UPC, CIMNE, ETSECCP

Computational Solid Mechanics
Assignment 1

Inocencio Castañar

April 7, 2017

CONTENTS

1 PART 1. RATE INDEPENDENT MODELS

The main purpose of this report is to show how the modified implementations in the supplied MATLAB code work, as well as to study their behaviour and validate their correctness. First of all, it is requested to implement in the supplied MATLAB code the integration algorithms for:

- The continuum isotropic damage "non-symmetric tension-compression damage" model.
- The "tension-only" damage model.

It has also been demanded to implement the case of exponential hardening/softening. All the implementations that have been added to the supplied code are shown in annex 1.

Let us assess the correctness of our implementations by showing some paths at the stress space with their corresponding strain-stress curves for each of the implemented models. Let us define appropriate loading paths starting at the point $\sigma_1^0 = 0$; $\sigma_2^0 = 0$ and described by three-segment paths in the strain space.

We assume a constant yield stress, $\boldsymbol{\sigma_u} = 200MPa$, constant Young Modulus, $\boldsymbol{E} = 20000MPa$ an a specific Poisson ratio, $\boldsymbol{\nu} = 0.3$ to see how the models behave depending only upon the properties of interest. Regarding the hardening/softening law the implemented exponential law is going to be used, defining the hardening/softening modulus H=1 as 1 when we want to reproduce hardening behaviour and -1 when we want to reproduce softening. Note that the second law of thermodynamics states that the value of $\boldsymbol{H}$ must be minor in absolute value than $q(r)/r$. This value is in the beginning 1 because $q_0 = r_0$ and for this reason it makes no physical sense for $\boldsymbol{H}$ to take values greater than 1.

Case 1.

$\Delta\bar{\sigma}_1^1 = 400MPa \quad ; \quad \Delta\bar{\sigma}_2^1 = 0$ (uniaxial tensile loading)
$\Delta\bar{\sigma}_1^2 = -1500MPa \quad ; \quad \Delta\bar{\sigma}_2^2 = 0$ (uniaxial tensile unloading/compressive loading)
$\Delta\bar{\sigma}_1^3 = 1100Mpa \quad ; \quad \Delta\bar{\sigma}_2^3 = 0$ (uniaxial compressive unloading/tensile loading)

Case 2.

$\Delta\bar{\sigma}_1^1 = 400Mpa \quad ; \quad \Delta\bar{\sigma}_2^1 = 0$ (uniaxial tensile loading)
$\Delta\bar{\sigma}_1^2 = -1500Mpa \quad ; \quad \Delta\bar{\sigma}_2^2 = -1500Mpa$ (biaxial tensile unloading/compressive loading)
$\Delta\bar{\sigma}_1^3 = 1100Mpa \quad ; \quad \Delta\bar{\sigma}_2^3 = 1100Mpa$ (biaxial compressive unloading/tensile loading)

Case 3.

$\Delta\bar{\sigma}_1^1 = \alpha \quad ; \quad \Delta\bar{\sigma}_2^1 = \alpha$ (biaxial tensile loading)
$\Delta\bar{\sigma}_1^2 = -\beta \quad ; \quad \Delta\bar{\sigma}_2^2 = -\beta$ (biaxial tensile unloading/compressive loading)
$\Delta\bar{\sigma}_1^3 = \gamma \quad ; \quad \Delta\bar{\sigma}_2^3 = \gamma$ (biaxial compressive unloading/tensile loading)

1.1 THE CONTINUUM ISOTROPIC DAMAGE "NON-SYMMETRIC TENSION-COMPRESSION DAMAGE" MODEL

As it is known, this model represents those materials which behave differently against compression or traction. Let us assume a ratio compression strength/traction strength , $\boldsymbol{n} = 2$.

- **Case 1**

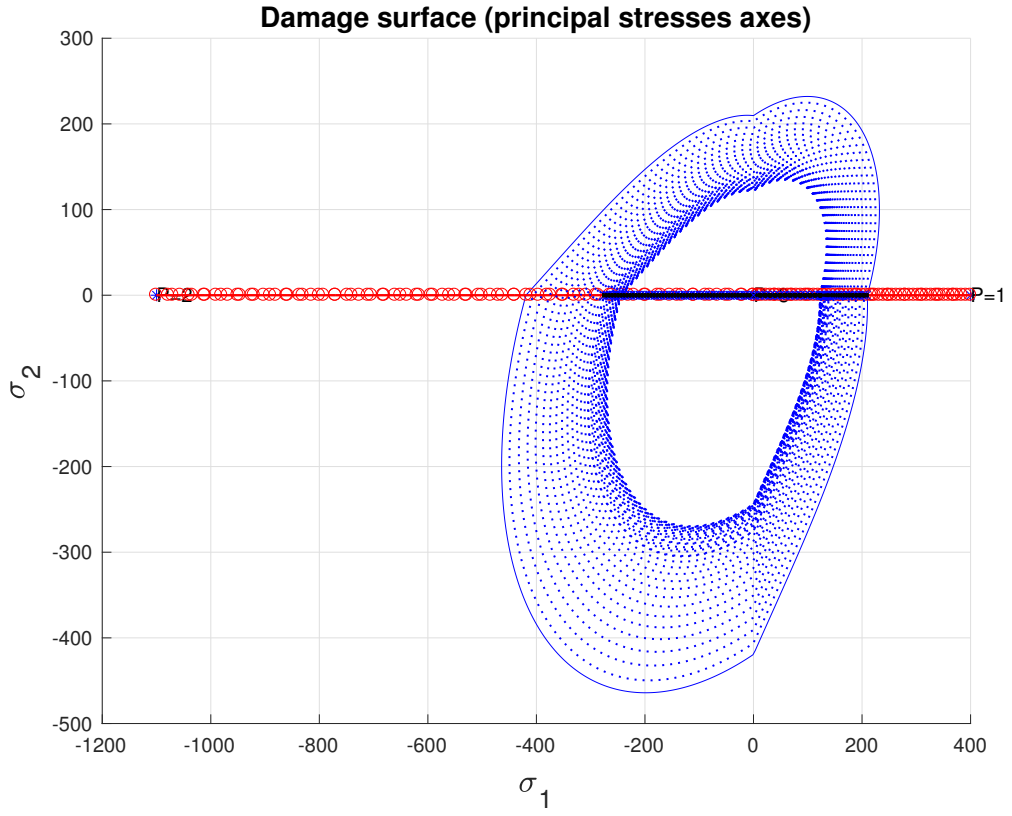

Figure 1.1: Path at the stress space for case 1

This figure shows the path followed in the stress space. It can be seen that as the elastic regime is surpassed twice, the effect of the softening parameter (-1) results in the

contraction of the elastic domain, which was expected.

