Master's Degree Numerical Methods in Engineering

Computational Solid Mechanics

Homework 1: Implementation of Damage Model using Matlab

Author:
Luis Ángel Avilés Murcia
luis.angel.aviles@upc.edu

Professors:
Alfredo, Huespe
Joaquín A., Hernández O.

March 29, 2020
Academic Year 2019-2020

Contents

1 Description

This work shows the implementation of some components of a Damage model for both time-independent and time-dependent materials. A code made in Matlab is used and completed to get the full stress-strain response inviscid and viscous materials. This report explains the assumptions made to programming the code, the restrictions or conditionals made to the code. New implemented parts of the code are shown and a set of figures proves the correct implementation of the code for different loading path conditions.

2 Part I - Rate independent models

Two models were implement in the supplied Matlab code: Tension-only damage model and Non-symmetric tension-compression damage model. Next sections explain where was implemented the code inside the Matlab file and show a plot of the initial damage surface for each case in comparison with the symmetric model (already implemented). Additional, the path at the stress space and the stress-strain curves are plotted for each case.

2.1 Loading paths

The following are the stress path used to assess the implementation.

Path 1 - Uniaxial

σ_1 = 300 kPa	σ_1 = -500 kPa	σ_1 = 800 kPa
σ_2 = 0 kPa	σ_2 = 0 kPa	σ_2 = 0 kPa

Path 2 - Uniaxial, biaxial

σ_1 = 300 kPa	σ_1 = -500 kPa	σ_1 = 800 kPa
σ_2 = 0 kPa	σ_2 = -500 kPa	σ_2 = 800 kPa

Path 3 - Biaxial

σ_1=300 kPa	σ_1=-500 kPa	σ_1=800 kPa
σ_2=300 kPa	σ_2=-500 kPa	σ_2=800 kPa

2.2 Material parameters

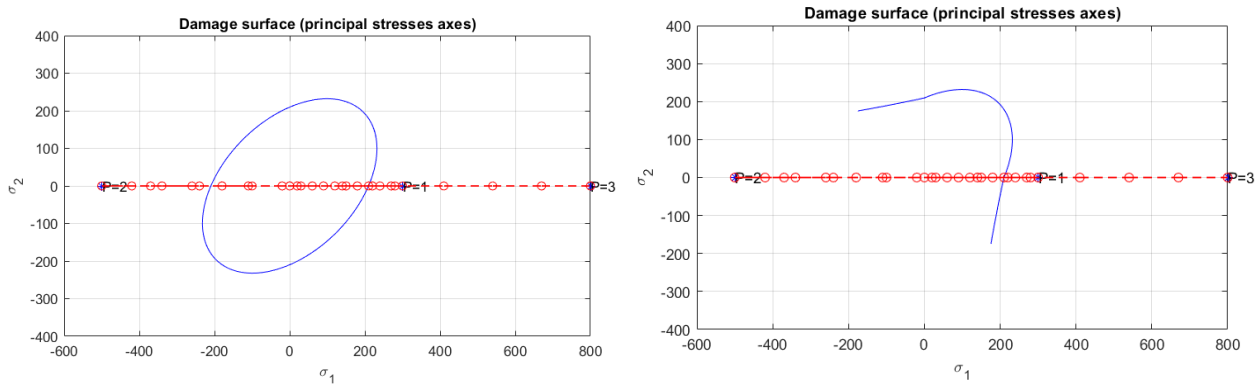
The material used to assess the correctness of the code implemented have the following properties for all the cases. When any parameter change it will be mentioned.

Parameter	Value
Young Modulus	20000 kPa
Poisson	0.3
Hardening Modulus	0.5
Softening Modulus	-0.5
Ratio compression/tension (n)	3.0
yield stress	200

Table 1: Material parameters

2.3 Tension-only damage model

The Matlab code for Tension-only was implemented in the function `Dibujar_criterio_dano1` for the case when `MDtype==2`, which correspond to Tension, defined in the file `main_nointeractive`. Path 1 was used to plot the initial surface.

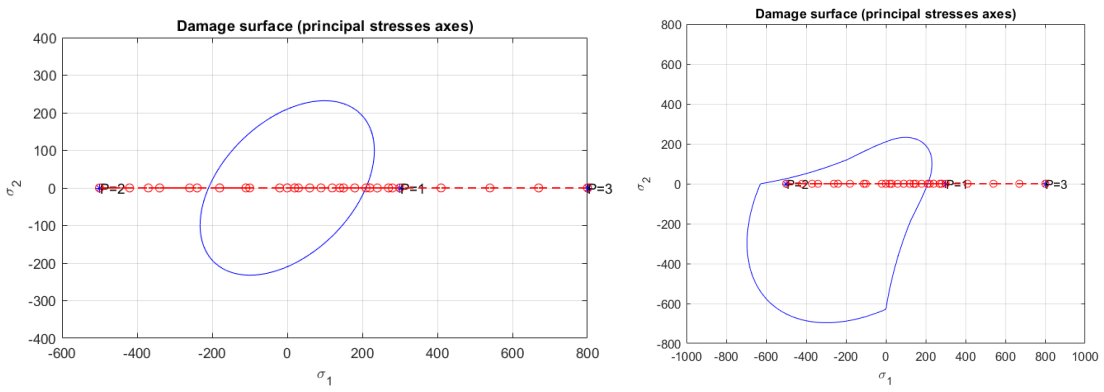


(a) Symmetric (b) Tension-only

Figure 1: Initial surface for Tension-only damage model implementation

2.4 Non-symmetric tension-compression damage model

This case of damage model correspond to `MDtype==3` option inside the Matlab code. This model was implement in `Dibujar_criterio_dano1` file, as well as, `modelos_dano1`.



(a) Symmetric (b) Non-symmetric

Figure 2: Initial surface for Non-symmetric tension-compression damage model implementation

2.5 Exponential hardening/softening

This additional case was implement in `rmap_dano1` and it is activated choosing `HARDTYPE` variable equal to `EXPONENTIAL` in the `main_nointeractive` file. To see the correctness of the implementation a symmetric case and the path 1 is used as show by figure 3. To see the difference between the two implementations, a values of $H = \pm 0.5$ was used.

2.6 Loading paths

The following are the stress path used to assess the implementation done.

