Plotting with R

Sebastian Walter

Seminar talk

03.07.2009

Index (1/2)

Index (2/2)

I. Graphics in R

i. Basic plotting

Elementary functions

Setting up the plot window

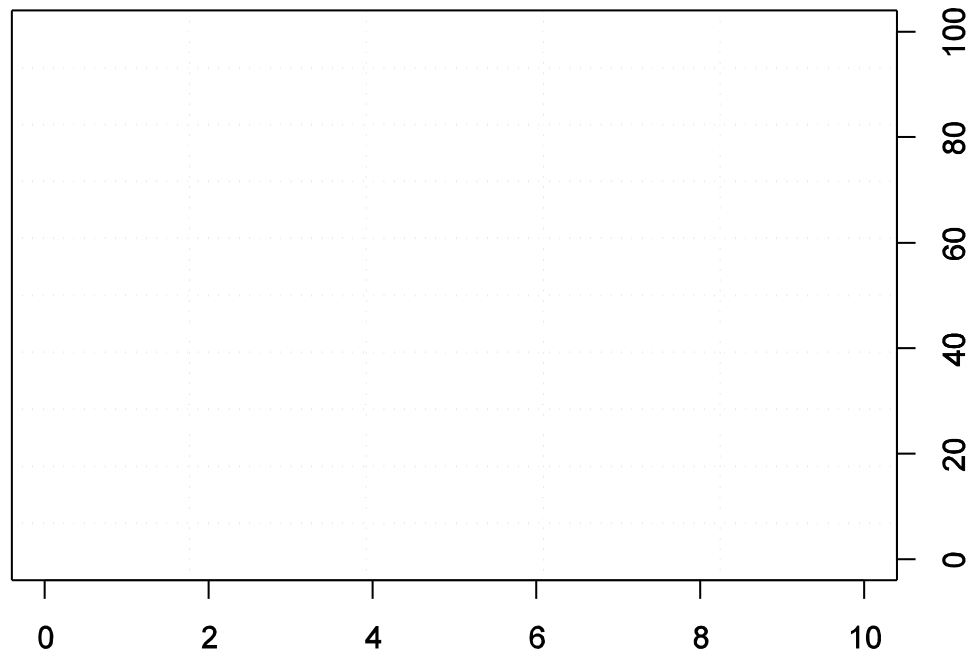
par(mfrow)	Defines the number of plots and their layout per window
plot.new()	Creates a new window
plot.window()	Defines the window‘s limits
axis()	Draws an axis, location specified by a number
grid()	Draws a grid into the window
box()	Draws a box around the window

Setting up the plot window

Code

```
> plot.new()
> plot.window(c(0, 10), c(0, 100))
> axis(1)
> axis(4)
> grid(5, 10)
> box("plot")
```

Setting up the plot window



Labelling the plot

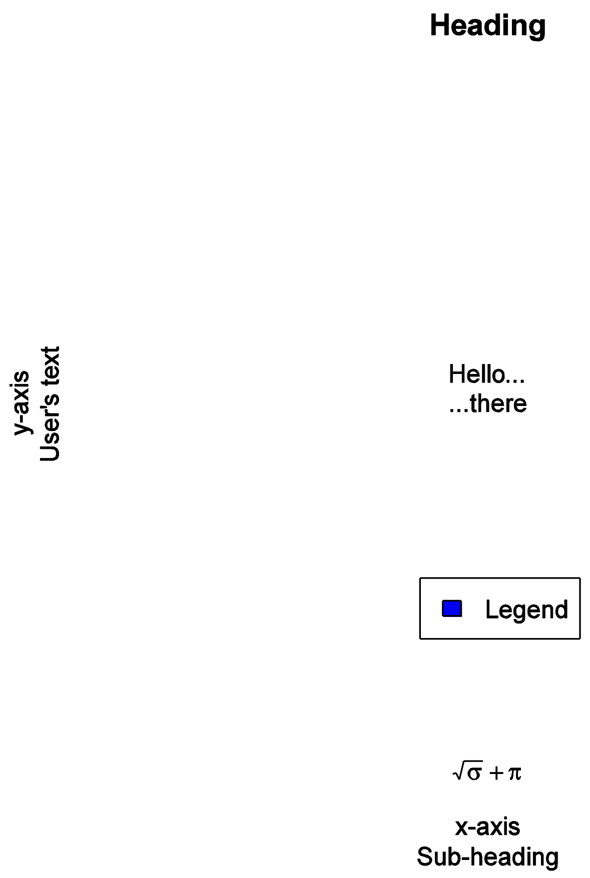
title()	Inserts a heading, axis‘ labels or a subheading at the bottom
mtext()	Inserts a text or expression at the specified marginal line (=> dealing with expressions, try ‚demo(plotmath)‘)
text()	Inserts a text or expression at the specified coordinates
legend()	Inserts a legend at the speciefied coordinates (=> try ‚locator(1)‘ instead of x-, y-coordinates to make live easier)

