ST1301 Bioberegninger - Introduksjon

Jarle Tufto

11. januar 2005

1 Om kurset

Innenfor en del retninger av statistikk og biologifaget er behovet for programmeringsferdigheter relativt store. I forbindelse med analyse av data, vil man særlig ha behov for å utføre ulike typer numeriske beregninger og å implementere ulike typer algoritmer for å simulere fra til dels kompliserte sannsynlighetsfordelinger.

Da bachelorprogrammet i biomatematikk ble satt sammen sto valget mellom å legge inn et av Institutt for informatikks standard grunnkurs i programmering i første studieår eller å lage vårt eget kurs. Vi har valgt det siste fordi dette i langt større grad gir oss muligheten til å fokusere på programmeringsteknikker som vi som statiskere rent faktisk vil ha behov for. Det gir oss også muligheten til å velge et høynivå programmeringsspråk som er mer tilpasset den type beregninger vi som statiskere møter. Samtidig er tanken at vi samtidig vil kunne styrke forståelsen av viktige deler av brukerkursene i sannsynlighetsregning, statistiske metoder og matematikk ved å gjøre deler av stoffet i disse kursenen mer konkret og håndgripelig.

Et grunnkurs i programmering vil være nyttig særlig i et senere masterstudium når en skal jobbe med mer avanserte statistiske teknikker, men slik vi ser det kan et slikt kurs tas først når behovet melder seg i forbindelse med mastergrad.

Dette kurset vil også kunne være interessant for masterstudenter i biologi og andre interesserte som ønsker å lære seg programmeringsferdigheter tilpasset den type problem en som masterstudent vil kunne møte i forbindelse med analyse av innsamlede data.

Øvingene vil i praksis være den viktigste delen av kurset. Det beste er å jobbe med disse i grupper på en eller to personer.

2 Hvorfor R?

Når vi som statistikere skal trekke slutninger fra data, f.eks. fra en klinisk undersøkelse eller tidseriedata fra en populasjon ute i naturen, bygger vi slutningene på en statistisk modell. En slik modell representerer generellt et sett antakelser om de prosessene som har generert dataene. Hvilke antakelser vi finner rimelige varierer selvsagt fra situasjon til situasjon.

I mange tilfeller vi modellantakelsene svare til standard statistiske modeller som variansanalyse, lineær regresjon, logistisk regresjon eller levetidsanalyse. I slike tilfeller kan vi benytte standard metoder for estimering og hypotesetesting under slike modeller. Slike standardmetoder finnes innebyggd i en rekke forskjellige statistiske programpakker som SPSS, SAS, Minitab, S-plus og i R som vi skal benytte i dette kurset.

I andre tilfeller vil ikke de antakelsene vi finner rimelige svare til noen standard statistisk modell og metode. I slike tilfeller vil vi måtte analysere dataene ved hjelp av egenutviklede statistiske modeller som vi forsøker å skreddersy for den aktuelle situasjonen vi studerer. Programpakken R er velegnet til å utføre den type beregninger vi trenger å utføre i slike situasjoner.

I biologi og medisin blir mye av teorien stadig mer matematisert og stokastiske, ikke-deterministiske modeller spiller en stadig viktigere rolle. For å oppnå innsikt i de dynamiske egenskapene til en modell vil vi ofte trenge å undersøke modellen ved hjelp av stokastisk simulering og numeriske beregninger på datamaskin fordi matematisk analyse av alle sider ved modellen blir for vanskelig. Også på dette området er R relativt egnet og vi har valgt å bruke R for begrense antall programpakker som studentene vil måtte sette seg inn i.

