2019-2020 Master Numerical Methods in Engineering
COMPUTATIONAL SOLID MECHANICS
ASSIGNMENT # 1

Zichen Ding

Introduction

This assignment consists of the implementation of inviscid and viscous damage models. Firstly, tension-only and non-symmetric tension-compression damage models with exponential hardening/softening law were applied. Both damage models were evaluated under inviscid conditions considering representative load paths. Secondly, the viscous damage model was implemented and evaluated in a symmetric elastic domain and linear hardening/softening law. The effects of viscosity parameters, strain rate and the coefficient alpha were studied for the viscous damage model. An evaluation of the evolution of component C_{11} of tangential and algorithmic constitutive tensors was also considered for different values of alpha.

PART I (rate independent models):

(a) Implementation of Rate Independent Case for

1) The continuum isotropic "non-symmetric tension-compression" damage model

2) The "tension-only" damage model

In function ***dibujar_criterio_dano1.m***, the part of code defining the damage surface has been modified to add both cases above. For tension-only damage model, the essential modification is to change the expression for the stress vector to the positive counterpart. The details can be found in Annex and a representative damage surface for the implemented model is shown below in Figure 1. It can be clearly observed that when the principal stresses are negative (compression), the elastic domain can't be reached up to infinity.

The non-symmetrical damage model is useful to simulate materials, such as concrete, rocks, whose tension domain compression. The damage surface for the implemented model considering the ratio of compression strength over tension strength as 3, has been shown below in the Figure 2 with principal stresses axes. It can be clearly observed that the elastic limits of the elastic domain are different for tension and compression.

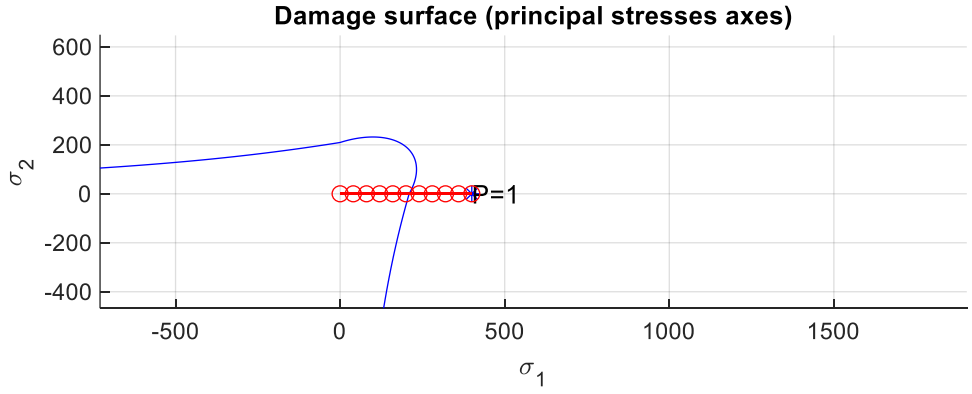


Figure 1 Damage surface for tensile-damage model

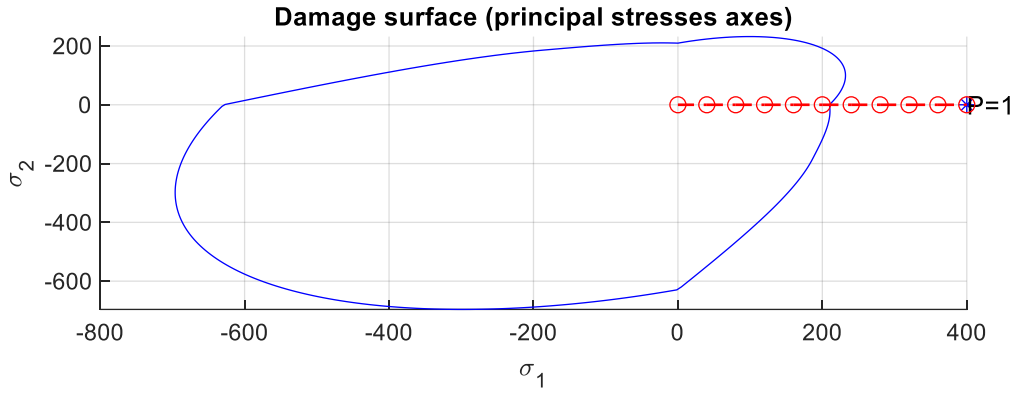


Figure 2 Damage surface for non-symmetric *tension-compression* damage model

(b) Implement the following cases for each of those models:
linear and exponential hardening/softening (H<0 and H>0)

The algorithm had implemented only the linear hardening/softening case. For that, the function ***rmap_dano1.m*** has been modified. The parameter $q\infty$ has been added to the ***Eprop*** vector variable to add this extra required parameter. The parameter *A* has been computed assuming that the input *H* used for the linear case now represents the value at $r = r0$. The implementation is immediate as shown:

```
if hard_type == 0
        %  Linear
        q_n1= q_n+ H*delta_r;
    else
        % Hardening/Softening exponential law implemented ;
        A = H * r0 / (q_inf - r0);
        q_n1= q_inf - (q_inf - r0) * exp(A*(1-r_n1/r0));
```

c) Accessing the correctness of the implementation of the models:

in all cases the following data has been used:
- Young's Modulus: $E = 2 \times 10^4 Pa$
- Hardening Modulus: H = 0.1
- Poisson coefficient: $\nu = 0.1$
- Yield stress: $\sigma Y = 200 Pa$

The loading paths start from origin of the stress space and are described by three-segment paths, with stress increments specified respectively for each case.

