Table of Contents


```
clear; close all; clc;
load('given.mat')
A = Ebar\Abar
B = Ebar\Bbar
C =  [0 0 0 1;
      0 1 0 0]
% C = eye(4)
% x = [alpha, q, VT, theta]
% angle of attack, pitch rate, true air speed, pitch angle
```

```
A =

  -12.7883    0.9363   -0.0318         0
 -154.1308  -11.7971    0.0344         0
   54.7213         0   -0.0018  -32.2000
         0    1.0000         0         0
```

```
B =

   -0.7117
 -131.7397
   -3.3493
         0
```

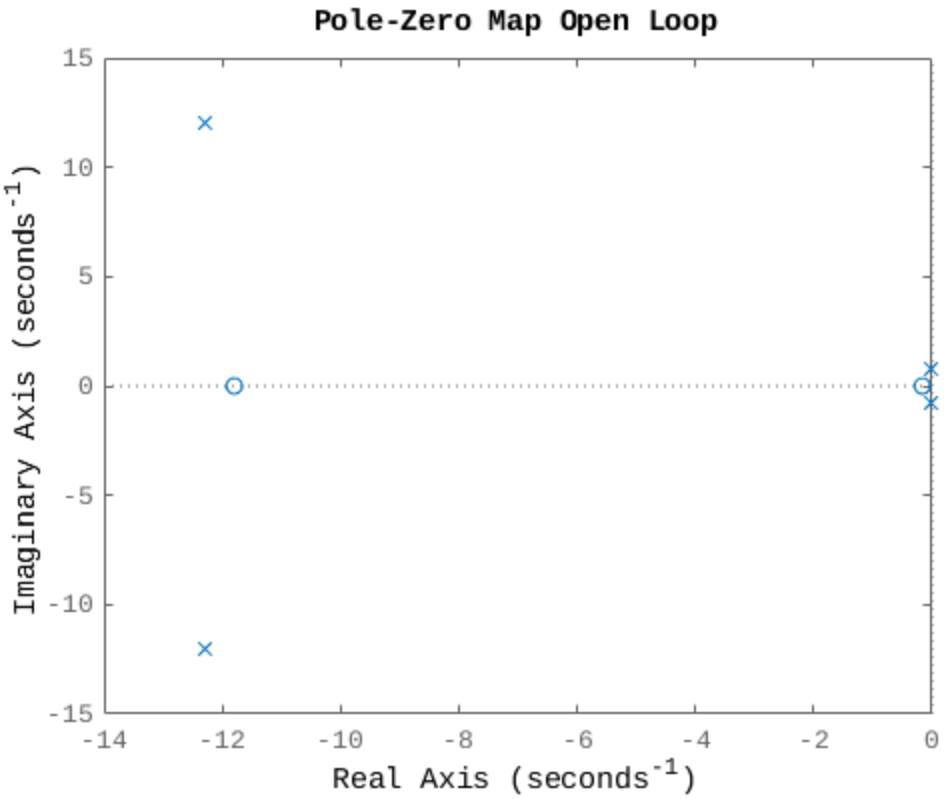
```
C =

     0     0     0     1
     0     1     0     0
```

