Kalman Filtering

This case study illustrates Kalman filter design and simulation.
Both steady-state and time-varying Kalman filters are considered.

Overview of the Case Study

This case study illustrates Kalman filter design and simulation. Both steady-state and time-varying Kalman filters are considered.

Consider a discrete plant with additive Gaussian noise w_n on the input $u[n]$:

$$x[n+1] = Ax[n] + B(u[n] + w[n])$$
$$y[n] = Cx[n].$$

The following matrices represent the dynamics of this plant.

```
A = [1.1269   -0.4940    0.1129;
     1.0000         0         0;
          0    1.0000         0];

B = [-0.3832;
      0.5919;
      0.5191];

C = [1 0 0];
```

Discrete Kalman Filter

The equations of the steady-state Kalman filter for this problem are given as follows.

- Measurement update:

$$\hat{x}[n|n] = \hat{x}[n|n-1] + M(y_v[n] - C\hat{x}[n|n-1])$$

- Time update:

$$\hat{x}[n+1|n] = A\hat{x}[n|n] + Bu[n]$$

In these equations:

- $\hat{x}[n|n-1]$ is the estimate of $x[n]$, given past measurements up to $y_v[n-1]$.
- $\hat{x}[n|n]$ is the updated estimate based on the last measurement $y_v[n]$.

Given the current estimate $\hat{x}[n|n]$, the time update predicts the state value at the next sample *n* + 1 (one-step-ahead predictor). The measurement update then adjusts this prediction based on the new measurement $y_v[n+1]$. The correction term is a function of the innovation, that is, the discrepancy between the measured and predicted values of $y[n+1]$. This discrepancy is given by:

$$y_v[n+1] - C\hat{x}[n+1|n]$$

The innovation gain M is chosen to minimize the steady-state covariance of the estimation error, given the noise covariances:

$$E\left(w[n]\,w[n]^T\right) = Q \qquad E\left(v[n]\,v[n]^T\right) = R \qquad N = E\left(w[n]\,v[n]^T\right) = 0$$

You can combine the time and measurement update equations into one state-space model, the Kalman filter:

$$\hat{x}[n+1|n] = A(I - MC)\hat{x}[n|n-1] + \begin{bmatrix} B & AM \end{bmatrix}\begin{bmatrix} u[n] \\ y_v[n] \end{bmatrix}$$
$$\hat{y}[n|n] = C(I - MC)\hat{x}[n|n-1] + CMy_v[n].$$

This filter generates an optimal estimate $\hat{y}[n|n]$ of y_n. Note that the filter state is $\hat{x}[n|n-1]$.

Steady-State Design

You can design the steady-state Kalman filter described above with the function `kalman`. First specify the plant model with the process noise:

$$\begin{aligned} x[n+1] &= Ax[n] + Bu[n] + Bw[n] \\ y[n] &= Cx[n] \end{aligned}$$

Here, the first expression is the state equation, and the second is the measurement equation.

The following command specifies this plant model. The sample time is set to -1, to mark the model as discrete without specifying a sample time.

A

```
Plant = ss(A,[B B],C,0,-1,'inputname',{'u' 'w'},'outputname','y');
```

Assuming that *Q* = *R* = 1, design the discrete Kalman filter.

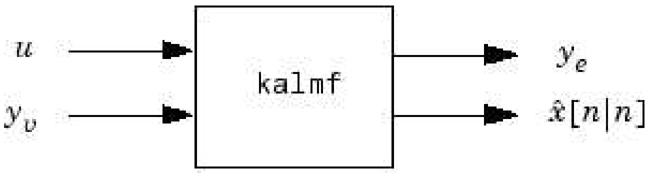
```
Q = 1;
R = 1;
[kalmf,L,P,M] = kalman(Plant,Q,R);
```

This command returns a state-space model `kalmf` of the filter, as well as the innovation gain `M`.

```
M
```

```
M =

    0.3798
    0.0817
   -0.2570
```

The inputs of `kalmf` are *u* and y_v, and. The outputs are the plant output and the state estimates, $y_e = \hat{y}[n|n]$ and $\hat{x}[n|n]$.



