Student Workshop
Discrete IMC

- Using the 3-state system
- Compare design and performance of
  - IMC
  - State Deadbeat
- Consider model error

Consider the state space model for 3 tanks in series, used in the previous workshops

$$\dot{x} = Ax + Bu \qquad y = Cx + Du$$

where

$$A = \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \qquad D = 0$$

and the time unit is minutes.

1. For the discrete-time model, based on a sample time of 0.5 minutes, perform two different controller designs: (**a**) **IMC**, where zeros outside the unit circle and negative zeros inside the unit circle are removed; note that an "all-pass" factorization is not performed since the zero outside the unit circle is negative. (**b**) **State Deadbeat control**.

Always make certain that the gain of the "good stuff" (g_{p-}) is the same as the original model (1 in this case), and that the "bad stuff" (g_{p+}) has a gain of 1. (Remember that gains are found by setting z = 1)

$$\tilde{g}_p(z) = \tilde{g}_{p-}(z)\tilde{g}_{p+}(z)$$

Use a continuous representation of the *plant* for simulations, but a discrete *model* for control system design and implementation. Make setpoint changes at t = 1 minute. What happens as λ is decreased for the IMC design? (how does the performance compare with state deadbeat?)

$$q(z) = \tilde{g}_{p-}^{-1}(z) \cdot f(z) \qquad \text{where} \qquad f(z) = \frac{(1-\alpha)z^{-1}}{1-\alpha z^{-1}} = \frac{(1-\alpha)}{z-\alpha}$$

$$\alpha = e^{-\Delta t/\lambda}$$

Implement this control strategy within the simulation framework developed in Workshops 1 & 2.

For a sample time of 0.5 minutes, the discrete process transfer function is

$$\tilde{g}_p(z) = \frac{0.014388(z+0.1831)(z+2.579)}{(z-0.6065)^3} = \frac{0.01439z^2 + 0.03973z + 0.006794}{z^3 - 1.82z^2 + 1.104z - 0.2231}$$

$$= \frac{0.01439z^{-1} + 0.03973z^{-2} + 0.006794z^{-3}}{1 - 1.82z^{-1} + 1.104z^{-2} - 0.2231z^{-3}}$$

The zeros are -0.1831 and -2.579 min^{-1}, and there are three poles at 0.6065 min^{-1}. Factor out both the zero outside the unit circle (-2.579) and the negative zero that is inside the unit circle (-0.1831). Do not use an "all-pass" factorization. Make certain that the "gain" of the "bad stuff" is 1 and that the gain of the "good stuff" is the same as the gain of the original model.

$$\tilde{g}_p(z) = \frac{0.014388(z+0.1831)(z+2.579)}{(z-0.6065)^3} = \underbrace{\frac{0.014388(1.1831)(3.579)z^2}{(z-0.6065)^3}}_{\tilde{g}_{p-}} \cdot \underbrace{\frac{(z+0.1831)(z+2.579)}{(1.1831)(3.579)z^2}}_{\tilde{g}_{p+}}$$

The internal model controller is

$$q(z) = \tilde{g}_{p-}^{-1}(z) \cdot f(z) \qquad \text{where} \qquad f(z) = \frac{(1-\alpha)z^{-1}}{1-\alpha z^{-1}} = \frac{(1-\alpha)}{z-\alpha}$$

and the discrete filter parameter is $\alpha = e^{-\Delta t/\lambda}$ where λ is the closed-loop tuning parameter in continuous-time.

(a) IMC

The IMC controller is

$$q(z) = \frac{(1/0.06092)(z-0.6065)^3}{z^2} \cdot \frac{(1-\alpha)}{(z-\alpha)}$$

which in pole-zero form is expressed as

$$q(z) = 16.414(1-\alpha)\frac{(z-0.6065)(z-0.6065)(z-0.6065)}{(z-\alpha)(z-0)(z-0)}$$

