Main_J2Plasticity.m

```matlab
clear; close all; clc ;
disp('');
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%');
disp('%                                                    %');
disp('%               J2 PLASTICITY ALGORITHM              %');
disp('%                                                    %');
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%');


%% INPUT DATA
disp('-----------------   INPUT DATA   ---------------')

% Choose number of caes to run (this is because in this way I can
%run more than on case and plot them on the same figure for
%comparison. By default only one case is executed.)

ncases = cinput('How many cases do you want to run?: ', 1);
for i_cases = 1 : ncases

% General material parameters (for all models)
MaterialData.E = cinput('Young modulus, E [N/m2]: ', 2e11); % Default value is 2E+11 N/m2
MaterialData.sigma_y = cinput('Yield stress, sigma_y [N/m2]: ', 4e8); % Default value is 4e8 N/m2
MaterialData.nu = cinput('Poisson coefficient, nu [-]: ', 0.3);

% Matrix of elastic constants in terms of E and nu

% Lame coefficients -> lambda and mu
MaterialData.lambda = (MaterialData.E*MaterialData.nu)/((1 + MaterialData.nu)*(1 - 2*MaterialData.nu));
MaterialData.mu = MaterialData.E/(2*(1 + MaterialData.nu));
MaterialData.k = MaterialData.lambda+(2/3)*MaterialData.mu; %bulk modulus

aux_vec = [1, 1, 1, 0, 0, 0]'; % auxiliar vector
%deviatoric operator (Voigt's notation)
Dev_op = eye(6)-(1/3)*(aux_vec*aux_vec');
MaterialData.C = MaterialData.k*(aux_vec*aux_vec') + 2*MaterialData.mu*Dev_op;


% Models: perfect plasticity, isotropic hardening only, kinematic
% hardening only or combination of isotropic & kinematic hardening
fprintf(' \n ')
disp('The following models are available: ');
disp('[1]: Perfect plasticity')
disp('[2]: Isotropic hardening only')
disp('[3]: Kinematic hardening only')
disp('[4]: Isotropic hardening + Kinematic hardening')
```

```
% Choose model to run
MODEL = cinput('What case do you want to run?: ', 1);


% Rate-dependent/Rate-independent case
fprintf(' \n ')
disp('Rate-dependent/Rate-independent model: ');
% Specify if you want to run a viscous case: 'Y' (YES) --> viscous
RateDep = input('Do you want to run a rate-dependent model? [Y/N]: ','s');
if RateDep == 'Y'
    % If viscous case is chosen, enter value for viscosity
    % eta = 1000e8 Pa s by default
    MaterialData.visc = cinput('Introduce a value for the viscosity [Pa*s]: ', 1000e8);
else
    % Otherwise, set viscosity value to zero
    MaterialData.visc=0;
end

% Enter specific hardening data for the model, K and H
switch MODEL

    case 2 % Isotropic hardening
        disp('Isotropic hardening model');
        HARDENING = 'YES'; % Variable to control hardening
        % Introduce value for K
        MaterialData.K = cinput('Isotropic hardening modulus, K [N/m2]: ', 3e10);
        MaterialData.H = 0; % Set H=0 in this case

    case 3 % Kinematic hardening model
        disp('Kinematic hardening model');
        HARDENING = 'YES';
        ISO_HARDENING = 'NO'; % Variable to control isotropic hardening
        MaterialData.H = cinput('Kinematic hardening modulus, H [N/m2]: ', 1.5e10);
        MaterialData.K = 0; % Set K=0 for kinematic case

    case 4 % Isotropic and Kinematic hardening model
        disp('Isotropic + Kinematic hardening model');
        HARDENING = 'YES';
        % Then introduce values for K and H
        MaterialData.K = cinput('Isotropic hardening modulus, K [N/m2]: ', 3e10);
        MaterialData.H = cinput('Kinematic hardening modulus, H [N/m2]: ', 1.5e10);

    otherwise % Perfect plasticity
