COMPUTATIONAL SOLID MECHANICS

ASSIGNMENT 1

Kiran Sagar Kollepara
Computational Mechanics

Part 1: Inviscid Damage Model

Figure 1 shows the simplest of all cases, uniaxial loading on a Linear, symmetric damage model. Some of the expected characteristics that can be seen:

- The stress strain curve clearly shows change in slope caused due to the damage, which occurs at both tensile and compressive loading.
- The slope of the constitutive modulus decreses as predicted by the thermodynamic laws. Also, the stress point never moves out of the damage surface.
- The stress point at the end step (31) lies exactly on the damage surface.
- $\sigma_2 = 0$, but $\epsilon_2 \neq 0$ because of Poisson effect

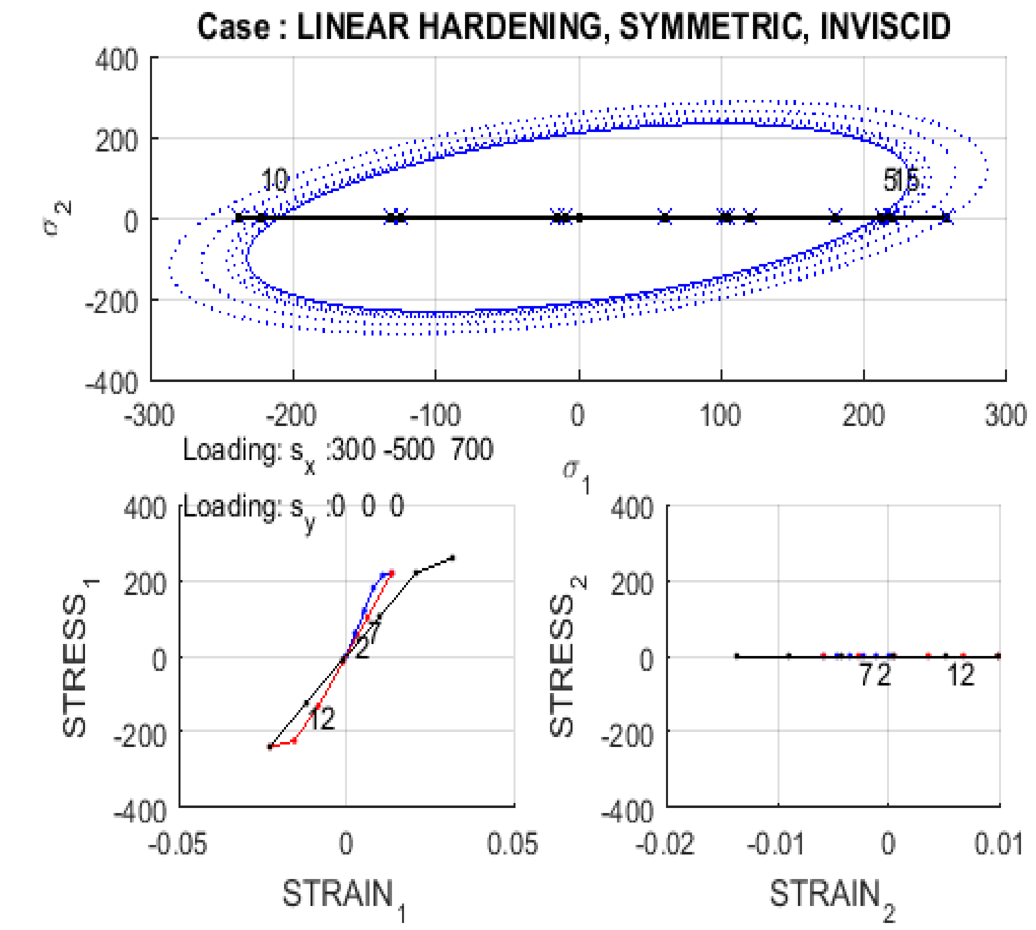

Figure 1: Uniaxial loading with Symmetric Linear Hardening.

Figure 2 shows the 'Tension only' case. Observations:

- No change of slope observed in stress-strain curve in the compression region.
- The unloading path in the compression region is the same as the loading path. Also, the reloading path in tension region is same as the first unloading path.
- The damage surface evolves only with tensile loading in direction 1.

