Student Workshop
Digital PID applied to 3-Tank Process

- Find discrete time model
- Digital PID
- Tune for Verge of Instability (continuous oscillations)
  - Find closed-loop poles
- Increase gain for unstable closed-loop
  - Find closed-loop poles

Consider the following state space model for 3 tanks in series, used in previous Workshop.

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

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. Feel free to use MATLAB and the Control Toolbox for the following problems.

1. For a sample time of 0.5 minutes, find the discrete transfer function in discrete pole-zero form. What are the values of the poles and zeros? You should find that one zero is outside the unit circle (magnitude greater than 1). Compare the discrete step response for this sample time with the continuous system step response.

```
% 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
 [yc,tc] = step(lintank);       % generate continuous step response
```

```
%   Problem 1
% discrete time model
  delt = 0.5;                    % sample time of 0.5 minutes
  tankssz = c2d(lintank,delt,'zoh')  % create discrete state space model from continuous
  [yd,td] = step(tankssz);          % generate step response of discrete model
  figure(11)
  plot(tc,yc,'k',td,yd,'k--')       % compare step responses of continuous and discrete models
  xlabel('t, min')
  ylabel('y')
  legend('continuous','discrete')
  title('continuous vs. discrete, three tank step response')
% 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)
```

Polynomial transfer function form:

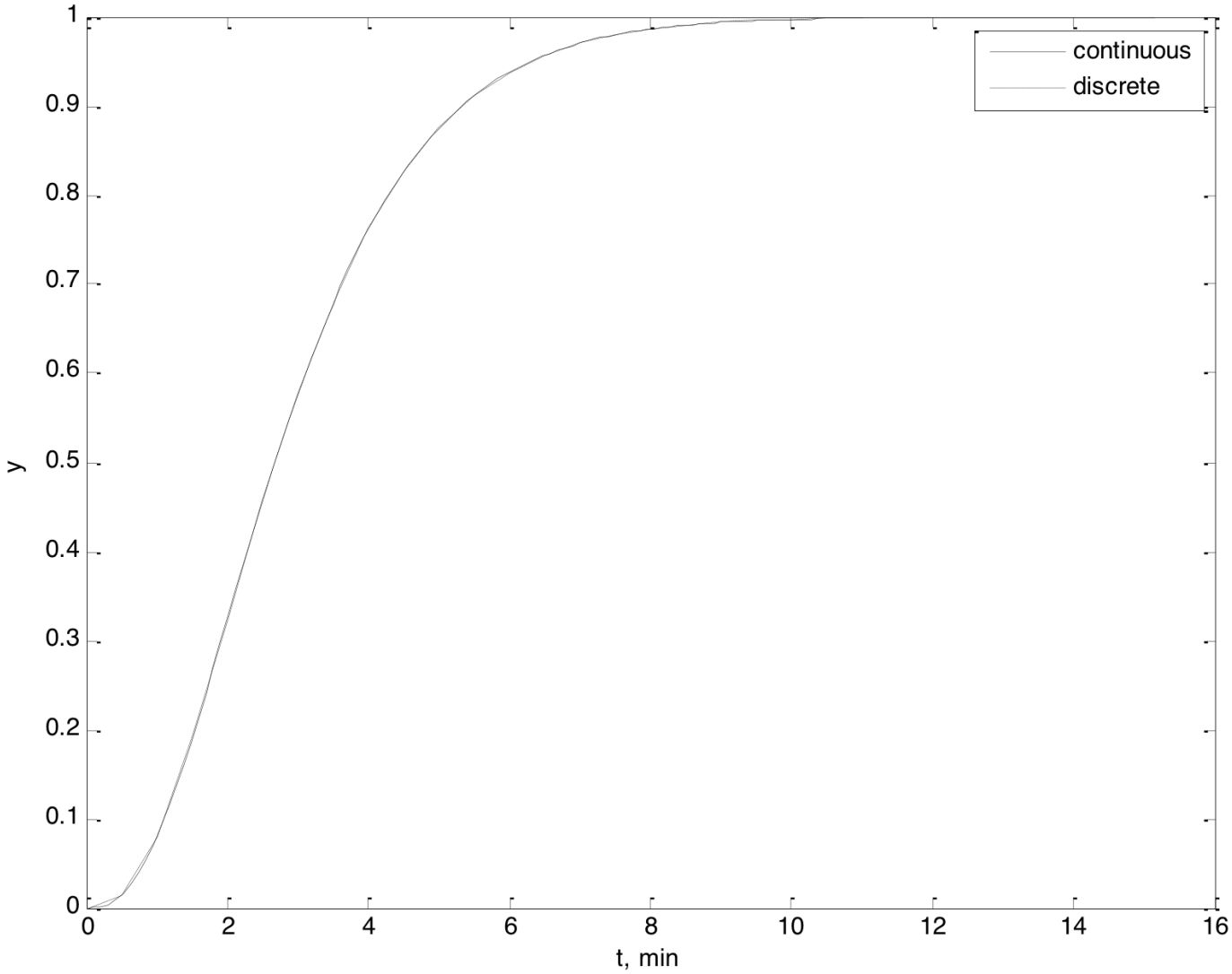
$$\frac{0.01439\,z^2 + 0.03973\,z + 0.006794}{z^3 - 1.82\,z^2 + 1.104\,z - 0.2231}$$

