Master's Degree Numerical Methods in Engineering

Computational Solid Mechanics

Homework 2: Implementation of the 1D and J2 Plasticity Models

Author:
Luis Ángel Avilés Murcia
luis.angel.aviles@upc.edu

Professors:
Carlos, Agelet de Saracibar

May 1, 2020
Academic Year 2019-2020

Contents

1 Description

This report presents the implementation of two plasticity models. First, the 1D rate-independent and rate-dependent hardening plasticity models, including linear and nonlinear isotropic hardening and linear kinematic hardening. Second, the J2 plasticity model or known as Von Mises models in 3D. Two software were used to carry on the implementation. The 1D plasticity model was implemented using a Matlab code due to the simplicity and J2 model was implemented in a fortran code, following the syntax of a User Material (UMAT) used in Abaqus but at the level of a Gauss point. For this last task, a code called `IncrementalDriver.f`[1] was used. This code allow test models using a single gauss point.

2 Part I - 1D Plasticity Model

The 1D plasticity model was implemented in Matlab following the algorithm from the slides. The implementation is a strain drive implementation, which means that strain vector is known for any step and stresses and internal variables are computed and updated according with this strain vector.

2.1 Loading paths

In order to validate and assess the correctness of the implementation, the following strain path was used. High values of strains are not necessary because the material to be tested is steel, which have high stress to low strains values.

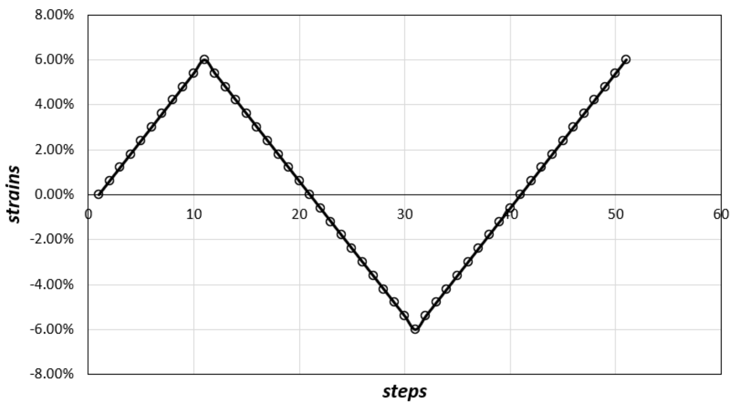


Figure 1: Strain path used to test the model

2.2 Material parameters

Three sets of material parameters will be used to assess the correctness of the implementation: a reference value, a lower value (under the reference) and a higher value (below of the reference value).

Parameter	Ref value	Value min	Value max
Young Modulus	100 MPa	50 MPa	125 MPa
Isotropic hardening modulus	50 MPa	20 MPa	75 MPa
Kinematic hardening modulus	50 MPa	20 MPa	75 MPa
Yield stress	1.2 MPa	0.5 MPa	2.0 MPa
Viscosity parameter	0.5 MPa	0.25 MPa	1.0 MPa
delta coefficient	0.5	0.3	1.0
Infinite stress	2.5 MPa	2.5MPa	2.5 MPa

Table 1: Material parameters

2.3 Numerical simulations

The following simulations were carry on using the implemented algorithm.

- Perfect Plasticity
- Linear isotropic hardening plasticity
- Nonlinear isotropic hardening plasticity with exponential saturation law
- Linear kinematic hardening plasticity
- Nonlinear isotropic and linear hardening plasticity

2.4 Results

- **Perfect Plasticity**

For this case, all hardening parameters are set to zero.

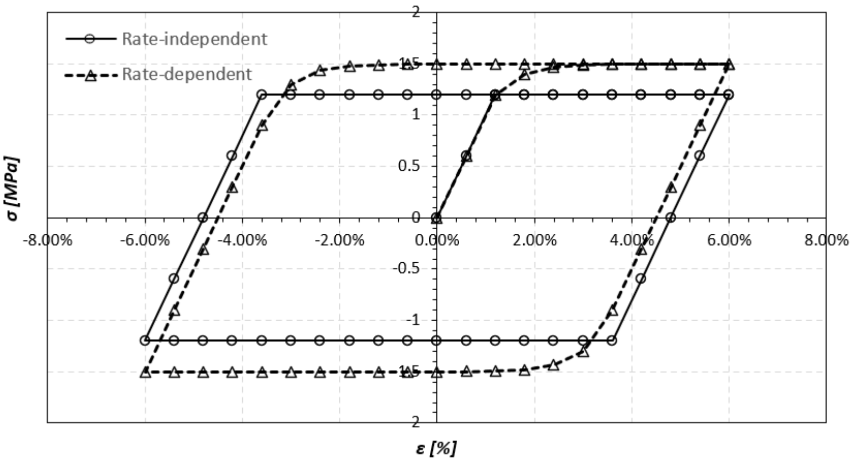
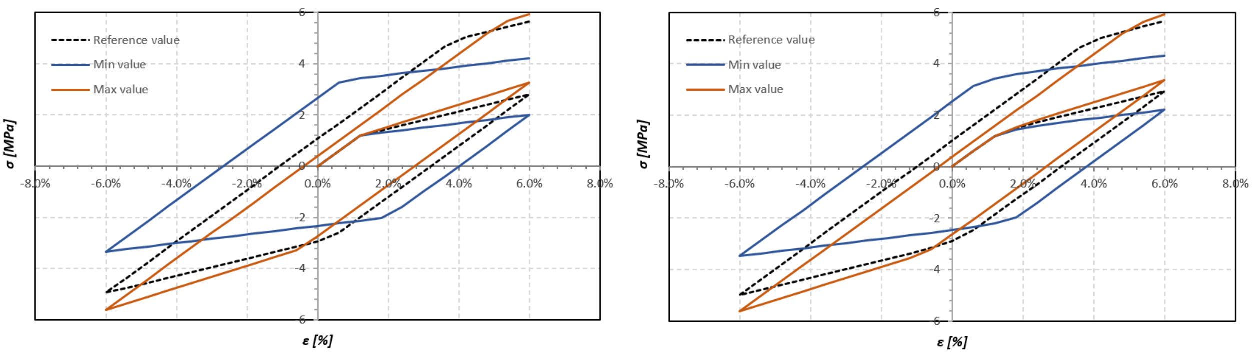


Figure 2: Perfect plasticity curves for rate-independent and rate-dependent

It can be seen that for rate-dependent materials, the stress response is higher than for rate-independent. It is because beyond the yield stress the material stiffness increase as plasticity have place. At the same time, for rate-dependent material the transition from elastic to plastic is smoother than for rate-independent.

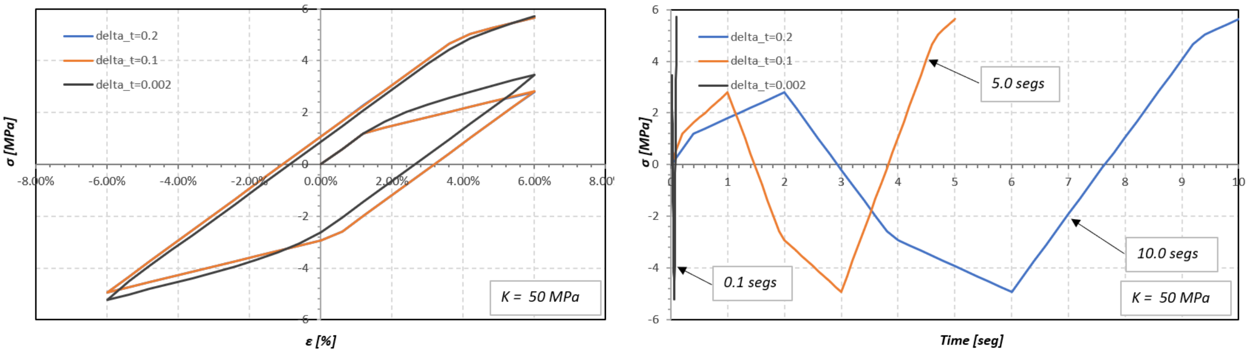
- **Linear isotropic hardening plasticity**



(a) Rate-independent (b) Rate-dependent

Figure 3: Isotropic hardening, variation of stress with strain rate

As the isotropic hardening modulus increase, the stress response increase becoming the material capable of wide the elastic regimen of the load.

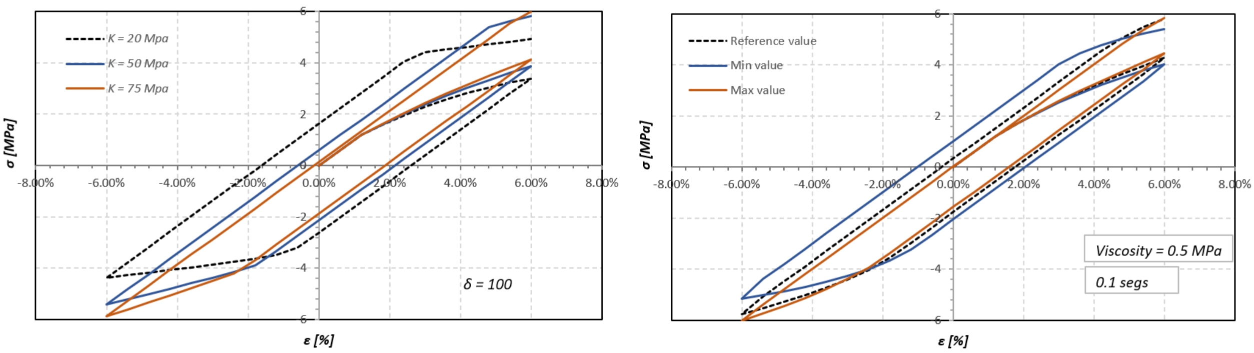


(a) Stress-strain (b) Stress-time

Figure 4: Isotropic hardening, variation of stress with strain rate

From figure 4 it can be seen that to rate-dependent material, for smaller time (lower rate strain) results in stiffer response.

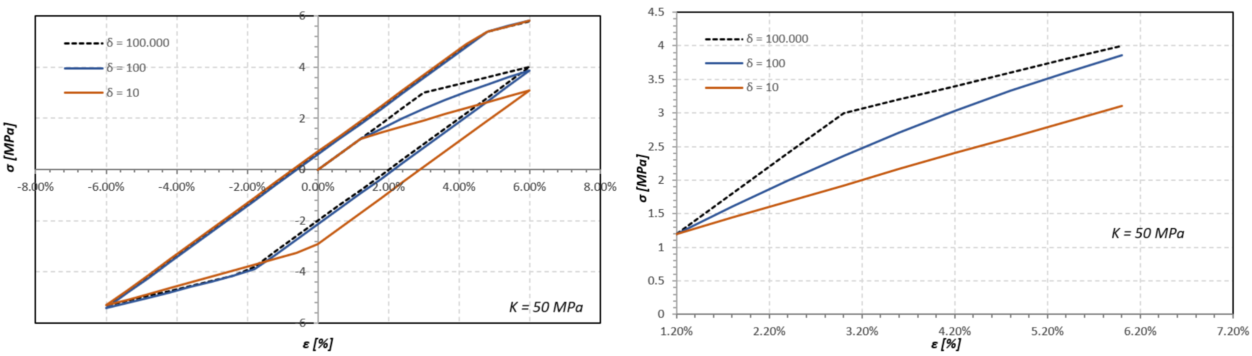
- **Nonlinear isotropic hardening plasticity**



(a) Rate independent (b) Rate dependent

Figure 5: Nonlinear isotropic hardening, variation of isotropic hardening modulus

When the material becomes rate-dependent, the curves shrink along the cycle of load but the strength increase is not significant.