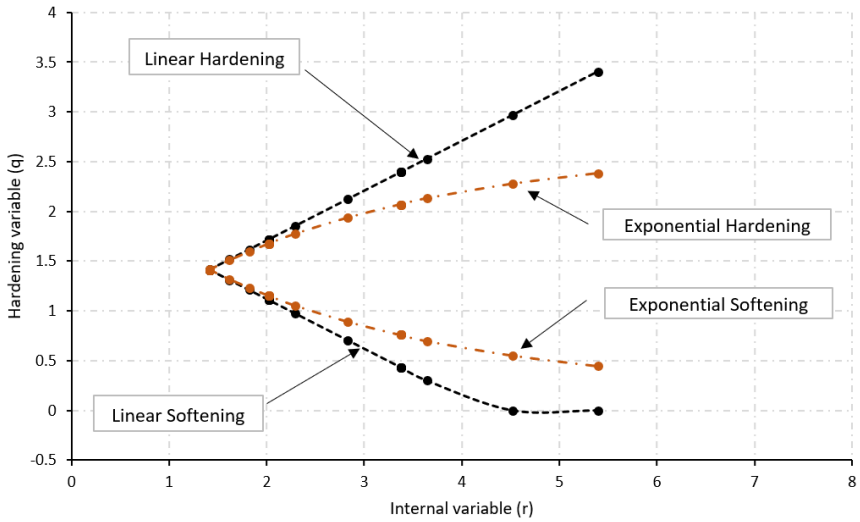
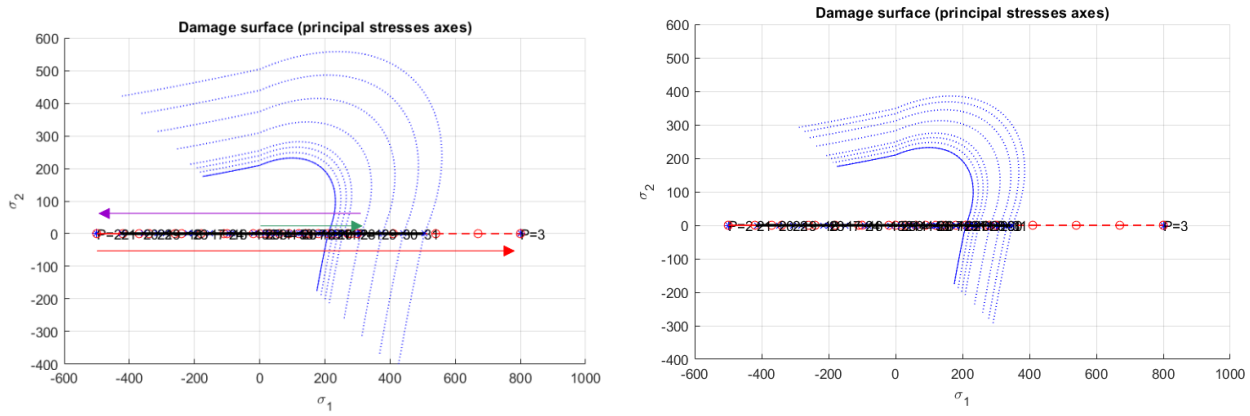


Figure 3: Linear and Exponential behaviour of the damage for Hardening and Softening

The plot has been created using the stress path 1 and the material parameters given in Table 1. The smoothing of the exponential law of evolution and the tendency to an infinity value of hardening variable (q) can be evidenced. On the other hand, the linear value in softening show the constant value of hardening variable after reach a certain value of stress that makes internal variable increase indefinitely at constant q.

2.7 Results

- **Path 1**

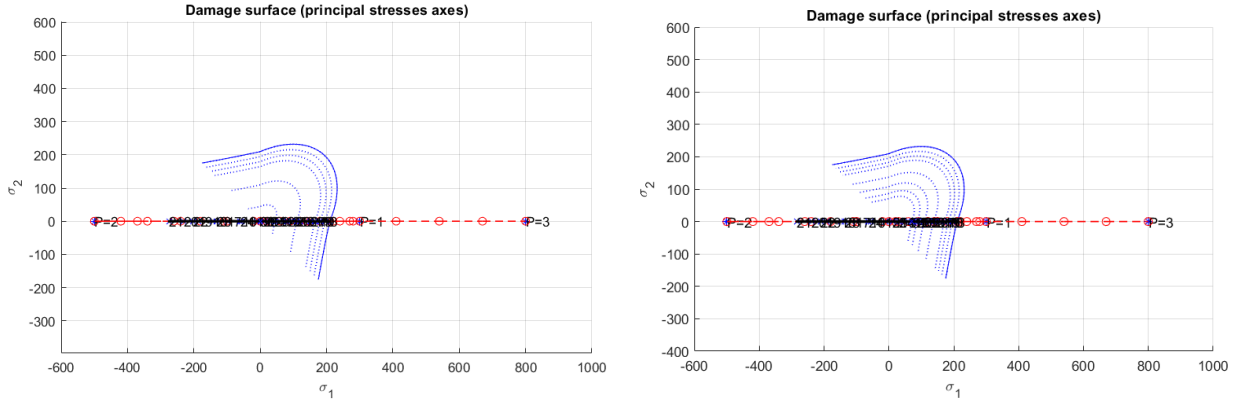


(a) Tension-only Linear

(b) Tension-only Exponential

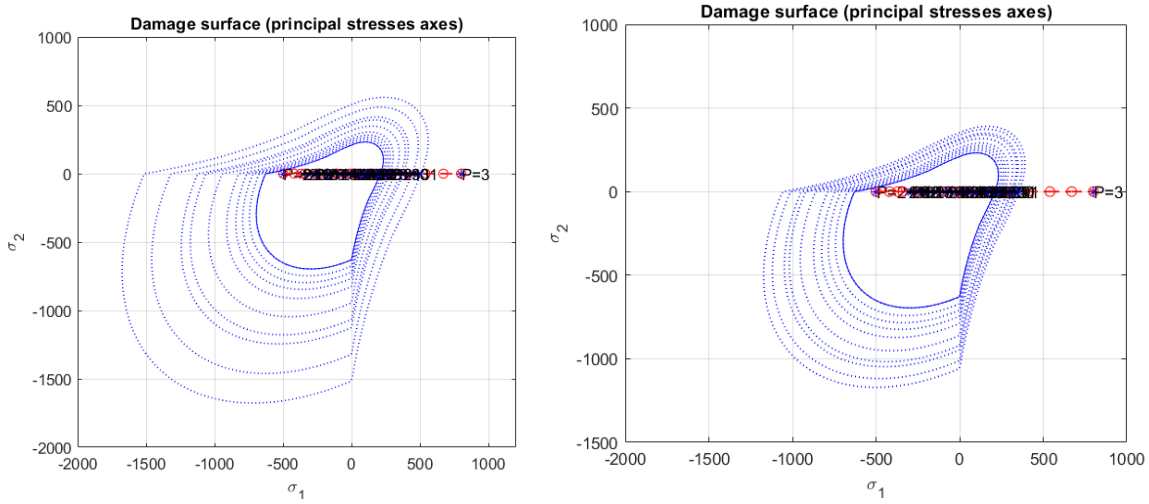
Figure 4: Tension-only damage surface Evolution. Linear and Exponential evolution of **Hardening H=0.5**

