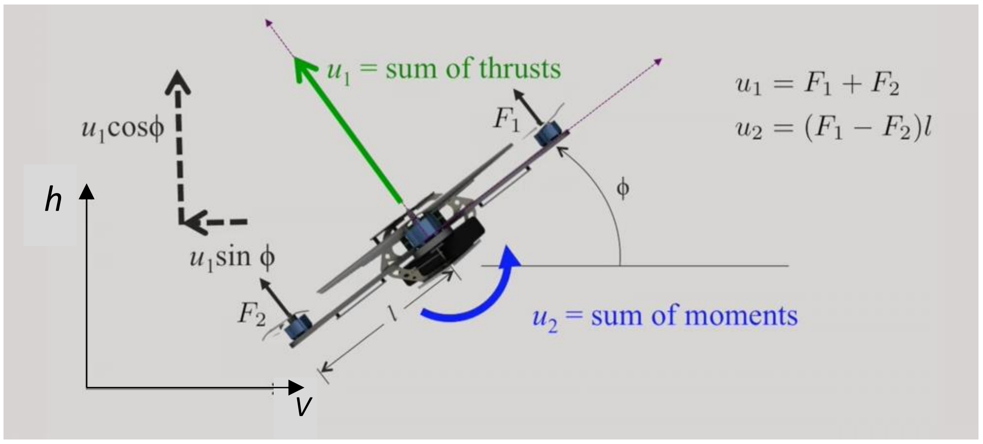
OPTIMAL CONTROL AND KALMAN FILTERING ON A 2D QUADCOPTER

Quadcopters are a useful tool for researchers to test and evaluate new ideas in a number of different fields, including flight control theory, navigation, real time systems, and robotics. Academics are working together to make significant improvements to the way quadcopters perform increasingly complex aerial maneuvers.

In this project we consider the dynamics of a quadcopter in the 2D vertical plane and provide an optimal control using concepts from Linear Quadratic Regulator and Algebraic Riccatti Equation. A Kalman Filter is implemented later to generate estimates of the states of the system from noisy measurements.

Part 1 . Mathematical modelling of a 2D Quadcopter:



[1]. Free Body Diagram of the Quadcopter in 2D vertical plane

The equations of motion of the quadcopter in the vertical plane is given by :

$$m\ddot{h} = u_1 \sin\phi,$$
$$m\ddot{v} = -mg + u_1 \cos\phi,$$
$$I\ddot{\phi} = u_2$$

where (*h*, *v*) denote the quadrotor (horizontal, vertical) position and ϕ denotes the quadrotor's rotation, (*m*, *I*) denote quadrotor (mass, inertia), *g* is acceleration due to gravity, and (*u*1, *u*2) denote the net (thrust, torque) applied by the spinning rotors.

If we measure or observe positions (*h*, *v*), e.g. with GPS, then the control system model is :

$$\frac{d}{dt}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ F((q,\dot{q}),u) \end{bmatrix} = f((q,\dot{q}),u),\ y = h(q,\dot{q})$$

where $q = (h, v, \theta) \in \mathbb{R}^3$, $u = (u_1, u_2) \in \mathbb{R}^2$, $F : \mathbb{R}^3 \times \mathbb{R}^2 \rightarrow \mathbb{R}^3$ is defined by

$$F((q,\dot{q}),u) = \ddot{q} = \begin{bmatrix} \frac{u_1}{m}\sin\phi \\ -g + \frac{u_1}{m}\cos\phi \\ \frac{u_2}{I} \end{bmatrix},$$

and $h : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ is defined by

$$h(q,\dot{q}) = (h, v).$$

Simulation of the Nonlinear dynamics:

1(a). The Quadcopter dynamics were simulated using Forward Euler numerical approximation. Functions *fun* and *h*, that take *(t, u, x)* as arguments and return the dynamics *fun*(*t*,*x*,*u*) and the observation *h*(*t*,*x*,*u*) respectively.

Taylor expansion of the function y around t_0 is given by:

$$y(t_0+h) = y(t_0) + hy'(t_0) + \frac{1}{2}h^2 y''(t_0) + O(h^3).$$

So the finite difference formula for the derivative becomes:

$$y'(t_0) \approx \frac{y(t_0+h) - y(t_0)}{h}$$

Function fun:

$$f(t,x,u) = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \\ \dot{x_6} \end{bmatrix} = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ \frac{sin(x_3)u_1}{m} \\ \frac{cos(x_3)u_1}{m} - g \\ \frac{u_2}{I} \end{bmatrix}$$

Matlab:
```
function xdot=fun(t,x,U)

        g=9.81;m=1;I=1;w=1;
        u = U(:,t);

        h=x(1); v=x(2); th=x(3); dh=x(4); dv=x(5); dth=x(6); u1=u(1); u2=u(2);

        dot2h = (u1*sin(th))/m;
        doth  = dh;
        dot2v = -m*g +(cos(th)*u1);
        dotv = dv;
        dot2th = (u2*1/I);
        dotth = dth;

        xdot = [doth;dotv;dotth;dot2h;dot2v;dot2th];
```

1(b) . Equilibrium states and inputs

For equilibrium df = 0. This will help us find the equilibrium states. Substituting u1 and u2 in f, we get equilibrium to be (0,0,0,0,0,0) for (h,v,θ,$\dot{h}$,$\dot{v}$,$\dot{\theta}$)

1(c) . Forward Euler Simulation

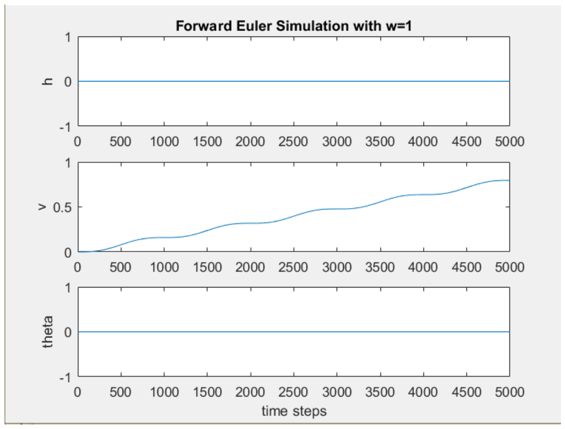
$$u(t) = \begin{bmatrix} mg + \sin(2t\pi\omega) \\ 0 \end{bmatrix}$$

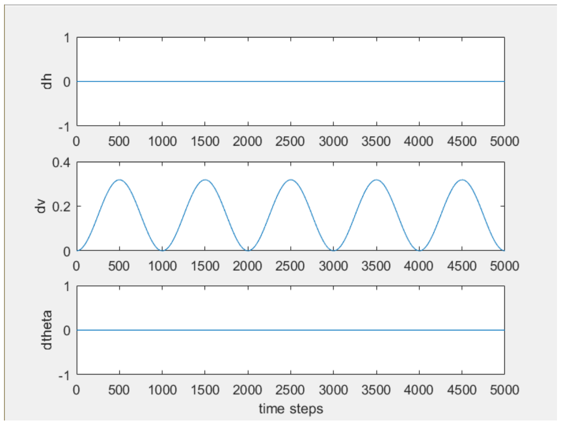
Matlab:
```
N=10000; %number of time steps
        t=linspace(0,10, N+1); %taking the time step
        step = t(2)-t(1);
        w=1;
```

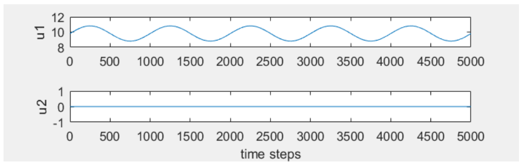
```
Qx = zeros(6, N+1); %initial condition is set at x[:,0] to [0;0;0;0;0;0]
    for i =1:N+1
        u(:,i)= [(m*g + sin(2*t(i)*pi*w));0];
    end
% Simulation forward euler:
    for i=2:N+1
        % defining the U vector
        Qx(:,i) = step*(fun(i,Qx(:,i-1),u))+ Qx(:,i-1);
    end
```

Plots :

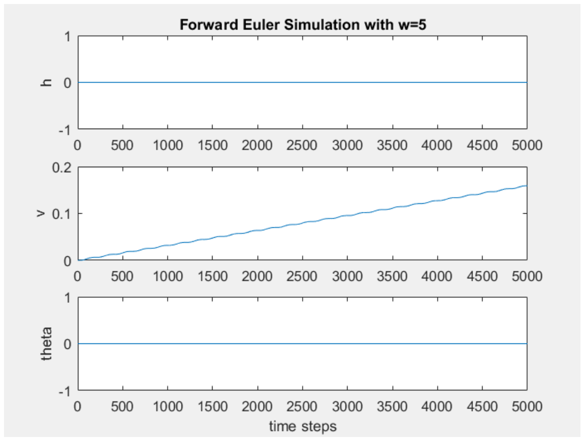
Forward Euler Simulation with w=1 :

