# 1. Discrete-Time System.

A discrete time system is given by

, 

Write a MATLAB m file to simulate the system, i.e. to compute  for a given input , initial condition , and range of the time index *k= 1,2,…,N.*

## a. Simulate the system



for  equal to the unit step and =0. Plot  vs. k for 100 time samples.

Find the period and percent overshoot.

**

*Period and percent overshoot where estimated from the plotted values of the simulation.*

*Period, T = 31-11 = 20 steps (assume units is seconds)*

*Percent Overshoot = abs(17.27-11.13)/11.13 = 55%*

## b. Simulate the same system but now add process noise so that

.

Take the noise  as uniformly distributed between 0 and 0.2. Use MATLAB function rand. Plot  vs. k for 100 time samples.



*Period and percent overshoot where estimated from the plotted values of the simulation.*

*Period, T = 31-11 = 20 steps (assume units is seconds)*

*Percent Overshoot = abs(17.42-11.38)/11.38 = 53%*

# 2. DT Kalman Filter

Write a MATLAB m file to simulate a DT system





plus DT Kalman Filter.

Simulate the optimal time-varying DT Kalman Filter for the system

,

.

Take the process noise  as a normal 2-vector (MATLAB function randn) with each component having zero mean and variance 0.1. Take measurement noise  as normal (0,0.1). Use  equal to the unit step and =0.

Plot the states and their estimates on the same graphs. i.e.  on one graph, and  on another graph.





For a choice of R =0.1, Q= G =

the result is  and P =

Please notice that this not a unique solution, it is dependent on the choice of R,Q, and G.

# 3. Steady-state DT Kalman Filter

## a. Find the steady-state Kalman gain by iteration on the time-varying Riccati difference equation.

For a choice of R =0.1, Q= G =

the result is  and P =

Please notice that this not a unique solution, it is dependent on the choice of R,Q, and G.

## b. Find the steady-state Kalman gain by solution of the ARE using dlqe in MATLAB.

For a choice of R =0.1, Q= G =

the result is  and P =

Please notice that this not a unique solution, it is dependent on the choice of R,Q, and G.

## c. Simulate the system in problem 2 with the steady-state Kalman Filter, which has a constant gain. Compare the results with the optimal Kalman filter in problem 2.

**

**

**

*The result of the steady state Kalman filter is more robust than the time varying one because the error covariance matrix which is generating the Kalman gain is calculated prior to the simulation.*

# MATLAB® Codes:

Problem 1;

function Homework3\_P1

clear all; close all; clc

A = [0,1;-0.89,1.8];

B = [0 1]';

N = 100; % Time Range

u = ones(N,1); % Unit Step

x0 = [0 0]'; % Initial Conditions

[Time1,States1] = Sim1(A,B,x0,u,0,0);

%%

figure(1); hold on;

plot(Time1,States1); title('Discrete Time System Simulation "NO Noise"');

xlabel('Time (Seconds)'); ylabel('Amplitude'); grid on

h1= legend('$x\_1(t)$','$x\_2(t)$'); set(h1, 'interpreter', 'latex');

[Time2,States2] = Sim1(A,B,x0,u,0,0.2);

figure(2); hold on;

plot(Time2,States2); title('Discrete Time System Simulation " with Noise"');

xlabel('Time (Seconds)'); ylabel('Amplitude'); grid on

h2= legend('$x\_1(t)$','$x\_2(t)$'); set(h2, 'interpreter', 'latex');

%%

function [Time,State] = Sim1(A,B,InCon,u,NoiseMin,NoiseMax)

N =size(u,1);

Time(1)= 1;

n = size(InCon,1);

State=zeros(N,n);

State(1,:)=InCon';

for Count = 1:N-1

Time(Count+1)=Count+1;

State(Count+1,:)=...

(A\*State(Count,:)' + B\*u(Count,:)' +...

(NoiseMin + (NoiseMax\*rand)))';

end

Time=Time';

Problem 2;

function Homework3\_P2

clear all; close all; clc

A = [0,1;-0.89,1.8]; % State System Matrix

B = [0 1]'; % System Input Matrix

N = 100; % Time Range (Number of Samples)

G\_ = eye(2); % Process noise

u =[0 ones(1,N-1)]; % Unit Step

InCon =[0 0]'; % Initial Conditions

Wlim = [0 0.1]; % process noise zero mean and variance 0.1

Vlim = [0 0.1]; % Measurement noise normal (0,0.1).

