E6225 Continious Assignment 2

Msc CCA

Consider a temperature control problem of Tyreus case 4 (T4), the transfer function matrix is obtained by empirical modeling technique as:

Try to design following controllers using ETF and compare with the classical decentralized control based on BLT method

1. Decentralized control
2. Sparse control
3. Decoupling control

Compare the performance for the control systems by simulating the closed-loop performances for step response R1(t)=1 for t>0, R2(t)=1 for t>250 and R3(t)=1 for t>500.

# Calculate The Equivalent Transfer Functions for

1. **Loop Pairing**

Steady-state gain and Relative Gain Array (RGA) :

According to the Loop paring rules, select 1-1、2-3、3-2, and rearrange :

Since , the loop paring for the system is stable.

1. **Equivalent Transfer Function Based on RNGA Approximation**

In the following sections, I use and in place of the original ones to get the diagnal paring in . Thus,

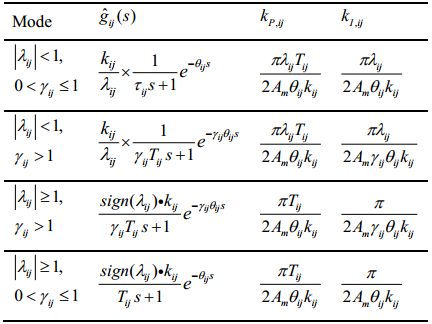
Steady-state gain and Relative Gain Array (RGA) :

Normalized Gain and Relative Normalized Gain Array (RNGA) :

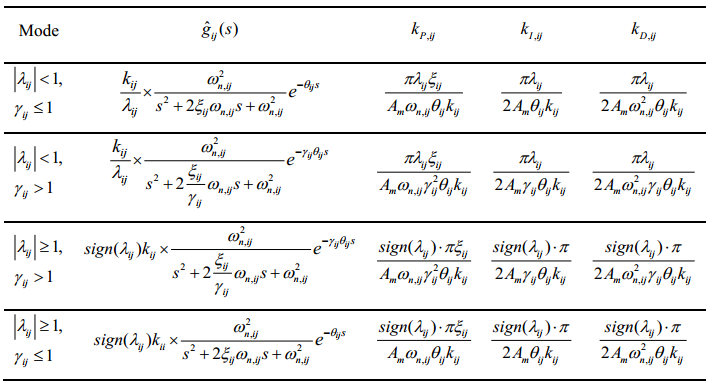
Relative Average Residence Time Array (RARTA) :

