Universitat Politecnica De Catalunya

Computational Solid Mechanics

Numerical Integration of Continuum Damage Models

Perfect Kudakwashe Marenga

March, 2020

Contents

1 Introduction

1.1 Isotropic Damage Models

Firstly, it is worth to mention that the term isotropic damage model comes from the fact that the model depends on a single variable, the damage parameter d, meaning we are assuming a mechanical behavior in which the degradation is independent of the orientation. Also these models are key in understanding the behavior of materials whose mechanical properties are degrading due to the presence of small cracks that propagate during loading.

In this report two main scenarios have been studied in relation to this model namely, rate-dependent and rate-independent models. In rate-independent models, only loading history is of essence, the rate at which the load is being applied is neglected under the assumption that the load is being applied slowly and material have time to react and deform.

However, in rate-dependent model, not only loading history but loading time and material viscosity play significant roles because different materials react differently to the rate at which the load is being applied to them. Due to the appearance of the time derivative (since time is now an independent variable) in this model, the finite difference methods also known as α methods have been implemented to approximate the resulting differential equation.

2 Rate-Independent Model

2.1 Hardening/Softening Law

There are basically two hardening/softening laws namely, linear and exponential law, their respective equations are shown below,

$$q(r) = \begin{cases} r_0 & r \leqslant 0 \\ r_0 + H(r - r_0) & r > r_0 \end{cases} \tag{1}$$

$$q(r) = q_\infty - (q_\infty - r_0) \exp^{A(1-\frac{r}{r_0})} \tag{2}$$

The implementation of the exponential law was compared with the linear law which was implemented before hand for correctness purposes. A symmetric, uni-axial tensile loading case was considered with the following, stress path $\Delta\bar{\sigma}_1^{(1)} = 200$; $\Delta\bar{\sigma}_1^{(2)} = 500$; $\Delta\bar{\sigma}_1^{(3)} = 800$, Hardening/softening parameter $(H) = -1$ (implying softening case). The results are shown below,

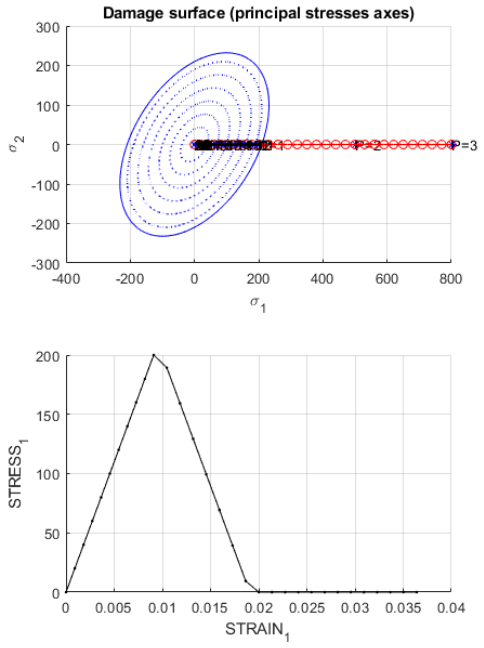

Figure 1: Linear Softening

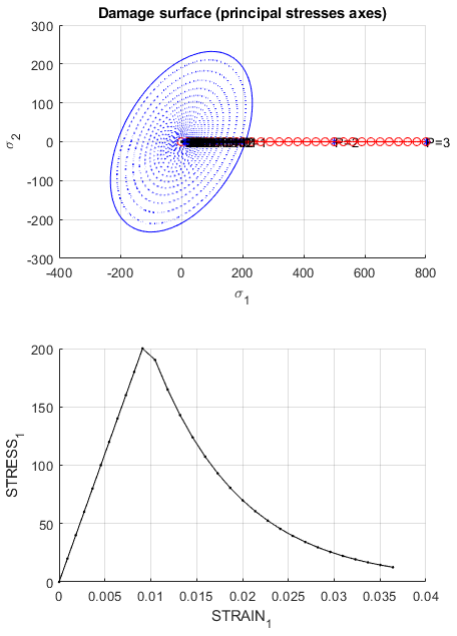

Figure 2: Exponential Softening

From the above figures a distinct difference in the laws can be seen from the $\sigma - \epsilon$ plots, in Figure 1 we see the material undergoes elastic tensile loading up-to $\sigma_u = 200$, after which tensile loading with damage initiates until the material loses all its load bearing capacity. In Figure 2, a different scenario is observed were after reaching σ_u, exponential softening initiates and continues into an asymptotic behavior. The corresponding damage variable (d) plots are shown below respectively.

