# **Notes & Screenshots**

## Servo-Compensators

**Example 1** – **20th Order Heat System**

We start with servo-compensators that just attempt to track a constant setpoint:

**HeatServoComp.m**

**% 20-stage RC Filter**

**% Lecture #16**

**% Servo Compensators at DC**

**% System description**

**R = 0.2; Cap = 0.2;**

**A = zeros(20,20);**

**for i=1:19**

**A(i,i) = -2/(R\*Cap);**

**A(i,i+1) = 1/(R\*Cap);**

**A(i+1,i) = 1/(R\*Cap);**

**end**

**A(20,20) = -1/(R\*Cap);**

**B = zeros(20,1);**

**B(1) = 1/(R\*Cap);**

**C = zeros(1,20);**

**C(20) = 1;**

**D = 0;**

**% 4th Order Approximation**

**Rapprox = 5\*R; Capprox = 5\*Cap;**

**aa = 1/(Rapprox\*Capprox);**

**Aapprox = zeros(4,4);**

**for i=1:3**

**Aapprox(i,i) = -2\*aa;**

**Aapprox(i,i+1) = aa;**

**Aapprox(i+1,i) = aa;**

**end**

**Aapprox(4,4) = -aa;**

**Bapprox = zeros(4,1);**

**Bapprox(1) = aa;**

**Capprox = zeros(1,4);**

**Capprox(4) = 1;**

**Dapprox = D;**

**% Simple Compensator**

**% des\_poles = [-1, -2, -2.3473, -3.5321];**

**% [Kx, Kr] = placePoles(Aapprox, Bapprox, Capprox, des\_poles);**

**% Servo Compensator**

**Aaug = [Aapprox, Bapprox\*0; Capprox, 0];**

**Baug = [Bapprox; 0];**

**Caug = [Capprox, 0];**

**des\_poles = [-1 -2 -2 -2 -2];**

**[Kx\_aug, Kr\_aug] = placePoles(Aaug,Baug,Caug, des\_poles);**

**Kx = Kx\_aug(1:4) %#ok<NOPTS>**

**Kz = Kx\_aug(5) %#ok<NOPTS>**

**% % % Check /w approximation**

**% AA = [Aapprox - Bapprox\*Kx, -Bapprox\*Kz; Capprox, 0];**

**% % eg1 = [eig(Aapprox); 1.2345];**

**% % fprintf('[Eigenvalues of Aapprox, Eigenvalues of