For writing your own code in discrete-time, multiply and place in polynomial form to find

$$q(z) = 16.414*(1-\alpha)\cdot\frac{z^3 - 1.8195z^2 + 1.1035z - 0.2231}{z^3 - \alpha z^2}$$

$$= 16.414*(1-\alpha)\cdot\frac{1 - 1.8195z^{-1} + 1.1035z^{-2} - 0.2231z^{-3}}{1 - \alpha z^{-1}}$$

which is implemented in discrete time as

$$u_k = \alpha u_{k-1} + 16.414(1-\alpha)\left[\tilde{r}_k - 1.8195\tilde{r}_{k-1} + 1.1035\tilde{r}_{k-2} - 0.2231\tilde{r}_{k-3}\right]$$

where $\tilde{r}_k = r_k - \tilde{d}_k = r_k - (y_k - \tilde{y}_k)$

and the model output is calculated by (from $\widetilde{g}_p(z)$, above)

$$\widetilde{y}_k = 1.82\widetilde{y}_{k-1} - 1.104\widetilde{y}_{k-2} + 0.2231\widetilde{y}_{k-3} + 0.01439u_{k-1} + 0.03973u_{k-2} + 0.006794u_{k-3}$$

An example code is shown in the Appendix. The closed-loop result for a perfect model is shown in Figure 1, for λ = 0.25, 0.5 and 1 min.

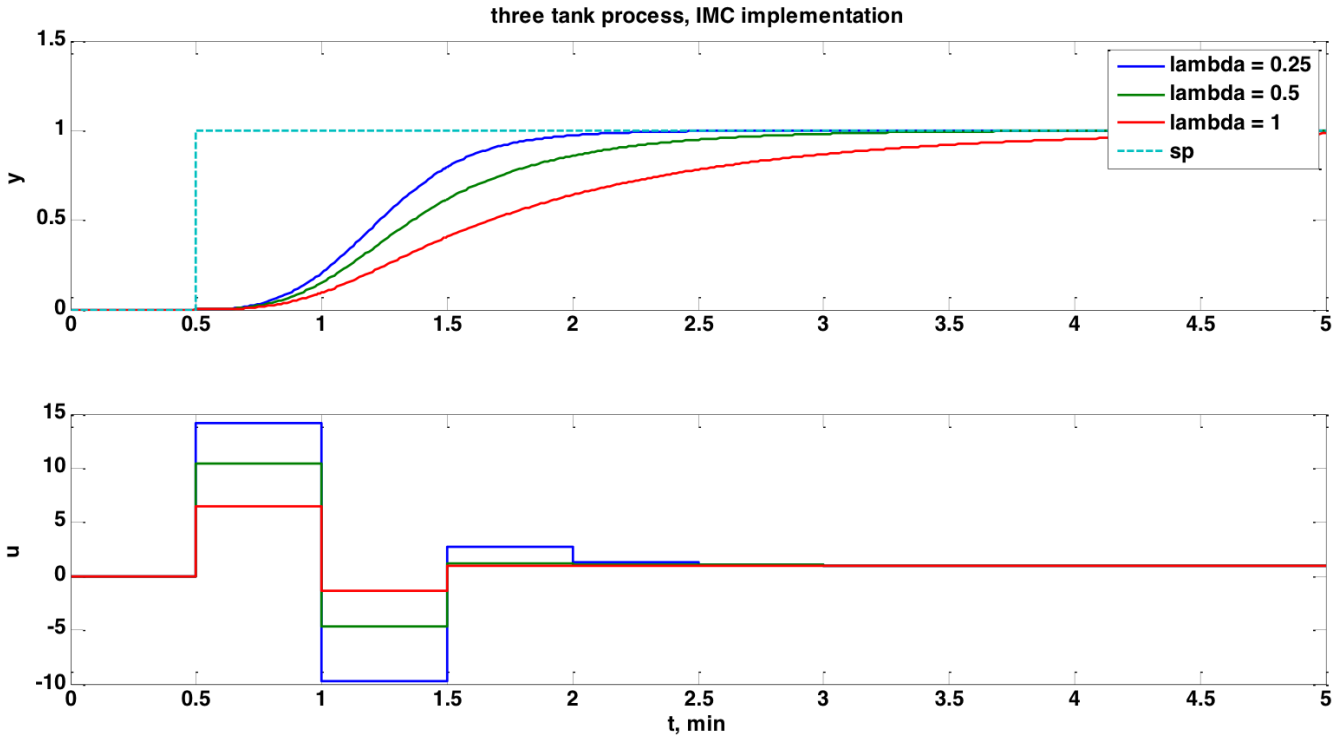

Figure 1. Comparison of the effect of the tuning parameter on the closed-loop response for a step setpoint change. Discrete IMC design procedure.