Figure 4.a shows the stress path. The green line represents the loading condition (300 kPa, 0) that increases until it reaches the initial damage surface and then, because of the hardening behavior the surface increase. The purple one represents the unloading and compression condition (-300,0). Finally, the red is the reloading condition until the last surface reached by the green route and continually increasing due to the hardening behavior again. The same procedure is followed for other paths.



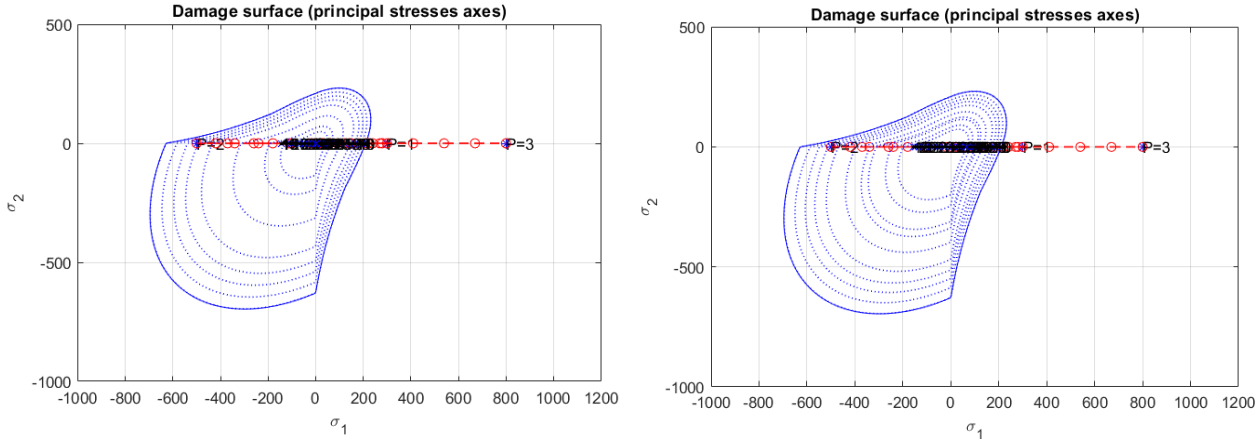
(a) Tension-only Linear (b) Tension-only Exponential

Figure 5: Tension-only damage surface evolution. Linear and Exponential evolution of **Softening H=-0.5**



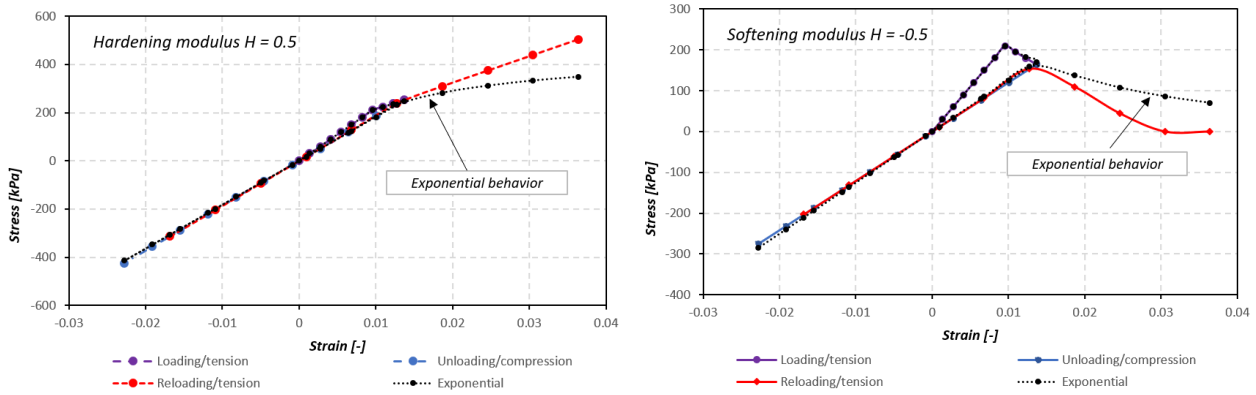
(a) Non-symmetric Linear (b) Non-symmetric Exponential

Figure 6: Non-symmetric model (n=3). Linear and Exponential evolution of **Hardening H=0.5**



(a) Non-symmetric Linear (b) Non-symmetric Exponential

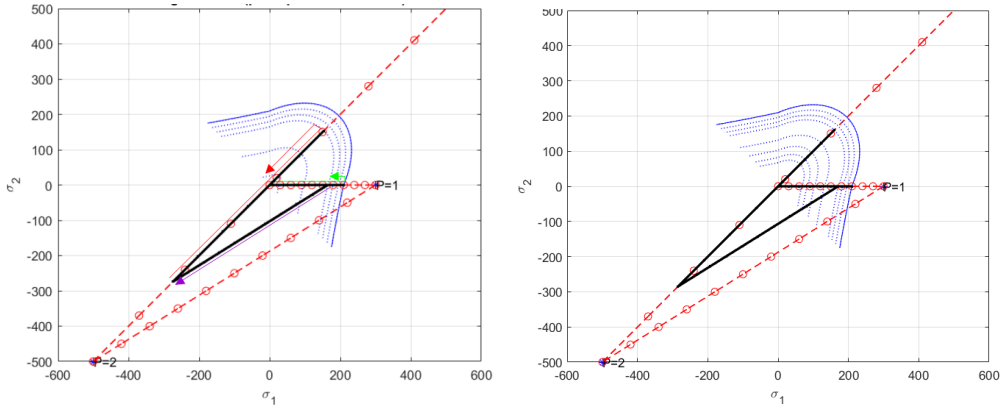
Figure 7: Non-symmetric model (n=3). Linear and Exponential evolution of **Softening H=-0.5**