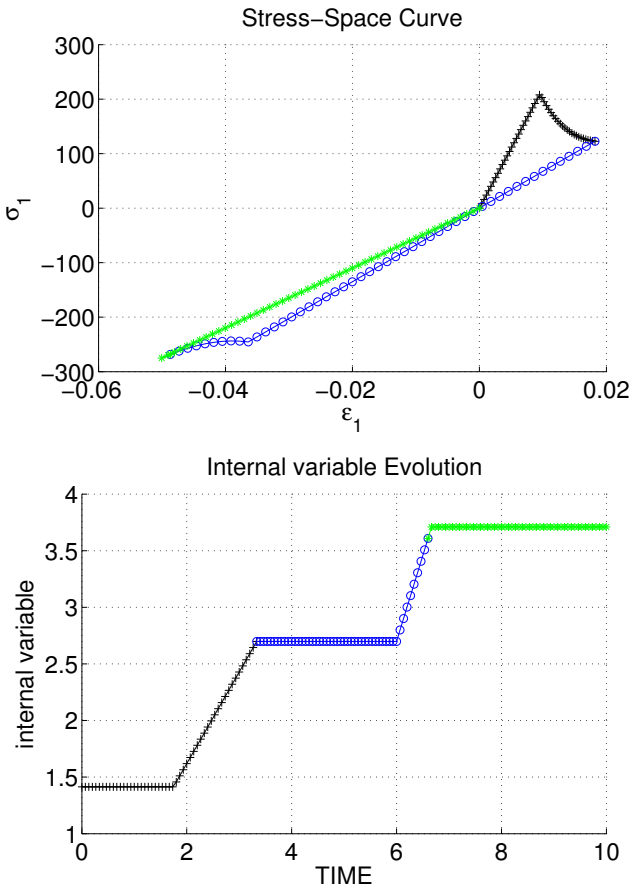

Figure 1.2: Stress-space curve (up) and internal variable evolution (down) for case 1

From the previous figure we can validate the correctness of our implementation. First of all the relation between stresses and strains is defined by the constitutive operator. Once the elastic regime has been surpassed, the damage variable increases and for this reason the slope of the unloading is less steep than the first one. Eventually as the inelastic regime has been achieved, our material damage increases again and the third loading case is incrementing with a minor operator.

This can also be seen in the below picture, where it can be observed that the internal variable r is not decreasing, and it increases at those steps where our effective stress has surpassed the elastic regime.

- **Case 3**

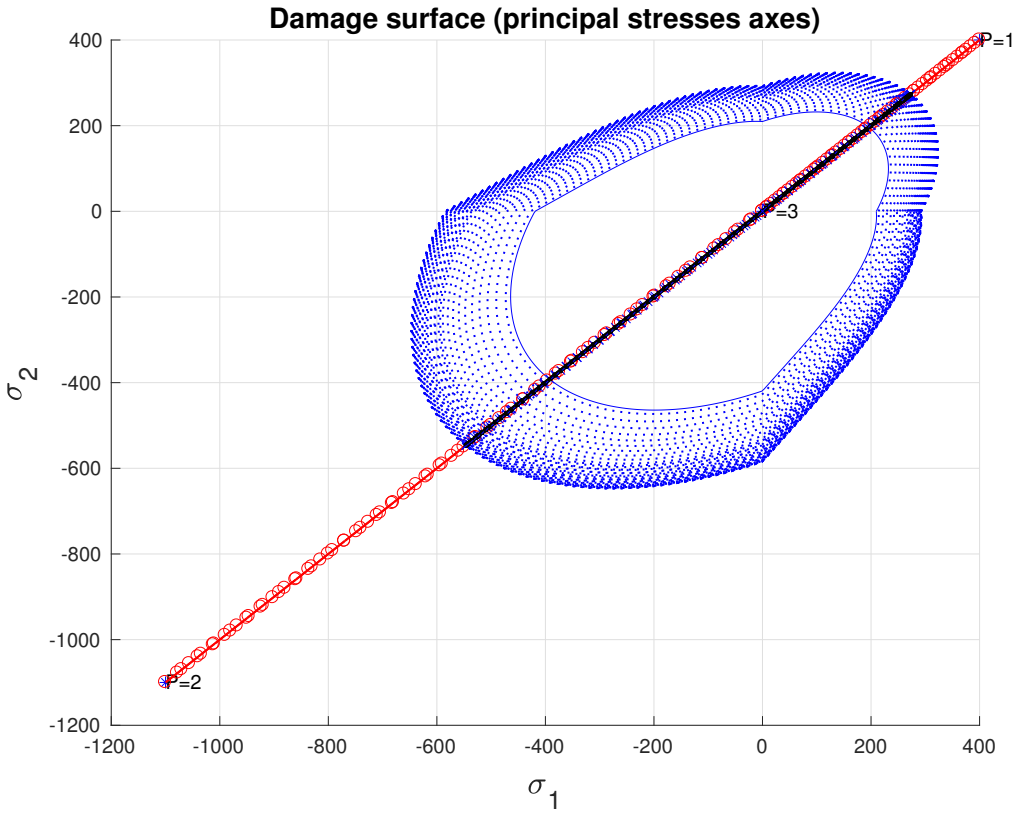

Figure 1.3: Path at the stress space for case 3

In this third case a hardening modulus has been imposed ($\boldsymbol{H} = 1$), and the elastic domain increases when the material surpasses the inelastic regime. Note that when the inelastic regime has been reached for the compressive loading, the elastic domain decreases. It is important to bear in mind that the elastic domain in the stress space can be reduced or increased in function of the hardening variable q.