(a) Rate dependent (b) Rate dependent

Figure 6: Variation of Nonlinear isotropic hardening with delta coefficient

The influence of the delta coefficient over the stiffness of the material is more marked for higher values of this parameter. Very high values of delta expand the elastic regime, lower values present linear development of the hardening.

Figure 6b shows the evolution of the stress according to an exponential law, however, because of the isotropic and kinematic hardening, the behavior seems linear and it extends beyond the infinite yield stress because of the hardening itself.

The following figure shows the influence of delta parameter over the stress response. It can see that for high values of the delta parameter more quickly the stress reach the yield infinite stress given by the material.

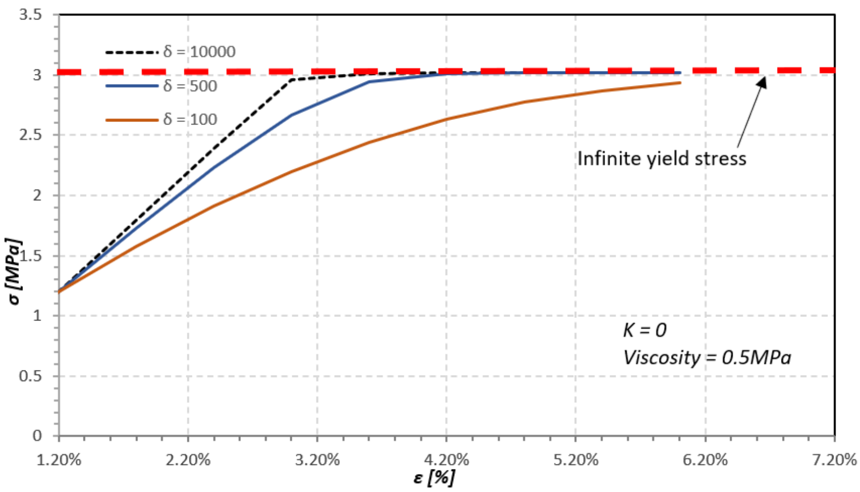
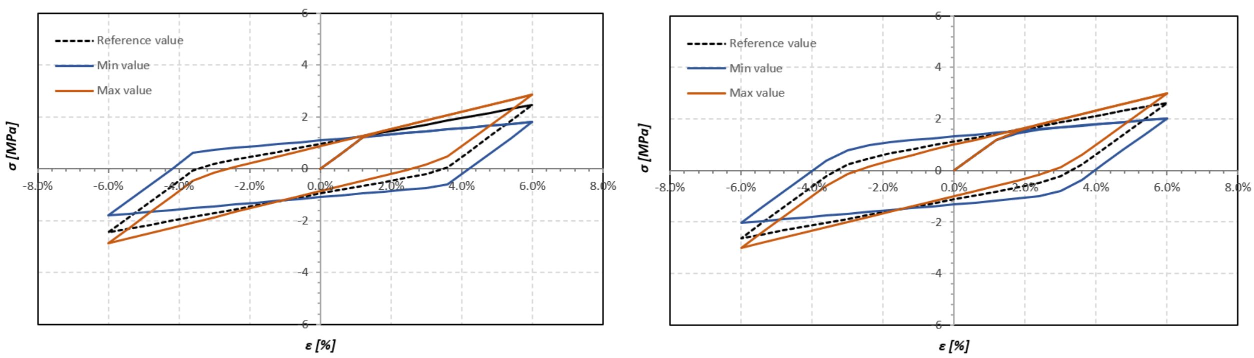


Figure 7: Influence of the delta parameter on the stress response in nonlinear plasticity behaviour

- **Linear kinematic hardening plasticity**

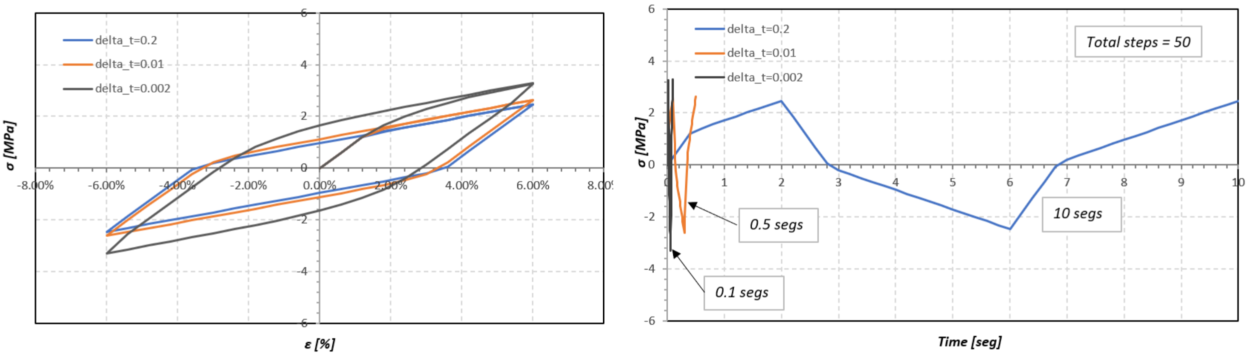


(a) Rate independent

(b) Rate dependent

Figure 8: Kinematic hardening, variation of kinematic hardening modulus

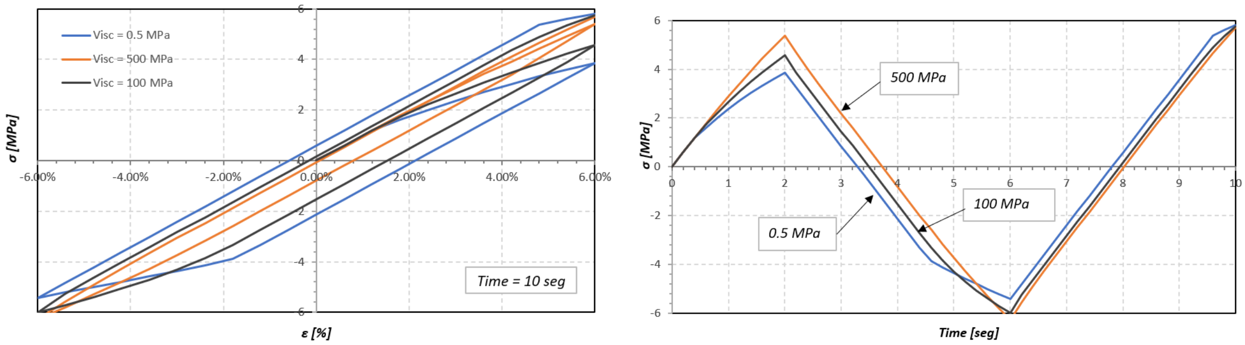
The increase of the kinematic hardening is not as high as isotropic hardening, which means isotropic hardening is more relevant to increase the resistance of materials and expand the elastic regime once the yield state has been overcome.



(a) Stress-strain (b) Stress-time

Figure 9: Kinematic hardening, variation of stress with strain rate

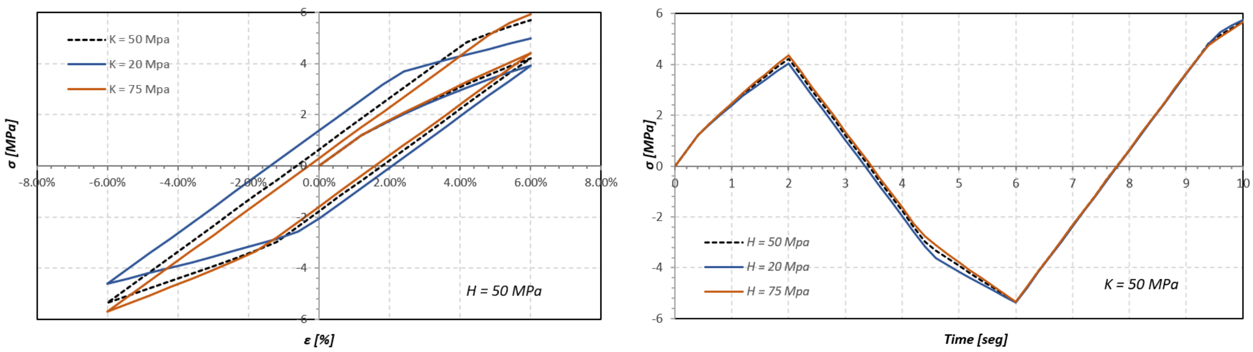
Shorter time in the load application means the material can resist more load. Physically, this is due because there is not time to material experiment a redistribution of the load through the domain. Just a very small time, or strain rate, present significant changes in the material response, from a certain time (in this case approximately 1 seg) the rate-strain does no have a relevant effect on the material response, it can be seen because the almost overlapping in the stress-strain in 9a.



(a) Stress-strain (b) Stress-time

Figure 10: Influence of viscosity on the stress-strain response

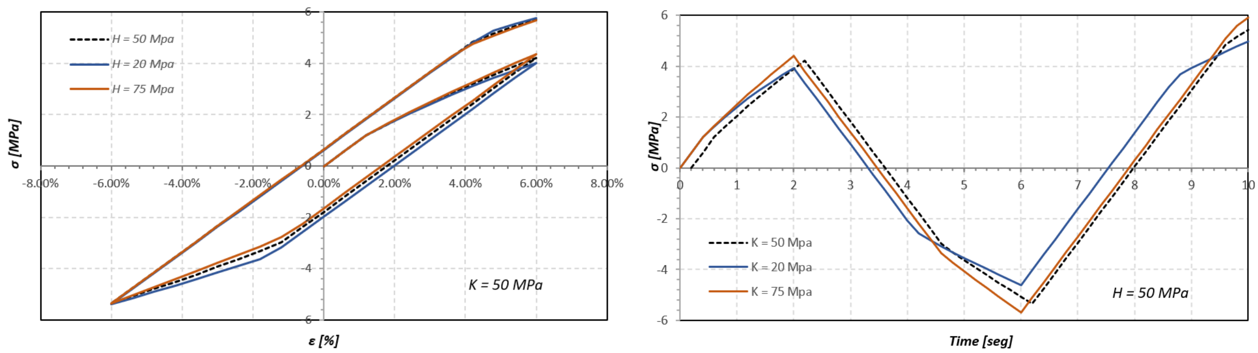
- **Nonlinear isotropic and linear kinematic hardening plasticity**



(a) Isotropic variation (b) Variation isotropic hardening with time

Figure 11: stress-strain response to nonlinear isotropic and linear kinematic hardening, rate-dependent

The nonlinear behavior is not to clear when materials have hardened, this cause that material becomes stiffener and do not show very clear the exponential law of hardening.



