Estimation of Randomly Occurring Deterministic Disturbances

Guide to using the MATLAB code in this directory

Author: Bill Tubbs

Date: December 2022

Main files used in this tutorial:

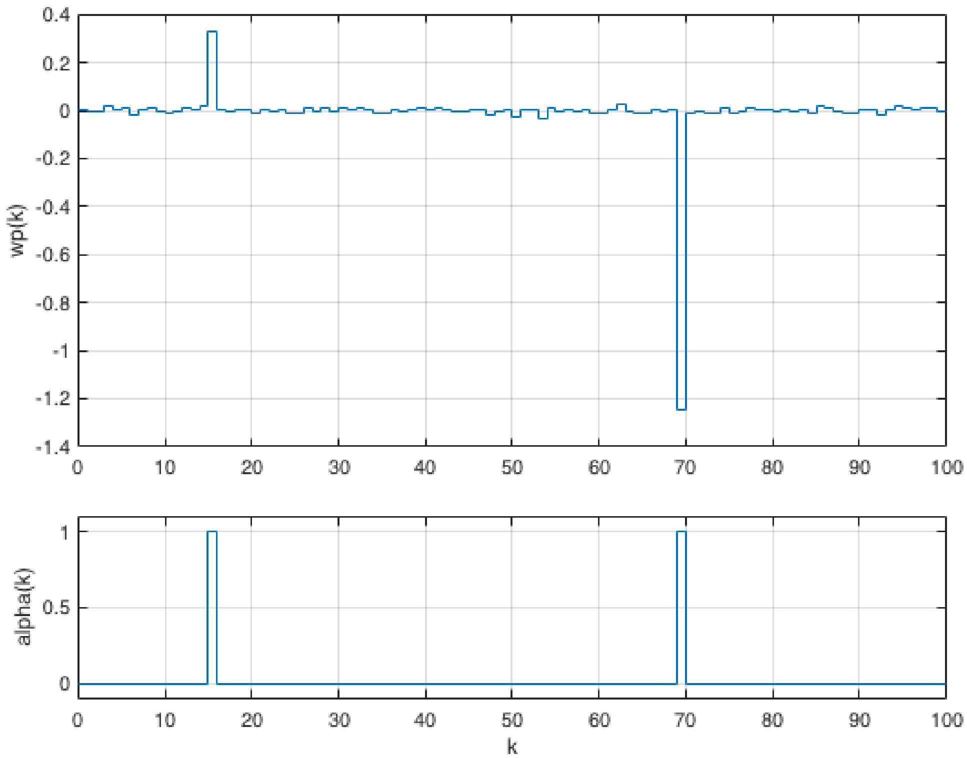
- KalmanFilter.m
- MKFObserverSF_RODD.m
- MKFObserverSP_RODD.m
- sample_random_shocks.m
- sys_rodin_step.m

```
clear all
% Set default parameters for plots
set(0, 'DefaultTextInterpreter', 'none')
set(0, 'DefaultLegendInterpreter', 'none')
set(0, 'DefaultAxesTickLabelInterpreter', 'none')
```

