Inicializace odhadu směsi distribucí

Odhad směsi spočívá v odhadu parametrů jednotlivých komponent a modelu ukazovátka. předpokládá se, že měřená data přicházejí postupně z různých pracovních módů systému a patří tedy do různých komponent modelu směsi. Prvním krokem při zpracování každého datového vektoru je tedy určení, ke které komponentě (nebo několika komponent) data patří. To se děje především pomocí proximity (blízkost jednotlivých komponent ke změřenému datovému vektoru). Tato proximity se určí tak, že se do komponent dosadí existující bodové odhady parametrů a zjišťuje se, jakou hodnotu mají komponenty (jejich hustoty pravděpodobnosti) ve změřených datech. Čím dále budou data od centra komponenty, tím bude hodnota proximity menší. Z proximity se pak určí váhy komponent vzhledem k naměřenému datovému vektoru a podle toho se data použijí k přepočtu statistik komponent.

Z uvedeného plyne: Budou-li data daleko od počátečních komponent, budou váhy prakticky nulové a k žádnému odhadu nedojde. Proto je potřeba

1. Nastavit počáteční polohy tak, aby ležely v oblasti, kde se vyskytují data.
2. Zabránit tomu, aby hned na začátku některá komponenta "neutekla" nebo aby s komponenty nepřekryly.

K tomu slouží inicializace odhadu směsi - tj. vhodné rozmístění počátečních komponent a jejich částečná fixace (pomalejší pohyb) během začátku odhadování. Přitom předpokládáme, že máme k dispozici apriorní vzorek dat (tj. data, získaná v minulosti, která jsou k dispozici ještě před začátkem odhadu z průběžně měřených dat).

Poznámka

Pokud apriorní data nejsou, zbývá jediná cesta. Všechny veličiny škálovat (upravit tak, aby měly přibližně nulovou střední hodnotu a jednotkový rozptyl a dále pracovat na této normalizovaná datové oblasti. Výsledky pal lze opět převést do původní metriky.

Obecné zásady pro inicializaci jsou následující:

1. Najít si oblast, kde se vyskytují měřená data. Např. zjistit minima a maxima u jednotlivých veličin, nebo lépe prohlédnout si jejich histogramy.
2. Nastavit počáteční hodnoty odhadu parametrů tak dobře, jak je to jen možné (s využitím apriorních dat i expertní informace).
3. Na začátku odhadování přidržíme apriorní odhady center komponent, aby se nerozutekly příliš daleko nebo se nepřekryly.
4. U komponent necháme fixní malé kovariance (pokud nám záleží na tvaru klastrů, spustíme jejich odhadování později, kdy centra komponent již budou více méně správně určeny).
5. Provést opakovaný odhad na stejném datovém vzorku. Při tom je třeba místo apriorních parametrů zadat dosavadní odhady a statistiky nechat apriorní (aby nebyly utažené).
6. Pro dynamické směsi je dobré začít s komponentami s potlačenou dynamikou (tedy ve tvaru statických). Počáteční umístění komponenty je pak dáno přepočtenou konstantou.
7. Použít uměle vytvořené regresní modely a k nim expertně určit odpovídající výstup. Tato umělá data použít pro inicializaci.

8. Provést expertní klasifikaci několika apriorních nebo uměle vytvořených dat a využít je pro inicializaci.

Jednotlivé body budeme dále ilustrovat teoreticky a v příkladech. Budeme uvažovat statické komponenty, tedy modely ve tvaru $y_t = k_i + e_t$ (kde k_i jsou centra komponent pro $i = 1, 2, \cdots, nc$)

1. Oblast dat

Dejme tomu, že máme 3 veličiny x_1, x_2 a x_3 uspořádané v datové matici x se třemi řádky a tolika sloupci, kolik je počet měření. Potom

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

dá 3-prvkové vektory minimálních a maximálních hodnot veličin. Souřadnice počátečních komponent můžeme do tohoto prostoru rozmístit příkazem

for i=1:nc

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

end

kde (mi+ma)/2 je střed oblasti, (ma-mi) je šířka oblasti a nd je počet komponent.

