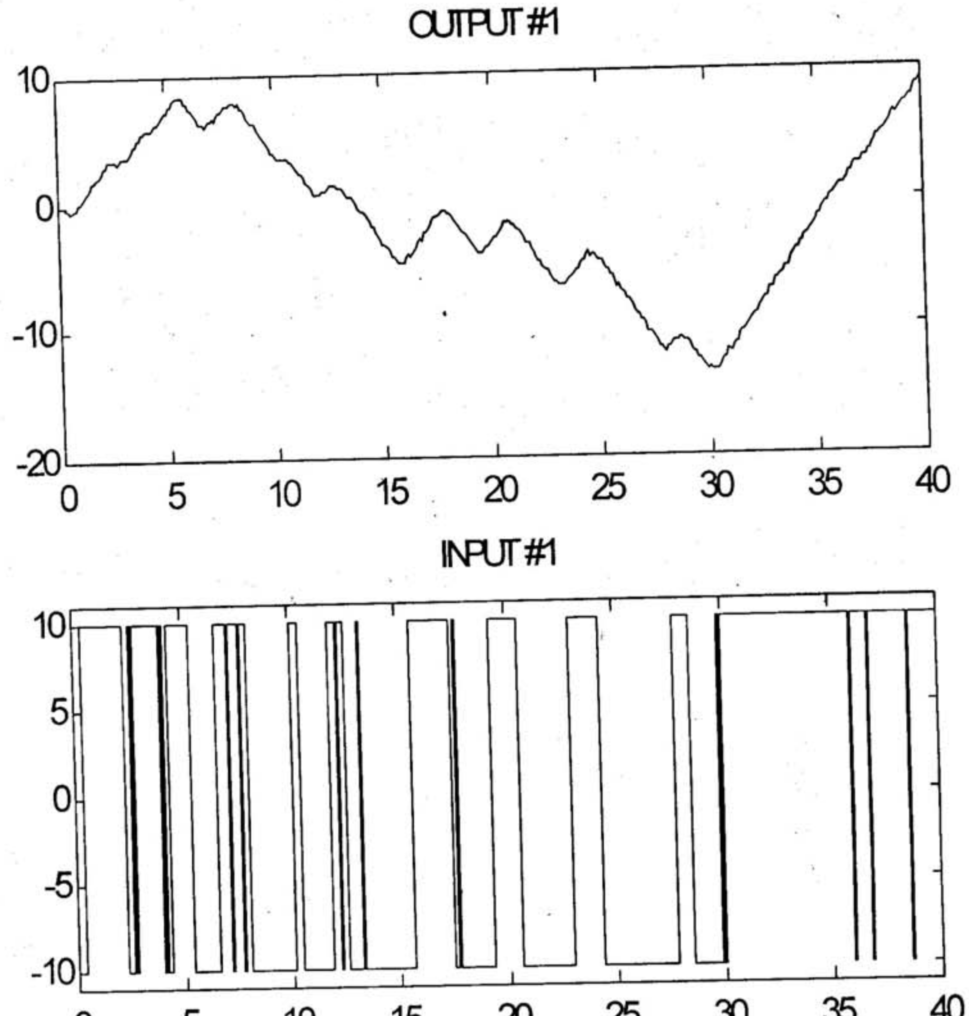
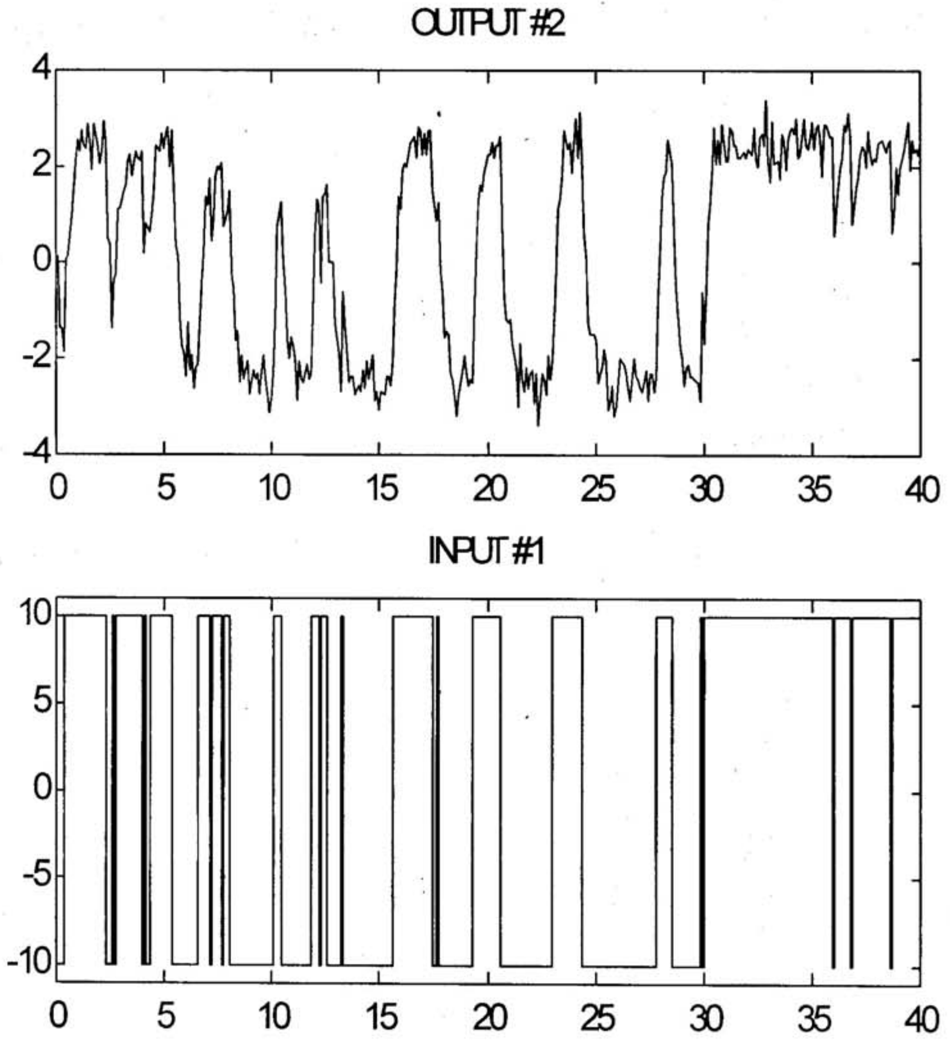
State Space Modeling Example

```
load dcmdata

%	The matrix y contains the two outputs: y1 is the angular position of
%	the motor shaft and y2 is the angular velocity. There are 400 data
%	points and the sampling interval is 0.1 seconds. The input is
%	contained in the vector u. It is the input voltage to the motor.
%	Press a key for plots of the input-output data.

z=[y u];

pause,idplot(z,[],0.1,2),pause
```

