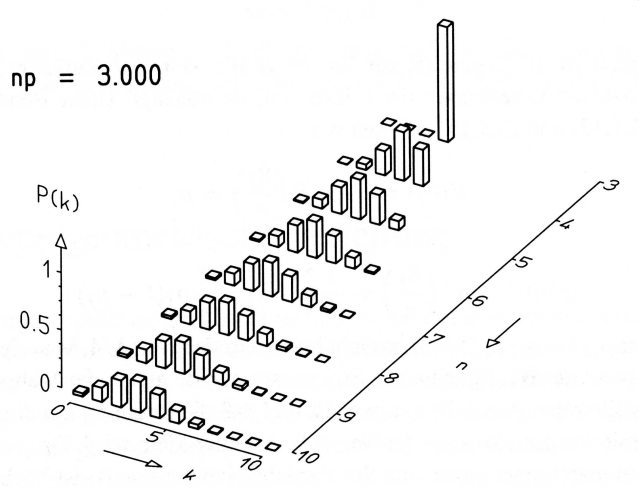
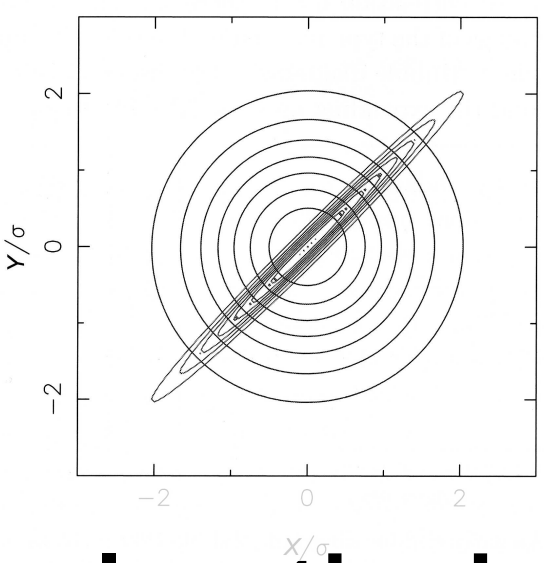
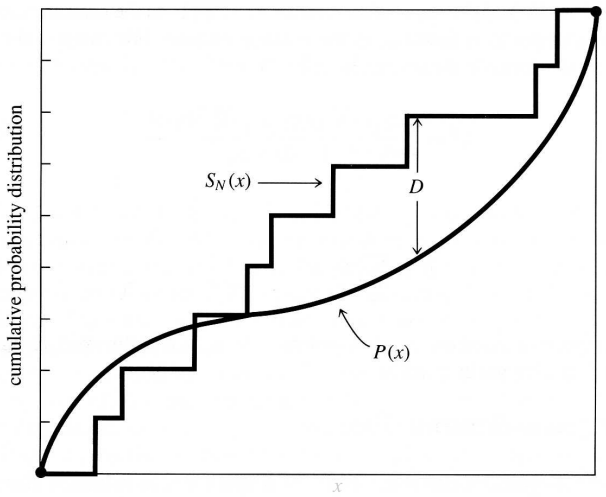






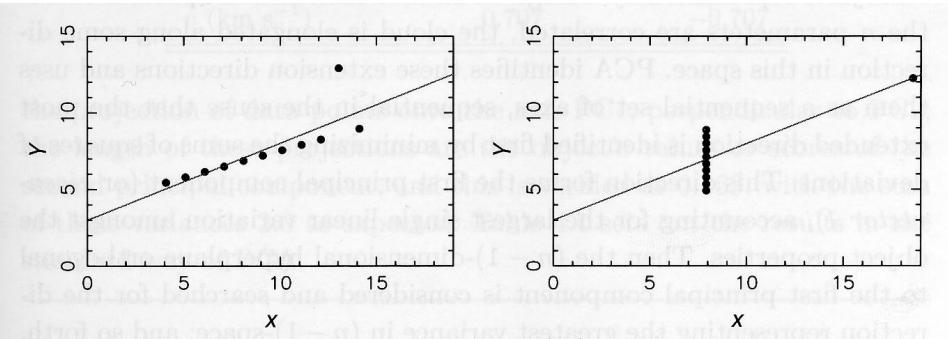
Statistical methods (UKSta)