Program

```
// Určení oblasti dat
// ---------------------------------------
exec SCIHOME/ScIntro.sce, mode(0)

nd=100;             // počet dat
for t=1:nd
  y(:,t)=[5;3;8]+[2;1;3].*rand(3,1,'n');
end

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

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

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

Popis programu

V programu jsou počáteční centra "rozházena" kolem středu oblasti dat (max(y)+min(y))/2. Velikost "rozházení" je dána jakýmsi poloměrem oblasti dat max(y)-min(y).

2. Počáteční odhady parametrů

Na počátečním umístění parametrů (center klastrů) extrémně záleží. Centra by rozhodně měla ležet v oblasti, kde se vyskytují data a v ideálním případě by jednotlivá centra měla ležet poblíž vrcholků dat, tj. v místech, kde s vyskytují hustotní maxima předpokládaných klastrů (pracovních módů systému).

Pokud **nemáme apriorní data**, postupujeme podle předchozího bodu (data škálujeme a centra rozhodíme kolem počátku).

Pokud **jsou apriorní data** k dispozici (a měly by být, protože proces nějak už běžel - třeba jen zkušebně - a většinou se stačí jen vynaložit určité úsilí a data se seženou), rozhodně je chceme využít. Především určíme oblast, kde se data vyskytují (viz předchozí bod) a potom hledáme hustotní vrcholky - buď v histogramech jednotlivých veličin nebo ve dvojicích veličin (více nepřehlédneme). Histogramy jsou jasné. Ukážeme postup pro dvojice:

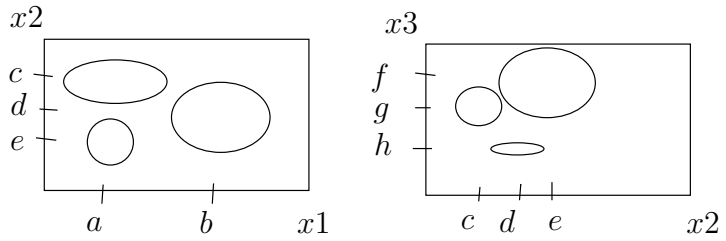
Máme 3 veličiny $x1$, $x2$ a $x3$. Vykreslíme xy-grafy pro dvojice $x1$-$x2$ a $x2$-$x3$.

$$\text{plot(x1,x2,':')} \quad \text{a} \quad \text{plot(x2,x3,'.')}$$

Na ose x prvního grafu najdeme souřadnice, korespondující s viditelnými klastry a na ose y jim odpovídající y-ove souřadnice. Může se stát, že k jedné x-ové souřadnici bude více y-ových - pak zaznamenáme všechny s tím, že x-ové se opakují.

Přejdeme k druhému obrázku a na jeho x-ové ose vyznačíme všechny y-ové souřadnice z prvního obrázku. K nim dourčíme y-ové souřadnice z druhého obrázku a přidáme je jako třetí číslo k existujícím souřadnicím. Tak můžeme pokračovat i pro více souřadnic. Výsledné souřadnice použijeme jako centra komponent, tj. apriorní parametry statických komponent.

Postup je znázorněn na obrázku:



Centra z prvního obrázku budou:

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

Z druhého obrázku doplníme

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

To nemusí nutně být pravá centra mnohorozměrných komponent, ale alespoň víme, že tady se něco děje a počáteční centra nějak patří hustotním vrcholkům. K doladění by mělo dojít při vlastním odhadu.

Program

```
// Počáteční odhady parametrů a k nim statistiky
// ---------------------------------------
exec SCIHOME/ScIntro.sce, mode(0), rand('seed',0)