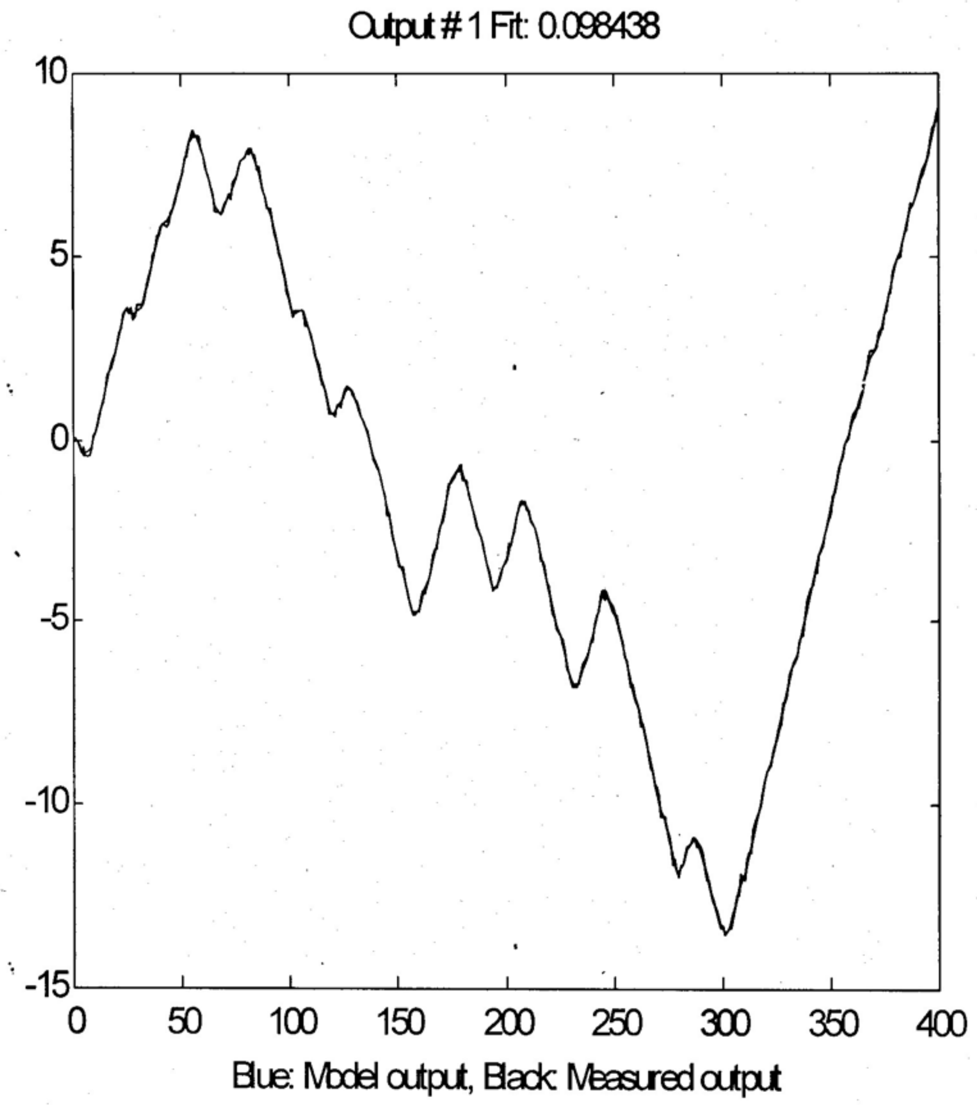




```
%      We shall build a model of the dc-motor. The dynamics of the motor is
%      well known. If we choose x1 as the angular position and x2 as the
%      angular velocity it is easy to set up a state-space model of the
%      following character neglecting disturbances:
%      (see page 84 in Ljung(1987):

%            |0    1 |       |0   |
% d/dt x =   |       |x  +   |    |u
%            |0  -th1|       |th2 |
%
%
%        |1  0|
%   y =  |    |x
%        |0  1|
%
%
%      The parameter th1 is here the inverse time-constant of the motor and
%       We shall build a model of the dc-motor. The dynamics of the motor is
%       well known. If we choose x1 as the angular position and x2 as the
%       angular velocity it is easy to set up a state-space model of the
%       following character neglecting disturbances:
%       (see page 84 in Ljung(1987):

%           | 0     1 |       | 0   |
% d/dt x =  |         | x  +  |     | u
%           | 0   -th1 |      | th2 |
%
%
%       | 1  0 |
%   y = |      | x
%       | 0  1 |
%
%
%       The parameter th1 is here the inverse time-constant of the motor and
%       th2 is such that th2/th1 is the static gain from input to the angular
%       velocity. (See Ljung(1987) for how th1 and th2 relate to the physical
%       parameters of the motor.)
%       We shall estimate these two parameters from the observed data. Strike
%       a key to continue.

pause

%       [redacted]
%
%       We first have to define the structure in terms of the general
%       description
%
%       d/dt x = A x + B u + K e
%
%       y = C x + D u + e
%
%       Strike a key to continue.

Pause

%
%       Any parameter to be estimated is entered as NaN (Not a Number).
%       Thus we have
A=[0 1 ; 0  -th1] ;
B=[0  th2] ;
C=[1 0 ; 0  1]
D=[0
K=[0 0
X0=[0
        %entered as parameters to be identified.

%       The model structure is now defined by

ms = modstruc(A,B,C,D,K,X0);pause     % Strike a key to continue

%       We shall produce an initial guess for the parameters. Let us guess
%     that the time constant is one second and that the static gain is 0.28.
%     This gives:

th_guess = [-1 0.28];

%     We now collect all this information into the "theta-format" by

dcmodel = ms2th(ms,'c',th_guess,[],0.1);

%     [redacted]denotes that the parametrization refers to a [redacted]
%     The last argument, 0.1,is the sampling interval for the collected data

%     The prediction error (maximum likelihood) estimate of the parameters
%     is now computed by (Press a key to continue)

pause, dcmodel = pem(z,dcmodel,'trace');
0
   ITERATION # 1
Current loss: 0.18012  Previous loss: 1.5092
Current th  prev. th   gn-dir.
  -1.7485  -1.0000  -0.7485
   0.4550   0.2800   0.1750

Norm of gn-vector: 0.76868
Expected improvement: 173.3237 %
Achieved improvement: 88.0655 %
0
   ITERATION # 2
Current loss: 0.0085866  Previous loss: 0.18012
Current th  prev. th   gn-dir.
  -2.8479  -1.7485  -1.0994
   0.7186   0.4550   0.2636

Norm of gn-vector: 1.1306
Expected improvement: 155.6873 %
Achieved improvement: 95.2327 %
0
   ITERATION # 3
Current loss: 0.0012423  Previous loss: 0.0085866
Current th  prev. th   gn-dir.
  -3.7279  -2.8479  -0.8800
   0.9325   0.7186   0.2139

Norm of gn-vector: 0.90564
Expected improvement: 118.1639 %
Achieved improvement: 85.5326 %
0
   ITERATION # 4
Current loss: 0.0010614  Previous loss: 0.0012423
Current th  prev. th   gn-dir.
  -3.9952  -3.7279  -0.2672
   0.9978   0.9325   0.0654

Norm of gn-vector: 0.27512
Expected improvement: 15.0203 %
Achieved improvement: 14.5559 %
```

```
0
   ITERATION # 5
Current loss: 0.0010608   Previous loss: 0.0010614
Current th  prev. th   gn-dir.
  -4.0129  -3.9952  -0.0177
   1.0022   0.9978   0.0044

Norm of gn-vector: 0.018254
Expected improvement: 0.057146 %
Achieved improvement: 0.057647 %
0
   ITERATION # 6
Current loss: 0.0010608   Previous loss: 0.0010608
Current th  prev. th   gn-dir.
  -4.0131  -4.0129  -0.0002
   1.0023   1.0022   0.0001

Norm of gn-vector: 0.000243
Expected improvement: 9.9615e-006 %
Achieved improvement: 1.0061e-005 %

%      We can display the result by the 'present' command.
%      The imaginary parts denote standard deviation.
%      Strike a key to continue.

pause, present(dcmodel), pause

This matrix was created by the command PEM      on 9/22 1999 at 16:4
Loss fcn: 0.0010608   Akaike's FPE: 0.0010715 Continuous time model estimated using sampling interval
0.1
The state-space matrices with standard deviations given as imaginary parts are

a =

      0          1.0000
      0         -4.0131 + 0.0374i


b =

      0
  1.0023 + 0.0092i


c =

    1    0
    0    1

d =

    0
    0

k =
    0    0
    0    0


x0 =

    0
    0


lambda =

   0.0097  -0.0004
  -0.0004   0.1095

%      The estimated values of the parameters are quite close to those used
%      when the data were simulated (-4 and 1).

%      To evaluate the model's quality we can simulate the model with the
%      actual input by and compare it with the actual output.
%      Strike a key to continue.

pause, compare(z,dcmodel); pause
```