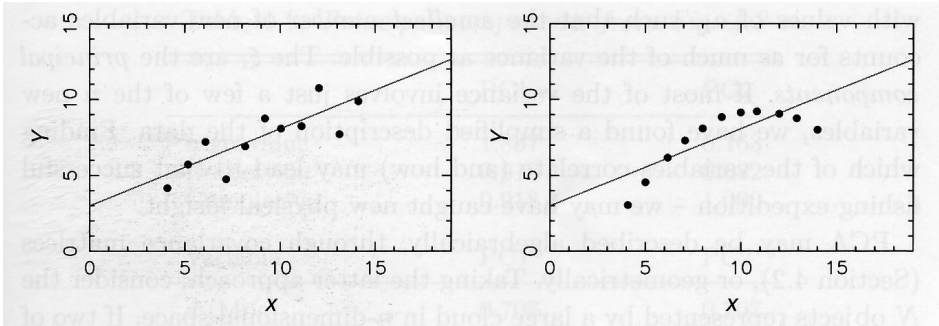
R tutorial

Dr. Hans-Günter Ludwig

Summer term 2018

(original lecture by N. Christlieb)





R

- Programming language and environment for statistics and data analysis
- Platforms: Linux, MacOS X, Windows
- Published under GNU General Public License (GPL); i.e., freely available (see `www.r-project.org`)
- Command-line; interpreter
- Object oriented
- Own programs can easily be integrated
- Extensive statistics library
- Very powerful graphics package

Starting and quitting R

```
$ cd                        # Change to home directory
$ mkdir UKStaSS18           # Create directory
$ cd UKStaSS18              # Change to that directory
$ mkdir Monday              # (see above)
$ cd Monday                 # (see above)
$ R                         # Starting R
>                           # This is the R prompt
> [COMMANDS…]
> q()                       # Quitting R
> Save workspace image? [y/n/c]
```

R basics

- Commands
  a) Expressions: evaluated, displayed, value lost.
     > 1+1
  b) Assignments: general: variable <- object

     variable <- *expression*

     variable <- *function(parameters)*

     Right hand side is evaluated, value assigned to *variable*.

     > y <- 1+1
     > (y <- 1+1) # Print result on screen
     > z <- cos(pi) ; cos(pi) ->z
- The beginning of a comment is indicated with a hashmark (‘#’).

R basics

- Commands are case sensitive.
- Commands are separated by a semicolon (‘;’), or by a newline.
- Command history: use up/down arrow keys. Previously executed commands can be edited and executed again.
- Command editing analogous to Linux shell
- To search in the command history, type ctrl-r.

Arithmetic operators

^	raise to the power
**	raise to the power
*	multiplication
/	division
+	adding
-	subtracting
%/%	integer division
%%	modulo division

Logical operators and functions

==, !=	equal, not equal
<, <=	smaller, smaller or equal
>, >=	larger, larger or equal
!	not
&, &&	and
|, ||	or
xor()	exclusive or

Vector/matrix operations and functions

%*%	inner product
%o%, outer()	outer product
dim(), ncol(), nrow()	number of columns and rows
diag()	reading/setting diagonale
eigen()	eigenvalues/eigenvectors
solve()	inverting a matrix, etc.
t()	transpose

Pre-defined functions

max(), min()	minimum/maximum value
abs()	absolute value
sqrt()	square root
round()	rounding
sum(), prod()	sum, product
log(), log10()	logarithms
exp()	exponential function
sin(), cos(), tan()	trigonometric functions

Creating your own functions

Syntax:

```
FunctionName <- function(arguments){
        # Commands here
}
```

Example:

```
TimesTwo <- function(z){
    y <- 2.0 * z
    y
}
```

Remark: Define each function in a separate file; invoke them with source().

Passing arguments to functions

Definition:

```
f <-function(a,b,c=3){
    # Commands here
}
```

Envocation:

```
result <- f(1,2,3)
result <- f(c=3, a=1, b=2) # equivalent to above
result <- f(1,2)           # equivalent default c=3
```

Assignments to variables in function body are local.

“Superassignment“ <<- for changing variables in host environment

Miscellaneous

getwd()

setwd(“[PATH]”)

system(“[LINUX-COMMAND]”)

source(“[FILENAME]”)

source(…, echo = TRUE)

sink(“[FILENAME]”)

sink()

Resources

- ?, ?<cmd>, help(), help.start(), help.search()
- R reference card (is on course page in Moodle).
- Venables et al. (2017), *An Introduction to R*
- R reference manual;
  use table of contents, index, or full text search.

