UPC, CIMNE, ETSECCP

Computational Solid Mechanics - Assignment 1

Rafael Pacheco
77128580N

April 7, 2017

DAMAGE MODELS

Universitat Politècnica De Catalunya
BARCELONATECH
Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona
CIMNE
Centre Internacional de Mètodes Numèrics en Enginyeria

Contents

1 Introduction

The purpose of this assignment is to implement the rate independent and rate dependent damage models algorithm. A Matlab environment algorithm is provided for the student to append the so-called models. The student has to implement the inviscid and viscid models, together with the hardening law.

The models have 3 different constitutive equations:

1 Symmetric tension-compression model:

$$\tau_{\varepsilon} = \sqrt{\varepsilon : \mathbb{C} : \varepsilon} \tag{1.1}$$

2 Tension only model:

$$\tau_{\varepsilon} = \sqrt{\langle\varepsilon\rangle : \mathbb{C} : \varepsilon} \tag{1.2}$$

3 Non-symmetric tension-compression model:

$$\theta = \frac{\sum_{i=1}^{3} <\sigma_i>}{\sum_{i=1}^{3} |\sigma_i|} \quad , \quad \tau_{\varepsilon} = \left(\theta + \frac{1-\theta}{n}\right)\sqrt{\varepsilon : \mathbb{C} : \varepsilon} \tag{1.3}$$

2 Rate Independent damage models

The inviscid model has been already implemented for Symmetric tension-compression model (Eqn. 1.1). The student has to implement Eqn. (1.2 and 1.3).

2.1 Tension-only damage model.

The tension only damage model has yield only working in tensile, therefore the 3rd quadrant is undefined, and the limits with it tend to two asymptotes ($\sigma_1 = 0$ and $\sigma_2 = 0$). To implement this model, the DIBUJAR_CRITERIO_DANO1.M has to be modified:

$$\tau_{\sigma} = \sqrt{\langle\sigma\rangle : \mathbb{C}^{-1} : \sigma} \tag{2.1}$$

```
% Mcaulay brackets
m1_2(m1_2<0)=0;
m2_2(m2_2<0)=0;
m3_2(m3_2<0)=0;
```

```
% loop
radio(i)= q/sqrt([m1_2(i) m2_2(i) 0 m3_2(i)]*ce_inv*[m1(i) m2(i) 0 ...
m3(i)]');
```

The plot obtained is:

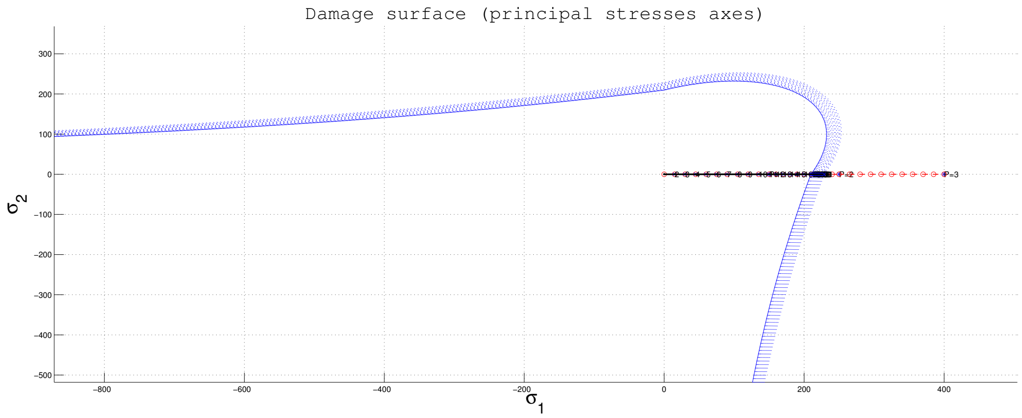

Figure 2.1: Tension only damage surface.

2.2 Non-symmetric tension-compression damage model.

Here the compression yielding stress is n times greater the tension yield strength. Using Eqn. 1.3 and implementing it inside DIBUJAR_CRITERIO_DANO1.M :

$$\tau_\sigma = \left(\theta + \frac{1-\theta}{n}\right)\sqrt{\sigma : \mathbb{C}^{-1} : \sigma} \tag{2.2}$$

```
TETHA = (m1_2(i) + m2_2(i) + m3_2(i))/(abs(m1(i))+abs(m2(i))+abs(m3(i)))
    ;
Const = TETHA + (1 - TETHA)/n;
radio(i)= q/(Const*sqrt([m1(i) m2(i) 0 nu*(m1(i)+m2(i))]*ce_inv*[m1(i)
    m2(i) 0 ...
nu*(m1(i)+m2(i))]'));
```

The plot obtained is:

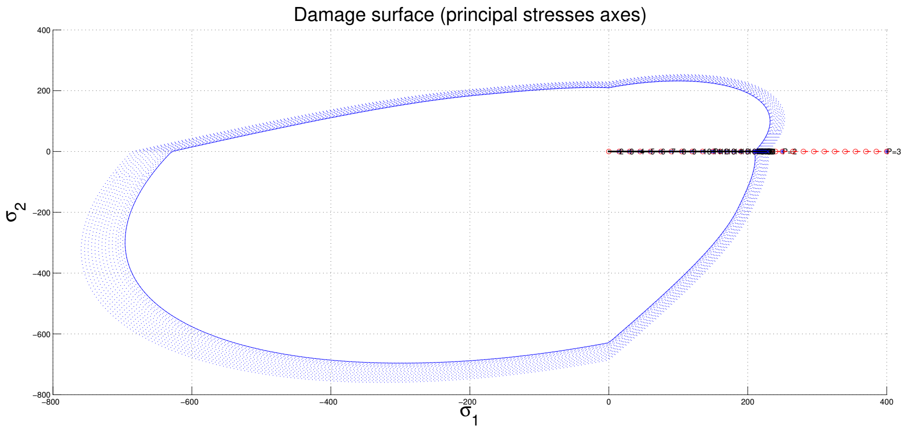

Figure 2.2: Non-symmetric tension-compression damage surface.

