R-ohjelmointi

Kurssimoniste

Mari Myllymäki

Santtu Tikka

31. elokuuta 2017

R:n kotisivu: `www.r-project.org`

Kirjallisuutta

W.N. Venables, D.M. Smith and the R Development Core Team (2004). *An Introduction to R.* Revised and updated.
Ladattavissa netistä osoitteesta `http://cran.r-project.org/doc/manuals/R-intro.pdf`

Peter Dalgaard (2002). *Introductory Statistics with R.* Springer. Second edition, 2008.

- - -

A. Penttinen (2001). S-plus/R-ohjelmointi. Kurssimoniste.

P. Koikkalainen (2007). R-ohjelmointi. Kurssimoniste. `http://erin.mit.jyu.fi/pako/CourseR/`

Mari Myllymäki (2008). R-ohjelmointi. Kurssimoniste, Jyväskylän yliopisto.

M.-L. Hannila, V. Kiviniemi (2008). R-opas. Alkeista tilastollisiin perusmenetelmiin. Kuopion yliopisto, tietotekniikkakeskus.

- - -

J.C. Pinheiro and D.M. Bates (2002). Mixed-Effects Models in S and S-PLUS. Springer-Verlag New York, Inc.

Sisältö

1 Johdanto

R on interaktiivinen tietokoneohjelma, joka on tarkoitettu tilastolliseen laskemiseen ja grafiikkaan. Sitä voidaan käyttää useissa Unix-käyttöjärjestelmissä sekä Windows- ja MacOS-käyttöjärjestelmissä.

R on osa GNU-projektia (`www.gnu.org`) ja se perustuu paljolti S kieleen ja ympäristöön, jotka Rick Becker, John Chambers ja Allan Wilks kehittivät BELL laboratoriossa 1980-luvulla. R:n tekemisen aloittivat Robert Gentleman ja Ross Ihaka noin 10 vuotta S ohjelman syntymisen jälkeen. Alkuperäinen S on kaupallinen tuote, kun taas R on vapaasti käytettävissä ja jaettavissa. Kenellä tahansa on oikeus käyttää, kopioida, jakaa ja kehittää R-ohjelmaa.

R:n voi ladata itselleen R:n kotisivuilta `www.r-project.org`.

1.1 Miksi kannattaa käyttää R:ää?

• Hyvin kehittynyt, objektiorientoitunut rakenteellinen ohjelmointikieli.

• Perusfunktioiden lisäksi on saatavilla yli tuhat kirjastoa, jotka sisältävät valmiita funktioita erilaisiin tehtäviin (`http://cran.r-project.org`).

• Helposti laajennettavissa (voi tehdä omia funktioita ja kirjastoja).

• R "osaa" lukuisia toimintoja:

  · Toimii laskukoneena.

  · Käsittelee vektori- ja matriisialgebraa.

  · Osaa 2- ja 3-ulotteisen grafiikan.

  · Mahdollistaa monipuoliset tietorakenteet.

  · Mahdollistaa interaktiivisen työskentelyn.

  · Tilastollista analyysiä. R:stä löytyy valmiita kirjastoja/funktioita mm. (yleistettyjen) lineaaristen mallien ja sekamallien sovitukseen, aikasarja-analyysiin, epidemiologian menetelmiin, pääkomponenttianalyysiin, pisteprosessien simulointiin, geostatistiikkaan.

  · Ehtolauseet, silmukat ja käyttäjän määrittelemät rekursiiviset funktiot.

  · Käyttäjä voi ohjelmoida uusia funktioita C/C++ ja FORTRAN -kielillä.

• Helppokäyttöinen, suhteellisen helppo oppia.

• Syntaksi on kansainvälinen standardi.

• R on vapaasti saatavilla ja käy erilaisiin käyttöympäristöihin.

1.2 R:n käynnistys ja lopetus

R:n käyttöliittymä on tekstimuotoinen komentotulkki, jonka avulla käyttäjä käskyttää ohjelmistoa. Komennot kirjoitetaan tietokoneen näppäimistön avulla; menuja ja graafista käyttöliittymää ei juuri käytetä (voi käyttää joissakin toiminnoissa, esim. tallentamisessa, kirjastojen lataamisessa). Tässä luvussa käsitellään R:n käynnistystä ja lopetusta Windows-käyttöjärjestelmässä.

Windows-käyttöjärjestelmässä ohjelma käynnistyy esim. klikkaamalla ohjelman ikonia joko Käynnistä-valikosta tai työpöydältä. Tällöin R käynnistyy ja ohjelmaan aukeaa konsoli-ikkuna (R Console):

```
R version 3.0.3 (2014-03-06) -- "Warm Puppy"
Copyright (C) 2014 The R Foundation for Statistical Computing

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

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

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

>
```

'>' -merkki on R:n kehoite. Käynnistyessään R siis siirtyy odottamaan käyttäjän antamia kommentteja. Komentojen suoritus tapahtuu, kun käyttäjä on kirjoittanut komennon ja painanut `ENTER`-näppäintä. Komennon suorituksen voi keskeyttää käyttöliittymän punaisella `Stop`-kuvakkeella tai painamalla `ESC`-näppäintä.

Yleensä, erityisesti jos R:llä on tekemässä työtä, jonka haluaa tallentaa myöhempää käyttöä varten, on hyödyllistä luoda itselleen työkansio. Kun käynnistää R:n, niin oletushakemisto kannattaa vaihtaa kyseiseksi työkansioksi. Tämän saa tehtyä valitsemalla `File`-valikosta `Change dir...` . Tällöin esimerkiksi R-istunnon aikana tallennettavat kuvat ja mahdolliset muut tiedostot tallentuvat automaattisesti työkansioon. Lisäksi ladattaessa R:ään tiedostoja, jotka sijaitsevat työkansiossa, ei ole välttämätöntä kirjoittaa koko hakemistopolkua annettavaan komentoon.

Kun ohjelman sulkee (`File` → `Exit` tai × oikeassa ylänurkassa tai komento `q()`), R kysyy "Save workspace image?". Jos kysymykseen vastaa "Yes", niin R:n työympäristö (objektit) tallentuu työkansioon (joka oletusarvoisesti on `C:/Program Files/R/R-2.7.1` tai vastaava). R luo tällöin kaksi tiedostoa, joista `.Rdata` sisältää R:ssä luodut objektit ja `.Rhistory` komentohistorian. Jos nämä tallentuvat oletuskansioon `C:/Program Files/R/R-2.7.1` (tai vastaava), niin tällöin R aina käynnistyessään (ikonista) automaattisesti lataa viimeksi tallennetun datan ja historian. Suositeltavampi tapa on käyttää työkansiota: Työkansioon tallennetun R-datan

(ja mahdollisen historian) saa käyttöönsä avaamalla kyseisen `.Rdata`-tiedoston esim. resurssienhallinnasta tai lataamalla sen R:n valikosta `File` → `Load Workspace...`.

Jos kysymykseen "Save workspace image?" vastaa "No", niin mitään ei tallenneta ja R sulkeutuu. Vastauksella "Cancel" palataan takaisin R:ään.

1.3 R:n koodieditori (script window) vs muu tekstieditori

Yleisesti ottaen 'oikea tapa' käyttää R:ää on tallentaa kirjoittamansa koodi ja funktiot .r- tai .txt-tiedostoihin. Tällöin suoritetut toimenpiteet ovat helposti toistettavissa ja muokattavissa ilman, että käyttäjän tarvitsee kirjoittaa komentoja uudelleen.

Windows-käyttöliittymässä R sisältää oman yksinkertaisen koodieditorin. Tämä `R editor` aukeaa `File`-valikosta valitsemalla `New script`. Editoriin kirjoitetun komentorivin saa ajettua komennolla `Ctrl + r`, kun kursori on kyseisellä rivillä. Vastaavasti jos useampi rivi on valittuna, niin kaikki nämä ajetaan komennolla `Ctrl + r`. Tämä komento itse asiassa kopioi kirjoitetun koodin `R Console`-ikkunaan ja näin R suorittaa komennot.

Vaihtoehtoisesti koodieditorina voi käyttää mitä tahansa tekstinkäsittelyohjelmaa (esim. Muistio, WordPad, Crimson Editor, Emacs). Tällöin komentonsa saa ajettua kopioimalla ne `R Console` -ikkunaan tai lataamalla koodin sisältämän tiedoston R:ään `source("tiedosto.r")` -komennolla.

Erityisesti R:ää varten suunniteltu kehitysympäristö on RStudio, joka on vapaasti ladattavasti netistä. RStudio -editoria voidaan käyttää kuten muitakin tekstinkäsittelyohjelmia, mutta se sisältää myös erityisiä R:ää varten kehitettyjä toimintoja.

1.4 R:n ohjesivustot

R-ohjelmointia, kuten mitä tahansa muutakin ohjelmointikieltä, oppii vain käyttämällä sitä. Erittäin tärkeää oppimisen ja käyttämisen kannalta on osata etsiä tietoa. Useimpiin tilastollisiin menetelmiin on jo olemassa valmiita R-funktioita ja monet uusimmat tilastotieteen algoritmit ja menetelmät tulevat nopeasti saataville R-kirjastoissa. Joskus halutun R-funktion löytäminen saattaa tosin vaatia hieman vaivannäköä. Paras keino etsiä tietoa jostain aiheesta (kun ei tiedä tarkoitukseen sopivaa R-kirjastoa taikka funktiota) lienee R:n kotisivujen kautta (R site search). Lisäksi R-ohjelma sisältää ohjesivustot, mutta näiden sisältö rajoittuu koneeseen asennettuihin kirjastoihin/funktioihin.

Kaikista R:n valmiista funktioista (jotka ovat koneelle asennetuissa kirjastoissa) löytyy tietoa R:n `HELP`-tiedostoista. Komento

```
> help.start()
```

avaa nettisivun (offline), josta löytyy ohjeita:

- Valitsemalla 'Packages' pääsee selaamaan asennettua kirjastoja ja näiden sisältämiä funktioita.

- "An Introduction to R" on hyvä johdanto R-kieleen. Sisältö on sama kuin kirjassa, jonka ovat toimittaneet W.N. Venables, D.M. Smith and the R Development Core Team (2004). Manuaalista "R Data Import/Export" löytyy ohjeita tutkimusaineistojen lukemisesta R:ään. Muut manuaalit ovat ohjeistusta R:n edistyneesempään käyttöön.

- Osioista "Frequently Asked Questions" ja "FAQ for Windows port" saattaa myös löytyä apua, erityisesti R:n käyttöä opetellessa, sillä näihin on koottu vastauksia yleisiin kysymyksiin.

Samat kokoelmat löytyvät myös R:n kotisivuilta (`www.r-project.org` → `Manuals`). Lisäksi osiosta `CRAN` → `Contributed` löytyy useita R:n käyttäjien kirjoittamia dokumentteja.

Ohjesivu yksittäisestä funktiosta avautuu seuraavalla komennolla (ehdolla että kyseinen funktio on olemassa ladatuissa kirjastoissa):

```
> help(mean)
> # tai
> ?mean
```

Huom. R:ssä '`#`' on kommentin merkki. Rivillä kaikki sen jälkeen tuleva sivuutetaan.

Tällä tavoin aukeava yksittäisen funktion ohjesivu on hyödyllinen aina kun käyttää yksittäistä funktiota (joka ei ole todella tuttu käyttäjälle). Tältä sivulta selviää funktion kutsu, sille annettavat argumentit, funktion palauttama tulos, ja lisäksi sivu sisältää useimmiten esimerkkejä funktion käytöstä.

Jos ei tiedä funktion nimeä, voi funktiota etsiä avainsanalla `help.search`-funktion avulla, esim.

```
> help.search("t-test")
```

antaa luettelon niistä funktioista, joiden nimessä tai esittelyssä esiintyy "`t-test`". Seuraavanlainen tulostus aukeaa edellisellä komennolla:

```
Vignettes:

Rcpp::Rcpp-unitTests Rcpp-unitTests PDF source R code
RcppArmadillo::RcppArmadillo-unitTests RcppArmadillo-unitTests PDF source R code
Code demonstrations:

tcltk::tkttest t-test example of GUI interface to a function call. (Run demo in console)
Help pages:

bnlearn::test.counter Manipulating the test counter
Hmisc::dataDensityString Internal Hmisc functions
Hmisc::summary.formula Summarize Data for Making Tables and Plots
Hmisc::t.test.cluster t-test for Clustered Data
Rcpp::RcppUnitTests Rcpp : unit tests results
vcd::goodfit Goodness-of-fit Tests for Discrete Data
stats::bartlett.test Bartlett Test of Homogeneity of Variances
```

```
stats::fisher.test Fisher's Exact Test for Count Data
stats::pairwise.t.test Pairwise t tests
stats::power.t.test Power calculations for one and two sample t tests
stats::t.test Student's t-Test
```

Tässä on listattuna funktioita, joihin hakusana t-test liittyy. Aluksi ilmoitetaan, mistä kirjastosta funktio löytyy, ja tämän jälkeen kahdella kaksoispisteellä eroteltuna funktion nimi. Lisäksi annetaan määrittely, joka kertoo mitä kyseinen funktio tekee. Esimerkiksi Studentin t-testin tekevästä funktiosta saa lisätietoa antamalla komennon `?t.test`.

1.5 R-kirjastot

Kaikki R-funktiot ja datat ovat tallennettuna R-kirjastoihin. R:n mukana tulevat tietyt peruskirjastot automaattisesti. Näillä pääsee hyvin alkuun. Komennolla

```
> library()
```

saa näkyviin R:ään asennettuna olevat kirjastot. Samaiset kirjastot löytyvät R:n help-sivuston (`help.start()`) alta kohdasta `packages`. Muita kirjastoja voi ladata R:n kotisivuilta. Jos tietokone on kytkettynä verkkoon, niin paketteja voi asentaa Window-käyttöliittymässä R:ään suoraan R:n kautta `Packages` → `Install package(s)...` .

Kirjaston sisältö on käytettävissä, kun kirjasto on ladattu R:ään. Osa kirjastoista ladataan automaattisesti R:ään, kun R käynnistetään. Muita kirjastoja saa ladattua R:ään komennolla

```
> library(kirjaston_nimi)
```

Esim. sekamallien kirjaston `nlme` saa käyttöön komennolla

```
> library(nlme)
```

kunhan kirjasto on ensin asennettu.

1.6 Esittelyesimerkkejä

1.6.1 Keskiarvo, varianssi, histogrammi, t-testi

Olkoon alla oleva data tiedostossa data1.dat (oletushakemistossa).

```
1.28 -1.84 1.47 6.1 10.36 6.68 1.49 -1.54 4.49 3.35 -0.1 2.33
-0.92 1.75 4.48 1.99 4.73 0.88 -0.78 -0.93 -2.46 -1.18 -0.45 4.71
-1.09 1.08 4.61 1.1 6.16 1.8 -2.94 -1.95 2.17 -2.89 3.4 -0.85
-0.74 1.63 -5.28 1.56 2.24 0.77 6.66 3.69 4.4 5.26 0.4 6.35 1.49
2.62 0.3 -3.3 4.19 5.44 1.61 0.18 2.64 4.46 0.77 6.53 -4.02 4.14
-0.89 2.84 3.12 4.09 3.14 6.99 -0.44 1.64 -2.84 5.28 0.5 -1.27
-3.81 3.91 4.47 6.7 5.17 1.44 7.18 1.33 6.05 -3.36 0.45 5.49 2.21
1.48 -2.18 3.54 -2.5 3.77 1.29 0.17 5.4 2.01 8.19 2.91 1.38 1.92
-0.53 2.64 4.13 2.31 3.28 2.75 0.11 6.15
```

Tämä data saadaan luettua R:ään `scan`-funktiolla:

```
> y <- scan("data1.dat")
Read 108 items
```

Tässä syntyi uusi objekti `y`, joka sisältää tiedostossa olevat arvot.

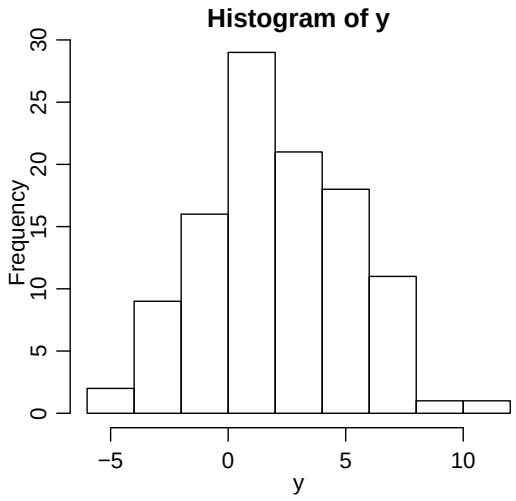
```
> objects()
[1] "y"
> y
  [1]  1.28 -1.84  1.47  6.10 10.36  6.68  1.49 -1.54  4.49  3.35 -0.10  2.33
 [13] -0.92  1.75  4.48  1.99  4.73  0.88 -0.78 -0.93 -2.46 -1.18 -0.45  4.71
 [25] -1.09  1.08  4.61  1.10  6.16  1.80 -2.94 -1.95  2.17 -2.89  3.40 -0.85
 [37] -0.74  1.63 -5.28  1.56  2.24  0.77  6.66  3.69  4.40  5.26  0.40  6.35
 [49]  1.49  2.62  0.30 -3.30  4.19  5.44  1.61  0.18  2.64  4.46  0.77  6.53
 [61] -4.02  4.14 -0.89  2.84  3.12  4.09  3.14  6.99 -0.44  1.64 -2.84  5.28
 [73]  0.50 -1.27 -3.81  3.91  4.47  6.70  5.17  1.44  7.18  1.33  6.05 -3.36
 [85]  0.45  5.49  2.21  1.48 -2.18  3.54 -2.50  3.77  1.29  0.17  5.40  2.01
 [97]  8.19  2.91  1.38  1.92 -0.53  2.64  4.13  2.31  3.28  2.75  0.11  6.15
```

Lasketaan keskiarvo ja varianssi ja piirretään histogrammi.

```
> mean(y)   # keskiarvo
[1] 2.019352
> var(y)    # varianssi
[1] 9.138286
> X11()     # avaa uuden grafiikkaikkunan
> hist(y)   # piirtää histogrammin y:n arvoista
>
> # Tulostetaan kuva tiedostoon:
> postscript(file="data1_histrogram.eps",height=4,width=4,paper="special",
+  horizontal=F)
> # grafiikan koon ja muodon määrittely
> par(mfrow=c(1,1),mar=c(4,2.5,1.5,0.5),mgp=c(1.5,0.7,0))
> hist(y)
> dev.off()
```

Testataan hypoteesia $\mu = 2.5$ kaksisuuntaisella t-testillä:

```
> tulos <- t.test(y, alternative="two.sided", mu=2.5, conf.level = 0.95)
> tulos

        One Sample t-test

data:  y
t = -1.6524, df = 107, p-value = 0.1014
alternative hypothesis: true mean is not equal to 2.5
```



Kuva 1: Histogrammi.

```
95 percent confidence interval:
 1.442707 2.595996
sample estimates:
mean of x
 2.019352
```

Huomaa, että edellä syntyi uusi objekti `tulos`.

```
> objects()
[1] "tulos" "y"
```

1.6.2 Oma funktio

Lasketaan edellisen tehtävän aineiston `y` 4 ensimmäistä momenttia. Kirjoitetaan funktio `moments`, joka laskee `K` ensimmäistä momenttia: $\frac{1}{n}\sum_{i=1}^{n} y_i^k$, $k = 1, 2, \ldots, K$.

```
moments <- function(x, K)
{
# laskee K ensimmäistä momenttia datavektorista x
    m <- NULL
    for(k in 1:K) {
        m[k] <- mean( x^k )
    }
    m
}
```

Tallennetaan funktio tiedostoon `moments.r`. Funktio voidaan lukea R:ään komennolla

```
> source("moments.r")
```

ja funktiota voidaan kutsua seuraavasti

```
> moments(y, K=4)
[1]   2.019352  13.131455  63.223633 456.974915
```

Tulos voidaan pyöristää:

```
> round(moments(y, K=4), digits=2)
[1]   2.02  13.13  63.22 456.97
```

Huomaa, että funktio `round` pyöristää aina lähimpään parilliseen lukuun tai desimaaliin:

```
> round(0.5)
[1] 0
> round(1.5)
[1] 2
```

1.6.3 Vektori- ja matriisilaskentaa

Vektoreiden ja matriisien käsittely R:ssä on yksinkertaista, mutta käsittelyssä tulee olla tarkkana, jotta tekee sitä, mitä haluaa. Esimerkkejä:

1. Lasketaan kahden kolmiulotteisen avaruuden pisteen $u = (2.2, -1.4, 6.7)$ ja $v = (1.9, 2.5, -2.2)$ välinen etäisyys $d = \sqrt{(2.2 - 1.9)^2 + (-1.4 - 2.5)^2 + (6.7 + 2.2)^2}$.

   ```
   > u <- c(2.2,-1.4,6.7)
   > v <- c(1.9,2.5,-2.2)
   > u
   [1]  2.2 -1.4  6.7
   > v
   [1]  1.9  2.5 -2.2
   > u-v
   [1]  0.3 -3.9  8.9
   > (u-v)^2
   [1]  0.09 15.21 79.21
   > sum((u-v)^2)
   [1] 94.51
   > sqrt( sum((u-v)^2) )
   [1] 9.721625
   ```

   Laskutoimituksen suorittamiseksi vektoreista `u` ja `v` tarvitaan vain viimeinen komento; välivaiheiden tarkoituksena on havainnollistaa, mitä komennon eri osat tekevät.

   Etäisyyden laskemiseen pisteiden välillä on olemassa myös valmis funktio `dist`, jolle pitää antaa parametrina matriisi tai datakehikko, jossa yksi rivi vastaa yhtä pistettä.

```
> rbind(u,v)
  [,1] [,2] [,3]
u  2.2 -1.4  6.7
v  1.9  2.5 -2.2
>  dist(rbind(u,v))
         u
v 9.721625
```

Funktio `dist` on hyödyllinen laskettaessa useamman kuin kahden pisteen välisiä etäisyyksiä.

2. Käännetään matriisi

$$A = \begin{bmatrix} 5.2 & 2.9 & 3.7 \\ 1.2 & 2.4 & 3.5 \\ 1.2 & 3.4 & 6.7 \end{bmatrix}.$$

```
> A <- matrix(c(5.2,2.9,3.7,1.2,2.4,3.5,1.2,3.4,6.7), ncol=3, byrow=T)
> A
     [,1] [,2] [,3]
[1,]  5.2  2.9  3.7
[2,]  1.2  2.4  3.5
[3,]  1.2  3.4  6.7
> solve(A)
            [,1]       [,2]        [,3]
[1,]  0.27792553 -0.4554521  0.08444149
[2,] -0.25531915  2.0212766 -0.91489362
[3,]  0.07978723 -0.9441489  0.59840426
```

3. Lasketaan matriisitulo

$$A'B = \begin{bmatrix} 5.2 & 2.9 & 3.7 \\ 1.2 & 2.4 & 3.5 \\ 1.2 & 3.4 & 6.7 \end{bmatrix}' \begin{bmatrix} 0.6 & 4.1 \\ 2.0 & 2.1 \\ 4.6 & 3.0 \end{bmatrix}.$$

```
> B <- matrix(c(0.6,2.0,4.6,4.1,2.2,3.0), ncol=2, byrow=F)
> B
     [,1] [,2]
[1,]  0.6  4.1
[2,]  2.0  2.2
[3,]  4.6  3.0
> t(A) # A:n transpoosi
     [,1] [,2] [,3]
[1,]  5.2  1.2  1.2
[2,]  2.9  2.4  3.4
[3,]  3.7  3.5  6.7
> # matriisitulo
> t(A)%*%B
      [,1]  [,2]
```

```
     [1,] 11.04 27.56
     [2,] 22.18 27.37
     [3,] 40.04 42.97
```

1.6.4 Aikasarja

Tarkastellaan dataa `Nile`, jossa on Ashwanissa vuosina 1871-1970 havaitut vuosittaiset virtaamat (annual flow). `Nile` on 'aikasarjaobjekti':

```
> Nile
Time Series: Start = 1871 End = 1970 Frequency = 1
  [1] 1120 1160  963 1210 1160 1160  813 1230 1370 1140  995  935 1110  994 1020
 [16]  960 1180  799  958 1140 1100 1210 1150 1250 1260 1220 1030 1100  774  840
 [31]  874  694  940  833  701  916  692 1020 1050  969  831  726  456  824  702
 [46] 1120 1100  832  764  821  768  845  864  862  698  845  744  796 1040  759
 [61]  781  865  845  944  984  897  822 1010  771  676  649  846  812  742  801
 [76] 1040  860  874  848  890  744  749  838 1050  918  986  797  923  975  815
 [91] 1020  906  901 1170  912  746  919  718  714  740
```

Piirretään aikasarjan kuvaaja:

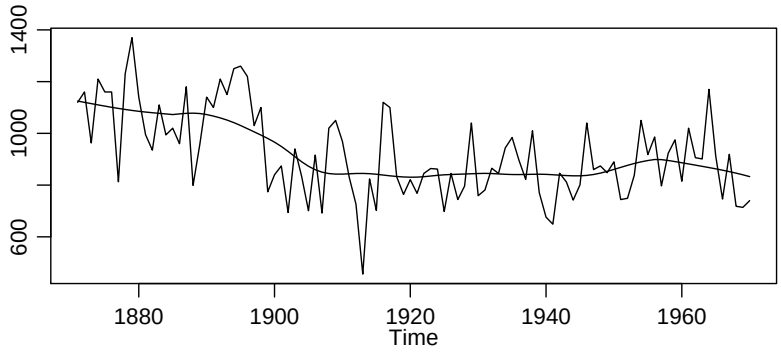
```
> par(pin=c(5,2))
> plot(Nile)
```

Estimoidaan aikasarjan trendi `loess`-funktiolla:

```
> a <- loess(Nile ~ time(Nile), span=30/length(Nile),
+      control=loess.control(surface="direct"), degree=1)
> trend <- ts(a$fit, start=time(Nile)[1])
```

ja piirretään trendi samaan kuvaan aikasarjan kanssa

```
> ts.plot(Nile, trend)
```



Kuva 2: Aikasarja ja siihen sovitettu trendikäyrä.

Lista funktioista, joita käytettiin

`help.start()`	R:n manuaalit
`?, help()`	'HELP'
`help.search()`	etsiminen help-tiedostoista hakusanalla
`q()`	R-session lopetus
`scan()`	(vektorimuotoisen) datan lukeminen tiedostosta
`objects()`	objektien listaaminen
`mean()`	keskiarvo
`var()`	varianssi
`hist()`	histogrammi
`par()`	grafiikan koon ja muodon määrittely
`t.test()`	Studentin t-testi
`function()`	funktion luominen
`source()`	funktion lataaminen
`round()`	lukujen pyöristys
`c()`	vektorin muodostaminen
`sum()`	summa parametrina annettavan vektorin alkioista
`sqrt()`	neliöjuuri
`rbind()`	vektorien tai matriisien sitominen toisiinsa riveittäin (=allekkain)
`dist()`	etäisyyksien laskeminen
`matrix()`	matriisin muodostaminen
`solve()`	lineaarinen yhtälöryhmän ratkaisu, matriisin kääntö
`plot()`	kuvaajien piirtäminen
`ts.plot()`	aikasarjojen piirtäminen
`loess()`	polynomipinnan sovittaminen
`ts()`	aikasarja-objektin luominen

2 R:n tietojärjestelmän piirteitä

+ rakenteellinen ohjelmointikieli

R-ohjelmointi on samankaltaista esim. C-ohjelmoinnin kanssa. R-ohjelman voi koostaa pienemmistä paloista, aliohjelmista. Aliohjelma on itsenäinen ohjelman osa, jota voidaan kutsua mistä tahansa pääohjelmasta tai muista aliohjelmista. Siis käsillä oleva ongelma voidaan ratkaista jakamalla se osatehtäviin. Tämä helpottaa ohjelmointia, tekee koodista selkeämpää ja auttaa luomaan funktiokirjastoja, joiden ohjelmia voidaan käyttää aliohjelmina toisissa ohjelmissa. Itse asiassa R:ssä on paljon valmiita funktiokirjastoja, jotka ovat vapaasti ladattavissa netistä ja kuka tahansa saa tehdä uuden tällaisen kirjaston kaikille jaettavaksi. Itse asiassa monet uusimmat tilastotieteen algoritmit ja menetelmät tulevat ensimmäisenä saataville juuri R-kirjastoissa.

+/- tietorakenne

R:ssä käyttäjä näkee 'datan' (=objektit) symbolisten nimien kautta. Nimien taakse voidaan 'kätkeä' hyvinkin monimutkaisia tietorakenteita. Näihin tietorakenteisiin päästään käsiksi nimien kautta. Nimet ovat itse asiassa osoitteita tietokoneen muistipaikkoihin, jotka R-ohjelmoinnissa ovat pääosin piilotettu käyttäjältä.

Tiedon eli datan tallentamiseen R tarjoaa useita esitysmuotoja ja tietorakenteita, kuten liukulukuja, taulukkoja, vektoreita, matriiseja, merkkijonoja, listoja. Lisäksi voidaan luoda myös paljon monimutkaisempia objekteja.

