Mixture models

Estimation of mixture models is based on classification of incoming data into the individual components. For each data record so called proximities to all components are constructed which, after normalization to the sum equal to one, give probabilities that the data record belongs to the component. During the data update, the statistics are recomputed adding data with the corresponding weights.

From this is immediately clear that if the initial positions of the components (or at least some of the components) lay too far from the area where the data occur, the weights are almost zero, the estimation is ill conditioned and it can (and usually will) fail. To avoid this, it is necessary to:

1. Set the initial positions to lie in the region where the data occur.
2. Prevent components from "escaping" or overlapping with the components right at the start.

This is done by initializing the mixture estimation using the prior data - i.e., setting the centers of the initial components appropriately and partially fixing them. In doing so, we assume that we have an prior sample of data (i.e., data obtained in the past that are available prior to the start of the estimation from continuously measured data).

Note

If prior data are not available, there is only one way to go. Scale all variables to have approximately zero mean and unit variance and continue to work on this normalized data domain. The results obtained can be again converted to the original metric.

The general principles for initialization are as follows:

1. Find a region where the measured data occurs. E.g., find out the minima and maxima for each variable, or better, look at their histograms.

2. Set the initial values of the parameter estimates as well as possible (using prior data and expert information).

3. Hold the prior estimates of the component centers at the beginning of the estimation so that they do not run too far or overlap.

4. It is good to set small covariances and fix them or at least to fix them at the beginning of estimation (till the regression parameters are approximately estimated).

5. Run the estimation repeatedly on the same data sample. When doing this, after each run we need to set the initial parameters to the values previously estimated and apply a suitable forgetting to enable the parameter estimates to move.

6. For dynamic mixtures, it is a good idea to start with components with suppressed dynamics (i.e., in the form of static ones). The initial location of the component is then given by a recalculated constant.

7. As prior data fictitious records specified by an expert can be used. Fictitious records can be specified in the following way. Take all important combinations of the values of explanatory variable (in the region of the application - e.g. specified points in a traffic microregion) and for each of them ask the expert what response he would guess. The values of the response and the corresponding vector of values of the explanatory variables form one fictitious record. If this record seems to be important it is possible to multiply it by a number expressing as if how man times it was measured.

8. Perform expert classification of several prior or artificial data and use them for initialization.

In the following text we are going to illustrate each point theoretically and in examples. To this end we will consider static components, i.e. models of the form $y_t = k_j + e_t$ (where k_j are the component centers for $j = 1, 2, \cdots, n_c$)

1. Data area

Suppose we have 3 variables x_1, x_2 and x_3 arranged in a data matrix x with three columns and as many rows as is the number of measurements. Then

```
mi=min(x,'c') ma=max(x,'c')
```

gives the 3-element vectors of the minimum and maximum values of the variables. We can accommodate the centers of the initial components thI with the command

```
for i=1:nc
   thI(:,i) = (mi+ma)/2 + .2*(ma-mi).*rand(3,1,'n')
end
```

where `(mi+ma)/2` is the center of the region, `(ma-mi)` is the width of the region, and `nd` is the number of components.

Program

```
// ini1Region.sce
// Determination of component centers
// ----------------------------------------
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data
```

```
nd=100;                                  // number of data
for t=1:nd
  y(:,t)=[5;3;8]+[2;1;3].*rand(3,1,'n'); // data simulation
end

mi=min(y,'c');                           // minimum
ma=max(y,'c');                           // maximum

nc=5;                                    // number of components
for i=1:nc
  thI(:,i)=(mi+ma)/2+.2*(ma-mi).*rand(3,1,'n');
                                         // centers of components
end

scf();
plot(y(1,:),y(2,:),'b.')
plot(thI(1,:),thI(2,:),'ro','markersize',12)
xlabel x1
ylabel x2
title 'Centers for the first two variables'
```

Program description

In the program, the initial centers are "scattered" around the center of the data region `(max(x)+min(x))/2`. The size

of the "scatter" is given by the radius of the data region `max(x)-min(x)`.

2. Initial parameter estimates

The initial location of the parameters (cluster centers) matters extremely. The centers should definitely lie in the region where the data occur, and ideally, the individual centers should lie close to the peaks of the data, i.e., where the density maxima of the expected clusters (working modes of the system) occur.

If we have some prior data (and in many practical applications, especially in transportation, they should be at disposal), we definitely should use them. First of all, we determine the region where the data occur (see previous point) and then we look for density peaks - either in histograms of individual variables or in pairs of variables. Usage the the histograms is clear. We show the procedure for pairs of variables:

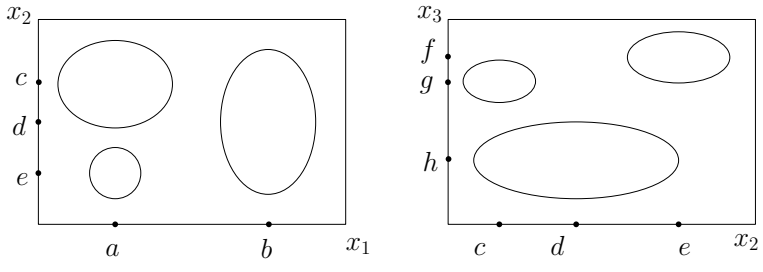
Let us have 3 variables $x1, x2$ and $x3$. We plot xy-graphs for the pairs $x1$-$x2$ and $x2$-$x3$.

`plot(x1,x2,':')` and `plot(x2,x3,'.')`

On the x-axis of the first plot we find the coordinates corresponding to the visible clusters and on the y-axis the corresponding y-coordinates. There may be multiple y-coordinates to one x-coordinate - then we record all of them, with the x-coordinates repeated.

