Q1（10 marks）

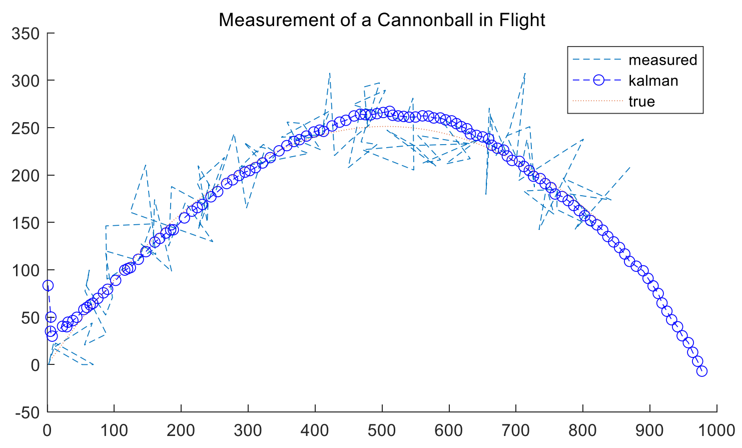
a. the Matlab code for implementing the Kalman filter for this particular example.

Codes:

```
T=14.4;%Simulation time
Tt=0.1;%Simulation interval
muzzle_velocity = 100; % How fast should the cannonball come out?
angle = 45; % Angle from the ground.
speedX = muzzle_velocity*cos(angle*pi/180);
speedY = muzzle_velocity*sin(angle*pi/180);
V=[speedX,speedY];%The initial velocity
noiselevel = 20;
x = [];
y = [];
x2 = [];
y2 = [];
nx = [];
ny = [];
kx = [];
ky = [];
control_vector=[0;0;0.5*-9.81*0.1*0.1;-9.81*0.91];
Q=zeros(4);
Z=eye(4);
H=eye(4);
R=0.2*eye(4);
I=eye(4);
A=[1,0.1,0,0;0,1,0,0;0,0,1,0.1;0,0,0,1];
B=[0,0,0,0;0,0,0,0;0,0,1,0;0,0,0,1];
current_state_estimate=[0;speedX;500;speedY];
current_prob_estimate=eye(4);
gravity = [0,-9.81];
velocity = [muzzle_velocity*cos(angle*pi/180), muzzle_velocity*sin(angle*pi/180)];
loc = [0,0]; % The initial location of the cannonball.
acceleration = [0,0]; % The initial acceleration of the cannonball.
figure;
hold on;
index = 1;
for i=0:Tt:T
    %%
    predicted_state_estimate=A*current_state_estimate+B*control_vector;
    predicted_prob_estimate = (A*current_prob_estimate)*A'+Q;
    Kg=predicted_prob_estimate*H'*(H*predicted_prob_estimate*H'+R)^(-1);
    Pn=(I-Kg*H)*predicted_prob_estimate;
    Xn=predicted_state_estimate+Kg*(Z-H*predicted_state_estimate);
    %%
```

```
        timeslicevec = [Tt,Tt];
        sliced_gravity = gravity.*timeslicevec;
        sliced_acceleration = sliced_gravity;
        velocity = velocity+sliced_acceleration;
        sliced_velocity = velocity.*timeslicevec;
        loc =loc+sliced_velocity;
        if loc(2) < 0
            loc(2) = 0
        end
        x2(index) =loc(1);
        y2(index) =loc(2);
        nx(index) = normrnd(x2(index),noiselevel);
        ny(index) = normrnd(y2(index),noiselevel);
        if ny(index) <= 0
        ny(index)=0;
        end
        kx(index) = Xn(1,1);
        ky(index) = Xn(3,1);
        if ky(index) <= 0
        break;
        end
        Z=[nx(index),V(1),ny(index),V(2)];
        current_state_estimate = Xn;
        current_prob_estimate = Pn;
        index=index+1;
end
plot(nx,ny,'--',kx,ky,'b--o',x2,y2,':')
title('Measurement of a Cannonball in Flight')
legend('measured','kalman','true')
```

b. Image of result



c. Comment on the effect of changing the process and measurement noise covariance matrices.

The changing process is mainly to figure out how to set the most important parameter matrices in the algorithm.The source of the measurement noise covariance is the sensor error, that is the inaccuracy of the sensor. Generally, the sensor will give an accuracy index when it is used.

Q2 (15 marks) In this problem, we shall look at the variable lamda RLS algorithm and the Kalman filtering algorithm for tracking experiments.