Järjestelmä luo muuttujia ja varaa näille tilaa automaattisesti ilman käyttäjän eksplisiittisesti antamaa komentoa, mikä toisaalta on käyttäjälle helppoa, mutta toisaalta myös kasvattaa ohjelmointivirheiden mahdollisuutta. R-ohjelmointi vaatiikin käyttäjältään huolellisuutta ja kurinalaisuutta. Esim. jos R:ssä kertoo kaksi eripituista vektoria keskenään, R toistaa lyhyemmän vektorin arvoja kunnes se on samanpituinen pidemmän vektorin kanssa. Tätä ominaisuutta voi käyttää hyödyksi, mutta kahden eripituisen vektorin kertominen, joka ei ole tarkoituksenmukainen, voi jäädä huomaamatta.

```
> # Luodaan kaksi vektoria x ja y
> x <- c(1,2,3)
> y <- c(1,1,1,2,2,2,3,3)
> # Kerrotaan vektorit keskenään
> z <- x*y
Warning message: longer object length
        is not a multiple of shorter object length in: x * y
> # R antaa varoituksen, mutta komento suoritetaan
> z
[1] 1 2 3 2 4 6 3 6
> # mutta jos y:n pituus on x:n pituuden monikerta
> y <- c(1,1,1,2,2,2,3,3,3)
> z <- x*y
> z
[1] 1 2 3 2 4 6 3 6 9
> # ei varoituksia
```

Huom. Nimet. Nimi on yksilöllinen tunniste objektille. Nimeksi kelpaavat kaikki kirjain-numero-yhdisteet ja nimiin saa sisältyä myös merkki ' . ', mutta pistettä ' . ' ei kannata käyttää nimen ensimmäisenä merkkinä (esim. `tulos`, `tulos2`, `data1.tulos` käyvät nimiksi). Isot ja pienet kirjaimet tulkitaan eri merkeiksi. R sallii myös luoda objektin (lähes mille tahansa) nimelle, joka on jo varattu ennalta johonkin muuhun käyttöön (esim. R:n valmiit funktiot, vakiot, käyttäjän omat objektit). Päällekirjoitus R:ssä on (vaarallisen) helppoa! R tekee tämän yleensä varoittamatta. Tosin (nimisuojatun) R:n funktion/vakion nimi palautuu tälle, kun käyttäjä poistaa luomansa samannimisen objektin (`rm(objekti)`-komennolla). Esim.

```
> pi
[1] 3.141593
> pi <- 2
> pi
[1] 2
> rm(pi)
> pi
[1] 3.141593
```

Käyttäjän omat objektit/funktiot sen sijaan tulevat ylikirjoitetuiksi, jos käyttäjä antaa saman nimen jollekin toiselle objektille.

+ objektiorientoitunut

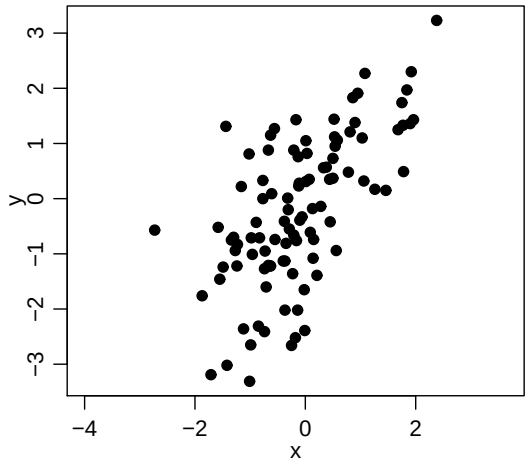
Monet R:n (valmiit) funktiot tunnistavat niille parametrina annetun objektin tyypin ja mahdollisesti suorittavat toiminnon riippuen objektin tyypistä. Esim. `plot`-funktio on tällainen.

Esim. Oletetaan että meillä on 100 mittausta muuttujista `x` ja `y`, joiden välistä lineaarista riippuvuutta haluamme tarkastella, ja muuttujat on ladattu objekteihin `x` ja `y` R:ssä.

```
> x
  [1] -1.58 -0.56 -1.02 -0.31 -1.44  1.75 -0.17 -0.89  0.95  2.38  1.84 -1.49
 [13]  0.38  0.56 -0.85 -0.67 -0.71  0.14  1.03  0.33 -0.37  0.03 -0.06 -0.63
 [25]  1.06 -0.32 -0.37 -0.96 -0.74  0.58 -0.83  0.50  1.68  0.45  0.21 -0.10
 [37] -0.77  1.26 -1.55  0.13 -1.24 -0.16  0.54 -0.14 -1.01 -1.23 -0.99 -0.64
 [49]  0.09  1.96  1.77 -0.21  0.07 -0.13 -0.02 -2.73  1.46  0.28  1.92  1.78
 [61] -0.73  0.15 -0.74  0.52 -1.16  0.01 -0.61  0.44 -0.11 -0.63  0.53  0.50
 [73] -0.67 -0.38 -1.71  0.78 -1.42  0.86  0.81 -0.01 -0.18 -0.55 -0.29 -1.34
 [85] -1.27 -1.12 -0.23 -0.40 -1.30  0.90 -0.12 -0.21 -0.98 -0.77  1.90  1.08
 [97]  0.01 -0.25 -0.35 -1.87
> y
  [1] -0.52  1.27  0.81 -0.20  1.31  1.74  1.43 -0.43  1.91  3.23  1.97 -1.24
 [13]  0.57 -0.94 -2.31  0.88 -1.60 -1.08  1.10  0.56 -2.02  0.82 -0.33 -1.22
 [25]  0.32  0.01 -1.13 -1.01 -2.41  1.06 -0.71  0.37  1.25 -0.42 -1.39 -0.39
 [37]  0.00  0.17 -1.46 -0.18 -1.22 -0.76  0.95 -2.02 -3.31 -0.83 -2.65 -1.22
 [49] -0.61  1.43  1.33 -0.66  0.35  0.76 -1.65 -0.57  0.15 -0.14  2.30  0.49
 [61] -0.95 -0.74 -1.27  1.44  0.22  1.05  0.09  0.35  0.28  1.15  1.12  0.73
 [73] -1.21 -0.41 -3.19  0.48 -3.02  1.83  1.21 -2.39 -2.52 -0.74 -0.55 -0.75
 [85] -0.94 -2.36 -1.36 -1.13 -0.70  1.38  0.23  0.88 -0.71  0.33  1.36  2.27
 [97]  0.31 -2.66 -0.81 -1.76
```

Hajontakuvion x:stä ja y:stä saamme plot-komennolla:

```
> plot(x, y, pch=16, asp=1)
> # pch määrittää pisteen tyylin, ks. ?points
> # asp = y/x sivusuhde
> # asp=1 -> x- ja y-koordinaattien akselien yksikkövälit samansuuruisia
```



Kuva 3: x:n ja y:n hajontakuvio.

Sovitetaan sitten regressiosuora `y = a + bx` simuloituun datajoukkoon. Tarkoituksena siis estimoida `a` ja `b`. Regressiosuoran saa R:ssä sovitettua funktiolla `lm`. Sijoitetaan `lm`-funktion antama tulos objektiin nimeltä `tulos`.

```
> tulos <- lm(y~x)
```

Funktion `lm` palauttaman objektin, joka on nyt objektissa `tulos`, voi antaa parametrina funktiolle `plot`. Funktio `plot` tunnistaa objektin tyypin ja tuottaa useita kuvia mallin sovituksen hyvyydestä.

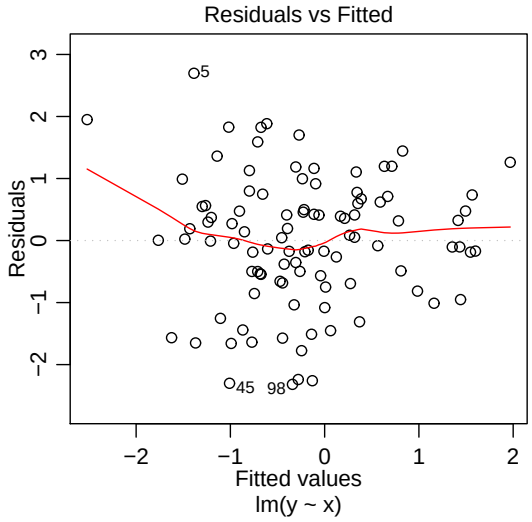
```
> plot(tulos)
```

- R on tulkkaava kieli.

Kieltä ei käännetä konekieliseen muotoon, vaan komennot suoritetaan välittömästi sen jälkeen kun käyttäjä on antanut ne, rivi riviltä. Tämän takia mm. silmukkarakenteet ovat hitaita ja niitä tulee välttää, jos mahdollista.

R:n suoritustehoa vaativissa sovelluksissa voidaan lisätä seuraavilla tavoilla:

- Käytetään valmiita funktioita. Valmiit funktiot on yleensä toteutettu tehokkaasti.
- Käytetään vektori- ja matriisilaskentaa, joka käsittelee yhdellä käskyllä suuren joukon dataa.



Kuva 4: Ensimmäinen komennon `plot(tulos)` tuottamista kuvista.

Esim. Simuloidaan 100000 standardinormaalijakautunutta muuttujaa ja etsitään ne simuloidut arvot, jotka ovat suurempia kuin 1.5.

```
> y <- rnorm(100000,0,1) # simulointi
> # Tapa 1.
> k <- 1
> x <- NULL
> system.time(           # mittaa ajan
+ for(i in 1:length(y)) {
+   if(y[i] > 1.5) {
+     x[k] <- y[i]
+     k <- k+1
+   }
+ }
+ )
   user  system elapsed
   0.41    0.00    0.42
> # Tapa 2. (suositeltava)
> system.time(
+ x <- y[y>1.5]
+ )
   user  system elapsed
      0       0       0
```

Lista funktioista, joita käytettiin

`c()`	vektorin luonti
`rm()`	objektin poistaminen
`plot()`	mm. hajontakuvion piirtäminen
`lm()`	lineaarisen mallin sovitus
`rnorm()`	normaalijakautuneiden muuttujien simulointi
`system.time()`	ajanotto

3 R:n yleisimmät dataobjektit

Vektorit, matriisit, listat ja datakehikot ovat dataobjektien perusmuotoja R:ssä. Tässä luvussa esitetään miten näitä objekteja voidaan luoda ja käsitellä.

Objektin rakennetta voidaan tarkastella funktiolla `str`. Funktiota voi soveltaa mihin tahansa objektiin, esimerkiksi numeeriseen vektoriin tai funktioon

```
> x <- c(1,2,3)
> str(x)
 num [1:3] 1 2 3

> str(lm)
function (formula, data, subset, weights, na.action, method = "qr", model = TRUE,
    x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL,
    offset, ...)
```

Monilla objekteilla on myös luokka ja tyyppi, jotka voi selvittää funktioilla `class` ja `typeof`. Esimerkiksi lineaarisen mallin tapauksessa tyyppi ja luokka määräytyvät seuraavasti

```
> x <- c(1,2,3); y <- c(3,2,1)
> z <- lm(y~x)
> class(z)
[1] "lm"
> typeof(z)
[1] "list"
```

Luokan voi ajatella kertovan, mitä metodeja luokan objektiin voidaan soveltaa ja kuinka niiden antamia tuloksia tulisi tulkita. Tyyppi puolestaan kertoo, minkälaista tietoa objekti sisältää.

Monet objektit saattavat olla todella suuria, jolloin niiden tarkasteleminen voi olla vaikeaa. Objektin alkuosan saa näkyviin funktion `head` avulla. Voimme esimerkiksi tulostaa 5×5-matriisin ensimmäisen rivin

```
> A <- matrix(runif(25),5,5)
> head(A,1)
          [,1]      [,2]      [,3]      [,4]     [,5]
[1,] 0.2546544 0.4106951 0.1644813 0.5611193 0.193146
```

Objektin nimet saa näkyviin funktiolla `names`. Esimerkiksi datakehikkoon sovellettuna

```
> x <- c(1,2,3); y <- c(3,2,1)
> d <- data.frame(x=x,y=y)
> names(d)
[1] "x" "y"
```

saadaan siis datakehikon komponenttien nimet.

3.1 Vektorit

Yksinkertaisin R:n tietorakenne on vektori. Vektori $x = (1.1, 3.5, -0.4, 5)$ saadaan muodostettua R:ssä komennolla

```
> x <- c(1.1,3.5,-0.4,5)
```

Tässä `<-` on sijoitusoperaattori ja `c()` on R-funktio, joka muodostaa vektorin sille parametreina annetuista alkioista. Yllä olevalla komennolla vektori sijoitetaan objektiin `x`, ts. vektorille annetaan nimi `x`. Objektin `x` sisällön saa tulostettua:

```
> x
[1]  1.1  3.5 -0.4  5.0
```

Vektorille suoritettava operaatio tehdään komponenteittain. Jos tulosta ei sijoiteta mihinkään objektiin, operaation tulos tulostetaan, `x`:n arvo ei muutu ja tulosta ei tallenneta.

```
> x^2
[1]  1.21 12.25  0.16 25.00
> 1/x
[1]  0.9090909  0.2857143 -2.5000000  0.2000000
> x
[1]  1.1  3.5 -0.4  5.0
```

Myös yksittäinen arvo on vektori, jonka pituus on 1.

```
> x <- 1
> x == c(1)
[1] TRUE
> c(x) == 1:1
[1] TRUE
>x[1] == 1
[1] TRUE
> length(x)
[1] 1
```

3.1.1 Vektoriaritmetiikkaa

Vektoreita voidaan siis käyttää aritmeettisissa lausekkeissa ja vektoreita käsitellään näissä komponenteittain. Tavallisimmat operaatiot ovat (ks. `?Arithmetic`)

```
+, -, *, /, ^
```

Lisäksi useimmat yleiset matemaattiset funktiot ovat käytettävissä: `log, exp, sin, cos, tan, sqrt` jne. Esim.

```
> y <- c(0.9, 1.5, 0.4, -3)
> y
[1]  0.9  1.5  0.4 -3.0
> z <- x+y
> z
[1] 2 5 0 2
> w <- exp(z)
> w
[1]   7.389056 148.413159   1.000000   7.389056
> log(w)
[1] 2 5 0 2
```

Lisäksi R:stä löytyy funktiot, jotka antavat vektorin alkoiden minimin, maksimin, vaihteluvälin, summan ja tulon sekä vektorin pituuden:

```
> x
[1]  1.1  3.5 -0.4  5.0
> min(x)                  # minimi
[1] -0.4
> max(x)                  # maksimi
[1] 5
> range(x)                # vaihteluväli, c(min(x),max(x))
[1] -0.4  5.0
> sum(x)                  # vektorin alkioiden summa
[1] 9.2
> prod(x)                 # vektorin alkioiden tulo
[1] -7.7
> length(x)               # vektorin pituus
[1] 4
```

Komennolla `rank(x)` saadaan `x`:n alkioiden järjestysluvut

```
> rank(x)
[1] 2 3 1 4
```

ja funktio `sort` järjestää vektorin alkiot järjestykseen joko pienimmästä suurimpaan tai toisin päin:

```
> sort(x)
[1] -0.4  1.1  3.5  5.0
> sort(x, decreasing = TRUE)
[1]  5.0  3.5  1.1 -0.4
```

Tilastolliset funktiot `mean`, `var` ja `sd` laskevat parametrina annetun vektorin keskiarvon, otosvarianssin ja otoskeskihajonnan.

```
> mean(x)                  # sum(x)/length(x)
[1] 2.3
> var(x)                   # sum((x-mean(x))^2)/(length(x)-1)
[1] 5.82
> sd(x)                    # sqrt(var(x))
[1] 2.412468
```

Funktio `cor` laskee kahden muuttujan välisen korrelaatiokertoimen, ks. `?cor`.

Huom. 1. Vektoreiden ei tarvitse välttämättä olla samanpituisia. Jos vektorit eivät ole samanpituisia, lausekkeen arvo on yhtä pitkä kuin mitä pisin vektori. Lyhyempien vektoreiden arvoja toistetaan niin monta kertaa peräkkäin että ne vastaavat pidemmän vektorin pituutta. Esimerkki kappaleesta 2:

```
> x <- c(1,2,3)
> y <- c(1,1,1,2,2,2,3,3)
> z <- x*y
Warning message: longer object length
        is not a multiple of shorter object length in: x * y
> # R antaa varoituksen, mutta komento suoritetaan
> z
[1] 1 2 3 2 4 6 3 6
> # mutta jos y:n pituus on x:n pituuden monikerta
> y <- c(1,1,1,2,2,2,3,3,3)
> z <- x*y
> z
[1] 1 2 3 2 4 6 3 6 9
> # ei varoituksia
```

Huom 2. Erityisesti vakiota toistetaan. Vakio on itse asiassa vektori, jossa on vain yksi alkio.

```
> x
[1]  1.1  3.5 -0.4  5.0
> 2*x
[1]  2.2  7.0 -0.8 10.0
```

Huom. 3. Suurimmaksi osaksi käyttäjän ei tarvitse välittää siitä, ovatko numeerisen vektorin alkiot kokonaislukuja, reaalilukuja vai jopa kompleksilukuja. Oletuksena R käsittelee kokonaisluvutkin reaalilukuina (double). Kokonaisluvun voi pakottaa integer-muotoon komennolla `as.integer(luku)` (tarpeellinen lähinnä kutsuttaessa C/Fortran-kielisiä funktioita). Objektin tyyppiä voi kysyä komennolla `typeof(objektinnimi)`. Jos haluaa työskennellä kompleksiluvuilla, tulee aina antaa luvun imaginaariosa (vaikka sen kerroin olisi 0). Esim.

```
> # Objektin tyypistä:
> a <- 2
> a
[1] 2
```

```
> typeof(a)
[1] "double"
> a <- as.integer(a)
> typeof(a)
[1] "integer"
>
> # Kompleksiluvuista:
> sqrt(-1)                    # Not a Number
[1] NaN
Warning message:
NaNs produced in: sqrt(-1)
> sqrt(-1+0i)
[1] 0+1i
```

R osaa siis myös kompleksiluvuilla laskemisen.

3.1.2 Jonojen luominen

R:ssä on erityisiä komentoja, joilla voidaan luoda jonoja.

Miten esimerkiksi voidaan generoida jono $1, 2, 3, 4, 5$? Tapoja on useita:

- Tapa 1. Käytetään funktiota `c` kuten edellä.

  ```
  > c(1,2,3,4,5)
  [1] 1 2 3 4 5
  ```

- Tapa 2.

  ```
  > 1:5
  [1] 1 2 3 4 5
  ```

- Tapa 3. `seq`-funktio, jolle annetaan parametreina jonon alku- ja loppupiste sekä joko jonon pituus (`length`) tai lisäys alkiosta toiseen (`by`).

  ```
  > seq(1, 5, by=1)
  [1] 1 2 3 4 5
  > # TAI pelkästään
  > seq(1, 5)
  [1] 1 2 3 4 5
  ```

Jonon ei tarvitse aina alkaa luvusta 1.

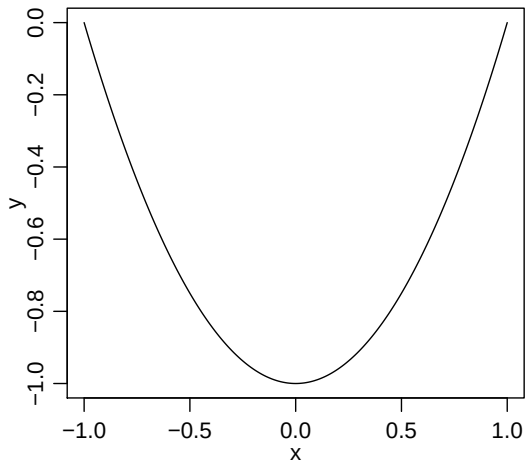
```
> 3:5
[1] 3 4 5
> seq(3, 5)
[1] 3 4 5
```

Tavalla 2 voidaan luoda vektoreita, jotka koostuvat kokonaisluvuista. Näitä voidaan kertoa ja jakaa vakioilla ja näin tuottaa myös ei-kokonaisluku-vektoreita. Mitä tekee $2 * 1 : 10 - 1$?

```
> 2*1:10
 [1]  2  4  6  8 10 12 14 16 18 20
> 2*1:10-1
[1]  1  3  5  7  9 11 13 15 17 19
> 1:10 / 10
 [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Toisaalta funktio `seq` lienee käyttökelpoisin tällaisten jonojen luonnissa. Se on hyödyllinen ja käytännöllinen myös kuvien piirtämisessä.

```
> x <- seq(-1,1,by=0.01)
> y <- x^2 - 1
> plot(x,y,type="l")
```



Entä miten luodaan jono $5, 4, 3, 2, 1$?

```
> c(5,4,3,2,1)
[1] 5 4 3 2 1
> 5:1
[1] 5 4 3 2 1
> seq(5,1)
[1] 5 4 3 2 1
> seq(5,1,by=-1)
[1] 5 4 3 2 1
```

Toistoja voidaan generoida `rep`-funktiolla seuraavasti:

```
> rep(1,times=5)
[1] 1 1 1 1 1
> rep(c(1,2,3),times=3)
[1] 1 2 3 1 2 3 1 2 3
```

3.1.3 Loogiset vektorit

Loogiset vektorit muodostetaan ehdollisella lausekkeella. Esim.

```
> x
[1]  1.1  3.5 -0.4  5.0
> x > 2
[1] FALSE  TRUE FALSE  TRUE
```

Tuloksena on ehdossa olevan vektorin (yllä `x`) pituinen vektori, jonka alkiot saavat arvoja `TRUE` ja `FALSE`. Loogiset operaattorit ovat

```
<     pienempi kuin
<=    pienempi tai yhtä suuri kuin
>     suurempi kuin
>=    suurempi tai yhtä suuri kuin
==    yhtä suuri kuin
!=    eri suuri kuin
```

Lisäksi jos `c1` ja `c2` ovat kaksi ehtolausetta, niin niiden leikkaus ("ja") saadaan komennolla `c1 & c2` ja yhdiste ("tai") komennolla `c1 | c2`. Esim.

```
> x
[1]  1.1  3.5 -0.4  5.0
> x > 1 & x <= 3.5
[1]  TRUE  TRUE FALSE FALSE
```

3.1.4 Merkkitietovektori

Kirjaimia ja sanoja käytetään R:ssä esimerkiksi kuvien akselien ja otsikkojen nimissä sekä objektien alkioiden nimeämisessä (ks. `?names`). Tähän tarvitaan merkkitietovektoreita. Merkkitieto annetaan R:ssä lainausmerkkien sisällä.

Esim. 1.

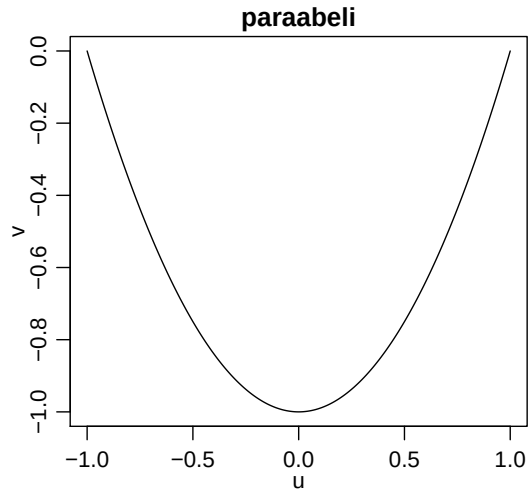
```
> c("a", "b", "c", "d")
[1] "a" "b" "c" "d"
```

Esim. 2.

```
> tulos <- c(5,4,8,6)
> tulos
[1] 5 4 8 6
> names(tulos) <- c("Helena", "Kalevi", "Tuija", "Petteri")
> tulos
 Helena  Kalevi   Tuija Petteri
      5       4       8       6
```

Esim. 3.

```
> x <- seq(-1,1,by=0.01)
> y <- x^2 - 1
> plot(x,y,type="l",xlab="u",ylab="v",main="paraabeli")
```



Kuva 5: Tälle kuvalle on annettu otsikko sekä x- ja y-akseleille nimet.

3.1.5 Vektorin alkioihin viittaaminen ja osajoukkojen valinta

Vektorin alkioihin viitataan hakasuluilla `[i]`, missä `i` on alkion indeksi. Ensimmäisen vektorin alkion indeksi R:ssä on 1. Esim.

```
> x
[1]  1.1  3.5 -0.4  5.0
> x[2]
[1] 3.5
```

Samalla kertaa voidaan valita myös useampi alkio.

```
> x[c(1,3)]
[1]  1.1 -0.4
> x[1:3]
[1]  1.1  3.5 -0.4
```

Vektorista voidaan myös jättää alkio tai alkiojoukko pois. Tämä ilmaistaan miinusmerkillä.

```
> x[-2]
[1]  1.1 -0.4  5.0
> x[-c(1,2)]
[1] -0.4  5.0
```

Loogista vektoria voidaan käyttää vektorin osajoukon valitsemiseen:

```
> x[x > 2]
[1] 3.5 5.0
> x[x > 1 & x <= 3.5]
[1] 1.1 3.5
```

Mikäli vektorin alkiot ovat nimettyjä, voidaan niihin viitata myös nimen perusteella.

```
> y <- c(a = 1, b = 2, c = 3)
> y
a b c
1 2 3
> y["a"]
a
1
> y[c("a", "c")]
a c
1 3
```

3.1.6 Puuttuva tieto

Aina kaikki muuttujan arvot eivät ole tunnettuja. Puuttuvan arvon symbolina R:ssä on `NA` (not available). Yleisesti ottaen, jos mille tahansa operaatiolle annetaan puuttuva arvo, operaatio antaa tulokseksi myös `NA`.

```
> xmiss
[1]  1.1   NA -0.4  5.0
> mean(xmiss)
[1] NA
```

Keskiarvoa ei siis voida laskea vektorista `xmiss`, mutta puuttuva tieto voidaan 'ohittaa' (jos puuttuminen ei ole selektiivistä) ja laskea keskiarvo vain tunnetuista havainnoista.

Funktio `is.na(x)` löytää vektorista `x` puuttuvan tiedon. Funktiolla `which` voidaan kysyä, mikä funktion `is.na` palauttamista arvoista on `TRUE` eli mikä vektorin `x` arvoista on puuttuva. Funktio `which` palauttaa kyseisen vektorin alkion indeksin.

```
> xmiss
[1]  1.1   NA -0.4  5.0
> is.na(xmiss)
[1] FALSE  TRUE FALSE FALSE
> which(is.na(xmiss))       # on sama kuin which(is.na(xmiss)==TRUE)
[1] 2
```

Tällöin vektorin `xmiss` ei-puuttuvista alkioista voidaan laskea keskiarvo komennolla

```
> mean(xmiss[!is.na(xmiss)])
[1] 1.9
```

Toisaalta funktiolle `mean` voi antaa parametrin `na.rm`, joka oletusarvoisesti on `FALSE`.

```
> mean(xmiss, na.rm=TRUE)
[1] 1.9
```

Jos `na.rm=TRUE`, niin `mean` poistaa vektorista puuttuvat havainnot ennen keskiarvon laskemista. Huomaa, että myös muilla R:n funktioilla saattaa olla valmiita toimintoja puuttuville havainnoille. Tämä selviää kyseisen funktion help-sivulla olevasta dokumentoinnista.

Jos havaintoja puuttuu, niin keskiarvon voi laskea myös helposti funktiolla `summary`. Funktion avulla voi myös selvittää puuttuvien havaintojen lukumäärän.

```
> summary(xmiss)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  -0.40    0.35    1.10    1.90    3.05    5.00       1
```

Huom. Toisenlainen puuttuva havainto on `NaN` (Not a Number). Se on seurausta numeerisesta laskusta, jonka tulosta ei ole määritelty. Esimerkkejä ovat

```
> sqrt(-1)
[1] NaN Warning message: NaNs produced in: sqrt(-1)
> 0/0
[1] NaN
> Inf - Inf
[1] NaN
```

3.1.7 Vektorien yhdistäminen ja muokkaaminen

Vektoreita voidaan yhdistää toisiinsa helposti `c`-funktiolla:

```
> x
[1]  1.1  3.5 -0.4  5.0
> c(x,1)
[1]  1.1  3.5 -0.4  5.0  1.0
> y <- c(-2.2, 2)
> c(x,y)
[1]  1.1  3.5 -0.4  5.0 -2.2  2.0
```

Vektoria voidaan muokata asettamalla mille tahansa alkiolle uusi arvo:

```
> x
[1]  1.1  3.5 -0.4  5.0
> x[2] <- 4.2
> x
[1]  1.1  4.2 -0.4  5.0
```

3.2 Matriisit

Matriisi (2-ulotteinen) voidaan R:ssä muodostaa `matrix`-funktiolla.

Esim. matriisi

$$A = \begin{bmatrix} 1.2 & 0.9 & 3.2 \\ 0.6 & 2.4 & 1.7 \end{bmatrix}$$

saadaan luotua objektiin A seuraavasti:

```
> A <- matrix(c(1.2,0.6,0.9,2.4,3.2,1.7), ncol=3)
> A
     [,1] [,2] [,3]
[1,]  1.2  0.9  3.2
[2,]  0.6  2.4  1.7
```

Ensimmäisessä parametrissa funktiolle `matrix` annetaan matriisin alkiot joko riveittäin tai sarakkeittain lueteltuna vektorimuodossa. Oletuksena on, että alkiot annetaan sarakkeittain. Jos alkiot halutaan antaa riveittäin, niin annetaan määrite `byrow=TRUE` (`T`).

