Notat 3 - ST1301

1. februar 2005

1 Simulering fra modell

Når vi skal analysere et gitt konkret innsamlet datasett vil vi gjøre dette med utgangspunkt i en statistisk modell. Vi kan si at en slik statistisk modell representerer hva vi på forhånd tror om den prosessen som har generert dataene vi studerer. Modellen består av et sett antakelser vi velger å tro på som spesifiserer hvordan dataene er fordelt gitt verdien av visse ukjente størrelser som inngår i modellformuleringen — modellens parametere. Vi ønsker å trekke slutninger om disse parameterne.

I mange tilfeller kan vi finne egenskaper ved interessante størrelser slik som estimatorer, testobservatorer og andre funksjoner av dataene analytisk. Dersom vi for eksempel antar at vi har uavhengig identisk normalfordelte data, vil størrelsen $(\bar{X} - \mu)/(S/\sqrt{n})$ være t-fordelt. I en lineær regresjonsmodell antas det at responsvariabelen Y_i for den i'te observasjonen normalfordelt med forventning $a + bx_i$ og varians σ^2. I denne modellen har estimatorene for de ulike parameterne kjente fordelinger. I mange tilfeller kan vi finne interessante egenskaper som forventningsverdi og varians til eksempelvis ulike estimatorer analytisk.

Avhengig av hvilke antakelser vi finner det rimelig å bygge inn i en modell vil det imidlertid ikke alltid være slik at vi kan finne egenskaper til interessante størrelser analytisk. Det er i slike tilfeller at stokastisk simulering vil være et nyttig hjelpemiddel.

1.1 Generell algoritme

Anta at vi har en modell som spesifiserer fordelingen til et sett observasjoner $X_1, X_2, \ldots, X_n$. I mange tilfeller vil vi være interessert i å se på egenskapene til en eller annen funksjon av observasjonene, la oss si,

$$W = W(X_1, X_2, \ldots, X_n). \tag{1}$$

Hvis vi for eksempel jobber med data fra exponentiell fordeling vil vi kunne være interessert i å studere egenskapene til estimatoren $\hat{\lambda} = n/\sum X_i$. Denne er en slik funksjon av dataene.

Generelt vil funksjonen W kunne være en estimator, en testobservator, eller en annen stokastisk variabel som vi ønsker å finne egenskapene til.

- I estimeringssammenheng trenger vi å bestemme standardfeil og eventuell forventningsfeil til estimatoren gitt ved $E(W)$ og $\mathrm{Var}(W)$.
- I forbindelse med hypotese testing vil vi kunne være interessert i sannsynligheter for eksempel av typen $P(W \geq a)$.
- Når vi skal undersøke reellt konfidensnivå til konfidensintervaller trenger vi å se på to funksjoner av dataene, nedre og øvre intervallgrense, gitt ved funksjonene $\underline{\theta}(X_1, X_2, \ldots, X_n)$ og $\bar{\theta}(X_1, X_2, \ldots, X_n)$. Vi er så interessert i sannsynligheten for at intervallgrensene ligger rundt den ukjente parameteren θ, altså $P(\underline{\theta} \leq \theta \leq \bar{\theta})$.
- Også utenfor området statistisk inferens vil stokastisk simulering være et nyttig verktøy. Senere i kurset og i videregående kurs vil vi lage stokastiske modeller som spesifiserer fordelingen til frekvensen av for eksempel en ny mutasjon i fremtidige generasjoner, la oss si, $p_1, p_2, \ldots, p_n$, gitt frekvensen i generasjon $t = 0$. Spørsmål som vil kunne være av interesse er sannsynligheten for at en mutasjonen er tilstede i generasjon n, $P(p_n > 0)$.

