MATEMATIK OG DATABEHANDLING

Noter om R

og

Noter om Regneark

Morten Larsen og Peter Sestoft

Institut for Grundvidenskab
Den Kongelige Veterinær- og Landbohøjskole, KVL

Version 2.0.1 af 2006-09-20

Indhold

Tak til Bo Martin Bibby, Claus Ekstrøm, Henrik Laurberg Pedersen, Jacob Engelbrecht. Mogens Flensted-Jensen og Thomas Vils Pedersen for kommentarer og forslag, og selvfølgelig tak til de mange der har bidraget til udviklingen af R-systemet og OpenOffice.

MATEMATIK OG DATABEHANDLING

Modul A, B og C

Noter om R

Morten Larsen og Peter Sestoft

Institut for Grundvidenskab
Den Kongelige Veterinær- og Landbohøjskole, KVL

Disse noter beskriver programsystemet R som det benyttes i kurset Matematik og Databehandling på KVL. Programsystemet R er velegnet til matematiske og statistiske beregninger, graftegning, og behandling af forsøgsdata. Du kan lovligt og gratis installere en kopi af R på din egen pc; systemet er open source software. Eksemplerne i disse noter benytter Windows-udgaven, men R fås også til MacOS og til Linux.

Det bedste udbytte af disse noter fås hvis du selv eksperimenterer med de præsenterede funktioner i R under læsningen. Alle eksempelindtastninger fra noterne samt de benyttede eksempelfiler findes her:

`http://www.matfys.kvl.dk/mat-dat/eksempler`

1 Effektiv brug af R

1.1 Første gang du starter R

- Opret en mappe kaldet `MatDat` specielt til dette kursus. Læg den fx på Skrivebordet (hvis du bruger din egen pc) eller på dit netværksdrev (hvis du bruger en af KVL's pcer). En ny mappe oprettes ved at højreklikke på Skrivebordet og vælge NY || MAPPE.
- Start R, der normalt ligger som en stor ikon R på skrivebordet, vælg FILE || SAVE WORKSPACE, navigér til din nye mappe `MatDat` og gem et workspace kaldet `matdat.RData`. Luk R og svar Nej til Save workspace.
- Nu er der en ny R-ikon kaldet `matdat` i din `MatDat`-mappe. *For fremtiden skal du dobbeltklikke denne ikon for at starte R.*
- Svar altid Nej til Save workspace når du lukker R.

Når R gemmer opgaveløsninger (afsnit 1.5) på disken eller læser datafiler (afsnit 10.1) fra disken, så bruges den *aktuelle filmappe*. Ved at bruge fremgangsmåden ovenfor bliver mappen `MatDat` altid den aktuelle filmappe, hvad der er meget praktisk. Ellers er du nødt til at udpege den aktuelle filmappe — ved at vælge FILE || CHANGE DIR i RGui — hver eneste gang du starter R.

1.2 Vinduerne i R

Når man arbejder med R under Microsoft Windows har man typisk flere vinduer, som det ses i figur 1:

- RGui er det store ydre vindue. Menuen i RGui afhænger af hvilket indre vindue der ligger øverst, dvs. hvilket vindue der er aktivt.
- R Console er et indre vindue hvor man kan skrive regneudtryk der skal udregnes; se afsnit 1.3.
- R Graphics er et indre vindue der viser det seneste plot; se afsnit 1.4.
- R Editor er et indre vindue hvor man kan redigere definitioner; se afsnit 1.5.
- R Help vinduer (ikke vist) dukker op hvis man har brugt hjælpefunktionen i R; se afsnit 1.6.

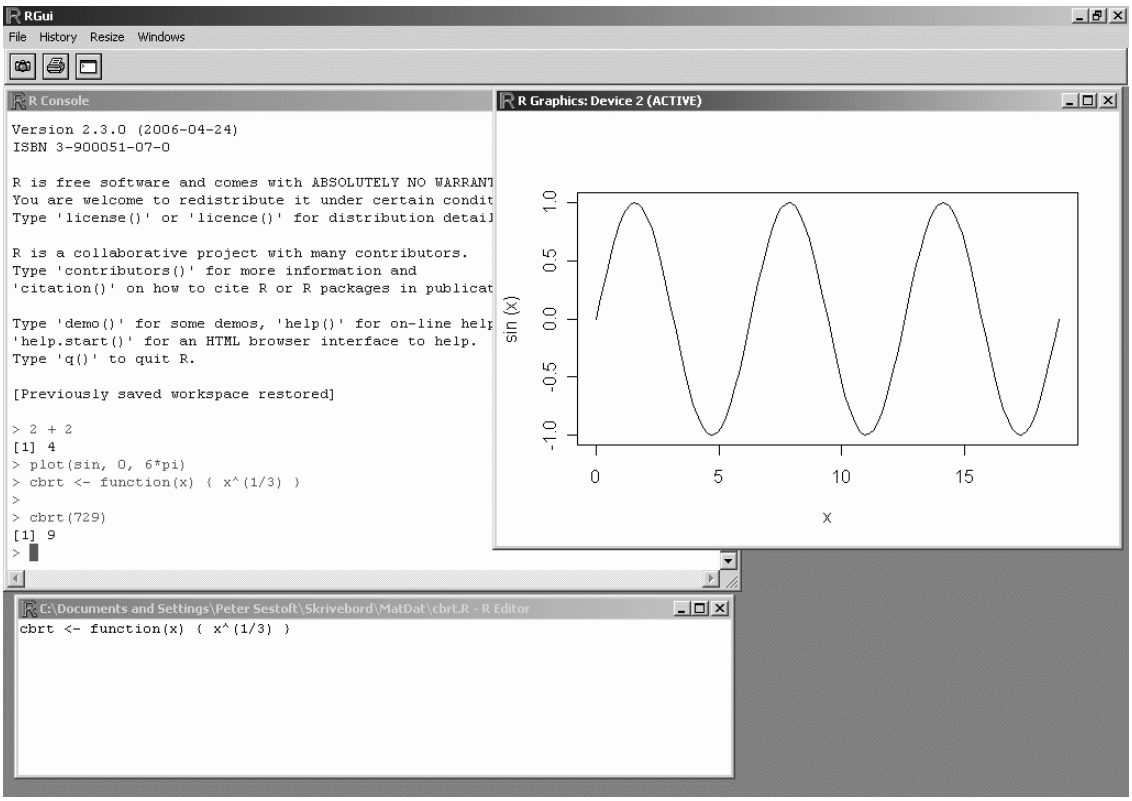
1.3 R Console vinduet

I R Console vinduet kan du skrive regneudtryk og definitioner. For at udregne `2 + 2` skal du skrive udtrykket efter prompten "`>`" og tryk på Enter-tasten:

```
> 2 + 2
[1] 4
```

Svaret fra R kommer på den næste linie. Resultatet af beregningen er naturligvis `4`, og klammen "`[1]`" betyder at `4` er det første tal i resultatet. Senere skal vi se at et svar fra R kan bestå af flere tal.

Du kan bladre i de seneste indtastningslinier ved hjælp af pil-op og pil-ned tasterne, rette i et udtryk og igen trykke Enter for at udregne det. Det er ikke smart at skrive lange definitioner direkte i R Console; brug i stedet R Editor som forklaret i afsnit 1.5.



Figur 1: RGui med et R Console vindue, et R Graphics vindue, og et R Editor vindue.

Hvis du kommer til at trykke Enter før udtrykket er færdigt vil R lave en "+" prompt på næste linie. Det angiver at R venter på resten af udtrykket:

```
> 5 -
+
```

Her ville vi have udregnet `5 - 3` men kom til at trykke Enter allerede efter minus. Da vi ikke har tastet et afsluttet udtryk kommer R med "+" prompten. Så kan vi indtaste det manglende tretal og trykke Enter, hvorefter R kommer med resultatet:

```
> 5 -
+ 3
[1] 2
```

I disse noter vil enkelte lange indtastninger være delt over flere linier på denne måde.

I eksemplet `5 - 3` var det nemt at gennemskue hvad det var vi manglede at indtaste og fejlen kunne udbedres ved blot at indtaste det manglende. Nogle gange kan man i et langt indviklet udtryk komme til at glemme en slutparentes eller noget andet og så få en uventet "+"-prompt. Desværre kan man ikke fra "+"-prompten gå direkte op på den foregående linie og rette fejlen. I disse tilfælde må man afbryde indtastningen med `Esc` (linux: `Ctrl+C`) og så kalde den fejlagtige linie frem igen med pil-op tasten og rette i den.

1.4 R Graphics vinduet

Du kan plotte funktioner og data ved at skrive kommandoer i R Console; den resulterende graf vises i R Graphics vinduet. For eksempel kan du plotte $\sin(x)$ for x fra 0 til 6π ved at skrive i R Console:

```
> plot(sin, 0, 6*pi)
```

Normalt vil R Graphics vinduet kun vise det seneste plot. Du kan få vinduet til at huske alle plot ved først at vælge HISTORY || RECORDING i RGui mens R Graphics vinduet ligger øverst. Du kan så bladre frem og tilbage gennem dine plots med tasterne PageUp og PageDown.

Du kan gemme det plot du ser ved at højreklikke på det og vælge COPY AS METAFILE og derefter indsætte i Word, OpenOffice, eller PowerPoint; se afsnit 6.1.

1.5 R Editor vinduet

Når du arbejder på en opgaveløsning skal du skrive definitionerne i R Editor, så du kan rette, udføre og gemme dem. Du kan nemlig ikke gemme indtastningerne du laver direkte i R Console.

- Du åbner en R Editor ved at vælge FILE || NEW SCRIPT eller FILE || OPEN SCRIPT i RGui.
- Skriv dine definitioner i R Editor. I eksemplet vist i figur 1 er der i R Editor defineret en funktion `cbrt` til at udregne kubikroden af `x`.
- Du kan udføre definitionerne ved at markere dem, højreklikke, og vælge RUN LINE OR SELECTION; så bliver de udført i R Console.

  I R Console kan man så bruge de definerede funktioner i regneudtryk, fx udregne `cbrt(729)`.
- I stedet for at højreklikke for at udføre markeret tekst fra R Editor kan man taste `Ctrl+R`.

  Det er faktisk oftest nemmest i R Editor slet ikke at markere de linier der skal udføres, men simpelthen flytte markøren til den første af linierne og så taste `Ctrl+R` på hver linie. Markøren rykker automatisk en linie ned hver gang du taster `Ctrl+R`. Hermed får du udført dine definitioner i R Console én efter én og det er nemt at finde ud af hvor der eventuelt opstår fejl.
- Du kan rette dine definitioner i R Editor og udføre dem så ofte du vil.
- **Husk** at gemme indholdet af R Editor med FILE || SAVE og vælg et passende filnavn, fx `Dat-A-1.R`. Du kan kun gemme indholdet når R Editor vinduet ligger øverst (og dermed er det aktive vindue).
- En gemt R-fil kan senere åbnes ved at vælge FILE || OPEN SCRIPT i RGui.

1.6 R Help vinduer

Hjælp til brugen af R fås under menupunktet `Help` i RGui, når R Console ligger øverst. Man kan få hjælp til en bestemt funktion ved at vælge HELP || R FUNCTIONS og skrive funktionsnavnet, fx `log`. Der dukker så et nyt R Help vindue op med en beskrivelse af funktionen og nogle eksempler, men forklaringerne kan være ret uforståelige. Det vil for funktioner der bruges i *Matematik og Databehandling* generelt være langt bedre at slå dem op i disse noter, fx ved hjælp af det omfattende alfabetiske indeks sidst i noterne.

Hvis man ikke kan huske navnet på en funktion eller en regneoperator kan man være heldig at finde den med HELP || SEARCH HELP eller HELP || APROPOS. Ofte finder man dog blot et antal meget specielle statistiske funktioner på denne måde.

Menuen HELP giver også adgang til en kort vejledning til R Console, og til R's manualer.

En *pakke* er en samling af R-funktioner til et bestemt formål. Man kan få en liste af allerede installerede pakker ved at vælge PACKAGES || LOAD PACKAGE. For at få en liste af pakker tilgængelige fra internettet skal man først vælge hvor de skal komme fra; se appendiks A.2.

2 Regneudtryk og indbyggede funktioner

Der er mange indbyggede regneoperatorer og funktioner i R, for det meste med de velkendte navne, men notationen i R er ofte lidt anderledes end i matematik. Funktioner og operatorer kan bruges i regneudtryk:

Regneudtryk i R	Resultat	Matematik	Betydning
`5 + 7 * 8`	`61`	$5+7\cdot 8$	Fem plus syv gange otte
`sqrt(10)`	`3.162278`	$\sqrt{10}$	Kvadratroden af 10
`10^3`	`1000`	10^3	10 opløftet i tredje
`log10(17)`	`1.230449`	$\log(17)$	Titalslogaritme af 17
`exp(3)`	`20.08554`	e^3	Eksponentialfunktionen af 3
`log(17)`	`2.833213`	$\ln(17)$	Naturlig logaritme af 17
`sin(3/2*pi)`	`-1`	$\sin(\frac{3}{2}\pi)$	Sinus af $\frac{3}{2}\pi$

For eksempel kan den tredje rod af 17 beregnes som $\sqrt[3]{17} = 17^{1/3}$ hvad der i R skrives sådan her:

```
> 17^(1/3)
[1] 2.571282
```

Bemærk at hat-symbolet (^) er en såkaldt død tast på nogle computere, så man skal trykke hat (^) efterfulgt af mellemrum for at få tegnet vist på skærmen.

2.1 Visning af tal, og intern regnenøjagtighed

Som det fremgår af eksemplet ovenfor viser R normalt kun 7 betydende cifre i resultater. Dette tal kan sættes op til 12 på denne måde:

```
> options(digits=12)
> 17^(1/3)
[1] 2.57128159066
```

Der er ikke nogen mening i at bede R om at vise mere end 16 betydende cifre, for moderne computere regner kun med 15–16 betydende cifre internt. Det gør pcer og R også, uanset hvor mange eller få cifre der vises i resultaterne. Visningen af resultater sættes tilbage til det normale sådan her:

```
> options(digits=7)
```

2.2 Resultater der ikke er tal

Nogle regneudtryk har ikke nogen fornuftig talværdi: hvad skulle resultatet af `1/0` eller `log(0)` eller `0/0` være? Disse udtryk giver nogle specielle resultater, nemlig `Inf`, der betyder uendelig; og `-Inf`, der betyder minus uendelig; og `NaN`, der betyder "Not a Number". Hvis `NaN` dukker op i resultatet af en beregning, så er der som regel gået noget galt og man må hellere tjekke sine regneudtryk.

Derimod betegner den specielle værdi `NA`, der betyder "not available" eller "not applicable", en manglende observation i et datasæt, fx en manglende aflæsning fra et forsøg.

Funktion i R	Matematik	Betydning
`x + y`	$x+y$	`x` plus `y`
`x - y`	$x-y$	`x` minus `y`
`x * y`	$x \cdot y$	`x` gange `y`
`x / y`	x/y	`x` divideret med `y`
`x %/% y`		`x` heltalsdivideret med `y`; fx vil `7%/%3` give `2`
`x %% y`	$x \bmod y$	`x` modulo `y`, rest ved heltalsdivision; fx vil `7%%3` give `1`
`x ^ y`	x^y	`x` opløftet i `y`
`abs(x)`	$\lvert x \rvert$	numerisk (absolut) værdi af `x`
`sqrt(x)`	$\sqrt{x}$	kvadratrod af `x`
`log(x)`	$\ln(x)$	naturlig logaritme af `x`
`log10(x)`	$\log(x)$	titalslogaritme af `x`
`exp(x)`	e^x	eksponentialfunktionen af `x`, dvs. e^x
`sin(x)`	$\sin(x)$	sinus til `x` radianer
`cos(x)`	$\cos(x)$	cosinus til `x` radianer
`tan(x)`	$\tan(x)$	tangens til `x` radianer
`asin(x)`	$\sin^{-1}(x)$	arcus sinus til `x`
`acos(x)`	$\cos^{-1}(x)$	arcus cosinus til `x`
`atan(x)`	$\tan^{-1}(x)$	arcus tangens til `x`
`floor(x)`	$\lfloor x \rfloor$	x rundet *ned* til nærmeste heltal
`ceiling(x)`	$\lceil x \rceil$	x rundet *op* til nærmeste heltal
`round(x)`	$[x]$	x afrundet til nærmeste heltal; halve rundes til lige tal
`pi`	π	enhedscirklens areal $\pi = 3.14159\ldots$
`exp(1)`	e	den naturlige logaritmes grundtal $e = 2.718282\ldots$

Figur 2: Numeriske operatorer, funktioner og konstanter i R.

3 Variable

Man kan med en *tildeling* binde resultatet af et regneudtryk til en variabel, for eksempel `z`. Pilen (`<-`) betyder at udtrykket på højresiden udregnes og værdien gemmes i variablen `z`:

```
> z <- 17^(1/3)
```

Resultatet af beregningen bliver ikke vist, men lagret i variablen `z`. Man kan få oplyst en variabels værdi blot ved at skrive dens navn:

```
> z
[1] 2.571282
```

Variable kan bruges i efterfølgende beregninger. Det er nyttigt fordi man undgår at skrive det samme komplicerede regneudtryk flere gange:

```
> z^6 - z^3
[1] 272
```

Et variabelnavn kan indeholde punktummer, så `antal.planter` er et lovligt variabelnavn. I R anses store og små bogstaver for at være forskellige, så `z` og `Z` er to forskellige variable. Undgå at bruge navnene `c`, `F`, `t` og `T`, da de allerede bruges af R til forskellige formål. Se også afsnit 19 om programmering i R.

Værdien af en variabel kan ændres ved en ny tildeling:

```
> z <- 5
> z
[1] 5
> z <- z + 3
> z
[1] 8
```

Det er muligt med funktionen `ls` at få vist en liste over hvilke variable, man har defineret:

```
> ls()
[1] "z"
```

Endelig kan man en sjælden gang have brug for helt at fjerne en variabel igen. Dette gøres med funktionen `rm`:

```
> rm(z)
> z
Error: object "z" not found
```

4 Funktioner

Man kan nemt definere sine egne funktioner i R. For eksempel kan man definere en funktion `cbrt(x)` til at finde tredje rod ("cubic root") af et vilkårligt positivt tal `x` sådan her:

```
> cbrt <- function(x) { x^(1/3) }
```

Variablen `x` kaldes en *formel parameter* for funktionen `cbrt`, og krølle-parentesen `{ x^(1/3) }` er funktionens *krop*. Udtrykket `x^(1/3)` beregner $\sqrt[3]{x}$. Den nye funktion bruges som alle andre funktioner ved at skrive funktionsnavnet efterfulgt af en argumentparentes:

```
> cbrt(1000)
[1] 10
```

Værdien af dette udtryk udregnes ved at sætte `x` lig 1000 og så udregne funktionskroppen `x^(1/3)`, hvilket giver 10.

Funktioner med flere formelle parametre defineres på samme måde. Her er en funktion `root(x, n)` til at beregne den `n`'te rod af `x`:

```
> root <- function(x, n) { x^(1/n) }
```

Funktioner med flere parametre kaldes på sædvanlig vis ved at skrive argumentudtrykkene i parentes efter funktionsnavnet:

```
> root(625, 4)
[1] 5
```

Som et yderligere eksempel kan den såkaldte logistiske funktion:

$$logis(t) = \frac{K}{1 + ae^{-rt}}$$

defineres således i R:

```
> logis <- function(t) { K / (1 + a*exp(-r*t)) }
```

Bemærk at der ikke er givet nogen konkrete værdier for `a`, `r` og `K`. Man kan derfor ikke bruge funktionen til noget før `a`, `r` og `K` er blevet defineret:

```
> logis(5)
Error in logis(5) : Object "K" not found
```

Når `a`, `r` og `K` er blevet defineret kan man kalde funktionen for at beregne fx `logis(5)`, eller plotte funktionen. Hvis man ændrer `a`, `r` eller `K`, så ændres funktionen:

```
> a <- 50
> r <- 1
> K <- 1
> logis(5)
[1] 0.7480006
> K <- 10
> logis(5)
[1] 7.480006
```

5 Funktionsplot

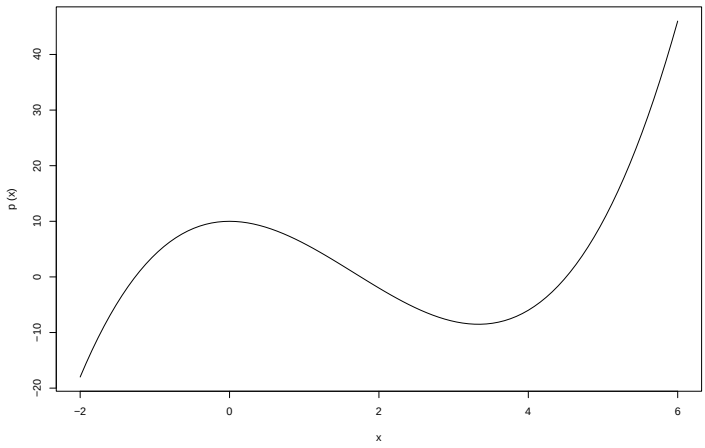
Man kan plotte indbyggede funktioner såsom $\sin(x)$ for x fra 0 til 6π ved brug af funktionen `plot`:

```
> plot(sin, 0, 6*pi)
```

For at plotte mere indviklede udtryk, for eksempel polynomiet $p(x) = 10 - 5x^2 + x^3$ for x løbende fra -2 til 6, kan man definere en funktion `p` med en parameter `x` og med polynomiet som funktionskrop:

```
> p <- function(x) { 10 - 5*x^2 +  x^3 }
> plot(p, -2, 6)
```

Det resulterende plot ses i figur 3.



Figur 3: Plot af polynomiet $p(x) = 10 - 5x^2 + x^3$ fremstillet med `plot(p, -2, 6)`.

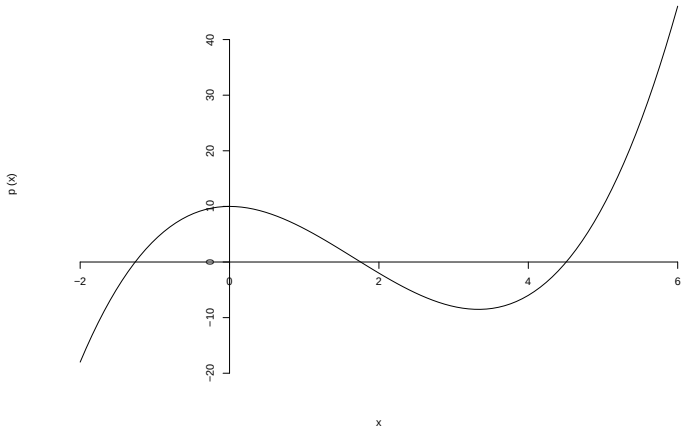
5.1 Akser i plot

Som det ses, sætter R gerne akserne som en kasse uden om selve funktionsgrafen. I matematik er det mere normalt at akserne skærer hinanden i $(0,0)$. For at opnå dette skal man i `plot`-funktionen angive `axes=FALSE` (med store bogstaver), og dernæst bruge funktionen `axis` til at angive at første-aksen skal gå gennem $y = 0$ og at anden-aksen skal gå gennem $x = 0$:

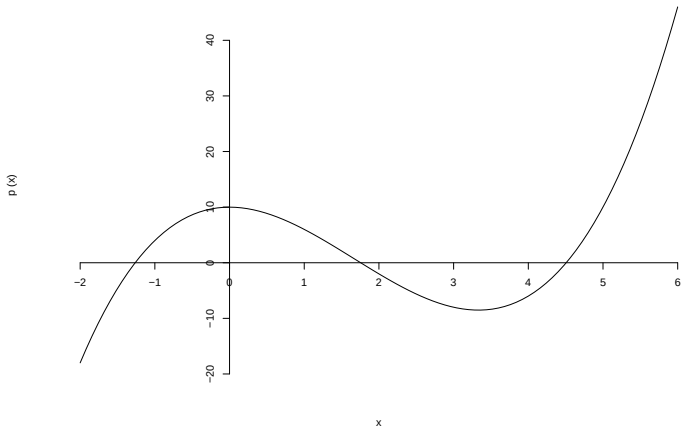
```
> plot(p, -2, 6, axes=FALSE)
> axis(1, pos=0)
> axis(2, pos=0)
```

Det resulterende plot ses i figur 4. Man kan også bestemme præcis hvilke aksemærker ("ticks") der skal sættes, med parameteren `at`. For eksempel kan man sætte mærker på x-aksen for hvert heltal fra -2 til 6 på denne måde:

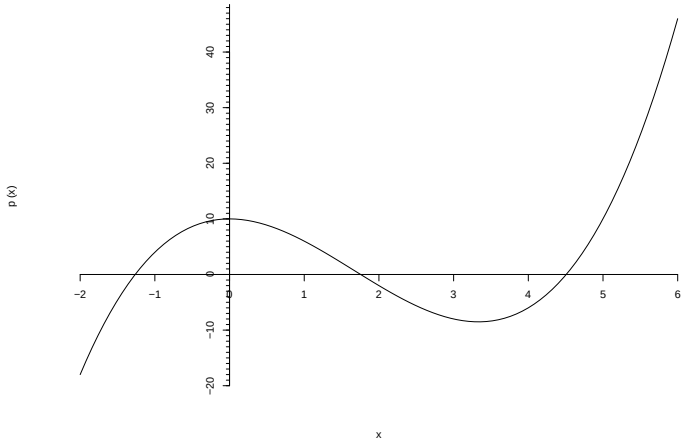
```
> plot(p, -2, 6, axes=FALSE)
> axis(1, at=seq(-2,6), pos=0)
> axis(2, pos=0)
```



Figur 4: Som figur 3, men med `axes=FALSE`, `axis(1, pos=0)` og `axis(2, pos=0)`.



Figur 5: Som figur 4, men med `axis(1, at=seq(-2,6), pos=0)`.



Figur 6: Som figur 5, men tilføjet `axis(2, pos=0, at=seq(-20,50), labels=FALSE, tcl=-0.25)`.

Det resulterende plot ses i figur 5. Funktionen `seq`, der her bruges til at generere talrækken -2, -1, . . . , 6, forklares i afsnit 14.1.

Man kan med `labels=FALSE` sætte aksemærker *uden* etiketter. Dette kan være nyttigt til med to kald af `axis` først at sætte mærker *med* etiketter og derefter sætte mærker i en finere inddeling *uden* etiketter. De sekundære mærker bør man så få tegnet med en lidt kortere længde, for eksempel:

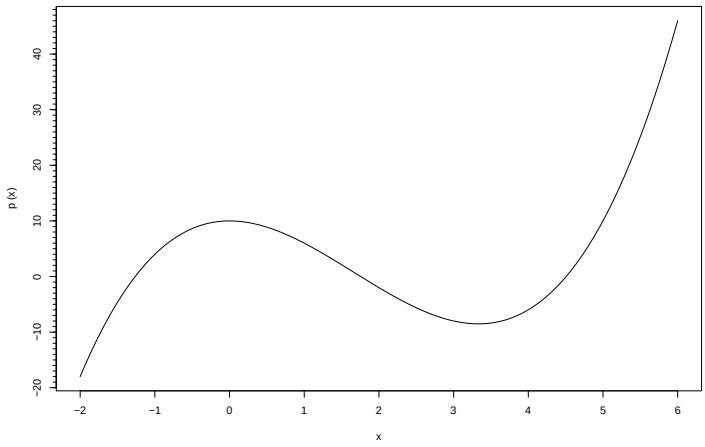
```
> plot(p, -2, 6, axes=FALSE)
> axis(1, at=seq(-2,6), pos=0)
> axis(2, pos=0)
> axis(2, pos=0, at=seq(-20,50), labels=FALSE, tcl=-0.25)
```

Det resulterende plot ses i figur 6. Parameteren `tcl` regulerer længden af stregerne, der tegnes som mærker; de normale mærker svarer til `tcl=-0.5` og med `tcl=-0.25` får vi altså halvdelen af den normale længde. Med positive værdier af `tcl` tegnes mærkerne på den anden side af akserne (altså modsat teksterne).

Hvis man udelader parameteren `pos` i `axis` vil aksen blive tegnet i siden af plottet, på samme position som hvis man havde ladet `plot` tegne den i første omgang. Hvis man gerne vil bibeholde den sædvanlige position af akserne som en kasse uden om plottet men selv vil regulere mærkerne kan man gøre som følger:

```
> plot(p, -2, 6, axes=FALSE)
> axis(1, at=seq(-2,6))
> axis(2)
> axis(2, at=seq(-20,50), labels=FALSE, tcl=-0.25)
> box()
```

Resultatet ses i figur 7. Den sidste funktion, `box`, tegner selve rammen. I stedet for at bruge `box` kunne man i selve kaldet af `plot` have angivet `frame.plot=TRUE` (sammen med `axes=FALSE`).



Figur 7: Som figur 6, men uden `pos=0` og til gengæld tilføjet `box()`.

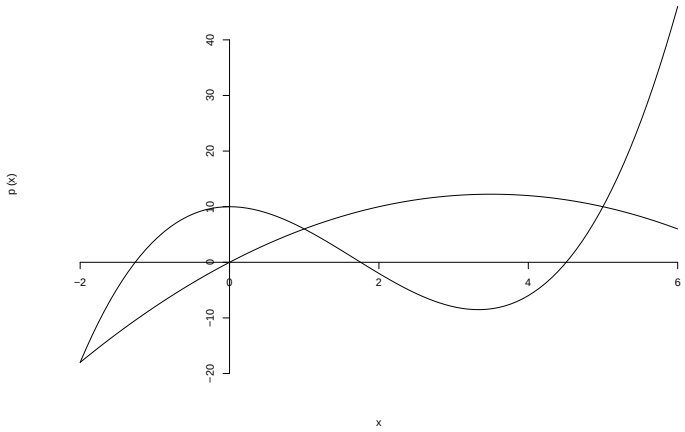
Der er flere andre parametre man kan bruge til at styre udseendet af mærker og akseetiketter; den vigtigste af disse er nok `las` som styrer hvilken led, akseetiketterne skrives på. Standard er `las=0`, som giver etiketter parallelt med akserne. Andre muligheder er `las=1`, hvilket giver vandrette etiketter, `las=2`, hvilket giver etiketter vinkelret på akserne, og `las=3`, hvilket giver lodrette etiketter. Parameteren `las` kan også anvendes direkte i `plot`, hvis man ikke angiver `axes=FALSE`.

5.2 Tilføjelse af funktionsgrafer til plot

Man kan tilføje flere funktionsgrafer til et eksisterende plot ved at bruge `add=TRUE` i `plot`-ordren.

```
> q <- function(x) { 7*x - x^2 }
> plot(q, -2, 6, add=TRUE)
```

Resultatet ses i figur 8. Man kan tilføje lige så mange funktionsgrafer man ønsker.



Figur 8: Som figur 4, men med `plot(q, -2, 6, add=TRUE)` tilføjet.

5.3 Aksegrænser i plot

Når man tilføjer en ny funktionsgraf til et eksisterende plot, bevares alle andre egenskaber ved plottet. Især bliver y-aksens grænser ikke tilpasset til den nye funktion, så man risikerer at en stor del af grafen falder uden for plottet.

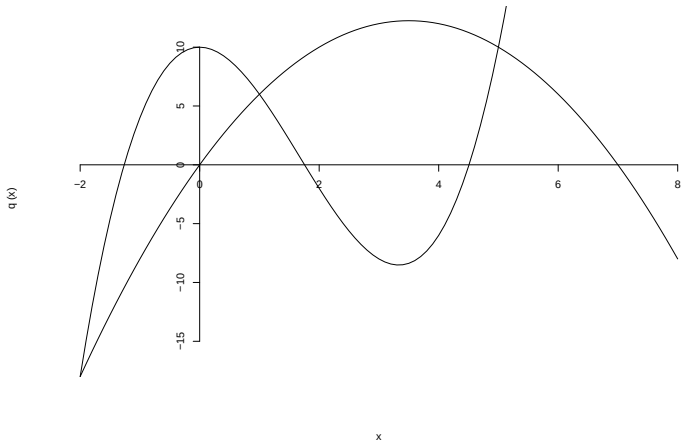
For eksempel, hvis man plotter de to funktioner `p` og `q` for $x \in [-2, 8]$ og plotter `q` først, så bliver y-aksen så kort at meget af grafen for `p` falder uden for plottet, jævnfør figur 9:

```
> plot(q, -2, 8)
> plot(p, -2, 8, add=TRUE)
```

I denne situation må man finde (fx gætte) største og mindste y-værdi for de to funktioner, og sætte passende aksegrænser for y-aksen allerede når den første funktion tegnes. Det gøres med parameteren `ylim`. For at få y-aksen til at gå fra cirka -16 til cirka 200 kan man gøre sådan her:

```
> plot(q, -2, 8, ylim=c(-16, 200))
> plot(p, -2, 8, add=TRUE)
```

Tilsvarende kan man sætte aksegrænser for x-aksen med parameteren `xlim`. Notationen `c(-16, 200)` binder de to tal sammen til en *vektor*, hvilket vi skal se flere eksempler på andre steder hvor vi skal angive en parameterværdi der består af mere end ét tal. Meget mere om vektorer følger i afsnit 14.



Figur 9: Som figur 8, men med `q` tegnet før `p` og y-aksen derfor for kort.

5.4 Farver i plot

Når man tegner flere funktionskurver i samme plot er det nyttigt at give kurverne hver sin farve. Det kan man gøre med parameteren `col`:

```
> q <- function(x) { 7*x - x^2 }
> plot(q, -2, 6, add=TRUE, col="red")
```

Figur 10 viser nogle få af standardfarverne i R.

`black`	`blue`
`brown`	`cyan`
`green`	`magenta`
`purple`	`red`
`white`	`yellow`

Figur 10: Navne på nogle standardfarver i R.

Tast `colors()` i R Console for at få en liste af de mere end 600 andre navngivne farver. Mange flere farver kan fremstilles med funktionen `rgb` der benyttes i afsnit 18.1. Farven på akser, tekster, signaturforklaringer og titler i et plot kan styres på lignende vis (se nedenfor).

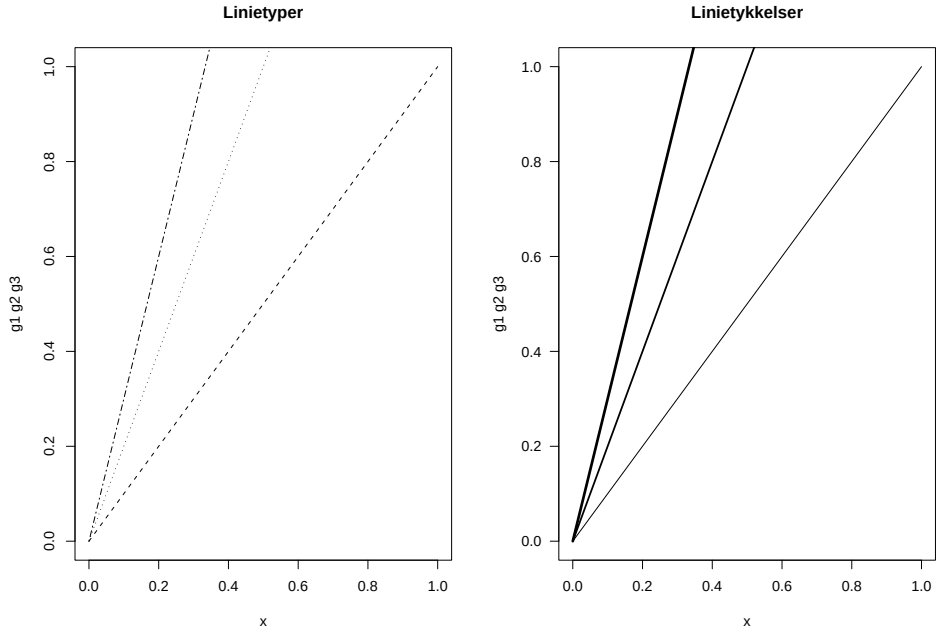
5.5 Linietype, linietykkelse, aksetitler i plot

Linietype, linietykkelse og aksetitler styres også med parametre til `plot`. Her plottes tre funktioner to gange for at demonstrere forskellige linietyper og linietykkelser:

```
> g1 <- function(x) { x }
> g2 <- function(x) { 2 * x }
> g3 <- function(x) { 3 * x }
> plot(g1, xlab="x", ylab="g1 g2 g3", main="Linietyper", lty="dashed", las=0)
> plot(g2, add=TRUE, lty="dotted")
> plot(g3, add=TRUE, lty="twodash")
> plot(g1, xlab="x", ylab="g1 g2 g3", main="Linietykkelser", lwd=1, las=1)
```

```
> plot(g2, add=TRUE, lwd=2)
> plot(g3, add=TRUE, lwd=3)
```

Eksemplet bruger `xlab` og `ylab` til at sætte aksetitler, og `main` til at sætte plottitler. De resulterende plot ses i figur 11.



Figur 11: Effekt af nogle grafikparametre. Til venstre ses fra neden linietyperne (`lty`) dashed, dotted og twodash; og til højre linietykkelserne (`lwd`) 1, 2 og 3. Parameteren `las` styrer om akseetiketterne skrives parallelt med aksen (`las=0`) eller vandret (`las=1`).

Det kan være mere bekvemt at tilføje titler til et plot efter at det er tegnet. Hertil bruger man funktionen `title`, der tager de samme parametre som `plot`. Parameteren (`main` sætter tekst for oven, `sub` sætter tekst for neden, `xlab` sætter tekst langs x-aksen, og `ylab` sætter tekst langs y-aksen. Desuden kan parametrene `col.main`, `col.sub` bruges til at bestemme farve på teksten. For eksempel:

```
> title(main="Lineære funktioner")
> title(sub="Normalt plot", col.sub="blue")
```

Talrige andre egenskaber ved funktionsplot styres med andre parametre til `plot`. Nogle få er vist i figur 12; flere kan findes i R's hjælp under `?par`, `?plot.default` og `?title`.

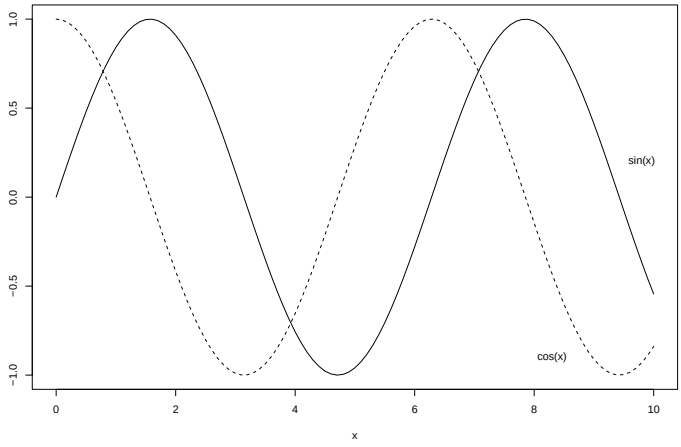
5.6 Tekst og signaturforklaring i plot

Funktionen `text` bruges til at tilføje en forklarende tekst til et eksisterende plot. Man angiver først (x, y)-koordinaterne for tekstens midtpunkt, og så den tekst der skal indsættes:

```
> plot(sin, 0,10, ylab="")
> plot(cos, 0,10, add=TRUE, lty="dashed")
> text(9.8, 0.2, "sin(x)")
> text(8.3, -0.9, "cos(x)")
```

Parameter	Effekt	Mulige værdier
`asp`	aspect ratio	Tal, fx `asp=1` for lige lange enheder på akserne eller `asp=2` for dobbelt så lange enheder på y-aksen som på x-aksen
`axes`	aksestyring	Brug `axes=FALSE` hvis du styrer akserne med `axis` fra afsnit 5.1
`col`	farve	Se figur 10, fx `col="red"`
`las`	akseetiketstil	`0` (parallel med akse), `1` (vandret), `2` (vinkelret på akse), `3` (lodret)
`log`	logaritmisk plot	`log="x"` (log til x), `log="y"` (log til y), `log="xy"` (dobbeltlog)
`lty`	linietype	`"solid"` `"dashed"` `"dotted"` `"dotdash"` `"longdash"` `"twodash"`
`lwd`	linietykkelse	1, 2, 3, . . . , hvor standard er 1
`main`	overtitel	Tekst, fx `main="Logistisk funktion"`
`sub`	undertitel	Tekst, fx `sub="Logistisk funktion"`
`xlab`	x-aksetekst	Tekst, fx `xlab="Tid i timer"`
`xlim`	x-aksegrænser	Interval, fx `xlim=c(-2, 6)`
`ylab`	y-aksetekst	Tekst, fx `ylab="Antal individer"`
`ylim`	y-aksegrænser	Interval, fx `ylim=c(-20, 40)`

Figur 12: Nogle grafikparametre til brug i funktionen `plot`.