```
> A <- matrix(c(1.2,0.9,3.2,0.6,2.4,1.7), ncol=3, byrow=T)
> A
     [,1] [,2] [,3]
[1,]  1.2  0.9  3.2
[2,]  0.6  2.4  1.7
```

Määritteessä `ncol` kerrotaan, kuinka monta saraketta matriisissa on. Vastaavasti voitaisiin käyttää määritettä `nrow`, joka määräisi matriisin rivien lukumäärän.

Huom. Jos annettujen alkioiden lukumäärä ei täsmää matriisin alkioiden määrään, joka on määritelty `ncol/nrow`-määritteillä, R monistaa annettuja alkioita.

Diagonaalimatriisi voidaan luoda `diag`-funktiolla. Esim.

```
> diag(1, nrow=3, ncol=3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> diag(c(1.2,2.4,3), nrow=3, ncol=3)
     [,1] [,2] [,3]
[1,]  1.2  0.0    0
[2,]  0.0  2.4    0
[3,]  0.0  0.0    3
```

Matriisilta voidaan kysyä sen rivien ja sakekkeiden lukumäärää:

```
> nrow(A)
[1] 2
> ncol(A)
[1] 3
```

tai dimensiota:

```
> dim(A)
[1] 2 3
```

Komennolla `is.matrix(A)` objektilta voidaan kysyä, onko se matriisi. Lisäksi esim. vektorista voidaan tehdä matriisi `as.matrix`-funktiolla.

```
> is.matrix(A)
[1] TRUE
> x
[1]  1.1  3.5 -0.4  5.0
> as.matrix(x)
     [,1]
[1,]  1.1
[2,]  3.5
[3,] -0.4
[4,]  5.0
```

Matriisista voidaan tehdä vektori seuraavasti:

```
> as.vector(A)
[1] 1.2 0.6 0.9 2.4 3.2 1.7
```

Komennolla `as.vector()` matriisin alkiot luetaan siis sarakkeittain vektoriksi.

Huom. Moniulotteisia (ja myös 1- ja 2-ulotteisia) vektoreita voidaan luoda `array`-funktiolla, ks. `?array`. Matriisi `matrix` on 2-ulotteinen 'array'.

3.2.1 Matriisin alkioihin viittaaminen

Matriisin alkioihin voidaan viitata samaan tapaan kuin vektorin alkioihin. Viittaus matriisin alkioon (i,j) tapahtuu hakasulkeilla `[i,j]`:

```
> A
     [,1] [,2] [,3]
[1,]  1.2  0.9  3.2
[2,]  0.6  2.4  1.7
> A[2,3]
[1] 1.7
```

Matriisin rivi i saadaan valittua hakasuluilla `[i,]` ja sarake j hakasuluilla `[,j]`. Esim.

```
> A[2,]
[1] 0.6 2.4 1.7
> A[,3]
[1] 3.2 1.7
```

```
> A[,1:2]
     [,1] [,2]
[1,]  1.2  0.9
[2,]  0.6  2.4
```

Mikäli matriisit rivit tai sarakkeet ovat nimettyjä, voidaan niihin viitata myös nimen perusteella samaan tapaan kuin vektoreiden tapauksessa.

3.2.2 Matriisien ja vektoreiden yhdistely

Funktioilla `rbind` ja `cbind` voidaan yhdistää matriiseja ja vektoreita isommiksi matriiseiksi riveittäin ja sarakkeittain. Esim.

```
> A
     [,1] [,2] [,3]
[1,]  1.2  0.9  3.2
[2,]  0.6  2.4  1.7
> x <- 1:3
> B <- matrix(c(2,2,2,2),ncol=2)
> rbind(A,x)
  [,1] [,2] [,3]
   1.2  0.9  3.2
   0.6  2.4  1.7
x  1.0  2.0  3.0
> rbind(A,matrix(x,nrow=1))
     [,1] [,2] [,3]
[1,]  1.2  0.9  3.2
[2,]  0.6  2.4  1.7
[3,]  1.0  2.0  3.0
> cbind(A,B)
     [,1] [,2] [,3] [,4] [,5]
[1,]  1.2  0.9  3.2    2    2
[2,]  0.6  2.4  1.7    2    2
> cbind(x,x)
     x x
[1,] 1 1
[2,] 2 2
[3,] 3 3
```

3.2.3 Matriisien summa ja tulo sekä matriisin transpoosi

Matriiseille ovat sallittuja samat laskutoimitukset kuin mitä vektoreillekin. Näissä operaatioissa, kuten `+,-,*,/` tai `sqrt, sin` matriiseja käsitellään kuten vektoreitakin: alkioittain! Siis esimerkiksi:

```
> A
```

```
     [,1] [,2] [,3]
[1,]  1.2  0.9  3.2
[2,]  0.6  2.4  1.7
> B <- matrix(c(1,2,3,4,5,6),ncol=3)
> B
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> A + B
     [,1] [,2] [,3]
[1,]  2.2  3.9  8.2
[2,]  2.6  6.4  7.7
> A * B
     [,1] [,2] [,3]
[1,]  1.2  2.7 16.0
[2,]  1.2  9.6 10.2
```

Näissä operaatioissa matriisien dimensioiden tulee tietenkin olla samat.

Matriisien kertolaskun saa suoritettua operaatiolla `%*%`. Tällöin matriisien dimensioiden pitää olla yhteensopivia matriisikertolaskun sääntöjen mukaisesti. Matriisin transpoosi saadaan funktiolla `t`. Esim.

```
> t(B)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> t(B) %*% A
     [,1] [,2] [,3]
[1,]  2.4  5.7  6.6
[2,]  6.0 12.3 16.4
[3,]  9.6 18.9 26.2
> A %*% t(B)
     [,1] [,2]
[1,] 19.9 25.2
[2,] 16.3 21.0
```

3.2.4 Matriisin kääntäminen, determinantin laskeminen ja lineaarisen yhtälöryhmän ratkaiseminen

Neliömatriisin käänteismatriisi ja determinantti saadaan helposti `solve`- ja `det`-funktioilla:

```
> C <- matrix(c(1,0.3,0.3,1.5),ncol=2)
> C
     [,1] [,2]
[1,]  1.0  0.3
```

```
[2,]  0.3  1.5
> solve(C)
           [,1]       [,2]
[1,]  1.0638298 -0.2127660
[2,] -0.2127660  0.7092199
> det(C)
[1] 1.41
```

Itse asiassa `solve`-funktio on yleisimmin tarkoitettu lineaaristen yhtälöiden ratkaisemiseen. Esim. yhtälön `Cx=b`, missä `C` on kuten edellä ja `b=(1,2)'` ratkaisu `x` saadaan seuraavasti:

```
> b <- as.matrix(c(1,2))
> b
     [,1]
[1,]    1
[2,]    2
> x <- solve(C, b)
> x
          [,1]
[1,] 0.6382979
[2,] 1.2056738
```

`b` voi olla myös matriisi:

```
> b <- matrix(c(1,2,3,4),ncol=2)
> b
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> solve(C, b)
          [,1]     [,2]
[1,] 0.6382979 2.340426
[2,] 1.2056738 2.198582
```

Oletusarvoisesti `b` on yksikkömatriisi ja näin ollen jos `b`:tä ei määritellä funktiokutsussa, niin `solve` kääntää parametrina annetun matriisin.

3.2.5 Matriisin hajotelmia

R:ssä on funktioita matriisin hajotelmien laskemiseen.

Esimerkiksi pääkomponenttianalyysi ja faktorianalyysi perustuvat matriisin omainaisarvohajotelmaan. Matriisin X omainaisarvohajotelman $X = V\Lambda V'$, missä $V = (e_1, e_2, \ldots, e_n)$ (ominaisvektorit) ja $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ (ominaisarvot) saa laskettua funktiolla `eigen`. Esim.

```
> C.eigen <- eigen(C)
> C.eigen
```

```
$values
[1] 1.6405125 0.8594875

$vectors
          [,1]       [,2]
[1,] 0.4241554 -0.9055894
[2,] 0.9055894  0.4241554

> C.eigen$values
[1] 1.6405125 0.8594875
> C.eigen$vectors
          [,1]       [,2]
[1,] 0.4241554 -0.9055894
[2,] 0.9055894  0.4241554
```

Vektori `C.eigen$values` sisältää ominaisarvot ja ominaisvektorit ovat matriisin `C.eigen$vectors` sarakkeissa.

Choleskyn hajotelman reaaliselle symmetriselle positiivisesti definiitille neliömatriisille (kovarianssimatriisi on tällainen) voi laskea funktiolla `chol`.

```
> C
     [,1] [,2]
[1,]  1.0  0.7
[2,]  0.3  1.5
> chol(C)
     [,1]     [,2]
[1,]    1 0.700000
[2,]    0 1.004988
```

Funktio `chol` palauttaa yläkolmiomatriisin U siten, että $U'U = C$.

Lisäksi esim. singulaariarvohajotelma, ks. `?svd`.

3.3 Taulukot

Taulukot ovat vektorien ja matriisien moniulotteinen yleistys. Vektorin voi ajatella olevan yksiulotteinen taulukko ja vastaavasti matriisin kaksiulotteinen taulukko. Taulukoita voidaan hyödyntää esimerkiksi tilanteissa, joissa ollaan kiinnostuneita useiden muuttujien arvoista yhtäaikaisesti. Taulukoita voi R:ssä luoda monella eri tavalla. Hyödynnetään ensin funktiota `array`, jolle annetaan parametreina vektori taulukon alkiosta sekä vektori, joka määrää taulukon dimensiot.

Luodaan kolmiulotteinen taulukko, jolla on kolme riviä, kaksi saraketta ja kaksi "alitaulukkoa". Dimensiot voi kolmiulotteisessa tapauksessa mieltää esimerkiksi pituudeksi, leveydeksi ja korkeudeksi.

```
> A <- array(1:12, dim = c(3, 2, 2))
```

```
> A
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

Dimensioiden ei tarvitse rajoittua kolmeen. Tarkastellaan neliulotteista taulukkoa, jolla on kaksi riviä, kolme saraketta ja kaksi alitaulukkoa, joista jokaisella on edelleen kaksi alitaulukkoa. Alitaulukot luetellaan nyt kahdella indeksillä.

```
> B <- array(1:24, dim = c(2, 3, 2, 2))
> B
, , 1, 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2, 1

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 1, 2

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 2, 2

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```

Taulukoita voi luoda myös määrittämällä vektorille dimensioparametri. Tämä on hyödyllinen tapa järjestää esimerkiksi valmis aineisto, joka on jo luettu vektoriksi. Myös matriiseja voi luoda vektoreista tällä tavalla.

```
> x <- 1:12
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> dim(x) <- c(3, 2, 2)
> x
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

3.4 Listat

R:n lista on objekti, joka on järjestetty joukko objekteja, ns. komponentteja. Komponenttien ei tarvitse olla samaa tyyppiä eikä yhtä pitkiä. Lista voi esimerkiksi sisältää numeerisia ja merkkitietoa sisältäviä vektoreita sekä matriiseja. Listan komponenteille on tapana antaa nimet, jolloin niihin on helppo viitata.

Esim. olkoon data objekteissa `x`, `y` ja `z` siten, että `x` ja `y` ovat puiden x- ja y-koordinaatit ja `z` on puiden halkaisija.

```
> x
 [1] 0.3 0.7 0.8 1.1 1.5 2.5 3.8 4.1 4.6 4.6 5.0 5.1 6.4 6.5 6.5 7.3
> y
 [1] 4.6 7.1 4.3 6.6 6.1 3.5 3.7 4.7 0.7 6.5 6.4 1.4 4.0 2.0 1.2 1.5
> z
 [1] 15 10  4 15 13 26 33 15 11 14 20 25 12  5 10 15
```

Muodostetaan lista:

```
> puut <- list(x1=x, x2=y, halkaisija=z)
> puut
$x1
 [1] 0.3 0.7 0.8 1.1 1.5 2.5 3.8 4.1 4.6 4.6 5.0 5.1 6.4 6.5 6.5 7.3

$x2
 [1] 4.6 7.1 4.3 6.6 6.1 3.5 3.7 4.7 0.7 6.5 6.4 1.4 4.0 2.0 1.2 1.5

$halkaisija
 [1] 15 10  4 15 13 26 33 15 11 14 20 25 12  5 10 15
```

Tässä listassa on 3 komponenttia. Huomaa, että listan pituus on 3:

```
> length(puut)
[1] 3
```

Listan komponentteihin voidaan viitata:

```
> puut$x1
 [1] 0.3 0.7 0.8 1.1 1.5 2.5 3.8 4.1 4.6 4.6 5.0 5.1 6.4 6.5 6.5 7.3
> puut$x2
 [1] 4.6 7.1 4.3 6.6 6.1 3.5 3.7 4.7 0.7 6.5 6.4 1.4 4.0 2.0 1.2 1.5
> puut$halkaisija
 [1] 15 10  4 15 13 26 33 15 11 14 20 25 12  5 10 15
```

Tässä komponentit ovat vektoreita ja niitä voidaan käsitellä kuin mitä tahansa vektoreita, esim.

```
> mean(puut$halkaisija)
[1] 15.1875
```

Listan komponentteihin voidaan viitata suoraan, jos suoritetaan `attach`-komento:

```
> attach(puut)
> halkaisija
 [1] 15 10  4 15 13 26 33 15 11 14 20 25 12  5 10 15
> mean(halkaisija)
[1] 15.1875
```

Käytön jälkeen on syytä suorittaa komento

```
> detach(puut)
```

jotta vältytään sekaannuksilta.

Komponenttien ei siis tarvitse olla samaa tyyppiä. Edellä käsiteltyyn listaan voidaan myös liittää merkkitietovektori:

```
> laji
 [1] "mänty" "koivu" "koivu" "mänty" "mänty" "mänty" "mänty" "koivu" "mänty"
[10] "koivu" "mänty" "mänty" "mänty" "koivu" "koivu" "mänty"
> puut <- list(x1=x, x2=y, halkaisija=z, laji=laji)
> puut
$x1
 [1] 0.3 0.7 0.8 1.1 1.5 2.5 3.8 4.1 4.6 4.6 5.0 5.1 6.4 6.5 6.5 7.3

$x2
 [1] 4.6 7.1 4.3 6.6 6.1 3.5 3.7 4.7 0.7 6.5 6.4 1.4 4.0 2.0 1.2 1.5
```

```
$halkaisija
 [1] 15 10  4 15 13 26 33 15 11 14 20 25 12  5 10 15

$laji
 [1] "mänty" "koivu" "koivu" "mänty" "mänty" "mänty" "mänty" "koivu" "mänty"
[10] "koivu" "mänty" "mänty" "mänty" "koivu" "koivu" "mänty"
```

Esimerkiksi mäntyjen halkaisijoiden keskiarvo voidaan nyt laskea seuraavasti:

```
> mean(puut$halkaisija[puut$laji=="mänty"])
[1] 18.5
```

Lasketaan mäntyjen sekä koivujen halkaisijoiden keskiarvot käyttäen hyväksi `tapply`-funktiota:

```
> tapply(puut$halkaisija, as.factor(puut$laji), mean)
    koivu     mänty
 9.666667 18.500000
```

Yllä funktio `as.factor` muuttaa merkkitietovektorin `puut$laji` faktoriksi. (Itse asiassa yllä tämä komento ei ole välttämätön, ks. `?tapply`.)

```
> as.factor(puut$laji)
 [1] mänty koivu koivu mänty mänty mänty mänty koivu mänty koivu mänty mänty
[13] mänty koivu koivu mänty
Levels: koivu mänty
```

3.5 Datakehikot

Samanpituisista, toisiinsa liittyvistä vektoreista voidaan muodostaa datakehikko, jolloin vektoreiden data on hyvin hallittavissa yhtenä kokonaisuutena. Datakehikossa yksi rivi voi esimerkiksi vastata yhtä yksilöä ja sarakkeet sisältää mittauksia yksilöistä. Datakehikkona (tai listana) on myös kätevä palauttaa suurempikin määrä tuloksia omatekoisesta funktiosta.

Datakehikko luodaan `data.frame`-funktiolla seuraavasti:

```
> puut.frame <- data.frame(x1=x,x2=y,halkaisija=z,laji=laji)
> puut.frame
    x1  x2 halkaisija  laji
1  0.3 4.6         15 mänty
2  0.7 7.1         10 koivu
3  0.8 4.3          4 koivu
4  1.1 6.6         15 mänty
5  1.5 6.1         13 mänty
6  2.5 3.5         26 mänty
7  3.8 3.7         33 mänty
8  4.1 4.7         15 koivu
9  4.6 0.7         11 mänty
```

```
10 4.6 6.5         14 koivu
11 5.0 6.4         20 mänty
12 5.1 1.4         25 mänty
13 6.4 4.0         12 mänty
14 6.5 2.0          5 koivu
15 6.5 1.2         10 koivu
16 7.3 1.5         15 mänty
```

Merkkitieto luetaan oletusarvoisesti faktorityyppiseksi muuttujaksi. Jos jostain syystä halutaan, että `laji` säilyy merkkitietona, voidaan yllä olevassa komennossa kirjoittaa `laji=I(laji)` (yleensä tähän ei ole syytä). Esim. faktoria voidaan käyttää:

```
> tapply(puut.frame$halkaisija, puut.frame$laji, mean)
    koivu     mänty
 9.666667 18.500000
```

Datakehikon komponentteihin voidaan viitata kuten listan komponentteihin:

```
> puut.frame$halkaisija
 [1] 15 10  4 15 13 26 33 15 11 14 20 25 12  5 10 15
```

`attach()` ja `detach()` toimivat datakehikolle kuten listalle. Datakehikosta voidaan valita rivejä tai sarakkeita myös samaan tapaan kuin matriiseista

```
> puut.frame[ ,"halkaisija"]
 [1] 15 10  4 15 13 26 33 15 11 14 20 25 12  5 10 15
> puut.frame[ ,c("x1", "halkaisija")]
    x1 halkaisija
1  0.3         15
2  0.7         10
3  0.8          4
4  1.1         15
5  1.5         13
6  2.5         26
7  3.8         33
8  4.1         15
9  4.6         11
10 4.6         14
11 5.0         20
12 5.1         25
13 6.4         12
14 6.5          5
15 6.5         10
16 7.3         15
> puut.frame[5, ]
    x1  x2 halkaisija  laji
5  1.5 6.1         13 mänty
```

```
> puut.frame[2:5, ]
    x1  x2 halkaisija  laji
2  0.7 7.1         10 koivu
3  0.8 4.3          4 koivu
4  1.1 6.6         15 mänty
5  1.5 6.1         13 mänty
```

Lista funktioista, joita käytettiin

`str()`	objektin rakenne
`class()`	objektin luokka
`head()`	objektin alkuosa
`names()`	objektin nimet
`typeof()`	objektin tyyppi
`c()`	vektorin tai listan luominen
`min()`	minimi
`max()`	maksimi
`range()`	vaihteluväli
`sum()`	vektorin alkioiden summa
`prod()`	vektorin alkioiden tulo
`length()`	vektorin pituus
`rank()`	komponenttien järjestysluvut
`sort()`	järjestäminen suuruusjärjestykseen
`mean()`	keskiarvo
`var()`	varianssi
`sd()`	keskihajonta
`sqrt()`	neliöjuuri
`seq()`	jonon luominen
`rep()`	toistaminen
`is.na()`	puuttuvan tiedon etsiminen
`summary()`	objektin sisällön yleinen tarkastelu
`matrix()`	matriisin luominen
`diag()`	diagonaalimatriisin luominen
`nrow()`	matriisin rivien lukumäärä
`ncol()`	matriisin sarakkeitten lukumäärä
`dim()`	matriisin dimensio
`is.matrix()`	onko matriisi?
`as.matrix()`	matriisiksi asettaminen
`as.vector()`	vektoriksi asettaminen
`array()`	taulukon luominen
`rbind()`	matriisien/vektoreiden yhdistäminen riveittäin
`cbind()`	matriisien/vektoreiden yhdistäminen sarakkeittain
`solve()`	yhtälöryhmän ratkaiseminen, matriisin kääntö
`det()`	matriisin determinantti
`eigen()`	ominaisarvohajotelma
`chol()`	Choleskyn hajotelma

`list()`	listan muodostaminen
`attach()`	listan/datakehikon komponenttien nimien käyttöönottaminen
`detach()`	komponenttien nimien vapauttaminen
`tapply()`	suorita toiminto (FUN) jokaiselle faktorin (tai faktoreiden) määrittelemälle vektorin osajoukolle
`as.factor()`	faktoriksi muuttaminen
`data.frame()`	datakehikon luominen

4 Datan lukeminen

Edellä on käsitelty datan lukemista näppäimistöltä, ts. kirjoittamalla arvot R:ään.

```
> x <- c(1,2.3,4,6.5)
>
> # Tähän on myös toinen tapa:
> x <- scan()
1: 1 2.3
3: 4
4: 6.5
5:
Read 4 items
> x
[1] 1.0 2.3 4.0 6.5
```

Todelliset datat ovat kuitenkin yleensä tallennettuna tiedostoissa ja myös halutaan tallentaa tiedostoihin, jotta ne ovat aina uudelleen käsiteltävissä. Seuraavassa käsitellään merkkipohjaisen (ASCII) tiedoston lukemista R:ään. Lisäksi esitetään, miten aineisto, joka on SAS- tai Excel-muodossa, voidaan lukea myös suoraan sellaisenaan R:ään.

Esimerkkiaineistona on `pigsx`-aineisto (R. Littell, G. Milliken and W. Stroup, 2006. SAS for Mixed Models, Second Edition. http://ftp.sas.com/samples/A59882). Kyseessä on sikojen kasvuaineisto, jossa 60 sikaa on täydellisesti satunnaistettu kolmeen käsittelyryhmään (trt). Sikojen painot on mitattu vieroituksen jälkeen kuusi kertaa kuuden päivän välein. Aineisto on tekstimuotoisena `pigsx.txt`-tiedostossa.

Lisätietoja tiedostojen lukemisesta löytyy R:n ohjesivustojen kautta:

```
> help.start()
```

ja sieltä '`R Data Import/Export`'.

4.1 Tekstitiedosto

4.1.1 Vektorimuotoisen datan lukeminen

Vektorimuotoiset datat kuten

```
1.28 -1.84 1.47 6.1 10.36 6.68 1.49 -1.54 4.49 3.35 -0.1 2.33
-0.92 1.75 4.48 1.99 4.73 0.88 -0.78 -0.93 -2.46 -1.18 -0.45 4.71
-1.09 1.08 4.61 1.1 6.16 1.8 -2.94 -1.95 2.17 -2.89 3.4 -0.85
-0.74 1.63 -5.28 1.56 2.24 0.77 6.66 3.69 4.4 5.26 0.4 6.35 1.49
2.62 0.3 -3.3 4.19 5.44 1.61 0.18 2.64 4.46 0.77 6.53 -4.02 4.14
-0.89 2.84 3.12 4.09 3.14 6.99 -0.44 1.64 -2.84 5.28 0.5 -1.27
-3.81 3.91 4.47 6.7 5.17 1.44 7.18 1.33 6.05 -3.36 0.45 5.49 2.21
1.48 -2.18 3.54 -2.5 3.77 1.29 0.17 5.4 2.01 8.19 2.91 1.38 1.92
-0.53 2.64 4.13 2.31 3.28 2.75 0.11 6.15
```

joka on tiedostossa `data1.dat`, voidaan lukea R:ään funktion `scan` avulla seuraavasti:

```
> x <- scan("data1.dat")
Read 108 items
```

Tämän jälkeen `x` sisältää yllä olevan datan arvot.

Jos luetaan merkkitietovektoria, niin komento on muotoa

```
> y <- scan("merkkitietodata.dat", what=character())
```

Parametri `what` määrittää luettavan datan tyypin.

Myös numeerinen matriisi/datakehikko voidaan myös lukea `scan`-funktiolla, mutta helpompaa on käyttää `read.table`-funktiota.

4.1.2 Datakehikon lukeminen

Aineisto `pigsx.txt` on seuraavanlaista muotoa

```
trt pig day0    day6    day12   day18   day24   day30
1   1   14      22.1    27.7    31.8    35.3    32.6
1   2   15.2    23.8    32.6    40      42.4    44.6
1   3   13.9    22.1    28.1    29.8    33.6    32.9
1   4   16.7    27.1    32.7    38      39.9    43.8
1   5   17      26.9    36      41.9    47.7    50.8
1   6   16.2    25.4    28.7    39.8    43.4    40.2
1   7   11.5    24.8    29.9    37.1    42.4    43.5
1   8   14.9    25      28.1    35.9    36.2    35.6
1   9   16.9    21      29.8    31.6    34.6    33.7
...
3   19  16      25.2    31.3    37.7    39.6    43.2
3   20  15.8    20.6    29.7    31.2    32.2    33.2
```

Tässä siis jokainen sarake vastaa yhtä muuttujaa ja rivi yksilöä, ensimmäisellä rivillä on muuttujien nimet. Tällainen ASCII-muodossa oleva data voidaan lukea R:ään kätevimmin `read.table`-funktiolla:

```
> pigs <- read.table("C:\\omat\\datat\\pigsx.txt", header=TRUE)
```

Ensimmäisenä parametrina annetaan tiedosto, josta aineisto halutaan lukea. Tiedoston hakemistopolku voidaan antaa kokonaisuudessaan, jolloin R löytää tiedoston riippumatta R:n (sen hetkisestä) työskentelykansiosta. Huomaa, että kenoviiva täytyy kirjoittaa R:ssä tuplana (tai vaihtoehtoisesti se tulee korvata yksinkertaisella kauttaviivalla "/"). Parametri `header=TRUE` kertoo, että annetussa aineistossa on otsikkorivi. Jos otsikkoriviä ei ole, niin silloin `header=FALSE` (tämä on oletuksena).

Luotu objekti `pigs` on datakehikko ja se voidaan tulostaa `R console` -ikkunaan (kuten mikä tahansa R-objekti):

```
> pigs
   trt pig day0 day6 day12 day18 day24 day30
1    1   1 14.0 22.1  27.7  31.8  35.3  32.6
2    1   2 15.2 23.8  32.6  40.0  42.4  44.6
3    1   3 13.9 22.1  28.1  29.8  33.6  32.9
4    1   4 16.7 27.1  32.7  38.0  39.9  43.8
...
59   3  19 16.0 25.2  31.3  37.7  39.6  43.2
60   3  20 15.8 20.6  29.7  31.2  32.2  33.2
```

Jos sarakkeiden arvot luettavassa tiedostossa on erotettu toisistaan jollakin muulla merkillä kuin välilyönnillä, tämä merkki tulee kertoa funktiolle `read.table` parametrissa `sep`. Lisää funktion toiminnasta voi lukea help-sivulta, joka aukeaa komennolla

```
> ?read.table
```

Suurien aineistojen lukeminen saattaa olla nopeampaa `scan`-funktiolla. Yllä oleva esimerkkiaineisto voidaan lukea `scan`-funktiolla seuraavasti muuttaen se samalla matriisimuotoon:

```
> pigs2 <- matrix(scan("C:\\omat\\datat\\pigsx.txt", skip=1),
+   ncol=8, byrow=TRUE)
Read 480 items
```

Funktio `scan` lukee tiedoston arvot riveittäin yhteen vektoriin. Komennolla `skip=1` määrätään, että funktio jättää tiedoston alusta 1:n rivin (otsikkorivin) huomioimatta ja aloittaa datan lukemisen vasta toiselta riviltä (oletuksena `skip=0`). Funktio `matrix` muuttaa funktion `scan` palauttaman vektorin matriisimuotoon. Tässä tulee käyttää määritettä `byrow=TRUE`, sillä funktio `scan` lukee aineiston riveittäin.

Huom. Huomaa, että toisen funktion palauttama tulos voidaan antaa suoraan toiselle funktiolle ilman, että sitä tarvitsee ensin sijoittaa johonkin objektiin.

Edellä olevan komennon jälkeen `pigs2`-objektin sisältö on (tulostetaan tässä vain 5 ensimmäistä riviä)

```
> pigs2[1:5,]
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    1 14.0 22.1 27.7 31.8 35.3 32.6
[2,]    1    2 15.2 23.8 32.6 40.0 42.4 44.6
[3,]    1    3 13.9 22.1 28.1 29.8 33.6 32.9
[4,]    1    4 16.7 27.1 32.7 38.0 39.9 43.8
[5,]    1    5 17.0 26.9 36.0 41.9 47.7 50.8
```

Tämä matriisi voidaan muuttaa datakehikoksi, jolloin sen sarakkeet voidaan myös nimetä ja niihin voidaan viitata näillä nimillä.

```
> pigs2 <- as.data.frame(pigs2)
> names(pigs2) <- c("trt","pig","day0","day6","day12","day18","day24","day30")
```

```
> pigs2[1:5,]
  trt pig day0 day6 day12 day18 day24 day30
1   1   1 14.0 22.1  27.7  31.8  35.3  32.6
2   1   2 15.2 23.8  32.6  40.0  42.4  44.6
3   1   3 13.9 22.1  28.1  29.8  33.6  32.9
4   1   4 16.7 27.1  32.7  38.0  39.9  43.8
5   1   5 17.0 26.9  36.0  41.9  47.7  50.8
```

4.1.3 Datakehikon lukeminen listaksi

Joskus voi olla mielekästä lukea aineisto lista-objektiksi R:ään. Data `pigsx.dat` voidaan lukea myös listaksi:

```
> pigs.lista <- scan("C:\\omat\\datat\\pigsx.txt",
+  list(trt=0,pig=0,day0=0,day6=0,day12=0,day18=0,day24=0,day30=0), skip=1)
Read 60 records
```

Toisessa argumentissa `what` kerrotaan minkä tyyppistä dataa ollaan lukemassa, sekä annetaan niille komponenteille nimet. Määrite 0 tarkoittaa, että kyseessä on numeerinen tieto. Merkkitiedolle voidaan lyhyesti kirjoittaa lainausmerkit.