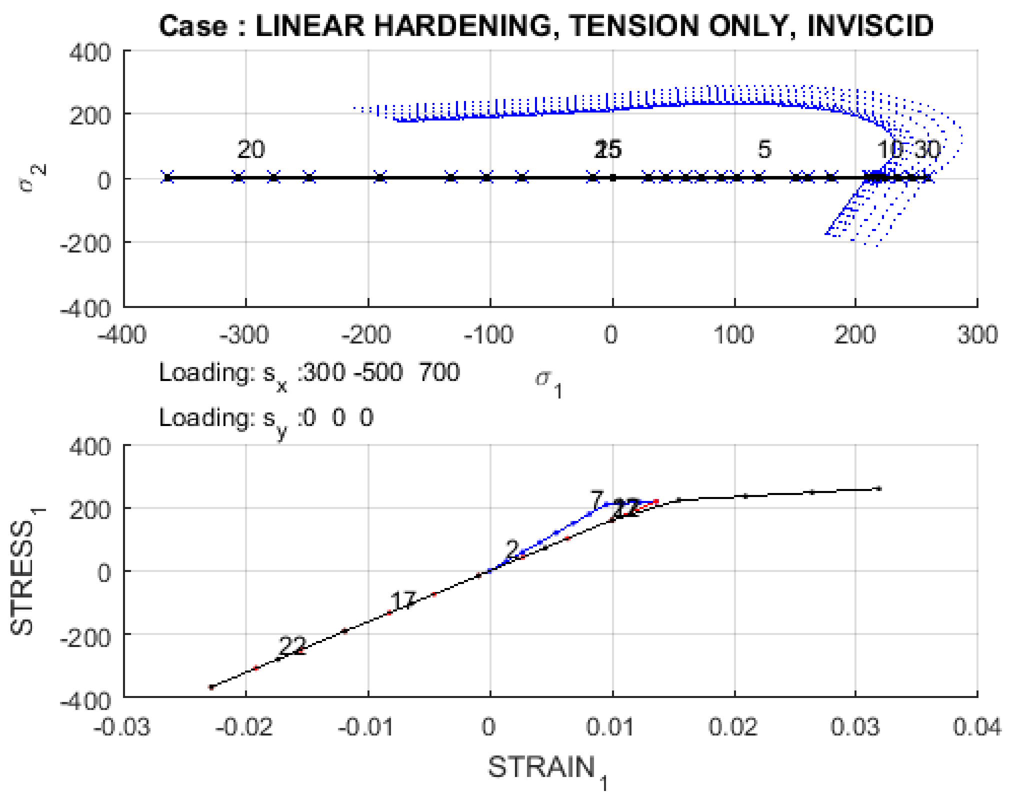

Figure 2: Uniaxial loading with Tension only damage, Linear Hardening

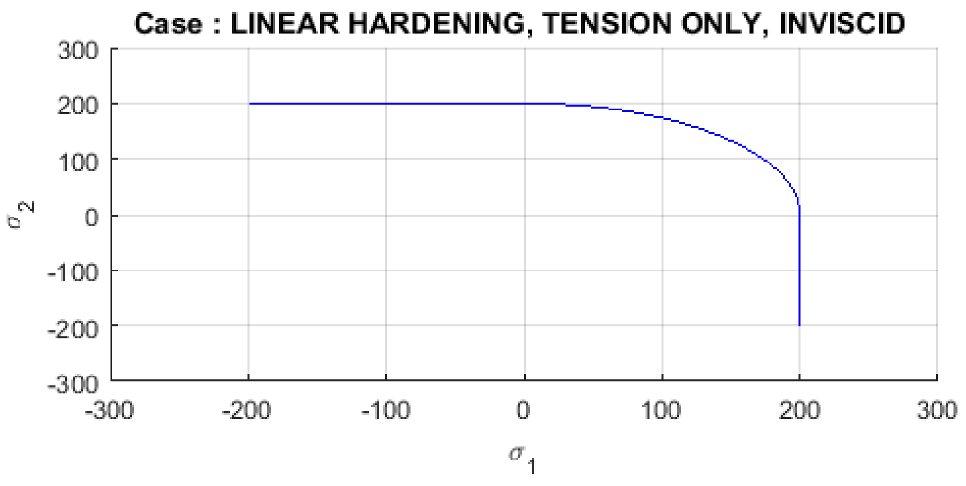

Figure 3: Tension only model with $\nu = 0$

In Figure 2, the Poisson coefficient is taken as 0.3. In contrast, Figure 3 shows the damage surface of a tension only model with Poisson coefficient $\nu = 0$.

- The surface runs parallel to the axes in first and third quadrants fo stress space. This means that any strain in a given direction doesn't contribute to stress in other direction.
- The damage surface in the first quadrant is not elliptical, but circular. This is also reflective of absence of Poisson's effect, where in positive strains in one direction have negative contribution to the stresses in other direction.

