KØBENHAVNS UNIVERSITET
AFDELING FOR ANVENDT
MATEMATIK OG STATISTIK

Statistik 1
Søren Feodor Nielsen
15. december 2006

R for Statistik 1

Formålet med "R for Statistik 1" er at give en introduktion til den interaktive statistiske programpakke R. Der er ikke tale om en manual men om et undervisningamteriale. Noterne skal altså bruges til at lære de basale dele af R at kende. Derefter bør man benytte de indbyggede hjælpesider og de manualer eller bøger, der findes. Et udvalg af gratis manualer findes på CRAN (http://cran.r-project.org) under Manuals (officielle manualer) og under Contributed (uofficielle manualer); det anbefales at få fat på den officielle manual "An introduction to R" med det samme.

Hvert kapitel i noterne indeholder en beskrivelse af dele af R, eksempler og nogle mere eller mindre simple opgaver. Du bør gentage eksemplerne, så du kan se hvad der sker. Nogle af de objekter, der dannes i eksemplerne skal du bruge senere i opgaverne eller andre eksempler. En del af opgaverne er utvivlsomt en fornærmelse mod din intelligens men når de nu er så nemme, hvorfor så ikke løse dem? Enkelte af opgaverne er markeret med en stjerne; det skal forstås som et tegn på at dele af opgaven kan være lidt sværere end de øvrige og/eller mere tidskrævende. Det er ikke meningen at stjernen skal skræmme dig væk men kun være et tegn på at du skal afsætte et bestemt tidsrum til denne opgave og hvis du ikke får den løst inden for dette tidsrum skal du overveje at gå videre med noget andet.

Indhold

Uge 1 Introduktion

Uge 1-A: Tænd og sluk

R er freeware og er derfor gratis. Det kan downloades fra nettet i versioner til Linux, Windows og Macintosh (http://cran.r-project.org). Installation af R under Windows synes at være uproblematisk. Hvis man har Linux/Mac på sin pc er man sikkert selv i stand til at downloade og installere udfra den vejledning, der findes på nettet (http://cran.r-project.org). Her på instituttet er R installeret på både Linux- og Windows-maskiner; skulle man finde en undtagelse fra denne regel, bør man orientere helpdesk@math.ku.dk med det samme.

R startes på Linux systemet ved at give kommandoen R i en xterminal:

```
shannon:~/> R

R : Copyright 2005, The R Foundation for Statistical Computing
Version 2.1.0  (2005-04-18), 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.

  Natural language support but running in an English locale

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 a HTML browser interface to help.
Type 'q()' to quit R.


>
```

">" er prompten; det er her du skriver dine kommandoer. Man afslutter sin R-session ved at give kommandoen q(). Bemærk parenteserne; de er vigtige! q er en funktion, som kaldes uden argumenter (eller rettere med nogle default-argumenter, som der ikke er grund til at rette på).

Når man afslutter sin R-session bliver man spurgt:

```
> q()
Save workspace image? [y/n/c]:
```

workspace image indeholder alle de variable, vektorer, funktioner, data etc, man har oprettet i løbet af sin session. Svarer man y, gemmes de og indlæses næste gang man starter R. Svarer man n er de tabt. Om man svarer y eller n er lidt en smagssag, men denne forfatter svarer altid y. *Du bør så længe du er i gang med at lære R svare y til at gemme dit workspace image*

Under Windows startes `R` fra Start-menuen. Resultatet er essentielt det samme. Man afslutter igen med `q()`.

Opgave a) Tænd og sluk for `R`. ○

Uge 1-B: Test i binomialfordelingen, hjælpesider

Resten af denne uge vil fokusere på at vise hvorledes nogle af de statistiske test, I kender fra SaSt 1 og SaSt 2 er implementeret i `R`.

I opgave 1 på opgavearket til uge 1 omtales en undersøgelse om fødselsmisdannelser og lykkepiller. I den pågældende undersøgelse er der observeret 1054 fødsler af mødre, der har benyttet lykkepiller; der blev observeret misdannelser ved 56 af disse fødsler[1]. Test af om hyppigheden af fødselsmisdannelser adskiller sig fra den almindelige hyppighed, som er 4%, er implementeret ved `binom.test(56,1054,.04)`.

Opgave a) Test hypotesen og undersøg om du forstår output. ○

`R`s indbyggede funktioner har hjælpesider. Hjælpesiden for f.eks. `binom.test` findes ved at skrive `?binom.test` eller `help(binom.test)`.

Opgave b) Læs hjælpesiden for `binom.test`; er det de rette argumenter du har givet til `binom.test`-funktionen? ○

Uge 1-C: T-test, tilknytning af data

For at kigge på t-test, skal vi først have fat på et datamateriale. Vi vil her benytte et klassisk datamateriale, som er indeholdt i `R`.

Opgave a) Gør data tilgængeligt ved kommandoen `data(sleep)`. ○

Opgave b) Læs hjælpesiden: `?sleep`. ○

Opgave c) Kig på selve data ved bare at skrive `sleep` ved prompten og trykke return. ○

T-testet er implementeret som funktionen `t-test`. Argumenter til denne funktion kan angives på flere forskellige måder. Det nemmeste er nok `t.test(extra~group, data=sleep, var.equal=TRUE)` Her angiver vi altså først at vi vil undersøge søvnforlængelsen (`extra`) som funktion af hvilken behandling (`group`), der er givet, i form af *modelformelen* `extra~group`. Data-materialet er `sleep`. Det sidste argument, `var.equal=TRUE`, sikrer at vi benytter det t-test vi kender fra SaSt2.

Opgave d) Udfør t-testet. ○

[1] Data er delvist rekonstrueret fra publicerede oplysninger.

Opgave e) Giv den samme kommando –det nemmeste er at trykke en enkelt gang på pil op og så redigere i kommandoen før der trykkes `return`– og fjern `data=sleep` fra kommandoen. ○

Vi har altså ikke umiddelbart adgang til de enkelte variable i data-materialet. For at få dette skal vi først tilknytte data: `attach(sleep)`. Kommandoen `search` giver en liste over tilknyttede datamaterialer, biblioteker (mere herom senere) o.a. Der vil være biblioteker (`package:navn`) på listen selv om du ikke har tilknyttet disse; de bør under ingen omstændigheder fjernes.

Opgave f) Giv først kommandoen `search()`. Tilknyt derefter `sleep` og gentag så `search()` og udfør så t-testet uden argumentet `data=sleep`. Husk at du kan benytte "pil-op" til at få gamle kommandoer frem igen. ○

Opgave g) Udfør også t-testet ved kommandoen `t.test(extra[group==1], extra[group==2], var.equal=TRUE)`. Hvad er `extra[group==1]`[2] i øvrigt? ○

Data afknyttes igen med kommandoen `detach(navn)`.

Opgave h) Afknyt data og se efter at det ikke længere optræder når du giver kommandoen `search()`. ○

Uge 1-D: Lineær regression, indlæsning af data

Data, som er gemt i en tekstfil, indlæses i `R` med kommandoen `read.table`. Her skal man som argument angive fil-navn (i gåse-øjne) og hvis den første linie i filen er variabel-navnene så også `header=TRUE`:

```
> my.data<-read.table('filnavn',header=TRUE)
```

Resultatet af denne kommando er at data lægges ind i `my.data` som en `data.frame`. Argumentet `header=TRUE` fortæller `R` at den første linie i tekstfilen ikke indeholder data men derimod variabel-navne. Bemærk at hvis filnavnet indeholder backslash (\) (Windows-stil), skal du enten skrive dobbelt-backslash (\\) eller en almindelig slash (/; Unix-stil).

Her vil vi benytte et datamateriale `puzzle.dat` og sammenhængen mellem studentereksamenssnit (`StudEksamen`), resultat af en "intelligenstest" (`Tid`) og Statistik 1A resultat (`Point`).

Opgave a) Download datamaterialet fra kursushjemmesiden og indlæs det i `R` som en `data.frame`. Giv denne `data.frame` navnet `puzzle` (du skal altså erstatte `my.data` med `puzzle` i kommandoen ovenfor). Kig på det med `summary(puzzle)`. Tilknyt det (`attach`) så du får adgang til de enkelte variable. ○

Vi vil her undersøge hvordan resultatet af eksamen `Point` afhænger af snittet fra studentereksamen `StudEksamen`.

[2] Indicering er noget vi vil vende tilbage til.

Opgave b) Start med at plotte data: `plot(StudEksamen, Point)`. ○

Lineær regression fittes med funktionen `lm`. Her skal vi –som med t-testet– angive en modelformel som argument (vi er interesserede i `Point` som funktion af `StudEksamen`).

Opgave c) Fit den lineære regression. ○

Resultatet er sparsomt: Vi får essentielt kun estimaterne af hældning (`StudEksamen`) og skæring (`(Intercept)`) ud. Vi kan benytte funktionen `summary` til at få langt mere information frem.

Opgave d) Fit igen den lineære regression og gem denne gang resultatet: `fit<-lm(Point~StudEksamen)`. Benyt så `summary` med `fit` som argument. Hvor meget af output forstår du? ○

Opgave e) Læg en regressionlinie ind i scatterplottet ved at kalde kommandoen `abline` med `fit` som argument. ○

Uge 1-E: Objekter og funktioner

Du har sandsynligvis bemærket at vi lige har benyttet funktionen `summary` på resultatet af `lm` mens vi tidligere benyttede `summary` på et datamateriale. Vi fik to vidt forskellige resultater frem. En lang række funktioner gør forskellige ting alt efter hvilken type argument vi propper ind.

Opgave a) Kald funktionen `class` på `puzzle` og derefter på `fit`. ○

Data og resultatet af en lineær regressionsanalyse er altså objekter af forskellig klasse og `summary` behandler dem derfor forskelligt. Vi har tidligere set en slags eksempel på dette ifm t-testet, som gør noget forskelligt alt efter om vi angiver en modelformel eller to vektorer.

Opgave b) Kald `plot` på `puzzle` og derefter på `fit` (outputtet af det sidste er det ikke meningen du skal kunne forstå). ○

Uge 1-F: Hvad har du lært?