1. **Determine Equivalent Transfer Functions with Integrity Rules**

**Table 1 EFTs and PI parameters for FOPDT process**



**Table 2 EFTs and PI parameters for SOPDT process**



According to Table 1 and Table 2, the ETFs of can be calculated as

1. **Determine Equivalent Transfer Functions without Integrity Rules**

For FOPTD, we have

For SOPTD, we have

Therefore,

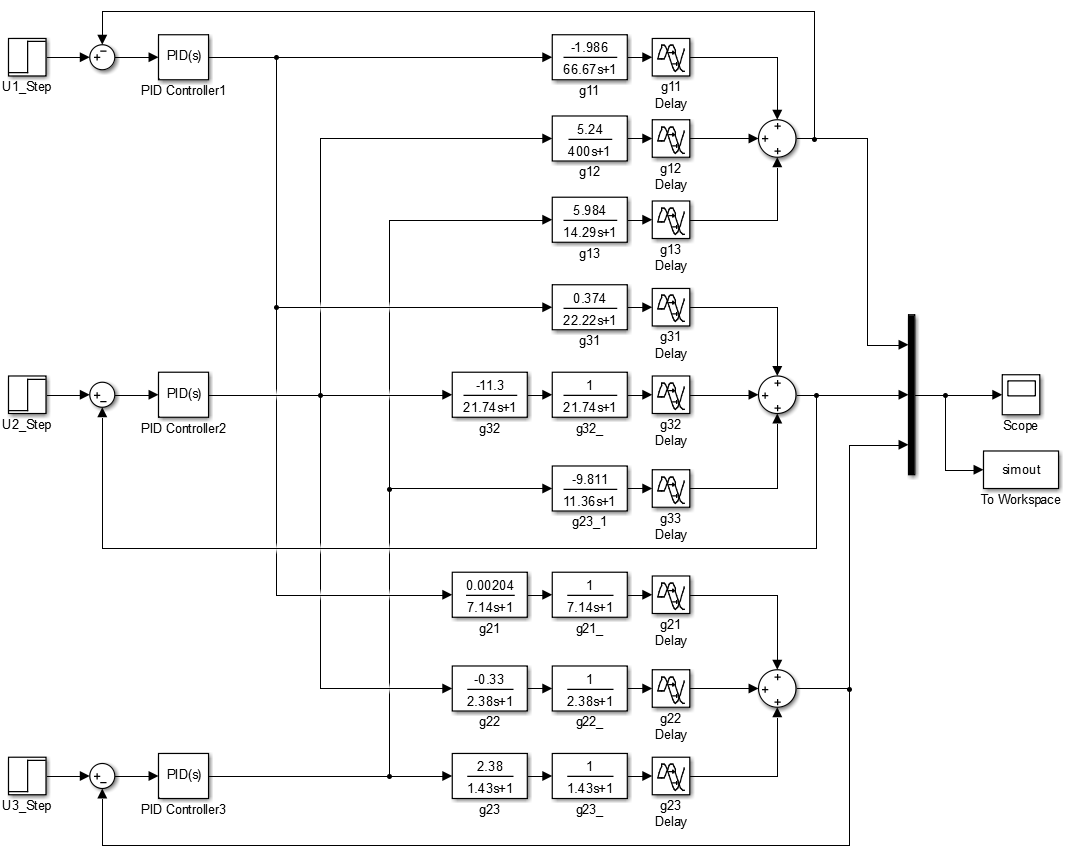
1. **Matlab Code**

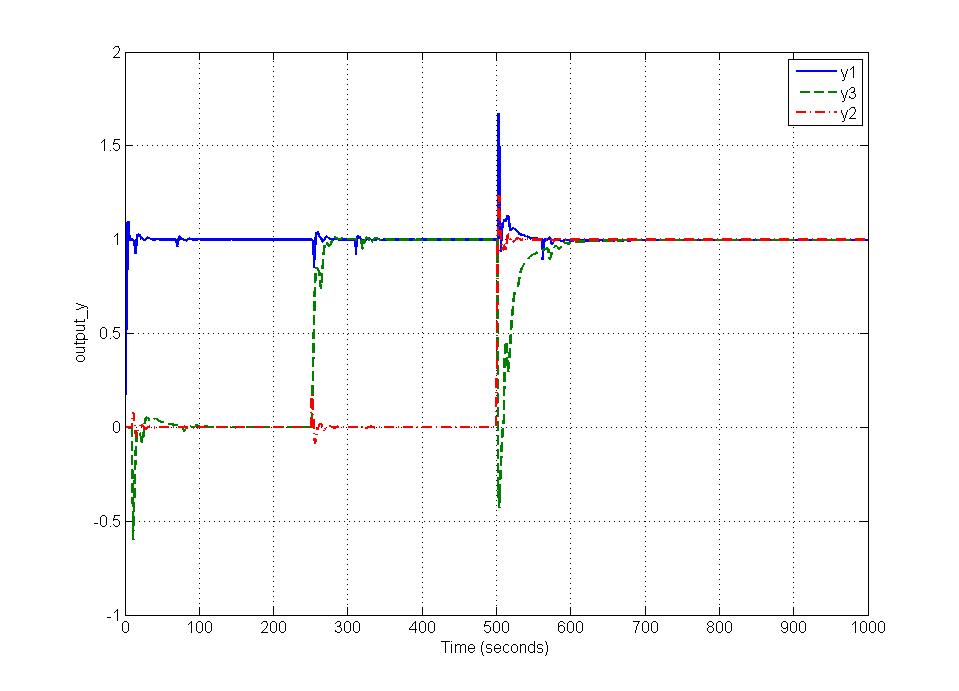
|  |
| --- |
| %select y1-u1¡¢y3-u2¡¢y2-u3, rearrange K for NI calculation  G = [tf([-1.986],[66.67 1],'inputdelay',0.71), tf([5.24],[400 1],'inputdelay',60), tf([5.984],[14.29 1],'inputdelay',2.24);  tf([0.374],[22.22 1],'inputdelay',7.75), tf([-11.3],[21.74^2 21.74\*2 1],'inputdelay',3.79), tf([-9.811],[11.36 1],'inputdelay',1.59);  tf([0.00204],[7.14^2 7.14\*2 1],'inputdelay',0.59), tf([-0.33],[2.38^2 2.38\*2 1],'inputdelay',0.68), tf([2.38],[1.43^2 1.43\*2 1],'inputdelay',0.42)];    G\_size = size(G);  for i = 1:G\_size(1)  for j = 1:G\_size(2)  K(i,j) = G(i,j).num{1}(end);  L(i,j) = G(i,j).InputDelay + G(i,j).ioDelay;  T(i,j) = G(i,j).den;  if size(T{i,j}) == [1 2]  K\_N(i,j) = K(i,j)/( T{i,j}(1)+L(i,j));  else  K\_N(i,j) = K(i,j)/( T{i,j}(2)+L(i,j));  end  end  end  %RGA  Lambda = K.\*(inv(K))';  NI = det(K)/(K(1,1)\*K(2,2)\*K(3,3));  %RNGA  Lambda\_N = K\_N.\*(inv(K\_N))';  Gamma=Lambda\_N./Lambda;    %With Integrity Rules  [ETF\_IR, PID\_IR] = eft\_PID\_integrity\_rules(G,Lambda,Gamma);  %Without Integrity Rules  ETF = eft(G,Lambda,Gamma); |
| function[G\_hat,PID]= eft\_PID\_integrity\_rules(G, Lambda, Gamma )  Am = 3;  G\_size = size(G);  PID = cell(G\_size(1),G\_size(2));  for i = 1:G\_size(1)  for j = 1:G\_size(2)  K(i,j) = G(i,j).num{1}(end);  L(i,j) = G(i,j).InputDelay + G(i,j).ioDelay;  T(i,j) = G(i,j).den;  if size(T{i,j}) == [1 2]  if abs(Lambda(i,j)) < 1  if Gamma(i,j) <= 1 && Gamma(i,j) > 0  G\_hat(i,j)=tf([K(i,j)/Lambda(i,j)],[T{i,j}(1) 1],'inputdelay',L(i,j));  PID{i,j}={pi\*Lambda(i,j)\*T{i,j}(1)/2/Am/L(i,j)/K(i,j) pi\*Lambda(i,j)/2/Am/L(i,j)/K(i,j) 0};  else %Gamma(i,j) > 1  G\_hat(i,j)=tf([K(i,j)/Lambda(i,j)],[Gamma(i,j)\*T{i,j}(1) 1],'inputdelay',Gamma(i,j)\*L(i,j));  PID{i,j}={pi\*Lambda(i,j)\*T{i,j}(1)/2/Am/L(i,j)/K(i,j) pi\*Lambda(i,j)/2/Am/Gamma(i,j)/L(i,j)/K(i,j) 0};  end  else %abs(Lambda(i,j)) >= 1  if Gamma(i,j) > 1  G\_hat(i,j)=tf([sign(Lambda(i,j))\*K(i,j)],[Gamma(i,j)\*T{i,j}(1) 1],'inputdelay',Gamma(i,j)\*L(i,j));  PID{i,j}={pi\*T{i,j}(1)/2/Am/L(i,j)/K(i,j) pi/2/Am/Gamma(i,j)/L(i,j)/K(i,j) 0};  else %Gamma(i,j) <= 1 && Gamma(i,j) > 0  G\_hat(i,j)=tf([sign(Lambda(i,j))\*K(i,j)],[T{i,j}(1) 1],'inputdelay',L(i,j));  PID{i,j}={pi\*T{i,j}(1)/2/Am/L(i,j)/K(i,j), pi/2/Am/L(i,j)/K(i,j), 0};  end  end  else %size(T{i,j}) == [1 3]  if abs(Lambda(i,j)) < 1  if Gamma(i,j) <= 1  G\_hat(i,j)=tf([K(i,j)/Lambda(i,j)],[T{i,j}(1) T{i,j}(2) 1],'inputdelay',L(i,j));  PID{i,j}={pi\*Lambda(i,j)\*T{i,j}(2)/2/Am/L(i,j)/K(i,j) pi\*Lambda(i,j)/2/Am/L(i,j)/K(i,j) pi\*Lambda(i,j)\*T{i,j}(1)/2/Am/L(i,j)/K(i,j)};  else % Gamma(i,j) > 1  G\_hat(i,j)=tf([K(i,j)/Lambda(i,j)],[T{i,j}(1) T{i,j}(2)/Gamma(i,j) 1],'inputdelay',L(i,j)\*Gamma(i,j));  PID{i,j}={pi\*Lambda(i,j)\*T{i,j}(2)/2/Am/Gamma(i,j)^2/L(i,j)/K(i,j) pi\*Lambda(i,j)/2/Am/Gamma(i,j)/L(i,j)/K(i,j) pi\*Lambda(i,j)\*T{i,j}(1)/2/Am/Gamma(i,j)/L(i,j)/K(i,j)};  end  else %abs(Lambda(i,j)) >= 1  if Gamma(i,j) > 1  G\_hat(i,j)=tf([sign(Lambda(i,j))\*K(i,j)],[T{i,j}(1) T{i,j}(2)/Gamma(i,j) 1],'inputdelay',Gamma(i,j)\*L(i,j));  PID{i,j}={sign(Lambda(i,j))\*pi\*T{i,j}(2)/2/Am/Gamma(i,j)^2/L(i,j)/K(i,j) sign(Lambda(i,j))\*pi/2/Am/Gamma(i,j)/L(i,j)/K(i,j) sign(Lambda(i,j))\*pi\*T{i,j}(1)/2/Am/Gamma(i,j)/L(i,j)/K(i,j)};  else % Gamma(i,j) <= 1  G\_hat(i,j)=tf([sign(Lambda(i,j))\*K(i,j)],[T{i,j}(1) T{i,j}(2) 1],'inputdelay',L(i,j));  PID{i,j}={pi\*sign(Lambda(i,j))\*T{i,j}(2)/2/Am/L(i,j)/K(i,j) pi\*sign(Lambda(i,j))/2/Am/L(i,j)/K(i,j) pi\*sign(Lambda(i,j))\*T{i,j}(1)/2/Am/L(i,j)/K(i,j)};  end  end  end  end  end |
| function[G\_hat]=eft(G, Lambda, Gamma )  Am = 3;  G\_size = size(G);  for i = 1:G\_size(1)  for j = 1:G\_size(2)  K(i,j) = G(i,j).num{1}(end);  L(i,j) = G(i,j).InputDelay + G(i,j).ioDelay;  T(i,j) = G(i,j).den;  if size(T{i,j}) == [1 2]  G\_hat(i,j)=tf([K(i,j)/Lambda(i,j)],[Gamma(i,j)\*T{i,j}(1) 1],'inputdelay',Gamma(i,j)\*L(i,j));  else %size(T{i,j}) == [1 3]  G\_hat(i,j)=tf([K(i,j)/Lambda(i,j)],[T{i,j}(1) T{i,j}(2)/Gamma(i,j) 1],'inputdelay',L(i,j)\*Gamma(i,j));  end  end  end |