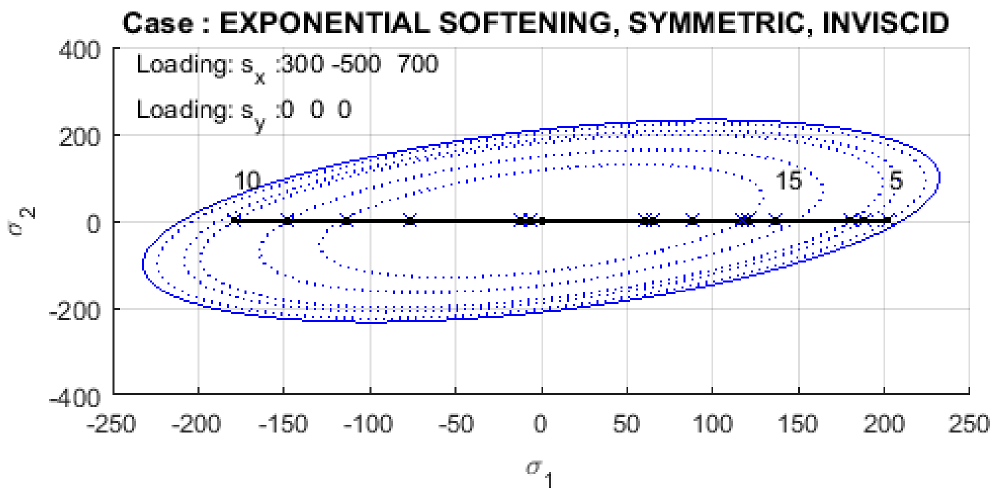

Figure 4: Uniaxial loading with Symmetric Exponential Softening

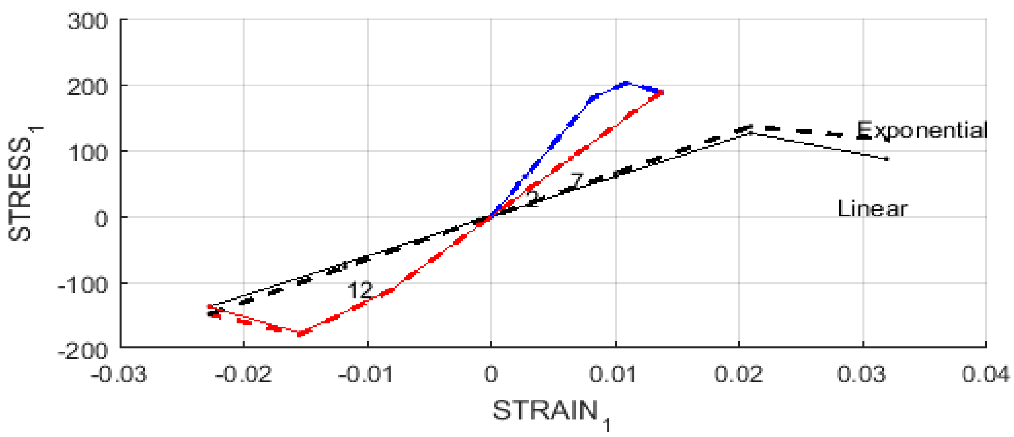

Figure 5: Comparison of Linear and Exponential softening for same initial H_d

Figure 4 shows the exponential *Softening* behaviour. Figure 5 shows a comparison of Linear and Exponential softening behaviour.

- The size of damage surface in stress domain reduces with evolution, dragging back the stress point along with it.
- The softening can be seen slowing down approaching perfect plasticity in exponential case, whereas it remains constant in Linear case.

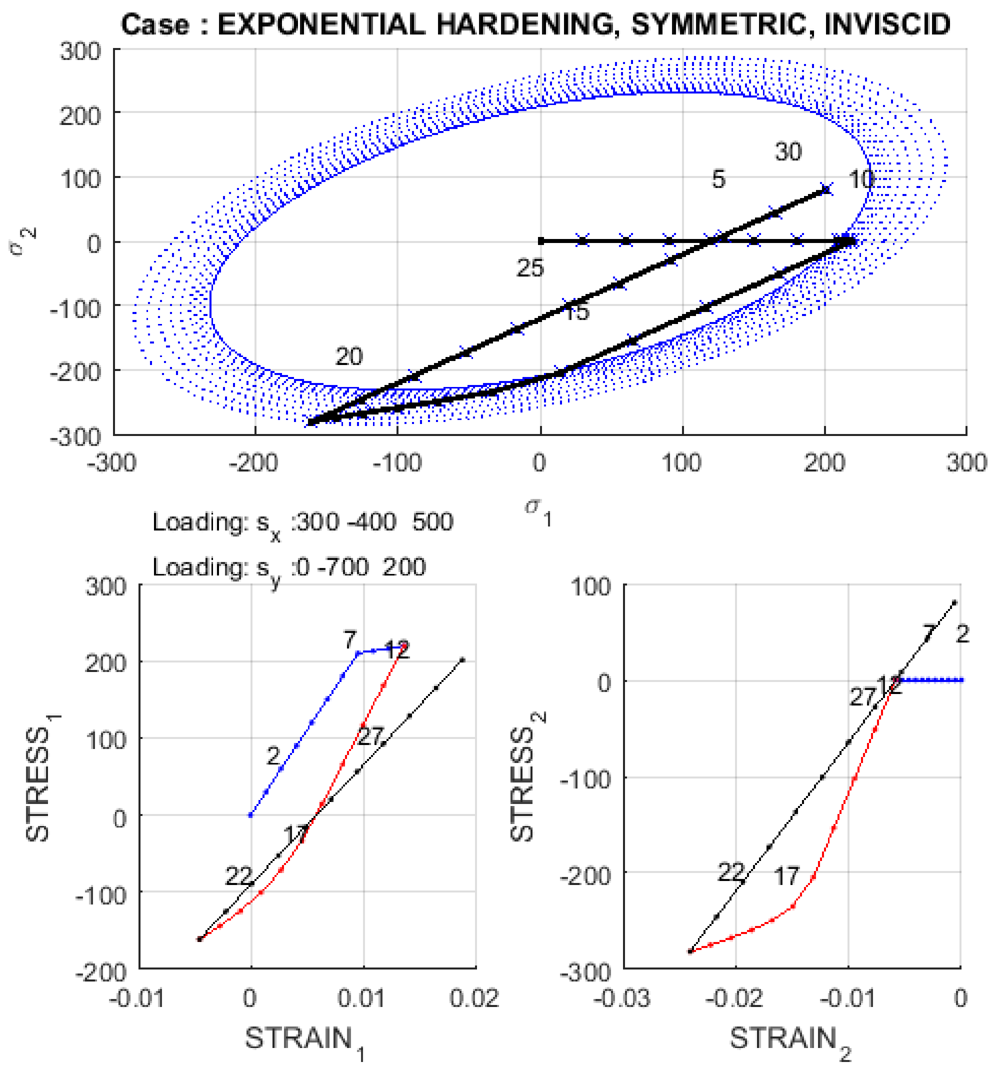

Figure 6: Bi-axial loading with Symmetric Exponential Hardening

Figure 6 and Figure 7 show the same loading for symmetric and assymetric tension-compression model. The following observations/comparisons can be made:

- The first step of loading(tensile) is identical in both cases. This is because the tensile damage surface is identical at the beginning.
- The second step, compressive loading, shows a different behaviour because the loading never reaches the damage surface in asymmetric model. Hence, the damage surface doesn't evolve in this model.
- The lack of evolution in the second load step influences the 3rd step. In assymetric model, the stress point reaches the damage surface earlier than this.