(a) Non-symmetric Linear (b) Non-symmetric Exponential

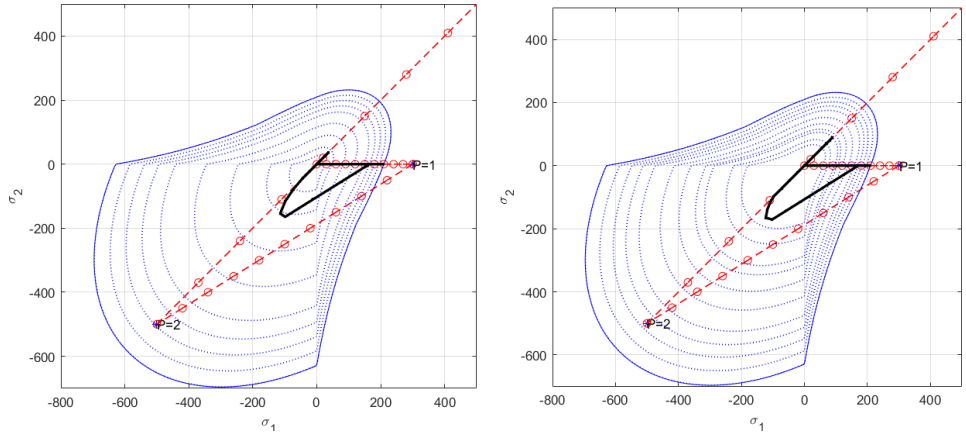
Figure 8: Tension-only Stress-strain Curves. Linear and Exponential evolution

- **Path 2**
  From this part, just figures for softening case are shown, which is the real physical behaviour for damage model.



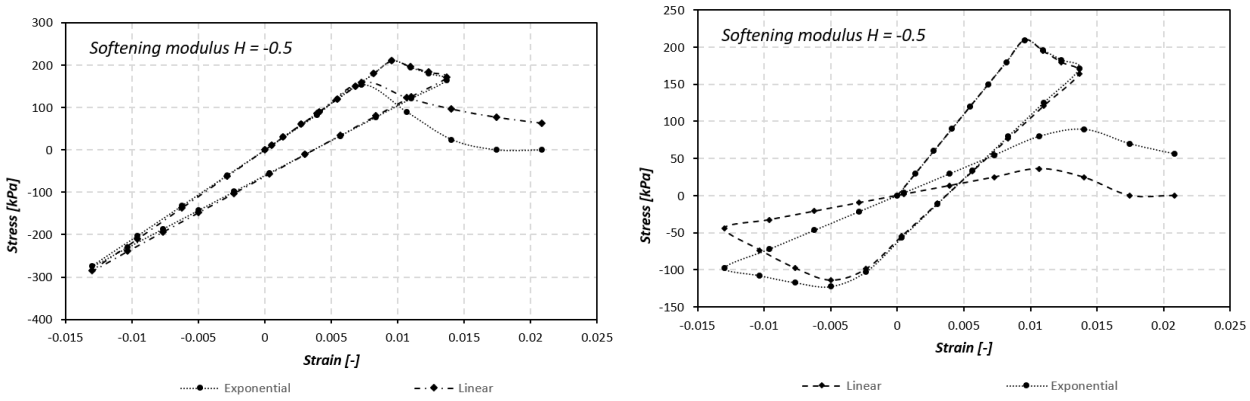
(a) Tension-only Linear (b) Tension-only Exponential

Figure 9: Tension-only damage surface Evolution. Linear and Exponential evolution of **Softening**



(a) Non-symmetric Linear (b) Non-symmetric Exponential

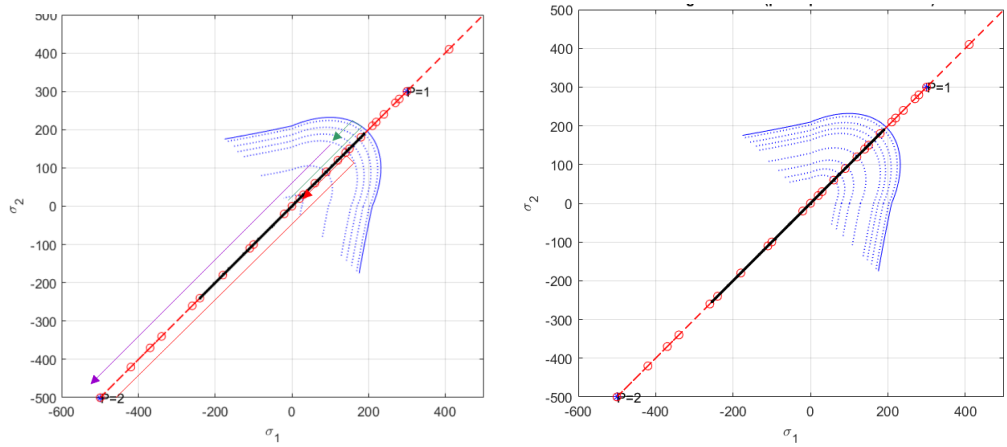
Figure 10: Non-symmetric damage evolution. Linear and Exponential evolution of **Softening H=0.5**



(a) Tension-only (b) Non-symmetric

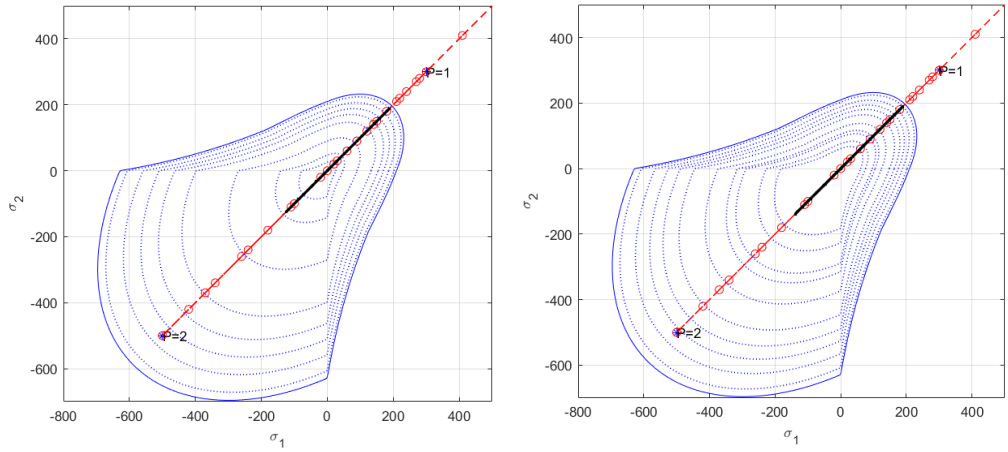
Figure 11: Stress-strain Curves for Tension-only and Non-symmetric damage model. Linear and Exponential evolution

