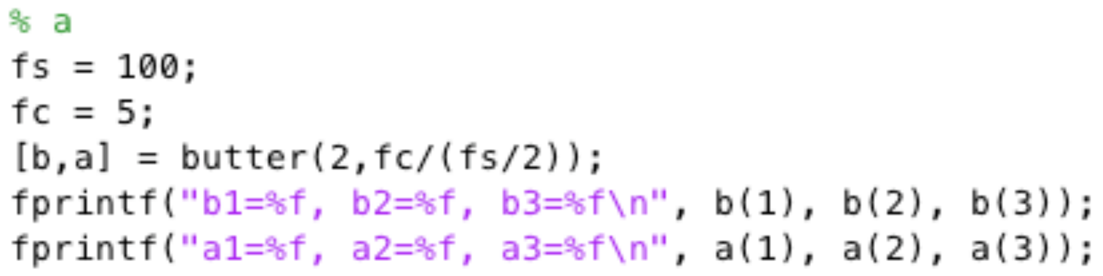
CSCI 545 Homework 3

Problem 1

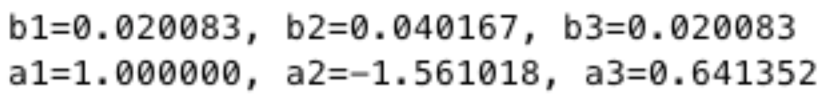
(a)

Code:



```
% a
fs = 100;
fc = 5;
[b,a] = butter(2,fc/(fs/2));
fprintf("b1=%f, b2=%f, b3=%f\n", b(1), b(2), b(3));
fprintf("a1=%f, a2=%f, a3=%f\n", a(1), a(2), a(3));
```

Result:



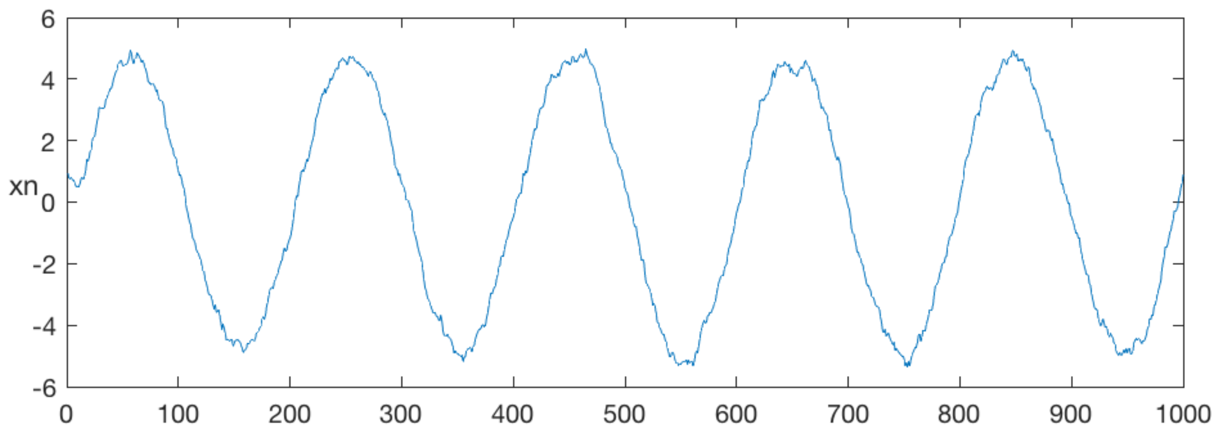
```
b1=0.020083, b2=0.040167, b3=0.020083
a1=1.000000, a2=-1.561018, a3=0.641352
```

