Odhad směsi se statickým ukazovátkem i komponentami[1]

- dvourozměrný výstup, bez řízení
- simulovaná data
- inicializace odhadu - zašuměné parametry ze simulace
- standardní odhad / odhad s pevnými kovariancemi šumů komponent
- model ukazovátka $f(c_t|\alpha) = \alpha_{c_t}$

Simulují se data ze směsi normálních, dvourozměrných regresních komponent a statického ukazovátka

$$y_t = \theta_i + e_t, \;\; i = 1, 2, \cdots, n_c$$

kde y_t je výstup v čase t,

$\theta_i = \begin{bmatrix} (\theta_1)_i \\ (\theta_2)_i \end{bmatrix}$ jsou parametry statických komponent s dvourozměrným výstupem,

n_c je počet komponent.

Předpoklady: $e \sim N(0, r)$, r konstantní.

Sci značení: y - yt, θ - th, r - cv.

Úloha: Simulace a odhad s modelem směsi statických komponent - základní konfigurace pro odhad směsi.

Poznámka

Inicializace odhadu směsi komponent je velice důležitá. Pokud nejsou počáteční centra komponent dostatečně blízko dat, jsou váhy w_t prakticky nulové (hodnota distribuce normálního rozdělení velmi rychle klesá se vzdáleností od střední hodnoty) a odhad je velmi nepřesný nebo dokonce selhává. Proto je vždy třeba počátečnímu nastavení center (středních hodnot) a šířce (kovarianční matici) odhadovaných komponent věnovat patřičnou péči.

Zde pro inicializaci využijeme znalost "skutečných" parametrů ze simulace. Jako počáteční hodnoty parametrů v odhadu použijeme skutečné hodnoty parametrů, a ty více nebo méně zašumíme.

Tento trik v inicializaci, který je z praktického hlediska "nefér" použijeme proto, že nám zde jde spíše o testování odhadu směsi (při kterém zkoušíme, co tento odhad dokáže), než o skutečný odhad, např. pro reálná data, kde samozřejmě "skutečné" parametry neznáme.

Doporučené experimenty

1. Komponenty modelu jsou statické. Tedy každá komponenta je gaussovský kopeček se středem daným regresními koeficienty *th* a šířkou úměrnou rozptylu *cv*. Protože komponenty jsou dvourozměrné, pohybujeme se v rovině a středy jsou body. Volbou středů a rozptylů můžeme volit klastry, které se překrývají nebo naopak jsou prakticky disjunktní. Podle

[1] Tato úloha je paralelní s úlohou T71MixDyn a liší se jen modelem ukazovátka, který je zde statický.

toho je možno nastavit situaci, kdy klasifikace je triviální a čekáme, že všechny rozhodnutí budou správné nebo naopak, kdy správní klasifikace je velmi obtížná a jsme spokojeni i s určitým procentem správných rozhodnutí.
Zkuste nastavit své vlastní soustavy a ověřte procento správných klasifikací.

2. Odhad modelu směsi je dosti obtížný a lze konstatovat, že bez nějaké inicializace (nalezení středů komponent poblíže center skutečných klastrů) nebude úspěšný. V této úloze se používají počáteční parametry odvozené od skutečných parametrů soustavy (parametry ze simulace).
Zkuste tyto parametry nastavit ručně a sledujte jejich vliv na odhad (zejména jeho počáteční fázi).

3. Jedna z velice úspěšných možností jak přimět odhad, aby se na začátku "chytil", je buď neodhadovat rozptyly komponent a ponechat je počáteční malé nebo s jejich odhadem začít až v průběhu odhadování. Odhad/neodhad rozptylů je možno zvolit pomocí parametru *IstCov*. Zpožděné odhadování je třeba doprogramovat: if t>100, $\cdots$, end
Zkuste odhadovat různými způsoby a porovnejte výsledky.

