Kalman Filter Bank for Failure Detection
Problem Statement

A process in a factory can be modeled by a simple second order ODE as follows:

$$\ddot{x} + b\dot{x} + bx = u$$
$$\dot{x}(0) = 0$$
$$x(0) = 0$$
$$b = 0.5$$

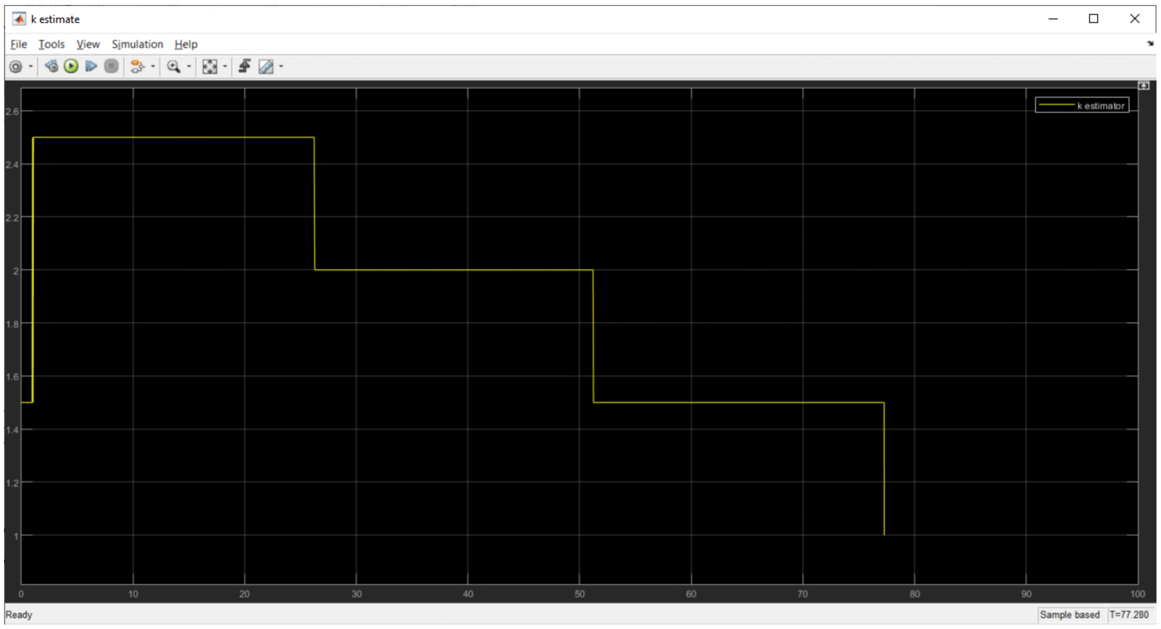
The process operates in response to a square wave input, *u*. The value of b is constant, but the value of k changes as failures in components occurs. Nominally, *k* = 2.5. When one failure occurs, the value of *k* becomes 2. When the second failure occurs, the value of *k* becomes 1.5. Similarly, with three failures, *k* becomes 1. The process can withstand three failures and continue operating, but a fourth failure would cause significant damage to the system, so it is imperative that the process be stopped upon the third failure. Your job is to develop a discrete-time, linear Kalman filter bank to predict the value of *k* based solely upon input (u) and measured output values (z) and stop the process when the third failure occurs.

Here are the particulars of the problem:

1) Implement the simple 2nd order system with a square wave of amplitude 1 and frequency of 0.05 Hz as the input, *u*, in Simulink with a fixed-step solver with a fixed-step time step of 0.01 seconds. Set the simulation to run 100 seconds. To test the system, have *k* start at 2.5 but changing to 2 at 23 seconds (first failure), changing to 1.5 at 49 seconds (second failure), and finally changing to 1 at 75 seconds (third failure).
2) To generate the measurements for the Kalman filters, add uncorrelated Gaussian noise (i.e. different seeds) to the states with zero mean and 0.001 variance.
3) While the simulation runs with a 0.01 fixed time-step, have the Kalman filters run updates (corrections) at 1 Hz instead of 100 Hz. You will have to have the corrector pass the predicted value as its output most of the time, only update the estimate once per second. Set up 4 filters, one set with *k* = 2.5, one set with *k* = 2, one set with *k* = 1.5, and the fourth set with *k* = 1. The choice of Q and R is up to you as the engineer, as is the initial value of covariance, P.
4) Determine which filter bank is most correct and use that information to indicate the current value of *k*. Set the simulation to stop when you determine that the third

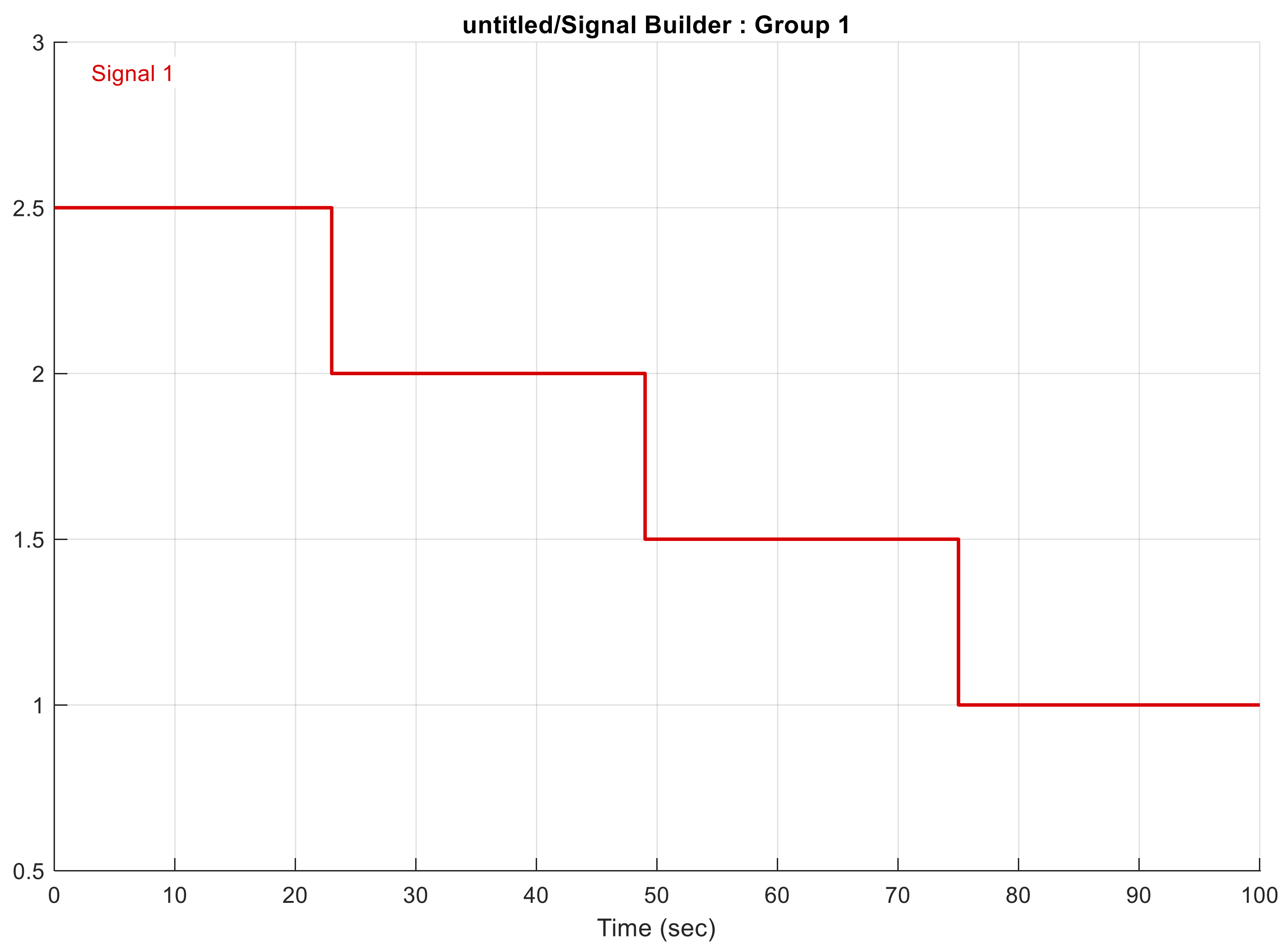
failure has occurred (i.e. when the estimated value of *k* = 1). It seems appropriate to use either the difference between the predictions and the measurements or the square root of the covariance as a way to determine which filter matches the current system dynamics. However, the metric is up to you as the engineer. Note that the estimates will be noisy and you'll have to use a moving average or low pass filter to help determine which filter is most accurate. The choice is up to you.

5) Document your approach to this complete project, especially step 4), and show your model and simulation results, including reporting how much time it takes your system to identify when a failure happens. I'd expect to see state time histories, state estimate time histories, and square root of the covariance time histories plotted with the estimation error, all which document the filter performance. – *As a side note: since the filters are all starting up at the same time and since it takes a bit for them to settle, I suggest you add logic to prevent the simulation from stopping in the first 1 or 2 seconds, regardless of the estimate of k in that first 1 or 2 seconds.*

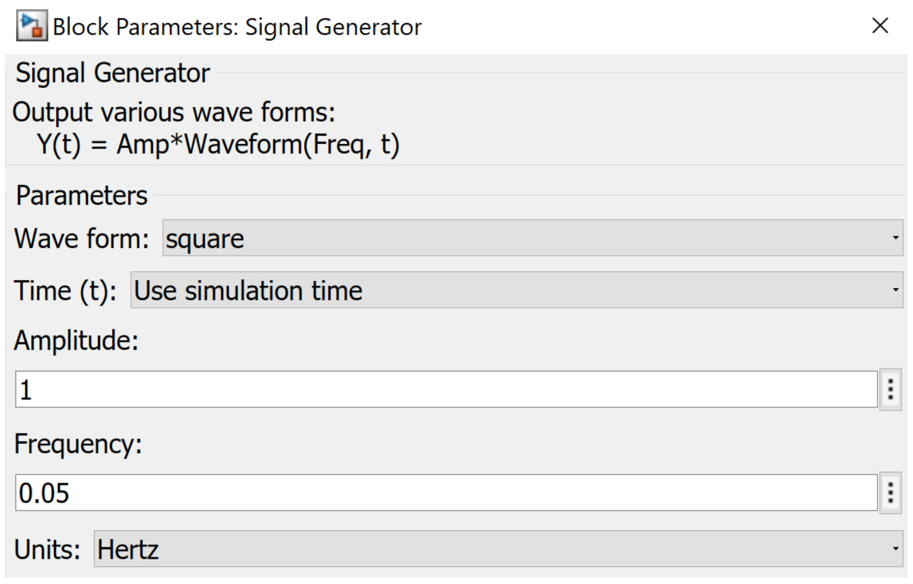


Note: The above figure is representative of what I'd expect the *k* estimate to look like. However, the above is not very fast at identifying the changes in *k*. I'd like to think that you could do better, but that's not a specific requirement for this project.

1. System Simulation



Signal Builder for changing the values of k at specified times


Block Parameters: Signal Generator

Signal Generator

Output various wave forms:
Y(t) = Amp*Waveform(Freq, t)

Parameters

Wave form: square

Time (t): Use simulation time

Amplitude:

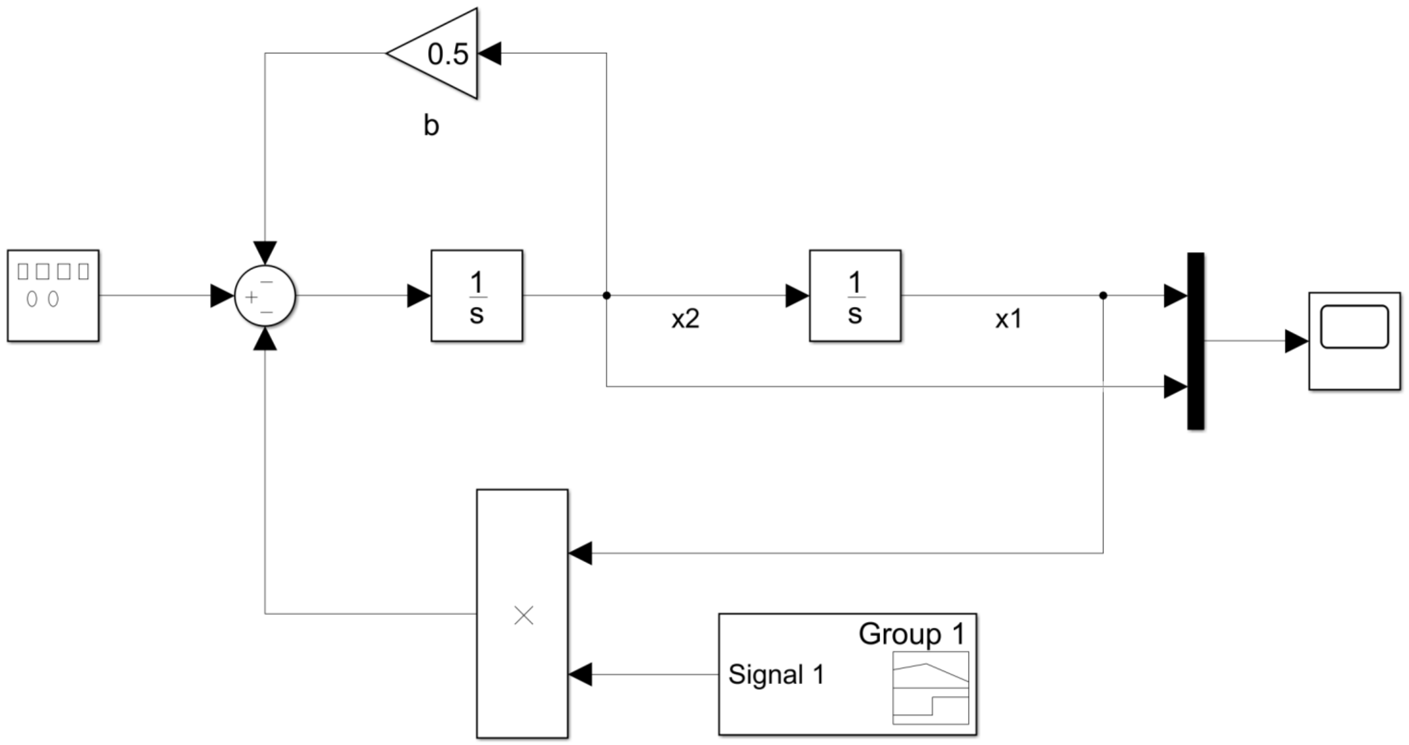
1

Frequency:

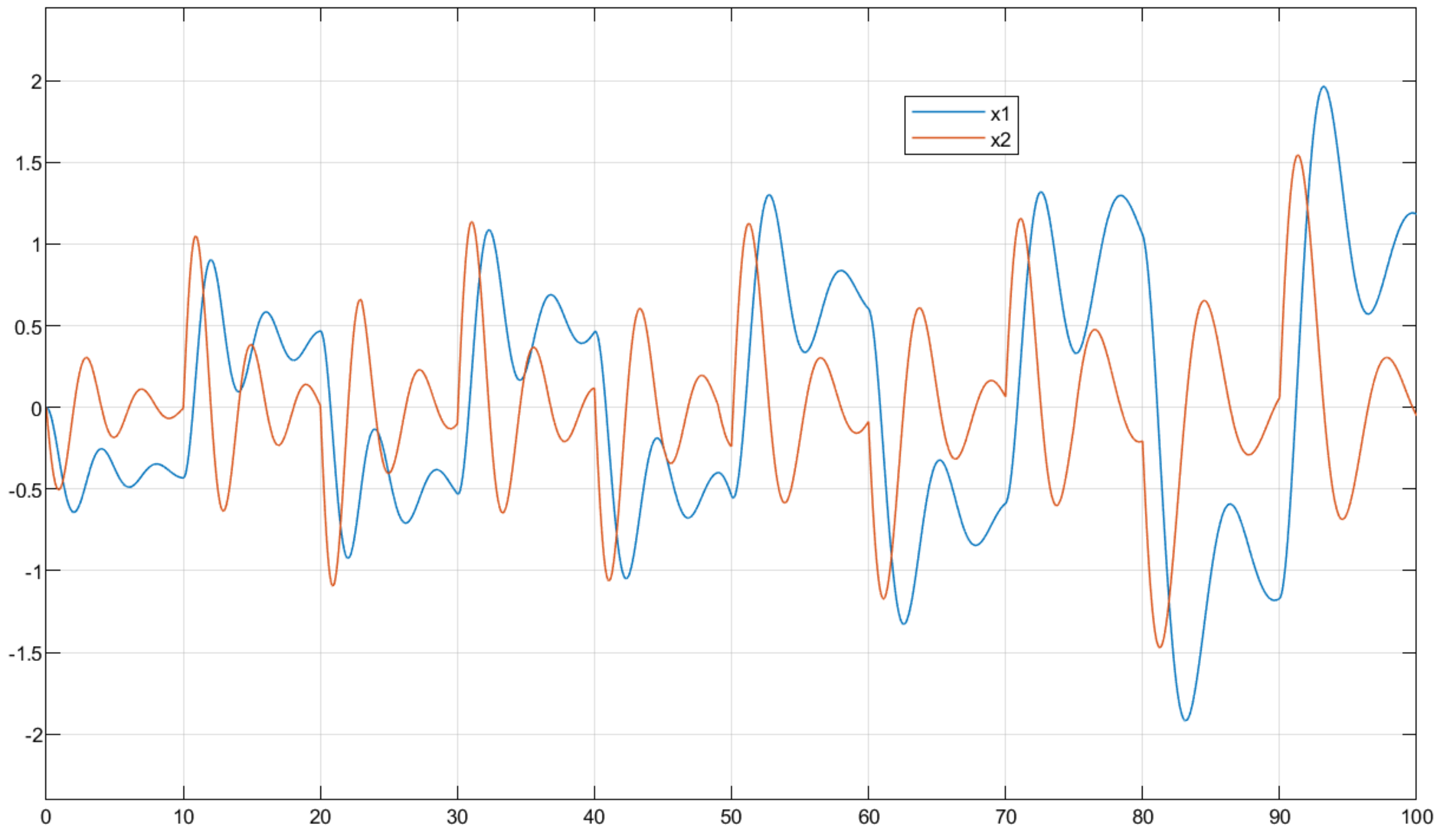
0.05

Units: Hertz

Parameters for input signal

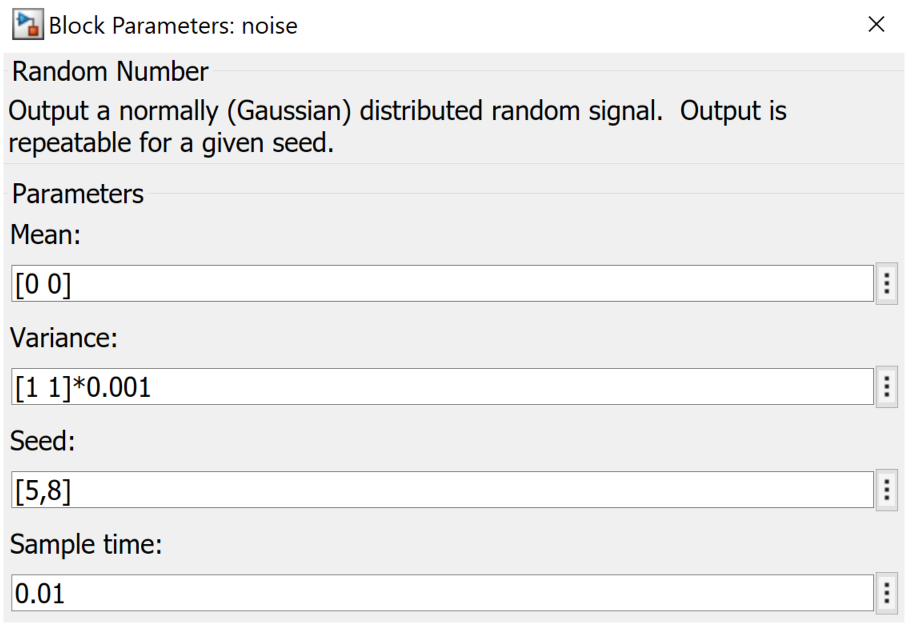


Simulink Diagram for the system



Output of states

2. Adding noise to the system



Block Parameters: noise

Random Number

Output a normally (Gaussian) distributed random signal. Output is repeatable for a given seed.

Parameters

Mean:

[0 0]

Variance:

[1 1]*0.001

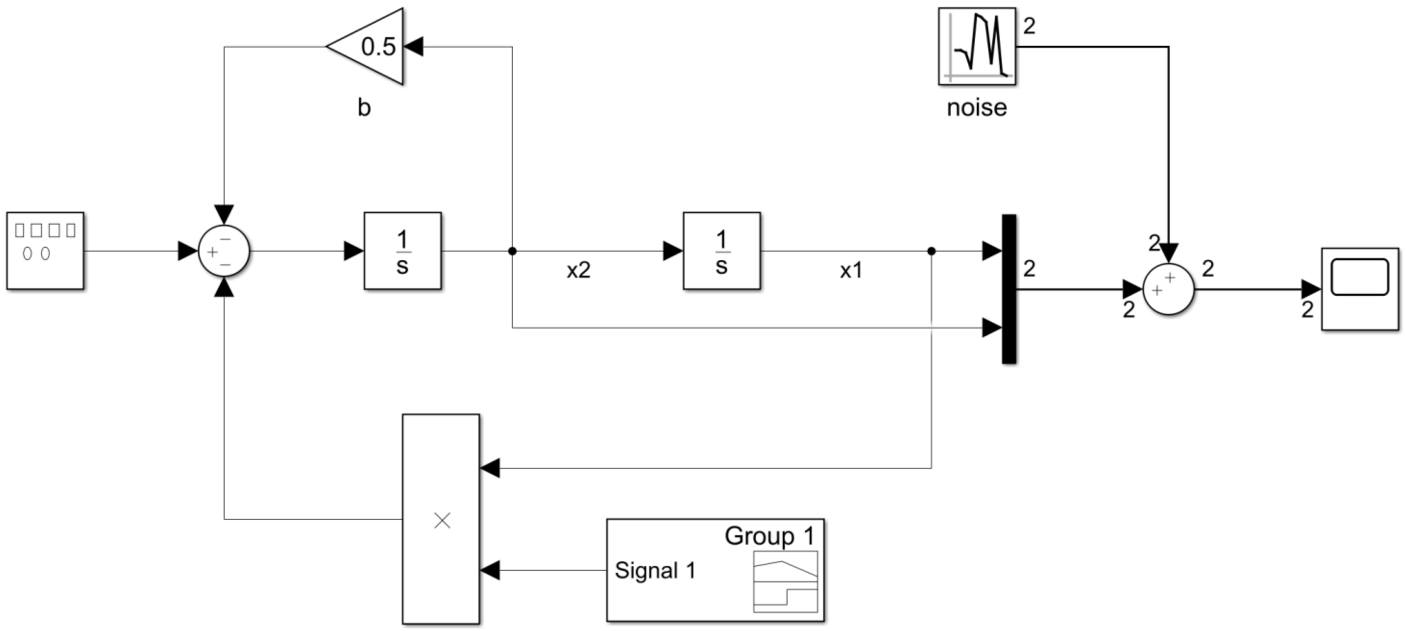
Seed:

[5,8]

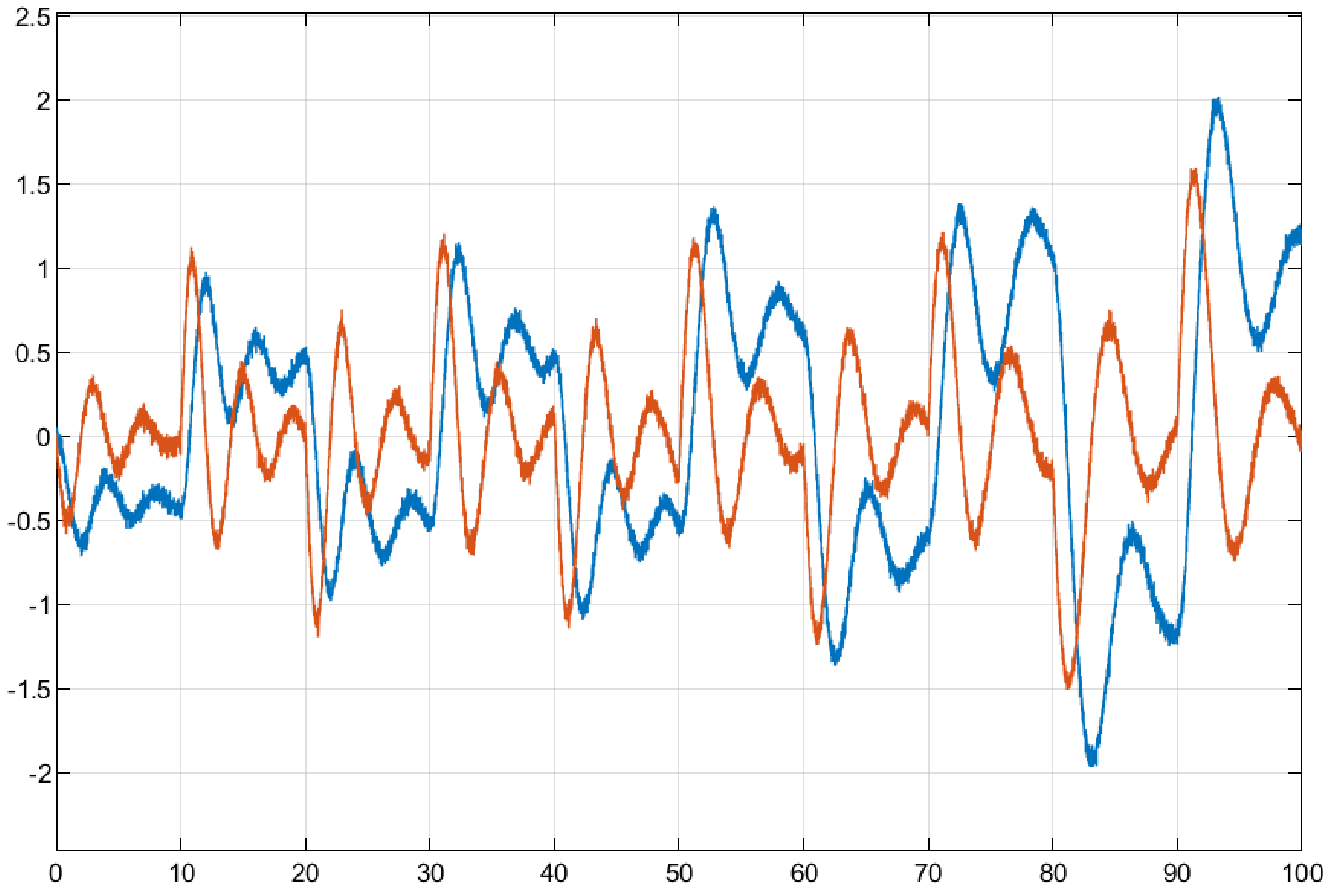
Sample time:

0.01

Noise Parameters

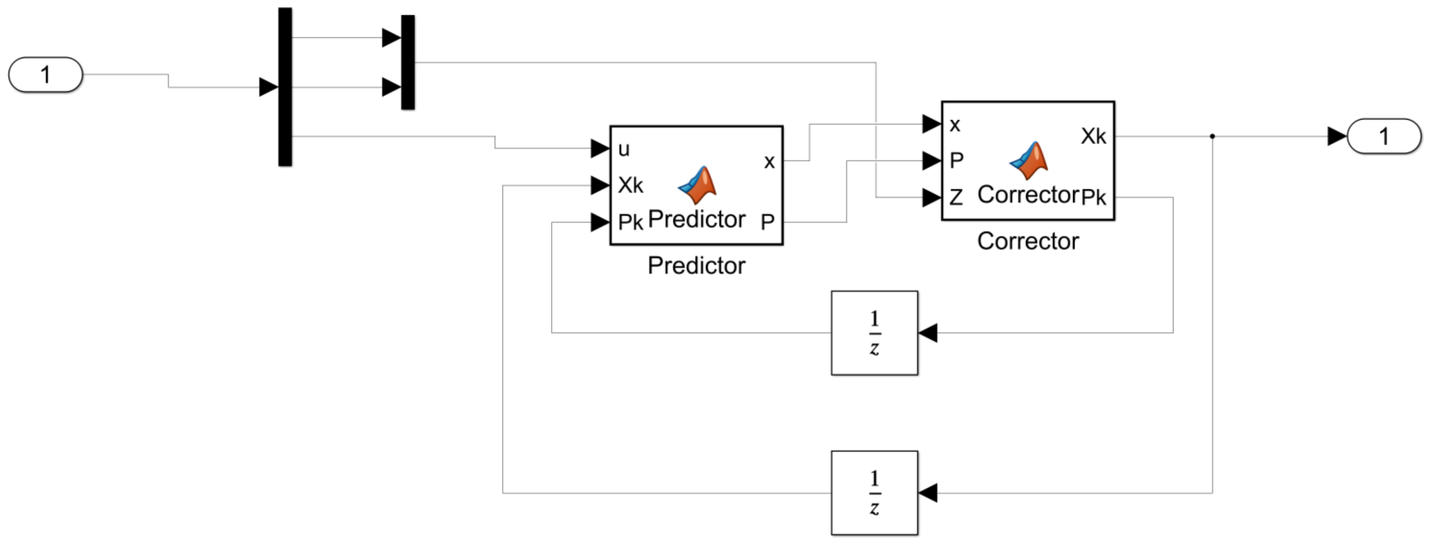


Simulink Diagram for adding noise



System Output after addition of noise

3. Implementing the Kalman filters



Kalman Filter Subsystem Diagram

Note: The code for all Kalman filters is the same, the only value which changes is of k in the predictor block which is 2.5, 2, 1.5 and 1 respectively.

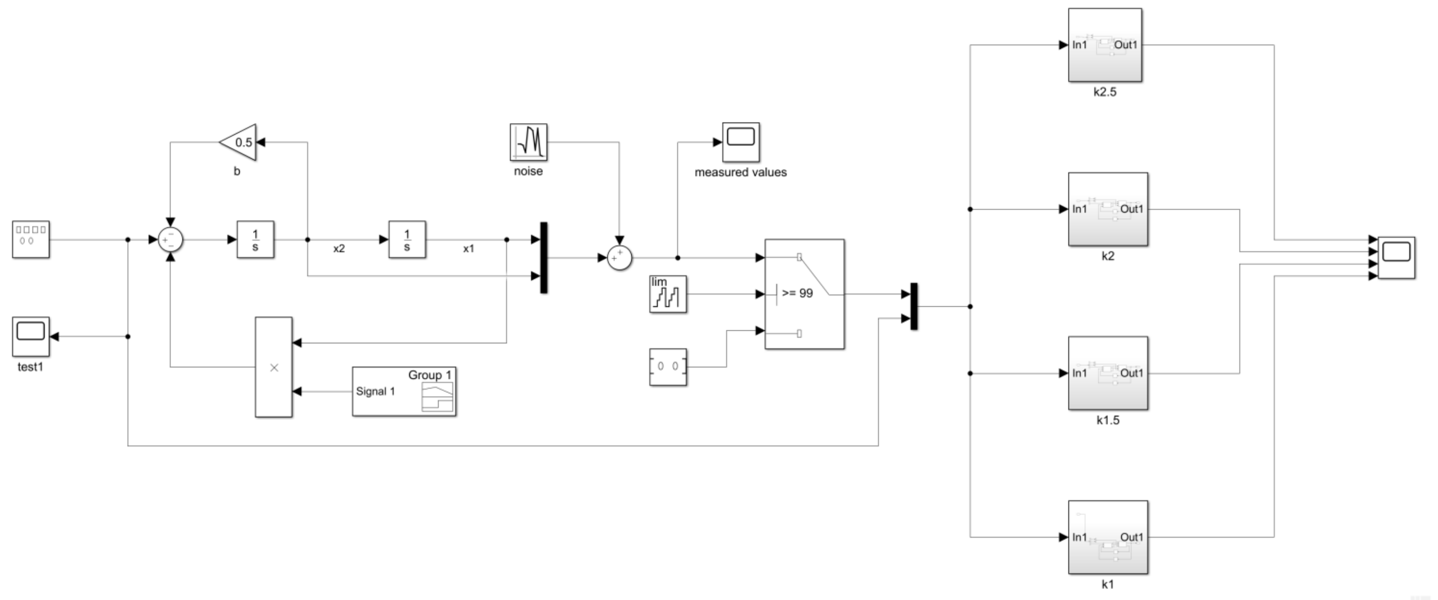
Predictor Block MATLAB Code:

```
function [x,P] = Predictor(u,Xk,Pk)

k=2.5; b=0.5;

A = [0 1;-k -b];

Ae=expm(A*0.01);

B= [0;1];

Bd = (inv(A))*(Ae - eye(2))*B;

Q=eye(2);

F = [-A Q; zeros(2,2) transpose(A)];

G = expm(F*0.01);

Qd=transpose(G(3:4,3:4))*G(1:2,3:4);

x=Ae*Xk + Bd*u;

P=(Ae*Pk*transpose(Ae)+Qd);

end
```

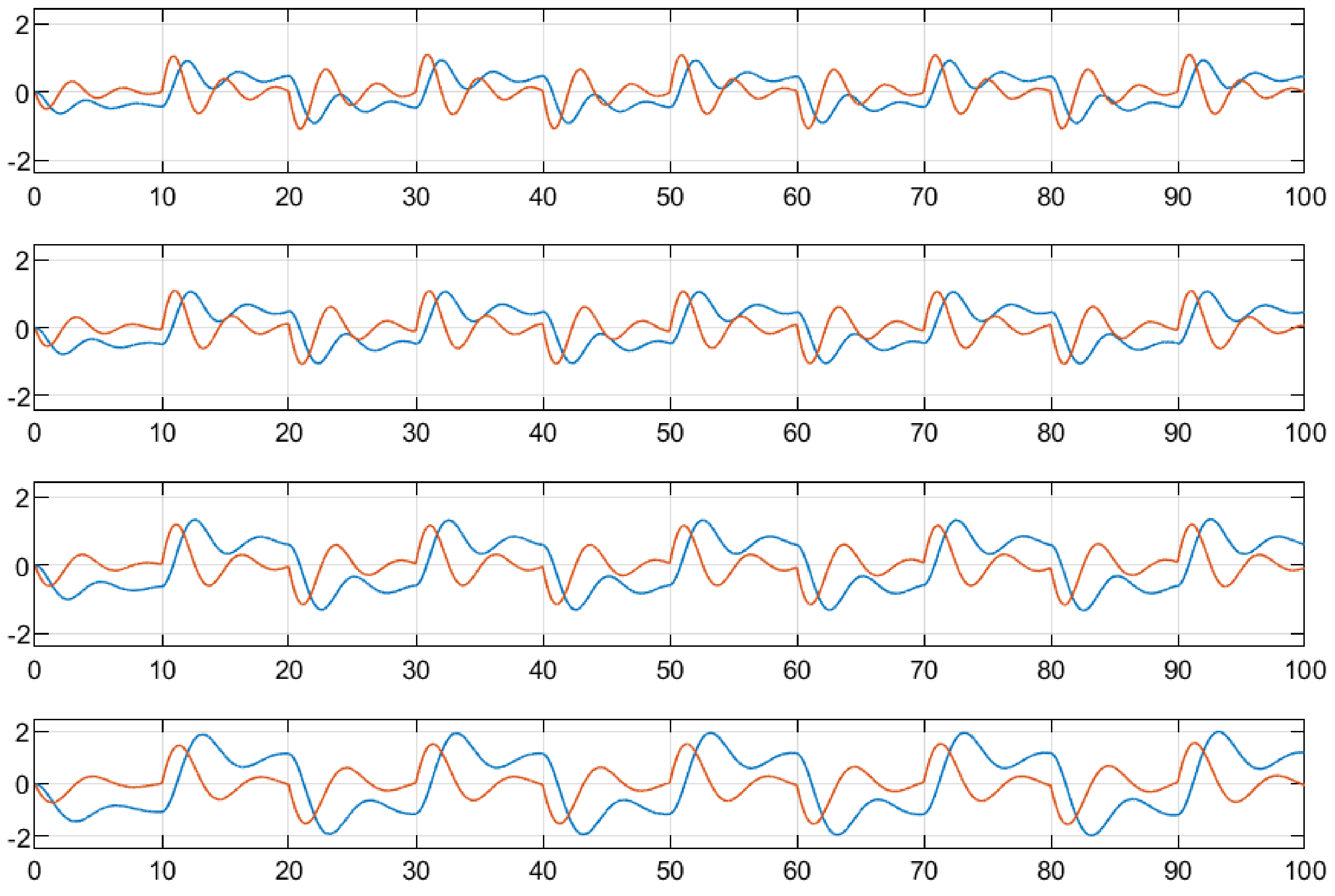
Corrector Block MATLAB Code:

```
function [Xk,Pk] = Corrector(x,P,Z)

H=eye(2); C=eye(2); R=eye(2)/0.01;

if Z(1)==0 && Z(2)==0

    Xk=x;

    Pk=P;

else

    K=P*transpose(H)*inv(H*P*transpose(H)+R);

    Pk=(eye(2)-K*H)*P*transpose(eye(2)-K*H)+K*R*transpose(K);

    Xk=x+K*(Z-H*x);

end

end
```

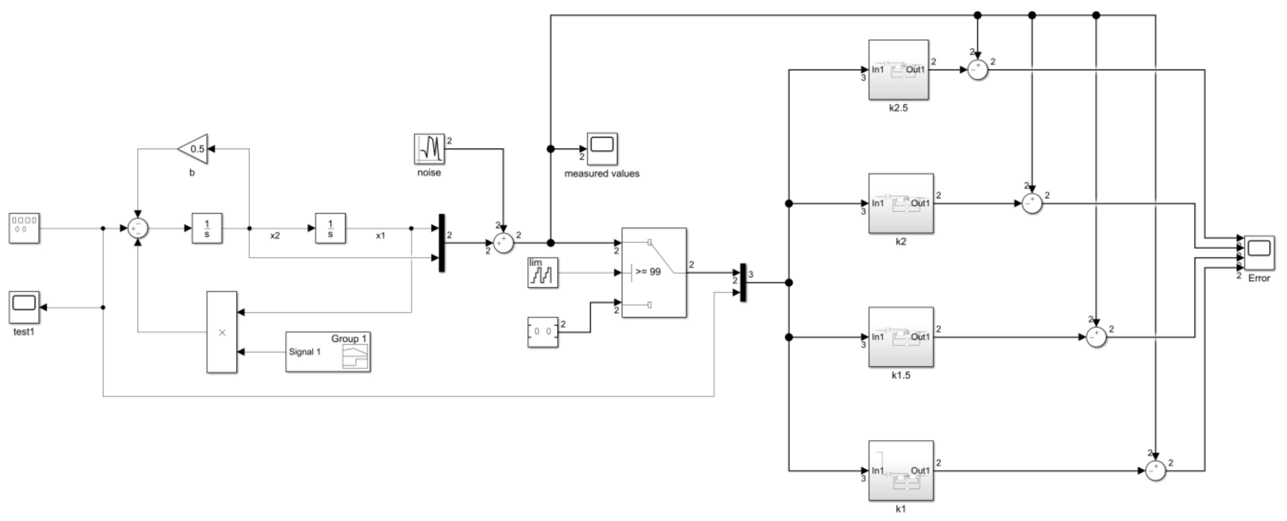
Four Kalman Filters are made by modifying the code as mentioned above. These are then put into a subsystem and their outputs are compared as shown in the following pages.