3 Hardening

3.1 Linear Hardening/Softening

The hardening is calculated by:

$$q(r) = \begin{cases} r_0 + H(r - r_0) & if \quad q < q_\infty \\ q_\infty & if \quad q \geq q_\infty \end{cases} \tag{3.1}$$

The evolution of the variable $q(r)$ is linear until it reaches the value q_∞. Once it reaches, the evolution is zero. Therefore the scheme would be:

$$q_{i+1} = \begin{cases} q_i H(r_{i+1} - r_i) & if \quad q_{i+1} < q_\infty \\ q_\infty & if \quad q_{i+1} \geq q_\infty \end{cases} \tag{3.2}$$

Eqn. 3.1 and Eqn. 3.2 are implemented into RMAP_DANO1.M :

```
if(rtrial > r_n)
%* Loading
fload=1;delta_r=rtrial-r_n;r_n1= rtrial ;
if hard_type == 0
% Linear
q_inf = q_fact * r0;
if q_fact>=1 % Hardening
if q_n>=q_inf
q_n1= q_inf;
else
```

```
q_n1= q_n+ H*delta_r;
if q_n1>=q_inf
q_n1 = q_inf;
end
end
else % Softening
if q_n<=q_inf
q_n1= q_inf;
else
q_n1= q_n+ H*delta_r;
if q_n1<=q_inf
q_n1 = q_inf;
end
end
end
hvar_n1(7) = H;%Exporting hardening variable
```

3.2 Exponential Hardening/Softening

In this case, the equation is:

$$q(r) = q_\infty - (q_\infty - r_0)\exp\Big(A(1-\frac{r}{r_0})\Big) \quad , \quad A \geq 0 \tag{3.3}$$

The hardening modulus is computed as:

$$H(r) = \frac{dq(r)}{dr} = A\Big(\frac{q_\infty - r_0}{r_0}\Big)\exp\Big(A(1-\frac{r}{r_0})\Big) \tag{3.4}$$

An important remark is that if q_∞ has a high value, negatives damages would be obtained and therefore healing. As first picked value in order to compute q_∞ would be considering the same value as for the constant hardening modulus as in the linear case, also imposing that r_{max} is proportional to r_0:

$$\begin{array}{lll} r_{max} = \beta r_0, & e.g. \beta = 2, & r_{max} = 2r_0 \\ \rightarrow & r_0 = q_0, & q_{fact} = \frac{q_\infty}{r_0} = \frac{H}{A}\exp(A) + 1 \end{array} \tag{3.5}$$

E.g. H=0.3 the values of $q\infty$ for $A \in [0.1, 1.5]$:

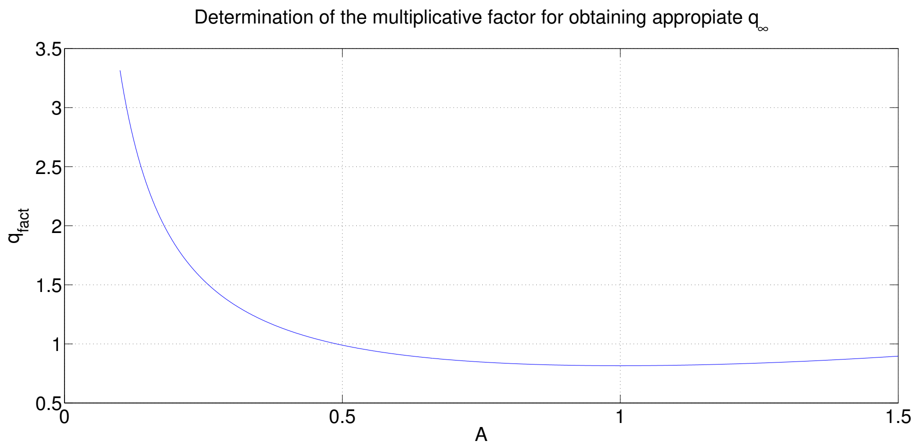

Figure 3.1: q_{fact} vs A.

Note that for values over 1 the material suffers Hardening and under 1 it suffers Softening. A Matlab file has been added, named Q_FACT_CALCULUS.M (Annex) , where it gives the relation between A and q_{fact}:

```
q_inf = q_fact * r0;
q_n1 = q_inf - ((q_inf - r0) * exp(A * (1 - (rtrial / r0))));
hvar_n1(7) = (A*(q_inf-r0)/r0) * exp(A * (1 - (rtrial / r0)));
```

And the generated output can be represented as:

4 Assessment of the inviscid model

To assess the code the parameters were defined as:

- $\alpha = 400$.
- $\beta = 150$.
- $\gamma = 250$.

First case is solved using symmetric tension-compression damage model for linear and exponential hardening schemes. The second case will be for tension only damage model and linear hardening scheme and the third for non-symmetric tension-compression damage model

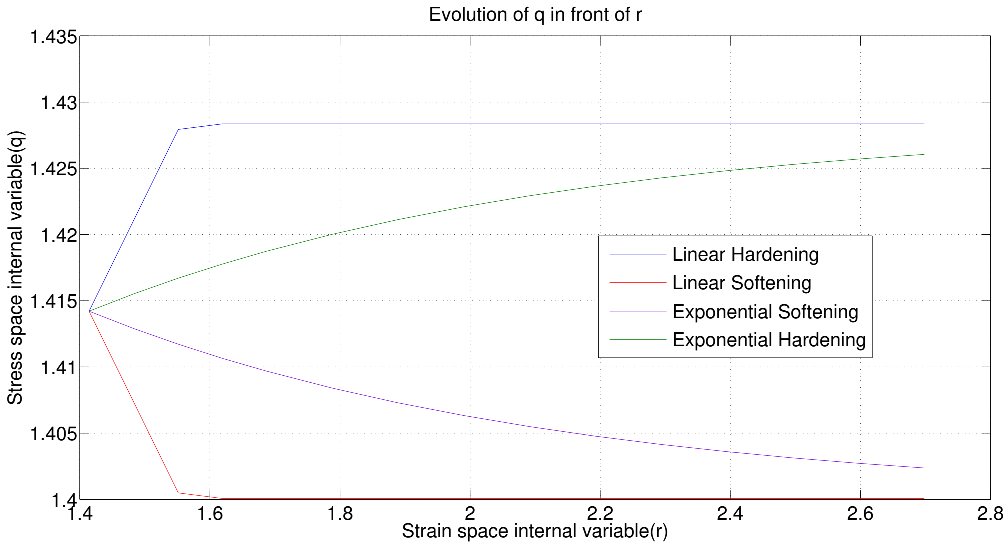

Figure 3.2: Hardening and Softening schemes.

for linear hardening scheme. Further comparison respect the models and hardening schemes will be explained in the following paragraphs.