(b) State Deadbeat control.

In this design no zeros are inverted at all. Also, there is no "filter" in the design technique.

$$\tilde{g}_p(z) = \frac{0.014388(z+0.1831)(z+2.579)}{(z-0.6065)^3} = \underbrace{\frac{0.014388(1.1831)(3.579)z^3}{(z-0.6065)^3}}_{\tilde{g}_{p-}} \cdot \underbrace{\frac{(z+0.1831)(z+2.579)}{(1.1831)(3.579)z^3}}_{\tilde{g}_{p+}}$$

$$q(z) = \frac{(1/0.06092)(z-0.6065)^3}{z^3}$$

$$q(z) = 16.414 \cdot \frac{z^3 - 1.8195z^2 + 1.1035z - 0.2231}{z^3}$$

$$= 16.414 \cdot \frac{1 - 1.8195z^{-1} + 1.1035z^{-2} - 0.2231z^{-3}}{1}$$

What happens as λ is decreased for the IMC design? Notice that, when λ is decreased to 0 in the IMC design, α decreases to 0, resulting in the same controller as state deadbeat control! The state deadbeat performance is shown in Figure 2.

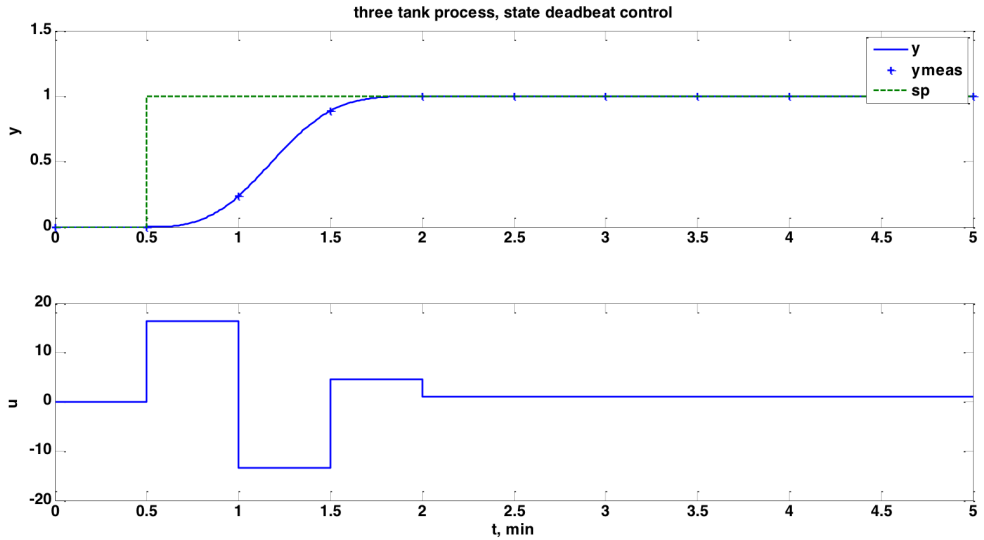

Figure 2. Closed-loop response for state deadbeat control.

Simulation Structure

```
for k = 1:length(time)-1;
    rmodvec = rsp(k-3:k)-distmod(k-3:k)';
   uvec = [u(k-1)];
   u(k) = dimc(qnum,qden,rmodvec,uvec);  % IMC controller calcuation
   umodvec = [u(k-2);u(k-1);u(k)];
   ymodvec = [ymod(k-2);ymod(k-1);ymod(k)];
   ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec); % model pred
%    integrate plant equations:
  [tdummy,xdummy] = ode45('linodepar',[time(k) time(k+1)],xdis(:,k),[],a,b,u(k));
   ndum = length(tdummy);
   xdis(:,k+1) = xdummy(ndum,:);   % plant state
   ydis(k+1) = c*xdis(:,k+1);      % plant output (measured)
   distmod(k+1) = ydis(k+1) - ymod(k+1);  % plant-model mismatch (additive dist)
   tplot = [tplot;tdummy];
   yplot = [yplot;xdummy*c'];
end
```

2. Now consider model error. Let the discrete model and controller remain the same as the ones developed in problem 1, but let the gain of the continuous plant be 0.7 and 1.3 rather than 1 (simply let the element in the B matrix be 0.7 or 1.3). Does your controller exhibit offset?