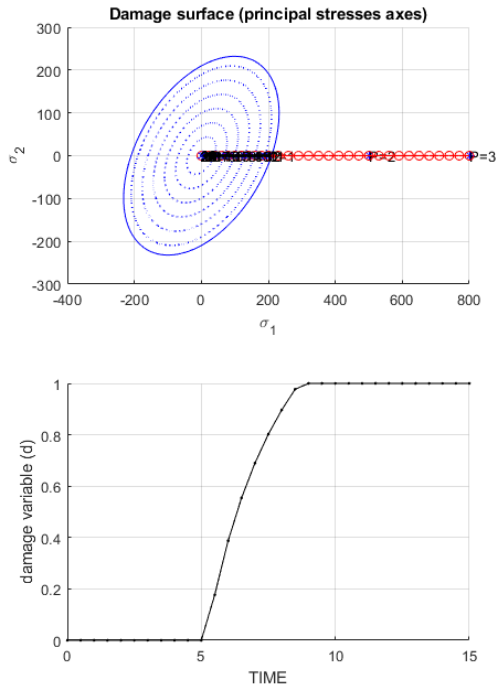

Figure 3: Damage with Linear Softening

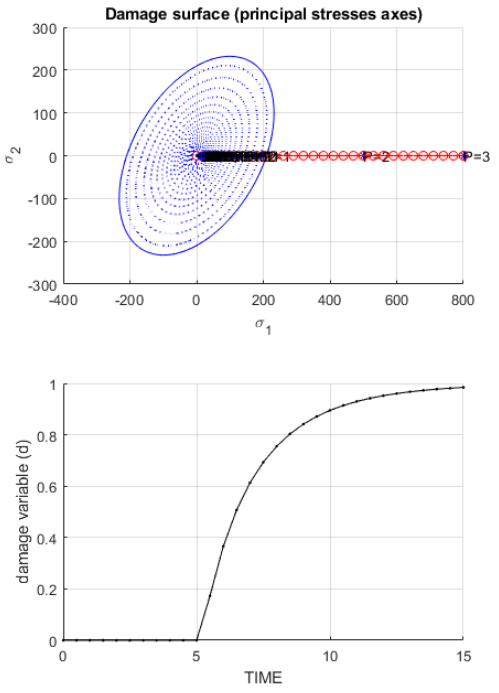

Figure 4: Damage with Exponential Softening

2.2 Tension-only damage model

In this model damage by compression is not taken into account. The effective stress tensor for this model becomes $\bar{\sigma}^+$. The surface equation characterising the stress state then becomes,

$$\tau_\sigma = \sqrt{\sigma^+ : C^{e^{-1}} : \sigma} \tag{3}$$

In the given Matlab code this model is implemented by setting all the negative values of the effective stress tensor ($\bar{\sigma}$) to zero in the *Modelos_de_dano1* function.

2.3 Non-symmetric tension-compression damage model

The non-symmetrical damage model is important in simulating materials whose tension domain differs with respect to compression. The implemented surface equation characterising the stress state is given below,

$$\tau_\sigma = \left(\theta + \frac{1-\theta}{n}\right) \sqrt{\sigma : C^{e^{-1}} : \sigma} \tag{4}$$

where n compression-tension ration and θ is the weight factor given by,

$$\theta = \frac{\sum_{i=1}^{3} \langle \sigma_i \rangle}{\sum_{i=1}^{3} |\sigma_i|} \tag{5}$$

The implementation of this model was also done in the given Matlab *Modelos_de_dano1* function.

2.4 Validation of Implementation

For validation purposes only the implemented exponential law was used and the following parameters, hardening parameter ($H = 1$) (damage with hardening case), $\sigma_y = 200$, $\nu = 0.3$, $E = 20000$. The values of α, β, γ, were taken as shown below,

$$\alpha = 400, \ \beta = 700, \ \gamma = 1000$$

2.4.1 Load scenario 1: uni-axial loading/unloading

The loading path for this case is shown below,

$$\begin{array}{llll} \Delta\bar{\sigma}_1^{(1)} = \alpha & \Delta\bar{\sigma}_2^{(1)} = 0 & \text{(uni-axial tensile loading)} \\ \Delta\bar{\sigma}_1^{(2)} = -\beta & \Delta\bar{\sigma}_2^{(2)} = 0 & \text{(uni-axial tensile unloading/compressive loading)} \\ \Delta\bar{\sigma}_1^{(3)} = \gamma & \Delta\bar{\sigma}_2^{(3)} = 0 & \text{(uni-axial compressive unloading/ tensile loading)} \end{array}$$

The figure below shows the damage surfaces and the stress-strain curve for both the tension-only and the non-symmetrical models. From the figures, it can be noted that