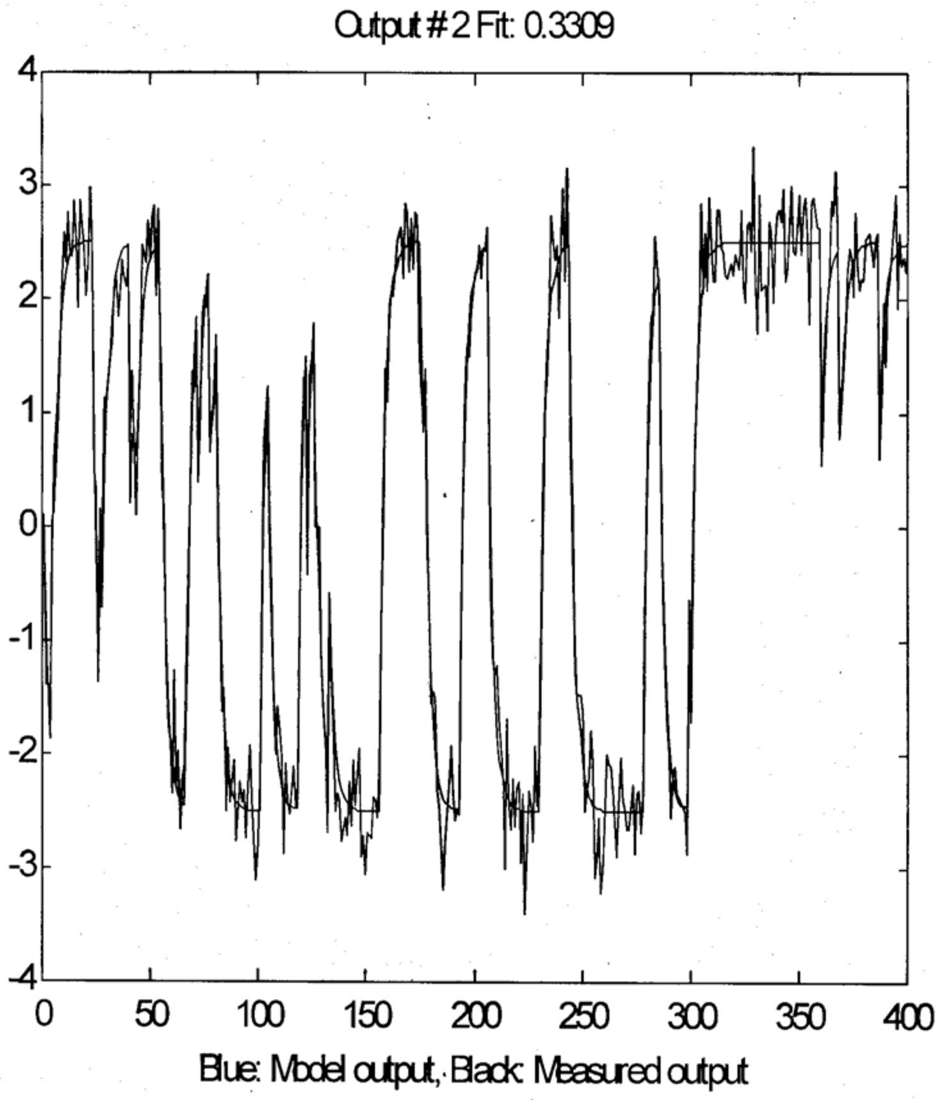
```
%      We can now, for example plot zeros and poles and their uncer-
%      tainty regions. We will draw the regions corresponding to 10 standard
%     To evaluate the model's quality we can simulate the model with the
%     actual input by and compare it with the actual output.
%     Strike a key to continue.

pause, compare(z,dcmodel); pause
```



```
%     We can now, for example plot zeros and poles and their uncer-
%     tainty regions. We will draw the regions corresponding to 10 standard
%     deviations, since the model is quite accurate. Note that the pole at
%     the origin is absolutely certain, since it is part of the model
%     structure; the integrator from angular velocity to position.
%     Strike a key to continue.
%      deviations, since the model is quite accurate. Note that the pole at
%      the origin is absolutely certain, since it is part of the model
%      structure; the integrator from angular velocity to position.
%      Strike a key to continue.

pause, zpplot(th2zp(dcmodel),10), pause
```