Go to the second figure and mark all the y-coordinates from the first figure on its x-axis. To these, we determine the y-coordinates from the second figure and add them as a third number to the existing list of coordinates. We can continue in this way for more variables. We use the resulting coordinates as the component centers, i.e. the prior parameters of the static components.

The procedure is illustrated in the figure



The centers from the first figure will be:

$C1 = [a, c], C2 = [a, e], C3 = [b, d]$

From the second figure, add

$C1 = [a, c, g], C2 = [a, e, f], C3 = [b, d, h]$

These are not necessarily the true centers of the multivariate components, but at least we know that something is going on here and the initial centers somehow belong to the density vertices. The fine-tuning should happen in the actual estimation.

Program

```
// ini2IniPar.sce
// Initial parameter estimates and associated statistics
// ----------------------------------------------------
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
```

```
clear("u","t","n")                        // clear auxiliary data
getd functions
rand('seed',0)

nd=500;              // number of data
I_estCov=0;          // estimation of noise covariances = 0|1 no|yes

// regression coefficients for simulation
Cy(1).th=[1 2 4]';
Cy(2).th=[5 3 2]';
Cy(3).th=[2 5 6]';
Cy(4).th=[3 8 8]';
nc=length(Cy);
// noise covariances for simulation
for i=1:4
  rn=.3*randn(3,3);
  Cy(i).cv=.1*(eye(3,3)+rn+rn');
end
// pointer parameters for simulation
Cp.th=fnorm(randu(1,3)+.1,2);

ct(1)=1;                                  // initial poiner
for t=1:nd
```

```
  ct(1,t)=sum(cumsum(Cp.th)<randu())+1; // pointer generation
  j=ct(t);                              // active component
  y=Cy(j).th+uut(Cy(j).cv)*randn(3,1);  // output generation
  yt(:,t)=y;                            // remember output
end

// 2d (marginal) clusters
set(scf(),'position',[600 100 800 600])
for i=1:2
  for j=i+1:3
    subplot(3,3,3*(i-1)+j)
    plot(yt(i,:),yt(j,:),'.','markersize',3)
    title('Var '+string(i)+' '+string(j))
  end
end
```

Program description

The program simulates the 4 components of a mixture and plots xy-plots of the quantities y1-y2, y2-y3 and y2-y3. From these we can read the three-dimensional centers of the components.

3. Holding the prior centers

This is a very important method, used more or less in every estimation!!!

At the beginning of the estimation, information about the parameters is extracted from only a small number of data. If the parameters were left completely free, they could "rush" meaninglessly after each measured data record and could easily stray somewhere from which there would be no return. That's why we need to start with statistics that already have some prior information in them - either from the data or from an expert.

We will demonstrate the situation for normal components. In other cases, the situation is similar.

The update of the statistics for estimation regression coefficients is performed as follows $V_t = V_{t-1} + \Psi\Psi'$ where V is the information matrix, $\Psi = [y_t, 1]'$ is the extended regression vector with the new data. This shows that the matrix V grows gradually as data are loaded into it.

At the same time, it is clear that if the matrix V is zero at the beginning, immediately the first data will change it a lot. But if it is large enough, the initial data do not affect it very much.

The point estimates of the regression coefficients follow the formula $\hat{\theta} = V_\psi^{-1} V_{y\psi}$ where $V_y, V_{y\psi}$ and V_ψ are the submatrices of V divided according to the regression vector. If we multiply the matrix V by a constant c, then $\hat{\theta} = (cV_\psi)^{-1}(cV_{y\psi}) = V_\psi^{-1} V_{y\psi}$ and the estimates will not change - only (for large c) the product cV will be larger, and hence more robust to changes due to initial data.

So the conclusion is quite simple: The large information matrix is used to hold the initial component centers.

Note

If we have some initial parameters in mind $\hat{\theta}_0$ and we want to construct an information matrix for them, we proceed as follows

$$V = \begin{bmatrix} 1 & \theta_0' \\ \theta_0 & 1 \end{bmatrix}.$$

Then the first guess $\hat{\theta} = V_\psi^{-1} V_{y\psi} = \hat{\theta}_0$.

Program

```
// ini3Hold.sce
// Holding initial parametrs
// - run for ni = 1, 10, 100.
// ----------------------------------------
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data

nd=200;                                  // number of data
ni=1;                                    // number of initial data

b0=1; a1=.3; b1=-.2; k=1.5; sd=1; y(1)=-2;   // sim. parameters
B0=2; A1=.1; B1=.2; K=-1;                    // ini. est. parameters

u=sin(8*%pi*(1:nd)/nd)+rand(1,nd,'n');       // control
// Simulation
for t=2:nd
  y(t)=b0*u(t)+a1*y(t-1)+b1*u(t-1)+k+sd*rand(1,1,'n');
end

// Initialization
V=[1 B0 A1 B1 k                              // information matrix
```

```
  b0 1  0  0  0
  A1 0  1  0  0
  B1 0  0  1  0
  K  0  0  0  1];

V=V*ni;        // Inf. matrix is set to give initial parameters,
               //  ni sais how many initial data records have been used.

// Estimation
for t=2:nd
  Ps=[y(t) u(t) y(t-1) u(t-1) 1]';          // ext. reg. vector
  V=V+Ps*Ps';                               // updt of V
  Vy=V(1:2,1);                              // partitioning of V
  Vyp=V(2:$,1);
  Vp=V(2:$,2:$);
  th=inv(Vp+1e-12*eye(Vp))*Vyp;             // point estimates
  tht(:,t)=th';                             // remember estimates
end

// Results
s=nd-50:nd;
tx=['b';'r';'k';'m';'c';'g'];
scf();
```

```
for i=1:size(tht,1)
  plot(tht(i,2:$),tx(i))
end
plot(s,ones(s)*b0,':'+tx(1))
plot(s,ones(s)*a1,':'+tx(2))
plot(s,ones(s)*b1,':'+tx(3))
plot(s,ones(s)*k,':'+tx(4))
title 'Evolution of estimated parameters and theit true values'
```

Program description

Estimates a first-order scalar regression model, with parameters denoted by lowercase letters. The initialization of the information matrix V is set to compute the initial parameters (denoted by capital letters). The constructed matrix V is multiplied by ni (that is, the number of data from which the matrix V was determined). The parameter estimates are further varied according to the size of the number ni. For small ni fast, for large ni slow. In the output, we observe the evolution of the parameter estimates over time.