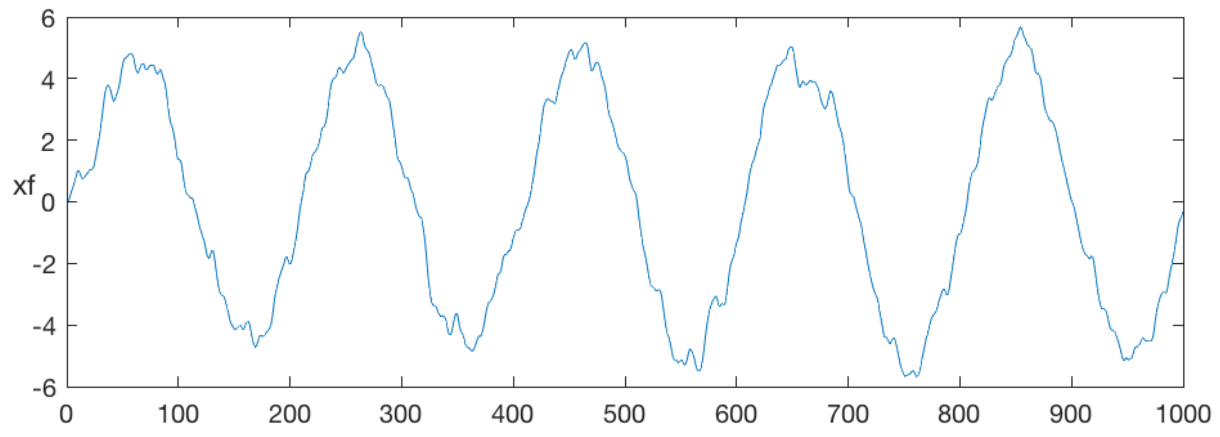
(b)
Code:

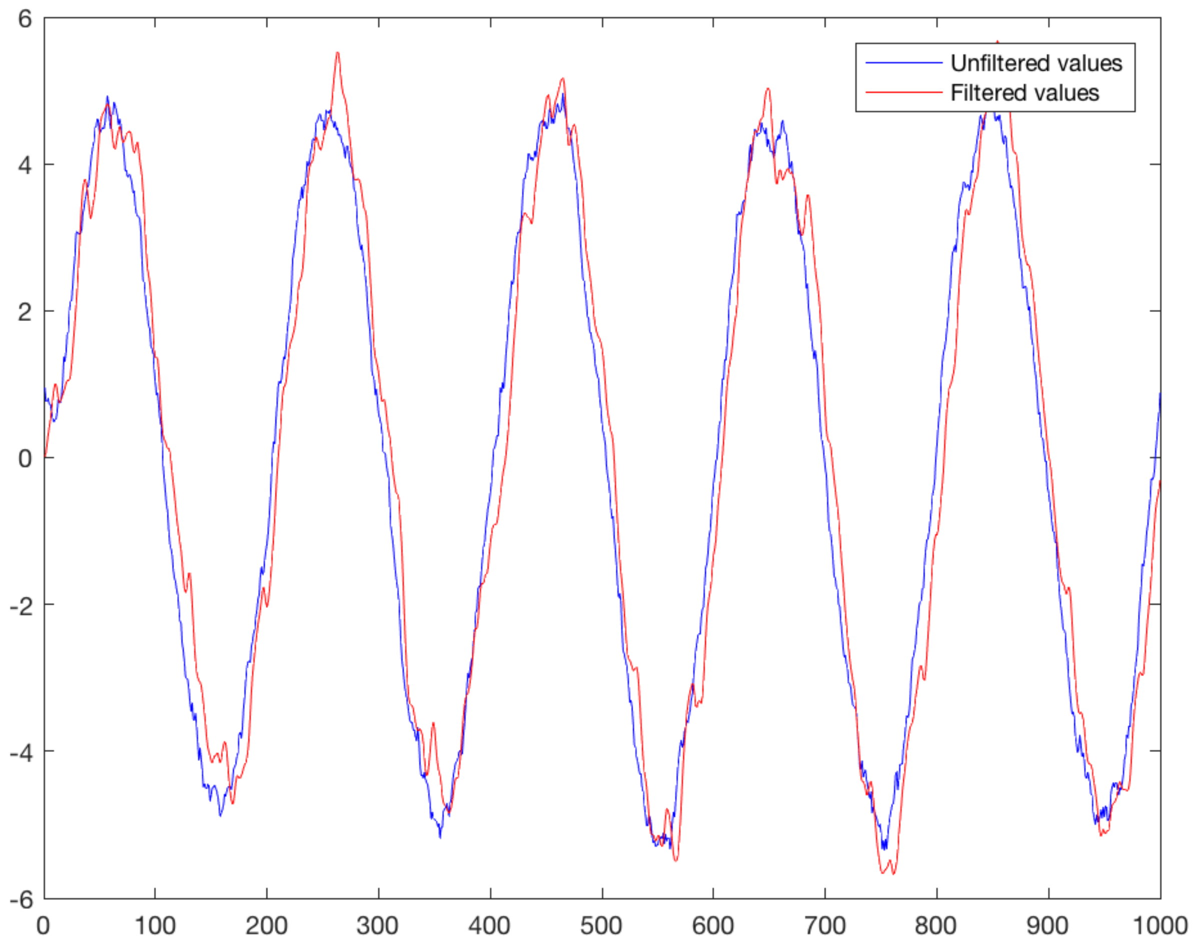
```
% b
filename = 'noisy.data';
delimiter=' ';
data=importdata(filename,delimiter);
yn=data(:,1);
xn=data(:,2);
un=data(:,3);
xf = filter(b, a, yn);
figure(1);
subplot(2,1,1)
plot(xn);
ylabel('xn','rotation',0);
subplot(2,1,2)
plot(xf);
ylabel('xf','rotation',0);
figure(2);
p1 = plot(xn,'b');
hold on;
p2 = plot(xf, 'r');
legend([p1,p2], 'Unfiltered values','Filtered values');
hold off;

disp("delay of Filtered values and Unfiltered values is: " + finddelay(xn,xf));
```

