Notat 2, ST1301

27. januar 2006

1 Sammensatte uttrykk

Vi har sett at funksjoner ikke trenger å bestå av annet enn ett enkeltuttrykk som angir hva funksjonen skal returnere uttrykkt ved de variable funksjonen tar som argument. I mange sammenhenger vil det imidlertid være nødvendig å sette sammen flere uttrykk for å beregne funksjonens utdata. Også i andre sammenhenger (se nedenfor) vil vi ha bruk for såkalte sammensatte uttrykk.

Et sammensatt uttrykk kan skrives på en eller flere linjer, f.eks. slik

```
{a <- 3; b <- 2; a*b; a+b}
```

eller slik

```
{
  a <- 3
  b <- 2
  a*b
  a+b
}
```

Hvis flere deluttrykk står på samme linje må disse skilles med tegnet semikolon. Plasseringen av klammeparantesene er ikke vesentlig for annet enn lesbarheten av programkoden (dette kommer vi tilbake til). Verdien av sammensatte uttrykk er lik verdien av siste deluttrykk i det sammensatte uttrykket. Om skriver inn uttrykket ovenfor i R-skallet får vi dermed

```
> {a <- 3; b <- 2; a*b; a+b}
[1] 5
```

La oss anta at vi ønsker å lage en funksjon som skal beregne løsningene av en annengradsligning på formen

$$ax^2 + bx + c = 0. \tag{1}$$

Denne har to løsninger på formen

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

$$x = \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \tag{3}$$

gitt at uttykket under rot-tegnet er positivt. Fordi de samme leddene inngår i begge løsningene kan det være hensiktsmessig å beregne disse først i to tilordningsuttrykk og så beregne de to løsningene. Setter vi disse operasjonene sammen får vi følgende funksjonsdefinisjon.

```
annengradsligning <- function(a,b,c) {
  ledd1 <- -b/(2*a)
  ledd2 <- sqrt(b^2-4*a*c)/(2*a)
  c(ledd1-ledd2,ledd1+ledd2)
}
```

Merk at det siste uttrykket (som returneres som funksjonsverdi) her består av et kall til funksjonene `c` som setter sammen en vektor bestående av verdien til til to løsningene av annengradsligningen.

Tester vi funksjonen får vi

```
> annengradsligning(1,-3,2)
[1] 1 2
```

som er riktig fordi $x^2 - 3x + 2 = (x - 2)(x - 1)$.

2 If-setninger

If-setninger får vi bruk for når vi ønsker å beregne ulike alternative uttrykk avhengig av verdien på andre variable. I eksempelet over kan det tenkes at annengradsligningen (for gitte argument) også kan ha én eller ingen løsninger. Dette vil avhenge av verdien av uttrykket under rot-tegnet.

Generellt har en if-setning syntaxen

`if (`*logisk uttrykk*`)` *gren 1* `else` *gren 2*

eller bare

`if (`*logisk uttrykk*`)` *gren 1*

hvor *logisk uttrykk* er et uttrykk med verdi av logisk datatype (verdi lik `TRUE` eller `FALSE`). *gren 1* er et uttrykk som utføres hvis det logiske uttrykket har verdien `TRUE`, hvis ikke utføres *gren 2*. Alle uttrykk som inngår i if-setningen kan være sammensatte. Hele if-setningen er i seg selv å betrakte som et uttrykk som får verdi lik verdien av *gren 1* eller *gren 2*. Merk også at hver gren selv kan være egne if-setninger (if-setninger kan være nøstede).

Gjør vi følgende

```
> if (3>2) z <- "sant" else z <- "usant"
> z
[1] "sant"
```

utføres tilordningsuttrykket `z <- "sant"` fordi det logiske uttrykket `3>2` har verdi `TRUE`. Bruker vi vanlige uttrykk og ikke tilordningsuttrykk i hver gren vil i stedet hele if-setningen få verdi lik uttrykket i den grenen som blir utført:

```
> if (3>2) "sant" else "usant"
[1] "sant"
```

Derfor kan hele if-setningen brukes på høyre side i tilordningsuttrykk:

```
> z <- if (3>2) "sant" else "usant"
> z
[1] "sant"
```

La oss skrive om funksjonen `annengradsligning` slik at denne fungerer returnerer alle løsninger for vilkårlige koeffisienter a, b og c. Vi må nå gjennomføre visse nye aktiviteter når vi skal programmere funksjonen, se tabell 1.