Simulink Diagram for Comparing Kalman Filter Outputs

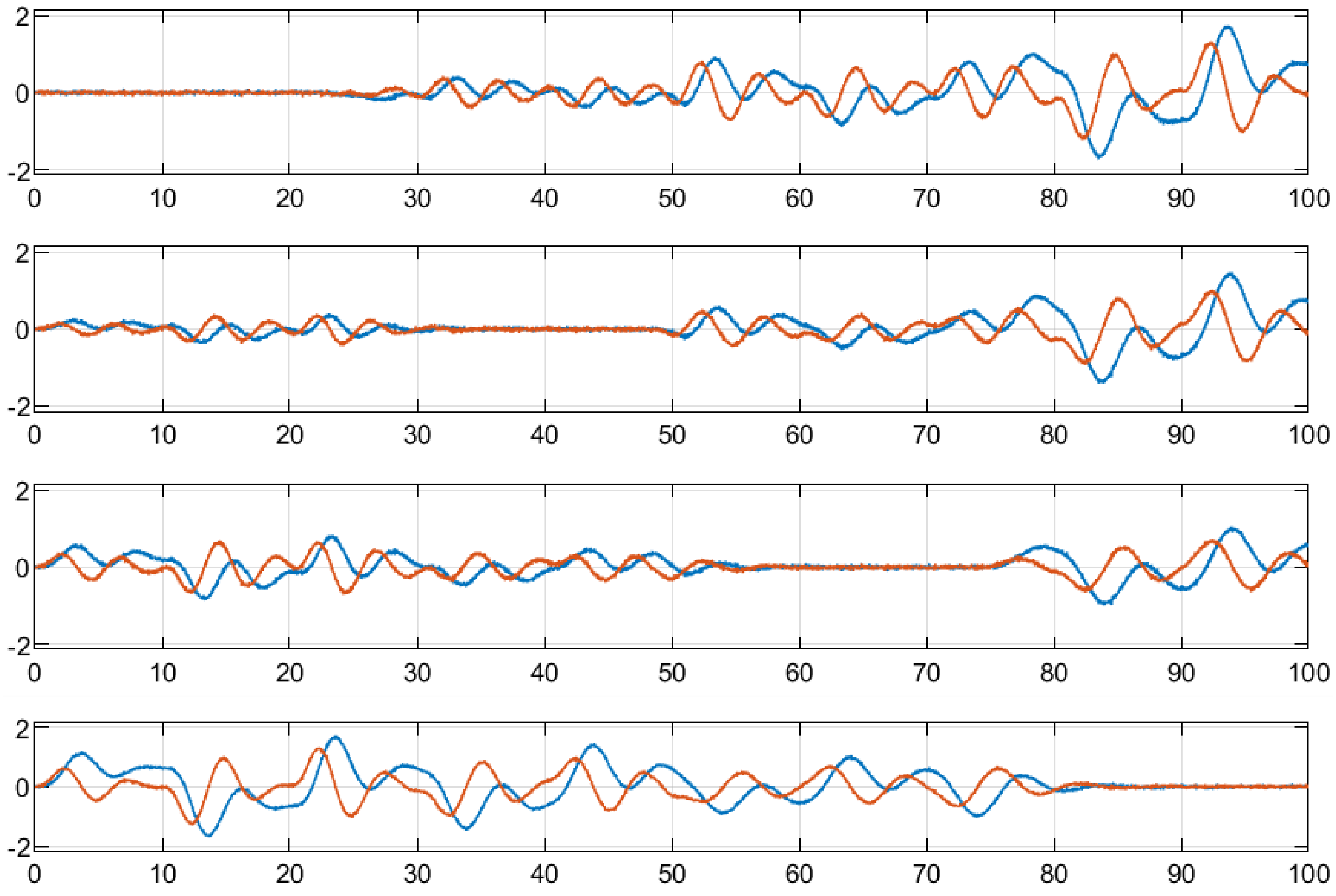


Output Comparison of Kalman Filters



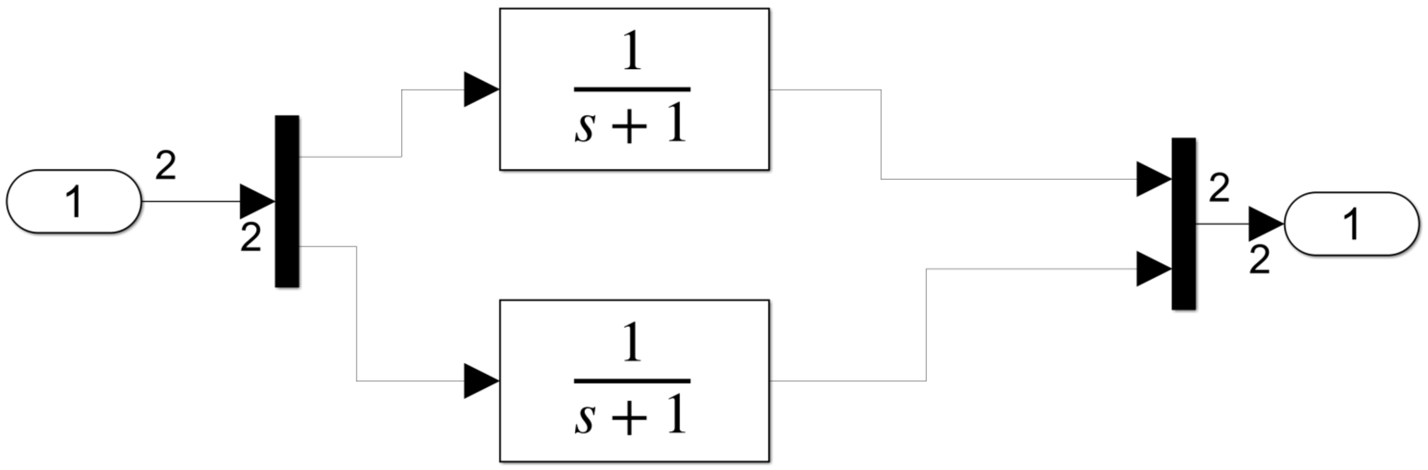
4.

Simulink Diagram for Error Comparison

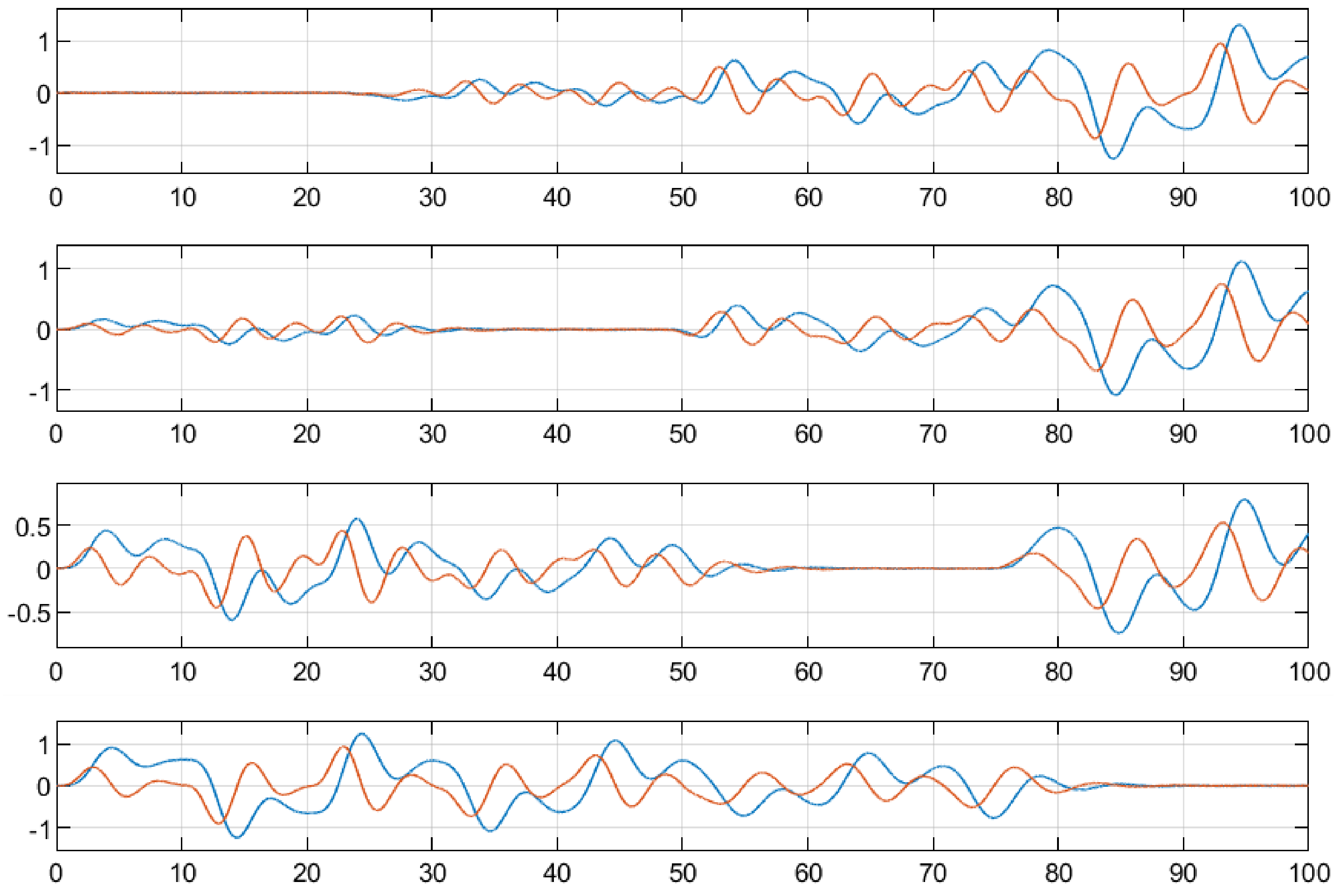


Comparison of Errors

As it can be seen from the above plots, the filters perform well and the errors are minimized when a filter has the same value of “k” as the system. This data can be used for selecting the best Kalman filter and to stop the system when the third failure occurs.