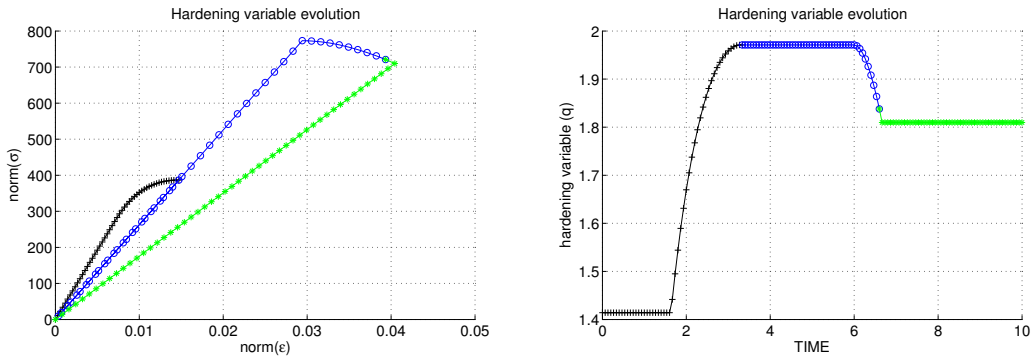

Figure 1.4: Stress norm-space norm curve (left) and hardening variable evolution (right)

In this case we can see that the norm of the stresses and the strains behave identically as the ones obtained for case 1. In addition, we can see that the hardening variable q increases while the material is in the inelastic regime and decreases when it surpasses it during compression.

1.2 THE "TENSION-ONLY" DAMAGE MODEL

- **Case 1**

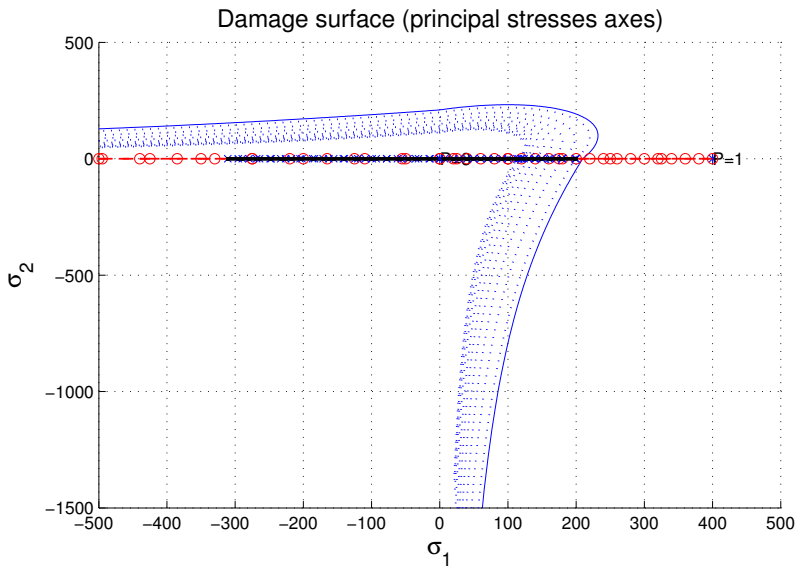

Figure 1.5: Path at the stress space for case 1

In this case, as a softening modulus ($\boldsymbol{H} = -1$) has been imposed, it can be seen that the elastic domain in the path at the stress space is decreases while the material surpasses the elastic regime.

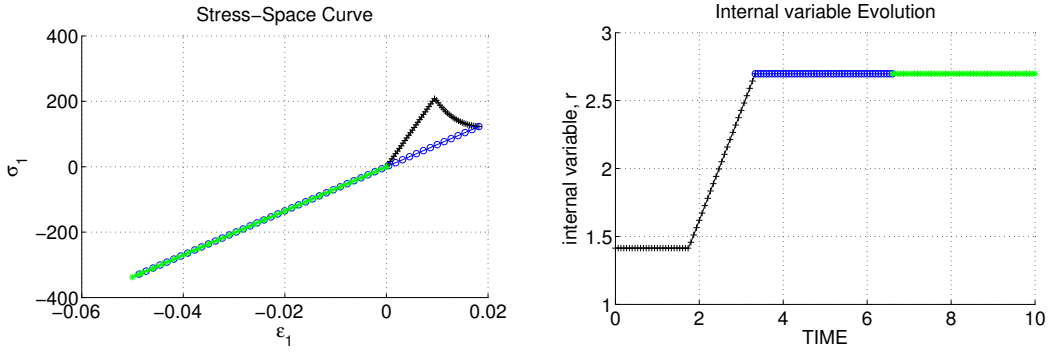

Figure 1.6: Stress-space curve (left) and internal variable evolution (right)

Unlike in case 1 for the non-symmetric damage model, in this case our material is allowed to be compressed up to infinite, without restrictions on its elastic domain for compression. Once the material has achieved the elastic domain in the first loading, as the internal variable increases, the constitutive operator decreases and the slope becomes less steep than before. In the following step the material can be compressed without surpassing the elastic domain, and for this same reason in the third step it evolves with the same slope than the second one.

In the picture on the right the evolution of the internal variable r can be observed, which increases when tensile loading surpasses the elastic regime, but as it has no restriction for compressive loading, it remains constant.

- **Case 2**

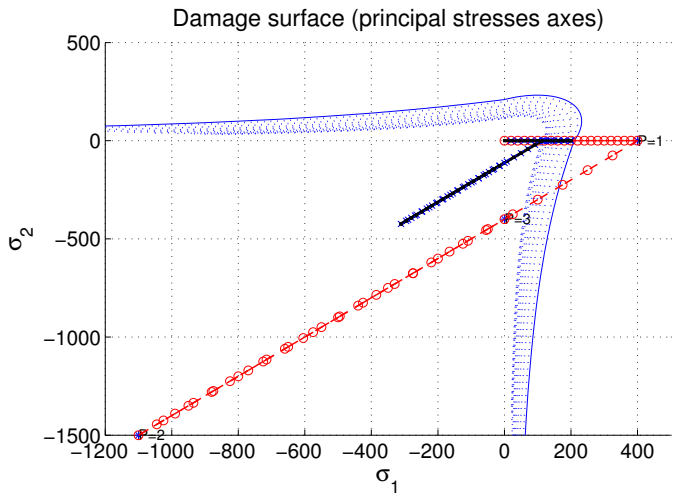

Figure 1.7: Path at the stress space for case 2

In this case, as a hardening modulus ($\boldsymbol{H} = 1$) has been imposed we can see that our elastic domain is increasing when the elastic regime is surpassed.