- **Path 3**



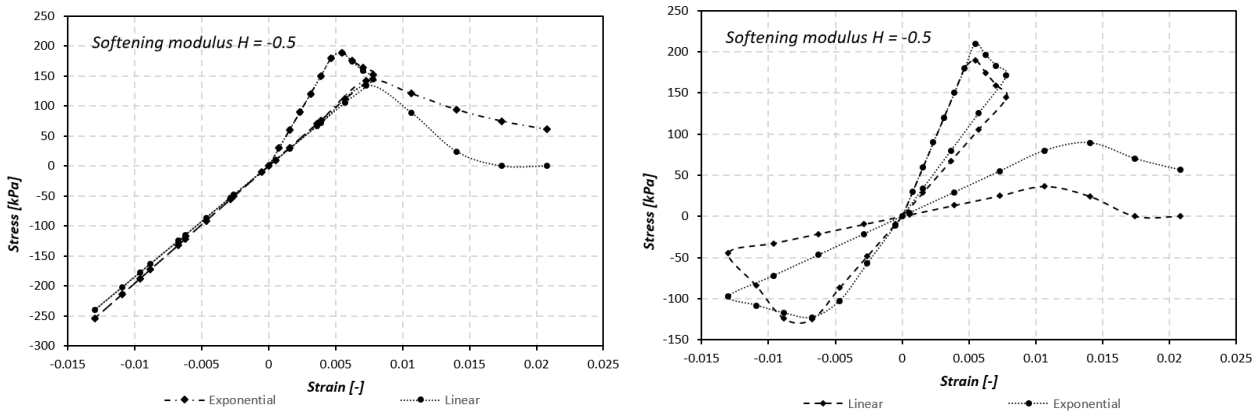
(a) Tension-only Linear (b) Tension-only Exponential

Figure 12: Tension-only damage evolution curves. Linear and Exponential behavior.



(a) Non-symmetric Linear (b) Non-symmetric Exponential

Figure 13: Non-symmetric damage evolution. Linear and Exponential behavior



(a) Tension-only (b) Non-symmetric

Figure 14: Stress-strain Curves for Tension-only and Non-symmetric damage model. Linear and Exponential evolution

2.8 Conclusions Part I

- Tension only damage model wide the area of elastic stresses and restrict the damage surface to positive values of stresses (quadrant 1 of the stress space).
- Linear behavior of the damage produce bigger damage surface than exponential behavior. This is because of unrestricted value of hardening variable (q) for Linear behavior and the limited value of q (q infinite) for the exponential case.
- Non-symmetric damage model scale the damage surface n values for negative stresses (quadrant 3 of the stress space). This model allows taking into account that some material has a different response to compression than to tension loading conditions.
- Stress-strain curves show the reduction of the stiffness value after unloading cycle.
- The damaged surface decreases faster for softening behavior than it could increases for hardening behavior. This means that softening degrade the materials in a faster way (real physical condition) that hardening could do.

3 Part II - Rate dependent models

3.1 Reference parameters and stress configuration

To evaluate the correctness of the implementation the following data and configurations are used for all the cases:

- Symmetric tension-compression model will be used.
- Uniaxial direction stress path will be consider as path of analysis. This path was selected to have constants strain rate increments for all the cases.

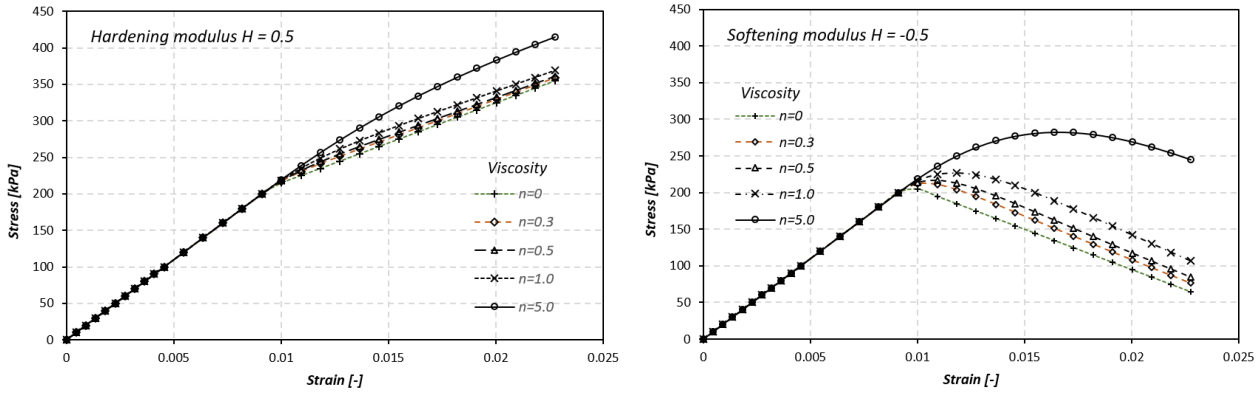
Path - Uniaxial

$\sigma_1 = 200$ kPa	$\sigma_1 = 400$ kPa	$\sigma_1 = 600$ kPa
$\sigma_2 = 0$ kPa	$\sigma_2 = 0$ kPa	$\sigma_2 = 0$ kPa

- Linear evolution will be considered.
- Constant parameter of Young Modulus, Poisson and yield stress are used. The values taken are the same from independent rate model.
- A softening and Hardening modulus of -0.5 and 0.5 are used, respectively.

3.2 Influence of the viscosity parameter η

Five values of the viscosity parameter were used, as it is shown in the figure. For this case, a value of $\alpha = 1.0$ and time increment of 0.33 seg. were taken.