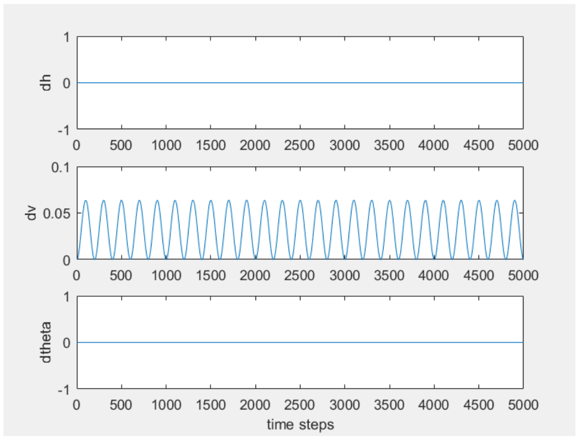


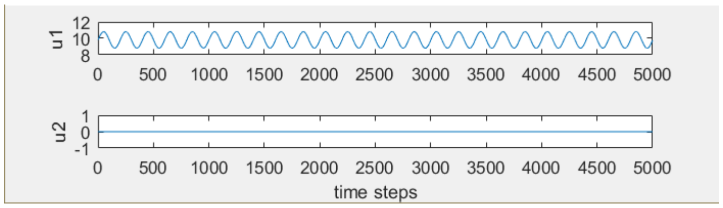




Forward Euler Simulation with w=5 :







Part 2 . <u>Stabilisation of the 2D Quadcopter:</u>

2(a) . Linearization

The system is linearised around the equilibrium [0 0.1 0 0 0 0]^T using first order approximation using Jacobian matrix.

$$\frac{\partial \dot{f}}{\partial x_1} = \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix};\ \frac{\partial \dot{f}}{\partial x_2} = \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix};\ \frac{\partial \dot{f}}{\partial x_3} = \begin{bmatrix}0\\0\\0\\\frac{\cos(x_3)u_1}{u_1}\\-\frac{\sin(x_3)u_1}{m}\\0\end{bmatrix};\ \frac{\partial \dot{f}}{\partial x_4} = \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \qquad \frac{\partial \dot{f}}{\partial x_5} = \begin{bmatrix}0\\1\\0\\0\\0\\0\end{bmatrix};\ \frac{\partial \dot{f}}{\partial x_6} = \begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix};\ \frac{\partial \dot{f}}{\partial u_1} = \begin{bmatrix}0\\0\\0\\\frac{\sin(x_3)}{m}\\\frac{cos(x_3)}{m}\\0\end{bmatrix};\ \frac{\partial \dot{f}}{\partial u_2} = \begin{bmatrix}0\\0\\0\\0\\0\\1\end{bmatrix}$$

Substituting values of equlibrium states [0 0.1 0 0 0 0]^T and inputs $[mg\ 0]^T$, we get :

$$A = \begin{bmatrix}0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\\0&0&g&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix};\ B = \begin{bmatrix}0&0\\0&0\\0&0\\\frac{sin(x_3)}{m}&0\\\frac{cos(x_3)}{m}&0\\0&1\end{bmatrix}$$

2(b) . Controllability

The controllability matrix of the system is given by : Ć = [A , AB , A^{2}B , A^{3}B , A^{4}B , A^{5}B]

The rank of the controllability matrix needs to be equal to the rank of the A matrix for the system to be controllable. Ć =

$$Ć\ = \ : \begin{bmatrix}0&0&0&0&0&0&0&9.8&0&0&0&0\\0&0&1&0&0&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&0&0&0\\0&0&0&0&0&g&0&0&0&0&0&0\\1&0&0&0&0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0&0&0&0\end{bmatrix}; \qquad \text{Rank(Ć) = 6 = Rank(A)}$$

Thus the system is controllable.

2(c) . Stabilizing state feedback control law using Lyapunov :

2(c) 1 . If (A, B) is controllable so is $(-\lambda I - A, B)$ for every $\lambda \in \mathbb{R}$.

Let us consider A which is a controllable matrix. First we write:

$$Ax = \lambda * x$$

Where x and λ are the eigenvector and eigenvalue pair of A. If we manipulate this expression by multiplying by -1 and subtracting μx on both sides we obtain:

$$-Ax - \mu * x = -\lambda * x - \mu * x$$
$$(-A - \mu)x = (-\lambda - \mu)x$$

Clearly, given (λ, x) is the eigenvalue and eigenvector pair for the controllable A then it follows from above that $(-A - \mu)$ is also controllable. Since they both have the same eigenvectors and therefore (λ, x) is an eigenvalue-eigenvector pair for A if and only if $(-\mu - \lambda, x)$ is an eigenvalue-eigenvector pair for $(-\mu I \ - \ A)$, implying that all these vector directions are controllable.

2(c) 2 . Finding K that stabilizes the system :

If $(-\mu I \ - \ A)$ is stable, the Lyapunov equation can be expressed as,

$$(-\mu I \ - \ A)W + W(-\mu I \ - \ A)^T = -BB^T$$

$$(A)W + W(\ A)^T - BB^T = -2\mu W$$

We can rewrite this in the Lyapunov form by multiplying by W^{-1} [ ] W^{-1} and letting $W^{-1} = P$

$$P(\ A) + (\ A)^T P - PBB^T P = -2\mu P$$

Let $B^T P = 2K$, then

$$P(\ A) + (\ A)^T P - 2PBK = -2\mu P$$

$$P(\ A) - PBK + (\ A)^T P - PBK = -2\mu P$$

$$P(\ A - BK) + (\ A - BK)^T P = -2\mu P$$

$$\text{Thus } K = \frac{1}{2} B^T P$$

Let the desired eigen values be, $p \ = [-3.5;\ -3.6;\ -3.7;\ -3.8;\ -4.5;\ -4.9]$

```
Matlab:%(-muI-A)W + W(-muI-A)' + BB' =  lyapunov
              lA = (diag(p) - A);
              lQ = B*B';
              W = lyap(lA,lQ);

      % from the calculation from 2(C)(3) we know that inv(P) = W; sp P = inv(W)
               lP = inv(W);
               K0= 0.5*B'*lP ;
```

The $K0$ for the desired poles is found to be,

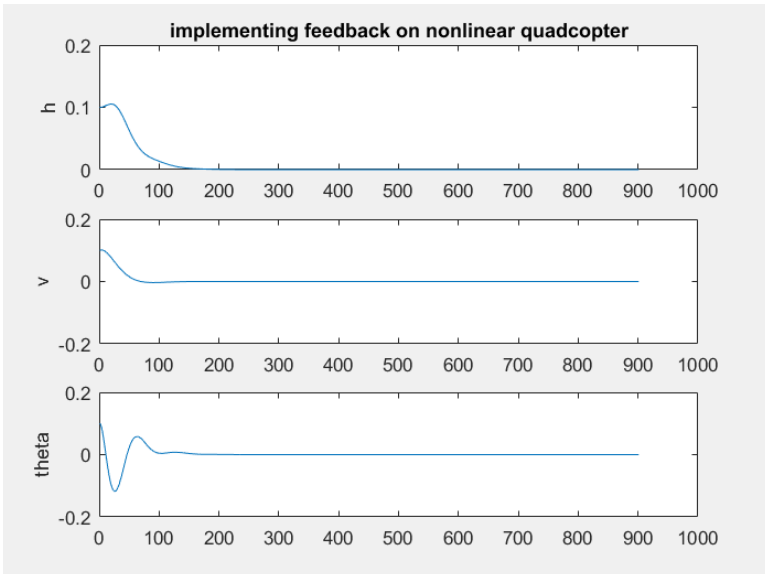
$$K0 = \begin{bmatrix} 0 & 29.1600 & 0 & 0 & 8.1000 & 0 \\ 157.6800 & 0 & 174.9000 & 90.0582 & 0 & 15.9000 \end{bmatrix}$$

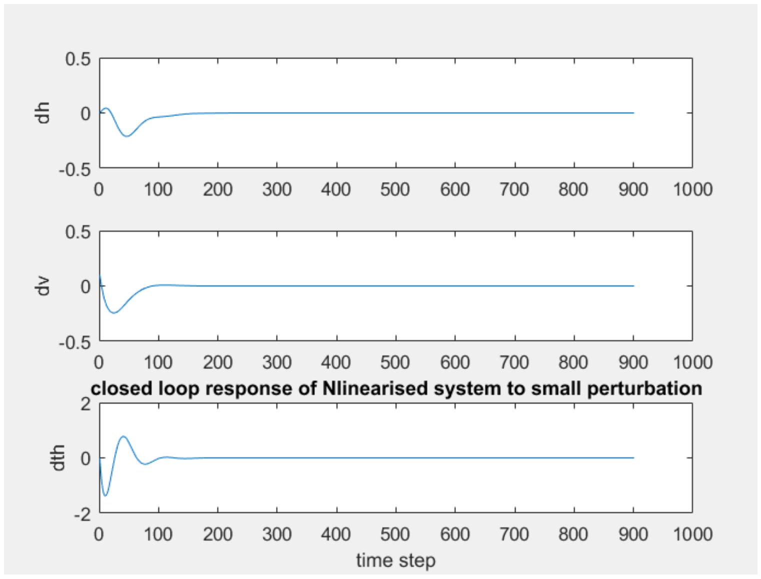
The eigen values of the closed loop system , $(A - B\ K0)$ are stable :

