Bioberegninger - notat 4: Mer om sannsynlighetsmaksimering

8. mars 2004

1 Kort om Newton's metode i flere dimensjoner

Newton's metode kan generaliseres til å løse sett av n ligninger med n ukjente. Skal vi f.eks. finne sannsynlighetsmaksimeringsestimater i en modell med n ukjente parametere vil dette gi opphav til et slikt sett av ligninger når vi setter de deriverte med hensyn på de ulike parameterne lik 0.

La oss se på det tilfelle at vi har et sett av to ikke-lineære ligninger med to ukjente. Vi har altså et problem på formen

$$\begin{aligned} f(x,y) &= 0, \\ g(x,y) &= 0. \end{aligned} \tag{1}$$

og ønsker å finne en løsning $\mathbf{z} = (x, y)$ slik at (1) er tilfredsstilt.

Vi gjetter først på en løsning $\mathbf{z}_0 = (x_0, y_0)$. Ideen som ligger bak løsningsalgoritmen er den samme som når vi bruker Newton's metode for å løse én ligning med én ukjent. Vi lager oss tilnærminger til funksjonene f og g rundt punktet $\mathbf{z}_0 = (x_0, y_0)$ — lineariseringene

$$\begin{aligned} f(x,y) &\approx f(x_0,y_0) + \frac{\partial f}{\partial x}(x - x_0) + \frac{\partial f}{\partial y}(y - y_0) \\ g(x,y) &\approx g(x_0,y_0) + \frac{\partial g}{\partial x}(x - x_0) + \frac{\partial g}{\partial y}(y - y_0) \end{aligned} \tag{2}$$

De partiellderiverte beregnes i punktet $\mathbf{z}_0$. Vi ser altså på tangentplanene til funksjonene f og g i punktet $\mathbf{z}_0$.

I stedet for å forsøke å løse det virkelige ikke-lineære ligningssystemet er ideen så at vi lager vi oss en forbedret tilnærmet løsning ved å sette

tilnærmingene av funksjonene lik null. Vi får da det lineære ligningssystemet

$$\begin{aligned} f(x_0, y_0) + \frac{\partial f}{\partial x}(x - x_0) + \frac{\partial f}{\partial y}(y - y_0) = 0, \\ g(x_0, y_0) + \frac{\partial g}{\partial x}(x - x_0) + \frac{\partial g}{\partial y}(y - y_0) = 0, \end{aligned} \tag{3}$$

som på matriseform kan skrives som

$$\mathbf{F}(\mathbf{z}_0) + \mathbf{J}(\mathbf{z} - \mathbf{z}_0) = 0, \tag{4}$$

hvor

$$\mathbf{F}(\mathbf{z}) = \begin{bmatrix} f(x, y) \\ g(x, y) \end{bmatrix}, \tag{5}$$

altså en funksjon som har en vektor som funksjonsverdi, og hvor

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix}. \tag{6}$$

Vi lar løsningen på det lineariserte ligningssystemet (4) være neste tilnærmede løsning av (1) slik at vi generelt får iterasjonsformelen

$$\mathbf{z}_{n+1} = \mathbf{z}_n - \mathbf{J}^{-1}\mathbf{F}(\mathbf{z}_n). \tag{7}$$

2 Optimeringsalgoritmer i R

Generelt vil Newton's metode ikke alltid konvergere. I praksis er det også arbeidskrevende å programmere algoritmen dersom vi jobber med ligningssystem med mange ukjente, f.eks. i forbindelse med sannsynlighetsmaksimering for modeller med et stort antall ukjente parametere.

I praksis bruker vi derfor gjerne mer generelle algoritmer som er innebygd i R når vi skal finne sannsynlighetsmaksimeringsestimater eller når vi jobber med andre optimeringsproblemer. Slike optimeringsalgoritmer finnes i en rekke varianter tilgjengelig i R bl.a. gjennom funksjonen `optim`. Noen av disse metodene er også modifikasjoner av Newton's metode, f.eks. den såkalte kvasi-Newton metoden. Denne bygger på at de deriverte beregnes numerisk ved at funksjonen som skal maksimaliseres evalueres gjentatte ganger i området rundt $\mathbf{z}_n$ (se figur 1c). Generelt virker de ulike optimeringsalgoritmene ved at den funksjonen som skal minimaliseres (eller maksimaliseres) evalueres i et antall punkter i parameterrommet slik at algoritmen kan "danne seg et bilde" av funksjonens form og slik jobbe seg i retning av funksjonens minimum.

