

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



Centre Internacional
de Mètodes Numèrics
en Enginyeria

CONTENTS

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Rate Independent damage models | 3 |
| 2.1 | Tension-only damage model. | 3 |
| 2.2 | Non-symmetric tension-compression damage model. | 4 |
| 3 | Hardening | 5 |
| 3.1 | Linear Hardening/Softening | 5 |
| 3.2 | Exponential Hardening/Softening | 6 |
| 4 | Assessment of the inviscid model | 7 |
| 4.1 | Case 1 | 9 |
| 4.2 | Case 2 | 9 |
| 4.3 | Case 3 | 10 |
| 5 | Rate dependent damage models | 11 |
| 5.1 | α method | 11 |
| 5.2 | Implementation | 11 |
| 6 | Assessment of the viscid model | 12 |
| 6.1 | Stress-Strain behaviour | 13 |
| 6.1.1 | Case 1 | 13 |
| 6.1.2 | Case 2 | 13 |
| 6.1.3 | Case 3 | 14 |
| 6.2 | effects of α against C_{11} | 15 |
| | Appendices | 16 |
| | Appendix 2: <code>_criterio_dano1.m</code> | 16 |
| | Appendix 2: <code>rmap_dano1.m</code> | 18 |
| | Appendix 3: <code>q_fact_value.m</code> | 20 |
| | Appendix 4: <code>rmap_danovisc.m</code> | 21 |
| | Appendix 5: <code>damage_main.m</code> | 23 |
| | Appendix 6: <code>Modelos_de_dano1.m</code> | 30 |
| | Appendix 7: <code>main_nointeractive.m</code> | 30 |

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:

$$\tau_{\varepsilon} = \sqrt{\varepsilon : \mathbb{C} : \varepsilon} \quad (1.1)$$

- 2 Tension only model:

$$\tau_{\varepsilon} = \sqrt{\langle \varepsilon \rangle : \mathbb{C} : \varepsilon} \quad (1.2)$$

- 3 Non-symmetric tension-compression model:

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

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:

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

```

1 % Mcaulay brackets
2 m1_2(m1_2<0)=0;
3 m2_2(m2_2<0)=0;
4 m3_2(m3_2<0)=0;

```

```

5 % loop
6 radio(i) = q/sqrt([m1_2(i) m2_2(i) 0 m3_2(i)]*ce_inv*[m1(i) m2(i) 0 ...
7 m3(i)]');

```

The plot obtained is:

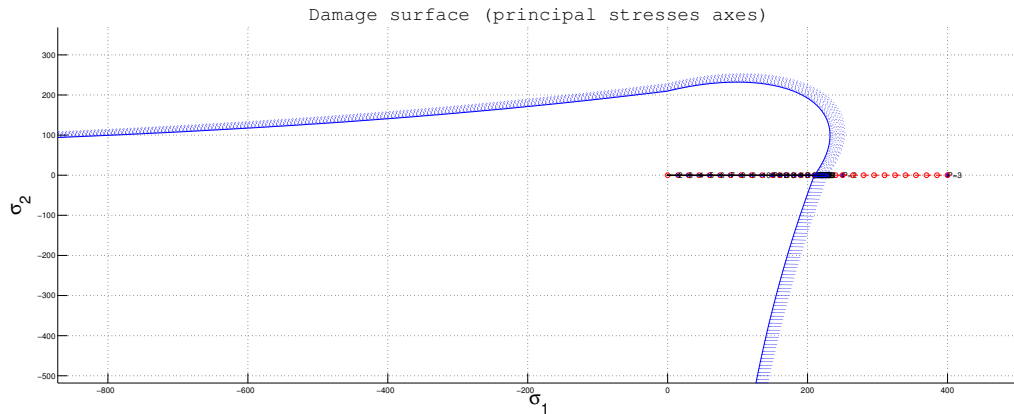


Figure 2.1: Tension only damage surface.

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 :

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

```

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

```

The plot obtained is:

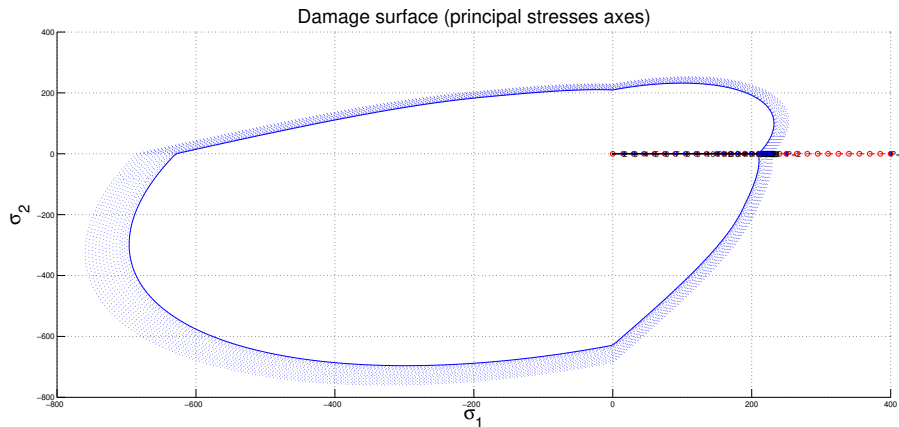


Figure 2.2: Non-symmetric tension-compression damage surface.

3 HARDENING

3.1 LINEAR HARDENING/SOFTENING

The hardening is calculated by:

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

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:

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

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

```

1  if(rtrial > r_n)
2  %* Loading
3  fload=1;delta_r=rtrial-r_n;r_n1= rtrial ;
4  if hard_type == 0
5  % Linear
6  q_inf = q_fact * r0;
7  if q_fact>=1 % Hardening
8  if q_n>=q_inf
9  q_n1= q_inf;
10 else

```

```

11 q_n1= q_n+ H*delta_r;
12 if q_n1>=q_inf
13 q_n1 = q_inf;
14 end
15 end
16 else % Softening
17 if q_n<=q_inf
18 q_n1= q_inf;
19 else
20 q_n1= q_n+ H*delta_r;
21 if q_n1<=q_inf
22 q_n1 = q_inf;
23 end
24 end
25 end
26 hvar_n1(7) = H;%Exporting hardening variable

```

3.2 EXPONENTIAL HARDENING/SOFTENING

In this case, the equation is:

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

The hardening modulus is computed as:

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

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 :