Case 1

$\left(\Delta\sigma_1^{(1)} = 400, \Delta\sigma_2^{(1)} = 0\right)$ uniaxial tensile loading,

$\left(\Delta\sigma_1^{(2)} = -1400, \Delta\sigma_2^{(2)} = 0\right)$ uniaxial tensile unloading/compressive loading,
$\left(\Delta\sigma_1^{(3)} = 1600, \Delta\sigma_2^{(3)} = 0\right)$ uniaxial compressive unloading/ tensile loading,

which correspond to stress states [400,0], [-1000,0] and [600,0] in the code.

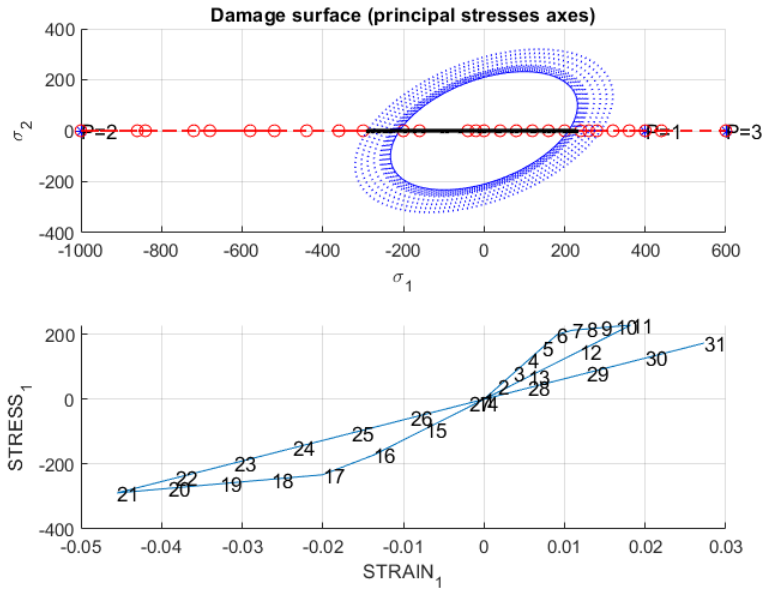


Figure 3 Case 1, Symmetric norm

In the stress-strain relationship curve, it is seen that, as expected, the points of inelastic loading 6-11 follow the path of a straight line and the points 17-21 in the anti-symmetric line. The shape of the stress admissible space is elliptic as expected. The line of the load state is along the horizontal line $\sigma_2 = 0$.

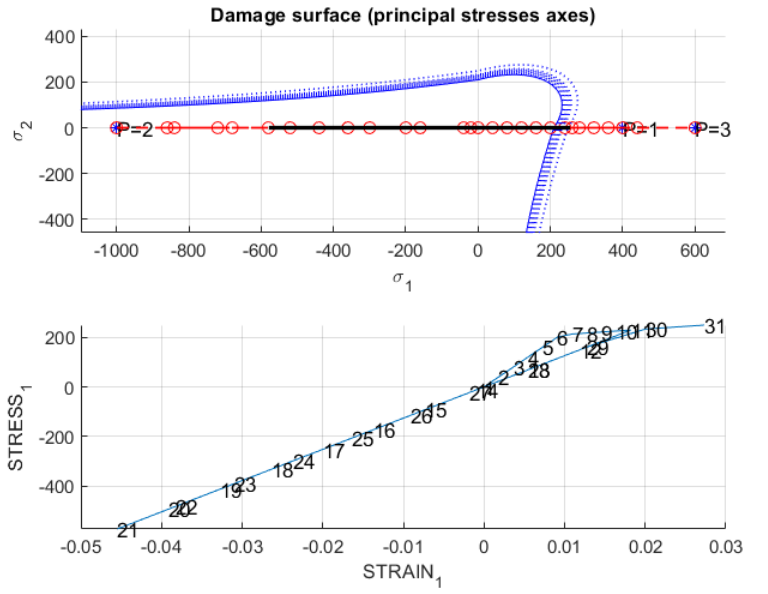


Figure 4 Case 1, Tension-only norm

In this simulation is seen that the behavior is similar to the linear of the symmetric case. However, here the compressive state reaches the state of around $-500Pa$ but it produces no damage as shown in the damage surface plot. This is consistent with the definition of tension-only damage.

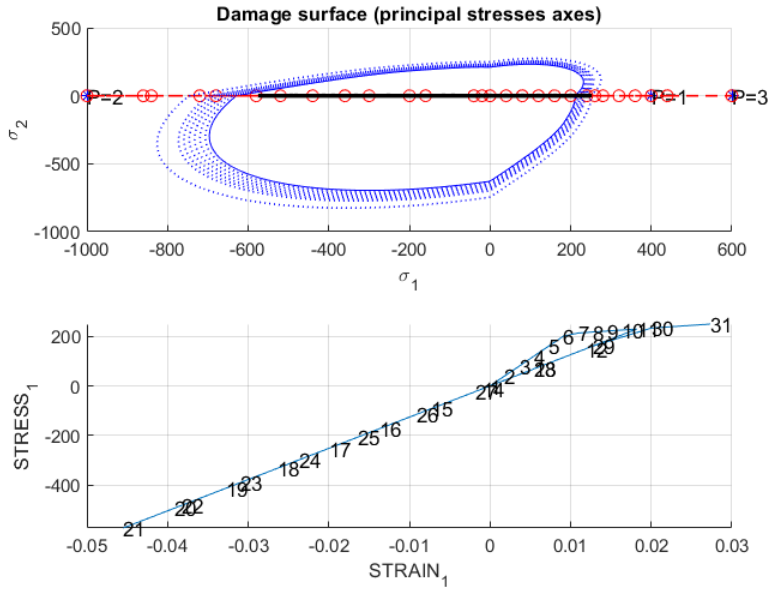


Figure 5 Case 1, Non-symmetric norm

In the scenario of non-symmetric norm, the damage surface has a larger radius in the compressive state. The evolution of damage surfaces and stress-strain curve are presented in Figure 5.