Notes: b and a are results in question (a) which are calculated by butter function.

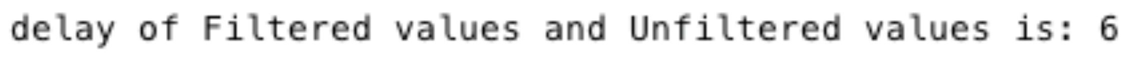
Result:


xn
6
4
2
0
-2
-4
-6
0
100
200
300
400
500
600
700
800
900
1000


xf
6
4
2
0
-2
-4
-6
0
100
200
300
400
500
600
700
800
900
1000



According to the image above, we can see the second order Butterworth filter works well. The blue line is the filtered values and the red line is the unfiltered values. It filters out most noisy signals. But it has some delay to the filtered data.



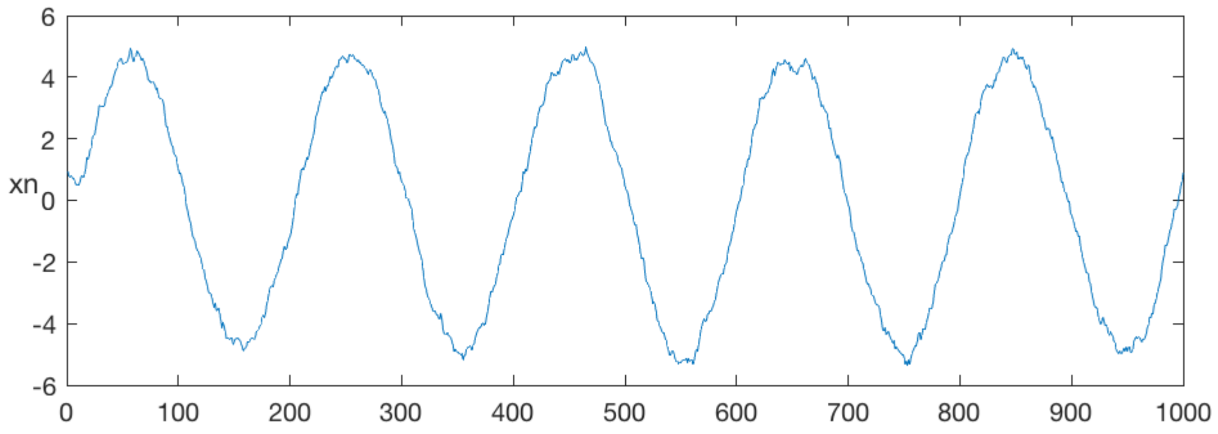
```
delay of Filtered values and Unfiltered values is: 6
```

