************


%*****************************************************************
***************
%*       initializing
%*
 r0 = sigma_u/sqrt(E);
 zero_q=1.d-6*r0;
% if(r_n<=0.d0)
%     r_n=r0;
%     q_n=r0;
% end
%*****************************************************************
***************


%*****************************************************************
***************
%*       Damage surface
%*
[rtrial,taw_n1] = Modelos_de_dano1
(MDtype,ce,eps_n1,n,eps_n,viscpr,ALPHA);
%*****************************************************************
***************
```

```
%*****************************************************************
***************
%*   Ver el Estado de Carga
%*
%*   --------->    fload=0 : elastic unload
%*
%*   --------->    fload=1 : damage (compute algorithmic constitutive
tensor)          %*
%% implementing exponential hardening for inviscid and viscous case

fload=0;

if(rtrial > r_n)
%     *   Loading
    fload=1;
    delta_r = rtrial-r_n;

    if viscpr == 0  %inviscid case
        r_n1= rtrial  ;
    elseif viscpr == 1 %viscous case
        Z = (eta-(1-ALPHA)*delta_t)/(eta+ALPHA*delta_t);
        Y = delta_t/(eta+ALPHA*delta_t);
        r_n1 = Z*r_n + Y*rtrial;
    end

    if hard_type == 0
%          Linear
        H_n1 = H;
        q_n1= q_n+ H*delta_r;
    elseif hard_type == 1
        q_inf = r0+(r0-zero_q);
%         exponential
        if H>0
            H_n1 = H*((q_inf-r0)/r0)*exp(H*(1-rtrial/r0));
            q_n1= q_n+ ((H*(q_inf-r0)/r0)*exp(H*(1-
rtrial/r0)))*delta_r;
        elseif H<0
            H_n1 = H*((q_inf-r0)/r0)*(1/exp(H*(1-rtrial/r0)));
            q_n1 = q_n+ ((H*(q_inf-r0)/r0)*(1/exp(H*(1-
rtrial/r0))))*delta_r;
        end
    end

    if(q_n1<zero_q)
        q_n1=zero_q;
    end
else
%     *     Elastic load/unload
    fload=0;
    r_n1= r_n  ;
    q_n1= q_n  ;
end

%%
% Damage variable
% ---------------
dano_n1 = 1-(q_n1/r_n1);  %damage parameter
sigma_n1  =(1-dano_n1)*ce*eps_n1';

%hold on
```

```
%plot(sigma_n1(1),sigma_n1(2),'bx')


%% tangent constitutive equation
if viscpr == 1
   if rtrial > r_n
       Ce_alg_n1 = (1-
dano_n1)*ce+((ALPHA*delta_t)/(eta+ALPHA*delta_t))*...
           (1/taw_n1)*((H_n1*r_n1-
q_n1)/(r_n1^2))*((ce*eps_n1')'*(ce*eps_n1'));
       C_alg = Ce_alg_n1(1,1);

       Ce_tan_n1=(1-dano_n1)*ce;
       C_tan = Ce_tan_n1(1,1);
       hvar_n1(8)= C_alg;
       hvar_n1(9)= C_tan;
   elseif rtrial <= r_n

       Ce_alg_n1 = (1-dano_n1)*ce;
       C_alg = Ce_alg_n1(1,1);

       Ce_tan_n1 = Ce_alg_n1;
       C_tan = Ce_tan_n1(1,1);
       hvar_n1(8)= C_alg;
       hvar_n1(9)= C_tan;
   end
end

%%
%*****************************************************************
***************

%  Computing stress
%  ****************
%sigma_n1  =(1.d0-dano_n1)*ce*eps_n1';
%hold on
%plot(sigma_n1(1),sigma_n1(2),'bx')

%*****************************************************************
***************


%*****************************************************************
***************
%* Updating historic variables
%*
%  hvar_n1(1:4)  = eps_n1p;

hvar_n1(5)= r_n1;
hvar_n1(6)= q_n1 ;
hvar_n1(7)=dano_n1;