R er også en fritt tilgjengelig programpakke med fritt modifiserbar kildekode som kan kjøres under alle vanlige operativsystemer som Unix, Linux, og ulike versjon av Microsoft Windows. Dette har gjort R til et attraktivt alternativ til tilsvarende kommersielle programpakker. Mye av nyere avanserte statistiske metoder utvikles og implementeres av ledende statistikere som tilleggspakker ("packages") som kan kjøres under R. I bruk er R svært likt den kommersielle programpakken S-plus. R, ulike tilleggspakker, og mer informasjon om systemet er tilgjengelig fra `http://www.r-project.org`. Som student må du gjerne installere R på din hjemmemaskin.

3 Om datamaskiner og programmering

Noe av framstillingen her og i neste kapittel bygger bygger på Felleisen et al. (2001).

Vi lærer å utføre ulike beregninger i ung alder. Noe av det første vi lærer er addisjon og subtraksjon.

> En pluss en er lik to. Seks minus fire er lik to.

Etterhvert som vi blir eldre lærer vi mer kompliserte matematiske operasjoner som eksponering og sinus, og vi lærer også å beskrive regler for ulike typer beregninger.

> Gitt en sirkel med radius r er sirkelens omkrets r gange to gange π. Sirkelens areal er gitt ved kvadratet av r ganget med med π. Tjener en person 100 kr per time og personen jobber i x antall timer vil personen totalt tjene 100 kr ganget med x.

Sannheten er at vi gjennom 12 års skolegang er blitt gjort til datamaskiner programmert av vår lærere til å utføre enkle programmer.

Datamaskiner er ikke annet en svært raske studenter som kan utføre millioner av f.eks. addisjonsoperasjoner i løpet av noen hundredels sekund. Men datamaskiner kan også gjøre mye annet som å beregne framtidig vær, gjennkjenne ansikter, eller styre et dataspill. Kort sagt kan en si at datamaskiner kan prosessere alle typer informasjon.

Vanligvis uttrykker vi informasjon og instruksjoner på vanlig norsk.

> Temperaturen er 35°C. Konverter denne temperaturen til grader Fahrenheit. En bil akselererer fra 0 til 100 km/t i løpet av 35 sekunder. Beregn hvor langt bilen kommer i løpet av 20 sekunder.

Datamaskiner kan imidlertid ikke forstå komplekse instruksjoner uttrykkt på norsk. I stedet må vi lære et *programmeringsspråk* for å kommunisere informasjon og instruksjoner til en datamaskin.

Informasjon uttrykt i et programmeringsspråk kalles *data*. Data kan være av mange typer, såkalte *datatyper*. En type data er tall. *Sammensatte data* er satt sammen av enklere datatyper. En sekvens bestemte tall vil være et eksempel på sammensatt data. Bokstaver vil er en annen datatype, et navn er en type sammensatte data. Et familietre vil være en annen sammensatt type data.

Data representerer informasjon men det konkrete tolkningen av denne informasjonen er opp til oss. Et tall som 37.51 kan f.eks. representere en temperatur, en tid eller en distanse. Boktaven B kan representere en karakter eller en del av en adresse. Merk at begrepet data slik det er brukt her er det samme som men også omfatter mye mer en begrepet data slik det brukes innenfor statistisk inferens.

Instruksjoner, også kalt funksjoner (eller operasjoner; det er egentlig ingens vesens forskjell på dette) er, på samme måte som data, av ulike typer. Noen operasjoner som "+", "-", "*", og "/" er mer primitive enn andre operasjoner og funksjoner. En programmerer setter sammen primitive operasjoner eller funksjoner sammen til et *program*. Vi kan derfor tenke på de primitive operasjoner og funksjoner som ordene i et fremmed språk og det å programmere som det å forme setninger i dette fremmede språket.

Program varierer i størrelse. Det å skrive et program krever nøye planlegging på samme måte som det å skrive en bok eller et essay gjør det. Programmet må *designes*. Hver enkelt detalj må vies oppmerksomhet og skriving av ett større program bør følge en planlagt strategi.