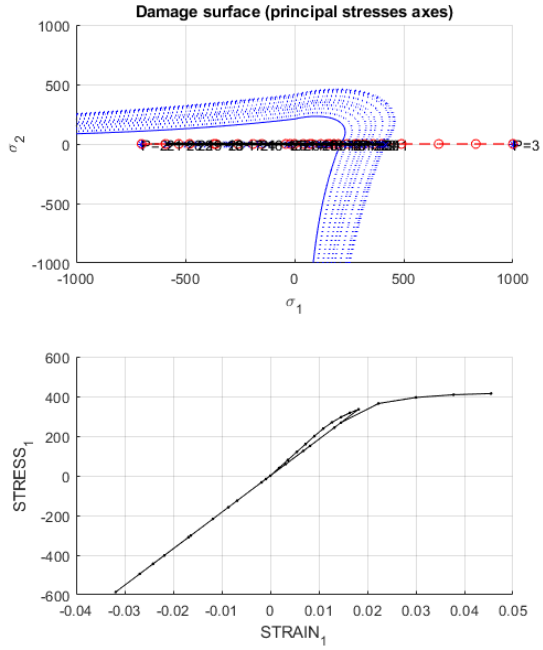

Figure 5: Tension-only damage model

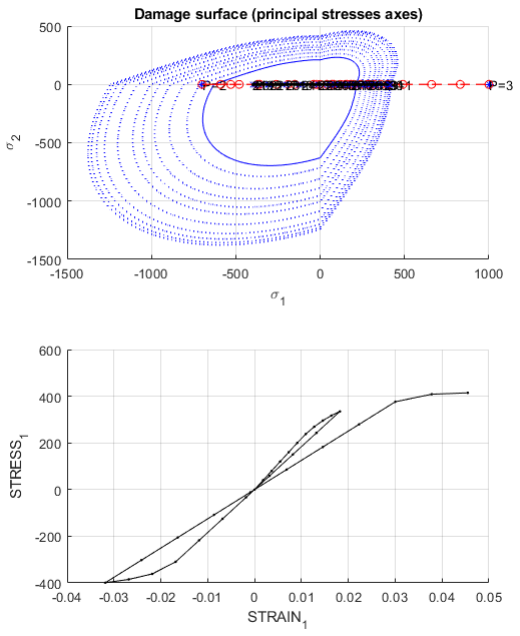

Figure 6: Non-symmetrical model

the damage surfaces expands, and this corresponds to the damage with hardening case chosen for implementation, also we can see the actual stress path (black) as compared to the theoretical path chosen (red). Considering the stress-strain curves, for the tension-only damage model we can observe a linear relationship up-to $\sigma_u = 200$ (elastic region), after which an exponential hardening can be observed before tensile unloading/compressive loading begins. Since the tension-only damage model doesn't take into account damage by compression, a linear (elastic behaviour) dependence is shown during compressive loading and unloading, after which tensile

loading begins again at ($\sigma_y \approx 300$), and a continued hardening can be observed. In the non-symmetrical model, damage by compression can be observed due to the change of slope during compressive loading/unloading.

2.4.2 Load scenario 2: uni-axial and bi-axial loading/unloading

The loading path for this case is shown below,

$$\begin{array}{lll} \Delta\bar{\sigma}_1^{(1)} = \alpha & \Delta\bar{\sigma}_2^{(1)} = 0 & \text{(uni-axial tensile loading)} \\ \Delta\bar{\sigma}_1^{(2)} = -\beta & \Delta\bar{\sigma}_2^{(2)} = -\beta & \text{(bi-axial tensile unloading/compressive loading)} \\ \Delta\bar{\sigma}_1^{(3)} = \gamma & \Delta\bar{\sigma}_2^{(3)} = \gamma & \text{(bi-axial compressive unloading/tensile loading)} \end{array}$$

The figure below shows the damage surfaces and the stress-strain curve for both the tension-only and the non-symmetrical models. From the figures shown the main