Program

```
// P71Mix1Stat.sce
// Mixture estimation - static components and pointer model
// - simulated data two-dimensional data
// - initialization by parameters from simulation + noise
// - with or withou estimation of covariances of components
//   FOR NC>5 IT IS NECESSARY TO SET PARAMETERS FOR SIMULATION
[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=100;             // number of data
nc=3;               // number of componentd
I_estCov=0;         // estimation of noise covariances ? 0|1 no|yes

// simulated reg.coef.
Sim.nc=nc;
Sim.Cy(1).th=[0.9 -.5]';
Sim.Cy(2).th=[0.4 0.6]';
Sim.Cy(3).th=[-.7 -.5]';
Sim.Cy(4).th=[-.1  .8]';
Sim.Cy(5).th=[ .5 -.1]';
// simulated noise covariances
Sim.Cy(1).sd=0.1*[1 0;0,1];
Sim.Cy(2).sd=0.1*[1 0;0,1];
Sim.Cy(3).sd=0.1*[1 0;0,1];
Sim.Cy(4).sd=0.1*[1 0;0,1];
Sim.Cy(5).sd=0.1*[1 0;0,1];
// simulated pointer parameters
```

```
Sim.Cp.th=fnorm(ones(1,nc)+.1,2);

Sim.ct(1)=1;                  // initial poiner

// initial parameters
a=.8;                         // std of scattering initial parametrs
for j=1:nc                    //  from those used in simulation
  [mr,mc]=size(Sim.Cy(j).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=.1*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

// SIMULATION ============================================================
for t=2:nd
  Sim.ct(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

// ESTIMATION ============================================================
printf(' ')
for t=2:nd
  if t/10==fix(t/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
    Vyy=Est.Cy(i).V(1:2,1:2);               // part Vyy - psi.psi'
```

```
    Vy=Est.Cy(i).V($,1:2);                  // part Vy  - psi.y
    V1=Est.Cy(i).V($,$);                    // part V1  - y.y
    Est.Cy(i).th=inv(V1+1e-8*eye(V1))*Vy;   // 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=(Vyy-Vy'*inv(V1+1e-8*eye(V1))*Vy)/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

// Results
disp(Sim.Cp.th,'pt.pars_sim')
disp(Est.Cp.th,'pt.pars_est')

s=2:nd;
wr=sum(Sim.ct(s)~=Est.ct(s)');
printf('\n Wrong classifications %d from %d\n',wr,length(s))

set(scf(1),'position',[50 50 400 400])
plot(s,Sim.ct(s),'bo',s,Est.ct(s),'rx')
legend('simulated','estimated');
title 'Values of the pointer'

set(scf(2),'position',[550 50 400 600])
for i=1:nc
  subplot(nc,1,i)
  plot(Est.Cy(i).tht')
  title('Parameters estimate evolution '+string(i))
end
```

```
// P71Mix1StRe.sce
// Mixture estimation - static components and pointer model
// - real two-dimensional data
// - initialization from expert knowledge
```

```
//   initial parameters are set through simulation structures + noise
// - with or without noise covariances estimation
[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=100;             // number of data
nc=3;               // number of components
I_estCov=0;         // estimation of noise covariances ? 0|1 no|yes

// MODEL INITIALIZATION
// Here expertly based initial parameters are set
// (they are copied to estimation structure Est)
// initial regression coefficients
Sim.nc=nc;
Sim.Cy(1).th=[0.9 -.5]'; // HERE SET THE INITIAL CENTERS OF COMPONENTS
Sim.Cy(2).th=[0.4 0.6]'; //  -"-
Sim.Cy(3).th=[-.7 -.5]'; //  -"-
// initial (or fixed) covariances
Sim.Cy(1).sd=0.1*[1 0;0,1];   // covariances of components
Sim.Cy(2).sd=0.1*[1 0;0,1];   //  for I_estCov=0; stay fixed
Sim.Cy(3).sd=0.1*[1 0;0,1];   //  (they are not estimated)
// initial pointer probabilities
Sim.Cp.th=fnorm([1 1 1]+.1,2);

Sim.ct(1)=1;                  // initial poiner value

// initial parameters
a=.8;                         // std of scattering initial parametrs
for j=1:nc                    //  from those used in simulation
  Ps=[Sim.Cy(j).th;1];        // 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=.1*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

// DATA ========================================================
load _data/dataAuto.dat        // HERE, THE DATA ARE LOADED
Sim.yt=dt([1 2],1:nd);   // DATA ASSIGNMENT (time runs in rows)

// ESTIMATION ==================================================
printf(' ')
for t=2:nd
```

```
  if t/10==fix(t/10), printf('.'); end
  for j=1:nc
    [xxx,G(j)]=GaussN(Sim.yt(:,t),Est.Cy(j).th,Est.Cy(j).sd); // likelihood
  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
    Vyy=Est.Cy(i).V(1:2,1:2);                // part Vyy - psi.psi'
    Vy=Est.Cy(i).V($,1:2);                   // part Vy  - psi.y
    V1=Est.Cy(i).V($,$);                     // part V1  - y.y
    Est.Cy(i).th=inv(V1+1e-8*eye(V1))*Vy;    // 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=(Vyy-Vy'*inv(V1+1e-8*eye(V1))*Vy)/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

// Results
s=2:nd;
set(scf(2),'position',[550 50 400 600])
for i=1:nc
  subplot(nc,1,i)
  plot(Est.Cy(i).tht')
  title('Evolution of parameter estimatess '+string(i))
end
```

Popis programu

1. Simulace hodnot ukazovátka a podle toho generování dat z příslušné komponenty.
2. Odhad se provádí podle standardních vzorců:
   (a) výpočet vah w_t pro změřený výstup y_t.
   (b) Přepočet statistik komponent a ukazovátka.
   (c) Konstrukce bodových odhadů parametrů. Tady je možnost volby:
      i. průběžné odhadování kovariančních matic komponent,
      ii. použití počátečních kovariančních matic bez průběžného přepočtu.
      Tato varianta je bezpečnější. Při průběžném odhadu se může stát, že jedna komponenta překryje ostatní a výsledek je daný právě jen touto komponentou.
      Možná je také varianta, kdy v prvé části odhadování ponecháme kovarianční matice pevné, a v další části je již odhadujeme.
3. Jako výsledek se ukazují hodnoty odhadovaného ukazovátka ve srovnání se simulovaným. Tím úloha získává charakter klasifikace.