Servo-comp Approx System]:\n');**

**% % [eg1, eig(AA)]**

**% % pause**

**% %**

**% % % Check /w full system**

**% Kx\_full = zeros(1,20); Kx\_full([5,10,15,20]) = Kx;**

**% AAA = [A-B\*Kx\_full, -B\*Kz; C, 0];**

**% % eg2 = [eig(A); 1.2345];**

**% % fprintf('[Eigenvalues of Original System, Eigenvalues of Servo-comp System]:\n');**

**% % [eg2, eig(AAA)]**

**% % pause**

**%**

**% % Check using step-response**

**% BB = [zeros(4,1);1];**

**% BBB = [zeros(20,1);1];**

**% CCy = [Capprox 0];**

**% CCu = [-Kx -Kz];**

**% CCCy = [C 0];**

**% CCCu = [-Kx\_full -Kz];**

**% ss\_approx\_y = ss(AA,BB,CCy,D);**

**% ss\_approx\_u = ss(AA,BB,CCu,D);**

**% ss\_orig\_y = ss(AAA,BBB,CCCy,D);**

**% ss\_orig\_u = ss(AAA,BBB,CCCu,D);**

**% % subplot(1,2,1);**

**% % step(ss\_approx\_y);**

**% % hold on;**

**% % step(ss\_approx\_u);**

**% % title('Step Response of Approx. System Servo-Comp');**

**% % legend('y\_{approx}(t)', 'u\_{approx}(t)');**

**% % grid on;**

**% % subplot(1,2,2);**

**% % step(ss\_orig\_y);**

**% % hold on;**

**% % step(ss\_orig\_u);**

**% % title('Step Response of Original System Servo-Comp');**

**% % legend('y(t)', 'u(t)');**

**% % grid on;**

**% % pause**

**% % hold off;**

**% % clf;**

**% Simulate**

**V = zeros(20,1);**

**Z = 0;**

**dt = 100e-6;**

**T\_end = 15;**

**t = 0;**

**V0 = 0;**

**Ref = 1;**

**N = (T\_end / dt) + 1;**

**DATA = zeros(N,3);**

**VMAX = 2;**

**% % Change system and see if servo-comp still manages /w disturbance**

**% for i=1:19**

**% A(i,i) = (-2/(R\*Cap))\*1.1; % Extra losses**