The results for an uncertain plant (gain = 1.3 and 0.7) are compared in Figure 3, while Figure 4 compares the estimated disturbance for both real plants (gain = 1.3 and 0.7). Clearly the controller does not exhibit offset.

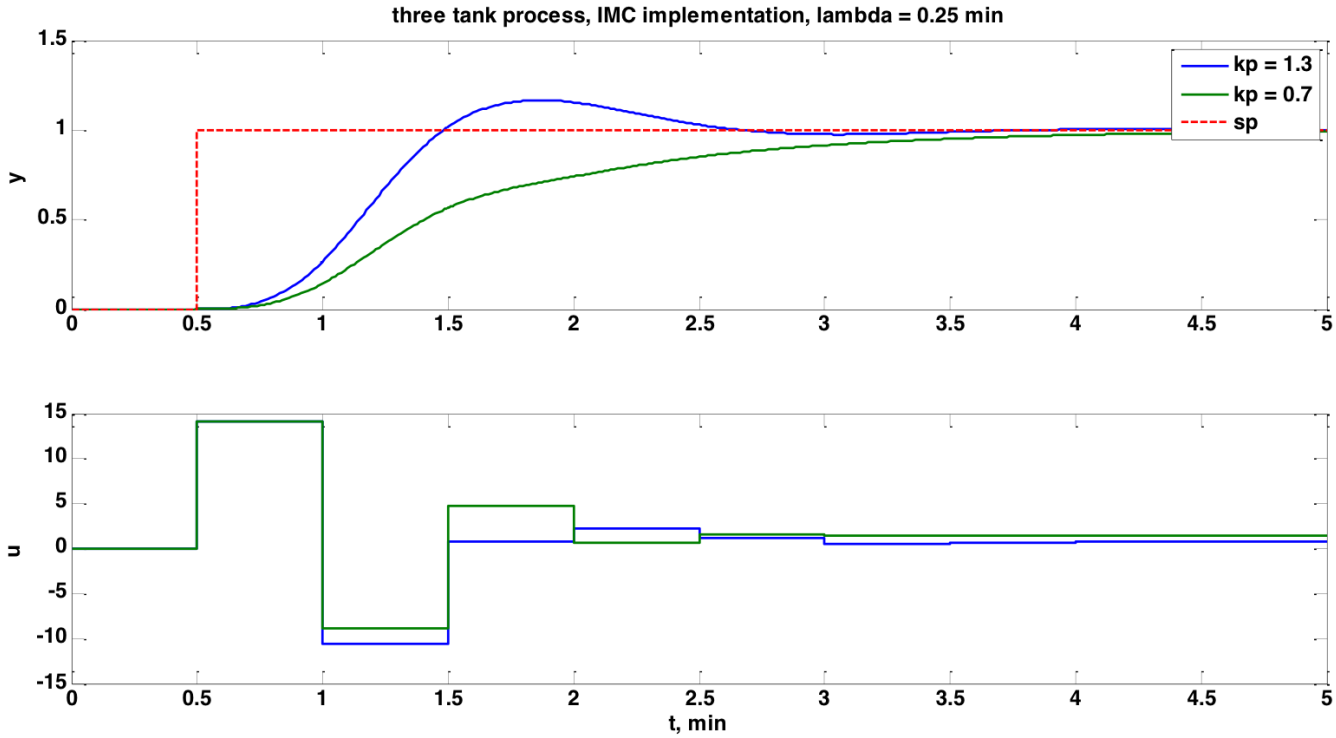

Figure 3. Comparison of the closed-loop responses for an uncertain plant (k_p = 1.3 and 0.7, while the model gain remains 1), for λ = 0.25 min. Discrete IMC design procedure.