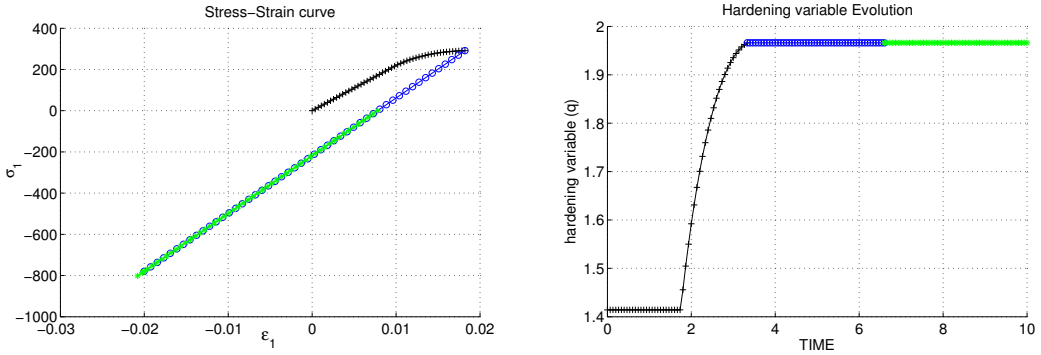

Figure 1.8: Stress-space curve (left) and hardening variable evolution (right)

Eventually we can see how the hardening variable q is increasing when the first tensile loading surpasses the elastic regime, as expected.

Observing and analysing the different figures and loading paths that have been obtained with our computations we have been able to confirm that our implementations are correct and that they behave as they theoretically should.

2 PART 2. RATE DEPENDENT MODELS

In the second part of this assignment we have implemented in the supplied MATLAB code the integration algorithm for the continuum isotropic visco-damage "symmetric tension-compression" model, and we have afterwards assessed its correctness by considering the effects of some parameters such as the viscosity parameter, the strain rate and the alpha value from the time integration scheme on the stress-strain curves. The main idea behind these analyses is to verify whether the implemented code behaves as it should theoretically.

Let us consider a specific Poisson ratio $\boldsymbol{\nu} = 0.3$ and linear hardening/softening, as well as the Young's Modulus and yield stress mentioned in the previous part of this assignment.

2.1 VISCOSITY PARAMETER INFLUENCE ON THE STRESS-STRAIN CURVE

As it is known the effect of the viscosity allows the material to achieve higher stresses once it is inside the inelastic regime. To show whether our implementation reproduces this behaviour or not, let us fix all the different parameters except for the viscosity one and then compute the stress-strain curve for the same path, a simple loading case in which the elastic domain is surpassed.

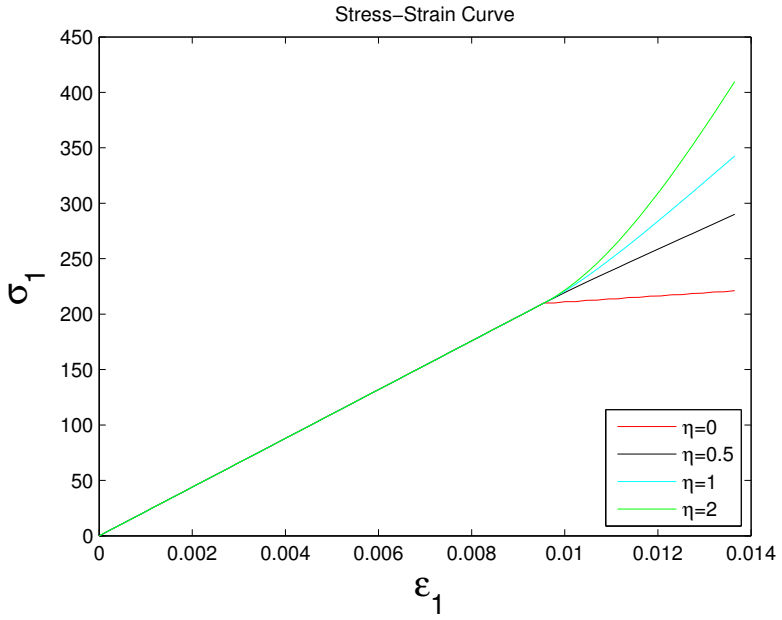

Figure 2.1: Evolution along time of the stress-space curve for different values of viscosity, η

Figure 2.1 shows the effect of the viscosity parameter on the stress-strain curve. While η increases it allows the material to reach higher stresses before starting the inelastic regime. It also can be seen that all the models behave identically while they are in the elastic domain due to the fact that the strain associated to the viscosity is zero in the elastic regime. We can therefore affirm thatthe implementation behaves as it was expected.

2.2 STRAIN RATE INFLUENCE ON THE STRESS-STRAIN CURVE

In order to show the evolution of the stress-strain curve as a function of the strain rate we have fixed all the parameters and paths except for the total time. The total time allows us to control how the strains change along time, so the higher total time, the minor strain rate during the process.

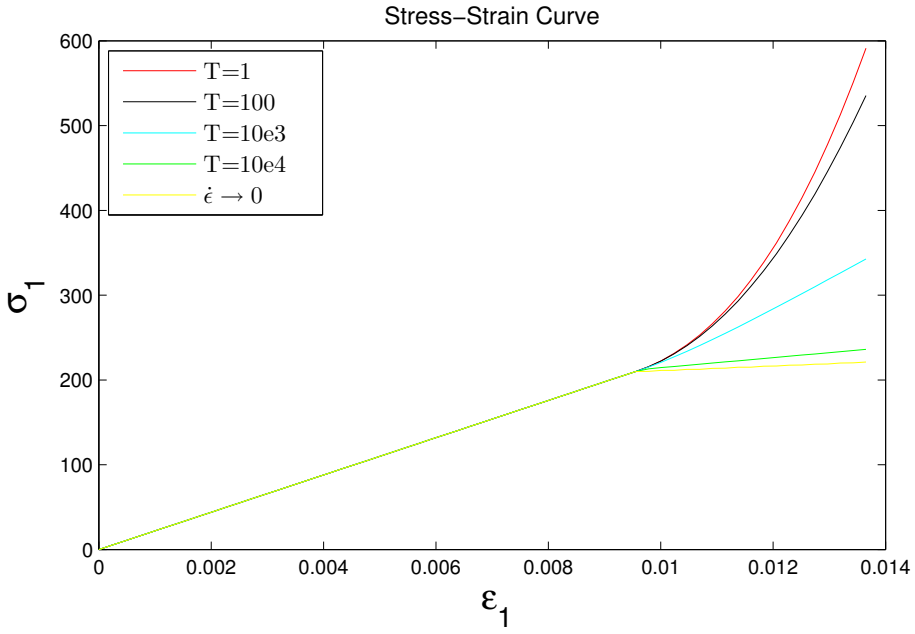

Figure 2.2: Evolution along time of the stress-space curve for different values of the strain rate $\dot{\epsilon}$