1. Generating randomly-occurring shocks

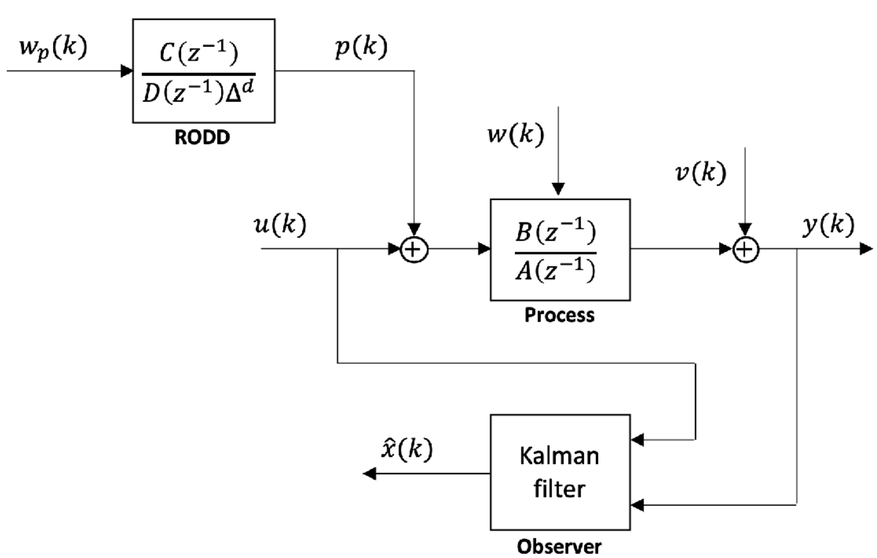
Generate a sample sequence of the random variable $w_{p,i}(k)$ described in Robertson et al. (1995).

```
% Reset random number generator
seed = 22;
rng(seed)

% Sequence length
nT = 100;

% RODD random variable parameters
epsilon = 0.01;
sigma_w = {[0.01; 1]};

% Generate random shock sequence
[Wp, alpha] = sample_random_shocks(nT+1, epsilon, sigma_w{1}(2), sigma_w{1}(1));

figure(1)
subplot(3,1,[1 2])
stairs(0:nT,Wp); grid on
ylabel('wp(k)')
subplot(3,1,3)
stairs(0:nT, alpha); grid on
ylim([-0.1 1.1])
xlabel('k')
ylabel('alpha(k)')
```



2. First order SISO system with one input disturbance

Consider the following linear dynamic system which consists of a process transfer function, an additive input disturbance, a measurement noise at the output, and a Kalman filter:



In this example, we will use a discrete system model defined in the file:

- `sys_rodin_step.m`

```
% Import system
sys_rodin_step
% Process transfer function
Gd
```

```
Gd =

    0.3
  -------
  z - 0.7

Sample time: 0.5 seconds
Discrete-time transfer function.
```

```
% RODD transfer function
HDd
```

```
HDd =

    1
  -----
  z - 1

Sample time: 0.5 seconds
Discrete-time transfer function.
```

The state-space representation of the augmented system is included in `sys_rodin_step.m`.

```
A, B, C, D, Ts
```

```
A = 2×2
    0.7000    1.0000
         0    1.0000
B = 2×2
     1     0
     0     1
C = 1×2
    0.3000         0
D = 1×2
     0     0
Ts = 0.5000
```

```
Gpss
```

```
Gpss =

  A = 
        x1   x2
   x1  0.7    1
   x2    0    1

  B = 
       u1  u2
   x1   1   0
   x2   0   1

  C = 
        x1   x2
   y1  0.3    0

  D = 
       u1  u2
```

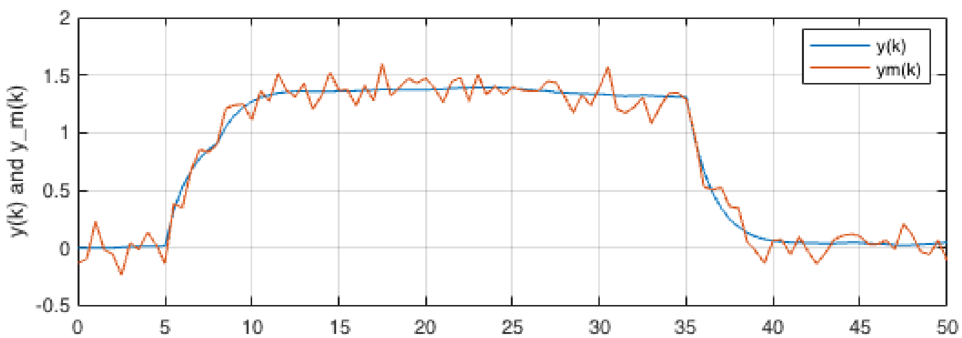
```
    y1   0   0

  Sample time: 0.5 seconds
  Discrete-time state-space model.
```