Å skrive et program er en kreativ aktivitet som belønnes med en god følelse når man har laget et velfungerende program som gjør det det er ment å gjøre. For å nå dette målet må vi tilegne oss endel ferdigheter. Progammeringsspråk er primitive og deres grammatikk er streng. Dessuten er ikke datamaskiner intelligente. En liten grammatisk feil kan derfor gjøre at et program ikke vil fungere. Og selv uten grammatiske feil kan vi ha gjort logiske feil som gjør at et program ikke gjør det vi ønsker.

4 Grunnleggende bruk av R

Se Dalgaard (2002), kap 1.1-1.3.

5 Programmering

I mange tilfeller vil vi jobbe med R interaktivt i R-skallet. Dersom vi trenger å utføre en større mengde operasjoner i en bestemt rekkefølge er det praktisk å skrive disse inn i som et *skript* i en egen editor, f.eks. winedt og så kopiere inneholdet inn i R (Alt-P eller Ctr-Alt-P) for å utføre operasjonene i R.

For å løse mer komplekse oppgaver er det ofte hensiktsmessig å skrive en eller flere egendefinerte funksjoner. Et slikt sett funksjoner sammen med et eventuellt skript vil utgjøre et program.

I matematikk skriver vi ofte matematiske uttrykk hvor matematiske variabler inngår. Arealet av en sirkel er f.eks. gitt ved det matematiske uttrykket

$$3.14 \cdot r^2. \tag{1}$$

I dette uttrykket står r for en hvilken som helst positivt reellt tall. Hvis vi kommer over en bestemt sirkel med radius lik 4 kan vi beregne denne sirkelens

areal ved å sette inn tallet 4 for r i (1) og så redusere uttrykket til et enkelt tall:

$$3.14 \cdot 4^2 = 3.14 \cdot 16 = 50.24. \tag{2}$$

Vi har sett hvordan referanser til variabler også kan inngå i uttrykk i R. Skriver vi følgende uttrykk i R får vi

```
> 3.14*r^2
[1] 28.26
```

dersom vi på forhånd har gjort tilordningen

```
> r <- 3
```

Generellt er slike uttrykk hvor variabler inngår regler som beskriver hvordan et visst resultat skal beregnes *når* verdien av en variablene er gitt.

Et funksjon i R er i sin enkleste form ikke annet en en slik regel. Det er en regel som forteller oss og datamaskinen hvordan visse type data skal produseres fra visse andre data. Store program består av mange slike regler (funskjoner). Derfor er det viktig å gi hver regel meningsbærende navn etter hvert som vi skriver dem ned. Skriver vi

```
areal.av.sirkel <- function(r)
  3.14*r^2
```

inn i winedt og kopierer dette inn i R har vi definert vår egen funksjon `areal.av.sirkel`. Første linje angir navnet på funksjonen (`areal.av.sirkel`), funskjonen skal ta et enkelt innargument (`r`), beregne verdien av uttrykket `3.14*r^2` og så returnere dette som funksjonsverdi. Vi kan merke oss at vi gjør en tilordning når vi definerer en funksjon; vi tilordner en type objekt (en funksjonsdefinisjon) til en variabel med navn `areal.av.sirkel`.

Nå kan vi anvende funksjonen på samme måte som andre funksjoner ved å skrive

```
> areal.av.sirkel(5)
[1] 78.53982
```

Programmer består av mer primitive operasjoner, i dette tilfelle operasjonene multiplikasjon og eksponentiering. Så snart vi har definert en funksjon kan vi bruke den gjenntatte ganger og som en den var en primitiv operasjon.

```
> areal.av.sirkel(3)
[1] 28.27433
> areal.av.sirkel(4)
[1] 50.26548
```

En funksjon kan ta flere innargument. Anta at vi ønsker å definere et funksjon som beregner arealet av en ring, d.v.s. en sirkulær skive med et hull i midten. Areal av skiven er gitt ved gitt ved arealet av den ytre sirkelen minus areal av den indre sirkelen. La oss kalle disse variablene (radius til ytre og indre sirkel) `ytre` og `indre`. Da vil en funksjon som beregner arealet av ringen være

```
areal.av.ring <- function(ytre,indre)
  areal.av.sirkel(ytre)-areal.av.sirkel(indre)
```