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

// simulated reg.coef.
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);
nc=nc;
// simulated noise covariances
for i=1:4
  rn=rand(3,3,'u');
  Cy(i).cv=.2*(eye(3,3)+rn+rn');
end
// simulated pointer parameters
Cp.th=fnorm(ones(1,3)+.1,2);

ct(1)=1;                  // initial poiner
// SIMULATION ===========================================================
for t=1:nd
  ct(1,t)=sum(rand(1,1,'u')>cumsum(Cp.th))+1;    // pointer
  j=ct(t); // active component
  y=Cy(j).th+uut(Cy(j).cv)*rand(3,1,'norm');        // output
  yt(:,t)=y;
end

ii=[2 3 6];
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,:),'.')
  end
end
```

Popis programu

V programu jsou simulovány 4 komponenty směsi, jsou vykresleny xy-grafy veličin $y1 - y2$, $y2 - y3$ a $y2 - y3$. Z nich můžeme odečíst tří-rozměrná centra komponent.

3. Přidržení apriorních center

Je to velmi důležitá metoda, používaná více nebo méně v každém odhadu!!!

Na počátku odhadu se informace o parametrech čerpá jen z malého počtu dat. Pokud bychom parametry nechali zcela volné, "vrhaly" by se nesmyslně za každým změřeným datovým vektorem

a snadno by mohly zabloudit někam, odkud by už nebyl návratu. Proto je třeba startovat se statistikami, které v sobě již mají nějakou informaci - buď z dat nebo expertní.

Situaci budeme demonstrovat pro normální komponenty. V ostatních případech je situace podobná.

Přepočet statistik pro odhad regresních koeficientů se děje takto

$$V_t = V_{t-1} + \Psi\Psi'$$

kde V je informační matice, $\Psi = [y_t,\, 1]'$ je rozšířený regresí vektor s novými daty. Z tohoto je patrné, že matice V postupně roste jak se do ní načítají data.

Zároveň je zřejmé, že je-li matice V na začátku nulová, ihned první data ji velmi změní. Je-li ale dostatečně veliká, počáteční data na ní nemají příliš velký vliv.

Bodové odhady regresních koeficientů se dějí podle vzorce

$$\hat{\theta} = V_\psi^{-1} V_{y\psi}$$

kde V_y, $V_{y\psi}$a V_ψ jsou submatice V dělené podle regresního vektoru (obecně je $\Psi = [y_t,\, \psi_t]'$; tady u statických komponent je $\psi_t = 1$). Násobíme-li matici V číslem c bude

$$\hat{\theta} = \left(cV_\psi\right)^{-1} \left(cV_{y\psi}\right) = V_\psi^{-1} V_{y\psi}$$

tedy odhady se nezmění - jen (pro velké c) bude cV větší, a tedy odolnější proti změnám vlivem počátečních dat.

Závěr je tedy docela jednoduchý: Velká informační matice slouží k přidržení počátečních center komponent.

Poznámka

Pokud máme na mysli nějaké počáteční parametry $\hat{\theta}_0$ a chceme k nim sestrojit informační matici, postupujeme takto

$$V = \begin{bmatrix} 1 & \hat{\theta}_0' \\ \hat{\theta}_0 & 1 \end{bmatrix}.$$

Potom první odhad $\hat{\theta} = V_\psi^{-1} V_{y\psi} = \hat{\theta}_0$.

Program

```
// Přidržení apriorních parametrů
// ---------------------------------------
exec SCIHOME/ScIntro.sce, mode(0)

nd=200;             // number of data for estimation
ni=1;               // number of initial data  ni>0

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
               //   as if how many initial data items have been used.

// Estimation
for t=2:nd
  Ps=[y(t) u(t) y(t-1) u(t-1) 1]';
  V=V+Ps*Ps';
  Vy=V(1:2,1);
  Vyp=V(2:$,1);
  Vp=V(2:$,2:$);
  th=inv(Vp+1e-12*eye(Vp))*Vyp;
  tht(:,t)=th';                              // evolution of 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'
```

Popis programu

Odhaduje se skalární regresní model 1. řádu, s parametry značenými malými písmeny. Inicializace informační matice V je nastavena tak, aby se z ní spočetly počáteční parametry (označené velkými písmeny). Zkonstruovaná matice V se násobí číslem ni (to je jako počet dat, ze kterých byla matice V určena). Podle velikosti čísla ni se se dále mění odhady parametrů. Pro malé ni rychle, pro velké ni pomalu. Na výstupu sledujeme vývoj odhadu parametrů v čase.

4. Fixní kovariance šumu

Kovariance šumu určují tvar klastrů. Pokud nám jde především o nalezení center klastrů, můžeme kovariance ponechat malé a fixní (neodhadovat je). Zadáváme je jako jednotková matice násobené desetinou až setinou rozsahu (poloměru) předpokládané datové oblasti.

Pokud nám na tvaru klastrů záleží, třeba při klasifikaci, kdy data dělíme do jednotlivých komponent, doporučuje se, zapnout jejich odhadování až v průběhu odhadu (třeba v polovině), kdy už budou centra v podstatě nalezena. Pořád ale hrozí nebezpečí, že kovariance utečou nebo že, jedna komponenta překryje ostatní - proto pozor.

Program