Figur 13: Tilføjelse af tekster med funktionen `text(...)`.

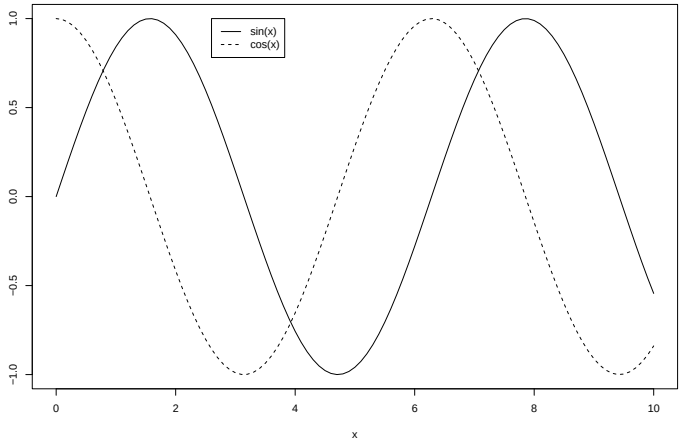
Resultatet ses i figur 13. Parameteren `col` kan bruges til at styre tekstens farve.

Funktionen `legend` bruges til at indsætte en boks med signaturforklaring i et eksisterende plot. Man skal angive (x, y)-koordinat for øverste venstre hjørne af signaturforklaringen. For eksempel kan vi plotte sinus med fuldt optrukken linie og cosinus med stiplet linie, og tilføje en forklaring:

```
> plot(sin, 0, 10, ylab="")
> plot(cos, 0, 10, add=TRUE, lty="dashed")
> legend(2.6, 1, c("sin(x)","cos(x)"), lty=c("solid","dashed"))
```

Resultatet ses i figur 14. R har ikke nogen erindring om hvad man har plottet og i hvilken rækkefølge, så der kan ikke genereres en automatisk signaturforklaring. Såvel teksterne som signaturerne er man derfor nødt til selv at angive og man må selv sørge for at signaturernes udseende rent faktisk kommer til at svare til det man har plottet. Det vil sige at type, bredde og farve for linierne er man nødt til at specificere ligesom man har gjort i de kald af `plot` der har tegnet graferne. Heldigvis hedder parametrene det samme i `legend` som i `plot`, så man bruger alså `lty` og `lwd` henholdsvis `col` til at angive type og bredde henholdsvis farve af linierne i signaturforklaringen. **Bemærk:** Man er nødt til at angive mindst én af parametrene `lty` eller `lwd` for overhovedet at få tegnet linier med `legend`; det er ikke tilstrækkeligt at angive `col`. Hvis man i kaldene af `plot` alene har brugt `col` og ingen af de andre to parametre (og kurverne dermed har forskellig farve men iøvrigt er "standardlinier") kan man så angive `lty` til `"solid"` i parametrene til `legend` for at få tegnet linier i signaturforklaringen.

Bemærk den specielle måde såvel signaturteksterne som parameteren `lty` angives på i eksemplet ovenfor. Da der er to signaturer skal der angives såvel to signaturtekster som to værdier af `lty`, og dette gøres i begge tilfælde ved som parameterværdi at angive en vektor med de to værdier pakket ind i `c(...)`. Den første tekst `"sin(x)"` hører til den første værdi af `lty`, altså `"solid"`, og tilsvarende hører den anden tekst til den anden værdi af `lty`, altså `"dashed"`. Hvis nu den første linie skulle være blå og den anden rød ville vi tilsvarende skulle angive `col=c("blue","red")`. Havde der været tre signaturer skulle der have været angivet tre værdier i hver af vektorerne `c(...)`. Vektorer `c(...)` omtales for alvor i afsnit 14.



Figur 14: Tilføjelse af signaturforklaring med funktionen `legend(...)`.

5.7 Aflæsning af punkter i plot

Med funktionen `locator` kan man aflæse (x, y)-koordinaterne til punkter i et plot. Dette er nyttigt når man skal finde koordinater til brug i `text` og `legend` funktionerne omtalt ovenfor.

Desuden kan det bruges til at lave omtrentlige aflæsninger af skæringspunkter mellem kurver og akser. For eksempel kan vi forsøge at aflæse (x, y)-koordinaterne for det første skæringspunkt mellem grafen og x-aksen i figur 4 på denne måde:

- Tegn plottet
- I R Console kalder man `locator` med argumentet 1, for at sige at man vil aflæse 1 punkt i plottet:

  ```
  > locator(1)
  ```

- Dernæst klikker man med musen i plottet der hvor kurven skærer x-aksen, og så svarer `locator`-funktionen i R Console med (x, y)-koordinaterne:

  ```
  $x
  [1] -1.270422
  $y
  [1] 0.06869091
  ```

  Som det ses er x omtrent lig -1.27, men det lykkedes åbenbart ikke at ramme x-aksen præcist, for så ville y have været 0.

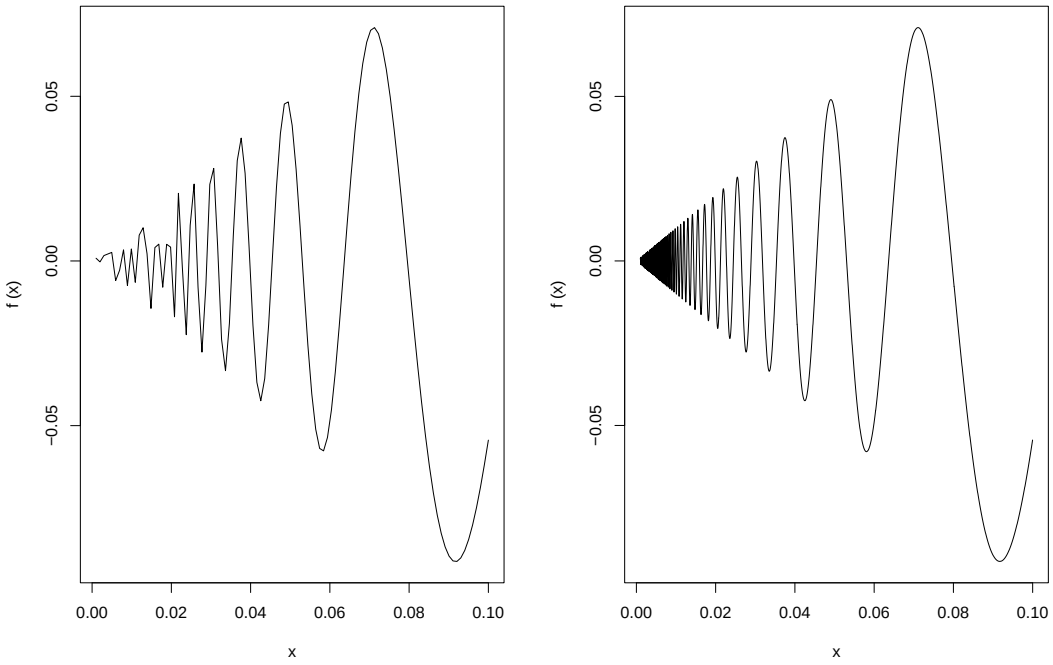
Hvis man vil aflæse alle tre skæringspunkter i én arbejdsgang, kalder man blot `locator(3)` og klikker på alle tre punkter, hvorefter funktionen returnerer en vektor af x-værdier med tre elementer, og en vektor af y-værdier med tre elementer.

5.8 Antal støttepunkter

Funktionen `plot` beregner normalt den givne funktions værdier i 100 støttepunkter. Hvis grafen for en funktion svinger meget i et lille interval kan det være nødvendigt at sætte antal punkter `n` til en højere værdi end 100. Prøv fx at tegne funktionen $f(x) = x\sin(\frac{1}{x})$, der svinger voldsomt når x er tæt på 0:

```
> f <- function(x) { x * sin(1/x) }
> plot(f, 0.001, 0.1)
> plot(f, 0.001, 0.1, n=5000)
```

De resulterende plot ses i figur 15.



Figur 15: Plot af $f(x) = x\sin(\frac{1}{x})$ for $x \in [0.001, 0.1]$, tegnet med 100 henholdsvis 5000 støttepunkter. Det venstre plot er klart forkert i nærheden af $x = 0$.

6 Eksport af plot fra R

6.1 Indsættelse af plot i rapporter og præsentationer

Under Windows kan man inkludere et plot (en graf) i en rapport eller præsentation ved at kopiere plottet i grafikformatet Windows (Enhanced) Metafile og derefter indsætte det i OpenOffice eller Microsoft Word eller PowerPoint:

- Lav plottet i R.
- Højreklik på plottet og vælg COPY AS METAFILE; eller peg på plottet og tryk `Ctrl+W`.
- Skift til OpenOffice eller Word eller PowerPoint, og indsæt figuren ved at vælge REDIGER || SÆT IND eller ved at trykke `Ctrl+V`.

Efter indsættelse kan plottet skaleres (formindskes eller forstørres) uden kvalitetsforringelse hvis det er overført som Windows Metafile. Hvis plottet er overført som bitmap, bliver det grimt ved skalering.

6.2 Indsættelse af plot på websider; grafikfiler

Hvis man skal lave mange plot, så er det praktisk at gemme hvert plot direkte i en separat grafikfil. At gemme et plot i filen `poly.emf` i formatet Windows Metafile gøres sådan her:

```
> p <- function(x) { 10 - 5*x^2 + x^3 }
> win.metafile("poly.emf")
> plot(p, -2, 8)
> dev.off()
```

Ordren `dev.off()` afslutter opbygning af plot-filen, og først derefter er filen `poly.emf` klar. En fil af denne type kan indsættes i OpenOffice (med INDSÆT || GRAFIK || FRA FIL) eller Word eller PowerPoint (med INDSÆT || BILLEDE || FRA FIL). Men direkte overførsel af plots er nu oftest nemmere; se afsnit 6.1.

Hvis et plot skal bruges på en webside duer Windows Enhanced Metafile ikke. I stedet kan grafen laves i grafikformatet Portable Network Graphics (PNG):

```
> png("poly.png")
> plot(p, -2, 8)
> dev.off()
```

PNG-filer er gode til websider fordi de er ret kompakte, men de bliver grimme hvis de skaleres, som det normalt er nødvendigt i dokumenter og præsentationer.

To andre gode grafikformater er Portable Document Format (PDF) og Encapsulated Postscript. Begge formater kan skaleres uden kvalitetstab og bruges i professionel bogproduktion. PDF-filer kan bruges uanset operativsystem; Postscript er mest egnet til tekstbehandling i Linux og MacOS. Plotfilerne laves på samme måde som ovenfor:

```
> postscript("poly.eps")
> plot(p, -2, 8)
> dev.off()
> pdf("poly.pdf")
> plot(p, -2, 8)
> dev.off()
```

Der er flere andre grafikfilformater end de fire nævnte; de ses ved at skrive `?device` i R Console.

7 Nulpunkt for funktion, og løsning af ligning med én ubekendt

I R kan man løse vilkårlige ligninger med én ubekendt x ved at finde nulpunkt for en passende funktion. For eksempel kan man løse ligningen $x = \cos(x)$ ved at definere funktionen $f(x) = x - \cos(x)$ og så bruge R til at finde en værdi af x for hvilken $f(x) = 0$:

```
> f <- function(x) { x - cos(x) }
> res <- uniroot(f, c(-10, 10))
> res
$root
[1] 0.7390799
$f.root
[1] -8.831189e-06
$iter
[1] 5
$estim.prec
[1] 6.103516e-05
```

Argumentet `c(-10,10)` til `uniroot` er en vektor (afsnit 14) som angiver at nulpunktet skal søges i intervallet $[-10, 10]$.

Funktionen `f` skal have forskelligt fortegn i de to endepunkter, og `uniroot` benytter så en iterativ procedure til at finde et nulpunkt for `f`. Resultatet `res` er en associationsliste (afsnit 17.1) med fire komponenter: `root` er løsningen til ligningen; `f.root` er funktionens værdi beregnet i det fundne punkt; `iter` angiver hvor meget arbejde der skulle til at finde løsningen; og `estim.prec` er den formodede fejl på løsningen.

For at få komponenten `root` ud af `res` (så man kan bruge tallet i et regneudtryk) skal man skrive `res$root`; mere herom i afsnit 17.1:

```
> res$root
[1] 0.7390799
```

Det er ikke alle funktioner der har nulpunkter og ikke alle ligninger der har løsninger, så man skal tjekke at `f.root` faktisk er tæt på nul. I eksemplet ovenfor er den `-8.831189e-06`, der som bekendt betyder $-8.831189 \cdot 10^{-6}$, altså -0.000008831178, som er ganske tæt på nul. På grund af computerens begrænsede regnenøjagtighed kan man ikke forvente at resultatet er eksakt nul.

Med `uniroot` kan man kun finde ét nulpunkt ad gangen. For at finde alle nulpunkter kan man plotte funktionen, og derefter søge nulpunkter separat for passende små intervaller omkring skæringspunkterne med første-aksen (hvor jo $f(x) = 0$). For eksempel ser man for polynomiet `p` i figur 4 at der må være et nulpunkt i intervallet $[-2, 0]$, et nulpunkt i intervallet $[0, 2]$, og et nulpunkt i intervallet $[4, 6]$:

```
> uniroot(p, c(-2,0)) $ root
[1] -1.263545
> uniroot(p, c(0,2)) $ root
[1] 1.755642
> uniroot(p, c(4,6)) $ root
[1] 4.507902
```

Men netop i specialtilfældet hvor funktionen er et polynomium $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ kan man finde alle rødder på en gang ved at bruge den specielle funktion `polyroot(c(`a_0`, `a_1`, ..., `a_n`))`.

8 Numerisk optimering: minimum og maksimum

Numerisk optimering af en funktion f går ud på at finde den værdi af x som giver den mindste (eller største) værdi $f(x)$ af funktionen. For eksempel kunne $f(x)$ udtrykke det økonomiske udbytte ved at tilføre x enheder kunstgødning til en mark. Dette udbytte består jo af en indtægt fra salg af høstudbyttet minus en udgift til indkøb og spredning af kunstgødningen. Lad os antage at høstudbyttet som funktion af gødningstilførslen t beskrives af funktionen

$$u(t) = \frac{20t}{t+1} + 10$$

og at vi tjener 4000 kroner per ton høstudbytte men skal betale 5000 kroner per ton gødning, så det økonomiske udbytte som funktion af gødningstilførslen t er

$$f(t) = 4000u(t) - 5000t$$

Nu kan vi bruge R til at finde den værdi af t der maksimerer fortjenesten:

```
> u <- function(t) { 20 * t / (t+1) + 10 }
> f <- function(t) { 4000 * u(t) - 5000 * t }
> optimize(f, interval=c(0,100), maximum=TRUE)
$maximum
[1] 2.999998
$objective
[1] 85000
```

Argumenterne til funktionen `optimize` er først funktionen $f(t)$ der skal maksimeres og dernæst det interval af t-værdier der skal undersøges, her $[0, 100]$. Parameteren `maximum=TRUE` angiver at $f(t)$ skal maksimeres, ikke minimeres. Resultatet er en associationsliste (afsnit 17.1) med to komponenter: `maximum` er den værdi af t der maksimerer $f(t)$, og `objective` er den tilsvarende maksimale værdi af $f(t)$.

Man skal tænke sig om når man bruger numerisk optimering, for der er flere faldgruber:

1. Det kan være at funktionen slet ikke har noget minimum (eller maksimum), enten fordi den er ubegrænset eller fordi den konvergerer mod en øvre (eller nedre) grænse for $x \to \pm\infty$. For eksempel har funktionen $g(x) = -x^2$ ikke noget minimum, og $h(x) = 1 - |\frac{1}{x}|$ har ikke noget maksimum.

2. Det kan være at funktionen har flere minima (eller maksima); i så fald giver R's optimeringsrutine bare et af dem, men ikke nødvendigvis det man forventede. Således er det vigtigt i eksemplet ovenfor at søge maksimum blandt ikke-negative værdier af gødningstilførslen t, her intervallet $[0, 100]$. Dels er det meningsløst i praksis at tilføre en negativ mængde gødning, dels kan funktionen f faktisk antage vilkårlig store værdier for store negative t.

3. Det kan være at funktionen har lokale minima (eller maksima) og at R's optimeringsrutine finder et af de lokale minima (eller maksima) i stedet for et af de globale.

Som eksempel på det første problem kan vi forsøge at minimere funktionen $g(x) = -x^2$. Som det ses er svaret fra `optimize` helt hen i vejret:

```
> g <- function(x) { -x^2 }
> optimize(g, interval=c(-100,100))
$minimum
[1] 23.6068
$objective
[1] -557.2809
```

Funktionen `optimize` kan kun bruges til at optimere (maksimere eller minimere) en funktion $f(x)$ af ét argument. En anden funktion, `optim`, kan bruges til at optimere funktioner af flere argumenter.

9 Numerisk integration

Man kan bruge R til at foretage numerisk integration af en funktion. For eksempel kan man integrere sinusfunktionen fra 0 til π:

```
> integrate(sin, 0, pi)
2 with absolute error < 2.2e-14
```

Dette virker naturligvis på samme måde med en funktion man selv har defineret:

```
> f <- function(x) { 7*x^2+x^4 }
> integrate(f, 0, 3)
111.6 with absolute error < 1.2e-12
```

Ud over værdien af integralet får man også en vurdering af størrelsen af fejlen.

Funktionen `integrate` giver ikke bare en talværdi, men en associationsliste (afsnit 17.1) med bl.a. komponenterne `value` (værdien af integralet) og `abs.error` (størrelsen af fejlen).[1] Hvis man ønsker at regne videre på resultatet af integrationen skal man altså udtage `value` fra listen med notationen `res$value`, fx som følger:

```
> res <- integrate(f, 0, 3)
> res$value * 3
[1] 334.8
```

Det er muligt at integrere til eller fra uendelig (`Inf`), så længe resultatet er endeligt:

```
> f1 <- function(x) { exp(-2*x) }
> integrate(f1, 0, Inf)
0.5 with absolute error < 7.7e-05
> f2 <- function(x) { 1/x^2 }
> integrate(f2, 1, Inf)
1 with absolute error < 1.1e-14
> f3 <- function(x) { exp(-x^2) }
> integrate(f3, -Inf, Inf)
1.772454 with absolute error < 4.3e-06
```

Funktionen `integrate` virker ved at beregne værdien af den givne funktion i passende mange punkter. Antallet kan begrænses med parameteren `subdivisions`, hvis standardværdi er 100. For funktioner såsom $f(x) = \sin(\frac{1}{x})$, som svinger meget så det er vanskeligt at beregne integralet, kan det være nødvendigt at forøge `subdivisions` for at få et resultat:

```
> f4 <- function(x) { sin(1/x) }
> integrate(f4, 0, 1)
Error in integrate(f4, 0, 1) :
        maximum number of subdivisions reached
> integrate(f4, 0, 1, subdivisions=10000)
0.5041151 with absolute error < 9.7e-05
```

Man kan ikke bruge R til at beregne stamfunktion ved symbolsk integration.

[1] Som det ses af eksemplerne vises denne associationsliste på en anden måde end dem man får fra `uniroot` og `optimize` beskrevet i de foregående afsnit, men det vil føre for vidt at forklare hvorfor.

10 Datasæt

Datasæt (data frames) er et centralt begreb i R, der jo netop er udviklet primært med henblik på statistisk analyse. Selv om det i disse noter ikke er de statistiske faciliteter der fokuseres på er det alligevel nyttigt og nødvendigt at beskrive de mest basale aspekter af datasæt i R, specielt hvad angår indlæsning, transformering og grafisk fremstilling af forsøgsdata.

Man kan tænke på et datasæt som en tabel eller et skema, hvor hver række er en sammenhængende observation og hver søjle er en variabel der er observeret. I figur 16 er som eksempel i tabelform vist et datasæt der stammer fra et forsøg med slagtesvin der har fået vækshormon. Datasættet har de tre variable `Tid`, `Kontrol` og `Vaekst`, og 20 observationer. Hver observation angiver et antal minutter efter slagtning (søjlen `Tid`) sammen med målt pH i kødet fra grise opdrættet normalt (søjlen `Kontrol`, dvs. kontrolgruppen) og fra grise der har fået væksthormon (søjlen `Vaekst`).

Tid	Kontrol	Vaekst
30	6.45	6.07
45	6.32	5.94
60	6.17	5.82
75	6.06	5.72
90	6.02	5.62
105	5.98	5.54
120	5.94	5.47
150	5.85	5.38
180	5.80	5.37
210	5.76	5.34
240	5.71	5.34
270	5.63	5.35
300	5.59	5.31
330	5.52	5.31
360	5.49	5.32
390	5.47	5.32
420	5.45	5.33
450	5.44	5.33
480	5.43	5.34
1440	5.43	5.34

Figur 16: Datasæt fra forsøg med slagtesvin. Der er tre variable og 20 observationer.

Afsnit 10.1 handler om hvordan datasæt kan indlæses fra tekstfiler til R. Appendiks C beskriver indlæsning til R fra andre kilder, fx regneark. Afsnit 10.2 beskriver kort hvordan datasæt kan eksporteres til tekstfiler der derefter kan indlæses i regnearksprogrammer. Afsnit 10.3 handler om hvordan man kan få et hurtigt overblik over et datasæt og dermed kontrollere at det er indlæst korrekt. Afsnit 10.4 handler om hvordan man kan regne på værdierne fra et datasæt og transformere variable.

Afsnit 12 beskriver hvordan man kan lave en grafisk fremstilling af målingerne i et datasæt. Afsnit 13 beskriver lineær regression og regressionskurver bl.a. ud fra datasæt.

Afsnit 17 beskriver lidt mere formelt hvilken type objekt et datasæt er i R og hvordan man kan oprette datasæt (uden indlæsning).

10.1 Indlæsning af forsøgsdata

Forsøgsdata i tekstfiler foreligger normalt enten i *mellemrumsseparereret format* eller *kommaseparereret format*. Figur 17 viser hvordan de samme data ser ud i de to formater hvis man bruger fx NotePad eller WordPad til at se på tekstfilerne. Indlæsning fra mellemrumsseparereret format er beskrevet i afsnit 10.1.1 og indlæsning fra kommaseparereret format er beskrevet i afsnit 10.1.2.

Kopiering fra regneark (OpenOffice og Excel) til R er beskrevet i appendiks C.1. En anden metode til dataoverførsel fra regneark, som formentlig er bedre ved meget store datasæt, er beskrevet i appendiks C.3.

10.1.1 Forsøgsdata i mellemrumsseparereret format

Figur 17 (a) viser en tekstfil med data opstillet i mellemrumsseparereret format. I eksemplet står overskrifter og data nydeligt opstillet under hinanden fordi der er tilpas mange mellemrumstegn på hver linie, men det vigtige er at kolonnerne er adskilt med mindst ét mellemrum.

```
 Tid Kontrol  Vaekst
  30    6,45    6,07
  45    6,32    5,94
  60    6,17    5,82
  75    6,06    5,72
  90    6,02    5,62
 105    5,98    5,54
 120    5,94    5,47
 150    5,85    5,38
 180    5,80    5,37
 210    5,76    5,34
 240    5,71    5,34
 270    5,63    5,35
 300    5,59    5,31
 330    5,52    5,31
 360    5,49    5,32
 390    5,47    5,32
 420    5,45    5,33
 450    5,44    5,33
 480    5,43    5,34
1440    5,43    5,34
```

(a) Mellemrumsseparereret `Grise2Fast.txt`

```
Tid;Kontrol;Vaekst
30;6,45;6,07
45;6,32;5,94
60;6,17;5,82
75;6,06;5,72
90;6,02;5,62
105;5,98;5,54
120;5,94;5,47
150;5,85;5,38
180;5,8;5,37
210;5,76;5,34
240;5,71;5,34
270;5,63;5,35
300;5,59;5,31
330;5,52;5,31
360;5,49;5,32
390;5,47;5,32
420;5,45;5,33
450;5,44;5,33
480;5,43;5,34
1440;5,43;5,34
```

(b) Kommaseparereret, DK `Grise2KommaDK.csv`

```
Tid,Kontrol,Vaekst
30,6.45,6.07
45,6.32,5.94
60,6.17,5.82
75,6.06,5.72
90,6.02,5.62
105,5.98,5.54
120,5.94,5.47
150,5.85,5.38
180,5.8,5.37
210,5.76,5.34
240,5.71,5.34
270,5.63,5.35
300,5.59,5.31
330,5.52,5.31
360,5.49,5.32
390,5.47,5.32
420,5.45,5.33
450,5.44,5.33
480,5.43,5.34
1440,5.43,5.34
```

(c) Kommaseparereret, US `Grise2KommaUS.csv`

Figur 17: Tre tekstfiler i (a) mellemrumsseparereret format, (b) dansk kommaseparereret format og (c) amerikansk kommaseparereret format. Dansk OpenOffice og Excel eksporterer normalt i format (b), mens engelsksprogede versioner eksporterer i format (c). Filtypen `.csv` betyder "comma-separated variables".

En tekstfil i mellemrumsseparereret format og med decimalkomma (a) kan indlæses i R med funktionen `read.table`:

```
> d <- read.table("Grise2Fast.txt", header=TRUE, dec=",")
```

Tekstfilen `Grise2Fast.txt` skal ligge i den aktuelle filmappe. Parameteren `header=TRUE` angiver at første linie af tekstfilen indeholder datasættets variabelnavne (kolonnenavne), og `dec=","` angiver at denne fil benytter decimalkomma i tal `6,45` (som på dansk, tysk og fransk) snarere end decimalpunktum `6.45` (som på amerikansk og engelsk).

Ved ovenstående ordre indlæses data fra filen til et R-datasæt `d`. Som med alle andre variable kan man få vist værdien af `d` bare ved at skrive navnet:

```
> d
    Tid Kontrol Vaekst
1    30    6.45   6.07
2    45    6.32   5.94
...
19  480    5.43   5.34
20 1440    5.43   5.34
```

R viser alle 20 rækker, men i disse noter er de midterste rækker udeladt af pladshensyn. Som det ses bruger R altid decimalpunktum i tal, uanset hvad der stod i den oprindelige tekstfil. Den venstre kolonne indeholder R's automatiske nummerering af observationerne.

Pas på, typisk fejl: Havde vi glemt `dec=","` i `read.table` ovenfor havde R alligevel uden at kny indlæst datasættet men opfattet de to kolonner med kommatal som *tekster* i stedet for som tal. Hvis man bare viser datasættet ser det hele meget tilforladeligt ud:

```
> fejl <- read.table("Grise2Fast.txt", header=TRUE)
> fejl
    Tid Kontrol Vaekst
1    30    6,45   6,07
2    45    6,32   5,94
...
19  480    5,43   5,34
20 1440    5,43   5,34
```

Man skal være meget vågen for at opdage at der her står kommaer hvor der burde være decimalpunktummer! Når man så begynder at ville lave en grafisk fremstilling af datasættet eller regne på værdierne vil man opleve det som om R opfører sig meget mystisk. Det sikreste er at bruge funktionen `summary` som beskrives i afsnit 10.3 til at kontrollere at datasættet er korrekt indlæst.

Pas også på: Havde vi glemt `header=TRUE` i `read.table` havde R også her alligevel uden at kny indlæst datasættet men så opfattet alle kolonner som tekster og selv opfundet nogle overskrifter. Hvis man bare viser datasættet ser det hele igen meget tilforladeligt ud, omend der er flere forskelle til det korrekt indlæste datasæt end før:

```
> fejl2 <- read.table("Grise2Fast.txt", dec=",")
> fejl2
     V1      V2     V3
1   Tid Kontrol Vaekst
2    30    6,45   6,07
3    45    6,32   5,94
...
20  480    5,43   5,34
21 1440    5,43   5,34
```

Bemærk at der nu er en observation for meget (21 i stedet for 20) og at overskrifterne er V1 V2 V3. Igen vil `summary` gøre det lettere at afsløre fejlen end hvis man bare viser selve datasættet.

10.1.2 Forsøgsdata i kommasepareret format

Forsøgsdata i såkaldt kommasepareret format ligger i en tekstfil hvor hver linie svarer til en observation og hvor variablene (kolonnerne) er adskilt af semikolon (`;`) eller komma (`,`) afhængig af om filen er af dansk eller amerikansk oprindelse. Tekstfiler i kommasepareret format har typisk et navn der ender på `.csv` for "comma-separated variables", men navnet kan også ende på `.txt`. Figur 17 (b) og (c) viser samme tekstfil i henholdsvis dansk og amerikansk kommasepareret format.

Tekstfiler i dansk kommasepareret format (b), med semikolon som skilletegn og med decimalkomma i tal, kan indlæses i R med funktionen `read.csv2`:

```
> d <- read.csv2("Grise2KommaDK.csv")
```

Tekstfiler i amerikansk kommasepareret format (c), med komma som skilletegn og med decimalpunktum i tal, kan indlæses i R med funktionen `read.csv`:

```
> d <- read.csv("Grise2KommaUS.csv")
```

10.2 Eksport af data fra R

Datasæt som `d` kan eksporteres fra R til tekstfil — og dermed til regneark — med funktionen `write.table`.

For at skrive i dansk kommasepareret format som i figur 17 (b), skal man angive semikolon (`;`) som separator og bruge decimalkomma (`,`) i tal:

```
> write.table(d, "output.csv", sep=";", dec=",", quote=FALSE, row.names=FALSE)
```

Parameteren `quote=FALSE` betyder at der ikke kommer anførselstegn om tekster (fx kolonneoverskrifter) i tekstfilen, og parameteren `row.names=FALSE` betyder at der ikke kommer rækkenumre på observationerne. Den resulterende tekstfil `"output.csv"` lægges i den aktuelle filmappe og kan importeres i dansk OpenOffice eller Excel med FILER || ÅBEN || FILTYPE ALLE FILER.

For at skrive i amerikansk kommasepareret format som i figur 17 (c), skal man angive komma (`,`) som separator; decimalpunktum (`.`) i tal er standard:

```
> write.table(d, "output.csv", sep=",", quote=FALSE, row.names=FALSE)
```

Den resulterende tekstfil `"output.csv"` kan åbnes direkte i (dansk) Excel ved at dobbeltklikke på den, og den kan importeres i amerikansk OpenOffice.

10.3 Overblik over et datasæt: funktionen `summary`

Funktionen `summary` beregner mindste værdi, største værdi, middelværdi, kvartiler og andre oplysninger om et datasæt. Hvis `d` er grisedatasættet indlæst i afsnit 10.1.1, giver `summary` dette resultat:

```
> summary(d)
      Tid            Kontrol          Vaekst
 Min.   :  30.0   Min.   :5.430   Min.   :5.310
 1st Qu.: 101.2   1st Qu.:5.485   1st Qu.:5.330
 Median : 225.0   Median :5.735   Median :5.345
 Mean   : 287.2   Mean   :5.776   Mean   :5.478
 3rd Qu.: 367.5   3rd Qu.:5.990   3rd Qu.:5.560
 Max.   :1440.0   Max.   :6.450   Max.   :6.070
```

Når du har indlæst et datasæt er det en god idé at bruge `summary` til at undersøge om det ser rimeligt ud. Desuden kan du bruge oplysningerne om mindste og største værdi til at vælge aksegrænser ud fra når du plotter datasættet; se afsnit 5.3.

Hvis grisedatasættet havde været indlæst forkert således at søjlerne `Kontrol` og `Vaekst` var blevet opfattet som tekst i stedet for tal (se afsnit 10.1.1) ville `summary` afsløre det med det samme:

```
> summary(fejl)
      Tid            Kontrol       Vaekst
 Min.   :  30.0   5,43   : 2   5,34   :4
 1st Qu.: 101.2   5,44   : 1   5,31   :2
 Median : 225.0   5,45   : 1   5,32   :2
 Mean   : 287.2   5,47   : 1   5,33   :2
 3rd Qu.: 367.5   5,49   : 1   5,35   :1
 Max.   :1440.0   5,52   : 1   5,37   :1
                  (Other):13   (Other):8
```

Bemærk at de to forkerte søjler ikke vises med de sædvanlige kvartiler.

10.4 Regning på værdier fra datasæt

De enkelte søjler i et datasæt fås ved at skrive datasættets og variablens navn adskilt af dollartegn ($):

```
> d$Kontrol
 [1] 6.45 6.32 6.17 6.06 6.02 5.98 5.94 5.85 5.80 5.76 5.71 5.63 5.59 5.52 5.49
[16] 5.47 5.45 5.44 5.43 5.43
```

Klammerne `[1]` og `[16]` i starten af første henholdsvis anden linie angiver at liniens første tal er observation nummer 1 henholdsvis 16.

En søjle i et datasæt er et eksempel på en *vektor* i R. Meget mere om vektorer og hvordan man kan udtage elementer fra dem og regne på dem følger i afsnit 14; her skal blot nævnes nogle få muligheder.

Man kan tage et enkelt tal ud fra rækken af observationer for en variabel, for eksempel værdi nummer 16 af variablen `Kontrol`:

```
> d$Kontrol[16]
[1] 5.47
```

Man kan bruge simple statistiske funktioner som `min` og `max` fra afsnit 11:

```
> min(d$Kontrol)
[1] 5.43
> max(d$Kontrol)
[1] 6.45
```

Man kan bruge matematiske funktioner til at *transformere* variable, for eksempel tage logaritmen:

```
> log(d$Kontrol)
 [1] 1.864080 1.843719 1.819699 1.801710 1.795087 1.788421 1.781709 1.766442
 [9] 1.757858 1.750937 1.742219 1.728109 1.720979 1.708378 1.702928 1.699279
[17] 1.695616 1.693779 1.691939 1.691939
```

Man kan tilføje en transformeret variabel som ny søjle i datasættet:

```
> d$LogK <- log(d$Kontrol)
> d$LogV <- log(d$Vaekst)
> d
    Tid Kontrol Vaekst     LogK     LogV
1    30    6.45   6.07 1.864080 1.803359
2    45    6.32   5.94 1.843719 1.781709
...
19  480    5.43   5.34 1.691939 1.675226
20 1440    5.43   5.34 1.691939 1.675226
```

Hvis man skal bruge en søjle fra et datasæt ofte kan man naturligvis binde den til en variabel for at undgå at skulle skrive lange udtryk som `d$Kontrol` hele tiden:

```
> K <- d$Kontrol
> K
 [1] 6.45 6.32 6.17 6.06 6.02 5.98 5.94 5.85 5.80 5.76 5.71 5.63 5.59 5.52 5.49
[16] 5.47 5.45 5.44 5.43 5.43
> K[16]
[1] 5.47
> min(K)
[1] 5.43
```

11 Simple statistiske funktioner

Figur 18 er en oversigt over nogle simple statistiske funktioner; mere avancerede statistiske funktioner i R forklares i KVL's statistikkurser. Mange af funktionerne, fx `max`, `mean`, `min`, `prod`, `sd`, `sum` og `var`, tager en *vektor* som argument. Vektorer beskrives nærmere i afsnit 14, men som allerede nævnt er en søjle fra et datasæt en vektor, så for eksempel kan man finde middelværdien (gennemsnittet) af de målte pH værdier for kontrolgruppen i grisedatasættet som følger:

```
> mean(d$Kontrol)
[1] 5.7755
```

Funktionerne `runif` og `rnorm`, der står for "random uniform" (ligefordeling) og "random normal" (normalfordeling), bruges til at generere pseudotilfældige tal. Dette er især nyttigt til simulering af biologiske eller økonomiske processer, se fx afsnit 15.1.

Funktion	Resultat
`lm(formel, data=datasæt)`	Tilpasning af lineær model, lineær regression (afsnit 13)
`max(v)`	Maksimum af elementerne i vektor `v`
`mean(v)`	Middelværdi af elementerne i vektor `v`
`median(v)`	Medianen af elementerne i vektor `v`
`min(v)`	Minimum af elementerne i vektor `v`
`prod(v)`	Produktet af elementerne i vektor `v`
`rnorm(n)`	Vektor med `n` tilfældige tal, normalfordelt $N(0,1)$
`rnorm(n,m,s)`	Vektor med `n` tilfældige tal, normalfordelt $N(\mathtt{m},\mathtt{s})$
`runif(n)`	Vektor med `n` ligefordelte tilfældige tal i $[0,1]$
`runif(n,min=1,max=4)`	Vektor med `n` ligefordelte tilfældige tal i $[1,4]$
`sd(v)`	Standardafvigelse af elementerne i vektor `v`
`sum(v)`	Summen af elementerne i vektor `v`
`summary(d)`	Min, max, middelværdi og kvartiler i `d` (afsnit 10.3)
`var(v)`	Varians af elementerne i vektor `v`

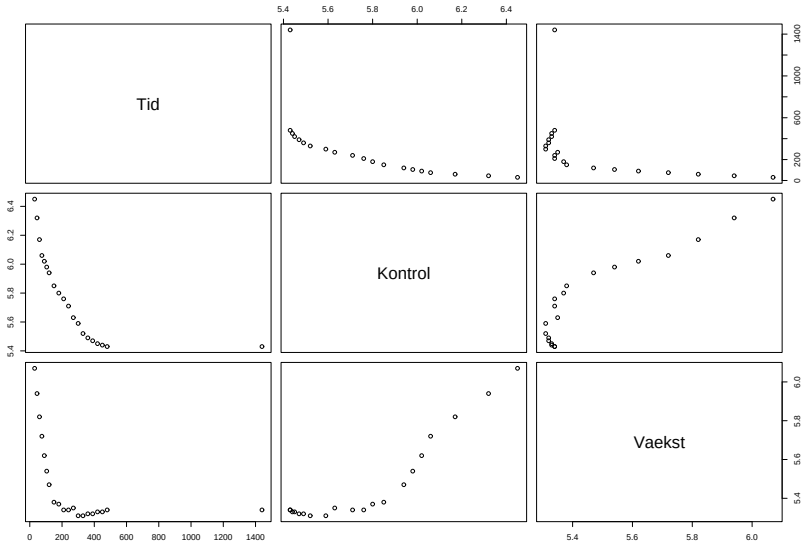
Figur 18: Nogle simple statistiske funktioner i R. Notationen $N(\mathtt{m},\mathtt{s})$ står for normalfordelingen med middelværdi `m` og standardafvigelse `s`.

12 XY-plot

Man kan plotte observationerne fra datasættet `d` indlæst ovenfor på talrige måder. Først kan man danne sig et overblik over sammenhænge i datasættet ved at plotte alle variable mod hinanden på én gang. Det gøres ved simpelthen at skrive:

```
> plot(d)
```

I dette tilfælde er der tre variable `Tid`, `Kontrol`, `Vaekst` hvilket giver $3 \cdot 2 = 6$ små XY-plots, som vist i figur 19.



Figur 19: Overbliksplot af datasæt med tre variable `Tid`, `Kontrol` og `Vaekst`.

Lige i dette tilfælde er det ikke så nyttigt, men generelt er det en god måde at få en idé om nogle af variablene for eksempel er lineært korrelerede, hvilket vil vise sig som en tilnærmelsesvis ret linie.

12.1 Plot af dataserier

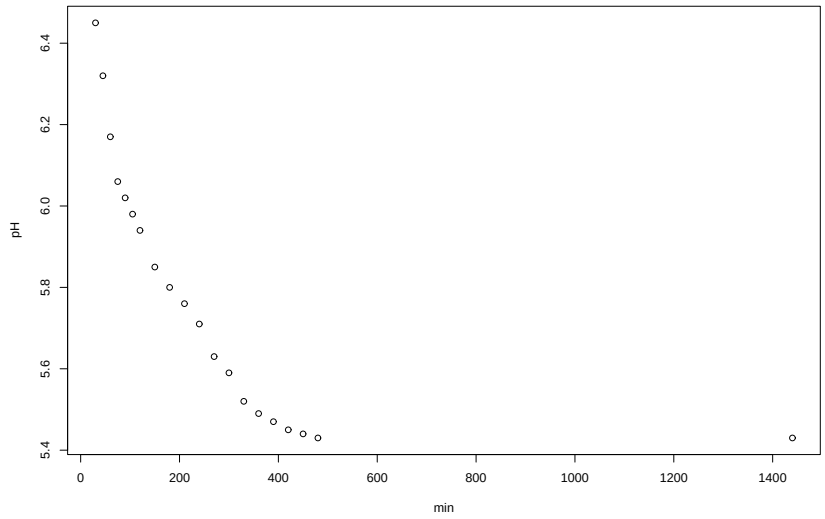
I det foreliggende datasæt er det mere nærliggende at plotte både kontrolgruppens pH `d$Kontrol` og væksthormongruppens pH `d$Vaekst` som funktion af tiden, så man kan sammenligne de to forløb.