Zero/pole/gain:

$$\frac{0.014388\,(z+2.579)\,(z+0.1831)}{(z-0.6065)^3}$$

So, there are three poles at 0.6065 (stable), and zeros at -2.579 and -0.1831. Clearly, one zero is outside the unit circle (magnitude 2.579).

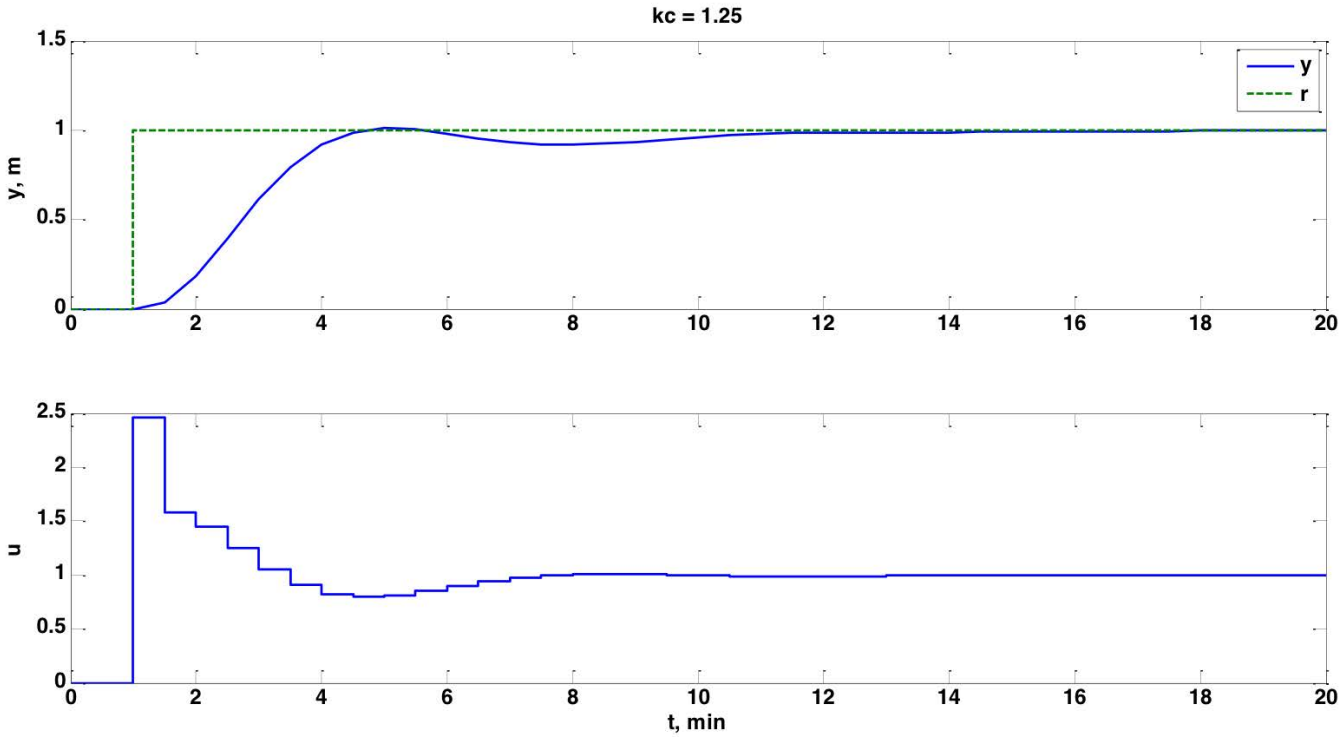
The continuous and discrete step responses are indistinguishable, as shown on the next slide, when the [yd,td] = step(tankssz) command is used. If the results are plotted directly, however, using step(tankssz), the result is a staircase because of the zero-order hold.


continuous
discrete
y
t, min
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
2
4
6
8
10
12
14
16

2. Design a digital *PID controller* with a sample time of 0.5 minutes. Use a continuous representation of the *plant* for simulations (using ode45), but implement a discrete controller on the simulated plant that you developed in Homework 1. The initial conditions for the input and output (and all states) are 0. Make a setpoint change to a value of r = 1 at t = 1 minute.

