Computational Solid Mechanics

Erasmus Mundas Masters in Computational Mechanics

Assignment:

Rate Independent and Dependent Model

BY

Md Tariqul Islam

Part 1: Rate Independent Model

Three types of Model are analysed in Rate independent case (namely: Symmetric model, Tension only model and Non-symmetric model). The algorithm for the symmetric model is already implemented in the given Matlab Code. The algorithm for the tension only model and non-symmetric model are now implemented (Matlab code can be found in the appendix) and the damage surface for both the models are retrieved and can be seen in the following diagrams below:

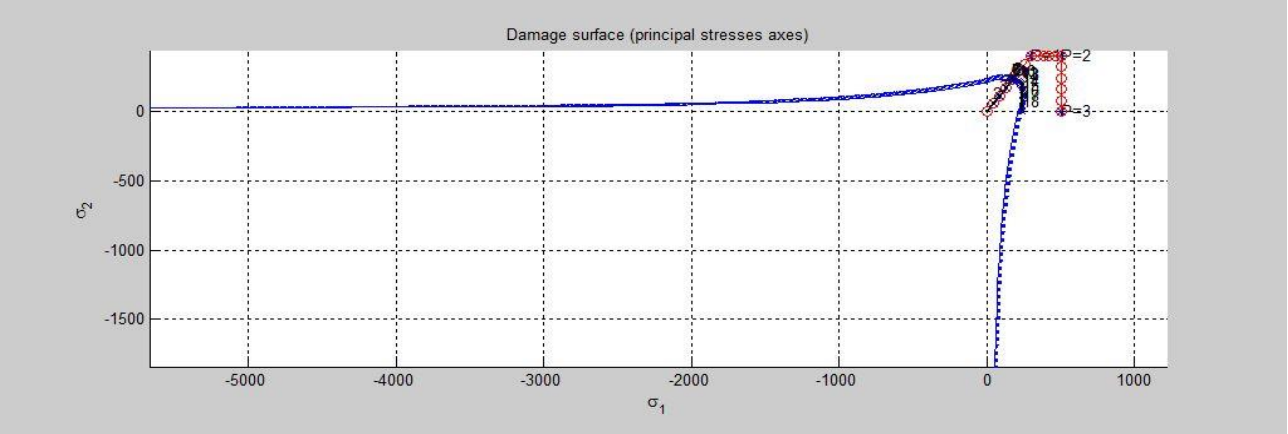

Fig 1: Damage surface for tension-only model.

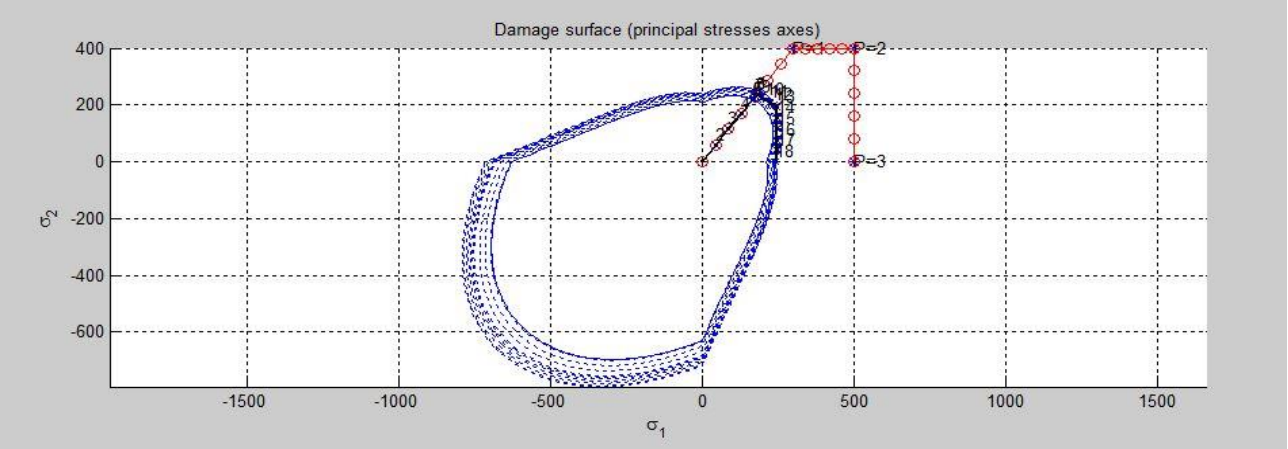

Fig2: Damage surface for Non-symmetric model.

It can be seen In the tension-only damage surface (Figure 1) that, it has the same ellipsoidal surface as the symmetric case in the positive part of the stress-space. But in the negative part of the stress-space(compression), the domain is open to infinity. This is because when the stress is compressive, the Macaulay bracket is zero. This implies that the norm is zero. Thus the r_0 is always negative and never approaches zero. On the other sides, the graph behaves asymptotically to infinity.

In the non-symmetric model (Figure 2), the positive stress-space part has the same ellipsoidal as symmetric model as before but in the negative (compression) part the ellipsoidal is further away than the usual. This is because the norm is amplified with the factor (1/n) where, ”n” being the ratio of the uniaxial stress in compression and tension. In the other two sides, the graph shows linear variation of stresses.

Next, the linear and exponential hardening/softening code (matlab code in the appendix) is implemented for both the above models. The results obtained are presented in the figures below:

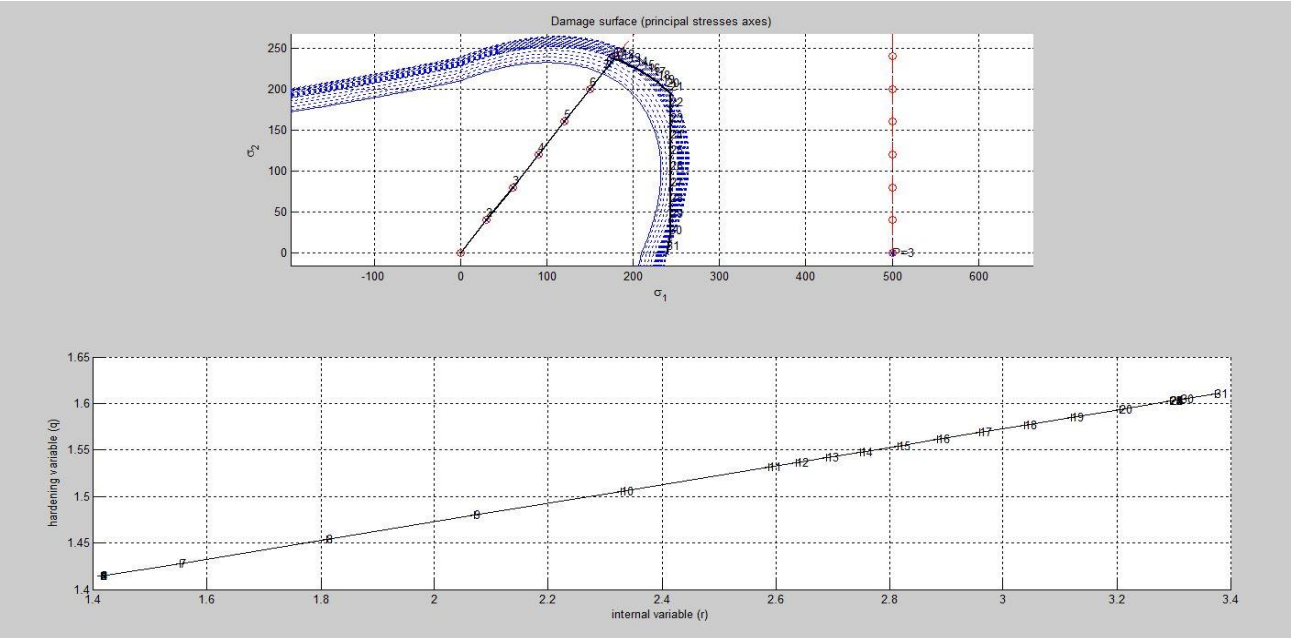

Fig3: Linear Hardening of Tension-only model

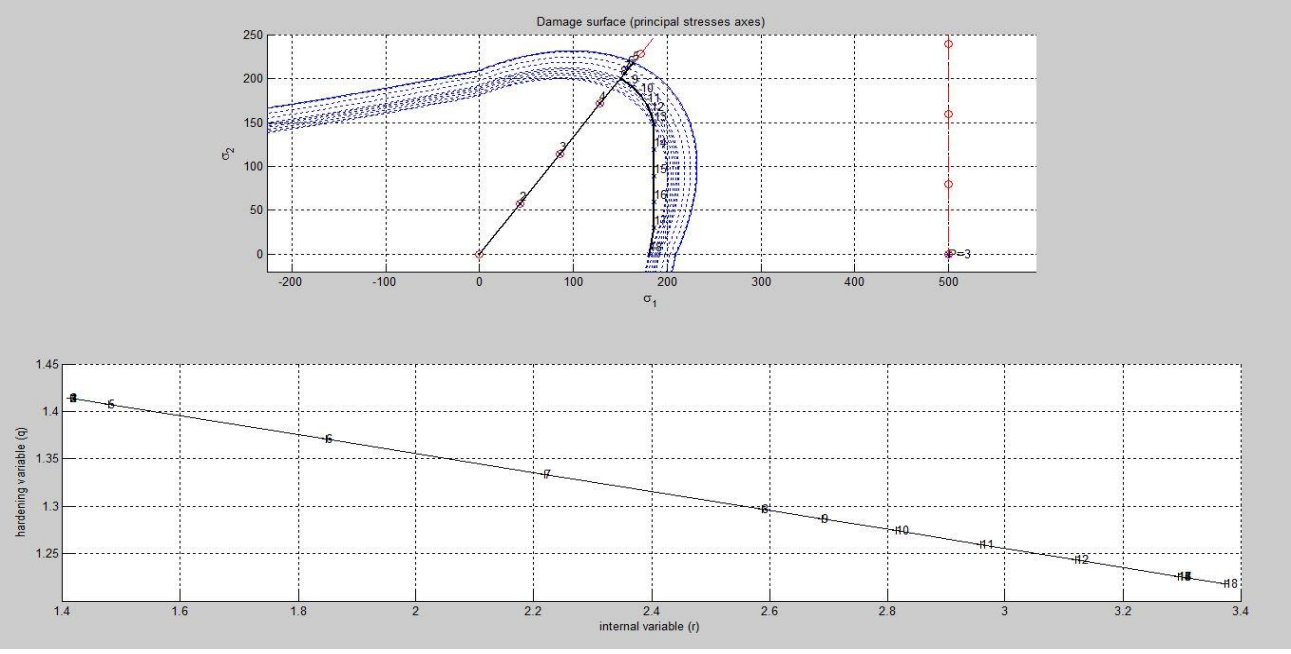

Fig4: Linear softening Tension-only model

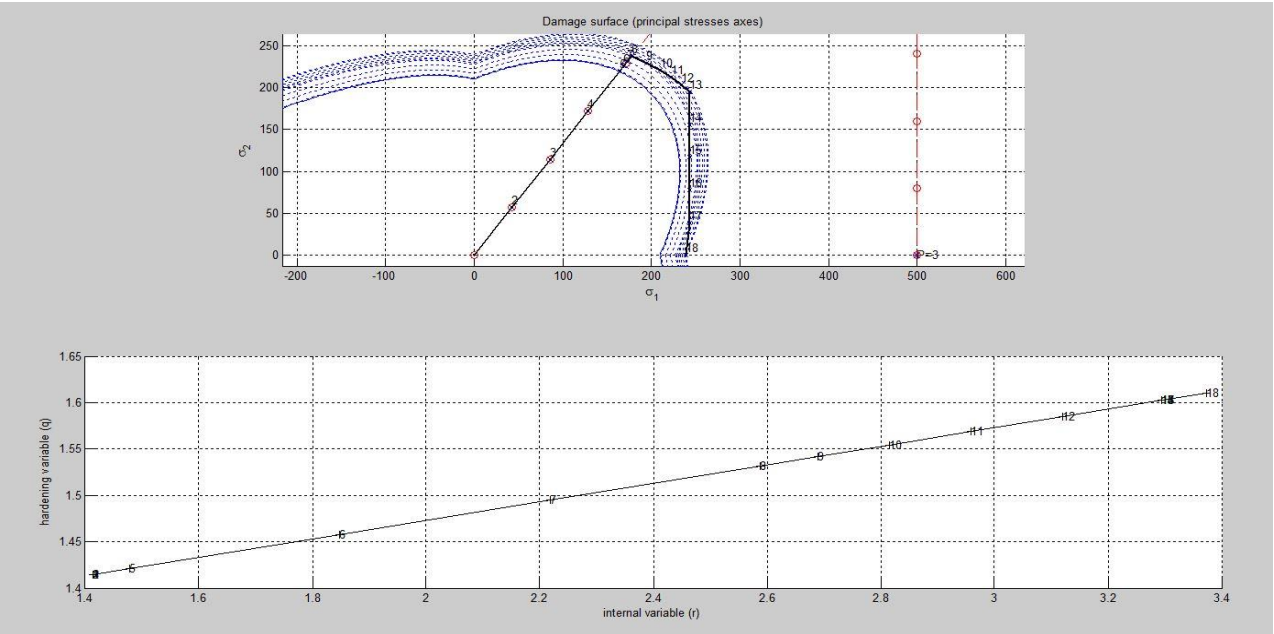

Fig5: Linear hardening non-symmetric model

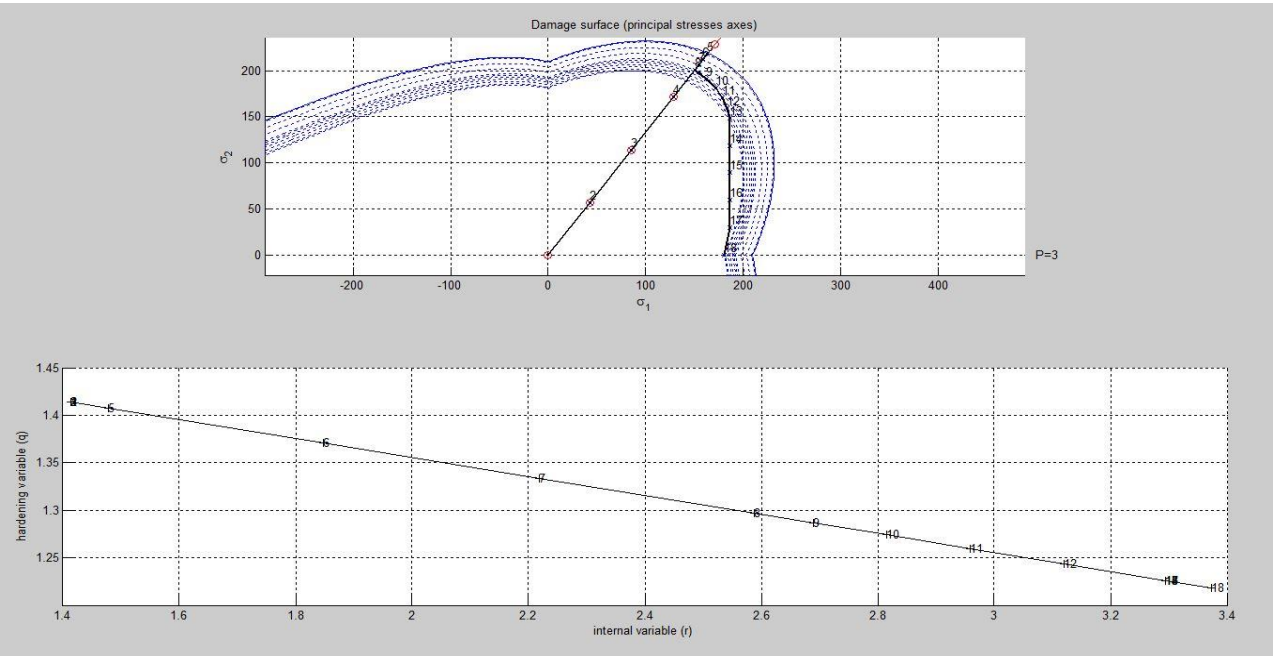

Fig6: Linear softening non-symmetric model

In the above 4 figures (figure2 to figure6), the linear hardening and softening for tension only and non-symmetric models are shown. The graphs show the variation of hardening variable q as internal variable increases. When H>0, hardening occurs and graphs has linear positive variation. When H<0, softening occurs and graphs has linear negative variation. For both the model the graphs shows similar pattern.

