```
// T49mixDesStat.sce
// MIXTURE ESTIMATION (descriptive, state-space)
// Experiments
// - change: simulated parameters, initial parameters, input signal,
//           type of proximity,
// ------------------------------------------------------------------
exec("ScIntro.sce",-1),
getd(), mode(0)

nd=500;                                                            // 1
// PARAMETERS                                                      // 2
c(1).mS=[-.5 .1                        // component parametrs      // 3
         .2 .7];                                                   // 4
c(2).mS=[.6 .2                                                     // 5
         .1 -.4];                                                  // 6
c(1).nS=[1 -1]';                                                   // 7
c(2).nS=[-1 1]';                                                   // 8
c(1).aS=[.9 1.5];                                                  // 9
c(2).aS=[1.2 .5];                                                  // 10
c(1).bS=10;                                                        // 11
c(2).bS=-1;                                                        // 12
c(1).swS=[.5 .1                        // component covariances    // 13
           0 .2]*.1;                                               // 14
```

```
c(2).swS=[.1 .5
           0 .2]*.1;
c(1).svS=.03;
c(2).svS=.05;
alS=[.9 .1                              // parameters of pointer model
     .1 .9];
nx=size(c(1).mS,1);
nc=length(c);                           // number of components
xS(:,1)=zeros(nx,1);                    // initail state
cS=1;                                   // initial pointer
u=.5*signal(nd,2,0,4);                  // input

// SIMULATION
for t=2:nd
  jS=sampCat(alS(:,cS(t-1)));           // pointer value
  cS(t)=jS;                             // stor pointer value
  xS(:,t)=c(jS).mS*xS(:,t-1)+c(jS).nS*u(t)+c(jS).swS*randn(2,1);
  y(t)=c(jS).aS*xS(:,t)+c(jS).bS*u(t)+c(jS).svS*randn();
end

// INITIALIZATION
ka=[1 1];                               // initial counter
c(1).x=randn(nx,1);                     // initial state x1
```

```
c(2).x=randn(nx,1);                    // initial state x2
c(1).R=1000*eye(nx,nx);                // state estimate covariance
c(2).R=1000*eye(nx,nx);                // state estimate covariance
c(1).rw=c(1).swS*c(1).swS';            // state model noise
c(2).rw=c(2).swS*c(2).swS';            // state model noise
c(1).rv=c(1).svS^2;                    // output model noise
c(2).rv=c(2).svS^2;                    // output model noise

// TIME LOOP
x=zeros(2,nd); yp=zeros(1,nd);
for t=2:nd
  //proximity
  for j=1:nc
    [nill,nill,py,ry] = Kalman(c(j).x(:,t-1),y(t),u(t),..
                          c(j).mS,c(j).nS,[],c(j).aS,c(j).bS,[],..
                          c(j).rw,c(j).rv,c(j).R); // prediction
     select 2            // select type of prox.: 1-quadratic, 2-pdf
     case 1, lq(j)=-2*log(abs(y(t)-py));     // quadratic proximity
     case 2, [nill,lq(j)]=GaussN(y(t),py,ry);// pdf proximity
     end
  end
  // weights
  q=exp(lq-max(lq));
```

```
  w=q/sum(q);                            // weights
  if 0, w=dDel(w); end   // 1=point estimates of weight
  wt(:,t)=w;                             // remember weights
  // estimation
  for j=1:nc
    [c(j).x(:,t),c(j).R,c(j).yp(t)] = Kalman(c(j).x(:,t-1),y(t),u(t),..
                         c(j).mS,c(j).nS,[],c(j).aS,c(j).bS,[],..
                         c(j).rw,c(j).rv,c(j).R);
     x(:,t)=x(:,t)+w(j)*c(j).x(:,t);    // resulting state estimate
     yp(t)=yp(t)+w(j)*c(j).yp(t);       // resulting prediction
  end
end

// RESULTS
tx=['b';'r';'g'];
set(scf(1),'position',[600 10 600 800]) // evolution of est. state
  title 'Simulated and estimated state'
for i=1:nx
  subplot(nx,1,i)
  plot(1:nd,xS(i,:),1:nd,x(i,:))
  xlabel('x'+string(i))
  legend('xS','x');
end
```

```
                                                                          // 84
set(scf(2),'position',[1200 10 600 400])// output prediction              // 85
plot(1:nd,y,1:nd,yp)                                                      // 86
legend('y','yp');                                                         // 87
title 'Output and its prediction'                                         // 88
                                                                          // 89
[nill,cp]=max(wt,'r');                  // accuracy of classification     // 90
disp 'Accuracy of classification'                                         // 91
ACC=acc(cS,cp)                                                            // 92
                                                                          // 93
disp 'Relative prediction error of y'   // relative prediction error     // 94
RPE1=rpe(y,yp)                                                            // 95
```

Description of the program

- Rows 3–18 define parameters of the model (both for simulation and for state estimation - the parameters are supposed to be known).
- Rows 19–20 introduce parameters of the pointer (switching) model.
- Rows 21–22 are dimension of the state and number of components.
- Rows 23–24 determine initial conditions for the state and pointer
- Row 25 defines the input signal.

- Rows 28–33 perform simulation using switched state-space model.
- Rows 36–44 set initialization for state estimation
  - Rows 37–38 introduce initial values for the estimated state.
  - Rows 39–40 set initial covariance of the state estimate.
  - Rows 41–44 copy the model covariances from the simulation.
- Rows 47–82 perform time loop for the state estimation.
  - Rows 50–58 compute proximity in a logarithmic form. It uses prediction of y and its covariance computed inside the Kalman filter procedure.

    Remark: two ways of computing are offered. 1-inverse value of the square of prediction error, 2-value of the predictive pdf of the predicted output.
  - Row 60 pre-normalizes and make exponent of the logarithmic proximity.
  - Row 61 computes weights

    Remark: Row 62 gives a possibility take a point estimate of the weight instead of its probabilistic form. The point estimate has one on position of maximal entry and zeros otherwise. Ti chooses just one, most probable, component.
  - Rows 64–72 perform state estimation using Kalman filtering.