4. Fixed noise covariance

The noise covariance determines the shape of the clusters. If we are primarily concerned with finding the cluster centers, we can keep the covariances small and fixed (not estimating them). We specify them as a unit matrix multiplied by a tenth to a hundredth of the range (radius) of the expected data region.

If we care about the shape of the clusters, it is recommended to turn on their estimation only during the estimation process (e.g. in the middle of it), when the centers are essentially found. But there is still a danger that covariances

will run away or that one component will overlap the others - so be careful.

Program

```
// ini4aCov.sce
// Fixed covariance
// ----------------------------------------
clc, clear,close(winsid()); mode(0)
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data
getd c:\functions

nd=500;             // number of data
I_estCov=0;         // estimation of noise covariances ? 0|1 no|yes

// simulated reg.coef.
Sim.Cy(1).th=[0.9 -.5]';
Sim.Cy(2).th=[0.4 0.6]';
nc=length(Sim.Cy);
Sim.nc=nc;
// simulated noise covariances
Sim.Cy(1).sd=0.5*[1 -.6;-.6,1];
Sim.Cy(2).sd=0.3*[1 .2;.2,1];
```

```
// simulated pointer parameters
Sim.Cp.th=fnorm(ones(1,2)+.1,2);

Sim.ct(1)=1;                                        // initial poiner
// SIMULATION ============================================================
for t=2:nd
  Sim.ct(1,t)=sum(randu(1,1)>cumsum(Sim.Cp.th))+1; // pointer gen.
  j=Sim.ct(t); // active component
  y=Sim.Cy(j).th+Sim.Cy(j).sd*randn(2,1);          // output gen.
  Sim.yt(:,t)=y;
end

// initial parameters
a=.5;                          // std of scattering initial parametrs
for j=1:nc                     //  from those used in simulation
  [mr,mc]=size(Sim.Cy(1).th);
  Ps=[Sim.Cy(j).th;1]+[a*randn(mr,mc);0];    // initial parameters
  Est.Cy(j).V=Ps*Ps';                        // statistics
  Est.Cy(j).th=Sim.Cy(j).th+a*randn(2,1);    // pt.est. of reg.coef
  Est.Cy(j).sd=.01*eye(2,2);                 // standard deviation
end
Est.ka=ones(1,nc);             // counter
Est.Cp.V=ones(1,nc);           // pointer statistics
```

```
Est.Cp.th=fnorm(ones(1,nc));  // pointer parameter
w=fnorm(ones(1,nc));          // weights


// ESTIMATION ============================================================
printf(' ')
for t=2:nd
  if t/fix(nd/10)==fix(t/fix(nd/10)), printf('.'); end
  for j=1:nc
    [xxx,G(j)]=GaussN(Sim.yt(:,t),Est.Cy(j).th,Est.Cy(j).sd);
                                              // proximity
  end
  Lq=G-max(G);
  q=exp(Lq);

  ww=q'.*Est.Cp.th; w=ww/sum(ww);             // generation of weights
  wt(:,t)=w';

  // Update of statistic
  Ps=[Sim.yt(:,t)' 1];                        // extended reg.vec.
  for i=1:nc
    Est.Cy(i).V=Est.Cy(i).V+w(i)*Ps'*Ps;      // information matrix
    Est.ka(i)=Est.ka(i)+w(i);                 // counter
```

```
    Est.Cp.V(i)=Est.Cp.V(i)+w(i);           // pointer statistics

    //nove rozdeleni informacni matice V
    Vy=Est.Cy(i).V(1:2,1:2);                // part Vy  - y.y
    Vyp=Est.Cy(i).V($,1:2);                 // part Vyp - psi.y
    Vp=Est.Cy(i).V($,$);                    // part Vp  - psi.psi'
    Est.Cy(i).th=inv(Vp+1e-8*eye(Vp))*Vyp;  // pt.est. - reg.coef.
    Est.Cy(i).tht(:,t)=Est.Cy(i).th';       // pt.est. - covar.
    if I_estCov~=0
      // PT.EST. OF NOISE COVARIANCE - USED OR NOT
      Est.Cy(i).cv=(Vy-Vyp'*inv(Vp+1e-8*eye(Vp))*Vyp)/Est.ka(i);
    end
  end
  Est.Cp.th=fnorm(Est.Cp.V,2);     // pt.est. of pointer parameter
  [ss,Est.ct(1,t)]=max(w);         // store
end

s=2:nd;
[q,T]=c2c(Sim.ct(s),Est.ct(s));
Ect=q(Est.ct(s));

S=list();                          // simulated clusters
for i=1:nc
```

```
  j=find(Sim.ct==i);
  S(i)=Sim.yt(:,j);
end

C=list();                          // estimated clusters
for i=1:nc
  j=find(Ect==i);
  C(i)=Sim.yt(:,j);
end

// Results
ACC=acc(Sim.ct(s),Ect');
//printf('\n Wrong classifications %d from %d\n',wr,length(s))

set(scf(1),'position',[300 100 1200 600])
subplot(121)
plot(Est.Cy(1).tht(1,:),Est.Cy(1).tht(2,:),'.')
plot(Est.Cy(1).tht(1,2),Est.Cy(1).tht(2,2),'g.','markersize',18)
plot(Sim.Cy(1).th(1),Sim.Cy(1).th(2),'ro','markersize',12)
subplot(122)
plot(Est.Cy(2).tht(1,:),Est.Cy(2).tht(2,:),'.')
plot(Est.Cy(2).tht(1,2),Est.Cy(2).tht(2,2),'g.','markersize',18)
plot(Sim.Cy(2).th(1),Sim.Cy(2).th(2),'ro','markersize',12)
```

```
set(scf(2),'position',[350 150 1200 600])
subplot(121)
plot(S(1)(1,:),S(1)(2,:),'b.')
plot(S(2)(1,:),S(2)(2,:),'r.')
title 'Simulated'
subplot(122)
plot(C(1)(1,:),C(1)(2,:),'b.')
plot(C(2)(1,:),C(2)(2,:),'r.')
title 'Estimated'
```