Alt dette er tallstørrelser som vi finne med den nøyaktighet vi måtte ønske ved hjelp av simuleringer. En generell algoritme for å gjennomføre dette er som følger:

1. Gjenta følgende for $i = 1, 2, \ldots, m$. Antall simuleringer velges gjerne lik $m = 1000$.
   (a) Simuler et utvalg $X_1^*, X_2^*, \ldots, X_n^*$ fra modellen. Utvalget kalles gjerne et *bootstrap-sample*.
   (b) Beregn $W_i^* = W(X_1^*, X_2^*, \ldots, X_n^*)$ og eventuelt andre variable av som er av interesse. Dette vil utgjøre i'te *bootstrap-replikat* av variabelen W.
2. Nå kan ulike størrelser estimeres fra bootstrap- replikatene på følgende måte:

(a) $E(W)$ kan estimeres ved hjelp av estimatoren

$$\bar{W}^* = \frac{1}{n}\sum_{i=1}^{m} W_i^*. \tag{2}$$

(b) $\mathrm{Var}(W)$ kan estimeres ved vanlig estimator for varians,

$$S^2_{W^*} = \frac{1}{m-1}\sum_{i=1}^{m}(W_i^* - \bar{W}^*)^2. \tag{3}$$

(c) Sannsynligheter av typen $P(W \geq a)$ kan estimeres ved m_A/m hvor m_A er antall bootstrap-replikater $W_i^* \geq a$.

I praksis utføres en slik algoritme lettest ved hjelp av datamaskin ved å lage et program i et passende programmeringsspråk. [1]

1.2 Eksempel - forventningsverdi til estimator

Det kan vises at

$$\hat{\lambda} = \frac{n}{\sum X_i} \tag{4}$$

er sannsynslighetsmaksimeringsestimatoren (SME) til parameteren λ i ekponentiell modell. La oss anta at vi ønsker å undersøke om denne er forventnignsrett, altså at $E(\hat{\lambda}) = \lambda$, ved hjelp av simuleringer.

Det er alltid hensiktsmessig å programmere den funksjonen vi vil se på som en egen funksjon i R. Estimatoren gitt (4) kan programmeres i R på følgende måte:

```
# Kontrakt: lambdahat: vektor -> flyttall
#
# Hensikt: Beregne SME for lambda i eksponentiell
#   modell fra et observert tilfeldig utvalg
#   x_1, x_2, ... , x_n
#
# Eksempel:
#   lambdahat(c(1,2,1,2)) skal returner 2/3=0.67
#
```

[1] Algoritmen over kalles gjerne parametrisk bootstrapping. I kursene anvendt statistikk og moderne statistiske metoder vil såkalt ikke-parametrisk bootstrapping bli behandlet. Dette er metoder hvor vi simulerer bootstrap-sample (trinn 1a i algoritmen) på en annen måte uten å gjøre antakelser om hvilken fordeling dataene har.

```
# Definisjon:
lambdahat <- function(x) {
   n <- length(x)
   n/sum(x)
}
# eventuellt bare:
# lambdahat <- function(x) 1/mean(x)
#
# Tester:
lambdahat(c(1,2,1,2))
```

Merk kallet til `length` som gjør at funksjonen `lambdahat` håndterer utvalg av vilkårlig størrelse (representert av vektoren `x` som er innargument.) Funksjonen svarer til funksjonen W gitt ved (1) i det generelle oppsettet.

Vi programmerer så en funksjon som utfører algoritmen over. Det er hensiktsmessig å la antall simuleringer m være et argument med defaultverdi 1000 slik at vi eventuelt kan utføre et større antall simuleringer senere om vi måtte ønske dette uten å endre funksjonsdefinisjonen. Når vi skal simulere fra modellen vår må vi også anta en eller annen verdi for parameteren λ. Denne (samt utvalgsstørrelsen n) bør derfor også være argument i funksjonen som skal utføre simuleringsalgoritmen:

```
# Hensikt: simulere 1000 realisasjoner av SME for lambda
#   i eksponentiell modell for utvalgsstørrelser på 10
#
lambdasim <- function(lambda,n=10,m=1000) {
   lambdaboot <- rep(NA,m)
   for (i in 1:m) {
      x <- rexp(n=n,rate=lambda)
      lambdaboot[i] <- lambdahat(x)
   }
   lambdaboot
}
```

La oss gå gjennom de ulike delene av funksjonsdefinisjonen over. Hoveddelen av funksjonen består av en for-løkke hvor løkkevariabelen `i` tar verdiene $1, 2, \ldots, 1000$ ved de gjentatte utførelsene av løkke-kroppen (uttrykkene mellom krøllparantesene).