**% end**

**% A(20,20) = (-1/(R\*Cap))\*1.1;**

**i=1;**

**tic**

**while(t < T\_end)**

**% while(abs(V(20) - Ref) > 0.005\*Ref)**

**V0 = -Kz\*Z - Kx\*V([5,10,15,20]);**

**dV = A\*V + B\*V0;**

**dZ = V(20) - Ref;**

**V = V + dV \* dt;**

**Z = Z + dZ \* dt;**

**t = t + dt;**

**DATA(i,:) = [V(20), V0, Ref];**

**i = i+1;**

**% plot([0:20], [V0;V], 'b.-', t, 0, 'b+',20,Ref,'b+');**

**% ylim([0,3\*VMAX]);**

**% xlim([0,20]);**

**% grid on;**

**% pause(100e-6);**

**% DATA = [DATA ; V(20), V0, Ref];**

**end**

**toc**

**t = [1:length(DATA)]' \* dt;**

**DATAds = downsample(DATA,10);**

**tds = downsample(t,10);**

**plot(t,DATA);**

**grid on;**

**legend('V\_{20}(t)','V\_{in}(t)','Ref');**

**title('Simulated Step Response of Full Servo-Comp System');**

**% subplot(1,2,1);**

**% plot(t,DATA);**

**% grid on;**

**% legend('V\_{20}(t)','V\_{in}(t)','Ref');**

**% title('Simulated Step Response of Full Servo-Comp System');**

**% subplot(1,2,2);**

**% step(ss\_orig\_y);**

**% hold on;**

**% step(ss\_orig\_u);**

**% grid on;**

**% title('Step Response of Original System Servo-Comp');**

**% legend('y(t)', 'u(t)');**

**% % ylim([0, VMAX\*1.3]);**

**placePoles.m**

**function [Kx, Kr] = placePoles(A, B, C, poles)**

**sz = size(A);**

**N = sz(1);**

**% Controllability matrix**

**T1 = zeros(N,N);**

**for i=0:N-1**

**T1(:,i+1) = (A^i) \* B;**

**end**

**if N <= 1**

**error('System needs to have an order greater than 1.');**

**elseif rank(T1) ~= N**

**error('System is not controllable');**

**end**

**if isControllerCanonical(A)**