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

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

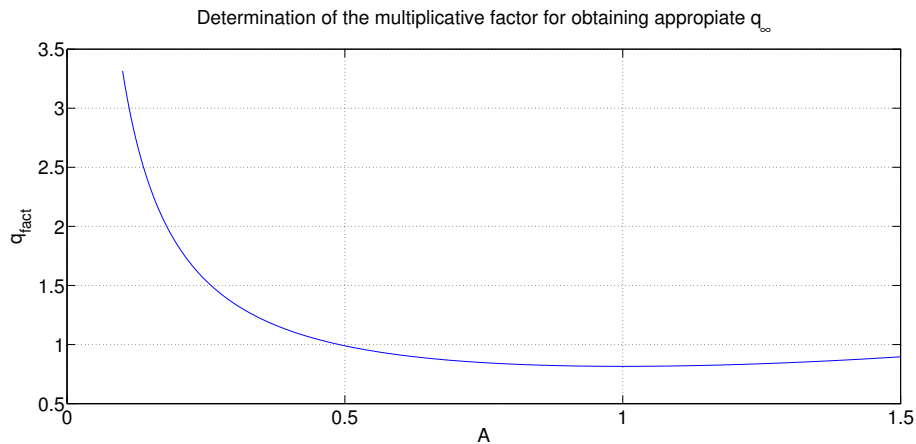


Figure 3.1: q_{fact} vs A .

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} :

```

1 q_inf = q_fact * r0;
2 q_n1 = q_inf - ((q_inf - r0) * exp(A * (1 - (rtrial / r0))));
3 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

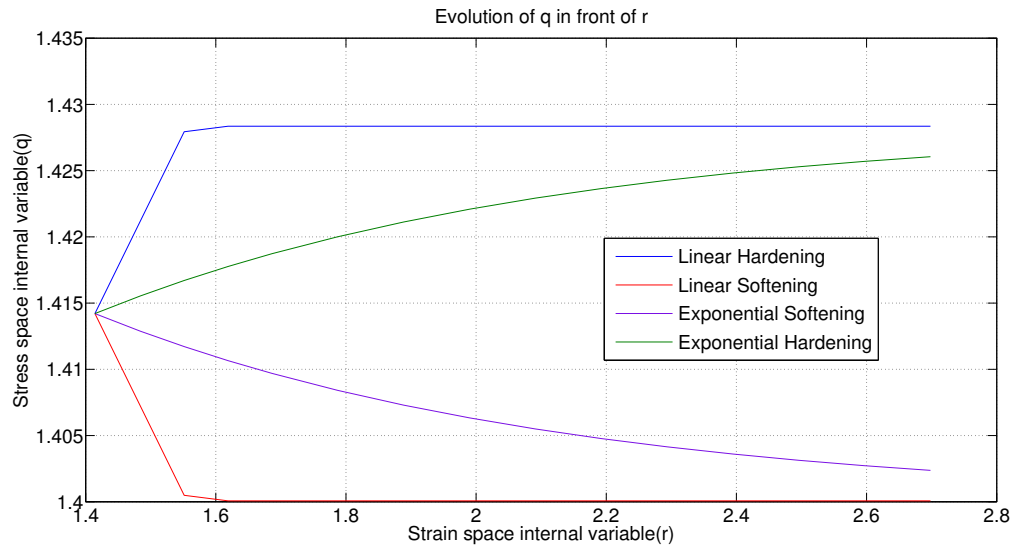


Figure 3.2: Hardening and Softening schemes.

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

4.1 CASE 1

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

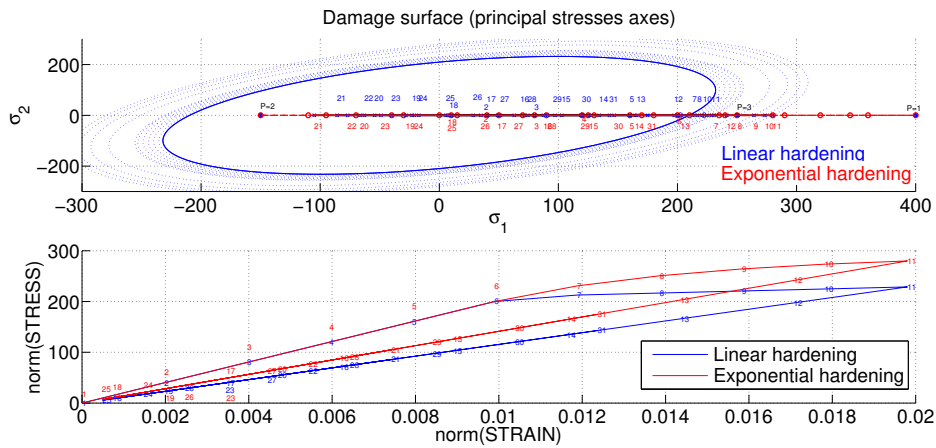


Figure 4.1: Damage surface and stress-strain plots for Case 1.

Note that the material is loaded for the first 11 steps, then is unloaded until 18th step, it keeps compressing until 21st 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

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

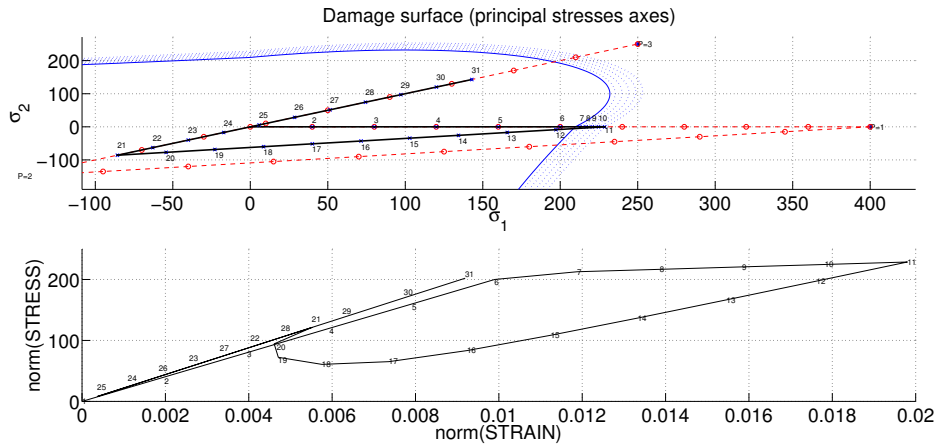


Figure 4.2: Damage surface and stress-strain plots for Case 2.

Note that the first eleven steps 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

$$\left\{ \begin{array}{l} \Delta\sigma_1 = \alpha \quad , \quad \Delta\sigma_2 = \alpha \text{ (biaxial tensile loading)} \\ \Delta\sigma_1 = -\beta \quad , \quad \Delta\sigma_2 = -\beta \text{ (biaxial tensile unloading/compressive loading)} \\ \Delta\sigma_1 = \gamma \quad , \quad \Delta\sigma_2 = \gamma \text{ (biaxial compressive unloading/tensile loading)} \end{array} \right. \quad (4.3)$$

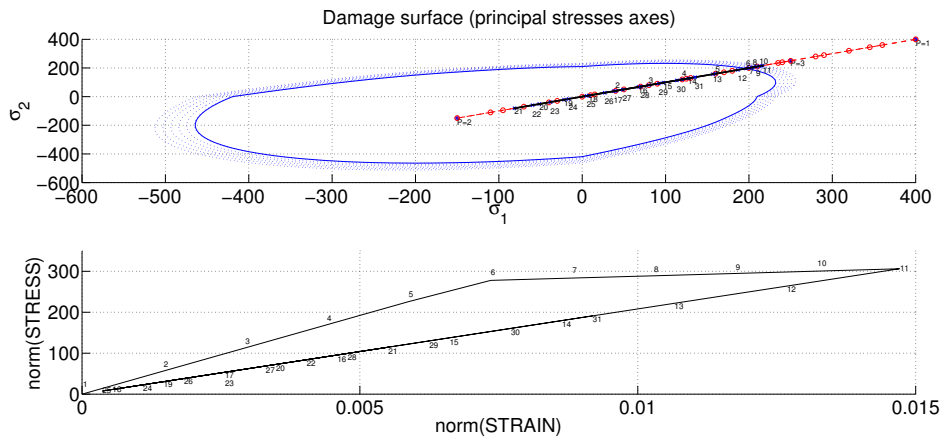


Figure 4.3: Damage surface and stress-strain plots for Case 3.

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:

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

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:

```

1 [ rtrial0 ] = Modelos_de_dano1 (MDtype, ce , eps_n0 , n) ;
2 [ rtrial1 ] = Modelos_de_dano1 (MDtype, ce , eps_n1 , n) ;
3 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} :

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

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

```

1 r_n1= ((eta-delta_t*(1-alpha))/(eta+alpha*delta_t))*r_n+...
2 (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:

```
1 if viscpr == 1
2 % Does computation for viscid model %
3 % ... Algorithm ... + call to rmap_danovisc.m
4 else
5 % Does computation for inviscid model %
6 % ... Algorithm ...
7 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:

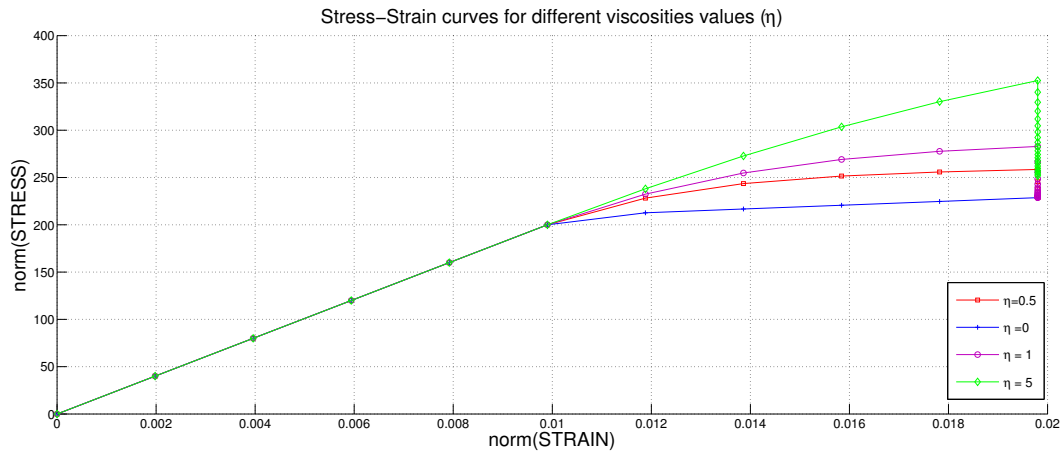


Figure 6.1: Stress-strain against viscosity.

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:

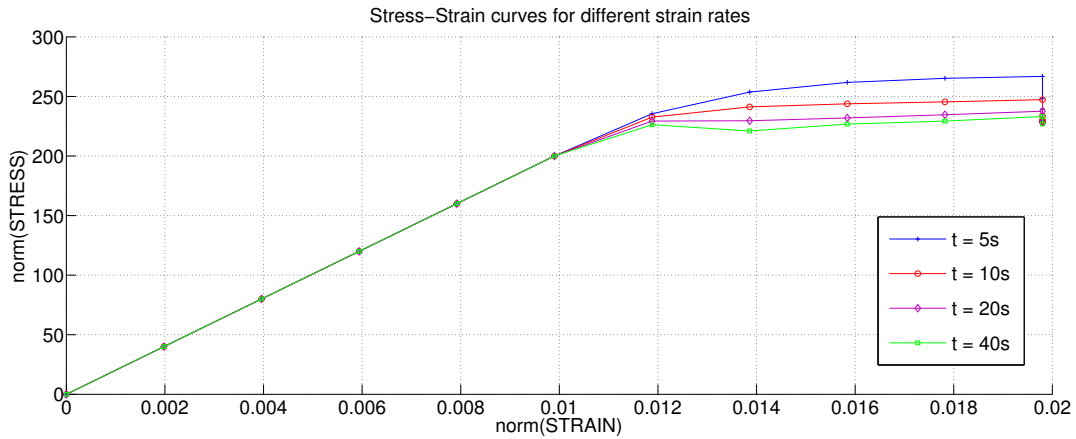


Figure 6.2: Stress-strain against time (proportionally inverse to stress-rate).

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:

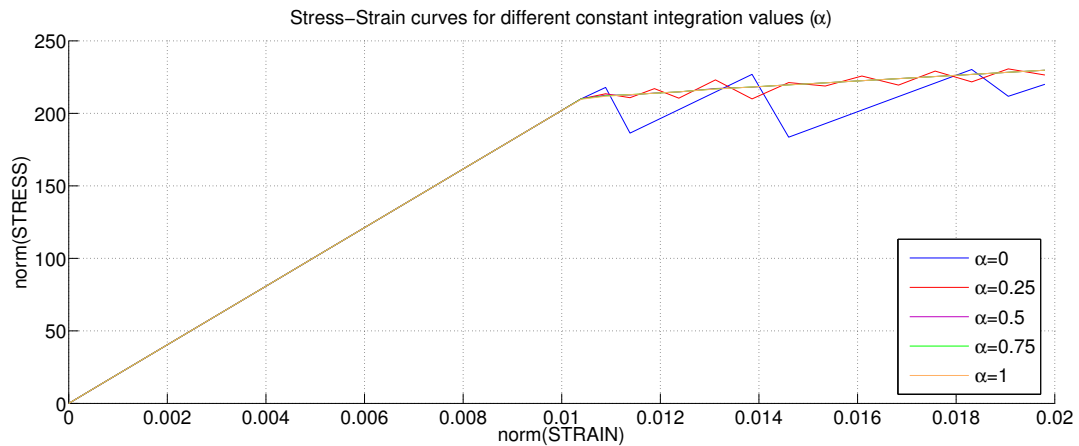


Figure 6.3: Stress-strain against α .