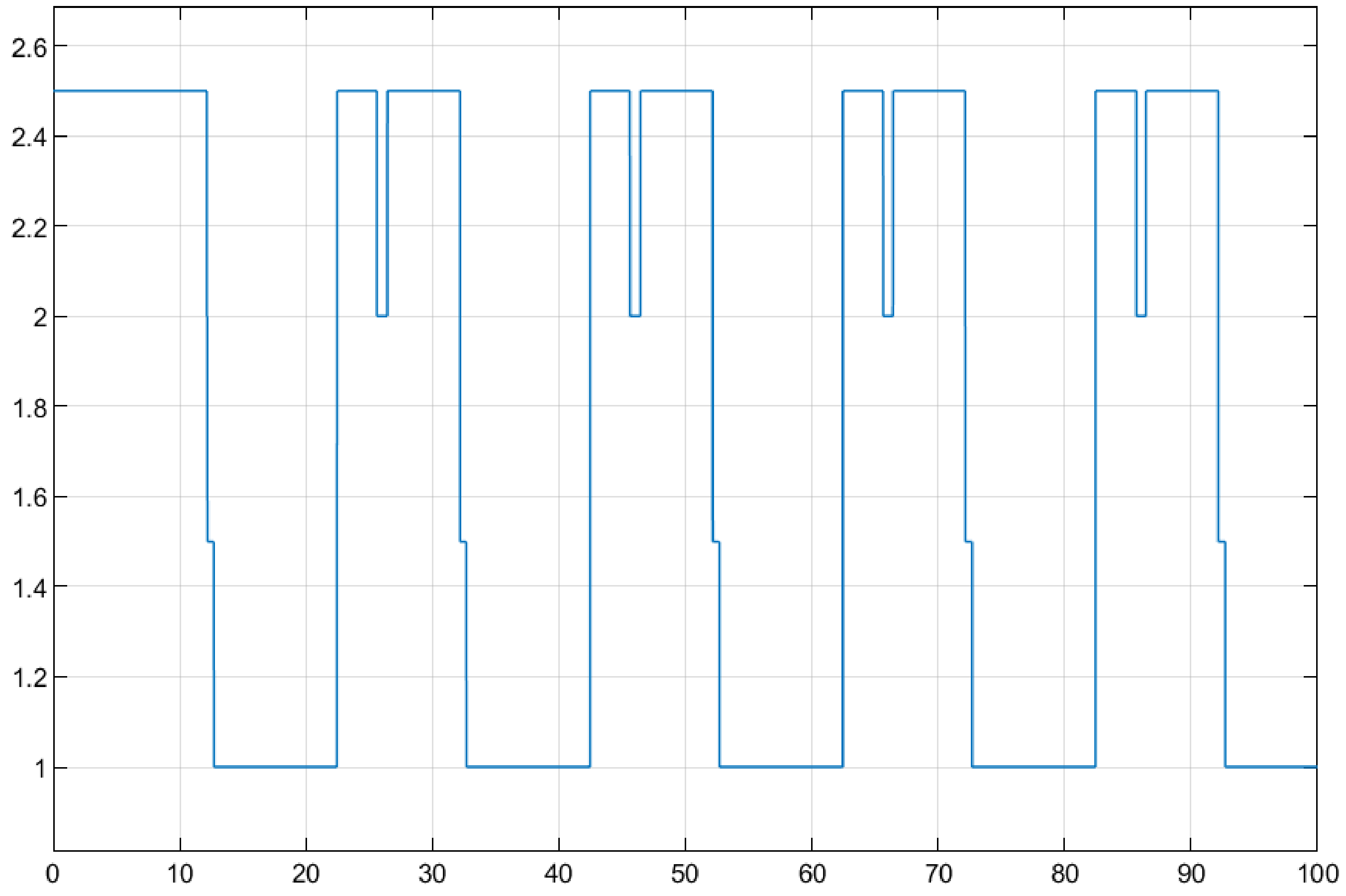
Filter to smooth out the errors



Comparison of errors after filtering

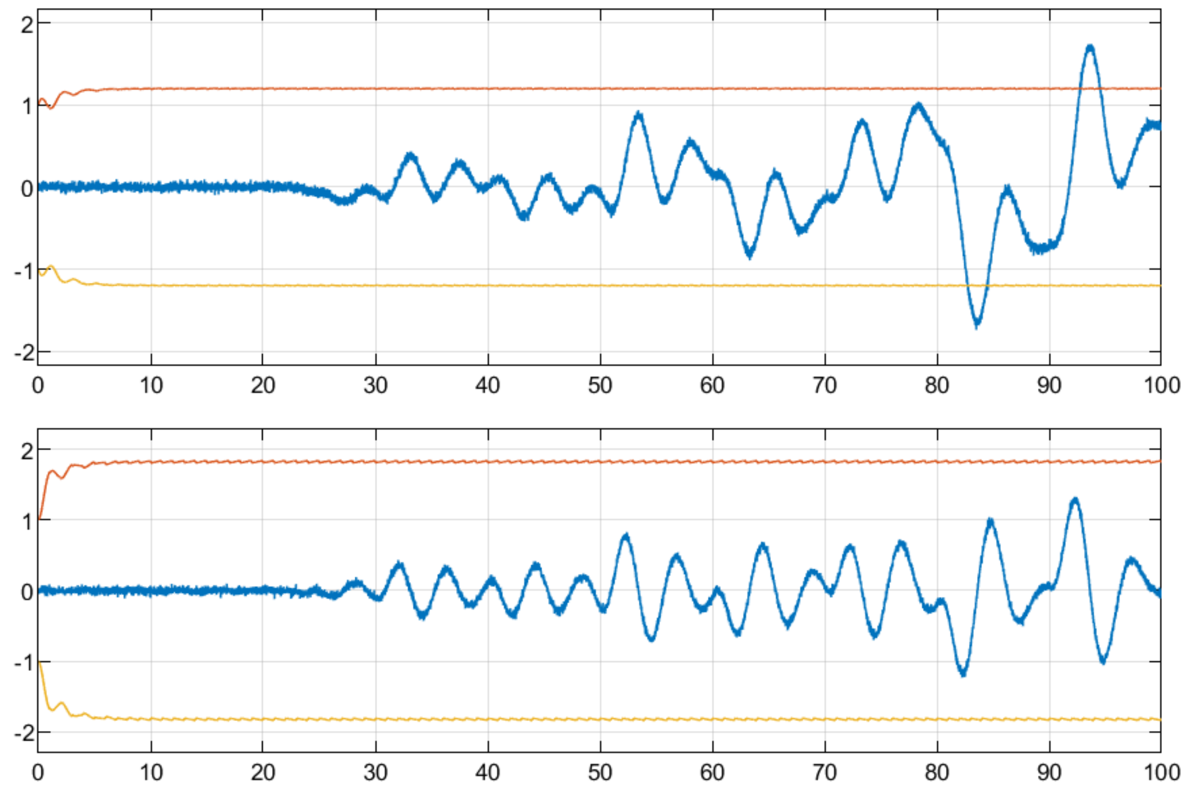
MATLAB Function for selecting the right value of k:

```
function K = selector(X1,X2,X3,X4)

[a,b]=min([X1(1),X2(1),X3(1),X4(1)]);

K=2.5;

switch b

    case 1

        K=2.5;

    case 2

        K=2;

    case 3

        K=1.5;

    case 4

        K=1;

end
```

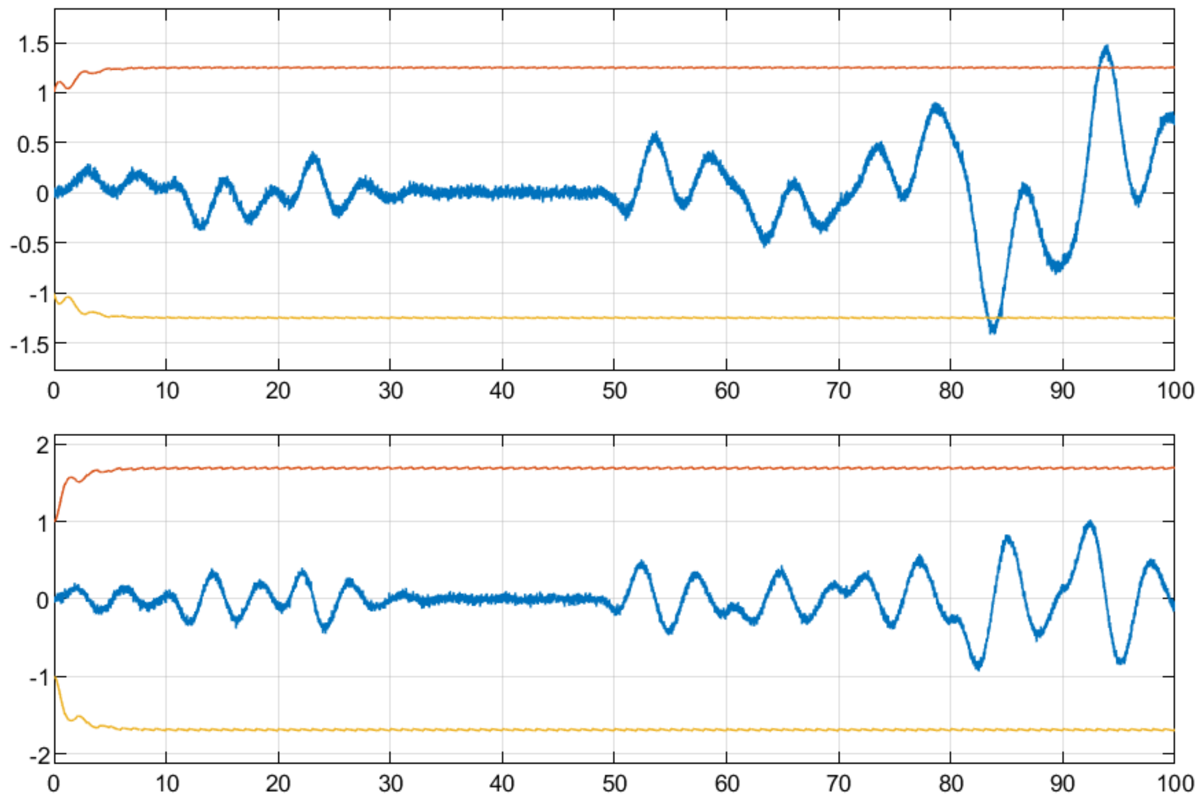


Estimated output for K values

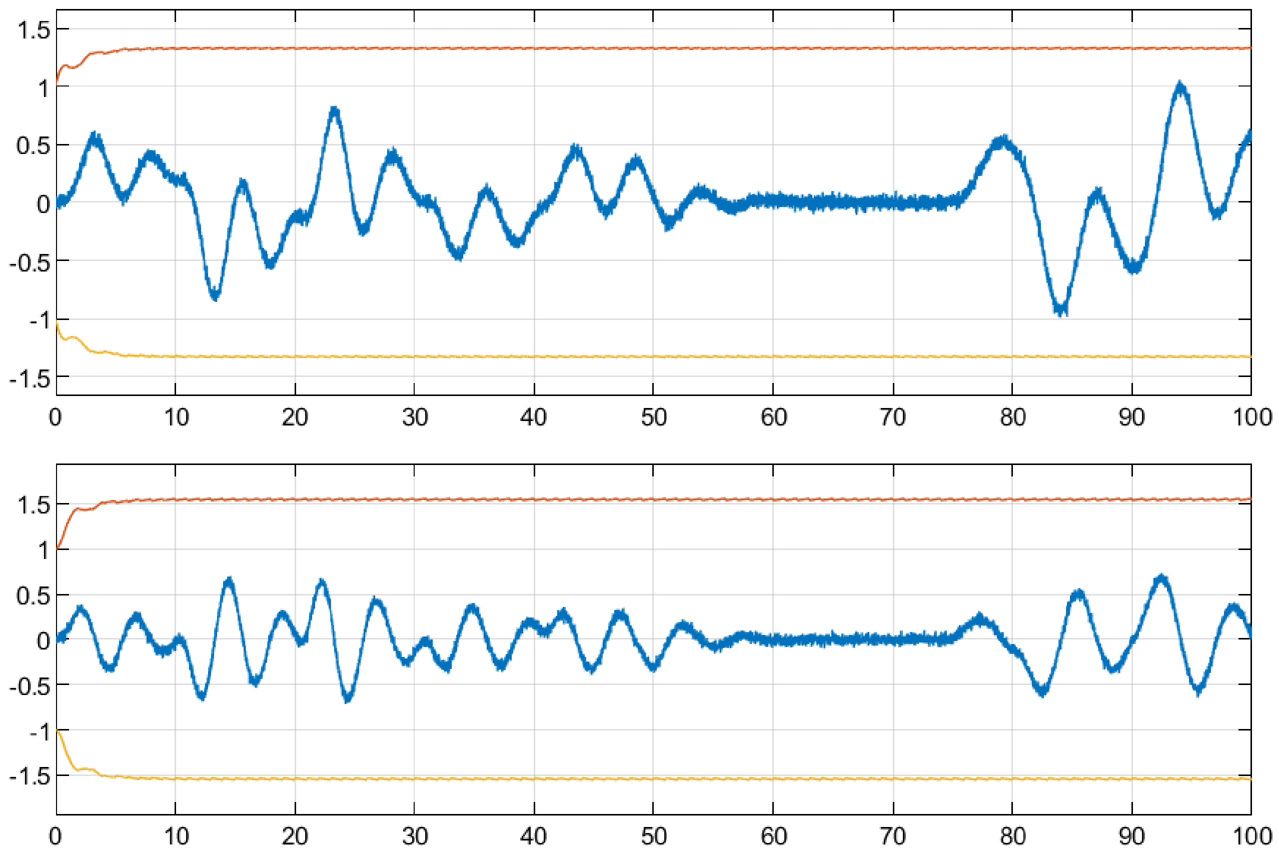
This doesn’t work as well so another attempt is made using the covariance method.