# Design Decentralized Control using ETF

According to Table 1 and 2, with and , the decentralized PID controller is

1. **Simulation**



****

As shown in the figure above, all three outputs converge to 1 responding to the three unit step inputs. Therefore, all three PID controllers are designed properly and successfully control the three outputs.

# Design Sparse Control using ETF

1. **Design the Structure of Sparse Control**

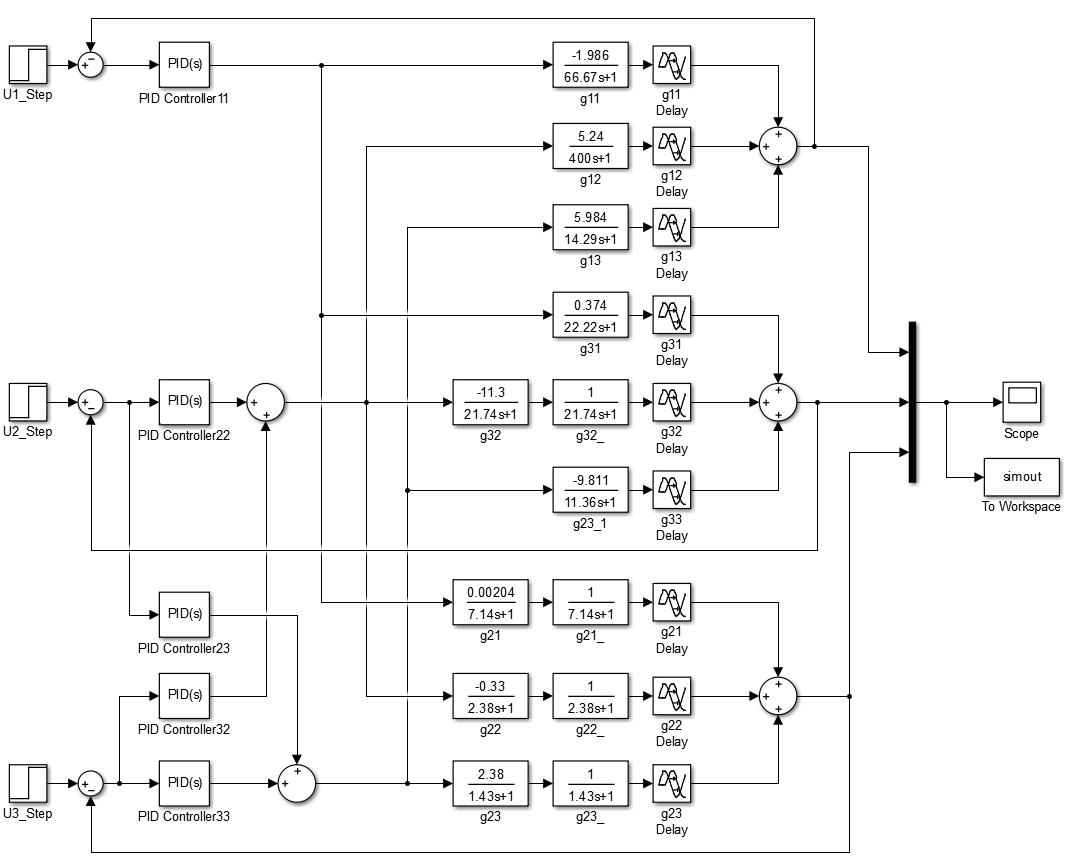
Based on the Interaction Index method, the index matrix *B* is calculated as

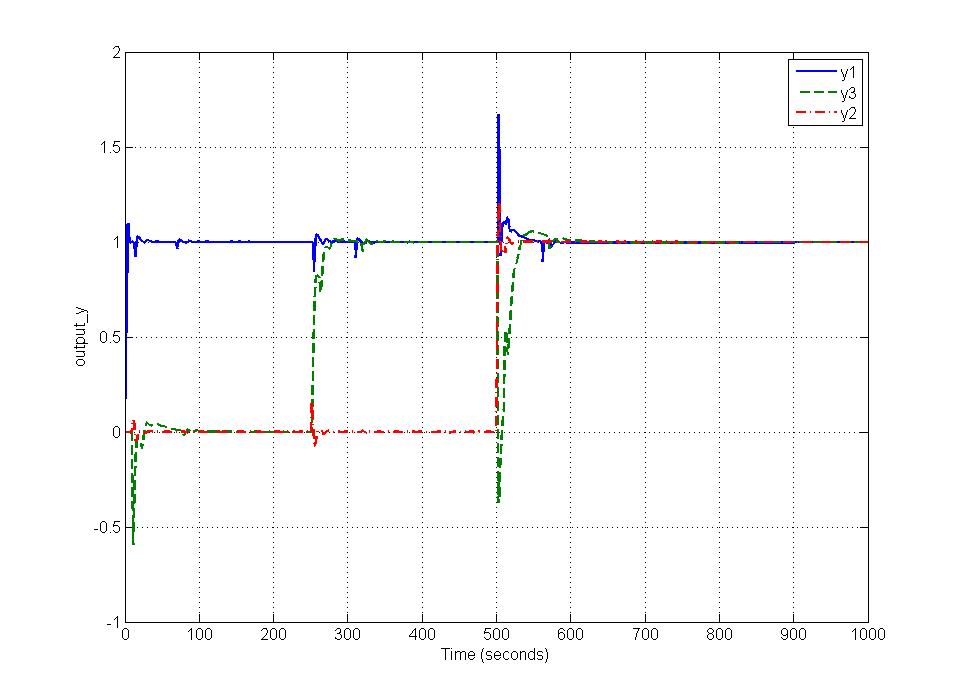
According to the control selection criterion , the sparse control should have the following structure

1. **Design Controllers based on Gain and Phase Margin Method**

According to Table 1 and 2, with and , the sparse PID controller is

1. **Simulation**



**

As shown in the figure above, all three outputs converge to 1 responding to the three unit step inputs. Therefore, all five PID controllers are designed properly and successfully control the three outputs.