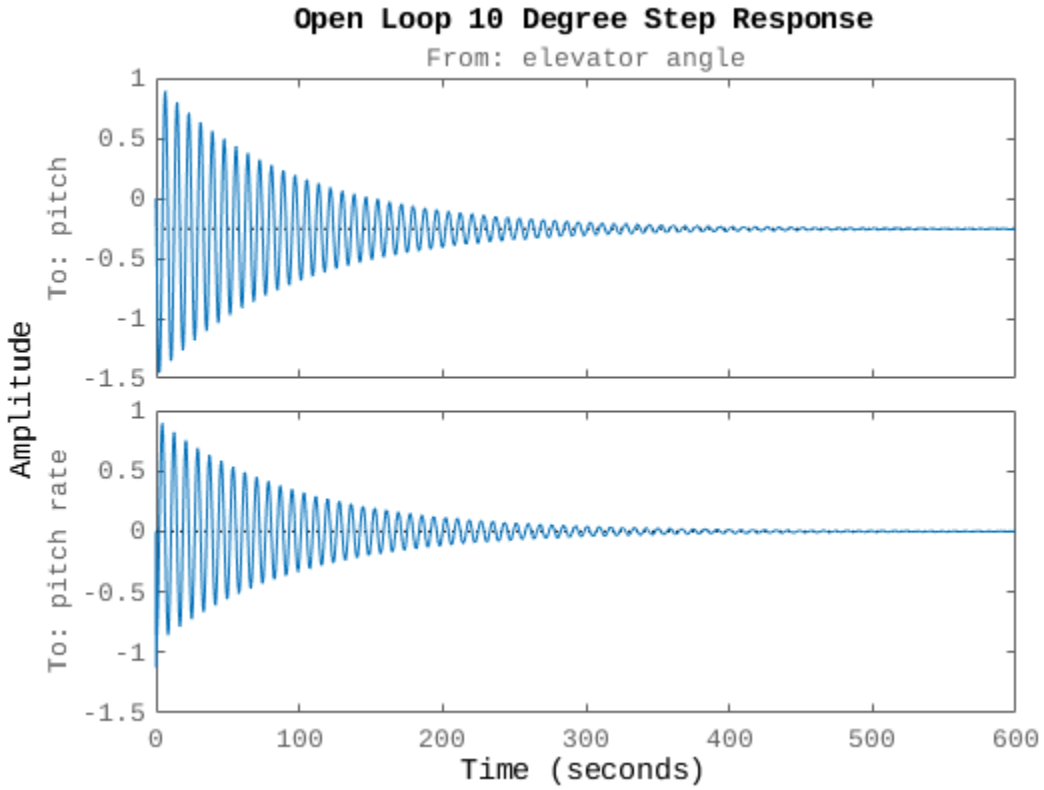
Part A

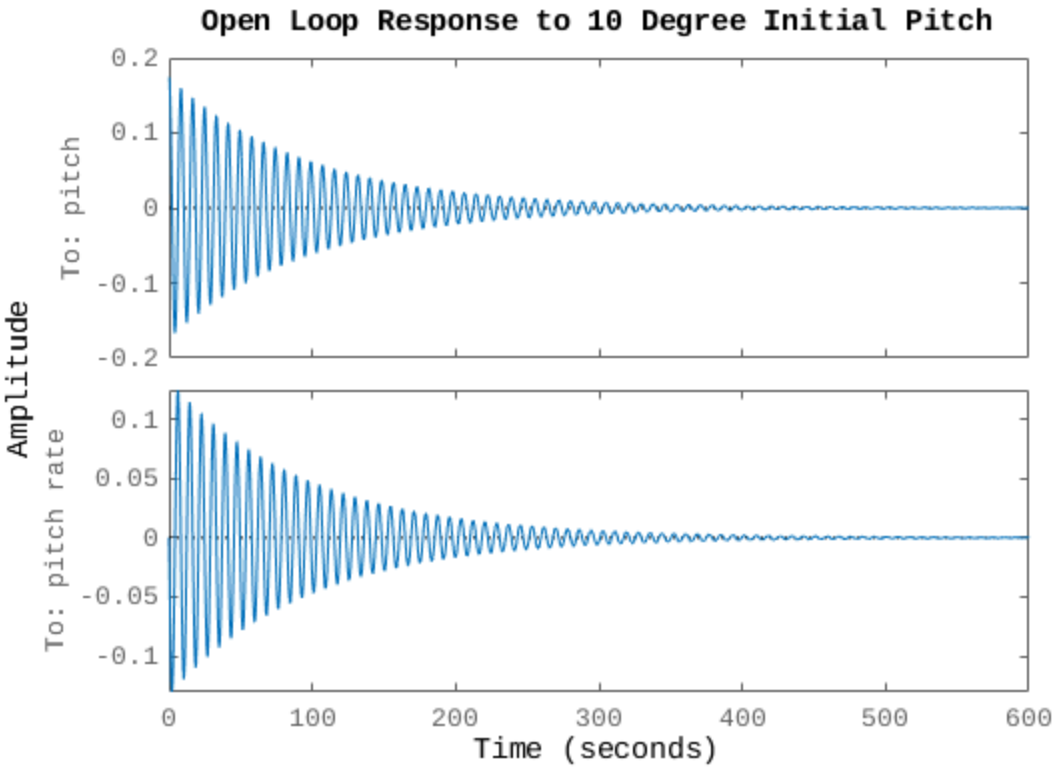
```
fprintf("***************Part A***************\n")
% Controllability
U1 = B;
U2 = A*B;
U3 = A*A*B;
U4 = A*A*A*B;
```

```
U = [U1 U2 U3 U4];
fprintf("Controllability\n")
disp(U)
fprintf("\tRank of controllability should be %i and is %i\n", [rank(A),
 rank(U)])
fprintf("\tTherefore is controllable\n\n")

% Observability with only pitch rate feedback
Cq = [0 1 0 0];
fprintf("C for pitch rate only is \n")
disp(Cq)
O1 = Cq;
O2 = Cq*A;
O3 = Cq*A*A;
O4 = Cq*A*A*A;
O = [O1; O2; O3; O4];
fprintf("Observability with only pitch rate feedback\n")
disp(O)
fprintf("\tRank of observability should be %i and is %i\n", [rank(A),
 rank(O)])
fprintf("\tTherefore is observable\n\n")

% Observability with only pitch feedback
Ctheta = [0 0 0 1];
fprintf("C for pitch only is \n")
disp(Ctheta)
O1 = Ctheta;
O2 = Ctheta*A;
O3 = Ctheta*A*A;
O4 = Ctheta*A*A*A;
O = [O1; O2; O3; O4];
fprintf("Observability with only pitch feedback\n")
disp(O)
fprintf("\tRank of observability should be %i and is %i\n", [rank(A),
 rank(O)])
fprintf("\tTherefore is observable\n\n")

sso = ss(A, B, C, 0, ...
    'StateName', {'alpha', 'q', 'VT', 'theta'}, ...
    'InputName', {'elevator angle'}, ...
    'OutputName', {'pitch', 'pitch rate'});
poles = pole(sso);
figure()
pzplot(sso)
title("Pole-Zero Map Open Loop")
fprintf("Open Loop System Stability:\n\tOpen Loop Poles\n")
disp(poles)
fprintf("\tAll real parts of poles are negative so open loop system is stable
\n\n")
figure()
opt = stepDataOptions;
opt.StepAmplitude = deg2rad(10) ;
step(sso,opt)
title("Open Loop 10 Degree Step Response")
```

```
figure()
initial(sso, [0, 0, 0, deg2rad(10)])
title("Open Loop Response to 10 Degree Initial Pitch")

**************Part A***************
Controllability
   1.0e+05 *

   -0.0000   -0.0011    0.0302   -0.4044
   -0.0013    0.0166   -0.0204   -4.4129
   -0.0000   -0.0004   -0.0200    1.1161
         0   -0.0013    0.0166   -0.0204

 Rank of controllability should be 4 and is 4
 Therefore is controllable

C for pitch rate only is
     0     1     0     0

Observability with only pitch rate feedback
   1.0e+04 *

         0    0.0001         0         0
   -0.0154   -0.0012    0.0000         0
    0.3791   -0.0005    0.0005   -0.0001
   -4.7446    0.3609   -0.0121   -0.0145

 Rank of observability should be 4 and is 4
 Therefore is observable

C for pitch only is
     0     0     0     1

Observability with only pitch feedback
   1.0e+03 *

         0         0         0    0.0010
         0    0.0010         0         0
   -0.1541   -0.0118    0.0000         0
    3.7912   -0.0051    0.0045   -0.0011

 Rank of observability should be 4 and is 4
 Therefore is observable

Open Loop System Stability:
 Open Loop Poles
 -12.2832 +12.0412i
 -12.2832 -12.0412i
  -0.0104 + 0.7629i
  -0.0104 - 0.7629i

 All real parts of poles are negative so open loop system is stable
```


Pole-Zero Map Open Loop
Imaginary Axis (seconds-1)
Real Axis (seconds-1)
15
10
5
0
-5
-10
-15
-14
-12
-10
-8
-6
-4
-2
0


Open Loop 10 Degree Step Response
From: elevator angle
To: pitch
To: pitch rate
Amplitude
Time (seconds)
1
0.5
0
-0.5
-1
-1.5
0
100
200
300
400
500
600