Note that:

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

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.

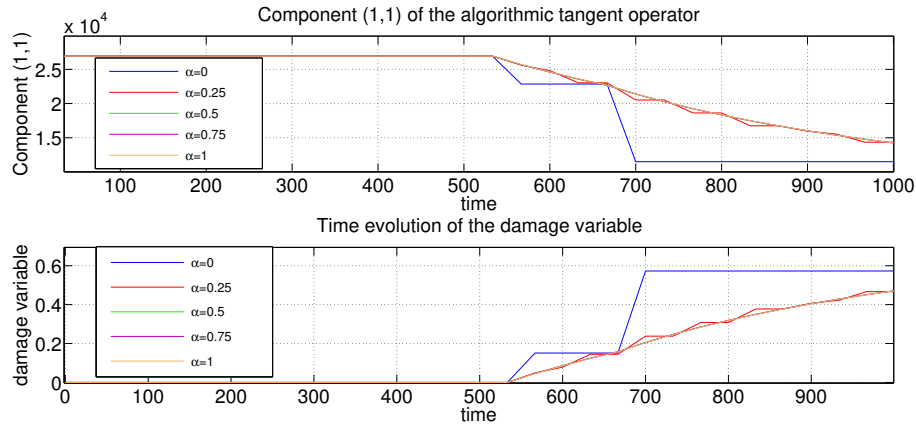


Figure 6.4: Stress-strain against α .

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

```
1 Annex 1: dibujar_criterio_dano1.m
2 function hplot = dibujar_criterio_dano1(ce,nu,q, tipo_linea ,MDtype,n)
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 if MDtype==1
14 tetha=[0:0.01:2*pi];
15 %*****
16 %* RADIUS
17 D=size(tetha); %* Range
18 m1=cos(tetha); %*
19 m2=sin(tetha); %*
20 Contador=D(1,2); %*
21 radio = zeros(1,Contador) ;
22 s1 = zeros(1,Contador) ;
23 s2 = zeros(1,Contador) ;
24 for i=1:Contador
25 radio(i)= q/sqrt([m1(i) m2(i) 0 nu*(m1(i)+m2(i))]*ce_inv*...
26 [m1(i) m2(i) 0 nu*(m1(i)+m2(i))]' );
27 s1(i)=radio(i)*m1(i);
28 s2(i)=radio(i)*m2(i);
29 end
30 hplot =plot(s1,s2,tipo_linea);
31 elseif MDtype==2
32 % Comment/delete lines below once you have implemented this case
33 % *****
34 tetha=[0:0.01:2*pi];
35 %*****
36 %* RADIUS
37 % Slide 18 lecture 4
38 D=size(tetha);
```



```
39 m1 = cos(tetha);
40 m2 = sin(tetha);
41 m3 = nu * (m1 + m2);
42 Contador=D(1,2);
43 %Macaulin bracket
44 m1_2(m1_2<0)=0;
45 m2_2(m2_2<0)=0;
46 m3_2(m3_2<0)=0;
47 radio = zeros(1,Contador) ;
48 s1 = zeros(1,Contador) ;
49 s2 = zeros(1,Contador) ;
50 for i=1:Contador
51 radio(i)= q/sqrt([m1_2(i) m2_2(i) 0 m3_2(i)]*ce_inv* ...
52 [m1(i) m2(i) 0 m3(i)]');
53 s1(i)=radio(i)*m1(i);
54 s2(i)=radio(i)*m2(i);
55 end
56 hplot =plot(s1,s2,tipolinea);
57 elseif MDtype==3
58 % Comment/delete lines below once you have implemented this case
59 % *****
60 % Slide 19 lecture 4
61 tetha=[0:0.01:2*pi];
62 D=size(tetha);
63 m1 = cos(tetha);
64 m2 = sin(tetha);
65 m3 = nu *(m1 + m2);
66 Contador=D(1,2);
67 % Macaulin bracket
68 m1_2(m1_2<0)=0;
69 m2_2(m2_2<0)=0;
70 m3_2(m3_2<0)=0;
71 radio = zeros(1,Contador) ;
72 s1 = zeros(1,Contador) ;
73 s2 = zeros(1,Contador) ;
74 for i=1:Contador
75 TETHA = (m1_2(i) + m2_2(i) + m3_2(i))/(abs(m1(i))+abs(m2(i))+ ...
76 abs(m3(i)));
77 Const = TETHA + (1 - TETHA)/n;
78 radio(i)= q/(Const*sqrt([m1(i) m2(i) 0 nu*(m1(i)+m2(i))] * ...
79 ce_inv*[m1(i) m2(i) 0 nu*(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 %*****
86 %*****
87 return
```