Til at begynde med plotter vi kontrolgruppens pH som funktion af tiden med funktionen `plot`:

```
> plot(d$Tid, d$Kontrol, type="p", xlab="min", ylab="pH")
```

Argumentet `type="p"` gør at datapunkterne vises i plottet uden at der trækkes linier mellem datapunkterne; andre plottyper ses i figur 22. Desuden er der brugt `xlab` og `ylab` (se afsnit 5.5) til at bestemme aksetitler. Det resulterende plot ses i figur 20.

Hvis man vil plotte både `Kontrol` og `Vaekst` som funktion af `Tid`, så kan man plotte den første dataserie med funktionen `plot` (som ovenfor), og derefter tilføje den anden dataserie med funktionen `points`. Når man skal tilføje nye datapunkter eller XY-punktkurver til et eksisterende plot skal man bruge funktionerne `points` og `lines`. Man kan *ikke* bruge `plot(..., add=TRUE)`; det duer kun når man skal tilføje en *funktionsgraf* som i afsnit 5.2.

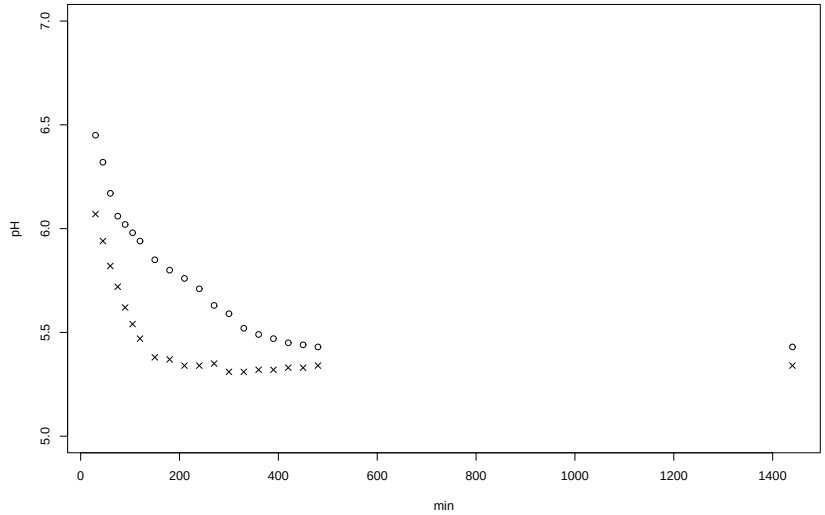


Figur 20: XY-plot af forsøgsdata: én dataserie tegnet med `plot`.

Funktionen `points` tager en vektor af x-værdier og en vektor af y-værdier og tilføjer datapunkterne (x, y) til et eksisterende plot. For eksempel kan vi tilføje et plot af `d$Vaekst` til et eksisterende plot af `d$Kontrol` sådan her:

```
> plot(d$Tid, d$Kontrol, type="p", ylim=c(5,7), xlab="min", ylab="pH")
> points(d$Tid, d$Vaekst, type="p", pch=4)
```

Her betyder parameteren `ylim=c(5,7)` at y-aksen skal gå fra 5 til 7 så begge kurver kan være der (se afsnit 5.3), og `pch=4` bruges til at vælge plotsymbol (se afsnit 12.2). Resultatet ses i figur 21.



Figur 21: XY-plot af forsøgsdata: to dataserier tegnet med `plot` og `points`.

Funktionen `points(xs, ys, ...)` virker grundlæggende som `plot(xs, ys, ...)`, bortset fra at den ikke starter et nyt plot, men tegner videre på et der findes. De samme parametre bruges til at styre

udseende af plotsymboler og eventuelle forbindelseslinier i både `plot` og `points`. Parameteren `type` vælger plottype, parameteren `pch`, der står for "plotting character" kan bruges til at vælge plotsymbol, parameteren `col` kan bruges til at vælge plotsymbolernes (og liniernes) farve, og parameteren `cex` bruges til at vælge plotsymbolernes størrelse; for eksempel betyder `cex=3` tredobling af standardstørrelsen, og `cex=0.5` betyder halvering. Hvis plottypen har linier mellem punkterne bestemmer parametrene `lty` og `lwd` henholdsvis liniernes type og tykkelse ganske som for funktionsgrafer, se figur 11.

Til forskel fra `plot` kan man ikke i `points` regulere titler, akser og udstrækningen af koordinatsystemet, ganske som man ikke kan det for funktionsgrafer man tilføjer til et eksisterende plot med `add=TRUE`. Det vil sige at man lige som for funktionsgrafer skal sørge for at koordinatområde osv. sættes korrekt op i det første kald af `plot` så datapunkter der efterfølgende tilføjes med `points` ikke falder udenfor.

Den generelle procedure for at plotte flere dataserier sammen er altså:

- Plot den første dataserie med `plot`. Sørg for at sætte koordinatområde, titler, akser osv. korrekt.
- Tilføj følgende dataserier med `points`.
- Tilføj eventuelle funktionsgrafer med `plot(...,add=TRUE)` og regressionslinier med `abline`, se afsnit 13.
- Tilføj eventuelle forklarende tekster og signaturforklaring.

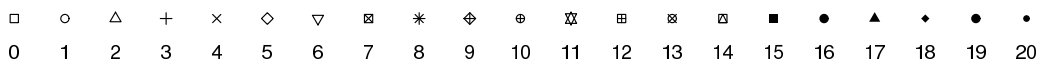
Funktionen `lines` er fuldstændig analog med `points` bortset fra at standardværdien for plottypen er `type="l"`, således at der (med mindre man angiver en anden `type`) tegnes uden plotsymboler og med linier mellem datapunkterne. Hvis man angiver parameteren `type` er der ingen forskel på `points` og `lines`.

12.2 Plottype og plotsymboler

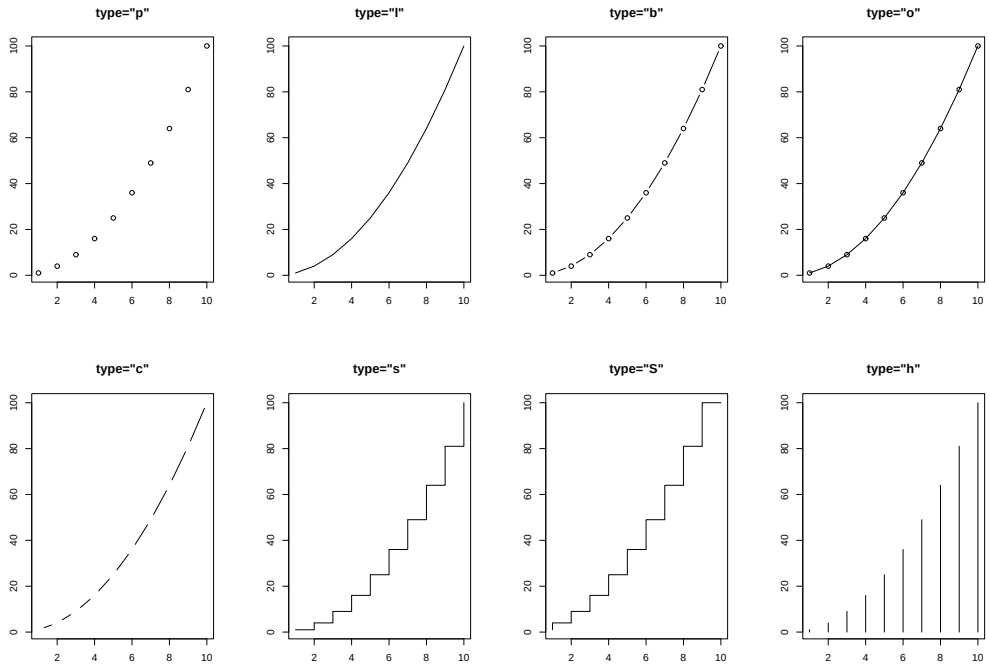
De fleste af `plot`-parametrene beskrevet i afsnit 5 kan anvendes også på XY-plot. Desuden kan man styre XY-plottets type (punkter, linier, begge dele, trinplot, histogram) med parameteren `type`, og man kan styre hvilket symbol der bruges til at plotte datapunkter med parameteren `pch`. Se tabellen i figur 22. Endvidere kan funktionen `locator` (se afsnit 5.7) bruges til at identificere datapunkter i et XY-plot ligesom i et funktionsplot.

Parameter	Effekt	Mulige værdier
`cex`	symbolskalering	Tal, fx giver `cex=2` fordobling, `cex=0.5` halvering
`pch`	plotsymbol	`pch=0` eller `1`, `2`, ..., `20`; se figur 23
`type`	plottype	`type="p"` eller `"l"`, `"b"`, `"o"`, `"c"`, `"s"`, `"S"`, `"h"`; se figur 24

Figur 22: Nogle grafikparametre til brug i XY-`plot`, `points` og `lines`.



Figur 23: Plotsymboler til `plot` og `points`. For eksempel giver `pch=2` en åben trekant.



Figur 24: Plottyper: punkter, linier, to kombinationer, linjestykker, to slags trinplot, og histogram.

12.3 Signaturforklaring i XY-plot

Funktionen `legend` som blev beskrevet i afsnit 5.6 kan også tegne signaturforklaringer med plotsymboler. Dette gøres ved at angive parameter `pch` og en liste af plotsymboler pakket ind i `c(...)`, analogt med måden man i `legend` angiver farver, linietyper og linietykkelser. Eksempel:

```
> plot(d$Tid, d$Kontrol, type="p", ylim=c(5,7), xlab="min", ylab="pH")
> points(d$Tid, d$Vaekst, type="p", pch=4)
> legend(800, 6.75, c("Kontrol","Vaekst"), pch=c(1,4))
```

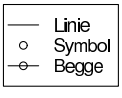
Resultatet ses i figur 25. Bemærk i eksemplet at der ikke er angivet nogen værdi for `pch` til `plot`; den første dataserie tegner R derfor med standard plotsymbol svarende til `pch=1` og derfor skal den første `pch`-værdi i `legend` være 1.

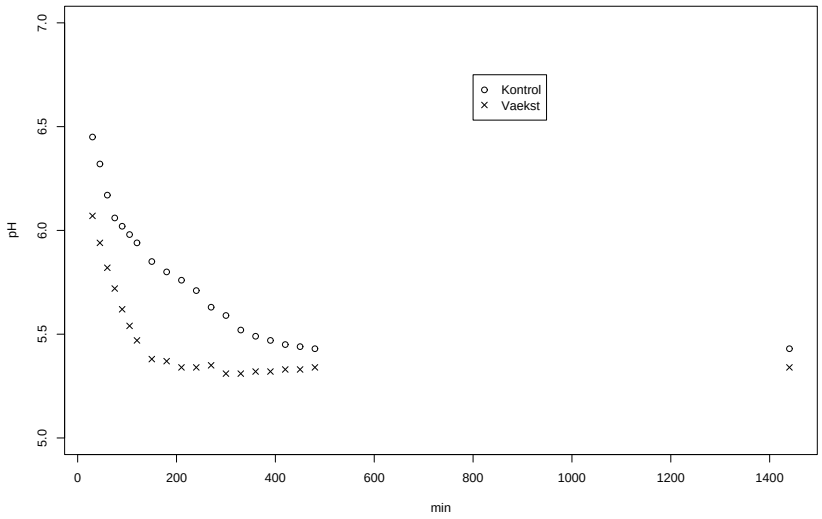
Hvis man har skaleret plotsymboler i XY-plottet med parameteren `cex` bør man foretage den samme skalering i signaturforklaringen. Her hedder den tilsvarende parameter imidlertid ikke `cex` men derimod `pt.cex`.

Man kan have brug for at tegne en signaturforklaring med både plotsymboler og linier og dette gøres ved at angive både `pch` og `lty`/`lwd` parametre til `legend`. Hvis en signatur kun skal have plotsymbol (og ingen linie) angives `"blank"` på den tilsvarende plads i `lty` listen og hvis omvendt en signatur alene er en linie angives -1 på den tilsvarende plads i `pch` listen. Eksempel:

```
> legend(x, y, c("Linie","Symbol","Begge"),
+        lty=c("solid","blank","solid"), pch=c(-1,1,1))
```

Her har vi delt den lange indtastning over to linier. Dette giver denne signaturforklaring:





Figur 25: XY-plot af forsøgsdata med signaturforklaring tegnet med `legend`.

12.4 Eksempel på et matematisk XY-plot

Dette afsnit bruger funktioner og begreber fra afsnit 14 nedenfor til at lave et "matematisk" plot ved hjælp af funktionerne `plot`, `lines` og `points`. For eksempel kan man tegne enhedscirklen, en spiral, og en lodret tangent til spiralen som vist i figur 26. Vi begynder med at lave en vektor `ts` med 100 vinkler fra 0 til 2π med funktionen `seq` fra afsnit 14.1:

```
> ts <- seq(0, 2*pi, len=100)
```

Dernæst beregner man vektorer af enhedscirklens x- og y-koordinater ud fra vinklerne, og plotter med `type="l"` de linier der forbinder disse (x, y)-koordinater:

```
> xs <- cos(ts)
> ys <- sin(ts)
> plot(xs, ys, type="l", asp=1, xlim=c(-2,2), ylim=c(-2,2))
```

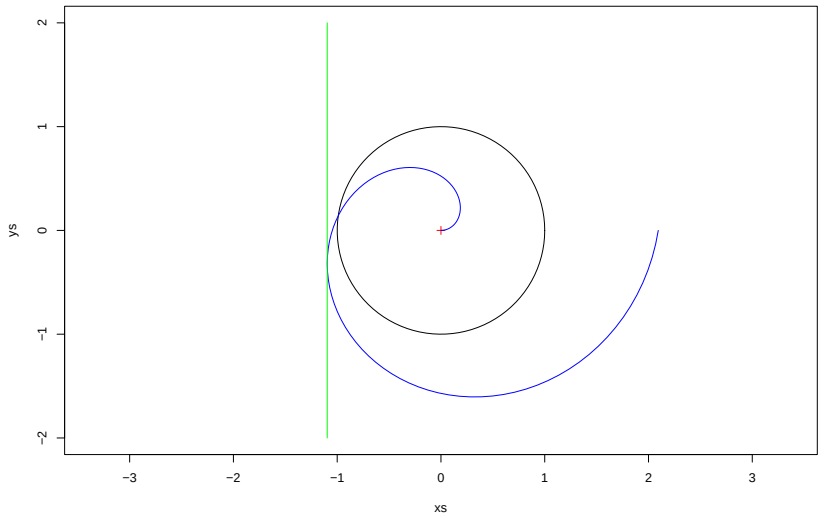
Resultatet er den sorte cirkel midt i figur 26. Parameteren `asp=1`, der står for "aspect ratio" betyder at en y-akseenhed skal være lige så lang som en x-akseenhed i plottet. Hvis ikke de er det, så ligner cirklen en ellipse i stedet.

Man kan bruge `points` til at indsætte enkeltsymboler i plottet, fx til at sætte plotsymbol 3, dvs. et kryds, i origo $(0, 0)$ (med rødt):

```
> points(0, 0, pch=3, col="red")
```

Hvis man ganger x- og y-koordinaterne for cirklen med en stigende sekvens af tal, fx vinklerne `ts`, så får man en spiral. Vi kan bruge funktionen `lines` til at tilføje en blå spiral til det eksisterende plot, idet vi dividerer begge koordinater med konstanten 3 for at spiralen bliver inden for plottet:

```
> lines(xs * ts/3, ys * ts/3, col="blue")
```



Figur 26: Enhedscirkel, nulpunkt (rødt kryds), spiral (blå) og lodret tangentlinie til spiralen (grøn).

Nu kunne vi ønske at tegne den lodrette tangent til spiralen gennem det punkt hvis koordinater er omtrent $(-1.1, -0.3)$. For at finde en mere præcis værdi for tangentens x-koordinat skal vi bruge at spiralens x-koordinat som funktion af vinklen t er $x(t) = t\cos(t)/3$ så dens afledede er $x'(t) = cos(t)/3 - t\sin(t)/3$. Til at finde den værdi t_0 mellem 1 og 5 hvor spiralens tangent er lodret, kan vi bruge `uniroot` fra afsnit 7 til at finde det t_0 der giver $x'(t_0) = 0$:

```
> dx <- function(t) { cos(t)/3 - t*sin(t)/3 }
> t0 <- uniroot(dx, c(1,5)) $ root
> t0
[1] 3.425601
```

Så kan vi ud fra t_0 beregne x-koordinaten x_0 for den lodrette tangent til spiralen og tegne tangentlinien med grønt, som et liniestykke fra $(x_0, -2)$ til $(x_0, 2)$:

```
> x0 <- cos(t0)*t0/3
> x0
[1] -1.096124
> lines(c(x0, x0), c(-2, 2), col="green")
```

13 Lineær regression og regressionskurve

Betragt et datasæt bestående af fem sammenhørende værdier af t og y:

t	1.0	1.5	2.0	2.5	3.0
y	1.4	0.3	-1.5	-3.1	-4.8

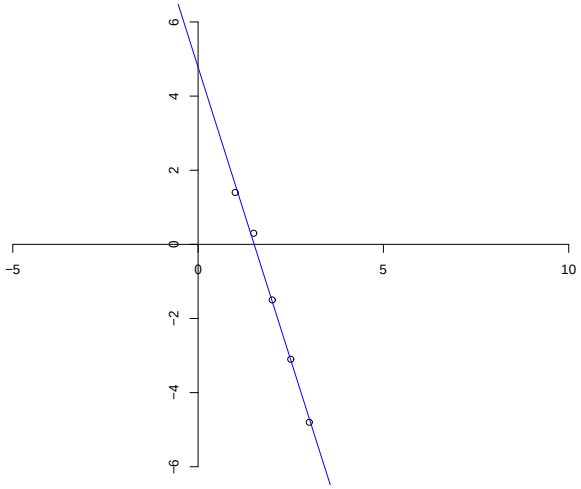
Hvis vi plotter disse fem punkter i et koordinatsystem falder de omtrent på en ret linie; se figur 27.

Lineær regression finder ligningen for den rette linie der bedst passer til punkterne. Hertil benyttes R-funktionen `lm`, der står for "linear model". Først definerer vi vektorerne `t` og `y`, og derefter kalder vi `lm` med en såkaldt R-formel `y ~ t` for at angive at vi vil modellere `y` som funktion af `t`. Funktionen `lm` finder så koefficienterne b og a i modellen $y = b + at$:

```
> t <- c(1.0, 1.5, 2.0, 2.5, 3.0)
> y <- c(1.4, 0.3, -1.5, -3.1, -4.8)
> lreg <- lm(y ~ t)
> lreg$coefficients
(Intercept)           t
       4.78       -3.16
```

Den beregnede regressionslinie er $y = 4.78 - 3.16t$. Resultatet `lreg` af `lm` er en associationsliste (afsnit 17) hvis komponent `coefficients` er en vektor der indeholder koefficienterne b og a. Ved at skrive `summary(lreg)` kan man få mange flere oplysninger om regressionen.

Nu kan man først plotte datasættets observationer med funktionen `plot` og dernæst tilføje regressionslinien til plottet med funktionen `abline`. Det resulterende plot ses i figur 27.



Figur 27: Forsøgsdata med lineær regressionslinie.

```
> plot(t, y, type="p", ylim=c(-6,6), axes=FALSE, asp=1)
> axis(1, pos=0)
> axis(2, pos=0)
> abline(lreg, col="blue")
```

Parameteren `asp=1` sætter plottets aspect ratio til 1, så enhederne på anden-aksen og på første-aksen er lige lange.

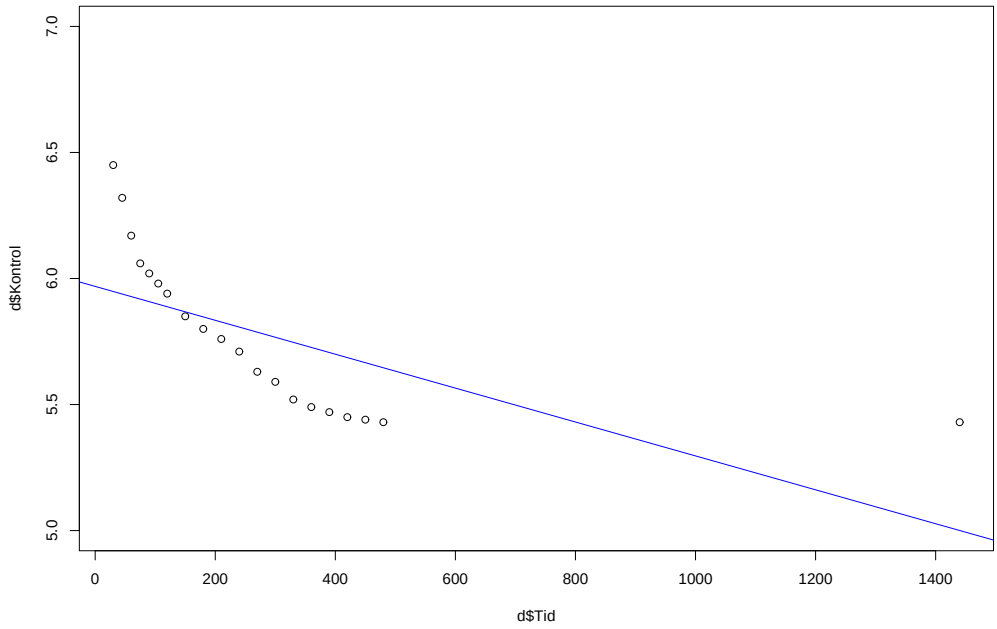
Man kan også lave lineær regression på eksisterende datasæt, som for eksempel grisedatasættet `d` fra afsnit 10.1. Lad os beregne den lineære regressionskurve for kontrolgruppens pH (variablen `Kontrol`) som funktion af tiden (variablen `Tid`). Vi kalder derfor funktionen `lm` med formlen `Kontrol ~ Tid` og skriver `data=d` fordi variablene skal tages fra datasættet `d`:

```
> lreg <- lm(Kontrol ~ Tid, data=d)
> lreg
Coefficients:
(Intercept)          Tid
  5.9686648   -0.0006725
```

Den beregnede regressionslinie `lreg` er $y = 5.9686648 - 0.0006725x$. Nu kan man først plotte datasættets observationer med funktionen `plot` og dernæst tilføje regressionslinien til plottet med funktionen `abline`; det giver datapunkterne og den rette linie i figur 28:

```
> plot(d$Tid, d$Kontrol, type="p", ylim=c(5, 7))
> abline(lreg, col="blue")
```

Det ses i figur 28 at observationerne dårligt forklares af den lineære model. I appendiks D vises det hvordan man kan lave polynomiel regression på de samme data.



Figur 28: Forsøgsdata med lineær regressionskurve.

14 Vektorer

En vektor i R er et talsæt. Vi har allerede brugt vektorer adskillige steder, fx i afsnit 5.3 hvor vi angav `ylim` i et plot med vektoren `c(-16, 200)`.

14.1 Oprettelse af vektorer

Man kan definere vektoren `v` til at være (3, 4), altså til have de to elementer 3 og 4, således:

```
> v <- c(3, 4)
```

Funktionen `c` betyder "combine" og bruges til at konstruere vektorværdier. Som altid kan man vise værdien af variablen `v` bare ved at skrive den:

```
> v
[1] 3 4
```

Man kan lave en vektor med elementerne 11 12 13 14 15 ved brug af funktionen `seq`, der er en forkortelse af "sequence":

```
> seq(11,15)
[1] 11 12 13 14 15
```

Hvis man bytter om på grænserne får man sekvensen i omvendt rækkefølge:

```
> seq(15,11)
[1] 15 14 13 12 11
```

Kolon-operatoren ":" er en kortere notation for den simple brug af `seq` som ovenfor:

```
> 11:15
[1] 11 12 13 14 15
> 15:11
[1] 15 14 13 12 11
```

Funktionen `seq` kan også bruges til at lave en inddeling af intervallet $[0, 2\pi]$ i 9 lige store delintervaller. Resultatet er en vektor af længde 10 hvis første element er 0 og sidste element er 2π:

```
> s <- seq(0, 2*pi, len=10)
> s
[1] 0.0000000 0.6981317 1.3962634 2.0943951 2.7925268 3.4906585 4.1887902
[8] 4.8869219 5.5850536 6.2831853
```

Alternativt kan man med parameteren `by` bestemme springenes størrelse direkte:

```
 > seq(0, 1, by=0.1)
  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Man kan også udelade endepunktet og kombinere `len` og `by`:

```
 > seq(0, by=0.1, len=14)
  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3
```

Med funktionen `rep`, der er en forkortelse af "replicate", kan man fx lave en vektor med 10 kopier af tallet 1:

```
> rep(1, 10)
[1] 1 1 1 1 1 1 1 1 1 1
```

Med `rep` kan man også gentage hele vektorer:

```
> rep(v, 5)
[1] 3 4 3 4 3 4 3 4 3 4
```

Funktionen `c(...)` kan bruges til at kombinere flere dele, også eksisterende vektorer, til en enkelt vektor:

```
> c(3, 4, rep(5, 10))
[1] 3 4 5 5 5 5 5 5 5 5 5 5
> c(v, v, 42)
[1]  3  4  3  4 42
```

14.2 Indeksering i vektorer

Et indeksudtryk `s[i]` giver det `i`'te element af vektoren. For eksempel får man det tredje element af `s` (fra afsnit 14.1) på denne måde:

```
> s[3]
[1] 1.396263
```

Man kan også indeksere en vektor med en vektor af indekser. På den måde kan vektor-elementer udelades, dubleres, eller omordnes:

```
> s[c(3,4,4,1)]
[1] 1.396263 2.094395 2.094395 0.000000
```

Det er således nemt med en sekvens af indeksværdier at udtage en del-vektor, for eksempel de første 5 elementer:

```
> s[1:5]
[1] 0.0000000 0.6981317 1.3962634 2.0943951 2.7925268
```

Man kan lave avanceret udvælgelse af elementer ved at indeksere med vektorer af sandhedsværdier; det kommer senere i afsnit 19.6.

Man kan ændre enkelte værdier i en vektor med tildelinger:

```
> w <- 1:10
> w
 [1]  1  2  3  4  5  6  7  8  9 10
> w[3] <- 17
> w
 [1]  1  2 17  4  5  6  7  8  9 10
> w[c(2,6,8)] <- 101:103
> w
 [1]   1 101  17   4   5 102   7 103   9  10
```

14.3 Regning med vektorer

Funktionerne `sum`, `mean`, `min` og `max` fra afsnit 11 beregner sum, gennemsnit, minimum og maksimum af elementerne i en vektor, her `v = c(3, 4)`:

```
> sum(v)
[1] 7
> mean(v)
[1] 3.5
> min(v)
[1] 3
> max(v)
[1] 4
```

Almindelige regneoperatorer kan også bruges på vektorer. Når man regner på to vektorer der har samme antal elementer, så anvendes regneoperatoren elementvis, og resultatet er en vektor med samme antal elementer:

```
> c(1,2,3) + c(7,9,13)
[1]  8 11 16
> c(1,2,3) * c(7,9,13)
[1]  7 18 39
```

Bemærk at ovenstående ikke beregner prikproduktet. Prikproduktet eller indre produkt eller af to vektorer med samme antal elementer beregnes som summen af de to vektorers elementvise produkter:

```
> sum(c(1,2,3) * c(7,9,13))
[1] 64
```

Alternativt kan prikproduktet beregnes som matrixprodukt, se afsnit 16.4. Længden (i planen eller rummet) af en vektor `v = c(3, 4)` kan beregnes som kvadratroden af prikproduktet af vektoren med sig selv, altså:

```
> sqrt(sum(v * v))
[1] 5
```

Pas på: Udtrykket `length(v)` giver antallet af elementer i vektoren, *ikke* vektorens geometriske længde (i planen eller rummet):

```
> length(v)
[1] 2
```

To vektorer med forskelligt antal elementer kan bruges i samme regneudtryk:

```
> c(3, 4) + c(11, 13, 17, 19)
[1] 14 17 20 23
```

I R-systemet gælder der følgende *genbrugsregel*: Når to vektorer med forskelligt antal elementer bruges i samme regneudtryk, så genbruges elementerne fra vektoren med færrest elementer så antallet passer med den vektor der har flest elementer. Udtrykket ovenfor beregnes som om elementerne fra `c(3, 4)` var blevet gentaget, på præcis samme måde som dette:

```
> c(3, 4, 3, 4) + c(11, 13, 17, 19)
[1] 14 17 20 23
```

Bemærk at dette er helt anderledes end i matematik, hvor det ikke giver nogen mening at lægge en vektor der har to elementer sammen med en vektor der har fire elementer.

Et simpelt tal og en vektor kan også bruges i samme regneudtryk. Faktisk opfatter R et almindeligt tal som en vektor af længde én. I overensstemmelse med genbrugsreglen vil det simple tal blive genbrugt for hvert element i vektoren. For eksempel kan man beregne kvadraterne på de første fem primtal sådan her:

```
> c(2, 3, 5, 7, 11) ^ 2
[1]   4   9  25  49 121
```

Dette regneudtryk betyder det samme som

```
> c(2, 3, 5, 7, 11) ^ c(2, 2, 2, 2, 2)
```

Tilsvarende gælder at hvis man anvender en indbygget matematisk funktion af én variabel på en vektor, så anvendes funktionen elementvis:

```
> sqrt(c(49, 81, 169))
[1]  7  9 13
```

Det fungerer fordi funktionen `sqrt` forventer et enkelt tal som argument, og den eneste fornuftige måde at anvende den på en vektor af tal er at anvende den på hvert tal for sig. Det samme gælder de funktioner man selv definerer, hvis de kun anvender de indbyggede matematiske funktioner og almindelige regneoperatorer:

```
> cbrt <- function(x) { x^(1/3) }
> cbrt(c(8, 27, 64))
[1] 2 3 4
```

Hvis der er tvivl om hvordan funktionen virker på en vektor, så er det bedre udtrykkeligt at angive at funktionen skal anvendes på hvert element for sig. Hertil bruges R-funktionen `sapply` der tager en vektor som første argument og en funktion som andet argument, anvender funktionen på hvert element af vektoren, og producerer en vektor af resultaterne:

```
> sapply(c(8, 27, 64), cbrt)
[1] 2 3 4
```

14.4 Vektorer af tekster

Nogle gange har man i R brug for at angive "vektorer" hvis elementer er tekster i stedet for tal. En del eksempler på dette så vi i forbindelse med signaturforklaringer i afsnit 5.6 og afsnit 12.3.

Vektorer af tekster angives med `c` som man forventer:

```
> c("red", "green", "blue")
[1] "red"   "green" "blue"
```

Det giver naturligvis ingen mening at regne på sådanne "vektorer", men indeksering virker som forventet.

15 `For`-løkker

En `for`-løkke udfører den samme beregning én gang for hvert element i en vektor[2] efter tur. Dette er forskelligt fra fx `sapply` (afsnit 14.3) som udfører den samme operation på hvert element i en vektor "på en gang" og giver en vektor med resultaterne.

En `for`-løkke har følgende form:

`for (` *variabel* `in` *udtryk1* `)` *udtryk2*

For-løkken evalueres ved at hvert element i vektoren *udtryk1* bindes efter tur til variablen *variabel* og *udtryk2* evalueres med den værdi af *variabel*. Efter løkken beholder variablen *variabel* den sidste værdi, den havde i løkken. Udtrykket *udtryk2* kaldes løkkens *krop*. Et udtryk i R kan, som vi skal se nærmere på i afsnit 19.1, være mange forskellige ting, for eksempel en sekvens af udtryk eller tildelinger til variable, så man kan lave vilkårligt komplekse ting i kroppen af en `for`-løkke. For eksempel er en `for`-løkke i sig selv et udtryk så man kan endda have flere løkker inde i hinanden. I eksemplerne i disse noter vil vi konsekvent skrive løkkekroppe med krøllede parenteser `{ ... }` omkring.

Den mest almindelige brug af en `for`-løkke er til trinvis beregning. Som et simpelt eksempel kan vi bruge en `for`-løkke til at udregne 6 fakultet, altså produktet $6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6$:

```
> y <- 1
> for (x in 2:6) { y <- y*x }
> y
[1] 720
> x
[1] 6
```

Ovenstående virker som følger: Vi vil gemme resultatet af beregningen i variablen `y`. Defor giver vi først `y` værdien 1 inden løkken. Så bruger vi `for`-løkken til at løbe gennem `x`-værdierne 2, 3, ..., 6 og for hver af disse værdier af `x` ændre værdien af `y` til at være den "gamle" værdi af `y` multipliceret med `x`. Hvis vi skrev de samme operationer uden en `for`-løkke ville det se således ud:

```
> y <- 1
> x <- 2
> y <- y*x
> x <- 3
> y <- y*x
> x <- 4
> y <- y*x
> x <- 5
> y <- y*x
> x <- 6
> y <- y*x
> y
[1] 720
> x
[1] 6
```

Det er ikke altid at man overhovedet bruger selve værdierne fra vektoren til noget – det kan være vi bare ønsker at gentage en operation et vist antal gange.

[2] `For`-løkker virker faktisk ikke kun for vektorer men også for associationslister (og dermed datasæt), se afsnit 17.

15.1 Eksempel: Brownsk bevægelse

Vi vil som eksempel på brug af `for`-løkker simulere Brownsk bevægelse[3] i én dimension. Vi betragter en partikel hvis position (i én dimension) til tiden t er et tal x_t. Mellem tid t og tid $t+1$ bevæger partiklen sig et lille stykke ϵ_t, hvor ϵ_t er et ligefordelt[4] tilfældigt tal mellem -1 og 1 uafhængigt af x_t og af senere og tidligere værdier af ϵ. Modellen er altså:

$$x_{t+1} = x_t + \epsilon_t,$$

med ϵ_t et tilfældigt tal mellem -1 og 1.

Vi vil nu starte med $x_0 = 0$ og simulere bevægelsen frem til $t = 100$, altså 100 trin:

```
> x <- 0
> for ( t in 1:100 ) { x <- x + runif(1, -1, 1) }
> x
[1] -3.419144
```

Funktionskaldet `runif(1, -1, 1)` giver et pseudotilfældigt tal mellem -1 og 1, se afsnit 11.

Efter 100 trin har partiklen altså flyttet sig cirka 3.42 enheder i lige netop denne simulation – hvis vi kører simulationen igen vil partiklen sandsynligvis flytte sig anderledes.

Men hvad hvis vi er interesseret i de mellemliggende positioner og ikke kun x_{100}? Så må vi gemme de mellemliggende x-værdier som følger.

Først opretter vi en vektor `X` med 100 pladser (hver med den vilkårlige startværdi 0) til at indeholde de 100 værdier x_1 til x_{100}. Derefter gentager vi eksperimentet, men gemmer nu i løkken hver værdi x_t i `X[t]`:

```
> X <- rep(0, 100)
> x <- 0
> for ( t in 1:100 ) { x <- x + runif(1, -1, 1) ; X[t] <- x }
> x
[1] -1.130062
```

I løkkekroppen har vi nu *to* tildelinger, adskilt med semikolon. Denne notation (som forklares nærmere i afsnit 19.2) betyder at tildelingerne udføres efter hinanden i rækkefølge fra venstre til højre. Det vil i dette tilfælde sige at først beregnes den nye værdi af `x` og derefter gemmes den i `X[t]`. I første gennemløb af løkken er `t`=1 og x_1 gemmes derfor i `X[1]`. I andet gennemløb er `t`=2 og x_2 gemmes derfor i `X[2]`, osv. Skrevet ud uden en `for`-løkke ville det se således ud:

```
> X <- rep(0, 100)
> x <- 0
> x <- x + runif(1, -1, 1) ; X[1] <- x
> x <- x + runif(1, -1, 1) ; X[2] <- x
   ... (ialt 100 linier af denne type) ...
> x <- x + runif(1, -1, 1) ; X[99] <- x
> x <- x + runif(1, -1, 1) ; X[100] <- x
> x
[1] -1.130062
```

[3] Opkaldt efter den skotske botaniker *Robert Brown* (1773-1858), som i 1827 da han kiggede på pollen i vand under mikroskop observerede, at små partikler i en væske bevægede sig rundt tilfældigt i små ryk (hvilket skyldes at væskens molekyler støder ind i partiklerne).

[4] I den gængse matematiske model for Brownsk bevægelse er ϵ ikke ligefordelt men derimod normalfordelt, men vi ønsker ikke at komme ind på normalfordelingen her.

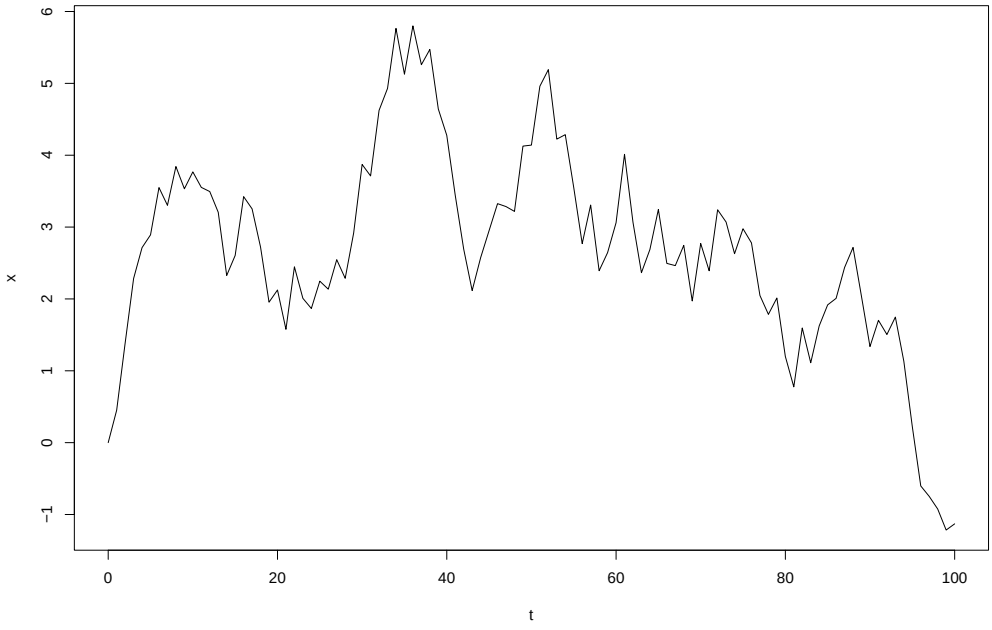
Vi har altså efter løkken værdierne x_1 til x_{100} gemt i X:

```
> X
  [1]  0.4516293  1.3836988  2.2839655  2.7141455  2.8914668  3.5494655
  [7]  3.3027996  3.8426133  3.5329267  3.7680208  3.5524554  3.4944244
   ...
 [91]  1.7025119  1.5051216  1.7475660  1.1323102  0.2250076 -0.6026496
 [97] -0.7479024 -0.9233399 -1.2161006 -1.1300624
```

Vi kan grafisk vise partiklens bevægelse som funktion af tiden:

```
> plot(0:100, c(0,X), type="l", xlab="t", ylab="x")
```

Bemærk at vi for at få (0,0) med som første punkt i grafen lod t-koordinaterne være sekvensen `0:100` og klistrede et ekstra nul på i starten af `X` med `c(0,X)`. Den resulterende graf ses i figur 29. Hvis vi gentog eksperimentet ville vi sandsynligvis få en anden graf.



Figur 29: Graf fra simulation af Brownsk bevægelse i én dimension.

15.2 Eksempel: Fibonacci-tal

Fibonacci-tallene[5] er den uendelige talrække

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

hvor Fibonacci-tal nummer n er defineret som følger:

$$\text{fib}(n) = \begin{cases} 0 & \text{hvis } n = 0 \\ 1 & \text{hvis } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{hvis } n > 1 \end{cases}$$

[5] Fibonacci: Leonardo af Pisa, kaldet Fibonacci (= søn af Bonaccio), omkr. 1180-1250, Ital. matematiker man tilskriver den første renaissance for matematikken på kristen jord.

Vi vil nu med en `for`-løkke udregne de første 30 Fibonacci-tal og gemme dem i en vektor `fib` således at `fib[1]` er Fibonacci-tal nummer 1, `fib[2]` er Fibonacci-tal nummer 2, og så videre:

```
> fib <- rep(1,30)
> for(i in 3:30) { fib[i] <- fib[i-1] + fib[i-2] }
> fib
 [1]      1      1      2      3      5      8     13     21     34     55
[11]     89    144    233    377    610    987   1597   2584   4181   6765
[21]  10946  17711  28657  46368  75025 121393 196418 317811 514229 832040
```

Virkemåden i detaljer: Først sættes `fib` til at være en vektor med 30 elementer, alle med værdien 1. Da Fibonacci-tal nummer 1 og 2 begge er 1 er `fib[1]` og `fib[2]` hermed korrekte mens `fib[3]` til `fib[30]` skal beregnes i `for`-løkken. Løkkekroppen udføres for hvert tal `i` mellem 3 og 30, og beregner `fib[i]` som summen af de to foregående (og allerede beregnede) tal `fib[i-1]` og `fib[i-2]`. Til sidst vises indholdet af `fib`.

Dette eksempel udbygges i afsnit 19.7.2.

16 Matricer

En matrix i R er ligesom i matematik et rektangulært skema af tal.

16.1 Oprettelse af matricer

Man kan definere matricen

$$A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

sådan her

```
> A <- matrix(c(1,2,3,4), 2)
```

Det sidste 2-tal betyder at matricen skal have to rækker. Det kan også skrives `nrow=2`, og tilsvarende kan man skrive `ncol=2` for at angive at der skal være to søjler. Som sædvanlig vises værdien af `A` ved at skrive navnet:

```
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Bemærk at `A` oprettes søjlevis: de to første elementer går til søjle 1, de to sidste til søjle 2. Hvis man i stedet vil bruge elementerne til rækkevis oprettelse af matricen kan man give argumentet `byrow=TRUE`, så de to første elementer går til række 1 og de to sidste til række 2:

```
> A <- matrix(c(1,3,2,4), 2, byrow=TRUE)
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

I udskrifterne betyder klammen `[,1]` søjle 1, og klammen `[1,]` betyder række 1.

Matricer kan have lige så mange rækker og søjler man ønsker, og behøver ikke være kvadratiske:

```
> M1 <- matrix(1:12, 3)
> M1
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> M2 <- matrix(1:8, 4)
> M2
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

Man kan definere enhedsmatricen

$$E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

med 2 rækker og 2 søjler ved brug af funktionen `diag`:

```
> E <- diag(2)
```

16.2 Indeksering i matricer

Hvis `A` er en matrix som ovenfor, så er `A[1,2]` elementet i første rækkes anden søjle, og `A[,1]` og `A[,2]` er første og anden søjle, og tilsvarende er `A[1,]` og `A[2,]` første og anden række:

```
> A[1,2]
[1] 3
> A[,1]
[1] 1 2
> A[1,]
[1] 1 3
```

Analogt med indeksering i vektorer (afsnit 14.2) kan man anvende en vektor som indeks til at udvælge rækker og/eller søjler. Herved kan man frit udtage delmatricer og bytte om på søjler eller rækker. Med `M1` defineret som ovenfor kan man udtrække hele række 3 og hele række 1 og få en ny matrix:

```
> M1[c(3,1),]
     [,1] [,2] [,3] [,4]
[1,]    3    6    9   12
[2,]    1    4    7   10
```

Eller man kan udtrække hele søjle 2 og hele søjle 4 og få en ny matrix:

```
> M1[,c(2,4)]
     [,1] [,2]
[1,]    4   10
[2,]    5   11
[3,]    6   12
```

Tilsvarende kan man udtrække række 1, søjle 2 og 4; eller række 1 og 2, søjle 3; eller række 1 og 2, søjle 3 og 4. I alle tilfælde er resultatet en ny matrix:

```
> M1[1, 2:4]
[1]  4  7 10
> M1[1:2, 3]
[1] 7 8
> M1[1:2, 3:4]
     [,1] [,2]
[1,]    7   10
[2,]    8   11
```

16.3 Opbygning af matricer med `cbind` og `rbind`

Funktionen `cbind` kombinerer vektorer eller matricer søjlevis (column) ved at "sætte dem ved siden af hinanden". Resultatet af `cbind(...)` er en matrix:

```
> A <- matrix(1:4, 2)
> v <- c(3,4)
> w <- c(5,6)

> cbind(v, w)
     v w
[1,] 3 5
[2,] 4 6
```

Funktion i R	Matematik	Betydning
`A + B`	$A+B$	`A` plus `B`, elementvis addition
`A - B`	$A-B$	`A` minus `B`, elementvis subtraktion
`A * B`	PAS PÅ	`A` gange `B`, elementvis multiplikation
`A %*% B`	AB	`A` gange `B`, matrixprodukt
`A / B`	PAS PÅ	`A` divideret med `B`, elementvis division
`A ^ y`	PAS PÅ	`A` opløftet i `y`, elementvis
`t(A)`	A'	den transponerede til `A`
`det(A)`	$\det(A)$, $\lvert A\rvert$	determinanten af `A`
`solve(A)`	A^{-1}	den inverse til `A`
`solve(A, v)`		find `u` der løser ligningen `A %*% u = v`
`eigen(A)`		egenvektorer og egenværdier for `A`
`A[i,j]`	A_{ij}	indeksering: `i`'te række, `j`'te søjle af A
`A[i,]`	$A_{i\cdot}$	indeksering: `i`'te række af A
`A[,j]`	$A_{\cdot j}$	indeksering: `j`'te søjle af A

Figur 30: Operatorer og funktioner på matricer og vektorer i R. Inden du bruger en operator mærket "PAS PÅ", så overvej om ikke der snarere er brug for fx `%*%` eller `solve(A)` eller matrixpotensopløftning.

```
> cbind(A, w)
     w
[1,] 1 3 5
[2,] 2 4 6
```

Funktionen `rbind` kombinerer vektorer eller matricer rækkevis (row) ved at "stable dem oven på hinanden". Resultatet af `rbind(...)` er en matrix:

```
> rbind(v, w)
  [,1] [,2]
v    3    4
w    5    6
> rbind(A,w)
  [,1] [,2]
     1    3
     2    4
w    5    6
```

Det er instruktivt at sammenligne med funktionen `c` fra afsnit 14.1, der sætter vektorer i forlængelse af hinanden. Resultatet af `c(...)` er en vektor:

```
> c(v, w)
[1] 3 4 5 6
```

16.4 Regning med matricer

Alle sædvanlige regneoperatorer på matricer findes også i R, men nogle af dem har uventede navne. Især kan det overraske at `A * B` ikke er matrixprodukt; det skal i stedet skrives `A %*% B`. Læg også mærke til at `A^-1` ikke beregner den inverse til matrix `A`; den skal beregnes med `solve(A)`. Nedenfor følger nogle eksempler på matrixoperationer.

Man kan definere B som den transponerede af A, altså A spejlet i diagonalen, således:

```
> B <- t(A)
> B
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Beregn resultatet af at addere matrix A og B:

```
> A + B
     [,1] [,2]
[1,]    2    5
[2,]    5    8
```

Beregn resultatet af at gange alle elementer af A med 7 (à la genbrugsreglen fra afsnit 14.3):

```
> 7 * A
     [,1] [,2]
[1,]    7   21
[2,]   14   28
```

Beregn resultatet af at lægge 7 til alle elementer af A (à la genbrugsreglen):

```
> 7 + A
     [,1] [,2]
[1,]    8   10
[2,]    9   11
```

Beregn resultatet af at gange matrix A med matrix B, som matrixprodukt:

```
> A %*% B
     [,1] [,2]
[1,]   10   14
[2,]   14   20
```

Dette viser at

$$AB = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 10 & 14 \\ 14 & 20 \end{pmatrix}$$

Tjek at det i almindelighed *ikke* er det samme som at gange matrix B med matrix A:

```
> B %*% A
     [,1] [,2]
[1,]    5   11
[2,]   11   25
```

Det er *slet ikke* det samme som `A * B`, der ganger A med B elementvis.

Beregn resultatet af at multiplicere matrix A med søjlevektoren (5, 6); det er en søjlevektor:

```
> A %*% c(5,6)
     [,1]
[1,]   23
[2,]   34
```

Dette viser at

$$A\begin{pmatrix} 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}\begin{pmatrix} 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 23 \\ 34 \end{pmatrix}$$

Beregn resultatet af at multiplicere rækkevektor (5, 6) med matrix A; det er en rækkevektor:

```
> c(5,6) %*% A
     [,1] [,2]
[1,]   17   39
```

Dette viser at

$$\begin{pmatrix} 5 & 6 \end{pmatrix} A = \begin{pmatrix} 5 & 6 \end{pmatrix}\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 17 & 39 \end{pmatrix}$$

Beregn prikproduktet eller indre produkt af vektorerne `c(1,2,3)` og `c(7,9,13)`:

```
> c(1,2,3) %*% c(7,9,13)
     [,1]
[1,]   64
```

Beregn determinanten af A:

```
> det(A)
[1] -2
```

Beregn den inverse til A, der i matematik ofte betegnes A^{-1}:

```
> solve(A)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

Beregn resultatet af at gange A med sin inverse, og omvendt. I begge tilfælde bør resultatet være enhedsmatricen, som det ses:

```
> A %*% solve(A)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> solve(A) %*% A
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Find den søjlevektor `u` der løser ligningen `A %*% u = v`, hvor `v = c(3, 4)` er defineret i afsnit 14.1:

```
> u <- solve(A, v)
> u
[1] 0 1
```

Tjek at `u` faktisk er en løsning ved at gange `A` med `u`; resultatet skal være `v`:

```
> A %*% u
     [,1]
[1,]    3
[2,]    4
```

Beregn egenvektorer og egenværdier for matrix A og gem dem i variablen `ev`:

```
> ev <- eigen(A)
> ev
$values
[1]  5.3722813 -0.3722813

$vectors
           [,1]       [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736
```

Resultatet `ev` indeholder dels egenværdierne `ev$values` og dels matricen af egenvektorer `ev$vectors`. Den første og anden egenvektor fås ved at udtrække første og anden søjle af denne matrix:

```
> ev1 <- ev$vectors[,1]
> ev2 <- ev$vectors[,2]
```

Eftersom `ev1` er en egenvektor for `A` og `ev$values[1]` er den tilhørende egenværdi, skal nedenstående to udregninger give samme resultat:

```
> A %*% ev1
          [,1]
[1,] -3.039462
[2,] -4.429794
> ev$values[1] * ev1
[1] -3.039462 -4.429794
```

Hvis A er en matrix og n er et naturligt tal, så betyder A^n opløftning af A til n'te potens, der er det samme som A matrixmultipliceret med sig selv n gange:

$$A^n = \underbrace{A \cdots A}_{n}$$

Der er ikke nogen indbygget funktion til dette formål i R; udtrykket `A^n` beregner noget helt andet som forklaret ovenfor. For mindre potenser, fx $n = 4$, kan man nemt beregne $R = A^n$ manuelt:

```
> R <- A
> R <- R %*% A
> R <- R %*% A
> R <- R %*% A
> R
     [,1] [,2]
[1,]  199  435
[2,]  290  634
```

Husk at man kan gentage en indtastning i R Console bare ved at bruge pil-op og Enter.

Mere bekvemt er det dog selv at definere en funktion `matpow(A, n)` til at beregne A^n som vist i afsnit 19.9.1, og derefter bruge den som enhver anden funktion:

```
> matpow(A, 4)
     [,1] [,2]
[1,]  199  435
[2,]  290  634
```

Når en matrix der har r rækker og k søjler multipliceres med en matrix der har k rækker og s søjler, så har resultatmatricen r rækker og s søjler. Ved matrixmultiplikationen `M1 %*% M2` med `M1` og `M2` som defineret i afsnit 16.1 ovenfor er $r = 3$, $k = 4$ og $s = 2$ og resultatet derfor en matrix med 3 rækker og 2 søjler:

```
> M1 %*% M2
     [,1] [,2]
[1,]   70  158
[2,]   80  184
[3,]   90  210
```

16.5 Eksempel: Fremskrivning med lineær afbildning (kaninpopulation)

Som eksempel vil vi lave fremskrivninger i modellen for en kaninpopulation givet i Anvendelseseksempel B.1 i *Noter om Matematik*. Modellen angiver hvordan en population af kaniner opdelt i unge individer x og gamle individer y udvikler sig fra år til år. Fra år t til åt $t+1$ er udviklingen som følger:

$$\begin{aligned} x_{t+1} &= 2.0 \cdot x_t + 1.5 \cdot y_t \\ y_{t+1} &= 0.7 \cdot x_t + 0.4 \cdot y_t \end{aligned}$$

Vi antager at i år 0 er der 100 unge og ingen gamle kaniner, dvs. at $x = 100$ og $y = 0$. Vi vil gerne vide hvordan populationen ser ud efter 20 år. Vi opskriver modellen på matrixform således:

$$\mathbf{v}_t = \begin{pmatrix} x_t \\ y_t \end{pmatrix}, \quad \mathbf{M} = \begin{pmatrix} 2.0 & 1.5 \\ 0.7 & 0.4 \end{pmatrix}, \quad \mathbf{v}_{t+1} = \mathbf{M}\mathbf{v}_t, \quad \mathbf{v}_0 = \begin{pmatrix} 100 \\ 0 \end{pmatrix}.$$

I R opskriver vi det som følger. Først oprettes matricen `M` (og vi kontrollerer at vi ikke har fået byttet om på rækker og søjler):

```
> M <- matrix(c(2.0, 0.7, 1.5, 0.4), 2)
> M
     [,1] [,2]
[1,]  2.0  1.5
[2,]  0.7  0.4
```

Herefter oprettes `v0` (svarende til $\mathbf{v}_0$) samt en variabel `v` som vi vil fremskrive i hvert trin og som indledningsvis er lig `v0`.

```
> v0 <- c(100,0)
> v <- v0
```

Nu er vi klar til at fremskrive og vi benytter en simpel `for`-løkke (afsnit 15) til dette:

```
> for(t in 1:20) { v <- M %*% v }
```

Efter løkken er resultatet ($\mathbf{v}_{20}$) i `v`:

```
> v
           [,1]
[1,] 7345918745
[2,] 2448639582
```

Efter 20 år er der `sum(v) = 9794558327` kaniner, altså næsten 10 milliarder ...

Men hvad hvis vi også ønskede kaninpopulationen i de mellemliggende år og ikke kun $\mathbf{v}_{20}$? Så må vi gemme resultatet i en matrix hvor første søjle er populationen til $t = 0$ (altså $\mathbf{v}_0$), anden søjle populationen efter et år ($\mathbf{v}_1$) osv. ind til sidste søjle som er populationen efter tyve år ($\mathbf{v}_{20}$). Resultatet er dermed en 2×21 matrix som vi vil kalde $\mathbf{V}$:

$$\mathbf{V} = (\mathbf{v}_0 \mathbf{v}_1 \cdots \mathbf{v}_{20}) = \begin{pmatrix} x_0 & x_1 & \cdots & x_{20} \\ y_0 & y_1 & \cdots & y_{20} \end{pmatrix}.$$

I R gør vi nu indledningsvis fuldstændig som før og opretter matricen `M`, vektoren `v0` og variablen `v`. Endvidere oprettes matricen $\mathbf{V}$ med kun én søjle; resten af `V` vil blive klistret på, søjle for søjle, efterhånden som vi fremskriver.

```
> M <- matrix(c(2.0, 0.7, 1.5, 0.4), 2)
> v0 <- c(100,0)
> v <- v0
> V <- matrix(v0,2)
```

Nu bruger vi en lidt udvidet udgave af `for`-løkken fra før til selve fremskrivningerne:

```
> for(t in 1:20) { v <- M %*% v ; V <- cbind(V,v) }
```

Løkken gennemløbes 20 gange og hver gang beregnes først den nye population med `v <- M %*% v` og derefter klistres den på `V` som sidste søjle med `V <- cbind(V,v)`. De to tildelinger i løkkekroppen udføres i rækkefølge når de er adskilt med semikolon, se afsnit 19.2. Når løkken er færdig er `v` kaninpopulationen efter 20 år og `V` har fået tilføjet 20 søjler svarende til populationerne i hver af de 20 år. Vi kan nu vise resultatet:

```
> v
           [,1]
 [1,] 7345918745
 [2,] 2448639582
> V
     [,1] [,2] [,3]   [,4]    [,5]     [,6]      [,7]     [,8]      [,9]
[1,]  100  200  505 1262.0 3155.05 7887.620 19719.051 49297.63 123244.07
[2,]    0   70  168  420.7 1051.68 2629.207  6573.017 16432.54  41081.36
        [,10]    [,11]     [,12]   [,13]    [,14]    [,15]    [,16]     [,17]
[1,] 308110.2 770275.4 1925688.5 4814221 12035553 30088883 75222208 188055520
[2,] 102703.4 256758.5  641896.2 1604740  4011851 10029628 25074069  62685173
         [,18]      [,19]      [,20]      [,21]
[1,] 470138800 1175346999 2938367498 7345918745
[2,] 156712933  391782333  979455833 2448639582
```

17 Associationslister og datasæt

Associationslister og datasæt i R bruges til at samle forskelligartede data i ét objekt. De opstår typisk når man indlæser data fra en tekstfil (afsnit 10.1) eller som resultat af funktioner der returnerer sammensatte data, såsom de indbyggede funktioner til nulpunktsfinding, numerisk optimering og numerisk integration i afsnit 7–9, eller lineær regression i afsnit 13. Men man kan også selv konstruere associationslister og datasæt.

17.1 Associationslister, funktionen `list`

En *associationsliste* er en værdi der kan indeholde flere navngivne komponenter. En associationsliste oprettes med funktionen `list`. I dette eksempel har associationslisten `xx` to komponenter `t` og `y`, som begge er vektorer:

```
> xx <- list(t=c(1.0, 1.5, 2.0, 2.5, 3.0), y=c(1.4, 0.3, -1.5, -3.1, -4.8))
> xx
$t
[1] 1.0 1.5 2.0 2.5 3.0
$y
[1]  1.4  0.3 -1.5 -3.1 -4.8
```

Komponenterne i en associationsliste behøver ikke være vektorer, og behøver ikke have samme længde:

```
> blandet <- list(A=matrix(c(1, 2, 3, 4), 2), d=42.2)
```

Man kan få fat på de enkelte komponenter af en associationsliste med operatoren `$`, og man kan få oplyst navnene på en associationslistes komponenter med funktionen `names`:

```
> xx$t
[1] 1.0 1.5 2.0 2.5 3.0
> xx$y
[1]  1.4  0.3 -1.5 -3.1 -4.8
> names(xx)
[1] "t" "y"
```

Man kan også indeksere med den sædvanlige indeksoperator og navnene på tekstform som indeksværdier, eller med talværdier som indeks:

```
> xx["y"]
$y
[1]  1.4  0.3 -1.5 -3.1 -4.8
> xx[2]
$y
[1]  1.4  0.3 -1.5 -3.1 -4.8
```

Man kan ændre og tilføje elementer med tildelinger:

```
> xx$t <- 1:8
> xx$ny = 10
> xx
$t
[1] 1 2 3 4 5 6 7 8
$y
[1]  1.4  0.3 -1.5 -3.1 -4.8
$ny
[1] 10
```

17.2 Datasæt, funktionen `data.frame`

Et *datasæt* i R, også kaldet en *data frame*, er en associationsliste hvis komponenter er vektorer af samme længde. Alle funktioner (fx `$`, `names`) der virker på associationslister kan også bruges på datasæt. Et datasæt kan indlæses med funktionerne `read.table` og `read.csv` som vist i afsnit 10.1.1 og 10.1.2, eller oprettes direkte med funktionen `data.frame`. I modsætning til en associationsliste vises et datasæt på tabelform med en kolonne for hver komponent. Det er muligt netop fordi alle komponenterne er vektorer af samme længde:

```
> d <- data.frame(t=c(1.0, 1.5, 2.0, 2.5, 3.0), y=c(1.4, 0.3, -1.5, -3.1, -4.8))
> d
    t    y
1 1.0  1.4
2 1.5  0.3
3 2.0 -1.5
4 2.5 -3.1
5 3.0 -4.8
```

I modsætning til for en almindelig associationsliste får man en fejlmelding hvis man forsøger at indsætte en ny søjle som ikke har den rigtige længde:

```
> d$ny=1:8
Error in "$<-.data.frame"(‘*tmp*‘, "ny", value = c(1, 2, 3, 4, 5, 6, 7,  :
        replacement has 8 rows, data has 5
```

Man kan indeksere et datasæt med række og søjle, lige som en matrix (se afsnit 16.2). For eksempel:

```
> d[1,2]
[1] 1.4
> d[2:4,2]
[1]  0.3 -1.5 -3.1
> d[3,]
  t    y
3 2 -1.5
> d[1:3,]
    t    y
1 1.0  1.4
2 1.5  0.3
3 2.0 -1.5
```

Resultatet af en indeksering der giver mere end én søjle er et nyt datasæt. Ved at bruge vektorer af sandhedsværdier til at indeksere rækkerne med (afsnit 19.6) kan man på en elegant måde udtrække bestemte observationer fra et datasæt på. Her udtrækker vi de observationer, altså rækker, hvor $t > y$:

```
> d1 <- d[ d$t > d$y ,]
> d1
    t    y
2 1.5  0.3
3 2.0 -1.5
4 2.5 -3.1
5 3.0 -4.8
```

18 Plot af funktioner af 2 variable

R har mulighed for grafisk fremstilling af funktioner af 2 variable. Fælles for de muligheder, der vil blive præsenteret her, er at man først er nødt til at udregne z koordinaterne (funktionsværdierne) for et net af støttepunkter for det (x, y) område, funktionen skal tegnes for. For eksempel, hvis vi ønsker at tegne funktionen $z = f(x, y) = \cos(x)\sin(2y)$ for $x \in [0, 2\pi]$ og $y \in [-\pi, \pi]$ med 50×50 støttepunkter:

```
> x <- seq(0, 2*pi, len=50)
> y <- seq(-pi, pi, len=50)
> f <- function(x,y) { cos(x)*sin(2*y) }
> z <- outer(x, y, f)
```

Ovenstående kald af funktionen `outer` genererer en matrix af værdier med lige så mange rækker, som der er elementer i `x` og lige så mange søjler som der er elementer i `y`, hvor værdien på hver plads udregnes ved at anvende funktionen `f` på de tilsvarende pladser i `x` og `y`, altså `z[i,j]=f(x[i],y[j])`. Vi antager i det følgende at `x`, `y` og `z` er genereret som oven for.

Vi skal i afsnit 19.5 se et eksempel på hvordan man kan definere en funktion `overflade` der indpakker ovenstående datagenereringsopskrift og kaldet af plotfunktionen, så det hele bliver lidt mere behageligt at arbejde med.

18.1 3D overfladeplot af en funktion af to variable

For at lave et simpelt 3D plot af en funktion af 2 variable kan man bruge funktionen `persp`:

```
> persp(x, y, z)
```

Overfladen tegnes som et net af firkanter udspændt af støttepunkterne, hvor hvert støttepunkt angiver et "knudepunkt" i nettet. Med 50×50 støttepunkter bliver der et net af 49×49 firkanter.

Udseendet af plot genereret med `persp` kan selvfølgelig ændres. Desværre har man i R ikke mulighed for interaktivt at trække i plottet med musen for at ændre betragtningsvinkel, så det kan kræve en del eksperimenteren med gentagne kald af `persp` indtil man finder den korrekte vinkel.[6] Betragtningsvinkel angives med parametrene `phi`, der er højden i grader over (x, y) planen, og `theta`, der er drejningsvinklen omkring z-aksen. For eksempel:

```
> persp(x, y, z, phi=35, theta=30)
```

Afstanden til betragtningspunktet fra centrum af "kassen" med den tegnede overflade (og dermed graden af perspektivisk fortegning) kan reguleres med parameteren `r`. Som standard er `r`=$\sqrt{3}$; større værdier giver mindre fortegning og mindre værdier en kraftigere fortegning, `r=10` giver udmærkede resultater (se figur 31):

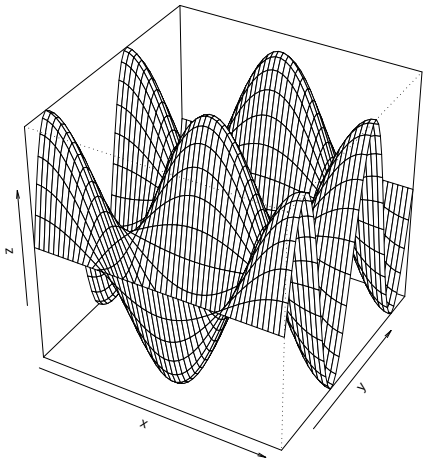
```
> persp(x, y, z, phi=35, theta=30, r=10)
```

Plottet skaleres som standard til at være i en kasse der er lige lang på alle sider. Hvis man ønsker at bevare det korrekte forhold mellem akserne skal man angive parameteren `scale=FALSE`:

```
> persp(x, y, z, phi=35, theta=30, scale=FALSE)
```

For at få enheder på akserne:

[6] Alt dette bliver gjort noget nemmere hvis man bruger funktionen `overflade` som kan hentes fra kursushjemmesiden; se appendix F.



Figur 31: Plot af $f(x,y) = \cos(x)\sin(2y)$ med `persp(x, y, z, phi=35, theta=30, r=10)`.

```
> persp(x, y, z, phi=35, theta=30, scale=FALSE, ticktype="detailed")
```

Det er muligt at farve overfladen ved at angive parameteren `col`:

```
> persp(x, y, z, phi=35, theta=30, scale=FALSE, col="red")
> persp(x, y, z, phi=35, theta=30, scale=FALSE, col=rgb(1,0.5,0))
```

Den sidste linie farver overfladen orange. Farven er angivet med dens koordinater i RGB-farverummet, hvor parametrene til funktionen `rgb` er intensiteten af rødt, grønt og blåt, hver angivet med et tal mellem 0 (ingen) og 1 (fuld intensitet). Man kan også angive værdien `NA` som farve, hvilket gør firkanterne "gennemsigtige". Man kan angive en sekvens af farver for at få en interessant flerfarvet effekt. Der er endda funktioner beregnet til at generere passende sekvenser af farver, for eksempel `rainbow`:

```
> persp(x, y, z, phi=35, theta=30, scale=FALSE, col="NA")
> persp(x, y, z, phi=35, theta=30, scale=FALSE, col=c("red","green","blue"))
> persp(x, y, z, phi=35, theta=30, scale=FALSE, col=rainbow(50))
```

Parameteren til `rainbow` angiver hvor mange farver der skal genereres og er med vilje her valgt så antallet af farver er 1 større end antallet af firkanter langs x-aksen. Eksperimentér eventuelt selv med værdier fra 47 til 52 for at se effekten.

Funktionen `persp` kan også farvelægge overfladen som om den belyses fra en given retning. Parameteren `shade` regulerer groft sagt hvor meget der skal lægges vægt på retningsbestemt lys og hvor meget der skal lægges vægt på diffust "baggrundslys", hvor `shade=0` er standard og svarer til 100% diffust lys og `shade=1` svarer til 100% retningsbestemt lys (værdier mellem 0.5 og 0.75 svarer nogenlunde til almindeligt dagslys).

```
> persp(x, y, z, phi=35, theta=30, scale=FALSE, shade=0.7)
> persp(x, y, z, phi=35, theta=30, scale=FALSE, shade=0.7, ltheta=110)
```

De to parametre `ltheta` og `lphi` angiver retning til det retningsbestemte lys, efter samme system som betragtningsvinklen angives med `theta` og `phi`.

Den sidste parameter til `persp` der skal nævnes her er parameteren `border` der bestemmer farven af det "net" der tegnes mellem støttepunkterne. Hvis man har mange støttepunkter kan linierne mellem dem næsten få overfladen til at forsvinde, så her kan man med fordel anvende `border=NA` for at slå linierne helt fra (dette virker bedst sammen med retningsbestemt belysning).

```
> persp(x, y, z, phi=35, theta=30, scale=FALSE, border="red")
> persp(x, y, z, phi=35, theta=30, scale=FALSE, shade=0.7, border=NA)
```

18.1.1 Avanceret farvning af overfladeplot

Hvis man vil have farven af overfladen i et `persp` plot til at afhænge af z værdi (højde) er man nødt til at angive en farve for hver firkant mellem støttepunkterne. For vores 50×50 støttepunkter i eksemplet er der 49×49 firkanter mellem dem. Det vil sige at som `col` parameter kan vi for eksempel angive en 49×49 matrix af farver. Lad os nu prøve at lave en matrix af farver der gør at de laveste værdier farves sorte og de højeste hvide. For at lave en grå farve kan man bruge funktionen `gray`, hvor `gray(0)` giver sort og `gray(1)` giver hvid og værdier af parameteren mellem 0 og 1 giver grå nuancer. Vi skal så gøre følgende:

1. Find niveauet for de 49×49 firkanter. Som tilnærmelse tager vi her bare værdien i det ene hjørne af hver firkant ved at tage de første 49 rækker og 49 søjler af `z` (se også opgave Dat-C-12).
2. Skaler værdierne således at de kommer til at ligge i intervallet 0 til 1.
3. Brug funktionen `gray` for at lave værdierne om til gråtonefarver.

```
> zv <- z[1:49,1:49]
> zv.skaleret <- (zv - min(zv)) / (max(zv) - min(zv))
> zcol <- gray(zv.skaleret)
> persp(x, y, z, phi=35, theta=30, scale=FALSE, col=zcol)
```

18.2 Plot med funktionen `image`

For at visualisere en funktion af to variable kan man i stedet for at tegne den i 3D med `persp` få tegnet den som et "kort" hvor funktionsværdierne (z-værdierne) afbildes over i en farveskala (for eksempel en gråtoneskala hvor laveste værdi svarer til sort og højeste til hvid). Lige som med `persp` kræver det en matrix af på forhånd udregnede funktionsværdier, z, samt vektorer x og y svarende til det anvendte net af støttepunkter.

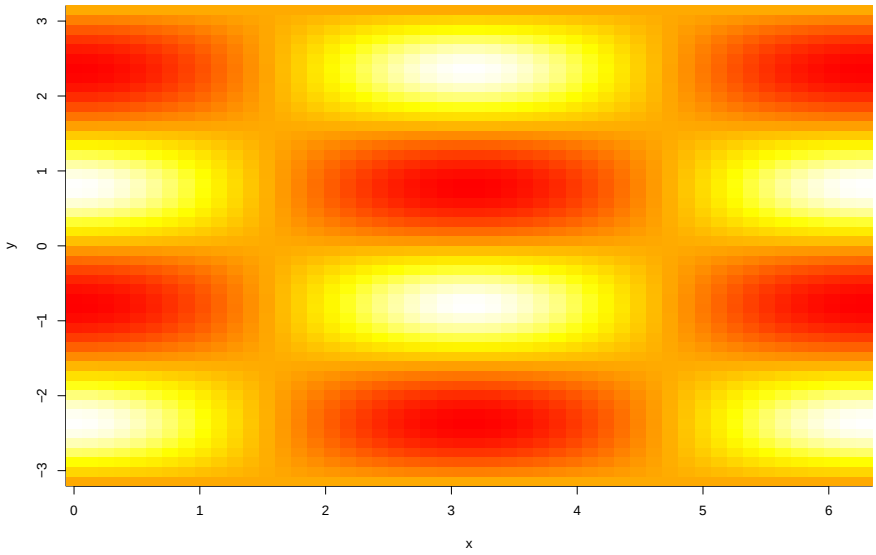
Funktionen, der tegner "kortet" hedder `image`:

```
> image(x, y, z)
```

Som standard afbildes funktionsværdierne med en "varme"-farveskala hvor de lave værdier bliver røde og de høje bliver hvide, med nuancer af orange og gult indimellem. Standardskalaen består af kun 12 farver, så det første man kan gøre for at få et bedre plot er at sætte antallet af farver i skalaen op:

```
> image(x, y, z, col=heat.colors(256))
```

Ovenstående anvender samme "varme"-farveskala som standardkaldet, men nu med 256 forskellige nuancer i skalaen. De 256 er valgt fordi farver i R (og i standard computer grafik i det hele taget) repræsenteres i et farverum af 256 nuancer af hver af komponenterne rød, grøn og blå, så med en lineær farveskala som den der produceres af `heat.colors` kan opdelingen ikke gøres finere.



Figur 32: Plot af $f(x,y) = \cos(x)\sin(2y)$ med `image(x, y, z, col=heat.colors(256))`.

Med parameteren `col` angives farveskalaen, der skal bruges, som en vektor af farvekoder. Sådanne vektorer er besværlige at angive manuelt, så man vil oftest anvende funktioner, der genererer vektoren af koder, som for eksempel `heat.colors`. Funktionen `rainbow` som blev brugt i nogle af eksemplerne med `persp` er et andet eksempel på en farveskalagenererende funktion, men den er begrænset nyttig til `image` da den som standard genererer en cyklisk liste af farver, hvilket bevirker at de højeste og laveste værdier får samme farve!

Man kan selv generere sin liste af farver, hvilket for eksempel er nødvendigt hvis man vil have en gråtoneskala hvor sort er lavt og hvidt er højt:

```
> image(x, y, z, col=gray(seq(0,1,len=256)))
```

I ovenstående genereres farverne ved at anvende funktionen `gray` (beskrevet i afsnit 18.1.1 om `persp`) på en vektor af 256 tal gående fra 0 til 1.

Parametre til at regulere titel, koordinatakser, skalering osv. er lige som for `plot`; se afsnit 5.

18.3 Plot af niveaukurver

En tredie måde at visualisere en funktion af to variable på er med niveaukurver (som højdekurver på et kort). Funktionen, der gør dette, hedder `contour`:

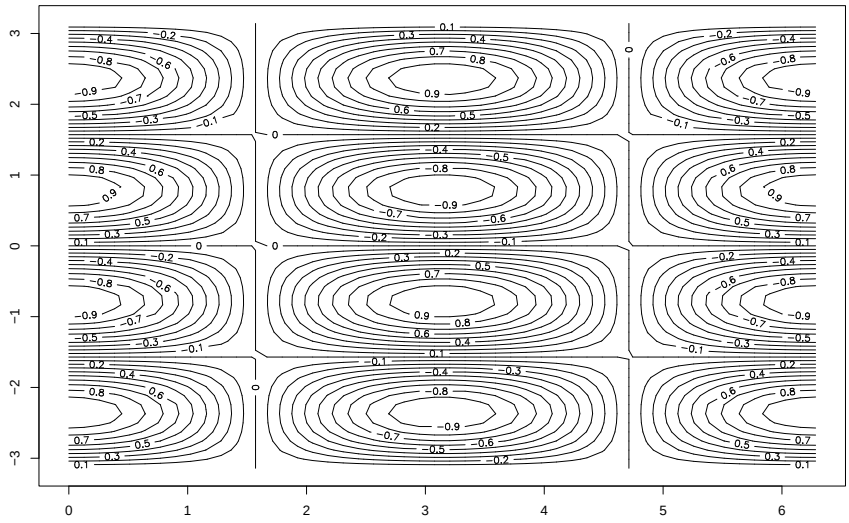
```
> contour(x, y, z)
```

Som standard tegnes 10 niveaukurver, men man kan angive hvor mange man ønsker med parameteren `nlevels`:

```
> contour(x, y, z, nlevels=15)
```

Man kan også eksplicit angive en vektor af de niveauer, der skal tegnes kurver for:

```
> contour(x, y, z, levels=c(-0.95,-0.75,-0.5,0,0.5,0.75,0.95))
```



Figur 33: Niveaukurveplot af $f(x, y) = \cos(x)\sin(2y)$ med `contour(x, y, z, nlevels=15)`.

Det er muligt at regulere farve af kurverne med parameteren `col` og placering af niveauangivelserne på kurverne med parameteren `method`:

```
> contour(x, y, z, method="edge", col="red")
```

Som `method` kan man angive `"simple"` (niveauangivelser i kanten af plottet, oven på konturerne), `"edge"` (niveauangivelser i kanten af plottet "indsat" i konturerne) eller `"flattest"` (standard, niveauangivelser indsat i den fladeste del af konturerne uden at de overlapper).

Hvis man slet ikke ønsker tekst på niveaukurverne kan man angive `drawlabels=FALSE`. Ud over det her nævnte kan man anvende mange af de samme parametre som til `plot` funktionen.

Man kan kombinere niveaukurveplot lavet af `contour` med et `image` plot som følger:

```
> image(x, y, z)
> contour(x, y, z, add=TRUE)
```

Dette virker bedst for et stort antal støttepunkter (for eksempel 500 gange 500) og et stort antal farver i `image` (for eksempel 256).

19 Programmering med R

Når den samme type beregning skal udføres mange gange, eller man for eksempel skal lave mange beslægtede grafer, kan man gøre det lettere for sig selv ved at definere nogle passende funktioner i R. Det er muligt at opbygge meget avancerede funktioner med fleksibilitet i den måde de kaldes på og med indre opbygning der kan gentage beregninger og/eller opbygge resultater trin for trin, og meget andet. R sproget har kort sagt faciliteter for programmering. De mest umiddelbart anvendelige af disse faciliteter vil blive beskrevet i det følgende, men først er det nødvendigt at tage et nærmere og lidt mere formelt kig på noget af alt det, der allerede er introduceret.

19.1 Udtryk, værdier og sideeffekter

De grundlæggende byggesten i R sproget er *udtryk*. Alt man kan taste ind er udtryk og ethvert udtryk giver en *værdi* når det beregnes. Vi har set talrige eksempler på udtryk i det foregående, især regneudtryk og funktionskald, men også funktionsdefinitioner og binding af værdier til variable og sågar `for`-løkker er faktisk udtryk. Altså har hver eneste linie i eksemplerne på R sproget indtil nu været udtryk.

Ud over at et udtryk ved beregning giver en værdi kan beregningen (og dermed udtrykket) have en *sideeffekt*. En sideeffekt er noget der ændrer tilstanden af "verden" uden for udtrykket, for eksempel ved at binde en værdi til en variabel (hviket har betydning for beregning af senere udtryk hvor variablen indgår) eller ved at tegne en graf (hvilket ændrer indholdet af grafvinduet eller skriver grafen i en fil på computerens disk).

19.1.1 Tildelingsudtryk og "usynlige" værdier

En binding af en værdi til en variabel, såsom `x <- 2`, kaldes en *tildeling* og er et udtryk. Værdien af et tildelingsudtryk er variablens nye værdi. Således kunne man bruge værdien direkte i en videre beregning, altså ved at have tildelingen som et deludtryk i et større udtryk. Dette kan dog absolut ikke anbefales, da det gør udtrykket meget svært at forstå. I en enkelt type udtryk kan det dog være i orden at bruge tildeling som et deludtryk, nemlig ved tildeling af samme værdi til flere variable:

```
> y <- x <- 2
```

Dette binder værdien 2 til både `x` og `y`.

Værdien af et tildelingsudtryk vises ikke i R Console. For eksempel giver nedenstående ikke noget direkte "svar" når det tastes ind i R Console:

```
> x <- 2
```

At en tildeling vitterlig er et udtryk som alle andre og har en værdi kan man overbevise sig om ved at sætte en parentes rundt om den:

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

Nu bliver værdien vist i R Console. Et tildelingsudtryk er et eksempel på et udtryk, hvis værdi er gjort "usynlig" i R Console (fordi man som hovedregel ikke er interesseret i at se den). Et andet eksempel på et udtryk med en usynlig værdi er en `for`-løkke, hvis værdi er værdien af den sidste evaluering af løkkens krop. Mange funktioner i R skjuler deres værdi på denne måde, nemlig de funktioner der ikke er funktioner i matematisk forstand men snarere er *procedurer* der "gør noget" og altså kaldes for deres sideeffekters skyld. Et eksempel er funktionen `plot`, der tegner en graf. Hvis vi gerne vil se værdien af et kald af `plot` kan vi lige som med en tildeling pakke udtrykket ind i en parentes:

```
> (plot(sin,0,6*pi))
NULL
```

Dette giver værdien `NULL` (som er en særlig værdi der betyder "ingenting"), og tegner samtidig en graf. Det er ikke kun funktioner som giver `NULL` der skjuler deres værdi. Funktionen `persp` (afsnit 18.1) giver for eksempel en 4×4 matrix der kan bruges til at projicere 3D punkter ind i det tegnede koordinatsystem (se hjælpeteksten for `?persp` hvis dette har interesse).

19.2 Sekvenser og blokke af udtryk

Hidtil har (næsten) alle indtastninger i disse eksempler været af en form hvor én linie svarer til ét udtryk. Vi har således afsluttet alle udtryk med linieskift. Man kan imidlertid afslutte et udtryk med semikolon (`;`) i stedet og dermed have flere udtryk på samme linie:

```
> 1+2; 2^5; sqrt(2)
[1] 3
[1] 32
[1] 1.414214
```

Dette svarer fuldstændig til at indtaste de tre regneudtryk på hver sin linie; R viser de tre resultater. Man kan altid afslutte et udtryk med et semikolon, uanset om det også står sidst på linien, så nedenstående er for eksempel også en lovlig indtastning:

```
> 1+2;
[1] 3
```

Semikolon og linieskift er en lille smule forskellige i den måde, de afslutter et udtryk på. Et semikolon afslutter altid et udtryk, og udtrykket før et semikolon skal derfor altid være "lukket"; det skal være et komplet udtryk. Hvis et udtryk foran et semikolon er "åbent" er det en fejl (for eksempel hvis man har glemt en slutparentes). Et linieskift afslutter kun et udtryk hvis udtrykket er lukket. Hvis udtrykket er åbent ved slutningen af linien fortsætter udtrykket simpelthen på næste linie. Ved indtastning i R Console skifter prompten fra `>` til `+` hvis linien fortsætter et åbent udtryk fra den foregående linie. Et eksempel, hvor udtrykket `(1+2-3)` deles over to linier:

```
> (1+2
+ -3)
[1] 0
```

Et andet eksempel på deling af samme udtryk:

```
> 1+2-
+ 3
[1] 0
```

I begge tilfælde er udtrykket åbent efter første linie. I første eksempel er det åbent fordi parentesen ikke er afsluttet, i andet eksempel fordi der er nødt til at følge en operand efter minustegnet.

Man kan samle en sekvens af udtryk i en `blok` ved at omslutte den med krølleparenteser ({) og (}):

```
> { 1+2; 2^5; sqrt(2) }
[1] 1.414214
```

Som det ses er værdien af sådan et *blokudtryk* simpelthen værdien af sidste udtryk. I ovenstående beregnes de to første udtryk såvel som det tredie, men værdierne af dem smides væk, hvorfor det lige i dette eksempel er meningsløst overhovedet at have dem med. Det giver imidlertid mening at have flere udtryk i en blok hvis de (eventuelt på nær det sidste) har sideeffekter som for eksempel at binde værdier til variable eller tegne grafer:

```
> { x<-2; y<-3; x+y }
[1] 5
> x
[1] 2
```

Dette tildeler værdier til `x` og `y` og giver endelig en værdi hvor de to variable indgår i beregningen. Bemærk at de to variable har deres nye værdier (2 og 3) også bagefter, uden for blokken.

En blok kan lige som andre udtryk fortsættes over flere linier. For eksempel:

```
> { x<-2
+   y<-3
+   x+y }
[1] 5
```

Bemærk at de første to linieskift afslutter de to tildelingsudtryk men ikke selve blokken, der jo skal afsluttes af med højre krølleparentes (}). Dermed kan semikolonerne mellem udtrykkene udelades. Eller sagt på en anden måde: et lukket (del)udtryk afsluttes af et linieskift, også inden i en blok[7]. Det er imidlertid en *meget god vane* altid at afslutte sine udtryk inde i en blok med semikolon, også selv om man har linieskift. Så risikerer man ikke uforvarende at komme til at efterlade et åbent udtryk der fortsætter på næste linie. Hvis man for eksempel havde glemt tallet 2 på første linie i blokken ovenfor havde `x` fået værdien 3. Så det var bedre at skrive ovenstående blok som følger:

```
> { x<-2 ;
+   y<-3 ;
+   x+y ; }
[1] 5
```

19.3 Funktionsdefinitioner

En funktionsdefinition `function(...) ...` er et udtryk hvis værdi er en funktion. Vi har hidtil kun set eksempler på definition af *navngivne funktioner*, hvor funktionen bindes til en variabel og derefter bruges ved navn. Nogle gange hvor vi kun bruger en funktion en enkelt gang, for eksempel for at tegne en kurve i en graf, kunne vi ligeså godt have anvendt funktionen uden at navngive den (som *anonym funktion*):

```
> plot(function(x) { x^3 - 5*x^2 }, -2, 8)
```

Hidtil har vi for læselighedens skyld konsekvent angivet kroppen af en funktion som et udtryk i en krølle-parentes. En funktionskrop kan imidlertid være et vilkårligt udtryk, og således er alle nedenstående definitioner ækvivalente:

```
> f <- function(x) x^3 - 5*x^2
> f <- function(x) ( x^3 - 5*x^2 )
> f <- function(x) { x^3 - 5*x^2 }
```

[7] Det gælder dog ikke `if`-udtryk hvis de efterfølges af `else`, se afsnit 19.7.

19.4 Kommentarer

Man kan angive kommentarer i sin R kode for at gøre den lettere at læse og forstå. Alt hvad der følger efter et #-tegn på en linie er en kommentar og ignoreres af R. For eksempel:

```
> 1:40   # giver mig tallene fra 1 til 40
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
> # Denne linie er kun en kommentar.
```

19.5 Mere avancerede funktionsdefinitioner

Som eksempel på definition af en funktion af procedural karakter vil vi se på hvordan det kan gøres smidigere at lave et overfladeplot af en funktion af 2 variable. Undervejs i dette afsnit vil vi se gradvist mere og mere avancerede definitioner af en funktion `overflade` til at tegne 3D overfladeplot. Den "endelige" udgave af `overflade` kan hentes fra kursushjemmesiden og med fordel bruges til nemt at lave overfladeplot, se appendiks F.

Givet at vi har defineret en matematisk funktion `f(x,y)` skal vi som beskrevet i afsnit 18 igennem en række trin for at få grafen tegnet:

```
> x <- seq(0, 2, len=50)        # 50 støttepunkter i intervallet [0,2]
> y <- seq(-1, 1, len=50)       # 50 støttepunkter i intervallet [-1,1]
> z <- outer(x, y, f)           # Funktionsværdier i støttepunkterne
> persp(x, y, z)                # Tegn grafen
```

Hvis vi ønsker at få tegnet grafen for andre intervaller, for et finere net af støttepunkter eller for en anden funktion end `f`, skal vi gentage flere af indtastningerne ovenfor med ændringer. Hvis det er noget vi skal gøre mere end nogle få gange kan det betale sig at definere en funktion der udfører alle ovenstående trin.

Vi definerer nu en funktion `overflade` der som argument tager en matematisk funktion af to variable samt intervaller for de to funktionsvariable og tegner en 3D graf for funktionen med `persp`:

```
> overflade <- function(f, x.min, x.max, y.min, y.max) {
+    x <- seq(x.min, x.max, len=50);    # 50 x-støttepunkter
+    y <- seq(y.min, y.max, len=50);    # 50 y-støttepunkter
+    z <- outer(x, y, f);               # Funktionsværdier i støttepunkterne
+    persp(x, y, z);                    # Tegn grafen
+ }
```

Bemærk at de tre tildelinger til `x`, `y` og `z` *ikke* ændrer på eventuelle bindinger til disse variabelnavne uden for funktionen. Variable, der tildeles en værdi i en funktionskrop, er *lokale variable* i funktionen. Lokale variable kendes ikke uden for funktionen, de er defineret i, og tildeling til dem ændrer ikke ved eventuelle ikke-lokale variable, der måtte hedde det samme.

Nu kan vi nemt få tegnet forskellige funktioner af to variable for forskellige intervaller:

```
> f <- function(x,y) { cos(x)*sin(2*y) }
> overflade(f, 0, 2, -1, 1)
> overflade(f, 0, 10, -5, 5)
> overflade(function(x,y) { x^2+cos(x*y) }, 0, 2, 0, 2*pi)
```

Ovenstående første forsøg på at definere funktionen `overflade` tillader os ikke at ændre antal støttepunkter. Vi udvider derfor funktionen lidt med en ny parameter `n`:

```
> overflade <- function(f, x.min, x.max, y.min, y.max, n) {
+    x <- seq(x.min, x.max, len=n);     # n x-støttepunkter
+    y <- seq(y.min, y.max, len=n);     # n y-støttepunkter
+    z <- outer(x, y, f);               # Funktionsværdier i støttepunkterne
+    persp(x, y, z);                    # Tegn grafen
+ }
```

Med ovenstående nye udgave af `overflade` kan vi nu skrive:

```
> overflade(f, 0, 2, -1, 1, 50)
> overflade(f, 0, 10, -5, 5, 200)
> overflade(function(x,y) { x^2+cos(x*y) }, 0, 2, 0, 2*pi, 100)
```

Der er efterhånden lidt mange parametre man skal huske at give til `overflade` for at få en graf (og vi vil indføre endnu flere senere). R har imidlertid mulighed for at angive *standardværdier* for parametrene i en funktionsdefinition. Hvis vi nu vælger, at intervallerne som standard skal være $[0, 1]$ og antal støttepunkter som standard skal være 50 kan vi definere funktionen som følger:

```
> overflade <- function(f, x.min=0, x.max=1, y.min=0, y.max=1, n=50) {
+    x <- seq(x.min, x.max, len=n);     # n x-støttepunkter
+    y <- seq(y.min, y.max, len=n);     # n y-støttepunkter
+    z <- outer(x, y, f);               # Funktionsværdier i støttepunkterne
+    persp(x, y, z);                    # Tegn grafen
+ }
```

Bemærk, at det kun er i selve listen af formelle parametre vi har ændret; funktionskroppen er den samme som før. De parametre, der er standardværdier for, kan man så udelade når man kalder funktionen:

```
> overflade(f)
> overflade(f, 0, 10, -5, 5)
```

Det første funktionskald ovenfor tegner `f` for intervallerne $x \in [0, 1]$ og $y \in [0, 1]$ (givet af standardværdierne) mens det andet tegner `f` for $x \in [0, 10]$ og $y \in [-5, 5]$.

Hvis nu vi ønsker at angive antal støttepunkter (som er sjette parameter) men ikke ønsker at angive intervalgrænserne (anden til femte parameter) kan vi kalde `overflade` med angivelse af antal støttepunkter som en *navngiven* parameter:

```
> overflade(f, n=100)
```

Tilsvarende kunne vi skrive

```
> overflade(f, x.max=10, y.max=10)
```

eller endda bytte om på rækkefølgen af parametrene ved at navngive dem alle:

```
> overflade(n=100, x.max=5, y.max=5, f=function(x,y) { x^2+cos(x*y) })
```

Vi har tidligere i set talrige eksempler på brug af navngivne parametre når vi brugte indbyggede funktioner i R, for eksempel `plot` med dens myriader af forskellige parametre til at styre akser, tekster, stregtykkelser, farver osv. Mekanismen med at have standardværdier for de fleste parametre og angive navngive parametre når man bruger funktionerne kan gøre selv funktioner med dusinvis af parametre anvendelige.

Udtryk for standardværdier for de formelle parametre i en funktionsdefinition kan være vilkårligt komplekse og kan referere til andre af de formelle parametre. Hvis vi eksempelvis synes at man til `overflade` burde have mulighed for at angive intervalgrænserne som vektorer `c`(*min*,*max*) som alternativ til `x.min`, `x.max`, `y.min` og `y.max` og hvis vi gerne ville give mulighed for forskelligt antal støttepunkter på de to akser kunne det gøres således:

```
> overflade <- function(f, x.min=min(x.interval), x.max=max(x.interval),
+                       y.min=min(y.interval), y.max=max(y.interval),
+                       n=50, x.n=n, y.n=n,
+                       x.interval=c(0,1), y.interval=c(0,1) ) {
+    x <- seq(x.min, x.max, len=x.n);
+    y <- seq(y.min, y.max, len=y.n);
+    z <- outer(x, y, f);
+    persp(x, y, z);
+ }
```

Bemærk at de to nye parametre `x.n` og `y.n` som standardværdi har parameteren `n`. De nye intervalparametre `x.interval` og `y.interval` bruges nu til at give standardværdierne for de "gamle" parametre `x.min`, `x.max`, `y.min` og `y.max`. Grunden til at vi fx i standardværdien for `x.min` anvender `min(x.interval)` i stedet for `x.interval[1]` er at det tillader brugeren at angive `x.interval` og `y.interval` i lidt forskellige former (for eksempel størst før mindst eller som en vektor med mere end to elementer).

Eksempler på brug af den nye udgave af `overflade`:

```
> overflade(f, x.max=10, y.interval=c(-5,5) )
> overflade(function(x,y) { x^2+cos(x*y) }, 0, 2, 0, 2*pi, y.n=100)
```

Hvis vi ville være endnu mere fleksible kunne vi flytte de lokale variable `x` og `y` op som formelle parametre således at brugeren kunne angive vektorer med støttepunkterne direkte (hvis de ikke ønskes jævnt fordelt over intervallet):

```
> overflade <- function(f, x.min=min(x.interval), x.max=max(x.interval),
+                       y.min=min(y.interval), y.max=max(y.interval),
+                       n=50, x.n=n, y.n=n,
+                       x.interval=c(0,1), y.interval=c(0,1),
+                       x=seq(x.min, x.max, len=x.n),
+                       y=seq(y.min, y.max, len=y.n) ) {
+    z <- outer(x, y, f);
+    persp(x, y, z);
+ }
```

Det ville dog være en dårlig idé at flytte `z` op som formel parameter på en tilsvarende måde.

Et enkelt eksempel på brug af den nye definition:

```
> overflade(function(x,y) { x^2+cos(x*y) }, y.max=2*pi, x=c((0:10)/5,3:5))
```

I ovenstående er `x` koordinaten for støttepunkterne angivet til $0, \frac{1}{5}, \frac{2}{5}, \ldots, 2, 3, 4, 5$, altså med større tæthed for værdier af `x` under 2 end for værdier over 2.

Ved at pakke `persp` ind i `overflade` har vi nu gjort det nemt at tegne en graf for en funktion af to variable ved at generere støttepunkterne som `persp` skal bruge. Til gengæld har vi mistet alle de muligheder som `persp` giver for at regulere betragtningsvinkel, farver, koordinatakser og så videre. Det er der imidlertid råd for. Man kan til sidst i listen af formelle parametre i en funktionsdefinition angive `...`, altså tre punktummer, hvilket betyder at funktionen accepterer et vilkårligt antal parametre ud over

Operator i R	Matematik	Betydning
`x == y`	$x = y$	x lig med y
`x != y`	$x \neq y$	x forskellig fra y
`x < y`	$x < y$	x mindre end y
`x > y`	$x > y$	x større end y
`x <= y`	$x \leq y$	x mindre end eller lig y
`x >= y`	$x \geq y$	x større end eller lig y

Figur 34: Sammenligningsoperatorer i R.

dem, der er specificeret. I funktionskroppen kan man så anvende (`...`) for at videregive parametrene til en anden funktion i et funktionskald. For at tillade at parametre til `persp` "passerer igennem" `overflade` kan vi skrive `overflade` som følger:

```
> overflade <- function(f, x.min=min(x.interval), x.max=max(x.interval),
+                       y.min=min(y.interval), y.max=max(y.interval),
+                       n=50, x.n=n, y.n=n,
+                       x.interval=c(0,1), y.interval=c(0,1),
+                       x=seq(x.min, x.max, len=x.n),
+                       y=seq(y.min, y.max, len=y.n),
+                       ... ) {
+    z <- outer(x, y, f);
+    persp(x, y, z, ...);
+ }
```

Nu kan vi bruge funktionen som følger:

```
> overflade(function(x,y) { x^2+cos(x*y) }, x.max=4, y.max=2*pi, col="red")
> overflade(f, 0, 10, 0, 10, col="blue", theta=30, phi=35)
```

I appendiks F vises en yderligere udbygget version af `overflade`.

19.6 Logiske udtryk

Logiske udtryk er udtryk der som resultat har en sandhedsværdi (altså sandt eller falsk; i R `TRUE` eller `FALSE`). Eksempler:

```
> x <- 3
> x > 5
[1] FALSE
> x < 5
[1] TRUE
```

"Mindre" end (<) og "større end" (>) er eksempler på *sammenligningsoperatorer*; de sammenligner to værdier og giver `TRUE` eller `FALSE`. Figur 34 giver en oversigt over sammenligningsoperatorerne i R.

Faktisk virker sammenligningsoperatorer lige som de aritmetiske operatorer på vektorer af værdier, med den sædvanlige genbrugsregel hvis vektoren på den ene side er kortere end på den anden:

```
> x <- 1:10
> x > 5
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> x == 3
 [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Operator i R	Matematik	Betydning
`!x`	$\neg x$	negation; “ikke x”
`x & y`	$x \wedge y$	x og y
`x | y`	$x \vee y$	x eller y
`x && y`		x og y, kun første element (se tekst)
`x || y`		x eller y, kun første element (se tekst)

Figur 35: Logiske operatorer.

En vektor af sandhedsværdier kan bruges til indeksering af en vektor. Det giver en ny vektor (eller matrix) med de elementer fra den indekserede vektor, hvor de tilsvarende pladser i indeksvektoren har værdien `TRUE`. Dette kan vi bruge sammen med logiske udtryk for at udtage elementer fra en vektor hvorom der gælder noget bestemt:

```
> x[x>5]
[1]  6  7  8  9 10
> x[x%%3==0]
[1] 3 6 9
```

Det sidste udtryk ovenfor giver de elementer af `x` hvis rest ved heltalsdivision med 3 er nul, altså de elementer, der er delelige med 3.

Logiske udtryk kan kombineres med *logiske operatorer*; se figur 35. Eksempler:

```
> x > 5 & x <= 7
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
> x <= 3 | x >= 7
 [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> x[!(x>2 & x<9) | x==5]
[1]  1  2  5  9 10
```

De to operatorer `&&` og `||` adskiller sig fra operatorerne `&` og `|` ved at de for vektorer af logiske værdier altid kun kigger på det første element og altid kun giver en enkelt sandhedsværdi som svar. For eksempel:

```
> x > 5 && x <= 7
[1] FALSE
> x <= 3 || x >= 7
[1] TRUE
```

Derudover er der den forskel at `&&` og `||` ikke evaluerer deres anden operand hvis det samlede udtryks sandhedsværdi kan afgøres alene ud fra den første operand; fx hvis deludtrykket x er sandt i udtrykket $x||y$ så evalueres deludtrykket y slet ikke. Dette kan have betydning hvis evalueringen af anden operand enten har en sideeffekt eller vil give en fejl. Dobbeltoperatorerne `&&` og `||` anvendes oftest sammen med `if` ... `else` udtryk som omtales i afsnit 19.7.

19.7 Betingede udtryk: `if` ... `else`

Et `if` ... `else` udtryk har følgende form:

`if` (*betingelse*) *udtryk1* `else` *udtryk2*

Effekt: Udtrykket *betingelse* evalueres. Hvis det giver `TRUE` evalueres *udtryk1* (kaldet `if`-grenen), ellers evalueres *udtryk2* (kaldet `else`-grenen). Værdien af det samlede `if` ... `else` udtryk er værdien af den af de to grene, der evalueres. Eksempler:

```
> x <- 5
> if (x<2) 2 else x
[1] 5
> if (x>2) 2 else x
[1] 2
```

Man kan udelade `else`-grenen:

```
> if (x<2) 2
> if (x>2) 2
[1] 2
```

Man skal derfor passe på hvis man (uden for en blok) deler et `if` ... `else` udtryk over flere linier, idet udtrykket uden `else` i sig selv er et gyldigt lukket udtryk:

```
> if (x<2) 2
> else x
Error: syntax error in "else"
> if (x<2) 2 else
+ x
[1] 5
```

Det er ofte en god idé at lade de to grenudtryk være blokke i stedet for simple udtryk, så skal man bare være sikker på at have `else` på samme linie som den afsluttende krøllede parentes fra `if`-grenens blok:

```
> if (x<2) {
+   2
+ } else {
+   x
+ }
[1] 5
```

Bemærk at ovenstående kun gælder uden for blokke. Inden i en blok afslutter linieskift *ikke* et `if`-udtryk hvis der følger en `else`-gren umiddelbart efter:

```
> { if (x<2) 2
+   else x }
[1] 5
```

19.7.1 Eksempel: Fakultetsfunktionen

Som eksempel på anvendelse af et `if` ... `else` udtryk vil vi implementere fakultetsfunktionen $x!$, defineret for naturlige tal x som

$$x! = \begin{cases} 1, & \text{hvis } x = 1, \\ x \cdot (x-1)! & \text{ellers.} \end{cases}$$

Bemærk at definitionen er *rekursiv*: $x!$ er defineret ved hjælp af $(x-1)!$, så fakultetsfunktionen er defineret ved hjælp af sig selv. Vi følger definitionen nøje og implementerer funktionen `fak` rekursivt:

```
> fak <- function(x) { if (x==1) 1 else x*fak(x-1) }
> fak(4)
[1] 24
```

For en god ordens skyld skal vi nævne at R allerede har en indbygget fakultetsfunktion; den hedder `factorial`.

19.7.2 Eksempel: Funktion der giver Fibonacci-tal

Vi vil konstruere en funktion der kan give os Fibonacci-tal nummer n for heltal $n \geq 1$. Til beregningen vil vi benyttes os af fremgangsmåden fra afsnit 15.2 hvor vi beregnede Fibonacci-tal ved hjælp af en `for`-løkke; dér er det også forklaret hvad Fibonacci-tal overhovedet er.

For at funktionen skal svare i brugen til de indbyggede R-funktioner skal den også kunne bruges når n er en vektor, dvs. at for eksempel `fibonacci(c(1,5,7))` skal give en vektor med det første, femte og syvende Fibonacci-tal.

Her er så en R-funktion `fibonacci`, der for heltal $n \geq 1$ giver Fibonacci-tal nummer n:

```
> fibonacci <- function(n) {  # Fibonacci-tal for n>0.
+   max.n <- max(n) ;         # Hvor mange fibonacci-tal skal bruges?
+   fib <- rep(1, max.n) ;    # Lav en vektor med plads til max.n tal.
+   if ( max.n > 2 ) {
+     for ( i in 3:max.n ) {
+        fib[i] <- fib[i-1] + fib[i-2] ; # Udregn Fibonacci-tal nummer i.
+     }
+   }
+   fib[n] ;                  # Returner de(t) n'te Fibonacci-tal.
+ }
```

Princippet i funktionen er at udregne Fibonacci-tallene fra nummer 1 og op til de(t) Fibonacci-tal, der skal beregnes. Undervejs gemmes Fibonacci-tallene i en vektor `fib` således at `fib[1]` er Fibonacci-tal nummer 1, `fib[2]` er Fibonacci-tal nummer 2, og så videre. Resultatet af funktionskaldet er så `fib[n]`, som er en vektor hvis `n` er en vektor.

Virkemåden i detaljer: Først bestemmes hvor mange Fibonacci-tal, det er nødvendigt at beregne, nemlig det samme som det største tal i vektoren `n`. (Husk at et tal i R repræsenteres som en vektor med ét element, så dette virker også hvis `n` bare er et enkelt tal). Dette største tal gemmes i variablen `max.n`. Så sættes `fib` til at være en vektor med `max.n` elementer som alle er 1. Da Fibonacci-tal nummer 1 og 2 begge er 1 er `fib[1]` og `fib[2]` hermed korrekte. Kun hvis Fibonacci-tal nummer 3 eller højere skal udregnes udføres `for`-løkken for at beregne tallene. Løkkekroppen udføres for hvert tal `i` mellem 3 og `max.n` og beregner `fib[i]` som summen af de to foregående tal i `fib`. Til slut udtages de(n) ønskede værdi(er) fra `fib` med indekseringen `fib[n]`.

Eksempler på brug:

```
> fibonacci(6)
[1] 8
> fibonacci(1:10)
[1]  1  1  2  3  5  8 13 21 34 55
```

19.8 Betingede udtryk: `ifelse`

Et `if ... else` udtryk som beskrevet i det foregående er beregnet til at operere på kun en enkelt sandhedsværdi i betingelsen. Hvis betingelsen giver en vektor med mere end én sandhedsværdi vil kun det første element blive brugt og R vil ydermere give en advarsel. Dette betyder at funktionen `fak` fra afsnit 19.7.1 i modsætning til R's indbyggede funktioner *ikke* virker på en vektor med mere end ét tal: Dels vil der blive genereret en masse advarsler fordi betingelsen `(x==1)` giver et vektorresultat når `x` er en vektor, dels vil det alene være det første tal i argumentvektoren der vil komme til at styre antallet af multiplikationer. Det betyder at `fak(1:10)` simpelthen giver tallet `1`, og `fak(c(3,4))` giver vektoren `c(6,12)`, hvilket kun er korrekt i første element.

For at lave en implementation af fakultetsfunktionen der virker på vektorer af tal lige som R's interne funktioner kan man i stedet benytte funktionen `ifelse`. Det er en funktion der tager tre (vektor) argumenter:

`ifelse(` *v1* `,` *v2* `,` *v3* `)`

Det første argument *v1* skal være vektor af logiske værdier og det andet og tredie to vektorer af samme længde som *v1* (omend den sædvanlige genbrugsregel gælder hvis den ene er for kort). Resultatet er en vektor af samme længde som *v1* sammensat af elementer fra *v2* og *v3* således at hvor *v1* er `TRUE` tages elementer fra *v2* og hvor *v1* er `FALSE` tages elementer fra *v3*. Eksempel:

```
> x <- 1:10
> ifelse(x>5,x,10-x)
 [1]  9  8  7  6  5  6  7  8  9 10
```

Hvis betingelsen *v1* er `TRUE` i alle elementer evalueres *v3* slet ikke. Tilsvarende vil *v2* ikke blive evalueret hvis *v1* er *FALSE* i alle elementer. Kun hvis *v1* har en blanding af sande og falske elementer evalueres både *v2* og *v3*.

Vi kan nu implementere `fak` på en bedre måde:

```
> fak <- function(x) { ifelse(x<=1, 1, x*fak(x-1)) }
> fak(1:5)
[1]   1   2   6  24 120
```

Bemærk at vi her brugte betingelsen `x<=1` i stedet for betingelsen `x==1`. Dette er fordi at på et tidspunkt i evalueringen vil der være nogle af elementerne i argumentvektoren `x-1` i det rekursive kald af `fak` der bliver mindre end 1. For eksempel når R evaluerer `fak(1:5)` resulterer det reelt i følgende kald af `fak`:

```
fak(c(1,2,3,4,5))
  fak(c(0,1,2,3,4))
    fak(c(-1,0,1,2,3))
      fak(c(-2,-1,0,1,2))
         fak(c(-3,-2,-1,0,1))
```

Med betingelsen `x==1` ville dette fortsætte i det uendelige (eller rettere: indtil R-systemet løb tør for computerhukommelse) idet der altid vil være mindst ét element i argumentvektoren der er forskelligt fra 1. Med betingelsen `x<=1` vil på et tidspunkt alle elementerne i argumentvektoren opfylde betingelsen og dermed stopper de rekursive kald af `fak`, idet det første argument til `ifelse` så er `TRUE` i alle elementer.

19.9 `while`-løkker

I afsnit 15 beskrev vi iteration over elementerne i en vektor med `for`-løkker. `For`-løkker er nemme at bruge når noget skal gentages et bestemt antal gange, man kender på forhånd. En mere generel form for løkke er `while`-løkken. Den kan bruges til at foretage samme operation gentagne gange sålænge en given betingelse er opfyldt – dvs. antallet af gentagelser behøver ikke at være kendt på forhånd. Et `while` udtryk har følgende form:

`while` (*betingelse*) *udtryk*

Effekt: Udtrykket *betingelse* evalueres. Hvis det giver `TRUE` evalueres *udtryk* (kaldet løkkekroppen). Dette gentages, så længe *betingelse* giver `TRUE`. Når *betingelse* giver `FALSE` stopper evalueringen af `while` udtrykket. Værdien af et `while` udtryk er værdien af den sidste evaluering af kroppen (og `NULL` hvis kroppen aldrig evalueres – altså hvis betingelsen var falsk fra start).

Eksempel: Find det mindste heltal *n* så summen af tallene fra 1 til *n* giver mindst 1000:

```
> s <- n <- 0
> while(s<1000) { n <- n+1; s <- s+n; }
> n
[1] 45
> s
[1] 1035
```

19.9.1 Eksempel: En funktion til potensopløftning af matricer

Som vi så tidligere kan man ikke skrive potensopløftning af en kvadratisk matrix `A` med den naturlige hat-notation; `A^2` giver ikke A^2, altså `A` matrixmultipliceret med sig selv, men derimod en matrix hvor hvert element er kvadratet af det tilsvarende element i `A`.

Vi vil nu definere en funktion `matpow(A,n)` der beregner A^n, det vil sige `A` matrixmultipliceret med sig selv `n` gange (hvor `n` er et heltal). Vi bemærker at A^0 = `matpow(A,0)` giver en enhedsmatrix af samme dimensioner som `A`. I første omgang begrænser vi os til $n \geq 0$.

```
> matpow <- function(A, n) {  # matrix-potensopløftning for heltal n>=0
+   res <- diag(dim(A)[1]);   # start: enhedsmatrix af samme størrelse som A
+   while (n > 0) {
+     res <- res %*% A;       # gang A på resultatet, n gange
+     n <- n-1;
+   }
+   res;                      # det færdige resultat
+ }
```

I ovenstående opbygger vi resultatet i variablen `res`. Vi starter med at sætte `res` lig enhedsmatricen af samme størrelse som `A` (funktionen `dim` giver dimensionerne af en matrix som en vektor med to elementer: antal rækker og antal søjler). Kroppen af `while`-løkken udføres `n` gange og hver gang sættes `res` lig den gamle værdi af `res` matrixmultipliceret med `A`. Efter løkken har vi

$$\texttt{res} = E\,\underbrace{\texttt{A}\cdots\texttt{A}}_{\texttt{n}} = \texttt{A}^{\texttt{n}}$$

og dette bliver funktionens værdi. Bemærk, at hvis `n` er nul udføres løkkekroppen nul gange, hvorved `res` forbliver enhedsmatricen (dette er grunden til at vi ikke kan anvende en `for`-løkke). Eksempler på brug:

```
> A <- matrix(c(1,2,3,4),2)
> matpow(A,0)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> matpow(A,1)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> matpow(A,2)
     [,1] [,2]
[1,]    7   15
[2,]   10   22
```

Ofte bruger man notationen A^{-1} for den inverse af A. Vi kan definere $A^{-n} = (A^{-1})^n$ for negative heltal $-n$. Dermed kan vi generalisere `matpow` til også at håndtere negative potenser af matricer:

```
> matpow <- function(A, n) {  # matrix-potensopløftning for heltal n
+   if (n < 0) {
+     n <- -n;                # for negative potenser, vend fortegn af n
+     A <- solve(A);          # og sæt A til den inverse af A
+   }
+   res <- diag(dim(A)[1]);   # start: enhedsmatrix af samme størrelse som A
+   while (n > 0) {
+     res <- res %*% A;       # gang A på resultatet, n gange
+     n <- n-1;
+   }
+   res;                      # det færdige resultat
+ }
```

Eksempler på brug:

```
> matpow(A,-1)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5

> matpow(A,-2)
     [,1]  [,2]
[1,]  5.5 -3.75
[2,] -2.5  1.75
```

19.9.2 Eksempel: En funktion til iteration af funktioner

Hvis f er en funktion og $n \geq 0$ er et heltal, kan en ny funktion $f^n(x)$ defineres som resultatet af n gange at bruge f på x:

$$f^n(x) = \underbrace{f(\ldots(f}_{n}(x))\ldots)$$

Specielt gælder at $f^0(x) = x$ og $f^1(x) = f(x)$ og $f^2(x) = f(f(x))$. Denne form for iteration af funktioner kan i R programmeres på samme måde som potensopløftning for matricer i foregående afsnit. Funktionen `funpow(f, n, x)` beregner $f^n(x)$ og kan defineres sådan her:

```
> funpow <- function(f, n, x) {
+   res <- x;
+   while (n > 0) {
+     res <- f(res);
+     n <- n-1;
+   }
+   res;
+ }
```

Til forskel fra `matpow` kan man dog ikke generelt udvide `funpow(f,n,x)` til at virke for negative `n`. Det ville kræve at der var en generel måde at finde den inverse f^{-1} til en given funktion f, og det er der ikke.

Som eksempel på brug af `funpow` kan vi betragte pensionsopsparing med en årlig rente på 4% og et årligt indskud på 41000 kroner. Funktionen $f(s) = s + 0.04s + 41000$ beregner næste års saldo $f(s)$ som summen af dette års saldo s, med renten og det årlige indskud konstant. Hvis man starter med 0 kroner og sparer op på den måde i 35 år, så får man $f^{35}(0)$ kroner, godt 3 millioner kroner:

```
> f <- function(s) { s + 0.04*s + 41000 }
> funpow(f, 35, 0)
[1] 3019741
```

Hvis man vil se hvordan opsparingen forløber over de første 7 år kan man beregne $f^0(0)$, $f^1(0), \ldots, f^7(0)$ ved at bruge funktionen `sapply` (afsnit 14.3) til at beregne $f^n(0)$ for $n = 0, \ldots, 7$ sådan her:

```
> sapply(0:7, function(n) { funpow(f, n, 0) })
[1]      0.0  41000.0  83640.0 127985.6 174105.0 222069.2 271952.0 323830.1
```

Dette resultat er nemmere at aflæse hvis man parrer saldoen med årnummeret `n`. Det kan man gøre sådan:

```
> sapply(0:7, function(n) { c(n, funpow(f, n, 0)) } )
     [,1]  [,2]  [,3]     [,4]     [,5]     [,6]   [,7]     [,8]
[1,]    0     1     2      3.0      4.0      5.0      6      7.0
[2,]    0 41000 83640 127985.6 174105.0 222069.2 271952 323830.1
```

Hvis resultatet er langt, bliver det mere overskueligt hvis man transponerer det med R-funktionen `t`:

```
> t(sapply(0:35, function(n) { c(n, funpow(f, n, 0)) } ))
      [,1]      [,2]
 [1,]    0       0.0
 [2,]    1   41000.0
  ...
[35,]   34 2864174.2
[36,]   35 3019741.2
```

Kombinationen af `sapply` og `funpow` er særdeles nyttig, og virker også i forbindelse med funktioner f der tager argumenter og producerer resultater som er matricer eller vektorer. Nedenstående er en simpel nationaløkonomisk model, hvor funktionen f som argument tager en søjlevektor der beskriver forbrug C og investeringer I i et givet år, og som resultat giver en søjlevektor der viser næste års forbrug og investeringer (se Anvendelseseksempel B.5 i *Noter om Matematik*):

```
> A <- matrix(c(0.8, -0.1, 0.8, 0.4), 2)
> b <- c(4, 2)
> f <- function(v) { A %*% v + b }
> f(c(15, 0))
     [,1]
[1,] 16.0
[2,]  0.5
```

Hvis man antager at $C = 15$ og $I = 0$ ved start, kan man lave økonomiske fremskrivninger for de næste 6 år på denne måde:

```
> sapply(c(0:6), function(i) { funpow(f, i, c(15, 0)) } )
     [,1] [,2] [,3]  [,4]   [,5]    [,6]     [,7]
[1,]   15 16.0 17.2 18.24 19.008 19.5136 19.81312
[2,]    0  0.5  0.6  0.52  0.384  0.2528  0.14976
```

Det egentlige emne for dette afsnit er løkker, men vi skal da nævne at `funpow(f,n,x)` også kan defineres rekursivt, uden en løkke:

```
> funpow <- function(f, n, x) {
+   if (n==0) x else funpow(f, n-1, f(x));
+ }
```

Denne version af `funpow` giver samme resultater som den løkkebaserede. Den svarer nøje til denne rekursive matematiske definition af $f^n(x)$ for $n \geq 0$:

$$f^n(x) = \begin{cases} x, & \text{hvis } n = 0, \\ f^{n-1}(f(x)) & \text{ellers.} \end{cases}$$

I stedet for `if-else` kan man bruge `ifelse` fra afsnit 19.8 i den rekursive `funpow`-funktion. Så virker funktionen også når `n` er en vektor, og man kan undgå brugen af `sapply`. Men det virker ikke når `f` returnerer en vektor eller matrix, så vi foretrækker kombinationen af `sapply` og `funpow`, der altid virker.

20 Opgaver til R

Når du løser disse opgaver kan du slå de forskellige R-funktioner op ved hjælp af det alfabetiske indeks bag i disse noter. Står der for eksempel "brug R-funktionen `uniroot` til ..." i en opgave og du ikke kan huske hvordan man bruger `uniroot` kan du slå op i indeks for at finde det afsnit hvor det er beskrevet.

Opgaver til modul A

Opgave Dat-A-1 — Start arbejde med R

Udfør trinnene beskrevet i afsnit 1.1 for at oprette mappen MatDat og en R-ikon til at starte R med.

Opgave Dat-A-2 — Definér og anvend simpel funktion

Definér funktionen $g(x) = 3e^{-2(x-1)^2}$ i R. Brug din funktion til at udregne $g(-1)$, $g(0)$ og $g(1)$.

Opgave Dat-A-3 — Definér, anvend og plot simpel funktion

(1) Definér funktionen $f(x) = 2\cos(\frac{x}{2}) + x$ i R. Brug din funktion til at udregne $f(-1)$, $f(0)$ og $f(1)$.

(2) Lav med R-funktionen `plot` en graf for funktionen f for intervallet $x \in [-10, 10]$. Giv grafen titlen "Funktionen f" (med `main=`) og aksetekserne "x" og "y" på første- hhv. andenaksen (med `xlab=` hhv. `ylab=`). Selve kurven skal være blå (brug `col=`).

Opgave Dat-A-4 — Funktion med parameter

Definér funtionen $f(x) = x^a$ hvor a er en parameter. Hvad sker der når du skriver `f(2)`?

Sæt nu $a = 3$ (med tildelingen `a <- 3`) og skriv igen `f(2)`. Udregn også værdien $f(-2)$.

Sæt nu $a = 0.5$ og udregn igen $f(2)$ og $f(-2)$. Forklar.

Opgave Dat-A-5 — Find nulpunkt for funktion

(1) Definér funktionen $f(x) = x^3 + 4x^2 - 17$ i R. Brug din funktion til at udregne $f(1)$, $f(2.3)$ og $f(-4)$.

(2) Lav med R-funtionen `plot` grafer for funktionen f for intervallerne $x \in [-5, 5]$ og $x \in [-2, 2]$.

(3) Find et nulpunkt for funktionen f ved hjælp af R-funktionen `uniroot`. Brug graferne fra før til at vælge et passende interval at søge i.

Opgave Dat-A-6 — Løs tredjegradsligning

Find de tre løsninger til tredjegradsligningen $-2x^3+6x^2-5=0$ ved hjælp af R.

Fremgangsmåde: Definér venstresiden som en funktion af x og lav en graf hvoraf du kan se cirka hvor nulpunkterne er. Brug derefter `uniroot` tre gange med forskellige intervaller for at finde de tre nulpunkter.

Opgave Dat-A-7 — Løs ligning

Det oplyses at ligningen $x^2-4\sin(3x)-1=0$ har netop fire løsninger. Find de fire løsninger ved hjælp af R.

Fremgangsmåde: Definér venstresiden som en funktion af x og lav en graf hvoraf du kan se cirka hvor nulpunkterne er. Brug derefter `uniroot` fire gange for at finde de fire nulpunkter.

Opgave Dat-A-8 — Plot flere funktioner sammen

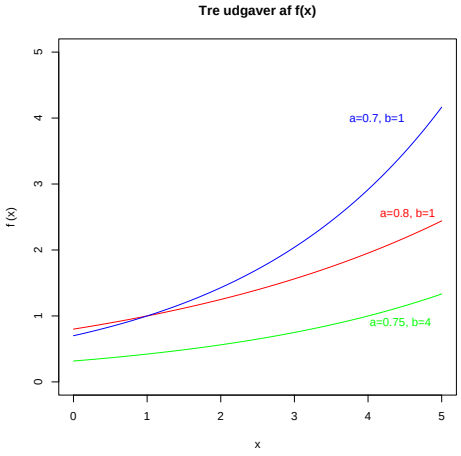
Definér funktionerne $g(x)=0.6\cdot e^{-x^2}$ og $h(x)=1-g(x+2)$. Tegn de to funktioner for intervallet $x\in[-5,5]$ i samme graf. Sørg for at akseteksten på andenaksen bliver "y". Lav de to kurver i forskellig farve.