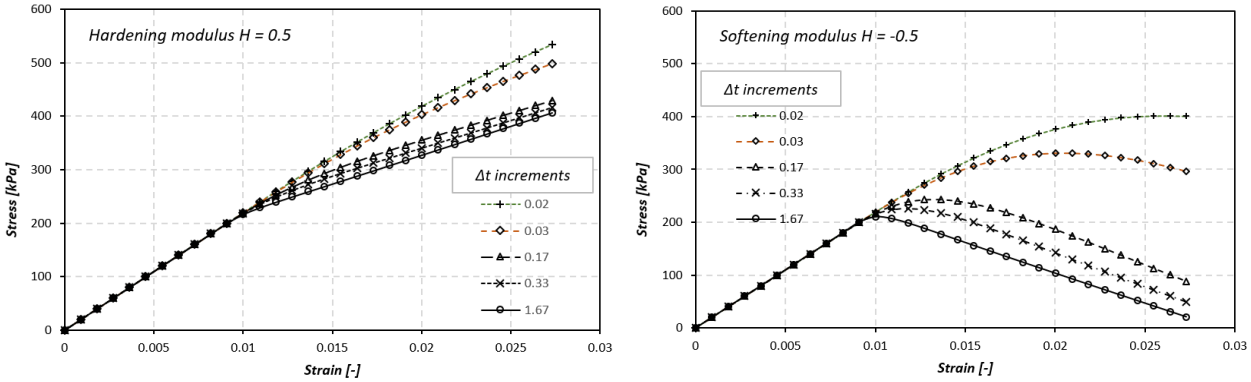


(a) Hardening (b) Softening

Figure 15: Variation of stress-strain relationship with viscosity

From figure 15 it can be seen that low-value viscosity reduces hardening and softening behavior. Viscosity acts like a dashpot, which means that high viscosity increases the resistance of materials for both cases hardening and softening. At the same time, viscosity variation affect more the softening behavior of materials, few changes in viscosity have large changes in softer materials (increase resistance) than harder materials, just when high changes of viscosity happens, hardening behaviour increase considerably.

3.3 Influence of the strain rate $\dot{\epsilon}$

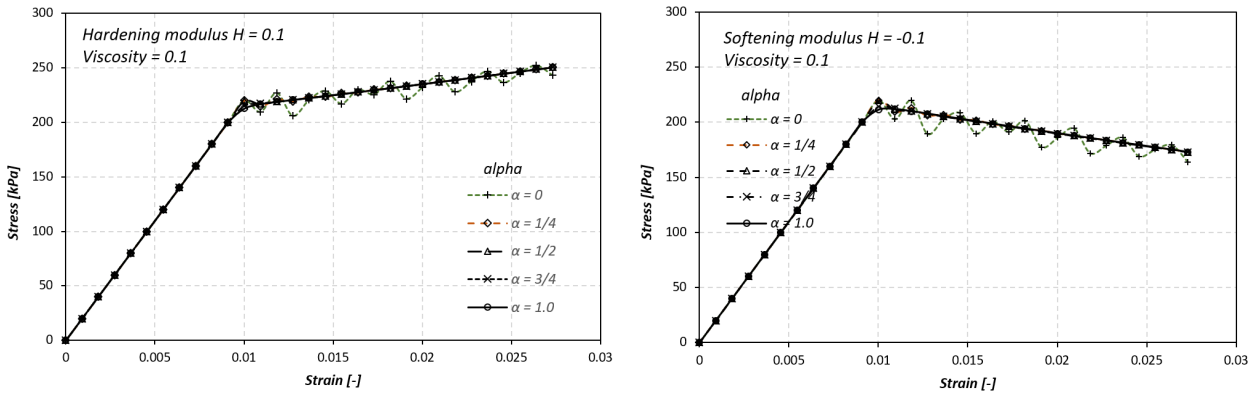


(a) Hardening (b) Softening

Figure 16: Variation of stress-strain relationship with viscosity

As it is expected, low strain rates increase the resistance of the materials. This high value of resistance is because the load is applied quickly and materials do not have time to redistribute the stresses.

3.4 Influence of α parameter

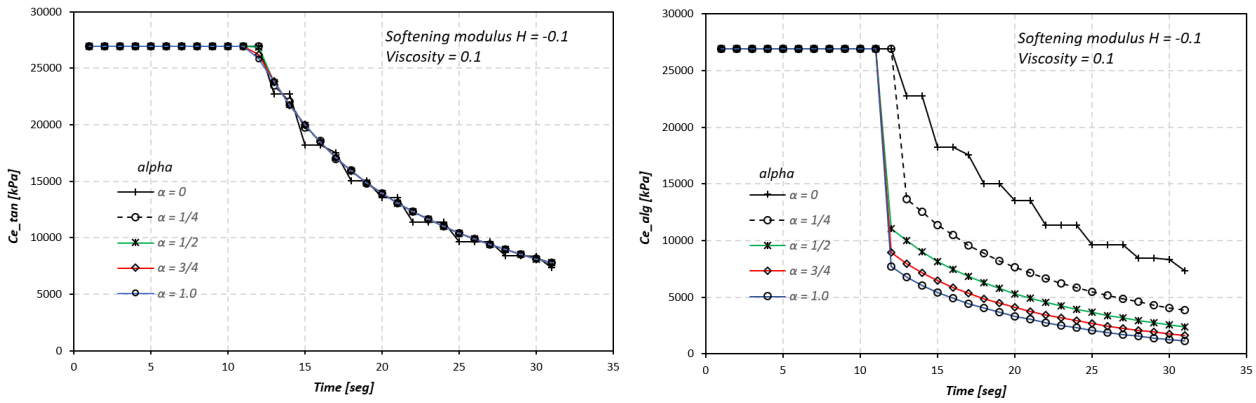


(a) Hardening (b) Softening

Figure 17: Variation of stress-strain relationship with viscosity

The alpha parameter seems to be relevant just for low values of hardening/softening modulus and low values of viscosity. Several tests were carried on with hardening/softening modulus of 0.5 and -0.5, but all the curves were the same. Reducing hardening modulus shows that low values of alpha affect the linear behavior creating oscillations like waves along the stress path. From the alpha value of 1/2, which means second order of accuracy, the stresses are stables.

3.5 Variation of $C_{tangent}$ and $C_{algorithm}$ with time



(a) $C_{tangent}$ (b) $C_{algorithm}$

Figure 18: Variation of stiffness with time

From figure 18 C_{tan} and C_{alg} are the same for alpha equal to zero and for elastic behavior of the material. However, for higher values of alpha C_{tan} seems be the same but C_{alg} reduce its value. Additional, there is a sharp change for C_{alg} when material start the plastic behavior and the starting point of this change decrease (C_{tan}) decrease as alpha increase.

3.6 Conclusions Part II

- Viscosity has a huge influence on material behavior, increasing the capacity to support more load. This influence is more clear in softening than in hardening.
- For lower values of strain rate of a material, higher is the load capacity.
- Low values of alpha parameters give unstable stress-strain behavior of materials. This is because for low values of alpha just first-order accuracy is obtained.
- Stiffness values C_{tan} and C_{alg} are the same for elastic behavior. For materials with damage, the stiffness is dependent of the alpha parameter selected for the integration.