4.1.4 Eripituisten datarivien lukeminen

Joskus data voi olla muotoa, jossa eri riveillä on eri määrä havaintoja. Esimerkki tällaisesta aineistosta on `kaenmunat.dat`, jonka sisältö on seuraava:

```
# rautiaisen ja ruokokerttusen pesistä löytyneiden käenmunien koot
rautiainen 22.0 23.9 20.9 23.8 25.0 24.0 21.7 23.8 22.8 23.1
ruokokerttunen 23.2 22.0 22.2 21.2 21.6 21.9 22.0 22.9 22.8
```

Aineisto saadaan luettua seuraavasti listaksi:

```
> # Lasketaan rivien havaintojen lukumäärät ('-1', koska rivinimet)
> nf <- count.fields("kaenmunat.dat", sep="") - 1
> # Luetaan aineisto read.table-funktiolla
> dat <- read.table("kaenmunat.dat", sep="", fill=TRUE,
+   colClass=(rep(numeric(),nf)), row.names=1)
> # Muutetaan objekti dat listaksi, poistetaan 'puuttuvat' (NA) havainnot
> lista <- lapply(as.data.frame(t(data.matrix(dat))), na.omit)
> # Jos halutaan poistaa 'ruokokerttusessa' olevat lisätiedot
> # NA:iden poistamisesta, se voidaan tehdä seuraavasti:
> lista2 <- list(rautiainen=lista$rau,ruokokerttunen=as.numeric(lista$ruoko))
> lista2
$rautiainen
 [1] 22.0 23.9 20.9 23.8 25.0 24.0 21.7 23.8 22.8 23.1

$ruokokerttunen
[1] 23.2 22.0 22.2 21.2 21.6 21.9 22.0 22.9 22.8
```

4.1.5 Aineisto sisältää puuttuvaa tietoa

Puuttuvan tiedon merkki R:ssä on `NA`. Jos puuttuva tieto on koodattu aineistoon `NA`:na, onnistuu aineiston lukeminen kuten yllä. Jos puuttuva tieto on koodattu aineistoon muulla merkillä, tämä tieto voidaan antaa määritteenä funktioille `scan` ja `read.table`, ks. `?scan` ja `?read.table`.

4.1.6 Aineisto sisältää merkkitietoa

Datan lukeminen `read.table`-funktiolla onnistuu kuten yllä. Kun aineisto luetaan R:ään, luettava merkkitieto muutetaan faktori-tyyppiseksi, ts. aineistossa esiintyvät 'nimet' ajatellaan faktorin eri tasoiksi.

Merkkitietoa voidaan lukea myös `scan`-funktiolla (määrite `what`).

4.2 SAS-tiedosto

1) Tallenna (`Export data`) data SAS-ohjelmasta `Table Delimited File (*.txt)` -muodossa ja toimi kuten edellä.

2) Luetaan SAS-data suoraan R:ään käyttäen hyväksi funktiota `read.ssd`, joka sijaitsee R:n kirjastossa `foreign`. Tämän funktion käyttäminen edellyttää, että myös SAS-ohjelma on käytettävissä! Kirjasto otetaan käyttöön komennolla `library(foreign)` ja funktion kutsu on alla olevaa muotoa.

```
> library(foreign)
> pigs3 <- read.ssd(libname="C:\\omat\\SASdatat", sectionnames="pigsx",
+  sascmd="C:\\Program Files\\SAS\\SAS 9.1\\sas.exe")
```

Ensimmäisenä parametrina funktiolle annetaan tiedoston nimi, mistä toisena parametrina annettava SAS-tiedosto löytyy. SAS-tiedoston päätettä ei pidä antaa. Määritteessä `sascmd` tulee antaa täysi polku SAS:iin (`sas.exe`).

```
> pigs3[1:5, ]
  TRT PIG DAY0 DAY6 DAY12 DAY18 DAY24 DAY30
1   1   1 14.0 22.1  27.7  31.8  35.3  32.6
2   1   2 15.2 23.8  32.6  40.0  42.4  44.6
3   1   3 13.9 22.1  28.1  29.8  33.6  32.9
4   1   4 16.7 27.1  32.7  38.0  39.9  43.8
5   1   5 17.0 26.9  36.0  41.9  47.7  50.8
```

Tällöin komponenttien nimet ovat isoilla kirjaimilla.

Huom. R:ssä isot ja pienet kirjaimet ovat eri merkkejä (siis `TRT` on eri kuin `trt`).

4.3 Excel-tiedosto

1) Tallennetaan aineisto ASCII-muotoon taulukkolaskentaohjelmasta (esim. Excel). Kätevintä lienee valita 'Text (Tab delimited)(*.txt)'. Tällöin aineisto talletetaan tekstimuotoisena, missä välilyönnit (tab) erottavat sarakkeet toisistaan. Lisäksi desimaalipilkut tulee korvata pisteillä.

2) Kirjastossa `gdata` on funktio `read.xls`, joka osaa lukea `.xls`-tiedoston R:ään. Kirjaston `gdata` voi ladata R:n nettisivuilta `http://www.r-project.org/ -> CRAN`.

3) Kirjaston `gdata` sijaan voidaan käyttää myös kirjastoa `XLConnect`.

Huom. funktio `read.xls` tarvitsee toimiakseen Perl-kääntäjän ja `XLConnect`ia käytettäessä tulee Javan olla asennettuna koneelle. Usein on helpompaa tallentaa haluttu Excel-tiedosto aluksi csv-tiedostoksi ja lukea tämä sisään funktiolla `read.csv2`.

```
> library(gdata)
> pigs4 <- read.xls(file="C:\\omat\\datat\\pigsx.xls")
> pigs4[1:5,]
  trt pig day0 day6 day12 day18 day24 day30
1   1   1 14.0 22.1  27.7  31.8  35.3  32.6
2   1   2 15.2 23.8  32.6  40.0  42.4  44.6
3   1   3 13.9 22.1  28.1  29.8  33.6  32.9
4   1   4 16.7 27.1  32.7  38.0  39.9  43.8
5   1   5 17.0 26.9  36.0  41.9  47.7  50.8
```

Tyhjä ruutu `.xls`-tiedostossa luetaan R:ään automaattisesti puuttuvaksi tiedoksi `NA`.

4.4 SPSS-tiedosto

1) Tallenna `Save as type:` data SPSS-ohjelmasta `Comma delimited (*.csv)` -muodossa. Tämän jälkeen aineston voi lukea R:ään esimerkiksi funktiolla `read.csv2`.

2) SPSS-ohjelmiston .sav-tiedostoja voi tuoda suoraan R:ään käyttämällä kirjaston `foreign` funktiota `read.spss`. Funktiota käytettäessä ei itse SPSS-ohjelmiston tarvitse olla asennettuna. Käyttökelpoinen optio aineistojen lukemiseksi suoraan datakehikoksi on `to.data.frame=TRUE`

```
> library(foreign)
> tomsato <- read.spss("tomsato.sav", to.data.frame = T)
Warning message:
In read.spss("tomsato.sav", to.data.frame = T) :
  tomsato.sav: Unrecognized record type 7, subtype 18 encountered in system file
> head(tomsato)
  SATO ISTUTUST LAJIKE
1  7.9       10      1
2  9.2       10      1
3 10.5       10      1
4 11.2       20      1
5 12.8       20      1
6 13.3       20      1
```

Funktio antaa käytettäessä usein varoituksia, mutta niistä ei yleensä tarvitse välittää. On kuitenkin syytä aina tarkastaa luettu aineisto virheellisten arvojen varalta.

Lista funktioista, joita käytettiin

`scan()`	datan lukeminen listaksi tai vektoriksi
`read.table()`	datakehikon lukeminen
`as.data.frame()`	muuttaminen datakehikoksi
`names()`	objektin komponenttien nimet
`count.fields()`	osajoukkojen komponenttien lukumäärien laskeminen
`lapply()`	suorita toiminto (FUN) jokaiselle vektorin/listan komponentille
`read.ssd()`	SAS-tiedoston lukeminen
`read.xls()`	Excel-tiedoston lukeminen
`read.spss()`	SPSS-tiedoston lukeminen

5 Datakehikko-datan käsittelyä

Oletaan, että aineisto `pigsx` on luettu R:ään objektiin nimeltä `pigs`. Tämä objekti on *datakehikko*. Seuraavassa sovelletaan osaksi kappaleen 3 asioita.

5.1 Osajoukkojen valinta

Datakehikon komponentteihin ja alkioihin viittaaminen.

1) Rivi- ja sarakenumeroiden/nimien avulla. Datakehikon komponentteihin (sarakkeisiin) voidaan viitata niiden nimillä seuraavasti

```
> pigs$day0
 [1] 14.0 15.2 13.9 16.7 17.0 16.2 11.5 14.9 16.9 13.5 15.3 14.5 15.7 13.2 18.0
[16] 17.1 14.8 13.9 15.7 10.9 15.1 14.6 15.3 14.6 18.7 13.4 17.7 14.3 11.5 15.6
[31] 16.2 16.6 15.1 14.9 16.6 11.8 15.1 12.9 17.1 15.4 16.6 13.0 14.0  9.2 17.3
[46] 16.5 13.5 15.1 17.8 15.0 11.3 17.6 14.3 14.7 13.0 10.0 13.5 13.7 16.0 15.8
> pigs$day6
 [1] 22.1 23.8 22.1 27.1 26.9 25.4 24.8 25.0 21.0 22.7 23.5 25.5 26.0 22.9 27.0
[16] 28.2 23.0 23.4 22.8 23.4 27.7 26.2 22.4 26.3 28.2 24.0 25.2 28.9 25.4 23.7
[31] 24.5 26.9 27.6 22.0 26.7 25.1 25.1 23.8 27.3 22.1 25.3 20.0 17.6 18.7 23.4
[46] 25.8 24.6 19.7 22.5 23.2 20.1 21.8 22.8 24.2 21.6 19.7 22.4 20.5 25.2 20.6
```

Vaihtoehtoisesti voidaan käyttää valintaa `pigs[,"day0"]`.

Samoihin datakehikon sarakkeisiin päästään käsiksi myös viittaamalla sarakkeen numeroon

```
> pigs[,3]
 [1] 14.0 15.2 13.9 16.7 17.0 16.2 11.5 14.9 16.9 13.5 15.3 14.5 15.7 13.2 18.0
[16] 17.1 14.8 13.9 15.7 10.9 15.1 14.6 15.3 14.6 18.7 13.4 17.7 14.3 11.5 15.6
[31] 16.2 16.6 15.1 14.9 16.6 11.8 15.1 12.9 17.1 15.4 16.6 13.0 14.0  9.2 17.3
[46] 16.5 13.5 15.1 17.8 15.0 11.3 17.6 14.3 14.7 13.0 10.0 13.5 13.7 16.0 15.8
> pigs[,4]
 [1] 22.1 23.8 22.1 27.1 26.9 25.4 24.8 25.0 21.0 22.7 23.5 25.5 26.0 22.9 27.0
[16] 28.2 23.0 23.4 22.8 23.4 27.7 26.2 22.4 26.3 28.2 24.0 25.2 28.9 25.4 23.7
[31] 24.5 26.9 27.6 22.0 26.7 25.1 25.1 23.8 27.3 22.1 25.3 20.0 17.6 18.7 23.4
[46] 25.8 24.6 19.7 22.5 23.2 20.1 21.8 22.8 24.2 21.6 19.7 22.4 20.5 25.2 20.6
```

tai voidaan myös valita kaksi saraketta kerralla

```
> pigs[,3:4]
   day0 day6
1  14.0 22.1
2  15.2 23.8
3  13.9 22.1
4  16.7 27.1
```

```
5  17.0 26.9
...
59 16.0 25.2
60 15.8 20.6
```

Vastaavasti voidaan valita rivi (yksilö)

```
> pigs[3,]
  trt pig day0 day6 day12 day18 day24 day30
3   1   3 13.9 22.1  28.1  29.8  33.6  32.9
```

tai rivejä (yksilöitä)

```
> pigs[1:3,]
  trt pig day0 day6 day12 day18 day24 day30
1   1   1 14.0 22.1  27.7  31.8  35.3  32.6
2   1   2 15.2 23.8  32.6  40.0  42.4  44.6
3   1   3 13.9 22.1  28.1  29.8  33.6  32.9
```

Yksittäinen alkio voidaan valita

```
> pigs[1, 3]
[1] 14
```

Esim. `pigs$day0` on *vektori*, jonka alkioihin voidaan viitata `pigs$day0[1], pigs$day0[2], ..., pigs$day0[60]`.

Valinnat viitaten rivi- ja sarakenumeroihin toimivat datakehikoille siis samoin kuin matriiseille.

2) Loogisten operaattoreiden avulla. Osajoukkojen valinnassa voidaan käyttää myös ehtolauseita, ks. kappale 3.1. Miten tätä voidaan soveltaa datakehikkoon? Valitaan siat, joille on annettu käsittely (`trt`) 1:

```
> pigs[pigs$trt==1,]
   trt pig day0 day6 day12 day18 day24 day30
1    1   1 14.0 22.1  27.7  31.8  35.3  32.6
2    1   2 15.2 23.8  32.6  40.0  42.4  44.6
3    1   3 13.9 22.1  28.1  29.8  33.6  32.9
...
20   1  20 10.9 23.4  30.0  35.6  37.4  38.7
```

Sika, jonka id on 1 ja on saanut käsittelyn 1:

```
> pigs[pigs$pig==1 & pigs$trt==1,]
  trt pig day0 day6 day12 day18 day24 day30
1   1   1   14 22.1  27.7  31.8  35.3  32.6
```

Tätä kyseistä sikaa vastaavat tiedot saadaan siis valituksi myös viitaten rivinumeroon, kun tämä tiedetään. Loogisten operaattoreiden avulla valittaessa sijaintia datakehikossa ei tarvitse tietää.

Osadatakehikon `pigs[pigs$trt==1,]` komponentteihin (sarakkeisiin) voidaan edelleen viitata

```
> pigs[pigs$trt==1,]$day0
 [1] 14.0 15.2 13.9 16.7 17.0 16.2 11.5 14.9 16.9 13.5 15.3 14.5 15.7 13.2
[15] 18.0 17.1 14.8 13.9 15.7 10.9
> pigs[pigs$trt==1,]$day30
 [1] 32.6 44.6 32.9 43.8 50.8 40.2 43.5 35.6 33.7 37.8 36.1 44.7 43.4 34.0 48.0
[16] 53.3 40.4 34.6 40.2 38.7
```

tai valitsemalla sarakkeet 3 tai `"day0"` ja 8 tai `"day30"`

```
> pigs[pigs$trt==1,"day0"]
 [1] 14.0 15.2 13.9 16.7 17.0 16.2 11.5 14.9 16.9 13.5 15.3 14.5 15.7 13.2
[15] 18.0 17.1 14.8 13.9 15.7 10.9
> pigs[pigs$trt==1,"day30"]
 [1] 32.6 44.6 32.9 43.8 50.8 40.2 43.5 35.6 33.7 37.8 36.1 44.7 43.4 34.0 48.0
[16] 53.3 40.4 34.6 40.2 38.7
```

5.2 Faktorit

Merkkitietoinen data luetaan R:ään oletusarvoisesti faktoriksi. Numeerinen tieto numeerisena. Faktori saattaa kuitenkin olla koodattuna numeroina kuten pigsx-aineistossa käsittely `trt`. Tämä muuttuja voidaan muuttaa faktoriksi `as.factor`-funktiolla:

```
> pigs$trt
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[39] 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
> # muutetaan faktoriksi:
> pigs$trt <- as.factor(pigs$trt)
> pigs$trt
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[37] 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
Levels: 1 2 3
```

Huomaa, että tulostettaessa faktori, tulostetaan myös sen tasot.

Huom. R:ssä on useita funktioita nimeltä `as.jotakin` ja `is.jotakin`, joilla voidaan yrittää muuttaa objekti johonkin tiettyyn muotoon tai voidaan kysyä objektilta, onko se jotakin tiettyä muotoa.

Faktoreita voi luoda myös funktiolla `gl`. Funktiolle annetaan parametreinä tasojen määrä ja tason koko. Esimerkiksi

```
> gl(3,5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
```

joka siis luo faktorin, jossa on kolme tasoa, ja jokaisen tason pituus on viisi. Funktiolle voidaan antaa lisäparametrinä myös haluttu faktorin pituus, jolloin kahden ensimmäisen parametrin määräämää rakennetta toistetaan tarvittaessa.

```
> gl(3,5,length=25)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2
Levels: 1 2 3
```

Olemassaolevan faktorin tasojen nimiä voidaan muokata funktiolla `levels`. Oletusarvoisesti esimerkiksi `gl`-funktiota käytettäessä tasojen nimet ovat vain kokonaislukuja.

```
> x <- gl(2,5)
> x
 [1] 1 1 1 1 1 2 2 2 2 2
Levels: 1 2
> levels(x) <- c("pieni", "suuri")
> x
 [1] pieni pieni pieni pieni pieni suuri suuri suuri suuri suuri
Levels: pieni suuri
```

5.3 Muuttujan arvojen muuttaminen

Datakehikon/vektorin arvoja voidaan muokata yksinkertaisesti sijoittamalla niiden paikalle jokin toinen arvo.

Aineistossa muuttuja `pig` on id sioille eri käsittelyryhmissä. Koska kyseessä ovat eri siat eri käsittelyryhmissä, on sekaannusten välttämiseksi jossakin tapauksissa välttämätöntä vaihtaa sikojen id:t yksikäsitteisiksi. Yksikäsitteisyyttä tarvitaan ainakin sekamallien sovittamisessa aineistoon (jos näin ei tehdä, esimerkiksi sekamallien sovittamiseen käytettävä funktio `lme` luulee, että yksittäiset siat 1, 2,..., 20 ovat saaneet kaikki kolme käsittelyä).

Luodaan uusi datakehikko (kopioidaan sisältö) ja tehdään muutos tähän.

```
> pigsW <- pigs
> pigsW$pig          # datakehikossa 'aiemmin'
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  1  2
[23]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  1  2  3  4
[45]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> 1:60               # vektori, joka sisältää alkiot 1, 2, ..., 60
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
[23] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
[45] 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
> pigsW$pig <- 1:60  # sijoitus datakehikon komponenttiin "pig"
> pigsW$pig          # sisältö sijoituksen jälkeen
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
[23] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
[45] 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
```

Aineistoa voidaan myös tarkastella ja yksittäisiä arvoja voidaan muuttaa myös komennon

```
> fix(pigsW)
```

avaamasta taulusta. Vastaavaan lopputulokseen päästään myös käyttämällä funktiota `edit` seuraavasti

```
> pigsW <- edit(pigsW)
```

Funktiolla `edit` voidaan myös pelkästään tarkastella aineistoa

```
> edit(pigsW)
```

Tällöin muutokset aineistoon eivät tallennu. Funktiota `fix` voidaan myös käyttää kokonaan uuden datakehikon luomisen yhteydessä.

```
> d <- data.frame()
> fix(d)
```

Useiden lukujen muuttaminen näin on kuitenkin työlästä.

5.4 Uuden muuttujan ja rivin luonti datakehikkoon

Datakehikkoon voidaan luoda uusi komponentti antamalla sille nimi ja sijoittamalla arvot samanaikaisesti. Luodaan seuraavassa `pigs`-datakehikkoon uusi muuttuja `d`, johon laskemme kasvun ajanhetkestä 0 ajanhetkeen 30.

```
> pigs$d <- pigs$day30-pigs$day0
```

Huom. jos datakehikko sisältäisi jo komponentin nimeltä `d`, tämän komponentin arvot korvattaisiin arvoilla `pigs$day30-pigs$day0`. Kun tämän nimistä komponenttia ei vielä ole, luodaan se sijoituksen yhteydessä. Tarkastetaan, mitä syntyi:

```
> pigs[1:5,]
  trt pig day0 day6 day12 day18 day24 day30    d
1   1   1 14.0 22.1  27.7  31.8  35.3  32.6 18.6
2   1   2 15.2 23.8  32.6  40.0  42.4  44.6 29.4
3   1   3 13.9 22.1  28.1  29.8  33.6  32.9 19.0
4   1   4 16.7 27.1  32.7  38.0  39.9  43.8 27.1
5   1   5 17.0 26.9  36.0  41.9  47.7  50.8 33.8
```

Loimme siis datakehikkoon uuden komponentin nimeltä `d`. Tämä on nyt myöhemminkin käytössä ja kätevästi tallessa samassa objektissa kuin muukin aineisto.

Huom. Vaihtoehtoisesti datakehikkoon voidaan luoda uusia komponentteja ja muokata jo olemassa olevia komponentteja käyttäen hyväksi funktiota `transform`. Esim. komponentin `d` luominen:

```
> pigs <- transform(pigs, d=day30-day0)
```

Datakehikkoon voidaan lisätä uusia rivejä `rbind`-funktiolla. Kyseisellä funktio yhdistää kaksi datakehikkoa toisiinsa jos niiden sarakkeet täsmäävät.

```
> new.pig <- data.frame(trt = 1, pig = nrow(pigs) + 1, day0 = 15.5, day6 = 25.2,
day12 = 33.1, day18 = 40.0, day24 = 45.7, day30 = 50.1, d = 50.1 - 15.5)
pigs <- rbind(pigs, new.pig)
```

5.5 Aineiston muokkaaminen toiseen muotoon

Usein aineisto on talletettu kuten `pigsx` aineisto: yksi rivi vastaa yhtä yksilöä ja sarakkeet sisältävät kaikki kyseiseen yksilöön liittyvät kovariaatit sekä mittaukset (muoto 1). Tilastolliset mallinnusfunktiot R:ssä kuitenkin yleensä vaativat, että aineistossa on vain yksi havainto per rivi, ts. kaikki havainnot on esitetty yhdessä sarakkeessa (muoto 2). Toisaalta kompakti esitysmuoto 1 saattaa olla muuten kätevämpi. Näin ollen datan muokkaus muodosta toiseen on tarpeen. Tämä onnistuu esimerkiksi `reshape`-funktiolla.

Objekti `pigsW` on datakehikko muodossa 1. Muutetaan tämä muotoon 2. Datakehikossa `pigsW` komponentti `pig` määrittelee yksilöt (siat) yksikäsitteisesti. Tämä kerrotaan argumentissa `idvar`. Jos yksilön määrittelee kaksi (tai useampi) muuttujaa, kuten datakehikossa `pigs`, annetaan määrite tyyliin `idvar = c("trt", "pig")`.

```
> pigsL <- reshape(data = pigsW, varying = list(names(pigsW)[3:8]),
+  v.names = "weight", timevar = "day", idvar = "pig", direction = "long",
+  times = seq(0, 30, by=6))
> pigsL[1:10, ]
     trt pig day weight
1.0    1   1   0   14.0
2.0    1   2   0   15.2
3.0    1   3   0   13.9
4.0    1   4   0   16.7
5.0    1   5   0   17.0
6.0    1   6   0   16.2
7.0    1   7   0   11.5
8.0    1   8   0   14.9
9.0    1   9   0   16.9
10.0   1  10   0   13.5
```

Ks. `?reshape` mitä muut argumentit ovat.

Vastaavasti muutos toiseen suuntaan saadaan:

```
> test <- reshape(pigsL, idvar="pig", varying=list(c("day0", "day6", "day12",
+  "day18", "day24", "day30")), v.names="weight", timevar = "day",
+  direction="wide")
> test[1:5,]
```

```
    trt pig day0 day6 day12 day18 day24 day30
1.0   1   1 14.0 22.1  27.7  31.8  35.3  32.6
2.0   1   2 15.2 23.8  32.6  40.0  42.4  44.6
3.0   1   3 13.9 22.1  28.1  29.8  33.6  32.9
4.0   1   4 16.7 27.1  32.7  38.0  39.9  43.8
5.0   1   5 17.0 26.9  36.0  41.9  47.7  50.8
```

5.6 Aineistojen järjestäminen sarakkeen mukaan

Aineisto voidaan järjestää jonkin sarakkeen mukaan helposti käyttäen kirjaston `doBy` funktiota `orderBy`. Ladataan kirjasto käyttöön

```
> library(doBy)
```

ja järjestetään aineisto esim. sikojen `day0`-painojen mukaan:

```
> orderBy(~day0, data=pigs)
   trt pig day0 day6 day12 day18 day24 day30    d
44   3   4  9.2 18.7  27.0  26.5  30.3  28.2 19.0
56   3  16 10.0 19.7  26.0  30.2  32.7  31.9 21.9
20   1  20 10.9 23.4  30.0  35.6  37.4  38.7 27.8
51   3  11 11.3 20.1  25.4  29.7  27.9  30.4 19.1
7    1   7 11.5 24.8  29.9  37.1  42.4  43.5 32.0
...
15   1  15 18.0 27.0  34.4  41.6  45.4  48.0 30.0
25   2   5 18.7 28.2  39.0  43.8  51.2  50.8 32.1
```

Ensimmäinen argumentti kertoo, minkä komponentin suhteen datakehikko järjestetään ja `data`-argumentissa annetaan datakehikko. Tulos voidaan tietenkin sijoittaa johonkin objektiin. Kirjasto `doBy` sisältää muitakin hyödyllisiä funktioita datakehikko-rakenteille, esim. `summaryBy`.

5.7 Toimintoja datakehikolle osaryhmissä

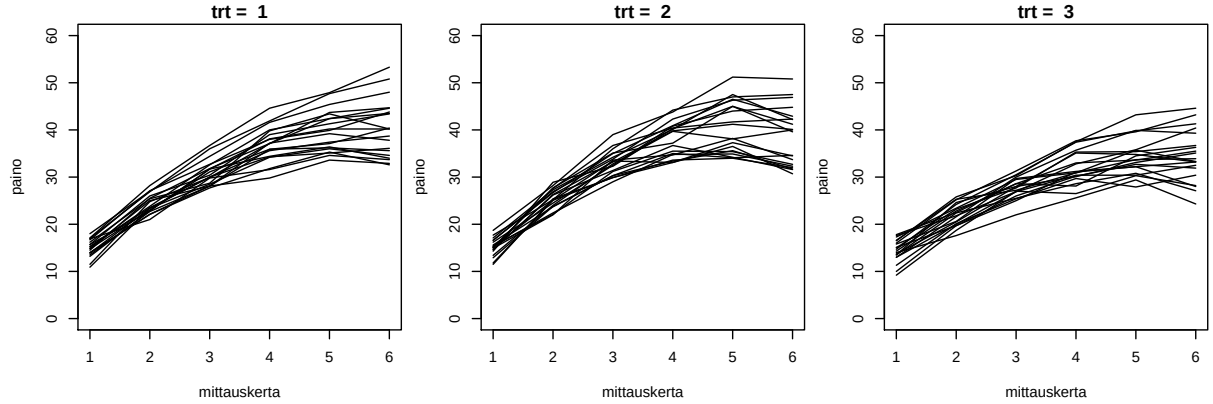
Funktio `by` on hyödyllinen, kun halutaan suorittaa komentoja datakehikolle osaryhmissä. Funktion kutsu on muotoa `by(data, INDICES, FUN, ...)`, missä `data` määrittelee datakehikon, `INDICES` faktorin/faktorit, jotka jakavat datakehikon ryhmiin. (Esimerkiksi `trt` jakaa datakehikon `pigs` kolmeen ryhmään.) Argumentissa `FUN` annetaan funktio, jota käytetään kuhunkin ryhmään erikseen. `...` viittaa lisäparametreihin, jotka mahdollisesti välitetään funktiolle `FUN`. Käyttäjä voi myös itse ohjelmoida käytettävän funktion.

Esimerkiksi piirretään sikojen kasvukäyrät käsittelyn `trt` ryhmissä erikseen (kuva 6):

```
> pigs.kkayrat <- function(x) {
+    plot(1:6, x[x$pig==1, 3:8], type="l", xlab="mittauskerta",
+         ylab="paino", ylim=c(0,60), main=paste("trt = ", x$trt[1]))
+    for(sikaid in 2:nrow(x)) {
```

```
+       lines(1:6,x[x$pig==sikaid, 3:8])
+    }
+ }
> X11()
> # postscript(file="kasvukayrat.eps",height=3,width=9,paper="special",horizontal=F)
> par(mfrow=c(1,3), mar=c(4,4,1.5,0.5))
> by(pigs, pigs$trt, pigs.kkayrat)
```

Huom. tämä funktio on spesifi `pigs`-datalle.



Kuva 6: Kasvukäyrät.