4.1 Case 1

$$\begin{cases} \Delta\sigma_1 = \alpha \quad , \quad \Delta\sigma_2 = 0 \text{(uniaxial tensile loading)} \\ \Delta\sigma_1 = -\beta \quad , \quad \Delta\sigma_2 = 0 \text{(uniaxial tensile unloading/compressive loading)} \\ \Delta\sigma_1 = \gamma \quad , \quad \Delta\sigma_2 = 0 \text{(uniaxial compressive unloading/tensile loading)} \end{cases} \tag{4.1}$$

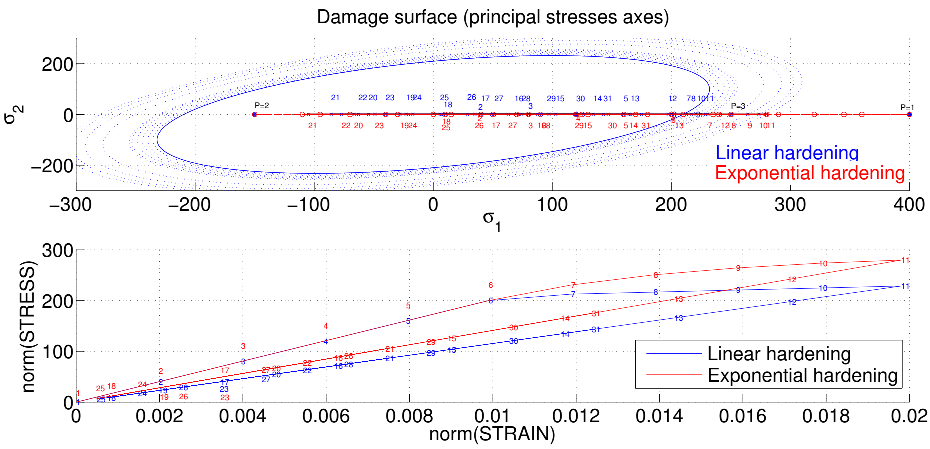

Figure 4.1: Damage surface and stress-strain plots for Case 1.

Note that the material is loaded for the first 11 steps, then is unloaded until 18^{th} step, it keeps compressing until 21^{st} step. Then compressive unloading until step 25 and tensile loading until step 31.

Observe that the results for exponential law grants bigger values of hardening modulus at the beginning and thus the stress for the exponential compared to the lineal is greater.

4.2 Case 2

$$\begin{cases} \Delta\sigma_1 = \alpha \quad , \quad \Delta\sigma_2 = 0 \text{(uniaxial tensile loading)} \\ \Delta\sigma_1 = -\beta \quad , \quad \Delta\sigma_2 = -\beta \text{(biaxial tensile unloading/compressive loading)} \\ \Delta\sigma_1 = \gamma \quad , \quad \Delta\sigma_2 = \gamma \text{(biaxial compressive unloading/tensile loading)} \end{cases} \tag{4.2}$$

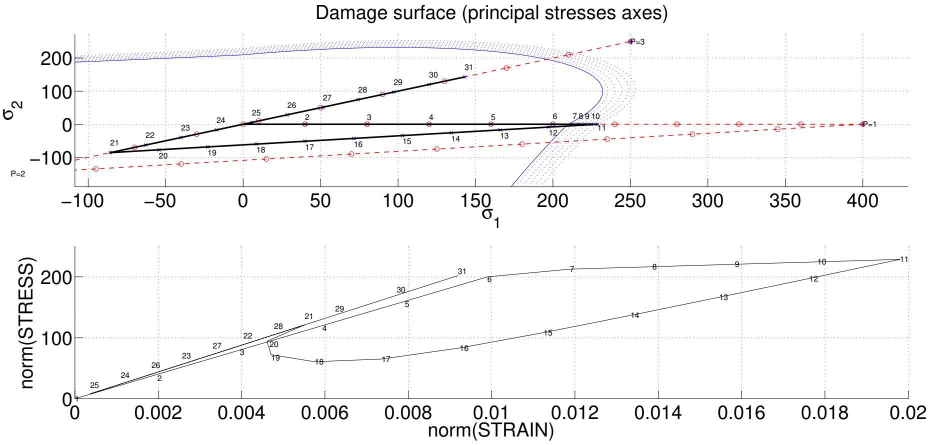

Figure 4.2: Damage surface and stress-strain plots for Case 2.

Note that the first eleven steeps are equal as for Case 1. Then the material unloads until step 17 where the biaxial compressive load begins until step 21. Again from step 21 to 25 the material unloads the compressive stress and until step 31 is under tensile loading.