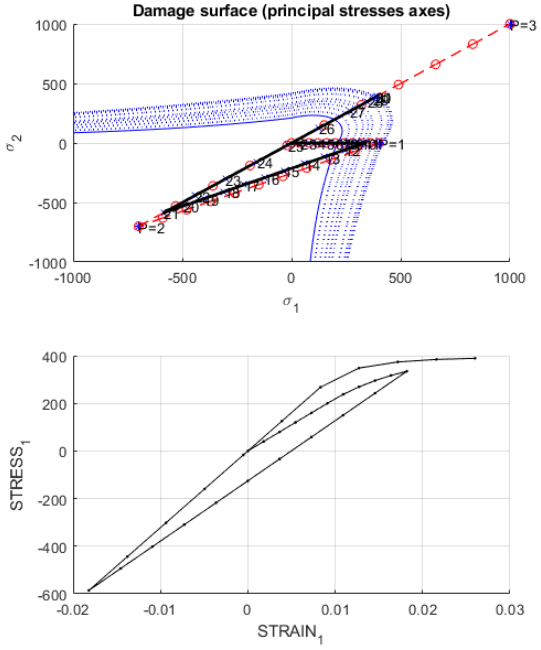

Figure 7: Tension-only damage model

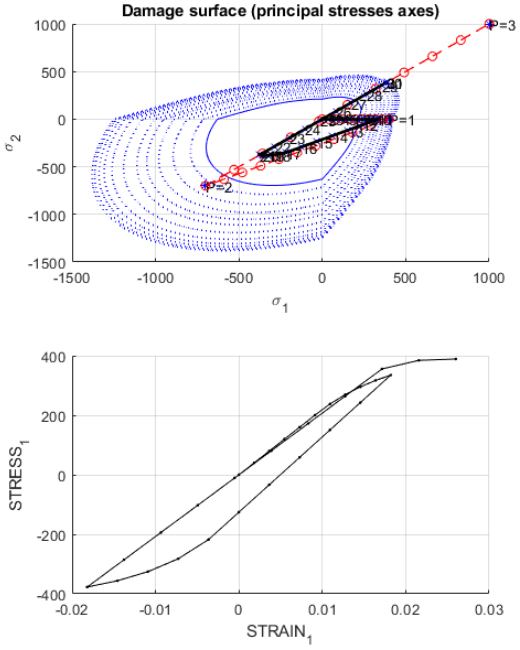

Figure 8: Non-symmetrical model

difference between this case and the previous one is the fact of unsymmetrical loading path caused due to change from uni-axial loading to bi-axial loading. This is evident as the second loading path doesn't pass through the origin (in the stress-strain curve) which is the case for symmetrical loading observed in case 1. Otherwise, similar features as those observed from case 1 can be seen, for example damage with hardening after the elastic region, in both models.

2.4.3 Load scenario 3: bi-axial loading/unloading

The loading path for this case is shown below,

$$\begin{array}{llll} \Delta\bar{\sigma}_1^{(1)} = \alpha & \Delta\bar{\sigma}_2^{(1)} = \alpha & \text{(bi-axial tensile loading)} \\ \Delta\bar{\sigma}_1^{(2)} = -\beta & \Delta\bar{\sigma}_2^{(2)} = -\beta & \text{(bi-axial tensile unloading/compressive loading)} \\ \Delta\bar{\sigma}_1^{(3)} = \gamma & \Delta\bar{\sigma}_2^{(3)} = \gamma & \text{(bi-axial compressive unloading/tensile loading)} \end{array}$$

The figure below shows the damage surfaces and the stress-strain curve for both the tension-only and the non-symmetrical models. In this case symmetrical bi-axial

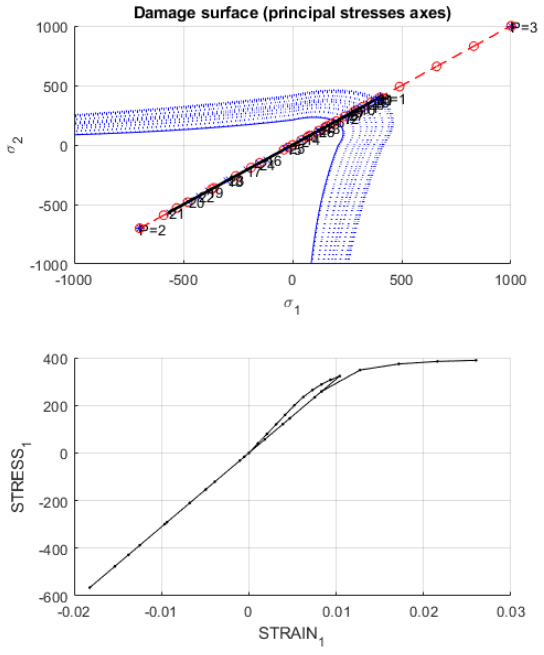

Figure 9: Tension-only damage model

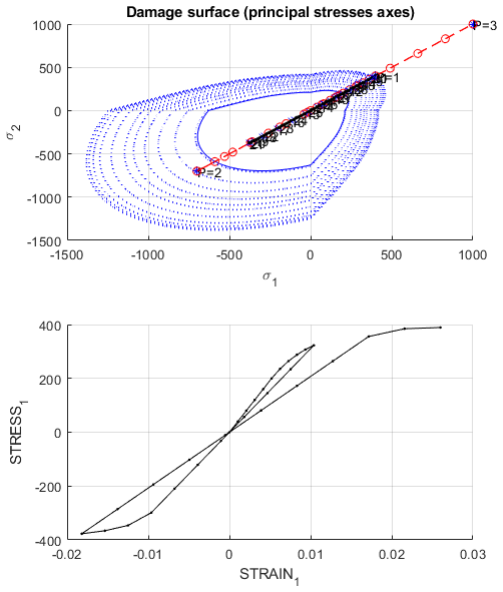

Figure 10: Non-symmetrical model

loading was implemented, and stress-strain curves looks the same as those obtained in case 1, the only difference is that in this case stress values corresponds to less values of strains and this is reasonable since we considering a bi-axial stress state.

