Odhad směsi s dynamickým ukazovátkem a statickými komponentami[1]

- dvourozměrný výstup, bez řízení
- simulovaná data
- inicializace odhadu - zašuměné parametry ze simulace
- standardní odhad / odhad s pevnými kovariancemi šumů komponent
- model ukazovátka $f\left(c_t|c_{t-1},\alpha\right)=\alpha_{c_t|c_{t-1}}$

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

Sci značení: y - yt, θ - th, r - cv.

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

Poznámky

1. *Dynamické ukazovátko umožňuje predikci aktivní komponenty, protože dává do souvislosti po sobě jdoucí aktivity komponent.*
2. *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.

[1] Tato úloha je paralelní s úlohou T71MixStat a liší se jen modelem ukazovátka, který je zde dynamický.

Doporučené experimenty

(Jsou stejné, jako pro statickou směs. Zde je uvedeme jen zkráceně. Podrobněji jsou v T71Mix1Stat)

1. Zkuste nastavit své vlastní soustavy a ověřte procento správných klasifikací.
2. Zkuste nastavit parametry 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.
4. (nové) Dynamické ukazovátko znamená, že výběr aktivní komponenty závisí na minulé aktivní komponentě. Tedy v přepínání aktivních komponent je určitý řád, daný modelem ukazovátka (kategorický dynamický model). Předvolený model ukazovátka je dán rovnoměrnou tabulkou *Sim.Cp.th*.
   Zvolte svůj vlastní model ukazovátka (např. deterministický) a sledujte efekt, který do odhadu takové směsi přináší.

Program

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 a 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.

Kód programu

```
// P71Mix2Dyn.sce
// Mixture estimation - static components and dynamic pointer model
// - simulated two-dimensional data
// - initialization by parameters from simulation + noise
// - dynamic components
[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]';
// noise covariances
r=.1;                               // amplitude of noise covariances
Sim.Cy(1).sd=r*[1 0;0,1];
Sim.Cy(2).sd=r*[1 0;0,1];
Sim.Cy(3).sd=r*[1 0;0,1];
// simulated noise covariances
Sim.Cp.th=fnorm(rand(nc,nc,'u')+.1,2);

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

// initial parameters
a=.5;   // std of scattering init.params from simulated ones
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=.1*ones(nc,nc);                     // pointer statistics
Est.Cp.th=fnorm(rand(nc,nc,'u')+.1);         // pointer parameter
w=fnorm(ones(1,nc));                         // weights

// SIMULATION ============================================================
for t=2:nd
  i=Sim.ct(t-1);    // last active component
  Sim.ct(t)=sum(rand(1,1,'u')>cumsum(Sim.Cp.th(i,:)))+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;                                           // stor
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); // likelihood
  end
  Lq=G-max(G);                // rough normlization
  q=exp(Lq);                  // exponent

  ww=(q*w)'.*Est.Cp.th;       // matrix weights
  W=ww/sum(ww);               // normalization    - f(c(t),c(t-1)|d(t))
  w=sum(W,1);                 // marginalization  - f(c(t)|d(t))
  wt(:,t)=w';                 // stor

  // Update of statistic
  Est.ka=Est.ka+w;                          // counter
  Est.Cp.V=Est.Cp.V+W;                      // ptr.stat. update
  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

    //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 pointer values
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')
title 'Pointer values and their predictions'

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 estimates '+string(i))
end
```