        disp('Perfect plasticity');
        HARDENING = 'NO'; ISO_HARDENING = 'NO' ;
```

```
        MaterialData.K = 0; MaterialData.H = 0 ; % Set both K and H to 0
end

% Choose linear or exponential isotropic hardening if needed
if MODEL == 2 || MODEL == 4 % If we choose isotropic model or isotropic + kinematic hardening
    disp('Choose an isotropic hardening type: ')
    disp('[1]: Linear isotropic hardening')
    disp('[2]: Non linear isotropic hardening (exponential saturation law)')
    isotropic_hardening = cinput('Type of isotropic hardening: ',1);
    % Default is linear isotropic hardening

    if isotropic_hardening == 2
        ISO_HARDENING = 'EXPONENTIAL';
        % Introduce values for the exponential law
        % Asymptotic value
        MaterialData.sigma_inf = cinput('Introduce a value for the saturation law, sigma_inf [N/m2]: ', 1e9);
        % Exponent value
        MaterialData.delta = cinput('Introduce the value for the exponential, delta: ', 130);
    else
        ISO_HARDENING = 'LINEAR';
    end
end

% Total time for each load step and time interval
fprintf(' \n ')
t_end = cinput('Total for each loading interval [s]: ', 2);
delta_t = cinput('Time increment [s]: ', 0.05);

% Strain history
fprintf(' \n ')
disp('Strain history is being computed')
% Uniaxial strain path
nloadstates = 5; % number of states for the strain path
eps_value = zeros(nloadstates,1); % initialize

% Define values of strain in the path. We go from 0 to 0.01,
%then from 0.01 to 0, then to -0.01 and so on until we
%close the cycle.
eps_value(1) = 0.0; eps_value(2) = 0.01 ;eps_value(3) = 0;
eps_value(4) = -0.01 ; eps_value(5) = 0; eps_value(6) = 0.01;
strain_history = zeros(nloadstates*t_end/delta_t, 1);
% compute all the strains on the path (done in the same way as in
% assignment 1).
for j = 2:(t_end/delta_t)+1
    strain_history(j) = (eps_value(2)/(t_end/delta_t))*(j-1);
    strain_history(j+t_end/delta_t) = eps_value(2) + ...
                        ((eps_value(3) - eps_value(2))/(t_end/delta_t))*(j-1) ;
```

```
    strain_history(j+ 2*(t_end/delta_t)) = eps_value(3) + ...
                             ((eps_value(4) - eps_value(3))/(t_end/
                                 delta_t))*(j-1) ;
    strain_history(j+ 3*(t_end/delta_t)) = eps_value(4) + ...
                             ((eps_value(5) - eps_value(4))/(t_end/
                                 delta_t))*(j-1) ;
    strain_history(j+ 4*(t_end/delta_t)) = eps_value(5) + ...
                             ((eps_value(6) - eps_value(5))/(t_end/
                                 delta_t))*(j-1) ;
end

% Time vector for plot stress-time curves
time=zeros((nloadstates)*(t_end/delta_t),1);
for k=1:(nloadstates*t_end/delta_t)
    time(k)=k*delta_t;
end

% Initialize sigma11 and dev_sigma11 for plotting
sigma11 = zeros(length(strain_history),1);
dev_sigma11 = zeros(length(strain_history),1);


fprintf(' \n ')
disp('--------------------    END INPUT DATA    ------------------')

%% ALGORITHM
% Initialize plastic internal variables (plastic strain, isotropic
% hardening variable and kinematic hardening variable)
eps_p = zeros(6,1);
xi = 0;
bar_xi = zeros(6,1);

% Initialize stress variables
sigma = zeros(6,1);
dev_sigma = zeros(6,1);
q = 0;
bar_q = zeros(6,1);

% Initialize plastic multiplier
gamma = zeros(length(strain_history),1);
fprintf(' \n ')
disp('Computing ...')

for i = 2 : length(strain_history) % Time loop

     % Compute trial state
     eps = [strain_history(i), 0 , 0 , 0 , 0 , 0]'; % uniaxial
     [f_trial, TrialState] = ComputeTrialStateJ2 (MaterialData,...