or

```
// ini4bCov.sce
// Fixní kovariance při odhadu směsi (odhad více komponent než v sim)
```

```
// ---------------------------------------
clc, clear, close(winsid()); mode(0)
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data
getd c:\functions

nd=500;                  // number of data
nc=8;                    // number of componentd
t_estCov=50;             // time when est. of covars is switched on

// simulated reg.coef.
Sim.nc=nc;
Sim.Cy(1).th=[0.9 -.5]';
Sim.Cy(2).th=[0.4 0.6]';
// simulated noise covariances
Sim.Cy(1).cv=0.2*[1 -.6;-.6,1];
Sim.Cy(2).cv=0.1*[1 .2;.2,1];
// simulated pointer parameters
Sim.Cp.th=fnorm(ones(1,2)+.1,2);
Sim.ct(1)=1;                  // initial poiner

// SIMULATION ============================================================
```

```
for t=2:nd
  Sim.ct(1,t)=sum(randu(1,1)>cumsum(Sim.Cp.th))+1;    // pointer
  j=Sim.ct(t); // active component
  y=Sim.Cy(j).th+uut(Sim.Cy(j).cv)*randn(2,1);        // output
  Sim.yt(:,t)=y;
end

// initial parameters
my=mean(Sim.yt,2);
a=.3;                          // std of scattering initial parametrs
for j=1:nc                     //  from those used in simulation
  [mr,mc]=size(Sim.Cy(1).th);
  th=my+a*rand(mr,mc,'n');
  Ps=[th;1];                   // initial parameters
  Est.Cy(j).V=Ps*Ps';          // statistics
  Est.Cy(j).th=th;             // pt.est. of reg.coef
  Est.Cy(j).cv=.01*eye(2,2);   // standard deviation
end
Est.ka=ones(1,nc);             // counter
Est.Cp.V=ones(1,nc);           // pointer statistics
Est.Cp.th=fnorm(ones(1,nc));   // pointer parameter
w=fnorm(ones(1,nc));           // weights
```

```
// ESTIMATION ============================================================
I_estCov=0;
printf(' ')
for t=2:nd
  if t/fix(nd/10)==fix(t/fix(nd/10)), printf('.'); end
  for j=1:nc
    [xxx,G(j)]=GaussN(Sim.yt(:,t),Est.Cy(j).th,Est.Cy(j).cv);
                                              // proximity
  end
  Lq=G-max(G);
  q=exp(Lq);

  ww=q'.*Est.Cp.th; w=ww/sum(ww);             // generation of weights

  // Update of statistic
  Ps=[Sim.yt(:,t)' 1];                        // extended reg.vec.
  for i=1:nc
    Est.Cy(i).V=Est.Cy(i).V+w(i)*Ps'*Ps;      // information matrix
    Est.ka(i)=Est.ka(i)+w(i);                 // counter
    Est.Cp.V(i)=Est.Cp.V(i)+w(i);             // pointer statistics

    //nove rozdeleni informacni matice V
```

```
    Vy=Est.Cy(i).V(1:2,1:2);                // part Vy  - y.y
    Vyp=Est.Cy(i).V($,1:2);                 // part Vyp - psi.y
    Vp=Est.Cy(i).V($,$);                    // part Vp  - psi.psi'
    Est.Cy(i).th=inv(Vp+1e-8*eye(Vp))*Vyp;  // pt.est. - reg.coef.
    Est.Cy(i).tht(:,t)=Est.Cy(i).th';       // pt.est. - covar.
    if t>t_estCov, I_estCov=1; end
    if I_estCov~=0
      // PT.EST. OF NOISE COVARIANCE - USED OR NOT
      Est.Cy(i).cv=(Vy-Vyp'*inv(Vp+1e-8*eye(Vp))*Vyp)/Est.ka(i);
    end
  end
  Est.Cp.th=fnorm(Est.Cp.V,2);     // pt.est. of pointer parameter
  [ss,Est.ct(1,t)]=max(w);         // store
end

s=2:nd;
Sc=Sim.ct(s);
Ec=Est.ct(s);
yt=Sim.yt(:,s);
S=list();                          // simulated clusters
for i=1:2
  j=find(Sc==i);
  S(i)=yt(:,j);
```

```
end

C=list();                          // estimated clusters
for i=1:nc
  j=find(Ec==i);
  C(i)=yt(:,j);
end

// Results
set(scf(1),'position',[300 100 1200 300])
for i=1:nc
subplot(1,nc,i)
plot(Est.Cy(i).tht(1,:),Est.Cy(i).tht(2,:),'.')
plot(Est.Cy(i).tht(1,2),Est.Cy(i).tht(2,2),'g.','markersize',18)
end
title 'green ring - beginning'

set(scf(2),'position',[300 550 1200 300])
subplot(1,nc+1,1)
plot(S(1)(1,:),S(1)(2,:),'b.')
plot(S(2)(1,:),S(2)(2,:),'r.')
set(gca(),'data_bounds',[-1 3 -1.5 2])
title 'Simulated'
```

```
for i=1:nc
subplot(1,nc+1,i+1)
if ~isempty(C(i))
plot(C(i)(1,:),C(i)(2,:),'.')
end
title 'Estimated'
set(gca(),'data_bounds',[-1 3 -1.5 2])
end
```

Program description

The `I_estCov` parameter decides whether or not to estimate the noise covariances. If it is 0, the covariances are not estimated and remain as set in initialization. For `I_estCov` = 1, they are estimated. A compromise is to delay their estimation, where they start estimating only from the specified time `t_estCov`.