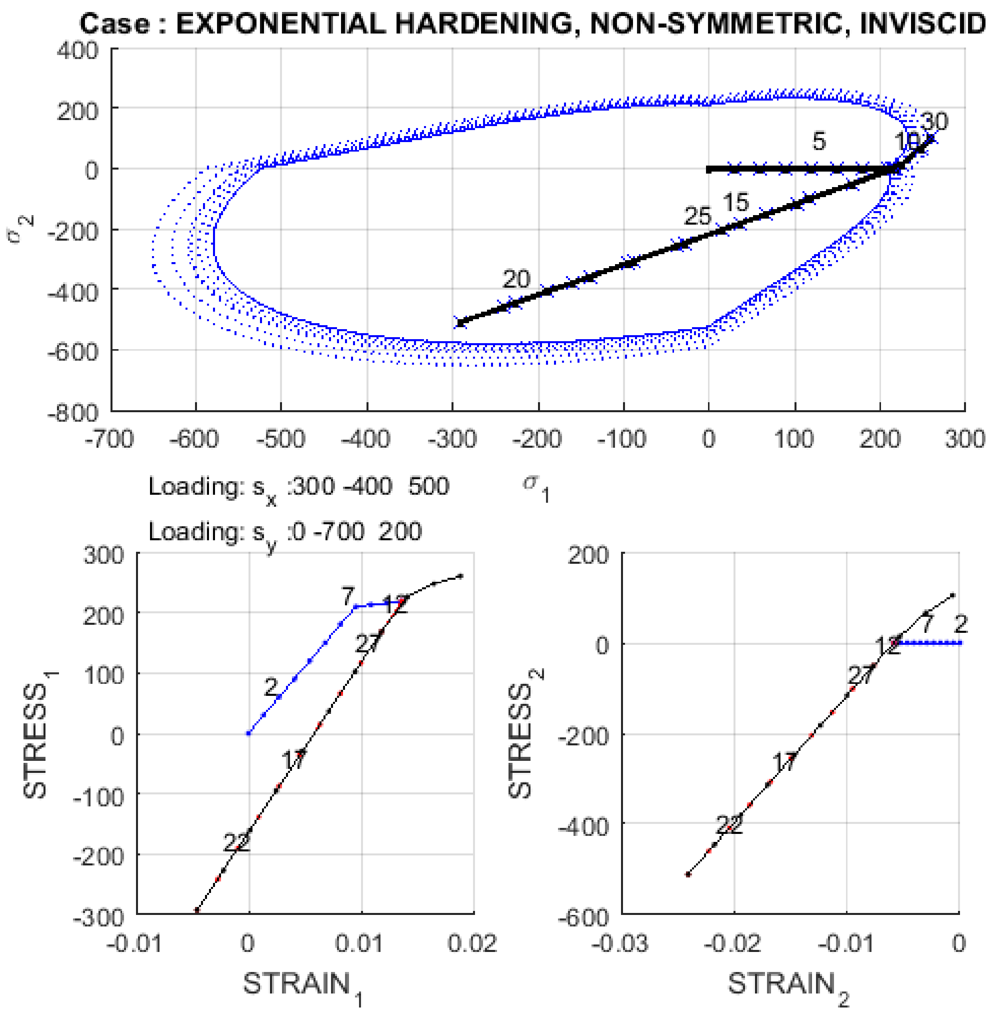

Figure 7: Bi-axial loading with Non-Symmetric Exponential Hardening

Figure 8 shows the exponential softening case for symmetric tensile/compression. Observations:

- Hardening parameter q and internal variable r , remain constant during elastic loading/unloading.
- Internal variable r , increases when the stress point is on the damage surface.
- Hardening variable, decreases when the stress point is on damage surface, because the model has exponential softening.
- Loss of linearity observed in stress-strain curves at all three evolution regions.