**Pd = poly(poles);**

**P = poly(eig(A));**

**dP = Pd - P;**

**Kx = dP([N+1:-1:2]);**

**else**

**P = poly(eig(A));**

**% T1 is controllability matrix [B AB A^2B ... A^N-1B]**

**T1 = zeros(N,N);**

**for i=0:N-1**

**T1(:,i+1) = (A^i) \* B;**

**end**

**% T2 involves the characteristic polynomial coefficients of A**

**T2 = zeros(N,N);**

**for i=0:N-1**

**T2(i+1,:) = [zeros(1,i), P(1:end-i-1)];**

**end**

**% Swap rows of identity matrix to get T3**

**T3 = eye(N);**

**for i=1:floor(N/2)**

**ii = T3(i,:);**

**T3(i,:) = T3(N-i+1,:);**

**T3(N-i+1,:) = ii;**

**end**

**% Similarity transform matrix**

**T = T1\*T2\*T3;**

**% Obtain state and input matrices of similar system**

**Az = inv(T)\*A\*T;**

**Bz = inv(T)\*B;**

**% Now obtain Kz needed for similar system**

**Pd = poly(poles);**

**dP = Pd - P;**

**Kz = dP([N+1:-1:2]);**

**% Perform similarity transform to obtain corresponding Kx**

**Kx = Kz\*inv(T);**

**end**

**Kr = -1 / (C\*inv(A-B\*Kx)\*B);**

**end**

**isControllerCanonical.m**

**function [val] = isControllerCanonical(A)**

**sz = size(A);**

**N = sz(1);**

**if sz(1)~= sz(2)**

**error('Matrix is not square!');**

**end**

**% Check first column to see if it's all zeros until last entry or not**

**col1 = A(:,1);**

**col1\_excluding\_bottom = col1([1:N-1]);**

**zero\_vec = zeros(N-1,1);**

**if ~isequal(col1\_excluding\_bottom,zero\_vec)**

**val = false;**

**return;**

**end**

**% Now check whether matrix formed by excluding first col and last row is**

**% identity**

**B = A(1:N-1,2:N);**

**if isequal(B,eye(N-1))**

**val = true;**

**return;**

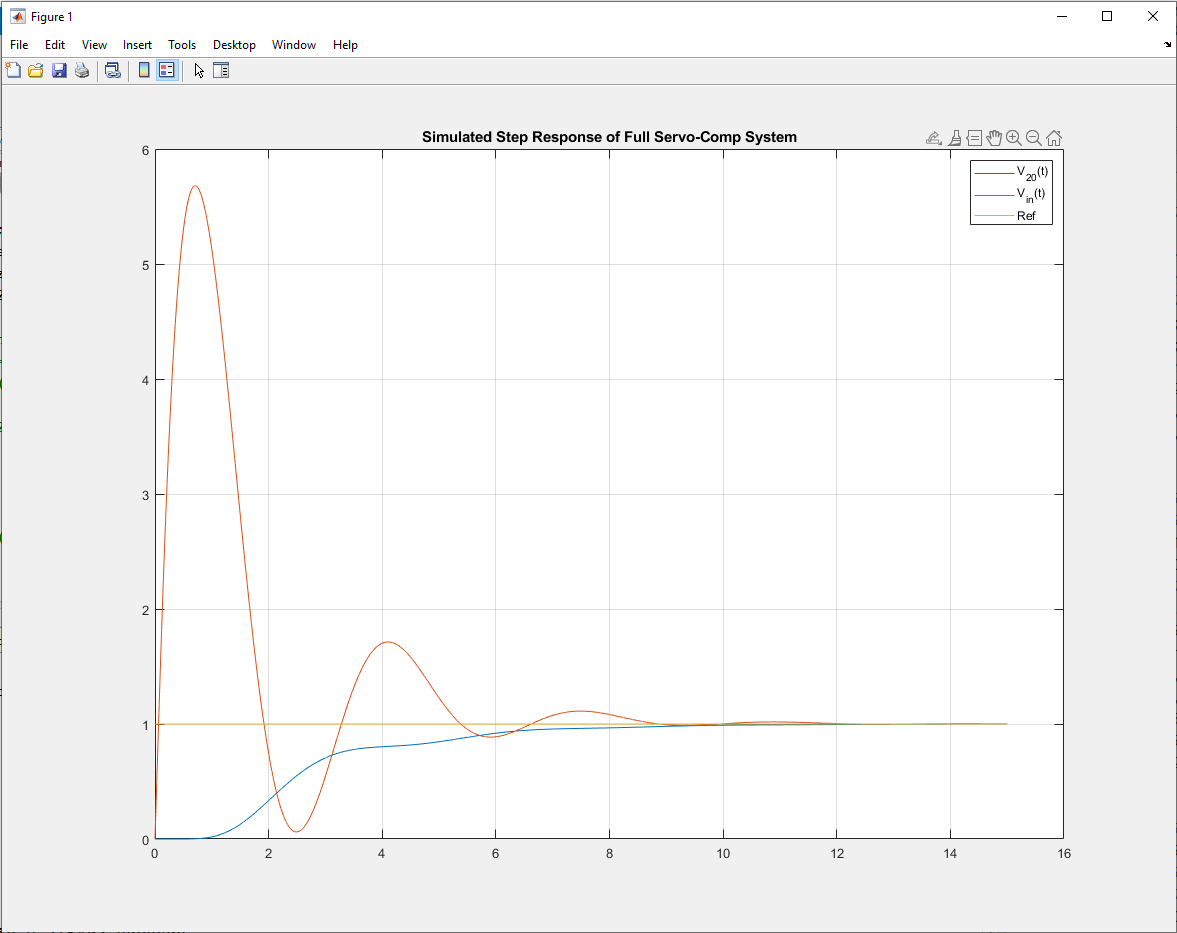