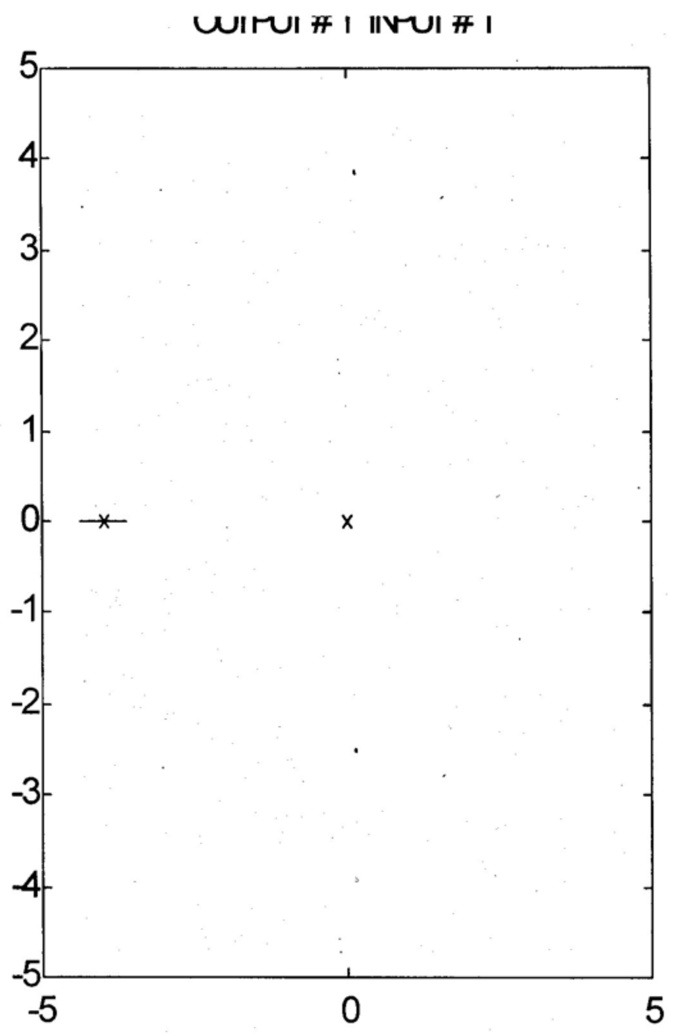
```
%      2. COUPLED PARAMETERS
%
%      Suppose that we accurately know the static gain of the dc-motor (from
%      input voltage to angular velocity, e.g. from a previous step-response
%      experiment. If the static gain is G, and the time constant of the
%      motor is t, then the state-space model becomes
%
%              |0    1|     | 0  |
%  d/dt x = |          |x + |      | u
%              |0  -1/t|     | G/t |
%
%          |1   0|
%   y  = |       | x
%          |0   1|
%

pause % Press a key to continue
```

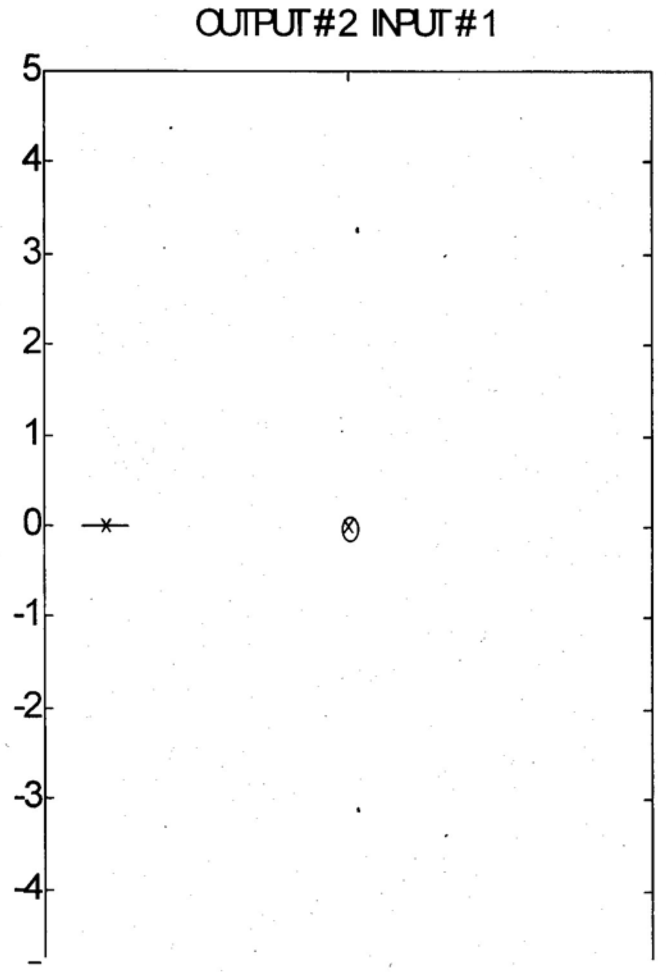
```
pause, zpplot(th2zp(dcmodel),10), pause
```



```
%      2. COUPLED PARAMETERS
%
%      Suppose that we accurately know the static gain of the dc-motor (from
%      input voltage to angular velocity, e.g. from a previous step-response
%      experiment. If the static gain is G, and the time constant of the
%      motor is t, then the state-space model becomes
%
%              |0    1|     | 0  |
%   d/dt x =   |      |x +  |    | u
%              |0  -1/t|    | G/t|
%
%              |1  0|
%     y    =   |    | x
%              |0  1|
```

6'