As figure 2.2 shows, as the total time is increasing (and consequently the strain rate is decreasing) less stress is seen for the same strain values. This figure also shows that in the rate dependent models stresses do not only depend on the strains, but also on the strain rate. This means that the stress tensor can change even if ϵ remains constant.

Moreover, it can be observed that when the total time is high enough as to consider $\dot{\epsilon} \to 0$

then the model matches the inviscid one.

2.3 α-PARAMETER EFFECTS ON STRESS-STRAIN CURVE

As in the previous sections, in order to see the behaviour of the stress-strain curve as a function of the time-integration coefficient α we will keep all the parameters fixed except for this one. This way:

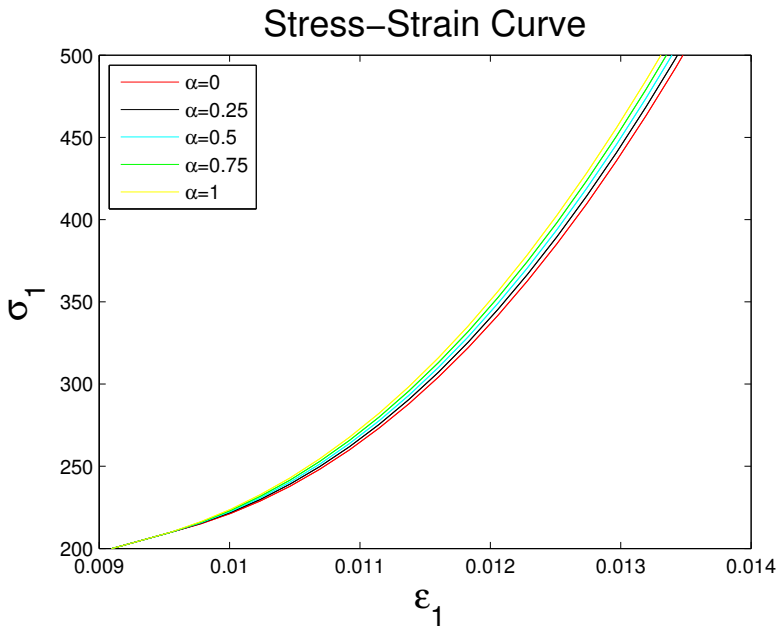

Figure 2.3: Evolution along time of the stress-space curve for different values of α

In figure 2.3 it can be observed that the evolution of the stress-strain curve depends on the time-integration coefficient, which means that we must be conscious of the importance of this parameter and its effects.

Provided we want to use an explicit scheme ($\alpha\epsilon[0,0.5)$) a little time step must be considered, since these methods are conditionally stable and could show oscillations. Implicit schemes ($\alpha\epsilon[0.5,1]$) are unconditionally stable, but if we want second order accuracy the best choice would be to select $\alpha = 0.5$. For this reason we have fixed α to 0.5 in all the other sections.

2.4 α-PARAMETER EFFECTS ON THE EVOLUTION ALONG TIME OF THE C_{11} COMPONENT OF THE TANGENT AND ALGORITHMIC CONSTITUTIVE OPERATORS.

To obtain the influence of α on the evolution of the tangent and algorithmic constitutive operators let us fix all the other parameters and obtain the evolution along time for both of them:

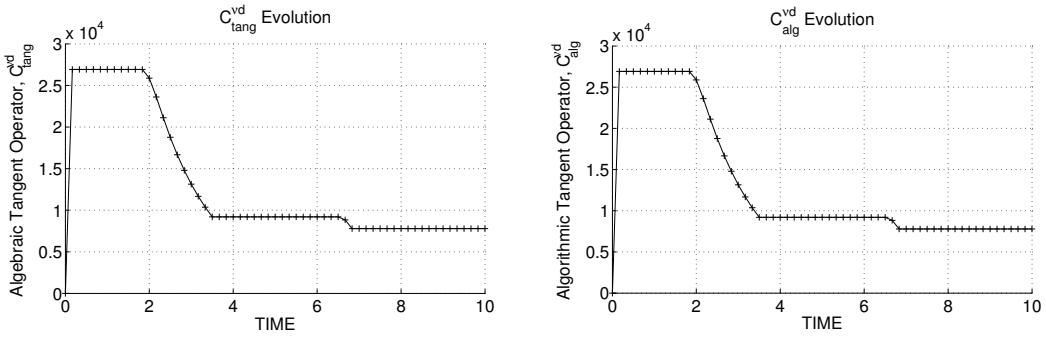

Figure 2.4: Evolution along time of the C_{11} component of the tangent(left) and algorithmic(right) constitutive operator for $\alpha = 0$

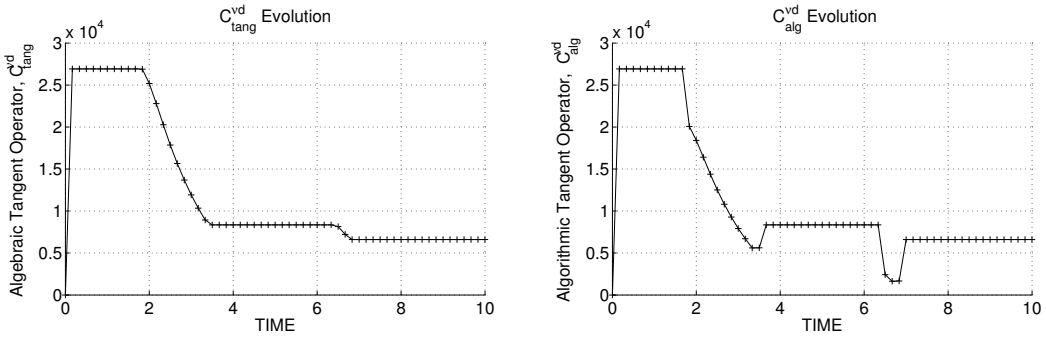

Figure 2.5: Evolution along time of the C_{11} component of the tangent(left) and algorithmic(right) constitutive operator for $\alpha = 0.5$