4 Annex

4.1 Tension-Only implementation

The Code implemented is:

```
elseif MDtype==2  %2 means Tension-only

    tetha=[0:0.01:2*pi];
    %*********************************************************************
    %* RADIUS
    D=size(tetha);                          %*  Range
    m1=cos(tetha);                          %*
    m2=sin(tetha);                          %*
    Contador=D(1,2);                        %*
    m1New=m1;
    m2New=m2;
    m1New(m1New<0)=0;
    m2New(m2New<0)=0;

    %Initialization of variables
    radio = zeros(1,Contador) ;
    s1    = zeros(1,Contador) ;
    s2    = zeros(1,Contador) ;

    for i=1:Contador
            radio(i)= q/sqrt([m1New(i) m2New(i) 0 ...
            nu*(m1New(i)+m2New(i))]*ce_inv*[m1(i) m2(i) 0 nu*(m1(i)+m2(i))]');
            s1(i)=radio(i)*m1New(i);
            s2(i)=radio(i)*m2New(i);

        if (m1(i)<0)&&(m2(i)>0) %conditio for the second quadrant
            radioAux= q/sqrt([m1New(i) sin(pi/2) 0 ...
            nu*(m1New(i)+sin(pi/2))]*ce_inv*[m1(i) sin(pi/2) 0 ...
            nu*(m1(i)+sin(pi/2))]');
            s1(i)=radioAux*m1(i);
            s2(i)=radioAux*sin(pi/2);
        end
        if (m2(i)<0)&&(m1(i)>0) %Condition for the fourth quadrant
            radioAux= q/sqrt([cos(0) m2New(i) 0 ...
            nu*(cos(0)+m2New(i))]*ce_inv*[cos(0) m2(i) 0 ...
            nu*(cos(0)+m2(i))]');

           s1(i)=radioAux*cos(0);
           s2(i)=radioAux*m2(i);
        end
    end
    % plotting the damage surface in stress space
    hplot =plot(s1,s2,tipo_linea);
```

4.2 Non-symmetric tension-compression damage model implementation

```
elseif MDtype==3 %Non-symmetric model
    tetha=[0:0.01:2*pi];
    %*********************************************************************
    %* Using polar coordinates
    D=size(tetha);                          %*  Range
```

```
m1=cos(tetha);                          %*
m2=sin(tetha);                          %*
m3=nu*(cos(tetha)+sin(tetha));
Contador=D(1,2);                        %*
m1New=m1;
m2New=m2;

% Creating new vriable to delete negative values
m3New=nu*(m1+m2);
m1New(m1New<0)=0;
m2New(m2New<0)=0;
m3New(m2New<0)=0;

% Initializing vectors
radio = zeros(1,Contador) ;
radio2 = zeros(1,Contador) ;
s1    = zeros(1,Contador) ;
s2    = zeros(1,Contador) ;
s3    = zeros(1,Contador) ;
teta    = zeros(1,Contador) ;

for i=1:Contador
    normSigma(i)=(abs(m1(i))+abs(m2(i))+abs(m3(i)));
    teta(i)= (m1New(i)+m2New(i)+m3New(i))/normSigma(i);
    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))]');

    %Recovering stress in cartesian coordinates
    s1(i)=radio(i)*m1(i);
    s2(i)=radio(i)*m2(i);

    if (m1(i)<0)&&(m2(i)<0) %Condition for Third quadrant
        normSigma(i)=(abs(m1(i))+abs(m2(i))); %Is not a norm
        teta(i)= 0; %(m2New(i))/normSigma(i);
        radioAux= q/((teta(i)+(1-teta(i))/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)=radioAux*m1(i);
        s2(i)=radioAux*m2(i);
    end

    if (m1(i)<=0)&&(m2(i)>=0) %Condition for Second quadrant
        normSigma(i)=(abs(m1(i))+abs(m2(i))+abs(m3(i)));

        teta(i)= (m2New(i)+m3New(i))/normSigma(i);
        radioAux= q/((teta(i)+(1-teta(i))/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)=radioAux*m1(i);

    end

    if (m1(i)>=0)&&(m2(i)<=0) %Condition for Fourth quadrant

        normSigma(i)=(abs(m1(i))+abs(m2(i))+abs(m3(i)));
```

```
            teta(i)= (m1New(i)+m3New(i))/normSigma(i);
            radioAux= q/((teta(i)+(1-teta(i))/n)*sqrt([m1(i) m2(i) 0...
            nu*(m1(i)+m2(i))]*ce_inv*[m1(i) m2(i) 0 ...
            nu*(m1(i)+m2(i))]'));

            s2(i)=radioAux*m2(i);
        end

    end
    % plotting the damage surface in stress space
    hplot =plot(s1,s2,tipo_linea);
```

4.3 Function rmap_dano1

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

%*********************************************************************
hvar_n1 = hvar_n;
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) ;

% Parameters for viscosity
viscpr = Eprop(6) ;
eta = Eprop(7) ;
ALPHA_COEFF = Eprop(8) ;
%delta_t = 0.333;
%*********************************************************************
%*       initializing                                                *
 r0 = sigma_u/sqrt(E);
 zero_q=1.d-6*r0;
 q_inf=r0+0.99*r0;
% if(r_n<=0.d0)
%     r_n=r0;
%     q_n=r0;
% end
%*********************************************************************
%*       Damage surface                                                                  %*
[rtrial] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
[tao_n]=Modelos_de_dano1 (MDtype,ce,eps_n,n); %rtrial;
[tao_n1]=Modelos_de_dano1 (MDtype,ce,eps_n1,n); %sqrt(eps_n1*ce*eps_n1');
tao_na=(1-ALPHA_COEFF)*tao_n+ALPHA_COEFF*tao_n1;