Labelling the plot

Code

```
> plot.new()
> plot.window(c(0, 10), c(0, 100))
> title(main = "Heading", sub = "Sub-heading", xlab = "x-axis", ylab = "y-axis")
> mtext("User's text", side = "2", line = "2")
> mtext(expression(sqrt(sigma) + pi), side = "1", line = "1")
> text(c(5, 5), c(55, 50), c("Hello...", "...there"))
> legend(4, 20, "Legend", fill = "blue2")
```

Labelling the plot



Elementary low-level plots

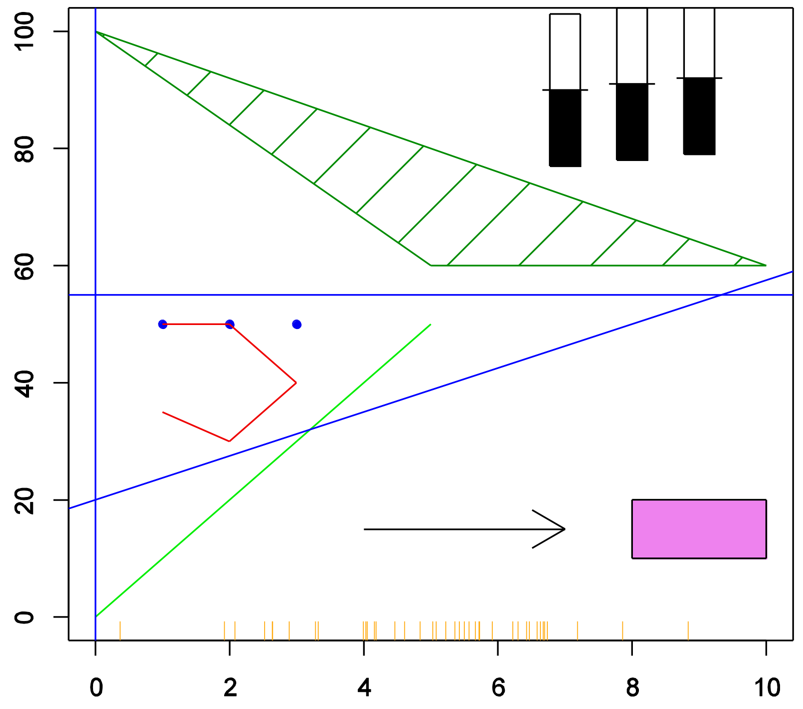
points()	Draws a point or any other ASCII-symbol
segements()	Draws the connection between two points
arrows()	Like ‚segements()' but with an arrowhead
lines()	Draws the connection between several points
rect()	Draws a rectangle, specified by two corners
polygon()	Draws a polygon which may be filled or shaded
abline()	Draws a line specified by ist slope and y-intercept
symbols()	Draws symbols suchs as circles or thermometers
rug()	Inserts a density-projection on an axis, given a data set

Elementary low-level plots

Code

```
> plot.new()
> plot.window(c(0, 10), c(0, 100))
> axis(1)
> axis(2)
> box("plot")
> points(c(1, 2, 3), c(50, 50, 50), pch = 20, col = "blue2")
> segments(0, 0, 5, 50, col = "green2")
> arrows(4, 15, 7, 15, code = 2)
> lines(c(1, 2, 3, 2, 1), c(35, 30, 40, 50, 50), col = "red2")
> rect(8, 10, 10, 20, col = "violet")
> polygon(c(0, 10, 5), c(100, 60, 60), density = 3, angle = 45, col = "green4")
> rug(rnorm(40, 5, 2), col = "orange")
> abline(20, 3.75, col = "blue", h = 55, v = 0)
> symbols(7:9, 90:92, thermometers = t(matrix(c(0.1, 0.5, 0.5), 3, 3)), add = T)
```

Elementary low-level plots



I. Graphics in R

ii. High level plots

Core graphical functions in R

High Level plots (1/3)