APPENDIX 2: RMAP_DANO1.M

```
1 function [sigma_n1 , hvar_n1 , aux_var] = rmap_dano1 (eps_n1 , hvar_n , ...
2 Eprop , ce , MDtype , n)
3 hvar_n1 = hvar_n ;
4 r_n = hvar_n (5) ;
5 q_n = hvar_n (6) ;
6 E = Eprop (1) ;
7 nu = Eprop (2) ;
8 H = Eprop (3) ;
9 sigma_u = Eprop (4) ;
10 hard_type = Eprop (5) ;
11 q_fact = Eprop (9) ;
12 A = Eprop (10) ;
13 %*****
14 %*****
15 %* initializing
16 r0 = sigma_u / sqrt (E) ;
17 zero_q = 1.d - 6 * r0 ;
18 % if (r_n <= 0.d0)
19 % r_n = r0 ;
20 % q_n = r0 ;
21 % end
22 %*****
23 %*****
24 %* Damage surface
25 [rtrial] = Modelos_de_dano1 (MDtype , ce , eps_n1 , n) ;
26 hvar_n1 (8) = rtrial ;
27 %*****
28 %*****
29 %* Ver el Estado de Carga
30 %* -----> fload=0 : elastic unload
31 %* -----> fload=1 : damage (compute algorithmic
32 %constitutive tensor)
33 fload=0 ;
```

```
34 if(rtrial > r_n)
35 %* Loading
36 fload=1;
37 delta_r=rtrial-r_n;
38 r_n1= rtrial ;
39 if hard_type == 0
40 % Linear
41 q_inf = q_fact * r0;
42 if q_fact>=1 % Hardening
43 if q_n>=q_inf
44 q_n1= q_inf;
45 else
46 q_n1= q_n+ H*delta_r;
47 if q_n1>=q_inf
48 q_n1 = q_inf;
49 end
50 end
51 else % Softening
52 if q_n<=q_inf
53 q_n1= q_inf;
54 else
55 q_n1= q_n+ H*delta_r;
56 if q_n1<=q_inf
57 q_n1 = q_inf;
58 end
59 end
60 end
61 hvar_n1(7) = H;
62 else
63 % We calculate q_inf from a factor of the intial value of q,
64 %which is r0.
65 % Then q_n1 of the timestep n+1 is calculated as:
66 q_inf = q_fact * r0;
67 q_n1 = q_inf - ((q_inf - r0) * exp(A * (1 - (rtrial / r0))));
68 hvar_n1(7) = (A*(q_inf-r0)/r0) * exp(A * (1 - (rtrial / r0)));
69 end
70 if(q_n1<zero_q)
71 q_n1=zero_q;
72 end
73 else
74 %* Elastic load/unload
75 fload=0;
76 r_n1= r_n ;
77 q_n1= q_n ;
```

```

78 end
79 % Damage variable
80 % -----
81 dano_n1 = 1.d0-(q_n1/r_n1);
82 % Computing stress
83 % *****
84 sigma_n1 = (1.d0-dano_n1)*ce*eps_n1';
85 %hold on
86 %plot(sigma_n1(1),sigma_n1(2),'bx')
87 %*****
88 %* Updating historic variables
89 % hvar_n1(1:4) = eps_nlp;
90 hvar_n1(5)= r_n1 ;
91 hvar_n1(6)= q_n1 ;
92 %*****
93 %*****
94 %* Auxiliar variables
95 aux_var(1) = fload;
96 aux_var(2) = q_n1/r_n1;
97 %*aux_var(3) = (q_n1-H*r_n1)/r_n1^3;
98 %*****

```

APPENDIX 3: Q_FACT_VALUE.M

```

1 clc;clear all;
2 % Young modulus
3 YOUNG = 20000;
4 % Yield strenght
5 sigma_u = 200;
6 %r0
7 r0=sigma_u;
8 %Select the maximum value of r
9 mult = 2;r_max = mult*r0;
10 %Select an average constant value for the hardening modulus. This
    constant
11 %value can be approximated by the value used for linear hardening.
    H=0.3;
12 % Vector of different A (constant) values.
13 A = linspace(0.1,1.5,1000);
14 % Calculation of q_fact, which is the value that multiplied by r0
15 % give us q_inf: q_inf=q_fact*r0.
16

```

```

17 q_fact = (H ./ A) .* exp(-A * (1 - (r_max / r0)));
18 q_fact2 =(H ./ A) .* exp(A);
19 %plot
20 plot(A, q_fact)
21 grid on
22 title('Determination of the multiplicative factor for obtaining
23     appropriate q_{\infty}')
24 xlabel('A')
25 ylabel('q_{fact}')

```

APPENDIX 4: RMAP_DANOVISC.M

```

1 function [sigma_n1, hvar_n1, aux_var] = rmap_danovisc (delta_t, eps_n0, ...
2     eps_n1, hvar_n, Eprop, ce, MDtype, n)
3 hvar_n1 = hvar_n;
4 r_n      = hvar_n(5);
5 q_n      = hvar_n(6);
6 E        = Eprop(1);
7 nu       = Eprop(2);
8 H        = Eprop(3);
9 sigma_u  = Eprop(4);
10 hard_type = Eprop(5);
11 alpha = Eprop(8);
12 eta = Eprop(7);
13 q_fact = Eprop(9);
14 A = Eprop(10);
15 %*****
16 %*****
17 %*      initializing                                     %*
18   r0 = sigma_u/sqrt(E);
19   zero_q=1.d-6*r0;
20 % if (r_n<=0.d0)
21 %   r_n=r0;
22 %   q_n=r0;
23 % end
24 %*****
25 %*****
26 %*      Damage surface                                   %*
27 %Calculate rtrial (tau_eps) for the time step n and n+1. Then calculate
28 %rtrial which is (tau_eps_n+alpha) as rtrial0*(1-alpha)+alpha*rtrial1
29 [rtrial0] = Modelos_de_dano1 (MDtype, ce, eps_n0, n);

```

```

30 [rtrial1] = Modelos_de_dano1 (MDtype, ce, eps_n1, n);
31 rtrial= rtrial0*(1-alpha)+alpha*rtrial1;
32 hvar_n1(8)= rtrial; % Exporting rtrial_n
33 hvar_n1(9)= rtrial1; % Exporting rtrial_n+1
34 %Both needed to calculate Calg
35 %*****
36 %*****
37 %*   Ver el Estado de Carga
38 %*   ----->   fload=0 : elastic unload
39 %*   ----->   fload=1 : damage (compute algorithmic constitutive
40 % tensor)      %*
41 fload=0;
42 if(rtrial > r_n)
43     %*   Loading
44     fload=1;
45     % As it is viscous, in the loading case we calculate rn+1 as:
46     r_n1= ((eta-delta_t*(1-alpha))/(eta+alpha*delta_t))*r_n+ ...
47           (delta_t/(eta+alpha*delta_t))*rtrial;
48     delta_r=r_n1-r_n;
49     if hard_type == 0
50         % Linear
51         q_inf = q_fact * r0;
52         if q_fact>=1 % Hardening
53             if q_n>=q_inf
54                 q_n1= q_inf;
55             else
56                 q_n1= q_n+ H*delta_r;
57                 if q_n1>=q_inf
58                     q_n1 = q_inf;
59                 end
60             end
61         else % Softening
62             if q_n<=q_inf
63                 q_n1= q_inf;
64             else
65                 q_n1= q_n+ H*delta_r;
66                 if q_n1<=q_inf
67                     q_n1 = q_inf;
68                 end
69             end
70         end
71         hvar_n1(7) = H;
72     else
73 % We calculate q_inf from a factor of the intial value of q,

```

```

74 % which is r0.
75 % Then q_n1 of the timestep n+1 is calculated as:
76     q_inf = q_fact * r0;
77     q_n1 = q_inf - ((q_inf - r0) * exp(A * (1 - (r_n1 / r0))));
78     hvar_n1(7) = (A*(q_inf-r0)/r0) * exp(A * (1 - (r_n1 / r0)));
79     end
80     if(q_n1<zero_q)
81         q_n1=zero_q;
82     end
83 else
84     %* Elastic load/unload
85     fload=0;
86     r_n1= r_n ;
87     q_n1= q_n ;
88 end
89 % Damage variable
90 % -----
91 dano_n1 = 1.d0-(q_n1/r_n1);
92 % Computing stress
93 % *****
94 sigma_n1 =(1.d0-dano_n1)*ce*eps_n1';
95 %hold on
96 %plot(sigma_n1(1),sigma_n1(2),'bx')
97 %*****
98 %*****
99 %* Updating historic variables
100 %hvar_n1(1:4) = eps_nlp;
101 hvar_n1(5)= r_n1 ;
102 hvar_n1(6)= q_n1 ;
103 %*****
104 %*****
105 %* Auxiliar variables
106 aux_var(1) = fload;
107 aux_var(2) = q_n1/r_n1;
108 %aux_var(3) = (q_n1-H*r_n1)/r_n1^3;
109 %*****