I vårt eksempel er det tre distinkte situasjoner som kan oppstå; avhengig av verdien av uttrykket under rot-tegnet $b^2 - 4ac$ kan vi ha ingen, én eller to reelle løsninger avhengig av verdien til uttrykket under rot-tegnet. Disse situasjonene bør vi legge inn som eksempler i eksempelfasen (se nedenfor). Legg merke til at når vi regner gjennom eksemplene for hånd ser vi hvilke uttrykk som må beregnes i hver enkelt distinkt situasjon.

Når selve kroppen til funksjonen skal skrives er det lurt å begynne med skjelletet av selve if-setningen. Fordi vi har tre distinkte situasjoner må den ene grenen nøstes. Grenene i if-setningen vil så bestå av de forskjellig uttrykk vi i eksempelfasen kom fram til at måtte beregnes i hver distinkt situasjon. Husk at verdien av if-setningen blir lik verdien av den grenen som beregnes. Derfor blir verdien som funksjonen returnerer lik den if-grenen som beregnes fordi if-setningen er siste uttrykk i det sammensatte utrykket som utgjør funksjonskroppen.

Fase	Mål	Aktivitet
Dataanalyse	Identifisere distinkte situasjoner som funksjonen må håndtere	Lete i oppgaveteksten etter distinkte situasjoner.
Eksempler	Lage eksempler for hver distinkt situasjon	Her kan det være spesielt viktig å se på unntakssituasjoner eller situasjoner på grensen mellom to intervaller.
Kropp	Å definere funksjonen	Skrive ned skjelettet av en if-setning eller en nøstet if-setning hvor hver if-gren svarer til hver distinkt situasjon. Utvikle uttrykk for de situasjoner som skal håndteres i hver gren av if-uttrykket.

Tabell 1: Utvidelser av designplanen for betingede funksjoner.

Merk bruken av sammensatte uttrykk og at `ledd1` og `ledd2` bare beregnes i de grener av if-setningen hvor de trengs i senere uttrykk. `NULL` gir oss en vektor av lengde null. Merk også bruken av innrykk for å markere ulike blokker av koden (hele funsksjonskroppen, og de nøstede if-grenene.

```
## annengradsligning : flyttall, flyttall, flyttall -> vektor
##
## Hensikt: Beregne alle reelle løsninger av en annengradsligning med
## koeffisienter a, b og c
##
## Eksempler:
## annengradsligning(1,-3,2) -> 2,1
## annengradsligning(1,2,1) -> 1
## annengradsligning(3,3,3) -> tom vektor
##
## Definisjon:
annengradsligning <- function(a,b,c) {
  rot.uttrykk <- b^2-4*a*c
  if (rot.uttrykk < 0)       # Negativt rotuttrykk
     NULL                    # Ingen reelle løsninger
  else {                     # Rotuttrykk lik null eller positivt
     ledd1 <- -b/(2*a)       # ledd1 trengs i begge undergrener
     if (rot.uttrykk > 0) {  # Positivt rotuttrykk
        ledd2 <- sqrt(rot.uttrykk)/(2*a)
        c(ledd1-ledd2,ledd1+ledd2) # To løsninger
     }
     else                    # Rotuttrykk må være lik null
        ledd1                # Én reell løsning (lik ledd1)
  }
}
```

3 For-løkker

Vi har sett at ulike aritmetiske operasjoner og de fleste funkjoner kan virke elementvis på vektorer slik at vi i realiteten kan utføre mange regneoperasjoner ved hjelp av ett uttrykk. Noen operasjoner er imidlertid genuint iterative og i slike tilfeller er det helt nødvendig å bruke såkalte løkker. Vi vil også bruke løkker i en del tilfeller hvor løkkebruk ikke er strengt tatt nødvendig men hvor bruk av løkker er enklere å forstå. I tillegg til for-løkker som vil bli introdusert nedenfor har vi også while-løkker og repeat-løkker i R (se ?) En for-løkke har syntaxen

`for (`*løkkevariabel* `in` *vektor*`)` *kropp*

hvor *vektor* er et uttrykk med verdi av datatypen vektor, *løkkevariabel* er navnet vi velger på en variabel som ved gjentatt utførelse av *kropp* vil ta verdier

fortløpende fra elementene i *vektor*. For-løkkens *kropp* kan være et uttrykk av en hvilken som helst type (sammensatt uttrykk eller tilordningsuttrykk) hvor *løkkevariabel* gjerne inngår.

Her er et eksempel:

```
> for (i in 1:5) x[i] <- i^2
> x
[1]  1  4  9 16 25
```

Tilordningsuttrykket `x[i] <- i^2` (løkkekroppen) vil her utføres fem ganger; ved hver enkeltutførelse tar løkkevariabelen `i` verdier hentet fra verdien av uttrykket `1:5` som jo blir en vektor med elementer lik $1, 2, \ldots, 5$. Resultatet blir at kvadratet av heltallene fra 1 til 5 blir tilordnet til de fem første elementene i vektor `x`.