See also UKSta on Moodle

R exercise 01: basic commands and operations

(1) Execute the commands listed in the handout one by one via the R prompt.
(2) See and understand what happens. In case of functions, carefully read the help pages and browse through them.
(3) Play around with the commands by changing the parameters, choosing different command options, etc.
(4) Write some of the commands into a file and invoke them with the function source(). Explore the parameters and options of this function.
(5) Using the function sink(), divert the screen output of some of the commands to a file. Check what has been written to the file. Explore the parameters and options of this function.

Data inspection

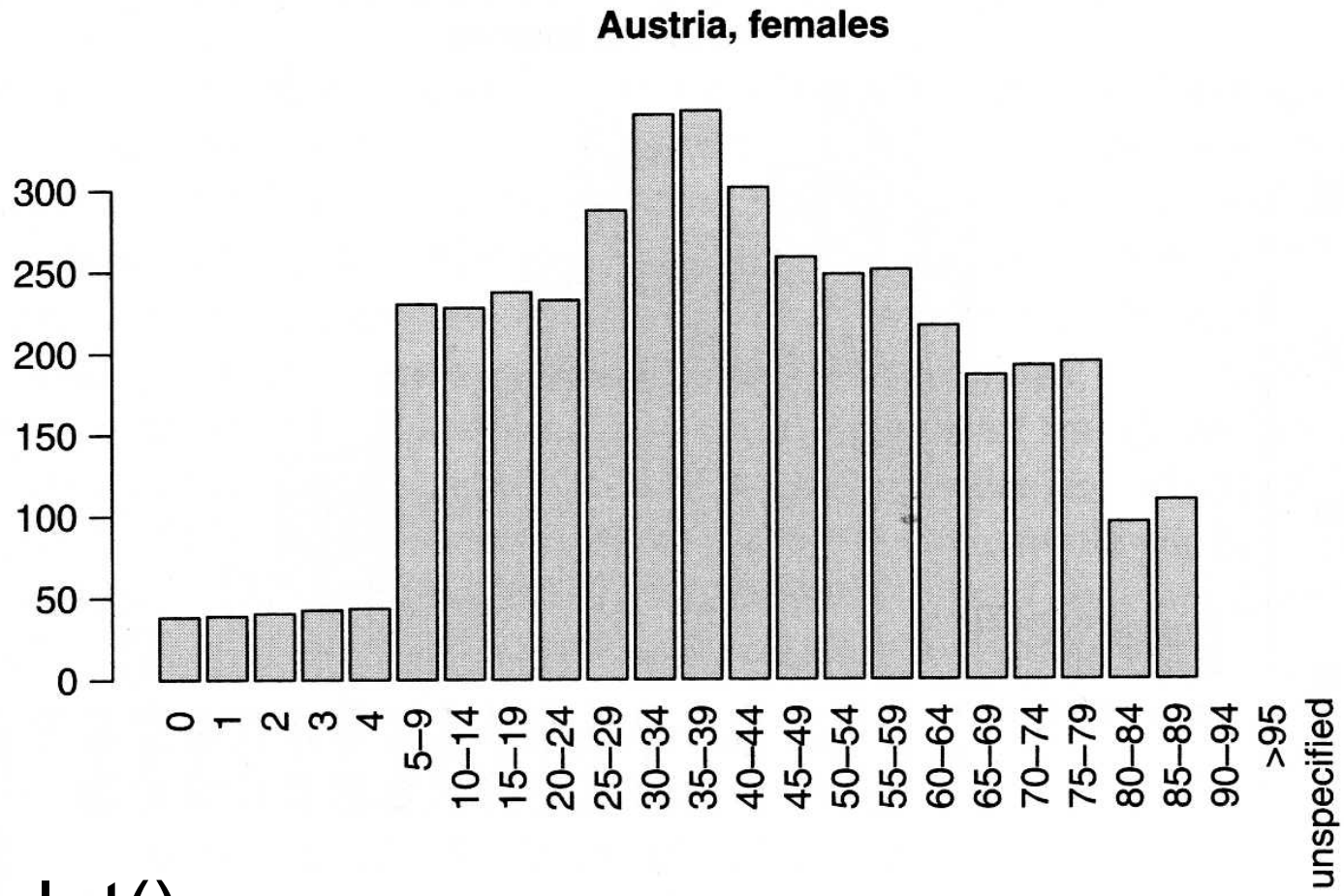
- **Always inspect your data** before you do any kind of statistical analysis, model fitting, etc.
  - How many variables are there?
  - Are they continuous, discrete, categorial, binary?
  - How much data are there?
  - What is the range of values covered by the dataset?
  - Are there any trends, outliers, peaks, correlations?
- Tools:
  - range(), min(), max()
  - Scatter plot; histogram; density plot; cumulative distribution
  - Contour plots