Når vi bruker funksjonen må begge innargumenter oppgis.

```
> areal.av.ring(4,2)
[1] 37.69911
```

6 Tekstoppgaver og designplan

Utgangspunktet når vi skal løse en programmeringsoppgave vil være en oppgavetekst. Fra denne teksten trenger vi å trekke ut relevant informasjon om hvilke inndata funksjonen skal ta som argument og hvilke data den skal produsere. Så må vi finne ut hvordan utdata er bestemt av inndataene og utvikle uttrykk som beregner utdataene basert på mer primitive operasjoner i R. Så må vi undersøke om programmet gjør det vi ønsker; dette kan føre til at vi finner syntaxfeil eller logiske feil.

For å løse de fleste oppgaver vil det være hensiktsmessig å følge en bestemt plan inndelt i bestemte faser hvor visse aktiviteter utføres. Resultatet av disse aktivitetene skrives inn som kommentarer rundt selve programkoden. For eksempelet over, beregning av arealet av en ring, vil sluttresultatet se slik ut

```
# Kontrakt: areal.av.ring : tall tall -> tall
#
# Hensikt: Å beregne arealet av en ring med radius lik ytre
# og med et hull i midten med radius lik indre.
#
# Eksempel: areal.av.ring(2,1) skal gi som svar 9.42
#
# Definisjon:
areal.av.ring <- function(ytre,indre)
  areal.av.sirkel(ytre)-areal.av.sirkel(indre)
#
# Tester:
```

```
areal.av.ring(2,1)
# skal gi forventet verdi 9.42
```

Designplanen består av følgende fire aktiviteter:

Å forstå funksjonens hensikt: Vi begynner med å gi funksjonen et meningsfullt navn og å ved å skrive ned hvilke datatyper funksjonen tar som innargument og hvilken datatype den produserer i form av en kommentar over programkoden (alle linjer som begynner med # er kommentarer som ignoreres av R)

```
# Kontrakt: areal.av.ring : flyttall flyttall -> flyttall
```

Når vi har skrevet kontrakten kan vi skrive første linje i selve programkoden for funksjonen; denne kalles funksjonens header:

```
areal.av.ring <- function(ytre,indre)
```

Her angis funksjonens navn igjen og vi velger navn på variablene som inneholder funksjonens inndata. Disse kalles gjerne funksjonens argumenter.

Ut i fra kontrakten og headeren skriver vi så ned funksjonens hensikt, d.v.s., en kort kommentar som beskriver *hva* funksjonen skal beregne uttrykkt ved de variabelnavn vi har valgt i headeren:

```
# Hensikt: Å beregne arealet av en ring med radius lik ytre
# og med et hull i midten med radius lik indre.
```

For tekstoppgaver må vi søke gjennom oppgaveteksten når vi skal bestemme hva som skal beregnes, hva som vi skal regne som gitt (dette vil dukke opp som en konstant i programkoden), og hva som er ukjente størrelser som først vil bli kjent senere (funksjonsargumenter).

Eksempler: For bedre å forstå hva programmet skal beregne og ikke minst hvordan er det svært hensiktsmessig å finne på eksempler på inndata og tilhørende utdata. For eksempel skal vår funksjonen `areal.av.ring` produsere 50.24 når inndataene er 2 og 1 fordi dette er forskjellen i areal mellom den ytre og indre sirkelen. Vi skriver inn slike eksempler som kommentarer over programkoden:

```
# Eksempel: areal.av.ring(2,1) skal gi som svar 9.42
```

Det å finne på eksempler, *før* vi skriver ned selve programkoden i funksjonskroppen, er til stor hjelp. Dersom vi gjør dette først når vi har en ferdig

funksjon kan det være lett å stole på at funksjonen produserer de utdata den skal og slik ikke oppdage eventeulle logiske feil.