```

APPENDIX 5: DAMAGE_MAIN.M

```

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

```

```
3 global hplotSURF
4 % SET LABEL OF "vartoplot" variables (it may be defined also outside
5 % this function)
6 % -----
7 LABELPLOT = {'hardening variable (q)', 'internal variable'};
8 E = Eprop(1) ; nu = Eprop(2) ;
9 viscpr = Eprop(6) ;
10 sigma_u = Eprop(4);
11 if ntype == 1
12     menu('PLANE STRESS has not been implemented yet', 'STOP');
13     error('OPTION NOT AVAILABLE')
14 elseif ntype == 3
15     menu('3-DIMENSIONAL PROBLEM has not been implemented yet', 'STOP');
16     error('OPTION NOT AVAILABLE')
17 else
18     mstrain = 4 ;
19     mhist = 8 ;
20 end
21 if viscpr == 1
22     % Here we solve for the viscous case, which is calling the function
23     % rmap_danovisc
24     totalstep = sum(istep) ;
25     % INITIALIZING GLOBAL CELL ARRAYS
26     % -----
27     sigma_v = cell(totalstep+1,1) ;
28     TIMEVECTOR = zeros(totalstep+1,1) ;
29     delta_t = TimeTotal./ istep/length(istep) ;
30     % Elastic constitutive tensor
31     % -----
32     [ce] = tensor_elastico1 (Eprop, ntype);
33     Calg1=zeros(totalstep+1,2);
34     Ctan1=zeros(totalstep+1,2);
35     % Initz.
36     % -----
37     % Strain vector for each timestep
38     % -----
39     eps_n0= zeros(mstrain,1);
40     eps_n1 = zeros(mstrain,1);
41     % Historic variables
42     % hvar_n(1:4) --> empty
43     % hvar_n(5) = q --> Hardening variable
44     % hvar_n(6) = r --> Internal variable
45     hvar_n = zeros(mhist,1) ;
46     % INITIALIZING (i = 1) !!!!
```



```
47 % ***** i*
48 i = 1 ;
49 r0 = sigma_u/sqrt(E) ;
50 hvar_n(5) = r0; % r_n
51 hvar_n(6) = r0; % q_n
52 eps_n1 = strain(i,:) ;
53 sigma_n1 =ce*eps_n1'; % Elastic
54 sigma_v{i} = [sigma_n1(1) sigma_n1(3) 0;sigma_n1(3) sigma_n1(2) 0 ;
55 ...
56 0 0 sigma_n1(4)];
57 nplot = 3 ;
58 vartoplot = cell(1,totalstep+1) ;
59 vartoplot{i}(1) = hvar_n(6) ; % Hardening variable (q)
60 vartoplot{i}(2) = hvar_n(5) ; % Internal variable (r)
61 vartoplot{i}(3) = 1-hvar_n(6)/hvar_n(5) ; % Damage variable (d)
62 for iload = 1:length(istep)
63 % Load states
64 for iloc = 1:istep(iload)
65 i = i + 1 ;
66 TIMEVECTOR(i) = TIMEVECTOR(i-1)+ delta_t(iload) ;
67 % Total strain at step "i"
68 % -----
69 eps_n0 = strain(i-1,:);
70 eps_n1 = strain(i,:) ;
71 %
72 %*****
73 %* DAMAGE MODEL
74 %
75 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
76 [sigma_n1 , hvar_n , aux_var] = rmap_danovisc(delta_t , eps_n0 ,
77 eps_n1 , ...
78 hvar_n , Eprop , ce , MDtype , n) ;
79 % PLOTTING DAMAGE SURFACE
80 if (aux_var(1)>0)
81 hplotSURF(i) = dibujar_criterio_dano1(ce , nu , hvar_n(6) ,
82 ...
83 'r' , MDtype , n ) ;
84 set(hplotSURF(i) , 'Color' , [0 0 1] , 'LineWidth' , 1)
85 ;
86 end
87 %
88 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

82      %
      *****
83      % GLOBAL VARIABLES
84      % *****
85      % Stress
86      % -----
87      m_sigma=[sigma_n1(1)  sigma_n1(3)  0;sigma_n1(3)  sigma_n1(2)
88                0 ; ...
89      0 0  sigma_n1(4)];
90      sigma_v{i} = m_sigma ;
91      % VARIABLES TO PLOT (set label on cell array LABELPLOT)
92      % -----
93      vartoplot{i}(1) = hvar_n(6) ; % Hardening variable (q)
94      vartoplot{i}(2) = hvar_n(5) ; % Internal variable (r)
95      vartoplot{i}(3) = 1-hvar_n(6)/hvar_n(5) ; % Damage
96      variable (d)
97      H_n1 = hvar_n(7);
98      q_n1 = vartoplot{i}(1);
99      r_n = vartoplot{i-1}(2);
100     r_n1 = vartoplot{i}(2);
101     d_n1 = vartoplot{i}(3);
102     sigma_bar = [sigma_n1(1)  sigma_n1(2)  sigma_n1(4)  2*sigma_n1
103                 (3)];
104     alpha = Eprop(8);
105     eta = Eprop(7);
106     rtrial = hvar_n(8);
107     rtrial1 = hvar_n(9);
108     % Calculation of the Algorithmic tangent operator and the
109     % tangent
110     % operator.
111     if rtrial>r_n1 %Loading case
112         d_diff = (q_n1-(H_n1*r_n1))/((r_n1)^2);
113         C1 = alpha*delta_t / (eta + (alpha*delta_t));
114         C2 = 1/rtrial1;
115         Calg=(1-d_n1)*ce-(C1*C2*d_diff*kron(sigma_bar',sigma_bar
116         ));
117         %Ctan = (1-d_n1)*ce - (((q_n1 - H_n1*r_n1)/r_n1^3)* ...
118         %kron(sigma_bar',sigma_bar));
119         Calg11(i,1)=TIMEVECTOR(i);
120         Calg11(i,2)=Calg(1,1);
121         %Ctan11(i,1) = TIMEVECTOR(i);
122         %Ctan11(i,2) = Ctan(1,1);