En teknikalitet vi bør merke oss er at variablen `x` allerede må eksistere hvis vi ønsker å gjøre tilordning til enkeltelementer av `x` (ved bruk av indeksering), hvis ikke får vi en feilmelding.

```
> x[1] <- 1
Error: Object "x" not found
```

Et tomt men eksisterende objekt kan vi opprette slik

```
> x <- NULL
> x[1] <- 1
> x[2] <- 2
```

Dette ligner på deklarering i av variable i andre programmeringsspråk som C og Pascal.

Vi kan og merke oss at vi kunne ha oppnådd samme resultat som over uten bruk av for-løkke ved å skrive

```
> x <- (1:5)^2
> x
[1]  1  4  9 16 25
```

La oss se på et eksempel hvor de operasjoner vi må utføre er genuint iterative. I økologi brukes den logistiske modellen ofte som modell for hvordan en populasjon endrer seg i størrelse fra år til år. Denne modellen (se `?`, side 104) kan skrives på formen

$$\Delta N_t = N_t R \left(1 - \frac{N_t}{K}\right), \tag{4}$$

eller, etter litt omskriving,

$$N_{t+1} = N_t \left[1 + R \left(1 - \frac{N_t}{K} \right) \right]. \tag{5}$$

R er her en parameter som angir vekstraten til populasjonen; vi ser fra ligning (4) at $\Delta N_t = 0$ for $R = 0$ men populasjonen vil vokse for positive R og $N_t < K$. Dersom $N \ll K$ og $R = 1$ vil populasjonen dobles i størrelse hvert år ($\Delta N_t \approx N_t R = N_t$).

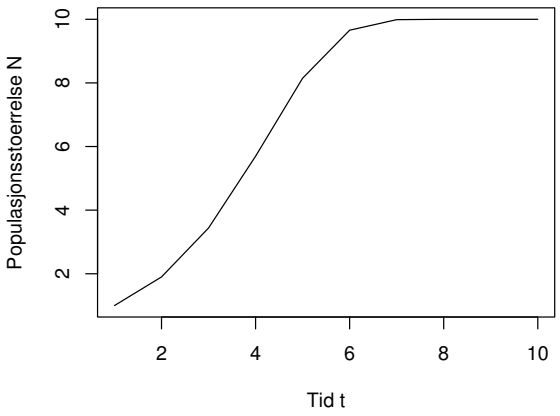
La oss lage en funksjon som beregner N_t fra $t = 2$ opptil $t = 10$ gitt N_1, vekstraten R, og bærekapasiteten K. Vi følger designplanen.

```
# logistisk : flyttall, flyttall, flytall -> vektor
#
# Hensikt: Beregne utviklingen i populasjonsstørrelse N gitt N ved
# t=1, bæreevne, og vektsrate.
#
# Eksempeler:
# logistisk(R=1,K=1000,N1=1) skal returnere tilnærmet 1,2,4,8,16
# o.s.v. siden vi er langt under bæreevnen K=1000
#
logistisk <- function(R,K,N1) {
  N <- N1
  for (t in 2:10)
     N[t] <- N[t-1]*(1+R*(1-N[t-1]/K))
  N
}
# Tester:
logistisk(1,1000,1)
logistisk(1,10,1)
plot(logistisk(1,10,1))
```

Ved vær utførelse av løkke-kroppen vil nå vektoren `N` utvides med ett element i posisjon `t`. Igjen må variabelen `N` eksistere før vi kan tilordne enkeltelementer. I og med initialbetingelsen $N_1 = 1$ gir det seg selv at `N` må tilordnes `N1` i første linje.

Testing av funksjonen viser at den fungerer som forventet; dersom $N \ll K$ vil populasjonsstørrelsen tilnærmet dobles de første årene dersom $R = 1$. Plotting av populasjonsutviklingen kan gjøres på følgende måte:

```
plot(1:10,logistisk(1,10,1),xlab="Tid t",
     ylab="Populasjonsstoerrelse N",type="l")
```



Figur 1: Populasjonsutvikling til populasjon med bæreevne K=10, $R = 1$ og $N_1 = 1$

Se hjelpesidene til `plot.default` for mer informasjon.

Selv om (5) ser ut som en enkel modell er det ikke mulig å uttrykke N_t ved N_1, R, og K på noen enkel måte — vi er nødt til å beregne løsningen iterativt numerisk.