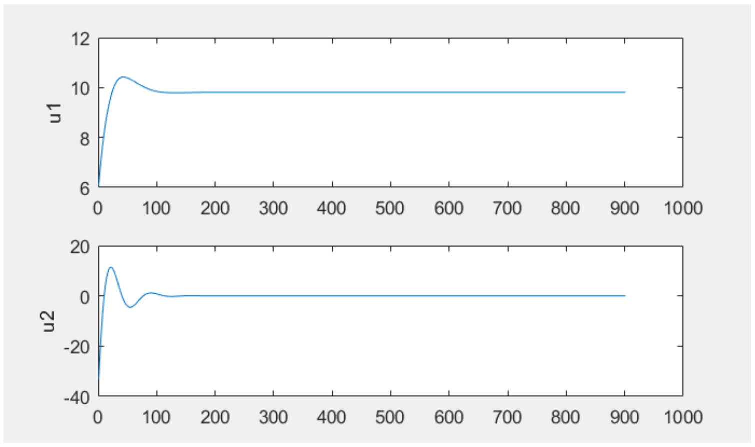
$$\begin{bmatrix} -4.0094 + 8.9135i \\ -4.0094 - 8.9135i \\ -3.9406 + 0.8051i \\ -3.9406 - 0.8051i \\ -4.0500 + 3.5718i \\ -4.0500 - 3.5718i \end{bmatrix}$$

2(d) . Simulation of the closed loop system in the equilibrium neighborhood:

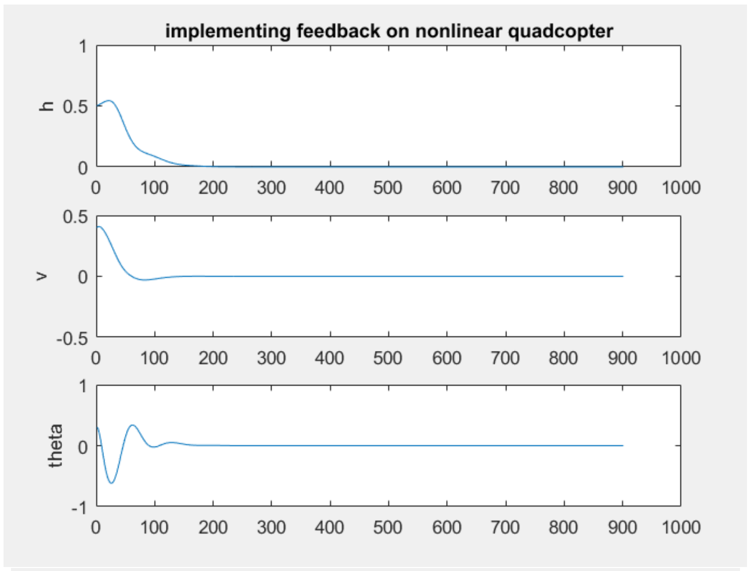
Closed Loop Response of Nonlinear System to $x0$ = $[0.1, 0.1, 0.1, 0, 0.1, 0]^T$**:**

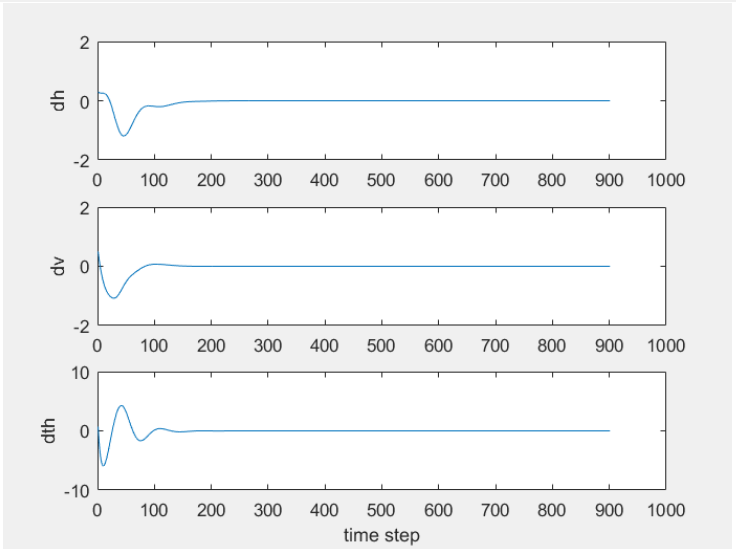


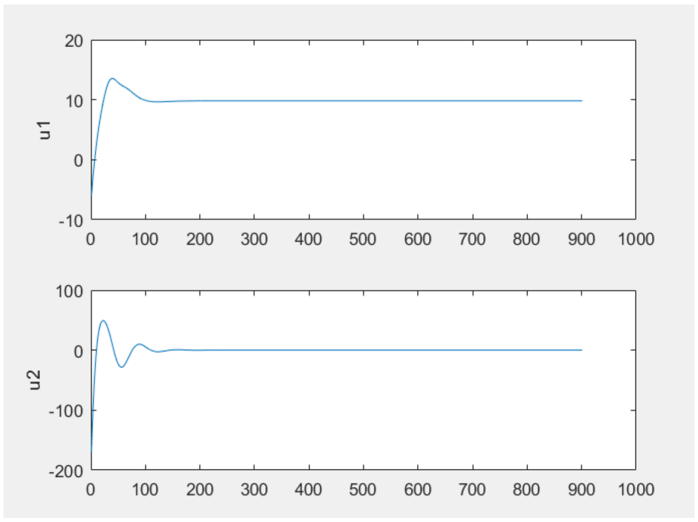




Closed Loop Response of Nonlinear System to $x0$ = $[0.1, 0.1, 0.1, 0, 0.1, 0]^T$:







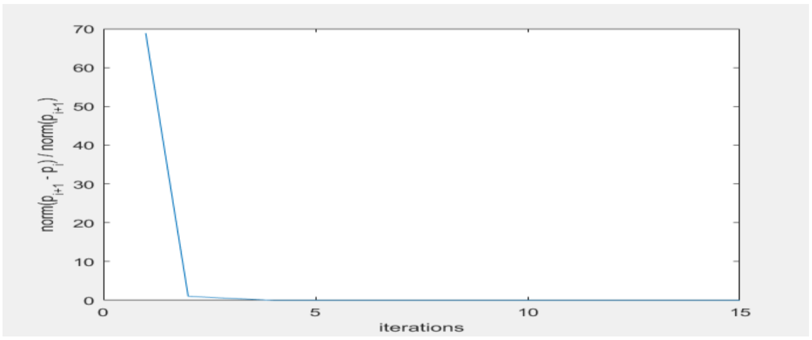
Part 3 . Continuous time LQR design:

3(a).1 Generating positive definite Matrices :

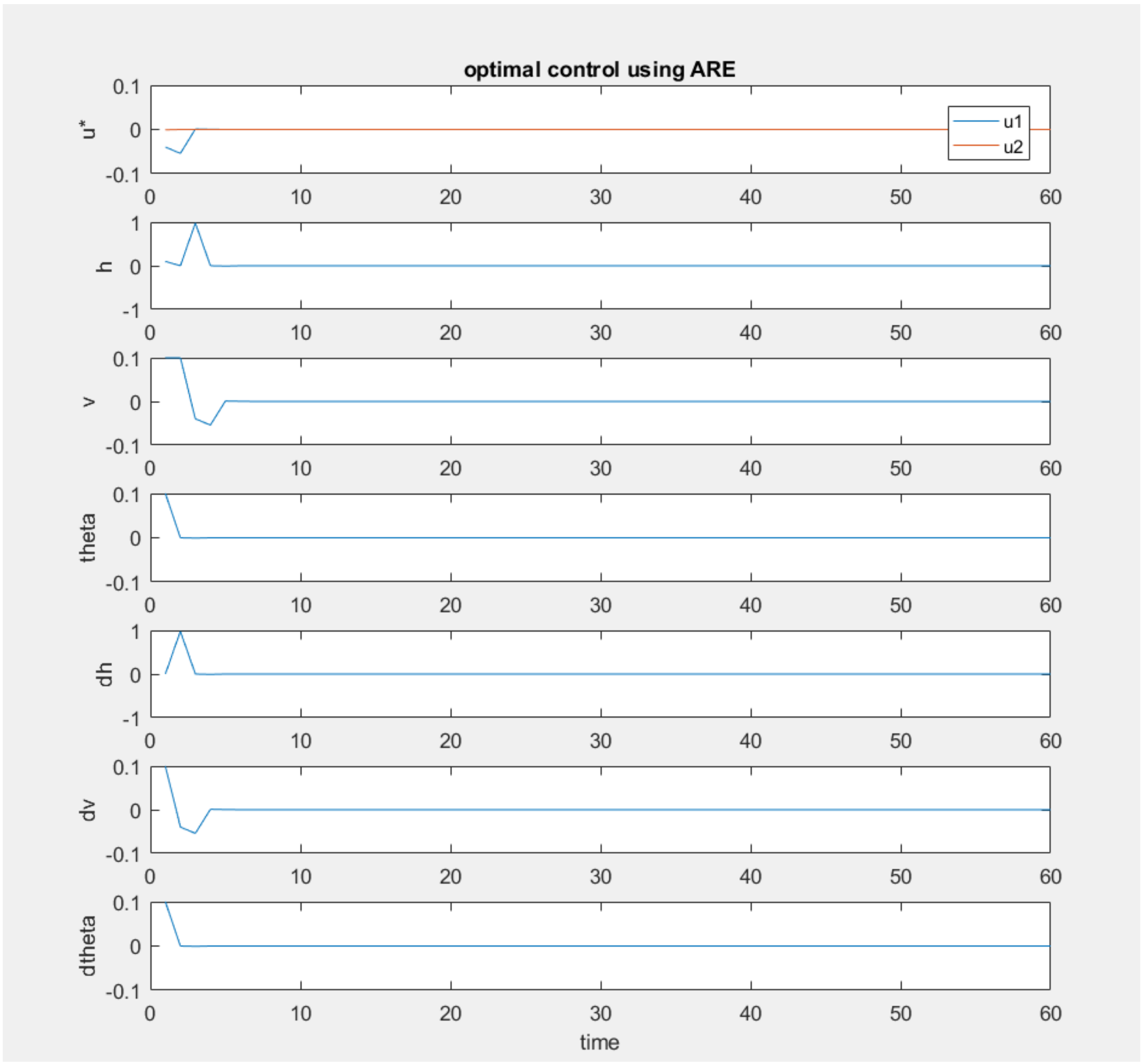
```
Matlab:function pd=pdef(size)
       % Generate a dense n x n symmetric, positive definite matrix
       pd = rand(size,size); % generate a random n x n matrix
       % construct a symmetric matrix using either
       pd = 0.5*(pd+pd');
       % or A = A*A';
       pd = pd + size*eye(size);
       pd;
```