I dette kapitel skal du have lært:

- At tænde og slukke `R`
- At indlæse data, tilknytte og afknytte det samt et par metoder til at kigge på det
- At foretage test i binomialfordelingen, t-test og fitte en lineær regression
- Lidt om `R`s objekt struktur.

Uge 2 Datastrukturer

Uge 2-A: Vektorer og recycling

Kommandoen `ls()` giver en liste over de variable etc, der er konstrueret ("indholdet af `workspace image`"). Hvis dette er tomt, får man

```
> ls()
character(0)
```

Hvis du har gemt dit `image` i sidste uge, skulle der gerne ligge lidt forskelligt.

Opgave a) Giv kommandoerne

```
> a<-1
> a
```

Pilen `<-` skrives med et mindre end-tegn og et minus. Tjek med `ls()` at `a` findes. Prøv dernæst

```
> 2->b
> b
```

Hvad er resultatet af `a<-b`? ○

`<-` og `->` er altså "assignment". En anden mulighed for "assignment", som jeg *ikke* vil anbefale, er `=`.

Opgave b) Hvad gør `a=b`? Hvad gør `a==b`? Hvorfor synes jeg at man bør undgå at bruge `=`? ○

Hvis man altid gemmer sit `workspace image` bliver det hurtigt meget stort. Derfor er det en god ide at slette ting man ikke længere skal bruge. Variablen `a` slettes vha `rm(a)`, både `a` og `b` med `rm(a,b)` og alting med `rm(list=ls())`. Bemærk at "=" her ikke er en assignment men en angivelse af værdien af argumentet `list`.

Opgave c) Slet `a` og `b`. ○

Vektorer er en grundlæggende struktur i `R`. En vektor `a` bestående af tallene $1, 2, 6, 88$ laves således

```
> a<-c(1,2,6,88)
```

Vi kan derefter gange `a` (elementvis) med 3:

```
> 3*a
[1]   3   6  18 264
```

En vektor `b` bestående af $1, 2, 3$ og 4 kan laves tilsvarende eller således

```
> b<-1:4
```

Opgave d) Lav de to vektorer `a` og `b` og se at de indeholder det forventede. Gang dem derefter sammen:

```
> a*b
```

Hvad er resultatet? ○

To vektorer ganges (og adderes og . . .) altså sammen elementvist. Men man kan også gange vektorer sammen selv om de ikke har samme længde:

```
> b<-1:2
> a*b
[1]   1   4   6 176
```

Det, der sker, er at den kortere vektor genbruges så mange gange som er nødvendigt ("recycling").

Opgave e) Hvad er resultatet af

```
> d<-1:3
> a+d
```

mon? ○

Andre typer vektorer fås ved kommandoerne `seq` og `rep`[1]:

```
> seq(7,14,by=2)      #vektor fra 7 til 14 med spring på 2
[1]  7  9 11 13
> seq(7,14,length=5)  #vektor fra 7 til 14 af længde 5
[1]  7.00  8.75 10.50 12.25 14.00
> rep(0,5)            #0-vektor af længde 5
[1] 0 0 0 0 0
```

Vektorer kan indeholde boolske værdier (`logical`), tal (`numeric`), eller bogstaver (`character`) men ikke blandinger; hvis man prøver at lave en blandet vektor bliver alle elementerne af den sidste type i den orden som er angivet ovenfor (boolske værdier bliver til 0 og 1, tal bliver til "bogstaver"). Typen kan aflæses med funktionen `mode`.

Opgave f) Prøv funktionerne `mode` og `length` på `a`, `b` og `d`. Hvad gør `length`? ○

Mere interessant er nok enten at tage matematiske funktioner så som `exp` og `sin` på en vektor –de virker så elementvis og returnerer en vektor af samme længe som den oprindelige– eller statistiske funktioner så som `mean` og `var`, som returnerer et tal (middelværdien og variansen).

Opgave g) Prøv disse funktioner (og evt. andre du kan gætte hvad hedder). ○

[1] Vi tilføjer kommentarer; de skal naturligvis ikke skrives med i din `R`-session. `#` er kommentartegn i `R`.

Uge 2-B: Indicering

Man har også ofte behov for at se på en del af en vektor. F.eks.

```
> a<-c(3,5,6,9)
> a[1]           #a's første element
[1] 3
> b<-2:1
> b
[1] 2 1          #bemærk 2 før 1
> a[b]
[1]  5 3         #a's andet og derefter det første element
> a[-b]
[1] 6 9          #a pånær det andet og det første element
> a[2*b]
[1] 9 5          #a's fjerde og andet element
```

Logisk indicering er meget nyttigt. Her er en række simple eksempler:

```
> a[a>5]
[1] 6 9          #de elementer af a der er større end 5
> a[a!=5]-a[a==5]
[1] -2  1  4     #de elementer af a der er forskellige fra 5 minus dem
                 #der er 5
> a[a%%3==0]
[1] 3 6 9        #de elementer af a der er delelige med 3
```

`sort` er naturligvis den funktion, der ordner:

```
> sort(a[2*b])
[1] 5 9
```

Hvis man har to vektorer `a` fra før og `b` som f.eks. er

```
> b<-c(b,4,-2)   #danner b udfra den gamle b ved at tilføje 4 og -2
> b
[1]  2  1  4 -2
```

så kunne man være interesseret i at ordne `b` med den mindste først og derefter ordne `a` så ordningen af `a` er den samme som `b`s, dvs da

```
> sort(b)
[1] -2  1  2  4
```

vil vi gerne have `a` ordnet `9 5 3 6`. Det gøres ved

```
> a[order(b)]
[1] 9 5 3 6
```

Uge 2-C: Matricer

Matricer kan i lighed med vektorer kun have elementer af én type. De konstrueres normalt ved at omforme en vektor:

```
> a
[1] 3 5 6 9
> b
[1]  2  1  4 -2
> A<-matrix(a,nrow=2)   #en matrix med to rækker
> A
     [,1] [,2]
[1,]    3    6
[2,]    5    9
> B<-matrix(b,ncol=2)   #en matrix med to søjler
> B
     [,1] [,2]
[1,]    2    4
[2,]    1   -2
```

I dette eksempel er det naturligvis ligegyldigt om vi angiver antal rækker (`nrow`) eller søjler (`ncol`). Man kan også benytte `cbind` til at binde vektorer sammen søjlevis:

```
> cbind(a,b)
     a  b
[1,] 3  1
[2,] 5  2
[3,] 6  4
[4,] 9 -2
```

`rbind` binder sammen rækkevis. Alternativt kan man gøre `b` til en matrix ved at give den dimensioner:

```
> dim(b)<-c(2,2)                        #b gives dimensioner
> b                                     #hvad er b?
     [,1] [,2]
[1,]    2    4
[2,]    1   -2
> is.matrix(b)                          #er b en matrix?
[1] TRUE
> b==B                                  #b er det samme som B
     [,1] [,2]
[1,] TRUE TRUE
[2,] TRUE TRUE
> dim(b)<-c(1,4)                        #nye dimensioner for b
> b
     [,1] [,2] [,3] [,4]
[1,]    2    1    4   -2
> is.matrix(b)                          #er b nu en vektor?
[1] TRUE                                #nej! stadig en matrix
> dim(b)<-NULL                          #fjern b's dimensioner
```

```
> is.matrix(b)                          #nu er b ikke længere en matrix
[1] FALSE
```

Den anden mulighed er at lave en "tom" matrix:

```
> E<-matrix(nrow=3,ncol=2)
> E
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
[3,]   NA   NA
```

og så fylde ind elementvis. Inden da bemærker vi at `NA` står for *not available*; `NA` indikerer altså at elementerne i matricen mangler.

Opgave a) Prøv følgende kommandoer og se efter *hver* kommando hvad `E` nu er:

```
> E[,1]<-a[1:3]
> E[-3,]<-B
> E[3,2]<-mean(b)
```

○

Endelig kan diagonalmatricer konstrueres med funktionen `diag`:

```
> diag(3)             #en 3x3 identitetsmatrix
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> diag(c(7,45))       #en diagonalmatrix
     [,1] [,2]
[1,]    7    0
[2,]    0   45
> diag(B)             #B's diagonal
[1]  2 -2
```

Hvis vi ganger matricerne `A` og `B` sammen, så får vi

```
> A*B
     [,1] [,2]
[1,]    6   24
[2,]    5  -18
```

Der er altså tale om elementvis multiplikation, ikke matrix-multiplikation. Matrix-multiplikation fås ved

```
> A%*%B
     [,1] [,2]
[1,]   12    0
[2,]   19    2
```

Naturligvis kan vi mere end bare gange sammen; `A` transponeres med funktionen `t`, den inverse til `A` findes som `solve(A)`, determinant fås vha `det`, egenværdier og -vektorer vha `eigen`. Derimod synes der ikke at være en funktion til beregning af sporet.

Opgave b)* Sporet af en kvadratisk matrix er summen af diagonalelementerne. Beregn sporet af `A` ved at "pille" diagonal elementerne ud og summere dem. ○

Opgave c) Betragt ligningssystemet

$$Ex = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

hvor E er matricen konstrueret ovenfor. Der er tale om 3 ligninger med 2 ubekendte, så det er ikke givet at der findes en løsning. Vi vil prøve at løse systemet alligevel. En måde er at gange med E^t på begge sider og så invertere E^tE. Derved får vi

$$x = (E^tE)^{-1}E^t \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

som vi så kan sætte ind i den oprindelige ligning ("gøre prøve") og se om det er en løsning. Find den mulige løsning og gør prøve. ○

Uge 2-D: Lister, dataframes, arrays

Data repræsenteres ofte som en "matrix" hvor hver række svarer til et individ/en observation mens hver søjle repræsenterer en variabel. Da variable ofte er af forskellig type –nogle er kontinuerte (`numeric`) og andre er diskrete (ofte `character`)– kan et sådant datamateriale ikke repræsenteres som en `R`-matrix. Man bruger i stedet objekter af typen `data.frame`. En `data.frame` er en `list` af klasse `data.frame`:

```
> mode(puzzle)
[1] "list"
> class(puzzle)
[1] "data.frame"
```

Et objekt af typen `list` er bare en slags generaliseret"vektor" hvor hvert element kan være hvad som helst (egentlige vektorer, matricer, andre lister, etc). Man kan hive elementer i lister ud på mange måder.