5. Repeated estimation on the same data

The component centers start at their initial centers and gradually travel to the density peaks. Each data record is shifted a little according to which component it belongs to (according to the weights w). If there is little data, it can and does happen that the estimation ends before the centers have arrived to their proper places. Then it is reasonable to continue the estimation with the same data again, but to start not from the initial centers, but from those that have arrived so far.

However, there is one problem. At the end of the estimation, the information matrices are large (we say that the estimation is tight) and the centers would either not move anymore or very little. Therefore, we need to forget statistics by dividing by a number that is approximately equal the the length of previous estimation. After this, the parameters will be equal the the latest estimates, however, the statistics will be as if only after one step of estimation (and thus ready to change).

This can be done repeatedly. In doing so, it is a good idea to follow the evolution of the centers, perhaps in a graph, and continue until the center estimates move.

The program illustrates the situation for two estimates

```
// ini5Opak.sce
// Opakovaný odhad na stejných datech
// ----------------------------------------
clc, clear, close(winsid()), mode(0)
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data
```

```
nd=10;                                  // numb. of data
k=[2; -1]; sd=.01;                      // reg. parameters

for t=1:nd
  y(:,t)=k+sd*rand(2,1,'n');            // simulation
end

// Inicializace k prvnímu odhadu
kI=[-5;8];                              // initial parametrs
V0(1:2,1:2)=kI*kI';
V0(3,1:2)=kI'; V0(1:2,3)=kI; V0(3,3)=1; // initial inf. matrix
V0=V0*10; V=V0;                         // initial strength = 10

// První odhad
for t=1:nd
  Ps=[y(:,t)' 1]';
  V=V+Ps*Ps';
  Vy=V(1:2,1:2);
  Vyp=V(3:$,1:2);
  Vp=V(3:$,3:$);
  th=inv(Vp+1e-12*eye(Vp))*Vyp;
  th1(:,t)=th';               // vývoj odhadů v první fázi
```

```
end
th1=[kI th1];                  // z výchozího odhadu kI se dále odhaduje

fi=50;     // Zkuste fi=1,10,50
V=V/fi;   // ZAPOMENUTÍ (uvolnění odhadu pro další postup)
// Druhý odhad na stejných datech
for t=1:nd
  Ps=[y(:,t)' 1]';
  V=V+Ps*Ps';
  Vy=V(1:2,1:2);
  Vyp=V(3:$,1:2);
  Vp=V(3:$,3:$);
  th=inv(Vp+1e-12*eye(Vp))*Vyp;
  th2(:,t)=th';                      //  vývoj odhadů v druhé fázi
end

// Výsledky
scf();
plot(th1(1,:),th1(2,:),'.:')       // první odhad
plot(th2(1,:),th2(2,:),'.:g')      // druhý odhad
plot(2,-1,'ro','markersize',12)    // správná hodnota paramrtru k
set(gca(),'data_bounds',[-6 4 -2 9])
title 'Vývoj odhadu konstanty - b=1.odhad. g=2.odhad'
```

Program description

The program considers the estimation of a static regression model with two-dimensional data. The estimator is initialized (by a constant kI) and run. The result of the first estimation is an information matrix V and the point estimates of the parameters computed from it. This is followed by a second estimation with the same data but with the information matrix computed in the previous run. It is convenient to subject this matrix to forgetting (dividing the integer matrix by fi). The estimation then starts from the point estimates of the parameters from the previous run. Forgetting the information matrix frees it to allow the estimates to move again. Try the offered forgetting coefficients fi and observe their effect.

6. Dynamic mixtures

In contrast to static mixtures, whose components have in reality fixed centers - the mean values of the components (the peaks of the density hills), dynamic mixtures have a mean value that is data dependent and therefore moving. Thus, estimates of their centers do not converge to fixed points, but to moving ones. Their initial setup is therefore more complicated.

We show a procedure that starts from static components and adds dynamics incrementally during estimation.

We will consider a single 1st order dynamic components $y_t = ay_{t-1} + bu_t + k + e_t$. The procedure is:

1. Take $a = 0$ and $b = 0$ as prior parameters. This corresponds to the static model $y_t = k + e_t$.
2. Find or estimate the average value of y as $\bar{y}$.
3. Set the initial constant k to $\bar{y}$.

4. Determine the initial information matrix as

$$V_0 = \begin{bmatrix} \bar{y}^2 & \Psi_0' \, \Psi_0 & E \end{bmatrix}$$

where $\Psi_0 = [0,\, 0,\, k]'$, $k = \bar{y}$ and E is the unit matrix.