Plotting commands

- **High-level plotting** functions create a new plot on the graphics device; usually with axes, labels, titles, etc.
- **Low-level plotting** functions add more information to an existing plot, such as extra points, lines, and labels.
- **Interactive** graphics functions allow you to interactively add information to, or extract information from, a graphics window, using the mouse.

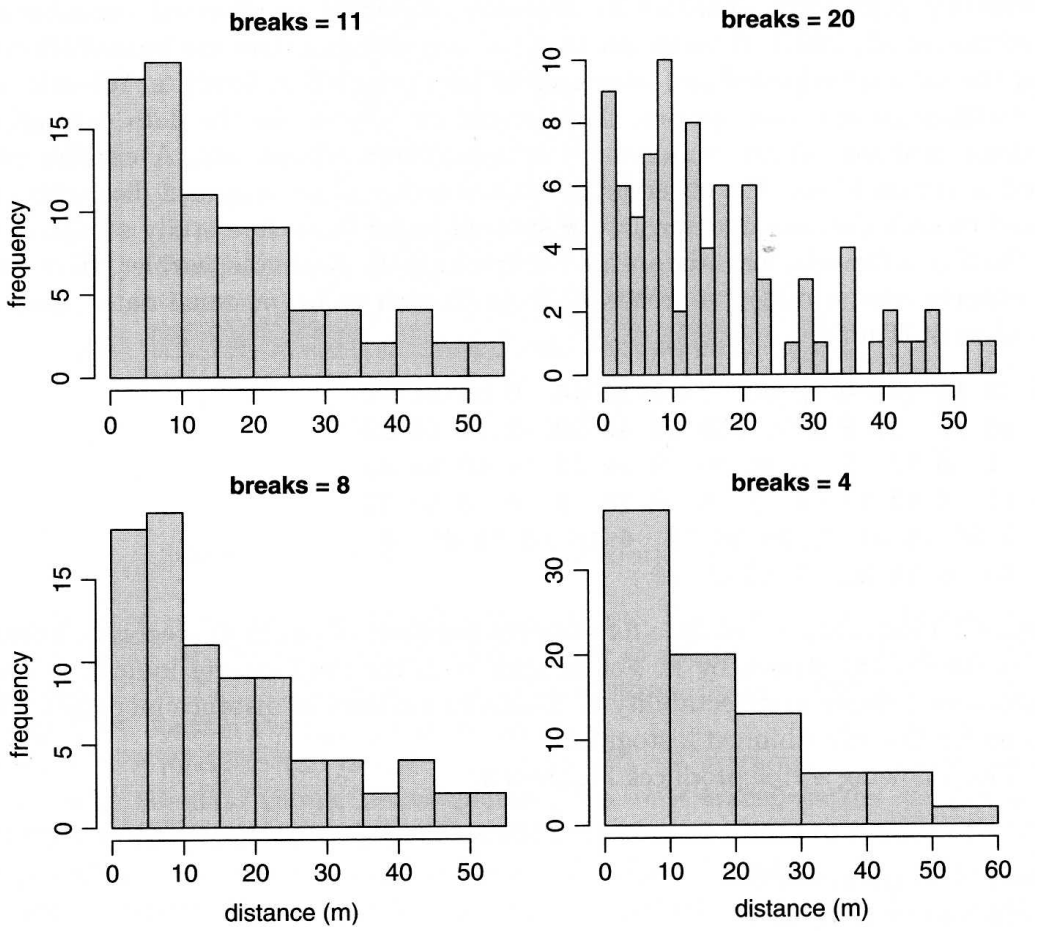
See chapter 12 of *An Introduction to R* for more information, and examples.