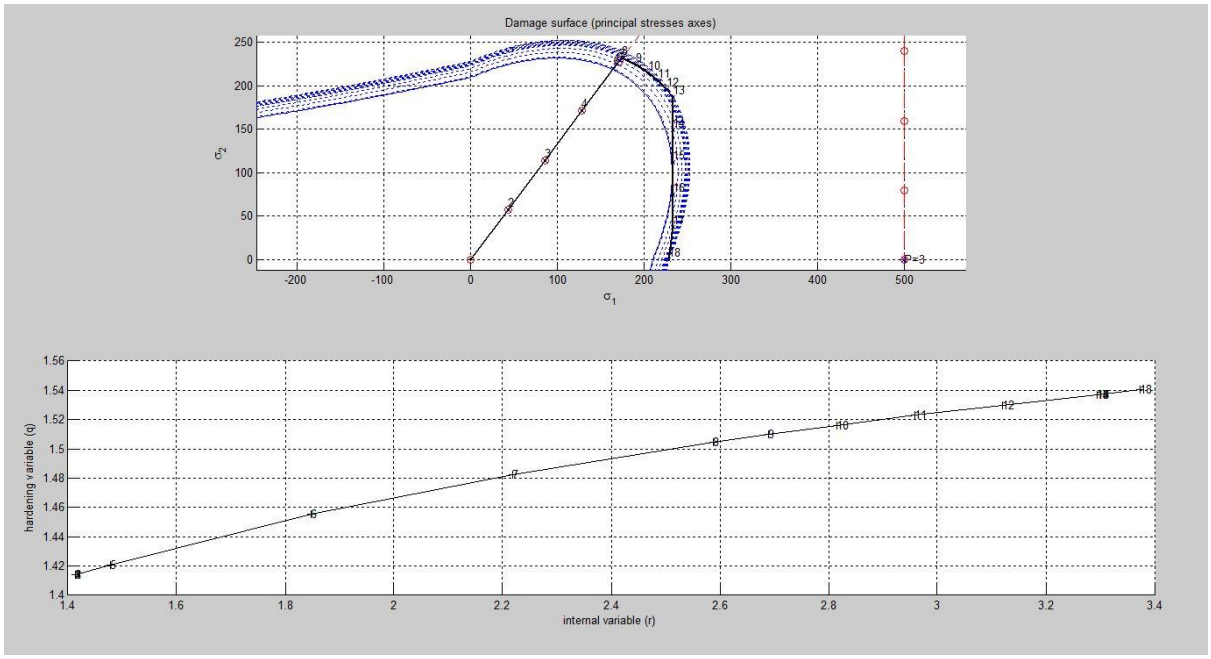

Fig7: exponential hardening Tension-only model

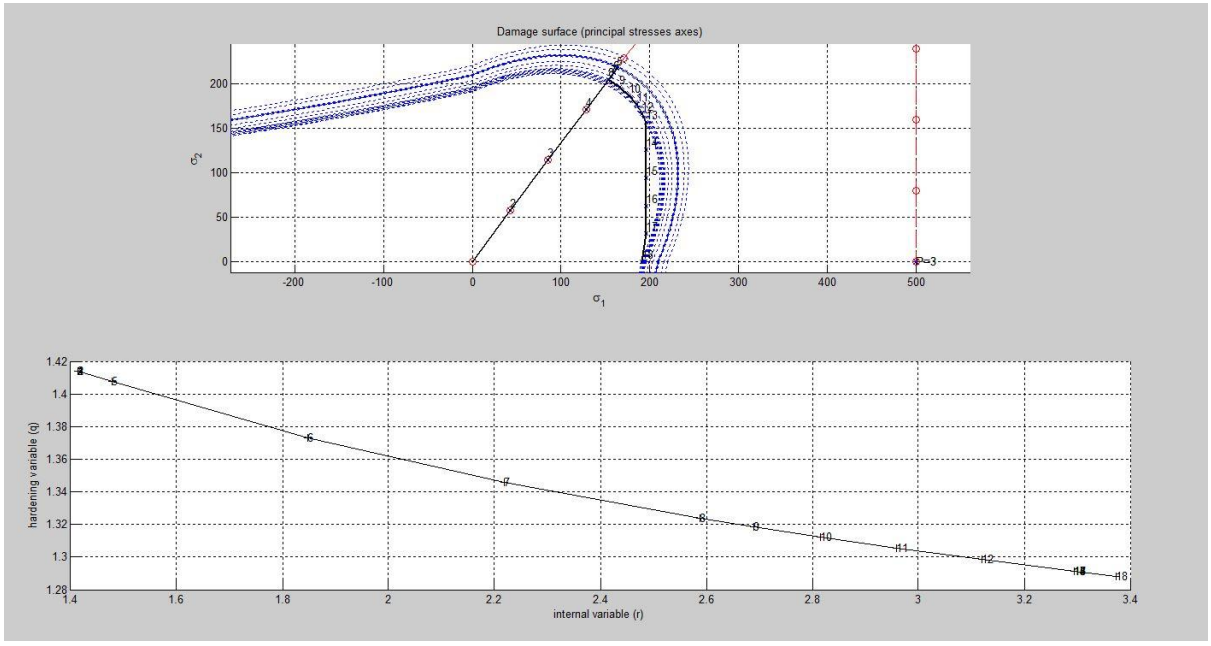

Fig8: exponential softening Tension-only model

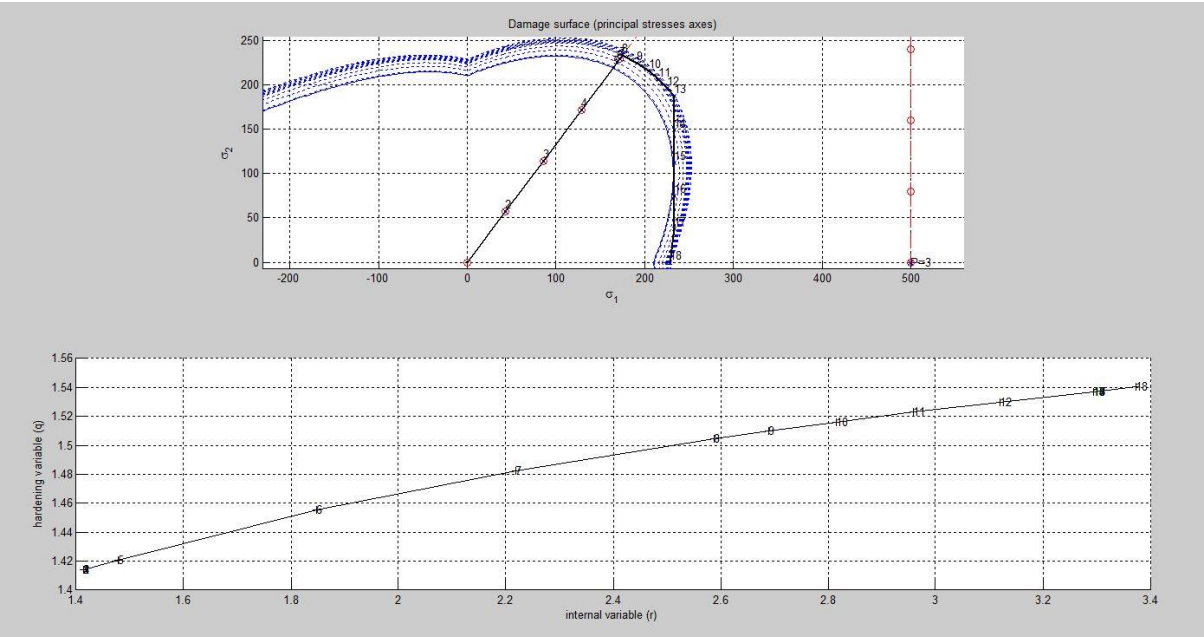

Fig9: exponential hardening non-symmetric model

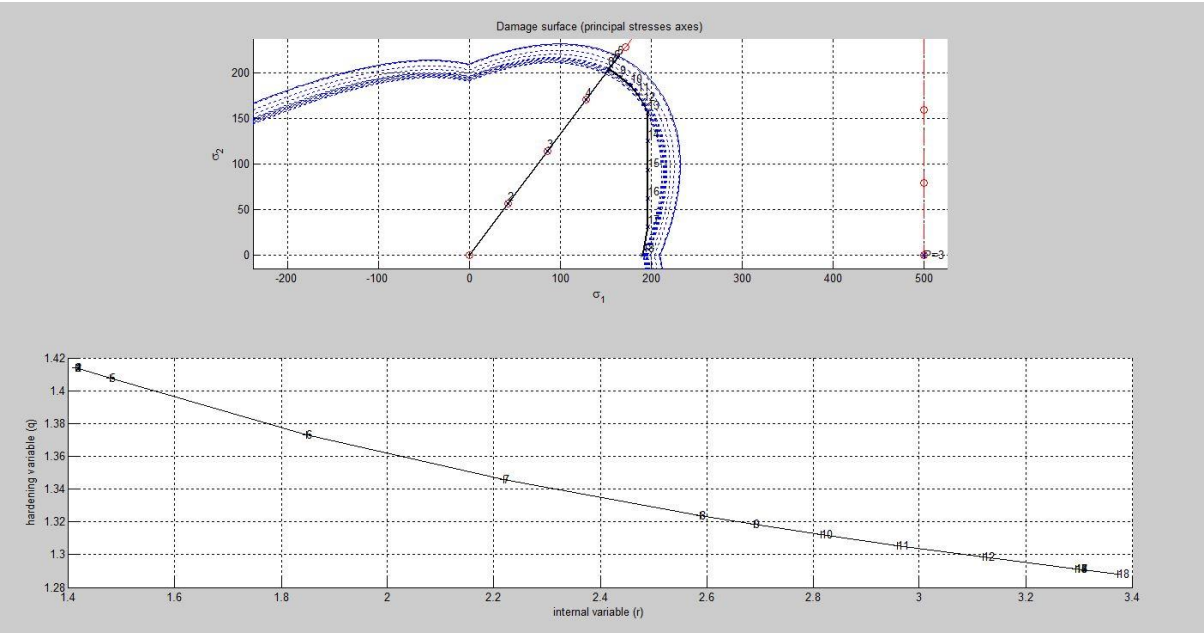

Fig10: exponential softening non-symmetric model

In the above 4 figures (figure7 to figure10), the exponential hardening and softening for tension only and non-symmetric models are shown. When H>0, hardening occurs and graphs has exponential positive variation. When H<0, softening occurs and graphs has exponential negative variation.