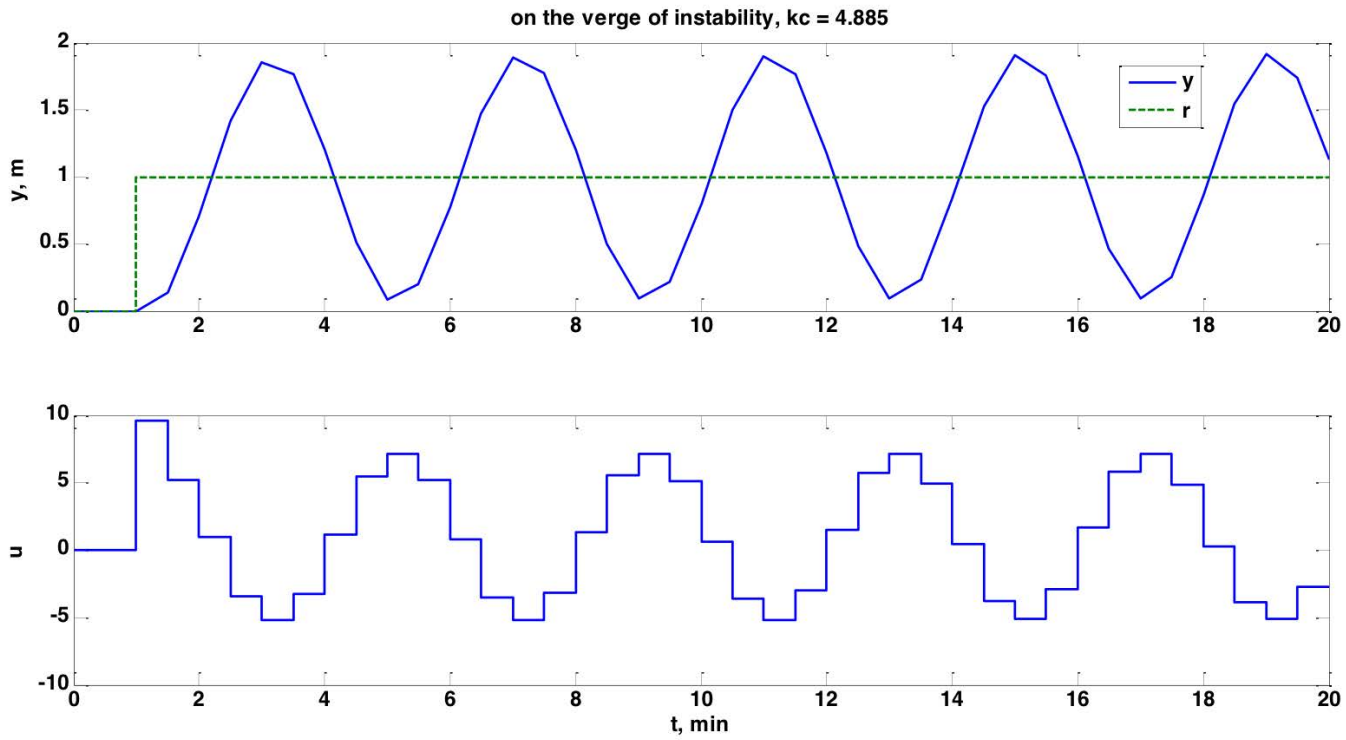
I suggest an IMC-based PID controller based on a first-order + deadtime approximation to the third-order process, with t_I = 3 minutes and t_D = 0.4 minutes. Adjust k_c (proportional gain) to obtain desirable performance.

Use subplot(2,1,1) to plot the setpoint and output on a "upper plot" and subplot(2,1,2) to plot the manipulated input on a "lower plot." The plots are shown below, for k_c = 1.25 (m^3/min/m, with t_I = 3 minutes and t_D = 0.4 minutes).


kc = 1.25
y, m
u
t, min
y
r
1.5
1
0.5
0
2.5
2
1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
20

3. Now, adjust k_c until the closed-loop system enters a continuous oscillation, on the verge of instability (constant amplitude output and manipulated input). Calculate the discrete closed-loop poles for this value of k_c. Is this result consistent with your simulation results?