Lista funktioista, joita käytettiin

`as.factor()`	aseta faktoriksi
`gl()`	faktorin luonti
`fix()`	editoi objektia
`edit()`	avaa tekstieditorin objektin tarkastelemiseksi
`data.frame()`	alusta datakehikko
`transform()`	datakehikon muunnos, esimerkiksi uuden muuttujan luomiseksi
`rbind()`	kahden datakehikon yhdistäminen
`reshape()`	muuta datakehikon muotoa 'pitkän' ja 'lyhyen' muodon välillä
`orderBy()`	järjestä datakehikko jonkin sarakkeen mukaan
`by()`	suorita toiminto (FUN) datakehikon osajoukoille
`paste()`	yhdistä vektorit merkkijonoksi muuttamisen jälkeen
`function()`	funktion luonti
`plot()`	R-objektien piirtäminen
`lines()`	käyrien piirtäminen

6 Tallentaminen

6.1 .Rdata:n tallentaminen

R-data voidaan tallentaa R:n `File`-valikosta `Save Workspace...` tai komennolla

```
save.image(file="nimi.Rdata")
```

Tällöin R tekee `.Rdata`-tiedoston, joka sisältää R:n työpöydällä (ks. `ls()`) olevat objektit. Tämä tiedosto voidaan uudelleen avata R:ään ja työskentelyä jatkaa siitä, mihin jäätiin edellisellä kerralla. Kun R-data on tallennettu näin, voidaan R:ää sulkiessa kysymykseen "Save workspace image?" vastata "Ei".

Osan R:n työpöydällä olevista objekteista voi tallentaa `save`-funktiolla, ks. `?save`. `.Rdata`-tiedosto voidaan ladata R:ään funktiolla `load` tai valikosta `File` → `Load Workspace...` tai vain 'klikkaamalla' kyseistä tiedostoa resurssienhallinnassa.

6.2 Tulostus tiedostoon

Vektorin tulostus tiedostoon

Tulostetaan esimerkiksi sikojen painojen muutokset d tiedostoon:

```
> write(pigs$d, file="pigs_d.dat", ncolumns=4)
```

Määritteessä `ncolumns` kerrotaan, montako arvoa tulostetaan yhdelle riville.

Matriisin tulostus tiedostoon

Tulostetaan matriisi

```
> A
     [,1] [,2] [,3]
[1,]  1.2  0.9  3.2
[2,]  0.6  2.4  1.7
```

tiedostoon A.dat:

```
> write(t(A),file="A.dat",ncol=3)
```

Parametrina täytyy antaa tulostettavan matriisin transpoosi, joka saadaan funktiolla `t`. Määritteessä `ncol` annetaan matriisin sarakeittein lukumäärä.

Datakehikon tulostus

Datakehikon `pigs` tulostus:

```
> write.table(pigs, file="pigs.dat")
> write.table(pigs, file="pigs.dat", row.names=FALSE, quote=FALSE)
> write.table(pigs, file="pigs.dat", sep=",")
```

Ensimmäinen komento käy, kun halutaan lukea aineisto myöhemmin takaisin R:ään. Toinen komento lienee yleiskäyttöisin. Se tuottaa tiedoston, joka on samaa muotoa kuin `pigsx.txt`.

Määritteessä `sep` voidaan antaa merkki, jolla arvot erotetaan tulostettavassa taulussa. Oletuksena on välilyönti. Vastaavasti funktiossa `read.table` pitää antaa määrite `sep`, kun luetaan tiedostoa, jossa erotusmerkkinä on jokin muu kuin väli.

Tulostus sink-funktiolla

Tulostus voidaan ohjata `sink`-funktiolla tiedostoon. Komennon

```
> sink(file="nimi.dat")
```

jälkeen kaikki tulostukset ohjautuvat R konsolin sijaan tiedostoon `nimi.dat` kunnes annetaan komento

```
> sink()
```

6.3 SAS-tiedoston kirjoittaminen R:stä

Funktion `write.foreign` avulla voidaan kirjoittaa datakehikko muiden tilastollisten ohjelmien (SAS, SPSS, Stata) muotoon. Esim.

```
> library(foreign)
> write.foreign(pigs, "C:\\omat\\datat\\mydata.sd2",
+               "C:\\omat\\datat\\mydata.sas", package="SAS")
```

Tässä luodaan kaksi tiedostoa: `mydata.sd2` ja `mydata.sas`. Ne sisältävät aineiston ja SAS-koodin aineiston lukemiseksi SAS-ohjelmaan.

Lista funktioista, joita käytettiin

`save.image()`	.Rdata:n tallennus
`write()`	vektorin/matriisin tulostus tiedostoon
`write.table()`	datakehikon tulostus tiedostoon
`sink()`	tulostuksen ohjaus tiedostoon
`write.foreign()`	SAS/SPSS/Stata-tiedoston luominen

7 Grafiikka

R:n grafiikkaominaisuudet ovat hyvin kehittyneet. Tässä käsitellään yleisimpiä grafiikkafunktioita.

Kuvan piirtoon liittyvät yleensä vaiheet:

1. Tulostuslaitteen määrittely: `R Graphics`-ikkuna (`X11()`) tai tiedosto (esim. funktiot `postscript`, `jpg`).
2. Asetetaan kuvakoko ja ulkoasu (`par`-funktiolla). Jos komentoja ei anneta, käytetään R:n oletusasetuksia.
3. Piirretään kuva. Esim.

   ```
   > plot(x,y)
   ```

4. Jos kyseessä on tulostus tiedostoon, suljetaan tulostusikkuna `dev.off()`-komennolla. Tällä komennolla voidaan sulkea myös R:n grafiikkaikkuna, joka on avattu `X11()`-komennolla.

7.1 Tulostuslaitteen määrittely

Tulostuslaitteen määrittely.
1) Uusi `R Graphics`-ikkuna voidaan avata komennolla

```
> X11()
```

Muussa tapauksessa uusi kuva piirretään edelliseen ikkunaan edellisen kuvan päälle. R avaa automaattisesti uuden ikkunan, jos yhtään ikkunaa ei ole auki.
2) Tulostus voidaan ohjata eps-tiedostoon:

```
> postscript(file="tiedosto.eps",height=4,width=4,
+            paper="special",horizontal=F)
```

Tässä `height` ja `width` määräävät kuvan koon ja määritteiden `paper="special"` ja `horizontal=F` avulla kuvasta saadaan määrätyn kokoinen sekä kuva oikein päin. Oletuksena kuva piirretään A4-paperille (`paper="a4"`) horisontaalisesti (`horizontal=T`), ks. lisätietoja `?postscript`. Komennon jälkeen kaikki graafiset toiminnot menevät tiedostoon `tiedosto.eps` kunnes annetaan komento

```
> dev.off()
```

3) `.pdf`-tiedostoon tulostus avataan komennolla

```
> pdf("tiedosto.pdf")
```

ja kuvat piirretään tiedostoon peräkkäin kunnes annetaan komento `dev.off()`.

7.2 Kuvakoon ja ulkoasun säätely

Piirrettävän kuvan kuvakokoa ja ulkoasua voidaan säätää `par`-funktiolla (ks. `?par`), jonka tyypillisimmät määritteet ovat

```
par(pin=c(2,2))       kuvakoko 2x2 tuumaa
par(mfrow=c(3,2))     piirtää samaan ikkunaan 6 kuviota 3x2-matriisin
                      (3 riviä, 2 saraketta), oletusarvo c(1,1)
```

Lisäksi hyödyllisiä ovat (etenkin tiedostoon tulostamisessa)

```
par(mar=c(4,2.5,1.5,0.5))   kuinka monta riviä jätetään marginaalia
                            kuvan eri puolille
                            c(alas, vasemmalle, yläpuolelle, oikealle)
                            (oletusarvo c(5, 4, 4, 2) + 0.1)
par(mgp=c(1.5,0.7,0))       kuinka paljon jätetään tilaa akselien nimille,
                            asteikoille ja akselille (oletusarvo c(3, 1, 0))
```

Näillä saadaan vähennettyä kuvien reunoille jäävää tyhjää tilaa. Komento voidaan antaa muodossa

```
> par(mfrow=c(1,1),mar=c(4,2.5,1.5,0.5),mgp=c(1.5,0.7,0))
```

Itselleen sopivat parametrit löytyvät kokeilemalla. Uudessa grafiikkaikkunassa, joka avataan komennolla `X11()` on oletusarvoiset määritteiden (kuten `pin`, `mfrow`) arvot.

Monipuolisempia kuvakokoelmia (esim. yksi iso kuva ja kaksi pientä samassa ikkunassa) voidaan luoda `layout`-funktiolla, ks. `?layout`.

7.3 Grafiikkatoimintoja

Grafiikkatoiminnot voidaan jakaa kahteen luokkaan. Ylemmän tason grafiikkatoiminnot (`plot`, `hist`, `image`) tekevät aina uuden kuvaajan, kun taas alemman tason grafiikkatoiminnot (`points`, `lines`, `abline`) lisäävät jo olemassaolevaan kuvaan joitain elementtejä.

Esimerkeissä käytetään datoja `cars` ja `quakes`. Datat ovat R:n `datasets`-paketissa, joka ladataan R:ään automaattisesti R:n käynnistyessä. Listan R:ssä olevista datoista saa komennolla `data()`. Nämä ovat vapaasti käytettävissä.

- Data `cars` sisältää 50 havaintoa kahdesta muuttujasta `speed` ja `dist`, jotka ovat auton nopeus (mph, mailia tunnissa) ja pysähtymismatka (ft, jalkoja). Huomaa, että data on 1920-luvulta, ks. `?cars`.
- Data `quakes` sisältää 1000 havaintoa maanjäristyksiin liittyvistä 5 muuttujasta: leveysaste (`lat`), pituusaste (`long`), syvyys (km, `depth`), suuruus (Richter, `mag`) ja kuinka monta asemaa raportoi järistyksestä (`stations`), ks. `?quakes`.

7.3.1 plot-funktio

Funktiolla `plot` voi piirtää hajontakuvioita ja viivagraafeja. Funktio osaa tunnistaa sille parametrina annetun objektin tyypin.

1. `> plot(cars$dist, xlab="Indeksi", ylab="Pysähtymismatka")`

   tulostaa vektorin `cars$dist` arvot indeksin funktiona (kuva 7). Määritteissä `xlab` ja `ylab` voidaan antaa nimet x- ja y-akseleille. Annettaessa kaksi muuttujaa, kuten

   ```
   > plot(cars$speed, cars$dist, xlab="Nopeus", ylab="Pysähtymismatka")
   ```

   tulee kuvaajasta näiden kahden muuttujan hajontakuvio. Voidaan antaa myös komento

   ```
   > plot(cars, pch=16, main="Hajontakuvio")
   ```

   jolloin piirretään objektin `cars` komponenttien hajontakuviot (kuva 7). Argumentti `pch` määrittää piirrettävän pisteen tyypin (ks. `?plot.default, ?points`) ja `main` on otsikko kuvalle. Jos parametrina annetaan datakehikko (tai lista), jossa on useampia muuttujia, niin piirretään hajontakuviot kaikkien näiden muuttujien välille.

2. Funktiolla `plot` voidaan myös piirtää viivagraafeja. Antamalla määrite `type="l"` pisteiden sijaan piirretään viiva (joka yhdistää pisteet). Esim.

   ```
   > x <- seq(-3,3,by=0.1)
   > y <- x^2
   > plot(x,y,type="l")
   ```

   Myös muita viivatyyppejä on, ks. `?plot`. Määritteellä `lty` voidaan määrätä viivan tyyppi ja määritteellä `lwd` viivan paksuus.

3. Jos parametrina annetaan `lm`-objekti, `plot` tuottaa useita kuvia.

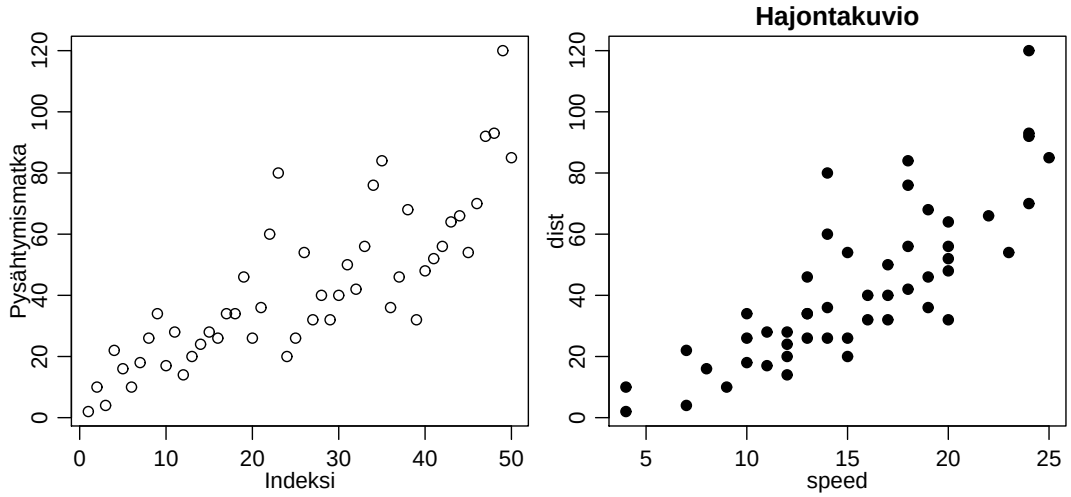
   ```
   > cars.lm <- lm(dist ~ speed, data=cars)
   > plot(cars.lm)
   ```

4. Jos annetaan parametrina aikasarja, kuten data `Nile`, niin `plot` piirtää aikasarjan. (On myös olemassa funktio `ts.plot`, jonka avulla voidaan helposti piirtää useampia aikasarjoja samaan kuvaan.)

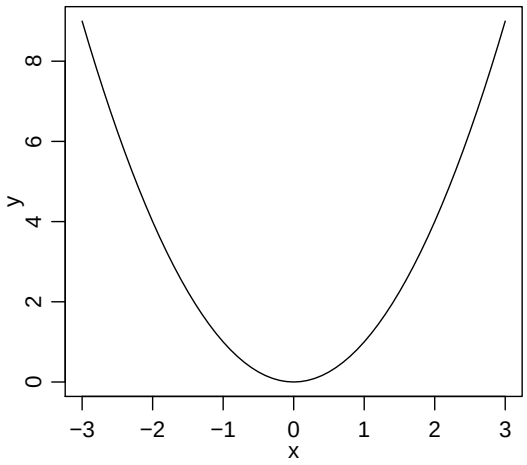
   ```
   > Nile
   Time Series: Start = 1871 End = 1970 Frequency = 1
     [1] 1120 1160  963 1210 1160 1160  813 1230 1370 1140  995  935 1110  994 1020
    [16]  960 1180  799  958 1140 1100 1210 1150 1250 1260 1220 1030 1100  774  840
    [31]  874  694  940  833  701  916  692 1020 1050  969  831  726  456  824  702
    [46] 1120 1100  832  764  821  768  845  864  862  698  845  744  796 1040  759
    [61]  781  865  845  944  984  897  822 1010  771  676  649  846  812  742  801
    [76] 1040  860  874  848  890  744  749  838 1050  918  986  797  923  975  815
    [91] 1020  906  901 1170  912  746  919  718  714  740
   > plot(Nile)
   ```

Funktiolle `plot` voidaan antaa myös monia muita parametreja kuin edellä mainitut (`xlab, ylab, main, pch, type, lty, lwd`), ks. `?plot.default`. Esimerkiksi `col` määrää kuvaajan värin.

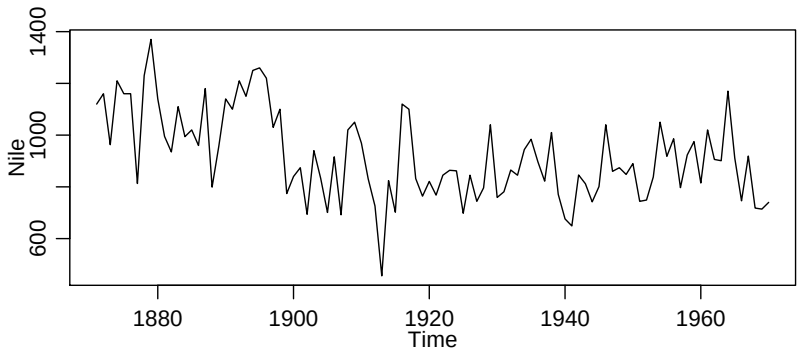
Huom. Funktio `plot` on geneerinen funktio objektin piirtämiseksi. Komento `plot(objekti)` piirtää kyseisen objektin oletusarvojen mukaisesti. Itse asiassa tietyn objektin piirtämiseen käytettävä funktio on `plot.objekti`, mutta nimen loppuosa on pääasiassa piilotettu käyttäjältä, eikä siitä tarvitse välittää. Kuitenkin ohjesivustoja kannattaa tarkastella. R:ssä olevat `plot`-funktiot saa tulostettua komennolla `methods(plot)` (samoin muille geneerisille funktioille).



Kuva 7: Komentojen `plot(cars$dist, xlab="Indeksi", ylab="Pysähtymismatka")` ja `plot(cars, pch=16, main="Hajontakuvio")` tuottamat kuvat.



Kuva 8: Komennon `plot(x,y,type="l")` tuottama kuva.



Kuva 9: Komennon `plot(Nile)` tuottama kuva.

7.3.2 Elementtien lisääminen kuvaan: points(), lines(), abline()

Funktioiden `points` ja `lines` avulla voidaan piirtää useampia kuvaajia samaan kuvaan. Näillä komennoilla annetut kuvat lisätään jo olemassa olevan kuvan päälle. Siis aiempi komento (esim. `plot`) määrää kuvan tyylin, esim. x- ja y-akselin rajat (voidaan säätää parametreilla `xlim`, `ylim`) ja nimet (`xlab`, `ylab`).

Esim. 1. points()

```
> x <- seq(-3,3,by=0.1)
> y <- x^2
> z <- x^3
> plot(x,y,type="l",ylim=c(-10,10))
> points(x,z)
```

Esim. 2. lines()

```
> plot(x,y,type="l",ylim=c(-10,10))
> lines(x,z) # on sama kuin points(x,z,type="l")
```

Esim. 3. abline()

Suorien piirtäminen:

```
>  x <- seq(-3,3,by=0.1)
>  y <- x^2
>  z <- x^3
>  plot(x,y,type="l")
>  abline(h=2)
>  abline(v=-2, lty=2)
>  abline(a=3, b=1, lwd=2)
```

Regressiosuoran lisääminen hajontakuvioon:

```
> cars.lm <- lm(dist ~ speed, data=cars)
> plot(cars)
> abline(cars.lm)
```

7.3.3 curve-funktio

Kätevä funktio käyrien piirtämisessä on **curve**. Esim.

```
> x <- seq(-3,3,by=0.1)
> y <- x^2
> plot(x,y,type="l")
```

voidaan vaihtoehtoisesti piirtää komennolla

```
> curve(x^2,from=-3,to=3,ylab="y")
```

Ensimmäisenä parametrina annetaan piirrettävä funktio tai lauseke argumenttina x.

7.3.4 Jakaumien visualisointi

Funktiolla `hist` voidaan piirtää histogrammi, joka approksimoi jakauman tiheysfunktiota.

```
> X11()
> par(mfrow=c(2,2))
> hist(quakes$depth, main="Histogrammi 1, oletusasetukset", xlab="syvyys")
```

Ylläolevassa perusmuodossaan `hist` pyrkii määrittelemään sopivan määrän histogrammin pylväslokeroita. Käyttäjä voi kuitenkin määrätä lokerot määritteellä `breaks`. Esim.

```
> hist(quakes$depth, 30, main="Histogrammi 2", xlab="syvyys")
> hist(quakes$depth, breaks=seq(min(quakes$depth),max(quakes$depth),length=30),
+  main="Histogrammi 3", xlab="syvyys")
```

Histogrammi voidaan skaalata 'tiheysfunktioksi' antamalla määrite `probability=TRUE`:

```
> hist(quakes$depth, breaks=seq(min(quakes$depth),max(quakes$depth),length=30),
+      probability=T)
```

Funktiolla `density` voidaan laskea tiheysfunktion ydinestimaatti, joka on muotoa

$$\hat{f}(x) = \frac{1}{n}\sum_{1}^{n} k_h(x_i - x), \tag{1}$$

missä $k_h(\cdot)$ on ydinfunktio (symmetrinen, ei-negatiivinen ja $\int k_h(udu) = 1$), $h > 0$ on ytimen leveys, joka määrää tasoituksen asteen. Oletusarvoisesti `density` käyttää Gaussista ydintä ja 'sopivasti valittua' ytimen leveyttä, ks. `?density`.

```
> plot(density(quakes$depth))
```

Nämä samaan kuvaan:

```
> hist(quakes$depth, breaks=seq(min(quakes$depth),max(quakes$depth),length=30),
+      probability=T)
> lines(density(quakes$depth))
```

Datasta voidaan myös laskea empiirinen kertymäfunktio. Piirretään kertymäfunktio:

```
> plot(ecdf(quakes$depth),do.points=FALSE,verticals=T,
+      main="empiirinen kertymäfunktio")
```

Viiksilaatikkodiagrammi voidaan piirtää seuraavasti:

```
> boxplot(quakes$depth, main="Box-plot")
```

7.3.5 Kahden muuttujan funktioiden visualisointi

Harmaasävykuva:

```
> x <- y <- seq(-2,2,by=0.1)
> z <- exp( - outer(x^2, y^2, "+") )
> image(x, y, z, main=expression(paste("Funktio ",f(x),"=",exp(-(x^2+y^2)))),asp=1)
```

Tasa-arvokäyrät:

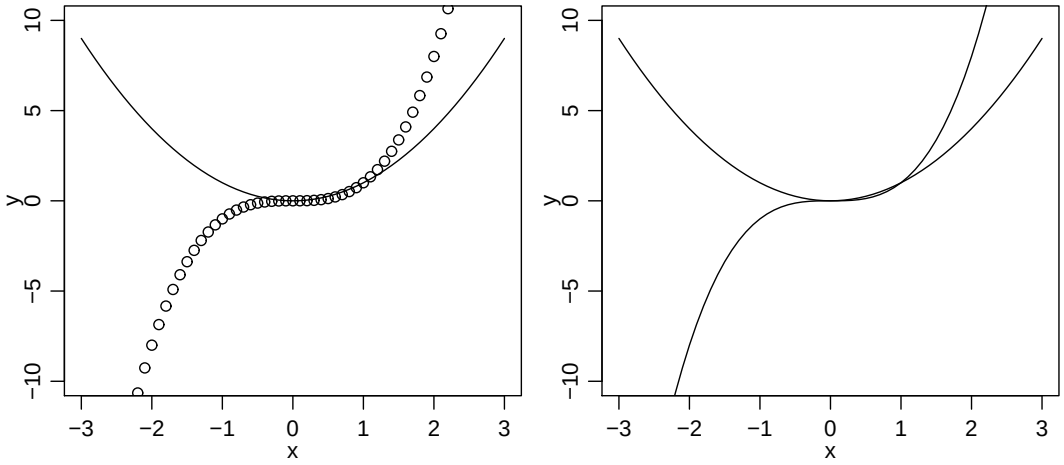
```
> contour(x, y, z, main="tasa-arvokäyrät", asp=1)
```

Funktiopinta:

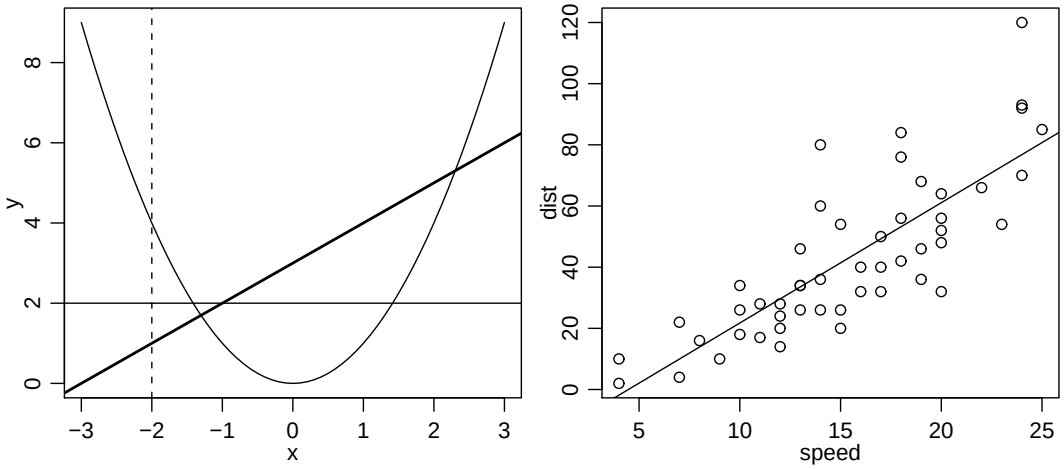
```
> persp(x, y, z, main="funktiopinta", asp=1)
```

Harmaasävykuvaan voidaan myös lisätä pisteitä `points`-funktiolla. Esim.

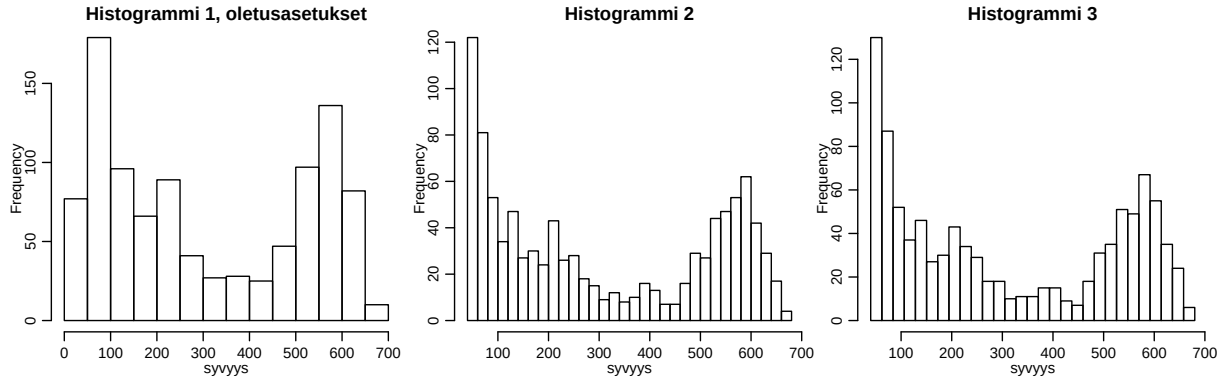
```
> u <- runif(20,-2,2)
> v <- runif(20,-2,2)
> image(x, y, z, main=expression(paste("Funktio ",f(x),"=",exp(-(x^2+y^2)),"
+       ja pisteet")), asp=1)
> points(u,v)
```

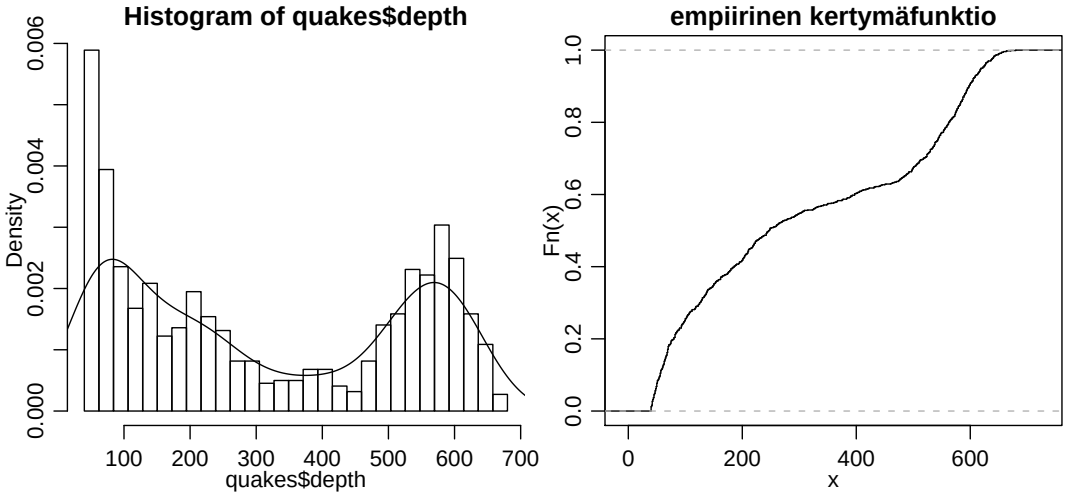


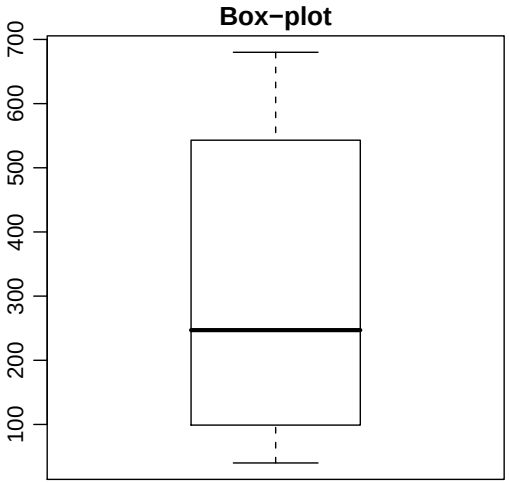
Kuva 10: points() ja lines() -funktioiden käyttö.