The last thing that has to be done in this rate independent model is to obtain the path at the stress space and stress strain curve for each of the model at appropriate loading paths. For the loading path, the value of alpha=400, beta= -200 and gamma=300 is considered in each case of uniaxial and biaxial loading/unloading. Only linear hardening/softening type is considered. The results obtained are displayed below:

For the first case: Uniaxial tensile loading, uniaxial tensile unloading, and uniaxial compressive unloading.

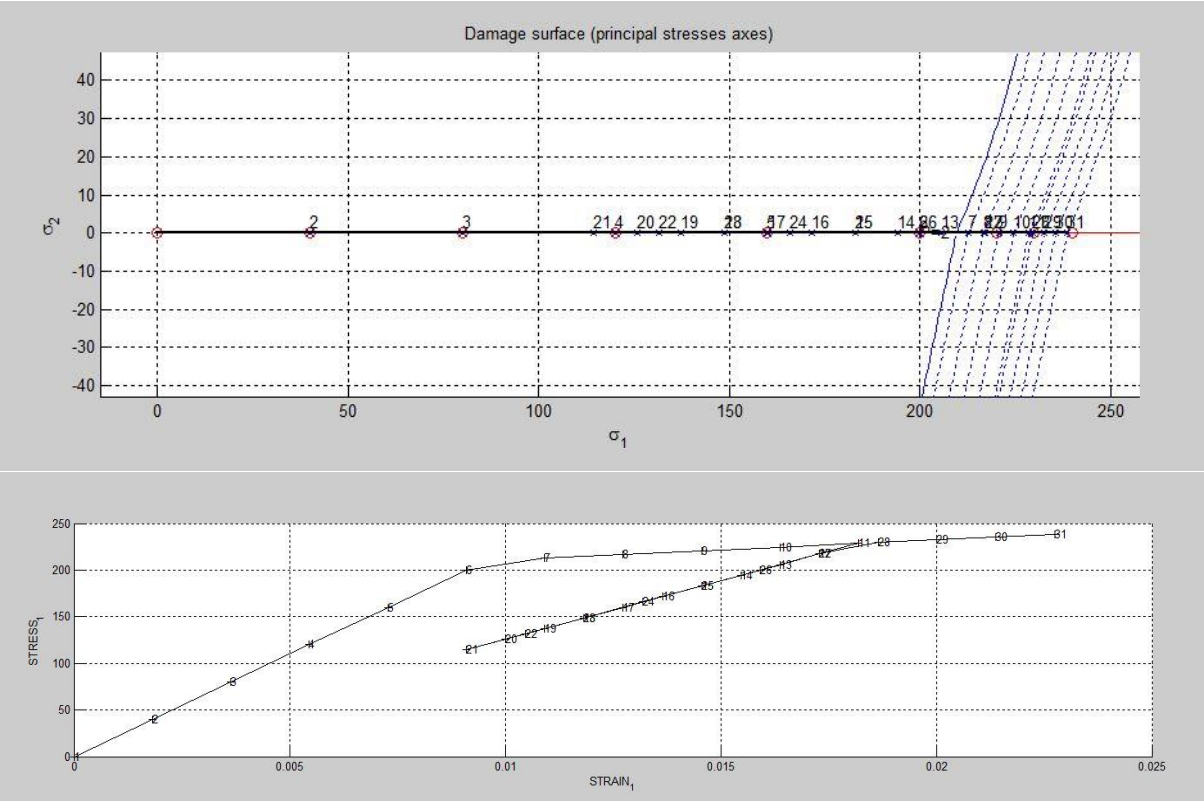

Fig11: path at the stress space and strain stress -curve for the first case load path.(Tension-only Model)

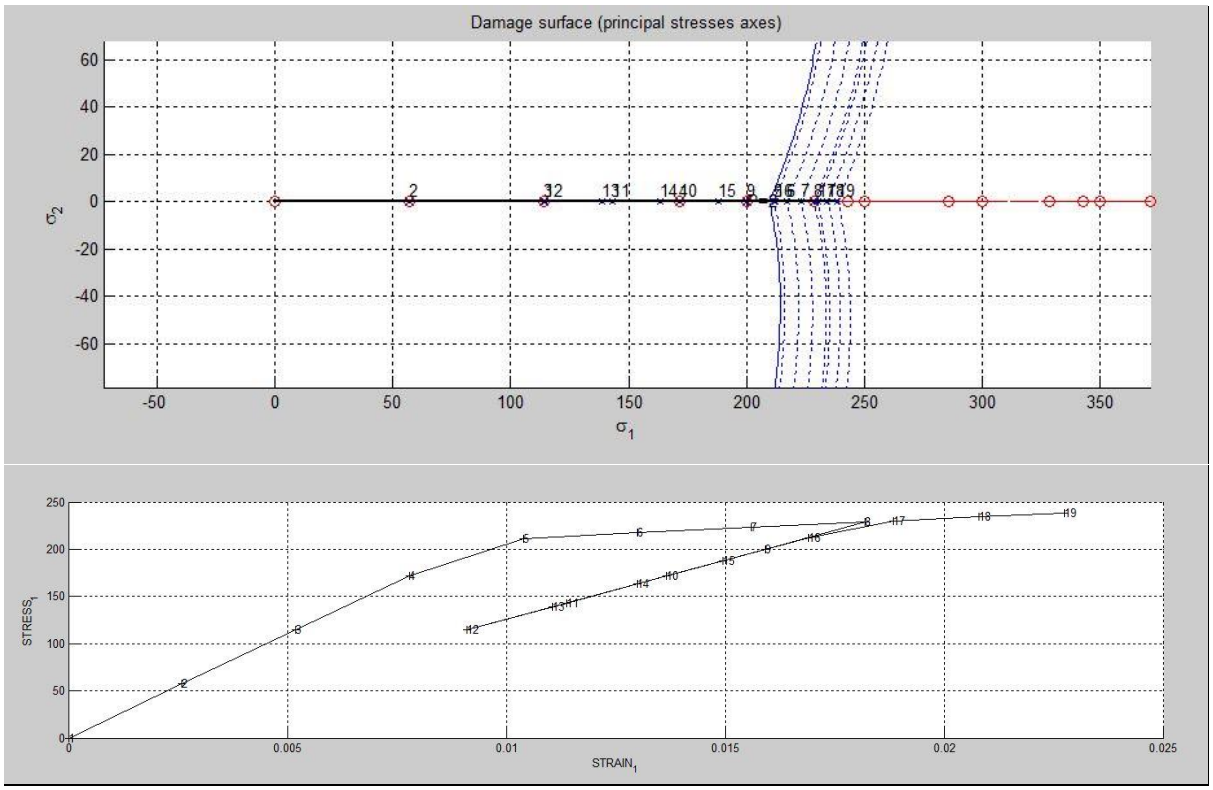

Fig12: path at the stress space and strain stress -curve for the first case load path. (Non-symmetric Model)

For the second case: Uniaxial tensile loading, biaxial tensile unloading, and biaxial compressive unloading.

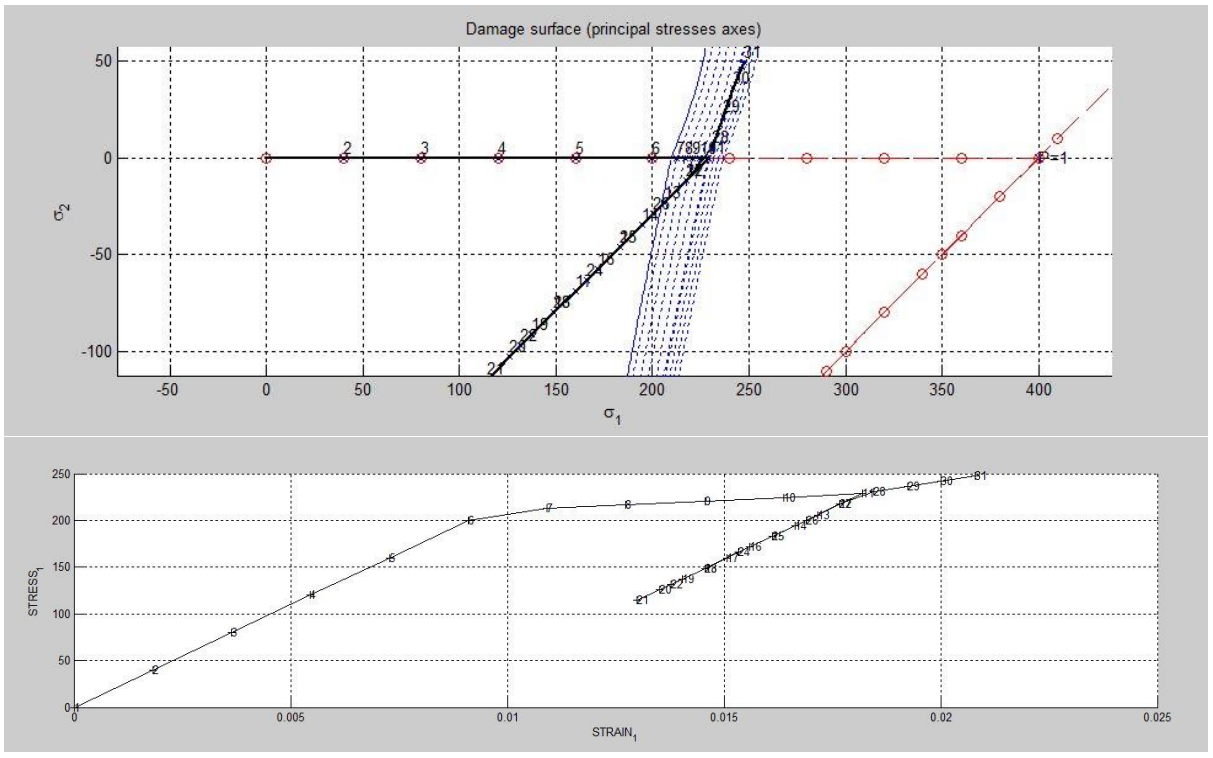

Fig13: path at the stress space and strain stress -curve for the second case load path. (Tension-only Model)