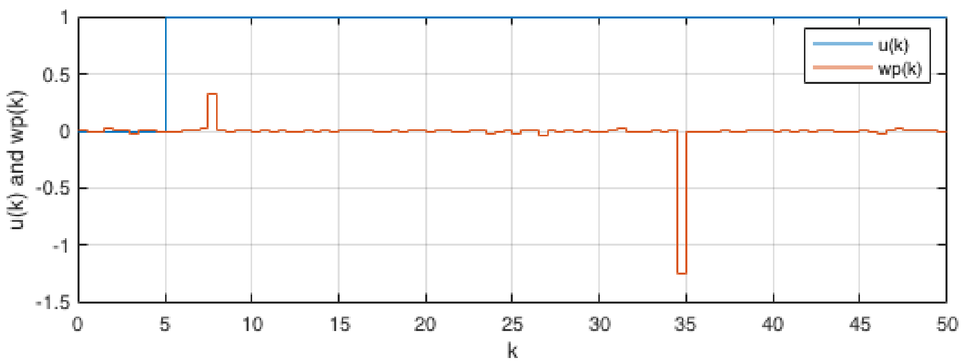
```
n, nu, ny
```

```
n = 2
nu = 1
ny = 1
```

Simulate system

```
X0 = zeros(n,1);
t = Ts*(0:nT)';
U = zeros(nT+1,1);
U(t>=5) = 1;
[Y,T,X] = lsim(Gpss,[U Wp],t,X0);
V = sigma_M*randn(nT+1, 1);
Ym = Y + V;  % measurement

figure(2)
subplot(2,1,1)
plot(t,Y,t,Ym); grid on
ylabel('y(k) and y_m(k)')
legend('y(k)', 'ym(k)')
subplot(2,1,2)
stairs(t, [U Wp]); grid on
xlabel('k')
ylabel('u(k) and wp(k)')
legend('u(k)', 'wp(k)')
```





3. Kalman filter simulation

The `KalmanFilter` class can be used to simulate a standard Kalman filter (prediction form):

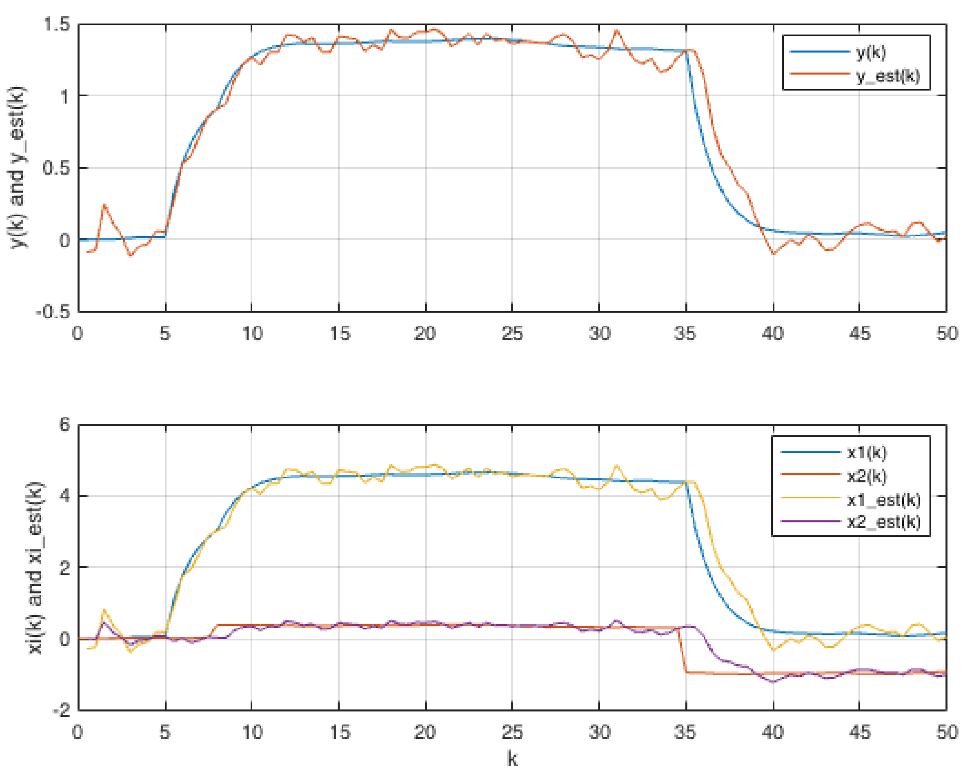
```
% Parameters
P0 = eye(n);
Q = diag([0.01^2 0.1^2]);
R = 0.1^2;
Bu = B(:,1);  % observer model without unmeasured inputs
KF1 = KalmanFilter(A,Bu,C,Ts,P0,Q,R,'KF1');
KF1
```

```
KF1 = 
  KalmanFilter with properties:

            n: 2
           nu: 1
           ny: 1
            A: [2×2 double]
            B: [2×1 double]
            C: [0.3000 0]
           Ts: 0.5000
    xkp1_est: [2×1 double]
    ykp1_est: 0
        Pkp1: [2×2 double]
            K: [2×1 double]
            Q: [2×2 double]
            R: 0.0100
           P0: [2×2 double]
        label: "KF1"
```

```
  x0: [2×1 double]
type: "KF"
```