4.3 Case 3

$$
\begin{cases} \Delta\sigma_1 = \alpha \quad , \quad \Delta\sigma_2 = \alpha (\text{biaxial tensile loading}) \\ \Delta\sigma_1 = -\beta \quad , \quad \Delta\sigma_2 = -\beta (\text{biaxial tensile unloading/compressive loading}) \\ \Delta\sigma_1 = \gamma \quad , \quad \Delta\sigma_2 = \gamma (\text{biaxial compressive unloading/tensile loading}) \end{cases} \tag{4.3}
$$

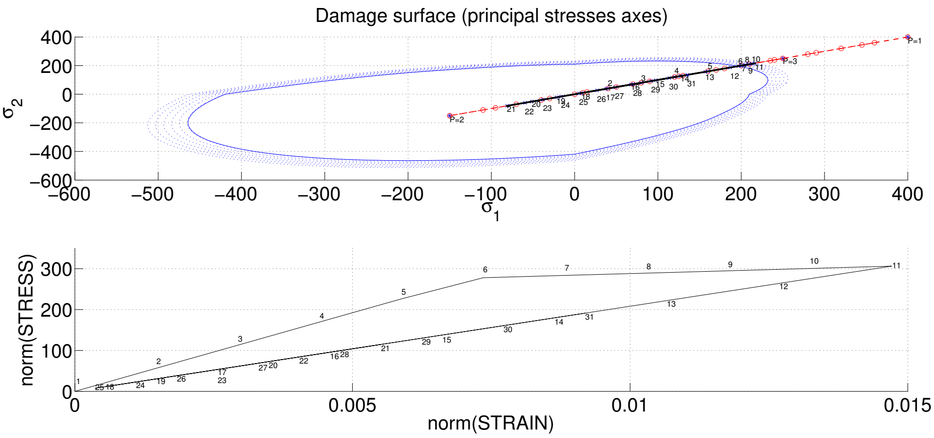

Figure 4.3: Damage surface and stress-strain plots for Case 3.

Note that in this case the result is identical to Case 1, with the particularity that is rotated with an angle of 45 degrees. This is because $\sigma_1 = \sigma_2$. Therefore the evolution of the norm of the stress respect to the strain is similar in shape as for Case 1 (linear case).

5 Rate dependent damage models

The implementation of the rate dependent damage model is undertaken in DAMAGE_MAIN.M and RMAP_DANOVISC.M.

5.1 α method

Here, the α method is introduced. The scheme to obtain $\tau_{\varepsilon_{n+\alpha}}$ is:

$$\tau_{\varepsilon_{n+\alpha}} = (1-\alpha)\tau_{\varepsilon_n} + \alpha\tau_{\varepsilon_{n+1}} \tag{5.1}$$

The scheme has been implemented by using the so-called function RMAP_DANOVISC.M, which is user implemented. The calculation of $\tau_{\varepsilon_{n+1}}$ is perform as:

```
[rtrial0] = Modelos_de_dano1 (MDtype,ce,eps_n0,n);
[rtrial1] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
rtrial= rtrial0*(1-alpha)+alpha*rtrial1;
```

Since this is a viscid model when loading, the equivalence $r_{n+1} = \tau_{\varepsilon_{n+\alpha}}$ does not hold any longer. The following scheme is used instead to calculated r_{n+1}:

$$r_{n+1} = \frac{\eta - \Delta t(1-\alpha)}{\eta + \alpha\Delta t} r_n + \frac{\Delta t}{\eta + \alpha\Delta t}\tau_{\varepsilon_{n+\alpha}} \tag{5.2}$$

Note that for $\eta = 0$ and $\alpha = 1$, the inviscid solution holds again. And this is implemented as:

```
r_n1= ((eta-delta_t*(1-alpha))/(eta+alpha*delta_t))*r_n+...
(delta_t/(eta+alpha*delta_t))*rtrial;
```

5.2 Implementation

In order to enable the viscous model, a if statement switches between one or the other. One way to implement both models (rate dependent and independent), is to add some specification as $\eta = 0$ and $\alpha = 1$ when inviscid model is selected and any other value the user wants for viscid model. To be simpler, the student has copied twice the same algorithm as the inviscid part for the viscid algorithm and added the function RMAP_DANOVISC.M. This would read:

```
if viscpr == 1
% Does computation for viscid model %
% ... Algorithm ... + call to rmap_danovisc.m
else
% Does computation for inviscid model %
% ... Algorithm ...
end
```

6 Assessment of the viscid model

The following parameters are defined for the viscous case:

- Symmetric damage model.
- $\nu = 0.3$.
- $H = 0.3$ and linear .
- $\sigma_1 = 400$.

Only constant uniaxial load step is introduced, since it is clearer to observe the effect of the viscosity, strain-rates and time-integration constant to the stresses.

6.1 Stress-Strain behaviour

6.1.1 Case 1

Stress-Strain curves against viscosities:

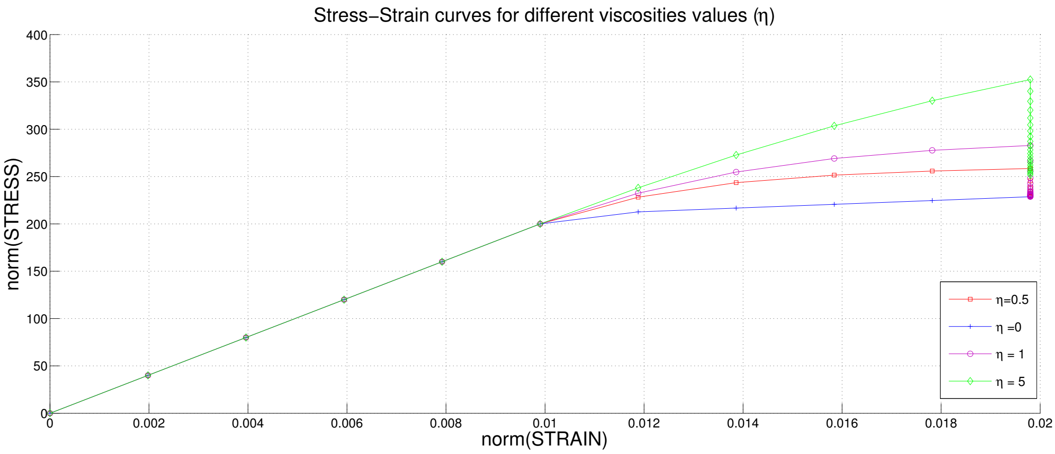

Figure 6.1: Stress-strain against viscosity.

As predicted by the formulation, if the viscosity is not zero, the stresses will increase. Notice that in the elastic region, viscosity is equal to 0.

6.1.2 Case 2

To assess the effect of the strain-rates, one has to to modify the total time of loading.