The corresponding program is as follows

```
// ini6Dyn.sce
// Inicializace dynamických komponent
// ----------------------------------------
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data
exec("ScIntro.sce",-1),mode(0)           // intro to sesion

// INITIALLIZATION OF ESTIMATION OD DYNAMIC MODEL AS STATIC ONE
function V=th2VN(nps,my);
  // nps  length of regression vector (without y(t))
  // my   average of y
  my=my(:);
  ny=length(my);
  nPs=ny+nps;
  V=eye(nPs,nPs);
```

```
  V(ny,ny)=my*my';
  V(nPs,1:ny)=my';
  V(1:ny,nPs)=my;
endfunction

nd=500;
b0=1; a=.2; b1=-.3; k=2; sd=.1; y(1)=2; y(2)=5;   //simul. parameters
u=1+.2*sin(10*%pi*(1:nd)/nd)+.1*rand(1,nd,'n');   // input
// sinulation
for t=3:nd
  y(t)=b0*u(t)+a*y(t-1)+b1*u(t-1)+k+sd*rand(1,1,'n');
end

my=mean(y);                          // mean of initial output
// Konstruction of V
V0=th2VN(4,my);
V=V0;

// Estimation
for t=3:nd
  Ps=[y(t),u(t),y(t-1),u(t-1),1]';
  V=V+Ps*Ps';
  Vy=V(1,1);
```

```
  Vyp=V(2:$,1);
  Vp=V(2:$,2:$);
  th=inv(Vp+1e-12*eye(Vp))*Vyp;

  tht(:,t)=th;                      // evolution of est. parameters
end

set(scf(1),'position',[300 300 500 400])
plot(tht')
title 'Evolution of par.estimates'
disp(th,'Esimated parameters')

// Prediction
for t=3:nd
  yp(t)=th(1)*u(t)+th(2)*y(t-1)+th(3)*u(t-1)+th(4);
end

s=2:nd;
set(scf(2),'position',[800 300 500 400])
plot(s,y(s),s,yp(s))
title 'Prediction'
```

In the program, we consider a dynamic regression model

$$y_t = b_0 u_t + a y_{t-1} + b_1 u_{t-1} + k + e_t$$

We initialize this model as static, i.e. of the form

$$y_t = k + e_t$$

where we expect k to be approximately equal to the average of the prior y. Here we hope that for the static model we have "hit the data area" and the dynamics will gradually be estimated.

7. Artificial regression vectors

A good way to convert often abstract expert knowledge into data that is suitable for estimation initialization is the creation of artificial data vectors. Each data vector consists of a regression vector and its corresponding output value. We will illustrate the situation with an example:

We observe the length of a queue in just one leg of a controlled intersection that collects traffic from a certain area. There are 5 critical points in this area, which may be at different traffic levels depending on the situation. Thus, the regression vector will contain 5 variables (traffic levels in the area) and the output is the length of the queue at the intersection. The expert can convert his knowledge into an output of the most important combinations of loads for each location in the area and assign (according to his belief) the corresponding value of the length of the queue at the intersection.

Note

If the prior data are available, it is of course possible to use them and select some important data vectors that are crucial for the situation and carry a lot of information. The selection can again be according to the expert's recommendation.

The constructed data vectors are then treated normally as measured data vectors in the initialization.

The method is illustrated by the following program. A description follows the program.

```
// ini7UmVekt.sce
// Umělé regresní vektory
// ---------------------------------------
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data
exec("ScIntro.sce",-1),mode(0)           // intro to sesion
rand('seed',0);

nd=200;        // all data (init + estim)
ni=50;         // initial data
I_init=0;      // I_init=1 - initialization (only)
               // I_init=0 - estimation with constructed initial V

// Simulation
u1=   sign(2*sin(12*%pi*(1:nd)/nd))+1;
u2= 2*sign(2*sin(10*%pi*(151:nd+150)/nd))+2;
u3=.2*sign(2*sin(8*%pi*(1:nd)/nd))+1;
```

```
a=.6; k=.5; y(1)=0;
for t=2:nd
  y(t)=a*y(t-1)+u1(t)+u2(t)+u3(t)+k+.5*rand(1,1,'n');
end

// Initialization (external data vectors)
if I_init==1
s=2:ni;
plot([y(s) u1(s)' u2(s)' u3(s)'])
legend('y','u1','u2','u3');
end

Psi=list();
// Psi = [y(t) y(t-1) u1(t) u2(t) u3(t) 1]   // data vector
// th0:         a      b1    b2    b3   k    // corresponding parameters
// extracted from prior data (can be also from expert knowledge)
Psi(1)=[10  9 2 0 1.2 1]';
Psi(2)=[17 15 2 4 1.2 1]';
Psi(3)=[15 16 0 4 1.2 1]';
Psi(4)=[14 14 0 4 .8  1]';
Psi(5)=[10 13 0 0 .8  1]';
Psi(6)=[11 10 2 0 1.2 1]';
```

```
Psi(7)=[11 11 2 0 1.2 1]';
Psi(8)=[8 7 2 0 .8 1]';
Psi(9)=[9 8 2 0 .8 1]';
Psi(10)=[9 8 2 0 .8 1]';

m=length(Psi(1));
n=length(Psi);
V=zeros(m,m);
for i=1:n
  V=V+Psi(i)*Psi(i)';
end

th0=v2thN(V/n,1)'
if I_init==1, return, end

// Estimation;
V=V*50;
for t=(ni+1):nd
  Psi=[y(t) y(t-1) u1(t) u2(t) u3(t) 1]';
  V=V+Psi*Psi';
  th=v2thN(V/(t-n),1);
  tht(:,t)=th;
end
```

```
th=th'
thSim=[.6 1 1 1 .5]

scf();
plot(tht')
```

Program description

The program shows the estimation of a first order regression model with three inputs

$$y_t = ay_{t-1} + u1_t + u2_t + u3_t + k + e_t$$

The program is split into 2 parts according to the `I_init` option. For option 1, the prior data is shown and the user can select from it the important items for the data vectors Psi, which are further used for initialization. Option `I_init=0` prepares a "hard" run, i.e. a prepared initialization with subsequent estimation. The evolution of the parameter estimation is shown.