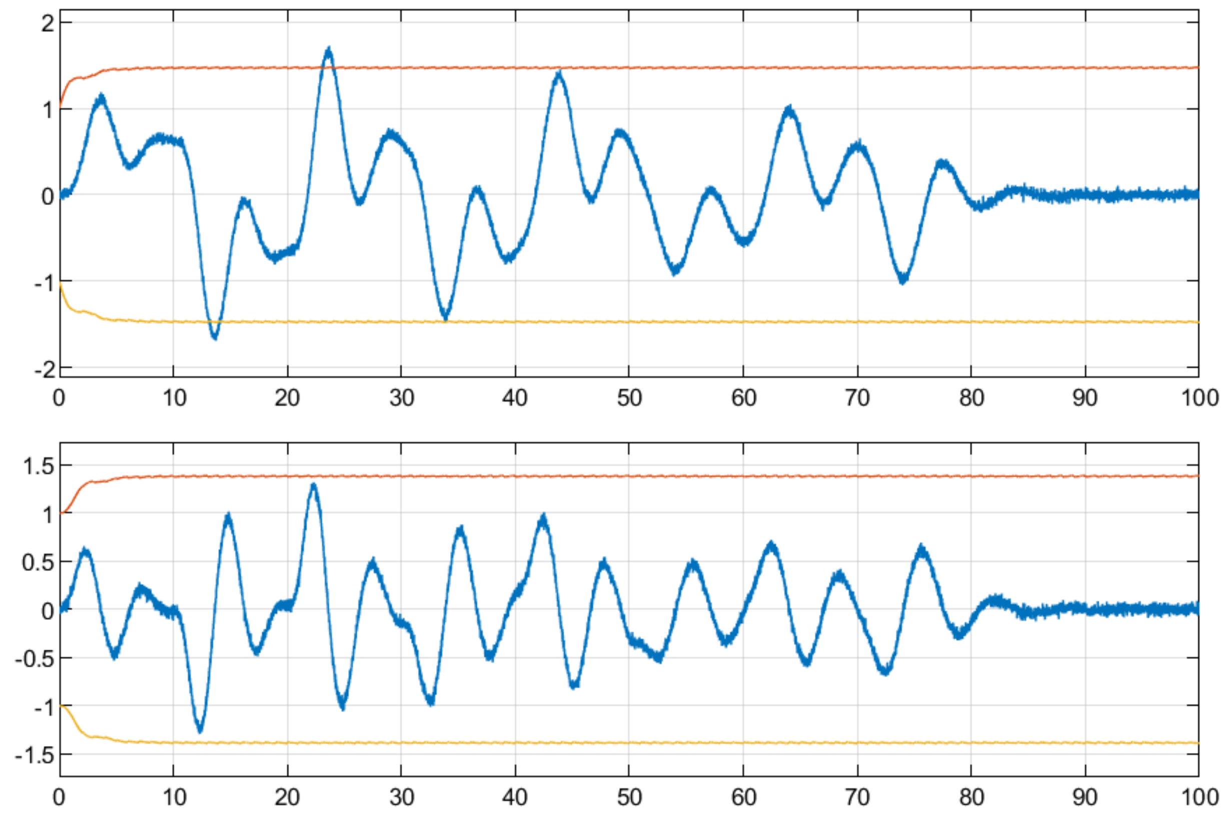
Covariance for k=2.5



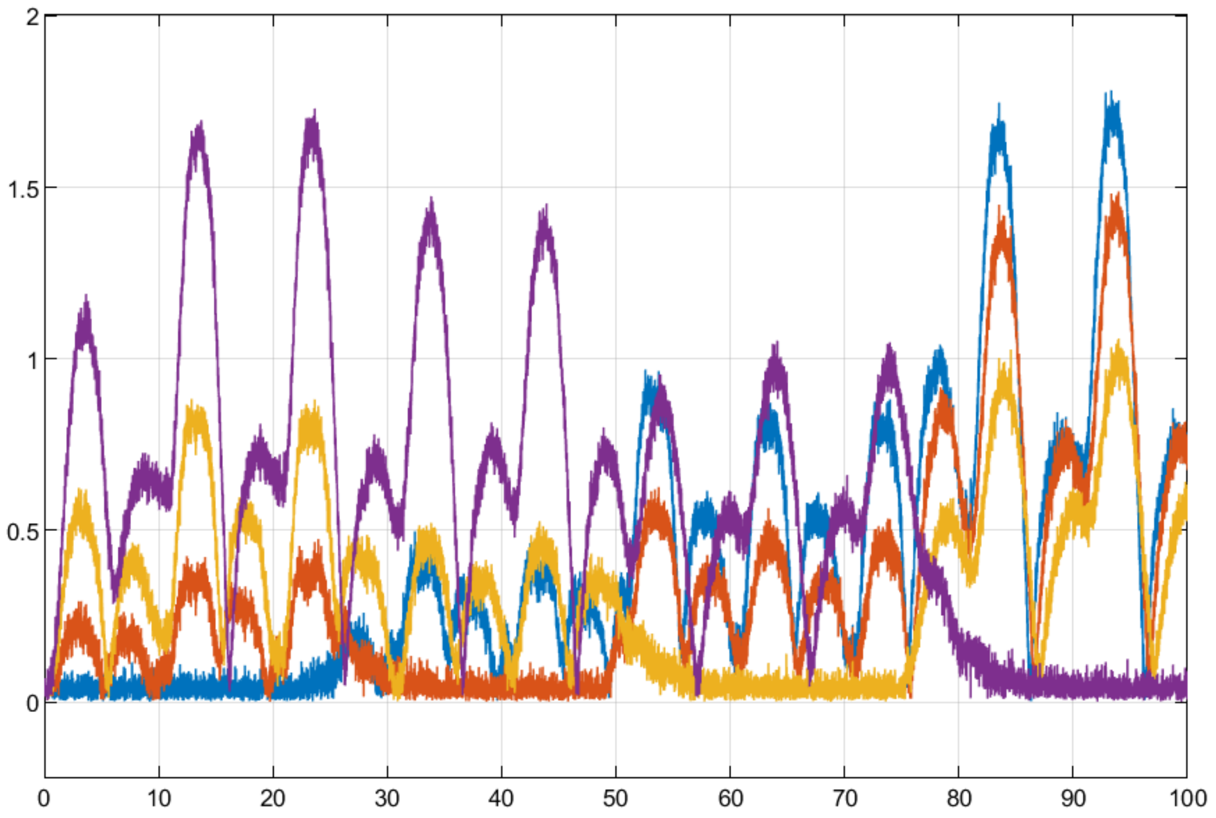
Covariance for k=2



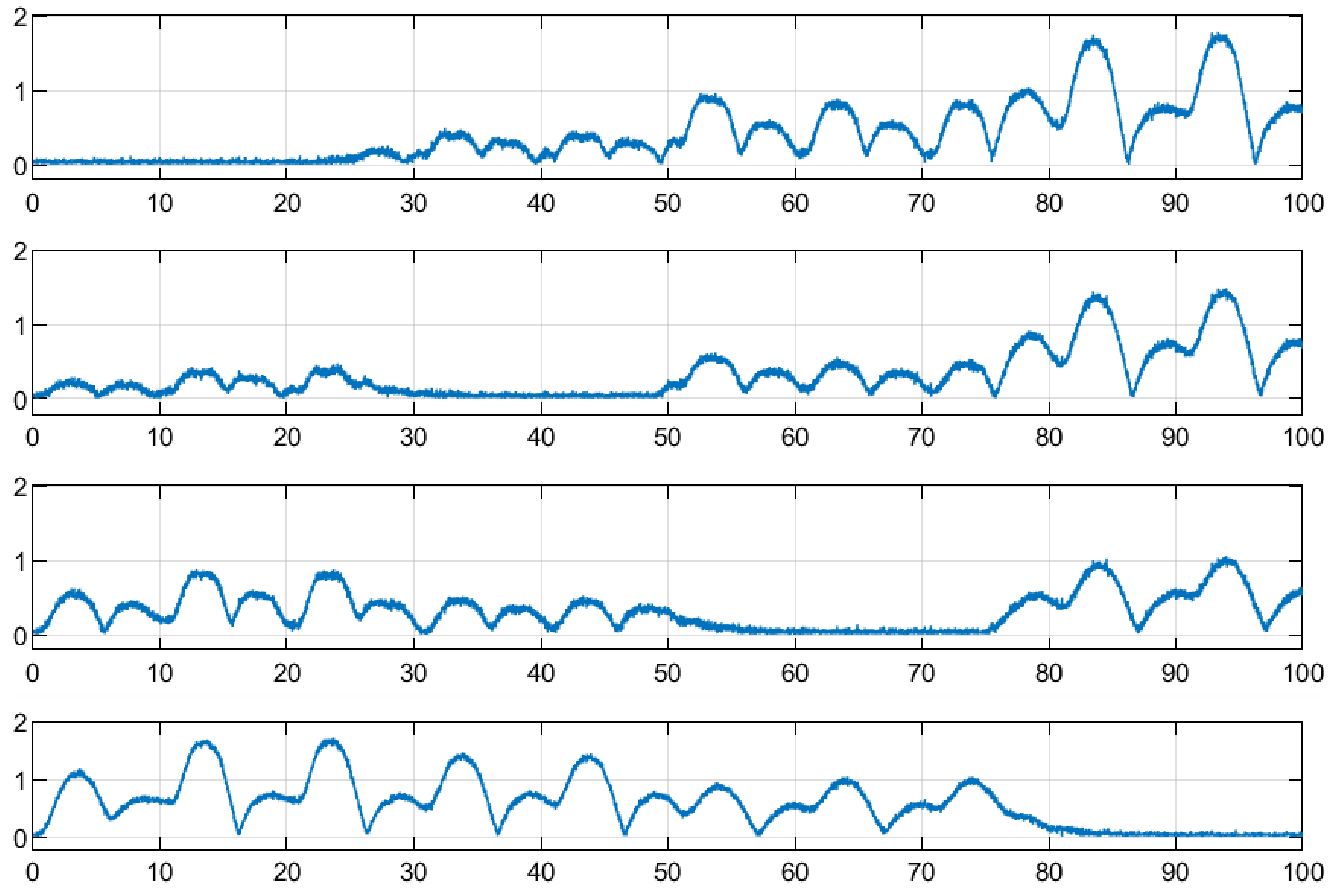
Covariance for k=1.5



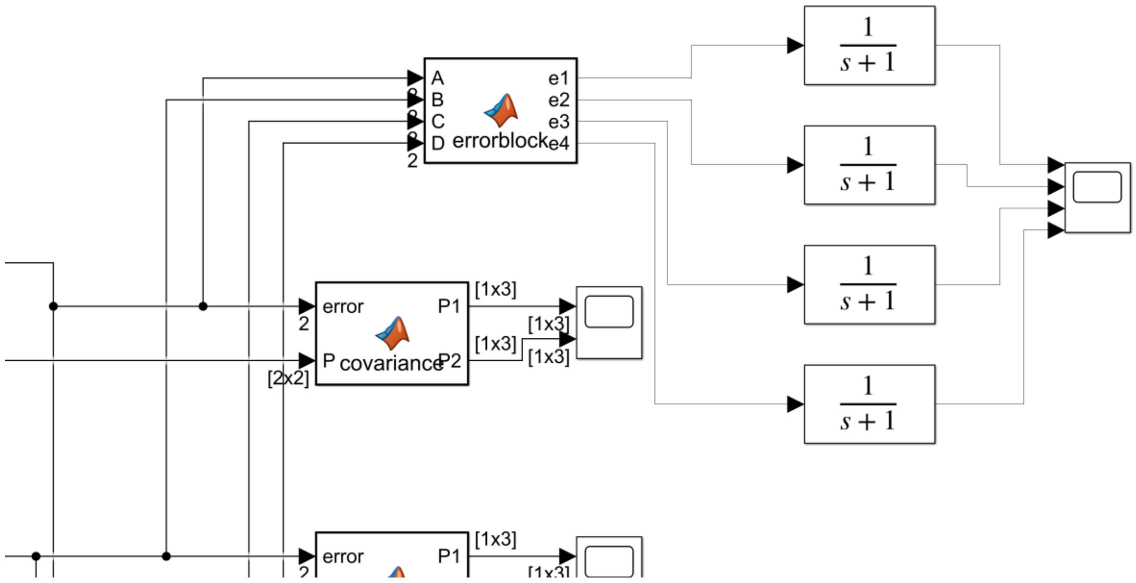
Covariance for k=1



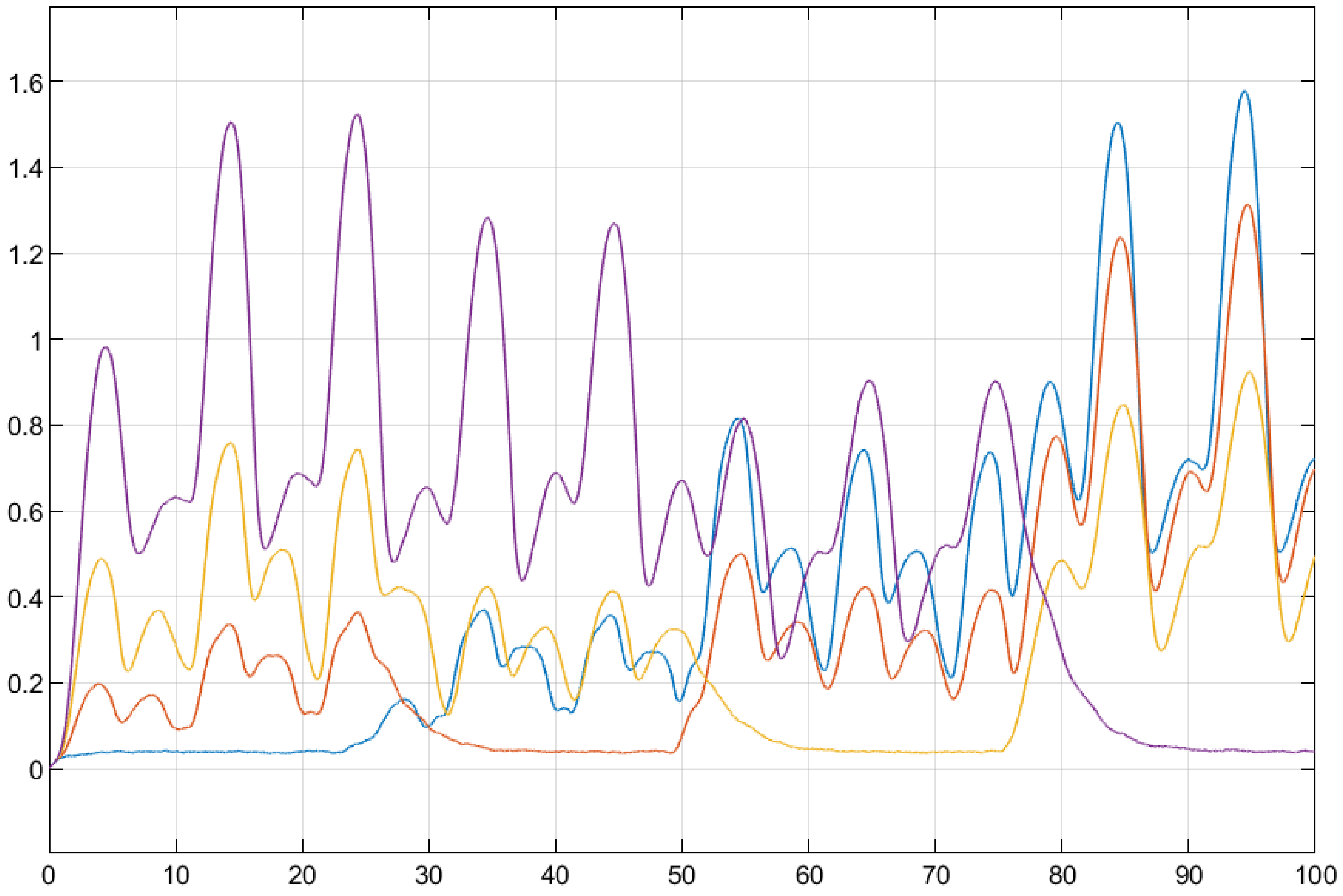
Error Block Outputs



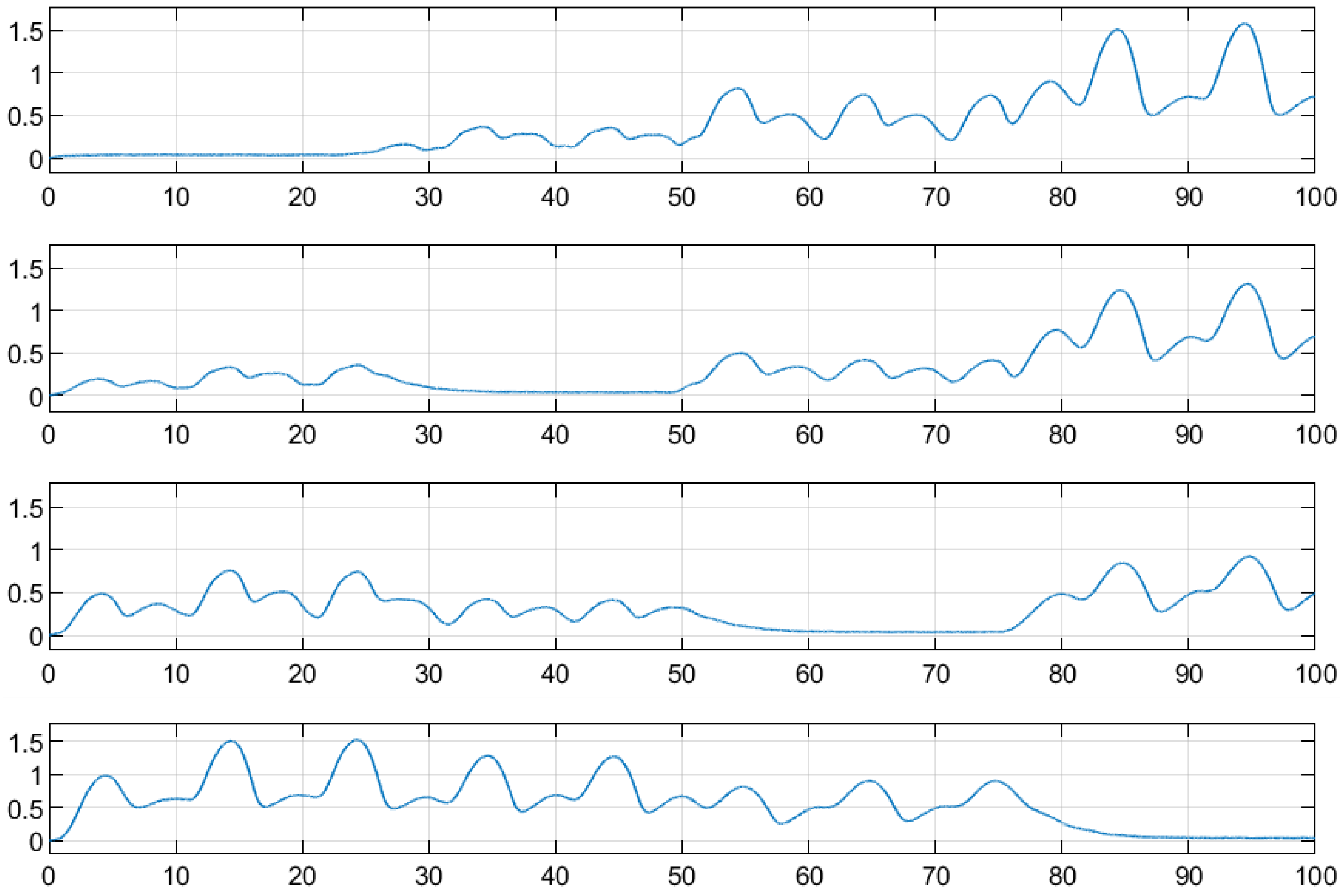
This output is noisy, so it might cause issues with selection of k. The errors have been filtered using a simple transfer function as shown ahead.



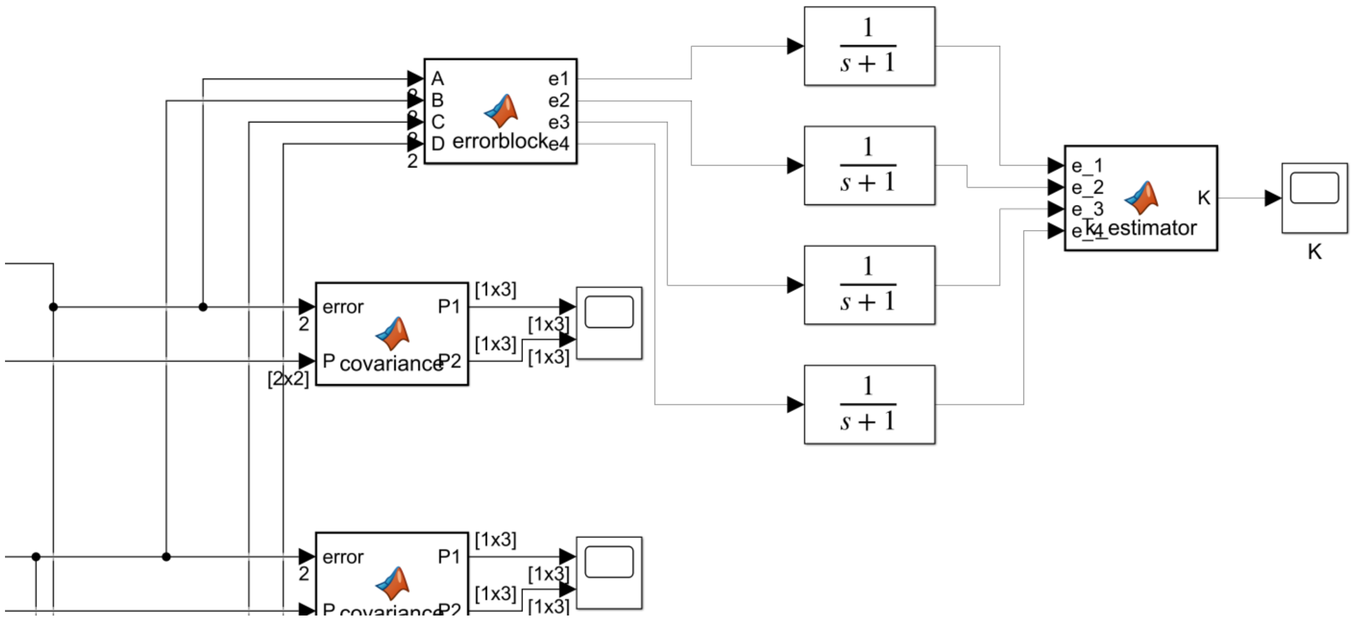
Simulink Diagram for filtering and estimating k



Error output after filtering



Individual error outputs after filtering