Enda viktigere er det at eksempler hjelper oss også til å tenke igjennom de operasjoner som må utføres for å produsere de ønskede utdata fra inndataene.

Eksempler hjelper også med å illustrere hva som menes i beskrivelsen av funksjonens hensikt.

Skriving av funksjonskroppen: Til slutt må vi formulere selve programkoden, d.v.s. utvilkle de nødvendige operasjoner og uttrykk som beregner det vi søker etter headeren ved å bruke mer primitive operasjoner og funksjoner som allerede finnes i R eller som vi planlegger å definere senere. I eksempelet vi har fulgt gjorde vi bruk av den mer primitive funksjonen `areal.av.sirkel` vi allerede hadde definert når vi utviklet uttrykket for å beregne areal av ringen.

```
areal.av.ring <- function(ytre,indre)
  areal.av.sirkel(ytre)-areal.av.sirkel(indre)
```

Hvis relasjonen mellom inndata og utdata er gitt ved et matematisk uttrykk trenger vi bare å oversette dette til en syntax som R forstår. For mer komplekse problemer er det ofte nyttig å gå tilbake til eksemplene vi laget for å undersøke nøyaktig *hvordan* vi kom fram til resultatet i hvert eksempel og så forsøke å uttrykke de beregningene vi foretok ved hjelp av uttrykk i R.

Testing: Når vi har skrevet programkoden bør vi som et minimum kontrollere at funksjonen produserer det den skal for de inndata vi brukte i eksemplene. Testene kan godt skrives slik at de kan kopieres rett i R-skallet ved hjelp av Alt-P.

Testing kan lede til at vi oppdager syntax feil eller logiske feil. I begge tilfelle kan det være nødvendig å gå helt tilbake til første fase i designplanen. Tabell 1 oppsummerer designplanen.

Referanser

Dalgaard, P., 2002. Introductory Statistics with R. Springer Verlag, New York.

Felleisen, M., R. B. Findler, M. Flatt, and S. Krisnamuurthi, 2001. How to Design Programs. An Introduction to Computing and Programming. MIT Press, Cambridge, Massachusetts.

Fase	Mål	Aktivitet
Kontrakt, hensikt, og header	Å navngi funksjonen, spesifisere datatype/klasse til inn- og utdata, beskrive hensikten; formulerere en header	Velge et navn som passer til problemet. Studere problemet og lete etter ledetråder for hvor mange ukjente gitte variable funksjonen konsumerer. Velge variabelnavn for hvert inndata; hvis mulig velge navn som er nevnt i oppgaveteksten. Beskrive hva funksjonene skal produsere uttrykt ved de valgt variabel navnene. Formulere kontrakt og header.
Eksempler	Å karakterisere sammenhengen mellom inndata og utdata gjennom eksempler	Søke etter eksempler i oppgaveteksten. Jobbe seg gjennom eksemplene (beregne utdata for hånd gitt funksjonens kontrakt og inndata). Validere resultatene hvis mulig. Konstruere eksempler på inndata og tilhørende utdata.
Kropp	Å definere funksjonen	Formulere hvordan funksjonen beregner sine utdata. Utvikle uttrykk bestående av mer primitive operasjoner, kall til andre funksjoner, og variabelreferanser. Oversette matematiske uttrykk i oppgaveteksten når disse er gitt.
Testing	Å oppdage feil (skrivefeil og logiske feil)	Anvende funksjonen på inndataene gitt i eksemplene. Undersøke om utdataene er som forventet

Tabell 1: En plan for hvordan en funksjon kan utvikles. Hentet fra Felleisen et al. (2001). Følgende aktiviteter gjennomføres (og kommentarer skrives over selve programkoden) før selv programkoden skrives.