8. Expert classification

This method follows the previous procedure, but instead of assigning an output value to a regression vector, the selected data is expertly assigned to the class (component) to which it (according to th expert knowledge) belong.

Again, there are several ways to perform this pre-classification:

1. Expertly create the entire data vector and classify it into a class.

2. Take some a priori measured data vector and expertly assign a class to it.

3. Use some "superior" tools (let a human observe the situation or rent some expensive measuring instrument) to measure not only the data records but also the corresponding classification classes for the prior measurements.

We use what we get for initialization, where we perform learning with the teacher (i.e., knowing the correct classification).

The following program shows this pre-classification initialization. The description follows the program.

```
// ini8aPreClass.sce
// Inicializace s učitelem
// ---------------------------------------
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data
exec("ScIntro.sce",-1),mode(0)           // intro to sesion
rand('seed',0)

nd=500;             // number of data
ni=20;              // lenght of initialization phase
t_estCov=1;         // when estimation of covariances is switched on

// simulated reg.coef.
Sim.Cy(1).th=[1.9 -.5]';
```

```
Sim.Cy(2).th=[0.4 0.6]';
Sim.Cy(3).th=[-1.2 1.6]';
Sim.Cy(4).th=[.1 -1.6]';
Sim.Cy(5).th=[2.1 2.2]';
nc=length(Sim.Cy);
Sim.nc=nc;
// simulated noise covariances
rcv=.8;                        // amplitude of covariances
for i=1:nc
  g=rand(2,2,'u');
  Sim.Cy(i).cv=rcv*(eye(2,2)+(g+g'));
end
// simulated pointer parameters
Sim.Cp.th=fnorm(ones(1,nc)+.1,2);
Sim.ct(1)=1;                   // initial poiner

// SIMULATION ============================================================
for t=1:(ni+nd)
  ct(1,t)=sum(rand(1,1,'u')>cumsum(Sim.Cp.th))+1;    // pointer
  j=ct(t); // active component
  dt(:,t)=Sim.Cy(j).th+uut(Sim.Cy(j).cv)*rand(2,1,'norm'); // output
end
// initial data
```

```
Sim.yi=dt(:,1:ni);
Sim.ci=ct(1:ni);
// simulated data
Sim.yt=dt(:,(ni+(1:nd)));
Sim.ct=ct(ni+(1:nd));

// INITIALIZATION ======================================================
// initial parameters
yi=list();
for i=1:nc
  j=find(Sim.ci==i);
  yi(i)=Sim.yi(:,j);
end

for i=1:nc                    //  from those used in simulation
  Est.Cy(i).V=zeros(3,3);
  nj=size(yi(i),2);
  for j=1:nj
    Ps=[yi(i)(:,j);1];          // initial parameters
    Est.Cy(i).V=Ps*Ps';         // statistics
  end
  Est.Cy(i).V=Est.Cy(i).V;      // the weight is proportional to data
  Est.Cy(i).th=v2thN(Est.Cy(i).V,2);            // pt.est. of reg.coef
```

```
  Est.Cy(i).cv=.1*eye(2,2);  // standard deviation
  Est.Cy(i).tht(:,1)=Est.Cy(i).th;
end
Est.ka=ones(1,nc);            // counter
Est.Cp.V=ones(1,nc);          // pointer statistics
Est.Cp.th=fnorm(ones(1,nc));  // pointer parameter
w=fnorm(ones(1,nc));          // weights

// ESTIMATION ============================================================
I_estCov=0;
printf(' ')
for t=2:nd
  if t/fix(nd/10)==fix(t/fix(nd/10)), printf('.'); end
  for j=1:nc
    [xxx,G(j)]=GaussN(Sim.yt(:,t),Est.Cy(j).th,Est.Cy(j).cv);
                                              // proximity
  end
  Lq=G-max(G);
  q=exp(Lq);

  ww=q'.*Est.Cp.th; w=ww/sum(ww);           // generation of weights
  wt(:,t)=w';
```

```
  // Update of statistic
  Ps=[Sim.yt(:,t)' 1];                        // extended reg.vec.
  for i=1:nc
    Est.Cy(i).V=Est.Cy(i).V+w(i)*Ps'*Ps;      // information matrix
    Est.ka(i)=Est.ka(i)+w(i);                 // counter
    Est.Cp.V(i)=Est.Cp.V(i)+w(i);             // pointer statistics

    //nove rozdeleni informacni matice V
    Vy=Est.Cy(i).V(1:2,1:2);                  // part Vy  - y.y
    Vyp=Est.Cy(i).V($,1:2);                   // part Vyp - psi.y
    Vp=Est.Cy(i).V($,$);                      // part Vp  - psi.psi'
    Est.Cy(i).th=inv(Vp+1e-8*eye(Vp))*Vyp;    // pt.est. - reg.coef.
    Est.Cy(i).tht(:,t)=Est.Cy(i).th';         // pt.est. - covar.
    if t>200, I_estCov=1; end
    if I_estCov~=0
      // PT.EST. OF NOISE COVARIANCE - USED OR NOT
      Est.Cy(i).cv=(Vy-Vyp'*inv(Vp+1e-8*eye(Vp))*Vyp)/Est.ka(i);
    end
  end
  Est.Cp.th=fnorm(Est.Cp.V,2);     // pt.est. of pointer parameter
  [ss,Est.ct(1,t)]=max(w);         // store
end
disp ' '
```

```
// Results
bigfig(1)
for i=1:nc
//subplot(1,nc,i)
plot(Est.Cy(i).tht(1,:),Est.Cy(i).tht(2,:),'.')
plot(Est.Cy(i).tht(1,1),Est.Cy(i).tht(2,1),'g.','markersize',14)
plot(Sim.Cy(i).th(1),Est.Cy(i).th(2),'r.','markersize',8)
end
title 'zelené kolečko - kde to začlo, červená tečka - správná centra'
```

Description of the program

The program shows a standard mixture estimator with normal components and simulated data. The pre-classified estimation method starts already in the simulation, where we prepare both the prior data for the initialization of *Sim.yi* and the actual measured data for the estimation of *Sim.yt*. In the initialization, we use `find()` to determine which of the prior data belong to which component. In the next section, we estimate each component from the prior data and complete the initialization of the pointer model in the standard way. Next, we use the results of the initialization as initial statistics and parameter estimates.