I første linje av løkke-kroppen simuleres et tilfeldig utvalg (et bootstrap-sample) fra modellen ved hjelp av et kall til funksjonen `rexp`. Resultatet

lagres i den lokale variabelen `X`. Dette svarer til trinn 1a) i den generelle algoritmen i avsnitt 1.1.

I andre linje av løkke-kroppen beregnes det i'te bootstrapreplikatet av variabelen vi ser på ved hjelp av et kall til vår egen funksjon `lambdahat`. Dette svarer til trinn 1b i algoritmen. Merk at det er det simulerte bootstrapsamplet som går inn som argument i kallet til `lambdahat`. Vi tar vare på resultatet i det i'te elementet av den lokale variabelen (vektoren) `lambdaboot`.

Det siste funksjonen gjør etter at for-løkken er gjennomløpt er at hele vektoren `lambdaboot` returneres som funksjonsverdi.

Merk også at hele vektoren `lambdaboot` opprettes som en tom vektor av lengde m i første linje ved at verdien av uttrykket `rep(NA,m)` tilordnes til `lambdaboot`. Dette er hensiktsmessig å gjøre dersom vi kjenner lengden til `lambdaboot` på forhånd slik som her — vi unngår at R må allokere minneplass til stadige utvidelser av vektoren `lambdaboot` ved gjennomkjøring av løkken slik tilfelle ville vært om vi hadde initiert `lambdaboot` til å ha bare ett element i linje 1. Resultatet er at kall til `lambdasim` utføres på langt kortere tid.[2]

Går vi tilbake til vårt opprinnelige problem kan vi nå finne forventningsverdien til estimatoren $\hat{\lambda}'$ ved å gjøre ett kall til `lambdasim` og så beregne gjennomsnittsverdien til de returnerte bootstrapreplikatene:

```
> mean(lambdasim(lambda=1))
[1] 1.120907
> mean(lambdasim(lambda=2))
[1] 2.218094
> mean(lambdasim(lambda=3))
[1] 3.257609
> mean(lambdasim(lambda=1,n=5))
[1] 1.221541
> mean(lambdasim(lambda=2,n=5))
[1] 2.501225
> mean(lambdasim(lambda=3,n=5))
[1] 3.828398
```

Vi ser at forventningsverdien estimert ved (2) nå blir betydelig større enn de sanne parameterverdiene vi har brukt når vi har simulert. For $n = 10$ (default verdi) ser vi at $\hat{\lambda}$ overestimeres med omlag 10%.

I tillegg til å se på forventingsretthet (ved å bruke `mean`) kan det og være interessant å se på standardavviket til estimatoren samt og lage histogram over hele fordelingen til estimatoren, gitt ulike verdier av den ukjente parameteren λ. Dette kan vi gjøre ved å skrive

[2]Ta eventuelt tiden ved å skrive f.eks. `system.time(mean(lambdasim(lambda=1)))`. Endre så første linje til `lambdaboot <- NA` og ta tiden på nytt.

```
> sd(lambdasim(1))
[1] 0.3940102
> hist(lambdasim(1))
>
```

1.3 Eksempel - beregning av dekningsgrad til konfidensintervall

Dersom $X_1, X_2, \ldots, X_n$ er uavhengige normalfordelte data med ukjent forventning og varians μ og σ^2 er

$$(\bar{X} - t_{n-1,\alpha/2}S/\sqrt{n}, \bar{X} + t_{n-1,\alpha/2}S/\sqrt{n}) \tag{5}$$

et $(1-\alpha)$ konfidensintervall for parameteren μ. Dette betyr at endepunktene i intervallet (som er funksjoner av dataene og dermed stokastiske variable) skal ligger rundt μ med sannsynlighet $(1-\alpha)$ (konfidensnivået). At dette faktiske er tilfelle kan kontrolleres ved hjelp av simuleringer.