(a) Kinematic variation

(b) Variation kinematic hardening with time

Figure 12: stress-strain response to nonlinear isotropic and linear kinematic hardening, rate-dependent

The figures showed before corresponding to rate-dependent materials, however, rate-independent material behaves similarly. The smooth transition in the stress-strain curves seen between rate-dependent and rate-independent is not to clear when the material has both isotropic and kinematic hardening.

2.5 Conclusions Part I

- The rate-dependent response increase the yield surface and apply a smoothening to the transition between elastic and plastic behavior.
- The linear isotropic parameter, K, play a role more significant in the hardening of the material than the kinematic modulus. Isotropic hardening modulus increase the slope for the plastic deformation and expand the elastic domain faster after each cycle.
- Time or rate-strain have a influence on the behaviour of the material for small values of time. At high values of time the material response in stress-strain does not change.
- For the nonlinear isotropic hardening, the exponential coefficient δ just affects the velocity with which the yield surface is reached, for high values, faster increase.
- As for isotropic hardening, for linear kinematic plasticity, greater values of the kinematic hardening parameter, H, produce that the plastic part of the stresses increase faster.
- The main difference between isotropic and kinematic cycle response is that for isotropic stress-strain the cycle remains open once the strain has complete the loop. For kinematic response, once the strain cycle is complete, the curves look closed.

3 Part II - J2 Plasticity Model

As mentioned before, J2 model is known as Von Mises model. This was implemented in Fortran using the code Incremental Driver, which is used to test material models at a gauss point level. The implementation was done following the algorithm given in class for rate-dependent model. The rate-independent model is a particular case of the model with viscosity equal to zero. Because this is a 3D model, some tensorial and vectorial operations are needed to operate with some stress or strains tensor, these vectorial operations were implemented as subroutines inside the code.

3.1 Material parameters

The following table shows the parameters used to evaluate the correctness of the J2 model implementation.

Parameter	Ref value	Value min	Value max
Young Modulus	20000 MPa	-	-
Isotropic hardening modulus	2000 MPa	1000 MPa	3000 MPa
Kinematic hardening modulus	3000 MPa	1500 MPa	4500 MPa
Yield stress	500 MPa	250 MPa	1200 MPa
Viscosity parameter	500 MPa	250 MPa	1000 MPa
delta coefficient	25	10	100
Infinite stress	2500 MPa	-	-

Table 2: Material parameters

All the state variables (internal variables) are zero at the beginning of the test. A triaxial initial condition was set with 100 MPa in all the three principal directions.

3.2 Loading path

In order to validate and assess the correctness of the implementation, the following strain path was used. The maximum strain is 10% in both compression and extension.

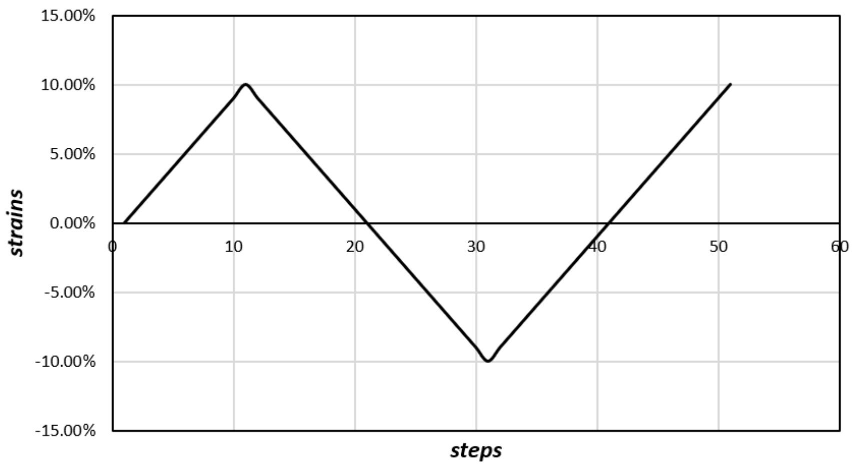


Figure 13: Strain path used to test the J2 model

The total time in the Figure is just orientative because a longer time can be used with the implementation.

3.3 Results

- **Perfect Plasticity**
  For this case, isotropic and kinematic hardening modulus are set to zero. In addition, to consider the rate effects, the mean value of viscosity was considered (500 MPa).

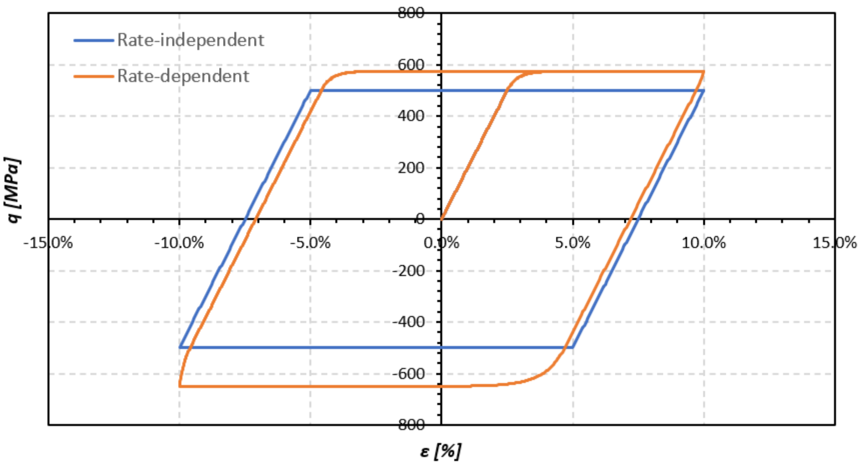
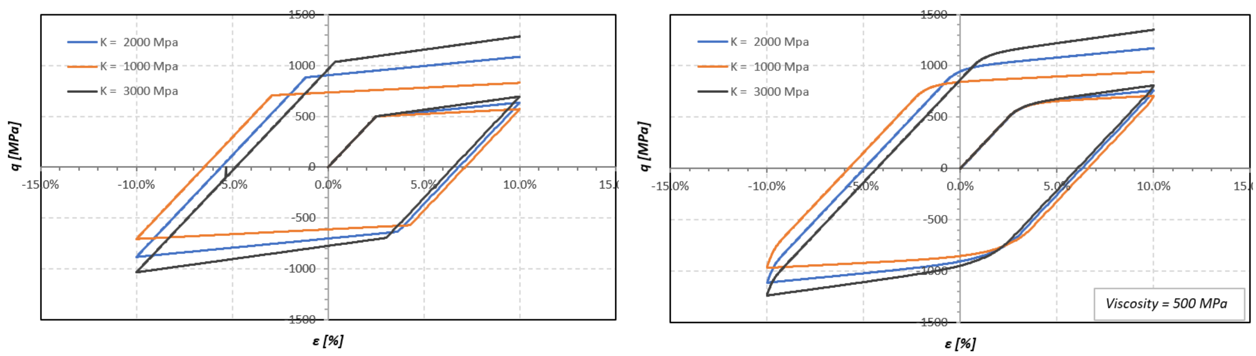


Figure 14: Perfect plasticity curves for rate-independent and rate-dependent

In all the stress-strain figures, the y-axis corresponds to the deviatoric stress (q). The behavior is very similar to 1D plasticity.

- **Linear isotropic hardening plasticity**
  In this case, the isotropic hardening modulus takes different values to see the influence of the parameter in the material response. A linear evolution of the hardening is considered.



(a) Rate-independent (b) Rate-dependent

Figure 15: Linear variation of stress-strain response with isotropic hardening modulus

As the isotropic hardening modulus increase, the stress response increase becoming the material capable of wide the elastic regimen of the load. At the same time, it is seen as the rate-dependent effect smooth the changes in the path of the cyclic load.

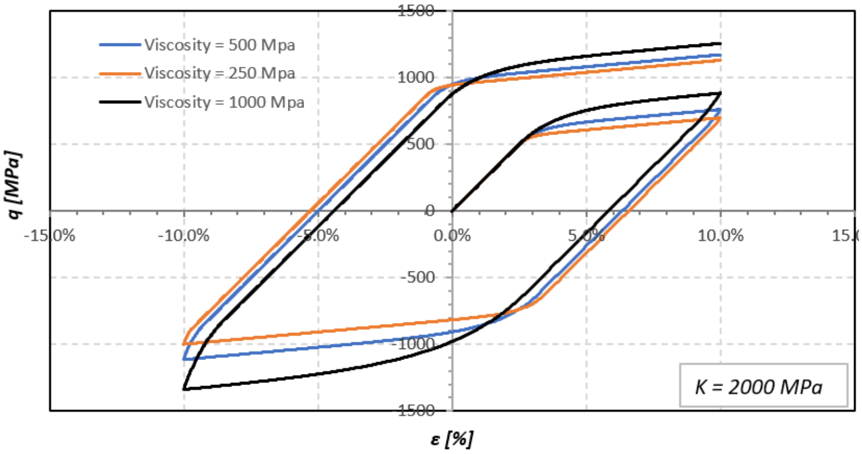
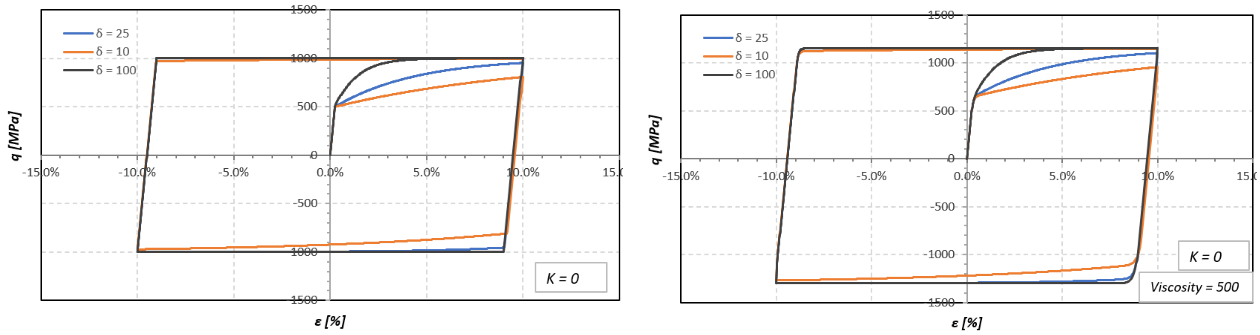


Figure 16: Influence of the viscosity on the stress-strain response for isotropic hardening

Higher values of viscosity increase the stiffness of the material. It can be seen as an additional term that stiffens the material during the load.

- **Nonlinear isotropic hardening plasticity**
  In order to see well the nonlinear behavior, the isotropic hardening modulus (K) was set to zero.