Stress-Strain curves against strain-rates:

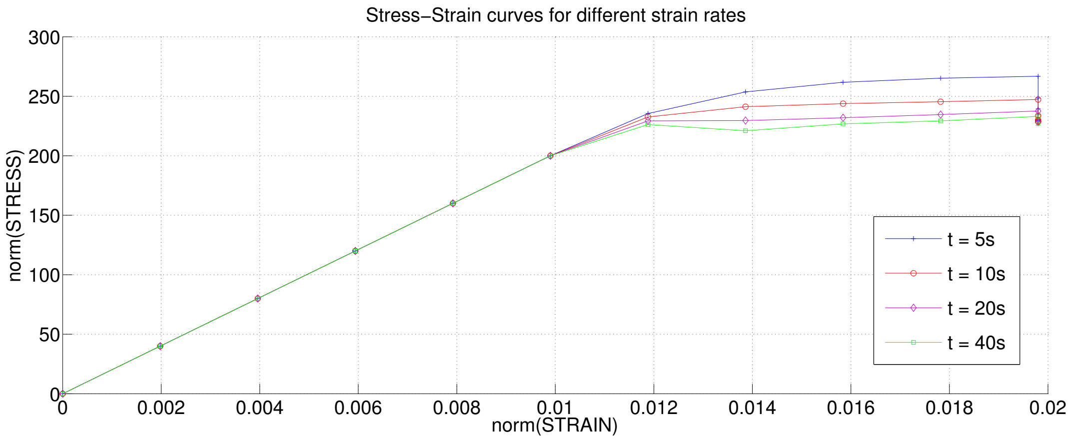

Figure 6.2: Stress-strain against time (proportionally inverse to stress-rate).

Obviously, the higher the strain-rate (inverse proportional to time), the higher the stresses.

6.1.3 Case 3

Stress-Stress curves against time-integration constants:

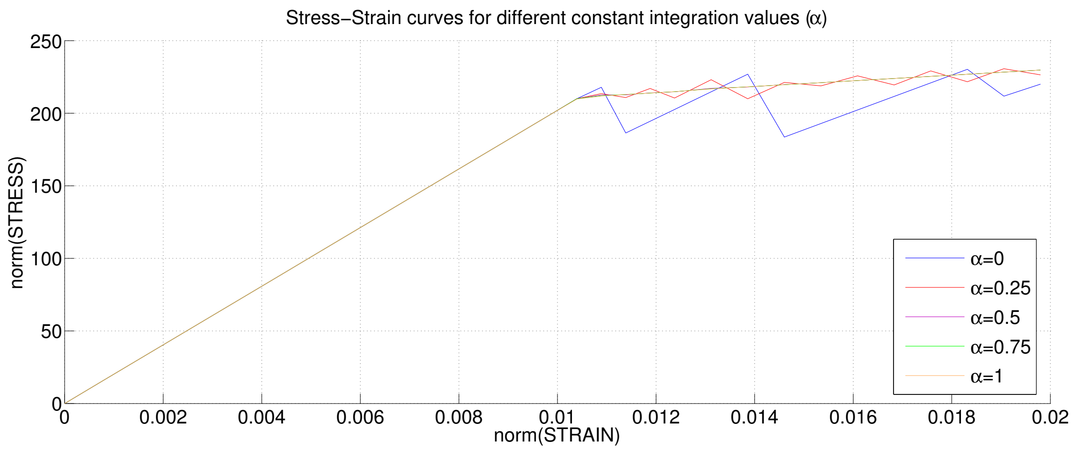

Figure 6.3: Stress-strain against α.

Note that:

$$\alpha = [0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1] \tag{6.1}$$

As the theory states, the results for $\alpha < 0.5$ are unstable. Also for $\alpha = 0.5$ the accuracy is second order, which is the so-called Crank-Nicholson method.

6.2 EFFECTS OF α AGAINST C_{11}

The algorithmic tangent constitutive operator (C_{11}) is computed within the DAMAGE_MAIN.M function.

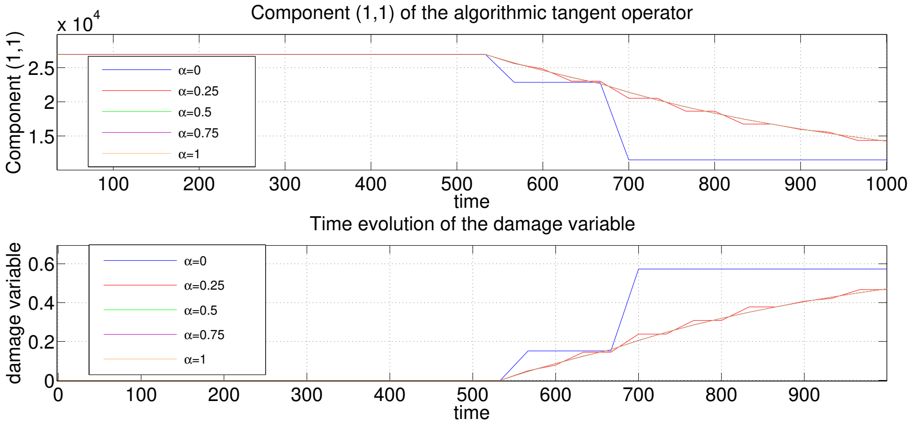

Figure 6.4: Stress-strain against α.

Note that for $\alpha < 0.5$ the solution oscillates and for $alpha \geq 0.5$ it stabilizes. Also observe that when the evolution of damage is 0, so the evolution of C_{11} is.

Finally, note that for the values of $\eta = 0$ and $\alpha = 1$ (inviscid case), the tangent constitutive operator is recovered again.