The object has an `update` method which is used to simulate it in an iterative loop.

```
% Arrays to store simulation results
Xk_est = nan(nT+1,n);
Yk_est = nan(nT+1,ny);
obs = KF1;
for i = 1:nT
    uk = U(i,:)';
    yk = Ym(i,:)';
    obs.update(yk, uk);
    Xk_est(i+1,:) = obs.xkp1_est';
    Yk_est(i+1,:) = obs.ykp1_est';
end

figure(3)
subplot(2,1,1)
plot(t,[Y Yk_est]); grid on
ylabel('y(k) and y_est(k)')
legend('y(k)','y_est(k)')
subplot(2,1,2)
plot(t, [X Xk_est]); grid on
xlabel('k')
ylabel('xi(k) and xi_est(k)')
legend('x1(k)','x2(k)','x1_est(k)','x2_est(k)')
```



```
% Calculate mean-squared error in state estimates
mse = mean((X(2:end,:) - Xk_est(2:end,:)).^2, [1 2])
```

```
mse = 0.0917
```

4. Sub-optimal multi-model observer simulation

The `MKFObserver_SF_RODD` class can be used to instantiate a sub-optimal multi-model observer which uses sequence fusion as described by Robertson *et al.* (1998).

Use the help function for details on this function.

```
help MKFObserverSF_RODD
```

```
  Multi-model Kalman Filter class definition

  obs = MKFObserverSF_RODD(model,io,P0,epsilon, ...
      sigma_wp,Q0,R,nf,m,d,label,x0,r0)

  Object class for simulating a multi-model observer for
  state estimation in the presence of randomly-occurring
  deterministic disturbances (RODDs) as described in
  Robertson et al. (1995, 1998).

  The observer object can be used recursively in an
  iteration loop or in a Simulink S-function block (see
  MKFObserver_sfunc.m)

  Arguments:
    model : struct
        Struct containing the parameters of a linear
        model of the system dynamics including disturbances
        and unmeasured inputs. These include: A, B,
        and C for the system matrices, and the sampling
        period, Ts.
    io : struct
        Struct containing logical vectors u_known and y_meas
        indicating which inputs are known/unknown and which
        outputs are measured/unmeasured.
    P0 : (n, n) double
        Initial value of covariance matrix of the state
        estimates.
    epsilon : (nw, 1) double
        Probability(s) of shock disturbance(s).
    sigma_wp : (1, nw) cell array
        Standard deviations of disturbances. Each element of
        the cell array is either a scalar for a standard (Gaussian)
        noise or a (1, 2) vector for a random shock disturbance.
    Q0 : (n, n)
        Matrix containing variance values for process
        states. Only values in the rows and columns corresponding
        to the process states are used, usually the upper left
        block from (1, 1) to (n-nw, n-nw). The remaining
        values corresponding to the covariances of the input
        disturbance model states are over-written at initialization.
    R : (ny, ny) double
        Output measurement noise covariance matrix.
    nf : integer double
        Mode sequence length in number of detection intervals.
    m : integer double
        Maximum number of disturbances over fusion horizon.
```

```
    d : integer double
        Detection interval length in number of sample periods.
    label : String (optional, default "MKF_SF_RODD")
        Arbitrary name to identify observer instance.
    x0 : (n, 1) double (optional, default zeros)
        Initial state estimates.
    r0 : (1, 1) or (nh, 1) integer (optional)
        Integer scalar or vector with values in the range