Case 2

$\left(\Delta\sigma_1^{(1)} = 400, \Delta\sigma_2^{(1)} = 0\right)$ uniaxial tensile loading,

$\left(\Delta\sigma_1^{(2)} = -1400, \Delta\sigma_2^{(2)} = -1000\right)$ biaxial tensile unloading/compressive loading,

$\left(\Delta\sigma_1^{(3)} = 1600, \Delta\sigma_2^{(3)} = 1600\right)$ biaxial compressive unloading/ tensile loading,

which correspond to stress states [400,0], [-1000, -1000] and [600,600] in the code.

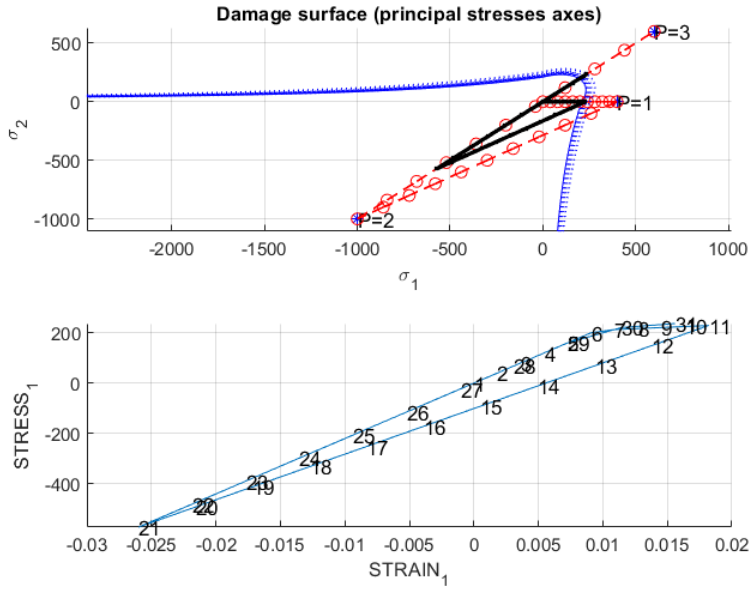


Figure 6 Case 2, tension-only norm

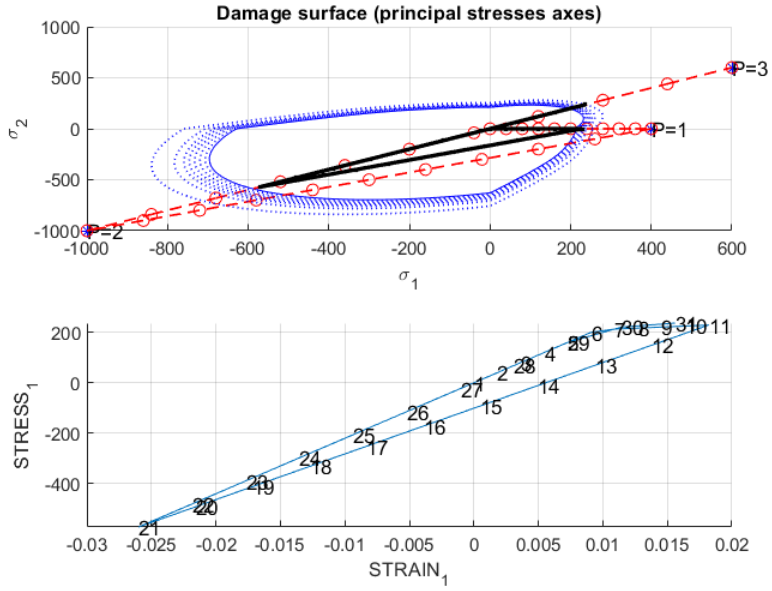


Figure 6 Case 2, Non-symmetric norm

Case 3

$\left(\Delta\sigma_1^{(1)} = 400, \Delta\sigma_2^{(1)} = 400\right)$ biaxial tensile loading,

$\left(\Delta\sigma_1^{(2)} = -1400, \Delta\sigma_2^{(2)} = -1400\right)$ biaxial tensile unloading/compressive loading,

$\left(\Delta\sigma_1^{(3)} = 1600, \Delta\sigma_2^{(3)} = 1600\right)$ biaxial compressive unloading/ tensile loading,

which correspond to stress states [400, 400], [-1000, -1000] and [600,600] in the code.

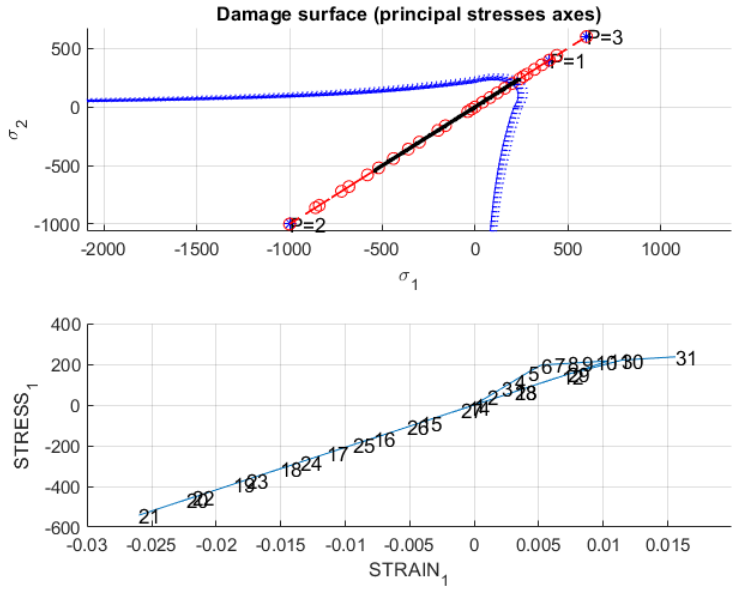


Figure 7 Case 3, tension-only norm

As all loading is biaxial, the stress state follows the line $\sigma_1 = \sigma_2$