# Design Decoupling Control using ETF with Integrity Rules

1. **Normalized Decoupling**

According to the normalized decoupling control system design rules, the forward transfer matrix is selected as

The decoupler can be obtained by

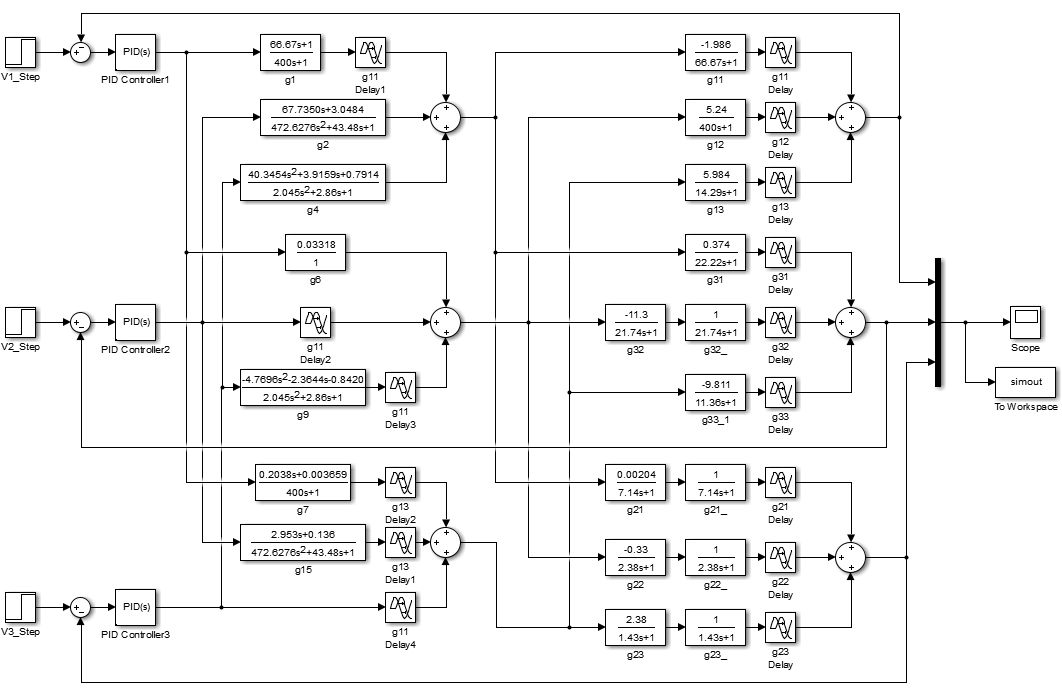
1. **Design Controllers based on Gain and Phase Margin Method**

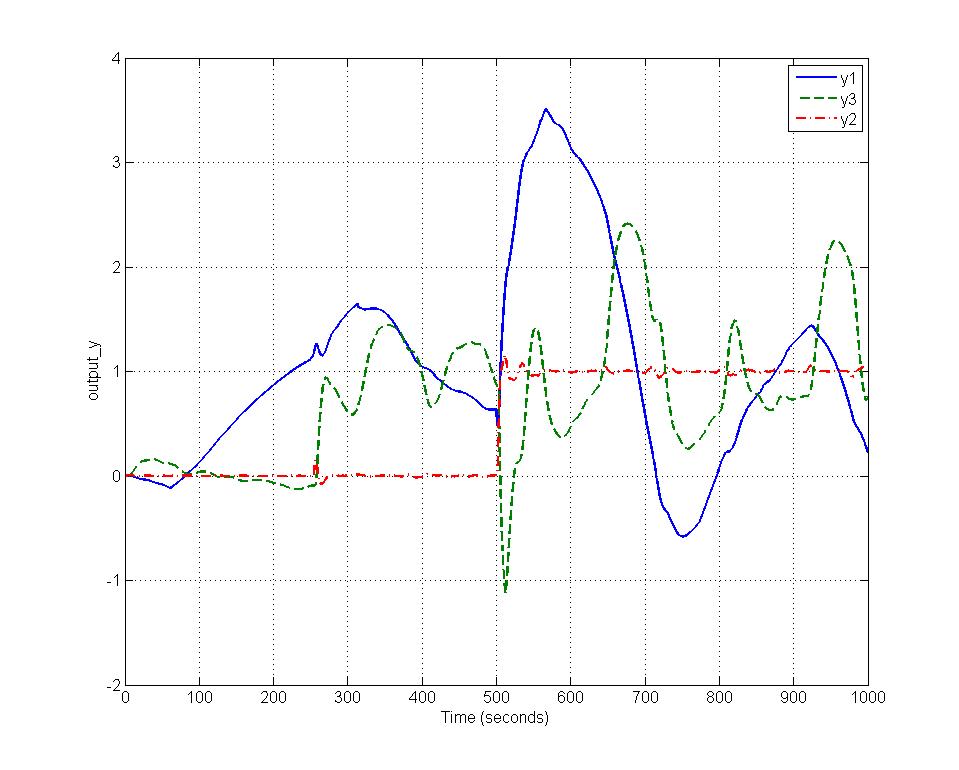
The decoupling PID controller is

1. **Matlab Code**

|  |
| --- |
| %Decoupling Control with Integrity Rules  G\_R=[tf([ETF(1,1).num{1}],[ETF(1,2).den{1}],'InputDelay',ETF(1,2).InputDelay + ETF(1,2).ioDelay), 0, 0;  0, tf([ETF(2,2).num{1}],[ETF(2,2).den{1}],'InputDelay',ETF(2,1).InputDelay + ETF(2,1).ioDelay), 0;  0, 0, tf([ETF(3,3).num{1}],[ETF(3,3).den{1}],'InputDelay',ETF(3,1).InputDelay + ETF(3,1).ioDelay)];    for i = 1:G\_size(1)  for j = 1:G\_size(2)  G\_hat\_I(i,j) =tf(G\_R(j,j).num{1}(end)/ETF(j,i).num{1}(end)\*ETF(j,i).den{1},G\_R(j,j).den{1},'InputDelay',abs(G\_R(j,j).InputDelay)+abs(G\_R(j,j).ioDelay) - abs(ETF(j,i).InputDelay)- abs(ETF(j,i).ioDelay));  end  end  PID\_decoupling = PID\_controller(G\_R);  sim('decoupling\_simulation');  plot(simout); |
| function[PID]=PID\_controller(G)  Am = 3;  G\_size = size(G);  PID = cell(G\_size(1),G\_size(2));  for i = 1:G\_size(1)  K(i,i) = G(i,i).num{1}(end);  L(i,i) = G(i,i).InputDelay + G(i,i).ioDelay;  T(i,i) = G(i,i).den;  if size(T{i,i}) == [1 2]  PID{i,i}={pi\*T{i,i}(1)/2/Am/L(i,i)/K(i,i), pi/2/Am/L(i,i)/K(i,i), 0};  else %size(T{i,i}) == [1 3]  PID{i,i}={pi\*T{i,i}(2)/2/Am/L(i,i)/K(i,i) pi/2/Am/L(i,i)/K(i,i) pi\*T{i,i}(1)/2/Am/L(i,i)/K(i,i)};  end  end  end |

1. **Simulation**





As shown in the figure above, only one output can converge to 1 responding to the three unit step inputs. Therefore, the decoupler and PID controllers are not designed properly. Therefore, I decided to redesign the decoupling control using ETF without integrity rules.