In concluding, the stress-strain curves obtained in all 3 implementations shows the correct expected behaviour which validates the correctness of the implementations.

3 Rate-Dependent Model

In this model, viscosity is introduced meaning the stress tensor no longer depend only on strain history but also on time. Since the time is an independent variable an ordinary first order differential equation for calculating the internal variable r of the damage variable d has to be solved. A well known numerical technique namely the finite difference method was implemented for approximating the differential equation, this technique of integration depends on the parameter α, typical the values are $\alpha = 0$ (forward Euler (explicit scheme)), $\alpha = 1$ (backward Euler (implicit scheme)) and $\alpha = 0.5$ (Crank-Nicolson (implicit scheme)).
A symmetric, uni-axial tensile loading case was considered with the following, stress path $\Delta\bar{\sigma}_1^{(1)} = 200$; $\Delta\bar{\sigma}_1^{(2)} = 400$; $\Delta\bar{\sigma}_1^{(3)} = 600$. Also it is worth to note that in this model (following the figure below), the actual stress path (in black) is allowed to surpass the elastic boundary, which is not the case in the inviscid model.

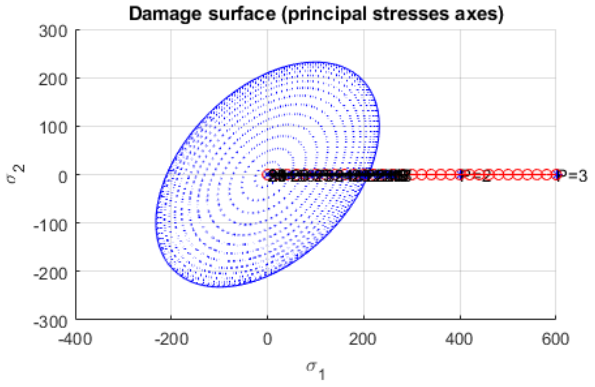

Figure 11: Damage surface Viscous Model

3.1 Effects of viscosity on stress-strain curve

In this section the effects of viscosity on the stress-strain curve was studied. For $\eta = 0$ the non-viscous case is retrieved. When viscosity is introduced (see figure below) the linear softening does not start immediately after reaching the prescribed yield stress. Rather hardening and a higher maximum stress in form of a rounded top can be observed.

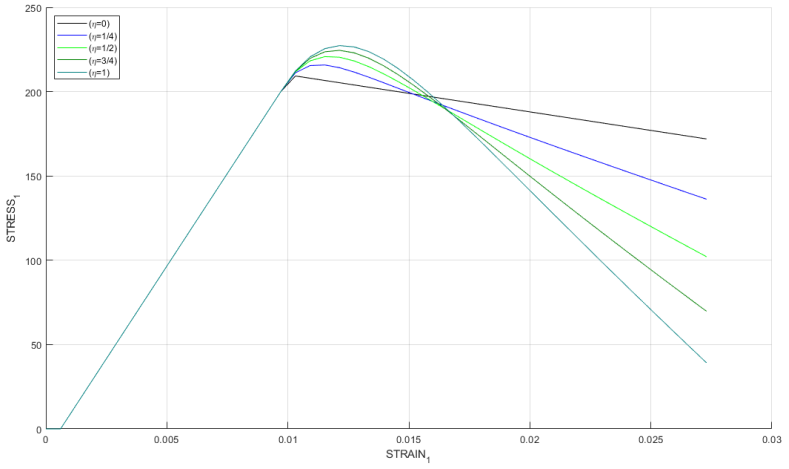

Figure 12: $\sigma - \epsilon$ curves for different values of η

3.2 Effects of strain rate on stress-strain curve

In this section the effects of strain rate has been studied. To evaluate the effects of strain rate, the viscosity coefficient has been set to a constant ($\eta = 0.5$). Since the strain rate is not a direct input parameter, it has been accounted for by manipulating the total loading time in the code, as it well known from theory that the strain rate is indirectly proportional to loading time. And the figure below shows the effects of strain rate on the stress values.