Implement in Matlab, the RLS algorithm with a variable memory parameter as describled in Haykin's textbook. ***(Full codes are in upload files.)***

a.

```
 end
 % RLS algorithm
 delta = 0.004; % Adjustment parameters
 w = zeros(M,L+1);
 epsilon = zeros(L,1);
 alfa = 1;
 P1 = eye(M)/delta;
 lamda = eye(1,L+1);
 temp = zeros(1,L+1);
 Fai = eye(1,2);
 for k = 1:L
     lamda(k)=0.98;
 end
 % RLS Iterative algorithm process
 for k=1:L
 PIn = P1 * A(:,k);
 denok = lamda(k)+ A(:,k)'*PIn;
 kn = PIn/denok;
 epsilon(k) = b(k)-w(:,k)'*A(:,k);
 w(:,k+1) = w(:,k) + kn*conj(epsilon(k));
 P1 = P1/lamda(k) - kn*A(:,k)'*P1/lamda(k);
 S = diff(P1,1);
 Fai = Fai*(I-kn*A(:,k)')+S*A(:,k)*conj(epsilon(k));
 temp = alfa*real(Fai*A(:,k)*conj(epsilon(k)));
 lamda(k+1) = lamda(k) + 0.0001;
 end
 w1(kk,:) = w(1,:);
 w2(kk,:) = w(2,:);
 MSE = abs(epsilon).^2;
 MSE_P(kk) = mean(MSE);
end
```

b. Use the above algorithm to track a time-varying system described by a first order Markov process given by $w_o(n+1)=aw_o(n)+b(n)$.

Where w_o=[1 0.5 0.1 -0.4 0.8], a is the model parameter equal to 0.67 and b(n) is a white noise process with a variance of 0.001.

```
clc,clear,close all % Clear the screen, clear the work area, close the window
```

```
warning off % Eliminate warning
feature jit off % Speed up code execution
N = 1000; % Signal observation length
a1 = 0.67; % First-order AR parameters
sigma = 0.001; % Additive white noise variance
for kk =1:100
  v = sqrt(sigma)*randn(N,1); % Generate v(n) additive white noise
u0 = [1,0.5,0.1,-0.4,0.8]; % Initial data
num = 1; % Numerator coefficient
den = [1,a1]; % Denominator coefficient
Zi = filtic(num,den,u0); % Initial conditions of the filter
un = filter(num,den,v,Zi); % Generate sample sequence u(n), N x 1 x trials
  % Generate expected response signal and observation data matrix
  n0 = 1; % Virtual realization of n0-step linear prediction
  M = 2; % Filter order
  b = un(n0+1:N); % Predicted expected response
  L = length(b);
  un1 = [zeros(M-1,1)',un']; % Extended data
  A = zeros(M,L);
  I =eye(M);
  for k=1:L
  A(:,k) = un1(M-1+k : -1 : k);% Construct observation data matrix
  end
  % Apply RLS algorithm to iteratively optimize and calculate the optimal weight vector
  delta = 0.004; % Adjustment parameters
  w = zeros(M,L+1);
  epsilon = zeros(L,1);
  alfa = 1;
  P1 = eye(M)/delta;
  lamda = eye(1,L+1);
  temp = zeros(1,L+1);
  Fai = eye(1,2);
  for k = 1:L
        lamda(k)=0.98;
  end
  % RLS Iterative algorithm process
  for k=1:L
  PIn = P1 * A(:,k);
  denok = lamda(k)+ A(:,k)'*PIn;
  kn = PIn/denok;
  epsilon(k) = b(k)-w(:,k)'*A(:,k);
  w(:,k+1) = w(:,k) + kn*conj(epsilon(k));
  P1 = P1/lamda(k) - kn*A(:,k)'*P1/lamda(k);
  S = diff(P1,1);
  Fai = Fai*(I-kn*A(:,k)')+S*A(:,k)*conj(epsilon(k));
  temp = alfa*real(Fai*A(:,k)*conj(epsilon(k)));
  lamda(k+1) = lamda(k) + 0.0001;
  end
  w1(kk,:) = w(1,:);
  w2(kk,:) = w(2,:);
  MSE = abs(epsilon).^2;
  MSE_P(kk) = mean(MSE);
end
W1 = mean(w1); %
W2 = mean(w2); %
figure,plot(1:kk,MSE_P,'r','linewidth',2),title('平均  MSE');grid on;
figure,plot(1:length(W1),W1,'r','linewidth',2),title('平均  MSE');hold on;
plot(1:length(W2),W2,'b','linewidth',2),title('权值');hold on;
grid on;legend('\alpha1=0','\alpha2=-1')
```

c. Plot the ensemble-averaged learning curves of the variable and fixed lamda RLS algorithms and compare their performance.