OUTPUT#1 INPUT#1
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
0
5


OUTPUT#2 INPUT#1
5
4
3
2
1
0
-1
-2
-3
-4

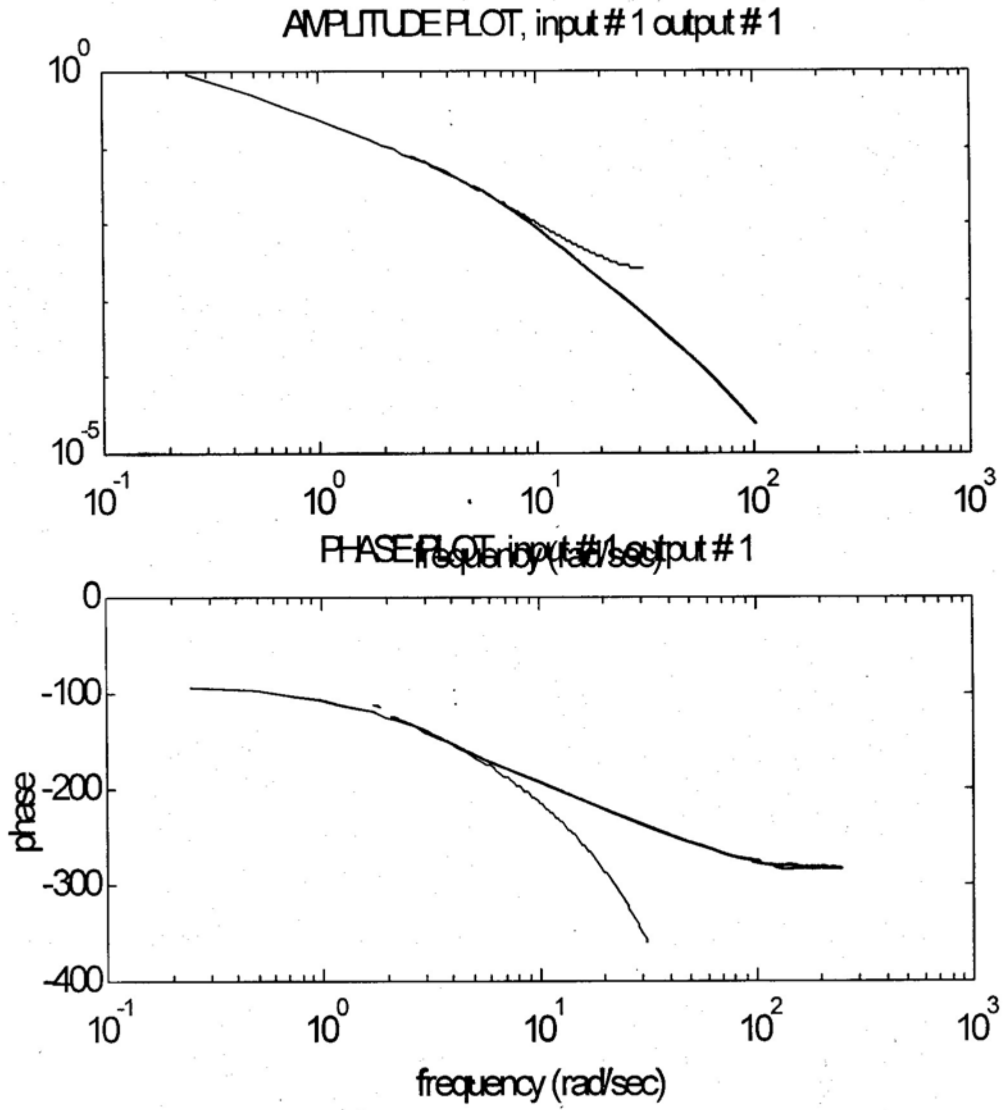
```
pause % Press a key to continue

%     With G known, there is a dependence between the entries in the
%     different matrices. In order to describe that, the earlier used way
%     with "NaN" will not be sufficient. We thus have to write an m-file
%     which produces the A,B,C,D,K and X0 matrices as outputs, for each
%     given parameter vector as input. It also takes auxiliary arguments as
%     inputs, so that the user can change certain things in the model
%     structure, without having to edit the m-file. In this case we let the
%     known static gain G be entered as such an argument. The m-file that
%     has been written has the name 'motor'. Press a key for a listing.

pause, type motor

function [A,B,C,D,K,X0] = motor(par,ts,aux)
%MOTOR  Describes the dc-motor with time-constant t (= par) and
%  known static gain G. The sampling interval is ts. The conversion to
%  a sampled model is inhibited if ts is entered as a negative number.
%  This is to allow for desirable features in the 'present'and 'eta2ss'
%  commands.

%  L. Ljung
%  Copyright (c) 1986-97 by The MathWorks, Inc.
%  $Revision: 2.2 $  $Date: 1997/04/16 20:02:22 $
```

```
t = par;
G=aux(1);
A=[0 1;0 -1/t];
B=[0;G/t];
C=eye(2);
D=[0;0];
K=zeros(2,2);
X0=[0;0];
if ts>0 % Sample the model with sampling interval ts
s = expm([[A B]*ts; zeros(1,3)]);
A = s(1:2,1:2);
B = s(1:2,3);
end

pause % Press a key to continue.

%     We now create a "THETA-structure" corresponding to this model
%     structure:The assumed time constant will be

tc_guess = 1;

%     We also give the value 0.25 to the auxiliary variable G (gain)
%     and sampling interval

aux = 0.25;

dcmm = mf2th('motor','c',tc_guess,aux,[],0.1);

pause   % Press a key to continue

%     The time constant is now estimated by

dcmm = pem([y u],dcmm,'trace');
0
   ITERATION # 1
Current loss: 0.014936   Previous loss: 1.5071
Current th  prev. th   gn-dir.
   0.1406    1.0000   -0.8594

Norm of gn-vector: 0.85943
Expected improvement: 176.1163 %
Achieved improvement: 99.0089 %
0
   ITERATION # 2
Current loss: 0.0012639   Previous loss: 0.014936
Current th  prev. th   gn-dir.
   0.2300    0.1406    0.0895

Norm of gn-vector: 0.089455
Expected improvement: 130.5844 %
Achieved improvement: 91.5382 %
0
   ITERATION # 3
Current loss: 0.0010652   Previous loss: 0.0012639
Current th  prev. th   gn-dir.
```

```
   0.2490    0.2300    0.0190

Norm of gn-vector: 0.018952
Expected improvement: 15.6655 %
Achieved improvement: 15.7151 %
0
   ITERATION # 4
Current loss: 0.001065   Previous loss: 0.0010652
Current th  prev. th   gn-dir.
   0.2496    0.2490    0.0007

Norm of gn-vector: 0.00066267
Expected improvement: 0.020871 %
Achieved improvement: 0.021129 %
0
   ITERATION # 5
Current loss: 0.001065   Previous loss: 0.001065
Current th  prev. th   gn-dir.
   0.2496    0.2496    0.0000

Norm of gn-vector: 8.2765e-006
Expected improvement: 3.2531e-006 %
Achieved improvement: 3.2913e-006 %

%     We have thus now estimated the time constant of the motor directly.
%     Its value is in good agreement with the previous estimate.
%     Strike a key to continue.

pause, present(dcmm),  pause

This matrix was created by the command PEM      on 9/22 1999 at 16:31
Loss fcn: 0.001065   Akaike's FPE: 0.0010703 Continuous time model estimated using sampling interval
0.1
The state-space matrices with standard deviations given as imaginary parts are

a =
        0          1.0000
        0         -4.0057 + 0.0368i


b =
        0
   1.0014 + 0.0092i


c =
     1    0
     0    1

d =
     0
     0

k =

     0    0
```

```
    0    0


x0 =
    0
    0

lambda =
   0.0097   -0.0004
  -0.0004    0.1095

%      With this model we can now proceed to test various aspects as before.
%      The syntax of all the commands is identical to the previous case.

%      5. MULTIVARIATE ARX-MODELS.
%
%      The state-space part of the toolbox also handles multivariate (several
%      outputs) ARX models. By a multivariate ARX-model we mean the
%      following: Strike a key to continue.

pause

%      A(q) y(t) = B(q) u(t) + e(t)
%
%      Here A(q) is a ny | ny matrix whose entries are polynomials in the
%      delay operator 1/q. The k-l element is denoted by a_kl(q):
%
%                       -1 →           -nakl →
%      a_kl(q) = 1 + a-1 q    + .... + a-nakl q
%
%      It is thus a polynomial in 1/q of degree nakl.

%      Similarly B(q) is a ny | nu matrix, whose kj-element is

%                    -nkkj       -nkkj-1              -nkkj-nbkj
%      b_kj(q) = b-0 q     +  b-1 q        + ... + b-nbkj q

%      There is thus a delay of nkkj from input number j to output number k

%      The most common way to create those would be to use the ARX-command.
%      The orders are specified as
%      nn = [na nb nk]
%      with na being a ny | ny matrix whose kj-entry is nakj; nb and nk are
%      defined similarly.  Strike a key to continue.

pause

%      Let's test some ARX-models on the dc-data. First we could simply build
%      a general second order model:

na = [ 2 2; 2 2];
nb = [2; 2];
nk = [1;1];

dcarx1 = arx([y u],[na nb nk]);
```

```
%     The result, dcarx1, is stored in the THETA-format, and all previous
%     commands apply. We could for example explicitly determine the
%     ARX-polynomials by

[A,B] = th2arx(dcarx1);

%     Strike a key to continue.

pause, A, B, pause

A =
   1.0000        0  -0.5545  -0.0355  -0.4454  -0.0640
        0   1.0000   0.0185  -0.2005  -0.0194  -0.2924

B =
        0   0.0042   0.0066
        0   0.0864   0.0388
%     We could also test a structure, where we know that y1 is obtained by
%     filtering y2 through a first order filter. (The angle is the integral
%     of the angular velocity). We could then also postulate a first order
%     dynamics from input to output number 2:

na = [1 1; 0 1];
nb = [0 ; 1];
nk = [1 ; 1];

dcarx2 = arx(z,[na nb nk]);
dcarx2 = sett(dcarx2,0.1);   % Setting the sampling interval to 0.1 seconds
[Am, Bm] = th2arx(dcarx2);

Am, Bm

Am =
   1.0000        0  -0.9992  -0.0960
        0   1.0000        0  -0.6254

Bm =
        0        0
        0   0.0897

%     Strike a key to continue

pause

%     Finally, we could compare the bodeplots obtained from the input to
%     output one for the different models by

g1 = th2ff(dcmodel); % The first calculated model
g2 = th2ff(dcmm);    % The second model (known dc-gain)
g3 = th2ff(dcarx2);  % The second arx-model

pause,bodeplot([g1 g2 g3]),pause

%     The two first models are in more or less exact agreement. The
%     arx-models are not so good, due to the bias caused by the non-white
%     equation error noise. (We had white measurement noise in the
%      simulations).
pause
echo off
```