Simulink Diagram for Selecting K

MATLAB code for covariance block

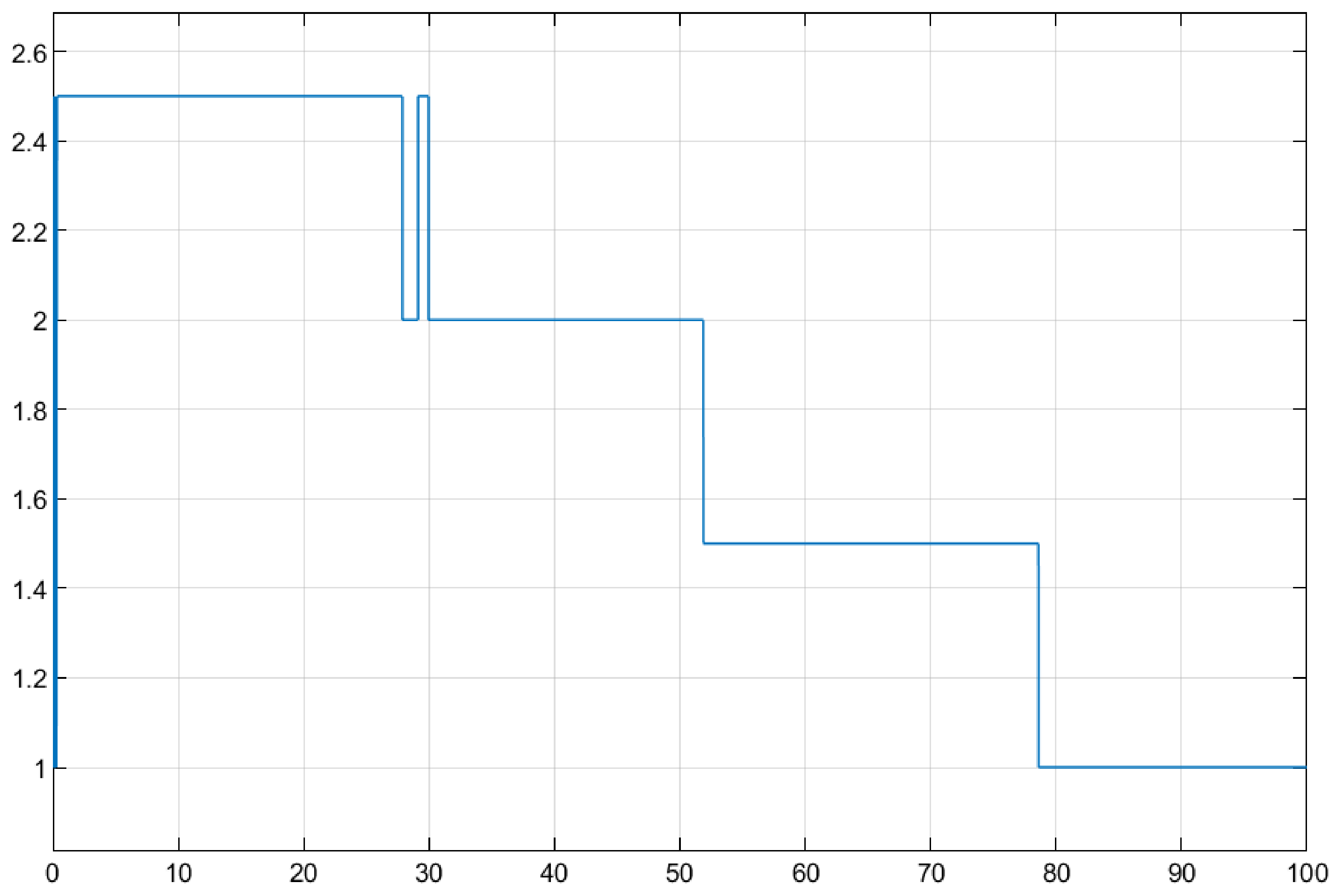
```
function [P1,P2]=covariance(error,P)
P1=[error(1) sqrt(P(1,1)) -sqrt(P(1,1))];
P2=[error(2) sqrt(P(2,2)) -sqrt(P(2,2))];
end
```

MATLAB code for errorblock

```
function [e1,e2,e3,e4] = errorblock(A,B,C,D)
e1=sqrt(A(1)^2+A(2)^2);
e2=sqrt(B(1)^2+B(2)^2);
e3=sqrt(C(1)^2+C(2)^2);
e4=sqrt(D(1)^2+D(2)^2);
end
```

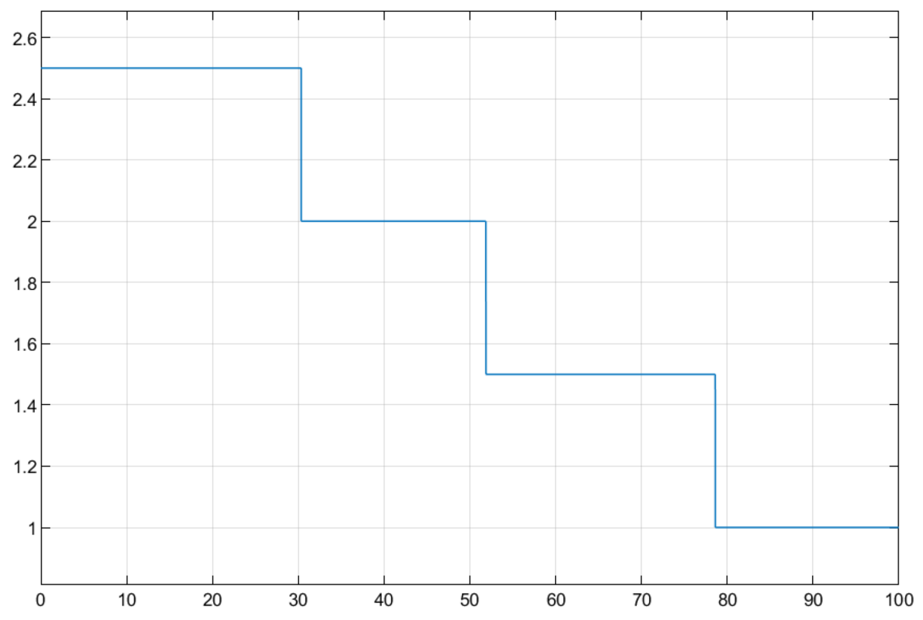
MATLAB code for k_estimator block

```
function K = k_estimator(e_1,e_2,e_3,e_4)
[i,j]=min([e_1,e_2,e_3,e_4]);
K=2.5;
switch j
    case 1
        K=2.5;
    case 2
        K=2;
    case 3
        K=1.5;
    case 4
        K=1;
end
```