%*****************************************************************
***************
%* Updating historic variables
%*
%  hvar_n1(1:4)  = eps_n1p;
hvar_n1(5)= r_n1 ;
hvar_n1(6)= q_n1 ;
```

```
%*********************************************************************
***************

%*********************************************************************
***************
%* Auxiliar variables
%*
aux_var(1) = fload;
aux_var(2) = q_n1/r_n1;
%*aux_var(3) = (q_n1-H*r_n1)/r_n1^3;
%*********************************************************************
***************

end
```

Annex 3: rmap_dano1

```
function [sigma_n1,hvar_n1,aux_var] = rmap_dano1
(eps_n1,hvar_n,Eprop,ce,MDtype,n,eps_n,delta_t)

%*********************************************************************
****************
%*                                          *
%*           Integration Algorithm for a isotropic damage model
%*
%*
*
%*            [sigma_n1,hvar_n1,aux_var] = rmap_dano1
(eps_n1,hvar_n,Eprop,ce)        *
%*
*
%* INPUTS              eps_n1(4)   strain (almansi)    step n+1
*
%*                                 vector R4    (exx eyy exy ezz)
*
%*                     hvar_n(6)   internal variables , step n
*
%*                                 hvar_n(1:4) (empty)
*
%*                                 hvar_n(5) = r  ; hvar_n(6)=q
*
%*                     Eprop(:)    Material parameters
*
%*
%*                     ce(4,4)     Constitutive elastic tensor
*
%*
*
%* OUTPUTS:            sigma_n1(4) Cauchy stress  , step n+1
*
%*                     hvar_n(6)   Internal variables , step n+1
*
```

```
%*                     aux_var(3)  Auxiliar variables for computing
const. tangent tensor  *
%*********************************************************************
*****************


hvar_n1 = hvar_n;       %DÈclaration des variables
r_n     = hvar_n(5);
q_n     = hvar_n(6);
E       = Eprop(1);
nu      = Eprop(2);
H       = Eprop(3);
sigma_u = Eprop(4);
hard_type = Eprop(5) ;
viscpr = Eprop(6) ;
eta = Eprop(7);
ALPHA = Eprop(8);
%*********************************************************************
***************


%*********************************************************************
***************
%*       initializing
%*
 r0 = sigma_u/sqrt(E);
 zero_q=1.d-6*r0;
% if(r_n<=0.d0)
%     r_n=r0;
%     q_n=r0;
% end
%*********************************************************************
***************


%*********************************************************************
***************
%*       Damage surface
%*
[rtrial,taw_n1] = Modelos_de_dano1
(MDtype,ce,eps_n1,n,eps_n,viscpr,ALPHA);
%*********************************************************************
***************


%*********************************************************************
***************
%*   Ver el Estado de Carga
%*
%*   --------->    fload=0 : elastic unload
%*
%*   --------->    fload=1 : damage (compute algorithmic constitutive
tensor)         %*
%% implementing exponential hardening for inviscid and viscous case

fload=0;

if(rtrial > r_n)
%     *   Loading
    fload=1;
    delta_r = rtrial-r_n;
```

```
    if viscpr == 0  %inviscid case
        r_n1= rtrial  ;
    elseif viscpr == 1 %viscous case
        Z = (eta-(1-ALPHA)*delta_t)/(eta+ALPHA*delta_t);
        Y = delta_t/(eta+ALPHA*delta_t);
        r_n1 = Z*r_n + Y*rtrial;
    end