(a) Rate-independent

(b) Rate-dependent

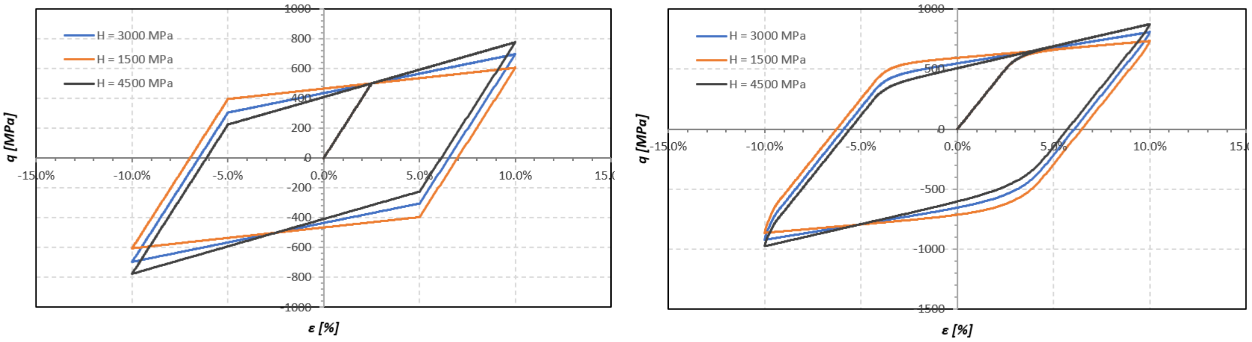
Figure 17: Nonlinear variation of stress-strain response with isotropic hardening modulus

It can be seen how the stress trend to 1000 MPa which is the value of the parameter infinite stress. High values of the parameter delta carry to a faster reach of the infinity yield stress value. Once the infinite stress value has been reached, the remaining parts of the load path behave as perfect plasticity when reaching the new yield stress.

The viscosity effect makes smooth the changes of direction on the load path. For both cases, it can be appreciated that not increment in the yield surface happen over the infinity yield stress, this is because the isotropic hardening parameter is zero.

- **Linear kinematic hardening plasticity**

Now the isotropic hardening modulus is zero and the kinematic hardening modulus takes different values according with the table of material parameters.



(a) Rate-independent (b) Rate-dependent

Figure 18: Variation of stress-strain response with kinematic hardening modulus

In kinematic hardening light changes in the hardening, modulus does not make a great change in the stress-strain response as in isotropic hardening. High values of kinematic hardening trend to close or reduce the area created by the curves.

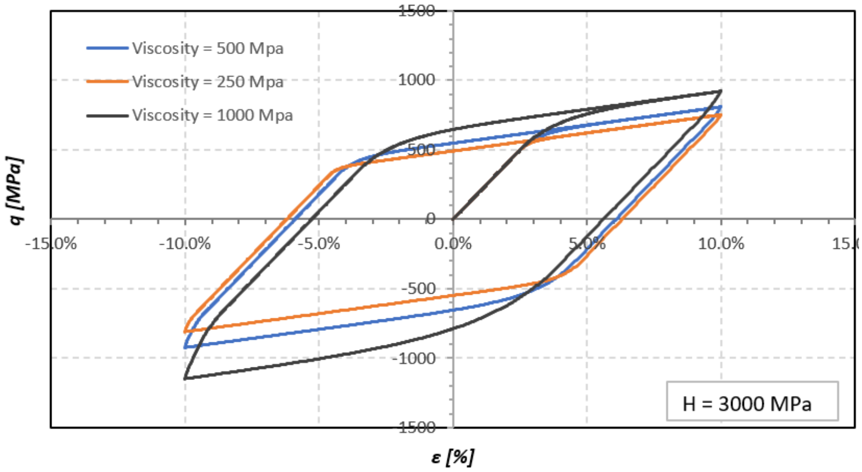
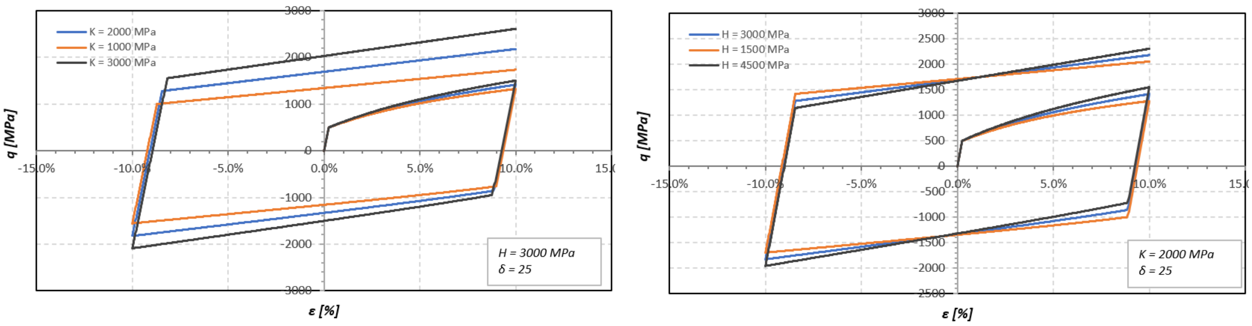


Figure 19: Influence of the viscosity on the stress-strain response for kinematic hardening

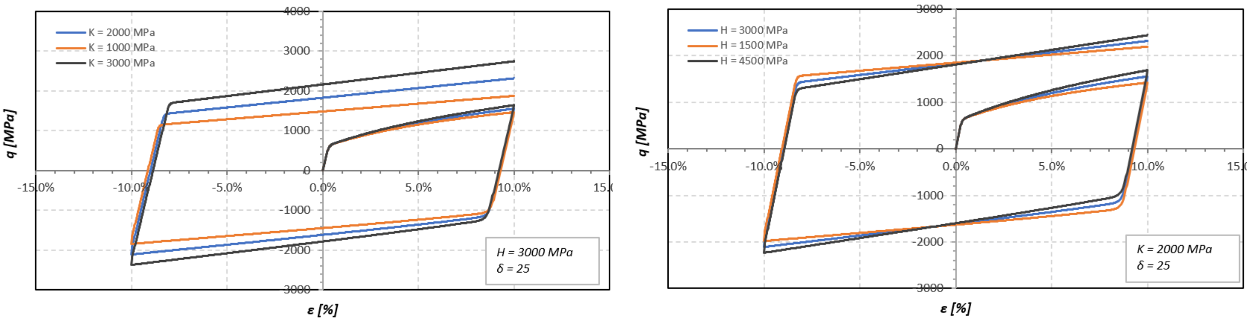
- **Nonlinear isotropic and linear kinematic hardening plasticity**



(a) Variation isotropic hardening (b) Variation kinematic hardening

Figure 20: Nonlinear isotropic and linear kinematic hardening, rate-independent behavior

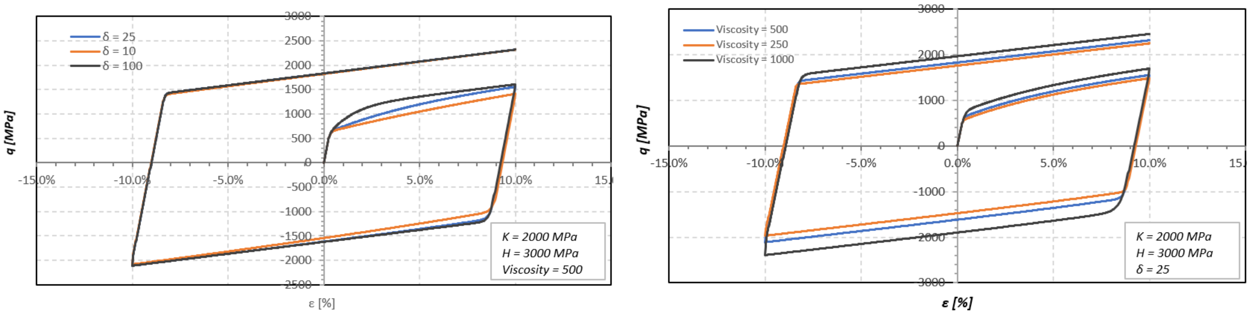
Changes in isotropic hardening modulus are more representative in the stress-strain response of the material, it increases the strength of the material, especially for the unloading and reloading part of the load cycle.



(a) Variation isotropic hardening (b) Variation kinematic hardening

Figure 21: Nonlinear isotropic and linear kinematic hardening, rate-dependent behavior

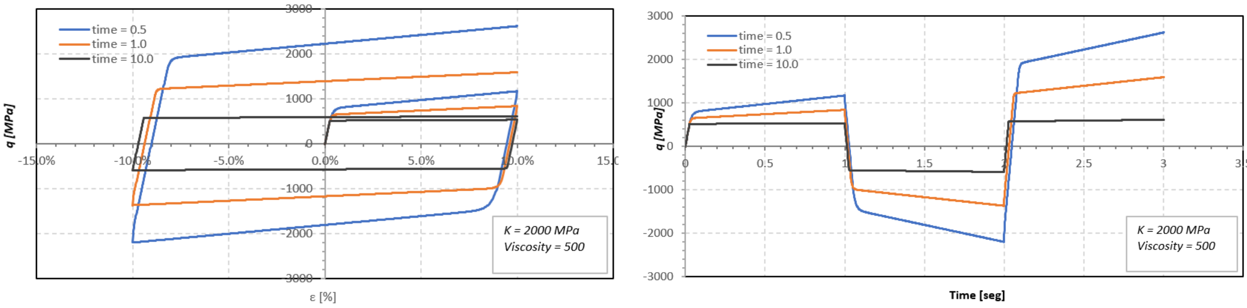
There are not huge differences between rate-independent and rate-dependent materials, at least at the level of stress analyzed in this work.



(a) Influence of delta parameter (b) Influence of viscosity

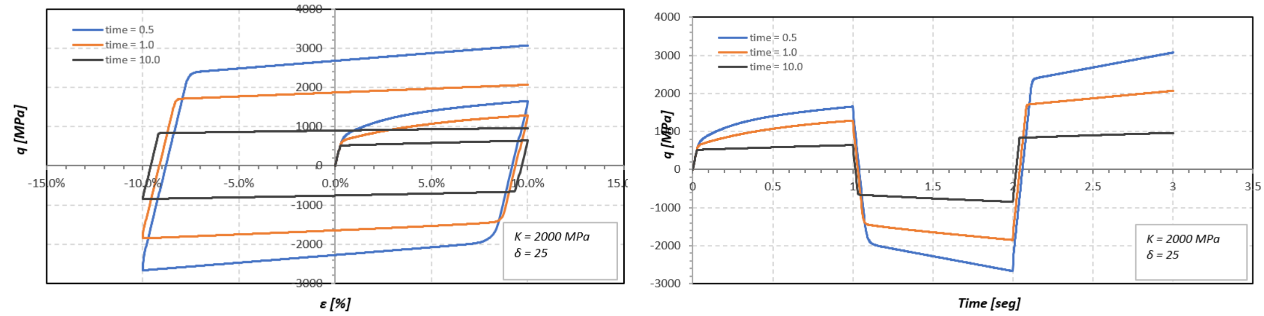
Figure 22: Nonlinear isotropic and linear kinematic hardening, influence of delta parameter and viscosity

- **Influence of the time (rate strain)**
  The influence of the time or rate-strain was considered changing the time between each strain increment. The following figures shows the stress-strain response and the stress-time behaviour for three time increments, for linear and nonlinear isotropic hardening.