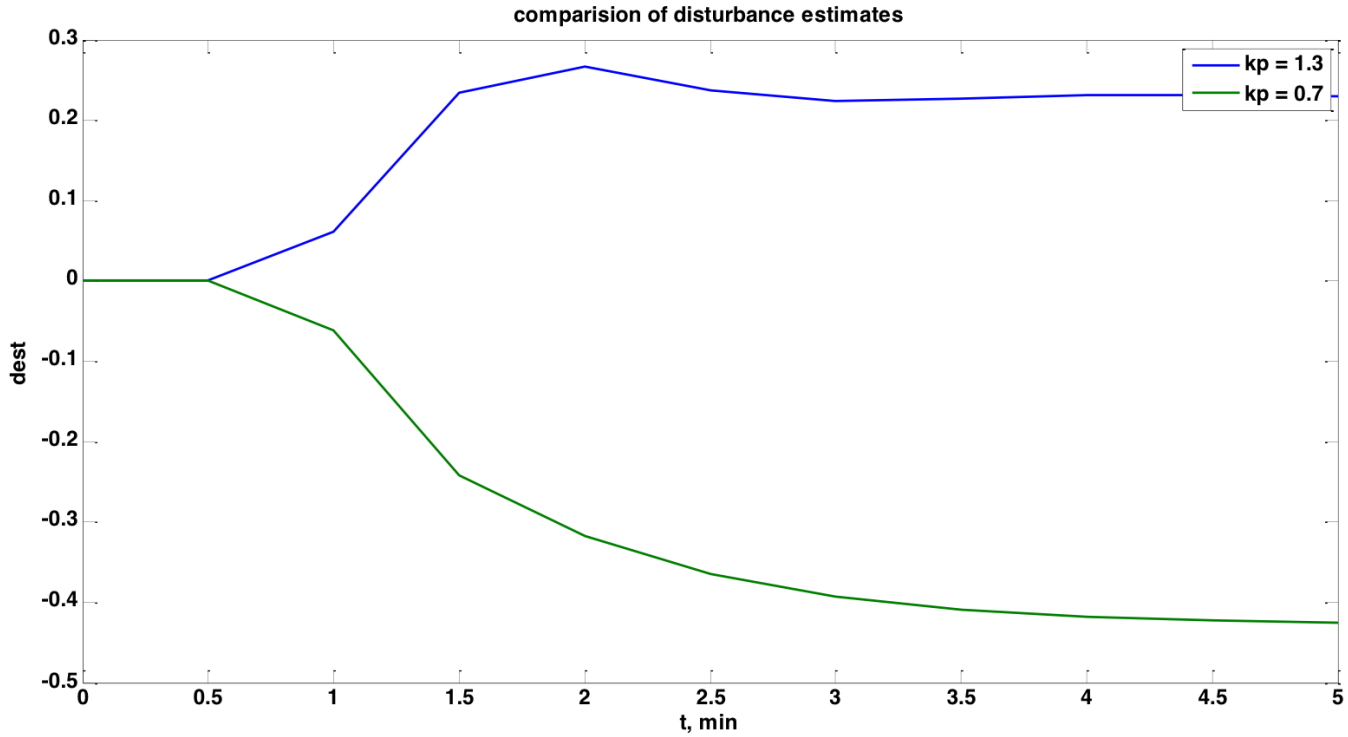

Figure 4. Comparison of the disturbance estimated for an uncertain plant (k_p = 1.3 and 0.7, while the model gain remains 1), for λ = 0.25 min. Discrete IMC design procedure.

```
%  function files:
%    dimc    -- IMC discrete controller q(z) -- calculates u(k)
%    dimcmod -- internal model -- calculates y(k)
% first, continuous state space model
  a = [-1 0 0;1 -1 0;0 1 -1]
  b = [1;0;0]
  c = [0 0 1]
  d = 0
%
  lintank = ss(a,b,c,d)             % defines continuous state space model
% ------------------------------------------------------------------
%
% discrete time model
  delt = 0.5;                       % sample time of 0.5 minutes
  tankssz = c2d(lintank,delt,'zoh')  % create discrete state space model from continuous
% discrete process transfer function
  tanktfz = tf(tankssz)             % create discrete t.f. from discrete state space model
  [nump,denp,tsample] = tfdata(tanktfz,'v')  % get the numerator and denominator
polynomials
  roots(nump)                       % finds zeros of discrete transfer function
  roots(denp)                       % finds poles of discrete transfer function
  tanktzpk = zpk(tankssz)           % zero-pole-gain form (consistency check)
```

```
%------  State deadbeat implementation using IMC formulation
  lambda = 1
  alpha = exp(-delt/lambda)
  qnum = 16.414*(1-alpha).*[1 -1.8195 1.1035  -0.2231];  % IMC numerator
  qden = [1 -alpha]                    % IMC denominator
  modnum = [0.01439  0.03973 0.006794]          % IM numerator
  modden = [1  -1.82  1.104  -0.2231]       % IM denominator
%
% now, set up a for loop and integrate over 1 minute time steps
% currently set up for single input - single output
%
  tplot = [];
  yplot = [];  % save more points for plotting in between samples
  tbeg = 0;    % simulation start time
  tend = 5;   % simulation end time
  time = tbeg:delt:tend;       % generate the time vector
  rsp = [zeros(1,1);ones(length(time)-1,1)]; % setpoint change at second step
  x0  = zeros(3,1);
  y0  = 0;
  u0  = 0;
  xdis(:,1) = x0;             % discrete plant state
  ydis(1)   = y0;             % discrete plant output (measurement) at first step
  ymod0     = y0;              % initially, model = plant output
  ymod(1)   = ymod0;
```

```
for k = 1:length(time)-1;
   if k == 1;
      rmodvec = [0;0;0;rsp(1)];
      uvec = [u0];
      u(k) = dimc(qnum,qden,rmodvec,uvec);
      umodvec = [0;u0;u(k)];
      ymodvec = [0;0;y0];
      ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec);
   elseif k == 2;
      rmodvec = [0;0;rsp(1);rsp(2)-distmod(2)];
      uvec = [u(k-1)];
      u(k) = dimc(qnum,qden,rmodvec,uvec);
       umodvec = [u0;u(k-1);u(k)];
       ymodvec = [y0;y0;ymod(k)];
       ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec);
   elseif k == 3;
       rmodvec = [0;rsp(k-2:k)-distmod(k-2:k)'];
       uvec =  [u(k-1)];
       u(k) = dimc(qnum,qden,rmodvec,uvec);
       umodvec = [u(k-2);u(k-1);u(k)];
       ymodvec = [y0;ymod(k-1);ymod(k)];
       ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec);
```

```
 else
        rmodvec = rsp(k-3:k)-distmod(k-3:k)';
        uvec = [u(k-1)];
        u(k) = dimc(qnum,qden,rmodvec,uvec);
        umodvec = [u(k-2);u(k-1);u(k)];
        ymodvec = [ymod(k-2);ymod(k-1);ymod(k)];
        ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec);
    end
