R syntaks

Denne note er en introduktion[1] til syntaksen i R. Den kode, vi skal bruge til modellerne, står i bogen eller kommer til at være på hjemmesiden i den takt, vi gennemgår teorien. Så det, vi skal lære her, er de basale kommandoer i R, der gør, at vi kan regne og lave andet sjov som fx flotte grafer.

Installation af R

Eftersom R er et shareware program, der udvikles løbende af brugerne, kan det hentes gratis på nettet. Det findes på hjemmesiden www.r-project.org. Ude til venstre trykkes på download linket med teksten CRAN, så vælger i et mirror at downloade fra (vælg fx Denmark), derefter vælges Windows (hvis det er det system man køre), så trykker i på "base" linket og endelig skal I downloade installationsfilen, der hedder R-2.5.1-win32.exe (eller noget i den stil). Når I har kørt filen, så er I klar til at regne i R.

R platformen

Når I åbner R (fx ved at dobbelt-klikke på en genvej til programmet), så vil vinduet "R Console" automatisk åbne (alt dette refererer til Windows versionen). Afhængig af den R version i downloader vil udseendet være følgende:

```
R : Copyright 2006, The R Foundation for Statistical Computing
Version 2.3.1 (2006-06-01)
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Den sidste linje, der starter med en '>' er en command line eller prompten, hvor vi indtaster de kommandoer, som R skal udfører. Prøv at bruge R som en lommeregner ved at indtaste følgende og derefter taste på `<enter>` :

```
> 1+1
[1] 2
```

[1] Noten er stærkt inspireret af appendix A i Murrell, 2005, "R Graphics", Chapman & Hall og Søren Feodor Nielsens note "R for Statistik 1".

Ud over at være en lommeregner har R også alle de funktioner, I kender fra. Funktionerne hedder blot noget lidt andet, men de er der alle sammen et sted. Prøv fx:

```
> log(2)
[1] 0.6931472
```

Alle funktioner starter med et navn, som her er log, og derefter tager de et eller flere parametre. Det er ikke altid alle parametre, vi behøver at angive. Hvis vi udelader nogle, så har R nogle, den bruger som en standard. Et eksempel på dette er funktionen seq(). Den laver en vektor af tal eller en sekvens af tal:

```
> seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Men faktisk tager denne funktion mange flere parametre. Den samme række fås som:

```
> seq(from=1,to=10,by=1)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Hvis man er i tvivl om, hvor mange parametre, man skal indsætte i en funktion, eller hvilke muligheder en funktion egentlig giver, så har R en meget omfattende hjælpefunktion. Så hvis vi vil vide noget mere om seq() funktionen, så skriver vi:

```
> help(seq)
```

eller

```
> ?seq
```

Større programmer

Det er måske lidt kedeligt i længden at indtaste store programmer en linje af gangen, og det kan være lidt svært at reproducere nogle dage efter. I stedet kan man skrive R programmer i en text editor og derefter køre dem. Inde i R vælges `File > New Script`. Herefter kommer der en R editor frem. Her kan man skrive sine kommandoer og derefter køre dem. Det er mange gange bedst at skrive en kommando pr linje. Så er det ikke nødvendigt at afslutte linjen med et ; , som I kender det fra C++. Når I har skrevet de linjer, I gerne vil køre, så er det selvfølgelig en god ide at gemme programmet. Kommandoerne vil så blive gemt i en fil, der hedder noget med .R, og I kan åbne den igen senere ved at vælge `file > open script`. Når I ønsker at køre jeres program, så kan det gøres ved at vælge `edit > run all` eller ved at markere de linjer af programmet, der skal køres og derefter trykke `ctrl + r`.
Det er på denne måde I kører de programmer, der ligger på hjemmesiden.

Variable

I R kan man arbejde med variable, vektorer og matricer på samme måde som i C eller Mathematica. Hvis vi vil gemme en værdi i en variabel kan vi skrive:

```
> x <- log(2)
> x
[1] 0.6931472
```

Når vi gemmer tal eller bogstaver i en variabel, så brug <- til at tildele værdien til en bestemt variabel. I mange tilfælde kan man også bruge et =, men ikke altid. Så <- er god R syntaks, mens = er dårlig syntaks.
Vi gør på samme måde med en vektor:

```
> x <- seq(1,10)
> x <- x + 5
> mean(x)
[1] 10.5
```

Man kan lave sin egen vektor og vælge det tredje element ud ved at skrive:

```
> v <- c(-1,3,-2,5)
> v
[1] -1  3 -2  5
> v[3]
[1] -2
```

Vi kan også vælge ud på andre måder:

```
> v[2:3]
[1]  3 -2
> v[v>0]
[1] 3 5
```

Når vi vil lave en matrix er det nyttigt at kende funktionerne matrix(), rbind(), cbind() og diag(). Lad os lave en matrix med to søjler, der hver har tre elementer:

```
> x1 <- c(1,2,3)
> x2 <- c(4,5,6)
> m <- cbind(x1,x2)
> m
     x1 x2
[1,]  1  4
[2,]  2  5
[3,]  3  6
```

Vi kunne også lave matricen med 2 rækker og 3 søjler:

```
> m <- rbind(x1,x2)
> m
   [,1] [,2] [,3]
x1    1    2    3
x2    4    5    6
```

Når vi så vil tage den anden søjle ud eller den første række, så skriver vi:

```
> m[,2]
x1 x2
 2  5
> m[1,]
[1] 1 2 3

> m[2,3]
[1] 6
```

Vi kunne også lave en matrix med en bestemt dimension i et enkelt forsøg:

```
> m <- matrix(0,2,3)
> m
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
```

Det første argument i funktionen matrix fortæller, hvad R skal fylde matricen med. Det andet argument fortæller antallet af rækker og endelig giver det tredje argument antallet af søjler. En mere kompleks matrix er:

```
> m <- matrix(1:3,3,2)
> m
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
```

Endelig laver diag() en diagonal matrix. Prøv at se i hjælpfunktionen ?diag.

Der er mange flere eksempler på, hvad man kan lave af matricer m.v. i dokumentet ”An Introduction to R”, som findes under `Help > Manuals (in pdf) > ”An Introduction to R”`. Dette dokument er helt undværligt. Se på eksempler her, når I er i tvivl om noget. Det er også muligt at se eksempler direkte i R ved at skrive fx:

```
> example(matrix)
```

På denne måde er der eksempler på alle de R funktioner, I kunne finde på at bruge.

Funktioner

Når man skal bruge de samme kommandoer mange gange, men med forskellige parametre, kan det være en fordel at lave en funktion. Men det vidste I nok godt. I R kan vi også lave funktioner. Her er et eksempel på en simpel funktion:

```
f1 <- function(x,y){

      z <- abs(x)/y

      return(z)
}
```

Når den køres ser det sådan ud:

```
> f1 <- function(x,y){
+
+ z <- abs(x)/y
+
+ return(z)
+ }
>
> f1(-2,10)
[1] 0.2
```

Indlæse data fra en fil

Det er ret nemt at indlæse data i R, hvis data ellers ikke ligger på en meget mærkelig form. Når vi indlæser data, sker det tit med funktionerne read.table() eller read.csv(), og i bogen kan de også godt lide at bruge scan(). Disse funktioner tager rigtigt mange parametre som input, så man kan næsten indlæse hvad som helst. Når vi indlæser så er data ofte på standard form, så vi kan nøjes med kun få argumenter. Et eksempel kunne være:

```
> data <- read.table("C:\\port10.txt",header=FALSE)
```

Det sidste argument angiver om filen indeholder en overskrift på søljerne med data eller ej. Når vi indlæser en tabel, antages det, at hver observation har en linje for sig selv. Bemærk at når I angiver stien, hvor filen har hjemme, så brug \\ eller /.
Det data, som I indlæser, er nu indeholdt i matricen data. Faktisk er data, det der i R hedder en data.frame(), men det vender vi tilbage til.

Plot i R

En af de store fordele ved R er, at man kan lave utroligt smukke grafer. Vi vil dog nøjes med at lave illustrative grafer. Som standard indeholder R en mængde datasæt, som er umiddelbart tilgængelige. Lad os se på et af dem. Vi vælger det, der hedder pressure. Prøv at skrive:

```
> pressure
```

Og prøv så at skrive:

```
> str(pressure)
```

Selve datasættet ligger som en data.frame(). Det er en samling af dataelementer, som kan være både numeriske og/eller bogstaver/ord. I statistisk kan det både være kontinuerte vriable og faktorer. I en normal matrix kan vi ikke både have bogstaver og tal, man kan regne på, men kun en af delene. Derfor kan det være en fordel med disse data.frames. Men det er ikke noget, vi vil går meget i dybden med her.
Den sidste kommando, vi skrev, giver et overblik over de variable, vi har til rådighed i den pågældende data.frame. Når vi vil have en af de variable ud og arbejde med den, skriver vi:

```
> pressure$temperature
```

Så fungerer den som en almindelig vektor. Lad os plotte pressure mod temperatur. Det kan gøres på flere måde. Her er et udvalg:

```
> plot(pressure)
> plot(pressure$temperature,pressure$pressure)
> plot(pressure$pressure~pressure$temperature)
> plot(pressure~temperature,data=pressure)
```

Hvis I læser om plot i hjælpefunktionen (?plot), kan I se, at plot() funktionen tager utroligt mange parametre. Her er nogle af dem:

```
> plot(pressure,type='l',main="Top",xlab="x",ylab="y",xlim=c(0,400))
```

Af andre spændende argumenter er der lty, der styrer linjens udformning, lwd, der styrer linjens tykkelse, pch, der styrer punkters udseende osv. Se alle argumenterne under ?plot eller under ?par, som styrer de parametre, der gives til plot som default. Når I først har lavet et plot, så kan I gemme det i alle mulige formater ved at klikke på plottet, og gemme det under `file` menuen.
Vi kan tilføje nye plots til det eksisterende plot ved hjælp af funktionerne points(), lines() eller abline(). Den første funktion tegner et sæt punkter, den næste funktion tegner streger mellem et sæt punkter og den sidste linje tegner en ret linje efter en forskrift.

Langt de fleste funktioner vi skal bruge ligger direkte i R, men enkelte skal læses ind ud fra en pakke. Et eksempel kunne være:

```
> library(lattice)
> xyplot(pressure$pressure~pressure$temperature)
```

Løkker og logiske test

Ligesom I C har vi mulighed for at lave for-løkker, while-løkker eller if-sætninger mv. En for-løkke får udseendet:

```
> for(i in 1:10) { et-eller-andet }
```

Fx

```
> for(i in 1:10) {print(i)}
```

Mens en while løkke, ikke overraskende, så får udseendet:

```
> while(et-eller-andet) {noget-andet}
```

Og endelig var der en if sætning:

```
> for(i in 1:10) { if(i>=4) { print(i) } }
```

Hvis I får lavet en uendelig løkke, så er der en rød stopknap øverst oppe på værktøjslinjen. Generelt er for-løkker og while-løkker noget, der går langsomt i R, ligesom i mange andre programmer. Hvis man kan undgå at bruge en for-løkke, men i stedet lave det som matrixregning er det ofte at fortrække. Jeg kan dog ikke forestille mig, at vi i vores kursus skal køre så store for-løkker, at det gør en reel forskel.

Fordelinger og simulering

En af fordelene ved R er, at det er en statistisk programpakke, så det har en masse indbygget funktioner til statistik. Vi kan simulere, finde fraktiler, beregne fordelingsfunktioner og tætheder. Eftersom vi mest skal bruge en normalfordeling i vores tidsrækker, vil vi tage udgangspunkt i den.

Simulering:
Det gøres ved hjælp af funktionen rnorm(n,mean,sd). Den giver en vektor af længde n med tilfældige tal, der er trukket fra normalfordelingen med middelværdi mean og standardafvigelse sd. Default er en standardnormalfordeling. Funktionen rnorm er sammensat af r, som står for random og norm, som angiver normalfordelingen. Hvis man ville trække tal op af hatten fra en anden fordeling, så bruges en anden forkortelse end norm. Andre fordelinger kan ses i manualen eller man kan gætte sig frem.

```
> rnorm(100,1,2)
```

Fordelingsfunktion:
pnorm(q,mean,sd) giver fordelingsfunktionen i punktet q. Hvis q er en vektor, så er output også en vektor.

Tæthed:
dnorm(x,mean,sd) giver tætheden i x.

Fraktiler:
qnorm(p,mean,sd) giver fraktilen svarende til p.

Lineær regression

R kan estimere de fleste statistiske modeller som standard, og ellers så ligger de nok i en pakke et sted. De analyser, vi vil lave i vores fag, kræver dog, at vi selv programmere noget til sidste del af pensum. Men før vi kommer så langt, så er det meget godt at kunne lave noget lineær regression.

Det kan selvfølgelig gøres ved hjælp af nogle designmatricer, men R har også den indbyggede funktion lm(). Her er et lille eksempel:

```
> lm(pressure$pressure~pressure$temperature)

Call:
lm(formula = pressure$pressure ~ pressure$temperature)

Coefficients:
         (Intercept)  pressure$temperature
            -147.899                 1.512
```

Det er ikke meget information, man får her, men der ligger meget mere gemt derinde et sted. Prøv fx at skrive:

```
> fit <- lm(pressure$pressure~pressure$temperature)
> summary(fit)

Call:
lm(formula = pressure$pressure ~ pressure$temperature)

Residuals:
    Min      1Q  Median      3Q     Max
-158.08 -117.06  -32.84   72.30  409.43

Coefficients:
                       Estimate          Std. Error  t value     Pr(>|t|)
(Intercept)            -147.8989         66.5529     -2.222      0.040124 *
pressure$temperature   1.5124            0.3158      4.788       0.000171 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 150.8 on 17 degrees of freedom
Multiple R-Squared: 0.5742,     Adjusted R-squared: 0.5492
F-statistic: 22.93 on 1 and 17 DF,  p-value: 0.0001710
```

Der er næsten alt den information, vi kunne ønske os. Det er muligt at få endnu mere frem. Se under ?lm.
Hvis vi nu ønsker at plotte linjen sammen med vores data, så gør vi følgende:

```
> plot(pressure)
> abline(fit)
```

Det er en usædvanlig dårlig model. Så man kan i stedet prøve:

```
> fit<-lm(pressure~temperature+I(temperature^2),data=pressure)
> lines(pressure$temperature,fit$fitted.values)
```

Denne model er dog ikke meget bedre.