A value of k_c = 4.885 (with t_I = 3 minutes and t_D = 0.4 minutes) results in a closed-loop that is on the verge of instability, as shown below.



The closed-loop poles are the solution to $1 + g_c(z)g_c(z) = 0$. Written in terms of the numerator and denominator polynomials, this is

$$1 + \frac{N_p(z)}{D_p(z)} \cdot \frac{N_c(z)}{D_c(z)} = 0$$

Which we can write as

$$D_p(z)D_c(z) + N_p(z)N_c(z) = 0$$

Which, for $k_c = 4.885$, has the polynomial

$$1.0000z^5 - 2.6814z^4 + 3.1222z^3 - 1.7099z^2 + 0.2921z + 0.0266 = 0$$

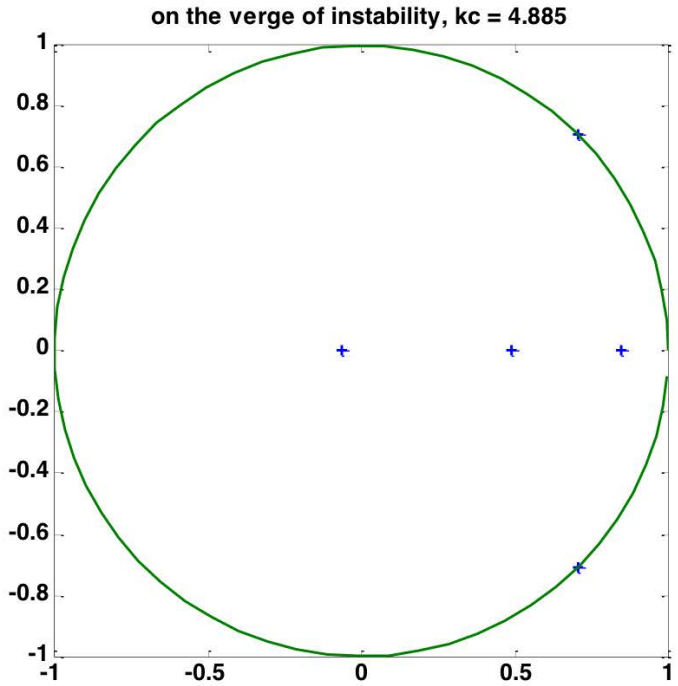
The five roots of this polynomial are

0.7046 + 0.7095i
0.7046 - 0.7095i
0.8449
0.4912
-0.0640

Which have the magnitudes

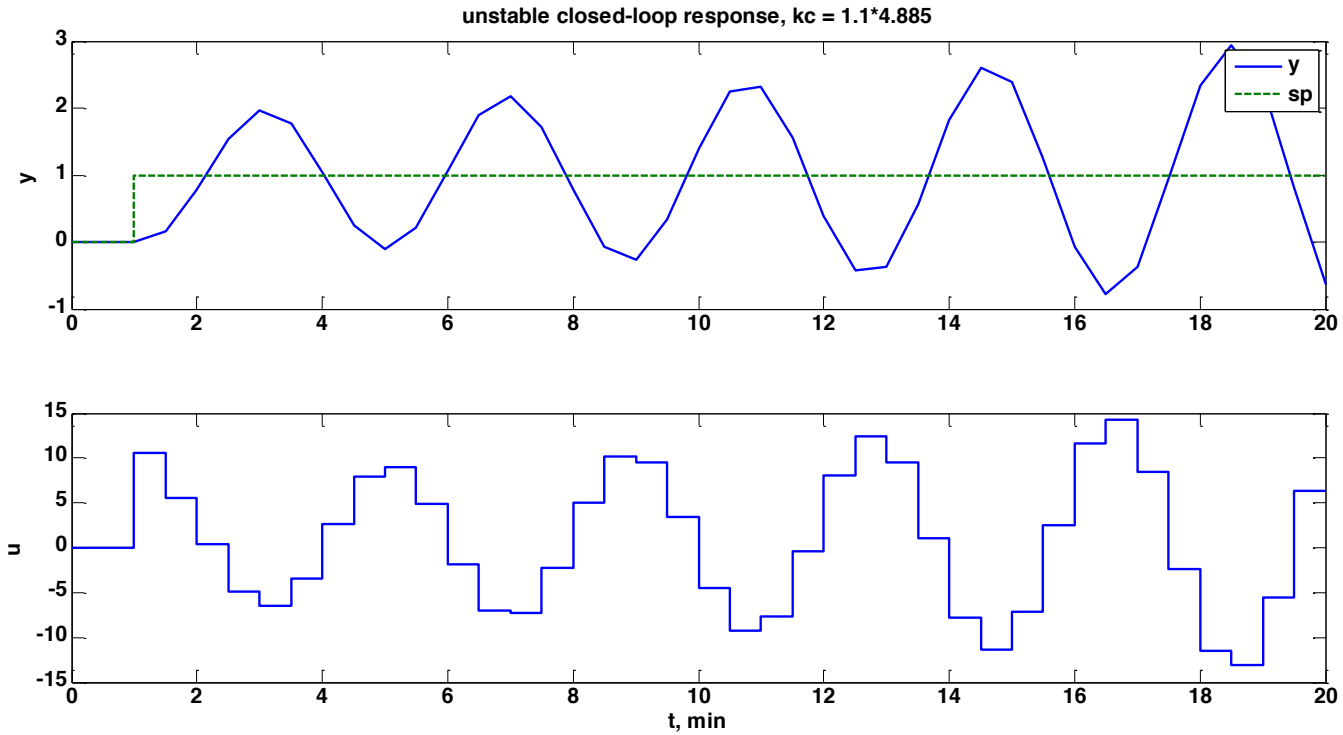
1.0000
1.0000
0.8449
0.4912
0.0640

So, clearly, two roots are on the verge of instability. We can see the magnitudes by plotting the poles on the following complex-pole plot

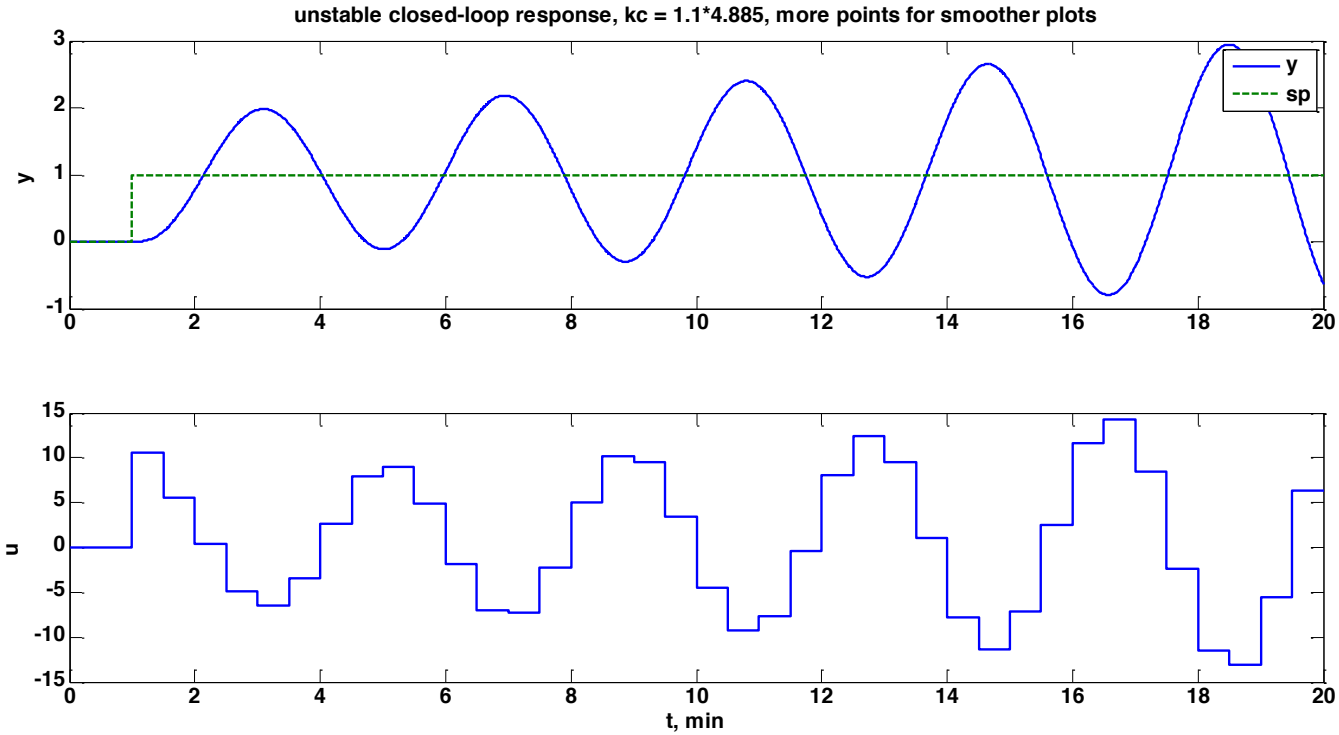


4. Further increase k_c by 10% and show the simulation results. What are the values of the closed-loop poles for this case?