     [tdummy,xdummy] = ode45('linodepar',[time(k) time(k+1)],xdis(:,k),[],a,b,u(k));
     ndum = length(tdummy);
     xdis(:,k+1) = xdummy(ndum,:);   % plant state
     ydis(k+1) = c*xdis(:,k+1);      % plant output (measured)
     distmod(k+1) = ydis(k+1) - ymod(k+1);
     tplot = [tplot;tdummy];
     yplot = [yplot;xdummy*c'];
  end
%
  u(k+1) = u(k);    % makes the input vector the same length as the time vector
```

```
%
 [tt,uu] = stairs(time,u);
 [ttrsp,rrsp] = stairs(time,rsp);
 figure(211)
 subplot(2,1,1)
 plot(tplot,yplot,ttrsp,rrsp,'--')
 legend('y','sp')
 ylabel('y')
 title('three tank process, IMC implementation')
 subplot(2,1,2)
 plot(tt,uu)
 xlabel('t, min')
 ylabel('u')
 %  save vectors for plotting
  tplot1 = tplot; yplot1 = yplot; tt1 = tt; uu1 = uu; rrsp1 = rrsp; ttrsp1 = ttrsp;
 % ------------------------------------------------------
```

```
---------------------------------------------------------
 lambda = 0.5
 alpha = exp(-delt/lambda)
 qnum = 16.414*(1-alpha).*[1 -1.8195 1.1035  -0.2231];  % IMC numerator
 qden = [1 -alpha]                  % IMC denominator
 modnum = [0.01439  0.03973 0.006794]          % IM numerator
 modden = [1  -1.82  1.104  -0.2231]       % IM denominator
%
% now, set up a for loop and integrate over 0.5 minute time steps
% currently set up for single input - single output
%
 tplot = [];
 yplot = [];  % save more points for plotting in between samples
 tbeg = 0;    % simulation start time
 tend = 5;   % simulation end time
 time = tbeg:delt:tend;       % generate the time vector
 rsp = [zeros(1,1);ones(length(time)-1,1)]; % setpoint change at second step
 x0  = zeros(3,1);
 y0  = 0;
 u0  = 0;
 xdis(:,1) = x0;            % discrete plant state
 ydis(1)   = y0;            % discrete plant output (measurement) at first step
 ymod0     = y0;              % initially, model = plant output
 ymod(1)   = ymod0;
```

```
for k = 1:length(time)-1;
   if k == 1;
      rmodvec = [0;0;0;rsp(1)];
      uvec = [u0];
      u(k) = dimc(qnum,qden,rmodvec,uvec);
      umodvec = [0;u0;u(k)];
      ymodvec = [0;0;y0];
      ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec);
   elseif k == 2;
      rmodvec = [0;0;rsp(1);rsp(2)-distmod(2)];
      uvec = [u(k-1)];
      u(k) = dimc(qnum,qden,rmodvec,uvec);
       umodvec = [u0;u(k-1);u(k)];
       ymodvec = [y0;y0;ymod(k)];
       ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec);
   elseif k == 3;
       rmodvec = [0;rsp(k-2:k)-distmod(k-2:k)'];
       uvec =  [u(k-1)];
       u(k) = dimc(qnum,qden,rmodvec,uvec);
       umodvec = [u(k-2);u(k-1);u(k)];
       ymodvec = [y0;ymod(k-1);ymod(k)];
       ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec);
   else
```

```
 rmodvec = rsp(k-3:k)-distmod(k-3:k)';
        uvec = [u(k-1)];
        u(k) = dimc(qnum,qden,rmodvec,uvec);
        umodvec = [u(k-2);u(k-1);u(k)];
        ymodvec = [ymod(k-2);ymod(k-1);ymod(k)];
        ymod(k+1) = dimcmod(modnum,modden,ymodvec,umodvec);
    end
     [tdummy,xdummy] = ode45('linodepar',[time(k) time(k+1)],xdis(:,k),[],a,b,u(k));
     ndum = length(tdummy);
     xdis(:,k+1) = xdummy(ndum,:);   % plant state
     ydis(k+1) = c*xdis(:,k+1);      % plant output (measured)
     distmod(k+1) = ydis(k+1) - ymod(k+1);
     tplot = [tplot;tdummy];
     yplot = [yplot;xdummy*c'];
  end
%
  u(k+1) = u(k);    % makes the input vector the same length as the time vector

%
  [tt,uu] = stairs(time,u);
  [ttrsp,rrsp] = stairs(time,rsp);
```