Bar plot



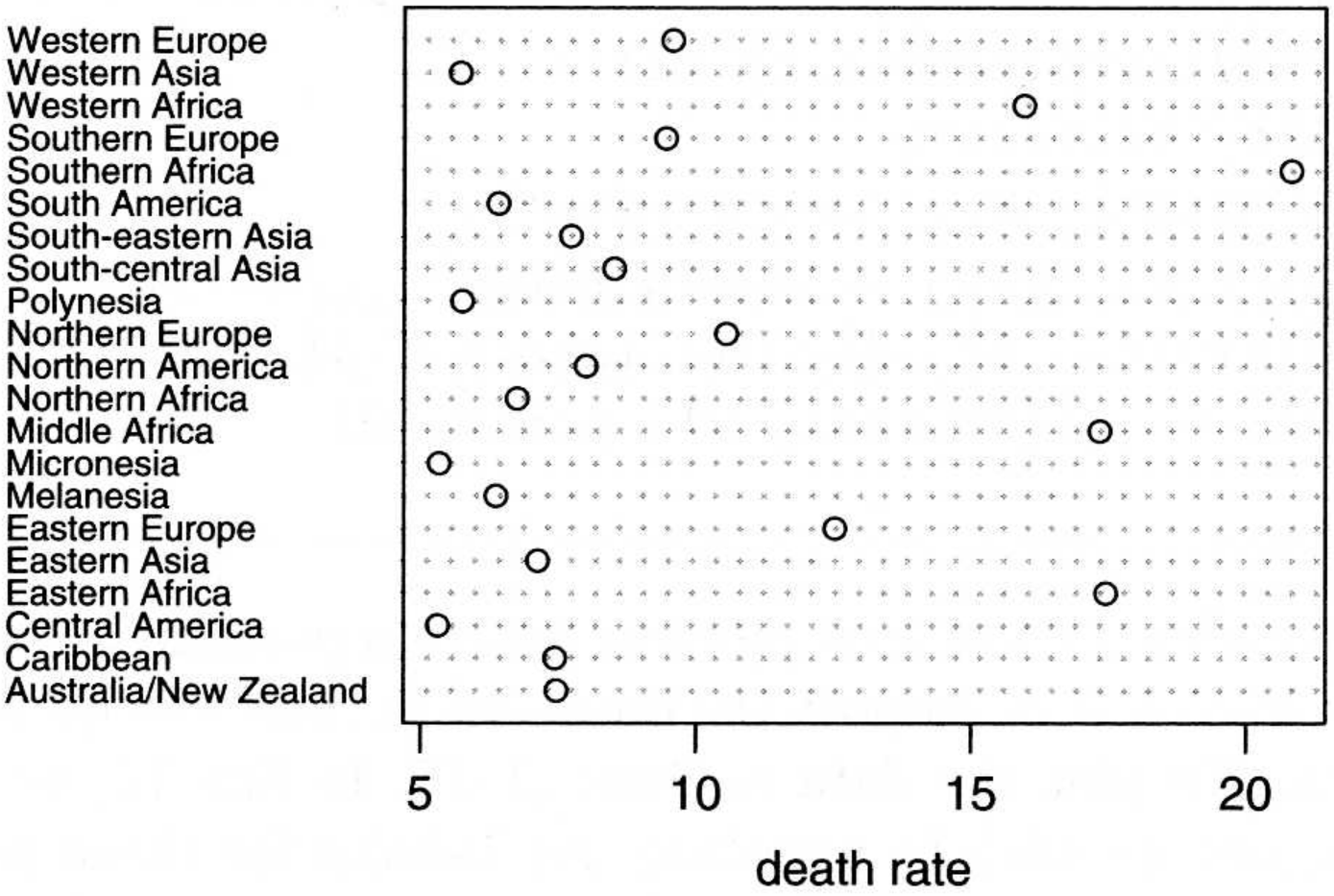
barplot()