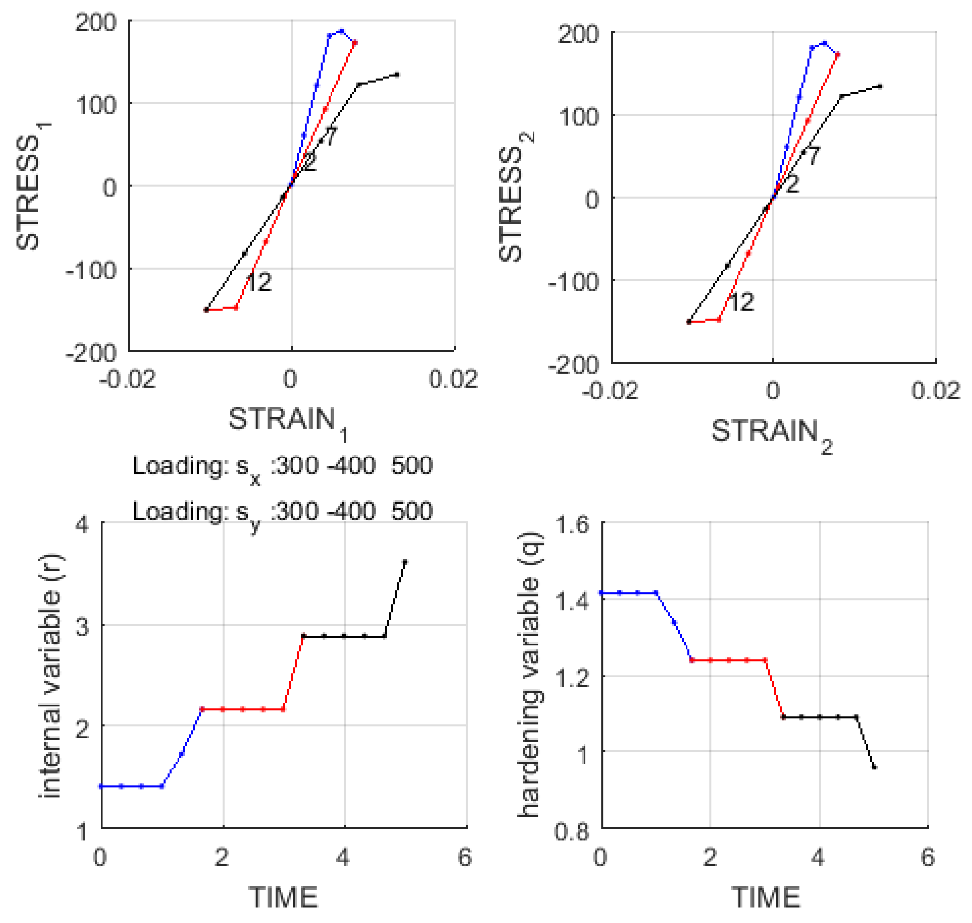

Figure 8: Exponential Softening case: Stress-Strain curves and Evolution of internal variable and hardening variables

COMPUTATIONAL SOLID MECHANICS

ASSIGNMENT 1

Kiran Sagar Kollepara
Computational Mechanics

Part 2: Visco-Damage Model

The algorithm for rate of evolution of damage variable r has been implemented. The correctness of the implementations will be checked for different values of η , α and $\dot{\epsilon}$.

Case 1:

Viscosity: $\eta=0.5$ and $\eta=5$. (Figure 1, 2 & 3).