La oss anta at vi ønsker å finne funksjonen

$$h(x, y) = (x - 2.5)^2 + (y - 1.5)^2 \tag{8}$$

sitt minimum ved bruk av `optim` i R. Vi trenger først å programmere h som en funksjon `h` i R. For at `h` skal kunne virke sammen med `optim` må den ta i mot de av sine innargumenter som `optim` skal optimalisere med hensyn til i form av en vektor, i dette tilfelle av lengde to, hvor verdiene av x og y ligger i første og andre element av denne vektoren. Vi kan programmere `h` f.eks. på følgende måte:

```
h <- function(p) {
  x <- p[1]
  y <- p[2]
  return((x-2.5)^2+(y-1.5)^2)
}
```

På samme måte som når vi bruker Newton's metode trenger vi å oppgi startverdier for x og y når vi bruker `optim`. Vi oppgir disse verdiene i form av vektor med startverdier. Neste argument til `optim` er en referanse til funksjonsdefinisjonen av på den funksjonen vi ønsker å minimalisere, vi oppgir med andre ord bare navnet på den funksjonen vi ønsker å minimalisere (uten etterfølgende parenteser slik som ved funksjonskall). Det er altså selve funksjonsdefinisjonen som er argument.

```
> optim(c(1,1),h)
$par
[1] 2.499977 1.499952

$value
[1] 2.88779e-09

$counts
function gradient
      61       NA

$convergence
[1] 0

$message
NULL
```

Resultatet av kallet til `optim` er en liste av ulike komponenter. Komponenten `$par` er vektoren som inneholder verdiene av x og y i h sitt minimum, og `$value` inneholder funksjonsverdien i dette minimumet. Vi ser også at `optim` har gjort 61 kall til vår funksjon `h` (listekomponenten `$counts`).

Dersom `optim` ikke finner noe minimum vil komponenten `$convergence` få verdi 1 og ikke 0.

Merk at `optim` generelt vil søke etter minimum. Oppgir vi argumentet `control=list(fnscale=-1)` søker `optim` etter maksimum.

3 Sannsynlighetsmaksimering

Anta at vi har observert dataene $x_1, x_2, \ldots, x_n$ fra en modell med parametervektor $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$. Generelt vil modellen, dersom vi har kontinuerlig fordelte data, spesifisere simultantettheten til dataene, $f_{X_1,\ldots,X_n}(x_1, \ldots, x_n)$. Likelihoodfunksjonen vil generelt være definert som denne simultantettheten beregnet i punktet $(x_1, x_2, \ldots, x_n)$, og betraktet som en funksjon av de ukjente parameterne, altså

$$L(\theta) = f_{X_1,\ldots,X_n}(x_1, \ldots, x_n). \tag{9}$$

Dersom dataene er uavhengig fordelt får vi at

$$L(\theta) = \prod_{i=1}^{n} f_{X_i}(x_i). \tag{10}$$

Dersom alle X_i'ene også er identisk fordelt (dette vil ikke være tilfelle f.eks. i en regresjonsmodell) har vi at

$$L(\theta) = \prod_{i=1}^{n} f_X(x_i). \tag{11}$$

For diskrete fordelte data for vi tilsvarende uttrykk.

3.1 Programmering av likelihoodfunksjonen

Anta at $t_1, t_2, \ldots, t_n$ er uavhengige identisk Weibullfordelte data med tetthet

$$f_T(t) = \frac{a}{b}\left(\frac{t}{b}\right)^{a-1} \exp\left(-\left(\frac{t}{b}\right)^{a}\right). \tag{12}$$