The second program shows simplified applications of the above method.

```
// ini8bPreClass.sce
// Inicializace s učitelem (jednoduchá varianta)
```

```
// ----------------------------------------
[u,t,n]=file();                          // find working directory
chdir(dirname(n(2)));                    // set working directory
clear("u","t","n")                       // clear auxiliary data
exec("ScIntro.sce",-1),mode(0)           // intro to sesion
rand('seed',0)

nd=500;             // number of data
ni=20;              // lenght of initialization phase
t_estCov=1;         // when estimation of covariances is switched on

// simulated reg.coef.
Sim.Cy(1).th=[1.9 -.5]';
Sim.Cy(2).th=[0.4 0.6]';
Sim.Cy(3).th=[-1.2 1.6]';
Sim.Cy(4).th=[.1 -1.6]';
Sim.Cy(5).th=[2.1 2.2]';
nc=length(Sim.Cy);
Sim.nc=nc;
// simulated noise covariances
rcv=.8;                        // amplitude of covariances
for i=1:nc
  g=rand(2,2,'u');
```

```
  Sim.Cy(i).cv=rcv*(eye(2,2)+(g+g'));
end
// simulated pointer parameters
Sim.Cp.th=fnorm(ones(1,nc)+.1,2);
Sim.ct(1)=1;                  // initial poiner

// SIMULATION ============================================================
for t=1:nd
  Sim.ct(1,t)=sum(rand(1,1,'u')>cumsum(Sim.Cp.th))+1;    // pointer
  j=Sim.ct(t); // active component
  Sim.yt(:,t)=Sim.Cy(j).th+uut(Sim.Cy(j).cv)*rand(2,1,'norm'); // output
end

// INITIALIZATION ========================================================
// initial parameters
for i=1:nc                    //  from those used in simulation
  Est.Cy(i).V=eye(3,3);
  Est.Cy(i).th=rand(2,1);            // pt.est. of reg.coef
  Est.Cy(i).cv=.1*eye(2,2);  // standard deviation
  Est.Cy(i).tht(:,1)=Est.Cy(i).th;
end
Est.ka=ones(1,nc);            // counter
Est.Cp.V=ones(1,nc);          // pointer statistics
```

```
Est.Cp.th=fnorm(ones(1,nc));  // pointer parameter
w=fnorm(ones(1,nc));          // weights

// ESTIMATION ============================================================
I_estCov=0;
printf(' ')
for t=2:nd
  if t/fix(nd/10)==fix(t/fix(nd/10)), printf('.'); end
  for j=1:nc
    [xxx,G(j)]=GaussN(Sim.yt(:,t),Est.Cy(j).th,Est.Cy(j).cv);
                                          // proximity
  end
  Lq=G-max(G);
  q=exp(Lq);

  ww=q'.*Est.Cp.th; w=ww/sum(ww);           // generation of weights
  // KNOWN POINTERS FOR INITIAL DATA
  if t<=ni, w=zeros(1,nc); w(Sim.ct(t))=1; end
  wt(:,t)=w';

  // Update of statistic
  Ps=[Sim.yt(:,t)' 1];                      // extended reg.vec.
  for i=1:nc
```

```
    Est.Cy(i).V=Est.Cy(i).V+w(i)*Ps'*Ps;     // information matrix
    Est.ka(i)=Est.ka(i)+w(i);                // counter
    Est.Cp.V(i)=Est.Cp.V(i)+w(i);            // pointer statistics

    //nove rozdeleni informacni matice V
    Vy=Est.Cy(i).V(1:2,1:2);                 // part Vy  - y.y
    Vyp=Est.Cy(i).V($,1:2);                  // part Vyp - psi.y
    Vp=Est.Cy(i).V($,$);                     // part Vp  - psi.psi'
    Est.Cy(i).th=inv(Vp+1e-8*eye(Vp))*Vyp;   // pt.est. - reg.coef.
    Est.Cy(i).tht(:,t)=Est.Cy(i).th';        // pt.est. - covar.
    if t>200, I_estCov=1; end
    if I_estCov~=0
      // PT.EST. OF NOISE COVARIANCE - USED OR NOT
      Est.Cy(i).cv=(Vy-Vyp'*inv(Vp+1e-8*eye(Vp))*Vyp)/Est.ka(i);
    end
  end
  Est.Cp.th=fnorm(Est.Cp.V,2);     // pt.est. of pointer parameter
  [ss,Est.ct(1,t)]=max(w);         // store
end
disp ' '

// Results
bigfig(1)
```

```
for i=1:nc
//subplot(1,nc,i)
plot(Est.Cy(i).tht(1,:),Est.Cy(i).tht(2,:),'.')
plot(Est.Cy(i).tht(1,1),Est.Cy(i).tht(2,1),'g.','markersize',14)
plot(Sim.Cy(i).th(1),Est.Cy(i).th(2),'r.','markersize',8)
end
title 'zelené kolečko - kde to začlo, červená tečka - správná centra'
```

This program is practically identical to the previous one. However, we do not complicate the situation by a special initialization, but simply use the prior data and the known pointer values at the beginning of the estimation. We enforce the pointer value by replacing the estimated weights for the prior data with a vector of zeros and with a one at the point where the pointer points.