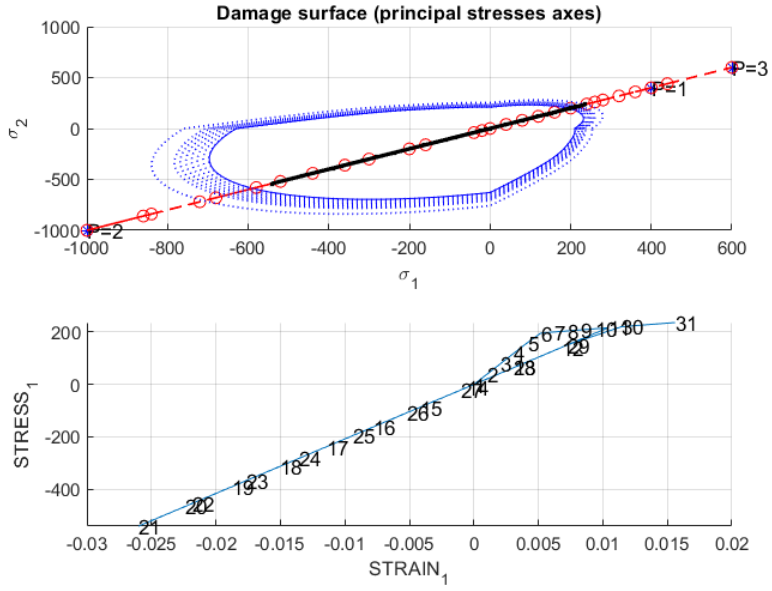


Figure 8 Case 3, Non-symmetric norm

PART II (rate dependent models):

d) Implementation of the integration algorithm (plane strain case) for the continuum isotropic visco-damage "symmetric tension compression" model

For viscous damage model, the stresses depend on both strains and strain rates. When the

strain rate is very low, it coincides with the inviscid case. In this type of model, the stress/strain state can lie outside the elastic domain.

Modification has been made to accommodate the viscous effect in ***rmap_dano1.m.*** More variables have been added to the array ***Eprop***, including ***viscous*** (logic value, assigned to 1 for viscous case), ***eta, alpha, q_inf***, ***delta_t***.

Then using the variable ***fload*** to distinguish elastic unloading (***fload***=0) from damage(***fload***=1)

Viscous and inviscid cases have different expressions of ***r_n1***, the internal variable at timestep n+1, as is shown in Annex. In viscous damage model, integration by the alpha method is used.

Effect of viscosity parameter η

In order to study the effect of viscosity on the material damage behavior, uniaxial tensile loading test has been performed.

using a symmetric norm and linear softening model with the following characteristics:

- **Alpha:** $\alpha = 0.5$
- **Total time:** $T = 10$
- **Load state:** $\sigma_1^{(1)} = 400, \sigma_2^{(1)} = 0$

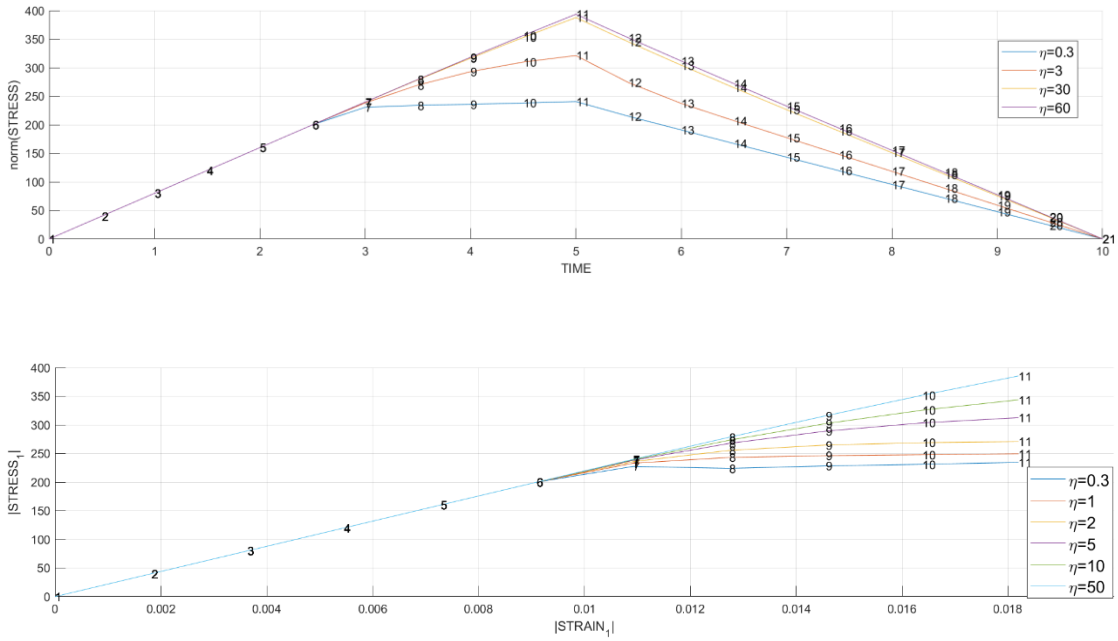


Figure 9 Viscous effect with varying viscosity constant values

As is shown in Figure 9, with higher viscosity values, the ultimate stress the material can resist is higher. When η is close to 0, the pattern of stress path is close to that of inviscid case; when η takes a large value, the slope of stress increase is close to that of elastic loading, with ultimate strength almost up to 400, as expected.