Part B

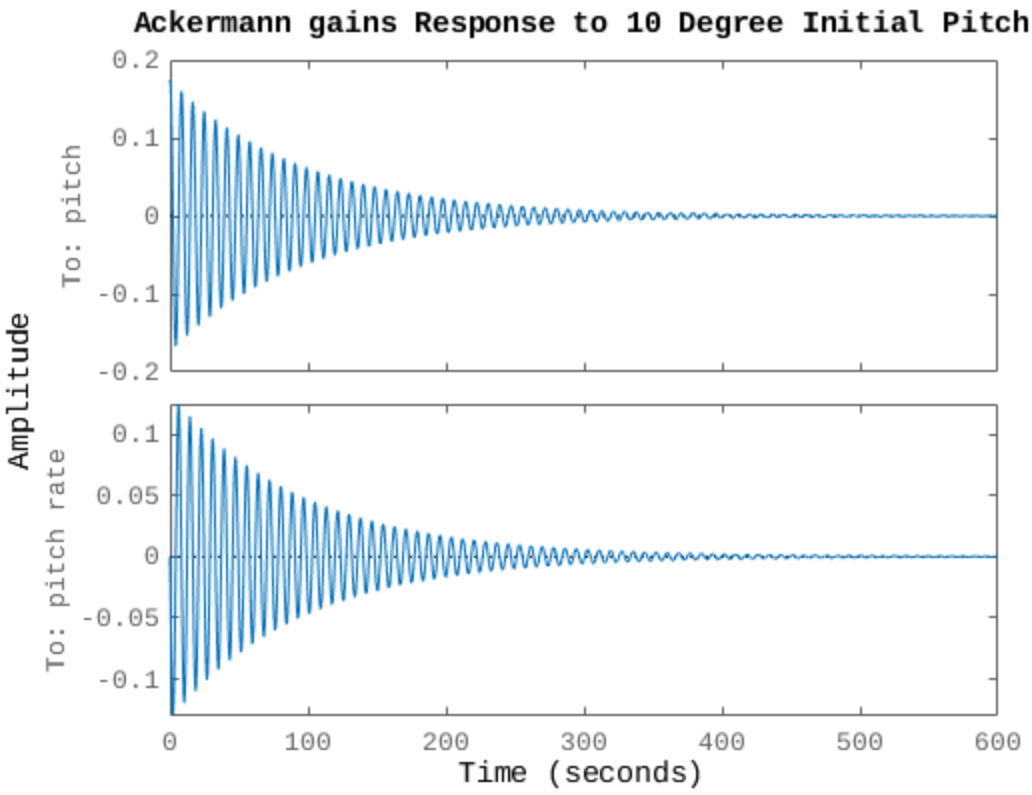
```
fprintf("***************Part B***************\n")
wn_sp = 16;
dr_sp = .8;
n = -wn_sp*(dr_sp);
w = wn_sp*sqrt(1-(dr_sp)^2);
lam1 = n + 1i*w;
lam2 = conj(lam1);
lambda = [lam1, lam2, poles(3), poles(4)];
Kack = acker(A, B, lambda);
% Kack = Ackermann(A, B, lambda);
fprintf("Ackermann's Formula\n")
fprintf("Desired Short Period Poles:\n")
disp([lam1; lam2])
fprintf("Gains are K = [%f  %f  %f  %f] \n", Kack)
fprintf("Poles using Ackermann gains:\n")
disp(eig(A-B*Kack))
ssack = ss(A-B*Kack, zeros(4,1), C, 0, ...
    'StateName', {'alpha', 'q', 'VT', 'theta'}, ...
    'InputName', {'elevator angle'}, ...
    'OutputName', {'pitch', 'pitch rate'});
figure()
initial(ssack, [0, 0, 0, deg2rad(10)])
title("Ackermann gains Response to 10 Degree Initial Pitch")
```

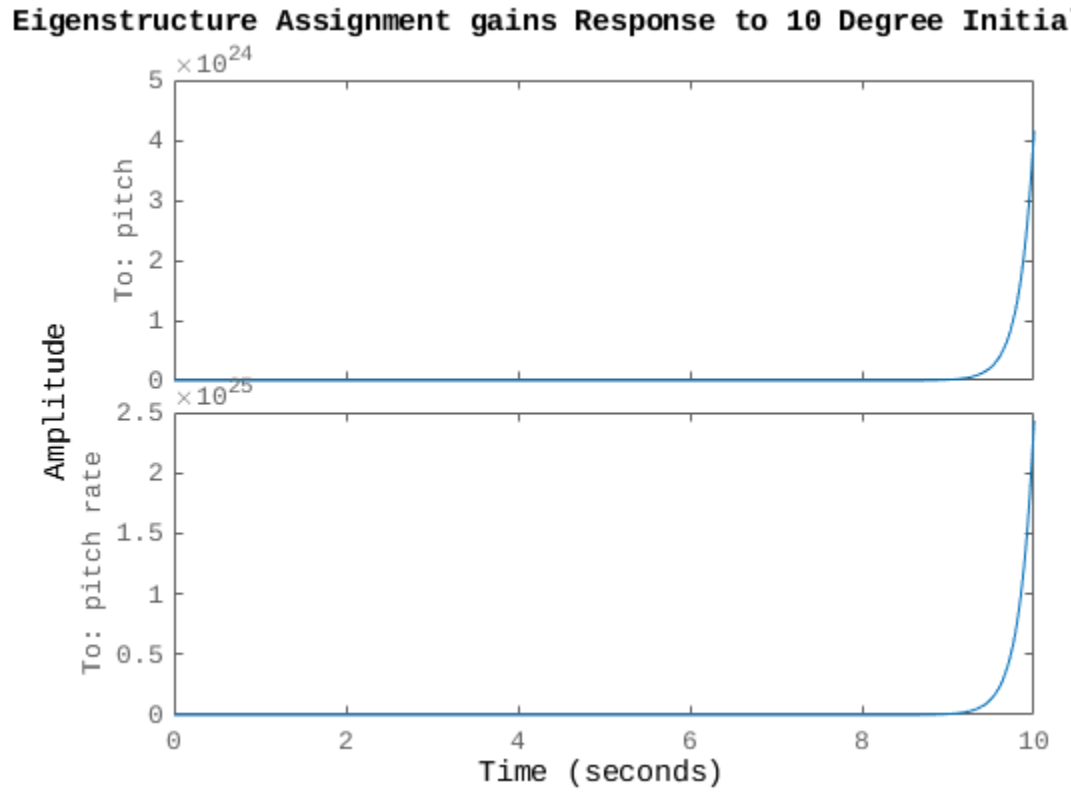
```
[ol_vec, ol_lam] = eig(A);
% new_vec1 = [1+i*-1; -1+i*1; 0; 0];
% new_vec3 = [0; 0; 1+i*-1; -1+i*1];
% new_vec2 = conj(new_vec1);
% new_vec4 = conj(new_vec2);
% new_vec = [new_vec1, new_vec2, new_vec3, new_vec4];
% new_vec = [1-i, 1+i;
%            -1+i, -1-i];
% new_vec = ol_vec([1,2],1:2)
% O = zeros(2,1);
% D = [1 0 0 0;
%      0 1 0 0;];

new_vec = ol_vec(:,1:2);
O = zeros(4,1);
D = eye(4);
I4 = eye(4);
M1 = [(lambda(1)*I4-A), B; D, O];
M1inv = pinv(M1);
M2 = [(lambda(2)*I4-A), B; D, O];
M2inv = pinv(M2);
top_zeros = zeros(4,2);
des = [top_zeros; new_vec];
final1 = M1inv*des(:,1);
v1 = final1(1:4);
u1 = final1(5);
final2 = M2inv*des(:,2);
v2 = final2(1:4);
u2 = final2(5);
U = [u1, u2];
V = [v1, v2];
Kstruc = U*inv(C*V);

% Kstruc = EigStructAssign(A, B, C, lambda(1:2), new_vec);
ssc = ss(A-B*Kstruc*C,B,C,0, ...
    'StateName', {'alpha', 'q', 'VT', 'theta'}, ...
    'InputName', {'elevator angle'}, ...
    'OutputName', {'pitch', 'pitch rate'});
figure()
initial(ssc, [0, 0, 0, deg2rad(10)])
title("Eigenstructure Assignment gains Response to 10 Degree Initial Pitch")

vd1 = [eye(4), zeros(4,1)];
fprintf("Eigenstructure Assignment using open loop eigen vectors\n")
fprintf("Desired Short Period Poles:\n")
disp([lam1; lam2])
fprintf("Gains are K = [%f  %f] \n", Kstruc)
fprintf("Poles using Eigenstructure Assignment gains:\n")
[es_vec, es_lam] = eig(A-B*Kstruc*C);
disp(es_lam)
fprintf("Achievable Eigen Vectors using Eigenstructure Assignment gains:\n")
disp(es_vec)
```