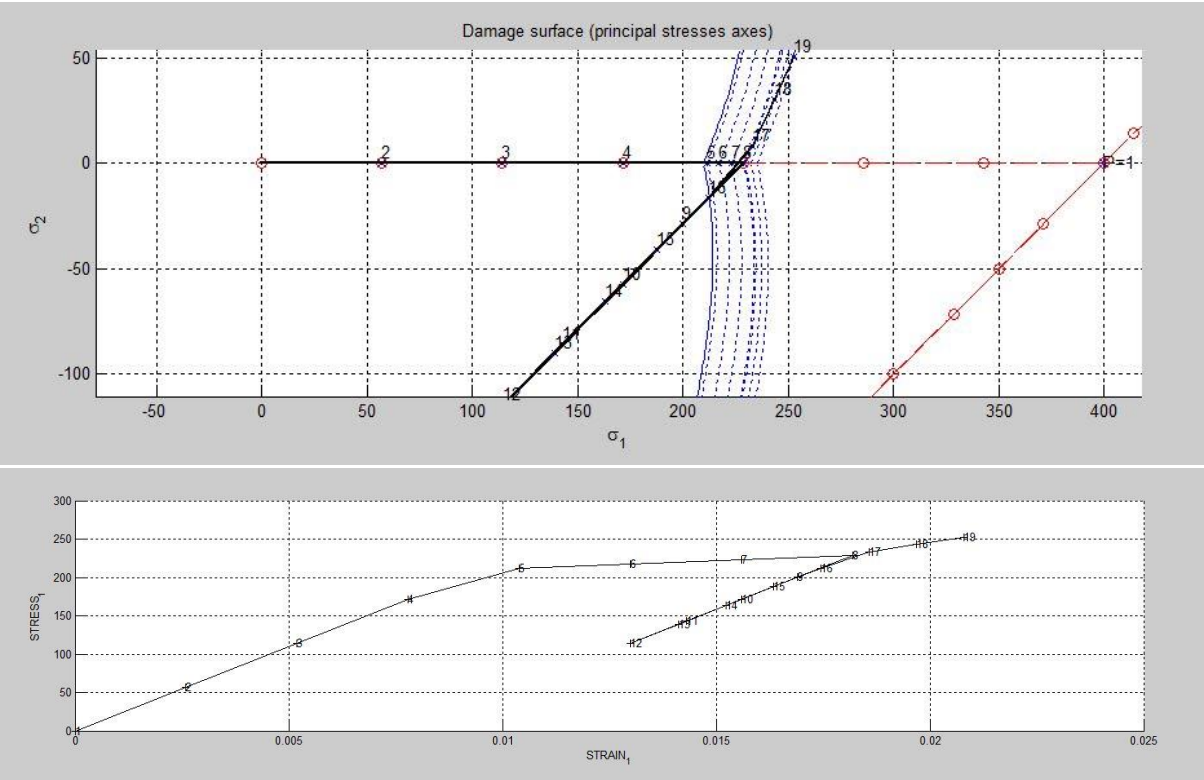

Fig14: path at the stress space and strain stress -curve for the second case load path. (Non-symmetric Model)

For the 3rd case: biaxial tensile loading, biaxial tensile unloading, and biaxial compressive unloading

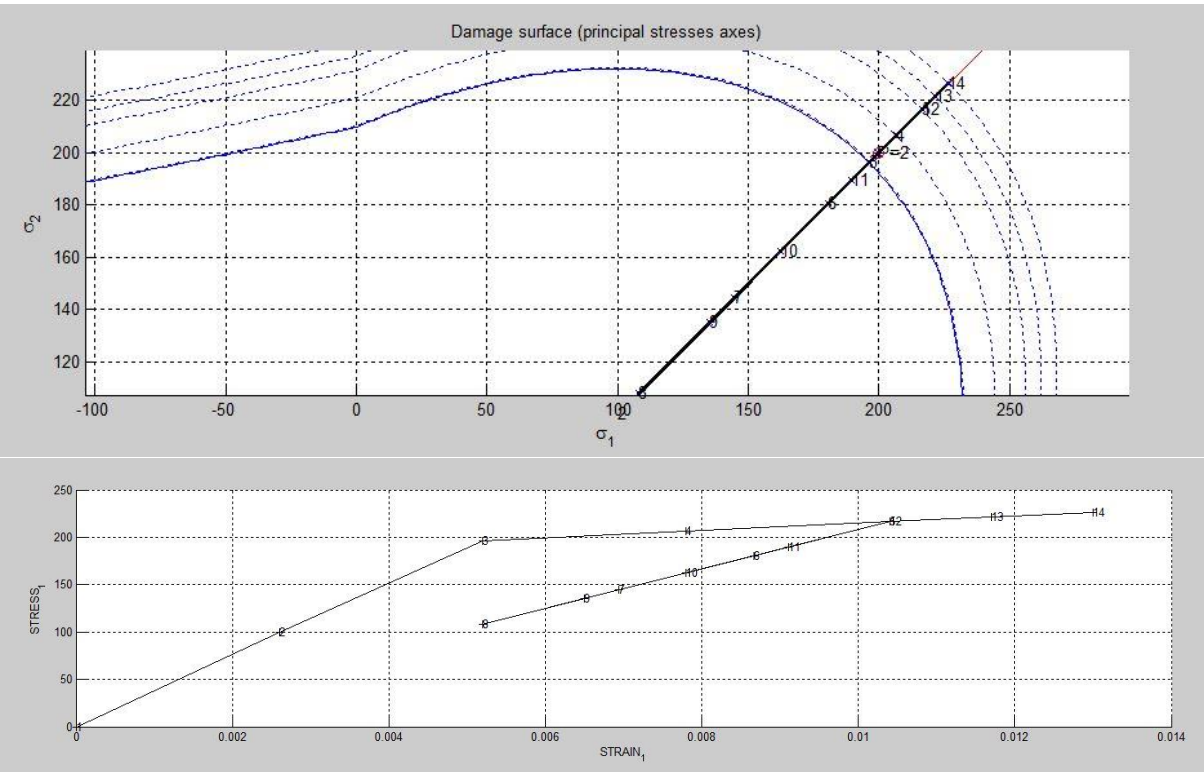

Fig15: path at the stress space and strain stress -curve for the second case load path. (Tension only Model)