        {1, ..., nj} which indicate the prior system modes at time
        k = -1. If not provided, the default initialization based
        on the mode sequence that is generated will be used.

  References:
   -  Robertson, D. G., Kesavan, P., & Lee, J. H. (1995).
      Detection and estimation of randomly occurring
      deterministic disturbances. Proceedings of 1995 American
      Control Conference - ACC'95, 6, 4453-4457.
      https://doi.org/10.1109/ACC.1995.532779
   -  Robertson, D. G., & Lee, J. H. (1998). A method for the
      estimation of infrequent abrupt changes in nonlinear
      systems. Automatica, 34(2), 261-270.
      https://doi.org/10.1016/S0005-1098(97)00192-1%

    Documentation for MKFObserverSF_RODD
```

```
% Collect system model parameters
model = struct();
model.A = A;
model.B = B;
model.C = C;
model.D = D;
model.Ts = Ts;

% Define multi-model observer
P0 = eye(n);
epsilon = 0.01;
sigma_wp = {[0.0100 1]};
Q0 = diag([0.01^2 0]);
R = 0.1^2;
nf = 5;  % length of fusion horizon in detection intervals
m = 1;  % maximum number of shocks during fusion horizon
d = 3;  % length of detection intervals in sample periods
io.u_known = [true false]';
io.y_meas = true;
MKF1 = MKFObserverSF_RODD(model,io,P0,epsilon,sigma_wp, ...
    Q0,R,nf,m,d,'MKF1');
MKF1
```

```
MKF1 =
  MKFObserverSF_RODD with properties:

               io: [1×1 struct]
               nw: 1
         n_shocks: 1
                m: 1
        sys_model: [1×1 struct]
            alpha: 0.0297
             beta: 0.9917
            p_seq: [6×1 double]
```

```
              p_rk: [2×1 double]
                Q0: [2×2 double]
                 R: 0.0100
           epsilon: 0.0100
          sigma_wp: {[0.0100 1]}
                 f: 15
                 d: 3
                id: 0
           id_next: 1
               seq: {6×1 cell}
                nf: 5
                 i: 5
            i_next: 1
        idx_branch: {[8×1 double]  [8×1 double]  [8×1 double]  [8×1 double]  [8×1 double]}
         idx_modes: {[8×1 double]  [8×1 double]  [8×1 double]  [8×1 double]  [8×1 double]}
         idx_merge: {[8×1 double]  [8×1 double]  [8×1 double]  [8×1 double]  [8×1 double]}
            nh_max: 8
            merged: [1×1 struct]
                nm: 6
                 n: 2
                nu: 1
                ny: 1
                P0: [2×2 double]
                x0: [2×1 double]
           filters: [1×1 struct]
                nj: 2
                nh: 8
            models: {[1×1 struct]  [1×1 struct]}
                Ts: 0.5000
                 T: [2×2 double]
                r0: [8×1 int16]
             label: "MKF1"
    p_seq_g_Yk_init: [8×1 double]
       p_seq_g_Ykm1: [8×1 double]
         p_seq_g_Yk: [8×1 double]
    p_yk_g_seq_Ykm1: [8×1 double]
        p_rk_g_Ykm1: [8×1 double]
        p_rk_g_rkm1: [8×1 double]
             xk_est: [2×1 double]
                 Pk: [2×2 double]
             yk_est: NaN
           xkp1_est: [2×1 double]
               Pkp1: [2×2 double]
                 rk: [8×1 int16]
               rkm1: [8×1 int16]
               type: "MKF_SF_RODD"
```

Inspect observer parameters

```
% Disturbance sequences for each filter
cell2mat(MKF1.seq)
```

```
ans = 6×5 int16 matrix
   1   1   1   1   1
   2   1   1   1   1
   1   2   1   1   1
   1   1   2   1   1
   1   1   1   2   1
   1   1   1   1   2
```

```
% Sequence probabilities (unconditioned)
MKF1.p_seq
```