(a) Stress-strain (b) Stress-time

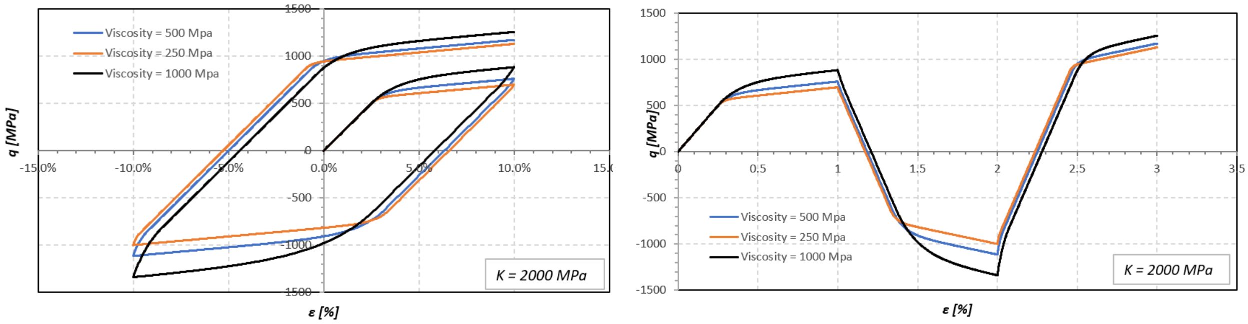
Figure 23: Linear rate-dependent isotropic hardening with time variation



(a) Stress-strain (b) Stress-time

Figure 24: Nonlinear rate-dependent isotropic hardening with time variation

It can be seen that the difference in the stress-strain response between Linear and Nonlinear behavior of the isotropic hardening is not significant when the time changes. The change in time or rate-strain controls the behavior of the material completely. A little increase or reduction of the time means an appreciable change in the load response.



(a) Stress-strain (b) Stress-time

Figure 25: Nonlinear rate-dependent isotropic hardening with time variation

The changes in the material response with viscosity changes are not too relevant like changes relate to time. Figure 25 shows that even when viscosity increase in 100 percent, changes are not to huge as when time increases a little.

3.4 Conclusions Part II

- Isotropic hardening have a more relevant effect in the strength of the materials.
- Lower rate-strains (lower time) increase the stiffness of the material and consequently their strength.
- High values of the delta parameter cause that materials reach the infinite stress value faster.
- Viscosity increment can be seen as an additional property that increases the stiffness of materials and smoothes the transition from loading to unloading and vice versa.

A Appendix

A.1 Plasticity_main 1D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implementation of 1D plasticity model
% Perfect plasticity, isotropic hardening and kinematic hardening
% Linear and Non linear hardening
% Writen by: Luis Angel Aviles Murcia
% Computational Solid Mechanics
% Master degree on numerical methods
% Professor: Carlos Agelet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all

%% Material properties
E_mod = 100E6;       % Eprop(1)  Young Modulus Pa
K_mod = 75E6;       % Eprop(2)  Isotropic hardening modulus in Pa
H_mod = 0E6;       % Eprop(3)  Kinematic hardening modulus in Pa
sigma_y = 1.2E6;    % Eprop(4)  Yield stress in Pa
viscosity = 0.5E6;    % Eprop(5)  Viscosity parameter
sigma_inf = 3.0E6;  % Eprop(6)  Sigma infinity used in exponential law
delta = 0.1E3;         % Eprop(7)  delta <- Parameter for exponential law
hard_law = 1;       % Eprop(8) Hardening law 0 = linear 1 = exponential
Eprop   = [E_mod K_mod H_mod sigma_y viscosity sigma_inf delta hard_law];

%% Strain path (cycle)
timeTotal = 10.0;   % time per path
nloadstates = 3 ;
npaths = 5;         % tramos de ruta de strain
noCycles = 1;

%% Strain points (just one load cycle)
strain(1) =6*10^-2;     % the strain vector start in 0
strain(2) =-6*10^-2;    % minimum point of strain vector
strain(3) =6*10^-2;     % last point of the strain vector

%% Number of time increments for each npath
% ---------------------------------------
istep = 10;                   % increments for each path
totalSteps = npaths*istep;  % total time

%% Initialisation strain vector
strainVector = zeros(totalSteps+1,1);
for i=1:istep
strainVector(i+1) = strain(1)*i/istep;
strainVector(11+i) = strain(1)-strain(1)*i/istep;
strainVector(21+i) = strain(2)*i/istep;
strainVector(31+i) = strain(2)+strain(1)*i/istep;
strainVector(41+i) = strain(3)*i/istep;
end

%% Initialisation time vector for rate dependent
timeVector = zeros(totalSteps+1,1) ;
delta_t = timeTotal/totalSteps;
for i=2:totalSteps+1
    timeVector(i) = timeVector(i-1)+delta_t;
end

%% Initialisation of the plastic state (internal variables)
eps_plas = zeros(totalSteps+1,1);   % epsilon plastico
Xi       = zeros(totalSteps+1,1);   % isotropic internal variable
Xibar    = zeros(totalSteps+1,1);   % kinematic internal variable
sigma    = zeros(totalSteps+1,1);   % stress vector
q        = zeros(totalSteps+1,1);   % isotropic hardening
qbar     = zeros(totalSteps+1,1);   % kinematic hardening
plastic_state  = zeros(totalSteps+1,1);
ce = E_mod;                         % elastic modulus


```

```

%% Problem solution

for i=2:totalSteps+1

   timeVector(i) = timeVector(i-1)+delta_t;

   %****** Solution of the model *****************
   [stress,eps_plasn1,Xi_n1,Xibar_n1,plastic_state(i)] = plasticity_one(eps_plas(i-1),Xi(i
    -1),Xibar(i-1),strainVector(i),Eprop,i,delta_t);

   %****** Updating variables for next step ******
   sigma(i)     = stress(1);
   q(i)         = stress(2);
   qbar(i)      = stress(3);
   eps_plas(i)  = eps_plasn1;
   Xi(i)        = Xi_n1;
   Xibar(i)     = Xibar_n1;

   % Saving data to plot externally
   %printResults(X,T,elemType,elementDegree,h);
end
% Printing variables in a .txt file to print data
printResults(K_mod,H_mod,viscosity,delta,sigma,q,qbar,strainVector,eps_plas,totalSteps,
    timeVector)
```

A.2 Function plasticity_one

```
function [stress,Eplas_n1,Xi_n1,Xibar_n1,plas_sta] = plasticity_one(Eplas,Xi,Xibar,strain,
    Eprop,i,delta_t)
% Time-stepping algorithm for a 1D hardening plasticity model
%
% Inputs:
% Eplas = epsilon plastico
% Xi = isotropic internal variable
% Xibar = hardening internal variable
% strain = vector with strain
% Eprop = material properties
%
% Outputs:
%  stress = sigma stress for next step n+1 (contains sigma, q, qbar)
%  Eplas_n1 = epsilo plastic for next step n+1
%  Xi_n1     = isotropic hardening variable (scalar), step n+1
%  Xibar_n1  = kinematic hardening variable (scalar), step n+1
%
%  plas_sta = [isplastic]
%  isplastic       variable that is 1 if plastic case or 0 otherwise
%*****************************************************************************
%
Eplas_n     = Eplas;
Xi_n        = Xi;
Xibar_n     = Xibar;
strain_n1   = strain;

E_mod    = Eprop(1);         % Young Modulus
K_mod    = Eprop(2);         % Isotropic hardening parameter
H_mod    = Eprop(3);         % Kinematic hardening parameter
sigma_y  = Eprop(4);         % Yield stress
h_law    = Eprop(8);         % Hardening Law 0=linear, 1=exponential
viscosity     = Eprop(5);    % Viscocity
sigma_inf = Eprop(6);        % Sigma infinity
delta = Eprop(7);            % delta


%************** Compute the trial state for step n+1 *********************
sigma_trial_n1 = E_mod*(strain_n1 - Eplas_n); %stress

%*** Isotropic hardening variable
if (h_law==0)   % Linear hardening law
    q_trial_n1 = -K_mod*Xi;
else            % Exponential hardening law
```

```
    q_trial_n1 = -(sigma_inf-sigma_y)*(1-exp(-delta*Xi))-K_mod*Xi;
end

%*** Kinematic hardening variable
qbar_trial_n1 = -(2/3)*H_mod*Xibar_n;

%*** Trial yield function
ftrial_n1 = abs(sigma_trial_n1 - qbar_trial_n1) - (sigma_y - q_trial_n1);

%**********************************************************************
% Definition of time step
if (viscosity==0) % Rate independent plasticity
    delta_t = 1;
% else
%     delta_t = Eprop(10); % Time step
end
% *********************************************************************
isplastic = 0; % start with elastic state

% **************************Plasticity algorithm*************************
if(ftrial_n1 > 0) % plastic state
    isplastic = 1;

    %*  Computing plastic multiplier
    if (h_law == 0) % Linear isotropic hardening
        gamma_n1 = ftrial_n1/(delta_t*(E_mod + K_mod + H_mod + (viscosity/delta_t))); %
    plastic multiplier

    else            % Exponential isotropic hardening (non linear)

        % Solve the equation using Newton_Raphson Algorithm
        % Initialize variables
        k = 0;
        gamma_n1 = 0;
        g_n1 = 100; % residual value
        % Solve Equation
        while ((g_n1 > 0.001) && (k < 100))

         % isotropic hardening with Xi slide 57
         if (h_law==0)   % Linear hardening law
         qXi = -K_mod*Xi_n;
         else            % Exponential hardening law
         qXi = -(sigma_inf-sigma_y)*(1-exp(-delta*Xi_n)) - K_mod*Xi_n;
         end

         % isotropic hardening with Xi+gamma_n1*delta_t slide 57
         if (h_law==0)   % Linear hardening law
         qXi_delta = -K_mod*(Xi_n+gamma_n1*delta_t);
         PI_2der = -K_mod;
         else            % Exponential hardening law
         qXi_delta = -(sigma_inf-sigma_y)*(1-exp(-delta*(Xi_n+gamma_n1*delta_t))) - K_mod
    *(Xi_n+gamma_n1*delta_t);
         PI_2der = -delta*(sigma_inf-sigma_y)*exp(-delta*(Xi_n+gamma_n1*delta_t)) - K_mod;
         end

         %[mPI12,mPI22] = pot_der(xi_n + gamma_n1*delta_t,Prop,h_law);
         g_n1 = ftrial_n1 - gamma_n1*delta_t*(E_mod + H_mod + viscosity/delta_t) - (qXi -
    qXi_delta);
         deltag_n1 = -(E_mod - PI_2der + H_mod + viscosity/delta_t)*delta_t;
         gamma_n1 = gamma_n1 - (g_n1/deltag_n1);
         k = k + 1;

         if (k == 100)
          fprintf('Maximum number of iterations exceded %d. \n',i)
         end
        end

    end

    % Return mapping algorithm
    sigma_n1=sigma_trial_n1-gamma_n1*delta_t*E_mod*sign(sigma_trial_n1-qbar_trial_n1);
```