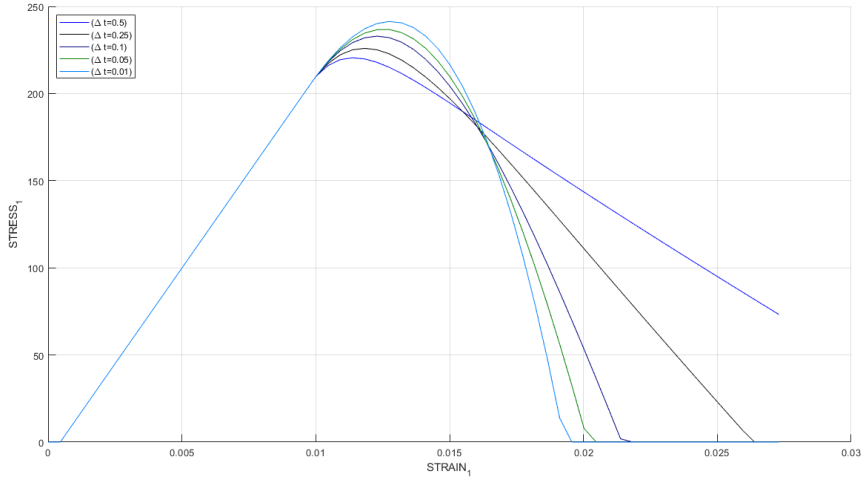

Figure 13: $\sigma - \epsilon$ curves for different values of $\dot{\epsilon}$

It can be seen from Figure 13, that the strain rate does have the same effect on the stress-strain curve as the viscosity coefficient.

3.3 Effect of α on Tangent and Algorithmic constitutive operator (C11)

Finally, the effect of the coefficient α, arising from integration scheme, on the constitutive operators is studied. Before an in-depth discussion, it is worthwhile to test the correctness of the implementation, this is done by setting $\alpha = 0$, for we know that in this case the constitutive algorithmic operator collapses into the tangent constitutive operator, the plot for this case is shown below, for a stable Δt,

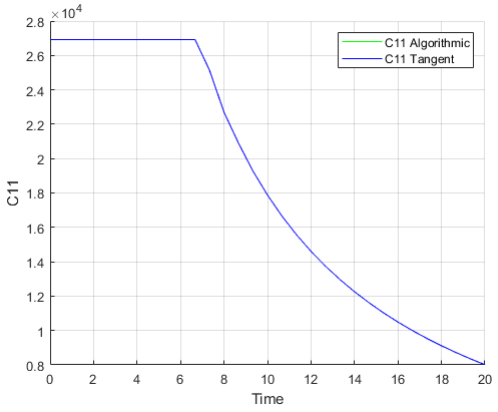

Figure 14: C11 Algorithmic and Tangent for $\alpha = 0$

Having validated the correctness of the implementation, now we can assess the effects of α on the constitutive operators, having chosen a bigger time step it can be evidently seen from both Fig. 15 and Fig. 16, that for values of $\alpha < 0.5$, oscillations are observed in the solution, this because of the discretization scheme used (Forward Euler), an explicit scheme which is conditionally stable. For $\alpha \leqslant 0.5$, no oscillations are observed, because of the implicit nature of the schemes meaning they are unconditionally stable.
Also it is worth to note that the constant straight line in the plots represents elastic loading where damage is absent, and there after damage begins and the values of both constitutive operators decreases rapidly.