Vi følger den samme generelle oppskrift som i avsnitt 1.1. Først programmer vi en funksjon som beregner endepunktene i intervallet som funksjon av dataene. Dette svarer til to funksjoner på samme form som (1). Kvantilen i t-fordelingen, $t_{n-1,\alpha/2}$, finner vi ved et kall til `qt`. Merk at kvantiler i R alltid er definert på grunnlag av nedre hale i fordelingene i motsetning til i mange lærebøker hvor øvre hale ofte brukes som utgangspunkt.

```
# Kontrakt: konfintmu: vektor, flyttall -> vektor
#
# Hensikt: Beregne konfidensintervall for parameter mu fra et
#   tilfeldig utvalg fra normalfordelingen
#
konfintmu <- function(x,alpha=.05) {
  n <- length(x)
  t <- qt(df=n-1,p=alpha/2,lower.tail=FALSE)
  xbar <- mean(x)
  s <- sd(x)
  c(xbar-t*s/sqrt(n),xbar+t*s/sqrt(n))
}
```

Uttrykket i siste linje (funksjonsverdien) angir en vektor av lengde 2 som inneholder endepunktene i konfindensintervallet som første og andre element.

Tester vi intervallet på ett simulert datasett får vi:

```
> x <- rnorm(n=30,mean=10,sd=2)
> konfintmu(x)
[1]  9.753085 11.405471
```

For det ene simulerte utvalget ser vi at konfindensintervallet blir liggende rundt den sanne parameterverdien ($\mu = 10$). Ut i fra måten intervallet er konstruert på vet vi at denne hendelsen skal inntreffe med sannsynlighet lik 0.95. La oss undersøke om dette virkelig er tilfelle.

En funksjon som beregner dekningsgraden kan se slik ut:

```
# Hensikt: Beregne dekningsgrad til konfidensinterval.
#
# Eksempel:
#    dekningsgrad(10,2,5) bør gi 0.95 som svar
#
dekningsgrad <- function(mu,sigma2,n,alpha=.05,nsim=10000) {
  ntreff <- 0
  for (i in 1:nsim) {
     X <- rnorm(n=n,mean=mu,sd=sqrt(sigma2))
     ki <- konfintmu(X,alpha)
     if (ki[1]<=mu & mu<=ki[2]) {
        ntreff <- ntreff + 1
     }
  }
  ntreff/nsim
}
```

Denne funksjonen følger essensielt samme oppsett som den generelle algoritmen i avsnitt 1.1. I første linje i løkke-kroppen simuleres et tilfeldig utvalg (et bootstrap-sample) fra den antatte modellen, og i neste linje beregnes konfidensintervallet og vi tar vare på dette i listen `ki`. Vi søker sannsynligheten for at intervallgrensene ligger rundt μ. Derfor tester vi dette med en if-setning; hvis det logiske uttrykket er oppfylt økes tellevariabelen `ntreff` med 1 i neste linje. Når for-løkken er gjennomløpt vil uttrykket `ntreff/nsim` gi oss et estimat av sannsynligheten vi er ute etter — intervallets reelle dekningsgrad. Vi må huske å initiere tellevariabelen i første linje — før vi har begynt å telle skal denne ha verdi 0.

Vi ser at dekningsgraden i dette tilfelle blir lik det nominelle nivået uansett hvilke verdier modellparameterne har. Dette er forventet i og med at intervallet bygger ikke på noen tilnærminger i motsetning til intervallet for p i øving 4 og 5:

```
> dekningsgrad(mu=10,sigma2=2^2,n=10,alpha=.05)
[1] 0.9495
> dekningsgrad(mu=10,sigma2=2^2,n=10,alpha=.1)
[1] 0.8979
> dekningsgrad(mu=10,sigma2=2^2,n=2,alpha=.05)
[1] 0.9506
> dekningsgrad(mu=10,sigma2=2^2,n=100,alpha=.05)
[1] 0.9496
```