Effect of strain rate $\dot{\varepsilon}$

To compare the behavior of the material under different strain rates, the simulation has been performed with the following parameters:

- **Alpha:** $\alpha = 0.5$

- **Viscosity:** $\eta = 0.3$
- **Load state:** $\sigma_1^{(1)} = 400, \sigma_2^{(1)} = 0$

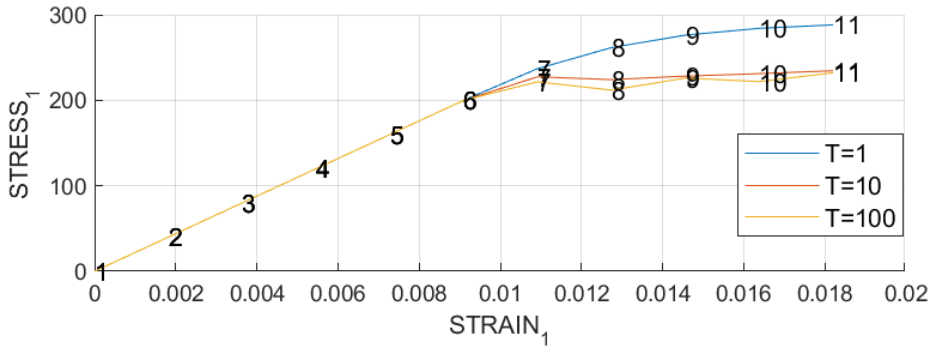


Figure 10 stress-strain curves with varying strain rates

The material behaves as expected: for high strain rates (T = 1), the material can resist stresses even higher than those of low strain rates(T=100). For extremely low strain rates (T = 1000), the rate-independent model is recovered.

Effect of α

The effect of different α values on the accuracy and the stability of the numerical integration have been studied. The simulation has been performed with the following data:

- **Total time:** $T = 10$
- **Viscosity:** $\eta = 0.3$
- **Load state:** $\sigma_1^{(1)} = 400, \sigma_2^{(1)} = 0$

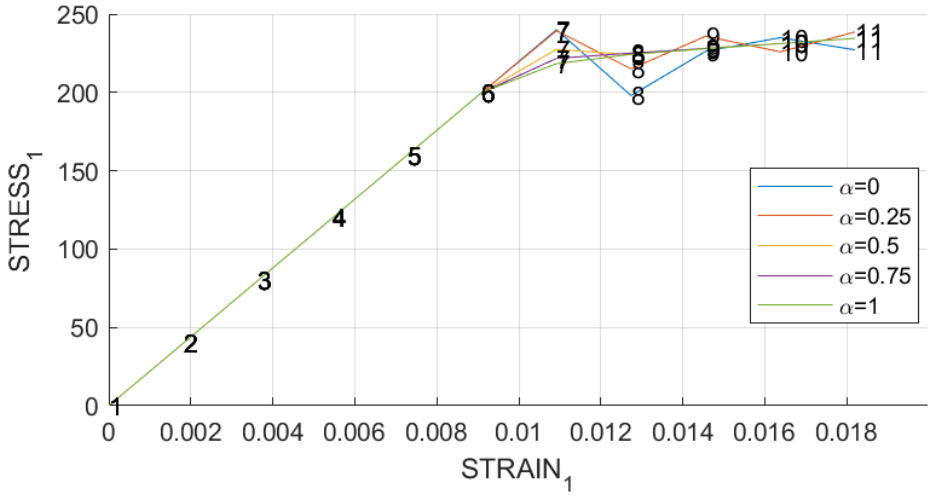


Figure 11 stress-strain curves with varying α values

From Figure 11, it can be observed that for $\alpha = 0$ and $\alpha = 0.25$, there are oscillations occurring in the process of loading. In fact, stability can only be achieved when α is in range $\left[\frac{1}{2}, 1\right]$, as is consistent with the results obtained. When $\alpha = \frac{1}{2}$, the mid-point rule is recovered while $\alpha = 1$ indicates Backward Euler.

Effect of α on the evolution of the C_{11} component of the tangent and algorithmic constitutive operators