```
// Fixní kovariance při odhadu směsi
// ---------------------------------------
exec SCIHOME/ScIntro.sce, mode(0)

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(rand(1,1,'u')>cumsum(Sim.Cp.th))+1;    // pointer
  j=Sim.ct(t); // active component
  y=Sim.Cy(j).th+Sim.Cy(j).sd*rand(2,1,'norm');        // output
  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*rand(mr,mc,'n');0]; // initial parameters
  Est.Cy(j).V=Ps*Ps';                       // statistics
  Est.Cy(j).th=Sim.Cy(j).th+a*rand(2,1,'n'); // 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
disp ' '
s=2:nd;
[q,T]=c2c(Sim.ct(s),Est.ct(s));
Ect=q(Est.ct(s));

S=list();
for i=1:nc
  j=find(Sim.ct==i);
  S(i)=Sim.yt(:,j);
end

C=list();
for i=1:nc
  j=find(Ect==i);
  C(i)=Sim.yt(:,j);
```

```
end

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

bigfig(1)
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)

bigfig(2)
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'
```

nebo

```
// Fixní kovariance při odhadu směsi (odhad více komponent než v sim)
// ----------------------------------------
exec SCIHOME/ScIntro.sce, mode(0)

nd=500;             // number of data
nc=8;               // number of componentd
t_estCov=50;       // when estimation of covariances 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(rand(1,1,'u')>cumsum(Sim.Cp.th))+1;    // pointer
  j=Sim.ct(t); // active component
  y=Sim.Cy(j).th+uut(Sim.Cy(j).cv)*rand(2,1,'norm');        // 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
  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>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
disp ' '
s=2:nd;
Sc=Sim.ct(s);
Ec=Est.ct(s);
yt=Sim.yt(:,s);
S=list();
for i=1:2
  j=find(Sc==i);
  S(i)=yt(:,j);
end

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

// 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(1,2),Est.Cy(i).tht(2,2),'g.','markersize',18)
end
title 'zelené klečko - kde to začlo'

bigfig(2)
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
```

Popis programu

O odhadu nebo ne-odhadu kovariancí šumů rozhoduje parametr I_estCov. Je-li roven 0, kovariance se neodhadují a zůstávají ty, co jsme nastavili v inicializaci. Pro I_estCov = 1 se odhadují. Kompromisem je jejich zpožděný odhad, při kterém se začnou odhadovat až od zadaného času t_estCov.

5. Opakovaný odhad na stejných datech

Centra komponent startují ve svých počátečních centrech a postupně putují k hustotním vrcholkům. Každý datový záznam je trochu posune podle toho, jak do které komponenty patří (podle vah w). Pokud je dat málo, může se stát a taky se stává, že odhad končí ještě před tím, než centra doputovala. Potom je rozumné pokračovat v odhadu se stejnými daty znovu, ale začít ne v počátečních centrech, ale z těch, ke kterým zatím doputovala.

Je tu ale jeden problém. Na konci odhadu jsou informační matice velké (říkáme, že odhad je utažen) a centra by se už buď nepohybovala, nebo jen velmi málo. Proto je třeba postupovat následovně:

1. Počáteční centra komponent nastavíme na hodnoty ze skončeného odhadu.
2. Statistiky z odhadu zahodíme a použijeme zase ty původní z prvního odhadu.

Takto můžeme postupovat opakovaně. Při tom je dobré sledovat vývoj center třeba v grafu a pokračovat do té doby, dokud se odhady center pohybují.

Program ilustruje situaci pro dva odhady