The plots for kc = 1.1*4.885 are shown below.



Notice that the output plots are somewhat "jagged." In the m-file I show how to plot additional time points between the sample time to make the output curves look smoother, as shown in the plot below.


unstable closed-loop response, kc = 1.1*4.885, more points for smoother plots
y
sp
u
t, min

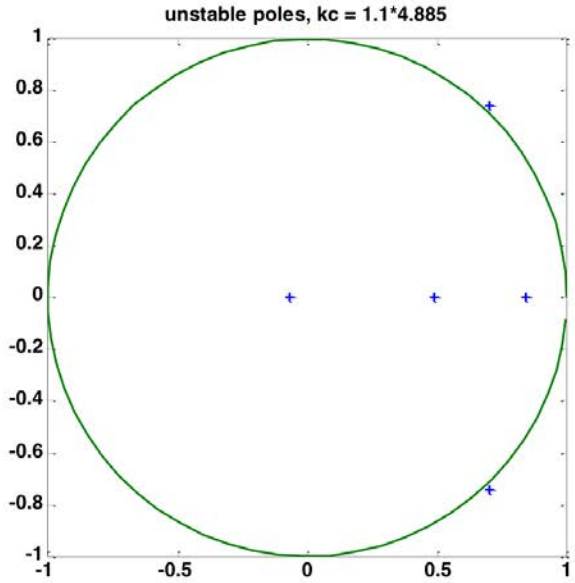
The closed-loop poles are

0.7001 + 0.7438i
0.7001 - 0.7438i
0.8439
0.4909
-0.0676

Which have the magnitudes:

1.0215
1.0215
0.8439
0.4909
0.0676

Clearly, there are two unstable closed-loop poles. We can also see this on the following pole plot:



MATLAB Script: Problems 2-4

```
%   Problem 2
% setup closed-loop simulation
  x0 = zeros(3,1);              % state vector initial condition
  y0 = c*x0;                    % initial output
  u0 =   0;                     % initial input
  tbeg = 0;                     % simulation start time
  tend =20;                     % simulation end time
  kc = 1.25;                    % controller proportional gain
  taui = 3;                     % controller integral time
  taud = 0.4;                   % controller derivative time
  par = [kc;taui;taud];         % tuning parameter vector
%
% now, set up a for loop and integrate over 0.5 minute time steps
% currently set up for single input - single output
%
  delt = 0.5;                   % sample time = 0.5 minutes
  time = tbeg:delt:tend;        % generate the time vector
  rsp = [zeros(2,1);ones(length(time)-2,1)]; % setpoint change at third time step
  xdis(:,1) = x0;               % initial state vector for discrete time simulation
  ydis(1)   = y0;               % initial output for discrete time simulation
```

```
   for k = 1:length(time)-1;         % k = index for simulation
%
   if k == 1;
      rvec = [0;0;rsp(1)];
      yvec = [0;0;y0];
      u(k) = dpid(delt,par,rvec,yvec,u0);
   elseif k == 2;
      rvec = [0;rsp(1);rsp(2)];
      yvec = [0;y0;ydis(1)];
       u(k) = dpid(delt,par,rvec,yvec,u(k-1));
   else
       rvec = rsp(k-2:k);
       yvec = ydis(k-2:k);
       u(k) = dpid(delt,par,rvec,yvec,u(k-1));
   end
 %
    [tdummy,xdummy] = ode45('linodepar',[time(k) time(k+1)],xdis(:,k),[],a,b,u(k));
    ndum = length(tdummy);
    xdis(:,k+1) = xdummy(ndum,:)';
    ydis(k+1) = c*xdis(:,k+1);
 end
%
 u(k+1) = u(k);    % makes the input vector the same length as the time vector
```

```
figure(12)
subplot(2,1,1)
plot(time,ydis,time,rsp,'--')
title('setpoint response, PID control')
xlabel('t, min')
ylabel('y, m')
legend('y','r')
subplot(2,1,2)
plot(time,u)
xlabel('t, min')
ylabel('u’)
% prefer zero order hold on inputs
[tt,uu] = stairs(time,u);          % zero-order hold on input
[ttrsp,rrsp] = stairs(time,rsp);    % zero-order hold on setpoint
figure(13)
subplot(2,1,1)
plot(time,ydis,ttrsp,rrsp,'--')
legend('y','r')
ylabel('y, m')
title('kc = 1.25')
subplot(2,1,2)
plot(tt,uu)
xlabel('t, min')
ylabel('u')
```

```
%  Problem 3 - simulation results
%           vary kc until closed-loop is on the verge of instability
  kc = 4.885;
%
  par = [kc;taui;taud];      % tuning parameter vector (passed into dpid function)
%
% now, set up a for loop and integrate over 0.5 minute time steps
% currently set up for single input - single output
%
  delt = 0.5;  % sample time = 0.5 minutes
  time = tbeg:delt:tend;                  % generate the time vector
  rsp = [zeros(2,1);ones(length(time)-2,1)]; % setpoint change at third step
  xdis(:,1) = x0;                         % discrete state initial condition
  ydis(1)   = y0;                         % output initial condition
```

```
for k = 1:length(time)-1;
   if k == 1;
      rvec = [0;0;rsp(1)];
      yvec = [0;0;y0];
      u(k) = dpid(delt,par,rvec,yvec,u0);
   elseif k == 2;
      rvec = [0;rsp(1);rsp(2)];
      yvec = [0;y0;ydis(1)];
       u(k) = dpid(delt,par,rvec,yvec,u(k-1));
   else
       rvec = rsp(k-2:k);
       yvec = ydis(k-2:k);
       u(k) = dpid(delt,par,rvec,yvec,u(k-1));
   end
    [tdummy,xdummy] = ode45('linodepar',[time(k) time(k+1)],xdis(:,k),[],a,b,u(k));
    ndum = length(tdummy);
    xdis(:,k+1) = xdummy(ndum,:)';
    ydis(k+1) = c*xdis(:,k+1);
 end
%
 u(k+1) = u(k);    % makes the input vector the same length as the time vector
```

```
%
% prefer zero order hold on inputs
%
  [tt,uu] = stairs(time,u);          % zero-order hold on input
  [ttrsp,rrsp] = stairs(time,rsp);    % zero-order hold on setpoint
%
  figure(14)
  subplot(2,1,1)
  plot(time,ydis,ttrsp,rrsp,'--')
  legend('y','r')
  ylabel('y, m')
  title('on the verge of instability, kc = 4.885')
  subplot(2,1,2)
  plot(tt,uu)
  xlabel('t, min')
  ylabel('u')
```

```
%   Problem 3 - find the closed-loop poles (poles of 1 + gc*gp)
%  first, coefficients of controller numerator (different order than some notes)
%        uses kc from previous simulation
  bc2 = kc*(1+(delt/taui)+(taud/delt));
  bc1 = -kc*(1+2*(taud/delt));
  bc0 = kc*taud/delt;
%
  numc = [bc2 bc1 bc0];          % numerator of controller transfer function
  denc = [1   -1    0];          % denominator of controller transfer function
%                              process numerator and denominator were found in problem 1
  denpdenc = conv(denp,denc)      % multiply process and controller denominators
  numpnumc = conv(nump,numc)      % multiply process and controller numerators