```

```

118         else %elastic/unloading
119             Calg = (1-d_n1)*ce;
120             %Ctan=Calg;
121             Calg11(i,1)=TIMEVECTOR(i);
122             Calg11(i,2)=Calg(1,1);
123             %Ctan11(i,1) = TIMEVECTOR(i);
124             %Ctan11(i,2) = Ctan(1,1);
125         end
126
127     end
128 end
129 else
130 % Here we solve for the inviscid case, which is calling the function
131 % rmap_danol
132 totalstep = sum(istep) ;
133 % INITIALIZING GLOBAL CELL ARRAYS
134 % -----
135 sigma_v = cell(totalstep+1,1) ;
136 TIMEVECTOR = zeros(totalstep+1,1) ;
137 delta_t = TimeTotal./ istep/length(istep) ;
138 Calg11=zeros(totalstep+1,2);
139 Ctan11=zeros(totalstep+1,2);
140 % Elastic constitutive tensor
141 % -----
142 [ce] = tensor_elastico1 (Eprop, ntype);
143 % Initz.
144 % -----
145 % Strain vector
146 % -----
147 eps_n1 = zeros(mstrain,1);
148 % Historic variables
149 % hvar_n(1:4) --> empty
150 % hvar_n(5) = q --> Hardening variable
151 % hvar_n(6) = r --> Internal variable
152 hvar_n = zeros(mhist,1) ;
153 % INITIALIZING (i = 1) !!!!
154 % ***** i*
155 i = 1 ;
156 r0 = sigma_u/sqrt(E);
157 hvar_n(5) = r0; % r_n
158 hvar_n(6) = r0; % q_n
159 eps_n1 = strain(i,:) ;
160 sigma_n1 =ce*eps_n1'; % Elastic

```

```
161     sigma_v{i} = [sigma_n1(1)  sigma_n1(3)  0;sigma_n1(3)  sigma_n1(2)  0 ;
162         ...
163             0  0  sigma_n1(4) ];
164     nplot = 3 ;
165     vartoplot = cell(1,totalstep+1) ;
166     vartoplot{i}(1) = hvar_n(6) ; % Hardening variable (q)
167     vartoplot{i}(2) = hvar_n(5) ; % Internal variable (r)
168     vartoplot{i}(3) = 1-hvar_n(6)/hvar_n(5) ; % Damage variable (d)
169     for iload = 1:length(istep)
170         % Load states
171         for iloc = 1:istep(iload)
172             i = i + 1 ;
173             TIMEVECTOR(i) = TIMEVECTOR(i-1)+ delta_t(iload) ;
174             % Total strain at step "i"
175             % -----
176             eps_n1 = strain(i,:) ;
177             %
178             % *****
179             %*      DAMAGE MODEL
180             %
181             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
182             [sigma_n1 , hvar_n , aux_var] = rmap_dano1(eps_n1 , hvar_n , Eprop
183                 , ...
184                 ce , MDtype , n) ;
185             % PLOTTING DAMAGE SURFACE
186             if (aux_var(1)>0)
187                 hplotSURF(i) = dibujar_criterio_dano1(ce , nu , hvar_n(6)
188                     , ...
189                     'r:' , MDtype , n ) ;
190                 set(hplotSURF(i) , 'Color' , [0 0 1] , 'LineWidth' , 1)
191                 ;
192             end
193             %
194             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
195             %
196             % *****
197             % GLOBAL VARIABLES
198             % *****
199             % Stress
200             % -----
```

```
193     m_sigma=[sigma_n1(1)  sigma_n1(3)  0;sigma_n1(3)  sigma_n1(2)
           0 ; ...
194             0 0  sigma_n1(4) ];
195     sigma_v{i} = m_sigma ;
196     % VARIABLES TO PLOT (set label on cell array LABELPLOT)
197     % -----
198     vartoplot{i}(1) = hvar_n(6) ; % Hardening variable (q)
199     vartoplot{i}(2) = hvar_n(5) ; % Internal variable (r)
200     vartoplot{i}(3) = 1-hvar_n(6)/hvar_n(5) ; % Damage
           variable (d)
201     H_n1 = hvar_n(7);
202     q_n1 = vartoplot{i}(1);
203     r_n = vartoplot{i-1}(2);
204     r_n1 = vartoplot{i}(2);
205     d_n1 = vartoplot{i}(3);
206     sigma_bar = [sigma_n1(1)  sigma_n1(2)  sigma_n1(4)  2*sigma_n1
           (3) ];
207     alpha = Eprop(8);
208     eta = Eprop(7);
209     rtrial = hvar_n(8);
210     % Calculation of the Algorithmic tangent operator.
211     if rtrial>r_n1 %Loading case
212         Ctan = (1-d_n1)*ce - (((q_n1 - H_n1*r_n1)/r_n1^3)* ...
213             (kron(sigma_bar', sigma_bar)));
214         Ctan11(i,1) = TIMEVECTOR(i);
215         Ctan11(i,2) = Ctan(1,1);
216     else %elastic/unloading
217         Ctan = (1-d_n1)*ce;
218         Ctan11(i,1) = TIMEVECTOR(i);
219         Ctan11(i,2) = Ctan(1,1);
220     end
221 end
222 end
223 end
```

APPENDIX 6: MODELOS_DE_DANO1.M

```

1 function [rtrial] = Modelos_de_dano1 (MDtype,ce,eps_n1,n)
2 %*****
3 if (MDtype==1) %* Symmetric
4 rtrial= sqrt(eps_n1*ce*eps_n1') ;
5 elseif (MDtype==2) %* Only tension
6 sigma = eps_n1*ce;
7 macsigma=sigma;
8 macsigma(macsigma<0)=0;
9 rtrial = sqrt(macsigma*eps_n1') ;
10 elseif (MDtype==3) %*Non-symmetric
11 sigma = eps_n1*ce;
12 macsigma = sigma;
13 macsigma(macsigma<0)=0;
14 theta = (macsigma(1) + macsigma(2) + macsigma(3)) / ...
15 (abs(sigma(1)) + abs(sigma(2)) + abs(sigma(3)));
16 rtrial =(theta + ((1-theta)/n)) * sqrt(eps_n1*ce*eps_n1') ;
17 end
18 %*****
19 return

```

APPENDIX 7: MAIN_NOINTERACTIVE.M

```

1 all
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3 % Program for modelling damage model
4 % (Elemental gauss point level)
5 % -----
6 % Developed by J.A. Hdez Ortega
7 % 20-May-2007, Universidad Polit cnica de Catalu a
8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9 %profile on
10 % -----
11 % *****
12 % INPUTS
13 % *****
14 % YOUNG' s MODULUS
15 % -----
16 YOUNG_M = 20000 ;
17 % Poisson' s coefficient