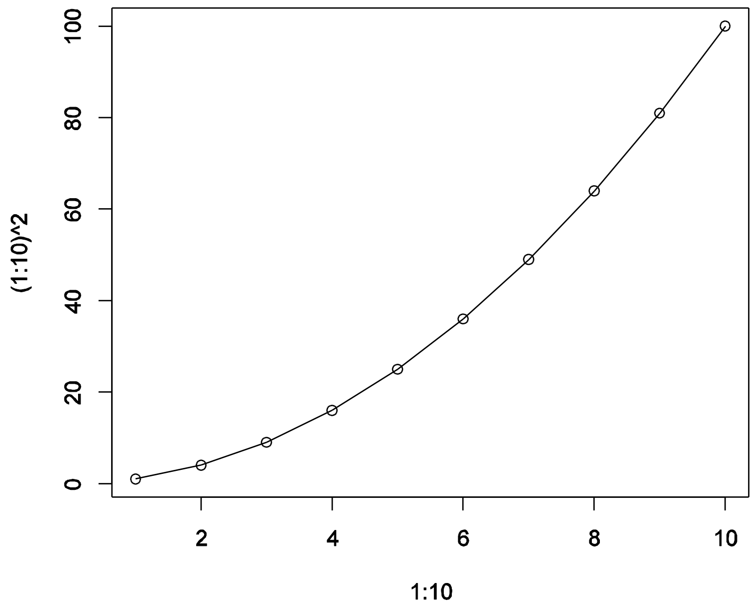
plot()	Basic 2-dimensional scatterplot, symbols can be customised
curve()	Plots a curve from a given expression or function
pie()	Plots a pie-diagram. Radius and appearance are to be set.
barplot()	Plots a 2-dimensional barplot. Several data sets can be plotted
hist()	Plots a 2-dimensional histogram, plots frequency or density
dotchart()	Plots a Cleveland dotchart, substitute for a barplot, supports various grouping options of data

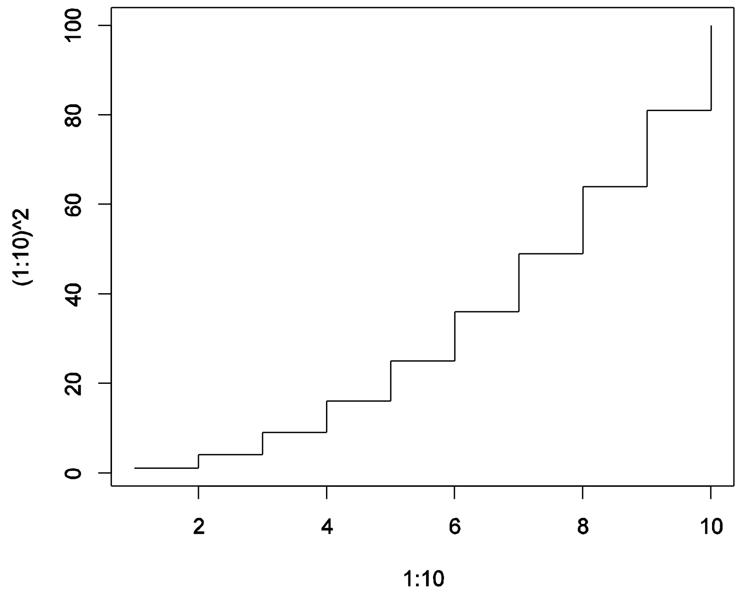
High Level plots (1/3)

Code

```
> par(mfrow = c(2, 1))
> plot(1:10, (1:10)^2, type = "o")
> plot(1:10, (1:10)^2, type = "s")
```

High Level plots (1/3)