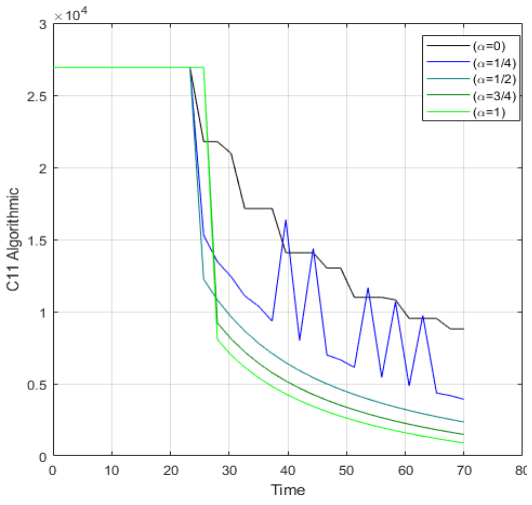

Figure 15: C11 Algorithmic for different values of α

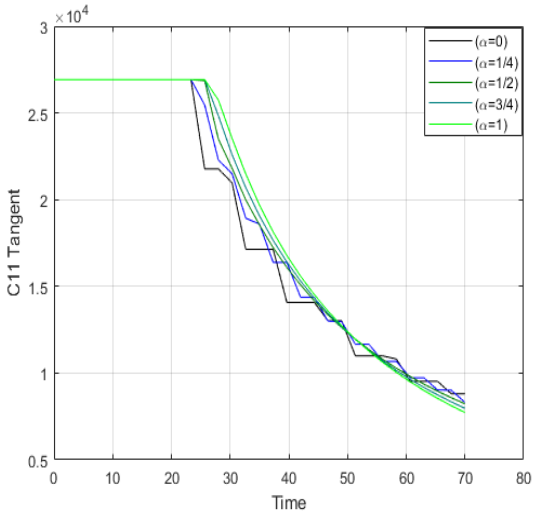

Figure 16: C11 Tangent for different values of α

A Codes for inviscid Model

Listing 1: Function Modelos de dano1

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

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

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

elseif (MDtype == 3)  %*Non-symmetric
    sigma_ = ce * eps_n1';
    sigma_p = sigma_;
    sigma_(sigma_p < 0) = 0;
    theta = sum(sigma_p)/sum(abs(sigma_));
    coeff = theta + (1 - theta)/n;
    rtrial = coeff * sqrt(eps_n1*ce*eps_n1');

end

return
```

Listing 2: Function rmap dano

```
function [sigma_n1,hvar_n1,aux_var] =...
                        rmap_dano(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);
eta       = Eprop(7);
alpha     = Eprop(8);

%*       initializing
 r0 = sigma_u/sqrt(E);
 zero_q = 1.d-6*r0;
 q_inf = r0  + sign(H)*(r0 - zero_q);
 A = abs(H);

%*       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;

if(rtrial > r_n)
    %*   Loading
    fload=1;
    Δ_r=rtrial-r_n;
    r_n1= rtrial  ;
    if hard_type == 0
        %  Linear
        q_n1= q_n+ H*Δ_r;
    else
        %  Exponential
        x = exp(A*(1 - r_n1/r0));
        q_n1 = q_inf - (q_inf - r0)*x;

    end

    if(q_n1<zero_q)
        q_n1=zero_q;
    end

else

    % Elastic load/unload
    fload = 0;
    r_n1 = r_n  ;
```

```
55      q_n1 = q_n  ;
56
57  end
58  % Damage variable
59  % ----------------
60  dano_n1 = 1.d0-(q_n1/r_n1);
61  %  Computing stress
62  %  ****************
63  sigma_n1 = (1.d0-dano_n1)*ce*eps_n1';
64
65  % Computing consistent algorithmic and tangent operators
66
67  %* Updating historic variables
68  hvar_n1(5)= r_n1 ;
69  hvar_n1(6)= q_n1 ;
70
71  %* Auxiliar variables
72  aux_var(1) = fload;
73  aux_var(2) = q_n1/r_n1;
```

Listing 3: Function dibujar criterio dano1

```
1   function hplot = dibujar_criterio_dano1(ce,nu,q,tipo_linea,MDtype,n)
2
3   %* Inverse ce
4   ce_inv=inv(ce);
5   c11=ce_inv(1,1);
6   c22=ce_inv(2,2);
7   c12=ce_inv(1,2);
8   c21=c12;
9   c14=ce_inv(1,4);
10  c24=ce_inv(2,4);
11
12  % POLAR COORDINATES
13  %*****************************
14  if MDtype==1
15      tetha=[0:0.01:2*pi];
16
17      %* RADIUS
18      D=size(tetha);                          %*  Range
19      m1=cos(tetha);                          %*
20      m2=sin(tetha);                          %*
21      Contador=D(1,2);                        %*
22
23
24      radio = zeros(1,Contador) ;
25      s1    = zeros(1,Contador) ;
26      s2    = zeros(1,Contador) ;
27
28      for i=1:Contador
29          radio(i)= q/sqrt([m1(i) m2(i) 0 nu*(m1(i)+m2(i))]*ce_inv*...
30                    [m1(i) m2(i) 0 nu*(m1(i)+m2(i))]');
31
32          s1(i)=radio(i)*m1(i);
```