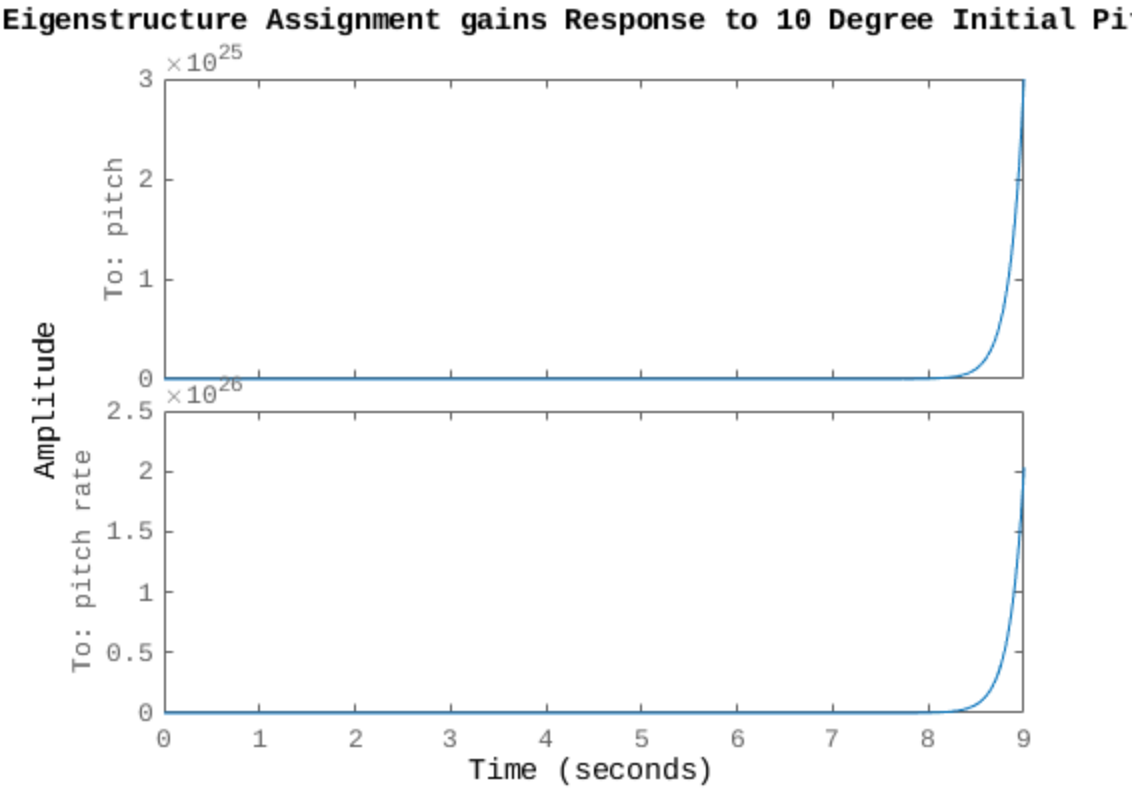
```
new_vec = [ol_vec(1,1), ol_vec(1,2)];
O = zeros(1,1);
D = [1 0 0 0];
I4 = eye(4);
M1 = [(lambda(1)*I4-A), B; D, O];
M1inv = inv(M1);
M2 = [(lambda(2)*I4-A), B; D, O];
M2inv = inv(M2);
top_zeros = zeros(4,2);
des = [top_zeros; new_vec];
final1 = M1inv*des(:,1);
v1 = final1(1:4);
u1 = final1(5);
final2 = M2inv*des(:,2);
v2 = final2(1:4);
u2 = final2(5);
U = [u1, u2];
V = [v1, v2];
Kstruc = U*inv(C*V);

% Kstruc = EigStructAssign(A, B, C, lambda(1:2), new_vec);
ssc = ss(A-B*Kstruc*C,B,C,0, ...
    'StateName', {'alpha', 'q', 'VT', 'theta'}, ...
    'InputName', {'elevator angle'}, ...
    'OutputName', {'pitch', 'pitch rate'});
figure()
initial(ssc, [0, 0, 0, deg2rad(10)])
title("Eigenstructure Assignment gains Response to 10 Degree Initial Pitch")

vd1 = [eye(4), zeros(4,1)];
fprintf("Eigenstructure Assignment of only alpha \n")
fprintf("Desired Short Period Poles:\n")
disp([lam1; lam2])
fprintf("Gains are K = [%f  %f] \n", Kstruc)
fprintf("Poles using Eigenstructure Assignment gains:\n")
[es_vec, es_lam] = eig(A-B*Kstruc*C);
disp(es_lam)
fprintf("Achievable Eigen Vectors using Eigenstructure Assignment gains:\n")
disp(es_vec)
fprintf("I am trying to change it as little as possible but for some reason\n"
 + ...
    "a pole I am not trying to change is going unstable. With just two \n"
 + ...
    "outputs only two sets of eigenvectors and values should be set\n")
```

```
**********Part B**********
Ackermann's Formula
Desired Short Period Poles:
 -12.8000 + 9.6000i
 -12.8000 - 9.6000i

Gains are K = [0.423237  -0.010149  0.000678  0.000308]
Poles using Ackermann gains:
 -12.8000 + 9.6000i
```

```
 -12.8000 - 9.6000i
  -0.0104 + 0.7629i
  -0.0104 - 0.7629i

Eigenstructure Assignment using open loop eigen vectors
Desired Short Period Poles:
 -12.8000 + 9.6000i
 -12.8000 - 9.6000i

Gains are K = [0.965489  0.037551]
Poles using Eigenstructure Assignment gains:
 -12.7248 + 9.8392i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i -12.7248 - 9.8392i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   5.8561 - 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0468 + 0.0000i

Achievable Eigen Vectors using Eigenstructure Assignment gains:
  -0.0024 - 0.0913i  -0.0024 + 0.0913i   0.0555 - 0.0000i  -0.0027 - 0.0000i
   0.9597 + 0.0000i   0.9597 + 0.0000i   0.9494 + 0.0000i   0.0002 - 0.0000i
  -0.2166 + 0.1423i  -0.2166 - 0.1423i  -0.2632 - 0.0000i   1.0000 + 0.0000i
  -0.0472 - 0.0365i  -0.0472 + 0.0365i   0.1621 + 0.0000i  -0.0035 - 0.0000i

Eigenstructure Assignment of only alpha
Desired Short Period Poles:
 -12.8000 + 9.6000i
 -12.8000 - 9.6000i

Gains are K = [1.096662  0.042946]
Poles using Eigenstructure Assignment gains:
 -12.8000 + 9.6000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i -12.8000 - 9.6000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   6.7308 - 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0603 + 0.0000i

Achievable Eigen Vectors using Eigenstructure Assignment gains:
  -0.0035 - 0.0932i  -0.0035 + 0.0932i   0.0544 - 0.0000i  -0.0027 - 0.0000i
   0.9571 + 0.0000i   0.9571 + 0.0000i   0.9762 + 0.0000i   0.0002 - 0.0000i
  -0.2184 + 0.1548i  -0.2184 - 0.1548i  -0.1515 - 0.0000i   1.0000 + 0.0000i
  -0.0479 - 0.0359i  -0.0479 + 0.0359i   0.1450 + 0.0000i  -0.0031 - 0.0000i

I am trying to change it as little as possible but for some reason
a pole I am not trying to change is going unstable. With just two
outputs only two sets of eigenvectors and values should be set
```