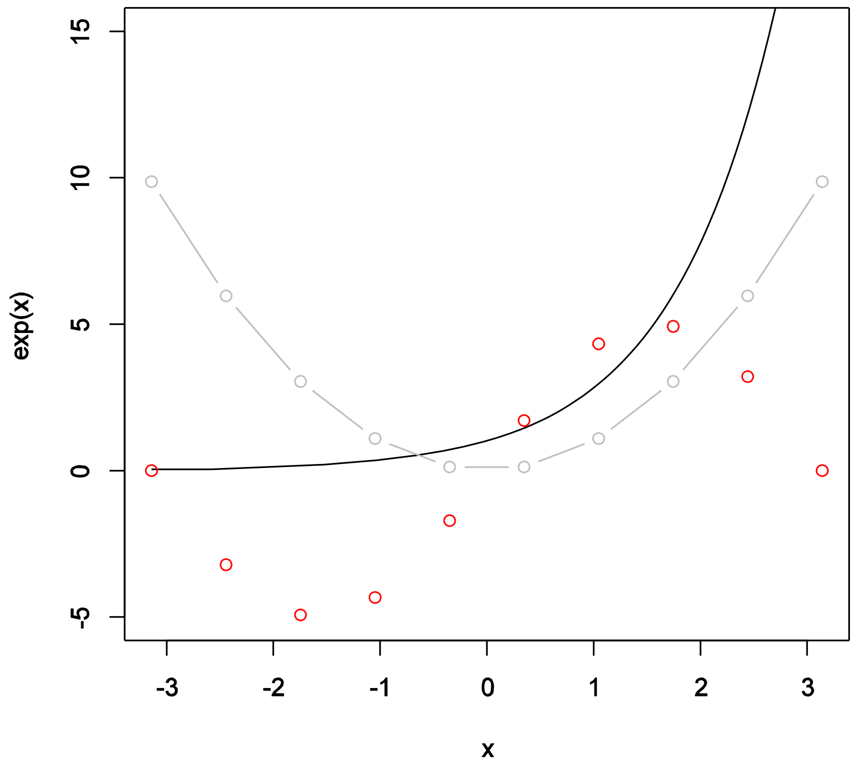


High Level plots (1/3)

Code

```
> curve(exp(x), xlim = c(-pi, pi), ylim = c(-5, 15))
> curve(x^2, add = T, col = "grey", type = "b", n = 10)
> curve(5 * sin(x), add = T, col = "red", type = "p", n = 10)
```

High Level plots (1/3)

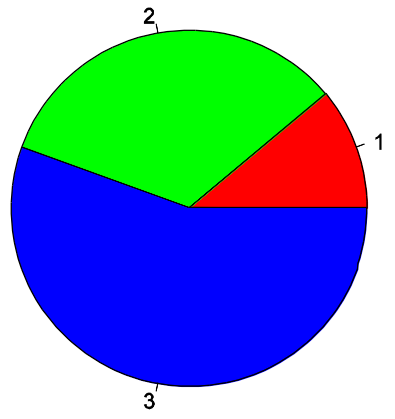


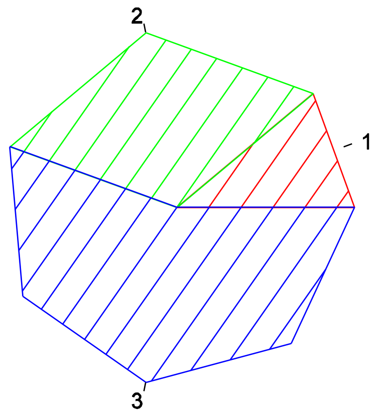
High Level plots (1/3)

Code

```
> par(mfrow = c(2, 1))
> pie(c(1, 3, 5), col = c("red", "green", "blue"), )
> pie(c(1, 3, 5), col = c("red", "green", "blue"), edges=10, density=5, angle=55)
```

High Level plots (1/3)



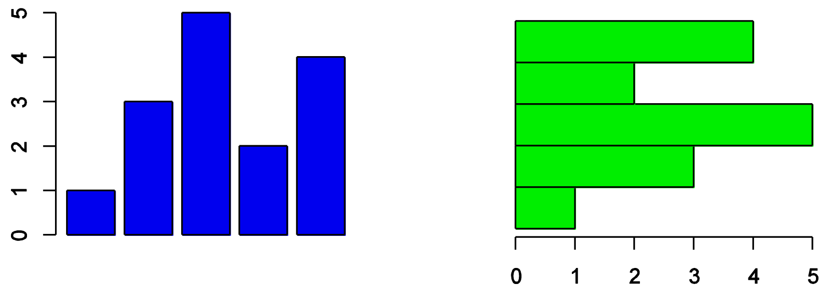


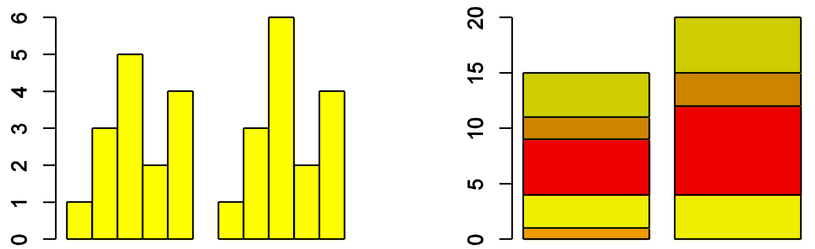
High Level plots (1/3)