Histogram



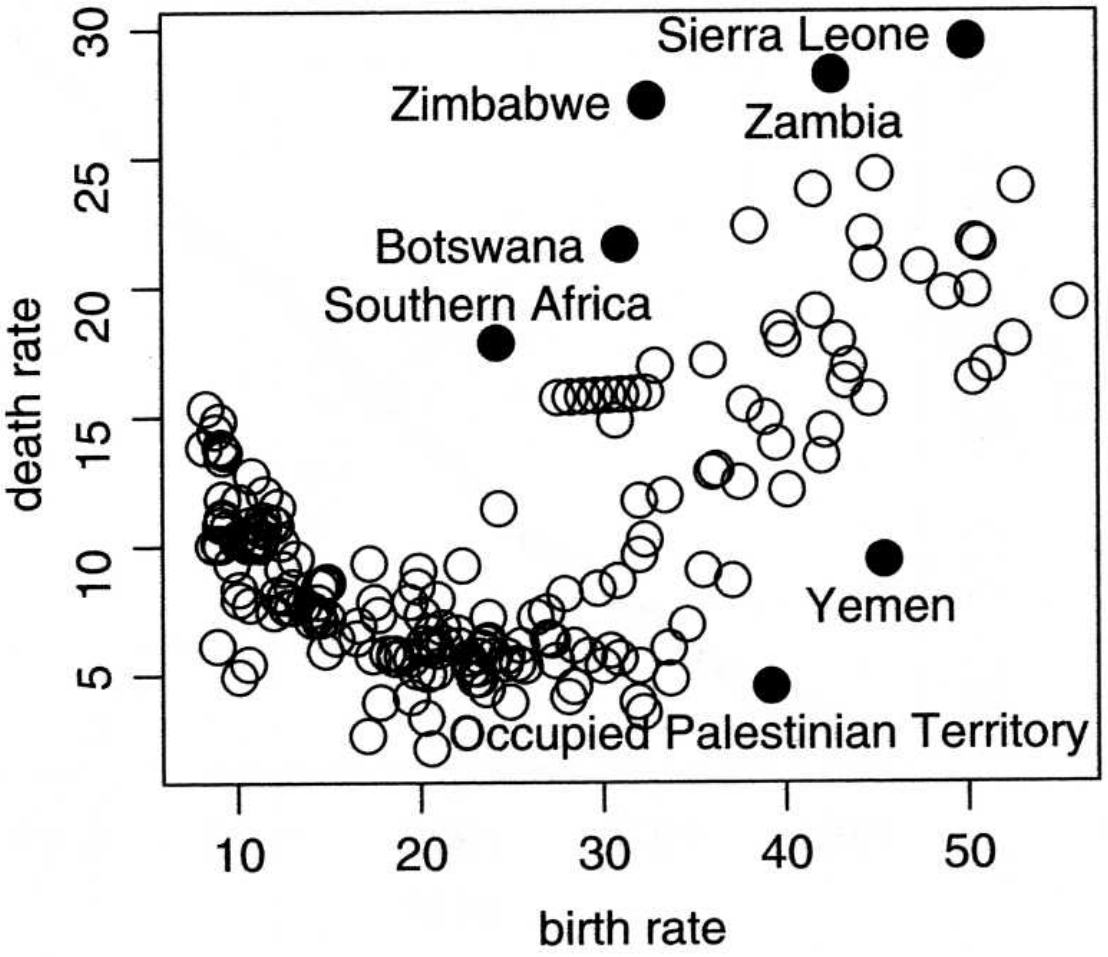
hist()