Appendices

APPENDIX 1: DIBUJAR_CRITERIO_DANO1.M

```
Annex 1: dibujar_criterio_dano1.m
function hplot = dibujar_criterio_dano1(ce,nu,q,tipo_linea,MDtype,n)
%* 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
% Comment/delete lines below once you have implemented this case
% *****************************************************************
tetha=[0:0.01:2*pi];
%*****************************************************************
%* RADIUS
% Slide 18 lecture 4
D=size(tetha);
```

```
m1 = cos(tetha);
m2 = sin(tetha);
m3 = nu * (m1 + m2);
Contador=D(1,2);
%Macaulin bracket
m1_2(m1_2<0)=0;
m2_2(m2_2<0)=0;
m3_2(m3_2<0)=0;
radio = zeros(1,Contador) ;
s1 = zeros(1,Contador) ;
s2 = zeros(1,Contador) ;
for i=1:Contador
radio(i)= q/sqrt([m1_2(i) m2_2(i) 0 m3_2(i)]*ce_inv* ...
[m1(i) m2(i) 0 m3(i)]');
s1(i)=radio(i)*m1(i);
s2(i)=radio(i)*m2(i);
end
hplot =plot(s1,s2,tipo_linea);
elseif MDtype==3
% Comment/delete lines below once you have implemented this case
% *******************************************************
% Slide 19 lecture 4
tetha=[0:0.01:2*pi];
D=size(tetha);
m1 = cos(tetha);
m2 = sin(tetha);
m3 = nu *(m1 + m2);
Contador=D(1,2);
% Macaulin bracket
m1_2(m1_2<0)=0;
m2_2(m2_2<0)=0;
m3_2(m3_2<0)=0;
radio = zeros(1,Contador) ;
s1 = zeros(1,Contador) ;
s2 = zeros(1,Contador) ;
for i=1:Contador
TETHA = (m1_2(i) + m2_2(i) + m3_2(i))/(abs(m1(i))+abs(m2(i))+ ...
abs(m3(i)));
Const = TETHA + (1 - TETHA)/n;
radio(i)= q/(Const*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
```

APPENDIX 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;
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) ;
q_fact = Eprop(9);
A = Eprop(10);
%***********************************************************************
%***********************************************************************
%* 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] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
hvar_n1(8) = rtrial;
%***********************************************************************
%***********************************************************************
%* Ver el Estado de Carga
%* ---------> fload=0 : elastic unload
%* ---------> fload=1 : damage (compute algorithmic
%constitutive tensor)
fload=0;
```

```
if(rtrial > r_n)
%* Loading
fload=1;
delta_r=rtrial-r_n;
r_n1= rtrial ;
if hard_type == 0
% Linear
q_inf = q_fact * r0;
if q_fact>=1 % Hardening
if q_n>=q_inf
q_n1= q_inf;
else
q_n1= q_n+ H*delta_r;
if q_n1>=q_inf
q_n1 = q_inf;
end
end
else % Softening
if q_n<=q_inf
q_n1= q_inf;
else
q_n1= q_n+ H*delta_r;
if q_n1<=q_inf
q_n1 = q_inf;
end
end
end
hvar_n1(7) = H;
else
% We calculate q_inf from a factor of the intial value of q,
%which is r0.
% Then q_n1 of the timestep n+1 is calculated as:
q_inf = q_fact * r0;
q_n1 = q_inf - ((q_inf - r0) * exp(A * (1 - (rtrial / r0))));
hvar_n1(7) = (A*(q_inf-r0)/r0) * exp(A * (1 - (rtrial / r0)));
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';
%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;
%*************************************************************
```

APPENDIX 3: Q_FACT_VALUE.M

```
clc;clear all;
% Young modulus
YOUNG = 20000;
% Yield strenght
sigma_u = 200;
%r0
r0=sigma_u;
%Select the maximum value of r
mult = 2;r_max = mult*r0;
%Select an average constant value for the hardening modulus. This
    constant
%value can be approximated by the value used for linear hardening.
H=0.3;
% Vector of different A (constant) values.
A = linspace(0.1,1.5,1000);
% Calculation of q_fact, which is the value that multiplied by r0
% give us q_inf: q_inf=q_fact*r0.
```

```
q_fact = (H ./ A) .* exp(-A * (1 - (r_max / r0)));
q_fact2 =(H ./ A) .* exp(A);
%plot
plot(A, q_fact)
grid on
title('Determination of the multiplicative factor for obtaining
    appropiate q_{\infty}')
xlabel('A')
ylabel('q_{fact}')
```

APPENDIX 4: RMAP_DANOVISC.M

```
function [sigma_n1,hvar_n1,aux_var] = rmap_danovisc (delta_t,eps_n0, ...
                                         eps_n1,hvar_n,Eprop,ce,MDtype,n)
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) ;
alpha=Eprop(8);
eta=Eprop(7);
q_fact = Eprop(9);
A = Eprop(10);
%*******************************************************************
%*******************************************************************
%*         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                                        %*
%Calculate rtrial (tau_eps) for the time step n and n+1. Then calculate
%rtrial which is (tau_eps_n+alpha) as rtrial0*(1-alpha)+alpha*rtrial1
[rtrial0] = Modelos_de_dano1 (MDtype,ce,eps_n0,n);
```

```
[rtrial1] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
rtrial= rtrial0*(1-alpha)+alpha*rtrial1;
hvar_n1(8)= rtrial; % Exporting rtrial_n
hvar_n1(9)= rtrial1; % Exporting rtrial_n+1
%Both needed to calculate Calg
%*******************************************************************
%*******************************************************************
%*   Ver el Estado de Carga
%*   --------->    fload=0 : elastic unload
%*   --------->    fload=1 : damage (compute algorithmic constitutive
% tensor)          %*
fload=0;
if(rtrial > r_n)
    %*   Loading
    fload=1;
    % As it is viscous, in the loading case we calculate rn+1 as:
    r_n1= ((eta-delta_t*(1-alpha))/(eta+alpha*delta_t))*r_n+ ...
                        (delta_t/(eta+alpha*delta_t))*rtrial;
    delta_r=r_n1-r_n;
    if hard_type == 0
        %  Linear
        q_inf = q_fact * r0;
      if q_fact>=1 % Hardening
        if q_n>=q_inf
            q_n1= q_inf;
        else
            q_n1= q_n+ H*delta_r;
            if q_n1>=q_inf
                q_n1 = q_inf;
            end
        end
      else % Softening
          if q_n<=q_inf
            q_n1= q_inf;
          else
            q_n1= q_n+ H*delta_r;
            if q_n1<=q_inf
                q_n1 = q_inf;
            end
        end
      end
        hvar_n1(7) = H;
    else
% We calculate q_inf from a factor of the intial value of q,
```