Code

```
> barplot(c(1, 3, 5, 2, 4), col = "blue2")
> barplot(c(1, 3, 5, 2, 4), col = "green2", horiz = T, space = 0)
> barplot(cbind(c(1, 3, 5, 2, 4), c(1, 3, 6, 2, 4)), col = "yellow", beside = T)
> barplot(cbind(c(1, 3, 5, 2, 4), c(0, 4, 8, 3, 5)), col = c("orange2", "yellow2",
+ "red2", "orange3", "yellow3"), beside = F)
```

High Level plots (1/3)



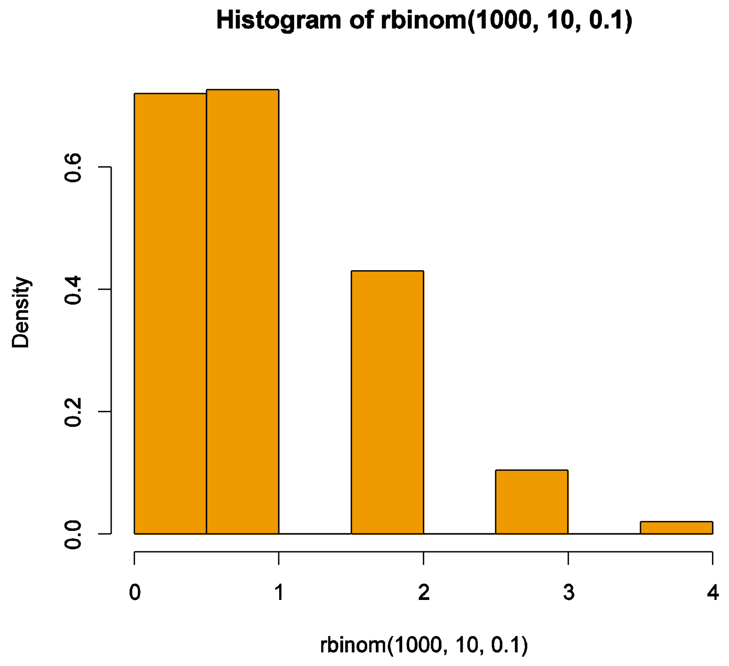


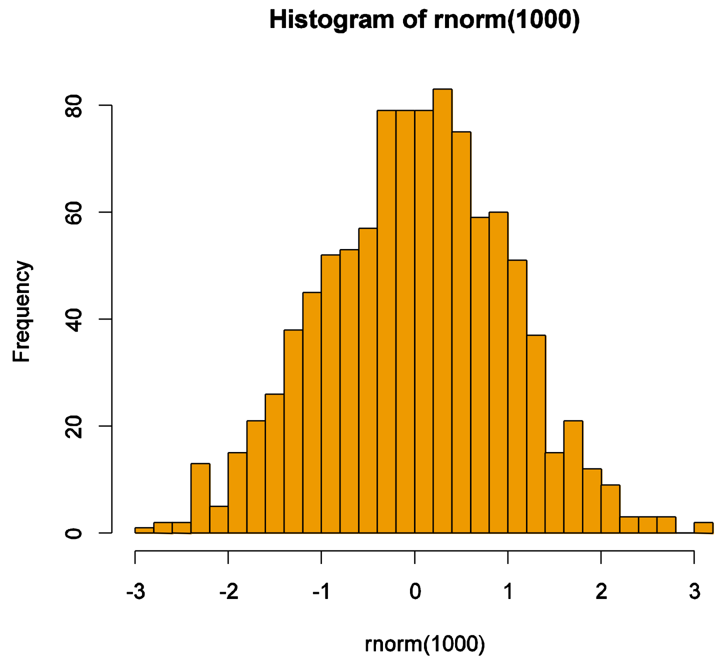
High Level plots (1/3)

Code

```
> par(mfrow = c(2, 1))
> hist(rbinom(1000, 10, 0.1), freq = F, col = "orange2")
> hist(rnorm(1000), breaks = 40, freq = T, col = "orange2")
```

High Level plots (1/3)