```
// Opakovaný odhad na stejných datech
```

```
// ----------------------------------------
exec SCIHOME/ScIntro.sce, mode(0)

nd=10;
k=[2; -1]; sd=.01;

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

// Inicializace k prvnímu odhadu
kk=[-5;8];                    // poč. hodnota par. je kk
V0(1:2,1:2)=kk*kk';
V0(3,1:2)=kk'; V0(1:2,3)=kk; V0(3,3)=1; // nastavení V
V0=V0*10; V=V0;               // V schválně trochu fixujeme - V0=V0*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=[kk th1];                 // z výchozího odhadu kk 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'
```

Popis programu

V programu se uvažuje odhad statického regresního modelu s dvourozměrnými daty. Odhad je inicializován (konstantou kk) a spuštěn. Výsledkem prvního odhadu je informační matice V a z ní spočtené bodové odhady parametrů. Následuje druhý odhad se stejnými daty, ale v informační maticí spočtenou v předchozím běhu. Tuto matici je vhodné podrobit zapomínání (dělení celé matice číslem fi). Odhad potom začne z bodových odhadů parametrů z předchozího kroku. Zapomínání informační matici uvolní, aby se odhady opět mohly pohybovat. Zkuste nabízené koeficienty zapomínání fi a sledujte jejich účinek.,

6. Dynamické směsi

Na rozdíl od statických směsí, jejichž komponenty mají ve skutečnosti pevná centra - střední hodnoty komponent (vrcholky hustotních kopečků), dynamické směsi mají střední hodnotu závislou na datech a tedy pohyblivou. Odhady jejich center tedy nekonvergují k pevným bodům, ale k pohyblivým. Jejich počáteční nastavení je tedy komplikovanější.

Ukážeme postup, který vychází ze statických komponent a dynamika se postupně přidává při odhadu.

Budeme uvažovat jednu dynamickou komponentu 1. řádu

$$y_t = ay_{t-1} + bu_t + k + e_t$$

1. Jako apriorní parametry vezmeme $a = 0$ a $b = 0$. Tomu odpovídá statický model $y_t = k + e_t$.
2. Zjistíme nebo odhadneme průměrnou hodnotu y jako $\bar{y}$.
3. Počáteční konstantu k nastavíme na hodnotu $\bar{y}$.
4. Počáteční informační matici určíme takto

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

   kde $\Psi_0 = [0,\, 0,\, k]'$, $k = \bar{y}$ a I je jednotková matice.

Odpovídající program je následující

```
// Inicializace dynamických komponent
// ------------------------------------------
exec SCIHOME/ScIntro.sce, mode(0)

// 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'

function [th,s2]=v2thN(v,m)
  // [th,s2]=v2thn(v,m)   computation of par. point estimates
  //                      from normalized information matrix
  // v    information matrix
  // m    dimension of y
  // th   regression coefficients
  // s2   noise cobariance estimate
```

```
  if argn(2)<2    // check for number of input arguments
    m=1;
  end

  // partitioning of information matrix
  vy=v(1:m,1:m);
  vyf=v(m+1:$,1:m);
  vf=v(m+1:$,m+1:$);