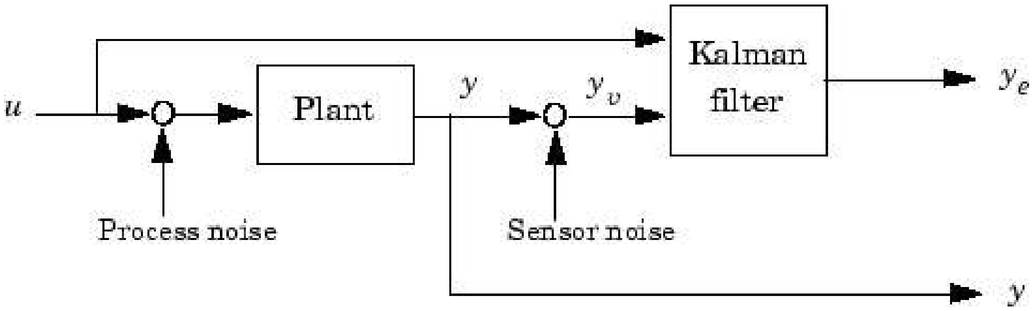
Kalman filter

Because you are interested in the output estimate y_e, select the first output of `kalmf` and discard the rest.

```
kalmf = kalmf(1,:);
```

To see how the filter works, generate some input data and random noise and compare the filtered response y_e with the true response *y*. You can either generate each response separately, or generate both together. To simulate each response separately, use `lsim` with the plant alone first, and then with the plant and filter hooked up together. The joint simulation alternative is detailed next.

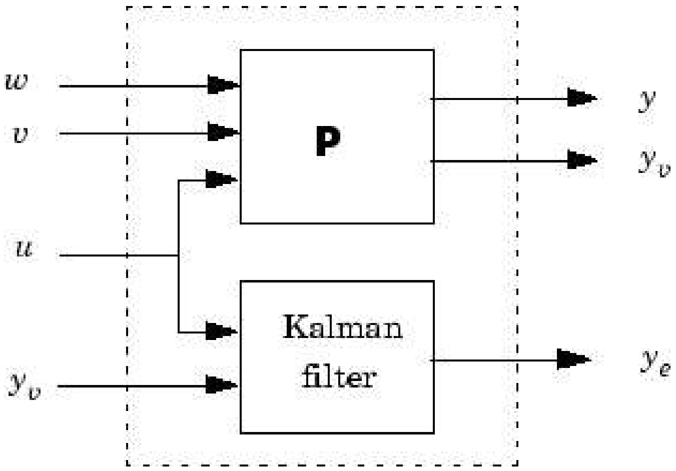
The block diagram below shows how to generate both true and filtered outputs.



You can construct a state-space model of this block diagram with the functions `parallel` and `feedback`. First build a complete plant model with *u*, *w*, *v* as inputs, and *y* and y_v (measurements) as outputs.

```
a = A;
b = [B B 0*B];
c = [C;C];
d = [0 0 0;0 0 1];
P = ss(a,b,c,d,-1,'inputname',{'u' 'w' 'v'},'outputname',{'y' 'yv'});
```

Then use `parallel` to form the parallel connection of the following illustration.




```
sys = parallel(P,kalmf,1,1,[],[]);
```

Finally, close the sensor loop by connecting the plant output y_v to filter input y_v with positive feedback.

```
SimModel = feedback(sys,1,4,2,1);   % Close loop around input #4 and output #2
SimModel = SimModel([1 3],[1 2 3]); % Delete yv from I/O list
```

The resulting simulation model has *w*, *v*, *u* as inputs, and *y* and y_e as outputs. View the `InputName` and `OutputName` properties to verify.

```
SimModel.InputName
```

```
ans =

  3×1 cell array

    'w'
    'v'
    'u'
```

```
SimModel.OutputName
```

```
ans =

  2×1 cell array

    'y'
    'y_e'
```

You are now ready to simulate the filter behavior. Generate a sinusoidal input *u* and process and measurement noise vectors *w* and *v*.