Ackermann gains Response to 10 Degree Initial Pitch
Amplitude
To: pitch
To: pitch rate
0.2
0.1
0
-0.1
-0.2
0.1
0.05
0
-0.05
-0.1
0
100
200
300
400
500
600
Time (seconds)

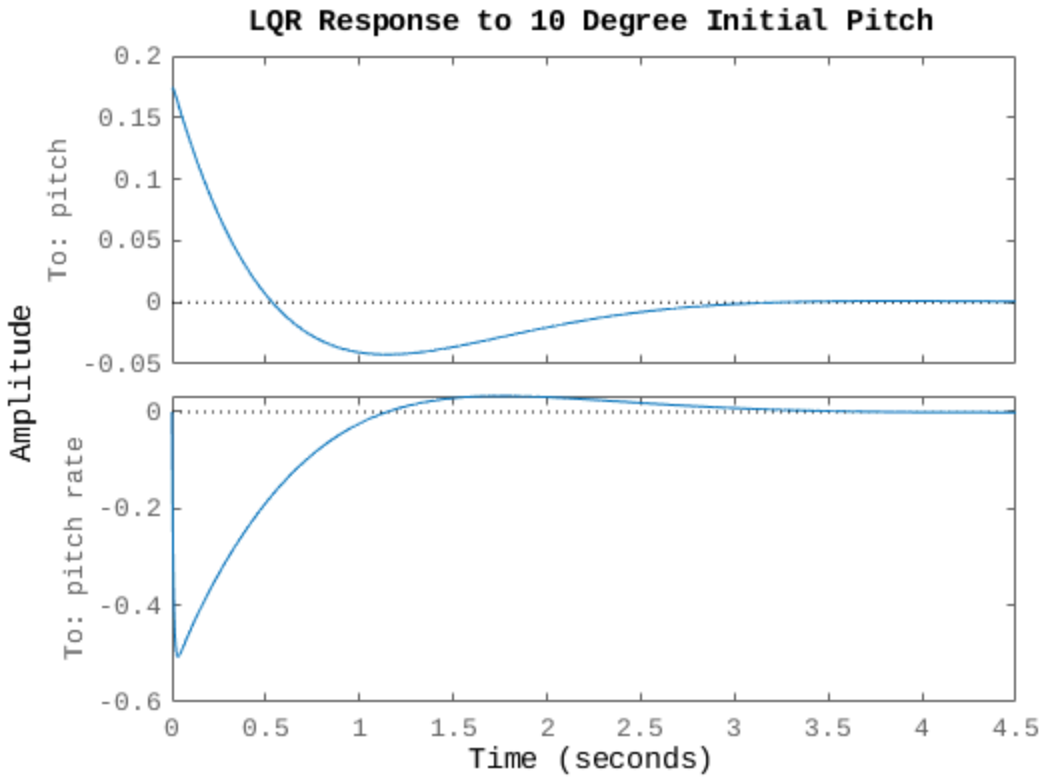

Eigenstructure Assignment gains Response to 10 Degree Initial Pi
Amplitude
To: pitch
To: pitch rate
×10^24
5
4
3
2
1
0
×10^25
2.5
2
1.5
1
0.5
0
0
2
4
6
8
10
Time (seconds)

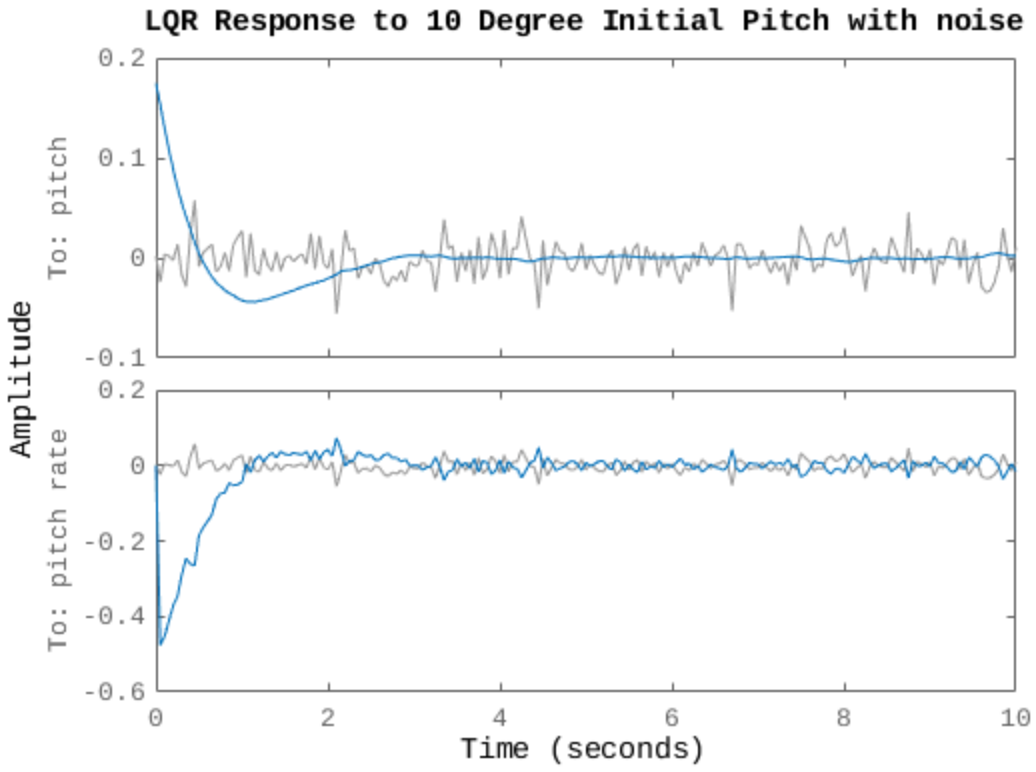


Part C

```
fprintf("***************Part C***************\n")
alpha_max = 5;
q_max = 10;
VT_max = 100;
theta_max = 10;
demax = 10;
Q = diag([1/alpha_max^2, 1/q_max^2, 1/VT_max^2, 1/theta_max^2]);
R = 1/demax^2;
% R = 1e-16;
[Klqr, Slqr, CLPlqr] = lqr(sso, Q, R);
ssc = ss(A-B*Klqr,B,C,0, ...
    'StateName', {'alpha', 'q', 'VT', 'theta'}, ...
    'InputName', {'elevator angle'}, ...
    'OutputName', {'pitch', 'pitch rate'});
disp("Solution to Riccati")
disp(Slqr)
disp("Gains")
disp(Klqr)
figure()
initial(ssc, [0, 0, 0, deg2rad(10)])
title("LQR Response to 10 Degree Initial Pitch")

inp1deg = deg2rad(randn(201,1));
figure()
```

```
lsim(ssc, inp1deg, linspace(0, 10, 201), [0 0 0 deg2rad(10)])
title("LQR Response to 10 Degree Initial Pitch with noise")

***************Part C***************
Solution to Riccati
    0.0043   -0.0001    0.0004   -0.0074
   -0.0001    0.0001   -0.0000    0.0003
    0.0004   -0.0000    0.0001   -0.0013
   -0.0074    0.0003   -0.0013    0.0389

Gains
    1.2657   -0.9316    0.0850   -3.0820
```