    if hard_type == 0
%          Linear
        H_n1 = H;
        q_n1= q_n+ H*delta_r;
    elseif hard_type == 1
        q_inf = r0+(r0-zero_q);
%         exponential
        if H>0
            H_n1 = H*((q_inf-r0)/r0)*exp(H*(1-rtrial/r0));
            q_n1= q_n+ ((H*(q_inf-r0)/r0)*exp(H*(1-
rtrial/r0)))*delta_r;
        elseif H<0
            H_n1 = H*((q_inf-r0)/r0)*(1/exp(H*(1-rtrial/r0)));
            q_n1 = q_n+ ((H*(q_inf-r0)/r0)*(1/exp(H*(1-
rtrial/r0))))*delta_r;
        end
    end

    if(q_n1<zero_q)
        q_n1=zero_q;
    end
else
%     *     Elastic load/unload
    fload=0;
    r_n1= r_n  ;
    q_n1= q_n  ;
end

%%
% Damage variable
% ---------------
dano_n1 = 1-(q_n1/r_n1);  %damage parameter
sigma_n1  =(1-dano_n1)*ce*eps_n1';

%hold on
%plot(sigma_n1(1),sigma_n1(2),'bx')


%% tangent constitutive equation
if viscpr == 1
   if rtrial > r_n
       Ce_alg_n1 = (1-
dano_n1)*ce+((ALPHA*delta_t)/(eta+ALPHA*delta_t))*...
           (1/taw_n1)*((H_n1*r_n1-
q_n1)/(r_n1^2))*((ce*eps_n1')'*(ce*eps_n1'));
       C_alg = Ce_alg_n1(1,1);

       Ce_tan_n1=(1-dano_n1)*ce;
       C_tan = Ce_tan_n1(1,1);
       hvar_n1(8)= C_alg;
       hvar_n1(9)= C_tan;
   elseif rtrial <= r_n
```

```
        Ce_alg_n1 = (1-dano_n1)*ce;
        C_alg = Ce_alg_n1(1,1);

        Ce_tan_n1 = Ce_alg_n1;
        C_tan = Ce_tan_n1(1,1);
        hvar_n1(8)= C_alg;
        hvar_n1(9)= C_tan;
    end
end

%%
%*****************************************************************
***************

%  Computing stress
%  ***************
%sigma_n1  =(1.d0-dano_n1)*ce*eps_n1';
%hold on
%plot(sigma_n1(1),sigma_n1(2),'bx')

%*****************************************************************
***************


%*****************************************************************
***************
%* Updating historic variables
%*
%  hvar_n1(1:4)  = eps_n1p;

hvar_n1(5)= r_n1;
hvar_n1(6)= q_n1 ;
hvar_n1(7)=dano_n1;


%*****************************************************************
***************
%* Updating historic variables
%*
%  hvar_n1(1:4)  = eps_n1p;
hvar_n1(5)= r_n1 ;
hvar_n1(6)= q_n1 ;
%*****************************************************************
***************




%*****************************************************************
***************
%* Auxiliar variables
%*
aux_var(1) = fload;
aux_var(2) = q_n1/r_n1;
%*aux_var(3) = (q_n1-H*r_n1)/r_n1^3;
%*****************************************************************
***************

end
```

Annex 4: dibujar_criterio_dano1

```
function hplot = dibujar_criterio_dano1(ce,nu,q,tipo_linea,MDtype,n)
%*****************************************************************
***************
%*                  PLOT DAMAGE SURFACE CRITERIUM: ISOTROPIC MODEL
%*
%*
%*
%*      function [ce] = tensor_elastico (Eprop, ntype)
%*
%*
%*
%*      INPUTS
%*
%*
%*
%*                     Eprop(4)    vector de propiedades de material
%*
%*                                          Eprop(1)=  E------>modulo de
Young          %*
%*                                          Eprop(2)=  nu----->modulo de
Poisson        %*
%*                                          Eprop(3)=  H----->modulo de
Softening/hard. %*
%*                                          Eprop(4)=sigma_u-----
>tensiÔøΩn ÔøΩltima        %*
%*                       ntype                                    %*
%*                                   ntype=1  plane stress
%*
%*                                   ntype=2  plane strain
%*
%*                                   ntype=3  3D
%*
%*                       ce(4,4)     Constitutive elastic tensor  (PLANE
S.       )    %*
%*                       ce(6,6)                                  ( 3D)
%*
%*****************************************************************
***************