```
    if (h_law==0)    % Linear hardening law
         mPI1 = -K_mod*(Xi_n+gamma_n1*delta_t);
    else             % Exponential hardening law
         mPI1 = -(sigma_inf-sigma_y)*(1-exp(-delta*(Xi_n+gamma_n1*delta_t))) - K_mod*(Xi_n
    +gamma_n1*delta_t);
    end

    if (h_law==0)    % Linear hardening law
         mPI2 = -K_mod*(Xi_n);
    else             % Exponential hardening law
         mPI2 = -(sigma_inf-sigma_y)*(1-exp(-delta*(Xi_n))) - K_mod*(Xi_n);
    end

    q_n1 = q_trial_n1 + (mPI1 - mPI2);
    qbar_n1 = qbar_trial_n1 + gamma_n1*delta_t*H_mod*sign(sigma_trial_n1 - qbar_trial_n1);

    % Update plastic internal variables database at time n+1
    Eplas_n1 = Eplas_n + gamma_n1*delta_t*sign(sigma_trial_n1 - qbar_trial_n1);
    Xi_n1 = Xi_n + gamma_n1*delta_t;
    Xibar_n1 = Xibar_n - gamma_n1*delta_t*sign(sigma_trial_n1 - qbar_trial_n1);

    % Compute the consistent elastoplastic tangent operator


else
    %*     Elastic load/unload
    sigma_n1 = sigma_trial_n1;

    if (h_law==0)    % Linear hardening law
         q_n1 = -K_mod*(Xi_n);
    else             % Exponential hardening law
         q_n1 = -(sigma_inf-sigma_y)*(1-exp(-delta*(Xi_n))) - K_mod*(Xi_n);
    end

    qbar_n1 = qbar_trial_n1;
    Eplas_n1 = Eplas_n;
    Xi_n1 = Xi_n;
    Xibar_n1 = Xibar_n;
end

%*************************************************************************************
% Outputs for next step
stress(1) = sigma_n1;
stress(2) = q_n1;
stress(3) = qbar_n1;
plas_sta(1)= isplastic;
```

A.3 Function Print_results

```
function []= printResults(K_mod,H_mod,viscosity,delta,sigma,q,qbar,strain,eps_plas,
    totalSteps,time)

fileID = fopen('results.txt','w');
fprintf(fileID,'%6s %6i\n','K_mod = ', K_mod);
fprintf(fileID,'%6s %6i\n','H_mod = ', H_mod);
fprintf(fileID,'%6s %6i\n','Visco = ', viscosity);
fprintf(fileID,'%6s %6i\n','delta = ', delta);
fprintf(fileID,'%6s          %6s           %6s           %6s            %6s     %6s\n' ,'
    Sigma', 'iso_q', 'kin_qbar','strain','strain_plast','time'); %printing number of nodes
for i = 1:totalSteps+1
    fprintf(fileID,'%6E     %6E     %6E     %6E     %6E     %6E\n',sigma(i),q(i),qbar(i),
    strain(i),eps_plas(i),time(i));
    %fprintf(fileID,'%12E\n', h(I));
end


fprintf(fileID,'\n');
fprintf(fileID,'\n');

```

```
fclose(fileID);
end
```

A.4 J2 Code main

```
*USER SUBROUTINES
C      Heading of UMAT
       SUBROUTINE UMAT(STRESS,STATEV,DDSDDE,SSE,SPD,SCD,
      1 RPL,DDSDDT,DRPLDE,DRPLDT,
      2 STRAN,DSTRAN,TIME,DTIME,TEMP,DTEMP,PREDEF,DPRED,CMNAME,
      3 NDI,NSHR,NTENS,NSTATEV,PROPS,NPROPS,COORDS,DROT,PNEWDT,
      4 CELENT,DFGRD0,DFGRD1,NOEL,NPT,LAYER,KSPT,KSTEP,KINC)
C
       USE subrutinas
       INCLUDE 'ABA_PARAM.INC'
C -----------------------------------------------------------------
C      Declarating UMAT vaiables and constants
       CHARACTER*80 CMNAME
       DIMENSION STRESS(NTENS),STATEV(NSTATEV),
      1 DDSDDE(NTENS,NTENS),DDSDDT(NTENS),DRPLDE(NTENS),
      2 STRAN(NTENS),DSTRAN(NTENS),TIME(2),PREDEF(1),DPRED(1),
      3 PROPS(NPROPS),COORDS(3),DROT(3,3),DFGRD0(3,3),DFGRD1(3,3),
      4 MAT(2,2)
! ------------------------------------------------------------------
! --------------variables usadas en el programa-------------------
       REAL*8 T(3,3),deps(3,3),void,lambda,JAC(3,3,3,3),aiso,akin(3,3)
      1 ,delt33(3,3),v,E,jac66(6,6),Nu,T0(3,3),D(3,3),Kc,G, Viscosity
      2 ,N(3,3),Idev(3,3,3,3),iso4(3,3,3,3),Iun(3,3,3,3),luis(3,3)
      3 ,dgamma,B_f(3,3),B_ft(3,3),H,Ty,T_tr(3,3),normet
      4 ,eta_tr(3,3),f_tr,Ce(3,3,3,3),Cep(3,3,3,3),ddeps(3,3),qiso_tr
      5 ,Tdev_tr(3,3),eta(3,3),Tnex_dev(3,3),a2,a1,qkin_tr(3,3),qkin(3,3)
      6 ,Ivol(3,3,3,3),akin_tr(3,3),aiso_tr,deps_tr(3,3),Tdev(3,3),g_n1
      6 ,countk, qiso, q_prime1, q_prime2, isoType,delta_t, b1, b2
      6 ,deltag_n1, qXi, qXi_delta, qXi_2prime
! ------------------------------------------------------------------
! ----------------Lectura de Parametros iniciales-----------------
       v=props(1)
       E=props(2)
       H=props(3)
       Ty=props(4)
       K=props(5)
       isoType=props(6) ! 0 = Linear; 1 = Non-linear
       sigma_inf=props(7)
       deltaParam=props(8)
       Viscosity = props(9) ! viscosity for rate dependent plasticity
       delta_t = 10.0 ! Use 1.0 when K |=0
! ----------------------Variables de Estado----------------------
       void=statev(1)
       aiso=statev(2)       ! isotropic hardening evolution
       akin(1,1)=statev(3) ! kinematic hardening evolution
       akin(2,2)=statev(4)
       akin(3,3)=statev(5)
       akin(1,2)=statev(6)
       akin(1,3)=statev(7)
       akin(3,2)=statev(8)
       akin(3,1)=statev(6)
       akin(2,1)=statev(7)
       akin(2,3)=statev(8)
! -----------------------------------------------------------------
       call Initial(STRESS,T, DSTRAN, DEPS, NTENS,NDI, NSHR)
       call D1(DSTRAN, D, dtime, NDI, NSHR, NTENS)
! -----------------------------------------------------------------
       if (v==0.5) then
       v=0.49999999
       endif
! ---------------------Calculo de Constante-----------------------
       Kc=E/(3.0d0*(1.0d0-2.0d0*v))              !Modulo volumetrico
       G=E/(2.0d0*(1.0d0+v))                     !Modulo de corte
       lambda=(v*E)/((1.0d0+v)*(1.0d0-2.0d0*v))  !Constante de LAME
       Nu=E/(2.0d0*(1.0d0+v))                    !Constante de LAME
! ------------------------------------------------------------------
```

```
! --------------Calculo de tensor Elastico de 4to orden-----------
      call Iunit(Iun)
      call Idesvi(Idev,iso4,Ivol)
      Ce=lambda*Iun+(2.0d0*Nu)*iso4
! ----------------------------------------------------------------
!     Trial Steps
      if (isoType==0) then
          qiso_tr=-K*aiso                              ! Scalar value
      else
          qiso_tr=-(sigma_inf-Ty)*(1-exp(-deltaParam*aiso))-K*aiso
      endif

      qkin_tr=-(((2.0d0/3.0d0)*H*Idev).doble.akin) ! Tensorial value
      T_tr=T+(Ce.doble.deps)                       ! Trial tensor  T = sigma
      Tdev_tr=(Idev.doble.T_tr)                    ! 2do order tensor
      Eta_tr=Tdev_tr+qkin_tr                       ! 2do order tensor (parte superior de la norma n)
      normet=nrm(eta_tr)
! ----------------------------------------------------------------
! -----------------------Funcion de fluencia----------------------
      f_tr=nrm(eta_tr)-sqrt(2.0d0/3.0d0)*(Ty-qiso_tr) ! trial yield function

! -----------------------Condicion de fuencia---------------------
      if (f_tr.lt.0)then           !El paso de prueba elastica esta bien
      T=T_tr                       !Tensor 2do orden
      aiso=aiso                    !Escalar
      deps=deps
      Cep=Ce
      statev(3)=akin(1,1)
      statev(4)=akin(2,2)
      statev(5)=akin(3,3)
      statev(6)=akin(1,2)
      statev(7)=akin(1,3)
      statev(8)=akin(3,2)
      statev(2)=aiso
      JAC=Cep
      call Solution(NTENS, NDI, NSHR, T, STRESS, JAC, DDSDDE)
      else                         !Se aplica corrector plastico (hay plasticidad)

          if (isoType==0) then !Linear case for isotropic hardening (include viscosity)

              !dgamma=f_tr/(2.0d0*G+(2.0d0/3.0d0)*(H+K)) !gamma_n+1
              if (Viscosity==0) then
               gamma_n1=f_tr/((2.0d0*G+(2.0d0/3.0d0)*(H+K))) !gamma_n+1
              else
               gamma_n1=f_tr/(delta_t*(2.0d0*G+(2.0d0/3.0d0)*(H+K)
     &                  +(Viscosity/delta_t))) !gamma_n+1
              end if


          else ! Exponential isotropic hardening (include viscosity)
              ! gamma_n+1 using Newton Raphson
              k=0
              gamma_n1=0
              g_n1 = 100

              do while ((g_n1>0.001).and.(k<100))


                  qXi = -(sigma_inf-Ty)*(1-exp(-deltaParam*aiso))-K*aiso

                  qXi_delta = -(sigma_inf-Ty)*(1-exp(-deltaParam
     &            *(aiso+sqrt(2.0d0/3.0d0)*gamma_n1*delta_t)))
     &               -K*(aiso+sqrt(2.0d0/3.0d0)*gamma_n1*delta_t)

                 qXi_2prime = -deltaParam*(sigma_inf-Ty)*exp(-deltaParam
     &            *(aiso+sqrt(2.0d0/3.0d0)*gamma_n1*delta_t))-K

                 ! Calculation of g function and gamma step n+1

                  g_n1 = f_tr - gamma_n1*delta_t*(2*G+(2.0d0/3.0d0)*H
```

```
     &              +(Viscosity/delta_t))-sqrt(2.0d0/3.0d0)
     &             *(qXi-qXi_delta)

                deltag_n1 = -(2*G+(2.0d0/3.0d0)*qXi_2prime
     &             +(2.0d0/3.0d0)*H+(Viscosity/delta_t))*delta_t

                gamma_n1 = gamma_n1 - (g_n1/deltag_n1)

                k=k+1
                !if (k==100)then
                !    write(,*),'Maximum number of iterations exceeded'
                !endif

            end do


         endif

            !****************** Return Mapping Algorithm*****************!
            eta = eta_tr/nrm(eta_tr)
            B_f=eta                    ! Vector de flujo plastico
           ! ---------------------------------------------------------------
! ---------------- Updating state variables -----------------------
            ddeps=gamma_n1*B_f           ! Incrementos de deformaciones plasticas
            deps=deps+ddeps              ! Deformaciones totales
            aiso=aiso+sqrt(2.0d0/3.0d0)*gamma_n1
            akin=akin+gamma_n1*B_f
            statev(3)=akin(1,1)
            statev(4)=akin(2,2)
            statev(5)=akin(3,3)
            statev(6)=akin(1,2)
            statev(7)=akin(1,3)
            statev(8)=akin(3,2)
            statev(2)=aiso
            ! sigma n+1
            if (Viscosity==0) then
                T = T_tr - gamma_n1*2*G*B_f
            else
                T = T_tr - gamma_n1*delta_t*2*G*B_f
            endif