Part D

```
fprintf("***************Part D***************\n")
Umat = [B, eye(4)];
sys2 = ss(A, Umat, C, 0);

inp_1deg = deg2rad(randn(201,1)*.1);
w_1 = [];
for ii = 1:length(inp_1deg)
    w_1(:,ii) = B.*inp_1deg(ii);
end
Qn_1 = cov(w_1');
% Qn = eye(4);
Rn_1 = cov(inp_1deg)*.1;
[Kest_1, L, P] = kalman(sys2, Qn_1, Rn_1);
reg_1 = lqgreg(Kest_1, Klqr);
closed_loop_1 = feedback(sso, -reg_1);
disp("Solution with noise on input of noise with normal distribution \sigma=.1
 degree")
disp("standard deviation of .1 degree")
disp("Solution to Observer Riccati")
disp(P)
disp("Observer Gains")
disp(L)
figure()
```

```
lsim(closed_loop_1, inp_1deg, linspace(0, 30, 201), [0 0 0 deg2rad(10) 0 0 0
 0])
title("LQG Response to 10 Degree Initial Pitch with input noise \sigma=.1
 degree")


w = [];
for ii = 1:length(inp1deg)
    w(:,ii) = B.*inp1deg(ii);
end
Qn = cov(w');
% Qn = eye(4);
Rn = cov(inp1deg)*.1;
[Kest, L, P] = kalman(sys2, Qn, Rn);
reg = lqgreg(Kest, Klqr);
% disp(eig(reg))
closed_loop = feedback(sso, -reg);

disp("Solution with noise on input of noise with normal distribution \sigma=1
 degree")
disp("standard deviation of 1 degree")
disp("Solution to Observer Riccati")
disp(P)
disp("Observer Gains")
disp(L)

figure()
initial(closed_loop, [0 0 0 deg2rad(10) 0 0 0 0], linspace(0, 30, 301))
title("LQG Response to 10 Degree Initial Pitch without noise")

figure()
lsim(closed_loop, inp1deg, linspace(0, 30, 201), [0 0 0 deg2rad(10) 0 0 0 0])
title("LQG Response to 10 Degree Initial Pitch with input noise \sigma=1
 degree")


inp10deg = deg2rad(randn(201,1)*10);
w10 = [];
for ii = 1:length(inp10deg)
    w10(:,ii) = B.*inp10deg(ii);
end
Qn10 = cov(w10');
% Qn = eye(4);
Rn10 = cov(inp10deg);
[Kest10, L, P] = kalman(sys2, Qn10, Rn10);
reg10 = lqgreg(Kest10, Klqr);
closed_loop10 = feedback(sso, -reg10);
disp("Solution with noise on input of noise with normal distribution \sigma=10
 degree")
disp("standard deviation of 10 degree")
disp("Solution to Observer Riccati")
disp(P)
disp("Observer Gains")
disp(L)
```

```
figure()
lsim(closed_loop10, inp10deg, linspace(0, 30, 201), [0 0 0 deg2rad(10) 0 0 0
 0])
title("LQG Response to 10 Degree Initial Pitch with input noise \sigma=10
 degree")

disp("After testing with different magnitudes of noise it is clear that the")
disp("the obvious conclusion is correct that the systems work better with ")
disp("less noise. It seems like 10 degree standard deviation for the noise")
disp("is far too much. The pitch rate seems more effected than the pitch by ")
disp("the noise for some reason. Noise was modeled as a noisy input in lsim ")
disp("and it was scaled to 10, 1, and .1 degrees. The input was shifted to  ")
disp("radians before being input to lsim")

% %% Functions
%
% function K = Ackermann(A, B, lambda)
%     n = length(A);
%
%     syms s
%     funs = 1;
%     for ii = 1:length(lambda)
%         funs = funs*(s-lambda(ii));
%     end
%     fdes_s = collect(funs);
%     coef = sym2poly(fdes_s);
%     fdes_A = 0;
%     m = length(coef);
%     for ii = 1:m
%         fdes_A = fdes_A + coef(ii)*A^(n-1);
%     end
% %     fdes_A = coef(1).*A*A + coef(2).*A + coef(3).*eye(n);
%
%     Con = [];
%     for ii = 1:n
%         Con = [Con, A^(ii-1)*B];
%     end
%
%     leftv = [zeros(1,n-1), 1];
%     K = leftv*inv(Con)*fdes_A;
% %     fprintf("Control Gains are K = \n")
% %     disp(K)
% end
%
% function K = EigStructAssign(A, B, C, des_lambda, des_vec)
%     n = length(A);
%     m = size(B,2);
%     p = size(des_vec,1);
%
%     I = eye(n);
%     D = diag(ones(1,p));
%     O = zeros(p,(n+m)-p);
%     top = zeros(n,1);
%     uv = @(lambda,vec) pinv([lambda.*I - A, B; D, O])*[top;vec];
```