H = [1 0]; % Output Matrix

R = 0.1; % R Symmetric and positive semidefinite here Scalar

Q = 0.1\* eye(size(A)); % Q symmetric and positive semidefinite

[Time,X,~,~,~,Xx,~, Kk] = Sim2(A,B,H,InCon,u,Wlim,Vlim,G\_,Q,R);

%% Plots

figure (1);

plot(Time,X(1,:),'b','LineWidth',2); hold on

plot(Time,Xx(1,:),'r','LineWidth',2);

xlabel('Time (Seconds)'); ylabel('Amplitude'); grid on

h1= legend('$x\_1(t)$','$\hat{x\_1}(t)$');

set(h1, 'interpreter', 'latex');

figure(2);

plot(Time,X(2,:),'b','LineWidth',2); hold on

plot(Time,Xx(2,:),'r','LineWidth',2);

xlabel('Time (Seconds)'); ylabel('Amplitude'); grid on

h2= legend('$x\_2(t)$','$\hat{x\_2}(t)$');

set(h2, 'interpreter', 'latex');

disp(Kk(:,end))

function [Time,X,Y,X\_,P\_,X1,P, Kk] = Sim2(A,B,H,InCon,u,Wlim,Vlim,G,Q,R)

Sz = size(A,1); % to Select the system order (Size)

N = length(u); % Time Range (Number of Samples) absed on imput

Time = zeros(1,N); % Keep track of the time

X = zeros(Sz,N); % Creating Vector for Faster Computing

X1 = X; X\_ = X; Kk = X; % Creating Vector for Faster Computing

P = ones(Sz); P\_ = P; % Creating Matrices for Faster Computing

Y = zeros(1,Sz); % Creating Vector for Faster Computing

X(:,1) = InCon; % Selecting initial condition

% creating the process noise zero mean and variance.

W = Wlim(1) + Wlim(2)\* randn(Sz,length(u));

for idx = 1:N-1

Time(1,idx+1)=idx;

X(:,idx+1) = A\*X(:,idx) + B\*u(:,idx) + W(:,idx);

Y(:,idx+1) = H\*X(:,idx) + (Vlim(1) + (Vlim(2)\*rand));

% Time Update (effect of system dynamics)

P\_(:,:,idx+1)= A\*P(:,:,idx)\*A' + G\*Q\*G';

X\_(:,idx+1) = A\*X1(:,idx) + B\*u(:,idx);

% Measurement Update (effect of measurement) Z\_k

P(:,:,idx+1) = inv(inv(P\_(:,:,idx+1))+ H'\* R^-1 \* H);

X1(:,idx+1) = X\_(:,idx+1)+ P(:,:,idx+1)\*H'\*R^-1.\*(Y(:,idx+1)-H\*X\_(:,idx+1));

% Kalman Filter Gain

Kk(:,idx+1) = P\_(:,:,idx) \*H' \* inv(H\* P\_(:,:,idx) \* H'+R);

end

Problem 3;

function Homework3\_P3

%% Initializing

clear all; close all; clc

A = [0,1;-0.89,1.8]; % State System Matrix

B = [0 1]'; % System Input Matrix

N = 100; % Time Range (Number of Samples)

G\_ = eye(2); % Process noise

u =[0 ones(1,N-1)]; % Unit Step

InCon =[0 0]'; % Initial Conditions

Wlim = [0 0.1]; % process noise zero mean and variance 0.1

Vlim = [0 0.1]; % Measurement noise normal (0,0.1).

H = [1 0]; % Output Matrix

R = 0.1; % R Symmetric and positive semidefinite here Scalar

Q = 0.1\* eye(size(A)); % Q symmetric and positive semidefinite

% Matlab Subrotine for Kalman Gain (Problem 3 B)

[Km,P\_,~,~] = dlqe(A,G\_,H,Q,R);

% Steady State Kalman Filter (Problem 3 A )

[Time,P, Kss] = Sim3\_a(A,H,G\_,Q,R,u);

% Simulation of SS Kalman (Problem 3 C)

[~,X2,Y2] = Sim3\_c (A,B,InCon,H,u,Kss,Vlim);

% Simulation of time Varying Kalman filter (Problem 2)

[~,Xo,Y1,~,~,X1,~, ~] = Sim2(A,B,H,InCon,u,Wlim,Vlim,G\_,Q,R);

% Displaying the Kalman Gains via Both ways in Command Line

disp([Kss(:,end-1), Km]);

disp([P(:,:,end), P\_]);

% Notice that the Matlab rotine and this rotine have the same Results

%% Plots

figure(1);

plot(Time,Xo(1,:),'b','LineWidth',2); hold on

plot(Time,X1(1,:),':r','LineWidth',2);

plot(Time,X2(1,:),'--g','LineWidth',2);

title('Estimated State1 Using Steady State Kalman Filter');

xlabel('Time (Seconds)'); ylabel('Amplitude'); grid on

h1= legend('$x\_1(t)$','$\hat{x}\_1.TV(t)$','$\hat{x}\_1.SS(t)$');

set(h1, 'interpreter', 'latex');

figure(2);

plot(Time,Xo(2,:),'b','LineWidth',2); hold on

plot(Time,X1(2,:),':r','LineWidth',2);