  // computation of point estimates
  th=inv(vf+1e-12*eye(vf))*vyf;
  s2=(vy-vyf'*th);
endfunction
```

V programu uvažujeme dynamický regresní model

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

Tento model inicializujeme jako statický, tedy ve tvaru

$$y_t = k + e_t$$

kde očekáváme hodnotu k přibližně rovnu průměru z apriorních y. Přito doufáme, že pro statický model jsme se "trefili do dat" a dynamika se postupně do odhadne.

7. Umělé regresní vektory

Dobrý způsob jak převést často abstraktní expertní znalost do podoby dat, která jsou vhodná pro inicializaci odhadu, je tzv. tvorba umělých datových vektorů. Každý datový vektor se skládá z regresního vektoru a jemu odpovídající hodnoty výstupu. Situaci budeme ilustrovat na příkladě:

Sledujeme délku kolony v jenom rameni řízené křižovatky, která sbírá dopravu z určité oblasti. V této oblasti je 5 kritických míst, která mohou být podle situace v různém stupni dopravy. Regresní vektor bude tedy obsahovat 5 veličin (stupně dopravy v daných místech oblasti) a výstupem je délka kolony v křižovatce. Expert svou znalost může převést na vývěr nejdůležitějších kombinaci zatížení jednotlivých míst v oblasti a jim přiřadit (podle svého přesvědčení) odpovídají hodnotu délky kolony v křižovatce.

Poznámka

Pokud jsou k dispozici apriorní data, je samozřejmě možno je využít a vybrat z nich některé důležité datové vektory, které jsou pro danou situaci rozhodující a nesou hodně informace. Výběr může být opět podle doporučení experta.

Zkonstruované datové vektory se potom zpravují normálně, jako měřené datové vektory v rámci inicializace.

Metodu ilustruje následující program. Popis následuje za programem.

```
// Umělé regresní vektory
// ----------------------------------------
```

```
exec SCIHOME/ScIntro.sce, mode(0), 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')
```

Popis programu

V programu se ukazuje odhad regresního modelu prvního řádu se třemi vstupy

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

Program je rozdělení na 2 části podle volby I_init. Pro volbu 1 se ukazují apriorní data a uživatel si z nich může vybrat důležité položky pro datové vektory Psi, které jsou dále využity pro inicializaci. Volba I_init=0 připraví běh "naostro", tj. připravenou inicializaci s následným odhadem. Ukazuje se vývoj odhadu parametrů.

8. Expertní klasifikace

Tato metoda sleduje předchozí postup, ale místo přiřazení hodnoty výstupu k regresnímu vektoru se vybraným datům expertně přiřazuje třída (komponenta), do které patří.

Opět je několik možností, jak tuto před-klasifikaci provést.

1. Expertně vytvoříme celý datový vektor a zařadíme jej do určité třídy.
2. Vezmeme nějaký apriori změřený datový vektor a expertně mu přiřadíme třídu.
3. Použijeme určité nadstandardní nástroje (necháme situaci pozorovat člověkem nebo půjčíme nějaký drahý měřící přístroj) abychom pro apriorní měření změřili nejen datové záznamy ale i příslušné třídy klasifikace.

To, co získáme, použijeme pro inicializaci, kdy provádíme učení s učitelem (tedy se znalostí správné klasifikace).

Následující program ukazuje tuto před-klasifikovanou inicializaci. Popis následuje za programem.

```
// Inicializace s učitelem
// ---------------------------------------
exec SCIHOME/ScIntro.sce, mode(0), 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'
```

Popis programu

Program ukazuje standardní odhad směsi s normálními komponentami a simulovanými daty. Metoda před-klasifikovaného odhadu začíná již v simulaci, kde si připravíme jednak apriorní data pro inicializaci Sim.yi, jednak vlastní měřená data pro odhad Sim.yt. V inicializaci pomocí příkazu find() určíme, která z apriorních dat patří které komponentě.V dalším odstavci provádíme odhad jednotlivých komponent z apriorních dat a standardním způsobem doplníme inicializaci modelu pointru. Následuje odhad, kde jako počáteční statistiky a odhady parametrů použijeme výsledky inicializace.

Druhý program ukazuje zjednodušená použití uvedené metody.

```
// Inicializace s učitelem (jednoduchá varianta)
// ---------------------------------------
exec SCIHOME/ScIntro.sce, mode(0), 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'
```

Tento program je prakticky shodný s předchozím. Situaci ale nekomplikujeme zvláštní inicializací, ale prostě apriorní data i se známým ukazovátkem použijeme na začátku odhadu. Ukazovátko vnutíme tak, že odhadnuté váhy pro apriorní data nahradíme vektorem nul a s jedničkou na místě, kam ukazuje ukazovátko.