```
% <----  Main Program for quadratic optimal control implemented using Kalman predictor
%      on Continuously Stirred Tank Reactor (CSTR) System
%      Model for control: State realization of identified ARMAX model
%      Plant Simulation : Discrete Linear Perturbation developed from
%                         mechanistic model
%                        OR Nonlinear Mechnistic model
% ------------------------------------------------------------------->

clear all ; clc ;  close all

  global CSTR_mod ;       %  Global Data structure containing System related parameters

  load CSTR_para         % Initialize CSTR_mod data structure and operating conditions
  load CSTR_LinMod_I  % Load discrete linear model obtained through linearization
                         % Used only for plant simulation

  % Following local variables are created only for improving readability of the program
  n_st = dmod_lin.n_st ; n_op = dmod_lin.n_op  ;
  n_ip = dmod_lin.n_ip ; n_ud = dmod_lin.n_ud ;
  Xs = dmod_lin.Xs ; Ys =  dmod_lin.Ys  ;    % Steady state operating conditions
  Us = dmod_lin.Us  ; Ws  = dmod_lin.Ws  ;
  phy = dmod_lin.phy ; gama_u = dmod_lin.gama_u ;
  gama_d = dmod_lin.gama_d ; C_mat = dmod_lin.C ;

  d_step = -0.2  ;      % Variable to introduce step disturbance

  % Note: It is possible to work directly with elements of dmod_lin object
  % without requiring creation of above local variables

  samp_T = dmod_lin.T ;       % Sampling interval
  N_samples = 401 ;   % Number of samples in open loop simulation run

  fprintf( '\n\n Continuously Stirred Tank Reactor (CSTR): Kalman Filtering Demo↙
Program' )
  fprintf( '\n\n\t'),
  mod_type = input('Plant Simulation using (0) : Linear (1) : Nonlinear model? ')  ;

%----Initialization for absolute and dev state variables -------
% Create dummy arrays for simulation

xk = zeros(n_st, N_samples) ;     % Matrices for saving deviation variables
uk = zeros(n_ip, N_samples) ;
```

```
yk = zeros(n_op, N_samples) ;
dk = zeros(n_ud, N_samples) ;

state_sigma = [ 0.01 ]' ;  % Generate state noise sequence for simulation
wk =  state_sigma * randn(n_ud, N_samples) ;
meas_sigma  = [ 0.1 ]';     % Generate state noise sequence for simulation
vk = meas_sigma * randn(1, N_samples) ;
xk(:,1) = [ 0.4 5 ]'  ;          % Initial deviation state in the plant (at k = 0)
yk(:,1) = C_mat * xk(:,1) + vk(1)   ;  %  Initial dev. Measurement (at k = 0)
dk(:,1) =  wk(:,1) ;

%  Matrices for saving Absolute variables
 Xk_abs = zeros(n_st, N_samples) ;
Uk_abs = zeros(n_ip, N_samples) ;
Yk_abs = zeros(n_op, N_samples) ;
Dk_abs = zeros(n_ud, N_samples) ;

Xk_abs(:,1) = Xs + xk(:,1)  ;              % Plant: Initial abs. state (at k = 0)
Yk_abs(:,1) = C_mat * Xs + yk(:,1) ;     %  Initial abs Measurement (at k = 0)
Uk_abs(:,1) = Us + uk(:,1) ;
Dk_abs(1) = Ws + wk(1) ;

% <---- Controller design based on state realization of identified ARMAX model ---->

% Generate state realization of identified ARMAX model for CSTR system
% Note: The state realization itself is equivalent to a Kalman predictor
%         and there is no need to carry out observer design separately

load CSTR_IdMod
[phy_id, gama_u_id, C_mat_id, D_mat_id, Lp_inf, z0 ] = ssdata( cstr_armax ) ;
n_st_id = length( phy_id ) ;

% Observer Initialization
zkpred = zeros(n_st_id, N_samples) ; %  Create Dummy matrices for estimated states
ek = zeros(n_op, N_samples) ;       % Innovation sequence
ek(:,1) = yk(:,1) - C_mat_id * zkpred(:,1) ;

% LQOC (Innovation Bias Formulation)  initialization
zsk = zeros(n_st_id, N_samples) ;    % Matrices for saving terget states
usk = zeros(n_ip, N_samples) ;
rk = zeros(n_op, N_samples) ;
```

```
ek_f = zeros(n_op, N_samples) ;       % Filtered Innovation sequence
phy_e = 0.98 * eye(n_op) ;             % Innovation filter parameter
phy_r = 0.9 * eye(n_op) ;             % Setpoint filter parameter

% Marices required in Target state Computation
Ku_mat = C_mat_id * inv(eye(n_st_id) - phy_id) * gama_u_id ;
Ke_mat = C_mat_id * inv(eye(n_st_id) - phy_id) * Lp_inf +eye(n_op) ;

% Compute Controller gain matrix by solving steady state Riccati eqns.