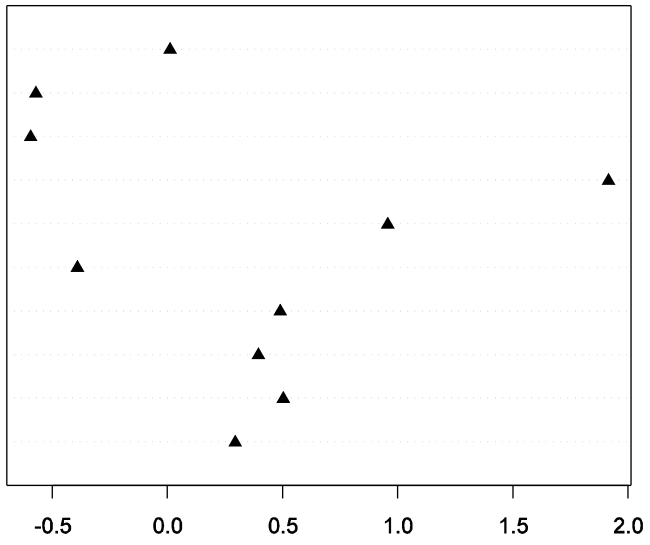


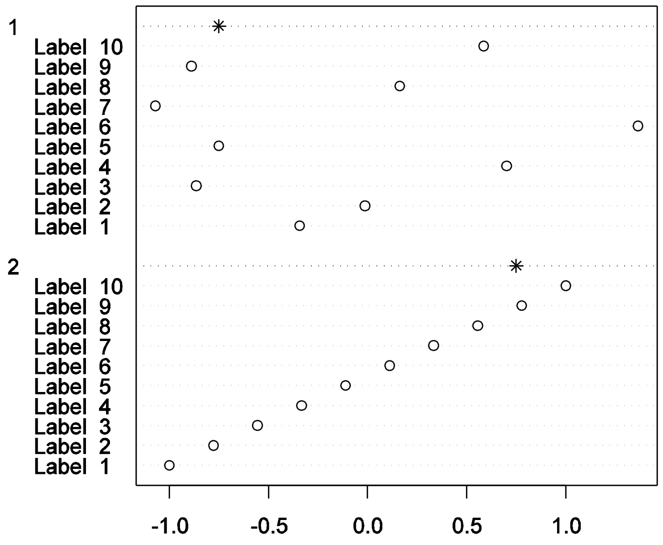
High Level plots (1/3)

Code

```
> par(mfrow = c(2, 1))
> dotchart(rnorm(10), pch = 17)
> dotchart(cbind(rnorm(10), seq(-1, 1, length = 10)), labels = c(paste("Label ",
+ 1:10)), gdata = c(-0.75, 0.75), gpch = 8)
```

High Level plots (1/3)





High level plots (2/3)

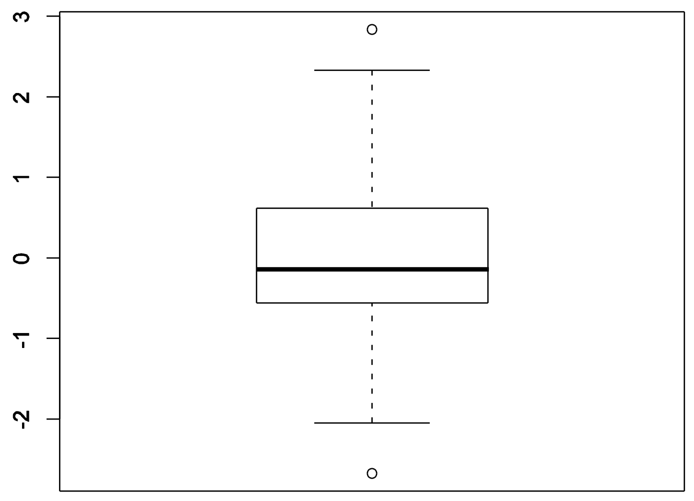
boxplot()	Plots a boxplot. Can compare multiple datasets in one window
pairs()	Extended scatterplot, good for more complex data
coplopt()	Plots a conditioning plot (scatterplot depending on a variable)
stripchart()	Plots a stripchart, substitute for a boxplot for discrete data
mosaicplot()	Plots a mosaicplot, multidimensional contigency tables. Usefull to spot dependencies, several graphical options
stars()	Plots stars or so called ‚radar diagrams‘, substitutes pie plot.