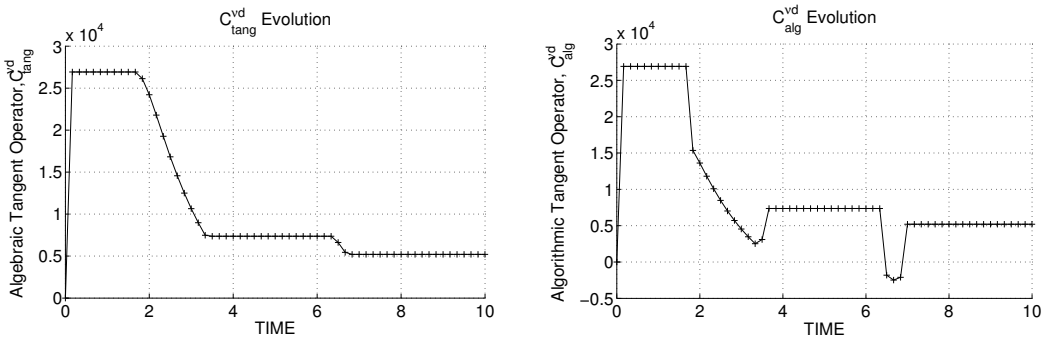

Figure 2.6: Evolution along time of the C_{11} component of the tangent(left) and algorithmic(right) constitutive operator for $\alpha = 1$

As it can be seen in figure 2.4 both components are identical when $\alpha = 0$, wherea when α increases they become different due to the needed additional term that must be computed to reproduce properly the behaviour of the material (see figure 2.5 and figure 2.6).

Bear in mind that this term depends on $\boldsymbol{\alpha}$, the damage parameter $\boldsymbol{d}$,the time step $\boldsymbol{\Delta t}$, the viscosity parameter $\boldsymbol{\eta}$ and the effective stress $\bar{\boldsymbol{\sigma}}$. It should vanish when either α or Δt tend to 0. In addition, it can be shown that when η vanishes it matches the inviscid constitutive operator.

Appendix 1: Listing of modified functions

- **damage_main.m**

```
function [sigma_v,vartoplot,LABELPLOT,TIMEVECTOR]=damage_main(Eprop,ntype,istep,strain,
    MDtype,n,TimeTotal)
global hplotSURF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CONTINUUM DAMAGE MODEL
% ------------------------------------------------------
% Given the almansi strain evolution ("strain(totalstep,mstrain)") and a set of
% parameters and properties, it returns the evolution of the cauchy stress and other
    variables
% that are listed below.
%
% INPUTS <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
% ----------------------------------------------------------------------------------------
% Eprop(1) = Young's modulus  (E)
% Eprop(2) = Poisson's coefficient (nu)
% Eprop(3) = Hardening(+)/Softening(-) modulus (H)
% Eprop(4) = Yield stress (sigma_y)
% Eprop(5) = Type of Hardening/Softening law  (hard_type)
%            0 --> LINEAR
%            1 --> Exponential
% Eprop(6) = Rate behavior (viscpr)
%            0 --> Rate-independent (inviscid)
%            1 --> Rate-dependent   (viscous)
%
% Eprop(7) = Viscosity coefficient (eta)  (dummy if inviscid)
% Eprop(8) = ALPHA coefficient (for time integration), (ALPHA)
%             0<=ALPHA<=1 , ALPHA = 1.0 --> Implicit
%                           ALPHA = 0.0 --> Explicit
%            (dummy if inviscid)
%
% ntype    = PROBLEM TYPE
%            1 : plane stress
%            2 : plane strain
%            3 : 3D
%
% istep = steps for each load state (istep1,istep2,istep3)
%
% strain(i,j) = j-th component of the linearized strain vector at the i-th
%               step, i = 1:totalstep+1
%
% MDtype      = Damage surface criterion %
%            1 : SYMMETRIC
%            2 : ONLY-TENSION
%            3 : NON-SYMMETRIC
%
%
% n          = Ratio compression/tension strength (dummy if MDtype is different from 3)
%
% TimeTotal  = Interval length
```

```
%
%  OUTPUTS <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
%  ------------------------------------------------------------------
%  1) sigma_v{itime}(icomp,jcomp)  --> Component (icomp,jcomp) of the cauchy
%                                      stress tensor at step "itime"
%                                     REMARK: sigma_v is a type of
%                                      variable called "cell array".
%
%
%  2) vartoplot{itime}              --> Cell array containing variables one wishes to
    plot
%                                       ---------------------------------------
%   vartoplot{itime}(1) =   Hardening variable (q)
%   vartoplot{itime}(2) =   Internal variable (r)%

%
%  3) LABELPLOT{ivar}              --> Cell array with the label string for
%                                      variables of "varplot"
%
%          LABELPLOT{1} => 'hardening variable (q)'
%          LABELPLOT{2} => 'internal variable'
%
%
%  4) TIME VECTOR  - >
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% SET LABEL OF "vartoplot" variables  (it may be defined also outside this function)
% ------------------------------------
LABELPLOT = {'hardening variable (q)','internal variable','Algebraic Tangent Operator',
     'Algorithmic Tangent Operator'};

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)

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
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          if viscpr ==1
            eps_n = strain(i-1,:);
            [sigma_n1,hvar_n,aux_var, C_alg,C_tan] = rmap_dano2(eps_n,eps_n1,delta_t,
                hvar_n,Eprop,ce,MDtype,n);
        else
        [sigma_n1,hvar_n,aux_var] = rmap_dano1(eps_n1,hvar_n,Eprop,ce,MDtype,n);
        end
```

```
        % 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) = C_alg(1,1);
%         vartoplot{i}(5) = C_tan(1,1);
    end
end
```