Opgave a) Prøv følgende kommandoer:

```
> puzzle$Point
> puzzle[2]
> puzzle[[2]]
```

Hvad får man ud af dem? ○

Ganske mange `R`-funktioner giver en liste som resultat; et eksempel er `eigen`, der giver en liste med to elementer, nemlig `values` og `vectors`. Resultatet af `lm` er også en liste.

Opgave b) Hvad er resultatet af at benytte funktionerne `is.list`, `class` og `names` på objektet `fit` dannet i sidste uge? ○

Opgave c) Find igen sporet af matricen `A`, denne gang ved at summere egenværdierne i stedet for diagonalelementerne. ○

Til tider kan det være nyttigt at opfatte et datamateriale som et objekt med mere end to dimensioner fremfor som en matrix (som jo har netop to dimensioner). Til det (og andre) formål findes datastrukturen `array`, som kan ses som en generalisering af vektorer og matricer.

Opgave d) Load datamaterialet `iris3` og find dets dimensioner. Hvad indeholder hver af dimensionerne? ○

Man kan selv lave et `array` ved at tage en vektor og give den passende dimensioner eller mere direkte ved

```
> x<-array(1:8,c(2,2,2))  #et 2x2x2 array
> x
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

Det er oftest en god ide at oprette et `array` af de ønskede dimensioner (f.eks. `x<-array(0,c(2,2,2))` og så fylde det ønskede indhold ind bagefter, men man kan naturligvis også oprette en vektor og derefter give den dimensioner som vi gjorde for matricer.

Uge 2-E: Løkker

Når man programmerer, har man ofte behov for løkker og if-then-else strukturer. Det kan man naturligvis også benytte sig af i `R`. Men hvis man kan undgå løkker ved at bruge f.eks. vektorer, kan man vinde meget. F.eks. kan man naturligvis beregne et matrix-produkt ved hjælp af for-løkker men det er da noget nemmere at benytte den indbyggede funktion `%*%`. Dertil kommer at det vil tage længere tid; for store matricer kan det blive ganske alvorligt:

```
> A<-matrix(rep(1,100^2),nrow=100)
> system.time(ans<-A%*%A)
[1] 0.01 0.00 0.01 0.00 0.00
> system.time(for (i in 1:100) {for (j in 1:100) {ans[i,j]<-sum(A[i,]*A[,j])}})
[1] 0.48 0.04 0.54 0.00 0.00
```

```
> system.time(for (i in 1:100) {for (j in 1:100) {
+ ans[i,j]<-0
+ for (k in 1:100) {ans[i,j]<-ans[i,j]+A[i,k]*A[k,j]}}})
[1] 25.15  0.03 25.81  0.00  0.00
```

Her er en masse vi skal forstå. Først og fremmest giver `system.time` den forbrugte tid målt i sekunder; det første er bruger cpu-tiden, det næste system cpu-tiden og det tredje er en total tid. Vi ser at det tager meget længere tid med to for-løkker og helt *latterligt* lang tid med tre.

Dernæst ser vi at for-løkker ser således ud:

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

n er naturligvis hvor mange gange vi skal igennem; `et-eller-andet` er det, der skal gøres i løkken. Hvis vi skal flere ting, indrammer vi dem i krøllede paranteser.

Det er også muligt at lave while-løkker:

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

`et-eller-andet` er et udsagn der er enten sandt eller falsk, `noget-andet` er det der skal udføres sålænge `et-eller-andet` er sandt. Uendelige løkker kan ofte afbrydes med `control-c`.

Hvis man skal benytte den samme funktion på alle søjlerne i en matrix (f.eks.) kan man i stedet for at bruge en løkke anvende kommandoen `apply`:

```
> E
     [,1]  [,2]
[1,]    2  4.00
[2,]    1 -2.00
[3,]    6  1.25
> mean(E)             #gennemsnit af alle elementer
[1] 2.041667
> apply(E,2,mean)     #søljevis gennemsnit
[1] 3.000000 1.083333
> apply(E,1,mean)     #rækkevis gennemsnit
[1]  3.000 -0.500  3.625
```

`apply` er muligvis en lille smule hurtigere end en `for`-løkke. Det hurtigste her er nu at gange med en vektor.

Opgave a) Find middelværdien af hver søjle ved at gange `E` med en vektor. ○

`apply` virker også på `arrayer`. For lister er der en tilsvarende funktion, `lapply(obj,f)`, som anvender funktionen `f` på hvert element af listen `obj`; resultatet er en liste. Funktionen `sapply` gør det samme som `lapply` men forsøger at strukturere resultatet som en matrix eller en vektor.

Opgave b) Find middelværdien af hver variabel i datamaterialet `puzzle` ved at give kommandoerne `mean(puzzle)`, `lapply(puzzle, mean)`, `sapply(puzzle,mean)`

if-then-else strukturer er også mulige:

```
> if (is.matrix(ans)){ cat('Determinanten er ',det(ans),'\n')} else
+ { cat('Hov! Dette er ikke en matrix\n')}
Determinanten er  0
```

Her er de krøllede parenteser overflødige; de behøves kun når der er flere ting man skal gøre i then-delen eller else-delen.

Hvis vi nu vil tage en vektor `x` af ikke negative tal og f.eks. finde `x[i]*log(x[i])` (med $0 \log 0 = 0$) kunne vi gøre følgende:

```
> x<-c(0,1:2,0,5)
> for (i in 1:length(x)) if (x[i]>0) x[i]<-x[i]*log(x[i])
> x
[1] 0.000000 0.000000 1.386294 0.000000 8.047190
```

men vi ved nu at løkker skal skys. Vi kan med fordel benytte

```
> x<-c(0,1:2,0,5)
> x<-ifelse(x>0,x*log(x),0)
> x
[1] 0.000000 0.000000 1.386294 0.000000 8.047190
```

`ifelse` virker altså elementvis på en vektor.

En teknisk forklaring. Når løkker er tidsrøvere i `R` skyldes det delvist at vektoroperationer er implementeret meget effektivt; de er derfor hurtige. Men fordi `R` er et interaktivt sprog skal det være muligt at komme videre selv om der går noget galt i en løkke. For at sikre det foretages en masse overflødig kopiering og det øger tidsforbruget, særligt for store løkker.

Uge 2-F: En vigtig advarsel

Man kan få lov at kalde variable, funktioner, datamaterialer, ... stort set hvad som helst i `R`. Man kunne f.eks. definere en konstant `pi` ved

```
> pi<-0
```

Dette er muligt selv om `pi` er den matematiske konstant π. Det siger sig selv at at omdefinere f.eks. `pi` kan have meget uheldige konsekvenser, hvis man derefter bruger en funktion, som benytter `pi` (dvs π).

Med tiden bliver `R` mindre og mindre følsomt overfor den slags men man bør alligevel undgå at omdefinere indbyggede konstanter, funktioner mm. Derfor bør du aldrig benytte `c` eller `t` som variabel/funktionsnavne eftersom disse to funktioner bruges et utal af steder. Heller ikke `T` og `F` bør omdefineres; de er forkortelser af `TRUE` og `FALSE`.

Uge 2-G: Hvad har du lært?

I dette kapitel skal du have lært:

- `R`s fundamentale datastrukturer at kende
- hvordan man indicerer i `R`
- hvordan man bruger løkker og if-then-else i `R`
- at løkker kun må bruges når der ikke er nogen anden udvej

Uge 3 Fordelinger, plots og egne funktioner

Uge 3-A: Fordelinger

I `R` er indbygget en lang række af de mest almindelige fordelinger i den forstand at `R` kan simulere fra disse fordelinger, beregne fordelingsfunktioner og tætheder samt finde fraktiler. Lad os se på normalfordelingen:

- Simulering: `rnorm(n,mean,sd)` giver en vektor med længde n af tilfældige tal fra normalfordelingen med middelværdi `mean` og spredning `sd` (bemærk det er spredning og ikke varians der skal angives). Hvis man ikke angiver middelværdi og spredning er defaulten standardnormalfordelingen.
- Fordelingsfunktion: `pnorm(q,mean,sd)` giver fordelingsfunktionen i punktet `q`; hvis dette er en vektor fås tilsvarende en vektor af værdier.
- Tæthed: `dnorm(x,mean,sd)` giver tætheden i `x` (evt en vektor).
- Fraktiler: `qnorm(p,mean,sd)` giver fraktil(er) svarende til (vektoren) `p`

Andre fordelinger fås på tilsvarende vis; man skal bare erstatte `norm` med den ønskede fordelings navn. Nedenfor er der en ufuldstændig liste med fordelinger, deres navn i `R` og deres parametre[1] i `R`; hvis der står et tal ved en parameter, så er det *default*-værdien for denne parameter.

Fordelinger i `R`

Fordeling	navn i `R`	Parametre (med evt defaultværdier)
Binomial	`binom`	længde og successandsynlighed
Geometrisk	`geom`	sandsynlighed
Poisson	`pois`	middelværdi
Negativ binomial	`nbinom`	”size” og sandsynlighed
Ligefordeling	`unif`	interval endepunkter=0 og 1
Eksponential	`exp`	intensitet=1
Cauchy	`cauchy`	position=0 og skala=1
Normal	`norm`	middelværdi=0 og spredning=1
Gamma	`gamma`	form og intensitet
χ^2	`chisq`	frihedsgrader
Beta	`beta`	formparametre
F	`f`	frihedsgrader
t	`t`	frihedsgrader
Weibull	`weibull`	form og skala=1

Opgave a) Find middelværdien af en fordeling efter eget valg ved at simulere en vektor af længde 5000 og så tage middelværdien af denne. Gør dette et par gange. Hvorfor ændrer værdien sig? ○

[1] For en del fordelingers vedkommende kan man også angive en *ikke-centralitetsparameter*; defaultværdien er her altid 0.

Uge 3-B: Egne funktioner

I sidste uge så vi en effektiv måde at beregne $x \log x$ på med konventionen $0 \log 0 = 0$. Hvis dette er noget, vi ofte har brug for, ville det være rart om vi kunne konstruere en permanent funktion `xlogx` som gav netop dette resultat. Heldigvis er dette muligt i `R`:

```
> xlogx<-function(x){ifelse(x>0,x*log(x),0)}
```

Funktionen `xlogx` er nu dannet og kan anvendes når som helst vi måtte ønske det:

```
> xlogx(c(0,1:2,0,5))
> 0.000000 0.000000 1.386294 0.000000 8.047190
```

Faktisk er mange af `R`s indbyggede funktioner af denne form, dvs funktioner som er skrevet i `R` og som kalder andre `R`-funktioner. Se f.eks. på funktionen `sd` som beregner standardafvigelser:

```
> sd
function (x, na.rm = FALSE)
{
    if (is.matrix(x))
        apply(x, 2, sd)
    else if (is.vector(x))
        sqrt(var(x, na.rm = na.rm))
    else if (is.data.frame(x))
        sapply(x, sd)
    else sqrt(var(as.vector(x), na.rm = na.rm))
}
```

Dette er jo næsten til at forstå: `is.matrix(x)` er sand hvis `x` er en matrix; `as.vector(x)` "tvinger" `x` til at være en vektor. Funktionen `sd` finder altså ud af hvad type objekt `x` er og beregner så en passende standardafvigelse (kvadratroden af variansen); for en matrix søjlevis (vha `apply`), for en `data.frame` variabelvis (`sapply`).

Vi bemærker at `sd` tager to argumenter: `x`, som er det objekt vi skal finde standardafvigelser på, og `na.rm`, som man ikke behøver at angive fordi der er en default-værdi (nemlig `FALSE`). Argumentet `na.rm` afgører hvad funktionen gør når der en manglende observationer i `x`; hvis `na.rm` er `FALSE` så får vi en fejlmeddelelse ud hvis der er manglende værdier (`=NA`), mens hvis `na.rm` er `TRUE` får vi standardafvigelse beregnet på de ikke-manglende værdier ud.

Opgave a) Tilknyt `puzzle` hvis det ikke allerede er tilknyttet og kald funktionen `sd` på `StudEksamen` både med `na.rm=FALSE` og med `na.rm=TRUE`. ○

Egentlige programmer i `R` skrives ofte med fordel som `R`-funktioner. Disse bliver ofte længere end den ene linie vi benyttede os af ovenfor (`xlogx`) og det gør det svært at undgå fejl undervejs. Hvis der er fejl, bliver man så nødt til at starte forfra (og så laver man en ny fejl ...). Det vil derfor være en fordel at skrive sine programmer i en flad tekstfil (vha Xemacs eller Notepad). Når filen så er gemt som (f.eks.) `myfct.R` kan man *source* den ind. I Windows kan man gøre det via menuen mens man i Linux er nødt til at kalde en funktion:

```
> source('myfct.R')  #det kan være nødvendigt at angive en hel sti
```

Hvis programmet indeholder fejl, vil man få en fejlmeddelelse og så må man jo i gang med at finde fejlen.

Filen behøver ikke hedder `R` til efternavn; det er bare praktisk så man ved at det er et `R`-program. En fil man sourcer ind, kan indeholde hvad som helst; evt kommandoer vil blive udført, så man kan også bare skrive sine kommandoer i en source-fil og så source dem ind. Typisk får man ikke noget synligt resultat af sine anstrengelser; for at få resultater ud, skal man bruge kommandoen `print`:

```
#Source-fil:
summary(puzzle)         #Intet resultat
print(summary(puzzle))  #Det ønskede resultat
```

Alternativt kan man "source med ekko":

```
> source('myfct.R',echo=T)
```

Så får man kommandoer og output frem.

Opgave b) Lav denne source-fil og source den ind i `R`. ○

Tilføjelse. Skal man køre store programmer er det nyttigt at kunne køre dem som batch, så man kan gå hjem og holde weekend imens. Dette gøres under Linux ved at give kommandoen

```
shannon:~/> R CMD BATCH myfct.R output.log &
```

hvor filen `myfct.R` indeholder ens kommandoer og filen `output.log` tager imod output; disse filer kan naturligvis hedde hvad som helst. Derefter kan man logge af og vende tilbage senere.

Uge 3-C: Plots

En af `R`'s stærke sider er de grafiske muligheder. Den grundlæggende funktioner er her funktionen `plot`, som laver forskellige plots alt efter hvad man bruger som argument.

Opgave a) Prøv følgende kommandoer:

```
> plot(puzzle)
> attach(puzzle)
> plot(StudEksamen,Point)
> plot(as.factor(round(StudEksamen)))
> plot(Tid)
```

○

Man kan opnå finere kontrol med grafen ved at specificere et yderligere antal argumenter. De umiddelbart vigtigste er nok

`type`**:** `plot(StudEksamen,Point,type='`*n*`')` hvor typen *n* kan være `p` (punkter), `l` (linier), `b` (begge; punkter og linier), `n` (intet; mere nyttigt end du tror!) mm. Defaultværdien er `p`.

`pch` og `lty`**:** `plot(StudEksamen,Point,pch=2)` erstatter cirklerne i grafen med trekanter; andre tal giver andre plot-symboler. `lty` giver tilsvarende andre linietyper.

`xlim` og `ylim`**:** `plot(StudEksamen,Point,ylim=c(50,85))` sikrer at anden aksen går fra 50 til 85 (plus en lille smule i hver ende).

`xlab`**,** `ylab`**,** `main`**:** `plot(StudEksamen,Point,xlab='`*navn*`')` skriver *navn* under første aksen; `main` giver overskrift.

Opgave b) Prøv følgende kommandoer

```
> plot(1:20,pch=1:20)
> plot(1:20,col=1:20)
```

Andre nyttige plotte-kommandoer er `points`, som sætter punkter ind på et allerede eksisterende plot, og `lines`, som tilsvarende sætter linier ind. `abline` sætter en ret linie ind i et eksisterende plot; man skal angive skæring og hældning eller for en vandret (horisontal) linie gennem 2 `h=2` og for en lodret (vertikal) `v=2`, f.eks.:

```
> plot(StudEksamen,Point)
> abline(v=9,lty=2)
```

Opgave c) Lav en graf med `StudEksamen` ud af første aksen og `Point` op af anden aksen med forskellige plotsymboler alt efter om `Tid` er skarpt mindre end 5 eller ej ved at angive argumentet `pch=1+(Tid<5)`. ○

Det er ofte nyttigt med flere grafer på en gang:

```
> par(mfrow=c(2,3))
```

opdeler grafikvinduet i 2×3 del-vinduer, som så udfyldes rækkevis (hvis de skal udfyldes søjlevis bruges `mfcol`).

Opgave d) Opdel grafikvinduet som ovenfor i 2 gange 3 felter og gentag så `hist(rnorm(50))` 6 gange (husk at du kan genkalde kommandoer med pil-op). ○

`par(mfrow=c(1,1))` fjerner opdelingen igen.

En anden mulighed er at åbne et nyt grafikvindue; det gøres med kommandoen `X11()`. Vinduerne får så numre (det første grafikvindue er `device 2`, det næste `device 3` etc) og man kan bestemme hvilket vindue er det "aktive" ved kommandoen `dev.set`:

```
> X11()          #åbner nyt vindue
> dev.cur()      #hvilket device er det aktive?
X11
  3              #nummer 3 er
> dev.set(2)     #gør nummer 2 aktivt
X11
```

```
  2
> dev.cur()
X11
  2
> dev.off()     #sletter det aktive device
X11
  3             #nummer 3 er nu aktivt
```

Man får hurtigt behov for at udskrive eller gemme sine grafer. I Windows gøres dette lettest ved at benytte grafikvinduets menu. Andre muligheder (som er nødvendige for Linux brugere) er:

```
> dev.print()     #printer grafen i det aktive vindue ud
> dev.copy2eps()  #gemmer grafen som eps-filen Rplot.eps
                  #(som kan inkluderes i LaTeX-dokumenter)
> dev.copy2eps(file='myplot.eps')
                  #gemmer i filen myplot.eps
```

Opgave e) Lav igen grafen fra Opgave c) og gem den i en fil. ○

Også pindediagrammer kan konstrueres:

```
> barplot(table(as.factor(round(StudEksamen))))
```

Her sikrer funktionen `as.factor` at variablen `StudEksamen` behandles som en variabel af typen `factor`, det vi ville kalde en kategorisk variabel. Funktionen `table` laver så en antalstabel:

```
> table(as.factor(round(StudEksamen)))
```

Vi bemærker at resultatet af vores `barplot`-kommando er det samme som det `plot(as.factor(round(StudEksamen)))` giver.

Ofte vil man være interesseret i at se på fordelingen af en variabel, dvs for en kontinuert variabel på et histogram:

```
> hist(StudEksamen)         #antal
> hist(StudEksamen,prob=T)  #andel: som regel det bedste!
```

Opgave f) Tegn histogrammet over studentereksamensresultater med andel op ad y-aksen. Tilføj en normalfordelingskurve ved først at lave en vektor `x` som antager 200 værdier jævnt fordelt over histogrammets x-akse (benyt funktionen `seq`) og derefter at kalde funktionen `lines` på følgende måde:
`lines(x,dnorm(x,mean(StudEksamen,na.rm=T),sd(StudEksamen,na.rm=T)))`. Tilføj et tæthedsestimat (som beskrevet i lærebogens afsnit 4.6) ved
`lines(density(StudEksamen,na.rm=T),lty=2)`. Ser det ud til at være rimeligt at tro at studentereksamens resultater er cirka normalfordelte? ○

Husker du da du i gymnasiet skulle sætte prikker på "normalfordelingspapir"? Det står i bekendtgørelsen[1] at det skal man lære i gymnasiet, så det har du sikkert

[1] . . . og det griner vi ofte af . . .

prøvet. I dag er der absolut ingen som bruger normalfordelingspapir fordi fornuftige computerprogrammer kan lave disse grafer for os. Sådanne grafer hedder QQ-plots eller fraktildiagrammer og er nærmere beskrevet i Sand 1 bogens afsnit 14.3. I `R` tegner man på "normalfordelingspapir" med kommandoen `qqnorm`:

```
> x<-rnorm(500)
> qqnorm(x)
> qqline(x)
```

Punkterne i grafen skulle gerne ligge omkring linien (som `qqline` tegner).

Opgave g) Tegn et normalfordelingsfraktildiagram for `StudEksamen`; husk at sætte en linie ind. Kan man med rimelighed antage at `StudEksamen` er normalfordelt? ○

Opgave h) Åbn et nyt grafikvindue og del det i 3 gange 3 felter. Gentag kommandoen `qqnorm(x<-rnorm(60));qqline(x)` 9 gange; læg mærke til hvordan vi på en gang simulere normalfordelinger gemmer resultatet i `x` og laver fraktildiagram, hvorefter vi sætter en ret linie ind (`;` adskiller de to plot-kommandoer). Hvordan ser graferne ud? Synes du på baggrund af disse grafer at det kunne være rimeligt at tro at studentereksamensresultaterne er normalfordelte? ○

Normalfordelingen er naturligvis vigtigere end så mange andre fordelinger, men i dag er det let nok at lave QQ-plot for andre fordelinger end normalfordelingen. Desværre skal man programmere en lille smule i `R`:

```
> qqt<-function(x,df){ plot(qt(ppoints(x),df),sort(x),
+  main=paste('t(',df,') QQ Plot',sep=""),
+  xlab='Theoretical quantiles', ylab='Sample quantiles') }
```

`qqline` skal også rettes; den bruger `qnorm`, dvs normalfordelingen.

Opgave i) Find ud af hvad den nævnte funktion gør helt præcist ved at starte fra en ende af og se hvad hver funktion gør i sig selv. Start altså med `ppoints`, så `qt`, `sort` og `paste` og slut med `plot`. ○

Opgave j)* Lav en QQ-plot funktion for en (eller flere) fordeling(er) efter eget valg (men ikke normal- og t-fordelingerne). Inkluder en `qqline`-funktionalitet i din funktion. Funktionen, som du bør skrive i en source-fil, skal altså se cirka sådan ud:

```
qqt<-funktion(x,df,line=T){
     #tegn qqplottet
     if (line) #tegn linien
     }
```

Særligt det med at inkludere en linie kan være svært. En forholdsvis nem (men ikke helt tilfredsstillende) "løsning" kunne være at lægge diagonalen ind. ○

Uge 3-D: Hvad har du lært?

I dette kapitel skal du have lært:

- hvilke fordelinger der findes i `R` og hvordan de bruges.
- hvordan du laver egne funktioner i `R`
- hvordan du laver de helt basale grafer -grafer med punkter og linier, histogrammer, pindediagrammer og søjlediagrammer- i `R` og hvordan du kan sætte tekst på aksen mm

Uge 4 Biblioteker, test og mere

Uge 4-A: Likelihood inferens

Opgave a) Start med at konstruere de data vi har set på ved forelæsningerne: `dat<-c(98,215,431,601,814,929)` ○

Opgave b) Plot data mod årstal (1984,...,1989). ○

Det ser rimeligt ud at antage at væksten er lineær altså at $E[X_y] = \alpha + \beta(y - 1984)$. Derudover vil vi antage $X_y$erne uafhængige og Poissonfordelte.

Opgave c)* Et simpelt estimat for (α, β) fås ved mindste kvadraters metode (OLS eller ordinary least squares). Det svarer i denne situation hvor middelværdien er en ret linie til almindelig lineær regression (se evt SaSt 2-noterne). Find et estimat ved at bruge `lm` (se Uge 1); du skal enten konstruere en vektor, som indeholder tallene 0,...,5, eller du kan anvende `0:5` direkte i `lm` hvis du "beskytter" den: `I(0:5)`[1]. Gem estimaterne for skæring og hældning som variablene `a` og `b`; du kan hive estimaterne ud af resultatet af `lm` med kommandoen `coef`:

```
> a<-as.numeric(coef(lm(...))[1])
> b<-as.numeric(coef(lm(...))[2])
```

(`as.numeric` tvinger resultatet til at være et tal; uden den vil der følge en label med hvilket kan give problemer senere). Sæt den rette linie ind i plottet med `abline(a-b*1984,b)`. ○

Opgave d) Lav log-likelihoodfunktionen:

```
> ll<-function(alfa=a,beta=b){-sum(dpois(dat,alfa+beta*0:5,log=T))}
```

Default-værdierne for parametrene α og β er ikke nødvendige men vil være nyttige sidenhen. Læg mærke til hvordan `ll` benytter vektoren `dat` og hvordan vi specificerer middelværdien; hvad er resultatet af `1+1*0:5`? Argumentet `log=T` kan angives for de fleste fordelinger; det sikrer at der tages logaritmen til tætheden. ○

Et sted at starte kunne være at plotte log-likelihood. Da der er tale om en funktion af to variable skal vi enten plotte i "tre dimensioner" eller lave plot af niveaukurver (se lærebogens figur 3.4). Det vil naturligvis være relevant at gøre dette i nærheden af maksimum, så lad os starte med et estimat:

Opgave e) Lav vektorer `alfa` og `beta` omkring dit estimat hver af længde 200; benyt funktionen `seq`. Husk fra forelæsningerne at særligt estimatet for α er for småt ift maksimum likelihood estimatoren så din `alfa`-vektor skal ikke ligge symmetrisk om estimatet. ○

Opgave f) Udregn herefter log-likelihood i disse 200 gange 200 punkter og gem resultatet i en 200 gange 200 matrix med navn `log.likelihood`: Start med at lave

[1]Kolon har en bestemt betydning i en modelformel, så `0:5` bliver misforstået af `lm`; funktionen `I` "beskytter" indmaden så den bliver opfattet som den er skrevet.

matricen `log.likelihood<-matrix(nrow=200,ncol=200)`, og fyld den derefter ud (en `for`-løkke indeni en `for`-løkke synes at være nødvendig. ○

Opgave g) Plot med niveaukurver laves med funktionen `contour`:

```
> contour(alfa,beta,log.likelihood)
```

`contour` vælger nogle niveauer selv og resultatet vil næppe være tilfredsstillende. Vælg bedre niveauer ved at kalde funktionen igen men denne gang også angive en vektor med niveauer som argumentet `levels` og sæt titler på de to akser (ved at angive argumentet `xlab` og `ylab`) samt en overskrift (`main`). ○

Opgave h) 3-d plot fås ved funktionen `persp` med de samme argumenter. Dog kan man ikke angive `levels`. I stedet kan man muntre sig med at ændre drejningen på plottet ved at angive vinklerne `theta` og `psi`. ○

`R` består at en basis-installation og en lang række af pakker eller biblioteker skrevet af forskellige statistikere. Listen af pakker er usandsynlig lang og det er svært at skaffe sig et overblik over hvad der er nyttigt. Du kan se hvilke pakker der er installeret ved kommandoen `library()`. Pakker installeres vha kommandoen `install.packages` eller i Windows via menuen. 3-d visualisering er meget populært og der er en række af pakker til `R` som indeholder funktioner til lige netop dette.

Opgave i) Fremkald listen over installerede biblioteker: `library()`. Se efter om pakker `rgl` er installeret; hvis ikke så installer den (på din egen computer). Load biblioteket ved `library(rgl)`, se hvad det indeholder ved `library(help=rgl)`. Biblioteket `rgl` optræder nu på den liste du får frem med `search()`. ○

Opgave j) Lav et nyt 3-d plot med `rgl.surface(alfa,beta,log.likelihood)`. Drej det ved at føre musen hen over det. Farverne mm kan naturligvis ændres; se hjælpesiderne til `rgl.bg` (for baggrundsfarven) og `rgl.surface`. ○

For at maksimalisere likelihoodfunktionen kan vi benytte `R`s indbyggede optimeringsrutine `optim` eller den udgave vi finder i pakken `stats4`.

Opgave k) Load biblioteket `stats4`. Funktionen `mle` giver maksimum likelihood estimatoren for den log-likelihood der angives som førsteargument: `mle(ll)`. Hvis man ikke har angivet default værdier i `ll` skal man give argumentet `start=list(alfa=a, beta=b)` (hvor start-værdierne naturligvis kan vælges som det passer dig men det vil da være dumt ikke at vælge dem i nærheden af det sande maksimumspunkt). I vores eksempel skal maksimaliseringen foretages over $(\alpha, \beta) \in \mathrm{R}_+^2$; dette sikrer man ved at angive `method="L-BFGS-B"` og `lower=c(0,0)` som argumenter også. Find MLE'en. ○

Advarsel: Pakken `stats4` (som hører til i basis-distributionen) er på mange måder temmelig atypisk konstrueret og de finere detaljer kan ikke forstås på baggrund af disse noter.

For at finde ud af hvilken fordeling estimatorerne har kan vi lave et simulationseksperiment.

Opgave l)* Følgende bør gøres i en source-fil! Start med at oprette to 2 gange `N` matricer til at gemme resultater i:

```
ans.mle<-ans.ols<-matrix(ncol=2,nrow=N)
```

`N` bør i praksis være forholdsvis stor (5000) men i denne øvelse bør du nok nøjes med et nogen mindre tal (50) for ikke at skulle vente for længe på resultaterne. Derefter skal vi bruge en `for`-løkke:

```
for (i in 1:N){
  #simuler data
  #find ols-estimater, dvs estimer med mindste kvadraters metode
  #find ml-estimater
  }
```

Vi kan simulere data med kommandoen: `dat<-rpois(6,alfa+beta*0:5)`. For `alfa` og `beta` skal vi bruge nogle estimater, f.eks. de ML-estimater vi har fundet ovenfor. OLS-estimaterne findes med `lm` og skal så lægges ind i den ene matrix:
`ans.ols[i,]<-as.numeric(coef(lm(dat~I(0:5))))`. Og tilsvarende for ML-estimaterne (også her skal du bruge funktionen `coef` som piller estimaterne ud af resultatet af `mle`) og `as.numeric`, som tvinger resultatet til at blive en numerisk vektor. Source dine kommandoer. ○

Opgave m) Resultaterne af simulationerne skal naturligvis præsenteres fornuftigt. En mulighed er at lave box-plot. For at kunne sammenligne de to estimater for α er det nok nemmest at samle estimaterne i en `data.frame` således:

```
> alfa.est<-data.frame(estimat=c(ans.ols[,1],ans.mle[,1]),
+ metode=rep(c('OLS','ML'),each=N))
```

(argumentet `each=N` sikrer at vi først tager tekststrengen `OLS` `N` gange og derefter `ML` `N` gange. Vi kan så lave boxplot med

```
> boxplot(estimat~metode,alfa.est)
```

Her angiver vi altså en modelformel (hvordan "afhænger" `estimat` af `metode`) og så `data.framen`. Gør dette for begge parametre så dine to boxplot står ved siden af hinanden i et grafikvindue. ○

Opgave n) Biblioteker fjernes igen med kommandoen `detach`: F.eks.
`detach(package:rgl)`. Fjern de to biblioteker `rgl` og `stats4`. ○

Uge 4-B: Test

Vi har tidligere set på `t.test` funktionen: `t.test(extra~group, data=sleep, var.equal=TRUE)`. Her kan vi også angive to vektorer i stedet for model-formelen. Eller vi kan bare angive en vektor og få et én-stikprøve t-test. I så fald vil vi få et test af hypotesen $H_0 : \mu = 0$. Hvis det er en anden nul-hypotese $H_0 : \mu = \mu_0$ kan den angives med argumentet `mu=`μ_0. Parrede t-test fås hvis man angiver to vektorer og argumentet `paired=TRUE`. Man kan ændre sit signifikansniveau ved at angive en værdi for `conf` (1 minus signifikansniveauet).

Vi har ved forelæsningerne snakket om styrkefunktioner. Styrkefunktionen for t-testet (eller rettere t-testene) er implementeret i `R` ved funktionen `power.t.test`. Her skal man angive 2 ud af de tre argumenter:

`n`	stikprøvestørrelse
`delta`	sand middelværdi minus middelværdi under hypotesen
`power`	styrke

Den sande spredning `sigma` er som default sat til 1, svarende til at `delta` i virkeligheden er sand middelværdi minus middelværdi under hypotesen delt med spredningen, og signifikansniveauet (`sig.level`) er sat til 5%; begge kan naturligvis ændres ligesom man kan vælge hvilken type t-test man er interesseret i (defaulten er to uafhængige stikprøver). Endelig bør man angive argumentet `strict=TRUE` (defaulten er `FALSE`) for at få begge haler med.

Opgave a) Find ud af hvor stor en stikprøve man skal bruge hvis man ved test af forskel mellem to (lige store) grupper vil have 80% sikkerhed (styrke) for at opdage en effektiv forskel på $\pm 0{,}5$. ○

Opgave b) Tegn styrkefunktioner op i et fælles plot for stikprøvestørrelserne 5, 10, 25 og 50, f.eks. ved først at tegne den ene styrkefunktion med `plot` og dernæst tilføje de andre med `lines`. Start med at oprette en vektor `delta` af længde 200, der går fra 0 til den største effektive forskel, du synes er interessant. Find så styrken i disse punkter med funktionskaldet:

```
> power.t.test(delta=delta,n=5, strict=T)$power
```

Resultatet af `power.t.test` er en liste af klasse `htest` og vi piller elementet `power` ud som vi vil plotte mod `delta`. Benyt forskellige linjetyper til de forskellige værdier af `n` og sæt til sidst en `legend` på med kommandoen

```
> legend(locator(1),paste('n=',c(5,10,25,50),sep=""),lty=1:4)
```

(hvis dine linjetyper er 1 til 4) ved at klikke på grafen der hvor du vil have din `legend` sat; `locator(1)` er den der gør det muligt at placere `legend`'en manuelt (alternativet er at angive koordinater), `paste` kommandoen konstruerer teksten ved at klistre `n=` på hver af værdierne i `c(5,10,25,50)` og argumentet `sep=` propper ingenting ind imellem `n=` og tallet (alternativt kan man angive teksten som en vektor af længde 4 med teksten i) og `lty` giver linjestumperne. ○

Også rang-teststørrelser er implementeret i `R` som funktionen `wilcoxon.test`. Opbygningen er helt analog: Vi får Wilcoxons rangsumsteststørrelse ved at angive en model-formel eller to vektorer, Wilcoxons signed rank test for én stikprøve ved bare at angive en vektor, og test for parrede data hvis man angiver to vektorer og `paired=T`.

Opgave c) Løs opgave 8.3 i lærebogen ved brug af `R`; data kan downloades fra nettet via kursushjemmesidens side om lærebogen. Vær sikker på at du forstår output. ○

Opgave d) Løs opgave 8.10 i lærebogen ved brug af `R`; data kan downloades fra nettet via kursushjemmesidens side om lærebogen. Vær sikker på at du forstår output.○

I opgaverne til kapitel 8 nævnes også Kendalls tau og Spearmans rangkorrrelationskoefficient. `R` beregner korrelationer med funktionen `cor`; her kan angives `method` som kan være `"pearson"` (almindelig korrelation) `"kendall"` eller `"spearman"`; det er nok at angive det første bogstav (`"p"`, `"k"` eller `"s"`). Hvis man vil have en p-værdi for test af hypotesen om at korrelationen er 0, skal man i stedet bruge funktionen `cor.test`.

Opgave e) Find de tre typer af korrelationer ved funktionen `cor.test` mellem ML-estimaterne for α og β fundet tidligere. ○

Uge 4-C: Hvad har du lært?

I dette kapitel skal du have lært:

- hvordan du bruger biblioteker/pakker i `R`
- hvordan du finder maksimum likelihood estimater
- mere om plots (niveukurver, flader, boxplot)
- hvordan du laver et simulationseksperiment og derved undersøger fordelingen af estimatorer eller lignende
- mere om de test der er implementeret i `R`

Uge 6 Den generelle linære model

Uge 6-A: Introduktion: Simpel lineær regression

Vi har tidligere set hvordan en simpel lineær regression kan fittes vha funktionen `lm`.

Opgave a) Start med at hentes om indlæse datamaterialet hørende til opgave 11.3 i lærebogen. Kald datamaterialet `dat.11.3` og tilknyt det. ○

Opgave b) Start med at kalde funktionerne `summary` og `plot` på `dat.11.3`. Bemærk at den kategoriske variabel `Faggruppe` opsummeres på en anden måde end de to andre variable. Undersøg `Faggruppes` "klasse". ○

I opgaven bliver vi bedt om at fitte en simpel lineær regression af `Blodtryk` på `Alder` for hver `Faggruppe`.

Opgave c) Start med at plotte `Blodtryk` mod `Alder` med forskellige farver/plottesymboler for hver `Faggruppe` (f.eks. `pch=1+(Faggruppe=='journalist')`) ○

Opgave d) For at fitte den lineære regression for `journalist` skal den del af materialet der hører til `Faggruppe=='journalist'` "pilles" ud. Det kan gøres på mindst tre forskellige måder:

```
> lm(Blodtryk[Faggruppe=='journalist']~Alder[Faggruppe=='journalist'])
```

er måske mest direkte mens

```
> lm(Blodtryk~Alder,data=dat.11.3[Faggruppe=='journalist',])
```

er mere subtil. Bedst er måske

```
> lm(Blodtryk~Alder,subset=(Faggruppe=='journalist')
```

Gør dette for hver faggruppe for sig, gem de to fit og tilføj de to regressionslinier til dit plot. ○

Opgave e) Inspicer de to fit vha `summary`. ○

Uge 6-B: lm, modelformler

Opgave 11.3 beder os derefter om at fitte en samlet model. Her vil det være rimeligt at starte med at lave et "Bartlett's test" dvs et test af om variansen i de to regressionsmodeller med rimelighed kan antages ens. Teststørrelsen er her forholdet mellem de to variansestimater som under hypotesen om ingen forskel er F-fordelt og har store og små værdier kritiske.

Opgave a) Udfør Bartlett's test ved at kalde funktionen `var.test` med de to `lm`-fit som argumenter. ○

Hvis Bartlett's test godkendes, så giver det mening at fitte en generel lineær model med middelværdi

$$E[\texttt{Blodtryk}] = \alpha_{\texttt{Faggruppe}} + \beta_{\texttt{Faggruppe}} \cdot \texttt{Alder}$$

Dette kunne gøres ved at opstille en designmatrix `A` og så kalde `lm`:

```
> lm(Blodtryk~A-1)
```

Men det er besværligt at opskrive designmatricer og bliver hurtigt uoverskueligt. ("-1" er vigtigt; hvorfor forklares nedenfor). I stedet specificeres modellen ved hjælp af en modelformel som kan ses som en kodning af designmatricen. Modelformler ser således ud

```
venstresiden ~ højresiden
```

På venstresiden står en enkelt variabel (her `Blodtryk`) som er den vi i lineære normale modeller vil modellere middelværdien af, altså vores observation. På højresiden står en formel bestående af en eller flere "baggrundsvariable/-kovariater/regressorer/-faktorer" –i vores eksempel `Alder` og `Faggruppe` kombineret med `+`, `-`, `:` og/eller `*`. Bemærk at disse symboler betyder noget helt bestemt i en modelformel. Hvis man vil have dem opfattet som almindelige regnesymboler (fordi man f.eks. vil lave lineær regression på `Alder-1`) så skal man beskytte sådanne udtryk med beskytterfunktionen `I`:

```
> lm(Blodtryk~I(Alder-1)) #lineær regression på alder-1
> lm(Blodtryk~Alder-1)    #lineær regression på alder med skæring i 0
```

Lad os starte med de simpleste:

```
Blodtryk~Alder
```

er modelformlen for lineær regression; højresiden specificerer en designmatrix med to søjler, en bestående af 1-taller og en bestående af værdierne i variablen `Alder`. Hvis højresiden består af en faktor[1] (en kategorisk variabel)