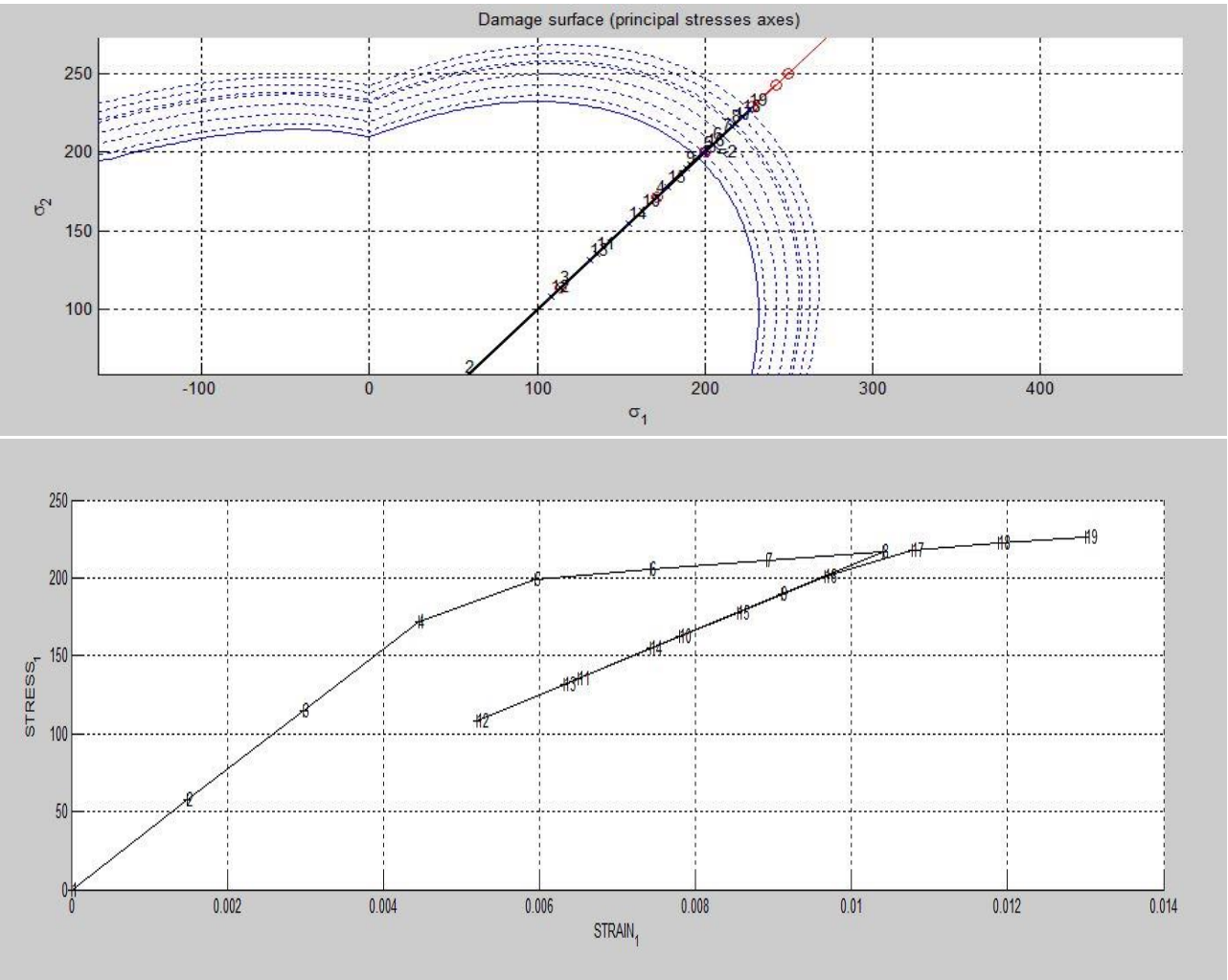

Fig16: path at the stress space and strain stress -curve for the 3rd case load path. (Non-symmetric Model)

From all the graphs above (Fig 11 to fig 16), it can be seen that initially there is elastic loading. Then it undergoes hardening for some time after which unloading happens. After unloading it starts to load elastically once again till certain point in the graph after which it starts to harden again. Similar pattern of behaviour can be seen in all the other cases of load path.

In all the three cases, the graph behaved coherently and all the possible loading unloading and hardening are visible at each stage. Therefore, it can be concluded that the implementation is correct.

Part 2: Rate Dependent Model

For the rate dependent case, only symmetric tension compression model is considered. The algorithm is implemented in Matlab for the viscous model and the resulting plot can be seen below:

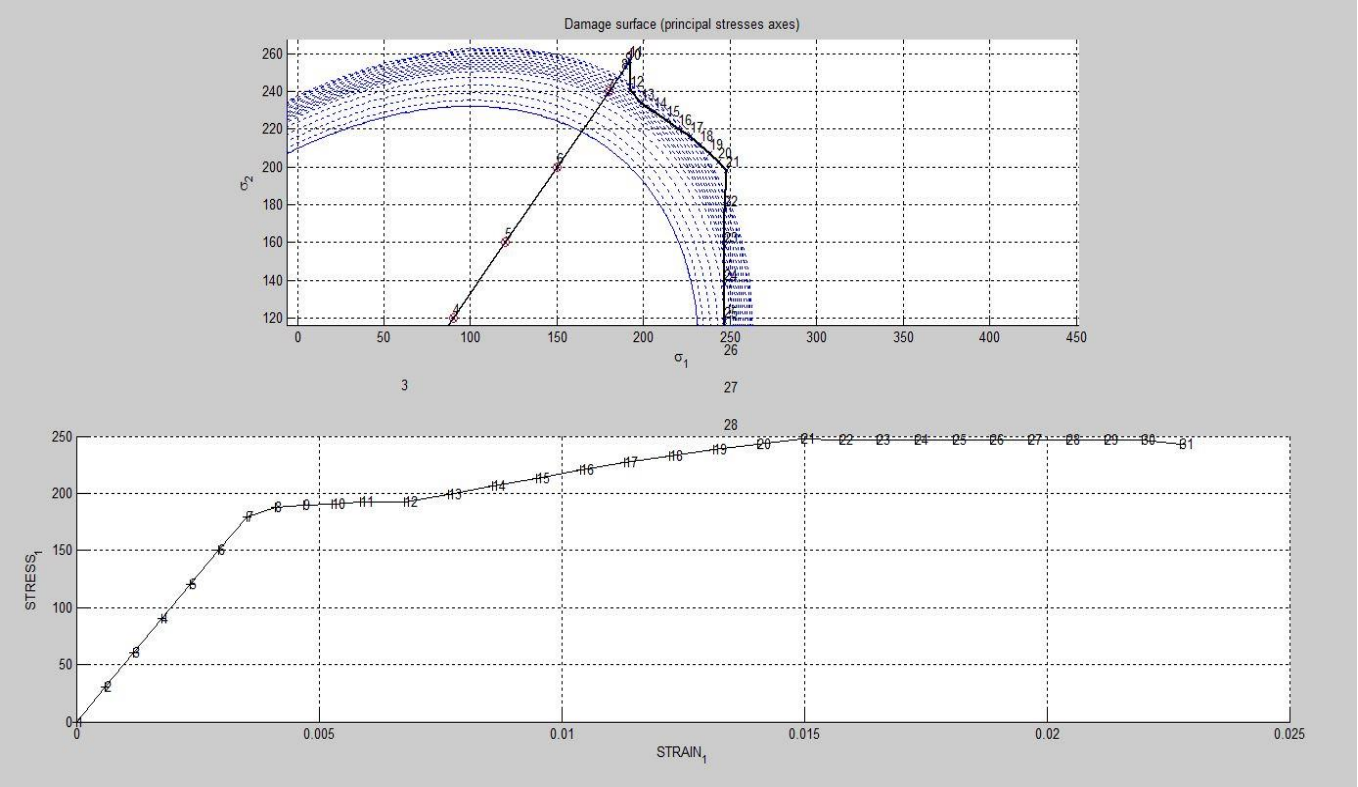

Fig17: Visco-damage model for symmetric tension-compression.

The correctness of the implementation of the algorithm can be verified from the above figure where it can be seen that the point can leave the elastic domain and reach the inelastic domain as viscosity increases. This is indeed expected from a Visco-damage model.

Also the correctness of the implementation can be assessed by varying the viscosity parameter, strain rate and alpha values and obtaining the result in stress-strain curve. For each of the cases, Poisson ratio of 0.3 and hardening/softening parameter of 0.1 is considered.

Different Viscosity parameter

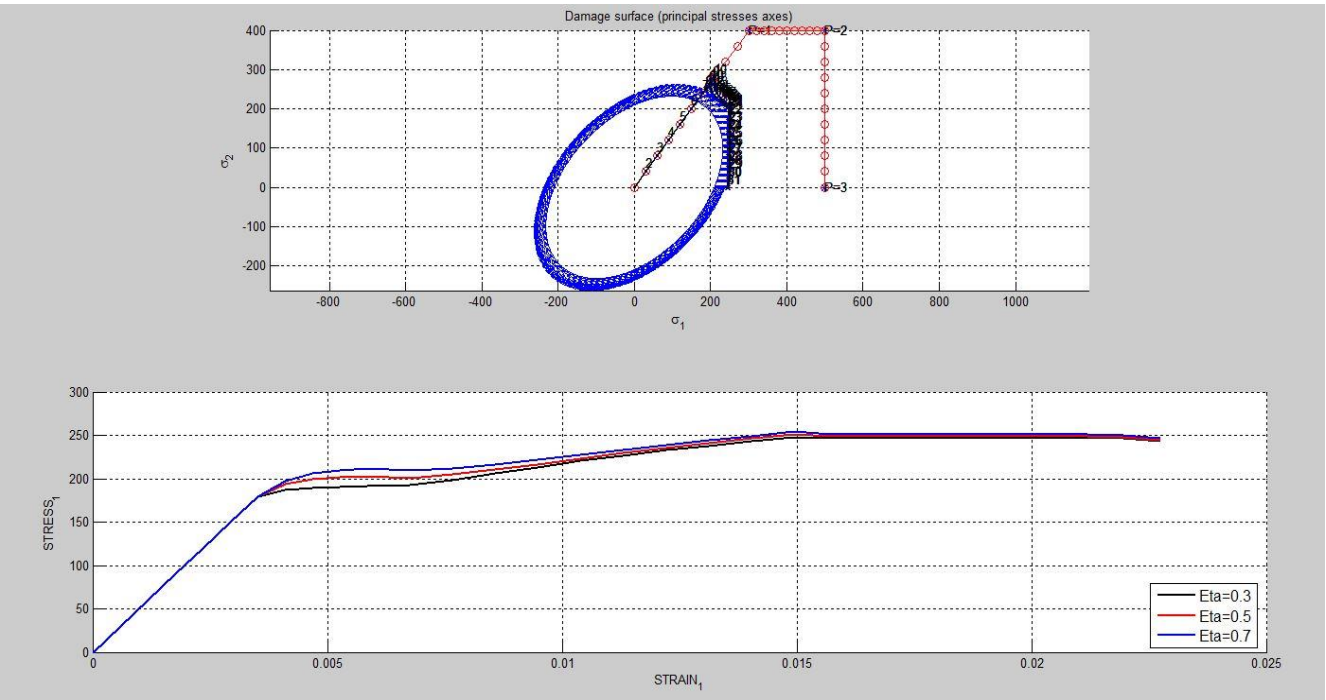

Fig18: stress-strain curve for different values of viscosity

Three different viscosity values (Eta= 0.3, 0.5, 0.7) are used to see the effect on the stress. It can be seen that the larger are the viscosity, the larger are the stresses. In the elastic part of the graph, there is no difference. But in the inelastic part, for the same strain value, the stresses are higher as viscosity increases.