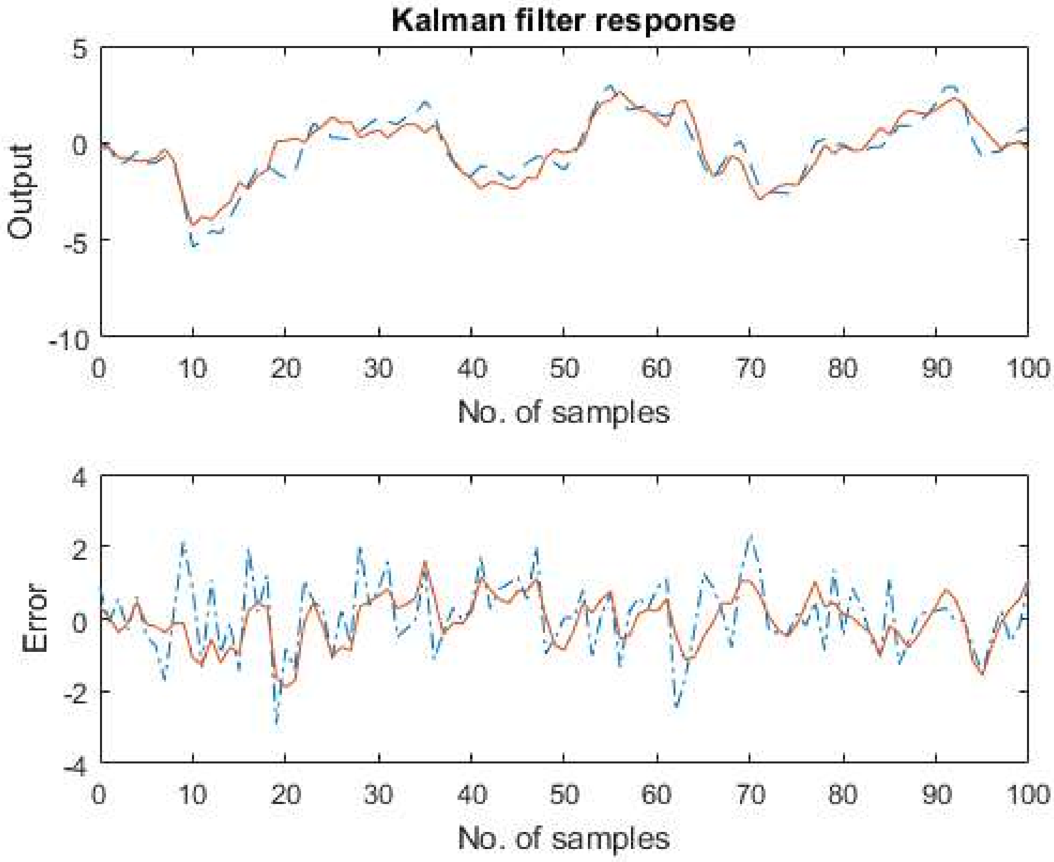
```
t = [0:100]';
u = sin(t/5);

n = length(t);
rng default
w = sqrt(Q)*randn(n,1);
v = sqrt(R)*randn(n,1);
```

Simulate the responses.

```
[out,x] = lsim(SimModel,[w,v,u]);

y = out(:,1);   % true response
ye = out(:,2);  % filtered response
yv = y + v;     % measured response
```

Compare the true and filtered responses graphically.

```
subplot(211), plot(t,y,'--',t,ye,'-'),
xlabel('No. of samples'), ylabel('Output')
title('Kalman filter response')
subplot(212), plot(t,y-yv,'-.',t,y-ye,'-'),
xlabel('No. of samples'), ylabel('Error')
```



The first plot shows the true response *y* (dashed line) and the filtered output y_e (solid line). The second plot compares the measurement error (dash-dot) with the estimation error (solid). This plot shows that the noise level has been significantly reduced. This is confirmed by calculating covariance errors. The error covariance before filtering (measurement error) is:

```
MeasErr = y-yv;
MeasErrCov = sum(MeasErr.*MeasErr)/length(MeasErr)
```

```
MeasErrCov =

    0.9992
```

The error covariance after filtering (estimation error) is reduced:

```
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr)
```

```
EstErrCov =

    0.4944
```

Time-Varying Kalman Filter

The time-varying Kalman filter is a generalization of the steady-state filter for time-varying systems or LTI systems with nonstationary noise covariance.

Consider the following plant state and measurement equations.

$$\begin{aligned} x[n+1] &= Ax[n] + Bu[n] + Gw[n] \\ y_v[n] &= Cx[n] + v[n]. \end{aligned}$$

The time-varying Kalman filter is given by the following recursions:

- Measurement update:

$$\begin{aligned}\hat{x}[n|n] &= \hat{x}[n|n-1] + M[n](y_v[n] - C\hat{x}[n|n-1]) \\ M[n] &= P[n|n-1]C^T(R[n] + CP[n|n-1]C^T)^{-1} \\ P[n|n] &= (I - M[n]C)P[n|n-1].\end{aligned}$$

- Time update:

$$\begin{aligned}\hat{x}[n+1|n] &= A\hat{x}[n|n] + Bu[n] \\ P[n+1|n] &= AP[n|n]A^T + GQ[n]G^T.\end{aligned}$$

Here, $\hat{x}[n|n-1]$ and $\hat{x}[n|n]$ are as described previously. Additionally:

$$\begin{aligned}&Q[n] = E(w[n]w[n]^T) \\ &R[n] = E(v[n]v[n]^T) \\ &P[n|n] = E(\{x[n] - x[n|n]\}\{x[n] - x[n|n]\}^T) \\ &P[n|n-1] = E(\{x[n] - x[n|n-1]\}\{x[n] - x[n|n-1]\}^T).\end{aligned}$$

For simplicity, the subscripts indicating the time dependence of the state-space matrices have been dropped.