- **dibujar_criterio_dano1.m**

```
1
2 function hplot = dibujar_criterio_dano1(ce,nu,q,tipo_linea,MDtype,n)
3 %*************************************************************************
4 %*                  PLOT DAMAGE SURFACE CRITERIUM: ISOTROPIC MODEL
5                                   %*
6 %*                                                                       %*
7 %*      function [ce] = tensor_elastico (Eprop, ntype)                   %*
8 %*                                                                       %*
9 %*      INPUTS                                                           %*
10%*                                                                       %*
11%*                  Eprop(4)    vector de propiedades de material        %*
12%*                                    Eprop(1)=  E------->modulo de Young %*
13%*                                    Eprop(2)=  nu------>modulo de Poisson %*
14%*                                    Eprop(3)=  H------>modulo de Softening/hard. %*
15%*                                    Eprop(4)=sigma_u------>tension ultima %*
16%*                    ntype                          %*
17%*                                 ntype=1  plane stress                 %*
18%*                                 ntype=2  plane strain                 %*
19%*                                 ntype=3  3D                           %*
20%*                    ce(4,4)     Constitutive elastic tensor  (PLANE S.    ) %*
21%*                    ce(6,6)                                  ( 3D)        %*
22%*************************************************************************
23
24
25%*************************************************************************
26%*       Inverse ce                                                      %*
27ce_inv=inv(ce);
28c11=ce_inv(1,1);
29c22=ce_inv(2,2);
30c12=ce_inv(1,2);
31c21=c12;
32c14=ce_inv(1,4);
33c24=ce_inv(2,4);
34%*************************************************************************
35
36
37
38
39
40
41%*************************************************************************
42% POLAR COORDINATES
43if MDtype==1
44    tetha=[0:0.01:2*pi];
45    %
          ***************************************************************************************

46    %* RADIUS
47    D=size(tetha);                    %*  Range
48    m1=cos(tetha);                    %*
49    m2=sin(tetha);                    %*
```

```
50      Contador=D(1,2);                        %*
51
52
53      radio = zeros(1,Contador) ;
54      s1    = zeros(1,Contador) ;
55      s2    = zeros(1,Contador) ;
56
57      for i=1:Contador
58          radio(i)= q/sqrt([m1(i) m2(i) 0 nu*(m1(i)+m2(i))]*ce_inv*[m1(i) m2(i) 0 ...
59              nu*(m1(i)+m2(i))]');
60
61          s1(i)=radio(i)*m1(i);
62          s2(i)=radio(i)*m2(i);
63
64      end
65      hplot =plot(s1,s2,tipo_linea);
66
67
68  elseif MDtype==2
69  %   Comment/delete lines below once you have implemented this case
70   %  ***************************************************
71     tetha=[(-pi/2)*0.99:0.01:pi*0.99];
72      %
           ********************************************************************************************

73      %* RADIUS
74      D=size(tetha);                          %*  Range
75      m1=cos(tetha);                          %*
76      m2=sin(tetha);                          %*
77      Contador=D(1,2);                        %*
78
79
80      radio = zeros(1,Contador) ;
81      s1    = zeros(1,Contador) ;
82      s2    = zeros(1,Contador) ;
83
84      for i=1:Contador
85          sigmaMcAuley=[m1(i) m2(i) 0 nu*(m1(i)+m2(i))];
86          sigmaMcAuley=sigmaMcAuley.*(sigmaMcAuley>0);
87          radio(i)= q/sqrt(sigmaMcAuley*ce_inv*[m1(i) m2(i) 0 nu*(m1(i)+m2(i))]');
88
89          s1(i)=radio(i)*(m1(i));
90          s2(i)=radio(i)*(m2(i));
91
92      end
93      hplot =plot(s1,s2,tipo_linea);
94
95
96
97
98  elseif MDtype==3
99
100     tetha=[0:0.01:2*pi];
101     %
```

```
        ********************************************************************************

    %* 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

        m1mcauley=m1(i)*(m1(i)>0);
        m2mcauley=m2(i)*(m2(i)>0);
        m1abs=abs(m1(i));
        m2abs=abs(m2(i));
        tetha_aux=(m1mcauley+m2mcauley)/(m1abs+m2abs);

        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))]')*(tetha_aux+((1-tetha_aux)/n)));

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

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


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


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

- **Modelos_de_dano1.m**

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


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

elseif (MDtype==2)  %* Only tension
rtrial= sqrt((eps_n1.*(eps_n1>0))*ce*eps_n1');

elseif (MDtype==3)  %*Non-symmetric
s_n1=ce*eps_n1';
s1=s_n1(1);
s2=s_n1(2);
tetha_aux=((s1*(s1>0))+(s2*(s2>0)))/(abs(s1)+abs(s2));
rtrial= (tetha_aux+((1-tetha_aux)/n))*sqrt(eps_n1*ce*eps_n1');

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

- **rmap_dano1.m**

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

%*************************************************************************************
%*                                                                                   *
%*            Integration Algorithm for a isotropic damage model                     *
%*                                                                                   *
%*                                                                                   *
%*             [sigma_n1,hvar_n1,aux_var] = rmap_dano1 (eps_n1,hvar_n,Eprop,ce)      *
%*                                                                                   *
%* INPUTS              eps_n1(4)    strain (almansi)     step n+1                    *
%*                                  vector R4     (exx eyy exy ezz)                 *
%*                     hvar_n(6)    internal variables , step n                    *
%*                                  hvar_n(1:4) (empty)                            *
%*                                  hvar_n(5) = r   ; hvar_n(6)=q                  *
%*                     Eprop(:)     Material parameters                            *
%*                                                                                   *
%*                     ce(4,4)      Constitutive elastic tensor                    *
%*                                                                                   *
%* OUTPUTS:            sigma_n1(4) Cauchy stress   , step n+1                      *
%*                     hvar_n(6)    Internal variables , step n+1
                                    *
%*                     aux_var(3)   Auxiliar variables for computing const. tangent
     tensor   *
%*************************************************************************************


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


%*************************************************************************************
%*        initializing                                                          %*
r0 = sigma_u/sqrt(E);
zero_q=1.d-6*r0;
inf_q= 2*r0-zero_q;
% if(r_n<=0.d0)
%      r_n=r0;
%      q_n=r0;
% end
%*************************************************************************************


%*************************************************************************************
%*        Damage surface                                                            %*
```

```
51[rtrial] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
52%*****************************************************************************
53
54
55%*****************************************************************************
56%*   Ver el Estado de Carga                                                  %*
57%*   ---------->    fload=0 : elastic unload                                 %*
58%*   ---------->    fload=1 : damage (compute algorithmic constitutive tensor) %*
59fload=0;
60
61if(rtrial > r_n)
62    %*   Loading
63    fload=1;
64    delta_r=rtrial-r_n;
65    r_n1= rtrial  ;
66    if hard_type == 0
67        %  Linear
68        q_n1= q_n+ H*delta_r;
69    else
70        %  Exponential
71        if H>0
72        dqdr = H*((inf_q-r_n)/r_n)*exp(H*(1-(r_n1/r_n)));
73        q_n1 = q_n + dqdr*delta_r;
74        elseif H<0
75        dqdr = H*((r_n-inf_q)/r_n)*exp(H*(1-(r_n1/r_n)));
76        q_n1 = q_n - dqdr*delta_r;
77        end
78    end
79
80    if(q_n1<zero_q)
81        q_n1=zero_q;
82        %Acotamos q por arriba con el valor q_infinito que se obtiene a
83        %partir de zero_q y r0.
84    elseif(q_n1>inf_q)
85        q_n1=inf_q;
86    end
87else
88    %*     Elastic load/unload
89    fload=0;
90    r_n1= r_n  ;
91    q_n1= q_n  ;
92end
93% Damage variable
94% ----------------
95dano_n1   = 1.d0-(q_n1/r_n1);
96%  Computing stress
97%  ****************
98sigma_n1  =(1.d0-dano_n1)*ce*eps_n1';
99%hold on
100%plot(sigma_n1(1),sigma_n1(2),'bx')
101
102%*****************************************************************************
103
104
```

```
%*****************************************************************************
%* 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;
%*****************************************************************************
```