% State weighting matrix  with output control
Wx =  C_mat_id' * C_mat_id  ;
Wu =  diag( [1  5]) ;    % Error weighting matrix
[G_inf, S_inf, EigVal_CL] = dlqr(phy_id, gama_u_id, Wx, Wu) ;

% <------------- Closed Loop Dynamic Simulation ------------------>
kT = zeros(N_samples,1) ;
kT(1) = 0 * samp_T ;   %  k = 1 corresponds to time = 0

for k = 2 : N_samples,
   k                                  % Print sampling time on screen
   kT(k) = (k-1) * samp_T ;
   %   <---- Plant simulation form instnat (k-1) to (k) ---->
   %   Integrate the model equations for U(k-1) and W(k-1) to compute X(k) and Y(k) ---->
   if ( mod_type == 0 )     % Process simulation using discrete linear perturbation model
      xk(:,k) = phy * xk(:,k-1) + gama_u * uk(:,k-1) +  gama_d * dk(:,k-1) ;
      yk(:,k) = C_mat * xk(:,k) + vk(:,k) ;
      Xk_abs(:,k) = Xs + xk(:,k) ;                % Save simulation data in absolute varriables
      Yk_abs(:,k) = Ys + yk(:,k) ;
   else                          % Process simulation using nonlinear ODE model
      CSTR_mod.Fc  = Uk_abs(1,k-1)  ;   % Assign manipulated and disturbance inputs
      CSTR_mod.F  =  Uk_abs(2,k-1)  ;
      CSTR_mod.Cao = Dk_abs(k-1)  ;
      %  Runge Kutta integration  over interval [(k-1)T, kT] using MATLAB ODE solver
      [t,Xt] = ode45( 'CSTR_Dynamics', [0 samp_T] , Xk_abs(:,k-1) ) ;
      Xk_abs(:,k) = Xt( length(t),:)' ;              % State at instnat (k+1)
      Yk_abs(:,k) = C_mat * Xk_abs(:,k) + vk(:,k) ;   % Measured Output at insthat (k)
      xk(:,k) =  Xk_abs(:,k)  - Xs ;         % Generate Perturbation variables
      yk(:,k) = Yk_abs(:,k) - Ys ;
   end

   % <-----Controller Calculations at k'th sampling instant ------>
   % Specify setpoint
```

```
    if ( k < 100 ), setpt = 0 ;
    elseif ( k >=101 ),  setpt = 5 ;   % Step chenge in setpoint
    end
    % Generate filtered setpoint trajectory
    rk(:,k) = phy_r * rk(:,k-1)+ (eye(n_op)-phy_r)* setpt ;

    %  Identified Model used as state observer
    zkpred(:,k) = phy_id * zkpred(:,k-1) + gama_u_id * uk(:,k-1) + Lp_inf * ek(:,k-1)  ;
    ek(:,k) =  yk(:,k) - C_mat_id *  zkpred(:,k)  ;   % Compute Innovation
    % Filtered innovation for innovation bias implementation
    ek_f(:,k) = phy_e * ek_f(:,k-1) + (eye(n_op) - phy_e) *  ek(:,k) ;

    % Compute target input and target states using idetified model
    usk(:,k) = pinv(Ku_mat) * ( rk(:,k) - Ke_mat * ek_f(:,k) ) ;
    zsk(:,k) =  inv(eye(n_st_id) - phy_id) * ( gama_u_id * usk(:,k) +  Lp_inf * ek_f(:,k) ) ;

    % Innovation Bias controller implementation
    uk(:,k) = usk(:,k)  - G_inf * (zkpred(:,k) - zsk(:,k)  )  ;
    Uk_abs(:,k) = Us + uk(:,k) ;

    % <---- Specify distrbance input at k'th instant for plant simulation ---->
    if ( k < 200 ),   dk(:,k)=  wk(:,k-1)  ;
    elseif ( k >=201 ),  dk(:,k)= d_step + wk(:,k-1)  ;   % Step chenge in setpoint
    end
    Dk_abs(k) = Ws + dk(k) ;

end   % <---- End of Closed Lopp Simulation Loop ---->

 %  <---- Display simulation results graphically ---->
 Init_Graphics_Style ;     % Set parameters for graphics (Optional)
 figure(1),subplot(211), plot( kT, xk(1,:) ) ;
  xlabel('Sampling Instant'), ylabel('Conc.(mod/m3)'), title( 'Plant State') ;
  figure(1),subplot(212), plot( kT , xk(2,:), 'xr', kT, rk, 'g' ) ;
  xlabel('Sampling Instant'), ylabel('Temp.(K)'), title( 'Controlled Output') ;
   figure(2),subplot(311), stairs( kT , uk(1,:) ) ;
   xlabel('Sampling Instant'), ylabel('Coolent Flow'), title( 'Man. Input Perturbations') ;
   figure(2),subplot(312), stairs(kT , uk(2,:) ) ;
   xlabel('Sampling Instant'), ylabel('Inflow')
   figure(2),subplot(313), stairs( kT, dk ) ;
   xlabel('Sampling Instant'), ylabel('Inlet Conc. (mol/m3)'), title( 'Unmeas. Dist.↙
Perturbations') ;
```