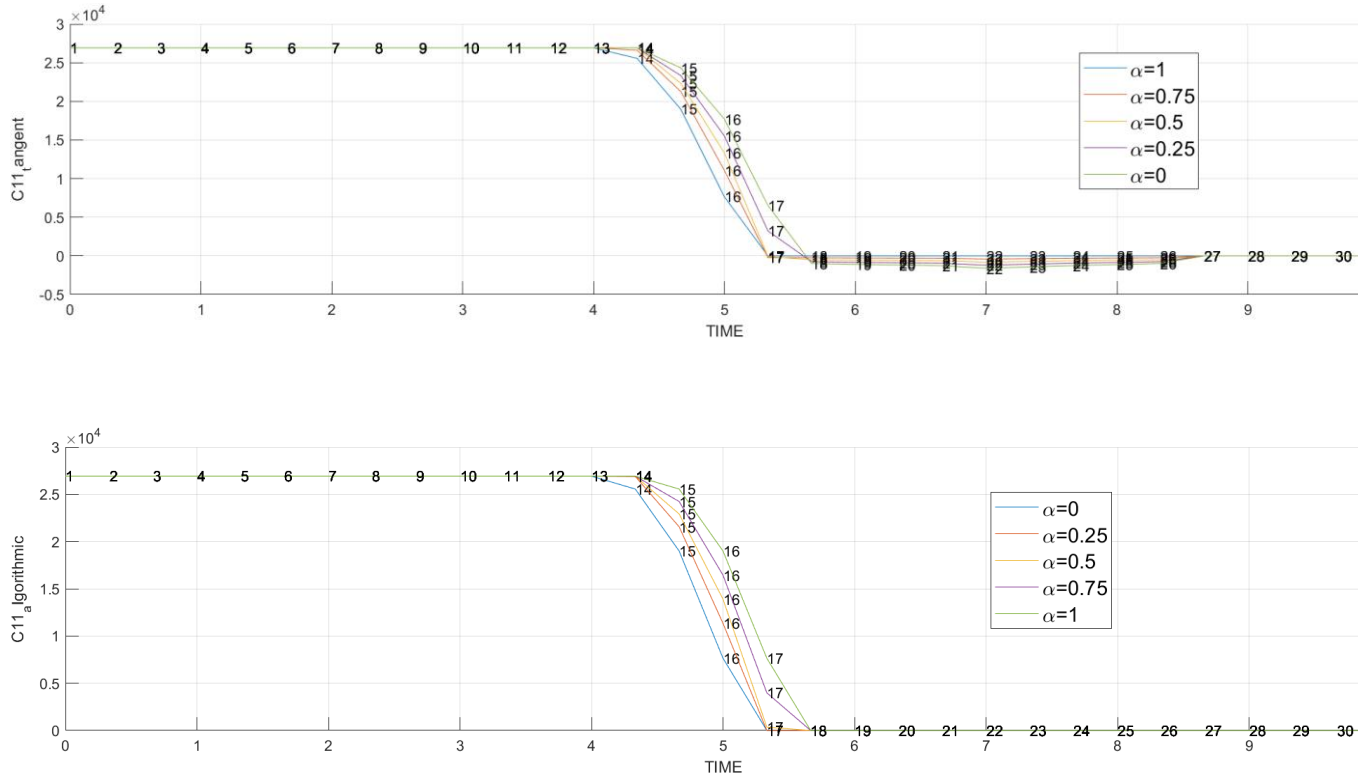
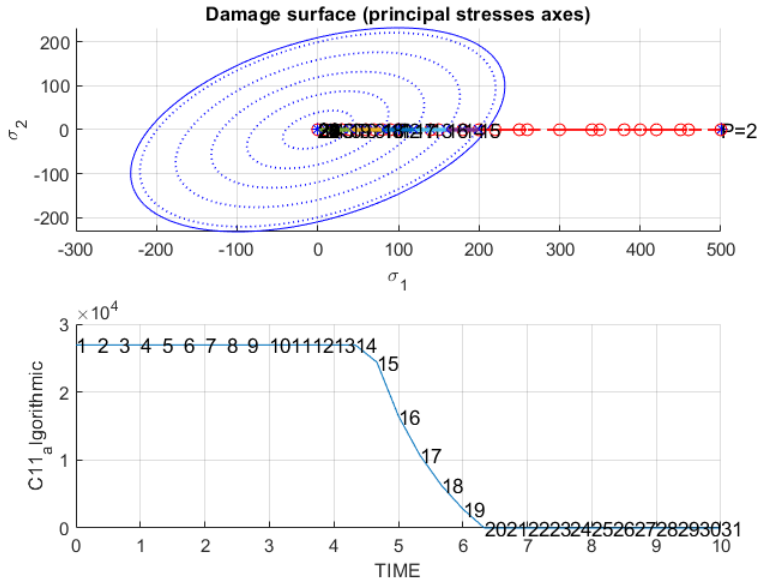


Figure 12 C_{11} component of the tangent and algorithmic constitutive operators



The stress path used for observation is (100,0), (300,0), (500,0). H=-1, $\eta = 10$. In the elastic domain, there is no change of C11 component taking place since no evolution of the internal

variable. Outside the elastic domain, for higher the α values, the values of C11 component are shown to be lower. Specially, when using $\alpha = 0$, both have the same value and show the same trend of evolution. When $\eta = 0$, the plot obtained is identical to that of the inviscid case, which is consistent with theory.

A Code for inviscid model

1. Modelos_de_dano1.m:

```
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
stress = ce * eps_n1';
stress_plus = max(0, stress);
rtrial = sqrt(stress_plus'*inv(ce)*stress);

elseif (MDtype==3)  %Non-symmetric
stress = ce * eps_n1';
theta = sum(max(0, stress)) / sum(abs(stress));
rtrial = (theta + (1-
theta)/n)*sqrt(stress'*inv(ce)*stress);
end
return
```

2. rmap_dano1.m:

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

hvar_n1 = hvar_n;
tau_n     = hvar_n(4);
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) ;
viscous      = Eprop(6);
eta       = Eprop(7);
alpha     = Eprop(8);
q_inf     = Eprop(9);
delta_t   = Eprop(10);

initializing                                                    %
*
 r0 = sigma_u/sqrt(E);
 zero_q=1.d-6*r0;
%*       Damage surface
[rtrial] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
%*   Ver el Estado de
Carga
%*
%*   --------->    fload=0 : elastic
unload                                                  %*
%*   --------->    fload=1 : damage (compute algorithmic
constitutive tensor)          %*
fload=0;