Da kan vi skrive log-likelihoodfunksjonen på formen

$$\ln L(a, b) = \sum_{i=1}^{n} \ln f_T(t_i), \tag{13}$$

Sannsynlighetstettheten $f_T(t_i)$ og logaritmen til denne kan beregnes i R med funksjonen `dweibull` ved å bruke tilleggsargumentet `log=T`. Alle leddene i summen kan beregnes ved at `dweibull` virker elementvis på datavektoren. Når vi skal programmere likelihoodfunksjonen i R slik at denne kan virke sammen med `optim` må også de variabler som funksjonen skal maksimaliseres med hensyn på oppgis som argument i form av en vektor. Hele likelihoodfunksjonen kan dermed programmeres f.eks. slik:

```
lnL <- function(par,t) {
   a <- par[1]
   b <- par[2]
   return(-sum(dweibull(t,shape=a,scale=b,log=T)))
}
```

Minustegnet i siste linje gjør at vi slipper å oppgi tilleggsargumentet `control=list(fnscale=-1)` når vi skal beregne maksimum — `optim` vil i stedet finne minimum til minus log-likelihoodfunksjonen, altså maksimum til pluss log-likelihoodfunksjonen.

3.2 Sannsynlighetsmaksimering ved bruk av `optim`

Leser vi først inn levetidsdataene vi har brukt tidligere kan vi nå finne sannsynlighetsmaksimeringsestimatene til både a og b:

```
> t <- scan("/home/jarlet/undervisning/bioberegninger/spurv.dat")
> optim(c(1.3, 2), lnL, t=t)
$par
[1] 1.333679 1.787965

$value
[1] 1242.049

$counts
function gradient
      69       NA

$convergence
[1] 0

$message
NULL
```

Det er fornuftig å oppgi startverdier på parameterne i nærheten av det vi tror er sannsynlighetsmaksimeringsestimatene (f.eks. `c(1.3,2)` som her).

Vi ser at estimatet av formparameteren nå blir $\hat{a} = 1.33$ og at estimatet av skalaparameteren blir $\hat{b} = 1.78$ når begge estimeres fritt (til forskjell fra resultatene i øving 6 hvor vi antok at at $b = 2$).

Når vi gjør et kall til en funksjon (her `optim`) som i sin tur kaller en annen funksjon (her `lnL`) vil vi i mange tilfelle ha behov for å oppgi argumenter til den andre funksjonen (i dette tilfelle datavektoren `t`) i funksjonskallet til den første. I eksempelet over har vi oppgitt argumentet `t=t` i funksjonskallet til `optim`. Dette argumentet sendes videre til `lnL` gjennom en spesiell type mekanisme, et såkalt "..."-argument, når `optim` i sin tur kaller `lnL`. Vi skal se nærmere på hvordan denne mekanismen fungerer nedenfor.

Alternativt kunne vi latt `lnL` referere til dataene gjennom en global variabel `t` men dette er uhensiktsmessig i tilfeller hvor vi ønsker å beregne sannsynlighetsmaksimeringsestimater både for observerte og simulerte data.

3.3 Detaljer

La oss se litt mer på detaljene i hva som skjer ved framgangsmåten brukt over. La oss først lage et konturplot av log-likelihoodfunksjonen. Funksjonen `contour` virker på samme måte som funksjonen `persp` (øving 7) men lager i stedet for en tredimensjonal overflate ett enklere todimensjonalt kart med "høydekvoter" som representerer punkter i parameterrommet med samme log-likelihood. En generell funksjon som lager et slikt konturplot kan se slik ut:

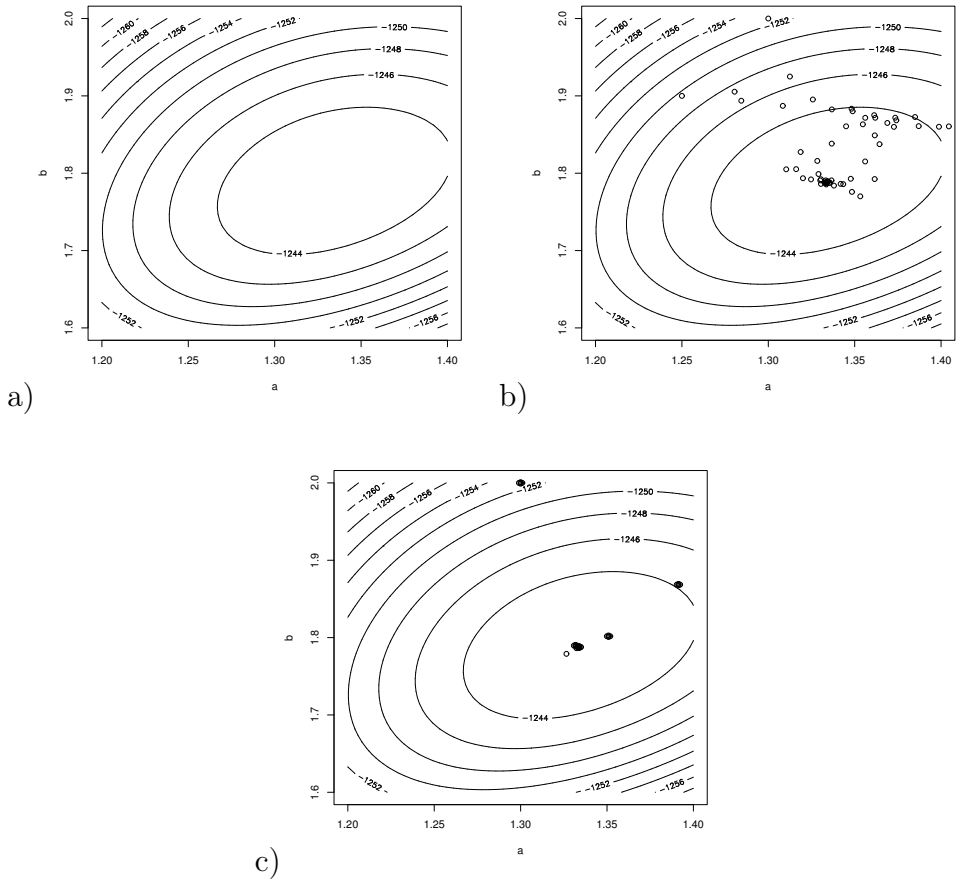
```
lnLplot <- function(lnL,min1,max1,min2,max2,xlab="",ylab="",gridsize=50,...){
   seq1 <- seq(min1,max1,length=gridsize)
   seq2 <- seq(min2,max2,length=gridsize)
   lnLmatrise <- matrix(NA, gridsize, gridsize)
   for (i in 1:gridsize) {
      for (j in 1:gridsize) {
         lnLmatrise[i,j] <- lnL(c(seq1[i],seq2[j]),...)
      }
   }
   contour(seq1,seq2,-lnLmatrise,xlab=xlab,ylab=ylab)
}
```

Her lar vi `lnLplot` ta i mot argumenter også gjennom det spesielle "..."-argumentet. Disse argumentene sender `lnLplot` så undret videre til funksjonen `lnL` inne i den nestede for-løkken.

Vi kan nå lage et konturplot over et passende område av parameterrommet rundt estimatene med kommandoen

```
> lnLplot(lnL,1.2, 1.4, 1.6, 2, t=t, xlab="a",ylab="b")
```

Se figur 1a.



Figur 1: Konturplot av log-likelihoodfunksjonen. I plot b) er punktene i parameterrommet hvor likelihoodfunksjonen blir evaluert av `optim` plottet (når optimaliseringen gjøres ved bruk av default metoden `method=''Nelder-Mead''`).

La oss nå se på hvordan `optim` leter seg fram gjennom parameterrommet. Dette kan lett illustreres ved at vi legger inn et par ekstra linjer i `lnL` som legger til punkter i konturplottet hver gang `lnL` kalles:

```
lnL <- function(par,t,vispunkter=F) {
```

```
   a <- par[1]
   b <- par[2]
   if (vispunkter) {
      points(a,b)  # Legg til et punkt med koordinater (a,b)
      Sys.sleep(1) # Vent ett sekund
   }
   return(-sum(dweibull(t,shape=a,scale=b,log=T)))
}
```

Beregner vi nå nye sannsynlighetsmaksimeringsestimater med `optim` ser vi hvordan algoritmen beveger seg gjennom parameterrommet:

```
> optim(c(1.3, 2), lnL, t=t, vispunkter=T)$par
[1] 1.333679 1.787965
```

Se figur 1b.

Samme plot med optimeringsalgoritmen kvasi-Newton (`method=''BFGS''`) er vist i figur 1c. I mange tilfeller vil den fungere bra og være noe raskere enn Nelder-Mead algoritmen som er default.