AMPLITUDE PLOT, input # 1 output # 1
10^0
10^{-5}
10^{-1}
10^0
10^1
10^2
10^3
frequency (rad/sec)
PHASE PLOT, input # 1 output # 1
0
-100
-200
-300
-400
phase
10^{-1}
10^0
10^1
10^2
10^3
frequency (rad/sec)

Optimal State-Feedback Gain

In LQG control, the regulation performance is measured by a quadratic performance criterion of the form

$$J(u) = \int_0^\infty \{x^T Qx + 2x^T Nu + u^T Ru\}\, dt$$

The weighting matrices Q, N, R are user-specified and define the trade-off between regulation performance (how fast $x(t)$ goes to zero) and control effort.

The first design step seeks a state-feedback law $u = -Kx$ that minimizes the cost function $J(u)$. The minimizing gain matrix K is obtained by solving an algebraic Riccati equation. This gain is called the *LQ-optimal* gain.

Kalman State Estimator

As for pole placement, the LQ-optimal state feedback $u = -Kx$ is not implementable without full state measurement. However, we can derive a state estimate $\hat{x}$ such that $u = -K\hat{x}$ remains optimal for the output-feedback problem. This state estimate is generated by the Kalman filter

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y_v - C\hat{x} - Du)$$

with inputs u (controls) and y_v (measurements). The noise covariance data

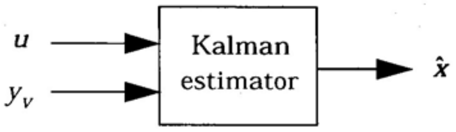
$$E(ww^T) = Q_n, \qquad E(vv^T) = R_n, \qquad E(wv^T) = N_n$$

determines the Kalman gain L through an algebraic Riccati equation.

The Kalman filter is an optimal estimator when dealing with Gaussian white noise. Specifically, it minimizes the asymptotic covariance

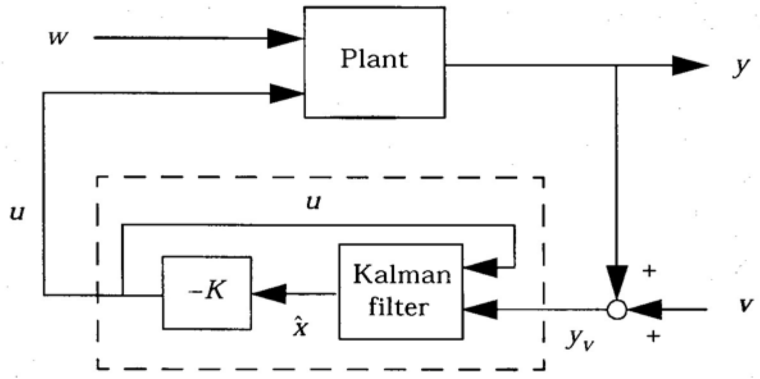
$$\lim_{t \to \infty} E((x - \hat{x})(x - \hat{x})^T)$$

of the estimation error $x - \hat{x}$.



LQG Regulator

To form the LQG regulator, simply connect the Kalman filter and LQ-optimal gain K as shown below:



LQG regulator

This regulator has state-space equations:

$$\dot{\hat{x}} = [A - LC - (B - LD)K]\,\hat{x} + Ly_v$$

$$u = -K\hat{x}$$

14

Kalman Filtering

This final case study illustrates the use of the Control System Toolbox for Kalman filter design and simulation. Both steady-state and time-varying Kalman filters are considered.