# Design Decoupling Control using ETF without Integrity Rules

1. **Normalized Decoupling**

According to the normalized decoupling control system design rules, the forward transfer matrix is selected as

The decoupler can be obtained by

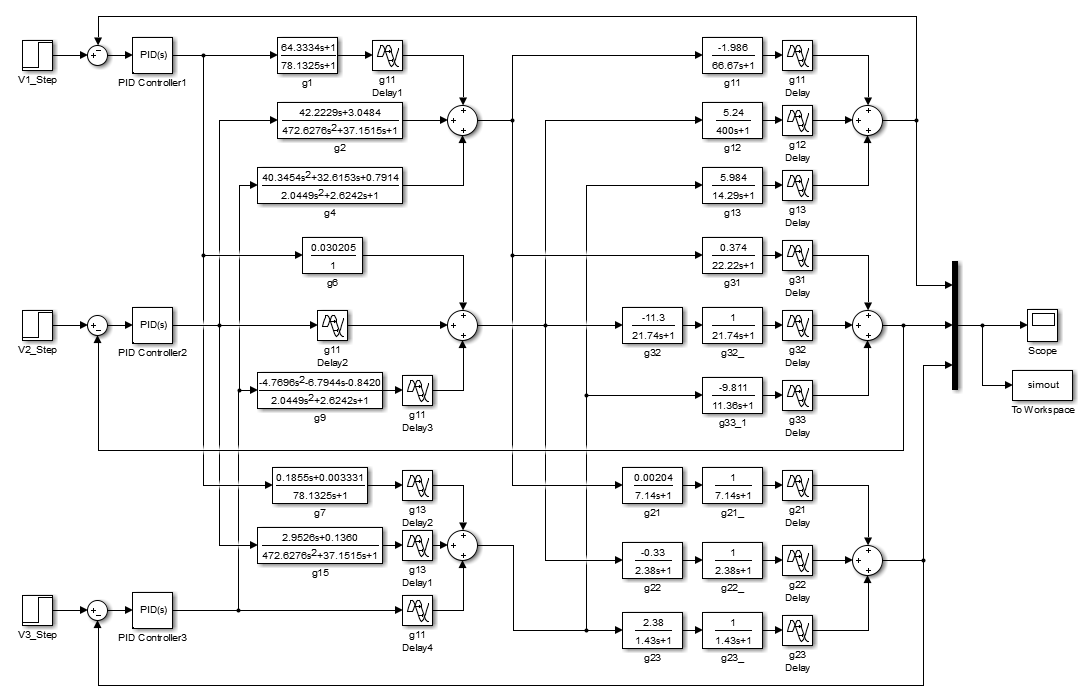
1. **Design Controllers based on Gain and Phase Margin Method**

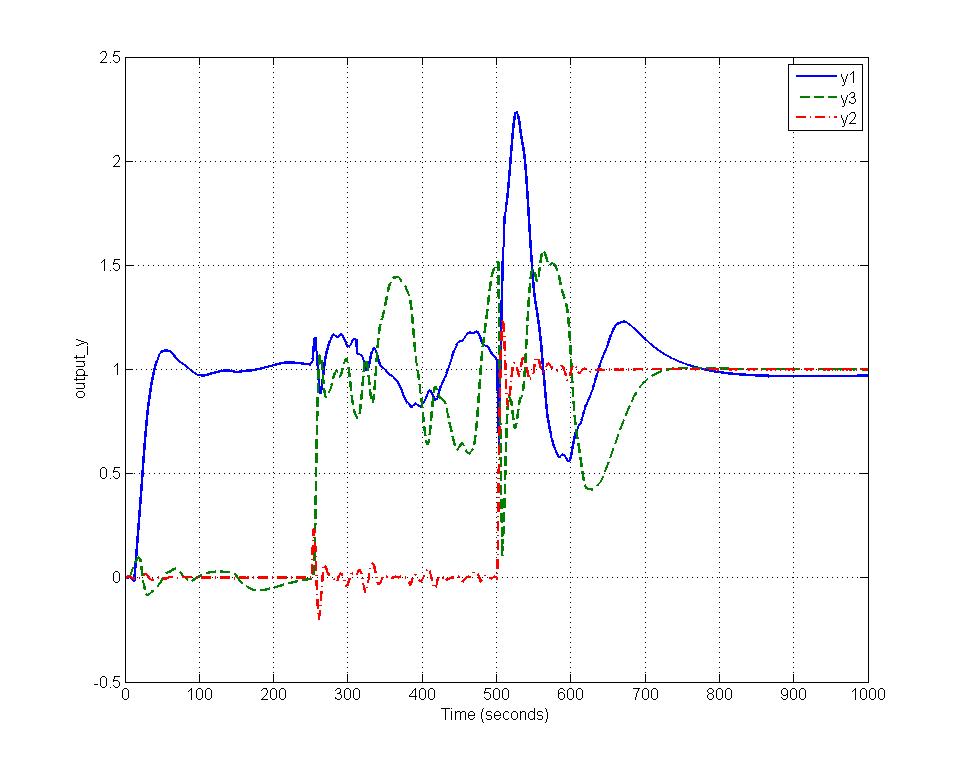
The decoupling PID controller is

1. **Matlab Code**

|  |
| --- |
| % Decoupling Control without Integrity Rules  G\_R=[tf([ETF(1,1).num{1}],[ETF(1,2).den{1}],'InputDelay',ETF(1,2).InputDelay + ETF(1,2).ioDelay), 0, 0;  0, tf([ETF(2,2).num{1}],[ETF(2,2).den{1}],'InputDelay',ETF(2,1).InputDelay + ETF(2,1).ioDelay), 0;  0, 0, tf([ETF(3,3).num{1}],[ETF(3,3).den{1}],'InputDelay',ETF(3,1).InputDelay + ETF(3,1).ioDelay)];    for i = 1:G\_size(1)  for j = 1:G\_size(2)  G\_hat\_I(i,j) =tf(G\_R(j,j).num{1}(end)/ETF(j,i).num{1}(end)\*ETF(j,i).den{1},G\_R(j,j).den{1},'InputDelay',abs(G\_R(j,j).InputDelay)+abs(G\_R(j,j).ioDelay) - abs(ETF(j,i).InputDelay)- abs(ETF(j,i).ioDelay));  end  end  PID\_decoupling = PID\_controller(G\_R);  sim('decoupling\_simulation');  plot(simout); |
| function[PID]=PID\_controller(G)  Am = 3;  G\_size = size(G);  PID = cell(G\_size(1),G\_size(2));  for i = 1:G\_size(1)  K(i,i) = G(i,i).num{1}(end);  L(i,i) = G(i,i).InputDelay + G(i,i).ioDelay;  T(i,i) = G(i,i).den;  if size(T{i,i}) == [1 2]  PID{i,i}={pi\*T{i,i}(1)/2/Am/L(i,i)/K(i,i), pi/2/Am/L(i,i)/K(i,i), 0};  else %size(T{i,i}) == [1 3]  PID{i,i}={pi\*T{i,i}(2)/2/Am/L(i,i)/K(i,i) pi/2/Am/L(i,i)/K(i,i) pi\*T{i,i}(1)/2/Am/L(i,i)/K(i,i)};  end  end  end |