Given initial conditions $x[1|0]$ and $P[1|0]$, you can iterate these equations to perform the filtering. You must update both the state estimates $x[n|.]$ and error covariance matrices $P[n|.]$ at each time sample.

Time-Varying Design

B

To implement these filter recursions, first genereate noisy output measurements. Use the process noise `w` and measurement noise `v` generated previously.

```
sys = ss(A,B,C,0,-1);
y = lsim(sys,u+w);
yv = y + v;
```

Assume the following initial conditions:

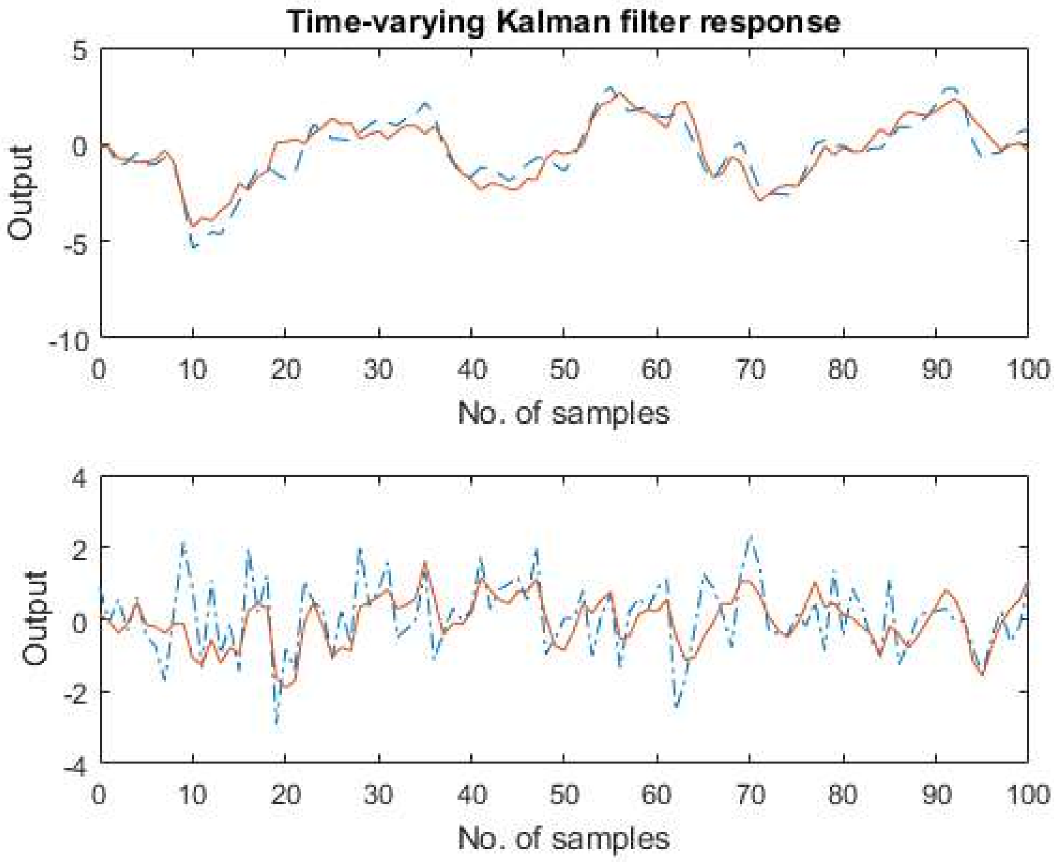
$$x[1|0] = 0, \quad P[1|0] = BQB^T$$

Implement the time-varying filter with a `for` loop.

```
P = B*Q*B';         % Initial error covariance
x = zeros(3,1);     % Initial condition on the state
ye = zeros(length(t),1);
ycov = zeros(length(t),1);

for i = 1:length(t)
  % Measurement update
  Mn = P*C'/(C*P*C'+R);
  x = x + Mn*(yv(i)-C*x);   % x[n|n]
  P = (eye(3)-Mn*C)*P;      % P[n|n]

  ye(i) = C*x;
  errcov(i) = C*P*C';

  % Time update
  x = A*x + B*u(i);        % x[n+1|n]
  P = A*P*A' + B*Q*B';     % P[n+1|n]
end
```

Compare the true and estimated output graphically.