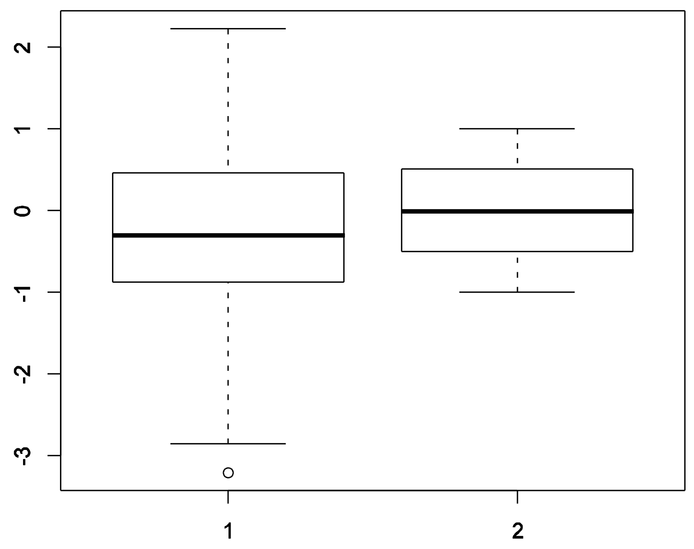
High level plots (2/3)

Code

```
> par(mfrow = c(2, 1))
> boxplot(rnorm(100))
> boxplot(cbind(rnorm(100), seq(-1, 1, length = 100)))
```

High level plots (2/3)



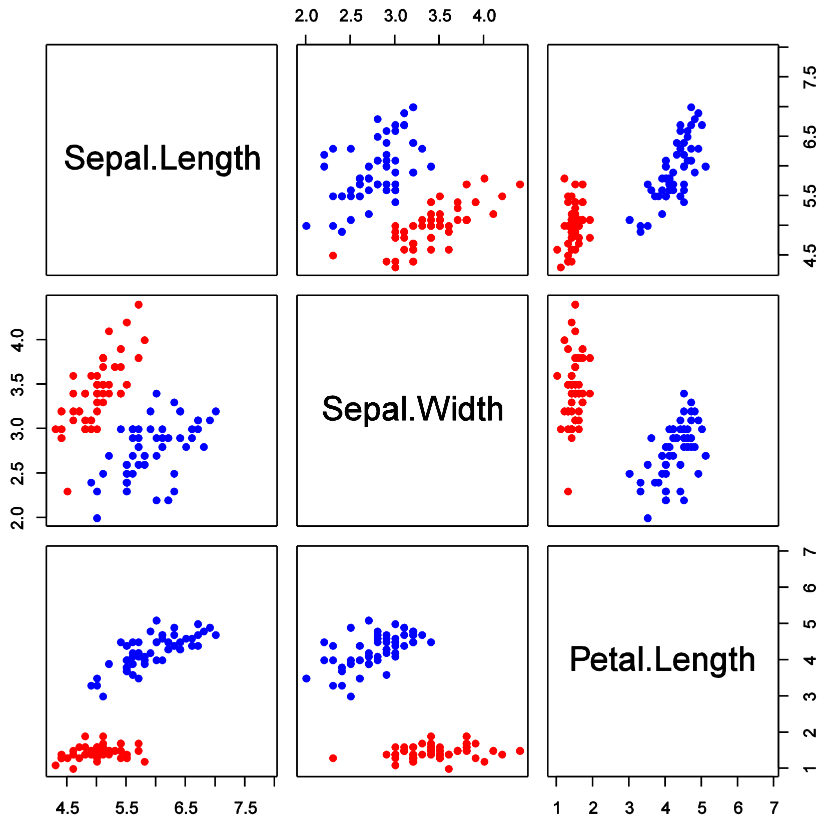


High level plots (2/3)

Code

```
> pairs(iris[1:3], col = c("red", "blue")[unclass(iris$Species)], pch = 16)
```

High level plots (2/3)