3(a).2 Numerical Scheme to solve Algebraic Riccatti Equation :

```
Matlab:m=2; n=6; rN=N; %rN =75
       rQ = pdef(n);
       rR = pdef(m)
       rP = zeros(n,n,rN+1); % time goes from zero to N, so indices from 1 to N+1
       rP(:,:,rN+1) = rQ; % we start by setting the final P value as Q
       rK = zeros(m,n,rN+1);
       Nrm = zeros(rN,1);
       for i = rN:-1:1
       % algebraic riccatti recursion eqn for P
       rP(:,:,i) = rQ + A'*rP(:,:,i+1)*A -
       A'*rP(:,:,i+1)*B*pinv(rR+B'*rP(:,:,i+1)*B)*B'*rP(:,:,i+1)*A;
       % optimal K_t( time varying feedback) associated with each iteration
       rK(:,:,i) = -pinv(rR+B'*rP(:,:,i+1)*B)*B'*rP(:,:,i+1)*A;
```



3(b). Numerical Scheme to solve Algebraic Riccatti Equation :



We see a clear of difference in the controlling patterns and the evolution of states over the time domain when comparing ARE response and Closed loop response.

Part 4 . <u>Kalman Filter on closed loop linearised system:</u>

4(a). Discretizing the Closed loop linearized system :

Considering the continuous time model

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{w}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) + \mathbf{v}(t)$$

where v and w are continuous zero-mean white noise sources with covariances $\mathbf{w}(t) \sim N(0, \mathbf{Q})$, $\mathbf{v}(t) \sim N(0, \mathbf{R})$

can be discretized, assuming zero-order hold for the input *u* and continuous integration for the noise *v*, to

$$\mathbf{x}[k+1] = \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_d\mathbf{u}[k] + \mathbf{w}[k]$$
$$\mathbf{y}[k] = \mathbf{C}_d\mathbf{x}[k] + \mathbf{D}_d\mathbf{u}[k] + \mathbf{v}[k]$$

with covariances

$$\mathbf{w}[k] \sim N(0, \mathbf{Q}_d)$$
$$\mathbf{v}[k] \sim N(0, \mathbf{R}_d)$$

where

$$\mathbf{A}_d = e^{\mathbf{A}T} = \mathcal{L}^{-1}\{(s\mathbf{I} - \mathbf{A})^{-1}\}_{t=T}$$
$$\mathbf{B}_d = \left(\int_{\tau=0}^{T} e^{\mathbf{A}\tau} d\tau\right) \mathbf{B} = \mathbf{A}^{-1}(\mathbf{A}_d - I)\mathbf{B},$$
$$\mathbf{C}_d = \mathbf{C}$$
$$\mathbf{D}_d = \mathbf{D}$$
$$\mathbf{Q}_d = \int_{\tau=0}^{T} e^{\mathbf{A}\tau}\mathbf{Q}e^{\mathbf{A}^\top\tau} d\tau$$
$$\mathbf{R}_d = \frac{1}{T}\mathbf{R}$$

$$x_{t+1} = A_{\text{cl},d} x_t$$

Matlab:
```
D = zeros(6,2);
T = step; %(discretization time step)
sysc = ss(Acl,B,C,D);
sysd = c2d(sysc,T);
Ad = sysd.A;
Bd = sysd.B;
Cd = sysd.C;

dA = expm(Acl*T)
dB = inv(Acl)*(dA -eye(6))*B;
dC = C;
dD = zeros(6,2)
```

4(b). Generating noisy samples of the discrete time system :

$$\begin{aligned} x_{t+1} &= A_{\text{cl},d} x_t + F_t w_t \\ y_t &= C_d x_t + H_t v_t \end{aligned}$$

We have :

$\mathbb{E}[w_t w_t^T] = W$,

$\mathbb{E}[v_t v_t^T] = V$,

$\mathbb{E}[x_0] = \bar{x}_0$ and

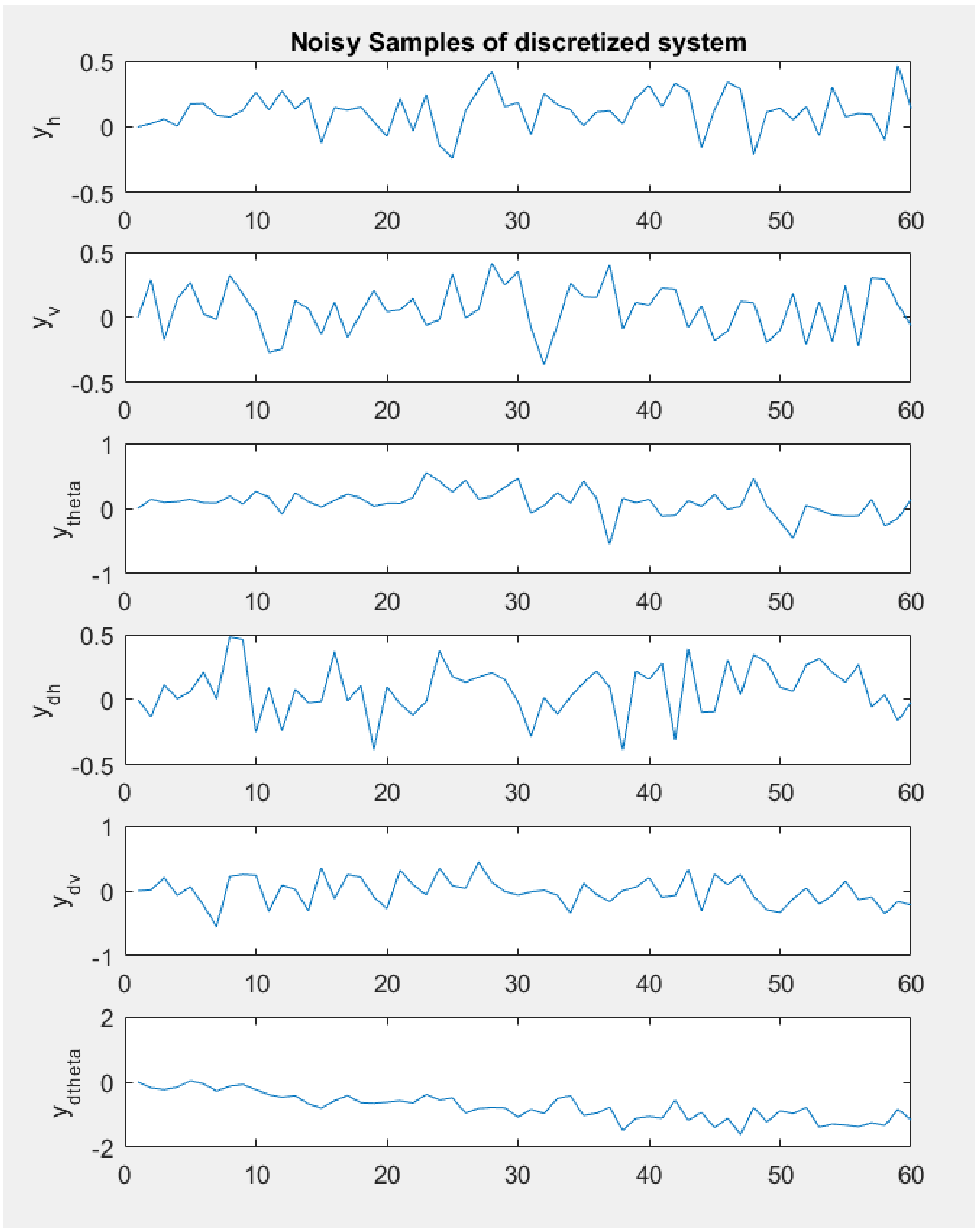
$\mathbb{E}[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T] = \Sigma_0$. Start with

$v_t \sim \mathcal{N}(0, 0.2 I_{2\times2})$ and $w_t \sim \mathcal{N}(0, 0.1 I_{2\times2})$, $\Sigma_0 = \text{diag}(0.1, 0.1, 0.1, 0.1, 0.1, 0.1)$.