```
Blodtryk~Faggruppe
```

får vi en ensidet variansanalyse. Her vil designmatricen igen bestå først af en søjle 1-taller og derefter af en søjle for hver værdi faktoren antager (her 2) således der i hver række er netop 2 1-taller, et fra den første søjle og et fra den søjle der svarer til den værdi af faktoren som hører til observationen. Modellen er tydeligt overparametriseret (hvorfor?) men den slags kan `R` godt klare for os.

Opgave b) Fit den ovenfor beskrevne ensidede variansanalyse og kald `summary` på resultatet. Fit derefter den samme model men med en anden parametrisering:

```
> summary(lm(Blodtryk~Faggruppe-1))
```

Læg mærke til at `-1` fjerner "interceptet" og dermed overparametriseringen, mens `R`s default er at "fjerne" det (alfabetisk) første niveau i faktoren. Hvordan skal parameterestimaterne tolkes? Hvilken parametrisering der er bedst er ikke entydigt klart; de har hver deres fordele og det er derfor nyttigt at kende dem begge. ○

Flere baggrundsvariable kan kombineres. F.eks. giver modelformlen

```
Blodtryk~Faggruppe+Alder
```

[1] En faktor i lærebogen er af afbildning $f : I \to F$; den tilsvarende faktor i `R` er vektoren $(\mathtt{f}(i))_{i \in I}$

en specifikation af middelværdien svarende til at tage designmatricen hørende til `Faggruppe` og den for `Alder` og "klistre" dem sammen; vi får altså en middelværdi i span af søjlerne i de to designmatricer (eller en sum af underrum). Bemærk at 1-søjlen automatisk kommer med med mindre vi beder om at få den hevet ud. Dermed får vi en model hvor middelværdien af `Blodtryk` beskrives af et niveau for hver `Faggruppe` og en lineær effekt af `Alder`, altså to *parallelle* regressionslinier.

Opgave c) Fit den beskrevne model både med intercept og uden. Hvilken parametrisering synes du er bedst? ○

For at få forskellige regressionslinier (altså en fælles model der svarer til de to simple lineære regressioner) skal vi have forskellige hældninger på de to linier.

Opgave d) Fit modellen

```
> lm(Blodtryk~Faggruppe+Alder:Faggruppe-1)
```

og se efter at du får de samme middelværdi-estimater som du fik i de to simple lineære regressioner. ○

`Alder:Faggruppe` giver altså "vekselvirkningen" mellem de to variable `Alder` og `Faggruppe`. Parametriseringen er valgt så estimaterne bliver som tidligere men i nogle sammenhænge kan andre parametriseringer være mere nyttige.

Opgave e) Fit modellen igen, nu med overparametrisering:

```
> lm(Blodtryk~Faggruppe+Alder+Alder:Faggruppe)
```

Hvad betyder parametrene her? (Sammenlign med estimaterne fra forrige spørgsmål). Samme model fittes lidt mere fikst ved

```
> lm(Blodtryk~Faggruppe*Alder)
```

Med andre ord er `Blodtryk~Faggruppe+Alder+Alder:Faggruppe` og `Blodtryk~Faggruppe*Alder` samme modelformel. Vær i øvrigt opmærksom på at `R`s parametrisering kan afhænge af "faktorenes orden". ○

Vi ved at estimatorerne i en lineær normal model er normalfordelte omkring den sande værdi og har et udtryk for variansmatricen.

Opgave f) Find den estimerede variansmatrix for estimaterne med funktionen `vcov` kaldt på dit `lm`-fit. ○

Uge 6-C: Mere om modelformler

Modelformler er nøglen til at fitte (lineære) modeller i `R`. Det er derfor vigtigt at vide hvordan en given model kan formuleres som modelformel og hvordan `R` fjerner overparametriseringer og dermed hvordan man kan fremtvinge en bestemt parametrisering.

Man bør tænke sig at `R` læser en modelformel fra venstre om højre og at `R` læser `+`

i en modelformel som en instruktion om at tilføje en eller flere søjler til designmatricen.

Der er to grundlæggende baggrundsvariable: En kovariat, som er en numerisk variabel, og faktoren som er en kategorisk variabel. Når en kovariat tilføjes i en modelformel, så tilføjes designmatricen en søjle som bare er kovariaten selv. Når en faktor tilføjes i en modelformel, så tilføjes søjler til designmatricen svarende til en designmatrix for den ensidede variansanalyse som er givet ved faktoren (se f.eks. Eksempel 12.5 i lærebogen). Rækkefølgen af niveauerne i faktoren er med mindre andet er angivet "alfabetisk"; man kan med kommandoen `levels(faktor)` se hvilken rækkefølge der er udgangspunktet og kan endvidere ændre rækkefølgen ved

```
> faktor<-factor(faktor,levels=ny.ordning)
```

hvor `ny.ordning` er en vektor af `character` som indeholder de gamle niveauer i en ny rækkefølge. En vekselvirkning `kovariat:faktor` giver søjler som er det koordinatvise produkt af kovaritaten og hver af faktor-designmatricens søjler; dermed kommer der i en søjle til at stå 0 hvis der i den tilsvarende søjle i faktor-designmatricen står 0 men værdien af kovariaten hvis der står 1 i designmatricen. En vekselvirkning mellem faktorer `faktor1:faktor2` giver bare designmatricen svarende til produktfaktoren.

Vi husker at den trivielle faktor `1` altid er med i modelformlen med mindre vi eksplict fortæller `R` at vi ikke ønsker den med. Og vi husker at `a*b` er kort notation for `1+a+b+a:b`.

Hver gang der tilføjes nye søjler til designmatricen, så er der en risiko for at matricen ikke længere har lineært uafhængige søjler. `R` undersøger for hver tilføjelse om søjlerne stadig er lineært uafhængige; hvis det ikke er tilfældet fjerner `R` søjler fra designmatricen ved først at fjerne den første søjle blandt de som lige er tilføjet, dernæst den anden, osv indtil designmatricen igen har fuld rang. Det betyder at ved overparametriseringer så er det altid det første niveau af faktoren som benyttes som referenceniveau (dvs dens parameter sættes lig 0) og parametrene for de øvrige værdier af faktoren er altså forskellen mellem det første niveau og de andre.

Altså i pseudo-kode:

```
while (not end of formula){
  nye.søjler<-søjler svarende til den næste led i modelformlen
  while ((cbind(design.matrix,nye.søjler) ikke fuld rang){
     nye.søjler<-nye.søjler[,-1]
     }
  design.matrix<-cbind(design.matrix,nye.søjler)
  }
```

Virkeligheden (se evt. `model.matrix.default`) er dog noget mere kompliceret; specielt skal man være opmærksom på at `1` fjernes først uanset hvor i modelformlen vi skriver `-1`. Derudover kan man instruere `R` om at benytte andre algoritmer til at fjerne overparametriseringer (via `options`) og man skal vide at designmatricen til en ordnet faktor (dannet med `ordered`) er noget speciel.

Uge 6-D: Test af hypoteser

I opgave 11.3 bliver vi bedt om at opstille et test for "parallelitet" af de to linier. Det svarer til at teste modellen

```
> hypotese<-lm(Blodtryk~Faggruppe+Alder-1)
```

mod

```
> model<-lm(Blodtryk~Faggruppe+Alder:Faggruppe-1)
```

Opgave a) Man kunne naturligvis pille variansestimater og frihedsgrader ud og selv konstruere sit test men det er naturligvis implementeret i `R`:

```
> anova(hypotese,model)
```

Alternativt kan man kalde `anova` på en passende parametrisering af `model`

```
> anova(lm(Blodtryk~Faggruppe*Alder))
```

og derved få (læst nedefra) test først af vekselvirkningen (H_0 : *linierne er parallelle*), dernæst test af hældning (H_0 : *linierne er sammenfaldende*) (som kun giver mening hvis der ikke er nogen vekselvirkning) og til sidst test af om der er en effekt af `Faggruppe` (H_0 : *linien er vandret*) (som for at give mening kræver at testet af ingen vekselvirkning og testet af ingen hældning begge accepteres). Bemærk dog at disse successive test ikke opdaterer variansestimatet mellem hvert test sådan som vi mener de burde; forskellen er dog sjældent betydelig. Hvis man hellere vil teste skæring (dvs effekt af `Faggruppe`) før hælding skal man bytte om på `Alder` og `Faggruppe` i modelformlen: Faktorernes orden er ikke ligegyldig! ○

Opgave b) Hvad er forskellen på de to test i den ensidede variansanalyse:

```
> anova(lm(Blodtryk~Faggruppe))
> anova(lm(Blodtryk~Faggruppe-1))
```

Hvilket test er du interesseret i? ○

Med det rette valg af parametrisering kan mange test formuleres som test af om en parameter er lig 0.

Opgave c) Lad nu

```
> model<-lm(Blodtryk~Faggruppe*Alder)
```

og kald `summary` på `model`. Eftersom parameteren `Faggruppe:Alder` er forskellen i hældning på de to linier, så er t-testet for denne parameter det samme test som det vi har lavet overfor vha `anova` i spørgsmål a. Se efter. ○

Advarsel: t-testet svarende til parameteren `Faggruppe` er et test af om linierne skærer hinanden i `Alder`=0, altså et test af om de to faggrupper har samme blodtryk ved fødslen. Det burde de nok have men der er absolut ingen grund til at tro at vores lineære model er en korrekt beskrivelse af blodtryks afhængighed af alder og job så langt før vi har nogen data! Testet er altså meningsløst.

Opgave d) Overvej de to andre t-test du får ud af `summary(model)`. Hvad er de test af og hvorfor er de meningsløse? ○

Omend meningsløse, så kan disse test bruge til at opstille konfidensintervaller med for de forskellige parametre. Det er dog nemmere at lade `R` om den slags.

Opgave e) Find konfidensintervaller med funktionen `confint`. ○

Uge 6-E: Model kontrol

Modelkontrol i den generelle linære model udføres med Bartletts test og diverse plots.

Opgave a) Hvis vi vil lave ensidet variansanalyse er det oplagt at teste om variansen er den samme i de forskellige grupper som faktoren angiver. Med to grupper giver