```
% which is r0.
% Then q_n1 of the timestep n+1 is calculated as:
            q_inf = q_fact * r0;
            q_n1 = q_inf - ((q_inf - r0) * exp(A * (1 - (r_n1 / r0))));
            hvar_n1(7) = (A*(q_inf-r0)/r0) * exp(A * (1 - (r_n1 / r0)));
      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';
%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;
%************************************************************
```

APPENDIX 5: DAMAGE_MAIN.M

```
function  [sigma_v,vartoplot,LABELPLOT,TIMEVECTOR,Calg11,Ctan11]=...
damage_main(Eprop,ntype,istep,strain,MDtype,n,TimeTotal)
```

```
global hplotSURF
% SET LABEL OF "vartoplot" variables (it may be defined also outside
% this function)
% ----------------------------------
 LABELPLOT = {'hardening variable (q)','internal variable'};
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   = 8    ;
end
if viscpr == 1
    % Here we solve for the viscous case, which is calling the function
    % rmap_danovisc
    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);
    Calg11=zeros(totalstep+1,2);
    Ctan11=zeros(totalstep+1,2);
    % Initz.
    % -----
    % Strain vector for each timestep
    % ----------------------------------
    eps_n0= zeros(mstrain,1);
    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)
    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_n0 = strain(i-1,:);
            eps_n1 = strain(i,:) ;
            %
                ***************************************************************
            %*        DAMAGE MODEL
            %
               %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            [sigma_n1,hvar_n,aux_var] = rmap_danovisc(delta_t,eps_n0,
                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)
    H_n1 = hvar_n(7);
    q_n1 = vartoplot{i}(1);
    r_n = vartoplot{i-1}(2);
    r_n1 = vartoplot{i}(2);
    d_n1 = vartoplot{i}(3);
    sigma_bar = [sigma_n1(1) sigma_n1(2) sigma_n1(4) 2*sigma_n1(3)];
    alpha = Eprop(8);
    eta = Eprop(7);
    rtrial = hvar_n(8);
    rtrial1 = hvar_n(9);
  % Calculation of the Algorithmic tangent operator and the tangent
  % operator.
    if rtrial>r_n1 %Loading case
        d_diff = (q_n1-(H_n1*r_n1))/((r_n1)^2);
        C1 = alpha*delta_t / (eta + (alpha*delta_t));
        C2 = 1/rtrial1;
        Calg=(1-d_n1)*ce-(C1*C2*d_diff*kron(sigma_bar',sigma_bar));
        %Ctan = (1-d_n1)*ce - (((q_n1 - H_n1*r_n1)/r_n1^3)* ...
        %(kron(sigma_bar',sigma_bar)));
        Calg11(i,1)=TIMEVECTOR(i);
        Calg11(i,2)=Calg(1,1);
        %Ctan11(i,1) = TIMEVECTOR(i);
        %Ctan11(i,2) = Ctan(1,1);
```

```
                else %elastic/unloading
                    Calg = (1-d_n1)*ce;
                    %Ctan=Calg;
                    Calg11(i,1)=TIMEVECTOR(i);
                    Calg11(i,2)=Calg(1,1);
                    %Ctan11(i,1) = TIMEVECTOR(i);
                    %Ctan11(i,2) = Ctan(1,1);
                end

            end
        end
else
% Here we solve for the inviscid case, which is calling the function
    % rmap_dano1
    totalstep = sum(istep) ;
    % INITIALIZING GLOBAL CELL ARRAYS
    % ----------------------------------
    sigma_v = cell(totalstep+1,1) ;
    TIMEVECTOR = zeros(totalstep+1,1) ;
    delta_t = TimeTotal./istep/length(istep) ;
    Calg11=zeros(totalstep+1,2);
    Ctan11=zeros(totalstep+1,2);
    % 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)
    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)
            H_n1 = hvar_n(7);
            q_n1 = vartoplot{i}(1);
            r_n = vartoplot{i-1}(2);
            r_n1 = vartoplot{i}(2);
            d_n1 = vartoplot{i}(3);
            sigma_bar = [sigma_n1(1) sigma_n1(2) sigma_n1(4) 2*sigma_n1
                (3)];
            alpha = Eprop(8);
            eta = Eprop(7);
            rtrial = hvar_n(8);
        % Calculation of the Algorithmic tangent operator.
            if rtrial>r_n1 %Loading case
                Ctan = (1-d_n1)*ce - (((q_n1 - H_n1*r_n1)/r_n1^3)* ...
                (kron(sigma_bar',sigma_bar)));
                Ctan11(i,1) = TIMEVECTOR(i);
                Ctan11(i,2) = Ctan(1,1);
            else %elastic/unloading
                Ctan = (1-d_n1)*ce;
                Ctan11(i,1) = TIMEVECTOR(i);
                Ctan11(i,2) = Ctan(1,1);
            end
        end
    end
end
```

APPENDIX 6: 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
sigma = eps_n1*ce;
macsigma=sigma;
macsigma(macsigma<0)=0;
 rtrial = sqrt(macsigma*eps_n1');
 elseif (MDtype==3) %*Non-symmetric
sigma = eps_n1*ce;
macsigma = sigma;
macsigma(macsigma<0)=0;
 theta = (macsigma(1) + macsigma(2) + macsigma(3)) / ...
 (abs(sigma(1)) + abs(sigma(2)) + abs(sigma(3)));
 rtrial =(theta + ((1-theta)/n)) * sqrt(eps_n1*ce*eps_n1');
end
%***************************************************************
return
```