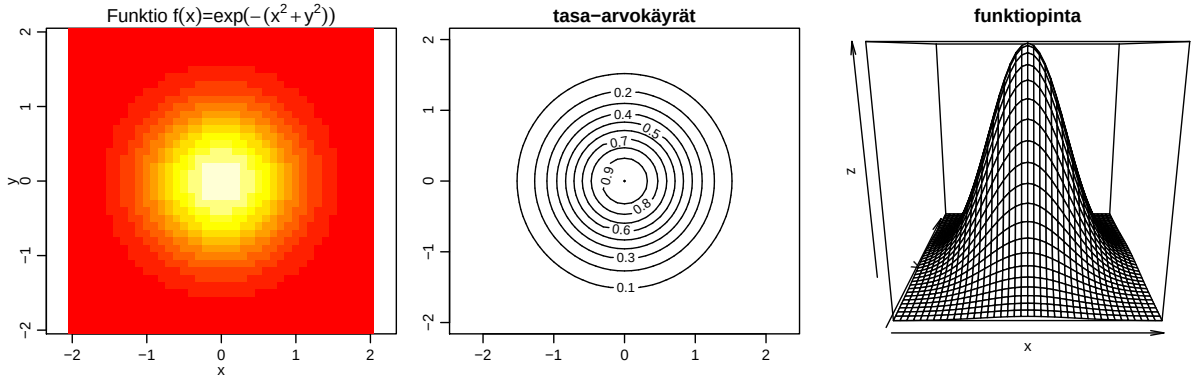


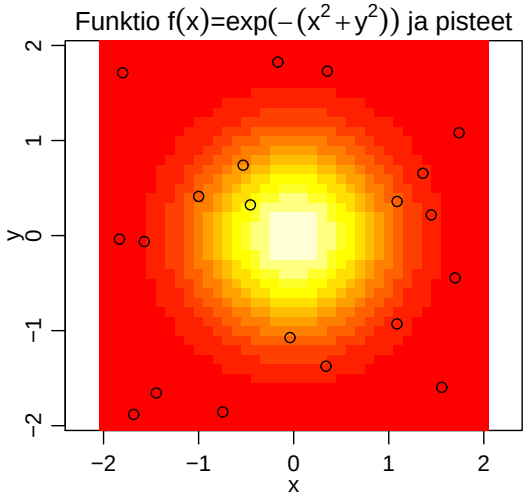
Kuva 11: Suorien lisääminen kuviin.


Histogrammi 1, oletusasetukset
Frequency
syvyys
Histogrammi 2
Frequency
syvyys
Histogrammi 3
Frequency
syvyys


Histogram of quakes$depth
Density
quakes$depth
empiirinen kertymäfunktio
Fn(x)
x


Box−plot





Lista funktioista, joita käytettiin

`X11()`	avaa R Graphics -ikkunan
`postscript()`	tulostus .eps-tiedostoon
`par()`	kuvakoon ja tyypin määrittely
`data()`	R:ssä olevat datat
`plot()`	hajontakuvio, viivagraafi yms.
`lm()`	lineaarisen mallin sovitus
`points()`	pisteiden lisääminen kuvioon
`lines()`	käyrän lisääminen kuvioon
`abline()`	suoran piirtäminen
`hist()`	histogrammi
`curve()`	käyrän piirtäminen
`density()`	tiheys-estimaatti kernel-menetelmällä
`ecdf()`	empiirinen kertymäfunktio
`boxplot()`	viiksilaatikkokuvion piirtäminen
`outer()`	ulkotulo
`image()`	harmaasävykuva
`contour()`	tasa-arvokäyrien piirtäminen
`persp()`	funktiopinnan piirtäminen

8 Jakaumat

R:stä löytyvät funktiot useimpien yleisesti käytettyjen jakaumien tarkasteluun. Esimerkiksi normaalijakaumalle on neljä funktiota:

```
dnorm(x, mean=0, sd=1, log = FALSE)
pnorm(q, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean=0, sd=1)
```

joista

`dnorm` laskee tiheysfunktion arvon argumentin `x` arvoilla

`pnorm` laskee kertymäfunktion arvon argumentin `q` arvoilla

`qnorm` laskee kvantiilin todennäköisyyden `p` arvoilla, $0 \leq$ `p` ≤ 1

`rnorm` simuloi jakaumasta `n` riippumatonta realisaatiota

Tarkastellaan näitä funktioita esimerkkien kautta:

1. **Tiheysfunktio.** Jakauman $N(2, 0.5)$ piirtäminen, missä 2 ja 0.5 ovat jakauman keskiarvo ja varianssi. Huomaa, että R-funktioille `dnorm`, `pnorm`, `qnorm` ja `rnorm` annetaan jakauman parametrit keskiarvo `mean` ja keskihajonta `sd` (ei varianssi).

   ```
   > curve(dnorm(x, mean=2, sd=sqrt(0.5)), from=-1, to=5, ylab="f(x)",
   +       main="Normaalijakauman N(2,0.5) tiheysfunktio")
   > # Kuvaajan piirtäminen tiedostoon:
   > postscript(file="dnorm_mean2_var05.eps",height=4,width=4,
   +            paper="special",horizontal=F)
   > par(mfrow=c(1,1),mar=c(4,2.5,1.5,0.5),mgp=c(1.5,0.7,0))
   > curve(dnorm(x, mean=2, sd=sqrt(0.5)), from=-1, to=5, ylab="f(x)",
   +       main="Normaalijakauman N(2,0.5) tiheysfunktio")
   > dev.off()
   ```

2. **Todennäköisyyksien laskeminen.** Mitä on $P(0 < X < 2)$, kun $X \sim N(2, 0.5)$? Entä $P(X > 4)$?

   ```
   > pnorm(2,2,sqrt(0.5))-pnorm(0,2,sqrt(0.5))
   [1] 0.4976611
   > 1-pnorm(4,2,sqrt(0.5))
   [1] 0.002338867
   ```

3. **Kvantiilien laskeminen.** Millä z on $P(Y > z) = 0.025$, kun $Y \sim N(0, 1)$? Mitkä ovat ala- ja yläkvartiilit ja mediaani?

   ```
   > qnorm(0.975)
   [1] 1.959964
   > # TAI
   ```

```
> qnorm(0.025,lower.tail = FALSE)
[1] 1.959964
> qnorm(c(0.25,0.5,0.75))
[1] -0.6744898  0.0000000  0.6744898
```

Määritteellä `lower.tail` voidaan määrätä lasketaanko todennäköisyyksiä $P(X \leq x)$ (`lower.tail=TRUE` oletus) vai todennäköisyyksiä $P(X > x)$ (`lower.tail=FALSE`).

4. **Simulointi ja jakaumien vertailu.** a) Lasketaan $P(X > 2.5)$ simuloimalla, kun $X \sim N(2, 0.5)$. Tehdään 5000 simulointia.

```
> x <- rnorm(5000,2,sqrt(0.5))
> length(x[x>2.5])/length(x)
[1] 0.2386
```

b) Simuloidaan 100 realisaatiota normaalijakaumasta $N(2, 0.5)$ ja piirretään niiden histogrammi.

```
> x <- rnorm(100,2,sqrt(0.5))
> hist(x,breaks=seq(min(x),max(x),length=10))
> # Kuvaajan piirtäminen tiedostoon:
> postscript(file="rnorm100_mean2_var05.eps",height=4,width=4,
+            paper="special",horizontal=F)
> par(mfrow=c(1,1),mar=c(4,2.5,1.5,0.5),mgp=c(1.5,0.7,0))
> hist(x,breaks=seq(min(x),max(x),length=10))
> dev.off()
```

Empiirisiä kvantiileja voidaan verrata jakauman $N(2, 0.5)$ kvantiileihin:

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1296  1.6500  2.0020  2.0630  2.5520  4.2080
>  qnorm(c(0.25,0.5,0.75),2,sqrt(0.5))
[1] 1.523064 2.000000 2.476936
```

Piirretään datasta lasketut (skaalattu) histogrammi ja tiheysfunktio sekä kertymäfunktio:

```
> hist(x,breaks=seq(min(x),max(x),length=10),prob=T)
> lines(density(x))
> plot(ecdf(x),do.points=FALSE,verticals=T,main="empiirinen kertymäfunktio")
```

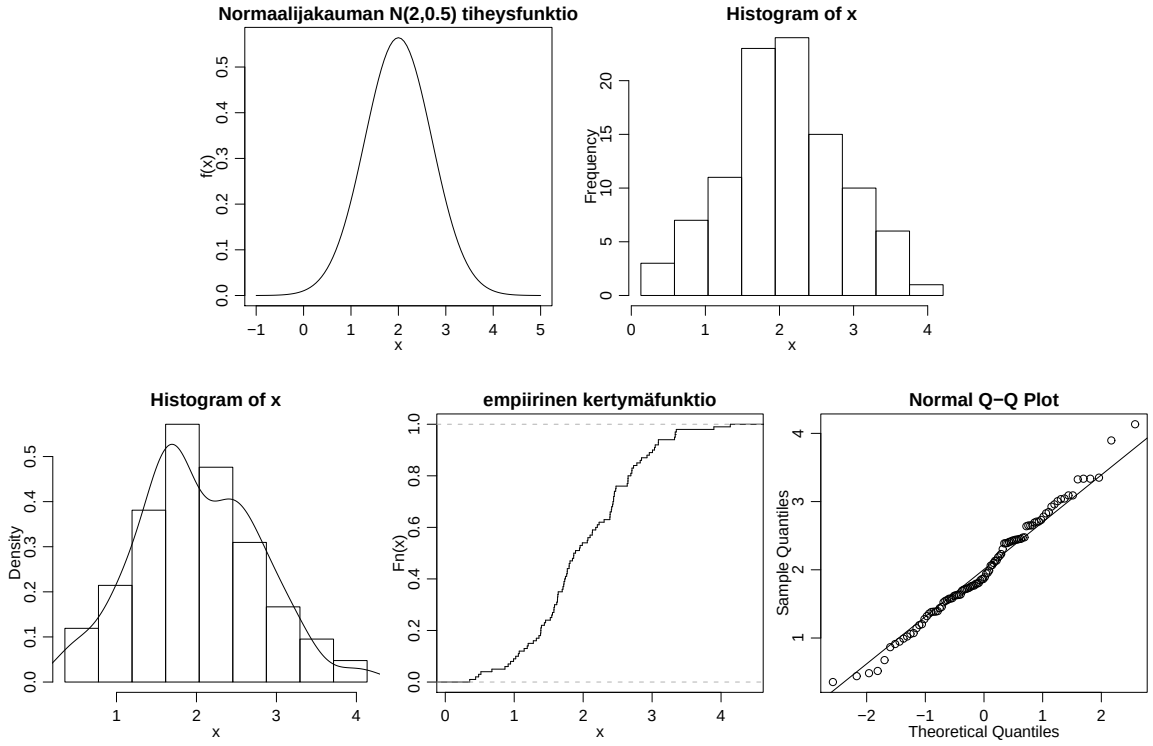
Yleinen tapa verrata kahta kahta otosta on käyttää Q-Q-hajontakuviota. Kvantiilifunktio määritellään

$$q(p) = \min(z|\ P(X \leq z) = p)$$

missä $P(X \leq z)$ lasketaan datasta. Jos $x = (x_1, x_2, \ldots, x_n)$ ja $y = (y_1, y_2, \ldots, y_n)$ ovat kaksi otosta, niin `qqplot(x,y),...` piirtää näiden kahden otoksen kvantiilifunktiot toisiaan vastaan. Jos halutaan verrata otosta normaalijakaumaan, se voidaan tehdä funktiolla `qqnorm`.

```
> qqnorm(x)
> qqline(x)
```

Funktio `qqline` lisää kuvaajaan suoran, joka kulkee ala- ja yläkvartiilin kautta.



Normaalijakauman lisäksi R:stä löytyvät ainakin seuraavat jakaumat:

Lista funktioista, joita käytettiin	
`dbeta()`	Beta-jakauma
`dbinom()`	Binomijakauma
`dcauchy()`	Cauchy-jakauma
`dchisq()`	χ^2-jakauma
`dexp`	Eksponenttijakauma
`df()`	F-jakauma
`dgamma()`	Gamma-jakauma
`dgeom()`	Geometrinen jakauma
`dhyper()`	Hypergeometrinen jakauma
`dlnorm()`	Log-normaalijakauma
`dlogis()`	Logistinen jakauma
`dnbinom()`	Negatiivinen binomijakauma
`dpois()`	Poisson-jakauma
`dt()`	t-jakauma
`dunif()`	Tasajakauma
`dweibull()`	Weibull-jakauma
`dwilcox()`	Wilcoxonin jakauma

Kaikille näille jakaumille on 4 funktiota kuten normaalijakaumallekin. Muut funktiot saadaan korvaamalla `d` kirjaimella `p, q` tai `r`. Jakaumien parametrit löytyvät R:n help-sivuilta.

Satunnaislukujen simulointi

Satunnaislukuja välillä $[a, b]$ voidaan simuloida funktiolla

```
runif(n, a, b)
```

missä `n` on simuloitavien satunnaislukujen määrä.

Satunnaisotos joukosta x (esim. $\{1, 2, \ldots, k\}$) voidaan poimia funktiolla

```
sample(x, size, replace = FALSE)
```

missä `size` määrää otoksen koon ja `replace` kertoo, onko otanta ilman takaisinsijoitusta (`replace=FALSE`, oletus) vai takaisinsijoittaen (`replace=TRUE`).

Esim.

```
> runif(5)
[1] 0.15565081 0.76423530 0.09210198 0.64080668 0.84525309
> runif(5, 1, 10)
[1] 4.048511 1.021403 1.016657 7.293052 9.802818
> sample(1:5, size=5, replace=T)
[1] 5 3 3 5 5
> sample(1:5, size=5, replace=F)
[1] 4 5 2 3 1
```

Lista funktioista, joita käytettiin

`dnorm()`	normaalijakauman tiheysfunktio
`pnorm()`	normaalijakauman kertymäfunktio
`qnorm()`	normaalijakauman kvantiilifunktio
`rnorm()`	normaalijakauman simulointi
`summary()`	tiivistelmä
`qqnorm(), qqplot()`	Q-Q-hajotelmakuvio
`qqline()`	suoran lisääminen qnorm()-kuvaajaan
`ecdf()`	empiirinen kertymäfunktio
`runif()`	satunnaislukujen generointi
`sample()`	otos jostakin joukosta

9 Omien funktioiden kirjoittaminen

Toistettavasta koodijonosta saattaa olla hyödyllistä tehdä oma funktionsa. Funktio tarvitsee kirjoittaa vain kertaalleen (ja ehkä parannella myöhemmin), minkä jälkeen toiminto on helposti toistettavissa.

R:ssä voidaan luoda omia funktioita ja se tapahtuu seuraavasti

```
nimi <- function(param1, param2, ... ) {
    R-komennot
}
```

Funktio toteuttaa R-komennot käyttäen parametreja `param1` , `param2`, ... Kun omatekoinen funktio on ladattu R:ään (`source`-komennolla), sitä voidaan käyttää kuten mitä tahansa R:n valmiita funktiota. Funktiokutsu on muotoa

```
nimi(param1, param2, ...)
```

missä parametreille annetaan jotkin arvot. Tämä kutsu voidaan tehdä myös toisten funktioiden sisällä.

Esimerkiksi vektorista neliösumman laskeva funktio.

```
> neliosum <- function(x) {
+    sum(x^2)
+ }
> a <- c(2,5,8,11,14,17)
> neliosum(a)
[1] 699
```

Toisena esimerkkinä momentit laskeva funktio (data `y` luvusta 1.6):

```
> moments <- function(x, K)  # funktion luonti
+ {
+ # laskee K ensimmäistä momenttia datavektorista x
+     m <- NULL
+     for(k in 1:K) {
+         m[k] <- mean( x^k )
+     }
+     m
+ }
> moments                     # funktion listaus
function(x, K)
{
# laskee K ensimmäistä momenttia datavektorista x
    m <- NULL
    for(k in 1:K) {
        m[k] <- mean( x^k )
```

```
    }
    m
}
> moments(x=y, K=4)  # funktion kutsu, tai lyhyesti moments(y, 4)
[1]   2.019352  13.131455  63.223633 456.974915  # Funktion palauttama tulos
```

Funktiot kannattaa kirjoittaa tekstieditorissa sen sijaan, että kirjoittaisi suoraan R-konsoliin. Funktio `moments`, joka on määritelty tiedostossa `moments.r` seuraavasti

```
moments <- function(x, K=3)
{
# laskee K ensimmäistä momenttia datavektorista x
    m <- NULL
    for(k in 1:K) {
        m[k] <- mean( x^k )
    }
    m
}
```

saadaan luettua R:ään komennolla

```
> source("moments.r")
```

Tämän jälkeen funktio `moments` on käytettävissä.

Huom. Funktioille kannattaa antaa nimiä, joista selviää, mitä ne tekevät.

9.1 Kommentteja funktioiden käytöstä

1) Funktiot palauttavat viimeisen komennon tuloksen, joka tyypillisesti on jotain, mikä on laskettu funktiossa. (Funktiossa tapahtuvat piirrot ja tulostukset, esim. print, sen sijaan piirtyvät/tulostuvat R:ään.) Siksi esimerkiksi funktio

```
tunnusluvut <- function(x) {
# laskee x:n vaihteluvälin, keskiarvon ja keskihajonnan
    range(x)
    mean(x)
    sd(x)
}
```

ei palauta kaikkia laskettuja suureita vaan vain viimeisen, keskihajonnan:

```
> tunnusluvut(pigs$d)
[1] 5.482252
```

Sen sijaan funktio

```
tunnusluvut <- function(x) {
# laskee x:n vaihteluvälin, keskiarvon ja keskihajonnan
    list(vaihteluväli=range(x),ka=mean(x),sd=sd(x))
}
```

palauttaa listan tunnusluvuista vaihteluväli, keskiarvo ja keskihajonta:

```
> tunnusluvut(pigs$d)
$vaihteluväli
[1] 10.3 36.2

$ka
[1] 23.00167

$sd
[1] 5.482252
```

2) Funktion parametreille voi antaa oletusarvoja, joita käytetään, jos parametrille ei anneta arvoa funktiokutsussa. Parametrille, jolle on annettu oletusarvo argumenttilistassa, ei siis tarvitse välttämättä antaa arvoa funktion kutsussa. Esim. jos funktio `moments` on määritelty seuraavasti

```
moments <- function(x, K=3)
{
# laskee K ensimmäistä momenttia datavektorista x
    m <- NULL
    for(k in 1:K) {
       m[k] <- mean( x^k )
    }
    m
}
```

niin tällöin funktio laskee `K=3` ensimmäistä momenttia parametrina annetusta datavektorista, jos parametria `K` ei anneta funktiokutsussa:

```
> moments(y)
[1]  2.019352 13.131455 63.223633
```

Jos parametri `K` annetaan funktiokutsussa, niin tällöin funktio käyttää tätä arvoa:

```
> moments(y,2)
[1]  2.019352 13.131455
```

3) Funktion parametrit ovat 'paikallisia nimiä', jotka ovat käytössä ainoastaan funktion sisällä. Funktion sisällä tehtävä muutos ei vaikuta funktion ulkopuolella olevan mahdollisesti samannimisen objektin sisältöön. Esim. funktio, joka kasvattaa parametrina annettavaa lukua (vektoria) `n`:llä.

```
> addn <- function(x, n=1) { x + n }
> x <- 5
> addn(x)
[1] 6
> x
[1] 5
```

4) Jos funktiossa käsitellään objektia nimeltä `x` (esim), joka ei ole määritelty funktion argumenttilistassa, funktio etsii objektia `x` työpöydältä. Jos `x` on olemassa työpöydällä, niin funktio käyttää tätä `x`:ää operaatioissaan. Tällainen käyttö on vaarallista. Tarkkuutta vaaditaan! Objektit, joita käsitellään funktiossa, kannattaa aina antaa funktiolle parametreina. Alla oleva on esimerkki huonosta funktiosta. Se ei ole yleiskäyttöinen.

```
> x
[1] 5
> add1 <- function() { x+1 }
> add1()
[1] 6
> x
[1] 5
```

9.2 Komentoryhmät, ehtolauseet ja silmukkarakenteet

Mielivaltainen määrä R:n komentoja voidaan koota yhdeksi laajemmaksi komennoksi kirjoittamalla nämä kaarisulkujen sisälle:

```
{
    komento1
    komento2
    komento3
    ...
    komentoN
}
```

Komennot voidaan erottaa toisistaan myös puolipisteellä, joten edellinen voidaan kirjoittaa vaihtoehtoisesti muodossa:

```
{
    komento1; komento2; komento3; ... komentoN;
}
```

Koska tällainen ryhmä komentoja on itsekin komento, se voi olla toisen komentoryhmän osa ja siten muodostamassa vielä isompaa kokonaisuutta.

Ehdollinen lauseke

R:n kontrollilauseet ovat samantapaisia kuin C-kielessä. Ehdollisen lausekkeen muoto on

```
if( ehto )
    komento1
else
    komento2
```

Ehdon `ehto` tulee palauttaa looginen arvo, joko `TRUE` tai `FALSE`. Jos ehto on tosi (`TRUE`), niin silloin suoritetaan komennot `komento1`. Jos ehto on epätosi (`FALSE`), niin suoritetaan `komento2`. Osio `else` ei ole pakollinen.

Esim. Tiedosto maksimi.r

```
maksimi <- function(luku1, luku2) {
    if(luku1 >= luku2)
        maks <- luku1
    else
        maks <- luku2
    maks
}
```

R:ssä:

```
> source("maksimi.r")
> maksimi(1,2)
[1] 2
```

Huom. R:ssä on funktio `max` maksimin etsimiseen ja siten yllä oleva esimerkkifunktio on turha.

Yksinkertaisempi ehtolauseke on toteutettu R:ssä funktiolla `ifelse`. Funktion parametrit määräytyvät seuraavasti:

```
ifelse( vektoria koskeva ehto, kyllä, ei )
```

Ne vektorit elementit, jotka toteuttavat ehdon, saavat arvokseen kyllä-muuttujan arvon. Vastaavasti ne, jotka eivät toteuta ehtoa, saavat arvokseen ei-muuttujan arvon. Esimerkiksi

```
> x <- c(0,1,1,0,1)
> ifelse(x > 0,3,-1)
[1] -1  3  3 -1  3
```

eli ne x:n arvot, jotka ovat > 0, saavat arvokseen 3. Muuten ne saavat arvon -1. Funktio `ifelse` on siis eräänlainen indikaattorifunktio.

Silmukat

Tyypillinen silmukka on

```
for(muuttuja in vektori) komento
```

Tässä esiintyvä `muuttuja` on ns. silmukkaindeksi, joka käy läpi kaikki `vektori`:ssa esiintyvät arvot. Silmukkaa siis suoritetaan yhtä monta kertaa kuin `vektori`:ssa on alkioita. Jokaisella silmukan kierroksella suoritetaan `komento`, joka käyttää sen hetkistä `muuttuja`:n arvoa. Useimmiten `for`-silmukka on muotoa:

```
for(i in 1:n) komento
```

R-kielen `while`-silmukka on muotoa

```
while( ehto ) komento
```

R-komentoa `komento` toistetaan niin kauan kuin `ehto` saa arvon `TRUE`. Jotta silmukka voisi päättyä, täytyy R-komennon muuttaa ehdossa esiintyvien muuttujien (objektien) arvoja siten, että `ehto` jossain vaiheessa saa arvon `FALSE`.

Silmukoita kannattaa välttää R-koodissa! Aina se ei ole kuitenkaan mahdollista.

9.3 Uuden objektityypin luominen

R:n perusobjekteja on käsitelty kappaleessa 3. Monet R-funktiot kuitenkin palauttavat objekteja, jotka ovat muuta muotoa, ja näille objekteille on myös määritelty omia toimintoja. Seuraavassa lyhyt yksinkertainen esimerkki oman objektiluokan luomisesta.

Oletetaan, että tarkastelemme usein kahden muuttujan välistä riippuvuutta. Voimme 'niputtaa' yhteen keskiarvojen, keskihajontojen ja kovarianssin laskemisen: muodostetaan funktio

```
xy.hajonta <- function(x, y) {
   dk <- data.frame(x, y)
   res <- list(x = x, y = y,
               keskiarvot = mean(dk),
               keskihajonnat = var(dk),
               korrelaatio = cor(x,y),
               call = match.call())
   oldClass(res) <- "xy.hajonta"
   res
}
```

Tämä funktio suorittaa laskutoimitukset ja palauttaa tulokset listana. Listaan on lisäksi laitettu vektorit `x` ja `y` myöhempiä toimintoja varten. Komponentti `call = match.call()` sisältää funktionkutsun. Funktion toiseksi alimmalla rivillä annetaan objektityypille nimi.

Luetaan funktio R:ään ja käytetään sitä normaalijakautuneisiin muuttujiin, jotka simuloimme seuraavassa:

```
> x <- rnorm(100, 0, 1)
> y <- rnorm(100, 5 + 2*x, 2)
> tul <- xy.hajonta(x, y)
```

Jos nyt tulostamme objektin `tul` sisällön, saamme pitkän tulostuksen. Itse asiassa kiinnostavia tulostettavia ovat keskiarvot, keskihajonnat ja korrelaatio, ei niinkään `x`:n ja `y`:n arvot. Voimme määritellä, miten `print`-funktio toimii, kun sille annetaan argumenttina `xy.hajonta`-objekti. Luodaan funktio nimeltä `print.xy.hajonta`:

```
print.xy.hajonta <- function(x, ...) {
   with(x, cat(" Keskiarvot ja keskihajonnat \n",
               "x:", keskiarvot[1], ",", keskihajonnat[1], "\n",
               "y:", keskiarvot[2], ",", keskihajonnat[2], "\n",
               "Korrelaatio \n",
               korrelaatio, "\n"))
   invisible(x)
}
```

Funktion määrittelyssä komponentti `x` on objekti, joka on tyyppiä `xy.hajonta`, joten se sisältää komponentit `keskiarvot`, `keskihajonnat` ja `korrelaatio`. Ladataan funktio R:ään ja tulostetaan objekti `tul`:

```
> tul
 Keskiarvot ja keskihajonnat
 x: -0.01442702 , 1.188969
 y: 4.732519 , 2.709823
 Korrelaatio
 0.7626715
```

Lisäksi voimme määritellä esimerkiksi, miten `xy.hajonta`-objekti oletusarvoisesti piirretään:

```
plot.xy.hajonta <- function(x, ...) {
   with(x, plot(x, y, main=paste("sd(x)=", round(keskihajonnat[1],3),
            ", sd(y)=", round(keskihajonnat[2],3), ",
            cor(x,y)=", round(korrelaatio,3)), ...))
}
```

Kun tämä funktio on ladattu R:ään, komento `plot(tul)` piirtää `x`:n ja `y`:n hajontakuvion siten, että kuvan otsikossa on keskihajonnat ja korrelaatio. Lisäksi `plot`-funktiolle voidaan antaa muita argumentteja kuten `col`, `pch`.

9.4 Debuggaus

Funktion kirjoittaminen ei aina ole aivan suoraviivaista. Omia funktioita kirjoittaessa, ja myös valmiita funktioita käytettäessä, voi törmätä erilaisiin varoitus- ja virheilmoituksiin, jotka kertovat virheellisestä toiminnasta. Debuggauksella tarkoitetaan ohjelmistotuotannossa tällaisen virheellisen toiminnan paikallistamista ja korjaamista. Debuggaus on usein pitkä ja vaativa prosessi, sillä virheet esiintyvät monesti vain harvinaisissa erikoistilanteissa, jolloin niiden paikallistaminen tai toistaminen on aikaavievää.

R:ssä on sisäänrakennettuna monia työkaluja debuggausta varten, joita ei tässä käsitellä kovin tarkasti. Sen sijaan keskitytään muutamaan yksinkertaiseen tapaan löytää ja korjata tavallisimpia virheitä.

Oletetaan, että keskiarvofunktio `mean` ei olisi käytettävissä, ja meidän tulisi ohjelmoida se itse. Tällöin funktio voisi näyttää seuraavalta

```
keskiarvo <- function(x) {
    s <- 0
    n <- length(x)
    for(i in 1:n) {
        s <- s + x[i]
    }
    s/n
}
```

Katsotaan mitä tapahtuu, jos funktiota yritetään soveltaa vektoriin, joka sisältää puuttuvaa tietoa.

```
> x <- c(1,2,NA)
> keskiarvo(x)
[1] NA
```

Funktio palauttaa `NA`, vaikka olisi haluttavaa, että funktio laskisi keskiarvon kaikista havaituista arvoista (alkuperäisessä `mean`-funktiossa on sama ongelma). Yksi tapa lähestyä ongelmaa on suorittaa funktion koodia rivi riviltä, ja tutkia missä kohtaa virhe tapahtuu. Selvästikään sijoitukset `s <- 0` ja `n <- length(x)` eivät ole syyllisiä virheelliseen toimintaan, joten virhe tapahtuu todennäköisesti silmukan sisällä, kun summaan `s` lisätään vektorin `x` arvoja. Kokeillaan luvun ja `NA`:n yhteenlaskua.

```
> 1 + NA
[1] NA
```

Virhe on löytynyt. Korjataan keskiarvofunktiota tallentamalla vektoriin `y` vektorin `x` havaitut arvot

```
keskiarvo <- function(x) {
    s <- 0
    y <- subset(x, !is.na(x))
    n <- length(y)
    for(i in 1:n) {
        s <- s + y[i]
    }
    s/n
}
```

Kokeillaan funktiota uudelleen.