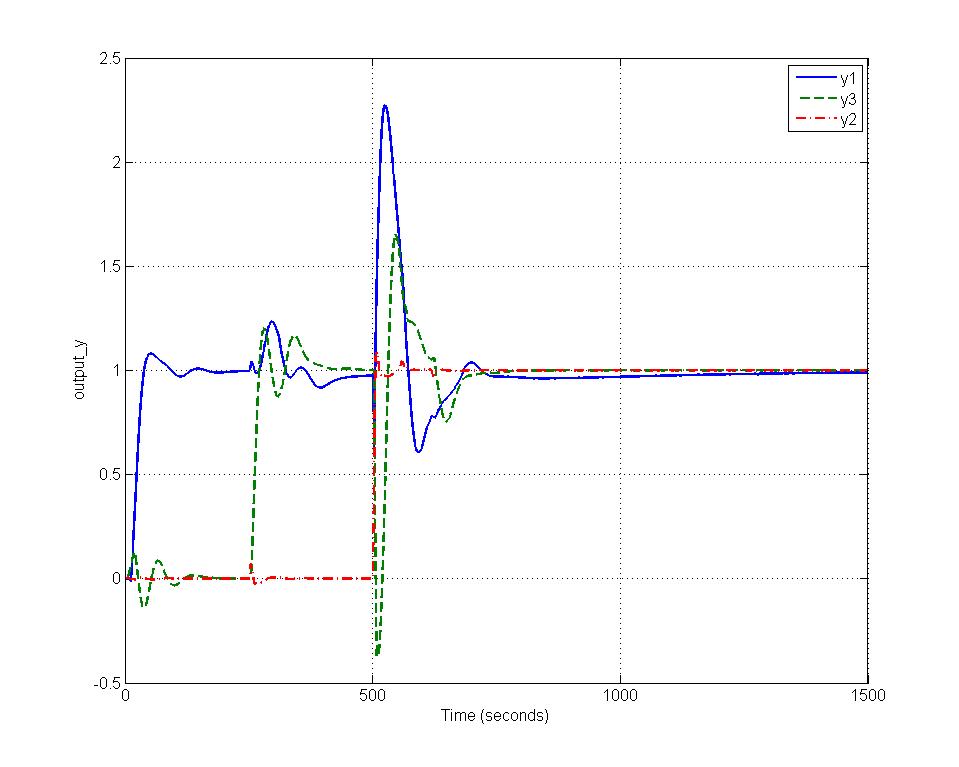
1. **Simulation**



****

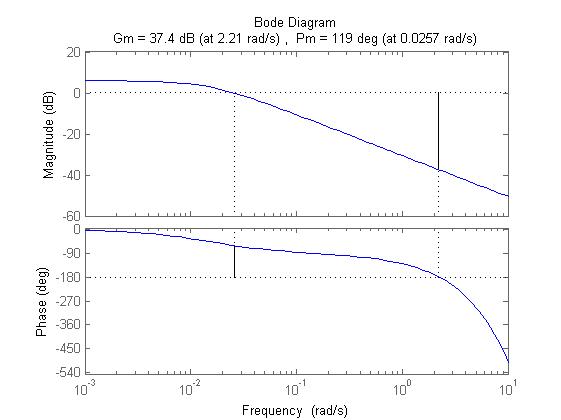
As shown in the figure above, all three output can converge to 1 responding to the three unit step inputs, while has a little bais toward 1. Therefore, the decoupler and PID controllers are designed properly but not well enough, as changes too rapidly and fiercely. Perhaps, this is due to the strong derivative factor of the PID controller over the two FOPTD model and . Thus, I adjust the derivative factor of the PID controller to, and the final decoupling PID controller is

As shown in the figure below, all three output can converge to 1 responding to the three unit step inputs with appropriate dynamic performence. Therefore, the decoupler and PID controllers are designed properly.

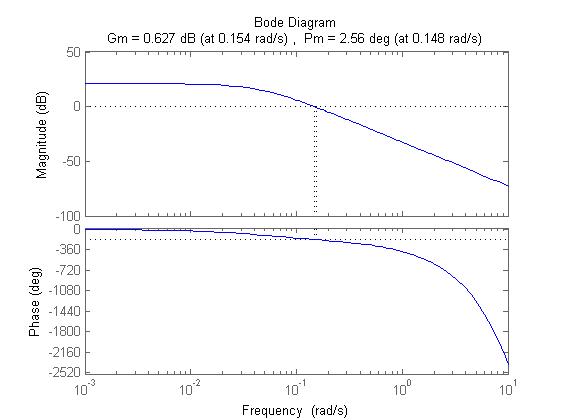


# Classical Decentralized Control based on Biggest Log Modulus Tuning Method

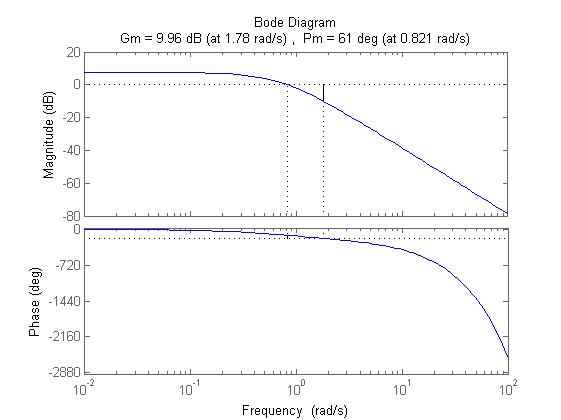
1. **Calculate the Ziegler-Nichols Settings for Each Individual Loop**



For , the ultimate gain and the ultimate frequncy .



For , the ultimate gain and the ultimate frequncy .



For , the ultimate gain and the ultimate frequncy .

According to the Ziegler–Nichols method, and for PI control.

1. **BLT-1 Tuning**

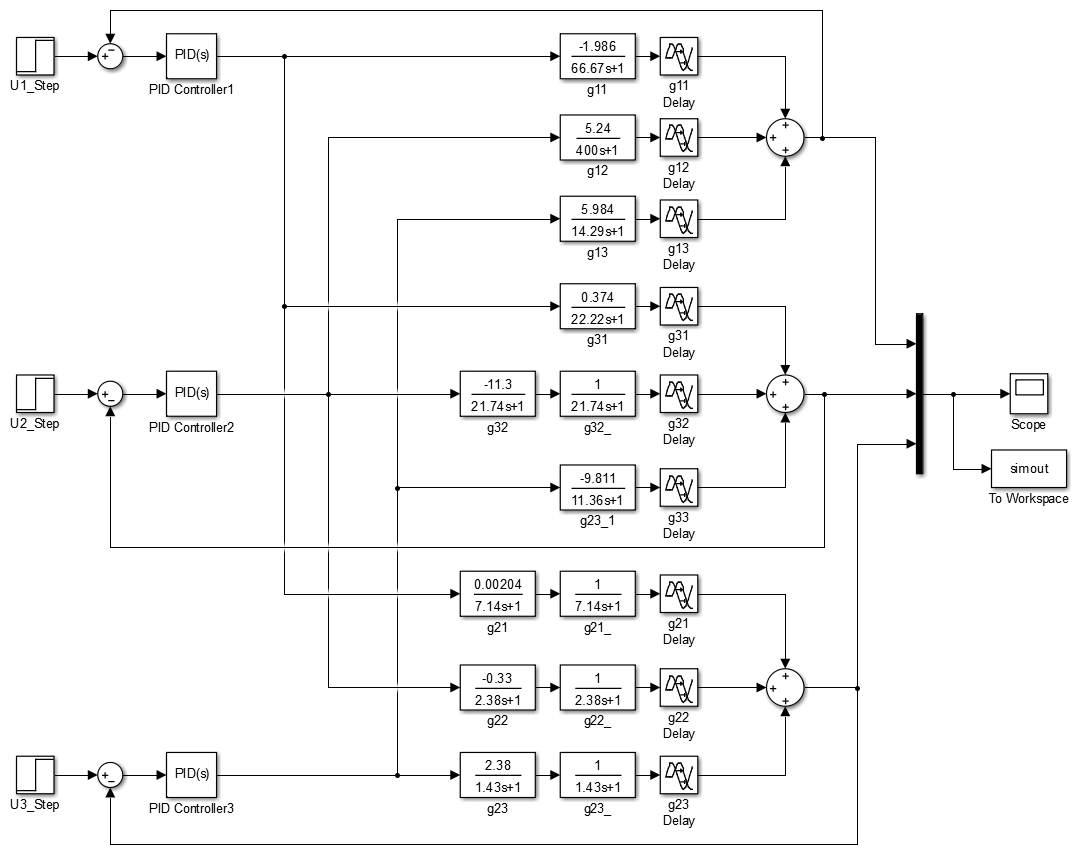
Choose a detuning factor , and adjust it until .