Dot plot



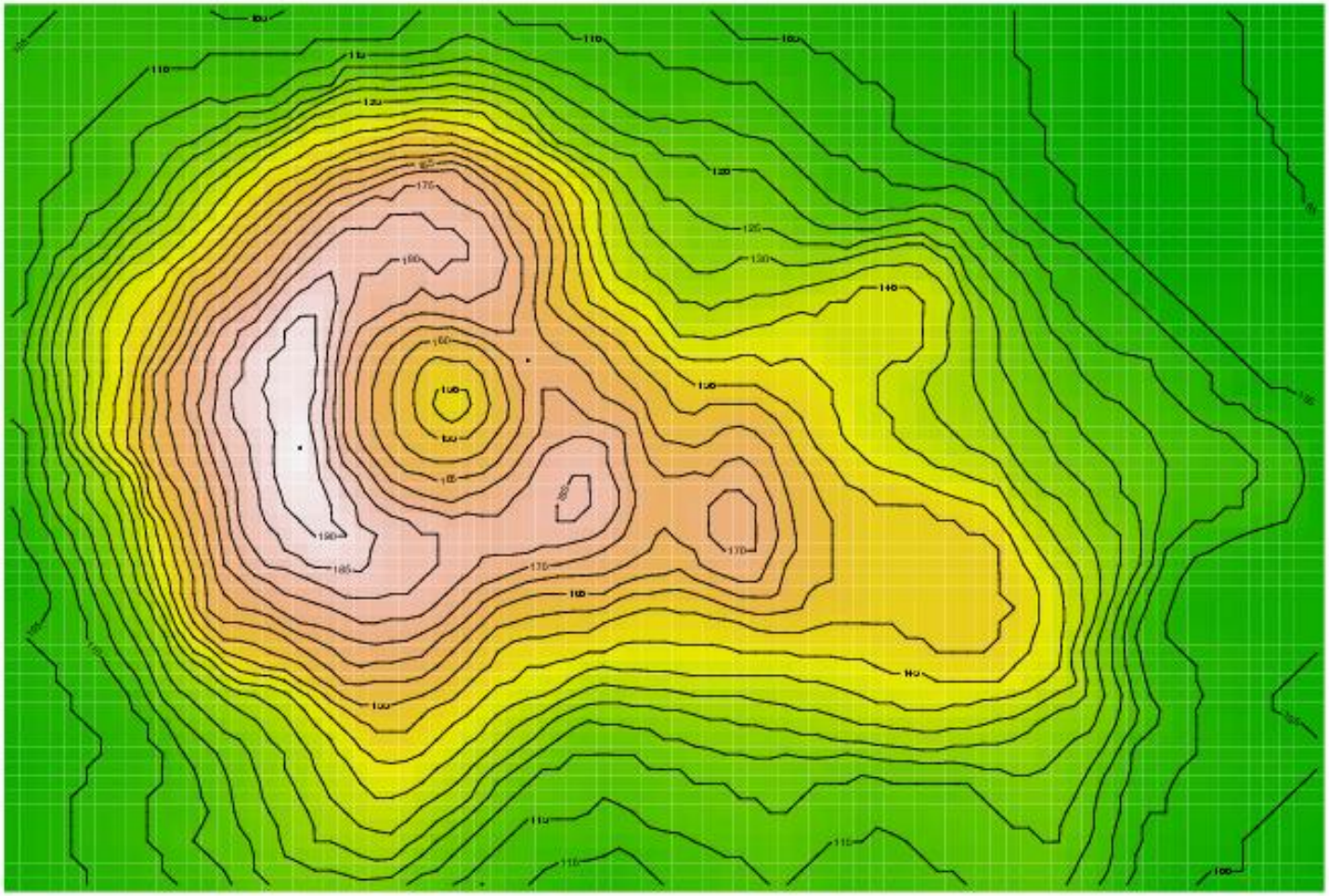
dotchart()

Scatter plot



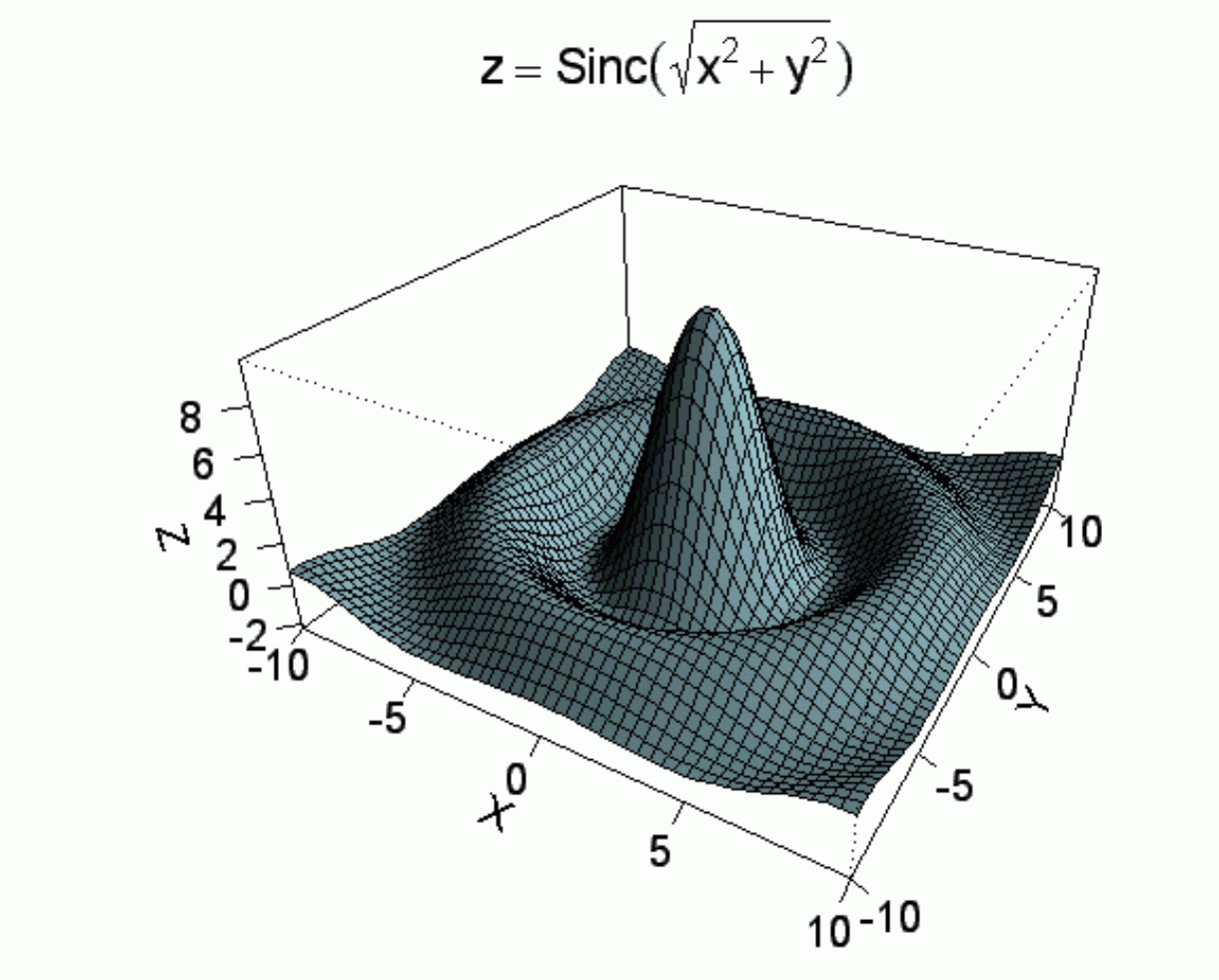
plot()