```
> x <- c(1,2,NA)
> keskiarvo(x)
[1] 1.5
```

Nyt funktio toimii kuten haluttiin. R:ssä on funktio vastaavaa debuggausta varten nimeltään `browser`. Funktio pysäyttää funktion suorituksen kohdassa, jossa sitä kutsutaan, ja antaa käyttäjälle mahdollisuuden tarkastella työtilan senhetkisiä muuttujia. Edellisen esimerkin yhteydessä funktiota voitaisiin kutsua for-silmukan jokaisella kierroksella, jolloin `s` muuttujan arvoa voitaisiin tarkastella seuraavasti

```
keskiarvo <- function(x) {
    s <- 0
    n <- length(x)
    for(i in 1:n) {
        s <- s + x[i]
        browser()
    }
    s/n
}
```

`s` muuttujan arvon saa selville aivan kuten tavallisessakin R-ympäristössä. Funktion suoritusta voi jatkaa kirjoittamalla browseriin `c` tai `cont`.

```
> x <- c(1,2,NA)
> keskiarvo(x)
Called from: keskiarvo(x)
Browse[1]> s
[1] 1
Browse[1]> c
Called from: keskiarvo(x)
Browse[1]> s
[1] 3
Browse[1]> c
Called from: keskiarvo(x)
Browse[1]> s
[1] NA
Browse[1]> Q
>
```

Nähdään, että summamuuttujan `s` arvo muuttuu `NA`:ksi viimeisellä iteraatiokierroksella. Browserista voi poistua kirjoittamalla `Q`.

Toinen hyödyllinen työkalu on funktio `traceback`, jolla voi useimmissa tapauksissa selvittää virheilmoituksen aiheuttajan. Esimerkiksi

```
> a <- function(x) { b(x) }
> b <- function(x) { c(x) }
```

```
> c <- function(x) { x + "b" + d(x) }
> d <- function(x) { x+3 }
> x <- 1
> a(x)
Error in x + "b" : non-numeric argument to binary operator
```

Virhe syntyy, kun numeerista muuttujaa yritetään lisätä merkkitietomuuttujaan. Ohjelman antaman virheilmoituksen perusteella ei kuitenkaan voida päätellä, mikä funktioista `a`, `b`, `c` vai `d` aiheutti virheen, mutta `traceback` antaa tähän vastauksen.

```
> traceback()
3: c(x) at #1
2: b(x) at #1
1: a(x)
```

Funktion antamaa tulostusta luetaan hieman hämäävästi alhaalta ylöspäin. Aluksi siis kutsuttiin funktiota `a`, joka kutsui funktiota `b`, joka edelleen kutsui funktiota `c`. Nähdään, että suoritus pysähtyy funktioon `c`, joka on siis virheen aiheuttaja.

Edellisestä esimerkistä on myös hyvä huomioida se, että alkuperäinen vektorinmuodostusfunktio `c` korvattiin virheellisellä funktiolla. Jos nyt koitetaan muodostaa vektori tavalliseen tapaan esim. seuraavasti

```
> c <- function(x) { x + "b" + d(x) }
> y <- c(1,2,3)
Error in c(1, 2, 3) : unused arguments (2, 3)
```

niin R antaa virheilmoituksen käyttämättömistä argumenteista.

9.5 Poikkeusten käsittely

Monissa ohjelmointiprojekteissa on tapana yrittää huomioida kaikki mahdolliset poikkeavat tilanteet jo ohjelman suunnitteluvaiheessa. Mitä monimutkaisempi ohjelma, sen vaikeampi tällaisia tilanteita on ennakoida täydellisen kattavasti, jolloin on yleensä parempi varautua poikkeuksiin. Ohjelman kannalta poikkeuksellisessa tilanteessa, joka johtaisi ohjelman kaatumiseen tai ei-haluttuun toimintaan, voidaan ohjelman suoritus keskeyttää ja käyttää poikkeuksen käsittelijää tilanteen ratkaisemiseksi.

R:ssä poikkeuksia voidaan käsitellä monilla eri työkaluilla, mutta kenties yksinkertaisin tapa on käyttää `tryCatch`-funktiota. Funktion syntaksi on seuraava

```
tulos = tryCatch({
    suoritettava lauseke
}, warning = function(w) {
    varoituksen käsittelevä koodi
}, error = function(e) {
    virheen käsittelevä koodi
}, finally = {
    lopuksi suoritettava koodi
})
```

Tarkastellaan R:n logaritmifunktiota, joka antaa varoituksen jos sitä yritetään soveltaa negatiiviseen lukuun. Varoitus voidaan siepata ja korvata halutulla toiminnalla: palautetaankin käyttäjän antaman luvun vastaluvun logaritmi. Otetaan huomioon myös tilanne, jossa käyttäjä yrittää soveltaa logaritmia merkkitietomuuttujaan. Siepataan virhe, ja korvataan merkkitieto luvulla 10.

```
> x <-list(1,2,-3,"4")
> y <- c()
> for(i in 1:4) {
+     y[i] = tryCatch({
+         log(x[[i]])
+     }, warning = function(w) {
+         log(-x[[i]])
+     }, error = function(e) {
+         log(10)
+     })
+ }
> y
[1] 0.0000000 0.6931472 1.0986123 2.3025851
> log(c(1,2,3,10))
[1] 0.0000000 0.6931472 1.0986123 2.3025851
```

Nähdään, että poikkeukset tulivat käsiteltyä halutulla tavalla.

9.6 Säännölliset lausekkeet

Säännöllinen lauseke, josta monesti käytetään lyhenteitä regex tai regexp (engl. regular expression), määrittelee ryhmän merkkijonoja. Lausekkeiden avulla voidaan esimerkiksi tarkistaa merkkijonon oikeellisuus tai etsiä tietyntyyppisiä merkkijonoja. Säännöllisten lausekkeiden syntaksin kattavaa dokumentaatiota voi R:ssä tarkastella kirjoittamalla `?regex` tai `?regexp`. Säännöllisillä lausekkeilla operointiin liittyvien funktioiden dokumentaatio löytyy puolestaan kirjoittamalla `?grep`.

Oletetaan, että tehtävänä olisi tunnistaa sanoja, jotka koostuvat kirjaimista a-z ja jotka ovat 5-8 merkkiä pitkiä. Tämä on tietysti mahdollista toteuttaa myös silmukoiden avulla käymällä sanoja läpi kirjain kirjaimelta. Säännölliset lausekkeet tarjoavat tähän kuitenkin paljon elegantimman ratkaisun.

```
> kuvio <- "^[a-z]{5,8}$"
> sanat <- c("kissa","koira","työ","auto","porkkana","tilastotiede")
> grep(pattern = kuvio, x = sanat, value = TRUE)
[1] "kissa"    "koira"    "porkkana"
```

Funktio `grep` palauttaa vektorin `sanat` ne indeksit, jotka toteuttavat määritetyn kuvion. Optiolla `value = TRUE` palautetaan indeksien sijaan vastaavat elementit. Kuvion määrittää säännöllinen lauseke, joka on tämän esimerkin yhteydessä melko yksinkertainen. Sulkeet `[ ]` määrittävät ryhmän, johon tässä tapauksessa kuuluvat kirjaimet a:sta z:aan, mikä ilmaistaan yhdistämällä kirjaimet viivalla. Sulkeiden `{ }` sisällä määritetään, kuinka monta edeltävän ryhmän jäsentä merkkijonossa tulisi olla. Vähimmäismäärä erotetaan pilkulla enimmäismäärästä.

Merkkejä ^ ja $ kutsutaan vasemmaksi ja oikeaksi ankkuriksi. Tässä esimerkissä ne aloittavat ja lopettavat sanan.

Tarkastellaan seuraavaksi hieman monimutkaisempaa esimerkkiä. Oletetaan, että tehtävänä olisi poimia käyttäjän IPv4-osoite pitkästä dokumentista, esim. html-sivulta, jonka sisältö on tallennettuna merkkitietovektoriin `sivu`. Tämä voidaan tehdä esimerkiksi seuraavalla tavalla

```
> sivu <- "charset=UTF-8\r\nConnection: close\r\n\r\n130.234.20.11\n"
> kuvio <- "[0-9]{1,3}\\.{1}[0-9]{1,3}\\.{1}[0-9]{1,3}\\.{1}[0-9]{1,3}"
> matchdata <- regexpr(pattern = kuvio, text = sivu)
> osoite <- regmatches(x = sivu, m = matchdata))
> osoite
[1] "130.234.20.11"
```

Funktio `regexpr` palauttaa kuvioon sopivan merkkijonon sijainnin ja pituuden, joiden avulla funktio `regmatches` etsii ja palauttaa kyseisen merkkijonon.

Funktioilla `sub` ja `gsub` voidaan korvata mahdollisesti löytyneitä kuvioon sopivia merkkijonoja. Funktio `sub` korvaa ensimmäisen sopivan merkkijonon ja `gsub` korvaa kaikki sopivat merkkijonot. Korvataan kaikki r-kirjaimet sanasta "r-kurssi"kirjaimella e.

```
> sana <- "r-kurssi"
> kuvio <- "r"
> korvaa < "e"
> gsub(pattern = kuvio, replacement = korvaa, x = sana)
[1] "e-kuessi"
```

Lista funktioista, joita käytettiin

`browser()`	suorituksen pysäytys ja istunnon tarkastelu
`traceback()`	käskyt, jotka johtivat virheeseen
`tryCatch()`	poikkeusten käsittely
`regexpr()`	sopivan merkkijonon sijainti ja pituus
`regmatches()`	funktion regexpr() löytämä merkkijono
`sub()`	ensimmäisen sopivan merkkijonon korvaaminen
`gsub()`	kaikkien sopivien merkkijonojen korvaaminen

10 Esimerkkejä

10.1 Lineaarinen malli

Edellä on jo sovitettu regressiomalli `lm`-funktiolla. Sovitettava malli ilmoitetaan parametrilla formula, jolla on oma syntaksinsa.

Syntaksi	Malli
Y ∼ X	$Y = \beta_0 + \beta_1 X + \epsilon$
Y ∼ X+Z	$Y = \beta_0 + \beta_1 X + \beta_2 Z + \epsilon$
Y ∼ X:Z	$Y = \beta_0 + \beta_1 XZ + \epsilon$
Y ∼ X*Z	$Y = \beta_0 + \beta_1 X + \beta_2 Z + \beta_3 XZ\epsilon$

Merkin ∼ vasemmalla puolella annetaan siis vastemuuttuja ja oikealla puolella selittäjät. Muitakin `formula` parametrin rakenteita on olemassa, mutta nämä ovat niistä yleisimpiä. Edellä olevat yleistyvät helposti useammillekin muuttujille, esimerkiksi kolmen muuttujan pelkät päävaikutukset saataisiin kirjoittamalla Y ∼ X_1+X_2+X_3. Tarkastellaan mallin sovitusta nyt hieman lähemmin.

```
> cars.lm <- lm(dist ~ speed, data=cars)
> plot(cars)
> abline(cars.lm)
> cars.lm

Call:
lm(formula = dist ~ speed, data = cars)

Coefficients:
(Intercept)        speed
    -17.579        3.932

> summary(cars.lm)

Call:
lm(formula = dist ~ speed, data = cars)

Residuals:
    Min      1Q  Median      3Q     Max
-29.069  -9.525  -2.272   9.215  43.201

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601   0.0123 *
speed         3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-Squared: 0.6511,     Adjusted R-squared: 0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.490e-12
```

Sovitimme siis lineaarisen mallin

$$y = \beta_0 + \beta_1 x + \epsilon,$$

missä $y = (y_1, y_2, \ldots, y_n)'$ on pysähtymismatka (`dist`), $x = (x_1, \ldots, x_n)'$ nopeus (`speed`) ja $\epsilon = (\epsilon_1, \ldots, \epsilon_n)'$ ovat riippumattomia, samoin jakautuneita, $\mathbb{E}\epsilon_i = 0$, $\mathrm{var}(\epsilon_i) = \sigma^2$. Nyt n oli siis 50.

Matriisimerkinnöin:

$$y = X\beta + \epsilon,$$

missä nyt

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}.$$

Funktion `lm` antamat estimaatit $\hat{\beta}_0 = -17.5791$ ja $\hat{\beta}_1 = 3.9324$ ovat pienimmän neliösumman estimaatit, jotka saadaan kaavasta $b = (X'X)^{-1}X'y$. Lasketaan tämä itse R:llä:

```
> y <- cars$dist
> X <- matrix(c(rep(1,times=length(cars$speed)),cars$speed),ncol=2,byrow=F)
> b <- solve(t(X)%*%X) %*% t(X) %*% y
> b
           [,1]
[1,] -17.579095
[2,]   3.932409
```

Edelleen harhaton estimaatti σ^2:lle on

$$\hat{\sigma^2} = \frac{1}{n-p}(y - Xb)'(y - Xb),$$

missä p on β-parametrien määrä.

```
> n <- length(cars$speed)
> sigma2 <- 1/(n-2)*t(y-X%*%b)%*%(y-X%*%b)
> sigma2
         [,1]
[1,] 236.5317
```

Huomaa, että objektin sisältämät komponentit saa näkyviin funktiolla `names`:

```
> names(cars.lm)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

Vektori `cars.lm$residuals` sisältää mallin jäännökset $y - Xb$. Vaihtoehtoisesti jäännökset saadaan funktiolla `resid`: `resid(cars.lm)`. Siis estimaatti σ^2:lle voidaan laskea myös esim.

```
> 1/(n-2)*t(as.matrix(cars.lm$residuals))%*%as.matrix(cars.lm$residuals)
         [,1]
[1,] 236.5317
> # tai: 1/(n-2)*cars.lm$residuals%*%cars.lm$residuals
> # tai: 1/(n-2)*sum(cars.lm$residuals*cars.lm$residuals)
```

Lisäksi estimaattorin b kovarianssimatriisin estimaattori on

$$\mathrm{cov}(b) = \hat{\sigma}^2 (X'X)^{-1}.$$

```
> b.cov <- as.numeric(sigma2) * solve(t(X)%*%X)
> b.cov
          [,1]       [,2]
[1,] 45.676514 -2.6588234
[2,] -2.658823  0.1726509
```

Edellä oleva `sigma2` on 1×1 matriisi, joten se on ensin muutettava numeeriseksi muuttujaksi funktiolla `as.numeric`. Estimaattorin b keskivirheet saadaan kovarianssimatriisin diagonaalialkoista niiden neliöjuurina:

```
> sqrt(diag(b.cov))
[1] 6.7584402 0.4155128
```

Tarkastellaan seuraavaksi aineistoa `Lapset2007`, jossa on tietoja vastasyntyneistä Tampereen seudun lapsista. Tutkitaan, selittävätkö pituus ja sukupuoli lapsen painoa. Tytöt ja pojat on koodattu aineistossa ykkösiksi ja nolliksi, joten luodaan aluksi hieman kuvaavampi faktori `fSukupuol`.

```
> lapset$fSukupuol <- factor(lapset$Sukupuol, labels=c("Poika", "Tyttö"))
```

Sovitetaan ensimmäiseksi malli, jossa selittäjänä on ainoastaan pituus.

```
> malli1 <- lm(lapset$Paino ~ lapset$Pituus)
```

Nähdään, että pituus selittää painoa. Sovitetaan nyt malli, jossa on painon lisäksi selittäjänä sukupuoli

```
> malli2 <- lm(lapset$Paino ~ lapset$Pituus + lapset$fSukupuol)
```

Malleja voidaan nyt verrata toisiinsa funktiolla `anova`.

```
> anova(malli1, malli2)
Analysis of Variance Table

Model 1: lapset$Paino ~ lapset$Pituus
Model 2: lapset$Paino ~ lapset$Pituus + lapset$fSukupuol
  Res.Df     RSS Df Sum of Sq      F  Pr(>F)
1     98 9513403
2     97 9244015  1    269388 2.8268 0.09592 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Nähdään, että mallit eivät poikkea merkitsevästi toisistaan. Sukupuolen vaikutus painoon ei siis ole merkitsevä. Tutkitaan kuitenkin vielä interaktion merkitsevyys.

```
malli3 <- lm(lapset$Paino ~ lapset$Pituus * lapset$fSukupuol)
> anova(malli2, malli3)
Analysis of Variance Table

Model 1: lapset$Paino ~ lapset$Pituus + lapset$fSukupuol
Model 2: lapset$Paino ~ lapset$Pituus * lapset$fSukupuol
  Res.Df     RSS Df Sum of Sq      F  Pr(>F)
1     97 9244015
2     96 8946580  1    297436 3.1916 0.07717 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Sukupuolen ja painon interaktio ei siis myöskään ole merkitsevä.

10.2 Yleistetty lineaarinen malli: logistinen regressio

Yleistettyjen lineaaristen mallien sovittamiseen R:ssä on funktio `glm`. Tarkastellaan esimerkkinä aineistoa `beetles`, joka sisältää kuolleiden kovakuoriaisten määrän `Killed` viiden tunnin altistumisen jälkeen hiilidisulfidille (`Dose` = $\log_{10}$(hiilidisulfidin määrä)).

```
Dose    Number  Killed
1.6907    59       6
1.7242    60      13
1.7552    62      18
1.7842    56      28
1.8113    63      52
1.8369    59      53
1.8610    62      61
1.8839    60      60
```

Kiinnostuksen kohteena on, onko hiilidisulfidilla vaikutusta kuolleiden kovariaisten lukumäärään. Sovitetaan tämän tutkimiseksi aineistoon logistinen malli

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_i.$$

missä x_i =Dose$_i$ ja p_i on kuolleisuus ehdolla hiilidisulfidin määrä.

Kun malli on sovitettu, voidaan kuolemien todennäköisyyksiä estimoida kaavalla

$$\hat{p}_i = \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}$$

Luetaan data R:ään

```
> beetles <- read.table("beetles.dat", header=T)
```

ja sovitetaan malli `glm`-funktiolla:

```
# Tapa 1
> reg <- glm(Killed/Number ~ Dose, data=beetles, weights=Number,
+   family=binomial(link = "logit"))
> # Tapa 2
> reg <- glm(cbind(Killed, Number-Killed) ~ Dose, data=beetles,
+   family = binomial())
```

Yksityiskohtainen tulostus:

```
> summary(reg)

Call:
glm(formula = cbind(Killed, Number - Killed) ~ Dose, family = binomial(),
    data = beetles)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5941  -0.3944   0.8329   1.2592   1.5940

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -60.717      5.181  -11.72   <2e-16 ***
Dose          34.270      2.912   11.77   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 284.202  on 7  degrees of freedom
```

```
Residual deviance:  11.232  on 6  degrees of freedom
AIC: 41.43

Number of Fisher Scoring iterations: 4
```

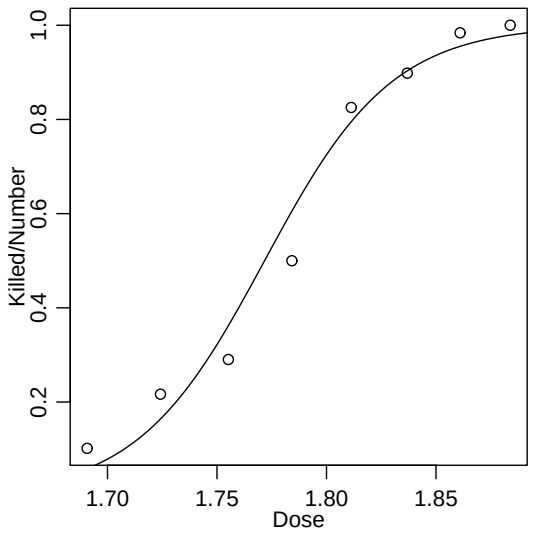
Hiilidisulfidiannoksella on siis vaikutusta kuolleiden lukumäärään.

Piirretään logistinen käyrä samaan kuvaan kuin havaitut arvot (x_i vs p_i).

```
> X11()
> plot(beetles$Dose, beetles$Killed/beetles$Number,
+      xlab="Dose", ylab="Killed/Number")
> curve(plogis(coef(reg)[1]+coef(reg)[2]*x), add=T)
```

Kuvasta voimme mm. arvioida kuinka suuri annos tappaa puolet kovariaisista. Voimme myös laskea tämän arvon. Se on $-\beta_0/\beta_1$:

```
> -coef(reg)[1]/coef(reg)[2]
(Intercept)
   1.771721
```



Mallin parametrien luottamusvälejä voidaan laskea funktiolla `confint`.

```
> confint(reg)
Waiting for profiling to be done...
                2.5 %    97.5 %
(Intercept) -71.44263 -51.07902
Dose         28.85403  40.30069
```

Yksittäisen regressiokertoimen 95 %:n luottamusväli saadaan seuraavasti.

```
> confint(reg, "Dose")
Waiting for profiling to be done...
                2.5 %    97.5 %
Dose         28.85403  40.30069
```

`predict`-funktiolla voidaan ennustaa log-oddseja, oddseja tai todennäköisyyksiä yksilöille, joilla on tietyt selittäjien arvot. Esimerkiksi kuoleman todennäköisyyden ennuste kovakuoriaiselle, jonka myrkkyannos on 1.8, saadaan seuraavasti.

```
> beetle <- data.frame(Dose = 1.8)
> predict(reg, newdata = beetle, type = "response")
        1
0.7249464
```

10.3 Varianssianalyysi

Tarkastellaan aineistoa `pigs` ja tutkitaan sikojen kasvua `d` ajanhetkestä 0 (`day0`) ajanhetkeen 30 (`day30`). Kysymys: onko sikojen kasvussa eroa eri käsittelyryhmien (`trt`) välillä?

Olkoon μ_1, μ_2 ja μ_3 teoreettiset ryhmäkeskiarvot kasvulle. Asetetaan nollahypoteesi $H_0 : \mu_1 = \mu_2 = \mu_3 = \mu$ eli ryhmien välillä ei ole eroa kasvun suhteen.

Teoreettisten keskiarvojen eroa voidaan testata varianssianalyysilla, jonka suorittamiseen R:ssä on funktio `aov`. Funktiota käytettäessä on ryhmiä kuvaavan muuttujan oltava faktori. Sijoitetaan funktion palauttama tulos objektiin nimeltä `tulos`:

```
> tulos <- aov(d ~ trt, data=pigs)
> tulos
Call:
   aov(formula = d ~ trt, data = pigs)

Terms:
                      trt Residuals
Sum of Squares   306.6203 1466.6295
Deg. of Freedom         2        57

Residual standard error: 5.072508
Estimated effects may be unbalanced
```

Varianssitaulu saadaan

```
> summary(tulos)
            Df  Sum Sq Mean Sq F value   Pr(>F)
trt          2  306.62  153.31  5.9583 0.004469 **
Residuals   57 1466.63   25.73
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Teoreettisia ryhmäkeskiarvoja ei siis voida aineiston perusteella pitää yhtä suurina. Käsittelyllä on vaikutusta sikojen kasvuun. Suoritetaan seuraavaksi monivertailut (parittaiset keskiarvovertailut) käyttäen Tukeyn parivertailua (HSD). Funktiolle `TukeyHSD` annetaan parametrina funktion `aov` palauttama tulos.

```
> TukeyHSD(tulos)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = d ~ trt, data = pigs)

$trt
      diff       lwr        upr     p adj
2-1 -2.020 -5.880058  1.8400584 0.4238501
3-1 -5.475 -9.335058 -1.6149416 0.0033558
3-2 -3.455 -7.315058  0.4050584 0.0881412
```

F-testillä (`aov`) saadaan siis tulos, että teoreettiset ryhmäkeskiarvot eroavat toisistaan. Tukeyn parivertailujen avulla saamme selville, mistä tämä ero johtuu.

Varianssianalyysissa on oletettu:
(- tasapainoinen koeasetelma)
- havainnot ovat toisistaan riippumattomia
- havainnot ovat ryhmissä normaalijakautuneita
- perusjoukkojen varianssit ovat yhtä suuret

Näitä oletuksia voidaan tutkia residuaalien avulla. Yllä varianssianalyysin tulos sijoitettiin objektiin `tulos`. Funktioilla `resid` ja `fitted` voidaan poimia argumenttina annettavasta mallinnusfunktion palauttamasta objektista mallin jäännökset ja sovitetut arvot. Näitä voidaan hyödyntää kuvien piirrossa. Esim. (kuva (13))

```
> X11()
> # postscript(file="plot.eps",height=3,width=3,paper="special",horizontal=F)
> par(mfrow=c(1,1), mar=c(4,4,1.5,0.5))
> plot(resid(tulos), pch=16, cex=0.5, xlab="indeksi", ylab="jäännökset")
> abline(h=0)
> # dev.off()
```

Komento `abline(h=0)` lisää kuvioon suoran.

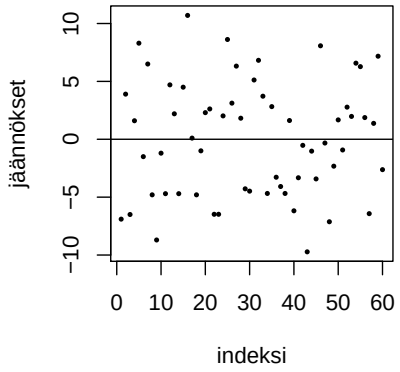
Komento

```
> plot(tulos)
```

tuottaa taas joitakin kuvia jäännöksistä.

10.4 Ristiintaulukko ja χ^2-testi

Jatketaan aineiston `Lapset2007` parissa. Tutkitaan, onko poikia suhteessa tyttöihin enemmän esikoisissa kuin ei-esikoisissa. Muodostetaan aluksi ristiintaulukko



Kuva 12: Jäännösten hajontakuvio.

```
> tt <- table(lapset2$Sukupuol, lapset2$Esikoine,
dnn = list("SP","Esikoinen"))
> tt
   Esikoinen
SP   0  1
  0 26 24
  1 32 18
```

Nollahypoteesina on, että poikien osuus esikoisten joukossa on sama kuin tyttöjen osuus. Hypoteesia voidaan testata χ^2-testillä, jonka testisuure on

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(y_{ij} - e_{ij})^2}{e_{ij}},$$

missä r on taulukon rivien lukumäärä ja c sarakkeiden lukumäärä. Testi vertaa havaittuja frekvenssejä y_{ij} odotettuihin frekvensseihin e_{ij}, ja testisuure noudattaa χ^2-jakaumaa vapausasteilla $(r-1)(c-1)$ nollahypoteesin ollessa voimassa. Taulukon marginaaleja voidaan tarkastella seuraavasti

```
> margin.table(tt,1)
SP
 0  1
50 50
> margin.table(tt,2)
Esikoinen
 0  1
58 42
```

Lasketaan sarakeprosentit eli sukupuolen jakauma muuttujan Esikoinen eri luokissa sekä odotetut frekvenssit

```
> round(prop.table(tt,2)*100,2)
   Esikoinen
```

```
SP      0     1
  0 44.83 57.14
  1 55.17 42.86
> chisq.test(tt)$expected
   Esikoinen
SP   0  1
  0 29 21
  1 29 21
```

Kaikkien solujen odotetut frekvenssit ovat suurempia kuin 5, joten χ^2-testin oletukset ovat voimassa, joten tehdään lopuksi vielä itse χ^2-testi.