     ISO_HARDENING, eps, eps_p, xi, bar_xi, Dev_op);

    if f_trial <= 0 % elastic step --> (*)_n+1 = (*)_trial

        % update variable values
```

```
        eps_p = TrialState.eps_p ;
        xi = TrialState.xi;
        bar_xi = TrialState.bar_xi;
        sigma11(i) = TrialState.sigma(1);  % Save sigma_11 for plotting
        dev_sigma11(i) = TrialState.dev_sigma(1);  % Save dev_sigma(11) for plotting
        q = TrialState.q ;
        bar_q = TrialState.bar_q;
        C_ep = MaterialData.C;

    else % Elasto - plastic step --> radial return mapping algorithm

        [C_ep, UpdatedVariables, gamma] = PlasticStepJ2 (MaterialData, ...
         f_trial,delta_t, ISO_HARDENING, TrialState, Dev_op) ;

        % Save updated variables to correspondent vector
        gamma(i) = gamma;
        sigma11(i) = UpdatedVariables.sigma(1);
        dev_sigma11(i) = UpdatedVariables.dev_sigma(1);
        q = UpdatedVariables.q;
        bar_q = UpdatedVariables.bar_q;

        eps_p = UpdatedVariables.eps_p;
        xi = UpdatedVariables.xi;
        bar_xi = UpdatedVariables.bar_xi;

    end
end
fprintf(' \n ')
disp('The algorithm has finished.')

%% POST-PROCESSING
colors = {'r','b','g','c','k'};
line_type = {'<-','o-','*-','h-','p-',};
% sigma11 - strain11 curve
figure(1)
plot(strain_history, sigma11,strcat(colors{i_cases},line_type{i_cases}));
hold on
legendInfo{i_cases} = ['$\eta = $' num2str(MaterialData.visc, '%1.2E')];

% Add labels and grids
grid on
grid minor
set(gca,'FontSize',16)
xlabel('$\varepsilon_{11}$ \ [-]','Interpreter','LaTex','FontSize',22)
ylabel('$\sigma_{11} \ [N/m^2]$','Interpreter','LaTex','FontSize',22)
legend(legendInfo,'Interpreter','LaTex','Location','Best',...
```

```
                                                'Orientation','Vertical', 'FontSize',16)
 % dev(sigma11)-strain11 curve
figure(2)
plot(strain_history, dev_sigma11, strcat(colors{i_cases}, line_type{i_cases}));
hold on
grid on
grid minor
set(gca,'FontSize',16)
xlabel('$\varepsilon_{11}$ \ [-]','Interpreter','LaTex','FontSize',22)
ylabel('$dev[\sigma_{11}] \ [N/m^2]$','Interpreter','LaTex','FontSize',22)
legend(legendInfo,'Interpreter','LaTex','Location','Best',...
                                                'Orientation','Vertical','FontSize', 16)

%For viscous model, also plot stress-time curves
if MaterialData.visc ~= 0
    figure(3)
    % stress11-strain11 curve
    plot([0;time], sigma11, strcat(colors{i_cases},line_type{i_cases}))
    hold on
    grid on
    grid minor
    set(gca,'FontSize',16)
    xlabel('$t \ [s]$','Interpreter','LaTex','FontSize',22)
    ylabel('$\sigma \ [N/m^2]$','Interpreter','LaTex','FontSize',22)
    legend(legendInfo,'Interpreter','LaTex','Location','Best',...
                                                'Orientation','Vertical','FontSize', 16)
    % dev(stress11)-strain11
    figure(4)
    plot([0;time], dev_sigma11, strcat(colors{i_cases},line_type{i_cases}));
    hold on
    grid on
    grid minor
    set(gca,'FontSize',16)
    xlabel('$t \ [s]$','Interpreter','LaTex','FontSize',22)
    ylabel('$dev[\sigma_{11}] \ [N/m^2]$','Interpreter','LaTex','FontSize',22)
    legend(legendInfo,'Interpreter','LaTex','Location','Best',...