  denpdenc + numpnumc             % add terms
  clpoles = roots(denpdenc + numpnumc)       % find closed-loop poles
  abs(clpoles)   % calculates the magnitude of each pole (important for complex conj poles)
%  create a unit circle
  angle = 0:0.1:2*pi;
  xcirc = cos(angle);
  ycirc = sin(angle);
%  plot the closed-loop poles, superimposed on the unit circle
  figure(16)
  plot(real(clpoles),imag(clpoles),'+',xcirc,ycirc)
  axis square                     % make a square plot
  title('on the verge of instability, kc = 4.885')
```

```
%     Problem 4 - simulation for unstable tuning
    kc = kc*1.1;              % increase previous kc by 10%
    par = [kc;taui;taud];     % tuning parameter vector
% now, set up a for loop and integrate over 0.5 minute time steps
% currently set up for single input - single output
  delt = 0.5;  % sample time = 0.5 minutes
  time = tbeg:delt:tend;      % generate the time vector
  rsp = [zeros(2,1);ones(length(time)-2,1)]; % setpoint change at third step
  xdis(:,1) = x0;
  ydis(1)   = y0;
  for k = 1:length(time)-1;
    if k == 1;
       rvec = [0;0;rsp(1)];
       yvec = [0;0;y0];
       u(k) = dpid(delt,par,rvec,yvec,u0);
    elseif k == 2;
       rvec = [0;rsp(1);rsp(2)];
       yvec = [0;y0;ydis(1)];
        u(k) = dpid(delt,par,rvec,yvec,u(k-1));
    else
        rvec = rsp(k-2:k);
        yvec = ydis(k-2:k);
        u(k) = dpid(delt,par,rvec,yvec,u(k-1));
    end
```

```
 [tdummy,xdummy] = ode45('linodepar',[time(k) time(k+1)],xdis(:,k),[],a,b,u(k));
     ndum = length(tdummy);
     xdis(:,k+1) = xdummy(ndum,:)';
     ydis(k+1) = c*xdis(:,k+1);
  end
%
  u(k+1) = u(k);    % makes the input vector the same length as the time vector
%
% prefer zero order hold on inputs
%
  [tt,uu] = stairs(time,u);
  [ttrsp,rrsp] = stairs(time,rsp);
  figure(18)
  subplot(2,1,1)
  plot(time,ydis,ttrsp,rrsp,'--')
  legend('y','sp')
  ylabel('y')
  title('unstable closed-loop response, kc = 1.1*4.885')
  subplot(2,1,2)
  plot(tt,uu)
  xlabel('t, min')
  ylabel('u')
```

```
%   Problem 4 - find the closed-loop poles (poles of 1 + gc*gp)
%  first, coefficients of controller numerator
%
  bc2 = kc*(1+(delt/taui)+(taud/delt));
  bc1 = -kc*(1+2*(taud/delt));
  bc0 = kc*taud/delt;
%
  numc = [bc2 bc1 bc0];
  denc = [1   -1    0];
%
  denpdenc = conv(denp,denc)
  numpnumc = conv(nump,numc)
  denpdenc + numpnumc
  clpoles = roots(denpdenc + numpnumc)      % find closed-loop poles
  abs(clpoles)
%     create unit circle
  angle = 0:0.1:2*pi;
  xcirc = cos(angle);
  ycirc = sin(angle);
%
  figure(19)
  plot(real(clpoles),imag(clpoles),'+',xcirc,ycirc)
  axis square
  title('unstable poles, kc = 1.1*4.885')
```

```
%  Again, simulating problem 4 (unstable), using more points to make the plots smoother
  tplot = [];  % new vectors for plotting that will yield smoother plots
  yplot = [];
%
  delt = 0.5;  % sample time = 0.5 minutes
  time = tbeg:delt:tend;       % generate the time vector
  rsp = [zeros(2,1);ones(length(time)-2,1)]; % setpoint change at third step
  xdis(:,1) = x0;
  ydis(1)   = y0;
  for k = 1:length(time)-1;
    if k == 1;
       rvec = [0;0;rsp(1)];
       yvec = [0;0;y0];
       u(k) = dpid(delt,par,rvec,yvec,u0);
    elseif k == 2;
       rvec = [0;rsp(1);rsp(2)];
       yvec = [0;y0;ydis(1)];
        u(k) = dpid(delt,par,rvec,yvec,u(k-1));
    else
        rvec = rsp(k-2:k);
        yvec = ydis(k-2:k);
        u(k) = dpid(delt,par,rvec,yvec,u(k-1));
    end
```

```
    [tdummy,xdummy] = ode45('linodepar',[time(k) time(k+1)],xdis(:,k),[],a,b,u(k));
    ndum = length(tdummy);
    xdis(:,k+1) = xdummy(ndum,:)';
    ydis(k+1) = c*xdis(:,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
%
% prefer zero order hold on inputs
%
  [tt,uu] = stairs(time,u);
  [ttrsp,rrsp] = stairs(time,rsp);
  figure(20)
  subplot(2,1,1)
  plot(tplot,yplot,ttrsp,rrsp,'--')
  legend('y','sp')
  ylabel('y')
  title('unstable closed-loop response, kc = 1.1*4.885, more points for smoother plots')
  subplot(2,1,2)
  plot(tt,uu)
  xlabel('t, min')
  ylabel('u')
```

Function Files

```
function u = dpid(delt,par,r,y,uprev)
% digital pid
% b.w. bequette
% 5 sept 07
% 10 sept 09 - revised for more logical b subscripts
% 12 sept 13 - used in homework 2, MPC course
%     y is a vector of the previous outputs (incl current)
%     uprev is the previous (step k-1) manipulated input
%
  kc = par(1);
  taui = par(2);
  taud = par(3);
  b2 = kc*(1+(delt/taui)+(taud/delt));
  b1 = -kc*(1+2*(taud/delt));
  b0 = kc*taud/delt;
%
  ekm2 = r(1) - y(1);  % step k-2
  ekm1 = r(2) - y(2);  % step k-1
  ek   = r(3) - y(3);  % step k
  u = uprev + b2*ek + b1*ekm1 + b0*ekm2;
```

```
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;
```