plot(Time,X2(2,:),'--g','LineWidth',2);

title('Estimated State2 Using Steady State Kalman Filter');

xlabel('Time (Seconds)'); ylabel('Amplitude'); grid on

h1= legend('$\hat{x}\_2(t)$','$\hat{x}\_2.TV(t)$','$\hat{x}\_2.SS(t)$');

set(h1, 'interpreter', 'latex');

figure(3);

plot(Time,Y1,'r','LineWidth',2); hold on

plot(Time,Y2,':g','LineWidth',2);

title('Estimated output using Steady State Kalman Filter');

xlabel('Time (Seconds)'); ylabel('Amplitude'); grid on

h1= legend('$Z(t)TV $','$Z(t) SS $');

set(h1, 'interpreter', 'latex');

function [Time,X,Y,X\_,P\_,X1,P, Kk] = Sim2(A,B,H,InCon,u,Wlim,Vlim,G\_,Q,R)

Sz = size(A,1); % to Select the system order (Size)

N = length(u); % Time Range (Number of Samples) absed on imput

Time = zeros(1,N); % Keep track of the time

X = zeros(Sz,N); % Creating Vector for Faster Computing

X1 = X; X\_ = X; Kk = X; % Creating Vector for Faster Computing

P = ones(Sz); P\_ = P; % Creating Matrices for Faster Computing

Y = zeros(1,Sz); % Creating Vector for Faster Computing

X(:,1) = InCon; % Selecting intial condtion

% creating the process noise zero mean and variance.

W = Wlim(1) + Wlim(2)\* randn(Sz,length(u));

for idx = 1:N-1

Time(1,idx+1)=idx;

X(:,idx+1) = A\*X(:,idx) + B\*u(:,idx) + W(:,idx);

Y(:,idx+1) = H\*X(:,idx) + (Vlim(1) + (Vlim(2)\*rand));

% Time Update (effect of system dynamics)

P\_(:,:,idx+1)= A\*P(:,:,idx)\*A' + G\_\*Q\*G\_';

X\_(:,idx+1) = A\*X1(:,idx) + B\*u(:,idx);

% Measurement Update (effect of measurement) Z\_k

P(:,:,idx+1) = inv(inv(P\_(:,:,idx+1))+ H'\* R^-1 \* H);

X1(:,idx+1) = X\_(:,idx+1)+ P(:,:,idx+1)\*H'\*R^-1.\*(Y(:,idx+1)-H\*X\_(:,idx+1));

% Kalman Filter Gain

Kk(:,idx+1) = P\_(:,:,idx) \*H' \* inv(H\* P\_(:,:,idx) \* H'+R);

end

function [Time,P, Kss] = Sim3\_a(A,H,G,Q,R,u)

% Sim3\_c is to Simulate the steady-state Kalman Filter

Sz = size(A,1); % to Select the system order (Size)

N = length(u); % Time Range (Number of Samples) absed on imput

P = zeros(Sz); % create Matrices for faster computing

Time = zeros(1,N); % Keep track of the time

Kss = zeros(Sz,N); % create Matrices for faster computing

for idx = 1:N-1

Time(1,idx+1)=idx; % Update Time

P(:,:,idx+1)= A\*(P(:,:,idx)-P(:,:,idx)\*H'...

\*inv(H\*P(:,:,idx)\*H'+R)\*H\*P(:,:,idx))\*A'+ G\*Q\*G'; % Update ErCov

Kss(:,idx) = P(:,:,idx) \*H' \* inv(H\* P(:,:,idx) \* H'+R); % update K gain

end

function [Time,X,Y,Xo] = Sim3\_c (A,B,InCon,H,u,K,Vlim)

% Sim3\_c is to Simulate the steady-state Kalman Filter

K = K(:,end); % Selecting The Last gain from pervious rotine

Sz = size(A,1); % Select the system order (Size)

N =length(u); % Time Range (Number of Samples) absed on imput

X = zeros(Sz,N); % Creat A vector for faster computing

X(:,1) = InCon; % Select the intial Condition

Xo = zeros(Sz,N); % Creat A vector for faster computing

Xo(:,1) = InCon; % Select the intial Condition

Y = zeros(1,N); % Creat A vector for faster computing

Time = zeros(1,N); % Creat A vector for faster computing

for idx = 1:N-1

% Update the Time

Time(1,idx+1)=idx;

% Update the States Values (SS Kalman)

X(:,idx+1) = A\*(eye(Sz)-K\*H)\*X(:,idx) + B\*u(:,idx) +(A\*K\*Y(:,idx));

% Update the States Values (Orignal)

Xo(:,idx+1)= A\*Xo(:,idx) + B\*u(:,idx) + (0 + (0.1\*rand));

% Update the Output Values (SS Kalman)

Y(:,idx+1) = H\*X(:,idx) + (Vlim(1) + (Vlim(2)\*rand));

end