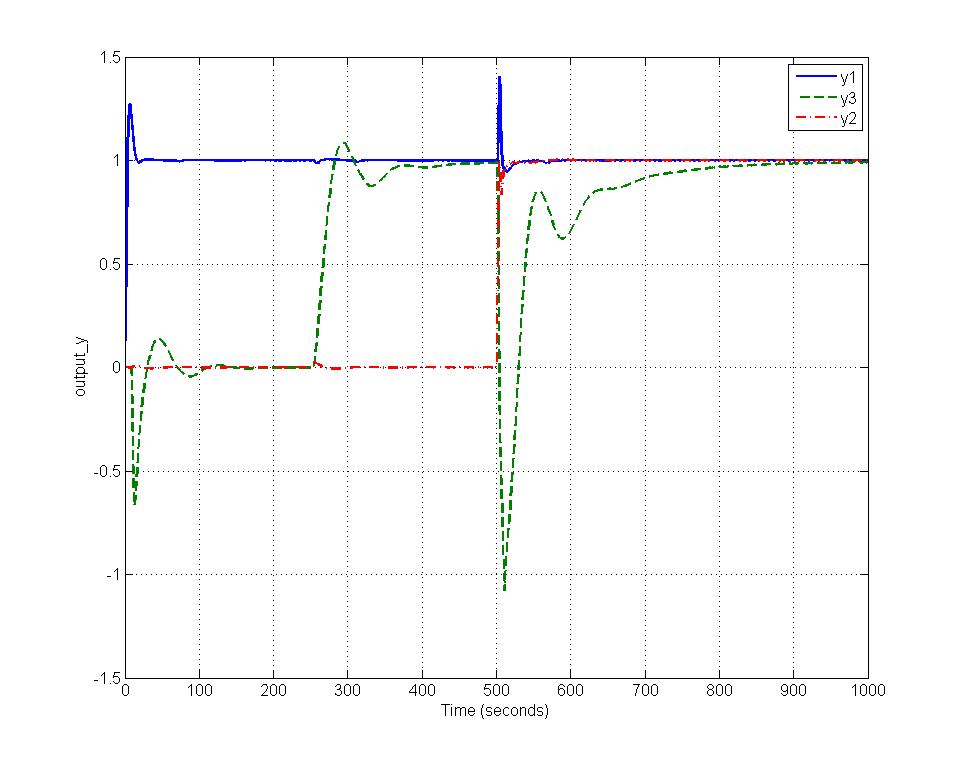
Considering the derivative action for SOPTD. Choose a second detuning factor , and compute . While adjusting , I find is minimized when . Therefore, no derivation action is needed.

1. **Matlab Code**

|  |
| --- |
| %Ku Wu  allmargin\_ = [allmargin(-G(1,1)),allmargin(-G(2,2)),allmargin(G(3,3))];  GM = [-allmargin\_(1).GainMargin(1),-allmargin\_(2).GainMargin(1),allmargin\_(3).GainMargin(1)];  GMF = [allmargin\_(1).GMFrequency(1),allmargin\_(2).GMFrequency(1),allmargin\_(3).GMFrequency(1)];  K\_ZN = GM./2.2;  T\_I\_ZN = 2\*pi./(1.2\*GMF);  T\_D\_ZN = 2\*pi./(8\*GMF);    min\_error=999;  for F=2.0:0.05:3  K\_C = K\_ZN/F;  T\_I = F\*T\_I\_ZN;  K\_I = K\_C./T\_I;  max\_Lc = 0;  Gc=[tf([K\_C(1) K\_I(1)],[1 0]),0,0;  0,tf([K\_C(2) K\_I(2)],[1 0]),0;  0,0,tf([K\_C(3) K\_I(3)],[1 0])];  for w=0.1:0.02:2.5  W = -1+det(eye(3)+freqresp(G\*Gc,w));  Lc = 20\*log10(abs(W/(1+W)));  if(Lc>max\_Lc)  max\_Lc=Lc;  end  end  error = abs(max\_Lc-6);  if(error<min\_error)  min\_error=error;  F\_match=F;  end  end    K\_C = K\_ZN./F\_match;  T\_I = F\_match.\*T\_I\_ZN;  K\_I = K\_C./T\_I;  min\_error\_2=999;  for FD=2.0:0.5:20  T\_D = T\_D\_ZN/FD;  K\_D=K\_C.\*T\_D;  max\_Lc\_2 = 0;  Gc=[tf([K\_C(1) K\_I(1)],[1 0]),0,0;  0,tf([K\_D(2) K\_C(2) K\_I(2)],[0 1 0]),0;  0,0,tf([K\_D(3) K\_C(3) K\_I(3)],[0 1 0])];  for w=0.1:0.05:2.5  W = -1+det(eye(3)+freqresp(G\*Gc,w));  Lc = 20\*log10(abs(W/(1+W)));  if(Lc>max\_Lc\_2)  max\_Lc\_2=Lc;  end  end  error = abs(max\_Lc\_2-6);  if(error<min\_error\_2)  min\_error\_2=error;  F\_match\_D=FD;  end  end    K\_C = K\_ZN./F\_match;  T\_I = F\_match.\*T\_I\_ZN;  K\_I = K\_C./T\_I;  T\_D = T\_D\_ZN/F\_match\_D;  K\_D = K\_C.\*T\_D;  sim('decentralized\_simulation\_BLT');  plot(simout); |