Let F_t be the discretized control dynamics B_d and

$H_t = I$.

Matlab:
```
kx = zeros(6,N+1);
        kx(:,1) = [0.1 0.1 0.1 0 0.1 0]';
        for i = 2:N+1
            kw = normrnd(0,0.1,2,1);
            kv = normrnd(0,0.2,6,1);
        kx(:,i) = Ad*kx(:,i-1) + F*kw;
        kynoN(:,i) = dC*kx(:,i);
        ky(:,i) = dC*kx(:,i)+ H*kv;
        end

        plot(ky(1,1:60))
        hold on;
        figure(12)
        subplot(6,1,1)
        plot(ky(1,1:60))
        ylabel('y_h')
        hold on;
        title('Noisy Samples of discretized system')
        subplot(6,1,2)
        plot(ky(2,1:60))
        ylabel('y_v')
        hold on;
        subplot(6,1,3)
        plot(ky(3,1:60))
        ylabel('y_t_h_e_t_a')
```


Noisy Samples of discretized system
y_h
y_v
y_{theta}
y_{dh}
y_{dv}
y_{dtheta}
0
10
20
30
40
50
60

4(c). Implementing the discrete time Kalman Filter:

Model and Observation:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \\ \mathbf{z}_k &= \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \end{aligned}$$

Initialization:

$$\mathbf{x}_0^a = \mu_0 \text{ with error covariance } \mathbf{P}_0$$

Model Forecast Step/Predictor:

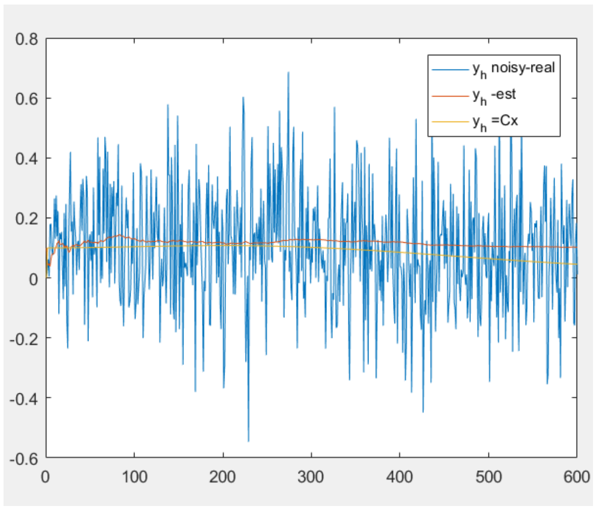
$$\begin{aligned} \mathbf{x}_k^f &= \mathbf{A}_{k-1}\mathbf{x}_{k-1}^a + \mathbf{B}_{k-1}\mathbf{u}_{k-1} \\ \mathbf{P}_k^f &= \mathbf{A}_{k-1}\mathbf{P}_{k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1} \end{aligned}$$

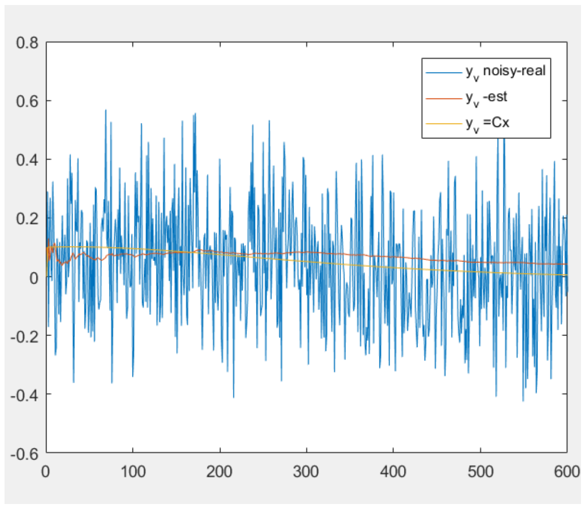
Data Assimilation Step/Corrector:

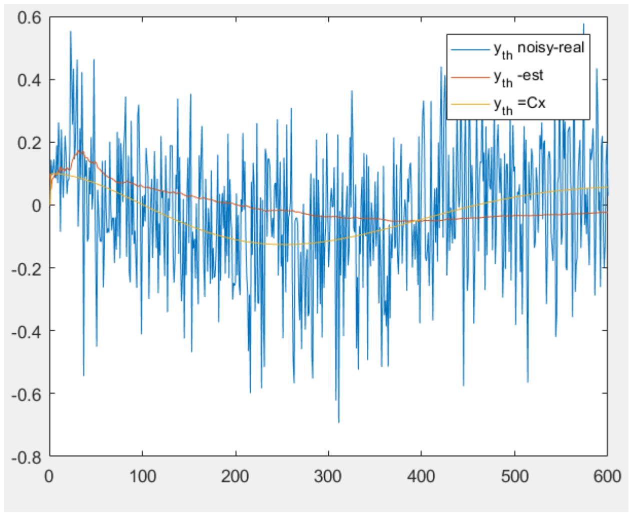
$$\begin{aligned} \mathbf{x}_k^a &= \mathbf{x}_k^f + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\mathbf{x}_k^f) \\ \mathbf{K}_k &= \mathbf{P}_k^f\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k^f\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \\ \mathbf{P}_k &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^f \end{aligned}$$

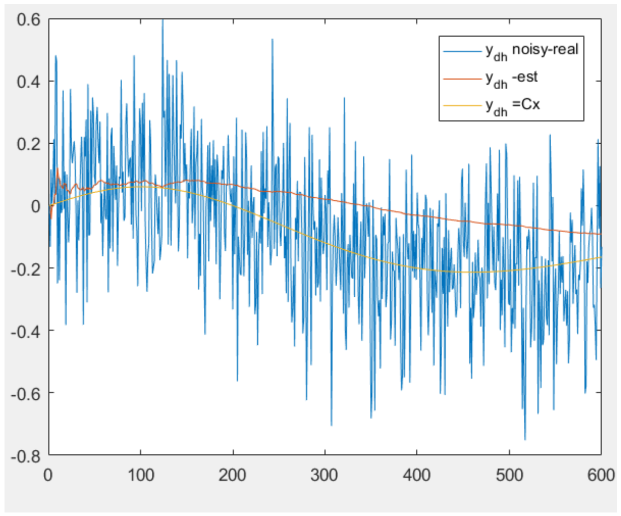
Matlab:
```
Q = [1      0       0       0       0       0
     0      1       0       0       0       0
     0      0       1       0       0       0
     0      0       0       1       0       0
     0      0       0       0       1       0
     0      0       0       0       0       1];  % process noise w

R = 0.1*eye(6); % measurement noise v
K = [];


xest_b(:,1) =kx0; %before
xest_c(:,1)= kx0; %current

Pest_b(:,:,1)  = ksigma;
Pest_c(:,:,1)  = ksigma;

for n=1:rN

xest_b(:,n+1)  = Ad*xest_c(:,n);%+dB*u; %estimate with data before current
Pest_b(:,:,n+1)  = Ad*Pest_c(:,:,n)*Ad' + Q; % estimate with data before current

K(:,:,n)  = Pest_c(:,:,n)*(dC'*inv(dC*Pest_c(:,:,n)*dC' + R));

xest_c(:,n+1)  = xest_c(:,n)+(K(:,:,n)*(ky(:,n)-dC*xest_c(:,n)));
Pest_c(:,:,n+1)  =  (eye(6)- K(:,:,n)*C)*Pest_c(:,:,n);
yest(:,n)  = dC*xest_c(:,n);

end
```

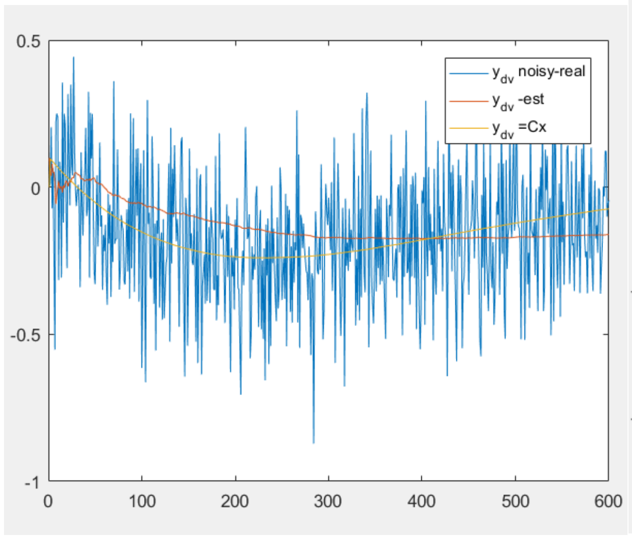
4(d). Applying the discrete time Kalman Filter to the system:

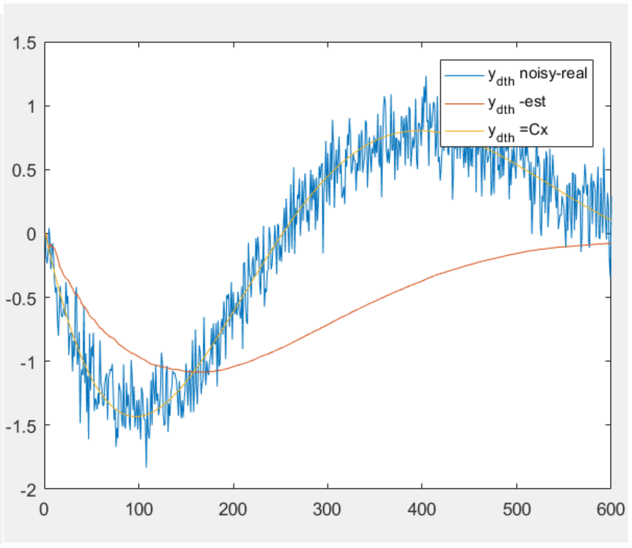




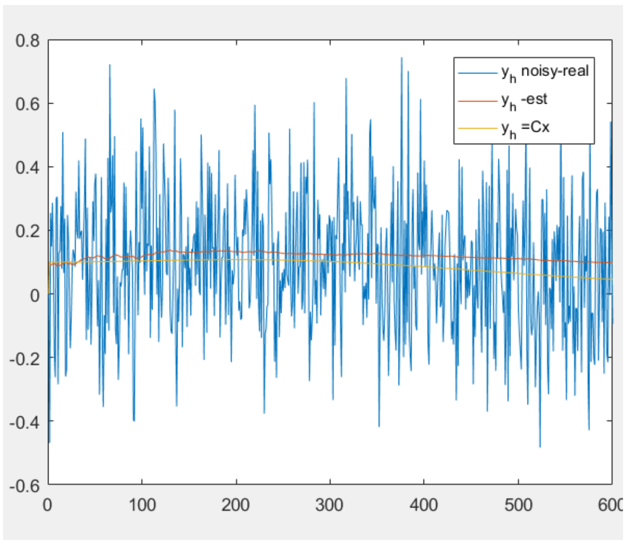


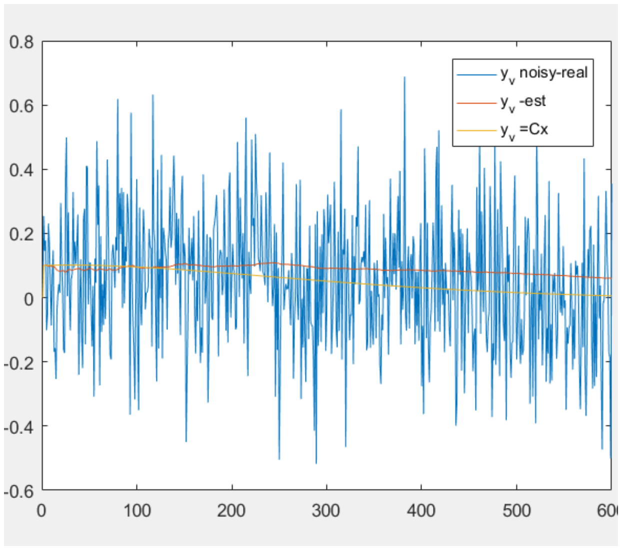


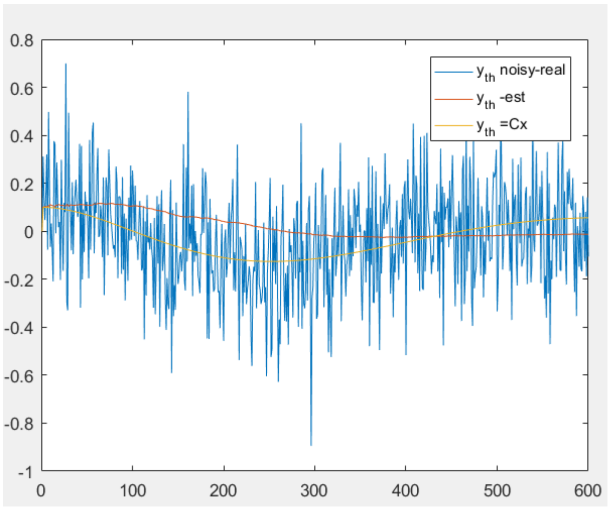


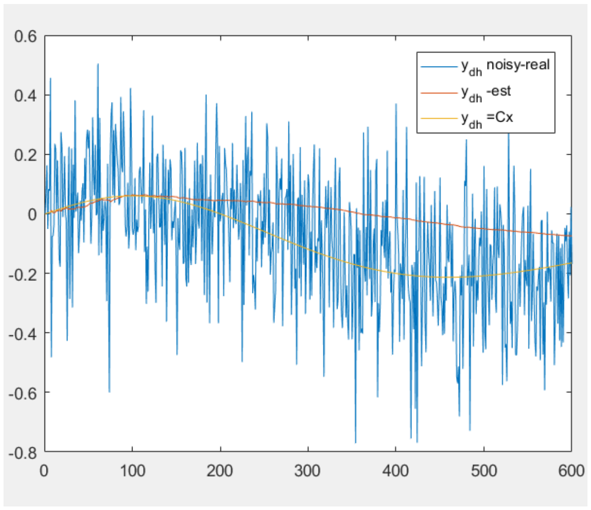


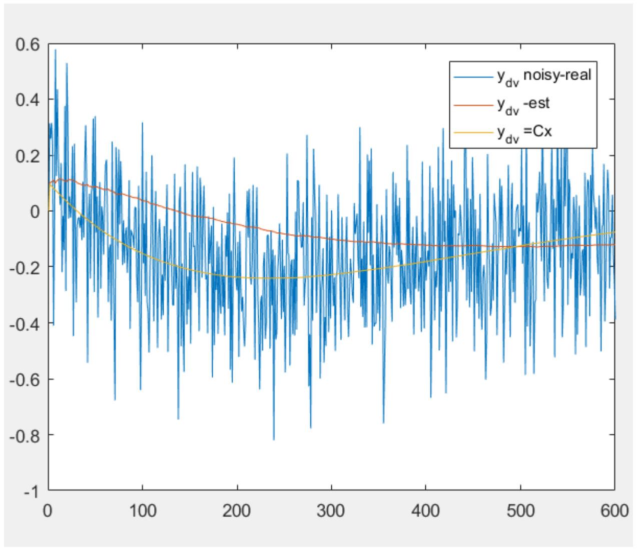
Plots for a randomly generated Q and R positive definite matrices:

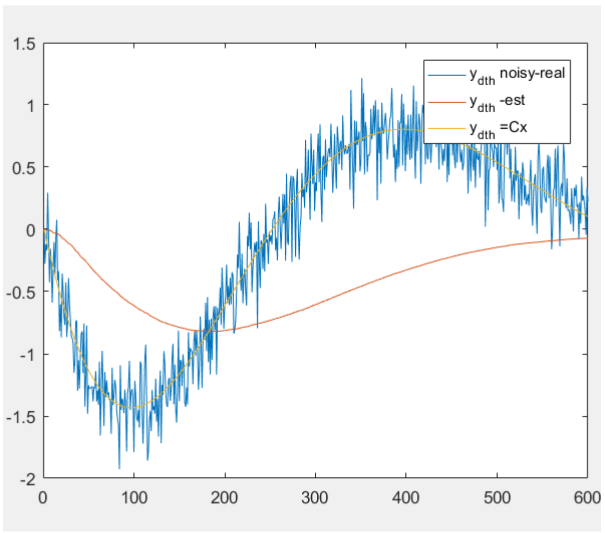












Comparing the performance:

On comparing the plots for randomly generated Q and R with the standard Q and R, the discrete time KF gives a decent performance for both the cases making a good estimate of the states from the noisy samples.

Bibliography :

-Dynamical systems theory: Nonlinear dynamics and chaosby Strogatz

-Sniedovich, M. (2010), Dynamic Programming: Foundations and Principles, Taylor & Francis, ISBN 978-0-8247-4099-3

-Stuart Dreyfus. "Richard Bellman on the birth of Dynamical Programming"

-O Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001), Introduction to Algorithms (2nd ed.), MIT Press & McGraw–Hill, ISBN 0-262-03293-7 . pp. 344.