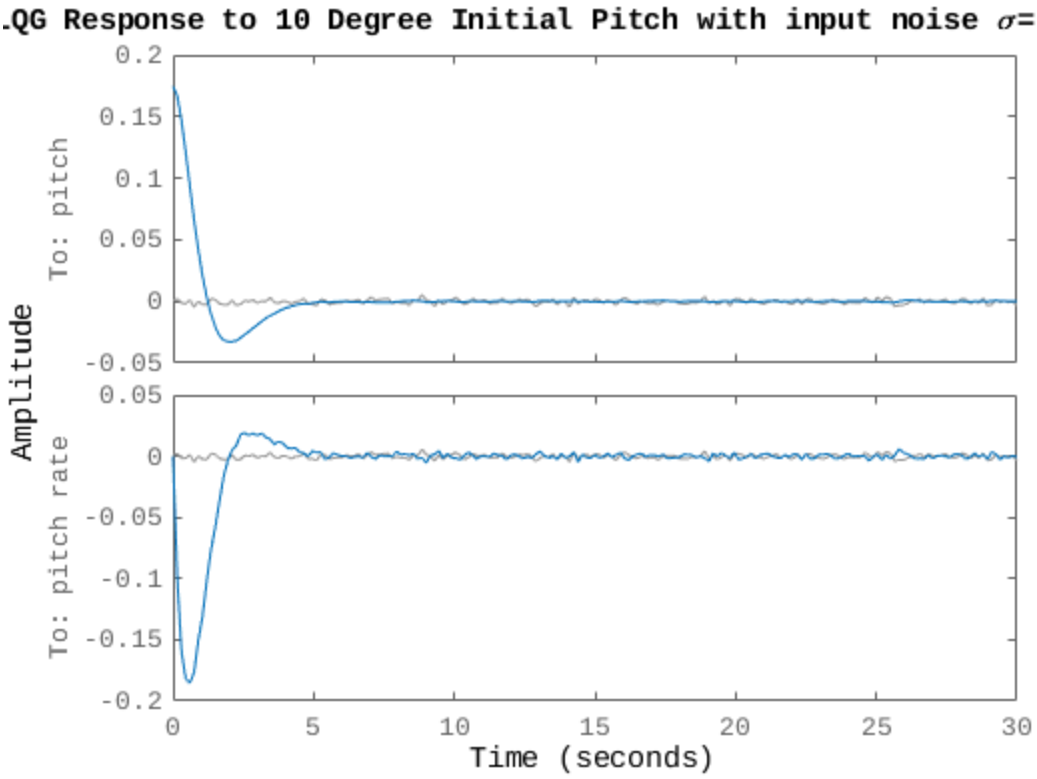
```
%
%     for ii = 1:length(des_lambda)
%         uvii = uv(des_lambda(ii), des_vec(:,ii));
%         vd(:,ii) = uvii(1:n);
%         ud(:,ii) = uvii(n+1:end);
%     end
%     K = ud*pinv(C*vd);
%     fprintf("Control Gains are K = \n")
%     disp(K)
%     poles = eig(A-B*K*C);
%     fprintf("Closed Loop Poles are \\lamda = \n")
%     disp(poles)
%     fprintf("Desired Eigen Vectors are v = \n")
%     disp(des_vec)
%     fprintf("Achievable Eigen Vectors are v = \n")
%     disp(vd)
% end
%
% function K = outputLQR(A, B, C, Q, R, x0, tol)
%     n = length(A);
%     m = size(B,2);
%     p = size(C,1);
%     P = sym("P", [n n]);
%     S = sym("S", [n n]);
% %     syms P S
%     if x0 == 0
%         x0 = zeros(n,1);
%     end
%     X = x0*x0';
%     X = eye(n);
%     kk = 0;
%     Kk = zeros(m,p);
%     Ak = A - B*Kk*C;
%     eqPk = 0 == Ak.'*P + P*Ak + C.'*Kk.'*R*Kk*C + Q;
%     eqSk = 0 == Ak*S + S*S.' + X;
%     sol = vpasolve([eqSk],[S]);
% %     Pk = sol.Pk
%     Sk = zeros(n);
%     Pk = zeros(n);
%     Sk1 = zeros(n);
%     Pk1 = zeros(n);
%     for ii = 1:n
%         for jj = 1:n
%             Sk(ii,jj) = eval(['sol.S', int2str(ii), '_', int2str(jj)]);
%             Pk(ii,jj) = eval(['sol.P', int2str(ii), '_', int2str(jj)]);
%         end
%     end
%     Jk = (1/2)*trace(Pk*X);
%     alpha = 1e-2;
%     err  = 1;
%     while err > tol
%         Ak = A - B*Kk*C;
%         eqPk = 0 == Ak'*P + P*Ak + C'*Kk'*R*Kk*C + Q;
%         eqSk = 0 == Ak*S + S*S' + X;
```

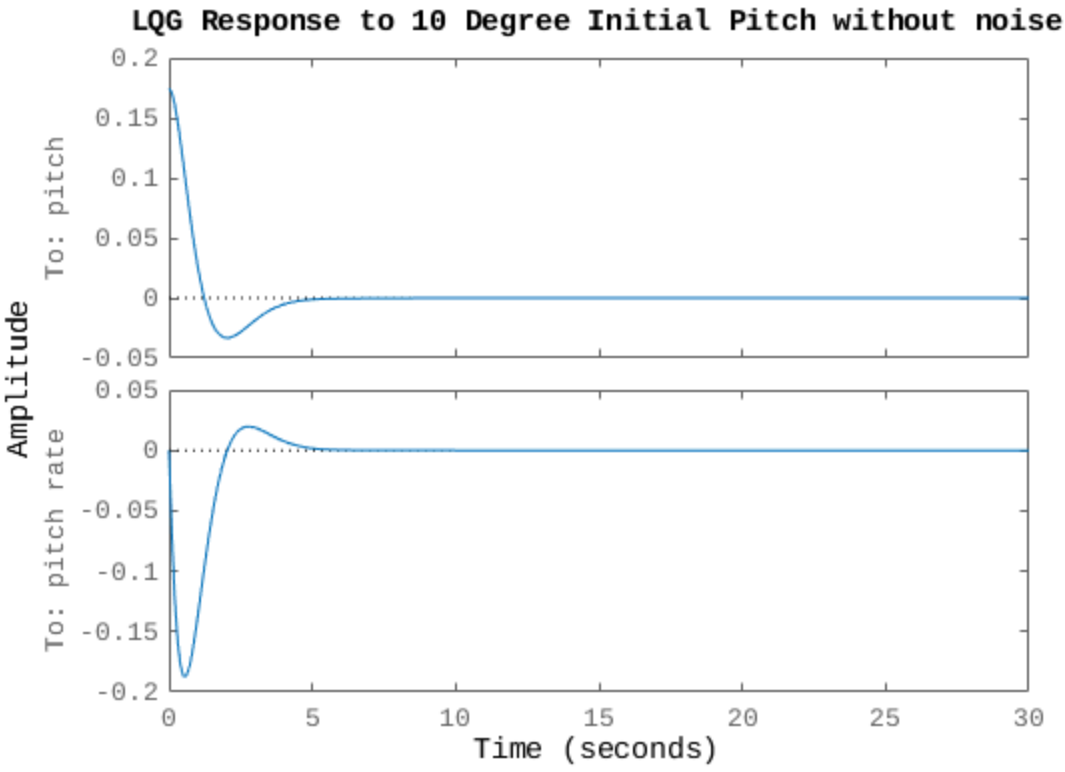
```
%         sol = vpasolve([eqPk,eqSk],[S,P]);
%         for ii = 1:n
%             for jj = 1:n
%                 Sk1(ii,jj) = eval(['sol.S', int2str(ii), '_', int2str(jj)]);
%                 Pk1(ii,jj) = eval(['sol.P', int2str(ii), '_', int2str(jj)]);
%             end
%         end
%         Jk1 = (1/2)*trace(Pk1*X);
%         deltaK = inv(R)*B'*Pk1*Sk1*C'*inv(C*Sk1*C')-Kk;
%         Kk1 = Kk + alpha*deltaK;
%         err = abs(Jk1-Jk);
%         kk = kk + 1;
%         Sk = Sk1;
%         Pk = Pk1;
%         Jk = Jk1;
%         Kk = Kk1;
%     end
%     K = Kk;
% end
```