1. **Simulation**

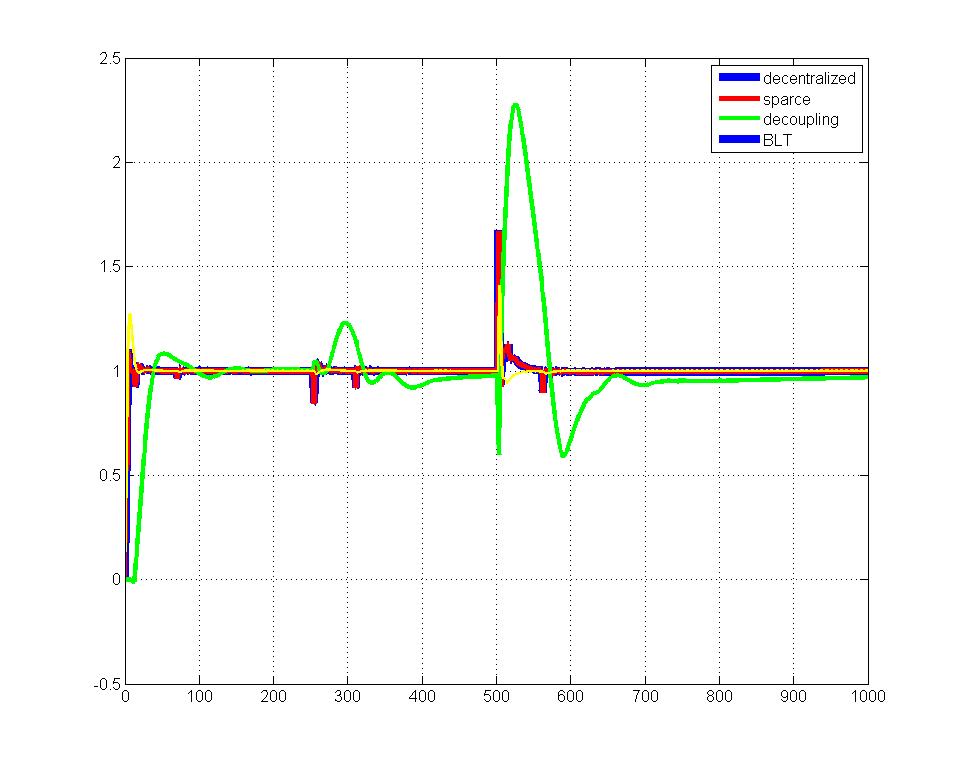


****

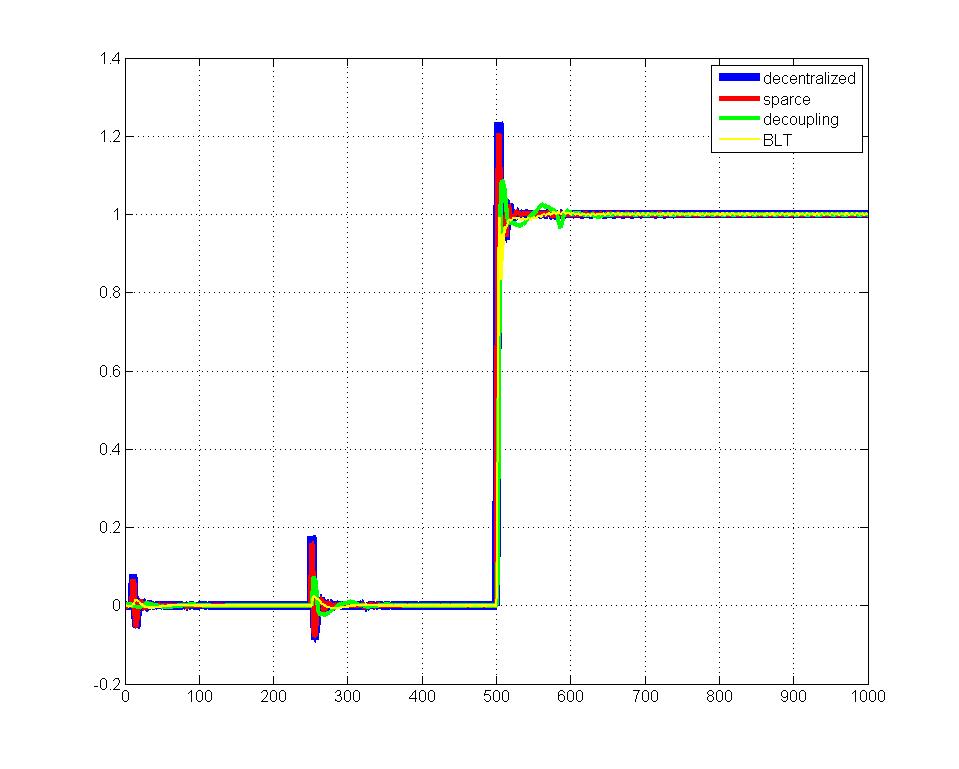
As shown in the figure above, all three outputs converge to 1 responding to the three unit step inputs. Therefore, all three PID controllers are designed properly and successfully control the three outputs.

# Comparison

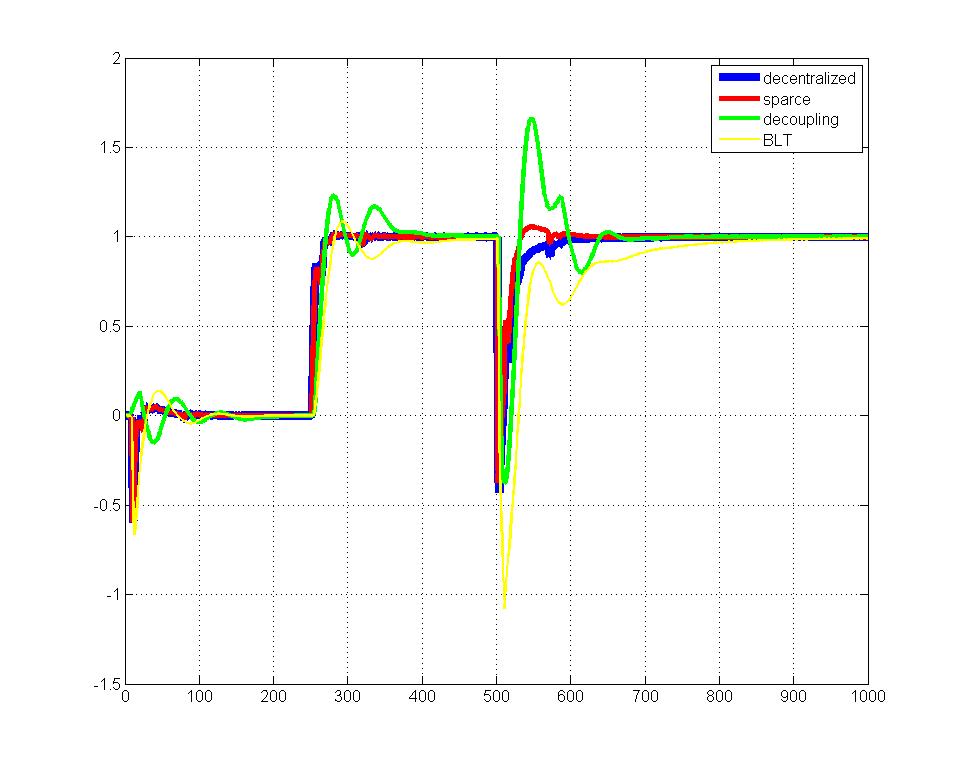
1. **Output**



1. **Output**



1. **Output**



According to all the three figures above, we can conclude that decentralized control method and parse control method using ETF have similar and satisfactory results. The classical decentralized control method based on BLT performs a little worse than the pervious two methods in dynamic responding. The decoupling method using ETF is sensitive to the modelling error and hard to obtain a good result, but gets the best dynamic performance.