- **Variable RLS algorithm:**

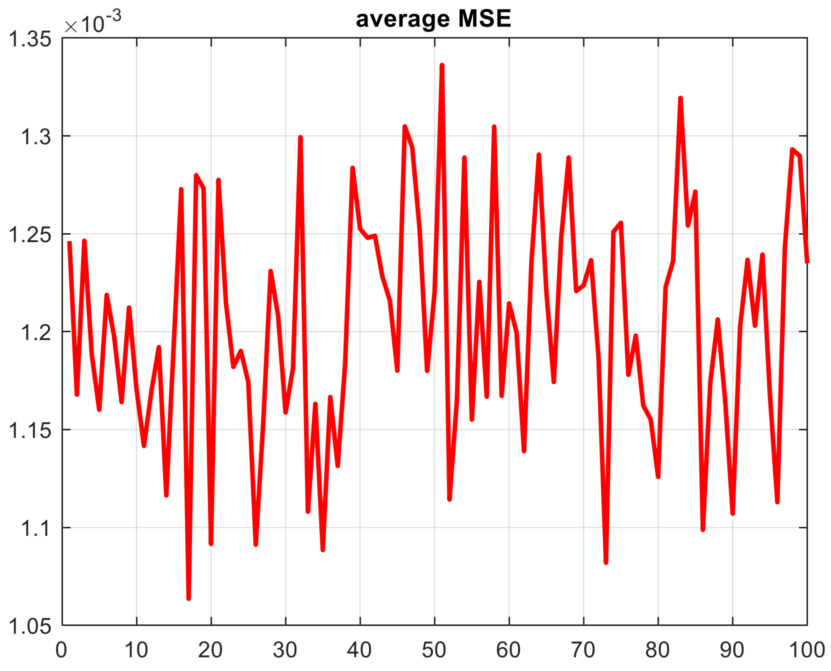


Figure 2. Average MSE of the variable lamda RLS algorithm

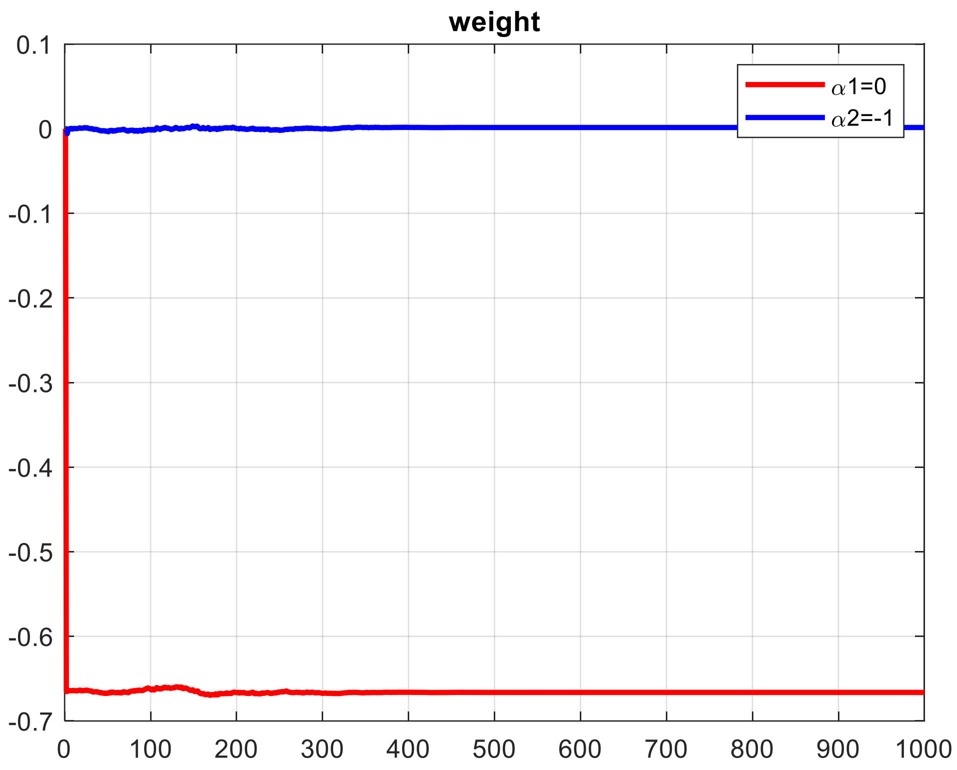


Figure 3. Weight of the variable lamda RLS algorithm

- Fixed lamada RLS algorithm

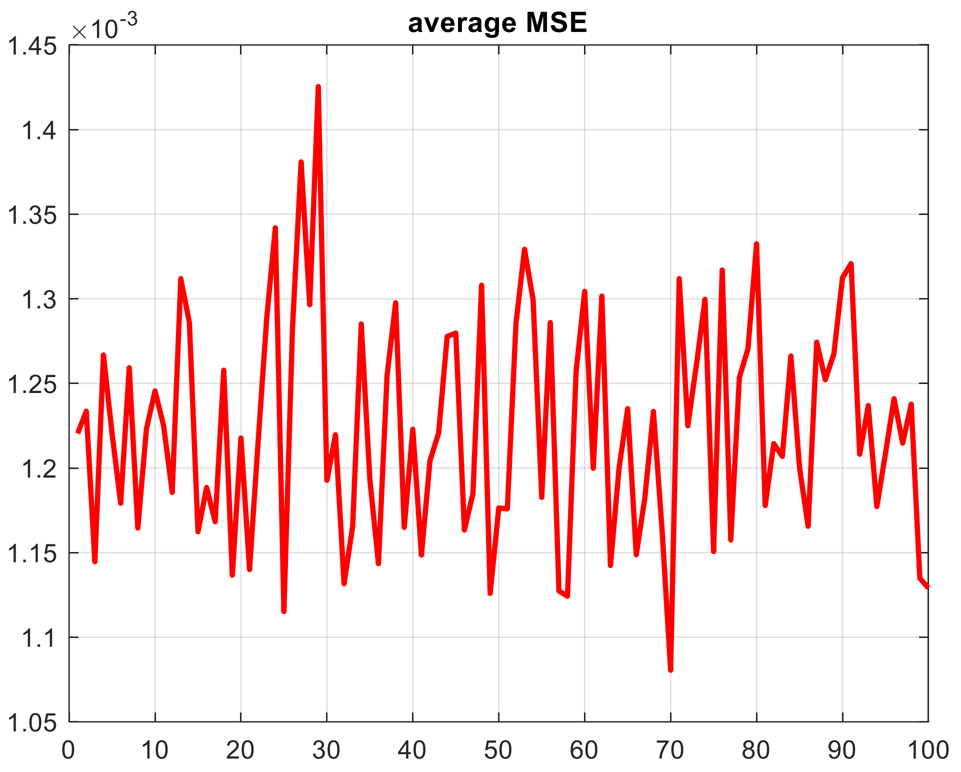


Figure 4. Average MSE of the fixed lamda RLS algorithm

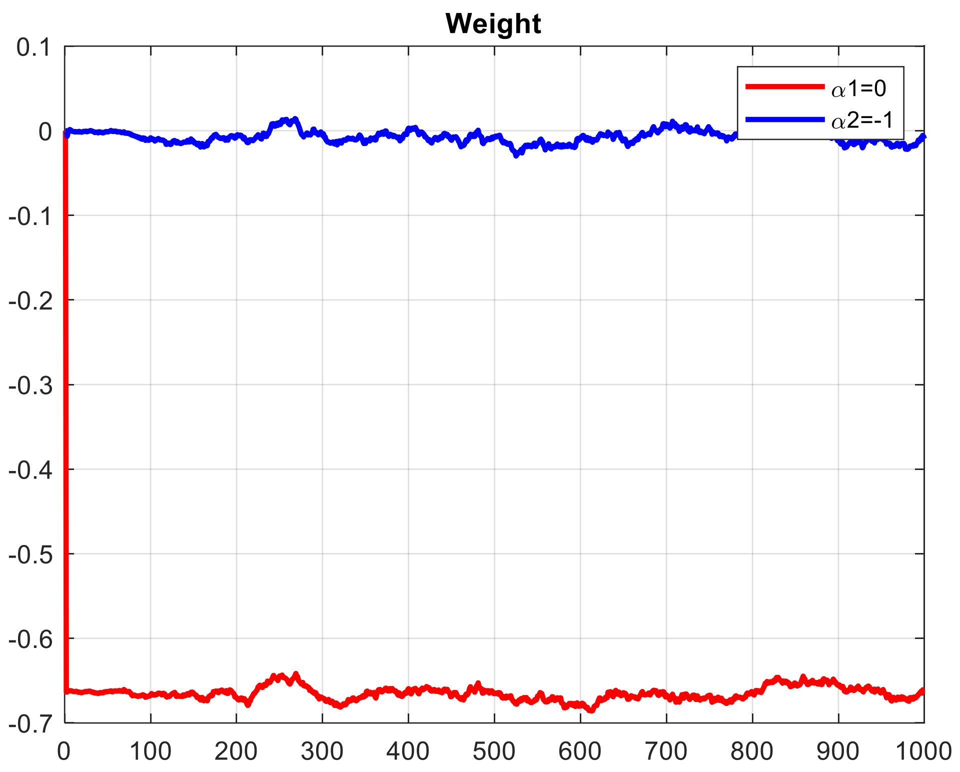


Figure 5. Weight of the Fixed lamda RLS algorithm

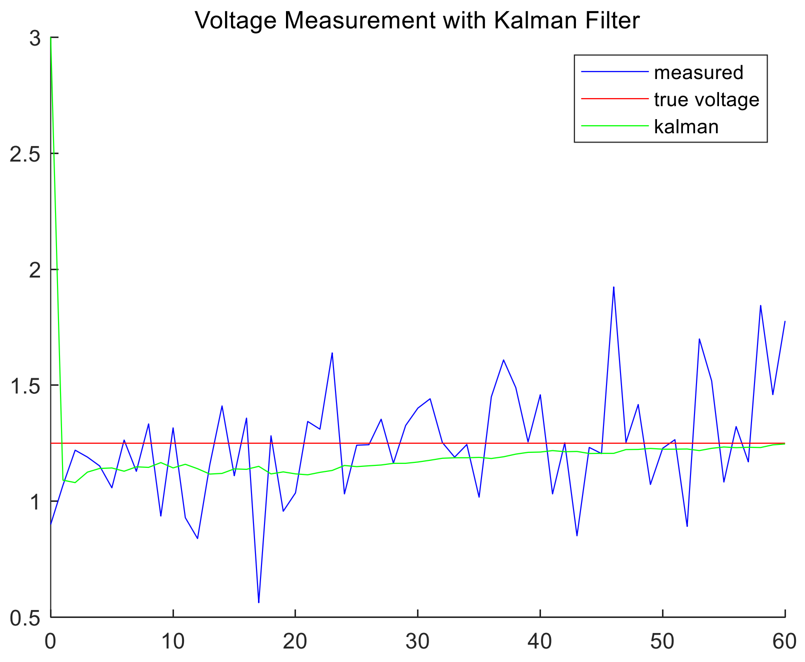
d. Formulate the Kalman filtering algorithm for estimating the time-varying system.Through simulations, compare the perfomances of Kalman filtering algorithm to the algorithms in step(c).

Codes:

```
clc,clear,close all
T=60;
Tt=1;
Q=0.00001*eye(1);