**else**

**val = false;**

**return;**

**end**

**end**



**Example 2 – Ball & Beam System**

**Beam2.m**

**% Ball & Beam System**

**% Lecture #16**

**% Servo Compensators at DC**

**%% System**

**% System setup**

**mball = 0.5; mbeam = 3;**

**Rball = 10e-2; L = 2;**

**% Obtain matrices**

**[A,B] = linearizedBeamBall(mball,Rball,mbeam,L);**

**C = [1 0 0 0]; % We are looking at the position of the ball**

**D = 0;**

**%% Servo-compensation**

**% Augmented system**

**Aaug = [A, 0\*B; C, 0];**

**Baug = [B; 0];**

**Caug = [C, 0];**

**des\_poles = [-1 -2 -2 -2 -2];**

**[Kx\_aug, Kr\_aug] = placePoles(Aaug,Baug,Caug, des\_poles);**

**Kx = Kx\_aug(1:4)**

**Kz = Kx\_aug(5)**

**% Servo Compensator Gains**

**%Kx = [ -56.77 102.00 -38.57 18.00];**

**%Kz = -20.57;**

**% %% % Check /w approximation**

**% AA = [A-B\*Kx, -B\*Kz; C, 0]; % A matrix of servo-compensated system**

**% eg1 = [eig(A); 1.2345]; % eigenvalues of original system**

**% fprintf('[Eigenvalues of Original System, Eigenvalues of Servo-comp System]:\n');**

**% [eg1, eig(AA)] %#ok<\*NOPTS>**

**% pause**

**%**

**% % Check using step-response**

**% BB = [zeros(4,1);1];**

**% CCy = [C 0];**

**% CCu = [-Kx -Kz];**

**% ss\_y = ss(AA,BB,CCy,D);**

**% ss\_u = ss(AA,BB,CCu,D);**

**%**

**% step(ss\_y);**

**% hold on;**

**% step(ss\_u);**

**% title('Step Response of Approx. System Servo-Comp');**

**% legend('y\_{approx}(t)', 'u\_{approx}(t)');**

**% grid on;**

**% pause**

**% clf**

**%% Simulate**