Full Matlab Code :

```
clc, clear all, close all

g=9.81;m=1;I=1;

 x0 = [0;0;0;0;0;0]';
% x0 = [0.1 0.1 0.1 0 0.1 0]';
Qxcl(:,1) = x0;
N=10000; %number of time steps
t=linspace(0,10, N+1); %taking the time step
step = t(2)-t(1);
w=5;

Qx = zeros(6, N+1); %initial condition is set at x[:,0] to [0;0;0;0;0;0]
for i =1:N+1
   u(:,i)= [(m*g + sin(2*t(i)*pi*w));0];
end
% Simulation forward euler:
for i=2:N+1
    % defining the U vector
    Qx(:,i) = step*(fun(i,Qx(:,i-1),[(m*g + sin(2*t(i)*pi*w));0]))+ Qx(:,i-1);
end

% % ode 45 verification
% [tq,q]=ode45(@(tq,q)fun(tq,q),t,Qx(:,1));
% q=q';

% plot(q(2,:))
% hold on;
% plot(q(5,:))

% figure(1)
% hold on
% plot(q(1,:))
% hold on;
% plot(q(2,:))
% hold on;
% plot(q(5,:))

figure(2)

subplot(3,1,1)
plot(Qx(1,1:5000))
ylabel('h')
hold on;
title('Forward Euler Simulation with w=5')
subplot(3,1,2)
plot(Qx(2,1:5000))
hold on;
ylabel('v')
subplot(3,1,3)
plot(Qx(3,1:5000))
ylabel('theta')
xlabel('time steps');

figure(3)
subplot(3,1,1)
plot(Qx(4,1:5000))
ylabel('dh')
hold on;
subplot(3,1,2)
plot(Qx(5,1:5000))
hold on;
ylabel('dv')
subplot(3,1,3)
plot(Qx(6,1:5000))
ylabel('dtheta')
xlabel('time steps');

figure(4)
subplot(2,1,1)
plot(u(1,1:5000))
hold on;
ylabel('u1')
subplot(2,1,2)
plot(u(2,1:5000))
ylabel('u2')
xlabel('time steps');


%% Part 2 : Stabilisation:

%Linearised Matrices

A = [0 0 0 1 0 0;
    0 0 0 0 1 0 ;
    0 0 0 0 0 1;
    0 0 9.8 0 0 0;
    0 0 0 0 0 0;
    0 0 0 0 0 0]

B = [0 0;
    0 0;
    0 0;
    0 0;
    1 0;
    0 1;]

CTRB = [B A*B A^2*B A^3*B A^4*B A^5*B]

C = [1,0,0,0,0,0;
    0,1,0,0,0,0;
    0,0,1,0,0,0;
    0,0,0,1,0,0;
    0,0,0,0,1,0;
    0,0,0,0,0,1]



            % Ans : CTRB =
            %
            %       [  0   0   0   0   0   0   0  9.8  0   0   0   0
            %          0   0   1   0   0   0   0   0   0   0   0   0
            %          0   0   0   1   0   0   0   0   0   0   0   0
            %          0   0   0   0   0  9.8  0   0   0   0   0   0
            %          1   0   0   0   0   0   0   0   0   0   0   0
            %          0   1   0   0   0   0   0   0   0   0   0   0  ]


rank(CTRB)
% Ans : 6; verified with rank(ctrb(A,B))

% thus Rank = 6;

% ---> 2(c)(1)
D=eig(A);
lambda = rand(10);
lambda= lambda(1,:);
for k=1:length(lambda)
rtest(k)=rank(ctrb(-lambda(k)*eye(6) - A,B));
end
% satisfies the controllability condition

% ---> 2(c)(2)

p = [-3.5; -3.6; -3.7; -3.8; -4.5; -4.9]; % be the mu values (the shift in poles)
K0 = place(A,B,p); % K0 from matlab solver


%(-muI-A)W + W(-muI-A)' + BB' =  lyapunov
        lA = (diag(p) - A);
        lQ = B*B';
        W = lyap(lA,lQ);

% from the calculation from 2(C)(3) we know that inv(P) = W; sp P = inv(W)

        lP = inv(W);

        K0= 0.5*B'*lP ;

%checking controllability
        eig(A-B*K0)
                % ans = yes controllable!
                %
                %   -4.0094 + 8.9135i
                %   -4.0094 - 8.9135i
                %   -3.9406 + 0.8051i
                %   -3.9406 - 0.8051i
                %   -4.0500 + 3.5718i
                %   -4.0500 - 3.5718i

% closed Loop A
Acl = A-B*K0;

%% -----> 2(d) Simulation of closed loop dynamics from a small perturbation
Qxlcl = zeros(6, N+1);
Qxcl = zeros(6, N+1); %initial condition is set at x[:,0] to [0;0;0;0;0;0]
Qxcl(:,1) = [0.1 0.1 0.1 0 0.1 0]';
% Qxcl(:,1) = [0.2 0.4 0.3 0.1 0.2 0.1]';
Qxlcl(:,1) = Qxcl(:,1);
unl = zeros(2, N+1);
% Simulation forward euler:
for i=2:N+1
    % defining the U vector
    unl(:,i) = -K0*Qxcl(:,i-1)+ [9.8;0];
     Qxcl(:,i) = step*(fun(i,Qx(:,i-1),unl(:,i)))+ Qx(:,i-1);
 Qxlcl(:,i) = step*(clloop(t(i),Qxlcl(:,i-1)))+ Qxlcl(:,i-1);
end

% ode 45 verification
% [tq,qcl]=ode45(@(tq,qcl)clloop(tq,qcl),t,Qxcl(:,1));
% qcl=qcl';
%
% plot(qcl(2,:))
% hold on;
% plot(qcl(5,:))

figure(3)
hold on
plot(Qxlcl(1,1:2000))
hold on;
plot(Qxlcl(2,1:2000))
hold on;
plot(Qxlcl(3,1:2000))
hold on;
plot(Qxlcl(4,1:2000))
hold on;
plot(Qxlcl(5,1:2000))
hold on;
plot(Qxlcl(6,1:2000))
title('closed loop response of linearised system to small perturbation');
xlabel('time step')
legend ('h','v','th','dh','dv','dth')

% figure(4)
%
subplot(3,1,1)
plot(Qx(1,1:5000))
ylabel('h')
hold on;
title('closed loop response of non linear system to small perturbation')
subplot(3,1,2)
plot(Qx(2,1:5000))
hold on;
ylabel('v')
subplot(3,1,3)
plot(Qx(3,1:5000))
ylabel('theta')
xlabel('time steps');

figure(5)
```

```
subplot(3,1,1)
plot(Qx(4,1:5000))
ylabel('dh')
hold on;
subplot(3,1,2)
plot(Qx(5,1:5000))
hold on;
ylabel('dv')
subplot(3,1,3)
plot(Qx(6,1:5000))
ylabel('dtheta')
xlabel('time steps');

figure(6)
subplot(2,1,1)
plot(u(1,1:5000))
hold on;
ylabel('u1')
subplot(2,1,2)
plot(u(2,1:5000))
ylabel('u2')
xlabel('time steps');
%% 3 (b) -> optimal control usin ARE

m=2; n=6; rN=N; %rN =75

rQ = pdef(n);
rR = pdef(m)


rP = zeros(n,n,rN+1); % time goes from zero to N, so indices from 1 to N+1
rP(:,:,rN+1) = rQ; % we start by setting the final P value as Q
rK = zeros(m,n,rN+1);
Nrm = zeros(rN,1);
for i = rN:-1:1

% algebraic riccatti recursion eqn for P
rP(:,:,i) = rQ + A'*rP(:,:,i+1)*A -
A'*rP(:,:,i+1)*B*pinv(rR+B'*rP(:,:,i+1)*B)*B'*rP(:,:,i+1)*A;

% optimal K_t( time varying feedback) associated with each iteration
rK(:,:,i) = -pinv(rR+B'*rP(:,:,i+1)*B)*B'*rP(:,:,i+1)*A;

%norm gives a scalar value for the P matrix. (2norm is used here to show
%convergence
Nrm(rN+1-i) = norm(rP(:,:,i+1)-rP(:,:,i))/norm(rP(:,:,i+1));
end

%showing convergence of P
figure(9)
plot (Nrm(1:15))
ylabel('norm(p_i_+_1 - p_i) / norm(p_i_+_1)')
xlabel('iterations')


for i=1:length(rP)
    Z(i) = norm(rP(:,:,i));
end

%%  optimal time varying
rKb = reshape(permute(rK, [2 1 3]), size(rK, 2), [])'
rk1 = zeros(76,6);
rk2 = zeros(76,6);
```

```
s=0;t=0;
for i=1:length(rKb)
    if(mod(i,2)==0)
        s=s+1;
        rk2(s,:)=rKb(i,:);
    else
        t=t+1;
        rk1(t,:)=rKb(i,:);
    end
end



figure(7)
subplot(2,1,1)
plot(rk1(:,1))
ylabel('K1')
hold on;

subplot(2,1,2)
plot(rk2(:,1))
ylabel('K2')
hold on;


%% finding initial condition x0, with norm not exceeding one, that maximises value of
J


% [v,d] = eig(rP(:,:,1))

%maximizing a quadratic form subject to ?x0?=1?x0?=1 constraints for a symmetric
matrix corresponds to a special matrix norm operator whose value is the maximum
eigenvalue and the maximizer is the associated eigenvector.
%
% rx0 = v(:,1);