APPENDIX 7: MAIN_NOINTERACTIVE.M

```
 all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program for modelling damage model
% (Elemental gauss point level)
% ------------------
% Developed by J.A. Hdez Ortega
% 20-May-2007, Universidad PolitÃlcnica de CataluÃ'sa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%profile on
% ---------------------------
% ***************
% INPUTS
% ***************
% YOUNG's MODULUS
% ----------------
YOUNG_M = 20000 ;
% Poisson's coefficient
```

```
% ——————————————————
POISSON = 0.3 ;
% Hardening/softening modulus
% —————————————————————
HARDSOFT_MOD = 0.1 ;%0.1
% Yield stress
% —————————
YIELD_STRESS = 200 ;
% Hardening/softening internal variable stress space factor (q_inf =
    q_fact * r0).
%
    ——————————————————————————————————————————————————————————————

q_fact = 1.01;%1.4
% Constant of the exponential law for hardering/softening. (A>0)
% ——————————————————————————————————————————
A = 2;
% Problem type TP = {'PLANE STRESS','PLANE STRAIN','3D'}
% —————————————————— = 1 =2 =3
% —————————
ntype= 2;
% Model PTC = {'SYMMETRIC','TENSION','NON-SYMMETRIC'} ;
% = 1 = 2 = 3
% ———————————————————————————————————————
MDtype =3;
% Ratio compression strength / tension strength
% ———————————————————————————————————
n = 3 ;%3
% SOFTENING/HARDENING TYPE
% ——————————————————
HARDTYPE = 'EXPONENTIAL' ; %{LINEAR,EXPONENTIAL}
% VISCOUS/INVISCID
% ——————————————————
VISCOUS = 'YES' ;
% Viscous coefficient ———
% ——————————————————
eta = 0;%0.3
% TimeTotal (initial = 0) ———
% ——————————————————
TimeTotal = 1000 ; %10
% Integration coefficient ALPHA
% ——————————————————
ALPHA_COEFF = 1 ;%0.5
% Points ————————————————————
```

```
% ---------------------------------------------------
nloadstates = 3 ;
SIGMAP = zeros(nloadstates,2) ;
SIGMAP(1,:) =[150 0];
SIGMAP(2,:) =[250 0];
SIGMAP(3,:) =[400 0];
% Number of time increments for each load state
% ---------------------------------------------------
istep = 10*ones(1,nloadstates) ;
% VARIABLES TO PLOT
vpx = 'norm(STRAIN)'; % AVAILABLE OPTIONS: 'STRAIN_1', 'STRAIN_2'
% '|STRAIN_1|', '|STRAIN_2|'
% 'norm(STRAIN)', 'TIME'
vpy = 'norm(STRESS)'; % AVAILABLE OPTIONS: 'STRESS_1', 'STRESS_2'
% '|STRESS_1|', '|STRESS_2|'
% 'norm(STRESS)', 'TIME', 'DAMAGE VAR.','hardening variable (q)','damage variable (d)'
% 'internal variable (r)'
% 3) LABELPLOT{ivar} --> Cell array with the label string for
% variables of "varplot"
%
LABELPLOT = {'hardening variable (q)','internal variable (r)','damage variable (d)'};
%%%%%%%%%%%%%%%%%%%%%%5 END INPUTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plot Initial Damage Surface and effective stress path
strain_history = PlotIniSurf(YOUNG_M,POISSON,YIELD_STRESS,...
SIGMAP,ntype,MDtype,n,istep);
E = YOUNG_M ;
nu = POISSON ;
sigma_u = YIELD_STRESS ;
switch HARDTYPE
case 'LINEAR'
hard_type = 0 ;
otherwise
hard_type = 1 ;
end
switch VISCOUS
case 'YES'
viscpr = 1 ;
otherwise
viscpr = 0 ;
end
Eprop = [E nu HARDSOFT_MOD sigma_u hard_type viscpr eta ...
ALPHA_COEFF q_fact A] ;
```

```
% DAMAGE MODEL
% ---------------
[sigma_v,vartoplot,LABELPLOT_out,TIMEVECTOR,Calg11,Ctan11]= ...
damage_main(Eprop,ntype,istep,strain_history,MDtype,n,TimeTotal);
try; LABELPLOT;catch;LABELPLOT = LABELPLOT_out ; end ;
% PLOTTING
% ---------
ncolores = 3 ;
colores = ColoresMatrix(ncolores);
markers = MarkerMatrix(ncolores) ;
hplotLLL = [] ;
for i = 2:length(sigma_v)
stress_eig = sigma_v{i} ; %eigs(sigma_v{i}) ;
tstress_eig = sigma_v{i-1}; %eigs(sigma_v{i-1}) ;
hplotLLL(end+1) = plot([tstress_eig(1,1) stress_eig(1,1) ],[tstress_eig(2,2) stress_'LineWidth',2,'color',colores(1,:),'Marker',markers{1},'MarkerSize',2);
plot(stress_eig(1,1),stress_eig(2,2),'bx')
text(stress_eig(1,1),stress_eig(2,2),num2str(i))
% SURFACES
% ------
end
% % SURFACES
% % ------
% 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
DATA.sigma_v = sigma_v ;
DATA.vartoplot = vartoplot ;
DATA.LABELPLOT = LABELPLOT ;
DATA.TIMEVECTOR = TIMEVECTOR ;
DATA.strain = strain_history ;
plotcurvesNEW(DATA,vpx,vpy,LABELPLOT,vartoplot) ;
d=zeros(length(TIMEVECTOR),1);
for i=1:length(TIMEVECTOR)
d(i) = vartoplot{i}(3);
end
figure(2)
subplot(2,1,1)
plot(Calg11(:,1),Calg11(:,2),'b-'),hold on
xlabel('time')
ylabel('Component (1,1)')
```

```
title('Component (1,1) of the algorithmic tangent operator')
grid on
subplot(2,1,2)
plot(TIMEVECTOR,d), hold on;
xlabel('time')
ylabel('damage variable')
title('Time evolution of the damage variable')
grid on
q = zeros(length(TIMEVECTOR),1);
r = zeros(length(TIMEVECTOR),1);
for i=1:length(TIMEVECTOR)
q(i) = vartoplot{i}(1);
r(i) = vartoplot{i}(2);
end
figure(3)
plot(r,q),hold on
xlabel('Strain space internal variable(r)')
ylabel('Stress space internal variable(q)')
title('Evolution of q in front of r')
grid on
```