**% Setting ICs and simulation config**

**X = [0, 0, 0, 0]'; % [r q dr dq]**

**Z = 0;**

**dt = 100e-6; T\_end = 15;**

**t = 0;**

**Ref = 1;**

**N = (T\_end / dt) + 1;**

**DATA = zeros(N,4);**

**% % Change system and see if servo-comp still manages /w disturbance**

**mball = mball\*5;**

**disturb = 0;**

**% Simulate**

**i=1;**

**tic**

**while(t < T\_end)**

**U = -Kz\*Z - Kx\*X + disturb; % Here, U is torque T**

**dX = BeamDynamics(X, U, mball, Rball, mbeam, L);**

**dZ = X(1) - Ref;**

**X = X + dX \* dt;**

**Z = Z + dZ \* dt;**

**t = t + dt;**

**DATA(i,:) = [Ref, X(1), X(2), U];**

**% if(n == 0)**

**% BeamDisplay(X, Ref);**

**% end**

**i = i+1;**

**end**

**toc**

**t = [1:length(DATA)]' \* dt; %#ok<NBRAK>**

**DATAds = downsample(DATA,10); tds = downsample(t,10);**

**plot(t,DATA);**

**grid on;**

**legend('Ref','r(t)','\theta(t)','u(t)');**

**title('Simulated Step Response of Full Servo-Comp Ball&Beam System'); xlabel('Time (s)');**

**linearizedBeamBall.m**

**function [A,B] = linearizedBeamBall(mball, Rball, mbeam, L)**

**g = 9.8;**

**Jball = (2/5)\*mball\*Rball^2;**

**Jbeam = (1/12)\*mbeam\*L^2;**

**A = [0 0 1 0; 0 0 0 1; 0 (-mball\*g/(mball+Jball\*Rball^2)) 0 0; (-mball\*g/Jbeam) 0 0 0];**

**B = [0;0;0;1/Jbeam];**

**end**

**BeamDynamics.m**

**function [dX] = BeamDynamics( X, T, mball, Rball, mbeam, L )**

**% Ball and Beam: Sp21 Version**

**% m = 0.5 kg**

**% J = 2.0 kg m^2**

**r = X(1);**

**q = X(2);**

**dr = X(3);**

**dq = X(4);**

**g = 9.8;**

**% mball = 0.5; Rball = 10e-2;**

**% mbeam = 1; L = 