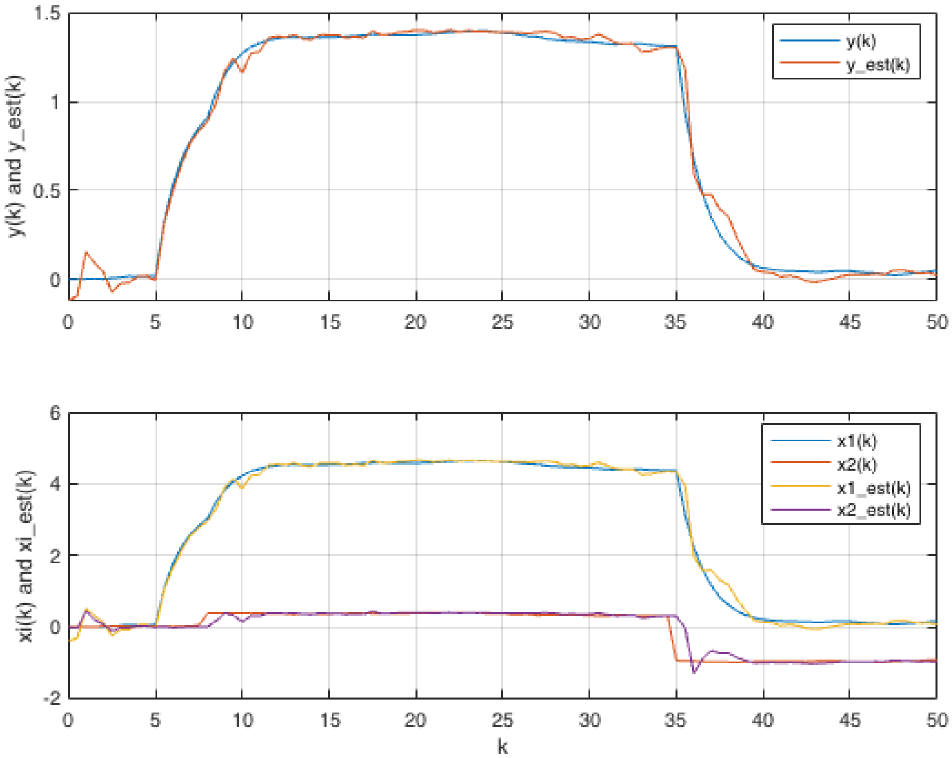
```
ans = 6×1
    0.8601
    0.0263
    0.0263
    0.0263
    0.0263
    0.0263
```

```
% Total probability of all sequences modelled (ideally, should be > 0.99)
MKF1.beta
```

```
ans = 0.9917
```

The `update` method is used to simulate the observer, same as for the Kalman filter above.

```
% Arrays to store simulation results
Xk_est = nan(nT+1,n);
Yk_est = nan(nT+1,ny);
obs = MKF1;
for i = 1:nT+1
    uk = U(i,:)';
    yk = Ym(i,:)';
    obs.update(yk, uk);
    Xk_est(i,:) = obs.xk_est';
    Yk_est(i,:) = obs.yk_est';
end

figure(4)
subplot(2,1,1)
plot(t,[Y Yk_est]); grid on
ylabel('y(k) and y_est(k)')
legend('y(k)','y_est(k)')
subplot(2,1,2)
plot(t, [X Xk_est]); grid on
xlabel('k')
ylabel('xi(k) and xi_est(k)')
legend('x1(k)','x2(k)','x1_est(k)',['x2_est(k)'])
```




```
% Calculate mean-squared error in state estimates
mse = mean((X(2:end,:) - Xk_est(2:end,:)).^2, [1 2])
```

```
mse = 0.0309
```

5. Multiple model observer with sequence pruning

The `MKFObserverSP_RODD` class can be used to instantiate a sub-optimal multi-model observer for estimating systems subject to randomly-occurring deterministic disturbances (RODDs). This implementation uses a sequence pruning algorithm described by Eriksson and Isaksson (1996).