%for viscous and inviscid cases,check if damage (fload value)
if viscous % Viscous case
    tau_n1 = rtrial;
    tau_alpha = (1-alpha)*tau_n + alpha*tau_n1;
    if(tau_alpha > r_n) % Inelastic state
        fload=1;
        r_n1 = ((eta - delta_t * (1-alpha))*r_n + delta_t *
tau_alpha) /...
            (eta + alpha * delta_t);
    else % Elastic state
        fload=0;
    end
else   % inviscid case
    if rtrial > r_n % Inelastic state
        fload = 1;
        r_n1 = rtrial;
    else % Elastic state
        fload = 0;
    end
end
%% if damage, then 2 hardening laws; else r q value remain
unchanged
if fload
    if hard_type == 0
        %  Linear
        q_n1= r0 + H * (r_n1 - r0);
    else
        %  Exponential
        A = H * r0 / (q_inf - r0);
        q_n1= q_inf - (q_inf - r0) * exp(A*(1-r_n1/r0));
```

```
    end

    if(q_n1<zero_q)
        q_n1=zero_q;     %prevent the q going below 0 in
softening case
    end

else

    %*     Elastic load/unload
    r_n1= r_n  ;
    q_n1= q_n  ;
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')

if viscous
    hvar_n1(4)= tau_n1;
end
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;
```

3. dibujar_criterio_dano1

```
function hplot =
dibujar_criterio_dano1(ce,nu,q,tipo_linea,MDtype,n)
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];
    %************************************************
**********************************
    %* 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(max(0,[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==3
    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
        stress = [m1(i) m2(i) 0 nu*(m1(i)+m2(i))];

        theta_s = sum(max(0, stress)) /
sum(abs(stress));
        radio(i)= q/((theta_s+(1-
theta_s)/n)*sqrt(stress*ce_inv*stress'));

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

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

end
return
```

4. damage_main.m

```
function
[sigma_v,vartoplot,LABELPLOT,TIMEVECTOR]=damage_main(Ep
rop,ntype,istep,strain,MDtype,n,TimeTotal)
global hplotSURF
LABELPLOT = {'hardening variable (q)','internal
variable','damage variable (d)','C11-Tangent','C11-
Algorithimic'};

E      = Eprop(1) ; nu = Eprop(2) ;
viscpr = Eprop(6) ;
sigma_u = Eprop(4);

if ntype == 1
    menu('PLANE STRESS has not been implemented
yet','STOP');
    error('OPTION NOT AVAILABLE')
elseif ntype == 3
    menu('3-DIMENSIONAL PROBLEM has not been
implemented yet','STOP');
    error('OPTION NOT AVAILABLE')
else
    mstrain = 4    ;
    mhist   = 6    ;
end
```

```
totalstep = sum(istep) ;


% INITIALIZING GLOBAL CELL ARRAYS
% -------------------------------
sigma_v = cell(totalstep+1,1) ;
TIMEVECTOR = zeros(totalstep+1,1) ;
delta_t = TimeTotal./istep/length(istep) ;


% Elastic constitutive tensor
% ---------------------------
[ce]    = tensor_elastico1 (Eprop, ntype);
% Initz.
% -----
% Strain vector
% -------------
eps_n1  = zeros(mstrain,1);
% Historic variables
% hvar_n(1:4) --> empty
% hvar_n(5) = q --> Hardening variable
% hvar_n(6) = r --> Internal variable
hvar_n  = zeros(mhist,1)  ;

% INITIALIZING  (i = 1) !!!!
% ***********i*
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) =(1-( 1-
hvar_n(6)/hvar_n(5)))*ce(1,1) ; %%C11 tangent
vartoplot{i}(5)=(1-( 1-hvar_n(6)/hvar_n(5)))*ce(1,1)+
(Eprop(8)*delta_t)/(Eprop(7)+Eprop(8)*delta_t)...
                 *(Eprop(3)*hvar_n(5)-
hvar_n(6))/( hvar_n(5) * hvar_n(5) )...
                 *sigma_v{i}(1,1)*sigma_v{i}(1,1)/(1-
( 1-hvar_n(6)/hvar_n(5)))^2; %C11 algorithm
% vartoplot{i}(5)=(1-( 1-hvar_n(6)/hvar_n(5)))*ce(1,1)+
(Eprop(8)*delta_t)/(Eprop(7)+Eprop(8)*delta_t)...
%                  *(Eprop(3)*hvar_n(5)-
hvar_n(6))/( hvar_n(5) * hvar_n(5) )...
```

```
%
*sigma_v{i}(1,1)*sigma_v{i}(1,1)*(1/tau_eps_1) %C11
algorithm
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_n1 = strain(i,:) ;
        %*       DAMAGE MODEL
        [sigma_n1,hvar_n,aux_var] =
rmap_dano1(eps_n1,hvar_n,Eprop,ce,MDtype,n);
        % 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) =(1-( 1-
hvar_n(6)/hvar_n(5)))*ce(1,1) ; %%C11 Tangent
        vartoplot{i}(5) =(1-( 1-
hvar_n(6)/hvar_n(5)))*ce(1,1)+
(Eprop(8)*delta_t)/(Eprop(7)+Eprop(8)*delta_t)...
                        *(Eprop(3)*hvar_n(5)-
hvar_n(6))/( hvar_n(5) *
hvar_n(5) )*sigma_v{i}(1,1)*sigma_v{i}(1,1)/((1-( 1-
hvar_n(6)/hvar_n(5))))^2;
                            % C11 Algorithmic
    end
end
```

B Code for viscous damage model

1. Modelos_de_dano1.m:

```
function [rtrial] = Modelos_de_dano1 (MDtype,ce,eps_n1,n)
if (MDtype==1)       %* Symmetric
rtrial= sqrt(eps_n1*ce*eps_n1');
end%*****************************************************
**************************
return
```

2. rmap_dano1.m:

```
function [sigma_n1,hvar_n1,aux_var,C11_alg,C11_ana] =
rmap_dano1 (eps_n,eps_n1,hvar_n,Eprop,ce,MDtype,n,dt)



hvar_n1   = hvar_n;
tau_n     = hvar_n(4);
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);
viscous      = Eprop(6);
eta       = Eprop(7);
alpha     = Eprop(8);
q_inf     = Eprop(9);
delta_t   = Eprop(10);


r0 = sigma_u/sqrt(E);
zero_q=1.d-6*r0;


 [rtrial_1] = Modelos_de_dano1 (MDtype,ce,eps_n,n);
[rtrial_n] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);

fload = 0;
rtrial = (1-alpha)*rtrial_1 + alpha*rtrial_n ;
if(rtrial > r_n)

    fload = 1;
    dr = rtrial-r_n;
    %r_n1= rtrial ;
    r_n1 = (eta-dt*(1-alpha))/(eta+alpha*dt)*r_n + ...
        dt/(eta+alpha*dt)*rtrial;
```

```
    if hard_type == 0
        % Linear
        q_n1 = q_n+ H*dr;
    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.d0-(q_n1/r_n1);
% Computing stress

sigma_n1 =(1.d0-dano_n1)*ce*eps_n1';

%Calculate the constitutive operator
if fload == 1
    C11_alg = (1.d0-dano_n1)*ce + alpha*dt/...
        (eta+alpha*dt)*(1/rtrial_1)*(H*r_n1-q_n1)...
        /(r_n1^2)*(ce*eps_n1')*(eps_n1*ce');
else
    C11_alg = (1.d0-dano_n1)*ce;
end
C11_ana = (1.d0-dano_n1)*ce;
%*********************************************************

%* Updating historic variables
hvar_n1(5)= r_n1 ;
hvar_n1(6)= q_n1 ;

%* Auxiliar variables
aux_var(1) = fload;

aux_var(2) = q_n1/r_n1;
```

3. dibujar_criterio_dano1

```
function hplot =
dibujar_criterio_dano1(ce,nu,q,tipo_linea,MDtype,n)
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];
    %*************************************************************
****************************
    %* 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(max(0,[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==3
    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
        stress = [m1(i) m2(i) 0 nu*(m1(i)+m2(i))];

        theta_s = sum(max(0, stress)) / sum(abs(stress));
        radio(i)= q/((theta_s+(1-
theta_s)/n)*sqrt(stress*ce_inv*stress'));

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

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

end
return
```

4. damage_main

```
function
[sigma_v,vartoplot,LABELPLOT,TIMEVECTOR]=damage_main(Eprop,nt
ype,istep,strain,MDtype,n,TimeTotal)
global hplotSURF

LABELPLOT = {'hardening variable (q)','internal
variable','damage variable (d)','C11-Tangent','C11-
Algorithimic'};

E      = Eprop(1) ; nu = Eprop(2) ;
viscpr = Eprop(6) ;
sigma_u = Eprop(4);



if ntype == 1
    menu('PLANE STRESS has not been implemented yet','STOP');
    error('OPTION NOT AVAILABLE')
elseif ntype == 3
```

```
    menu('3-DIMENSIONAL PROBLEM has not been implemented
yet','STOP');
    error('OPTION NOT AVAILABLE')
else
    mstrain = 4    ;
    mhist   = 6    ;
end


totalstep = sum(istep) ;


% INITIALIZING GLOBAL CELL ARRAYS
% -------------------------------
sigma_v = cell(totalstep+1,1) ;
TIMEVECTOR = zeros(totalstep+1,1) ;
dt = TimeTotal./istep/length(istep) ;


% Elastic constitutive tensor
% ---------------------------
[ce]    = tensor_elastico1 (Eprop, ntype);
% Initz.
% -----
% Strain vector
% -------------
eps_n1  = zeros(mstrain,1);
% Historic variables
% hvar_n(1:4) --> empty
% hvar_n(5) = q --> Hardening variable
% hvar_n(6) = r --> Internal variable
hvar_n  = zeros(mhist,1)  ;

% INITIALIZING  (i = 1) !!!!
% **********i*
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) =ce(1,1) ; %%C11 tangent
vartoplot{i}(5)=ce(1,1); %C11 algorithm

for iload= 1:length(istep)
    % Load states
```

```
    for iloc = 1:istep(iload)
        i = i + 1 ;
        TIMEVECTOR(i) = TIMEVECTOR(i-1)+ dt(iload) ;
        % Total strain at step "i"
        % ------------------------
        eps_n = strain(i,:) ;
        eps_n1 = strain(i-1,:) ;
        %*****************************************************
*********************************
        %*      DAMAGE MODEL
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
        [sigma_n1,hvar_n,aux_var,C11_alg,C11_ana] =...
            rmap_dano1(eps_n,eps_n1,hvar_n,Eprop,ce,...
            MDtype,n,dt(iload));
        C11_Algorithmic(i) = C11_alg(1,1);
        C11_Tangent(i) = C11_ana(1,1);

        % 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) =C11_alg(1,1) ; %%C11 Tangent
        vartoplot{i}(5) =C11_ana(1,1);
        % C11 Algorithmic
    end
end
```

END