```
> chisq.test(tt,correct=FALSE)

        Pearson's Chi-squared test

data:  tt
X-squared = 1.4778, df = 1, p-value = 0.2241
```

Testin mukaan ei ole näyttöä siitä, että poikia olisi enemmän suhteessa tyttöihin esikoisissa kuin ei-esikoisissa. HUOM! `chisq.test`-funktio käyttää oletuksena jatkuvuuskorjausta 2×2-tauluille. Korjauksen voi ottaa pois käytöstä asettamalla parametrin `correct` arvoksi `FALSE`.

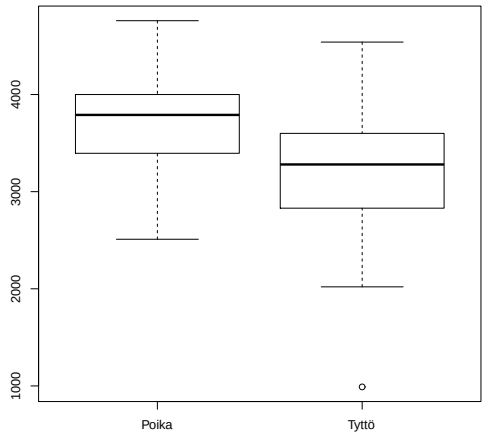
10.5 Kahden otoksen t-testi

Jatketaan aineiston `Lapset2007` parissa. Tutkitaan seuraavaksi, miten painot ovat jakautuneet tyttöjen ja poikien keskuudessa.

Tutkitaan painojen jakaumia

```
> tapply(lapset2$Paino,lapset2$fSukupuol,summary)
$Poika
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2510    3410    3790    3728    3998    4760

$Tyttö
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    990    2830    3280    3217    3588    4540

> tapply(lapset2$Paino,lapset2$fSukupuol,var)
   Poika    Tyttö
214801.8 396901.2
> boxplot(Paino~fSukupuol,data=lapset2)
```

Tunnuslukujen ja boxplotin perusteella näyttäisi siltä, että tytöt ovat keskimäärin kevyempiä kuin pojat. Olkoon μ_p poikien painon teoreettinen keskiarvo ja μ_t vastaavasti tyttöjen painon



Kuva 13: Tyttöjen ja poikien painot

teoreettinen keskiarvo. Asetetaan nollahypoteesi $H_0 : \mu_p = \mu_t$. Teoreettisten keskiarvojen eroa voidaan testata t-testillä. Jos ryhmien variansseja ei oleteta yhtäsuuriksi, niin testisuure on

$$t = \frac{\overline{y}_p - \overline{y}_t}{s_w}, \quad \text{missä} \quad s_w = \sqrt{\frac{s_p^2}{n_p} + \frac{s_t^2}{n_t}},$$

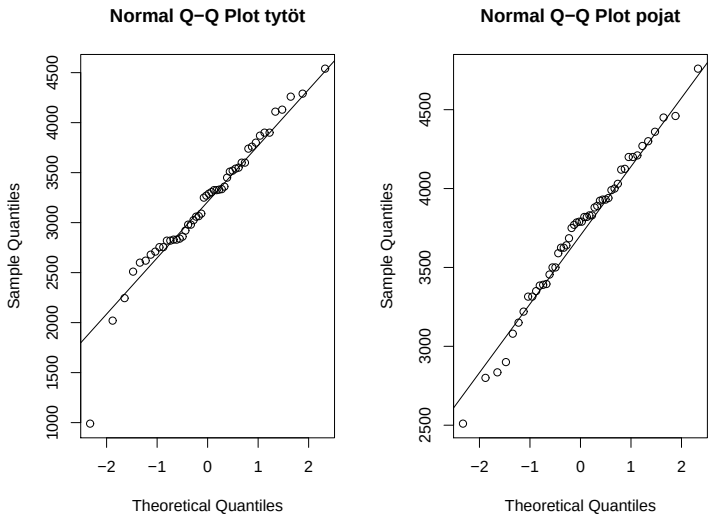
$\overline{y}_p$ ja $\overline{y}_t$ ovat poikien ja tyttöjen otoskeskiarvot, s_p^2 ja s_t^2 ovat vastaavat otosvarianssit, n_p ja n_t ovat ryhmien koot (tässä tapauksessa poikia ja tyttöjä on yhtä monta, eli $n_p = n_t = n$). Testisuure noudattaa t-jakaumaa nollahypoteestin ollessa voimassa, mutta vapausasteet joudutaan tässä tapauksessa estimoimaan.

```
> t.test(Paino~fSukupuol, data=lapset2)

        Welch Two Sample t-test

data:  Paino by fSukupuol
t = 4.6205, df = 90.022, p-value = 1.271e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 291.3193 730.8007
sample estimates:
mean in group Poika mean in group Tyttö
            3728.36              3217.30
```

Testin mukaan teoreettisissa keskipainoissa on eroa, mutta t-testiin liittyvät myös oletukset varianssien yhtäsuuruudesta ja ryhmittäisestä normaalisuudesta. Jaetaan aineisto sukupuolittain ja testataan näiden oletusten paikkansa pitävyyttä.

```
> library(car)
```

```
> tytot <- subset(lapset2, lapset2$fSukupuol=="Tyttö")
> pojat <- subset(lapset2, lapset2$fSukupuol=="Poika")
> leveneTest(lapset2$Paino, lapset2$fSukupuol)
Levene's Test for Homogeneity of Variance (center = median)
      Df F value  Pr(>F)
group  1  2.8755 0.09311 .
      98
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> shapiro.test(tytot$Paino)

        Shapiro-Wilk normality test

data:  tytot$Paino
W = 0.96, p-value = 0.08849

> shapiro.test(pojat$Paino)

        Shapiro-Wilk normality test

data:  pojat$Paino
W = 0.9844, p-value = 0.7437

> par(mfrow=c(1,2))
> qqnorm(tytot$Paino, main="Normal Q-Q Plot tytöt"); qqline(tytot$Paino)
> qqnorm(pojat$Paino, main="Normal Q-Q Plot pojat"); qqline(pojat$Paino)
```



Levenen testin mukaan ei ole näyttöä siitä, että varianssit olisivat erisuuret. Shapiro-Wilkin testien mukaan myös normaalisuusoletus on voimassa. Tehdään nyt t-testi uudelleen olettaen varianssien yhtäsuuruus:

```
> t.test(Paino~fSukupuol, data=lapset2, var.equal=T)

        Two Sample t-test

data:  Paino by fSukupuol
t = 4.6205, df = 98, p-value = 1.166e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 291.5626 730.5574
sample estimates:
mean in group Poika mean in group Tyttö
            3728.36             3217.30
```

Testin tulos on vastaava aiemman tuloksen kanssa. Aineistossa näyttäisi kuitenkin olevan yksi poikkeava havainto. Pieni tyttövauva, jonka paino on 990 g, voi vaikuttaa sekä keskiarvoon, varianssiin että normaalisuuteen. Suoritetaan parametriton testi jakaumien sijaintierojen tutkimiseksi käyttäen kaikkia vauvoja:

```
> kruskal.test(Paino~fSukupuol, data=lapset2)

        Kruskal-Wallis rank sum test

data:  Paino by fSukupuol
Kruskal-Wallis chi-squared = 19.1348, df = 1, p-value = 1.218e-05
```

Tulos on vastaava t-testien kanssa.

10.6 Newtonin menetelmä

Newtonin menetelmä on iteratiivinen algoritmi reaaliarvoisen funktion f juurien approksimoimiseksi. Aluksi on valittava alkuarvo x_0, joka voi olla esimerkiksi arvaus tai arvio todellisesta juuresta. Tietyin oletuksin parempi arvio juurelle saadaan kaavalla

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Iterointia jatketaan seuraavan kaavan mukaisesti kunnes haluttu tarkkuus on saavutettu.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Menetelmä ei toimi joka tilanteessa, sillä tarkasteltavan funktion tulee olla derivoituva, eikä yksikään luku x_n voi olla derivaatan $f'(x_n)$ juuri. Menetelmän toimivuus riippuu myös pitkälti valitusta alkuarvosta x_0.

Käytetään Newtonin menetelmää luvun $\sqrt{7}$ arvioimiseen. Tiedetään, että $\sqrt{7}$ on funktion $f(x) = x^2 - 7$ juuri. Funktion f derivaatta on $f'(x) = 2x$. Toteutetaan iteraatiokaava R:llä.

```
newton <- function(x0,n) {
    x <- c()
    x[1] <- x0
    for(i in 1:n) {
        x[i+1] <- x[i] - (x[i]^2 - 7) / (2 * x[i])
    }
    x[n+1]
}
```

R-funktiolle `newton` annetaan argumentteina alkuarvo x_0 ja haluttu iteraatioiden määrä n. Kokeillaan funktiota, kun $x_0 = 2$ ja $n = 10$, ja verrataan tulosta `sqrt`-funktion antamaan tulokseen.

```
> newton(2,10)
[1] 2.645751
> sqrt(7)
[1] 2.645751
```

Juuria voi etsiä myös funktiolla `uniroot`.

```
uniroot(f, interval, ...,
        lower = min(interval), upper = max(interval),
        f.lower = f(lower, ...), f.upper = f(upper, ...),
        tol = .Machine$double.eps^0.25, maxiter = 1000)
```

Funktion `f` juuria etsitään annetulta väliltä `interval`, jonka voi määrittää myös välin päätepisteinä `lower` tai `upper`. Etsitään funktion $g(x) = x^3 - 2x - 5$ juurta väliltä $(-5, 5)$.

```
> g <- function(x) { x^3 - 2*x - 5 }
> uniroot(g, interval = c(-5,5))
$root
[1] 2.094528

$g.root
[1] -0.0002653143

$iter
[1] 9

$estim.prec
[1] 6.103516e-05
```

Voisimme tehdä tämän myös newtonin menetelmällä halutessamme. Funktion g derivaatta on $g'(x) = 3x^2 - 2$. Toteutetaan iteraatiokaava R:llä.

```
newton2 <- function(x0,n) {
    x <- c()
```

```
    x[1] <- x0
    for(i in 1:n) {
        x[i+1] <- x[i] - (g(x[i])) / (3*x[i]^2-2)
    }
    x[n+1]
}
```

Valitaan alkuarvoksi $x_0 = 2$ ja iteraatiokierrosten lukumääräksi $n = 10$, ja verrataan tulosta funktion `uniroot` tulokseen.

```
> newton2(2,10)
[1] 2.094551
```

Tulos poikkeaa hieman viidennessä desimaalissa, mutta on muuten vastaava.

10.7 Uskottavuusfunktion piirtäminen ja numeerinen optimointi

Tarkastellaan otosta ($n = 25$) Poisson-jakaumasta, jonka keskiarvo on $\lambda = 10$.

```
> y <- rpois(25, 10)
> y
 [1]  5 14 11  8  6 11 10  9 11  6 14 12  7  5 16 11 13 15 10  4 10  8  8  6 11
```

Uskottavuusfunktio on

$$L(\lambda; y) = \prod_{i=1}^{n} \frac{1}{y_i!} \lambda^{y_i} e^{-\lambda}.$$

Logaritminen uskottavuusfunktio:

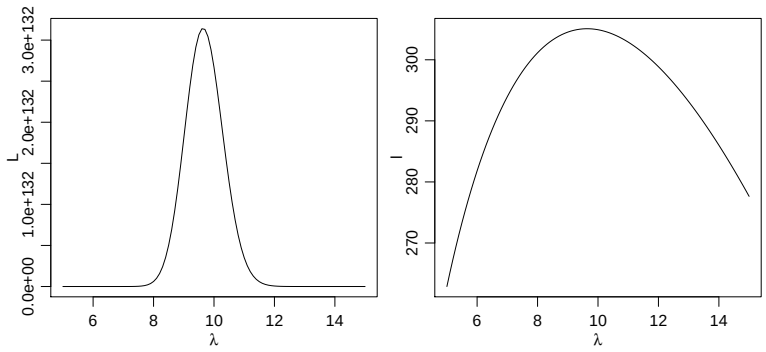
$$l(\lambda; y) \propto \left(\sum_{i=1}^{n} y_i \right) \log \lambda - n\lambda$$

Tehdään näistä R-funktiot:

```
L <- function(lambda, y) {
    lambda^(sum(y))*exp(-length(y)*lambda)
}
l <- function(lambda, y) {
    sum(y)*log(lambda)-length(y)*lambda
}
```

ja ladataan ne R:ään. Piirretään funktiot

```
> X11()
> par(mfrow=c(1,2))
> curve(L(x, y), 5, 15, xlab=expression(lambda), ylab="L")
> curve(l(x, y), 5, 15, xlab=expression(lambda), ylab="l")
```



Kuva 14: Uskottavuusfunktio ja logaritminen uskottavuusfunktio.

Nyt osataan helposti laskea su-estimaatti (ja se on $\hat{\lambda} = \bar{y} = 9.64$), mutta aina näin ei ole, vaan su-estimaatti joudutaan laskemaan numeerisesti esimerkiksi Newtonin algoritmilla:

$$\lambda_{new} = \lambda_{old} + \frac{S(\lambda; y)}{I(\lambda; y)}$$

missä siis etsitään pistemääräfunktion $S(\lambda; y)$ nollakohtaa. Nyt

$$S(\lambda; y) = \frac{\sum_{i=1}^{n} y_i}{\lambda} - n$$

ja

$$I(\lambda; y) = \frac{\sum_{i=1}^{n} y_i}{\lambda^2}$$

Ohjelmoidaan algoritmi R:llä:

```
nextlambda <- function(lambda, y) {
    S <- function(lambda, y) { sum(y)/lambda - length(y) }
    I <- function(lambda, y) { sum(y)/lambda^2 }
    lambda + S(lambda, y)/I(lambda, y)
}
lambda.hat <- function(lambda0, y, tarkkuus=0.001) {
# lambda0 = alkuarvaus
    lambda1 <- lambda0
    lambda2 <- nextlambda(lambda1, y)
    while(abs(lambda1-lambda2)>tarkkuus) {
        lambda1 <- lambda2
        lambda2 <- nextlambda(lambda1, y)
    }
    lambda2
}
```

Funktiota `lambda.hat` voidaan kutsua

```
> lambda.hat(lambda0=5, y=y)
[1] 9.64
```

Kaikkia tällaisia ohjelmia ei ole kuitenkaan tarve ohjelmoida itse, sillä R:stä löytyy valmiita funktioita numeeriseen optimointiin: Esim.

```
optimize(f = , interval = , lower = min(interval),
         upper = max(interval), maximum = FALSE,
         tol = .Machine$double.eps^0.25, ...)
```

joista `optimize` etsii funktion `f` maksimin/minimin annetulta väliltä funktion ensimmäisen parametrin suhteen.

Maksimoidaan logaritminen uskottavuusfunktio:

```
> optimize(f = l, interval = c(5,15), maximum = TRUE, tol = 0.00001, y=y)
$maximum
[1] 9.64

$objective
[1] 305.087
```

Funktion antama tuloste voi olla hieman harhaanjohtava, sillä `$maximum` ei ilmoita maksimiarvoa, vaan pisteen jossa maksimi saavutetaan. Maksimiarvon ilmoittaa `$objective`.

Funktio `optimize` on tarkoitettu pääasiassa optimointiin yhden parametrin suhteen. Funktioilla `optim` ja `nlm` voi optimoida funktioita useampien parametrien suhteen.

```
optim(par, fn, gr = NULL, ...,
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN",
                 "Brent"),
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE)
```

Parametri `par` määrittää optimoinnin alkuarvot, `fn` on optimoitava funktio ja `lower` ja `upper` määrittävät alueen, jolta optimaalista ratkaisua etsitään. Lisäksi valittavana on lukuisia eri optimointimenetelmiä ja kontrolliparametreja.

Etsitään funktion $f(x, y) = y^2 \exp(-0.5(y^2 + x^2))$ lokaali maksimi joukossa $-1 < x < 3, -1 < y < 3$. Annetaan alkuarvoiksi $x = 0.5$ ja $y = 0.5$. Kun halutaan rajoittaa joukkoa, josta optimaalista ratkaisua etsitään, on käytettävä optimointimenetelmää, joka tukee rajoitteiden asettamista. Tässä tapauksessa valitaan menetelmäksi "L-BFGS-B". `optim`-funktio etsii oletusarvoisesti funktion minimiä, joten vaihtamalla funktion merkki etsitäänkin maksimia.

Huomaa, että funktiolla `f` on vain yksi parametri, vaikka funktiolla f on kaksi parametria. Tämä johtuu siitä, että `optim`-funktion tapauksessa optimoitavien parametrien on esiinnyttävä funktion argumenteissa vektorina. Vektorin `x` ensimmäinen alkio `x[1]` vastaa siis muuttujaa x ja toinen alkio `x[2]` vastaa muuttujaa y. Tämä yleistyy useamman kuin kahden muuttujan funktioille, kun vektorin `x` pituutta kasvatetaan vastaavasti.

```
> f <- function(x) -x[2]^2*exp(-0.5*(x[2]^2+x[1]^2))
> optim(c(0.5, 0.5), f, lower = c(-1, -1), upper = c(3, 3), method = "L-BFGS-B")
$par
[1] -7.582426e-10  1.414214e+00

$value
[1] -0.7357589

$counts
function gradient
       8        8

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

`$par` kertoo maksimipisteen koordinaatit. Ensimmäinen luku kertoo maksimipisteen x-koordinaatin, ja toinen sen y-koordinaatin. `$value` ilmoittaa löydettyä optimia vastaavan arvon. Koska funktion merkki vaihdettiin maksimin etsimiseksi, on todellinen maksimiarvo siis löydetyn optimin vastaluku, eli ≈ 0.7357589. Muut tulostukset ovat optimoinnin konvergenssiin liittyviä lisätietoja. Vaihtoehtoisesti maksimia voi etsiä suoraankin vaihtamatta funktion merkkiä antamalla `optim`-funktiolle lisäparametri `control = list(fnscale = -1)`.

Etsitään vielä kolmen muuttujan funktion $g(x, y, z) = \exp(-x^2 - 3x - 7y^2 + 3y + z^3 - 2z - 3)$ lokaali maksimi joukossa $-2 < x < 2, -3 < y < 3, -3 < z < 0$.

```
> g <- function(x) exp(-x[1]^2-3*x[1]-7*x[2]^2+3*x[2]+x[3]^3-2*x[3]-3)
> optim(c(0.5, 0.5, -0.5), g, lower = c(-2, -3, -3), upper = c(2, 3, 0),
+       method = "L-BFGS-B", control = list(fnscale = -1))
$par
[1] -1.5000009  0.2142866 -0.8164968

$value
[1] 1.934968

$counts
function gradient
      22       22

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Kuten edellä, `$par` ilmoittaa maksimipisteen koordinaatit. Maksimi saavutetaan siis pisteessä $(x, y, z) \approx (-1.5000009, 0.2142866, -0.8164968)$ jolloin funktio g saa kohdan `$value` ilmoittaman arvon ≈ 1.934968.

Alkuarvot funktioille `optim` ja `optimize` tulee valita siten, että rajoitteet ovat voimassa. Alkuarvojen valintaan on vaikea antaa yleispätevää ohjetta, ja usein onkin hyvä kokeilla eri arvoja ja verrata niillä saatuja tuloksia. Yhden ja kahden muuttujan tapauksissa löydettyjen optimien mielekkyyttä voi tarkastella esimerkiksi piirtämällä funktion kuvaaja annetussa joukossa.

10.8 Numeerinen integrointi

R:n optimointityökaluihin kuuluu myös funktio `integrate`, jolla voi laskea useimpien funktioiden määrättyjä integraaleja.

```
integrate(f = , lower = , upper = , ..., subdivisions = 100L,
          rel.tol = .Machine$double.eps^0.25, abs.tol = rel.tol,
          stop.on.error = TRUE, keep.xy = FALSE, aux = NULL)
```

Ensimmäinen parametri `f` on funktio, jota halutaan integroida. Parametrit `lower` ja `upper` määräävät integrointivälin, jonka päätepisteet voivat olla myös äärettömiä. Tällöin asetetaan `lower = -Inf` tai vastaavasti `upper = Inf`.

Integroidaan funktiota $f(x) = x^2 + 3x - 2$ välin $[-2, 3]$ yli.

```
> poly <- function(x) { x^2 + 3*x - 2 }
> integrate(poly, -2, 3)
9.166667 with absolute error < 2.8e-13
```

10.9 AR(1)-aikasarjan simulointi

Tarkastellaan AR(1)-aikasarjan

$$y_t = \beta y_{t-1} + \epsilon_t, \quad \epsilon \sim N(0, \sigma^2),$$

simulointia. Tehdään funktio, joka simuloi AR(1)-prosessin annetuilla parametreilla β ja σ^2 ja alkuarvolla y_1.

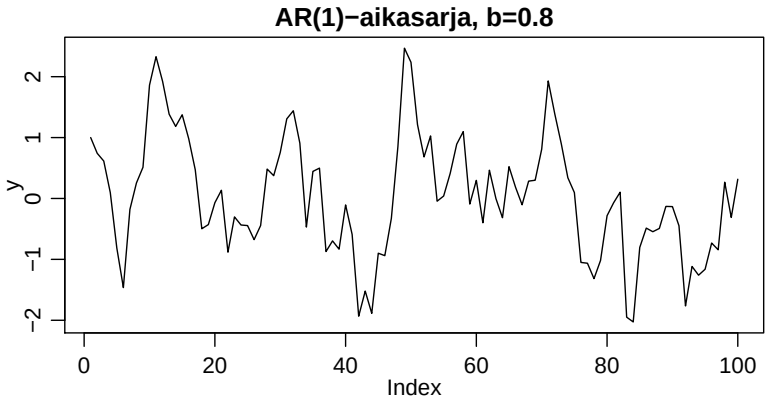
```
AR1 <- function(n, b, sigma2, y1)
{
# simuloi AR(1) prosessin y(t) = b*y(t-1)+e(t), e~N(0,sigma2)
# n  = aikasarjan pituus
# y1 = alkuarvo
    y <- NULL
    y[1] <- y1
    for(i in 2:n) {
        y[i] <- b*y[i-1] + rnorm(1, 0, sqrt(sigma2))
    }
    y
}
```

Funktiota voidaan kutsua

```
source("AR1.r")
y <- AR1(n=100, b=0.8, sigma2=1, y1=1)
```

ja kuvaaja piirtää

```
> X11()
> par(pin=c(5,2),mfrow=c(1,1))
> plot(y, type="l", main="AR(1)-aikasarja, b=0.8")
> # tiedostoon
> postscript(file="ar1_b08.eps",height=3,width=6,paper="special",horizontal=F)
> par(mfrow=c(1,1),mar=c(2.5,2.5,1.5,1.5),mgp=c(1.5,0.7,0))
> plot(y, type="l", main="AR(1)-aikasarja, b=0.8")
> dev.off()
windows
      2
```



10.10 Satunnaisten pisteiden simulointi yksikköneliöön

Simuloidaan täysin satunnainen pisteprosessi (homogeeninen Poisson prosessi) yksikköneliöön. Kirjoitetaan funktio `pois` tiedostoon `homogPoisson.r`:

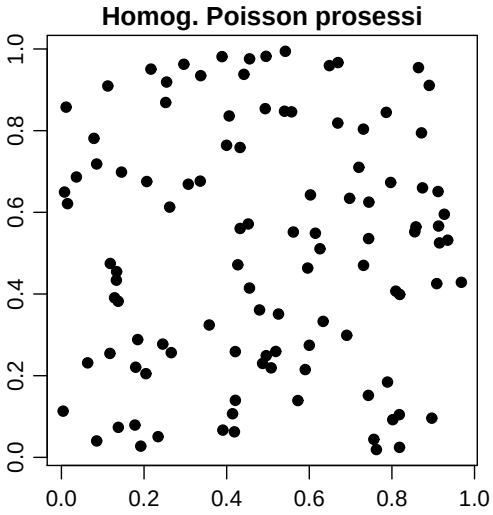
```
pois <- function(lambda, Plot=TRUE)
{
# simuloi homog. Poisson prosessin alueeseen [0,1]x[0,1]
# intensiteetilla lambda (odotettu pisteiden lukumäärä yksikköneliössä)
# piirtää kuvan, jos Plot=TRUE
    n<-rpois(1,lambda)   # pisteiden lukumäärä
    u<-runif(n)          # pisteiden x-koordinaatit
    v<-runif(n)          # pisteiden y-koordinaatit

    if(Plot) {
        X11()
```

```
        par(mfrow=c(1,1),mar=c(2.5,2.5,1.5,1.5),mgp=c(1.5,0.7,0))
        plot(u, v, xlab="", ylab="", main="Homog. Poisson prosessi", pch=16)
    }

    data.frame(x=u, y=v)
}
```

Funktion kutsu on

```
> source("homogPoisson.r")
> pisteet <- pois(100)
```

jolloin funktio piirtää myös kuvan R:n grafiikkaikkunaan.



Lista funktioista, joita käytettiin

`lm()`	lineaarinen mallin sovitus
`names()`	objektin komponenttien nimet
`as.numeric()`	asettaa objektin numeric-tyyppiseksi
`glm()`	yleistetyn lineaarisen mallin sovitus
`summary()`	tiivistelmä
`plogis()`	logistisen jakauman kertymäfunktio, käänteinen logit-funktio
`coef()`	sovitetun mallin kertoimet
`aov()`	varianssianalyysi
`TukeyHSD()`	Tukeyn parivertailut
`table()`	ristiintaulukon muodostaminen
`margin.table()`	taulukon marginaalien tarkastelu
`prop.table()`	suhteellisten osuuksien taulukko
`chisq.test()`	χ^2-testi
`leveneTest()`	Levenen testi varianssien yhtäsuuruudelle
`shapiro.test()`	Shapiro–Wilk-testi normaalisuudelle
`kruskal.test()`	Kurskal–Wallisin testi samoinjakautuneisuudelle
`uniroot()`	juurien etsintä
`rpois()`	Poisson-jakautuneiden muuttujien simulointi
`abs()`	itseisarvo
`optimize()`	optimointi (minimointi/maksimointi)
`optim()`	optimointi
`nlm()`	optimointi
`integrate()`	integrointi

11 Omien C/FORTRAN -ohjelmien liittäminen R:ään

Oleellinen ominaisuus R:ssä on, että käyttäjä voi itse kirjoittaa C/FORTRAN-kielisiä funktioita ja liittää ne R-funktioiksi. Tämä poistaa yhden tulkkaavan kielen ongelman, silmukoiden hitauden, sillä silmukat voidaan kirjoittaa C/FORTRAN-kielisiksi käyttäjän omiksi ohjelmiksi. Seuraavassa selvitetään, miten C-funktiota voi käyttää R:ssä.

Perusarkkitehtuuri on seuraava:

1. Kirjoitetaan C-funktio `funktio.c`, jonka parametrilista koostuu osoittimista (ehdoton vaatimus). C-funktion täytyy olla tyyppiä `void`. Tämän funktion on tarkoitus tehdä laskenta.
2. C-ohjelma käännetään .dll-tiedostoksi (Windows) tai .so-tiedostoksi (Unix) komennolla `R CMD SHLIB funktio.c` komentokehoitteessa tai suoraan R:ssä komennolla `system('R CMD SHLIB funktio.c')`.
3. Tehdään R:ssä R-kielinen "syöttöohjelma", joka linkittää R:ssä olevan datan ja parametrit C-ohjelmaan ja ottaa vastaan laskennan tulokset.
4. `.dll/.so`-tiedosto ladataan käyttöön R:ssä `dyn.load("funktio.dll")`-komennolla.
5. Käytetään funktiota.
6. Lopetetaan kytkentä C-ohjelmaan komennolla `dyn.unload("funktio.dll")`.

Unix-ympäristössä on valmiina C-ohjelmaan kääntämiseen tarvittavat työkalut, mutta Windowsissa nämä (Rtools.exe, sis. MinGW ja Perl, ladattavissa netistä) täytyy asentaa koneelle (ja lisätä path-muuttujaan).

Esimerkki. C-ohjelma, joka laskee keskiarvon.

```
void pkesk(int *n, double *y, double *ka){
// n = vektorin y pituus (=alkioiden määrä)
   double summa=0.0;
   int i;
   for(i=0; i<*n; i++) summa=summa+y[i];
   *ka=summa/(*n);
}
```

Ylläoleva koodi on tallennettuna tiedostossa `pkesk.c`.

Ohjelman kääntäminen tehdään komentokehotteessa hakemistossa, jossa `pkesk.c` on, komennolla

```
>R CMD SHLIB pkesk.c
```

Windows: Tämä tuottaa useita tiedostoja, joista `pkesk.dll` voidaan ladata käytettäväksi R:ssä.

Tehdään R-funktio koodieditorilla tiedostoon keskiarvo.r. Se on seuraavanlainen:

```
keskiarvo <- function(x) {
    n <- as.integer(length(x))
    .C("pkesk", n, as.double(x), ka=as.double(0))$ka
}
```

Funktiossa `.C` ensimmäisenä on C-ohjelman nimi. Sitä seuraa C-funktiolle annettavat muuttujat, jotka vastaavat C-funktion argumenttilistaa. R:n ohjelmassa tulee määritellä/vahvistaa kunkin muuttujan tyyppi tilavarausta varten. Tämä voidaan tehdä käyttäen funktioita `as.integer`, `as.double` jne. Tai seuraavasti

```
storage.mode(x)<-"double"
```

Vastaavuudet R:n ja C:n tyyppien välillä ovat seuraavat:

```
R storage mode      C type
---------------------------------
logical             int *
integer             int *
double              double *
complex             Rcomplex *
character           char **
raw                 char *
list                SEXP *  (ei funktion .C kanssa)
---------------------------------
```

Funktio `.C` palauttaa muuttujat, jotka sille on annettu parametreina ja joita `.c`-funktio mahdollisesti on muuttanut. Kiinnostavalle muuttujalle, jota `.c`-funktio (tässä `pkesk`) muuttaa, kannattaa antaa nimi: Funktion `.C` argumenttilistassa voidaan antaa 'nimi' kullekin muuttujalle, kuten yllä funktion `.C` kutsussa `ka`. Tämä mahdollistaa, että tähän muuttujaan voidaan viitata `$ka`.

Ohjelmien käyttö R:ssä

R-funktio luetaan R:ään:

```
> source("keskiarvo.r")
```

`.dll`-tiedosto ladataan R:ään (R Console -ikkunassa):

```
> dyn.load("pkesk.dll")
```

Ladattuja objekteja voi kysyä:

```
> is.loaded("pkesk")
[1] TRUE
```

Funktio toimii nyt (ladattuna) kuten mikä tahansa R-funktio. Esim.

```
> x <- c(7.9, 1.9, 1.5, 17.4, 9.4, 9.3, 2.5, 13.0, 2.6, 5.7)
> keskiarvo(x)
[1] 7.12
> # Tarkastus (valmiilla) R-funktiolla mean()
> mean(x)
[1] 7.12
```

Kytkentä kirjastoon lopetetaan komennolla

```
> dyn.unload("pkesk.dll")
```

Lista funktioista, joita käytettiin

`as.integer()`	kokonaisluvuksi, integer-objektiksi asettaminen
`as.double()`	double-objektiksi asettaminen
`storage.mode()`	talletustyypin asettaminen
`dyn.load()`	.dll-tiedoston lataaminen R:ään
`is.loaded()`	onko c/fortran-funktio ladattuna
`dyn.unload()`	.dll-tiedoston vapauttaminen