%norm_x0 = norm(x0)
%
% creating x vector

rx0 = [0.1 0.1 0.1 0 0.1 0]'  ;
rx = zeros(n,rN+1);
rx(:,1) = rx0;

ru = zeros(m,rN+1);
ru(:,1) = [rk1(1,:);rk2(1,:)] * rx(:,1);

% %% 2( Simulating system from x0 with u0 (normal state space simulation)
%
for (i=1:rN)
    rx(:,i+1) = A*rx(:,i) + B*ru(:,i);
%     u(:,i+1) = Kb(:,i+1)'*x(:,i+1); % u* calculated from Kb that we calculated
earlier
    ru(:,i+1) = [rk1(i+1,:);rk2(i+1,:)] * rx(:,i+1);
end

figure(3)
subplot(7,1,1)
plot(ru(1,1:60))
hold on;
title('optimal control using ARE')
plot(ru(2,1:60))
```

```
legend('u1','u2');
ylabel('u*')
hold on;

subplot(7,1,2)
plot(rx(1,1:60))
ylabel('h')
hold on;

subplot(7,1,3)
plot(rx(2,1:60))
ylabel('v')
hold on;

subplot(7,1,4)
plot(rx(3,1:60))
ylabel('theta')
hold on;

subplot(7,1,5)
plot(rx(4,1:60))
ylabel('dh')
hold on;

subplot(7,1,6)
plot(rx(5,1:60))
ylabel('dv')
hold on;

subplot(7,1,7)
plot(rx(3,1:60))
ylabel('dtheta')
xlabel('time')
hold on;

%
%% on nonlinear quadcopter - check if required

 % rn number of time steps
% t=linspace(0,100, rN+1); %taking the time step
% step = t(2)-t(1);
w=1;


ru1 = ru+[(1*g ) 0;0 0]*ones(2,rN+1);


Qx = zeros(6, rN+1); %initial condition is set at x[:,0] to [0;0;0;0;0;0]
Qx(:,1)=[0.1 0.1 0.1 0 0.1 0]';
% Simulation forward euler:
for i=2:rN+1

    % defining the U vector

     Qx(:,i) = step*(fun(i,Qx(:,i-1),ru1))+ Qx(:,i-1);
end

figure(9)
hold on
plot(Qx(1,1:80))
hold on;
plot(Qx(2,1:80))
hold on;
```

```
plot(Qx(5,1:80))


%% Part 4 Kalman Filter on closed loop system
%N=100;

%----> 4(a) Discretisation

D = zeros(6,2);
T = step; %(discretization time step)
sysc = ss(Acl,B,C,D);
sysd = c2d(sysc,T);
Ad = sysd.A;
Bd = sysd.B;
Cd = sysd.C;

dA = expm(Acl*T)
dB = inv(Acl)*(dA -eye(6))*B;
dC = C;
dD = zeros(6,2)


%----> 4(b) generating noisy samples of discrete system

ksigma = .1*eye(6);
% kw = normrnd(0,0.1,N+1,1);
% kv = normrnd(0,0.2,N+1,1);

% kw1 = zeros(2,N+1);
% kv1 = zeros(6,N+1);

H = eye(6);
F = dB;
% for i=1:N+1
%     kw1(:,i) = kw(i)*[1;1];
%     kv1(:,i) = kv(i)*[1;1;1;1;1;1];
% end

%Nonlinear Simulation
kx0 = [0.1 0.1 0.1 0 0.1 0]';


kx_real = zeros(6,N+1);
kx_real(:,1) = kx0;
for i = 2:N

kx_real(:,i) = Ad*kx_real(:,i-1) ;
end


figure(11)

plot(kx_real(1,1:60))
hold on;
% plot(kx_real(2,1:60))
% hold on;
% plot(kx_real(3,1:60))
% hold on;
% plot(kx_real(4,1:60))
% hold on;
% plot(kx_real(5,1:60))
% hold on;
% plot(kx_real(6,1:60))
% ylabel('x6')
% xlabel('time')

%% simulating noisy observation

kx = zeros(6,N+1);
kx(:,1) = [0.1 0.1 0.1 0 0.1 0]';
for i = 2:N+1
    kw = normrnd(0,0.1,2,1);
    kv = normrnd(0,0.2,6,1);
kx(:,i) = Ad*kx(:,i-1) + F*kw;
kynoN(:,i) = dC*kx(:,i);
ky(:,i) = dC*kx(:,i)+ H*kv;
end
%


plot(ky(1,1:60))
hold on;
figure(12)
subplot(6,1,1)
plot(ky(1,1:60))
ylabel('y_h')
hold on;
title('Noisy Samples of discretized system')
subplot(6,1,2)
plot(ky(2,1:60))
ylabel('y_v')
hold on;
subplot(6,1,3)
plot(ky(3,1:60))
ylabel('y_t_h_e_t_a')
hold on;
subplot(6,1,4)
plot(ky(4,1:60))
ylabel('y_d_h')
hold on;
subplot(6,1,5)
plot(ky(5,1:60))
ylabel('y_d_v')
hold on;
subplot(6,1,6)
plot(ky(6,1:60))
ylabel('y_d_t_h_e_t_a')

%% % kalman filter

Q = [1     0     0     0     0     0
     0     1     0     0     0     0
     0     0     1     0     0     0
     0     0     0     1     0     0
     0     0     0     0     1     0
     0     0     0     0     0     1];  % process noise w

R = 0.1*eye(6); % measurement noise v
K = [];


xest_b(:,1) =kx0; %before
xest_c(:,1)= kx0; %current

Pest_b(:,:,1) = ksigma;
Pest_c(:,:,1) = ksigma;
```

```
for n=1:rN

xest_b(:,n+1) = Ad*xest_c(:,n);%+dB*u; %estimate with data before current
Pest_b(:,:,n+1) = Ad*Pest_c(:,:,n)*Ad' + Q; % estimate with data before current

K(:,:,n) = Pest_c(:,:,n)*(dC'*inv(dC*Pest_c(:,:,n)*dC' + R));

xest_c(:,n+1) = xest_c(:,n)+(K(:,:,n)*(ky(:,n)-dC*xest_c(:,n)));
Pest_c(:,:,n+1) = (eye(6)- K(:,:,n)*C)*Pest_c(:,:,n);
yest(:,n) = dC*xest_c(:,n);

end
%
%
%  plot(1:100,ky(1,:),1:100,yest(1,:))
%    legend('true measurement','KF estimated measurement'), axis([0 100 -0.5
0.5]),xlabel('time'),ylabel('Position observations')
%

figure(20)

plot(ky(1,1:600));
hold on
plot (yest(1,1:600))
hold on
plot(kynoN(1,1:600))
legend('y_h noisy-real','y_h -est','y_h =Cx')

figure(21)
plot(ky(2,1:600));
hold on
plot (yest(2,1:600))
hold on
plot(kynoN(2,1:600))
legend('y_v noisy-real','y_v -est','y_v =Cx')

figure(22)
plot(ky(3,1:600));
hold on
plot (yest(3,1:600))
hold on
plot(kynoN(3,1:600))
legend('y_t_h noisy-real','y_t_h -est','y_t_h =Cx')

figure(23)
plot(ky(4,1:600));
hold on
plot (yest(4,1:600))
hold on
plot(kynoN(4,1:600))
legend('y_d_h noisy-real','y_d_h -est','y_d_h =Cx')


figure(24)
plot(ky(5,1:600));
hold on
plot (yest(5,1:600))
hold on
plot(kynoN(5,1:600))
legend('y_d_v noisy-real','y_d_v -est','y_d_v =Cx')

figure(25)
```

```
plot(ky(6,1:600));
hold on
plot (yest(6,1:600))

plot(kynoN(6,1:600))
legend('y_d_t_h noisy-real','y_d_t_h -est','y_d_t_h =Cx')

%% using psd to generate new kalman estimates

Q = pdef(6);
R = pdef(6);
K = [];


xest_b(:,1) =kx0; %before
xest_c(:,1)= kx0; %current

Pest_b(:,:,1) = ksigma;
Pest_c(:,:,1) = ksigma;

for n=1:rN

xest_b(:,n+1) = Ad*xest_c(:,n);%+dB*u; %estimate with data before current
Pest_b(:,:,n+1) = Ad*Pest_c(:,:,n)*Ad' + Q; % estimate with data before current

K(:,:,n) = Pest_c(:,:,n)*(dC'*inv(dC*Pest_c(:,:,n)*dC' + R));

xest_c(:,n+1) = xest_c(:,n)+(K(:,:,n)*(ky(:,n)-dC*xest_c(:,n)));
Pest_c(:,:,n+1) = (eye(6)- K(:,:,n)*C)*Pest_c(:,:,n);
yest(:,n) = dC*xest_c(:,n);

end

figure(26)

plot(ky(1,1:600));
hold on
plot (yest(1,1:600))
hold on
plot(kynoN(1,1:600))
legend('y_h noisy-real','y_h -est','y_h =Cx')

figure(27)
plot(ky(2,1:600));
hold on
plot (yest(2,1:600))
hold on
plot(kynoN(2,1:600))
legend('y_v noisy-real','y_v -est','y_v =Cx')

figure(28)
plot(ky(3,1:600));
hold on
plot (yest(3,1:600))
hold on
plot(kynoN(3,1:600))
legend('y_t_h noisy-real','y_t_h -est','y_t_h =Cx')

figure(29)
plot(ky(4,1:600));
hold on
plot (yest(4,1:600))
hold on
```

```
plot(kynoN(4,1:600))
legend('y_d_h noisy-real','y_d_h -est','y_d_h =Cx')


figure(30)
plot(ky(5,1:600));
hold on
plot (yest(5,1:600))
hold on
plot(kynoN(5,1:600))
legend('y_d_v noisy-real','y_d_v -est','y_d_v =Cx')

figure(31)
plot(ky(6,1:600));
hold on
plot (yest(6,1:600))
hold on
plot(kynoN(6,1:600))
legend('y_d_t_h noisy-real','y_d_t_h -est','y_d_t_h =Cx')
```