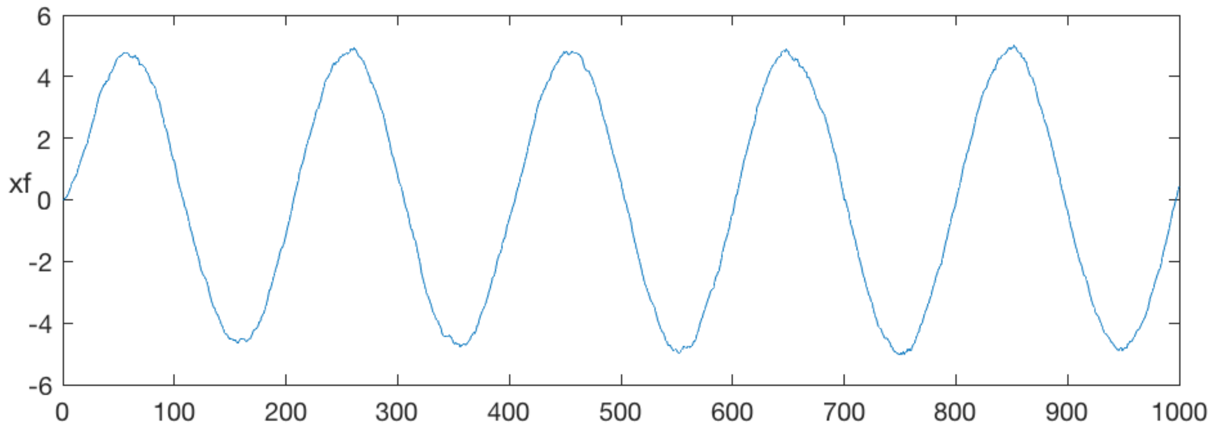
According to the code picture, the delay between filtered and unfiltered values are calculated by ‘finddelay’ function. The result is 6ms.

(c)
Code:

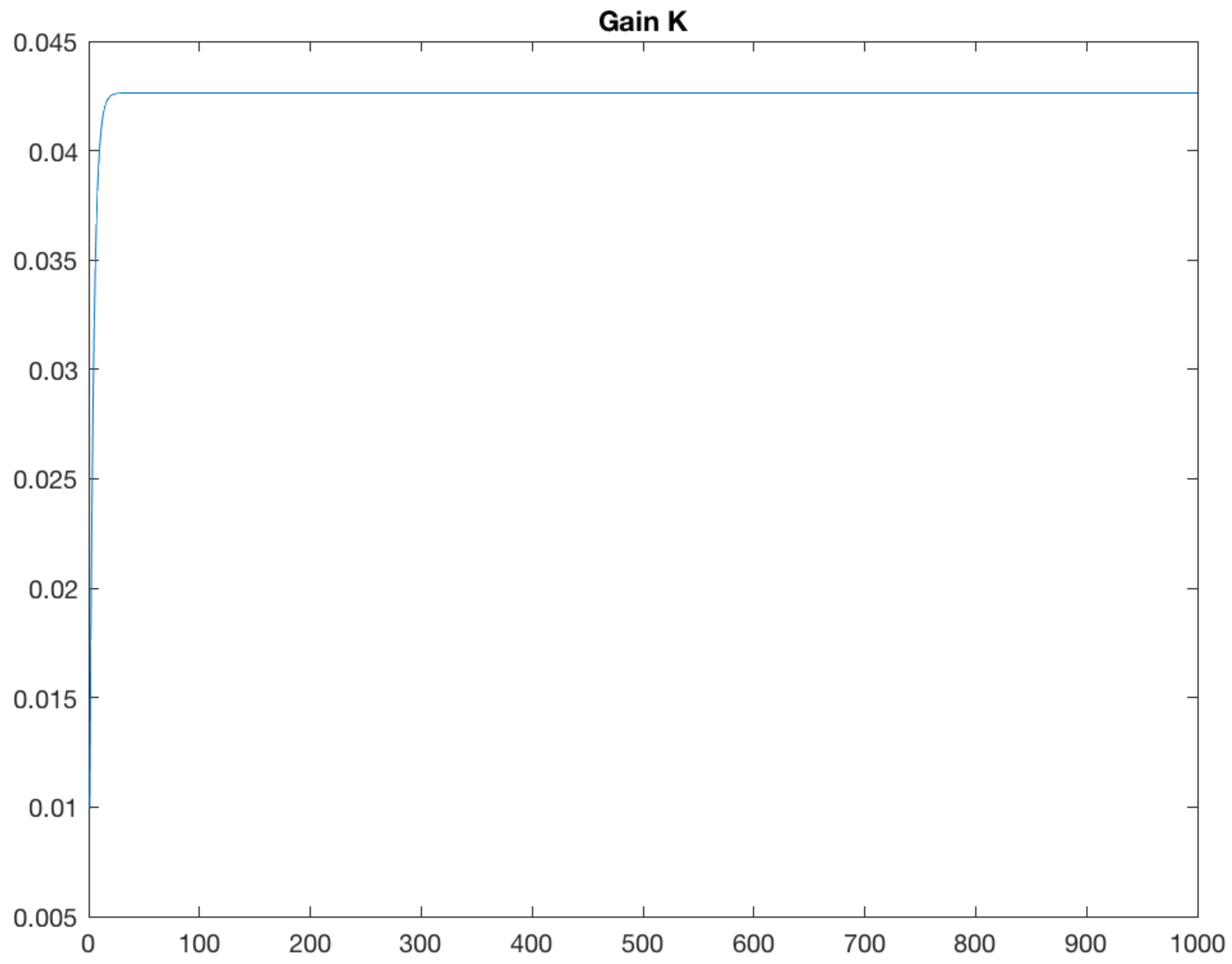
```matlab
% c
length = 1000;
A = 0.9;
B = 0.5;
C = 1;
R = 1; % the covariance of the observation noise;
Q = 0.01;% the covariance of the process noise;
P = zeros(1, length);
K = zeros(1, length);
x_post = zeros(1, length);
x_pri = 0;
p_pri = Q; % Assume P(0) = 1;
for t = 1:1000
    K(t) = p_pri * C' / ( C * p_pri * C' + R ); % Kn = pPriori * C' / (C * pPriori * C' + R);
    x_post(t) = x_pri + K(t) * ( yn(t) - C * x_pri ); % xPosteriori = xPriori + K * (yn - C * xPriori);
    P(t) = ( 1 - K(t) * C ) * p_pri; % Pn = (1 - K(n) * C) * pPriori;
    x_pri = A * x_post(t) + B * un(t); % xPriori = A * xPosteriori_n + B * un;
    p_pri = A * P(t) * A' + Q; %pPriori = A * Pn * A' + Q;
end

figure(3);
subplot(2,1,1);
plot(xn);
ylabel('xn','rotation',0);
subplot(2,1,2);
plot(x_post);
ylabel('xf','rotation',0);

figure(4);
plot(K);
title('Gain K');

figure(5);
plot(P);
title('Posterior Covariance Matrix P');

figure(6);
p3 = plot(xn,'b');
hold on;
p4 = plot(x_post, 'r');
legend([p3,p4], 'Unfiltered values','Filtered values');
hold off;

disp("delay of Filtered values and Unfiltered values is: " + finddelay(xn,x_post));
```