```

```
18 % -----
19 POISSON = 0.3 ;
20 % Hardening/softening modulus
21 % -----
22 HARDSOFT_MOD = 0.1 ;%0.1
23 % Yield stress
24 % -----
25 YIELD_STRESS = 200 ;
26 % Hardening/softening internal variable stress space factor (q_inf =
    q_fact * r0).
27 %
    -----
28 q_fact = 1.01;%1.4
29 % Constant of the exponential law for hardening/softening. (A>0)
30 % -----
31 A = 2;
32 % Problem type TP = {'PLANE STRESS', 'PLANE STRAIN', '3D'}
33 % ----- = 1 =2 =3
34 % -----
35 ntype= 2;
36 % Model PTC = {'SYMMETRIC', 'TENSION', 'NON-SYMMETRIC'} ;
37 % = 1 = 2 = 3
38 % -----
39 MDtype =3;
40 % Ratio compression strength / tension strength
41 % -----
42 n = 3 ;%3
43 % SOFTENING/HARDENING TYPE
44 % -----
45 HARDTYPE = 'EXPONENTIAL' ; %{LINEAR, EXPONENTIAL}
46 % VISCOUS/INVISCID
47 % -----
48 VISCOUS = 'YES' ;
49 % Viscous coefficient -----
50 % -----
51 eta = 0;%0.3
52 % TimeTotal (initial = 0) -----
53 % -----
54 TimeTotal = 1000 ; %10
55 % Integration coefficient ALPHA
56 % -----
57 ALPHA_COEFF = 1 ;%0.5
58 % Points -----
```

```

59 % -----
60 nloadstates = 3 ;
61 SIGMAP = zeros(nloadstates,2) ;
62 SIGMAP(1,:) =[150 0];
63 SIGMAP(2,:) =[250 0];
64 SIGMAP(3,:) =[400 0];
65 % Number of time increments for each load state
66 % -----
67 istep = 10*ones(1,nloadstates) ;
68 % VARIABLES TO PLOT
69 vpx = 'norm(STRAIN)'; % AVAILABLE OPTIONS: 'STRAIN_1', 'STRAIN_2'
70 % '|STRAIN_1|', '|STRAIN_2|'
71 % 'norm(STRAIN)', 'TIME'
72 vpy = 'norm(STRESS)'; % AVAILABLE OPTIONS: 'STRESS_1', 'STRESS_2'
73 % '|STRESS_1|', '|STRESS_2|'
74 % 'norm(STRESS)', 'TIME', 'DAMAGE VAR.', 'hardening variable (q)', 'damage
    variable (d)'
75 % 'internal variable (r)'
76 % 3) LABELPLOT{ivar} --> Cell array with the label string for
77 % variables of "varplot"
78 %
79 LABELPLOT = {'hardening variable (q)', 'internal variable (r)', 'damage
    variable (d)'};
80 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END INPUTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
81 %% Plot Initial Damage Surface and effective stress path
82 strain_history = PlotIniSurf(YOUNG_M,POISSON,YIELD_STRESS,...
83 SIGMAP, ntype,MDtype,n, istep);
84 E = YOUNG_M ;
85 nu = POISSON ;
86 sigma_u = YIELD_STRESS ;
87 switch HARDTYPE
88 case 'LINEAR'
89 hard_type = 0 ;
90 otherwise
91 hard_type = 1 ;
92 end
93 switch VISCOUS
94 case 'YES'
95 viscpr = 1 ;
96 otherwise
97 viscpr = 0 ;
98 end
99 Eprop = [E nu HARDSOFT_MOD sigma_u hard_type viscpr eta ...
100 ALPHA_COEFF q_fact A] ;

```



```
101 % DAMAGE MODEL
102 % -----
103 [sigma_v, vartoplot, LABELPLOT_out, TIMEVECTOR, Calg11, Ctan11]= ...
104 damage_main(Eprop, ntype, istep, strain_history, MDtype, n, TimeTotal);
105 try; LABELPLOT; catch; LABELPLOT = LABELPLOT_out; end;
106 % PLOTTING
107 % -----
108 ncolores = 3;
109 colores = ColoresMatrix(ncolores);
110 markers = MarkerMatrix(ncolores);
111 hplotLLL = [];
112 for i = 2:length(sigma_v)
113 stress_eig = sigma_v{i}; %eigs(sigma_v{i});
114 tstress_eig = sigma_v{i-1}; %eigs(sigma_v{i-1});
115 hplotLLL(end+1) = plot([tstress_eig(1,1) stress_eig(1,1)], [tstress_eig
    (2,2) stress_eig(2,2)], 'LineWidth', 2, 'color', colores(1,:), 'Marker', markers{1},
    'MarkerSize', 2);
116 plot(stress_eig(1,1), stress_eig(2,2), 'bx')
117 text(stress_eig(1,1), stress_eig(2,2), num2str(i))
118 % SURFACES
119 % -----
120 end
121 % % SURFACES
122 % % -----
123 % if (aux_var(1) > 0)
124 % hplotSURF(i) = dibujar_criterio_dano1(ce, nu, hvar_n(6), 'r:', MDtype, n
    );
125 % set(hplotSURF(i), 'Color', [0 0 1], 'LineWidth', 1);
126 % end
127 DATA.sigma_v = sigma_v;
128 DATA.vartoplot = vartoplot;
129 DATA.LABELPLOT = LABELPLOT;
130 DATA.TIMEVECTOR = TIMEVECTOR;
131 DATA.strain = strain_history;
132 plotcurvesNEW(DATA, vpx, vpy, LABELPLOT, vartoplot);
133 d=zeros(length(TIMEVECTOR), 1);
134 for i=1:length(TIMEVECTOR)
135 d(i) = vartoplot{i}(3);
136 end
137 figure(2)
138 subplot(2, 1, 1)
139 plot(Calg11(:, 1), Calg11(:, 2), 'b-'), hold on
140 xlabel('time')
141 ylabel('Component (1,1)')
```

```
142 title('Component (1,1) of the algorithmic tangent operator')
143 grid on
144 subplot(2,1,2)
145 plot(TIMEVECTOR,d), hold on;
146 xlabel('time')
147 ylabel('damage variable')
148 title('Time evolution of the damage variable')
149 grid on
150 q = zeros(length(TIMEVECTOR),1);
151 r = zeros(length(TIMEVECTOR),1);
152 for i=1:length(TIMEVECTOR)
153 q(i) = vartoplot{i}(1);
154 r(i) = vartoplot{i}(2);
155 end
156 figure(3)
157 plot(r,q),hold on
158 xlabel('Strain space internal variable(r)')
159 ylabel('Stress space internal variable(q)')
160 title('Evolution of q in front of r')
161 grid on
```