%*********************************************************************
%*   Ver el Estado de Carga                                                          %*
%*   --------->    fload=0 : elastic unload                                         %*
%*   --------->    fload=1 : damage (compute algorithmic constitutive tensor)       %*
fload=0;
if (viscpr==1) % Condition for Viscous case

    if (tao_na<=r_n)
```

```
        r_n1= r_n;
        %Ce_alg =
        %dano_n1   = 1.d0-(q_n1/r_n1);
    else %if tao_na>r_n
        fload=1;
        term1=(eta-delta_t(1)*(1-ALPHA_COEFF))/...
        (eta+ALPHA_COEFF*delta_t(1)); % delta_t=0.3333
        term2=delta_t(1)/(eta+ALPHA_COEFF*delta_t(1));
        r_n1= term1*r_n+term2*tao_na;
        %dano_n1   = 1.d0-(q_n1/r_n1);
    end

    delta_r=r_n1-r_n;
    if hard_type == 0
        %  Linear
            q_n1= q_n+ H*delta_r;
    else
        % Exponential
        A=0.5;
        if (H>0)
            H_n1=H*((q_inf-r0)/r0)*exp(H*(1-tao_na/r0));
        else
            H_n1 = H*((q_inf-r0)/r0)*1/(exp(H*(1-tao_na/r0)));
        end
            q_n1= q_n+ H_n1*delta_r;
            %q_n1=q_inf-(q_inf-r_n)*exp(A*(1-r_n1/r_n));
    end
    if(q_n1<zero_q)
            q_n1=zero_q;
    end
        %H=H_n1;
else % Condicion no viscosa
    if(rtrial > r_n)
    %*   Loading
    fload=1;
    delta_r=rtrial-r_n;

    r_n1= rtrial  ;

        if hard_type == 0
        %  Linear
            q_n1= q_n+ H*delta_r;
        else
        % Exponential
            A=0.5;
        if (H>0)
            H_n1=H*((q_inf-r0)/r0)*exp(H*(1-rtrial/r0));
        else
            H_n1 = H*((q_inf-r0)/r0)*(1/(exp(H*(1-rtrial/r0))));
        end
            q_n1= q_n+ H_n1*delta_r;
            %q_n1=q_inf-(q_inf-r_n)*exp(A*(1-r_n1/r_n));
        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
    %H=H_n1;
end
% Damage variable
% ---------------
dano_n1   = 1.d0-(q_n1/r_n1);
%  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 ;
%*************************************************************************
%* Auxiliar variables                                                    %*
aux_var(1) = fload;
aux_var(2) = q_n1/r_n1;
aux_var(3) = (q_n1-H*r_n1)/r_n1^3;
    if (tao_na<=r_n)
        Ce_tan=(1-dano_n1)*ce;
        Ce_alg=Ce_tan;
    else
        Ce_tan=(1-dano_n1)*ce;
        term_aux1 = (q_n1-H*r_n1)/r_n1^2;
        term_aux2 = ((ALPHA_COEFF*delta_t)/(eta+ALPHA_COEFF*delta_t));
        term_aux3 = ((ce*eps_n1')*(ce*eps_n1')');
        Ce_alg=Ce_tan-term_aux2*term_aux1*(1/tao_n1)*term_aux3;
    end
aux_var(4)=Ce_tan(1,1); %Ce_tan
aux_var(5)=Ce_alg(1,1); %Ce_alg
%*************************************************************************
```

4.4 Function modelos_de_dano1

```
    function [rtrial] = Modelos_de_dano1 (MDtype,ce,eps_n1,n)

if (MDtype==1)      %* Symmetric
rtrial= sqrt(eps_n1*ce*eps_n1');

elseif (MDtype==2)  %* Only tension
    test=eps_n1*ce; %sigma+
    test(test<0)=0;
    %if (test>0)
    rtrial= sqrt(test*eps_n1');
    %else
    %    rtrial= 0.0;
    %end

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

```
rtrial= sqrt(eps_n1*ce*eps_n1');

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

4.5 Function damage_main

From line 137 to 192

```
    i = 1 ;
r0 = sigma_u/sqrt(E);
hvar_n(5) = r0; % r_n
hvar_n(6) = r0; % q_n
eps_n1 = strain(i,:) ;
sigma_n1 =ce*eps_n1'; % Elastic
sigma_v{i} = [sigma_n1(1)  sigma_n1(3) 0;sigma_n1(3) sigma_n1(2) 0 ;...
0 0  sigma_n1(4)];

nplot = 3 ;
vartoplot = cell(1,totalstep+1) ;
vartoplot{i}(1) = hvar_n(6) ; % Hardening variable (q)
vartoplot{i}(2) = hvar_n(5) ; % Internal variable (r)
vartoplot{i}(3) = 1-hvar_n(6)/hvar_n(5)  ; %  Damage variable (d)
vartoplot{i}(4) = sqrt((sigma_n1(1))^2+ (sigma_n1(2))^2 +(sigma_n1(4))^2);
vartoplot{i}(5) = aux_var(4);
vartoplot{i}(6) = aux_var(5);
%aux_var(i,4) = 0;
%aux_var(i,5) = 0;
for  iload = 1:length(istep)
    % Load states
    for iloc = 1:istep(iload)
        i = i + 1 ;
        TIMEVECTOR(i) = TIMEVECTOR(i-1)+ delta_t(iload) ;
        % Total strain at step "i"
        % ------------------------
        eps_n = strain(i-1,:) ;
        eps_n1 = strain(i,:) ;
        %*********************************************************
        %*      DAMAGE MODEL
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        [sigma_n1,hvar_n,aux_var] =
        rmap_dano1(eps_n,eps_n1,hvar_n,Eprop,ce,MDtype,n,delta_t);
        % PLOTTING DAMAGE SURFACE
        if(aux_var(1)>0)
            hplotSURF(i) = dibujar_criterio_dano1(ce, nu, hvar_n(6),
            'r:',MDtype,n );
            set(hplotSURF(i),'Color',[0 0 1],'LineWidth',1)                          ;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %*****************************************************************
        % GLOBAL VARIABLES
        % ****************
        % Stress
        % ------
        m_sigma=[sigma_n1(1)  sigma_n1(3) 0;sigma_n1(3) sigma_n1(2) 0 ;
```

```
0 0  sigma_n1(4)];
sigma_v{i} =  m_sigma ;

% VARIABLES TO PLOT (set label on cell array LABELPLOT)
% ----------------
vartoplot{i}(1) = hvar_n(6) ; % Hardening variable (q)
vartoplot{i}(2) = hvar_n(5) ; % Internal variable (r)
vartoplot{i}(3) = 1-hvar_n(6)/hvar_n(5)  ; %  Damage variable (d)
vartoplot{i}(4) = sqrt((sigma_n1(1))^2+ (sigma_n1(2))^2
+(sigma_n1(4))^2);
vartoplot{i}(5) = aux_var(4);
vartoplot{i}(6) = aux_var(5);
%aux_var(i,4) = aux_var(4);
%aux_var(i,5) = aux_var(5);
```