4 Flere eksempler

Anta at vi ønsker å studere hvor raskt et legemiddel nedbrytes etter at det er tatt av n pasienter. Vi antar at konsentrasjonen etter tid t er $(a-b)e^{-ct}+b$ slik at konsentrasjonen går mot b når t går mot uendelig og at den er a ved tid $t=0$. Anta at vi observerer konsentrasjoner X_i ved tidspunkt t_i for ulike pasienter $i = 1, 2, \ldots, n$. På grunn av måleusikkerhet antar vi videre at observasjonene X_i er normalfordelte med varians σ^2.

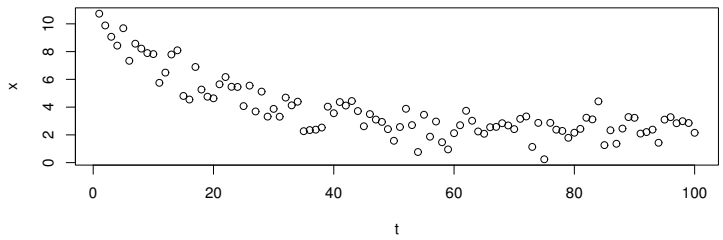
Oppsummerer vi dette har vi følgende statistiske modell:

$$X_i \sim N((a-b)e^{-ct_i}+b, \sigma^2). \tag{14}$$

Her er altså observasjonene uavhengige men ikke identisk fordelt fordi forventingsverdiene er forskjellige.

La oss først simulere ett datasett fra denne modellen. Anta at vi måler konsentrasjonen av legemiddelet i $n = 100$ forskjellige pasienter ved de valgte tidspunktene $1, 2, \ldots, 100$ timer. Anta også at $a = 10$, $b = 2$, $c = 0.05$, og $\sigma^2 = 1$. Et simulert datasett kan da lages på følgende måte:

```
> t <- 1:100
> x <- rnorm(n=100,mean=8*exp(-.1*t)+2,4)
> plot(t,x)
```



Figur 2: Simulerte data fra modell (14).

Plottet av dataene er vist i figur

Log-likelihoodfunksjonen for denne modellen kan programmeres på følgende måte:

```
lnL <- function(p,x,t) {
  a <- p[1]
  b <- p[2]
  c <- p[3]
  sigma2 <- p[4]
  return(-sum(dnorm(x,mean=(a-b)*exp(-c*t)+b,sd=sqrt(sigma2),log=T)))
}
```

Legg merke til at `dnorm` virker elementvis på vektorene `x` og `(a-b)*exp(-c*t)+b`, forventningene til observasjonene, slik at `dnorm` returnerer logaritmen til alle leddene i summen som inngår i log-likelihoodfunksjonen.

Vi kan nå finne sannsynlighetsmaksimeringsestimatene ved å bruke `optim`. Legg merke til at både `t` og `x` er argumenter som blir sendt videre til `lnL` via `optim`'s "..."-argument:

```
> optim(c(1,1,1,1),lnL,x=x,t=t)
$par
[1] 6.54997289 0.11863612 0.01616672 5.05129293

$value
[1] 191.0224

$counts
function gradient
     501       NA
```

```
$convergence
[1] 1

$message
NULL
```

Vi ser at vi får `$convergence` blir lik 1, som betyr at `optim` ikke fant fram til noe maksimum i løpet av 500 iterasjoner. Velger vi mer en mer fornuftig startverdi på parametervektoren får vi:

```
> optim(c(10,3,.1,1),lnL,x=x,t=t)
$par
[1] 10.50474828  2.14439504  0.05815284  0.94928235

$value
[1] 139.3044

$counts
function gradient
     239       NA

$convergence
[1] 0

$message
NULL
```

Nå ser vi at estimatene blir liggende i nærheten av de sanne verdiene samtidig som `$convergence` blir lik 0. Dette betyr ikke nødvendigvis at noe maksimum er funnet, bare at den siste relative endringen i $\ln L$ er mindre enn `reltol` som har defaultverdi i størrelsesorden 10^{-8}. Se hjelpesiden til `optim`. Dette kan inntreffe også om `optim` havner i ett området i parameterrommet hvor $\ln L$ "flater" ut. I slike tilfeller er det lurt å kontrollere at man har funnet et virkelig maksimum ved å kalle `optim` med forskjellige startverdier av parametervektoren.