Contour plot



image(), contour()

3D surface plot



persp()

R exercise 02: basic graphic commands

(1) Execute the commands listed in the handout one by one via the R prompt.
(2) See and understand what happens. In case of functions, carefully read the help pages and browse through them.
(3) Play around with the commands by changing the parameters, choosing different command options, etc.
(4) Export the graphics output of some of the commands to a PDF file, using the function pdf().
(5) Read the help pages of the commands dev.list(), dev.cur(), and dev.off(). Explore them by invoking them before and after the pdf() command.
(6) Read the help pages of the command savePlot(), and try it out.

R exercise 03: strong Fe I lines

(1) Download the file FeI_line.txt from Moodle

(2) Read that file in, using the read.table() function (see chapter 7 of *An Introduction to R* ; see chapter 6 for information on data frames and how to deal with them).

(3) Compute the predicted strengths of the lines, using the formula

$$\log W_\lambda \propto \log gf - \chi$$

(4) Identify the **3 strongest lines** in the line list by **plotting W_λ against λ**, and retrieving the **wavelengths of the lines** with the locator() function. Do you find a more elegant solution?

General remarks on programming exercises

- For each exercise, write all commands that you want to execute into one or more files, and load it/them with source().
- Name your files in a sensible way. Examples:
  Good: verify_central_limit_theorem.R
  Bad: test
  (Note the file extensions.)
- Carefully save and keep all the data files that you produce during the exercises (e.g., the formatted Fe I line list), since you will need most of them again in later exercises.

General remarks on programming exercises

- Write your programs step by step. Start with the core tasks, then increase the functionality and complexity stepwise.
- During each programming step, carefully test your program. Use the print() and cat() functions to produce screen output.
- Include comments in your programs, so that after 10 years you will still be able to understand how your program works.
- I will not immediately help you debugging your programs; you must learn how to do this yourself. (But I will answer general questions about R.)

R exercise 04: data exploration

- Load the library of standard datasets provided in the package “datasets” (see reference manual), with the library() function.
- Inspect and plot some of the datasets.

Helpful functions:

attributes(), mode()
dim(), dimnames()
attach()
length()
summary()

R exercise 05: control statements

Write a simple function that prints out the first *n* prime numbers on the screen, where *n* is the argument of the function (use small values for *n*).

Hints:

- Use nested loops, if statements (see chapter 9 of *An Introduction to R*), and logical operators.
- Use the modulo operator ('%%').