Use the help function for details on this function.

```
help MKFObserverSP_RODD
```

```
  Multi-model Kalman Filter class definition

  obs = MKFObserverSP_RODD(model,io,P0,epsilon, ...
      sigma_wp,Q0,R,nh,n_min,label,x0,r0)

  Object class for simulating a multi-model observer that
  uses the adaptive forgetting through multiple models
  (AFMM) algorithm for state estimation in the presence of
  infrequently-occurring deterministic disturbances, as
  described in Eriksson and Isaksson (1996).

  Uses a sequence pruning method described in Eriksson and
  Isaksson (1996):
  - At the updates, let only the most probable sequence,
    i.e. with the largest weight of all the sequences
```

```
  split into 2 branches.
- After the update is completed, remove the sequence
  with the smallest weight.
- Renormalize the remaining weights to have unit sum.

Restriction to above rules:
- Do not cut branches immediately after they are born.
  Let there be a certain minimum life length for all
  branches.

The observer object can be used recursively in an
iteration loop or in a Simulink S-function block (see
MKFObserver_sfunc.m)

Arguments:
  model : struct
      Struct containing the parameters of a linear
      model of the system dynamics including disturbances
      and unmeasured inputs. These include: A, B,
      and C for the system matrices, and the sampling
      period, Ts.
  io : struct
      Struct containing logical vectors u_known and y_meas
      indicating which inputs are known/unknown and which
      outputs are measured/unmeasured.
  P0 : (n, n) double
      Initial value of covariance matrix of the state
      estimates.
  epsilon : (nw, 1) double
      Probability(s) of shock disturbance(s).
  sigma_wp : (1, nw) cell array
      Standard deviations of disturbances. Each element of
      the cell array is either a scalar for a standard (Gaussian)
      noise or a (1, 2) vector for a random shock disturbance.
  Q0 : (n, n)
      Matrix containing variance values for process
      states. Only values in the rows and columns corresponding
      to the process states are used, usually the upper left
      block from (1, 1) to (n-nw, n-nw). The remaining
      values corresponding to the covariances of the input
      disturbance model states are over-written at initialization.
  R : (ny, ny) double
      Output measurement noise covariance matrix.
  nh : integer double
      Number of hypotheses to model (each with a separate
      Kalman filter).
  n_min : integer double
      Minimum life of cloned filters in number of sample
      periods.
  label : String (optional, default "MKF_SP_RODD")
      Arbitrary name to identify observer instance.
  d : integer double (optional, default 1)
     Detection interval length in number of sample periods.
  x0 : (n, 1) double (optional, default zeros)
      Initial state estimates.
  r0 : (1, 1) or (nh, 1) integer (optional)
      Integer scalar or vector with values in the range
      {1, ..., nj} which indicate the prior system modes at time
      k = -1. If not provided, the default initialization based
      on the mode sequence that is generated will be used.

NOTE:
- The adaptive forgetting component of the AFMM
  (Andersson, 1985) is not yet implemented.
```

```
References:
 - Eriksson, P.-G., & Isaksson, A. J. (1996). Classification
   of Infrequent Disturbances. IFAC Proceedings Volumes, 29(1),
    6614-6619. https://doi.org/10.1016/S1474-6670(17)58744-3
 - Andersson, P. (1985). Adaptive forgetting in recursive
   identification through multiple models. International
   Journal of Control, 42(5), 1175-1193.
   https://doi.org/10.1080/00207178508933420

  Documentation for MKFObserverSP_RODD
```

```
% Collect system model parameters
model = struct();
model.A = A;
model.B = B;
model.C = C;
model.D = D;
model.Ts = Ts;

% Define multi-model observer
P0 = eye(n);
epsilon = 0.01;
sigma_wp = {[0.0100 1]};
Q0 = diag([0.01^2 0]);
R = 0.1^2;
nh = 5;  % number of hypotheses
n_min = 2;  % minimum life of cloned filters
io.u_known = [true false]';
io.y_meas = true;
MKF2 = MKFObserverSP_RODD(model,io,P0,epsilon,sigma_wp, ...
    Q0,R,nh,n_min,'MKF2');
```