```
33          s2(i)=radio(i)*m2(i);
34
35      end
36      hplot =plot(s1,s2,tipo_linea);
37
38  elseif MDtype==2
39
40      tetha=[0:0.01:2*pi];
41      D = size (tetha);
42      m1 = cos (tetha);
43      m2 = sin (tetha);
44      Contador = D(1,2);
45      m1_p = m1;
46      m2_p = m2;
47      m1_p(m1_p < 0) = 0;
48      m2_p(m2_p < 0) = 0;
49      radio = zeros(1,Contador);
50      s1 = zeros(1,Contador);
51      s2 = zeros(1,Contador);
52
53      for i = 1:Contador
54          radio(i) = q/ sqrt([m1_p(i) m2_p(i) 0 nu*(m1_p( i ) + ...
55           m2_p(i))] * ce_inv * [m1(i) m2(i) 0 nu * (m1(i) + m2(i))]');
56          s1(i) = radio(i) * m1(i);
57          s2(i) = radio(i) * m2(i);
58      end
59      hplot =plot(s1,s2,tipo_linea);
60
61  elseif MDtype==3
62
63      tetha=[0:0.01:2*pi];
64      D = size(tetha);
65      m1 = cos (tetha);
66      m2 = sin (tetha);
67      Contador = D(1,2);
68      radio = zeros(1,Contador);
69      s1 = zeros (1,Contador);
70      s2 = zeros (1,Contador);
71      for i =1:Contador
72          sigma_ = [m1(i) m2(i) nu*(m1(i) + m2(i))];
73          sigma_p = sigma_;
74          sigma_p(sigma_p < 0) = 0;
75          theta = sum(sigma_p)/sum(abs(sigma_)) ;
76          coeff = theta + (1 - theta)/n ;
77          radio(i) = q /( coeff * sqrt([m1(i) m2(i) 0 nu * ...
78          (m1(i) + m2(i))] * ce_inv * [m1(i) m2(i) 0 nu *...
79          (m1(i) + m2(i))]'));
80          s1(i) = radio(i) * m1(i);
81          s2(i) = radio(i) * m2(i);
82      end
83      hplot = plot(s1,s2,tipo_linea);
84  end
85  return
```

Listing 4: Function damage main

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

% ---------------------------------------
 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   = 6    ;
end

totalstep = sum(istep) ;

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

% Elastic constitutive tensor
% -----------------------------
[ce]    = tensor_elastico1 (Eprop, ntype);
% Initz.
% ------
% Strain vector
% -------------
eps_n1  = zeros(mstrain,1);
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)+ Δ_t(iload) ;
        % Total strain at step "i"
        % ------------------------
        eps_n1 = strain(i,:) ;
        %*****************************************************
        %*      DAMAGE MODEL
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        [sigma_n1,hvar_n,aux_var] = ...
                          rmap_dano(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)
    end
end
end
```

B Codes for Viscous Model

Listing 5: Function Modelos de dano1

```
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');
end
%*********************************************************
return
```

Listing 6: Function rmap dano

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

hvar_n1 = hvar_n;
r_n     = hvar_n(5);
q_n     = hvar_n(6);
E       = Eprop(1);
nu      = Eprop(2);
H       = Eprop(3);
sigma_u = Eprop(4);
hard_type = Eprop(5) ;
eta     = Eprop(7);
alpha   = Eprop(8);

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

%*****************************************************
%*****************************************************
%*       Damage surface
[rtrial_1] = Modelos_de_dano1 (MDtype,ce,eps_n,n);
[rtrial_n] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
%*****************************************************

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

```
%*   Ver el Estado de Carga
%*   ---------->    fload=0 : elastic unload
%*   ---------->    fload=1 : damage
fload = 0;

rtrial = (1-alpha)*rtrial_1 + alpha*rtrial_n ;
if(rtrial > r_n)
    %   Loading

    fload = 1;
    Δ_r = rtrial-r_n;
    %r_n1= rtrial  ;
     r_n1 = (eta-Δ_t*(1-alpha))/(eta+alpha*Δ_t)*r_n + ...
                Δ_t/(eta+alpha*Δ_t)*rtrial;
    if hard_type == 0
        %  Linear
        q_n1 = q_n+ H*Δ_r;
    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*Δ_t/...
        (eta+alpha*Δ_t)*(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;
%*********************************************************
```

Listing 7: Function dibujar criterio dano1

```
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);
end
%*********************************************************
return
```

Listing 8: Function damage main

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

```

```
% SET LABEL OF "vartoplot" variables
% ------------------------------------
 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   = 6    ;
end

totalstep = sum(istep) ;

% INITIALIZING GLOBAL CELL ARRAYS
% -------------------------------
sigma_v = cell(totalstep+1,1) ;
TIMEVECTOR = zeros(totalstep+1,1) ;
Δ_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  = 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 Component
```

```
vartoplot{i}(5) = ce(1,1); % C11 Component
for  iload = 1:length(istep)
    % Load states
    for iloc = 1:istep(iload)
        i = i + 1 ;
        TIMEVECTOR(i) = TIMEVECTOR(i-1)+ Δ_t(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,Δ_t(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 Component
        vartoplot{i}(5) = C11_ana(1,1); % C11 Component
    end
end
```