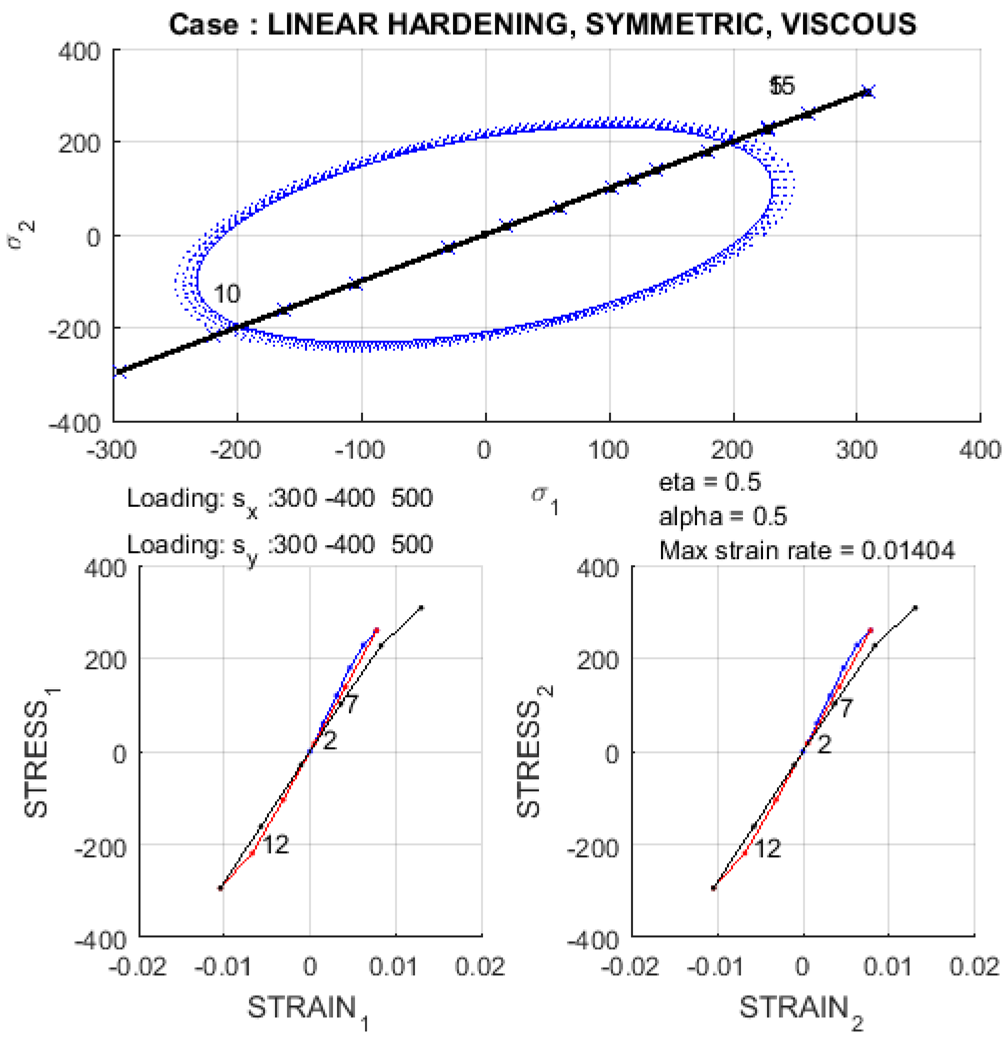

Figure 1: Loading-Unloading path for $\eta=0.5$

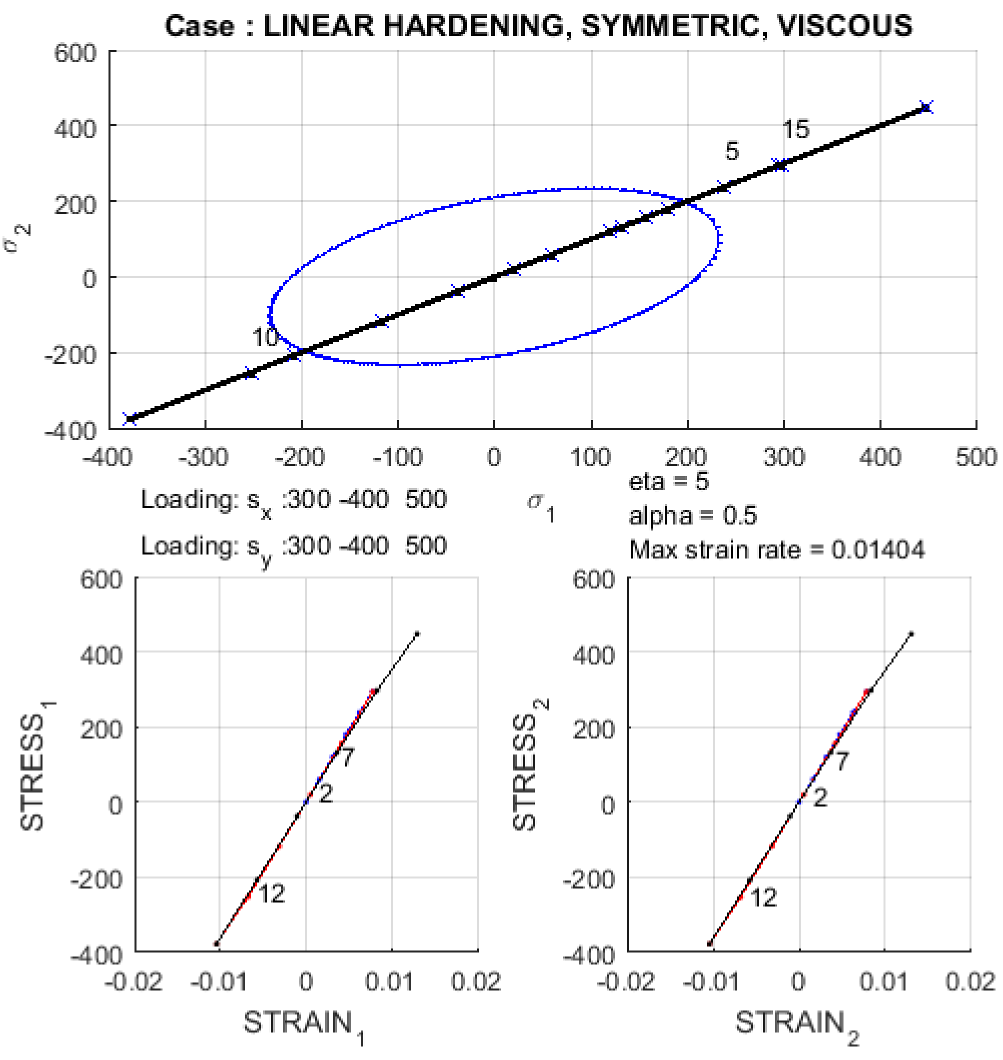

Figure 2: Loading/Unloading path for $\eta=5$

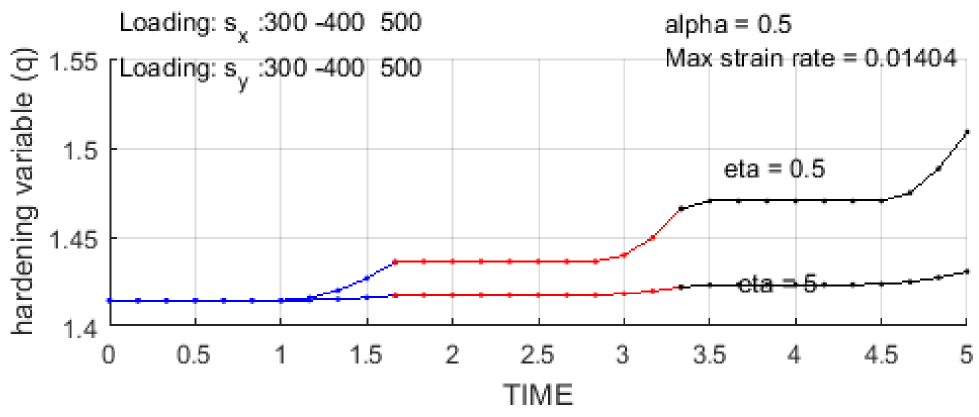

Figure 3: Hardening Variable (q *) vs. Time (* t *) for two different viscocities*

Observations: Figure 1,2 & 3)

- For $\eta=5$, the viscous effects are more visible, with the stress point moving far away from the damage surface, as compared to the stress curve of $\eta=0.5$.

- For $\eta=5$, the damage surface doesn't evolve much as compared to the $\eta=0.5$ case. This leads to a more linear Stress-Strain curve, because of the low damage.
- This is also evident in the q vs. Time plot, where q is higher for $\eta=0.5$.

Case 2:

Strain Rates: The efect of strain rates on evolution is studied in Figure 4 & 5. The same load steps are applied in different time steps.