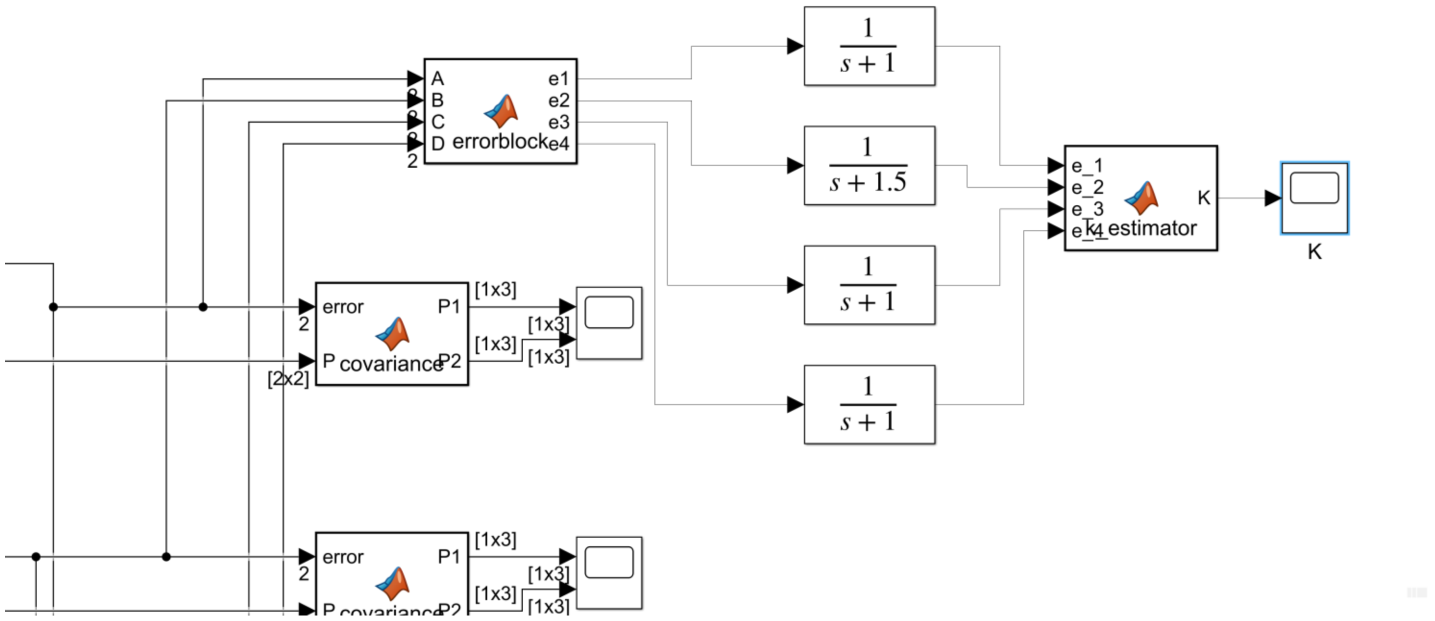
Estimated plot for K value

It can be seen that there is some issue at the start and at the point where k switches from 2.5 to 2. Manipulating the filter transfer functions may help.

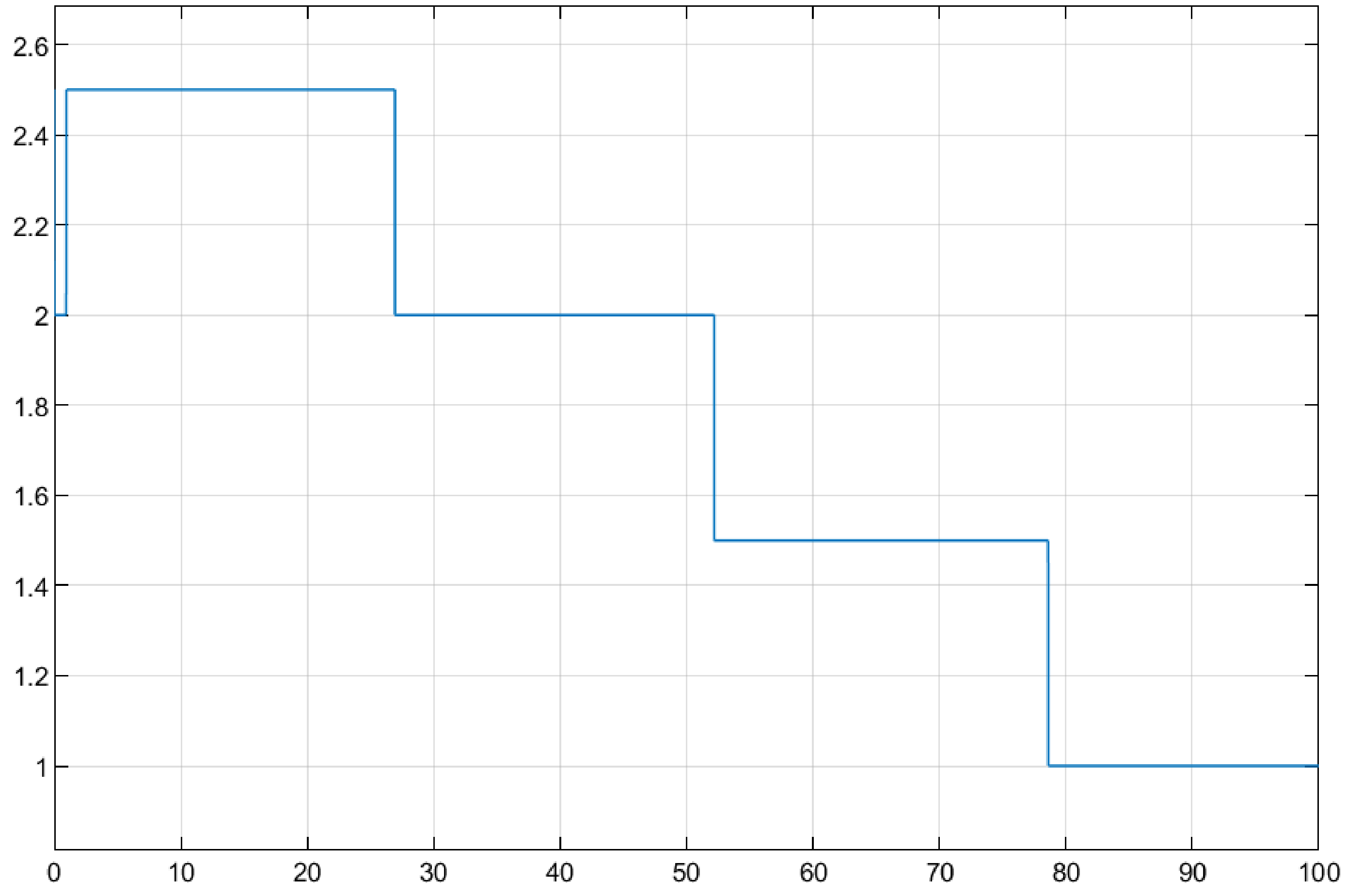


Estimated plot for K value at 1/s+1.25

Changing the value of first transfer function as 1/s+1.25 gives the following output. However, the system recognizes the change too late, changing it again to the following system provides a more accurate output.

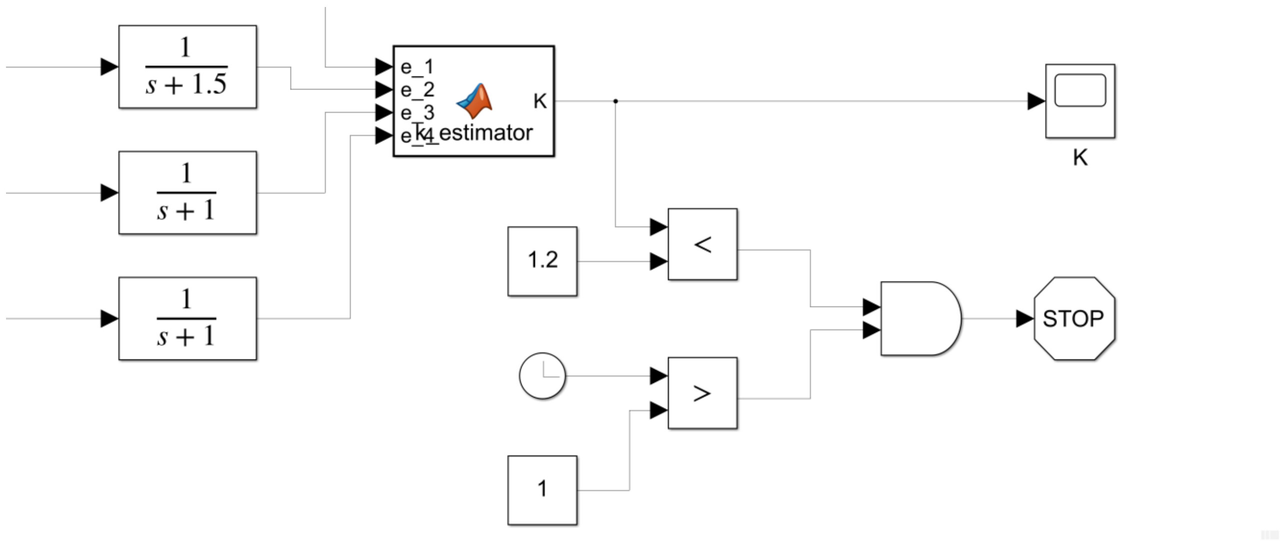


Simulink Diagram for recognizing K values accurately

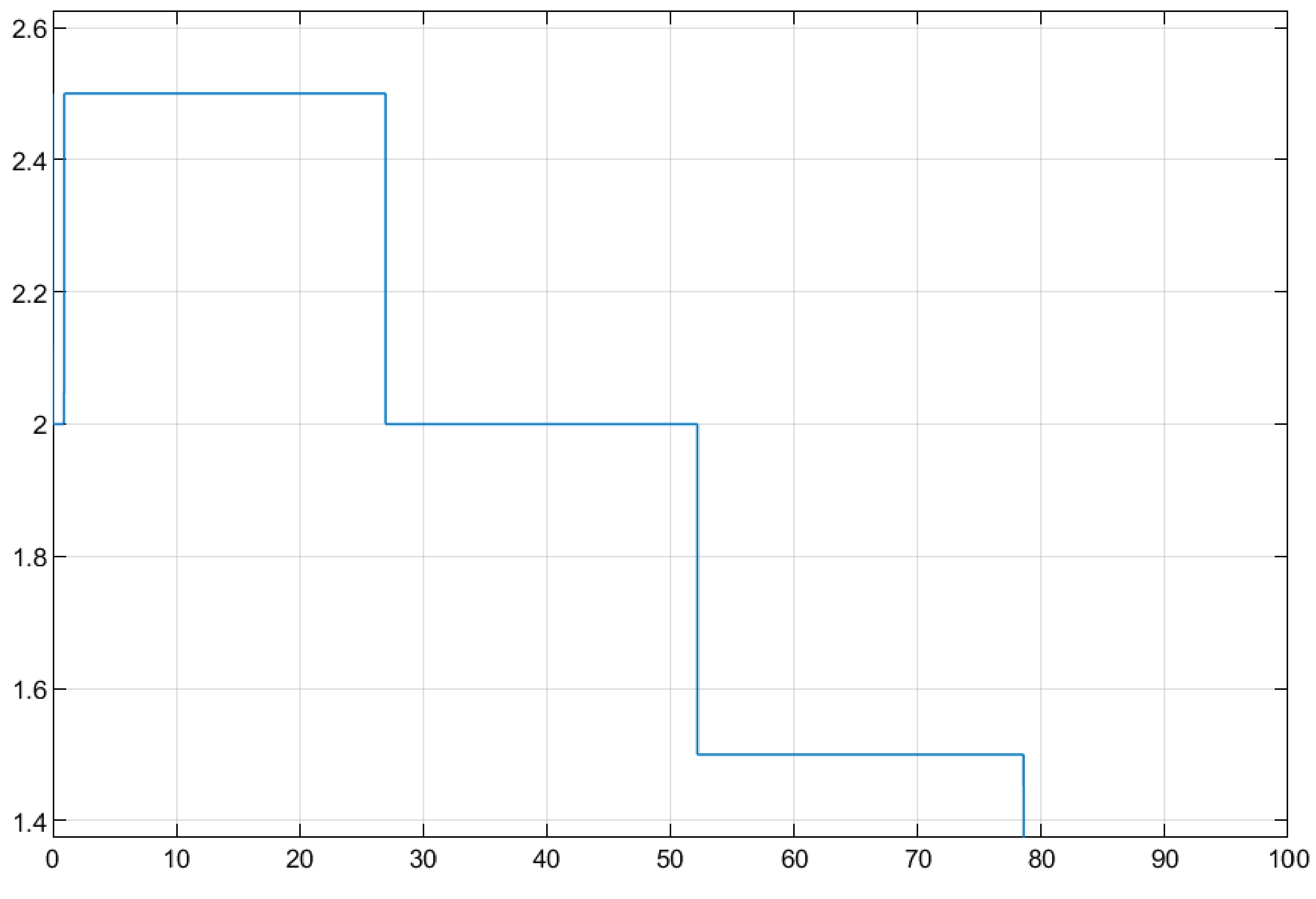


Estimated plot for K values

5. Implementing logic to stop the simulation after 3rd failure



Simulink Diagram



Estimated value of k

As it is observed, the simulation stops at 78.610 seconds i.e., almost 3.5 seconds after the last acceptable failure. Thus, the system works well and performs the task it was designed to do. There is a small initial error in the estimated value for k because the system is still adjusting to the inputs. The output response time and accuracy can further be improved by tuning the transfer functions further.