```
> var.test(Blodtryk~Faggruppe)
```

Bartletts test i form af forholdet mellem de to variansestimater. Bemærk at dette er en anden måde at anvende funktionen `var.test` på. Med mere end to grupper benyttes

```
> bartlett.test(Blodtryk~Faggruppe)
```

som giver et approksimativt (men ganske godt) test. ○

Af relevante plots kan nævnes plot af standardiserede residualer mod fittede værdier, mod baggrundsvariable, fraktildiagrammer af standardiserede residualer, plot af hat-værdier m.fl.

Opgave b) Start med at fitte og gemme modellen:

```
> model<-lm(Blodtryk~Alder*Faggruppe)
```

Fittede værdier findes med `fitted(model)` mens "rå" residualer fås med `resid(model)`. Vi vil dog altid bruge enten standardiserede residualer, som fås ved `rstandard(model)` (hvis vi ikke vil dividere med spredningsestimatet kan vi angive `sd=1` som argument efter `model`, eller studentiserede (deletion) residualer, som fås med `rstudent(model)`. Lav de ovenfor nævnte plots af residualer med fittede værdier og mod baggrundsvariable; hvor muligt/relevant kan du benytte forskellige plottesymboler for hver `Faggruppe`. ○

Opgave c) Residualplots skal laves mod normalfordelingen for standardiserede residualer og mod en passende t-fordeling for studentiserede residualer. Fraktildiagrammer bør *altid* tilføjes en linie så man kan se om der er nogen krumning i plottet. En funktion `resid.qq` (se afsnit Uge 6-I:) til konstruktion af fraktildiagrammer for studentiserede residualer findes på kursushjemmesiden i filen `uge6.r`. Lav fraktildiagrammer for både standardiserede og studentiserede residualer. ○

Opgave d) Hat-værdier fås med `hatvalues(model)`. Typisk plottes hat-værdier bare op:

```
> plot(hatvalues(model),hype='h')
```

En linie, der angiver gennemsnitsværdien fås med `abline(h=mean(hatvalues(model)))`. Plot også hatværdierne mod `Alder` og med forskellig farve for forskellig `Faggruppe`. Her er det nok nyttigt ikke at benytte `type='h'` fordi der er flere personer (fra hver `Faggruppe` med samme værdi af `Alder`. Evt kan man tillægge `Alder` en lille smule støj så værdierne skilles ad ved at erstatte `Alder` med `jitter(Alder)`. ○

Uge 6-F: Generelle linære modeller

Mere generelle modeller fittes let med `lm`. Det centrale er her at være i stand til at specificere middelværdiunderrummet i form af en modelformel.

Som et eksempel kan vi betragte to-sidet varians analyse. For `dat.11.3` kunne det være relevant at kategorisere alderen i stedet for at lave lineær regression f.eks. hvis man ikke tør tro på at effekten af alderen er lineær.

Opgave a) Vi vil inddele alderen i 10-årsgrupper. Dette kan gøre let ved

```
> grp.alder<-as.factor(round(Alder/10))
> summary(grp.alder)
```

Bemærk hvordan vi med `as.factor` sikrer at `R` vil betragte `grp.alder` som en faktor så vi ikke risikerer at lave lineær regression på en kategorisk variabel: Det er en klassisk fejl at lave lineær regression på ting som præparatnumre. Mere tilfredsstillende er måske:

```
> grp.alder<-factor(apply(matrix(round(Alder/10)),1,function(x){
+  switch(x-2,'<50','<50','50-59','60-69','70+')}),
+  levels=c('<50','<50','50-59','60-69','70+'))
> summary(grp.alder)
```

Bemærk hvordan vi i første linie har sørget for lidt færre kategorier end ovenfor[2]. Resultatet er nu en faktor med fornuftige navne. `R` vil altid "ordne" faktorer "alfabetisk" efter navnene på niveauerne; for at sikre en fornuftig ordning tvinger vi niveauerne ind i en bestemt rækkefølge med argumentet `levels=c('<50','<50','50-59','60-69','70+')`; det er overflødigt i dette eksempel, hvor ordningen af sig selv bliver rigtig.○

Opgave b) Hvad er resultatet af

```
> summary(factor(grp.alder,levels=levels(grp.alder)[4:1]))
```

○

Opgave c)* Fit en to-sidet variansanlyse model af `Blodtryk` på `Faggruppe` og `grp.alder` dels med en parametrisering som giver gruppegennemsnittene som estimater og dels med en parametrisering for hvilken `anova` giver først test af vekselvirkningen, dernæst test af `Faggruppe` og til sidst test af `grp.alder`. ○

Opgave d) Kontroller din model (Bartletts test, residualplot,...). ○

Kategorisering af en kontinuert baggrundsvariabel såsom `Alder` kan benyttes i forbindelse med modelkontrol.

[2] Vi vælger at gøre det fordi mindste kategori kun indeholder en person.

Opgave e) Fit en model med effekt af både `grp.alder` og `Alder` (gerne med vekselvirkning med `Faggruppe` hvis muligt). Undersøg om du kan teste `grp.alder` væk. Hvad siger resultaterne om hvor god modellen med to regressionslinier beskriver data? ○

Uge 6-G: Objektorientering

Som vi tidligere har bemærket er der en lang række af `R`s funktioner, som opfører sig forskelligt afhængigt af typen af argument vi benytter. Et eksempel er `summary`-funktionen. Kigger vi på selve funktionen ser vi følgende:

```
> summary
function (object, ...)
UseMethod("summary")
<environment: namespace:base>
```

Funktionen gør tilsyneladende ikke andet end at kalde en anden funktion, nemlig `UseMethod` (andre "generiske" funktioner kalder i steder `NextMethod`). Det denne funktion gør er at sende `object` og evt andre argumenter (repræsenteret ved `...`) videre til en funktion med navn `summary.klasse` hvor klasse selvfølgelig skal erstattes af `object`ets klasse. Vores kald af `summary(model)` sendes altså til `summary.lm(model)`. Vi vil kalde en funktion som `summary` en *generisk funktion* mens `summary.lm` vil blive omtalt som en *metode*.

Opgave a) Se efter at `summary.lm` er en "rigtig" funktion som gør en hel masse (men forsøg ikke at forstå præcis hvad; det er ikke umuligt men vil tage tid). Undersøg resultatets klasse (`class`) og type (`mode`). ○

Man kan undersøge hvilke metoder en generisk funktion eller hvilke generiske funktioner der har en metode svarende til en given type objekt med funktionen `methods`.

Opgave b) Undersøg hvilke metoder, der findes til `summary` med `methods(summary)`, og hvilke generiske funktioner der har en `lm`-metode med `methods(class=lm)`. ○

En del af disse funktioner er "'stjernet". Et eksempel er `vcoc.lm`. En sådan funktion kan ikke umiddelbart ses:

```
> vcov.lm
Error: Object "vcov.lm" not found
```

Skal man se en sådan funktion skal man enten kende/gætte hvilket "`namespace`" den hører til i[3] eller benytte funktionen `getAnywhere`.

Opgave c) Kig på `vcov.lm` ved kommandoen `getAnywhere(vcov.lm)` eller ved at gætte dens `namespace`, som er `stats`, og så benytte kommandoen

```
> stats:::vcov.lm
```
○

Som tidligere bemærket er output af `summary(model)` ikke umiddelbart det vi ville

[3] Dvs hvilken del/pakke af `R` den hører til i

tro når det nu bare er en speciel slags liste. Dette skyldes at R ikke bare giver resultatet af et funktionskald men i stedet printer det. For eksempel, hvis vi giver kommandoerne

```
> a<-1:4
> a
```

så giver R os i sidste ende ikke a men print(a). Naturligvis er print en generisk funktion.

Opgave d) Undersøg hvilke metoder der er til print og kig[4] på print.summary.lm .○

Uge 6-H: Hvad har du lært?

I dette kapitel skal du have lært

- hvordan du fitter generelle lineære modeller i R
- hvordan benytter modelformler til at skifte/vælge parametriseringer
- hvordan du tester hypoteser i generelle lineære modeller
- mere om Rs objektorientering

Uge 6-I: Appendix

resid.qq funktionen ser således ud:

```
resid.qq<-function(model,...){
  if (class(model)!='lm') stop('Argument must be a linear model')
  res<-rstudent(model)
  df<-model$df.residual
  qqt(res,df,...)
}

qqt<-function(y,df,line=T,xlab="Theoretical Quantiles",
 ylab="Sample Quantiles",main = paste("t(",df,") Q-Q Plot",sep='',col=''),
 ...){
  y<-sort(y)
  x<-qt(ppoints(y),df=df)
  plot(x,y,xlab=xlab,ylab=ylab,main=main,...)
  y <- quantile(y[!is.na(y)], c(0.25, 0.75))
  x <- qt(c(0.25, 0.75),df=df)
  if (line){
    slope <- diff(y)/diff(x)
    int <- y[1] - slope * x[1]
    }
  abline(int, slope, ...)
}
```

[4]Her benyttes naturligvis en metode, tilsyneladende print.default.

Den består altså af en funktion, som først ser efter at argumentet `model` er af klasse `lm` (og stopper med en fejlmeddelelse, hvis det ikke er tilfældet), hiver studentiserede residualer og frihedgrader ud af modellen, hvorefter den sender disse videre til en funktion som laver plottet. Fordelen ved at gøre dette er at man så kan fornøje sig med plot-funktionen `qqt` ved andre festlige lejligheder. Det andet argument `...` er en fiks `R`-konstruktion, som tillader at man specificerer andre argumenter (f.eks. `col='blue'` eller `xlab='Teoretiske t-fraktiler'`) som så sendes videre til plot-funktionen.

Plot-funktionen `qqt` laver bare fraktildiagrammet og sætter en linie ind hvis ikke man beder den lade være. `qqt` kunne med fordel forbedres så den var mere lig `qqnorm`, dvs inkluderede et tjek af at ikke alle observationer mangler, en mulighed for at bytte om på akserne (`datax`) og en mulighed for slet ikke at plotte (`plot.it`).