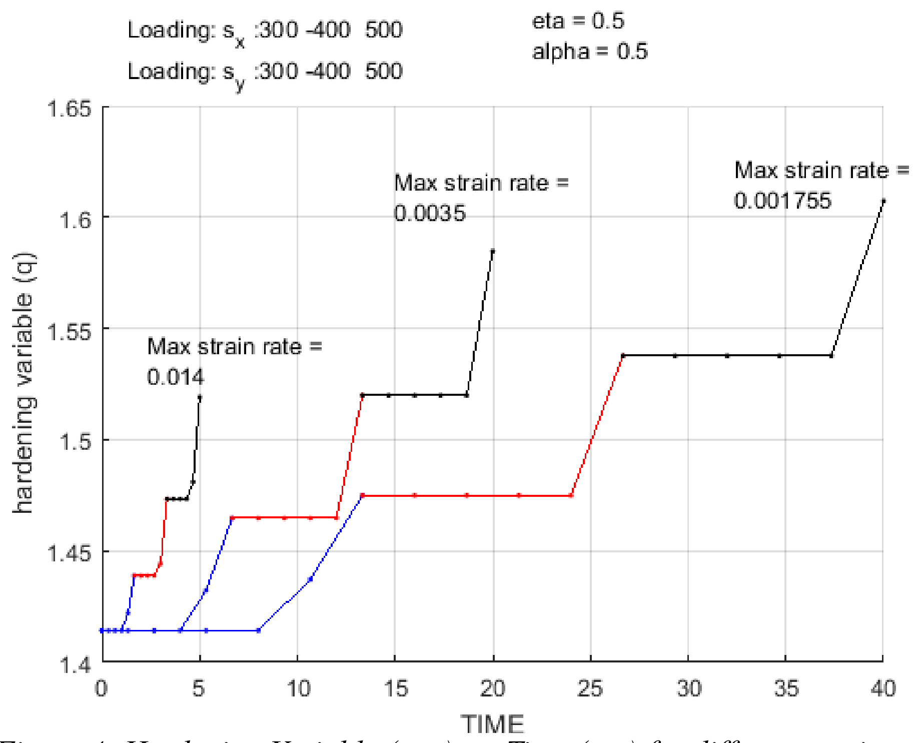

Figure 4: Hardening Variable (q) vs. Time (t) for different strain rates

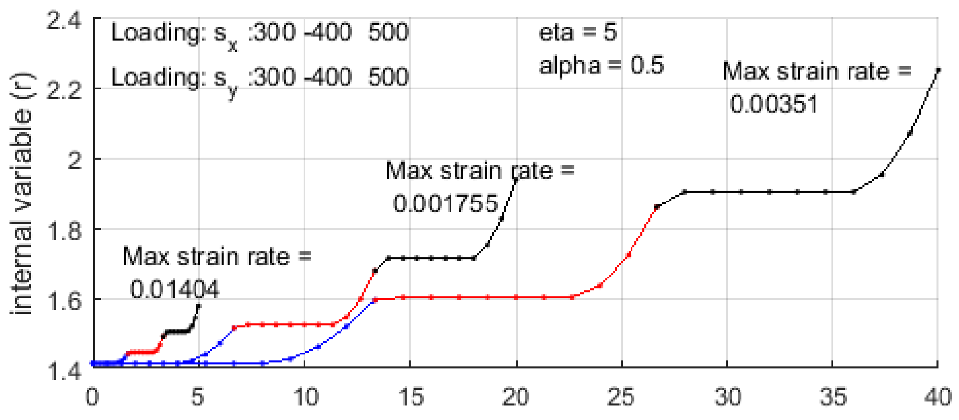

Figure 5: Internal Variable (r) vs. Time (t) for different strain rates

Observations:

- Lower strain rates have larger evolution of damage variables.
- Since loading is the same for all cases, strain rate decides how much time the stress point stays outside the damage surface. As the damage variable only evolves in this condition, the cases with lower strain rates evolve more.

Case 3:

$0<\alpha<1$

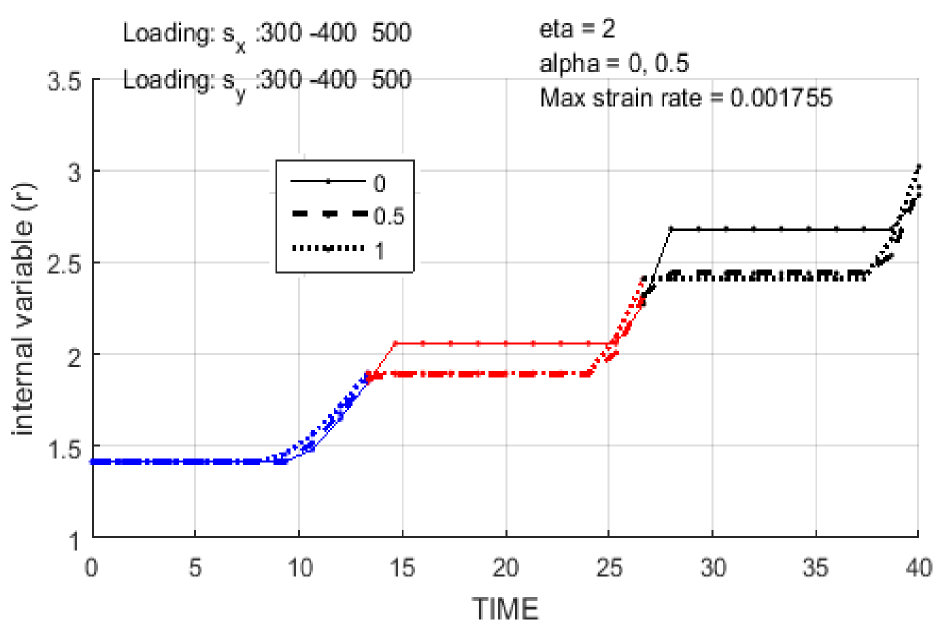

Figure 6: Evolution of internal variable over time

Observations:

- $\alpha=0$ over predicts the damage variable as compared to $\alpha=0.5$ & $\alpha=1$.
- For $\alpha=0$, the explicit nature of method delays the evolution of damage variable.