Different strain rate

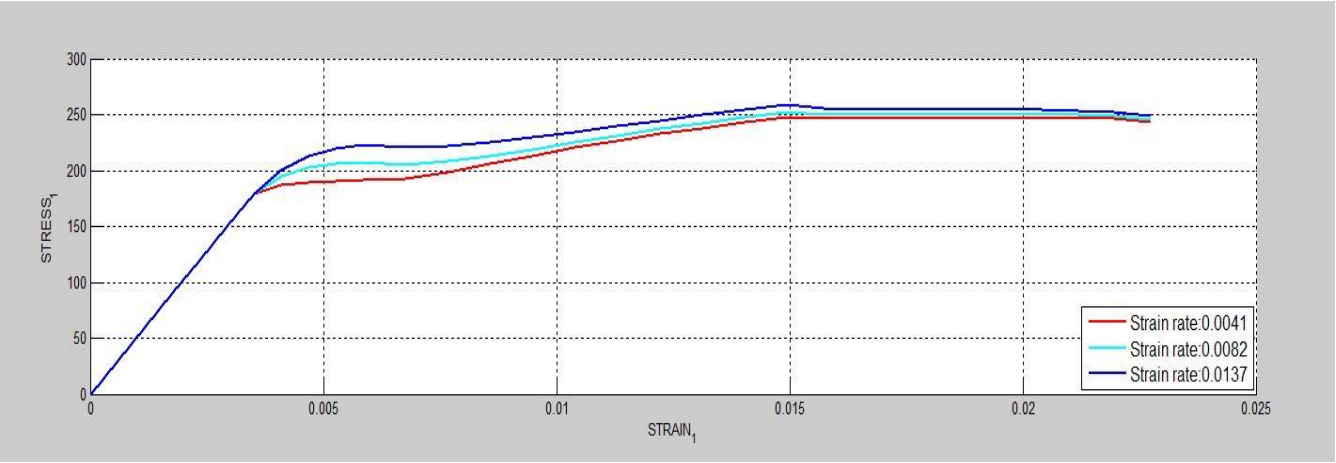

Fig19: stress-strain curve for different values of strain rate.

Here also, three different strain rates are used (0.0041, 0.0082, 0.0137). Similar behaviour

of graphs can be noticed when compare to previous graphs for viscosity. The larger the strain rates, larger are the stresses. The elastic part has no differences. But when the damage is triggered, the inelastic zone, the stresses increases as the rate of the strain increases. So, this shows that the stresses not only depend on the strain but also to the rate of the strain.

If we consider the viscosity to be 0, all the curves collapse to give one curve which is shown in the figure below.

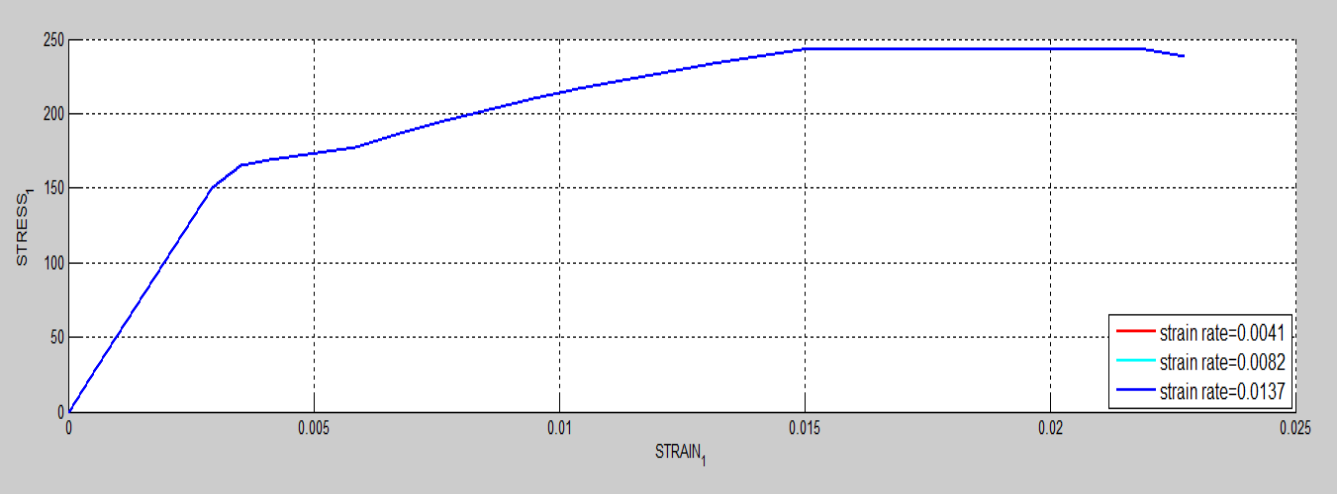

Fig20: collapsing of all the curves when viscosity is zero.

Different Alpha value

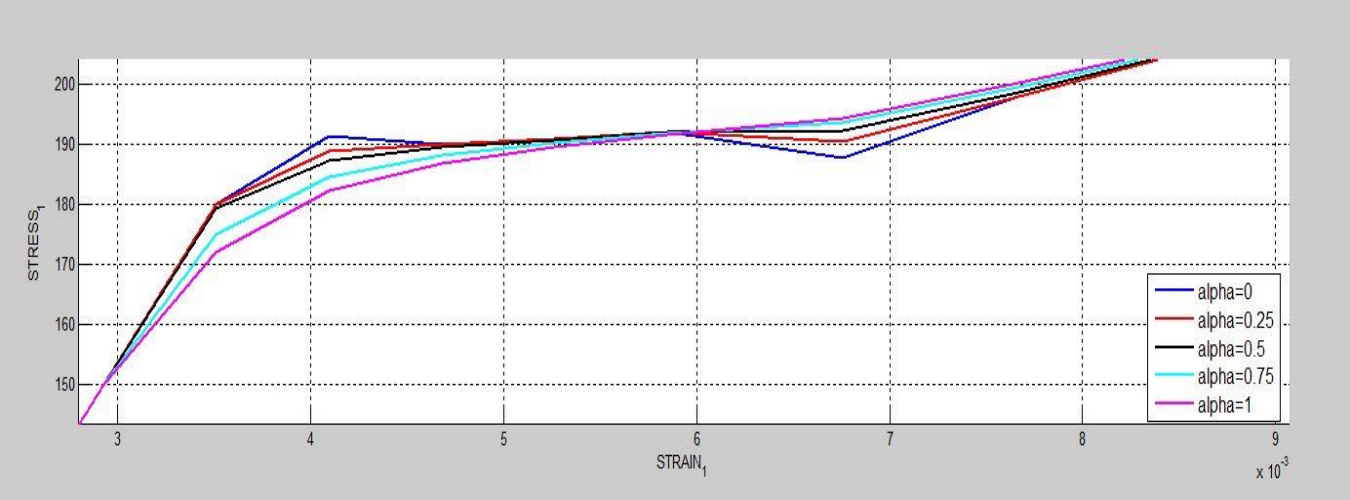

Fig21: stress strain curve for different value of alpha

Five different alpha value are considered in this case (alpha= 0, 0.25, 0.5, 0.75, 1). It can be seen that for all the alpha values the graph is quite similar. It is expected to behave like that as alpha value represents which integration method is being carried out. In between 0 to

0.5, there are some instability in the plot as the methods used are conditionally stable (forward Euler for example). But between alpha values of 0.5 to 1 the graphs are unconditionally stable (Crank Nickelson or Backward Euler for example).