            ! q_n+1
            if (isoType==0) then
              q_prime1 = - K*(aiso+gamma_n1*delta_t*sqrt(2.0d0/3.0d0))
              q_prime2 = - K*aiso
            else
                q_prime1 = -(sigma_inf-Ty)
     &   *(1-exp(-deltaParam*(aiso+gamma_n1*delta_t*sqrt(2.0d0/3.0d0))))
     &             -K*(aiso+gamma_n1*delta_t*sqrt(2.0d0/3.0d0))
                q_prime2 = -(sigma_inf-Ty)*(1-exp(-deltaParam*aiso))
     &             -K*aiso
            endif

            qiso = qiso_tr + (q_prime1-q_prime2) !gamma_n1*dtime*sqrt(2.0d0/3.0d0)*K

            ! qbar_n+1
            qkin = qkin_tr + gamma_n1*delta_t*(2.0d0/3.0d0)*(H)*B_f

! ----------------Calculo de proximo esfuerzo----------------------
            Tdev=Tdev_tr-(Ce.doble.ddeps)  !ddeps=delta epsilon
            !T=T_tr-(Ce.doble.ddeps)

! ----------------Modulo Elastoplastico consistente----------------
            call transpuesta(B_f,B_ft)
            b1=1-((2*G*gamma_n1*delta_t)/(nrm(eta_tr)))
            b2=2*G/(2*G+(2.0d0/3.0d0)*(H+K)+(Viscosity/delta_t))
     &          - (1-b1)
            Cep = Kc*Iun+2*G*b1*Idev-2*G*b2*(B_f.diad.B_ft)
            JAC=Cep
            call Solution(NTENS, NDI, NSHR, T, STRESS, JAC, DDSDDE)
```

```
      endif
      END SUBROUTINE UMAT
```

A.5 J2 Subroutines

```
! Initial conditions
    6   ntens
 -100    stress(1)    T11
 -100    stress(2)    T22
 -100    stress(3)    T33
  0.0    stress(4)    T12
  0.0    stress(5)    T13
  0.0   stress(ntens) T23
  8 nstatv number of state variables
  1.0 statev(1) evoid
  0.0 statev(2) aiso
    0 statev(3) akin(1,1)
    0 statev(4) akin(2,2)
    0 statev(5) akin(3,3)
    0 statev(6) akin(1,3)
    0 statev(7) akin(1,2)
    0 statev(8) akin(3,2)

! Parameters
  9   nprops
    0.3      props(1) v_Poisson
    200000   props(2) E_Elastic Module
  000   props(3) H_
  500    props(4) Ty_esfuerzo de fluencia
  2000   props(5) K
  1   props(6) linear or Non-linear
  1000  props(7) sigma_infinity
  25.0  props(8) delta parameter
  500000.00 props(9) viscosity

!     Modulo de Subprogramas
      MODULE subrutinas
      DOUBLE PRECISION delta(3,3)
      INTEGER i,j,k,l
      Public delta
      data delta/1.0d0,0.0d0,0.0d0,0.0d0,1.0d0,0.0d0,0.0d0,0.0d0,1.0d0/
!     ---------------------------------------------------------------
!     DECLARACION DE LOS OPERADORES DE LAS FUNCIONES A UTILIZAR
!     ---------------------------------------------------------------
      INTERFACE tr
      MODULE PROCEDURE traz
      END INTERFACE
!     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      INTERFACE operator(.doble.)
      MODULE PROCEDURE doble22,doble42
      END INTERFACE
!     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      INTERFACE nrm
      MODULE PROCEDURE norma
      END INTERFACE
!     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      INTERFACE operator(.diad.)
      MODULE PROCEDURE diada22
      END INTERFACE

!     ---------------------------------------------------------------
      contains
!     ---------------------------------------------------------------
!     %%%%%%%%%%%%%FUNCIONES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
!     ---------------------------------------------------------------
!     ------------------------------------------------
!     TRAZA DE UNA MATRIZ DE 3X3
!     ------------------------------------------------
      function traz(a) result (b)
      real*8, intent(IN):: a(3,3)
```

```
      real*8:: b
      b=a(1,1)+a(2,2)+a(3,3)
      return
      end function traz
!     ----------------------------------------------------
!     NORMA DE UN TENSOR DE SEGUNDO ORDEN DEFINIDA EN FUNCION DE SU DOBLE CONTRACCION
!     ----------------------------------------------------
      function norma(a) result(b)  !ESTA FUNCION NO QUIERE SERVIRME, REVISARLA
      real*8, intent(in):: a(3,3)
      real*8:: b
      b=sqrt(a(1,1)*a(1,1)+a(1,2)*a(1,2)+a(1,3)*a(1,3)+a(2,1)*a(2,1)
     & +a(2,2)*a(2,2)+a(2,3)*a(2,3)+a(3,1)*a(3,1)+a(3,2)*a(3,2)
     & +a(3,3)*a(3,3))
      endfunction norma
!     ----------------------------------------------------------
!     DOBLE CONTRACCION DE TENSORES DE 2DO ORDEN
!     ----------------------------------------------------------
      function doble22(a,b) result(c)
      real*8, intent(in), dimension(3,3)::a,b
      real*8:: c
      c=0.0d0
      do i=1,3
      do j=1,3
      c=c+a(i,j)*b(i,j)
      enddo
      enddo
      end function doble22
!     ----------------------------------------------------------
!     DOBLE CONTRACCION DE TENSORES DE 4tO ORDEN CON 2dO ORDEN
!     ----------------------------------------------------------
      function doble42(a,b) RESULT (c)
      DOUBLE PRECISION, INTENT(IN):: a(3,3,3,3),b(3,3)
      double precision c(3,3)
      do i=1,3
      do j=1,3
      c(i,j)=a(i,j,1,1)*b(1,1)+
     1 a(i,j,1,2)*b(1,2)+
     2 a(i,j,1,3)*b(1,3)+
     3 a(i,j,2,1)*b(2,1)+
     4 a(i,j,2,2)*b(2,2)+
     5 a(i,j,2,3)*b(2,3)+
     6 a(i,j,3,1)*b(3,1)+
     7 a(i,j,3,2)*b(3,2)+
     8 a(i,j,3,3)*b(3,3)
      enddo
      enddo
      return
      end function doble42
!    ------------------------------------------------------------
!    PRODUCTO DIADICO DE DOS TENSORES DE SEGUNDO ORDEN
!    ------------------------------------------------------------
      function diada22(a,b) result(c)
      real*8, intent(in):: a(3,3),b(3,3)
      real*8 c(3,3,3,3)
      integer i,j,k,l
      c=0.0d0
      do i=1,3
      do j=1,3
      do k=1,3
      do l=1,3
      c(i,j,k,l)=c(i,j,k,l)+a(i,j)*b(k,l)
      enddo
      enddo
      enddo
      enddo
      end function diada22
!     ----------------------------------------------
!     PRODUCTO PUNTO DE 2 VECTORES SIN USO DEL DK
!     ----------------------------------------------
      subroutine ppunto (a,b,c)
      real a(3),b(3),c
```

```
      do i=1,3
      c=c+a(i)*b(i)
      enddo
      end subroutine ppunto
!     ------------------------------------------------
!     PRODUCTO PUNTO DE 2 VECTORES CON DK
!     ------------------------------------------------
      subroutine ppunto2 (a,b,w)
      real u(3),v(3),c,delta(3,3)
      integer i,j
      w=0.0d0
      do i=1,3
      do j=1,3
      w=w+u(i)*v(j)*delta(i,j)
      enddo
      enddo
      end subroutine ppunto2
!     ------------------------------------------------
!     TRAZA DE UNA MATRIZ DE 3X3
!     ------------------------------------------------
      subroutine traza (a,c)
      real a(3,3),c
      do i=1,3
      do j=1,3
      c=a(1,1)+a(2,2)+a(3,3)
      end do
      end do
      end subroutine traza
!     ------------------------------------------------
!     TRANSPUESTA DE UNA MATRIZ DE 3X3
!     ------------------------------------------------
      subroutine transpuesta(a,b)
      real*8 a(3,3),b(3,3)
      integer i,j
      do i=1,3
      do j=1,3
      b(i,j)=a(j,i)
      enddo
      enddo
      end subroutine transpuesta
!     -------------------------------------------------
!     MULTIPLICACION DE MATRICES
!     -------------------------------------------------
      subroutine mmulti (a,b,c)
      real*8 a(3,3),b(3,3),c(3,3)
      integer i,j,k
      do i=1,3
      do j=1,3
      c(i,j)=0.0d0
      do k=1,3
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
      enddo
      enddo
      enddo
      endsubroutine mmulti
!     -------------------------------------------------
!     SUMA DE MATRICES
!     -------------------------------------------------
      subroutine msum (a,b,c)
      real*8 a(3,3),b(3,3),c(3,3)
      integer i,j,k
      do i=1,3
      do j=1,3
      c(i,j)=a(i,j)+b(i,j)
      enddo
      enddo
      end subroutine
!     ----------------------------------------------------
!     SUBRUTINA DE LA ADJUNTA DE UN TENSOR DE 2DO ORDEN
!     ----------------------------------------------------
      subroutine adjunta (A,Adj)
```

```fortran
      real*8 A(3,3), Adj(3,3)
      integer i,j,k
      Adj(1,1)=A(2,2)*A(3,3)-A(2,3)*A(3,2)
      Adj(1,2)=-(A(2,1)*A(3,3)-A(2,3)*A(3,1))
      Adj(1,3)=A(2,1)*A(3,2)-A(2,2)*A(3,1)
      Adj(2,1)=-(A(1,2)*A(3,3)-A(1,3)*A(3,2))
      Adj(2,2)=A(1,1)*A(3,3)-A(1,3)*A(3,1)
      Adj(2,3)=-(A(1,1)*A(3,2)-A(1,2)*A(3,1))
      Adj(3,1)=A(1,2)*A(2,3)-A(1,3)*A(2,2)
      Adj(3,2)=-(A(1,1)*A(2,3)-A(1,3)*A(2,1))
      Adj(3,3)=A(1,1)*A(2,2)-A(1,2)*A(2,1)
      return
      end subroutine adjunta
!     -------------------------------------------------------
!     SUBRUTINA DEL SIMBOLO DE PERMUTACION
!     -------------------------------------------------------
      subroutine simpermu(a)
      real*8 a(3,3,3)
      integer i,j,k
       i=1
      do j=1,3
      do k=1,3
      if (j==i) then
       a(i,j,k)=0
      elseif (k==i) then
       a(i,j,k)=0
      elseif(k==j) then
       a(i,j,k)=0
      elseif (j.gt.k) then
       a(i,j,k)=-1
      else
       a(i,j,k)=1
      end if
      enddo
      enddo
       i=2
      do j=1,3
      do k=1,3
      if (j==i) then
       a(i,j,k)=0
      elseif (k==i) then
       a(i,j,k)=0
      elseif(k==j) then
       a(i,j,k)=0
      elseif (j.gt.k) then
       a(i,j,k)=1
      else
       a(i,j,k)=-1
      endif
      enddo
      enddo
       i=3
      do j=1,3
      do k=1,3
      if (j==i) then
      a(i,j,k)=0
      elseif (k==i) then
       a(i,j,k)=0
      elseif(k==j) then
       a(i,j,k)=0
      elseif(j.lt.k) then
       a(i,j,k)=1
      else
       a(i,j,k)=-1
      endif
      enddo
      enddo
      end subroutine simpermu
!     -----------------------------------------------------
!     SUBRUTINA DEL DETERMINANTE DE UN TERSOR
!     ------------------------------------------------------
```