Fremgangsmåde: Brug `plot(...)` for at tegne første kurve og `plot(...,add=TRUE)` for at tegne anden kurve. Sørg i det første `plot`-kald for at intervallet på andenaksen er stort nok til at begge kurver kan vises (brug `ylim=`).

Opgave Dat-A-9 — Flere kurver i samme graf

Definér funktionen $f(x)=a^{b-x}$, hvor $a>0$ og b er parametre. Du skal nu lave en graf for funktionen for intervallet $x\in[0,5]$ for tre forskellige kombinationer af a og b. Du skal bruge `plot` tre gange, med `add=TRUE` i de to sidste kald. Sørg for at sætte andenaksens interval til $y\in[0,5]$ i det første kald af `plot` for at ingen af kurverne skal blive beskåret. Definér først $a=0.8$ og $b=1$ og tegn en rød kurve. Sæt derefter $a=0.7$ og $b=1$ og tegn en blå kurve. Sæt endelig $a=0.75$ og $b=4$ og tegn en grøn kurve.

Du kan eventuelt sætte forklarende tekst ind ved siden af hver kurve (for eksempel "a=0.8, b=1" ved den første) med R-funktionen `text`. Sæt eventuelt også farven på den forklarende tekst til at være den samme som farven på kurven. Grafen kan komme til at ligne nedenstående:



Opgave Dat-A-10 — Graddage og ukrudtsbekæmpelse

Denne opgave bygger på anvendelseseksempel A.15 "Graddage og ukrudtsbekæmpelse" fra *Noter om matematik*. Det er en god idé at følge med i anvendelseseksemplet i noterne efterhånden som du løser opgaven.

(1) Definer funktionen `T` (døgnets middeltemperatur som funktion af antal dage efter 15. april) fra anvendelseseksemplet, dvs.

$$T(t) = 8 + 8\sin\left(\frac{2\pi}{365}t\right).$$

(I R skriver man π som `pi`.)

Check at din definition er fornuftig ved at lave en graf for `T` for intervallet $t \in [0, 365]$.

(2) Funktionen Graddage defineres i *Noter om Matematik* som:

$$\text{Graddage}(t_2) = \int_0^{t_2} T(t)dt.$$

I modsætning til i matematiknoterne skal du ikke selv integrere T men derimod lade R om at lave numerisk integration. Skriv nu i R funnktionen `Graddage(t2)` svarende til ovenstående. Du skal bruge R-funktionen `integrate` og huske at tage `$value` for at få et tal vi kan regne videre på, altså `integrate(...)$value`. Check din funktion ved at tage værdien til $t_2 = 10$:

```
> Graddage(10)
[1] 86.8687
```

Note: Desværre kan man ikke lave en graf for `Graddage` direkte som man kan for `T`. Det fører for vidt her at forklare hvorfor.

(3) Bestem den værdi af t_2 for hvilken `Graddage`(t_2) når værdien 350, hvilket i R gøres nemmest ved at finde nulpunkt for funktionen $f(t_2) = \text{Graddage}(t_2) - 350$. Brug R funktionen `uniroot` til at finde en løsning i intervallet $t_2 \in [0, 100]$.

Opgave Dat-A-11 — Koncentrationsforøgelse i biokemisk proces

Denne opgave bygger på anvendelseseksempel A.19 "Koncentrationsforøgelse i biokemisk proces" fra *Noter om matematik.*

(1) Definer funktionen `v` (hastigheden hvormed et stof dannes i en given proces som funktion af tiden t) som

$$v(t) = (1.1 + 3.5t)e^{-1.5t}$$

og lav en graf for `v` for t mellem 0 og 3 svarende til figur A.63 i matematiknoterne.

(2) Bestem nu den totale koncentrationsforøgelse, dvs.

$$\int_0^\infty v(t)dt,$$

i R ved numerisk integration med `integrate`.

Opgave Dat-A-12 — Indlæsning, plot og transformation af data fra kemisk reaktion

(1) Gå ind på kursushjemmesiden `http://www.matfys.kvl.dk/mat-dat` og klik på "Eksempelfiler til opgaver og forelæsninger". Højreklik på filen `data/reaktion.txt` (fra mappen R) og vælg "gem som" og gem filen på din computer i den mappe du har oprettet til *Matematik og Databehandling* (mappen MatDat).

(2) Start med at åbne filen i fx. Notesblok (højreklik og "Åbn"). Undersøg om der er en overskrift, om der er brugt decimalkomma eller decimalpunktum, samt hvordan kolonnerne er adskilt (skilletegn).

Læs datasættet ind med `data <- read.table(...)` og husk at angive de nødvendige parametre så overskrift, tal og skilletegn læses korrekt.

Brug funktionen `summary` for at checke at data er indlæst korrekt. Det skal se således ud:

```
> summary(data)
      Tid              S1              S2              S3
 Min.   :  2.0   Min.   :0.050   Min.   :0.1000   Min.   :0.050
 1st Qu.: 26.5   1st Qu.:0.500   1st Qu.:0.5625   1st Qu.:4.088
 Median : 51.0   Median :1.275   Median :1.3000   Median :7.250
 Mean   : 51.0   Mean   :2.352   Mean   :1.3770   Mean   :6.264
 3rd Qu.: 75.5   3rd Qu.:3.413   3rd Qu.:2.1875   3rd Qu.:8.950
 Max.   :100.0   Max.   :9.400   Max.   :2.8500   Max.   :9.700
```

(3) Lav med `plot` et XY-plot af kolonnen `S2` som funktion af kolonnen `Tid`. Datapunkterne skal ikke forbindes med linier.

Vink: For at få kolonnen `S2` fra datasættet `data` skal du skrive `data$S2`.

(4) Lav et XY-plot af kvadratroden (`sqrt`) af kolonnen `S1` som funktion af kolonnen `Tid`. Datapunkterne skal ikke forbindes med linier.

Definér funktionen $f(t) = 2.66 - 0.025t$. Tilføj med `plot(...,add=TRUE)` grafen for f.

(5) Lav et XY-plot af den naturlige logaritme (`log`) til kolonnen `S1` som funktion af kolonnen `Tid`. Datapunkterne skal ikke forbindes med linier.

Definér funktionen $g(t) = 2.3 - 0.04t$. Tilføj med `plot(...,add=TRUE)` grafen for g.

Opgave Dat-A-13 — Indlæsning og plot af data fra vækstforsøg

Filen `data/Grise2Fast.txt` fra eksempelfil-samlingen på kursushjemmesiden indeholder data fra et forsøg med væksthormon til slagtesvin. Filen har tre variable `Tid`, `Kontrol` og `Vaekst`, og 20 observationer. Hver observation angiver et antal minutter efter slagtning sammen med målt pH i kødet fra grise der er opdrættet normalt (kontrolgruppen) og fra grise der har fået væksthormon.

(1) Hent fra kursushjemmesiden `http://www.matfys.kvl.dk/mat-dat` filen `data/Grise2Fast.txt` og gem filen på din computer i den mappe du har oprettet til *Matematik og Databehandling* (mappen MatDat).

(2) Undersøg filens indhold for oversrifter, skilletegn m.v. og læs så datasættet fra filen ind i R med `read.table`. Kald datasættet `data`. Check med `summary` at det er korrekt indlæst.

(3) Lav med `plot` og `points` et XY-plot med de to kolonner `Kontrol` og `Vaekst` som funktion af kolonnen `Tid`. Datapunkterne fra `Kontrol` skal være røde og datapunkterne fra `Vaekst` blå. Husk (med `ylim=` i `plot`) at sætte intervallet på andenaksen så ingen punkter falder udenfor (brug oplysningerne fra `summary` til at vælge et interval). Aksetekserne skal være "Minutter" på førsteaksen og "pH" på andenaksen (teksterne angives med `xlab=` og `ylab=` i `plot`).

(4) Tilføj med `legend` en signaturforklaring til grafen fra delopgave (3). Den skal have en rød cirkel med teksten "Kontrolgruppe" og en blå cirkel med teksten "Væksthormon".
Vink: Cirklen som R tegner som standard er plot-symbol (plotting character) nummer 1, så for at få cirkler ud for hver tekst i `legend` skal man huske `pch=1` (jævnfør figur 23 på side 32).

Opgave Dat-A-14 — Udklægningstid for flueæg

Denne opgave bygger på anvendelseseksempel A.1 "Udklækningstid for flueæg" fra *Noter om matematik*. Det er en god idé at kigge i anvendelseseksemplet i noterne under løsning af opgaven.

(1) Datasættet med måling af udklækningstid U som funktion af luftfugtighed L findes i tekstfilen `data/flueaeg.txt` under kursushjemmesidens eksempelsamling. Hent filen fra hjemmesiden og gem den i din MatDat mappe.

Indlæs datasættet med:

```
flueaeg <- read.table("flueaeg.txt",header=TRUE)
```

(2) Lav et XY-plot af `flueaeg$U` som funktion af `flueaeg$L`.

(3) Foretag med R-funktionen `lm` lineær regression for at bestemme den rette linie der bedst beskriver sammenhængen mellem L og U (kald resultatet for `lreg`). Angiv ud fra resultatet konstanterne a og b i modellen $U = aL + b$.

(4) Brug endelig R-funktionen `abline` til at tilføje linien $U = aL + b$ til plottet fra delopgave (2).

Opgave Dat-A-15 — Andemad

Denne opgave bygger på anvendelseseksempel A.4 "Andemad" fra *Noter om matematik*. Det er en god idé at kigge i anvendelseseksemplet i noterne under løsning af opgaven.

(1) Datasættet med antal blade N som funktion af antal dage siden forsøgets start t findes i filen `data/andemad.txt` under kursushjemmesidens eksempelsamling. Hent filen fra hjemmesiden og gem den i din MatDat mappe.

Indlæs datasættet med:

```
andemad <- read.table("andemad.txt",header=TRUE)
```

(2) Lav et XY-plot af logaritmen til `andemad$N` som funktion af `andemad$t`.

(3) Transformér kolonnen `N` ved at tage den naturlige logaritme. Kald de transformerede værdier for `logN`. Foretag med R-funktionen `lm` lineær regression for at bestemme den rette linie der bedst beskriver sammenhængen mellem $\ln N$ og t (kald resultatet for `lreg`). Angiv ud fra resultatet konstanterne b og r i modellen $\ln N = \ln b + rt$.

(4) Brug R-funktionen `abline` til at tilføje linien $\ln N = \ln b + rt$ til plottet fra delopgave (2).

(5) Definér funktionen $f(t) = be^{rt}$ med de fundne værdier for b og r fra delopgave (3).

Lav nu et XY-plot af `andemad$N` som funktion af `andemad$t` og brug `plot(...,add=TRUE)` til at tilføje grafen for $f(t)$ for intervallet $t \in [0, 13]$.

Opgave Dat-A-16 — Stofskifte og kropsvægt hos pattedyr

Denne opgave bygger på del (d) af anvendelseseksempel A.8 "Stofskifte og kropsvægt hos pattedyr" fra *Noter om matematik*. Det er en god idé at kigge i anvendelseseksemplet i noterne under løsning af opgaven.

(1) Datasættet med stofskifte S som funktion af kropsvægt K findes i filen `data/stofskifte.txt` under kursushjemmesidens eksempelsamling. Hent filen fra hjemmesiden og gem den i din MatDat mappe.

Indlæs datasættet med:

```
stofskifte <- read.table("stofskifte.txt",header=TRUE)
```

(2) Transformér kolonnerne `K` og `S` ved at tage den naturlige logaritme. Kald de transformerede data for `logK` og `logS`. Foretag med R-funktionen `lm` lineær regression for at bestemme den rette linie der bedst beskriver sammenhængen mellem $\ln S$ og $\ln K$ (kald resultatet for `lreg`). Angiv ud fra resultatet konstanterne a og b i modellen $\ln S = \ln b + a \ln K$.

(3) Lav et XY-plot af `logS` som funktion af `logK`. Tilføj derefter ved hjælp af `abline` den rette linie $\ln S = \ln b + a \ln K$ fundet i delopgave (2).

(4) Definér funktionen $f(K) = bK^a$ med de fundne værdier for a og b fra delopgave (2).

Lav nu et XY-plot af `stofskifte$S` som funktion af `stofskifte$K` og brug `plot(...,add=TRUE)` til at tilføje grafen for $f(K)$ for intervallet $K \in [0, 50]$.

Opgave Dat-A-17 — Arbejde med data (powerdata.csv)

Filen `powerdata.csv` indeholder data fra et eksperiment hvor man har målt den tid det tager en computer at "lægge" et puslespil. Man har prøvet dette for forskellige puslespil med forskellig antal brikker N. Der er målt tidsforbrug i sekunder for to forskellige alternative løsningsstrategier, A og B.

(1) Hent fra kursushjemmesiden `http://www.matfys.kvl.dk/mat-dat` filen `data/powerdata.csv` og gem filen på din computer i den mappe du har oprettet til *Matematik og Databehandling* (mappen MatDat).

(2) Læs datasættet fra filen ind i R. Kald det `data`. Check med `summary` at data er korrekt indlæst.

(3) Lav med `plot` og `points` et XY-plot med de to kolonner `TidA` og `TidB` som funktion af kolonnen `N`. Datapunkterne fra `TidA` skal være røde og datapunkterne fra `TidB` blå. Husk at sætte intervallet på andenaksen så ingen punkter falder udenfor (brug oplysningerne fra `summary` til at vælget et interval). Aksetekserne skal være "N" på førsteaksen og "Tid" på andenaksen.

(4) Foretag med R-funktionen `lm` lineær regression for at bestemme den rette linie der bedst beskriver sammenhængen mellem `TidA` og `N`. Lav herefter et XY-plot med `TidA` som funktion af `N` og tilføj den fundne regressionslinie ved hjælp af R-funktionen `abline`.

(5) Transformér kolonnerne `N`, `TidA` og `TidB` ved at tage den naturlige logaritme. Kald de transformerede data for `logN`, `logA` og `logB`.

Lav herefter med `plot` og `points` et XY-plot for de transformerede data med røde punkter for `logA` som funktion af `logN` og blå punkter for `logB` som funktion af `logN` (ligesom i delopgave (3)). Aksetekserne skal være "ln(N)" på førsteaksen og "ln(Tid)" på andenaksen.

(6) Foretag med R-funktionen `lm` lineær regression for at bestemme den rette linie der bedst beskriver sammenhængen mellem `logA` og `logN`. Gør det samme for at finde sammenhængen mellem `logB` og `logN`. Tilføj med `abline` regressionslinierne til plottet fra delopgave (5) i hhv. rød og blå.

(7) Tilføj nu med `legend` en signaturforklaring med følgende fire tekster: "Data A" (røde punkter), "Data B" (blå punkter), "Regression A" (rød linie) og "Regression B" (blå linie).

Opgaver til modul B

Opgave Dat-B-1 — Oprettelse af talrækker/vektorer

(1) Konstruér ved hjælp af R-funktionen `seq` følgende talrækker/vektorer:

- Tallene 1 til 10.
- Tallene 7, 8, 9, . . . , 15.
- Tallene 10, 9, . . . , 1.
- Tallene 0.25, 0.5, 0.75, . . . , 5.0.
- De ulige tal fra 1 til 29.
- Tallene 0, $\frac{1}{9}$, $\frac{2}{9}$, . . . , 1. (**Vink:** brug `len=`)

(2) Konstruér ved at kombinere R-funktionerne `seq`, `rep` og/eller `c` følgende talrækker/vektorer:

- Tallene 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1.
- Tallet 1 gentaget 10 gange efterfulgt af tallet 2 gentaget 8 gange.
- Tallene 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3.

(3) Konstruér ved hjælp af kolon-operatoren samt regneoperationer, for eksempel `(1:4)*2` (som giver 2, 4, 6, 8), følgende talrækker/vektorer:

- Tallene 10, 9, . . . , 1.
- Tallene 0.25, 0.5, 0.75, . . . , 5.0.
- De ulige tal fra 1 til 29.
- Tallene 0, $\frac{1}{9}$, $\frac{2}{9}$, . . . , 1.
- Tallene 100, 81, 64, 49, 36, 25, 16, 9, 4, 1. (**Vink:** brug potensopløftning)
- (Vanskelig) Tallene 0, 0, 2, 2, 4, 4, 6, 6, 8, 8.

Opgave Dat-B-2 — Flere talrækker/vektorer

(1) Konstruér ved hjælp af R-funktionen `seq` følgende talrækker/vektorer:

- Tallene 0.1, 0.2, 0.3, . . . , 1.0.
- Tallene 5, 10, 15, . . . , 100.
- Tallene 0, $\frac{2}{3}$, $\frac{4}{3}$, 2. (**Vink:** brug `len=`)

(2) Konstruér ved at kombinere R-funktionerne `seq`, `rep` og/eller `c` følgende talrækker/vektorer:

- Tallene 1, 2, . . . , 5 gentaget 4 gange.
- Tallene 3, 3, 3, 3, 3, 2, 2, 2, 1.

(3) Konstruér ved hjælp af kolon-operatoren samt regneoperationer, for eksempel `(0:3)*2+1` (som giver 1, 3, 5, 7), følgende talrækker/vektorer:

- Tallene 0.1, 0.2, 0.3, . . . , 1.0.
- Tallene 5, 10, 15, . . . , 100.
- Tallene 1.5, 2.5, . . . , 9.5.
- Tallene 0, $\frac{2}{3}$, $\frac{4}{3}$, 2.
- (Vanskelig) Tallene 2, 4, 6, 8, 10, 7, 9, 11, 13, 15.

Opgave Dat-B-3 — Endnu flere talrækker/vektorer

(1) Konstruér ved hjælp af R-funktionen `seq` følgende talrækker/vektorer:

- Tallene 0, 10, 20, . . . , 100.
- Tallene 100, 90, 80, . . . , 0.
- Tallene 0, $\frac{1}{3}$, $\frac{2}{3}$, . . . , 4. (**Vink:** brug `len=`)

(2) Konstruér ved at kombinere R-funktionerne `seq`, `rep` og/eller `c` følgende talrækker/vektorer:

- Tallene 1, 2, 3, . . . , 9, 10, 9, . . . , 2, 1, det hele gentaget 3 gange.
- Tallene 7, 9, 13 gentaget 5 gange.

(3) Konstruér ved hjælp af kolon-operatoren samt regneoperationer følgende talrækker/vektorer:

- Tallene 0, 10, 20, . . . , 100.
- Tallene 100, 90, 80, . . . , 0.
- Tallene 9.5, 8.5, . . . , 1.5.
- Tallene 0, $\frac{1}{3}$, $\frac{2}{3}$, . . . , 4.
- (Vanskelig) Tallene -1, 0, 3, -4, 0, 6, -7, 0, 9.
- (Meget vanskelig) Tallene 1, 2, 9, 4, 25, 6, 49, 8, 81, 10.

Opgave Dat-B-4 — Oprettelse af og regning med 2×2 matricer

(1) Opret følgende to 2×2 matricer:

$$A = \begin{pmatrix} 1 & 2 \\ 5 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 4 & 6 \\ 7 & 8 \end{pmatrix}.$$

(2) Udregn matrixprodukterne AB og BA. Er $AB = BA$?

(3) Find med `solve` den inverse matrix til `B`.

Opgave Dat-B-5 — Matrix invers og 2×2 ligningssystem

(1) Find ved at bruge `solve` den inverse til nedenstående matrix:

$$\begin{pmatrix} 1 & 7 \\ 9 & 13 \end{pmatrix}$$

(2) Løs følgende ligningssystem i R ved at omskrive til matrixform og bruge `solve`:

$$\begin{cases} 4.3x & + & 2.1y & = & 3 \\ -3.2x & + & 1.1y & = & -2 \end{cases}$$

Opgave Dat-B-6 — Opret større matrix; indeksering

(1) Opret en matrix A som følger:

$$A = \begin{pmatrix} 1 & 5 & 9 & 13 & 17 \\ 2 & 6 & 10 & 14 & 18 \\ 3 & 7 & 11 & 15 & 19 \\ 4 & 8 & 12 & 16 & 20 \end{pmatrix}.$$

(2) Udtag (ved hjælp af indeksering) fra A følgende dele:

- Elementet i 1. række, 3. søjle.
- Elementet i 4. række, 2. søjle.
- Hele 3. række.
- Hele 4. søjle.
- Delmatricen bestående af de første tre søjler.
- Delmatricen bestående af de to sidste rækker.
- 2×2 delmatricen bestående af de to sidste søjler og rækker ("nederste højre hjørne").
- Delmatricen bestående af alle søjler på nær den tredje.

Opgave Dat-B-7 — Opbygning af matrix med `cbind`

(1) Definér følgende matrix A og vektor v_0 (skriv `v0` for v_0 i R):

$$A = \begin{pmatrix} 0.20 & 0.80 \\ 0.90 & 0.15 \end{pmatrix}, \quad v_0 = \begin{pmatrix} 100 \\ 0 \end{pmatrix}.$$

Udregn herefter successivt følgende vektorer:

$$v_1 = Av_0, \;\; v_2 = Av_1, \;\; v_3 = Av_2, \;\; v_4 = Av_3, \;\; v_5 = Av_4.$$

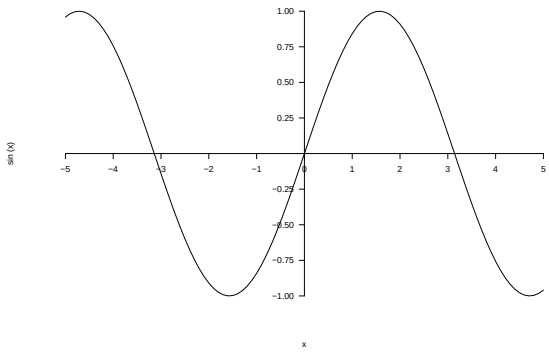
Konstruér endelig matricen V hvis søjler udgøres af vektorerne v_0 til v_5, altså hvis første søjle er v_0, anden søjle v_1 og så videre. Brug R-funktionen `cbind`.

(2) Lav med `plot` et XY-plot hvis punkter har første række af V som y-koordinater og talrækken fra 0 til 5 som x-koordinater. Punkterne skal indtegnes forbundet med røde linier (brug `type="o",col="red"`).

Tilføj med `points` den anden række af V, indtegnet på samme måde men i blåt.

Opgave Dat-B-8 — Lav plot med bestemte akser

Lav en graf for $\sin x$ for $x \in [-5, 5]$ svarende fuldstændig til denne (bemærk akserne):



Vink: Du skal bruge `axis`, se afsnit 5.1. Du skal bruge parameteren `at` med passende takrækker til at bestemme hvilke aksemærker, der skal sættes.

Opgave Dat-B-9 — Vektorkonstruktion med `for`-løkker

I denne opgave skal en række specielle vektorer, alle med 10 elementer, kontrueres ved hjælp af `for`-løkker. Fremgangsmåden er hver gang den samme:

1. Opret en vektor med 10 elementer med `rep(`*n*`,10)`, hvor *n* er tallet der skal stå på de(n) første plads(er) i vektoren. For eksempel oprettes den første vektor `v1` nedenfor med `v1 <- rep(3,10)`.
2. Kør en (eller om nødvendigt flere) `for`-løkke(r) der tildeler de ønskede værdier til pladserne 2 til 10 i vektoren. For eksempel vil den for den første vektor `v1` nedenfor have formen
   `for (i in 2:10) { v1[i] <-` *udtryk* `}`.

De vektorer, der ønskes konstrueret, er:

- Vektoren `v1`, hvorom det gælder at `v1[1]=3` og `v1[i]=v1[i-1]+i` for `i`=2, 3, ..., 10.
  Resultatet bliver talrækken 3, 5, 8, 12, 17, 23, 30, 38, 47, 57.
- Vektoren `v2`, hvorom det gælder at `v2[1]=1` og `v2[i]=i-2*v2[i-1]` for `i`=2, 3, ..., 10.
  Resultatet bliver talrækken 1, 0, 3, -2, 9, -12, 31, -54, 117, -224.
- Vektoren `v3`, hvorom det gælder at `v3[1]=10`, `v3[2]=10` og `v3[i]=v3[i-2]-v3[i-1]` for `i`=3, 4, ..., 10.
  Resultatet bliver talrækken 10, 10, 0, 10, -10, 20, -30, 50, -80, 130.
- Vektoren `v4`, hvorom det gælder at `v4[1]=1`, `v4[i]=v4[i-2]+i^2` for `i`=3, 5, ..., 9 og `v4[j]=v4[j-1]+3` for `j`=2, 4, ..., 10.
  Resultatet bliver talrækken 1, 4, 10, 13, 35, 38, 84, 87, 165, 168.
  Vink: Dette kræver to `for`-løkker.

Opgave Dat-B-10 — Reversering af vektor med `for`-løkke

Opret vektoren `v<-1:10` (talrækken 1, 2, ..., 10). Skriv derefter en `for`-løkke, der ved at bytte om på elementerne i `v` vender vektoren om (så den bliver talrækken 10, 9, ..., 1).

Vink: Løkken skal gå over tallene 1, 2, ..., 5. Det er nødvendigt at indføre en hjælpevariabel til midlertidigt at gemme det ene element når man vil bytte om på to elementer i vektoren.

Opgave Dat-B-11 — Fremskrivning i affin model med for-løkke

(1) Definér følgende matrix $\mathbf{M}$, vektor $\mathbf{q}$ og vektor $\mathbf{v}_0$ (skriv `v0` for $\mathbf{v}_0$ i R):

$$\mathbf{M} = \begin{pmatrix} 0.2 & 0.8 \\ 0.9 & 0.0 \end{pmatrix}, \quad \mathbf{q} = \begin{pmatrix} 10 \\ 0 \end{pmatrix}, \quad \mathbf{v}_0 = \begin{pmatrix} 100 \\ 0 \end{pmatrix}.$$

Udregn nu ved hjælp af en `for`-løkke vektorerne $\mathbf{v}_1$ til $\mathbf{v}_{20}$ ud fra den affine model

$$\mathbf{v}_{i+1} = \mathbf{M}\mathbf{v}_i + \mathbf{q}.$$

Vektorerne skal undervejs "opsamles" i en matrix `V` således at første søjle i `V` udgøres af $\mathbf{v}_0$, anden søjle af $\mathbf{v}_1$ og så videre.

Vink: Se på eksemplet i afsnit 16.5.

(2) Lav med `plot` et XY-plot hvis punkter har første række af `V` som y-koordinater og talrækken fra 0 til 20 som x-koordinater. Punkterne skal indtegnes forbundet med røde linier (brug `type="o",col="red"`).

Tilføj med `points` den anden række af `V`, indtegnet på samme måde men i blåt.

Opgave Dat-B-12 — Fårebestand

Denne opgave bygger på anvendelseseksempel B.4 fra *Noter om Matematik*. Det kan være en fordel at jævnføre med eksemplet i noterne undervejs.

En fårebestand med får i to klasser modelleres v.h.a. følgende affine model, jvf. anvendelseseksempel B.4 (b):

$$\mathbf{v}_{t+1} = \mathbf{M}\mathbf{v}_t + \mathbf{q},$$

hvor

$$\mathbf{v}_t = \begin{pmatrix} x_t \\ y_t \end{pmatrix}, \quad \mathbf{M} = \begin{pmatrix} 0.70 & 0.20 \\ 0.15 & 0.45 \end{pmatrix}, \quad \mathbf{q} = \begin{pmatrix} 0 \\ 900 \end{pmatrix}.$$

I året 1992 er der 500 hhv. 1400 får i de to klasser, d.v.s. $\mathbf{v}_{1992} = (500, 1400)$.

Vi ønsker at fremskive fårebestanden til året 2023 ved hjælp af modellen.

(1) Definér i R matricen `M` og vektorerne `q` og `v1992` svarende til modellen ovenfor.

Udregn nu ved hjælp af en `for`-løkke vektorerne $\mathbf{v}_{1993}$ til $\mathbf{v}_{2023}$ og opsaml undervejs resulatet i en 2×32 matrix `V`, hvor første søjle udgøres af $\mathbf{v}_{1992}$, anden søjle af $\mathbf{v}_{1993}$ osv.

Vink: Se på eksemplet i afsnit 16.5.

(2) Lav med `plot` et XY-plot som viser x_t som funktion af årstal fra 1992 til 2023. Punkterne skal indtegnes med åbne cirkler (plotsymbol 1) og forbundet med fuldt optrukne linier. Sørg for passende aksetitler ("år" og "bestand"). Vink: x_t værdierne udgør første række af `V`.

Tilføj med `points` y_t værdierne fra 1992 til 2023. Punkterne skal indtegnes med trekanter (plotsymbol 2) og forbindes med stiplede linier (`lty="dashed"`).

Tilføj titlen "Udvikling af fårebestand".

Tilføj en signaturforklaring med `legend` med de to signaturtekster "Klasse I" henholdsvis "Klasse II" for x_t henholdsvis y_t værdierne.

Opgave Dat-B-13 — Brownsk bevægelse i 2D

Denne opgave er en udbygning af eksemplet med simulering af Brownsk bevægelse i afsnit 15.1. Vi betragter nu en model for en partikels bevægelse i to dimensioner, hvor partiklens position til tidstrin t er (x_t, y_t). Partiklen starter i $(0, 0)$ til $t = 0$. Mellem tid t og tid $t + 1$ bevæger partiklen sig et lille stykke vej d_t i den tilfældige retning v_t (i radianer), hvor d_t er et (ligefordelt) tilfældigt tal mellem 0 og 1 og v_t er et (ligefordelt) tilfældigt tal mellem 0 og 2π. Vi har således:

$$\begin{aligned} x_{t+1} &= x_t + d_t \cdot \cos v_t, \\ y_{t+1} &= y_t + d_t \cdot \sin v_t, \end{aligned}$$

med d_t og v_t tilfældige af hinanden uafhængige tal som beskrevet ovenfor.

(1) Simuler 100 trin i ovenstående model, d.v.s. op til $t = 100$. Resulatet af simuleringen skal gemmes i to vektorer `X` og `Y` med hver 100 elementer hvor `X[1]`=x_1, `X[2]`=x_2 osv. og `Y[1]`=y_1, `Y[2]`=y_2 osv. Følg den grundlæggende fremgangsmåde fra afsnit 15.1.

(2) Lav et plot der viser partiklens bevægelse i 2D. Brug `plot` med `asp=1` for at sikre at enheder bliver lige lange på de to akser. Brug `type="l"` for at få tegnet "sporet" som en række forbundne liniestykker. Sørg for at også startpositionen (0,0) kommer med.

Tilføj med `points` en åben cirkel i startpositionen.

Tilføj med `points` en udfyldt cirkel (`pch=16`) i slutpositionen.

Opgaver til modul C

Opgave Dat-C-1 — Funktioner der giver sandhedsværdier

Du skal i denne opgave lave nogle funktioner der tager et tal som argument og giver en sandhedsværdi (`TRUE` eller `FALSE`) som svar. I hvert tilfælde vil funktionskroppen bestå af et logisk udtryk.

Eksempel: Funktion der giver `TRUE` hvis argumentet er et lige heltal, `FALSE` ellers:

```
lige <- function(x) { x%%2==0 }
```

(`x%%2` betyder resten ved heltalsdivision af `x` med 2, hvilket er 0 for et lige tal.)

Definér og afprøv følgende funktioner:

- Funktionen `ulige`, der giver `TRUE` hvis argumentet er et ulige heltal og `FALSE` ellers.
- Funktionen `positiv`, der giver `TRUE` hvis argumentet er et positivt tal og `FALSE` ellers.
- Funktionen `positiv.ulige`, der giver `TRUE` hvis argumentet er et positivt ulige heltal og `FALSE` ellers.
- Funktionen `timetal`, der giver `TRUE` hvis argumentet er et tal i intervallet $[0, 24[$ og `FALSE` ellers.
- Funktionen `skudaar`, der giver `TRUE` hvis argumentet er et årstal der er skudår og `FALSE` ellers. Et år er skudår hvis årstallet er deleligt med 4 men ikke med 100 *eller* hvis det er deleligt med 400 (således var år 2000 skudår mens år 1800 og 1900 ikke var).

Opgave Dat-C-2 — Numerisk værdi med `if`

Betragt nedenstående ufuldstændige funktion `num`:

```
num <- function(a) {
  ???
}
```

Færdiggør `num` så den som resultat giver den numeriske (absolutte) værdi af tallet `a`. Du skal bruge et `if`-udtryk i funktionen. Du skal *ikke* benytte R-funktionen `abs`.

Opgave Dat-C-3 — Indeksering med logiske værdier

Konstruér en vektor `x` indeholdende 100 ligefordelte tilfældige tal mellem 0 og 1 som følger:

```
> x <- runif(100)
```

Man kan med indeksering med et logisk udtryk udtage tal fra `x` der har bestemte egenskaber.
Eksempel: `x[x>0.5]` udtager de tal fra `x` der er større end $\frac{1}{2}$.

Ved at kombinere dette med `length` funktionen kan man bestemme hvor mange af tallene i `x` der har en bestemt egenskab.
Eksempel: `length( x[x>0.5] )` giver hvor mange tal i `x` der er større end $\frac{1}{2}$.

Brug nu indeksering med et logisk udtryk samt `length` til at bestemme følgende:

1. Hvor mange tal i `x` er mindre end 0.1?
2. Hvor mange tal i `x` er mellem 0.25 og 0.75?
3. Hvor mange tal i `x` er mindre end 0.1 *eller* mellem 0.25 og 0.75?
4. For hvor mange tal x_i i `x` er $\sin(1/x_i)$ større end nul?

Bemærk, at svarene til alle disse spørgsmål vil variere fra gang til gang man løser opgaven.

Opgave Dat-C-4 — Indeksering i datasæt med logiske udtryk

Indlæs datasættet fra filen `reaktion.txt` og kald det `data` (se opgave Dat-A-12).

Brug nu tildeling og indeksering af formen `data1 <- data[???,]`, hvor du udskifter de tre spørgsmålstegn med et logisk udtryk, til at fremstille følgende datasæt der alle er delmængder af `data`:

- `data1`, der indeholder de 25 observationer hvor `Tid` er mindre end eller lig `50`.
- `data2`, der indeholder de 21 observationer hvor `S2` er større end `S1`.
- `data3`, der indeholder de 13 observationer hvor den numeriske forskel på `S1` og `S2` er mindre end 0.1 (brug `abs` til at tage numerisk værdi).
- `data4`, der indeholder de 13 observationer hvor `S3` er mellem 2 og 6 inklusive.

Opgave Dat-C-5 — Nogle ikke-rekursive funktioner med `if`

(1) Betragt følgende funktion `f`:

```
f <- function(a,b) {
   if (a>b) a else b;
}
```

Beskriv i ord: Hvad giver kaldet `f(x,y)` for to tal `x` og `y`? Du behøver ikke at indtaste og afprøve funktionen for forskellige tal hvis du umiddelbart kan se hvad den gør.

(2) Betragt følgende funktion `g`:

```
g <- function(a,b) {
   if (a!=0 || b!=0) a/b else 1;
}
```

Beskriv i ord: Hvad giver kaldet `g(x,y)` for to tal `x` og `y`?

(3) Betragt følgende funktion `h`:

```
h <- function(a,b,c) {
   if (a>c && b>c) {
      c;
   } else {
      if (a>b) b else a;
   }
}
```

Beskriv i ord: Hvad giver kaldet `h(x,y,z)` for tre tal `x`, `y` og `z`?

Opgave Dat-C-6 — Midterste af tre tal

Skriv en funktion `midt <- function(a,b,c)` som giver det midterste af de tre tal `a`, `b` og `c` når de ordnes efter størrelse, altså fx:

```
> midt(1,2,3)
[1] 2
> midt(2,3,1)
[1] 2
> midt(3,1,2)
[1] 2
```

Brug udelukkende `if`-sætninger og ingen af R's statistiske funktioner.

Afprøv at din funktion virker ved alle rækkeføger af tallene 1, 2 og 3, altså såvel (1,2,3) som (1,3,2), (2,1,3), (2,3,1), (3,1,2) og (3,2,1).

Opgave Dat-C-7 — Nogle rekursive funktioner med `if`

Betragt følgende *rekursivt definerede* funktion `fra0`, der for et givet heltal $a \geq 0$ giver sekvensen af heltal fra 0 til `a`:

```
fra0 <- function(a) {
  if (a>0) c(fra0(a-1),a) else 0;
}
```

Eksempler på kald:

```
> fra0(8)
[1] 0 1 2 3 4 5 6 7 8
> fra0(0)
[1] 0
```

(1) Skriv nu en tilsvarende funktion `til0`, der for et heltal $a \leq 0$ giver sekvensen af heltal fra `a` til 0, fx.:

```
> til0(-4)
[1] -4 -3 -2 -1  0
```

(2) Skriv en funktion `med0` der afhængig af fortegnet af sit argument virker enten som `fra0` eller `til0` ovenfor, fx.:

```
> med0(8)
[1] 0 1 2 3 4 5 6 7 8
> med0(-4)
[1] -4 -3 -2 -1  0
```

Opgave Dat-C-8 — Flere rekursive funktioner med `if`

(1) Betragt funktionen `f` defineret her:

```
f <- function(a,b) { if (a>b) f(a-b,b)+1 else a; }
```

Besvar, *uden* at taste funktionen ind og prøve den, hvad giver `f(7,2)`?

(2) Lav en *rekursiv* funktion fib(n) der for heltal $n \geq 1$ giver Fibonacci-tal nummer n. Brug følgende rekursive definition af Fibonacci-tallene som udgangspunkt:

$$\text{fib}(n) = \begin{cases} 1 & \text{hvis } n \leq 2 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{hvis } n > 2 \end{cases}$$

(3) Lav en *rekursiv* funktion `sfd` der for to naturlige tal a og b finder deres *største fælles divisor* ved hjælp af *Euklids algoritme*[8], der kan opskrives som følger:

$$\text{sfd}(a,b) = \begin{cases} a & \text{hvis } b = 0, \\ \text{sfd}(b, a \bmod b) & \text{ellers.} \end{cases}$$

Notationen $a \bmod b$ betyder her a modulo b, dvs. resten ved heltalsdivision af a med b (i R skrives modulo operatoren `%%`).

Opgave Dat-C-9 — Simpel `while`-løkke

Brug en `while` løkke til at bestemme det mindste heltal n således at $n^2 > 500$. Check dit resultat ved at beregne $\sqrt{500}$.

Vink: Start med `n<-0` før løkken og tæl `n` op med én i løkkekroppen.

Opgave Dat-C-10 — Antal led i en sum

Brug en `while` løkke til at afgøre hvad der er mindste heltal n således at

$$\sum_{x=1}^{n} \frac{1}{x} \geq 10$$

Vink: Kig på første eksempel i afsnit 19.9.

[8] *Eukleides* af Alexandria (ca. 365 fvt. – 275 fvt.), græsk matematiker hvis kendteste værk *Elementer* bl.a. indeholder denne algortime til at finde største fælles divisor af to naturlige tal.

Opgave Dat-C-11 — Funktionen `overflade` fra kursushjemmesiden

Hent og indlæs filen `overflade.R` fra kursushjemmesiden som beskrevet i appendiks F.1.

Opret nu funktionen `g` som følgende matematiske funktion af to variable:

$$g(x, y) \;=\; \sin\left(x + \sqrt{xy}\right).$$

Tegn med `overflade` et 3D overfladeplot for g i området $x \in [0, 5]$, $y \in [0, 5]$. Prøv med forskellige værdier af parameteren `n` (antal støttepunkter) og prøv med `showangles=TRUE` at finde den bedste betragtningsvinkel (`phi` og `theta`). Eksperimentér også med parametrene `col` (feltfarver), `border` (stregfarver) og `shade` (retningsbestemt lys), se afsnit 18 (disse parametre gives alle videre fra `overflade` til `persp`).

Opgave Dat-C-12 — Farvning af overfladeplot efter funktionsværdi

I denne opgave skal du modificere funktionen `overflade`, som findes i R-filen `overflade.R`, der kan hentes fra kursushjemmesiden, se appendiks F. Start med at kopiere `overflade.R` til en ny fil `overflade2.R`, så du ikke ændrer i den oprindelige fil. Åbn `overflade2.R` i R Editor og lav dine ændringer dér.