- **rmap_dano2.m**

```
1
2 function [sigma_n1,hvar_n1,aux_var,C_alg,C_tan] = rmap_dano2 (eps_n,eps_n1,delta_t,
      hvar_n,Eprop,ce,MDtype,n)
3
4 %*****************************************************************************
5 %*                                                                           *
6 %*     Integration Algorithm for a isotropic damage model rate dependent     *
7 %*                                                                           *
8 %*                                                                           *
9 %*             [sigma_n1,hvar_n1,aux_var,C_tan] = rmap_dano2 (eps_n,eps_n1,hvar_n,Eprop,
      ce)          *
10 %*                                                                          *
11 %* INPUTS               eps_n1(4)    strain (almansi)     step n+1         *
12 %*                                     vector R4     (exx eyy exy ezz)     *
13 %*                      eps_n1(4)    strain (almansi)     step n           *
14 %*                                     vector R4     (exx eyy exy ezz)     *
15 %*                      hvar_n(6)    internal variables , step n           *
16 %*                                     hvar_n(1:4) (empty)                 *
17 %*                                     hvar_n(5) = r   ; hvar_n(6)=q       *
18 %*                      Eprop(:)     Material parameters                   *
19 %*                                                                          *
20 %*                      ce(4,4)      Constitutive elastic tensor           *
21 %*                                                                          *
22 %* OUTPUTS:            sigma_n1(4) Cauchy stress   , step n+1             *
23 %*                      hvar_n(6)    Internal variables , step n+1
                                *
24 %*                      aux_var(3)   Auxiliar variables for computing const. tangent
      tensor   *
25 %*****************************************************************************
26
27
28 hvar_n1 = hvar_n;
29 r_n     = hvar_n(5);
30 q_n     = hvar_n(6);
31 E       = Eprop(1);
32 nu      = Eprop(2);
33 H       = Eprop(3);
34 sigma_u = Eprop(4);
35 hard_type = Eprop(5) ;
36 eta = Eprop(7);
37 alpha = Eprop(8);
38 %*****************************************************************************
39
40
41 %*****************************************************************************
42 %*       initializing                                                      %*
43 r0 = sigma_u/sqrt(E);
44 zero_q=1.d-6*r0;
45 q_inf=2*r0-zero_q;
46 % if(r_n<=0.d0)
47 %     r_n=r0;
48 %     q_n=r0;
```

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

%*****************************************************************************
%*        Damage surface                                                     %*
[rtrial_n1] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
[rtrial_n] = Modelos_de_dano1 (MDtype,ce,eps_n,n);
%*****************************************************************************

rtrial_aux = (1-alpha)*rtrial_n+alpha*rtrial_n1;
%*****************************************************************************
%*   Ver el Estado de Carga                                                  %*
%*   ---------->    fload=0 : elastic unload                                 %*
%*   ---------->    fload=1 : damage (compute algorithmic constitutive tensor) %*
fload=0;

if(rtrial_aux > r_n)
    %*   Loading

    fload=1;
    delta_r=rtrial_aux-r_n;
    r_n1= ((eta-delta_t*(1-alpha))/(eta+alpha*delta_t))*r_n + (delta_t/(eta+alpha*
        delta_t))*rtrial_aux  ;
    if hard_type == 0
        %  Linear hardening
        q_n1= q_n+ H*delta_r;
    else
        % Exponential hardening
%         dqdr = H*(q_inf - r_n)/r_n*exp(H*(1-r_n1/r_n));
%         q_n1 = q_n + dqdr*delta_r;
    end

    if(q_n1<zero_q)
        q_n1=zero_q;
    elseif (q_n1 > q_inf)
        q_n1=q_inf;
    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')

%*****************************************************************************
% Algorithmic tangent operator

if (rtrial_aux > r_n)
sigma_barra = ce*eps_n1';
C_alg = (1-dano_n1)*ce + ...
    (alpha*delta_t)/((eta+alpha*delta_t)*rtrial_n1)*((H*r_n1-q_n1)/r_n1^2)*(sigma_barra
        '*sigma_barra);
else
    C_alg = (1-dano_n1)*ce;
end

%Analytic Tangent operator

C_tan = (1-dano_n1)*ce;



%*****************************************************************************
%* 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;
%*****************************************************************************
```