Z=eye(1);
H=eye(1);
R=0.1*eye(1);
I=eye(1);
A=[1];
B=[1];
P=[1];
xhat = [3]
measuredvoltage = [];
truevoltage = [];
kalman = [];
current_state_estimate=[3];
current_prob_estimate=eye(1);
```

```
figure;
hold on;
t=1.25;
e= 0.25;
control_vector = [0];
index = 1;
for i=0:Tt:T
        measured = normrnd(t,e);
        measuredvoltage(index) = measured;
        truevoltage(index) = t;
        kalman(index)= current_state_estimate;
        %%
        predicted_state_estimate=A*current_state_estimate+B*control_vector;
        predicted_prob_estimate = (A*current_prob_estimate)*A'+Q;
        Kg=predicted_prob_estimate*H'*(H*predicted_prob_estimate*H'+R)^(-1);
        current_prob_estimate=(I-Kg*H)*predicted_prob_estimate;
        Z = [measured];
        current_state_estimate=predicted_state_estimate+Kg*(Z-H*predicted_state_estimate);
        %%
        index=index+1;
end
plot(0:Tt:T,measuredvoltage,'b',0:Tt:T,truevoltage,'r',0:Tt:T,kalman,'g')
title('Voltage Measurement with Kalman Filter')
legend('measured','true voltage','kalman')
```

Result:



Comments:
RLS is actually Kalman's update step. This step is estimated using measurements. Kalman also has an extra prediction step, thanks to KF must have the support of the space-state model, this step Kalman can predict the state without using measurement. RLS can only perform estimation updates when there is a measurement update, while Kalman can actually be deduced by model when there is no measurement update.