                                                'Orientation','Vertical','FontSize', 16)
end
end
```

ComputeTrialStateJ2.m

```
function [f_trial, TrialState] = ComputeTrialStateJ2(MaterialData, ...
            ISO_HARDENING,eps, eps_p, xi, bar_xi, Dev_op)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function computes the variables for the trial state
%
% INPUTS :
%
%  MaterialData --> structure containing basic material info,
%  i.e. E, K, H, yield stress , etc.
%
%  ISO_HARDENING --> variable indicating if isotropic hardening is
%  considered or not. Values : 'NO ', 'LINEAR ', 'EXPONENTIAL '
%
%  eps --> strain vector for the present time step
%
%  eps_p --> plastic strain vector for the present time step
%
%  xi --> isotropic hardening variable value
%
%  bar_xi --> kinematic hardening variable value
%
%  Dev_op --> deviatoric operator
%
%  OUTPUTS:
%
%  f_trial --> yield function value at the trial state
%
%  TrialState --> strucutre containing the stress and plastic
%  strain variables at the trial state
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

eps_p_trial = eps_p ;
xi_trial = xi;
bar_xi_trial = bar_xi;

sigma_trial = MaterialData.C*(eps - eps_p_trial) ;

if strcmp(ISO_HARDENING,'LINEAR') == 1 || strcmp(ISO_HARDENING,'NO') == 1
    % isotropic linear hardening, kinematic hardening or perfect plasticity
    q_trial = - MaterialData.K * xi_trial ;
elseif strcmp(ISO_HARDENING,'EXPONENTIAL') == 1
    % isotropic exponential hardening (non-linear)
    q_trial = -(MaterialData.sigma_inf - MaterialData.sigma_y) * (1-exp(-MaterialData.delta*xi_trial));
end

```

```
bar_q_trial = - MaterialData.H*(2/3)*eye(6)*bar_xi_trial  ;

% Compute deviatoric part of the stress
dev_sigma_trial = Dev_op * sigma_trial;
f_trial = norm(dev_sigma_trial - bar_q_trial) - (sqrt(2/3))*(
    MaterialData.sigma_y - q_trial);

% Create structure of data with trial state information
TrialState.eps_p = eps_p_trial;
TrialState.xi = xi_trial;
TrialState.bar_xi = bar_xi_trial;
TrialState.sigma = sigma_trial;
TrialState.dev_sigma = dev_sigma_trial;
TrialState.q = q_trial;
TrialState.bar_q = bar_q_trial;
end
```

PlasticStepJ2.m

```
function [C_ep, UpdatedVariables, gamma] = PlasticStepJ2 (MaterialData,...
         f_trial,delta_t, ISO_HARDENING, TrialState, Dev_op)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function computes the consistent elastoplastic tangent
% modulus and updates the values of the plastic internal variables
% and the stress variables for a plastic step
%
% INPUTS :
%
%  MaterialData --> structure containing the different material
%  parameters : E, K, H, sigma_y and , if it is the case , sigma_inf
%  and delta (for the exponential law )
%
%  f_trial --> yield condition at trial state
%
%  delta_t --> time interval
%
%  HARDENING --> auxiliar variable , set to 'NO ' in the perfect
%  plasticity case
%
%  ISO_HARDENING --> auxiliar variable . Values are 'NO ' for the
%  "kinematic hardening only " model , 'LINEAR ' for linear
%  isotropic hardening and 'EXPONENTIAL ' for non - linear
%  isotropic hardening .