%*****************************************************************
***************
%*        Inverse ce
%*
ce_inv=inv(ce);
c11=ce_inv(1,1);
c22=ce_inv(2,2);
c12=ce_inv(1,2);
c21=c12;
c14=ce_inv(1,4);
c24=ce_inv(2,4);
```

```
%*********************************************************************
*****************
```

```
%*********************************************************************
*****************
% POLAR COORDINATES
if MDtype==1
    tetha=[0:0.01:2*pi];

%*********************************************************************
*****************
    %* RADIUS
    D=size(tetha);                          %*  Range
    m1=cos(tetha);                          %*
    m2=sin(tetha);                          %*
    Contador=D(1,2);                        %*


    radio = zeros(1,Contador) ;
    s1    = zeros(1,Contador) ;
    s2    = zeros(1,Contador) ;

    for i=1:Contador
        radio(i)= q/sqrt([m1(i) m2(i) 0
nu*(m1(i)+m2(i))]*ce_inv*[m1(i) m2(i) 0 ...
            nu*(m1(i)+m2(i))]');

        s1(i)=radio(i)*m1(i);
        s2(i)=radio(i)*m2(i);

    end
    hplot =plot(s1,s2,tipo_linea);


elseif MDtype==2

    tetha=[0:0.01:2*pi];
    D=size(tetha);                          %*  Range
    m1=cos(tetha);                          %* Ici on fait exactement
pareil que en MD=1
    m2=sin(tetha);                          %*
    Contador=D(1,2);                        %*


    radio = zeros(1,Contador) ;
    s1    = zeros(1,Contador) ;
    s2    = zeros(1,Contador) ;

    for i=1:Contador
        % McAuly Braket x*(x>0) if x>0 I obtain 1, Otherwise I obtain
a 0
        A = m1(i)*(m1(i)>0);
        B = m2(i)*(m2(i)>0);
```

```
        radio(i)= q/sqrt([A B 0 nu*(A+B)]*ce_inv*[m1(i) m2(i) 0 ...
            nu*(m1(i)+m2(i))]');

        s1(i)=radio(i)*m1(i);
        s2(i)=radio(i)*m2(i);

    end
    hplot =plot(s1,s2,tipo_linea);



elseif MDtype==3

    tetha=[0:0.01:2*pi];

%*********************************************************************
*****************
    %* RADIUS
    D=size(tetha);                          %*  Range
    m1=cos(tetha);                          %* Ici on fait exactement
pareil que en MD=1 et MD=2
    m2=sin(tetha);                          %*
    Contador=D(1,2);                        %*


    radio = zeros(1,Contador) ;
    s1    = zeros(1,Contador) ;
    s2    = zeros(1,Contador) ;

    for i=1:Contador
        % McAuly Braket x*(x>0) if x>0 I obtain 1, Otherwise I obtain
a 0
        A = m1(i)*(m1(i)>0);        %Cette partie l‡ Áa change un peut
car radio prend une autre forme
        B = m2(i)*(m2(i)>0);           %Simple application de la
theorie des diapos 4 ‡ partir de page 15
        alpha_N =A+B;
        alpha_D =abs(m1(i))+abs(m2(i));
        alpha = alpha_N/alpha_D;
        radio(i)= q/((alpha+(1-alpha)/n)*(sqrt([m1(i) m2(i) 0
nu*(m1(i)+m2(i))]*ce_inv*[m1(i) m2(i) 0 ...
            nu*(m1(i)+m2(i))]')));
        s1(i)=radio(i)*m1(i);
        s2(i)=radio(i)*m2(i);

    end
    hplot =plot(s1,s2,tipo_linea);


end
%*********************************************************************
****************


%*********************************************************************
*****************
return
```