```
      subroutine det (B,detA,a)
      real*8 B(3,3),detA,a(3,3,3)
      integer i,j,k
      call simpermu(a)
      detA=0.0d0
      do i=1,3
      do j=1,3
      do k=1,3
      detA=detA+a(i,j,k)*B(1,i)*B(2,j)*B(3,k)
      enddo
      enddo
      enddo
      return
      end subroutine det
!     -------------------------------------------------------------
!     SUBRUTINA PARA EL TENSOR DESVIADOR
!     -----------------------------------------------------------
      subroutine tdesv(T,Td) !subrutina para el tensor desviador
      real*8 T(3,3), Td(3,3)
      integer i,j
      do i=1,3
      do j=1,3
      Td(i,j)=T(i,j)-(1.0d0/3.0d0)*tr(T)
      enddo
      enddo
      end subroutine tdesv
c     ------------------------------------------------------------
!    TENSOR DE CUARTO ORDEN ISOTROPICO
!    -------------------------------------------------------------
      subroutine isotropico (a,b,c,d,ISO)
      real*8 a(3,3),b(3,3),c(3,3),d(3,3),ISO(3,3,3,3)
      integer i,j,k,l
      ISO=0.0d0
      do i=1,3
      do j=1,3
      do k=1,3
      do l=1,3
      if (i==k) then
      a(i,k)=1
      else
      a(i,k)=0
      endif
      if (j==l) then
      b(j,l)=1
      else
      b(j,l)=0
      endif
      if (i==l) then
      c(i,l)=1
      else
      c(i,l)=0
      endif
      if (j==k) then
      d(j,k)=1
      else
      d(j,k)=0
      endif
       ISO(i,j,k,l)=ISO(i,j,k,l)+(0.5*(a(i,k)*b(j,l)+c(i,l)*d(j,k)))
      enddo
      enddo
      enddo
      enddo
      end subroutine isotropico
!     --------------------------------------
c     ------------------------------------------------------------
!    TENSOR DE CUARTO ORDEN ISOTROPICO
!    -------------------------------------------------------------
      subroutine isotro4(iso4)
      real*8 iso4(3,3,3,3)
      integer i,j,k,l
      iso4=0.0d0
```

```fortran
      do i=1,3
      do j=1,3
      do k=1,3
      do l=1,3
      iso4(i,j,k,l)=iso4(i,j,k,l)+(0.5*(delta(i,k)*delta(j,l)
     1 +delta(i,l)*delta(j,k)))
      enddo
      enddo
      enddo
      enddo
      end subroutine isotro4

!       TENSOR UNITARIO DE CUARTO ORDEN
!      -------------------------------------------------------------
      subroutine Iunit(Iun)
      DOUBLE PRECISION Iun(3,3,3,3)
      integer i,j,k,l
      Iun=0.0D0
      Do i=1,3
      Do j=1,3
      Do k=1,3
      Do l=1,3
      Iun(i,j,k,l)=delta(i,j)*delta(k,l)
      Enddo
      Enddo
      Enddo
      Enddo
      end subroutine Iunit
!      -------------------------------------------------------------
!      -------------------------------------------------------------
!      TENSOR DE CUARTO ORDEN DESVIADOR
!      ------------------------------------------------------------
      subroutine Idesvi(Idev,iso4,Ivol)
      real*8 Ivol(3,3,3,3),delta(3,3),iso4(3,3,3,3),Idev(3,3,3,3)
      integer i,j,k,l
      Ivol=0.0d0
      do i=1,3
      do j=1,3
      if (i==j) then
       delta(i,j)=1
      else
       delta(i,j)=0
      endif
      enddo
      enddo
      do i=1,3
      do j=1,3
      do k=1,3
      do l=1,3
      Ivol(i,j,k,l)=Ivol(i,j,k,l)+1.0d0/3.0d0*(delta(i,j)*delta(k,l))
      iso4(i,j,k,l)=1.0d0/2.0d0*(delta(i,k)*delta(j,l)
     1 +delta(i,l)*delta(j,k))
      Idev(i,j,k,l)=iso4(i,j,k,l)-Ivol(i,j,k,l)
      enddo
      enddo
      enddo
      enddo
      end subroutine Idesvi

!      --------------------------------------------------------------
!      SUBRUTINA PARA PASAR DE TENSOR DE CUARTO ORDEN A MATRIZ DE 3X3
!      ----------------------------------------------------------
      subroutine tensortomatrix(a3333,  b66)   ! returns b(6,6)
      double precision a3333(3,3,3,3),b66(6,6)
      integer i,j,i9(6),j9(6)
      data i9/1,2,3,1,1,2/
     .     j9/1,2,3,2,3,3/
      do  i=1,6   !  switch to matrix notation
      do  j=1,6
      b66(i,j)=a3333(i9(i),j9(i),i9(j),j9(j))
      enddo
```

```
      enddo
      return
      end subroutine tensortomatrix
!     ---------------------------------------------------------------
!     SUBRUTINA INITIAL
!     ---------------------------------------------------------------
      subroutine Initial(STRESS,T, DSTRAN, DEPS, NTENS,NDI, NSHR)
      double precision STRESS(ntens), T(3,3)
     1 ,DSTRAN(ntens), DEPS(3,3)
      Integer ntens, nshr, ndi
      DEPS=0.0D0
      T=0.0D0
C
      do i=1,ndi
      T(i,i)=stress(i)
      DEPS(i,i)=DSTRAN(i)
      enddo
C
      if (nshr.ge.1) then
      T(1,2)=stress(4)
      T(2,1)=stress(4)
      DEPS(1,2)=0.5d0*DSTRAN(4)
      DEPS(2,1)=0.5d0*DSTRAN(4)
      endif
      if (nshr.ge.2) then
      T(1,3)=stress(5)
      T(3,1)=stress(5)
      DEPS(1,3)=0.5d0*DSTRAN(5)
      DEPS(3,1)=0.5d0*DSTRAN(5)
      endif
      if (nshr.ge.3) then
      T(2,3)=stress(6)
      T(3,2)=stress(6)
      DEPS(2,3)=0.5d0*DSTRAN(6)
      DEPS(3,2)=0.5d0*DSTRAN(6)
      endif
      return
      end subroutine Initial
c------------------------------------------------------
c------------------------------------------------------
       subroutine Solution(NTENS, NDI, NSHR, T, STRESS, JAC, DDSDDE)
       integer NTENS, NDI, NSHR, i, j, k, l
C     Subroutine for filling the stress and Jacobian matrix
       double precision T(3,3), JAC(3,3,3,3), STRESS(NTENS),
     1 DDSDDE(NTENS,NTENS), JAC66(6,6)
      k=1
      l=1
c------------------------------------------------------
      do i=1,ndi
      stress(i)=T(i,i)
      enddo
C
      if (nshr.ge.1) then
      stress(ndi+1)=T(1,2)
      endif
      if (nshr.ge.2) then
      stress(ndi+2)=T(1,3)
      endif
      if (nshr.ge.3) then
      stress(ndi+3)=T(2,3)
      endif
      call tensortomatrix(jac,  jac66)
        do i=1,ndi
        do j=1,ndi
          ddsdde(i,j)=jac66(i,j)
        enddo
      enddo
      do i=ndi+1,ndi+nshr
        do j=1,ndi
          ddsdde(i,j)=jac66(3+k,j)
        enddo
```

```
        k=k+1
      enddo
      do i=1,ndi
      l=1
        do j=ndi+1,ndi+nshr
          ddsdde(i,j)=jac66(i,3+l)
          l=l+1
        enddo
      enddo
      k=1
      do i=ndi+1,ndi+nshr
        l=1
        do j=ndi+1,ndi+nshr
          ddsdde(i,j)=jac66(3+k,3+l)
          l=l+1
        enddo
        k=k+1
      enddo
       Return
       end subroutine Solution
!     ----------------------------------------------------------------
!     SUBRUTINA TENSOR DE RIGIDEZ
!     ---------------------------------------------------------------
      subroutine Trigidez(lambda,Nu,delt33,rig)
      real*8 rig(3,3,3,3),delt33(3,3),lambda,Nu
      integer i,j,k,l
      do i=1,3
      do j=1,3
      if (i==j) then
       delt33(i,j)=1
      else
       delt33(i,j)=0
      endif
      enddo
      enddo
      do i=1,3
      do j=1,3
      do k=1,3
      do l=1,3
      rig(i,j,k,l)=lambda*(delt33(i,j)*delt33(k,l))
     & +2.0d0*Nu*(delt33(i,k)*delt33(j,l)+delt33(i,l)*delt33(k,l))
      enddo
      enddo
      enddo
      enddo
      end subroutine Trigidez
!     ----------------------------------------------------------------
!     SUBRUTINA DEL TENSOR TASA DE DEFORMACIONES
!     ----------------------------------------------------------------
      SUBROUTINE D1(DSTRAN, D, dtime, NDI, NSHR, NTENS)
C     Strain rate tensor D
      integer i,j, NDI, NSHR, NTENS
      double precision D(3,3), DSTRAN(6), dtime
      if (dtime==0.0d0) then
      D=0.0D0
      else
      D=0.0D0
      Do i=1,ndi
      D(i,i)=dstran(i)/dtime ! covariant components, matrix format
      Enddo
      if (nshr.ge.1) then
      D(1,2)=dstran(4)/(2.0d0*dtime)
      D(2,1)=D(1,2)
      endif
      if (nshr.ge.2) then
      D(1,3)=dstran(5)/(2.0d0*dtime)
      D(3,1)=D(1,3)
      endif
      if (nshr.ge.3) then
      D(2,3)=dstran(6)/(2.0d0*dtime)
      D(3,2)=D(2,3)
```

```
      endif
      endif
      END SUBROUTINE D1

      END MODULE subrutinas
```

References

[1] A. Niemunis. *INCREMENTAL DRIVER, user's manual*. Soils Models, Hub for Geotechnical Professionals, 2014. URL: https://soilmodels.com/idriver/.