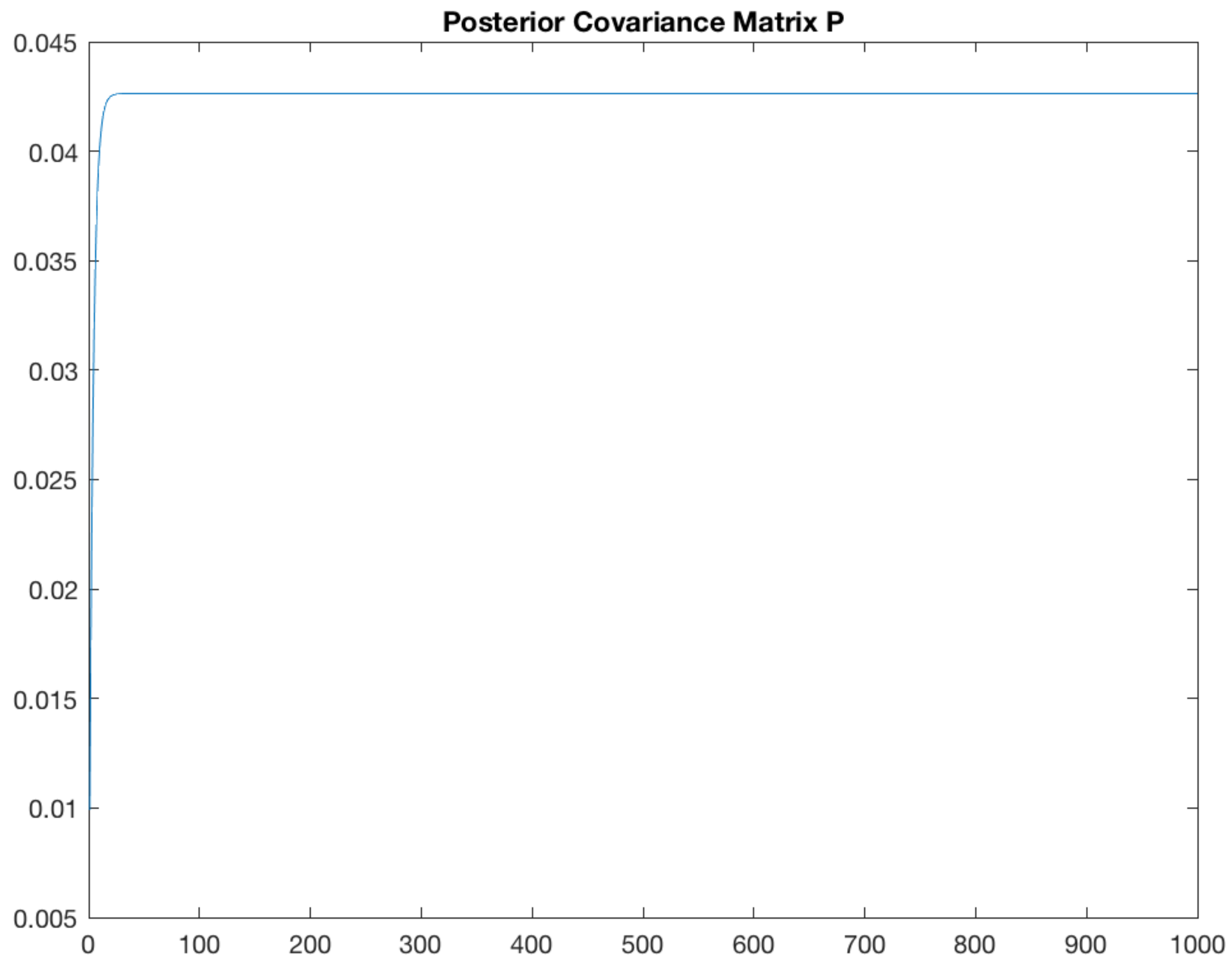
Note: in this code, there is a comment is wrong, the ‘Assume P(0) = 1’ should be ‘Assume P(0) = 0’.