Modified Codes

Damage surface

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

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

elseif (MDtype==2)  %* Only tension
    sigma_n1 = ce*eps_n1' ;
    sigma_n1t = [sigma_n1(1)  sigma_n1(3) ;sigma_n1(3) sigma_n1(2) ] ;
    sigma_z = sigma_n1(4) ;
    [ev,ps] = eig(sigma_n1t) ;
    ps = (ps+abs(ps))/2 ;
    sigma_z = (sigma_z+abs(sigma_z))/2 ;
    sigma_n1t = ev*ps/ev ;
    sigma_n1 = [sigma_n1t(1,1) ; sigma_n1t(2,2) ; sigma_n1t(1,2) ;
sigma_z ] ;
    rtrial = sqrt(eps_n1*sigma_n1) ;

elseif (MDtype==3)  %*Non-symmetric
    sigma_n1 = ce*eps_n1' ;
    sigma_n1t = [sigma_n1(1)  sigma_n1(3) ;sigma_n1(3) sigma_n1(2) ] ;
    sigma_z = sigma_n1(4) ;
    [ev,ps] = eig(sigma_n1t) ;
    p_m = diag((ps+abs(ps))/2) ;
    theta = sum(p_m) / sum(diag(abs(ps))) ;
    sigma_z = (sigma_z+abs(sigma_z))/2 ;
    sigma_n1t = ev*ps/ev ;
    sigma_n1 = [sigma_n1t(1,1) ; sigma_n1t(2,2) ; sigma_n1t(1,2) ;
sigma_z ] ;
    rtrial = ( theta + (1-theta)/n ) * sqrt(eps_n1*sigma_n1) ;

end
return
```

Evolution Law

```
function [sigma_n1,hvar_n1,aux_var] = rmap_dano1
(eps_n1,eps_n,hvar_n,Eprop,ce,del_t,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) ;
viscpr       = Eprop(6) ;
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] = Modelos_de_dano1 (MDtype,ce,eps_n1,n);
[t_n] = Modelos_de_dano1 (MDtype,ce,eps_n,n);
%************************************************************************
*************


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

switch viscpr
    case 0
        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 = H*r0/(zero_q - r0);
                q_n1 = zero_q - (zero_q-r0)*exp(A*(1-r_n1/r0)) ;
            end
            if(q_n1<zero_q)
                q_n1=zero_q;
            end
        else
            %*     Elastic load/unload
            fload=0;
            r_n1= r_n  ;
            q_n1= q_n  ;

        end

    case 1

        rtrial = (1-alpha)*t_n + alpha*rtrial ;
        if(rtrial >= r_n)
            %*   Loading
            fload=1;
            r_n1 = (eta-del_t*(1-alpha))/(eta+alpha*del_t)*r_n +
del_t*rtrial/(eta+alpha*del_t) ;
            delta_r=r_n1-r_n;
            if hard_type == 0
                %  Linear
                q_n1= q_n+ H*delta_r;
            else
                A = H*r0/(zero_q - r0);
                q_n1 = zero_q - (zero_q-r0)*exp(A*(1-r_n1/r0)) ;
            end
            if(q_n1<zero_q)
```

```
                q_n1=zero_q;
            end

        else
            %*     Elastic load/unload
            fload=0;
            r_n1= r_n  ;
            q_n1= q_n  ;
        end
end
% Damage variable
% ---------------
dano_n1    = 1-(q_n1/r_n1);
%  Computing stress
%  ****************
sigma_n1  =(1-dano_n1)*ce*eps_n1';
ce_(1-dano_n1)*ce ;
%hold on
%plot(sigma_n1(1),sigma_n1(2),'bx')

%*************************************************************************
*************

%*************************************************************************
*************
%* Updating historic variables
%*
%  hvar_n1(1:4)  = eps_n1p;
hvar_n1(5)= r_n1 ;
hvar_n1(6)= q_n1 ;
%*************************************************************************
*************

%*************************************************************************
*************
%* Auxiliar variables
%*
aux_var(1) = fload;
aux_var(2) = q_n1/r_n1;
%*aux_var(3) = (q_n1-H*r_n1)/r_n1^3;
%*************************************************************************
*************
```

Damage Criterion

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

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

elseif (MDtype==2)  %* Only tension
    sigma_n1 = ce*eps_n1' ;
    sigma_n1t = [sigma_n1(1)  sigma_n1(3) ;sigma_n1(3) sigma_n1(2) ] ;
    sigma_z = sigma_n1(4) ;
    [ev,ps] = eig(sigma_n1t) ;
    ps = (ps+abs(ps))/2 ;
    sigma_z = (sigma_z+abs(sigma_z))/2 ;
    sigma_n1t = ev*ps/ev ;
    sigma_n1 = [sigma_n1t(1,1) ; sigma_n1t(2,2) ; sigma_n1t(1,2) ;
sigma_z ] ;
    rtrial = sqrt(eps_n1*sigma_n1) ;

elseif (MDtype==3)  %*Non-symmetric
    sigma_n1 = ce*eps_n1' ;
    sigma_n1t = [sigma_n1(1)  sigma_n1(3) ;sigma_n1(3) sigma_n1(2) ] ;
    sigma_z = sigma_n1(4) ;
    [ev,ps] = eig(sigma_n1t) ;
    p_m = diag((ps+abs(ps))/2) ;
    theta = sum(p_m) / sum(diag(abs(ps))) ;
    sigma_z = (sigma_z+abs(sigma_z))/2 ;
    sigma_n1t = ev*ps/ev ;
    sigma_n1 = [sigma_n1t(1,1) ; sigma_n1t(2,2) ; sigma_n1t(1,2) ;
sigma_z ] ;
    rtrial = ( theta + (1-theta)/n ) * sqrt(eps_n1*sigma_n1) ;

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