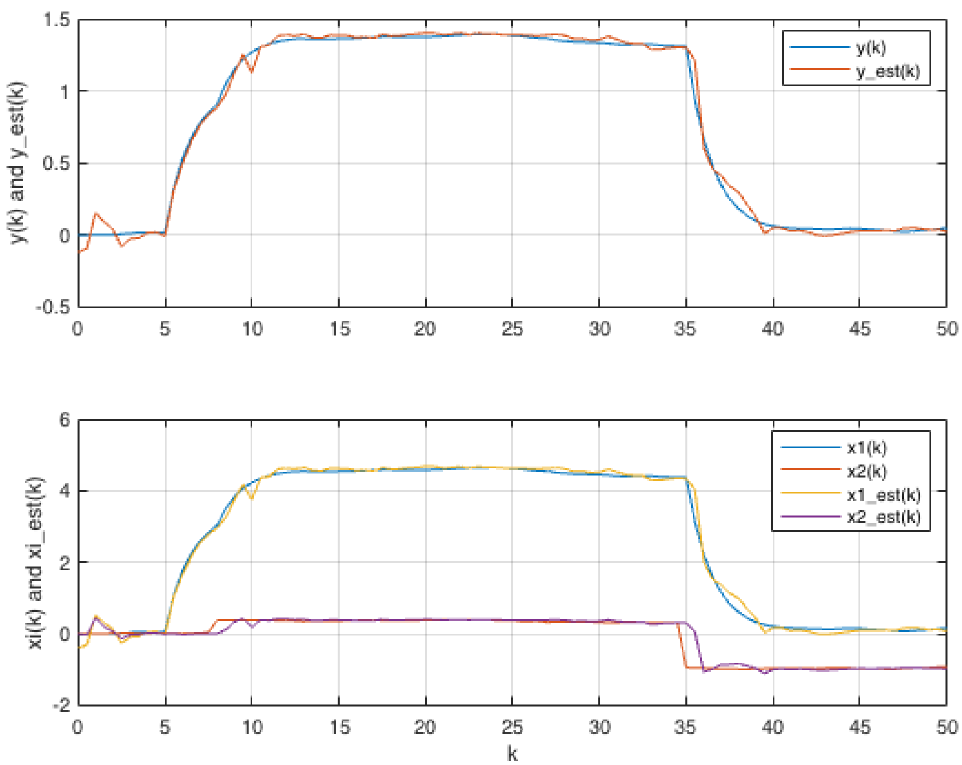
Inspect selected observer parameters

```
% Initial hypothesis probabilities
MKF2.p_seq_g_Yk
```

```
ans = 5×1
    1
    0
    0
    0
    0
```

```
% Arrays to store simulation results
Xk_est = nan(nT+1,n);
Yk_est = nan(nT+1,ny);
obs = MKF2;
for i = 1:nT+1
    uk = U(i,:)';
    yk = Ym(i,:)';
    obs.update(yk, uk);
    Xk_est(i,:) = obs.xk_est';
    Yk_est(i,:) = obs.yk_est';
end
```

```
figure(4)
subplot(2,1,1)
plot(t,[Y Yk_est]); grid on
ylabel('y(k) and y_est(k)')
legend('y(k)','y_est(k)')
subplot(2,1,2)
plot(t, [X Xk_est]); grid on
xlabel('k')
ylabel('xi(k) and xi_est(k)')
legend('x1(k)','x2(k)','x1_est(k)',['x2_est(k)'])
```




```
% Calculate mean-squared error in state estimates
mse = mean((X(2:end,:) - Xk_est(2:end,:)).^2, [1 2])
```

```
mse = 0.0292
```

References

1. Robertson, D. G., Kesavan, P., & Lee, J. H. (1995). Detection and estimation of randomly occurring deterministic disturbances. *Proceedings of 1995 American Control Conference - ACC'95*, *6*, 4453–4457. https://doi.org/10.1109/ACC.1995.532779.
2. Eriksson, P.-G., & Isaksson, A. J. (1996). Classification of Infrequent Disturbances. *IFAC Proceedings Volumes*, *29*(1), 6614–6619. https://doi.org/10.1016/S1474-6670(17)58744-3.
3. Robertson, D. G., & Lee, J. H. (1998). A method for the estimation of infrequent abrupt changes in nonlinear systems. Automatica, 34(2), 261-270. https://doi.org/10.1016/S0005-1098(97)00192-1.