4;**

**Jball = (2/5)\*mball\*Rball^2;**

**Jbeam = (1/12)\*mbeam\*L^2;**

**M = [(mball + Jball\*Rball^2), 0;**

**0, ((mball + Jball\*Rball^2)\*r^2 + Jbeam)];**

**B1 = (mball + Jball\*Rball^2)\*dq^2 - mball\*g\*sin(q);**

**B2 = T - 2\*(mball + Jball\*Rball^2)\*dq\*r\*dr - mball\*g\*cos(q);**

**ddX = inv(M)\*[B1; B2];**

**dX = [dr; dq; ddX];**

**end**

**BeamDisplay.m**

**function [] = BeamDisplay( X, Ref )**

**% BeamDisplay(X, Ref);**

**% ECE 463 Lecture #8**

**x = X(1);**

**q = X(2);**

**dx = X(3);**

**dq = X(4);**

**% draw the beam**

**x1b = -1.9\*cos(q);**

**y1b = -1.9\*sin(q);**

**x2b = 1.9\*cos(q);**

**y2b = 1.9\*sin(q);**

**% draw the ball**

**i = [0:0.1:1]' \* 2\*pi;**

**xb = x\*cos(q) + 0.05\*cos(i) - 0.05\*sin(q);**

**yb = x\*sin(q) + 0.05\*sin(i) + 0.05\*cos(q);**

**% draw a line through the ball so you can see it roll**

**Q = -x / 0.05;**

**x1 = x\*cos(q) - 0.05\*sin(q) + 0.05\*cos(Q);**

**y1 = x\*sin(q) + 0.05\*cos(q) + 0.05\*sin(Q);**

**Q = Q + pi;**

**x2 = x\*cos(q) - 0.05\*sin(q) + 0.05\*cos(Q);**

**y2 = x\*sin(q) + 0.05\*cos(q) + 0.05\*sin(Q);**

**plot(0,0,'bo',[x1b,x2b],[y1b,y2b],'b',xb,yb,'r',[x1,x2],[y1,y2],'r-', Ref\*cos(q), Ref\*sin(q), 'bx');**

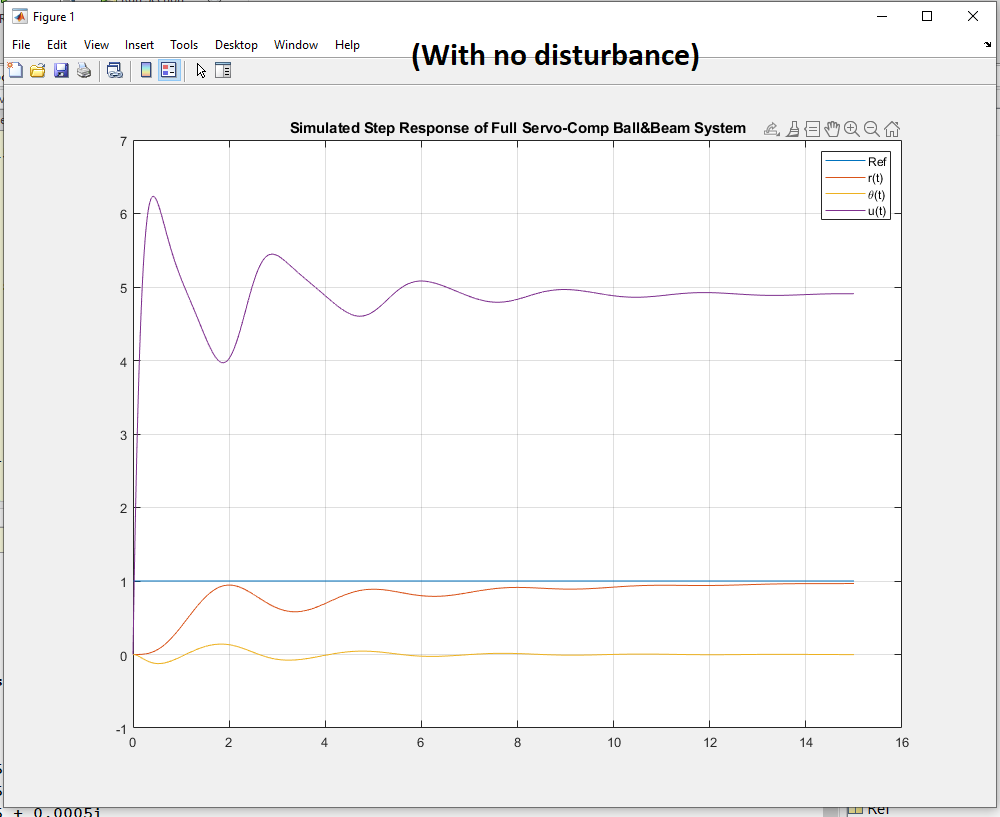
**ylim([-1.5,1.5]);**

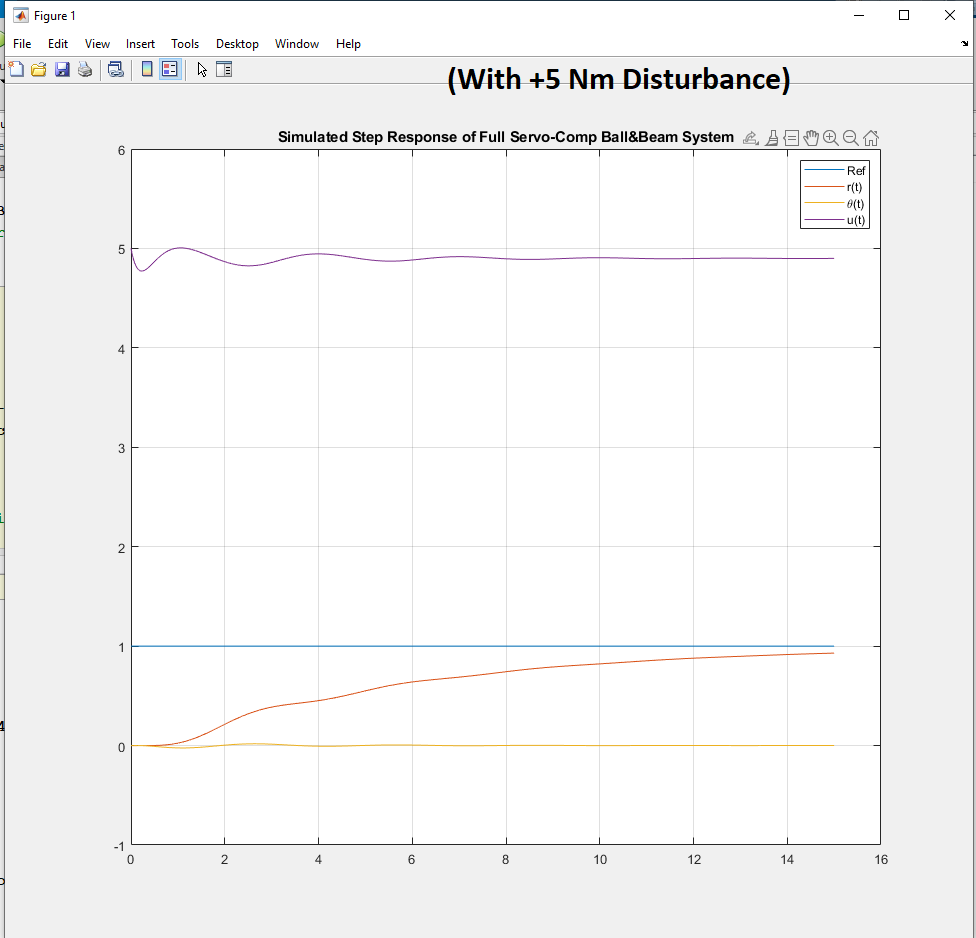
**xlim([-1.5,1.5]);**

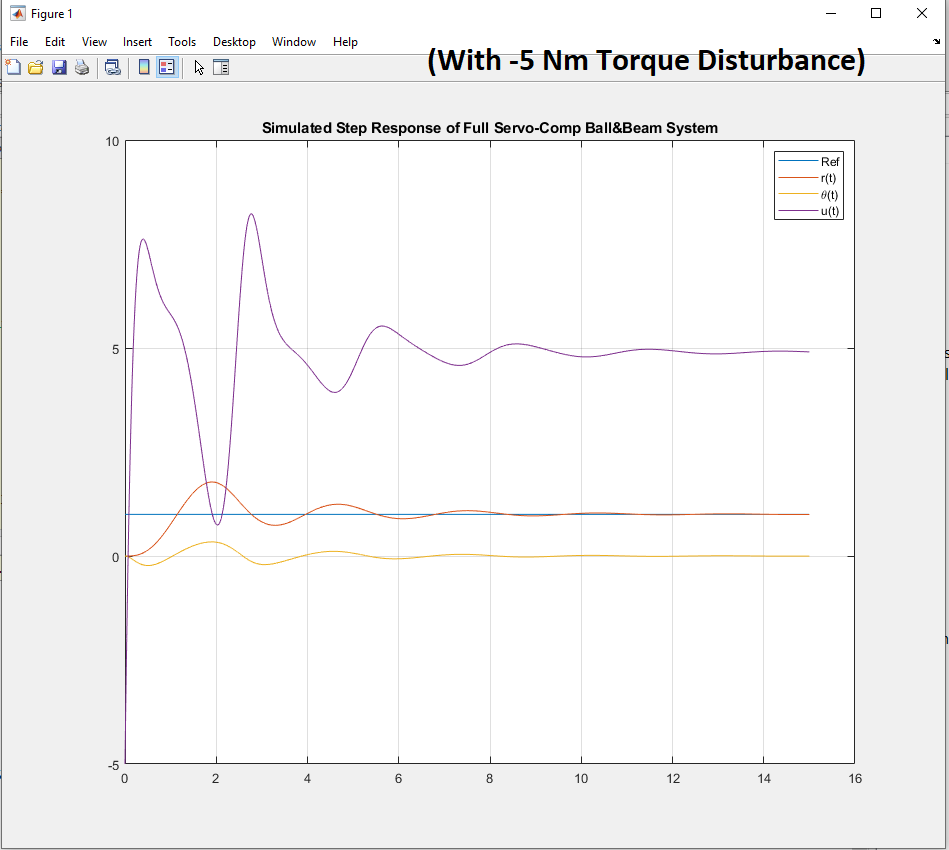
**pause(0.01);**

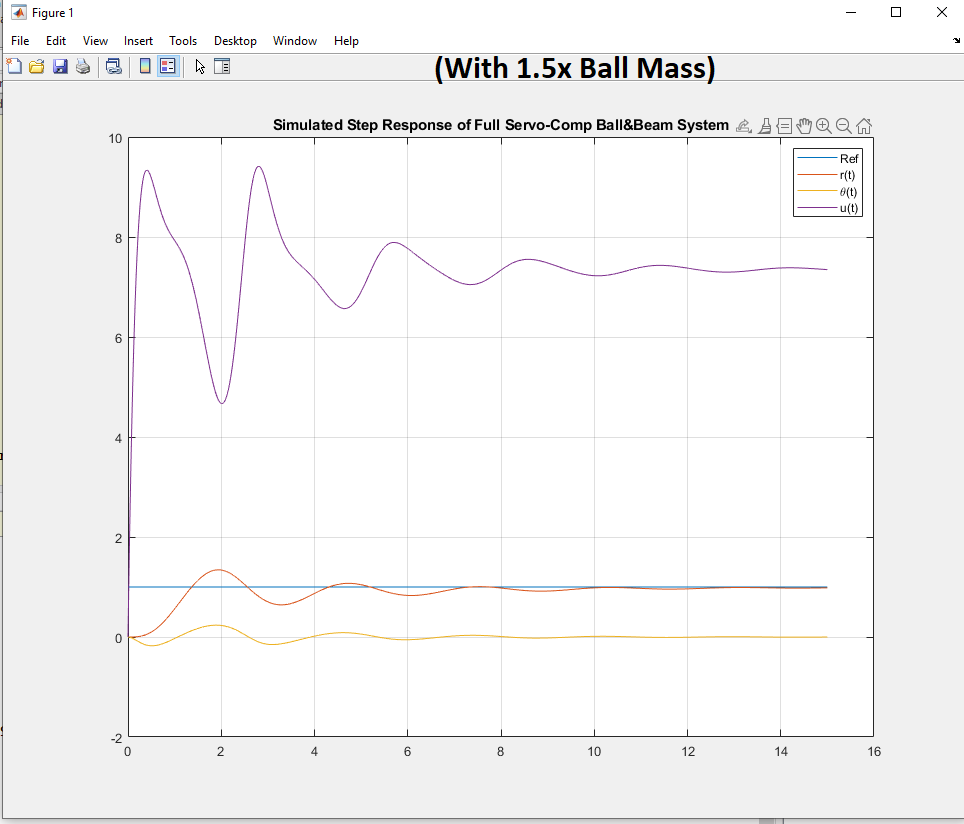
**hold off**

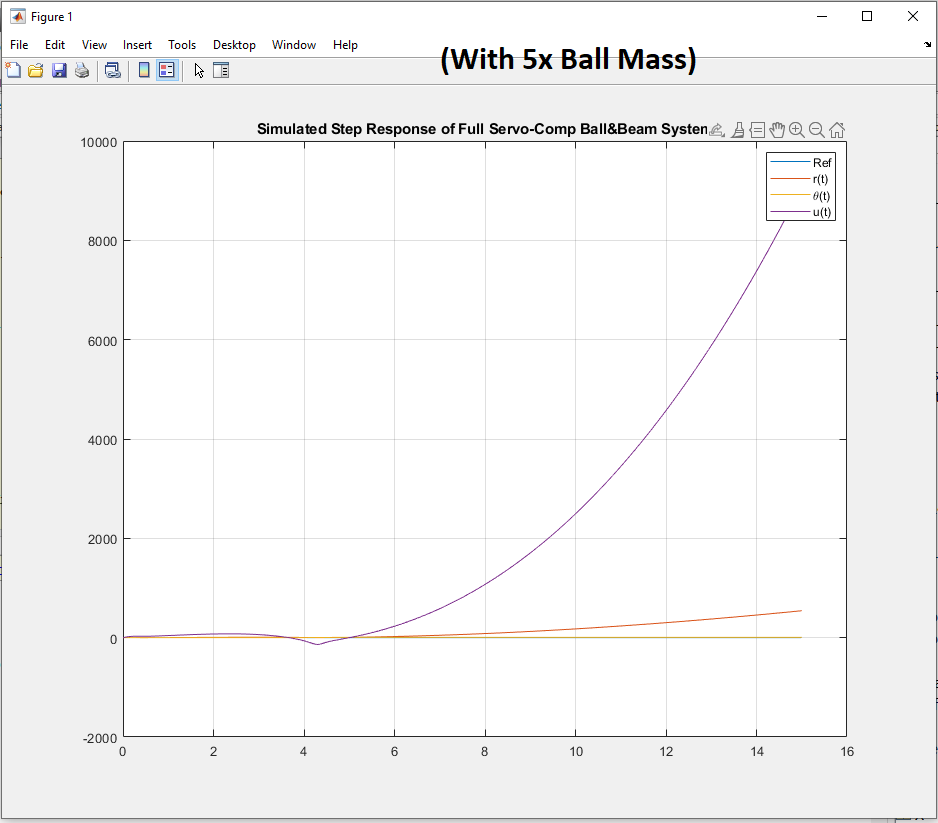
**end**











Now for servo-compensators that track a sinusoidal reference:

**% 20-stage RC Filter**

**% Lecture #16**

**% Servo Compensators at DC**

**%% System description**

**R = 0.2; Cap = 0.2;**

**A = zeros(20,20);**

**for i=1:19**

**A(i,i) = -2/(R\*Cap);**

**A(i,i+1) = 1/(R\*Cap);**

**A(i+1,i) = 1/(R\*Cap);**

**end**

**A(20,20) = -1/(R\*Cap);**

**B = zeros(20,1);**

**B(1) = 1/(R\*Cap);**

**C = zeros(1,20);**

**C(20) = 1;**

**D = 0;**

**% 4th Order Approximation**

**Rapprox = 5\*R; Capprox = 5\*Cap;**

**aa = 1/(Rapprox\*Capprox);**

**Aapprox = zeros(4,4);**

**for i=1:3**

**Aapprox(i,i) = -2\*aa;**

**Aapprox(i,i+1) = aa;**

**Aapprox(i+1,i) = aa;**

**end**

**Aapprox(4,4) = -aa;**

**Bapprox = zeros(4,1);**

**Bapprox(1) = aa;**

**Capprox = zeros(1,4);**

**Capprox(4) = 1;**

**Dapprox = D;**

**%% Simple Compensator**

**% des\_poles = [-1, -2, -2.3473, -3.5321];**

**% [Kx, Kr] = placePoles(Aapprox, Bapprox, Capprox, des\_poles);**

**%% Servo Compensator**

**Az = [0 2; -2 0]; Bz = [1;1];**

**Aaug = [Aapprox, zeros(4,2); Bz\*Capprox, Az];**

**Baug = [Bapprox; 0; 0];**

**Caug = [Capprox, 0, 0];**

**des\_poles = [-1 -2 -3 -4 -5 -6];**

**[Kx\_aug, Kr\_aug] = placePoles(Aaug,Baug,Caug, des\_poles);**

**Kx = Kx\_aug(1:4) %#ok<NOPTS>**

**Kz = Kx\_aug(5:6) %#ok<NOPTS>**

**% %% Check /w approximation**

**% AA = [Aapprox - Bapprox\*Kx, -Bapprox\*Kz; Bz\*Capprox, Az];**

**% eg1 = [eig(Aapprox); 1.2345; 1.2345];**

**% fprintf('[Eigenvalues of Aapprox, Eigenvalues of Servo-comp Approx System]:\n');**

**% [eg1, eig(AA)]**

**%**

**% % Check linearized response**

**% Acl = [Aapprox - Bapprox\*Kx, -Bapprox\*Kz; Bz\*Capprox, Az];**

**% Bcl = [zeros(4,1); -Bz];**

**% Ccl = Caug; Dcl = D;**

**% V0 = zeros(6,1); t = transpose(linspace(0,10,1001));**

**% R = sin(2\*t);**

**% Vout = step3(Acl,Bcl,Ccl,Dcl,t,V0,R);**

**% plot(t,Vout,t,R,'LineWidth',2);**

**% legend('Vout(t)','Vin(t)');**

**% ylim([-2,2]); grid on; title('Step3 Response Check');**

**%**

**% pause**

**%% Simulate**

**V = zeros(20,1);**

**Z = zeros(2,1);**

**dt = 100e-6;**

**T\_end = 15;**

**t = 0;**

**V0 = 0;**

**% Ref = sin(2\*t);**

**N = (T\_end / dt) + 1;**

**DATA = zeros(N,3);**

**VMAX = 2;**

**% % Change system and see if servo-comp still manages /w disturbance**

**% for i=1:19**

**% A(i,i) = (-2/(R\*Cap))\*1.1; % Extra losses**

**% end**

**% A(20,20) = (-1/(R\*Cap))\*1.1;**

**disturb = 0;**

**i=1;**

**tic**

**while(t < T\_end)**

**% while(abs(V(20) - Ref) > 0.005\*Ref)**

**V0 = -Kz\*Z - Kx\*V([5,10,15,20]) + disturb;**

**Ref = sin(2\*t);**

**dV = A\*V + B\*V0;**

**dZ = Bz\*(Capprox\*V([5,10,15,20]) - Ref) + Az\*Z;**

**V = V + dV \* dt;**

**Z = Z + dZ \* dt;**

**t = t + dt;**

**DATA(i,:) = [V(20), V0, Ref];**

**i = i+1;**

**% plot([0:20], [V0;V], 'b.-', t, 0, 'b+',20,Ref,'b+');**

**% ylim([0,3\*VMAX]);**

**% xlim([0,20]);**

**% grid on;**

**% pause(100e-6);**

**% DATA = [DATA ; V(20), V0, Ref];**

**end**

**toc**

**t = [1:length(DATA)]' \* dt;**

**DATAds = downsample(DATA,10);**

**tds = downsample(t,10);**

**plot(t,DATA, 'LineWidth',2);**

**grid on;**

**legend('V\_{20}(t)','V\_{in}(t)','Ref');**

**title('Simulated Step Response of Full Servo-Comp 20^{th} Order Heat System');**

**function [ y ] = step3( A, B, C, D, t, X0, U )**

**T = t(2) - t(1);**

**[m, n] = size(C);**

**npt = length(t);**

**Az = expm(A\*T);**

**Bz = B\*T;**

**X = X0;**

**y = zeros(npt, m);**

**y(1,:) = (C\*X + D \* ( U(1,:)' ) )';**

**for i=2:npt**

**X = Az\*X + Bz\*( U(i,:)' );**

**Y = C\*X + D \* ( U(i,:)' );**

**y(i,:) = Y';**

**end**

**end**