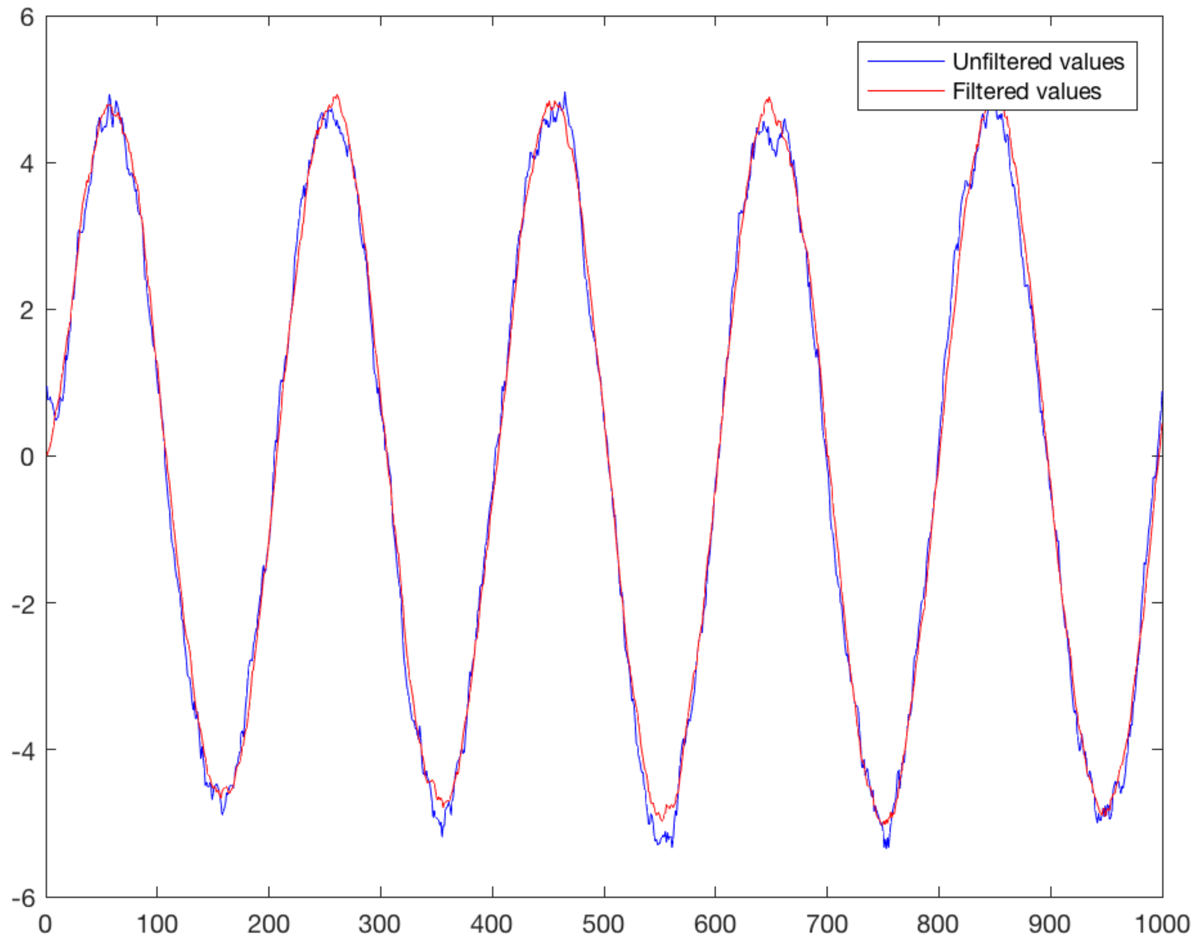

xn
6
4
2
0
-2
-4
-6
0
100
200
300
400
500
600
700
800
900
1000


xf
6
4
2
0
-2
-4
-6
0
100
200
300
400
500
600
700
800
900
1000

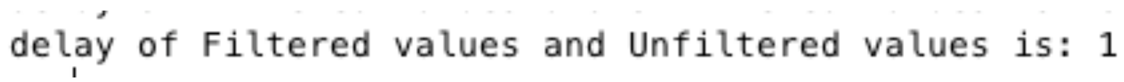
Result:


Gain K
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
100
200
300
400
500
600
700
800
900
1000


Posterior Covariance Matrix P
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
100
200
300
400
500
600
700
800
900
1000



In this case, I have initialize P to 1 and Q respectively. The results are similar but with different directions. If P equals to Q, the trends of 'Gain K' and 'Posterior Covariance Matrix P' images are increase from 0 to more than 0.4. If P is equals to 1, the trends are decrease, but the rate of convergency of both initialization are similar. As a result, any of the two values which I have tried can use in this case.



```
delay of Filtered values and Unfiltered values is: 1
```

The delay of Kalman filter is 1ms.
According to the figure above and the delay results of Kalman filter and Butterworth filter. Obviously, the Kalman filter is better than Butterworth filter. Kalman filter nearly filters out all noises. And its delay almost can be ignored.