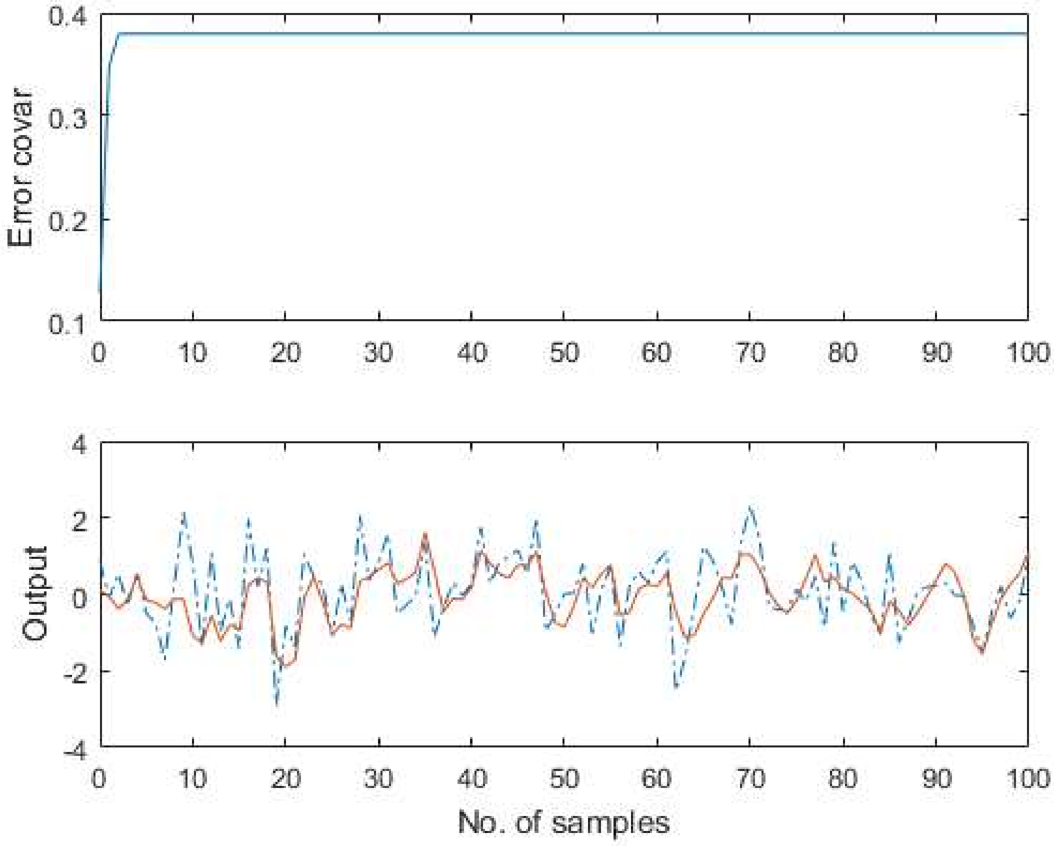
```
subplot(211), plot(t,y,'--',t,ye,'-')
title('Time-varying Kalman filter response')
xlabel('No. of samples'), ylabel('Output')
subplot(212), plot(t,y-yv,'-.',t,y-ye,'-')
xlabel('No. of samples'), ylabel('Output')
```



The first plot shows the true response *y* (dashed line) and the filtered response y_e (solid line). The second plot compares the measurement error (dash-dot) with the estimation error (solid).

The time-varying filter also estimates the covariance `errcov` of the estimation error $y - y_e$ at each sample. Plot it to see if your filter reached steady state (as you expect with stationary input noise).

```
subplot(211)
plot(t,errcov), ylabel('Error covar')
```



From this covariance plot, you can see that the output covariance did indeed reach a steady state in about five samples. From then on, your time-varying filter has the same performance as the steady-state version.

Compare with the estimation error covariance derived from the experimental data:

```
EstErr = y - ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr)
```

```
EstErrCov =

    0.4934
```

This value is smaller than the theoretical value `errcov` and close to the value obtained for the steady-state design.

Finally, note that the final value $M[n]$ and the steady-state value *M* of the innovation gain matrix coincide.

```
Mn
```

```
Mn =

    0.3798
    0.0817
   -0.2570
```

```
M
```

```
M =

    0.3798
    0.0817
   -0.2570
```

Bibliography

[1] Grimble, M.J., *Robust Industrial Control: Optimal Design Approach for Polynomial Systems*, Prentice Hall, 1994, p. 261 and pp. 443-456.