Consider the discrete plant

$$\begin{aligned} x[n+1] &= Ax[n] + B(u[n] + w[n]) \\ y[n] &= Cx[n] \end{aligned}$$

with additive Gaussian noise $w[n]$ on the input $u[n]$ and data

```
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];
```

Our goal is to design a Kalman filter that estimates the output $y[n]$ given the inputs $u[n]$ and the noisy output measurements

$$y_v[n] = Cx[n] + v[n]$$

where $v[n]$ is some Gaussian white noise.

Discrete Kalman Filter

The equations of the steady-state Kalman filter for this problem are:

Measurement update:

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

Time update:

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

۱۵

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*, i.e., the discrepancy

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

between the measured and predicted values of $y[n+1]$. The innovation gain M is chosen to minimize the steady-state covariance of the estimation error given the noise covariances

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

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

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

$$\hat{y}[n|n] = C(I-MC)\,\hat{x}[n|n-1] + CM\,y_v[n]$$

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:

$$x[n+1] = Ax[n] + Bu[n] + Bw[n] \qquad \text{(state equation)}$$
$$y[n] = Cx[n] \qquad \text{(measurement equation)}$$

This is done by

```
% Note: set sample time to -1 to mark model as discrete
Plant = ss(A,[B B],C,0,-1,'inputname',{'u' 'w'},...
                        'outputname','y');
```

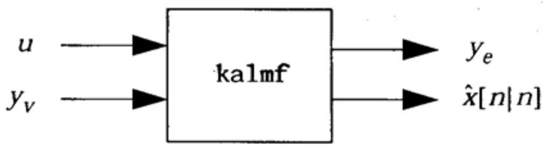
Assuming that $Q = R = 1$, you can now design the discrete Kalman filter by

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

This returns a state-space model kalmf of the filter as well as the innovation gain

```
» M

M =
   3.7980e-01
   8.1732e-02
  -2.5704e-01
```

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



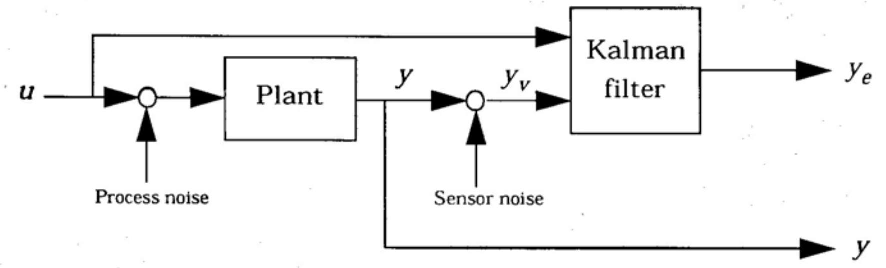
Kalman filter

Because you are interested in the output estimate y_e, keep only the first output of `kalmf`:

```
» kalmf = kalmf(1,:);
» kalmf

a =
                        x1_e          x2_e          x3_e
          x1_e       0.76830      -0.49400       0.11290
          x2_e       0.62020             0             0
          x3_e      -0.08173       1.00000             0

b =
                           u             y
          x1_e      -0.38320       0.35860
          x2_e       0.59190       0.37980
          x3_e       0.51910       0.08173

c =
                        x1_e          x2_e          x3_e
           y_e       0.62020             0             0

d =
                           u             y
           y_e             0       0.37980

Sampling time: unspecified
Discrete-time system.
```

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.

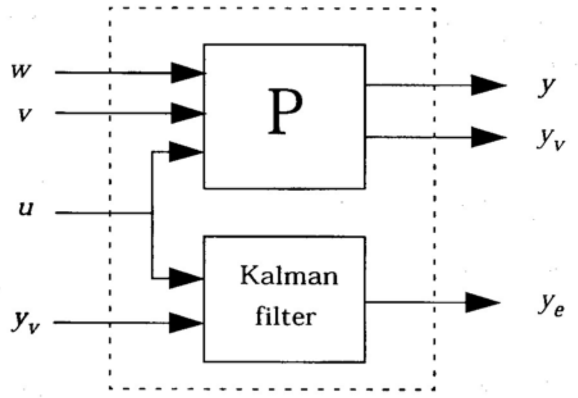
The block diagram below shows how to generate both true and filtered outputs.



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 following parallel connection:



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

18'

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

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

The resulting simulation model has w, v, u as inputs and y, y_e as outputs:

```
» SimModel.input

ans =
    'w'
    'v'
    'u'

» SimModel.output

ans =
    '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:

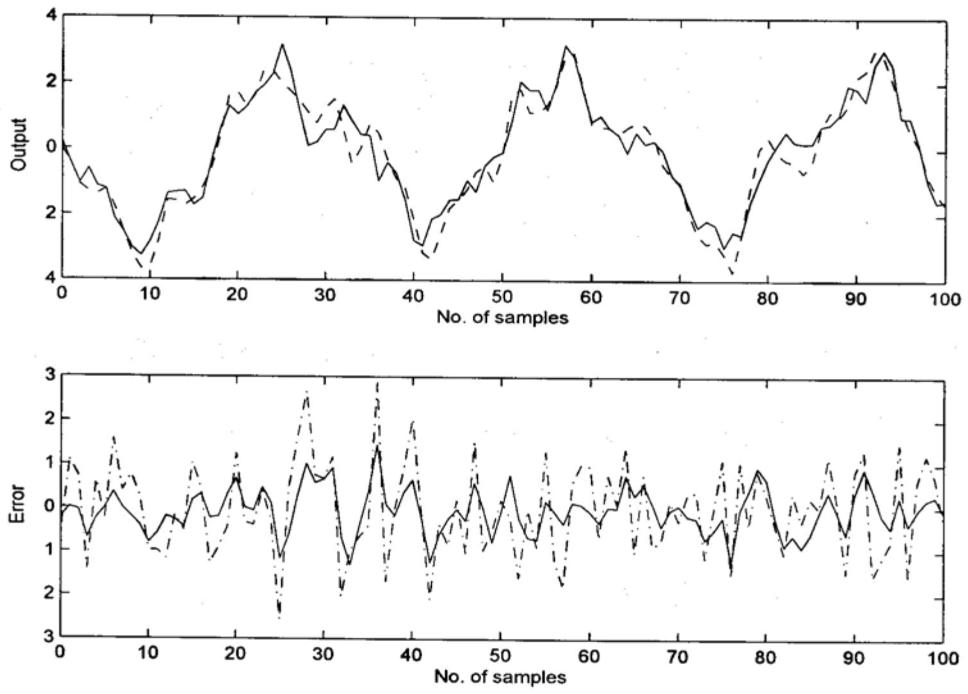
```
t = [0:100]';
u = sin(t/5);

n = length(t)
randn('seed',0)
w = sqrt(Q)*randn(n,1);
v = sqrt(R)*randn(n,1);
```

Now simulate with `lsim`:

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

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

and compare the true and filtered responses graphically:

```
subplot(211), plot(t,y,'--',t,ye,'-'),
xlabel('No. of samples'), ylabel('Output')
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 the following error covariance computations:

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

The error covariance before filtering (measurement error) is

```
» MeasErrCov

MeasErrCov =
    1.1138
```

while the error covariance after filtering (estimation error) is only

```
» EstErrCov

EstErrCov =
    0.2722
```

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 non-stationary noise covariance. Given the 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 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:

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

with $\hat{x}[n|n-1]$ and $\hat{x}[n|n]$ as defined on p. 6-51, and the notation

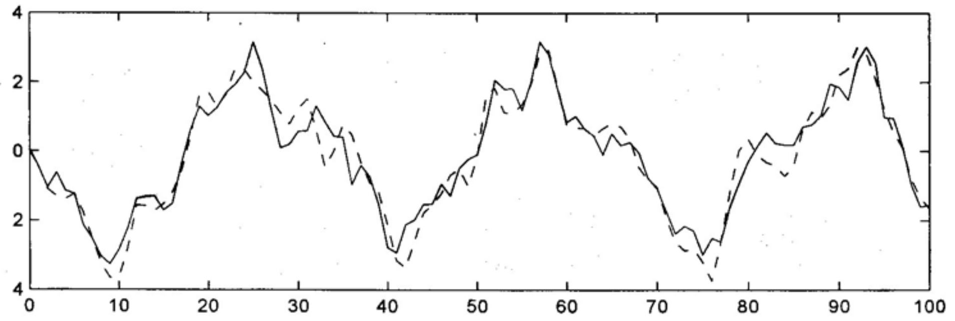
$$\begin{aligned} Q[n] &= E(w[n]w[n]^T) \\ R[n] &= E(v[n]v[n]^T) \\ P[n|n] &= E(\{x[n] - x[n|n]\}\{x[n] - x[n|n]\}^T) \\ P[n|n-1] &= E(\{x[n] - x[n|n-1]\}\{x[n] - x[n|n-1]\}^T) \end{aligned}$$

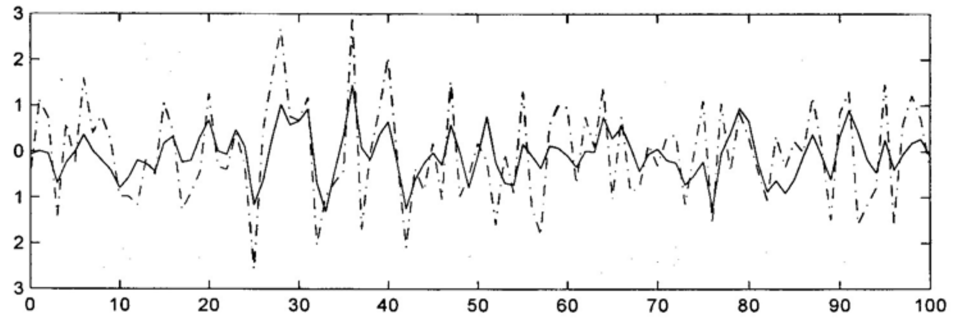
For simplicity, we have dropped the subscripts indicating the time dependence of the state-space matrices.

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

You can now compare the true and estimated output graphically:

```
subplot(211), plot(t,y,'--',t,ye,'-')
subplot(212), plot(t,y-yv,'-.',t,y-ye,'-')
```





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).

Time-Varying Design

Although the Control System Toolbox does not offer specific commands to perform time-varying Kalman filtering, it is easy to implement the filter recursions in MATLAB. This section shows how to do this for the stationary plant considered above.

First generate noisy output measurements

```
% Use process noise w and meas. noise v generated above
sys = ss(A,B,C,0,-1);
y = lsim(sys,u+w);      % w = process noise
yv = y + v;             % v = measurement noise
```

Given the initial conditions

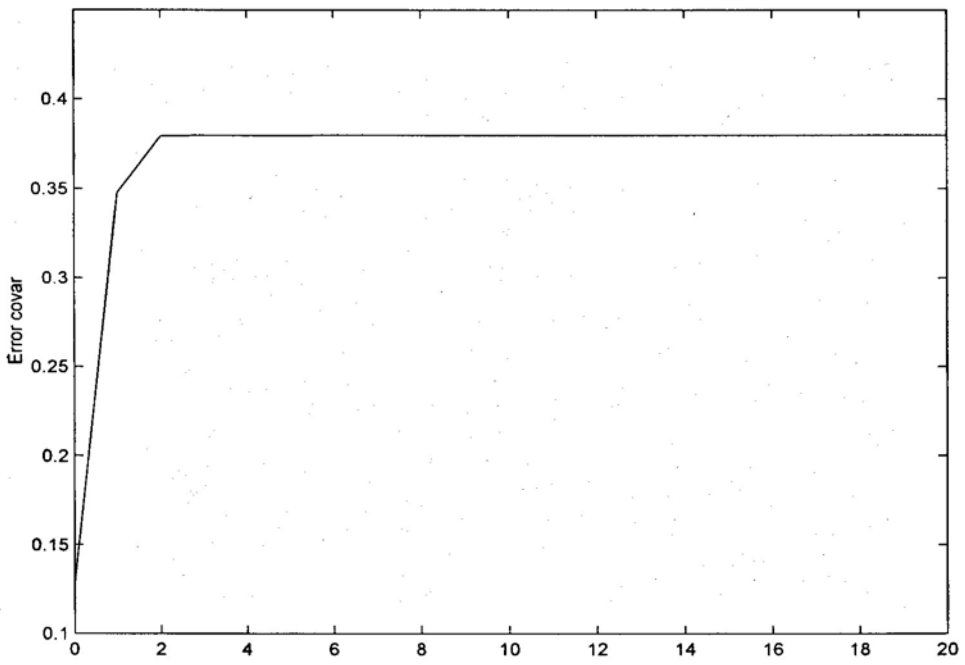
$$x[1|0] = 0 , \qquad P[1|0] = BQB^T$$

you can implement the time-varying filter with the following 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
```

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):

```
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 =
    0.2718
```

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, M

Mn =
    0.3798
    0.0817
   -0.2570

M =
    0.3798
    0.0817
   -0.2570
```