Function File: dimc.m

```
 function u = dimc(qnum,qden,rmodvec,uprev)
% discrete IMC -- used in MPC 2013 -- hw3
% b.w. bequette, 18 Sept 2013
%   rmodvec = modified setpoint vector, oriented rmod(k-2), rmod(k-1),
%             rmod(k)   (from top down)
%   uprev   = vector of previous manipulated inputs, oriented
%             uprev(k-2), uprev(k-1) (from top down)
% the numerator and denominator polynomials are oriented in the
%   opposite direction as the modified setpoint and manipulated
%   input vectors, so rback and uback are formed to reverse that
%   order
  rback = rmodvec(length(rmodvec):-1:1);   % reverse order
  uback = uprev(length(uprev):-1:1);       % reverse order
  u = qnum*rback -qden(2:length(qden))*uback;
```

Function File: dimcmod.m

```
 function ymod = dimcmod(modnum,modden,ymodvec,umodvec)
% calculates the next model output, given previous model
%   outputs and plant inputs
%   ymod(z) = gpmod(z)u(z)
%
% the model transfer function has the form
%              model numberator polynomial coefficients
%   gpmod =    ------------
%              model denominator polynomial coefficients
%
% the model output prediction has the form of
%   ymod(k+1) = modden(2)*ymod(k) + modden(3)*ymod(k-1) + ...
%              + modnum(1)*u(k) + modnum(2)*u(k-1) + ...
% where
%      modden(i) is a model denominator coefficient;
%        note that modden(1) is assumed to be 1
%      modnum(i) is a model numerator coefficient
%
% in our case, however, ymodvec is passed in as
%   [... ymodvec(k-2) ymodvec(k-1) ymodvec(k)]
% also, umodvec is passed in as
%   [... umodvec(k-2) umodvec(k-1) umodvec(k)]
```

Function File: linodepar.m

```
function xdot = linodepar(t,x,flag,a,b,u)
%
% pass through the state space a and b matrices
% pass through the input vector
  xdot = a*x + b*u;
```