%
%  TrialState --> structure containing stress and strain variables
%  of the trial state , i.e. sigma , q, bar_q , eps_p , xi , bar_xi
%
%  Dev_op --> deviatoric operator
%
% OUTPUTS:
%
%  C_ep --> consistent elastoplastic tangent modulus
%
%  UpdatedVariables --> structure containing the updated stres and
%  strain variables after a plastic step
%
%  gamma --> plastic multiplier
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Local variables
K=MaterialData.K ; H=MaterialData.H;
mu = MaterialData.mu; visc=MaterialData.visc; sigma_y=MaterialData.sigma_y;
k = MaterialData.k ;

% Compute plastic multiplier
if strcmp(ISO_HARDENING,'LINEAR') == 1 || strcmp(ISO_HARDENING,'NO') == 1
```

```
    % linear isotropic hardening or (perfect plasticity or just
       kinematic)
    gamma = f_trial/((2*mu+(2/3)*(K+H)+(visc/delta_t))*delta_t);
    [C_ep, UpdatedVariables] = ReturnMappingBasicJ2 (gamma,mu,H,K,k,
       TrialState,visc,delta_t, Dev_op);
else % exponential isotropic hardening -> Newton-Raphson (non-linear
    problem)
    gamma = NewtonRaphson (f_trial, MaterialData.visc, mu, H,
       sigma_y, MaterialData.sigma_inf, MaterialData.delta,
       TrialState.xi, delta_t) ;
    [C_ep, UpdatedVariables] = ReturnMappingNonLinearHardJ2 (gamma,
       mu,H,k, MaterialData.sigma_inf, sigma_y, TrialState, delta_t,
        MaterialData.visc, MaterialData.delta, Dev_op);
end
end
```

ReturnMappingBasicJ2.m

```
function[C_ep, UpdatedVariables] = ReturnMappingBasicJ2 (gamma,mu,H,
   K,k,TrialState,visc,delta_t, Dev_op)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function contains the return mapping algorithm for a
% plastic step for the cases of linear isotropic hardening
% only ( viscous and non - viscous ), kinematic hardening only
% (viscous and non - viscous) and linear isotropic hardening
% combined with kinematic hardening (viscous andnon - viscous).
% Same expressions can be used for these cases after setting
% the proper variables to zero.
%
% INPUTS :
%
%   gamma --> plastic multiplier
%
%   mu --> Lame coefficient
%
%   H --> Kinematic hardening modulus
%
%   k -- > Isotropic hardening modulus
%
%   k -- > bulk modulus
%
%   TrialState --> structure containing the information of the trial
%   state, i.e. stress and plastic strain variables
%
%   visc --> matierla viscosity
%
%   delta_t --> time interval
%
%   Dev-op --> deviatoric operator
%
%  OUTPUT:
%
%  C_ep --> elastoplastic tangent operator
%
%  UpdatedVariables --> structure containing the stress and
%  plastic strain updated variables after performing the
%  return mapping algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Local variables, just for convenience
 sigma_trial = TrialState.sigma; q_trial = TrialState.q ;
    bar_q_trial = TrialState.bar_q;
 eps_p_trial = TrialState.eps_p ; xi_trial = TrialState.xi;
    bar_xi_trial = TrialState.bar_xi;
 dev_sigma_trial = TrialState.dev_sigma;

 n = (dev_sigma_trial - bar_q_trial)/norm(dev_sigma_trial -
    bar_q_trial);
```

```

% Update variables
UpdatedVariables.sigma = sigma_trial - gamma*delta_t*2*mu*n;
UpdatedVariables.q = q_trial - gamma*delta_t*sqrt(2/3)*K;
UpdatedVariables.bar_q = bar_q_trial + gamma*delta_t*(2/3)*H*n;
UpdatedVariables.dev_sigma = Dev_op*UpdatedVariables.sigma;

UpdatedVariables.eps_p = eps_p_trial + gamma*delta_t*n;
UpdatedVariables.xi = xi_trial + gamma*delta_t*sqrt(2/3);
UpdatedVariables.bar_xi = bar_xi_trial - gamma*delta_t*n;

% auxiliar variables
aux_vec=[1 1 1 0 0 0]';
delta = 1 - (2*mu*gamma*delta_t)/norm(dev_sigma_trial - bar_q_trial);
bar_delta = 2*mu/(2*mu+2/3*(K+H)+visc/delta_t) - (1 - delta);
% elastoplastic tangent modulus
C_ep = k*(aux_vec*aux_vec') +2*mu*delta*Dev_op-2*mu*bar_delta*(n*n');
end
```