Lav nu om i funktionen `overflade` således at firkanterne i plottet farves med gråtoner efter funktionsværdierne som beskrevet i afsnit 18.1.1. I stedet for at bruge funktionsværdierne i firkanternes hjørner til at bestemme gråtonerne skal du dog bruge funktionsværdierne i midten af firkanterne.

Vink: Du skal beregne en ny matrix af funktionsværdier ved hjælp af `outer` ud fra to vektorer der indeholder midtpunkterne imellem støttepunkterne langs x og y akserne.

MATEMATIK OG DATABEHANDLING

Modul D

Noter om Regneark

Peter Sestoft

Institut for Grundvidenskab
Den Kongelige Veterinær- og Landbohøjskole, KVL

Disse noter beskriver brug af regneark i forbindelse med kurset Matematik og Databehandling på KVL, og skal ses i sammenhæng med de tilsvarende noter om statistikprogrammet R. Noterne beskriver regneark i OpenOffice men passer også til det meget udbredte Microsoft Excel. Du kan lovligt og gratis installere en kopi af OpenOffice på din egen pc; OpenOffice er open source software. Eksemplerne i disse noter benytter Windows-udgaven, men OpenOffice fås også til MacOS og til Linux. OpenOffice kan indlæse og gemme Microsoft Excel regneark og mange andre formater.

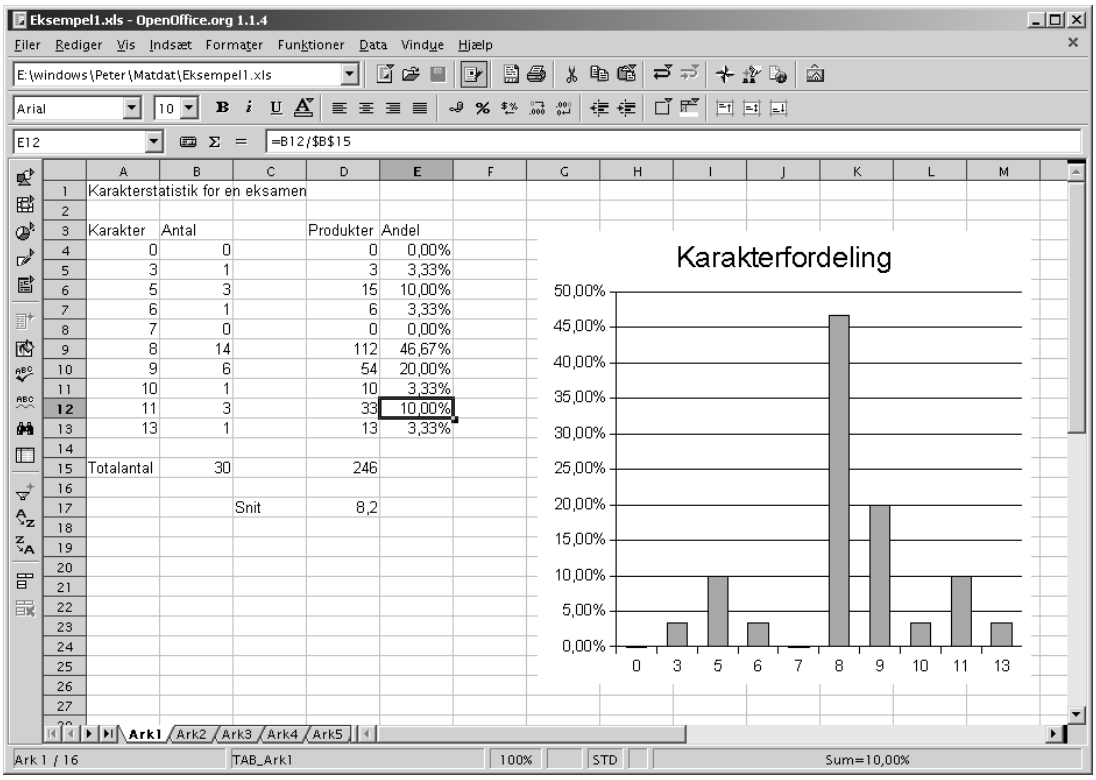
Det bedste udbytte af disse noter fås hvis du selv eksperimenterer med de præsenterede funktioner i regnearket under læsningen. Alle de benyttede eksempelfiler findes her:

`http://www.matfys.kvl.dk/mat-dat/eksempler/`

21 Kom i gang med regneark

Hvis OpenOffice er installeret findes programmet normalt som en lille genvejs-ikon i værktøjslinien nederst til højre på Windows-skrivebordet, ellers under START || PROGRAMMER || OPENOFFICE.ORG || REGNEARK. Se i appendiks B hvordan du kan installere OpenOffice på din egen computer.

1. OpenOffice Regneark startes ved at højreklikke på genvejs-ikonen nederst til højre på Windows-skrivebordet og vælge REGNEARK.
2. Regnearket `Eksempel1.xls` åbnes ved at vælge FILER || ÅBN. Som det ses i figur 36 er regnearket et rektangulært gitter af celler, og hver celle indeholder et tal, en tekst eller en formel.



	A	B	C	D	E
1	Karakterstatistik for en eksamen				
2					
3	Karakter	Antal		Produkter	Andel
4	0	0		0	0,00%
5	3	1		3	3,33%
6	5	3		15	10,00%
7	6	1		6	3,33%
8	7	0		0	0,00%
9	8	14		112	46,67%
10	9	6		54	20,00%
11	10	1		10	3,33%
12	11	3		33	10,00%
13	13	1		13	3,33%
14					
15	Totalantal	30		246	
16					
17			Snit	8,2	

Figur 36: Regneark i OpenOffice med tal, tekst, formler og et søjlediagram.

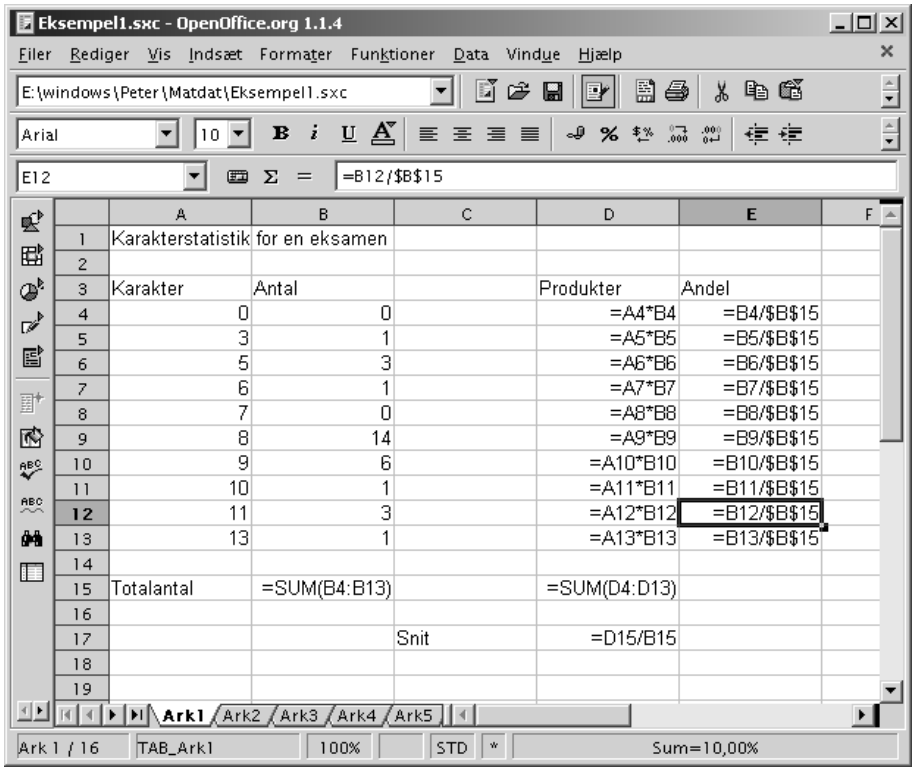
Cellerne A4:A13 indeholder de ti karakterer fra 13-skalaen, mens cellerne B4:B13 indeholder antallet af hver karakter givet ved en bestemt eksamen. For eksempel er der givet 14 otte-taller.

Figur 37 viser alle formlerne fra regnearket i figur 36. Celle B15 indeholder formlen `=SUM(B4:B13)` som beregner summen (30) af cellerne B4:B13, altså det samlede antal karakterer der givet.

Celle D17 beregner karaktergennemsnittet med formlen `=D15/B15`, idet celle D15 beregner karaktersummen med formlen `=SUM(D4:D13)`. Cellerne D4:D13 indeholder hver karakters bidrag, beregnet i fx celle D5 med formlen `=A5*B5`.

Cellerne E4:E13 beregner hver karakters procentvise andel af de givne karakterer, beregnet i fx celle E5 med formlen `=B5/$B$15`.

Søjlediagrammet oven i regnearket er tilføjet ved at markere celleområderne A4:A13 og E4:E13, og vælge INDSÆT || DIAGRAM og FØRSTE KOLONNE SOM ETIKET og SØJLER || NORMAL, hvorefter "Hovedoverskrift" er rettet til "Karakterfordeling".



	A	B	C	D	E	F
1	Karakterstatistik	for en eksamen				
2						
3	Karakter	Antal		Produkter	Andel	
4	0	0		=A4*B4	=B4/$B$15	
5	3	1		=A5*B5	=B5/$B$15	
6	5	3		=A6*B6	=B6/$B$15	
7	6	1		=A7*B7	=B7/$B$15	
8	7	0		=A8*B8	=B8/$B$15	
9	8	14		=A9*B9	=B9/$B$15	
10	9	6		=A10*B10	=B10/$B$15	
11	10	1		=A11*B11	=B11/$B$15	
12	11	3		=A12*B12	=B12/$B$15	
13	13	1		=A13*B13	=B13/$B$15	
14						
15	Totalantal	=SUM(B4:B13)		=SUM(D4:D13)		
16						
17			Snit	=D15/B15		
18						
19						

Figur 37: Visning af formlerne fra regnearket i figur 36.

22 Indhold og visning af celler

22.1 Indtastning af tal, formler og tekst

En celle indeholder enten et tal, en tekst eller en formel. Indholdets art bestemmes af indtastningens form:

Art	Form
Tal	Cifre, komma, fortegn, videnskabelig notation (`6,02E+23`)
Formel	Begynder med lighedstegn (`=`)
Tekst	Alt andet end tal og formel; eller begynder med enkelt anførselstegn (`'`)

Bemærk at i dansk OpenOffice og Excel skal tal indtastes med decimalkomma `6,45` (som på dansk, tysk og fransk) snarere end decimalpunktum `6.45` (som på amerikansk og engelsk). Hvis man indtaster `6.45` i dansk OpenOffice risikerer man at få den mærkelige resultat `01-06-45`, altså 1. juni 1945. Det er fordi OpenOffice forsøger at tolke indtastningen som en dato og skifter til datoformat i cellen; se afsnit 22.7.

I dansk Excel bliver `6.45` normalt opfattet som en tekst. Det viser sig typisk ved at tekster bliver venstrestillet mens tal bliver højrestillet i cellen.

22.2 Kopiering af cellers indhold

Indholdet af en celle, fx D4 i figur 37, kan kopieres til alle celler i et celleområde, fx D5:D13 på én gang:

- Marker den oprindelige celle og tryk `Ctrl+C`, eller højreklik og vælg KOPIER.
- Marker celleområdet du ønsker at kopiere til og tryk `Ctrl+V`, eller højreklik og vælg `Sæt ind`.

22.3 Absolutte og relative cellereferencer

I formler bruges cellereferencer til at referere til en celle (fx `B4`) eller til et celleområde (fx `B4:B13`). En cellereference `B4` består af et søjlebogstav (`B`) og et rækkenummer (`4`). Formlen `=A4*B4` i celle D4 i figur 37 indeholder således de to cellereferencer `A4` og `B4`.

Når man kopierer en formel der indeholder en cellereference `B4`, så justeres både søjle og række i cellereferencen. Det er nyttigt fordi man kan nøjes med at skrive formlen `=A4*B4` én gang i celle D4, og derefter markere og kopiere den til cellerne D5:D13. Herved justeres formlens cellereferencer automatisk, så formlen i celle D5 bliver `=A5*B5` og så videre.

Man kan undgå den automatiske justering af cellereferencer ved at sætte et dollartegn ($) foran søjlebogstav eller rækkenummer eller begge dele:

Reference	Betegnelse	Effekt ved kopiering
`B15`	Relativ	Både søjle og række justeres
`$B$15`	Absolut	Hverken søjle eller række justeres
`B$15`	Række absolut	Kun søjle justeres
`$B15`	Søjle absolut	Kun række justeres

For eksempel er formlerne i cellerne E4:E13 i figur 37 dannet ved at formlen `=B4/$B$15` er skrevet i celle E4. Derefter er formlen kopieret til E5:E13 hvorved cellereferencen `B4` justeres til `B5` osv. mens `$B$15` fastholdes som den er. Eftersom formålet med formlerne i E4:E13 er at beregne hver karakters andel af totalen, som står i celle B15, ville det ikke give nogen mening at justere referencen til B15.

Når man redigerer en formel kan man i OpenOffice bruge tasten `Shift+F4` til at skifte fra `B15` til `$B$15` til `B$15` til `$B15` og tilbage til `B15`; i Excel bruges bare `F4`.

22.4 Symbolske cellereferencer og navngivne celler

Hvis en formel indeholder referencer til mange andre celler, kan den blive meget svær at læse. Så er det bedre at bruge *symbolske cellereferencer*. Lad os sige at vi vil beregne den såkaldte logistiske funktion for mange forskellige værdier af t:

$$logis(t) = \frac{K}{1 + ae^{-rt}}$$

Antag at konstanterne $a = 50$, $r = 1$ og $K = 1$ står i cellerne C1, C2 og C3, at A6:A105 indeholder nogle t-værdier, og at vi ønsker at beregne den logistiske funktion i B6:B105. Vi kan nu skrive formlen:

```
=$C$3/(1+$C$1*EKSP(-$C$2*A6))
```

i celle B6 og kopiere den til cellerne B7:B105. Det virker, men det er svært at se om der ved en fejl er byttet om på a eller r eller K i denne formel i forhold den matematiske definition ovenfor.

Bedre er det at navngive cellerne der indeholder a, r og K og bruge symbolske cellereferencer. Marker C1 og vælg INDSÆT || NAVNE || DEFINER || A og tilsvarende for C2 og C3. Så kan formlen i B6 i stedet skrives sådan her:

```
=K/(1+A*EKSP(-R*A6))
```

Det er meget klarere at dette svarer til den matematiske definition ovenfor. Symbolske cellereferencer er altid absolutte, som om der var dollartegn foran både søjle og række, så den nye formel kan kopieres til B7:B105.

22.5 Tastaturgenveje til navigation, markering og redigering

Hvis man arbejder meget med regneark er det hurtigere (og mindre skadeligt for ens albue og skulder) at bruge tastaturet end at bruge musen. Her er nogle vink til effektiv brug af tastaturet:

Effekt	Tast
Flyt fokus helt til venstre (søjle A)	`Home`
Flyt fokus til øverste venstre celle (A1)	`Ctrl+Home`
Flyt fokus helt til højre (sidste søjle)	`End`
Flyt fokus til nederste højre celle	`Ctrl+End`
Flyt fokus i eller mellem celleklumper	`Ctrl`+piletaster
Marker celleområde	`Shift`+piletaster
Markér flere celleområder	`Shift+F8`
Marker hel række	`Shift`+mellemrum
Marker hel søjle	`Ctrl`+mellemrum
Kopiér indhold af markerede celler	`Ctrl+C`
Klip indhold fra markerede celler	`Ctrl+X`
Indsæt celleindhold	`Ctrl+V`
Redigér celleindhold	`F2`

Man kan kombinere brugen af `Shift` og `Ctrl`, så fx `Ctrl+Home` efterfulgt af `Shift+Ctrl+End` markerer hele den aktive del af regnearket. Derefter kan man bruge VIS || ZOOM || OPTIMAL til at zoome på netop den aktive del af arket. (Dette påvirker ikke udskrift af regnearket; se i stedet afsnit 25).

I meget store regneark kan man med fordel bruge VINDUE || OPDEL til at navigere rundt i adskilte dele af regnearket samtidig.

22.6 Talformater: antal decimaler, venstre- og højrestilling

Når en celle har fokus markeres den med indramning i regnearket, og dens indhold (tal, formel, tekst) vises i formellinien over selve regnearket. I figur 36 er der fokus på celle E12, og formellinien viser at cellen indeholder formlen `=B12/$B$15`. Man kan få vist alle formler som i figur 37 ved at vælge FUNKTIONER || INDSTILLINGER || REGNEARK || VISNING || VIS || FORMLER.

I selve regnearket vises resultatet af at udregne formlen. Cellens format bestemmer hvordan resultatet vises. Ved at vælge fx FORMATER || CELLER || TAL || TAL || ANTAL DECIMALER || 2 kan man få cellens værdi afrundet til 2 decimaler. Der er to grunde til at afrunde til færre decimaler: dels gør det tallene mere overskuelige, og dels er det misvisende at angive fx et måleresultat med 8 decimaler. Antallet af viste decimaler påvirker ikke regnenøjagtigheden.

Man kan angive at indholdet af en celle skal venstrestilles, højrestilles eller centreres (og meget andet) med FORMATER || CELLER || JUSTERING || VANDRET || ....

Hvis man vil have karaktererne 00 og 03 vist korrekt i regnearket i figur 36 kan man markere cellerne A4:A5 og vælge FORMATER || CELLER || TAL || TAL || FORANSTILLEDE NULLER || 2.

22.7 Talformater: procenter, datoer og klokkeslet

Tal kan vises på nogle specielle måder: som procenter, som datoer, og som klokkeslet.

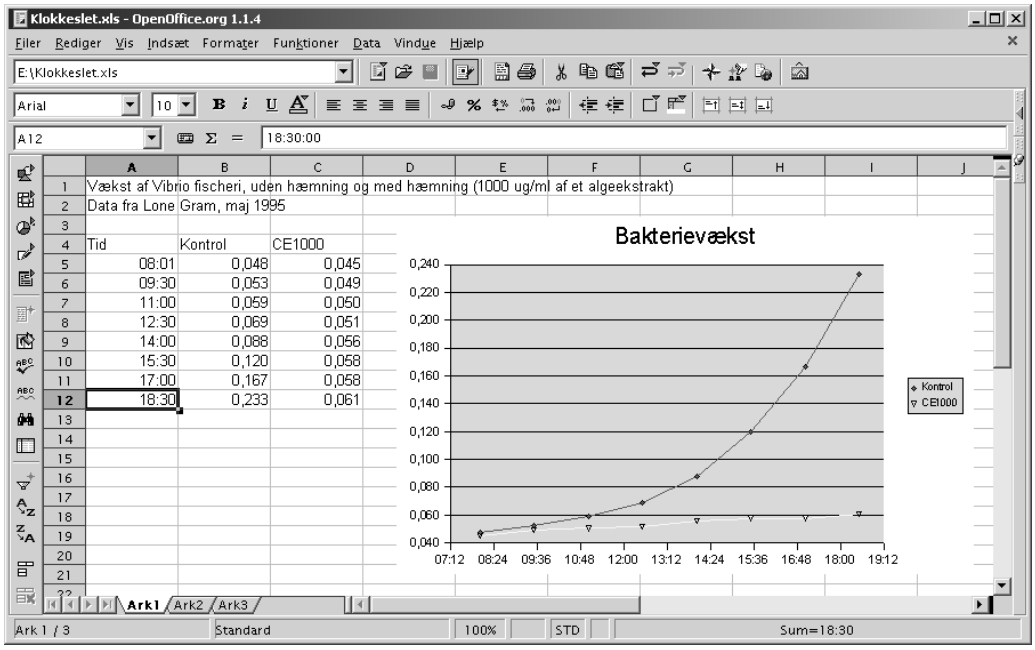
I figur 36 er der anvendt procentformat i cellerne E4:E13. Det er gjort ved at markere cellerne og vælge FORMATER || CELLER || TAL || PROCENT. Procentformat viser cellens indhold gange 100, efterfulgt af et procenttegn (%). Bemærk at procentvisningen kun påvirker hvordan cellens værdi vises, og ikke har

nogen effekt på beregninger hvori værdien indgår. Selv om celle E12 vises som `10.00%` så er dens værdi altså 0.1.

Datoformat viser antal dage siden 0. januar 1900, så 9. april 1940 er faktisk tallet 14710. Eftersom datoer blot er tal vist i et bestemt format, så kan man regne med datoer. Ved at finde differensen mellem to celler, hvoraf den ene indeholder 5.5.1945 og den anden indeholder 9.4.1940 kan man konstatere at anden verdenskrig varede 1852 dage (i Danmark).

Klokkesletformat viser brøkdele af døgn siden midnat, så 18:00 er i virkeligheden tallet 0.75, og kl 18:00 den 9. april 1940 er bare tallet 14710.75. Dermed kan man også regne med klokkeslet; forskellen mellem to klokkeslet er den brøkdel af et døgn der er mellem dem. Hvis man ganger den brøkdel med 1440 får man antal minutter mellem klokkeslettene.

Det vil sige at hvis man i laboratoriet aflæser fx bakterietæthed sammen med klokkeslet, som vist i figur 38, så kan man nemt beregne antal minutter mellem to aflæsninger. Hvis man har daterede målinger fra et vækstforsøg i mark eller skov så kan man beregne antal dage (og brøkdele af dage) mellem målingerne. Ligeledes kan datoer og klokkeslet uden videre bruges på første-aksen i xy-punktdiagrammer (afsnit 24), og man kan lave regression på aflæsningerne som funktion af tiden (afsnit 28.4).



Figur 38: Måleresultater med aflæsningstidspunkt i klokkesletformat, og et diagram.

22.8 Tekstformater

Tekster kan højrestilles osv. aldeles som tal, og man kan vælge typografi, fede typer, kursiv, tekstfarve, baggrundsfarve og meget meget andet, som i et tekstbehandlingsprogram. Nogle muligheder der er særlig nyttige i regneark er disse:

Effekt	Fremgangsmåde
Centrer tekst over flere celler	Marker cellerne, vælg FORMATER || FLET CELLER || DEFINER, og derefter FORMATER || CELLER || JUSTERING || CENTRER
Celle med flere linier tekst	FORMATER || CELLER || JUSTERING || AUTOMATISK LINIESKIFT

22.9 Serier

Regneark har særlige faciliteter til at oprette simple serier af værdier, såsom 1, 2, . . . , eller januar, februar, . . . , eller mandag, tirsdag, ... Man skriver det første led af serien i en celle, markerer cellen, og trækker så ud i det såkaldte fyldhåndtag nederst til højre i markeringsboksen. Derved udfyldes cellerne med værdier fra serien så langt man trækker i fyldhåndtaget.

Man kan også lave talserier med andre skridtlængder, fx 0.5, ved at udfylde to naboceller med 0.5 og 1.0, markere begge celler, og trække ud i fyldhåndtaget. Mere komplicerede serier kan fremstilles med REDIGER || FYLD || SERIER.

Hvis den oprindelige celle indholder en formel, så kan fyldhåndtaget bruges til at kopiere formlen til et tilstødende celleområde.

22.10 Genberegning og cirkularitet

Grundideen i et regneark er at en celle kan indeholde en formel der refererer til værdien i andre celler, som igen kan indeholde formler osv. Lige så vigtigt er det at når man ændrer indholdet af en celle (tal, formel eller tekst) i regnearket, så genberegnes alle celler i regnearket automatisk — i den rigtige rækkefølge — så deres værdier altid er korrekte.

Man kan eksplicit bestille en genberegning af alle celler ved at trykke `F9`; det er nyttigt hvis man laver simuleringer med `SLUMP()`-funktionen. Hvis regnearket indeholder tusinder af formler kan redigering virke tungt fordi der hele tiden genberegnes; så kan det hjælpe at slå automatisk genberegning fra med FUNKTIONER || CELLEINDHOLD || AUTOMATISK GENBEREGNING.

Hvis der er cirkulære referencer i regnearket, altså hvis en celle afhænger af sig selv, så duer genberegningsmekanismen imidlertid ikke. Prøv fx at indtaste formlen `=A2+1` i celle A1 og dernæst indtaste formlen `=A1*7` i celle A2. I OpenOffice viser en cirkulær afhængighed sig som en mystisk `Fejl:522` i de berørte celler. Brug FUNKTIONER || DETEKTIV || SPOR FEJL til at finde ud af hvilke celler der er involveret; se afsnit 28.1. I Excel kommer der en fejlboks op hvis man forsøger at lave en cirkulær afhængighed.

23 Indlæsning af data

OpenOffice kan læse Excel-regnearksmapper (`.xls`-filer), kommaseparerede filer (`.csv`-filer), OpenOffice regneark (`.sxc`-filer) og mange andre formater, og kan gemme igen i de fleste af disse formater.

En tekstfil i kommasepareret format (`.csv`-fil) indlæses med FILER || ÅBN hvorefter der dukker en tekstimport-boks op. Her skal man angive skilletegnsindstillinger. For filer i dansk kommasepareret format (eksempelfil `Grise2KommaDK.csv`) skal man angive DELT MED || SEMIKOLON, ikke delt med komma.

Filer i amerikansk kommasepareret format (eksempelfil `Grise2KommaUS.csv`) kan ikke indlæses i dansk OpenOffice. Derimod kan de godt indlæses i dansk Excel, for der kan man angive at filen bruger decimalpunktum (`6.45`) i stedet for decimalkomma (`6,45`).

Filer med type `.txt` skal indlæses ved at man vælger FILER || ÅBN || FILTYPE || TEKST CSV (*.CSV; *.TXT) og derefter filnavn, hvorefter der dukker en tekstimport-boks op. For filer i fastbreddeformat (eksempelfil `Grise2Fast.txt`) skal man angive FAST BREDDE og dernæst sætte kolonneskillestreger i vinduet nederst i tekstimport-boksen.

Undertiden vender de indlæste data "på den forkerte led", dvs. man ville gerne bytte om på rækker og søjler. Dette kan gøres ved transponering. Indlæs dataene som de er, marker de indlæste data og kopier dem med `Ctrl+C` eller højreklik efterfulgt af KOPIER. Vælg et ledigt område af regnearket eller et nyt ark, og udfør REDIGER || INDSÆT SPECIEL || TRANSPONER.

24 Diagrammer og plot

Diagramfaciliteterne i regneark er mest egnet til forretningsgrafik, men kan også bruges til fremstilling af forsøgsdata hvis man tænker sig om. Tre hovedtyper af diagrammer er særlig nyttige:

Type	Anvendelse
XY-diagram	Bruges til plot af sammenhørende værdier af fx tid og aflæsning. Brug enten undertypen der kun har datamærker eller undertypen der har linier og datamærker. Undertypen "kubisk spline" giver nydelige glatte kurver, men må ikke bruges til forsøgsdata.
Søjlediagram	Bruges til at sammenligne data for forskellige diskrete kategorier, fx at sammenligne hugsten af egetræ i 12 forskellige amter. Søjlerne kan være sammensatte, hvad der kan bruges til at vise hugst af både bøg og eg for hvert af 12 amter. Søjlediagrammer med 3D-effekter forvansker data og bør ikke bruges.
Cirkeldiagram	Viser den andel som forskellige størrelser udgør af en helhed, fx hvordan brugen af egetræ fordeler sig på gavntræ, brænde og skovflis. Cirkeldiagrammer er meget anvendte, men kritiseres af kognitionspsykologer, som siger at det er svært visuelt at opfatte forholdet mellem arealer korrekt. Brug hellere søjlediagrammer. Cirkeldiagrammer med 3D-effekter ("lagkagediagrammer") forvansker data yderligere og bør slet ikke bruges.

Hverken OpenOffice eller Excel har mulighed for at plotte en variabel som funktion af to andre. Hertil må man bruge fx `scatterplot3d` i R-programmet.

Diagrammer oprettes ved at man markerer data i regnearket og vælger INDSÆT || DIAGRAM, vælger diagramtype, undertype, overskrifter osv. De fleste af disse valg kan tilpasses i det færdige diagram ved at højreklikke på titler, akser, dataserier, baggrund osv.

I Excel kan man tilføje en tendenslinie til en dataserie i et XY-diagram ved at højreklikke på dataserien; i realiteten beregner Excel først en lineær regression og tegner så den fundne kurve.

25 Udskrift af regneark og eksport til PDF

Start med FILER || VIS || UDSKRIFT for at sikre at papiret bruges bedst muligt. For eksempel er det ofte en god idé at vælge SIDEFORMAT || SIDE || RETNING || LIGGENDE og måske SIDEFORMAT || ARK || TILPAS UDSKRIFT TIL SIDETAL || 1. Lav hellere en udskrift med reduceret skriftstørrelse end en udskrift hvor der er 12 kolonner af en tabel på ét ark og kun 1 kolonne af tabellen på det næste ark.

Hvis man skal distribuere et regneark til andre og gerne vil sikre at det ser ens ud for alle modtagere, så er det en god idé at distribuere det i PDF-format: vælg FILER || EKSPORTER SOM PDF. Det svarer til at sende en udskrift på elektronisk form i stedet for papir, og duer altså kun hvis modtagerne ikke skal kunne rette i regnearket. Men det er på den anden side nyttigt hvis man udsender et regnskab og vil forhindre at modtagerne retter i det ved en fejl.

Man kan få kolonneoverskrifterne til en (lang) tabel gentaget på hvert ark i udskriften ved at vælge FORMATER || UDSKRIFTSOMRÅDER || REDIGER || RÆKKER DER SKAL GENTAGES || BRUGERDEFINERET og så udpege den eller de rækker (typisk `$1`) der skal med på hvert ark. Det samme kan gøres for søjler.

I Excel skal man i stedet gå ind i FILER || SIDEOPSÆTNING || ARK || UDSKRIFTSTITLER. Samme indstillingsboks findes, men kan sært nok ikke bruges, under FILER || VIS UDSKRIFT || INDSTIL || ARK.

26 Indbyggede funktioner

OpenOffice Regneark og Microsoft Excel har et meget stort antal indbyggede funktioner, som kan ses under INDSÆT || FUNKTION eller INDSÆT || FUNKTIONSLISTE. De mest brugte numeriske funktioner er vist i figur 39 og de mest brugte statistiske funktioner er vist i figur 40.

Funktion i regneark	Matematik	Betydning
`x + y`	$x+y$	x plus y
`x - y`	$x-y$	x minus y
`x * y`	$x \cdot y$	x gange y
`x / y`	x/y	x divideret med y
`x ^ y`	x^y	x opløftet i y
`x%`	$x/100$	x divideret med 100
`ABS(x)`	$\|x\|$	numerisk (absolut) værdi af x
`KVROD(x)`	$\sqrt{x}$	kvadratrod af x
`LN(x)`	$\ln(x)$	naturlig logaritme af x
`LOG10(x)`	$\log(x)$	titalslogaritme af x
`EKSP(x)`	e^x	eksponentialfunktionen af x, dvs. e^x
`SIN(x)`	$\sin(x)$	sinus til x radianer
`COS(x)`	$\cos(x)$	cosinus til x radianer
`TAN(x)`	$\tan(x)$	tangens til x radianer
`ARCSIN(x)`	$\sin^{-1}(x)$	arcus sinus til x
`ARCCOS(x)`	$\cos^{-1}(x)$	arcus cosinus til x
`ARCTAN(x)`	$\tan^{-1}(x)$	arcus tangens til x
`PI()`	π	enhedscirklens areal $\pi = 3.14159\ldots$
`EKSP(1)`	e	den naturlige logaritmes grundtal $e = 2.718282\ldots$
`AFRUND(x; d)`		afrund x til d decimaler
`AFRUND.GULV(x; y)`		rund x ned til nærmeste multiplum af y
`AFRUND.LOFT(x; y)`		rund x op til nærmeste multiplum af y

Figur 39: Numeriske operatorer og funktioner i regneark.

Funktion i regneark	Resultat
`SLUMP()`	Et tilfældigt tal i $[0, 1]$ fra ligefordelingen
`PRODUKT(v)`	Produktet af elementerne i celleområdet v
`SUM(v)`	Summen af elementerne i celleområdet v
`MIDDEL(v)`	Middelværdi af elementerne i celleområdet v
`VARIANS(v)`	Varians af elementerne i celleområdet v
`MIN(v)`	Minimum af elementerne i celleområdet v
`MAKS(v)`	Maksimum af elementerne i celleområdet v

Figur 40: Nogle simple statistiske funktioner i regneark.

27 Disposition og subtotaler

Disposition kan bruges til at skjule rækker i regnearket som (kun) er der for at beregne indholdet af andre rækker. For eksempel kan man skjule underposter i et regnskab så kun hovedposterne vises. Disposition aktiveres med DATA || DISPOSITION || GRUPPERING.

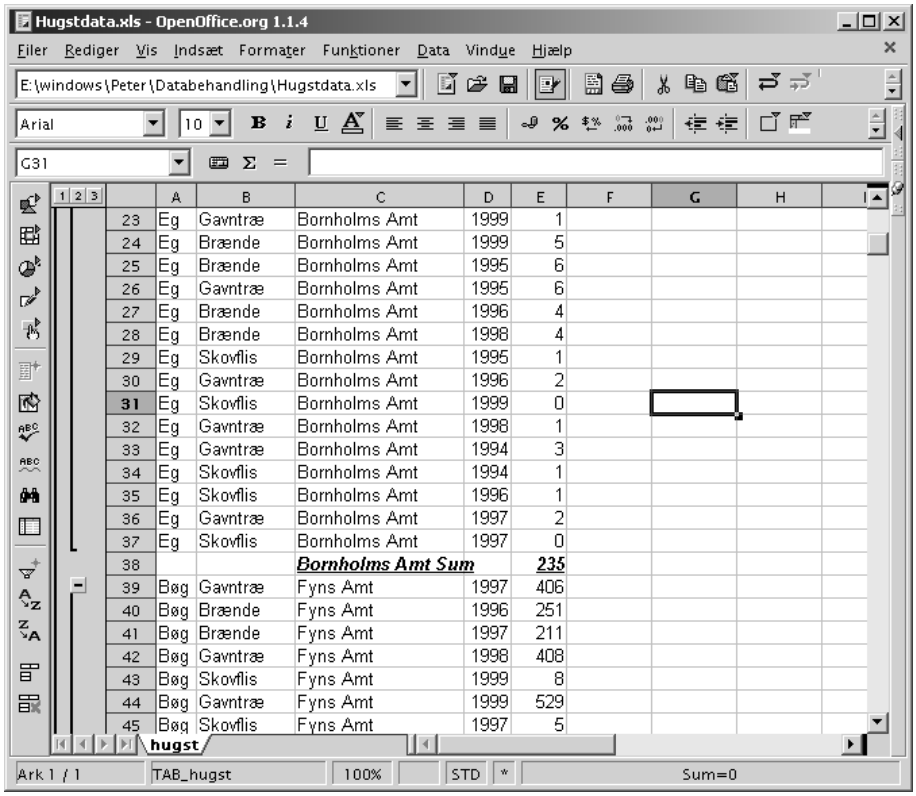
Subtotaler bruges til at tilføje nye rækker som er beregnet ud fra eksisterende rækker i et datasæt. Nedenfor er vist et datasæt `Hugstdata.xls` med fire uafhængige variable `Art`, `Anv`, `Amt` og `År`, og én afhængig variabel `Hugst`. Der er 2 træarter gange 3 anvendelser gange 12 amter gange 6 årstal, ialt 432 observationer:

```
Art     Anv      Amt                   År      Hugst
Bøg     Gavntræ  Hovedstadsregionen     1994    581
Bøg     Gavntræ  Vestsjællands Amt      1994    618
...
Bøg     Brænde   Hovedstadsregionen     1994    215
Bøg     Brænde   Vestsjællands Amt      1994    295
...
Bøg     Skovflis Hovedstadsregionen     1994    2
Bøg     Skovflis Vestsjællands Amt      1994    3
...
Eg      Gavntræ  Hovedstadsregionen     1994    103
Eg      Gavntræ  Vestsjællands Amt      1994    85
...
```

Med subtotaler kan man fx nemt beregne den samlede hugst for hvert amt for sig (men summeret over alle kombinationer af træarter, anvendelser og år). Man skal vælge DATA || SUBTOTALER || GRUPPER EFTER || AMT og BEREGN SUBTOTALER FOR || HUGST og BRUG FUNKTION || SUM. Derved sorteres dataene efter kolonnen `Amt`, og der indsættes 13 nye rækker: en subtotal for hvert amt og en hovedtotal. Se figur 41, hvor række 38 er en automatisk indsat subtotal.

Desuden oprettes der automatisk disposition så man kan skjule alt andet end subtotalerne (ved at trykke på boksen 2 i dispositionskolonnen yderst til venstre i regnearket) eller alt andet end hovedtotalen (ved at trykke på boksen 1 i dispositionskolonnen); se figur 42. Læg mærke til at rækkenumrene ikke er fortløbende når rækker skjules med disposition.

Man kan fjerne subtotalerne og den tilhørende disposition igen med DATA || SUBTOTALER || SLET.



	A	B	C	D	E
23	Eg	Gavntræ	Bornholms Amt	1999	1
24	Eg	Brænde	Bornholms Amt	1999	5
25	Eg	Brænde	Bornholms Amt	1995	6
26	Eg	Gavntræ	Bornholms Amt	1995	6
27	Eg	Brænde	Bornholms Amt	1996	4
28	Eg	Brænde	Bornholms Amt	1998	4
29	Eg	Skovflis	Bornholms Amt	1995	1
30	Eg	Gavntræ	Bornholms Amt	1996	2
31	Eg	Skovflis	Bornholms Amt	1999	0
32	Eg	Gavntræ	Bornholms Amt	1998	1
33	Eg	Gavntræ	Bornholms Amt	1994	3
34	Eg	Skovflis	Bornholms Amt	1994	1
35	Eg	Skovflis	Bornholms Amt	1996	1
36	Eg	Gavntræ	Bornholms Amt	1997	2
37	Eg	Skovflis	Bornholms Amt	1997	0
38			***Bornholms Amt Sum***		***235***
39	Bøg	Gavntræ	Fyns Amt	1997	406
40	Bøg	Brænde	Fyns Amt	1996	251
41	Bøg	Brænde	Fyns Amt	1997	211
42	Bøg	Gavntræ	Fyns Amt	1998	408
43	Bøg	Skovflis	Fyns Amt	1999	8
44	Bøg	Gavntræ	Fyns Amt	1999	529
45	Bøg	Skovflis	Fyns Amt	1997	5

Figur 41: Hugstdatasættet efter tilføjelse af subtotaler for hvert amt.

28 Avancerede regnearksfaciliteter

Her nævnes kort nogle få af de talrige andre faciliteter og indbyggede funktioner der kan være nyttige i naturvidenskabelig og økonomisk databehandling.

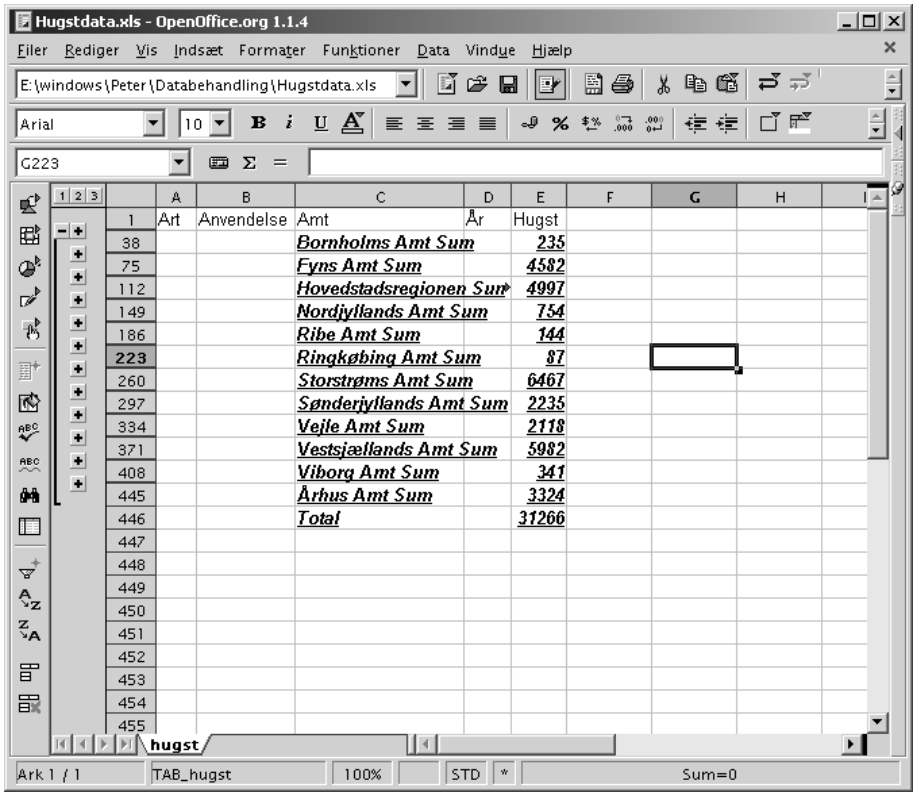
28.1 Detektiv (Revision)

Under FUNKTIONER || DETEKTIV finder man en facilitet til at vise hvilke ("overordnede") celler en given celle afhænger af, og til at vise hvilke andre ("underordnede") celler der afhænger af en given celle. Det er særdeles nyttigt til at opspore fejl i komplicerede regneark. I Excel hedder den samme facilitet FUNKTIONER || REVISION.