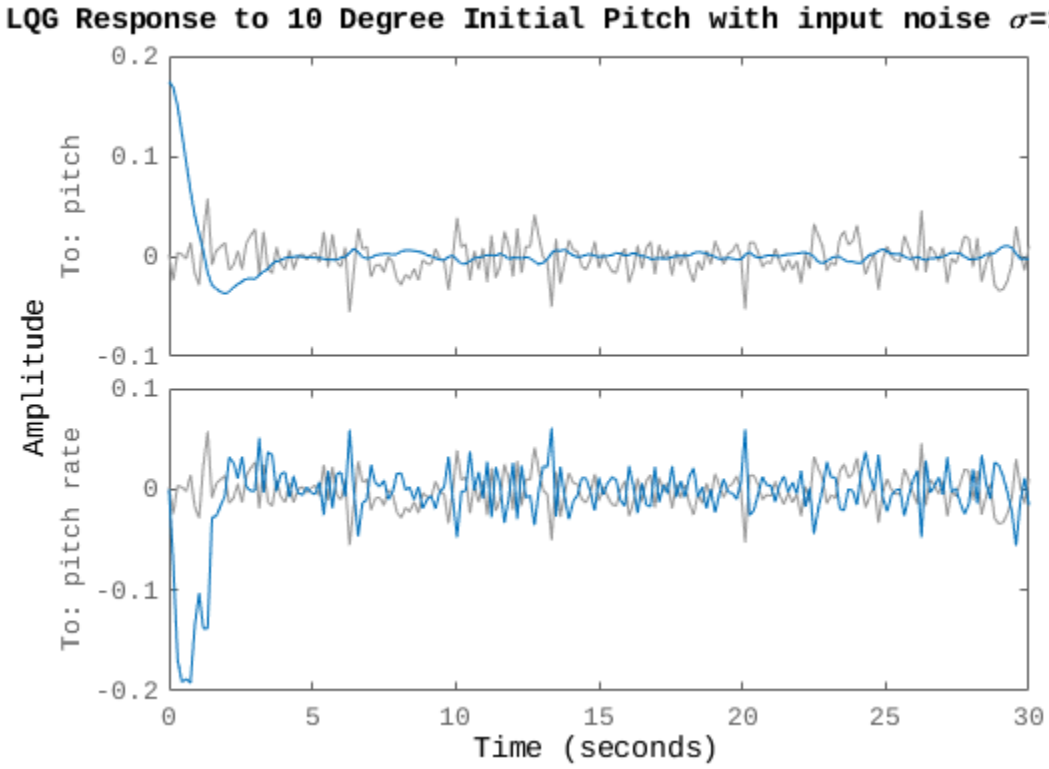
```
***************Part D***************
Solution with noise on input of noise with normal distribution \sigma=.1
 degree
standard deviation of .1 degree
Solution to Observer Riccati
    0.0000    0.0000   -0.0000    0.0000
    0.0000    0.0001    0.0000    0.0000
   -0.0000    0.0000    0.0011   -0.0000
    0.0000    0.0000   -0.0000    0.0000

Observer Gains
    0.1350    3.0409
    0.9174  403.8414
  -23.5701   14.1732
    0.9966    0.9174

Solution with noise on input of noise with normal distribution \sigma=1 degree
standard deviation of 1 degree
Solution to Observer Riccati
    0.0000    0.0001   -0.0002    0.0000
    0.0001    0.0120    0.0004    0.0000
   -0.0002    0.0004    0.0915   -0.0007
    0.0000    0.0000   -0.0007    0.0000

Observer Gains
    0.1350    3.0409
    0.9174  403.8414
  -23.5701   14.1732
    0.9966    0.9174

Solution with noise on input of noise with normal distribution \sigma=10
 degree
standard deviation of 10 degree
Solution to Observer Riccati
```

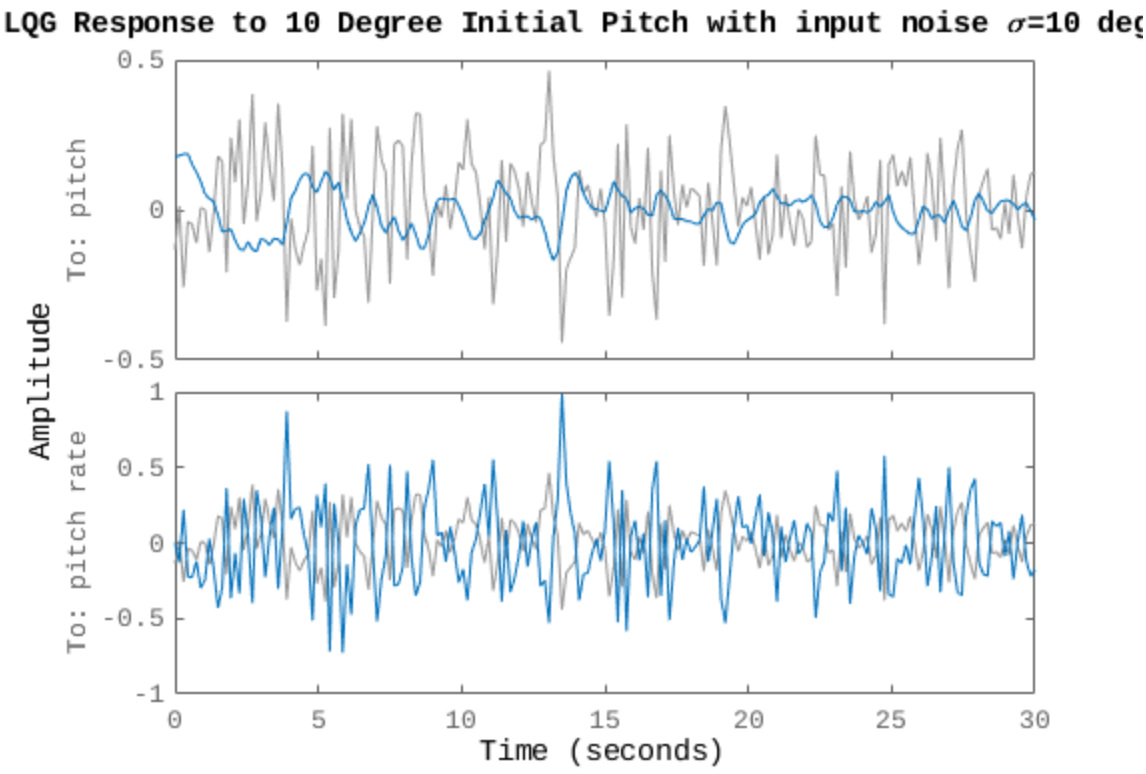
```
    0.0018    0.0391   -0.2011    0.0036
    0.0391    3.4278    0.3702    0.0216
   -0.2011    0.3702   80.0111   -0.6263
    0.0036    0.0216   -0.6263    0.0279

Observer Gains
    0.1260    1.3555
    0.7499  118.8819
  -21.7224   12.8388
    0.9682    0.7499
```

After testing with different magnitudes of noise it is clear that the the obvious conclusion is correct that the systems work better with less noise. It seems like 10 degree standard deviation for the noise is far too much. The pitch rate seems more effected than the pitch by the noise for some reason. Noise was modeled as a noisy input in lsim and it was scaled to 10, 1, and .1 degrees. The input was shifted to radians before being input to lsim




LQG Response to 10 Degree Initial Pitch without noise
Amplitude
To: pitch
To: pitch rate
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
5
10
15
20
25
30
Time (seconds)


LQG Response to 10 Degree Initial Pitch with input noise σ=1 deg
Amplitude
To: pitch
To: pitch rate
0.2
0.1
0
-0.1
-0.2
0
5
10
15
20
25
30
Time (seconds)



Published with MATLAB® R2022a