Finally, the effect of alpha values on the evolution along time of the C_{11} component of the tangent and algorithmic constitutive operator is measured. The results are displayed in the following two figures:

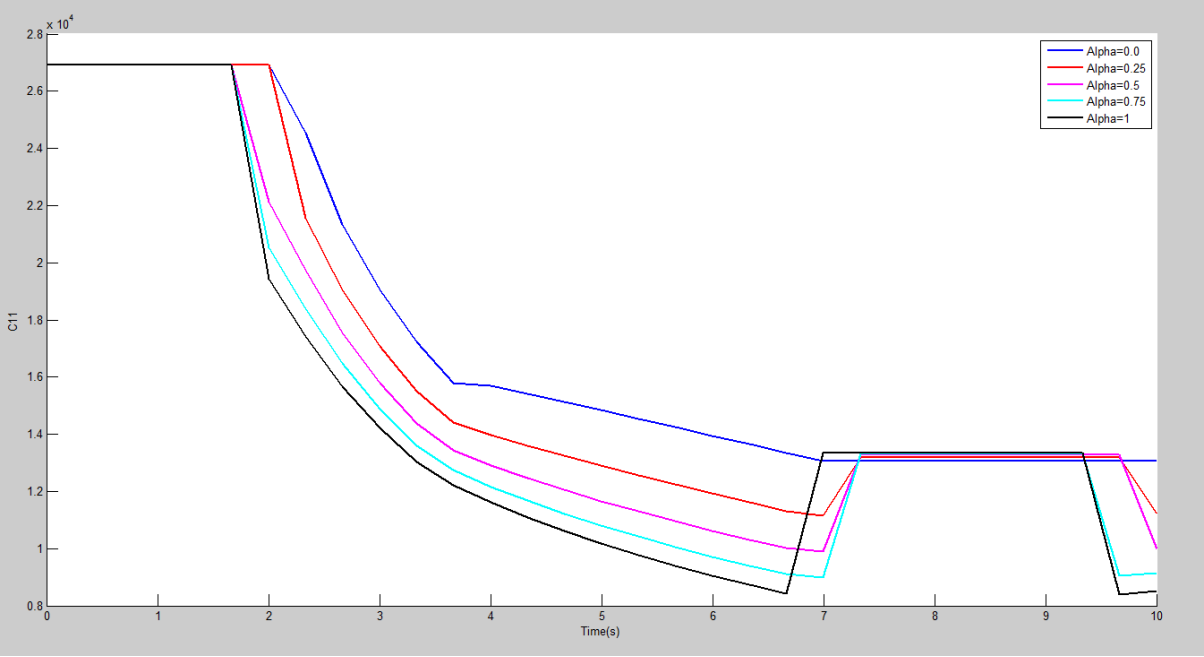

Fig22: effect of alpha on C_{11}component of algorithmic constitutive operator.

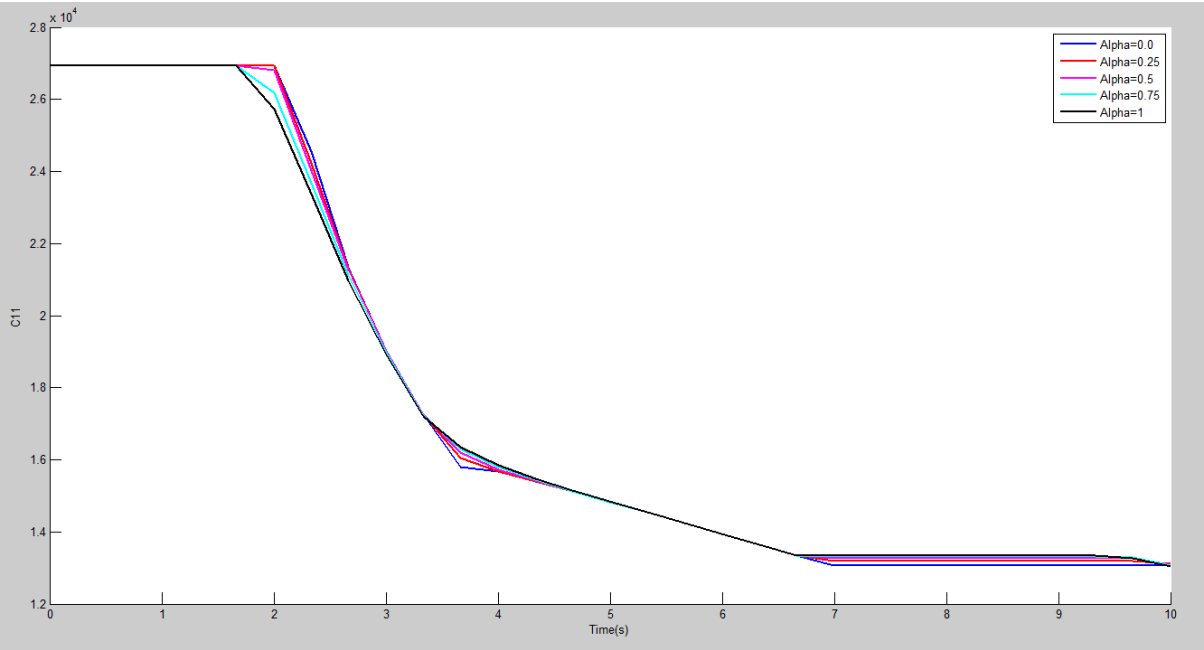

Fig21: effect of alpha on C_{11}component of tangent constitutive operator.

APPENDIX

For tension only and non- symmetric model.

```
%*************************************************************************************
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
sigma1 =eps_n1*ce;
thet = ((sigma1(1).*(sigma1(1)>0))+(sigma1(2).*(sigma1(2)>0)))/(abs(sigma1(1))+abs(sigma1(2)));
rtrial = (thet+(1-thet)/n).*sqrt(eps_n1*ce*eps_n1');
%rtrial= sqrt(eps_n1*ce*eps_n1')
end
%*************************************************************************************
```

```
elseif MDtype==2
   tetha=[0:0.01:2*pi];
    %*************************************************************************
    %* RADIUS
    D=size(tetha);                  %*  Range
    m1=cos(tetha);                  %*
    m2=sin(tetha);                  %*
    Contador=D(1,2);                %*


    radio = zeros(1,Contador) ;
    s1    = zeros(1,Contador) ;
    s2    = zeros(1,Contador) ;

    for i=1:Contador
        radio(i)= q/sqrt(([m1(i) m2(i) 0 nu*(m1(i)+m2(i))]).*(([m1(i) m2(i) 0 nu*(m1(i)+m2(i))])>0)*ce_inv*[m1(i) m2(i) 0 ...
            nu*(m1(i)+m2(i))]');

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

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

```
elseif MDtype==3
tetha=[0:0.01:2*pi];
    %*************************************************************************************
    %* RADIUS
    D=size(tetha);                      %*  Range
    m1=cos(tetha);                      %*
    m2=sin(tetha);                      %*
    Contador=D(1,2);                    %*


    radio = zeros(1,Contador) ;
    s1    = zeros(1,Contador) ;
    s2    = zeros(1,Contador) ;


    for i=1:Contador
        thet(i) = ((m1(i).*(m1(i)>0))+(m2(i).*(m2(i)>0)))/(abs(m1(i))+abs(m2(i)));
         radio(i)= q/((thet(i)+(1-thet(i))/n).*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

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

For hardening/softening

```
if(rtrial > r_n)
    %*   Loading

    fload=1;
    delta_r=rtrial-r_n;
    r_n1= rtrial  ;
    if hard_type == 0
        %  Linear
        q_n1= q_n+ H*delta_r;
    else
         A = abs(H);
         if H>0
        qmax = r0+r0-zero_q;
        q_n1 = qmax-(qmax-q_n)*exp(A*(1-r_n1/r_n)) ;
         elseif H<0
         qmin = zero_q;
         q_n1 = qmin-(qmin-q_n)*exp(A*(1-r_n1/r_n)) ;
         end
    end
```

For implementing Visco-model

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

%*************************************************************************************
%*                                        *
%*           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);
Eta     = Eprop(7);
Alpha = Eprop(8);
%*************************************************************************************
```

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


%*************************************************************************************
%*       Damage surface                                                                %*
[rtrial,rtrial_n] = Modelos_de_dano1 (MDtype,ce,eps_n1,n,eps);
%*************************************************************************************
rtrial_alpha= ((1-Alpha)*rtrial_n) + (Alpha*rtrial);

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

```
if(rtrial_alpha > r_n)
    %*   Loading

    fload=1;
    r_n1= (((Eta-delta_t.*(1-Alpha))/(Eta+Alpha*delta_t)).*r_n)+((delta_t/(Eta+Alpha*delta_t)).*rtrial_alpha)  ;
    delta_r=r_n1-r_n;
    if hard_type == 0
        %  Linear
        q_n1= q_n+ H*delta_r;
    else
         A = abs(H);
         if H>0
        qmax = r0+r0-zero_q;
        q_n1 = qmax-(qmax-q_n)*exp(A*(1-r_n1/r_n)) ;
         elseif H<0
         qmin = zero_q;
         q_n1 = qmin-(qmin-q_n)*exp(A*(1-r_n1/r_n)) ;
         end
    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
```

For C11 component

```
if fload==0
    C_al=(1-dano_n1)*ce;
else
    C_al=(1-dano_n1)*ce+((Alpha*delta_t)/(Eta+Alpha*delta_t))*(1/rtrial)*((H*r_n1-q_n1)/(r_n1^2))*(sigmabar*sigmabar');
end


C_tan=(1-dano_n1)*ce;

C11_al= C_al(1,1);
C11_tan=C_tan(1,1);
```