28.2 Pivottabeller

Pivottabeller er en meget effektiv måde at danne forskellige subtotaler i et datasæt med mange uafhængige variable. Betragt igen datasættet fra regnearket `Hugstdata.xls` vist i afsnit 27. Der er $2^4 = 16$ forskellige måder at beregne subtotaler i dette datasæt, hvert bestemt af hvilke af de 4 uafhængige variable `Art`, `Anv`, `Amt` og `År` der inddrages.

For at opbygge en pivottabel åbner man datasættet i et regneark, sætter fokus i datasættet og vælger DATA || DATAPILOT || START. Træk derefter en eller flere afhængige variable (her kun `Hugst`) til pivottabellens datafelt og træk en eller flere uafhængige variable til pivottabellens søjle- og række-overskrifter. Der bliver så beregnet subtotaler for alle kombinationer af de uafhængige variable der er taget med, med summation (eller gennemsnit eller lignende) over alle dem der ikke er taget med.



Figur 42: Hugstdatasættet efter disposition har skjult alt andet end subtotalerne.

Pivottabeller i Excel fungerer på næsten samme måde, og startes med og startes med DATA || PIVOTTABEL OG PIVOTDIAGRAM. I Excel er der yderligere mulighed for nemt at lave søjlediagrammer over beregnede pivottabeller.

28.3 Målsøgning

Med FUNKTIONER || MÅLSØGNING kan man finde en numerisk løsning til en ligning, ligesom med R-funktionen `uniroot`.

28.4 Matrixformler

En normal regnearksformel producerer et resultat der kan være i én celle. En matrixformel er en særlig formel hvis resultat kan fylde flere celler. For eksempel beregnes lineær regression $y = b + ax$ med en matrixformel, for resultatet består af to tal: hældningen a og skæringspunktet b. Ud over lineær regression kan matrixformler bruges til at beregne sum, produkt, invers, determinant, transponering osv. for matricer; se INDSÆT || FUNKTION || KATEGORI || MATRIX. Til de fleste af disse formål er R-programmet dog bedre egnet.

En matrixformel indtastes ved at man markerer de celler, fx A15:B15, som formlens resultater skal lagres i, indtaster formlen fx `=LINREGR(B4:B8; A4:A8)` og derefter trykker `Ctrl+Shift+Enter`. Dette sidste skridt er helt afgørende.

Efter indtastningen vises matrixformlen i krølleparenteser, fx `{=LINREGR(B4:B8; A4:A8)}`, men krølleparenteserne skal man altså ikke selv skrive. I eksemplet med lineær regression kommer hældningen a i celle A15 og skæringspunktet b i celle B15.

29 Opgaver til regneark

Opgaver til modul D

Opgave Dat-D-1 — Smørfedt

Nedenfor er der for 13 portioner mælk vist portionen størrelse (kg) og procent smørfedt i mælken. Start regnearksprogrammet og opret et nyt tomt regneark. Lav plads til kolonneoverskrifter og indtast nedenstående tal i regnearket i to kolonner:

22,9	4,4
25,2	4,5
22,5	4,65
21	4,7
20,1	4,75
19	4,85
18,2	4,9
17	5
15,5	5,15
13,9	5,2
10,2	5,35
5,7	5,75
0,5	6,05

Lav en separat kolonne med formler til at beregne fedtindholdet (kg) i hver portion mælk. Indtast formler at beregne den mælkemængde (kg), det samlede fedtindhold (kg), og den gennemsnitlige smørfedtprocent.

Hvorfor er det forkert at beregne den gennemsnitlige smørfedtprocent som middelværdien af smørfedtprocenterne?

Opgave Dat-D-2 — Rentes rente

Opstil et regneark der viser hvordan en kapital på 20 000 kroner vokser over 10 år når der tilskrives 4 procent i rente hvert år. Du skal *ikke* bruge regnearksprogrammets indbyggede finansielle funktioner. Det er en fordel at skelne mellem summen ved årets begyndelse (primosaldo) og summen ved årets slutning (ultimosaldo) — ellers kommer du nemt til at spare op i 9 eller 11 år i stedet for 10.

Opgave Dat-D-3 — Opsparing med årligt indskud

Opstil et regneark der viser hvordan en kapital vokser over 10 år når der i begyndelsen af hvert år indsættes 10 000 kroner på kontoen, og i slutningen af hvert år tilskrives 4 procent i rente af den aktuelle saldo. Du skal *ikke* bruge de indbyggede finansielle funktioner. Også her er det en fordel at skelne mellem primosaldo og ultimosaldo.

Lav regnearket sådan at man nemt kan ændre det årlige indskud og rentefoden: Skriv indskuddet i celle B2 og rentefoden i celle B3, sådan at når man redigerer en af de celler, så genberegnes opsparingen automatisk.

Lav et søjlediagram som viser hvordan opsparingen vokser med tiden.

Aflever udskrift søjlediagrammet, og udskrifter af regnearket både så det viser tallene, og så det viser de underliggende formler.

Opgave Dat-D-4 — Subtotaler, disposition og diagrammer

Regnearket `regneark/Webhit.xls` fra kursushjemmesidens eksempelsamling viser for hver dato hvor mange daglige hit (opslag) der var på en bestemt hjemmeside på World Wide Web.

(1) Lav først en xy-punktkurve der viser hvordan antal opslag udvikler sig dag for dag.

(2) Nu skal du lave et søjlediagram der viser hvordan det gennemsnitlige antal hit pr. dag udvikler sig måned for måned. Gennemsnittene skal beregnes som subtotaler.

Indsæt først en ny kolonne B med INDSÆT || KOLONNER, og brug den indbyggede funktion `MÅNED` til at beregne måneden svarende til hver dato i den kolonne. (Hvordan forhindrer du at resultatet vises som en dato?)

Sørg for at alle tre kolonner har passende overskrifter, og brug derefter subtotaler til at beregne det gennemsnitlige antal hit pr. dag inden for hver måned.

Brug disposition til at skjule alt andet end de beregnede gennemsnit. Lav endelig søjlediagrammet der viser hvordan det gennemsnitlige antal hit pr. dag udvikler sig måned for måned.

(3) Nu skal der på tilsvarende måde laves et cirkeldiagram der viser hvordan opslagene er fordelt på de syv ugedage. Start forfra i et nyt regneark. Beregn i en ny kolonne ugedagen for hver dato med funktionen `UGEDAG` (hvor søndag=1, ..., lørdag=7). Sorter efter ugedag, brug subtotaler til at beregne gennemsnitligt antal hit per dag for hver af de syv ugedage, og lav til slut et cirkeldiagram.

Aflever udskrifter af de tre grafer.

Opgave Dat-D-5 — Indlæsning af forsøgsdata fra tekstfil

Denne opgave handler om indlæsning af tekstfilen `data/Grise2Fast.txt` fra kursushjemmesidens eksempelsamling. Se opgave Dat-A-13 for en beskrivelse.

Indlæs tekstfilen i et nyt regneark som forklaret i afsnit 23.

Lav en xy-punktdiagram der viser både kontrolgruppens pH og vækshormongruppens pH som funktion af aflæsningstiden.

Lav et nyt tekstdokument (med OpenOffice eller Microsoft Word) og indsæt diagrammet i tekstdokumentet. Tilføj lidt forklaring.

Aflever udskrift af xy-punktdiagrammet.

Opgave Dat-D-6 — Matrixformler og lineær regression

Eksemplet `regneark/Soedmaelk.xls` indeholder data om forbruget af sødmælk i kg per person per år for årene 1990–2000. Tallene viser et støt fald i forbruget år for år. Brug matrixfunktionen LINREGR og matrixformler (husk `Ctrl+Shift+Enter`) til at lave lineær regression på sødmælksforbruget som funktion af årstallet, dvs. bestem koefficienterne a og b i ligningen: $y = ax + b$ hvor y er sødmælksforbruget og x er årstallet. Beregn dernæst det forventede sødmælksforbrug per person i år 2010 og 2015 (under forudsætning af at den fundne tendens holder). Tallene skal beregnes, ikke aflæses på en graf.

Aflever et regneark der viser de to koefficienter a og b, og de beregnede sødmælksforbrug for 2010 og 2015.

Opgave Dat-D-7 — Målsøgning

(1) Løs opgave Dat-D-3 og brug Målsøgning til regne på opsparing. Find først den årlige rente der er nødvendig for at man ender med en halv million kroner efter 10 år når man sparer 10000 kroner op om året. (Den rente minder påfaldende om den man betaler på kviklån og kontokort i forretninger).

Find derefter den årlige opsparing der er nødvendig for at man efter 10 år ender med en halv million kroner når renten kun er 4,5 % (som er nogenlunde det man får i en pensionskasse).

(2) Brug Målsøgning til at finde en løsning til følgende ligning:

$$x^5 - 8x^4 - 24x^3 + 242x^2 - 25x = 1050$$

Denne ligning har fem løsninger, men Målsøgning finder kun én — det er en af svaghederne ved Målsøgning.

Aflever et regneark der viser løsningen på begge delopgaver, med en kort forklaring på hvad du har gjort.

Opgave Dat-D-8 — Pivottabeller og pivotdiagrammer

Lav en pivottabel ud af eksemplet `regneark/Hugstdata.xls` fra kursushjemmesidens eksempelsamling.

Brug Art, Anvendelse, Amt og År, eller blot nogle af dem, som uafhængige variable, og brug (sum af) Hugst som afhængig variabel (data). Eksperimentér med at anbringe de fire indekser som rækkeindeks (til venstre), kolonneindeks (for oven til højre) og sideindeks (for oven til venstre).

Lav følgende tabeller; udskriv og aflever en af dem:

- En tabel der for hver træart (bøg, eg) viser hugsten summeret over alle amter, anvendelser og år. Der skal altså kun være 2 tal i resultatet, svarende til de 2 træarter.
- En tabel der for hver kombination af træart (bøg, eg) og amt viser hugsten summeret over alle anvendelser og år. Der skal altså være et 24 tal i resultatet (2 træarter gange 12 amter).
- En tabel der for hver kombination af anvendelse (gavntræ, brænde, skovflis) og 6 år viser hugsten summeret over træart og amter. Der skal være 18 tal i resultatet.
- En tabel der kun for bøg og for hver kombination af amt og år viser hugsten summeret over alle anvendelser. Der skal være 72 tal i resultatet.

APPENDIKS

A Sådan installerer du R

A.1 Grundlæggende installation af R

Hjemmesiden for R er `http://www.r-project.org/` hvorfra man kan hente den nyeste udgave af R, samt manualer, ekstra pakker osv. Der findes en kopi af hele hjemmesiden i Danmark på adressen `http://mirrors.sunsite.dk/cran/`. Brug denne adresse hvis du skal installere R.

Gør sådan her for at installere R under Microsoft Windows:

- Gå til `http://mirrors.sunsite.dk/cran/`
- Klik på Windows (95 and later) under Download and Install R.
- Klik på base
- Klik på R-2.3.0-win32.exe og vælg at gemme den på disk (fx Skrivebord). Filen er på 27 MB. I filnavnet er "2.3.0" versionsnummeret for R; denne del af navnet vil altså ændre sig når der kommer nye versioner af R.
- Dobbeltklik på `R-2.3.0-win32.exe` når download er færdig, vælg Dansk installationsdialog, acceptér licensen, vælg Brugerinstallation, og acceptér alle de foreslåede indstillinger.

  Når installationen er slut kan du slette `R-2.3.0-win32.exe` fra disken.

A.2 Installation af ekstra R-pakker

Der findes et stort antal ekstra pakker til R, især til specielle statistiske analyser og til import og eksport af data. For at installere fx pakken `RODBC`, som kan bruges til at indlæse forsøgsdata fra regnearksfiler i Excel-format, skal din pc være tilsluttet internettet. I Windows-udgaven af R skal du så gøre sådan her:

- I RGui vælges PACKAGES || INSTALL PACKAGE(S) || CRAN MIRROR || DENMARK
- Efter et øjeblik dukker der en liste af pakker op. Find `RODBC` på listen, marker den, og tryk OK.
- Hvis alt går godt kommer der en meddelelse om "successfully unpacked" i R Console.

B Sådan installerer du OpenOffice

Hjemmesiden for OpenOffice er `http://www.openoffice.org/` hvorfra man kan hente den nyeste udgave af OpenOffice samt vejledninger osv. Den danske udgave hentes nemmest fra `http://da.openoffice.org/`

Gør sådan her for at installere dansk OpenOffice under Microsoft Windows:

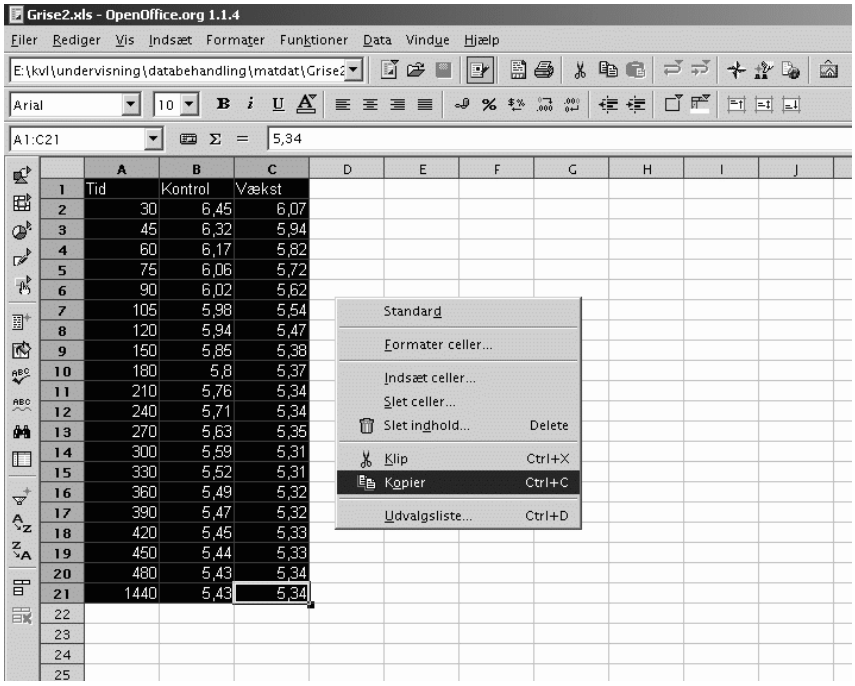
- Gå til `http://da.openoffice.org/`
- Klik på Windows under Hent Dansk OpenOffice.org og vælg at gemme installationsfilen på disk, fx på Skrivebord.
- Dobbeltklik på den nye skrivebordsikon, der vil hedde noget i retning af `OOo_2.0.2_Win32Intel_install_da.exe`, når download er færdig.
- Acceptér licensaftalen, indtast kundeinformation (om dig selv), vælg `Komplet installation`.

  Når installationen er slut kan du slette `OOo_2.0.2_Win32Intel_install_da.exe` fra disken.

C Flere former for indlæsning til R

C.1 Data fra regneark til R med klippe-klistre

Den nemmeste måde at flytte data fra et regneark til R er at markere dataene i regnearket og derefter enten trykke `Ctrl+C` eller højreklikke og vælge KOPIER som vist i figur 43. I begge tilfælde anbringes en kopi af dataene på det såkaldte klippebord (clipboard) i Windows.



Figur 43: Markering og kopiering af data i regneark.

Skift derefter til R Console, hvor man kan læse fra Windows-klippebordet `clipboard`:

```
> d <- read.table("clipboard", header=TRUE, dec=",")
```

Når man overfører data fra et engelsksproget regneark skal argumentet `dec=","` udelades:

```
> d <- read.table("clipboard", header=TRUE)
```

Bemærk at når man redigerer i R Console eller i et andet program, så forsvinder den "klippede" tekst fra klippebordet, og `read.table("clipboard", ...)` vil ikke virke før man igen har valgt data der skal kopieres.

C.2 Data fra regneark til R via tekstfil

I stedet for at læse direkte fra regneark til R med klippe-klistre kan man først eksportere data fra regneark til en tekstfil i kommasepareret format (`.csv`), og dernæst importere fra tekstfil til R med funktionen `read.csv2` eller `read.csv`. Fordelen ved dette er at man kan gemme tekstfilen og senere tjekke (med NotePad eller WordPad) at det var de rigtige tal man regnede videre på — betryggende hvis man bruger resultaterne i sit bachelor- eller specialeprojekt eller en anden videnskabelig publikation.

Fra dansk OpenOffice eller Excel eksporterer man til `.csv`-fil ved at vælge FILER || GEM SOM || FILTYPE TEKST CSV. I dansk OpenOffice skal man derefter vælge FELTSEPARATOR (;), altså semikolon. Den resulterende kommaseparerede fil kan derefter importeres med `read.csv2` som vist i afsnit 10.1.2.

Fra amerikansk OpenOffice eksporterer man til `.csv`-fil ved at vælge FILE || SAVE AS || FILE TYPE TEXT CSV, og her er det bedst at lade FIELD DELIMITER forblive komma (`,`). Den resulterende kommaseparerede fil kan derefter importeres med `read.csv` som vist i afsnit 10.1.2.

C.3 Direkte indlæsning af data fra regneark til R

Med R-systemets databaseinterface `RODBC` kan man indlæse data direkte fra regneark i Excel-format (`.xls`). Regneark lavet med OpenOffice Regneark skal altså først gemmes i Excel-format med FILER || GEM SOM || FILTYPE MICROSOFT EXCEL 97/2000/XP.

For at læse data direkte fra et Excel-regneark til R skal R-pakken `RODBC` være installeret, som forklaret i appendiks A.2. Desuden er det formentlig nødvendigt at have Microsoft Office installeret. Når det er på plads kan man læse `Ark1` fra regnearksmappen `Grise2.xls` på denne måde:

```
> library(RODBC)
> channel <- odbcConnectExcel("Grise2.xls")
> d <- sqlFetch(channel, "Ark1")
> odbcClose(channel)
```

Herved indlæses dataene til datasættet `d` som i afsnit 10.1. I ordren `sqlFetch` betyder argumentet `"Ark1"` at der skal læses data fra arket med navn `Ark1` i regnearksmappen `"Grise2.xls"` — en regnearksmappe indeholder jo gerne mere end ét regneark. I dansk OpenOffice og Excel hedder arkene normalt `Ark1`, `Ark2`, osv; mens de i amerikansk OpenOffice og Excel normalt hedder `Sheet1`, `Sheet2`, osv.

C.4 Indlejring af forsøgsdata i R-programmer

Ved hjælp af funktionen `textConnection` kan man indlægge datasæt direkte i et R-program. Dette er især nyttigt hvis man vil sende et komplet R-eksempel til andre eller vil gemme præcis de R-ordrer der blev brugt til at lave figurer til en artikel. Desuden er det praktisk for kendere af statistiksystemet SAS, hvis CARDS-kommando gerne bruges til at indlægge data i programmer.

For eksempel kan man indlægge grise-datasættet fra afsnit 10.1.1 i et R-program, "indlæse" det til en data frame `d` med `read.table`, og lave en plot-fil `grise.emf` sådan her:

```
d <- read.table(header=TRUE, dec=",", textConnection(
"    Tid Kontrol  Vaekst
      30    6,45    6,07
      45    6,32    5,94
     ...     ...     ...
     480    5,43    5,34
    1440    5,43    5,34"))
win.metafile("grise.emf")
plot(d)
dev.off()
```

Helt tilsvarende kan data opstillet i (dansk) kommasepareret format "indlæses" med `read.csv2` fra en `textConnection`:

```
d <- read.csv2(textConnection(
"Tid;Kontrol;Vaekst
```

```
30;6,45;6,07
45;6,32;5,94
...
480;5,43;5,34
1440;5,43;5,34"))
win.metafile("grise.emf")
plot(d)
dev.off()
```

Hvis et af ovenstående R-programmer er gemt i en fil `gris.R`, så kan det indlæses og udføres i R-systemet med funktionen `source("gris.R")` eller ved at vælge FILE || SOURCE R CODE || FILNAVN GRIS.R.

D Polynomiel regression og regressionskurve i R

I afsnit 13 så vi hvordan man i R nemt foretager lineær regression med funktionen `lm`.

Det er næsten lige så nemt at beregne en polynomiel regressionskurve, for eksempel at finde det andengradspolynomium der bedst passer til grisedatasættet `d` (se afsnit 13). Man bruger igen funktionen `lm` til at finde de koefficienter c, b og a der bedst passer med $y = c + bx + ax^2$ hvor y er `Kontrol` og x er `Tid`.

```
> preg <- lm(Kontrol ~ Tid + I(Tid^2), data=d)
> preg
Coefficients:
(Intercept)          Tid     I(Tid^2)
  6.318e+00   -2.801e-03    1.523e-06
```

Det beregnede andengradspolynomium `preg` er $y = 6.318 - 0.002801x + 0.000001523x^2$. I kaldet til `lm` betyder funktionen `I` i R-formlen at `Tid^2` er en variabel der indgår i den lineære model, ikke et udtryk der skal udregnes.

For at indtegne polynomiet på plottet over dataene skal man opbygge en vektor `m` af x-værdier, såkaldte "støttepunkter". For hvert af disse beregnes polynomiets y-koordinat ved hjælp af funktionen `predict`. Tilsammen udgør `m` og værdierne beregnet af `predict` et sæt (x, y)-koordinater for polynomiet, og de bruges i funktionen `lines` til at tilføje polynomiet til dataplottet:

```
> m <- seq(0, 1500, len=101)
> lines(m, predict(preg, data.frame(Tid=m)), col="red")
```

Polynomiel regression med hensyn til polynomier af højere grad, eller med hensyn til andre transformerede variable, laves på samme måde:

```
> preg3 <- lm(Kontrol ~ Tid + I(Tid^2) + I(Tid^3), data=d)
> lines(m, predict(preg3, data.frame(Tid=m)), col="green")

> regLog <- lm(Kontrol ~ I(log(Tid)), data=d)
> lines(m, predict(regLog, data.frame(Tid=m)), col="yellow")
```

E Plot af punkter og kurver i rummet med `scatterplot3d` i R

Der er ingen funktion til at lave plot af punkter i rummet (3D scatterplot) i standard R, man er nødt til at installere den ekstra pakke `scatterplot3d` (se afsnit A.2 for en vejledning).

Under forudsætning af, at pakken `scatterplot3d` er installeret, gør man nu følgende for at få adgang til funktionen `scatterplot3d`:

```
> library("scatterplot3d")
```

Lad os først producere nogle tilfældige data, der skal plottes. Først defineres `x` og `y` til at være to vektorer med hver 100 ligefordelte tilfældige tal mellem 0 og 10:

```
> x <- runif(100,0,10)
> y <- runif(100,0,10)
```

Herefter konstruerer vi `z` således at vi til hvert talpar `(x[i],y[i])` får `z[i]` på planen $z = x + 2y + 3$ plus et normalfordelt tilfældigt tal (støj):

```
> z <- x + 2*y + 3 + rnorm(100)
```

Herefter kan vi tegne et 3D scatterplot:

```
> scatterplot3d(x, y, z)
```

Det er muligt at variere 3D projektionen med parameteren `angle`. Vi kan også bruge `scatterplot3d` til at tegne kurver i rummet:

```
> t <- seq(0, 8*pi, len=300)
> x <- cos(t)
> y <- sin(t)
> z <- t
> scatterplot3d(x, y, z, type="l")
```

I ovenstående har vi 300 støttepunkter for værdier af parameteren `t` mellem 0 og 8π. Argumentet `type="l"` (bogstavet ℓ) angiver at `scatterplot3d` skal forbinde punkterne med linier.

F Funktionen `overflade`

Som gennemgået i afsnit 18.1 er det i R en lille smule omstændeligt at få tegnet et 3D overfladeplot af en funktion af to variable. Specielt kan det være besværligt at finde den betragtningsvinkel, der giver det bedste overblik over overfladen. For at gøre det nemmere er der til kurset *Matematik og Databehandling* fremstillet en funktion `overflade` som er tænkt til erstatning for funktionen `persp` når det drejer sig om at plotte en funktion af to variable med fokus på at forstå noget om funktionen frem for med fokus på at kunne lave plot i R.

Funktionen `overflade` kan hentes fra kursushjemmesiden og indlæses i R som beskrevet i afsnit F.1. Afsnit F.2 er en brugervejledning der beskriver hvordan man laver plot med funktionen. Afsnit F.3 er en beskrivelse af hvordan funktionen er implementeret og fungerer og er absolut kun tænkt som kursorisk læsning.

F.1 Hent og indlæs `overflade`

Da `overflade` ikke er en indbygget funktion i R skal den først indlæses fra en R-fil. Denne fil hentes fra kursushjemmesiden som følger:

- Gå ind med din browser på kursushjemmesiden `http://www.matfys.kvl.dk/mat-dat` og klik på "Eksempelfiler til opgaver og forelæsninger".
- Højreklik på R-filen `R/overflade.R` (fra mappen R) og vælg "gem som" og gem filen på din computer i mappen `MatDat`, som du har oprettet til *Matematik og Databehandling*. *Sørg for at filen kommer til at hedde `overflade.R` og ikke `overflade.R.txt`!*
- Hvis du bruger Internet Explorer kan det være nødvendigt at omdøbe filen efter du har hentet den ned for at give den det rigtige navn (`overflade.R` frem for `overflade.R.txt`).

Ovenstående behøver du kun at gøre én gang (så længe du ikke kommer til at slette `overflade.R` igen).

Når du skal til at bruge `overflade` skal filen `overflade.R` indlæses i R. Når filen er indlæst er `overflade` defineret ind til du forlader R (lukker programmet). Når du næste gang starter R vil `overflade` *ikke* være defineret, så hvis du skal bruge den må du indlæse den igen.

Indlæsning foregår som følger:

- Med R Console som aktivt vindue (som øverste vindue i RGui) vælger du fra menuen "File" punktet "Source R code...".
- Klik dig frem til filen `overflade.R`.
- Klik "Open".

Alternativt kan du skrive `source("overflade.R")`, hvis ellers MatDat mappen er den aktuelle filmappe (som den vil være hvis du ellers har gjort som beskrevet i afsnit 1.1 og altid starter R med R-ikonen matdat i MatDat-mappen).

Nu er du klar til at bruge `overflade`.

F.2 Brug af `overflade`

Funktionen `overflade` tegner et overfladeplot af en funktion af to variable. I eksemplerne nedenfor antager vi at vi har en passende funktion `f` defineret, for eksempel:

```
> f <- function(x,y) { cos(x)*sin(2*y) }
```

For at lave et overfladeplot for `f` for $x \in [0, 2\pi]$, $y \in [1, 5]$ kan man skrive:

```
> overflade(f, 0, 2*pi, 1, 5)
```

En alternativ måde at skrive intervallerne på er således:

```
> overflade(f, x.interval=c(0,2*pi), y.interval=c(1,5))
```

Der vil blive lavet et overfladeplot og der vil også komme en lille vejledning i R Console:

```
Drej grafen ved venstreklik i siderne af grafvinduet.
Afslut med højreklik i grafviduet.
```

Man kan nu dreje grafen (ændre betragtningsvinkel) ved at venstreklikke nær kanterne af R Graphics vinduet (inden i vinduet). Klik i venstre eller højre side drejer grafen om den lodrette akse. Tilsvarende vil klik i toppen eller bunden dreje grafen om den vandrette førsteakse. Klikker man nær kanten drejes grafen 5 grader pr. klik; klikker man længere inde mod midten drejes kun 1 grad pr. klik. R Console vinduet vil være blokeret ind til man går ud af `overflade` ved at højreklikke i R Graphics vinduet og vælge “Stop”.

Det er muligt at få betragtningsvinklen (`phi` og `theta`) vist i R Console mens man drejer grafen ved at angive parameteren `showangles=TRUE`:

```
> overflade(f, 0, 2*pi, 1, 5, showangles=TRUE)
Drej grafen ved venstreklik i siderne af grafvinduet.
Afslut med højreklik i grafviduet.
phi: 15  theta: 0
phi: 20  theta: 0
phi: 20  theta: -5
```

(Ovenstående er hvordan det ser ud efter først et klik øverst i vinduet og dernæst et klik til venstre i vinduet.)

Man kan angive startværdier af `phi` og `theta` (se afsnit 18.1 om betydningen af disse vinkler):

```
> overflade(f, 0, 2*pi, 1, 5, phi=40, theta=30)
```

Generelt kan man bruge alle de samme parametre som man kan bruge til `persp` for at ændre udseende af plottet, herunder `col` (farve af felterne), `border` (farve af stregerne), `shade` (belysning), `scale` (skalering af akser eller ej) og `r` (perspektivisk afstand), se afsnit 18.1. Specielt værd at nævne er at man med `ticktype="detailed"` får tegnet enheder på akserne.

Hvis man ikke ønsker det interaktive museklikkeri kan man angive `interactive=FALSE`, så tegner `overflade` bare grafen:

```
> overflade(f, 0, 2*pi, 1, 5, phi=40, theta=30, interactive=FALSE)
```

Som standard tegner `overflade` et net med 50×50 støttepunkter. Hvis man ønsker et grovere eller finere net kan man angive antal støttepunkter med parameteren `n`:

```
> overflade(f, 0, 2*pi, 1, 5, n=100)
```

Husk dog at jo flere støttepunkter, jo længere tid tager det at tegne grafen hver gang man klikker.

F.3 Implementationen af `overflade`

Dette afsnit forklarer i detaljer hvordan funktionen `overflade` er implementeret. Det er absolut *ikke* nødvendigt at læse dette afsnit for at kunne bruge funktionen. Afsnittet er heller *ikke* en del af pensum i *Matematik og Databehandling*. Forklaringen her er udelukkende givet for at tjene som et eksempel på konstruktion af en lidt mere kompleks R-funktion end dem der er vist i resten af noterne.

R-koden for funktionen `overflade` er vist i figur 44, med linienumre tilføjet i venstre side for at lette beskrivelsen.

Funktionens hoved er i linierne 1–6 identisk med udgaven beskrevet i slutningen af afsnit 19.5, se dette for en nærmere forklaring af disse parametre. I linie 7–8 er introduceret nogle ekstra parametre i forhold til afsnit 19.5, nemlig `phi` og `theta` som styrer betragtningsvinklen (svarende til `phi` og `theta` i `persp`) og så de tre parametre `interactive`, `showhelp` og `showangles` der styrer den interaktive brug af `overflade` (se foregående afsnit).

Linie 11–14 skriver en hjælpetekst i R Console hvis både `interactive` og `showhelp` er `TRUE`. R-funktionen `cat` skriver sine argumenter (her tekster) i R Console. I enden af hver af de to tekster forekommer tegnsekvensen "`\n`", der betyder linieskift. Dette er nødvendigt fordi `cat` ikke automatisk indsætter nogle linieskift, så hvis ikke de to linieskift var angivet eksplicit ville `cat` skrive det hele ud i én køre. Funktionen `flush.console` er nødvendig for at sikre at teksten vises i R Console med det samme (under Windows og Mac OS X).

I linie 16 beregnes værdierne i støttepunkterne, som beskrevet i afsnit 19.5. Kaldet af `persp` i linie 18 tegner overfladen, eneste forskel til afsnit 19.5 er at parametrene `phi` og `theta` er eksplicit angivet.

Linie 20 skriver værdierne af `phi` og `theta` i R console, hvis `showangles` er `TRUE`. Her er givet mere end ét argument til `cat`; de skrives i rækkefølge i R console.

Den interaktive del af `overflade` udgøres af linierne 23–47. Hvis parameteren `interactive` er `FALSE` springes denne del simpelthen over (linie 22).

Løkken i linie 24–47 udføres ind til brugeren klikker med højre musetast i grafvinduet. Brugerens museklik fås med `locator(1)` og gemmes i variablen `pos` (i linie 23 før løkken og i linie 46 i slutningen af hvert gennemløb af løkken). Så længe brugeren venstreklikker returnerer `locator(1)` en associationsliste med to komponenter, `$x` og `$y`, og dermed er løkkebetingelsen `length(pos)==2` i linie 24 opfyldt og løkkekroppen udføres. I det øjeblik brugeren højreklikker giver `locator(1)` værdien `NULL`, `pos` bliver derfor `NULL` og `length(pos)` dermed nul og løkkebetingelsen falsk, hvorefter R fortsætter med linie 48 efter løkken.

Inde i løkken skal koordinaterne, brugeren klikkede på, omsættes til en ændring af enten `phi` eller `theta`. Funktionen `locator` giver 2D-koordinater, der intet har at gøre med 3D koordinatsystemet som `persp` har tegnet. For at finde ud af hvor brugeren klikkede i forhold til grafvinduets kanter er vi nødt til at finde ud af hvordan `locator`-koordinaterne forholder sig til grafvinduets udstrækning. Vi vil gerne *normalisere* koordinaterne således at x går fra $x = 0$ i venstre side til $x = 1$ i højre side og y går fra $y = 0$ i bunden til $y = 1$ i toppen. Vektoren `par("usr")`, som i linie 26 gemmnes i variablen `extent`, giver nogenlunde[9] 2D koodinaterne der svarer til nederste venstre hjørne (*llx*,*lly*) og øverste højre hjørne (*urx*,*ury*) i grafvinduet som en vektor (*llx*,*urx*,*lly*,*ury*). En normaliseret værdi af x fås da som $\frac{x-llx}{urx-llx}$ hvilket er hvad der beregnes i linie 27 og gemmes i variablen `xrel`. Tilsvarende beregnes den relative y-koordinat $\frac{y-lly}{ury-lly}$ i linie 28 og gemmes i `yrel`.

Afhængig af om klikket er tættest på højre henholdsvis venstre side skal `theta` ændres ved at tillæge et positivt henholdsvis negativt tal. Ydermere vil vi gerne have det sådan at klik tæt på kanten giver en ændring på 5 grader mens klik længere inde mod midten kun ændrer vinklen 1 grad. Variablen `xdir` er det tal, vi vil lægge til `theta`. Variablen `xdist` angiver afstanden til den nærmeste kant. I linierne 30–

[9] Kun "nogenlunde" fordi der er afsat marginer langs grafvinduets kanter til titler og den slags, og `par("usr")` giver koordinaterne inden for disse marginer.

```
overflade <- function(f,
                       x.min=min(x.interval), x.max=max(x.interval),
                       y.min=min(y.interval), y.max=max(y.interval), n=50,
                       x.n=n, y.n=n, x.interval=c(0,1), y.interval=c(0,1),
                       x=seq(x.min, x.max, length=x.n),
                       y=seq(y.min, y.max, length=y.n),
                       theta=0, phi=15,
                       interactive=TRUE, showhelp=TRUE, showangles=FALSE,
                       ... ) {
  # Vis eventuelt hjælpetekst i R Console:
  if(interactive && showhelp) {
    cat("Drej grafen ved venstreklik nær kanterne af grafvinduet.\n");
    cat("Afslut med højreklik i grafviduet.\n"); flush.console();
  }
  # Udregn støttepunkter:
  z <- outer(x, y, f);
  # Tegn overfladen (første gang):
  persp(x, y, z, theta=theta, phi=phi, ...);
  # Vis evt. betragtningsvinkel (første gang):
  if (showangles) { cat("phi:",phi," theta:",theta,"\n"); flush.console(); }
  # Hvis interaktiv brug så gå i løkke styret af museklik:
  if (interactive) {
    pos <- locator(1); # Vent på første klik...
    while (length(pos)==2) {
      # Udregn relativ position af klik i grafvinduet:
      extent <- par("usr");
      xrel=(pos$x-extent[1])/(extent[2]-extent[1]);
      yrel=(pos$y-extent[3])/(extent[4]-extent[3]);
      # Fortegn for ændring bestemmes af klikposition i forhold til midten:
      if(xrel>0.5) { xdir=1; xdist=1-xrel; }
      else { xdir=-1; xdist=xrel; }
      # Større vinkelændring hvis klikket er tæt på kanten:
      if(xdist<0.1) xdir<-xdir*5;
      # Alt det samme for y:
      if(yrel>0.5) { ydir=1; ydist=1-yrel; }
      else { ydir=-1; ydist=yrel; }
      if(ydist<0.1) ydir<-ydir*5;
      # Afhængigt af hvilken kant klikket var tættest på
      # ændres enten phi eller theta:
      if (xdist<ydist) { theta <- theta+xdir; }
      else { phi <- phi+ydir; }
      # Tegn overfladen med ændrede vinkler:
      persp(x, y, z, theta=theta, phi=phi, ...);
      # Vis evt. den nye betragtningsvinkel:
      if(showangles) { cat("phi:",phi," theta:",theta,"\n"); flush.console(); }
      pos <- locator(1); # Vent på næste klik...
    }
  }
  invisible(list(theta=theta, phi=phi));
}
```

Figur 44: R-koden for funktionen `overflade`.

31 gives `xdir` og `xdist` værdier afhængigt af om klikket var tættest på højre kant (`xrel>0.5`) eller ej. Tildelingerne i `if`-grenen linie 30 udføres hvis klikket er tættest på højre kant (hvor $x = 1$ og afstanden til kanten dermed er `1-xrel`) mens tildelingerne i `else`-grenen linie 31 udføres hvis klikket er tættest på venstre kant (hvor $x = 0$ og afstanden til kanten simpelthen er `xrel`). I linie 33 skaleres `xdir` med en faktor 5 hvis klikket er "tæt" på kanten – inden for 10% af grafområdets udstrækning[10] (`xdist<0.1`). I linierne 35–37 udføres analoge beregninger med hensyn til den klikkede y-koordinat og resultaterne gemmes i variablene `ydir` og `ydist`.

I linie 40–41 ændres én af vinklerne `theta` og `phi` afhængigt af om klikket var tættest på en lodret kant (`xdist<ydist`) eller en vandret kant. Den anden vinkel lades uforandret.

I linie 43 tegnes overfladegrafen på ny, med de nye værdier af `phi` og `theta`. I linie 45 vises de nye værdier af vinklerne i R console hvis `showangles` er `TRUE`. Endelig som det sidste i løkkekroppen afventes et nyt museklik i linie 46.

Værdien af kaldet af `overflade` udregnes i linie 49. Funktionen returnerer en associationsliste med de senest brugte værdier af `phi` og `theta`. Da man som bruger oftest nok ikke er interesseret i at se disse værdier gøres listen "usynlig" med R-funktionen `invisible`.

[10] På grund af de afsatte marginer der ikke regnes med til grafområdet svarer de 10% i virkeligheden cirka til de yderste 20% af grafvinduet.

G Litteratur om R

- *An Introduction to R* er en introduktion til R-sproget, statistisk analyse og grafik.
  Fås fra `http://cran.r-project.org/doc/manuals/R-intro.pdf`
- *The R Language Definition* beskriver R-sproget i detaljer og er nyttig hvis man skal skrive sine egne R-funktioner. Fås fra `http://cran.r-project.org/doc/manuals/R-lang.pdf`
- *R Data Import/Export* beskriver hvordan man får data ind i og ud af R.
  Fås fra `http://cran.r-project.org/doc/manuals/R-data.pdf`
- *The R Reference Index* indeholder alle hjælpefiler fra R's standardpakker og anbefalede pakker.
  Fås fra `http://cran.r-project.org/doc/manuals/fullrefman.pdf`
- Peter Dalgaard: *Introductory Statistics with R*. Springer-Verlag 2002. ISBN 0-387-95475-9.
- Paul Murrell: *R Graphics*. Chapman & Hall/CRC Computer Science 2005. ISBN 158488486X.
- Flere bøger kan findes ved at søge efter `R statistics` på `http://www.amazon.com/`

Indeks