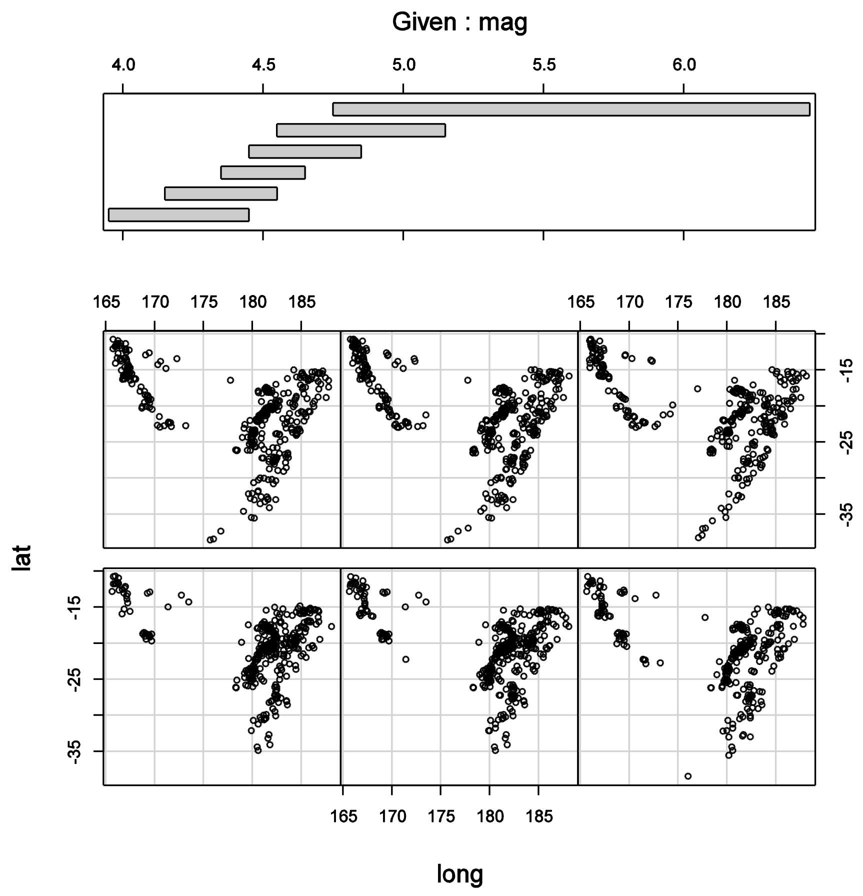


High level plots (2/3)

Code: [Source: R documentation]

```
> coplot(lat ~ long | mag, data = quakes, cex = 0.75)
```

High level plots (2/3)

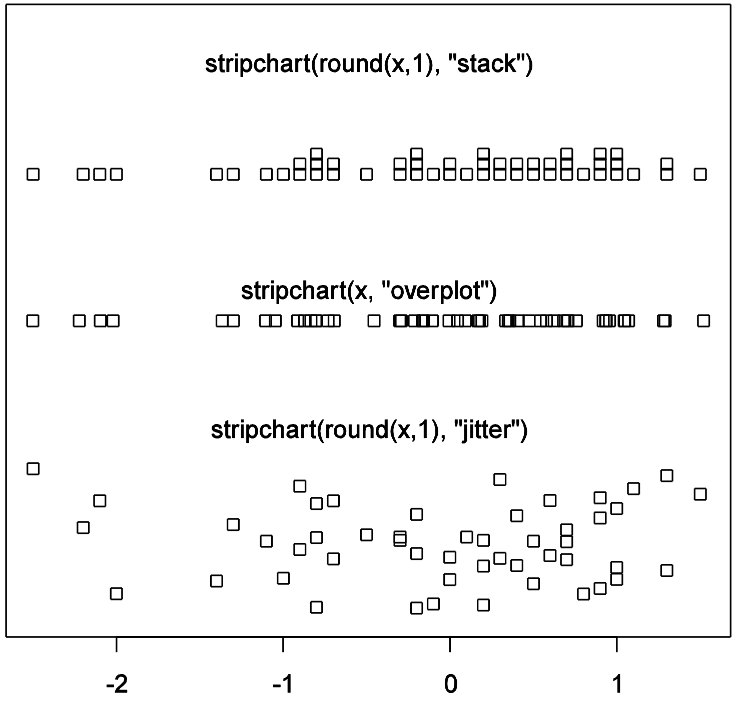


High level plots (2/3)

Code: [Source: R documentation]

```
> x <- stats::rnorm(50)
> xr <- round(x, 1)
> stripchart(x)
> m <- mean(par("usr")[1:2])
> text(m, 1.04, "stripchart(x, \"overplot\")")
> stripchart(xr, method = "stack", add = TRUE, at = 1.2)
> text(m, 1.35, "stripchart(round(x,1), \"stack\")")
> stripchart(xr, method = "jitter", add = TRUE, at = 0.7)
> text(m, 0.85, "stripchart(round(x,1), \"jitter\")")
```

High level plots (2/3)

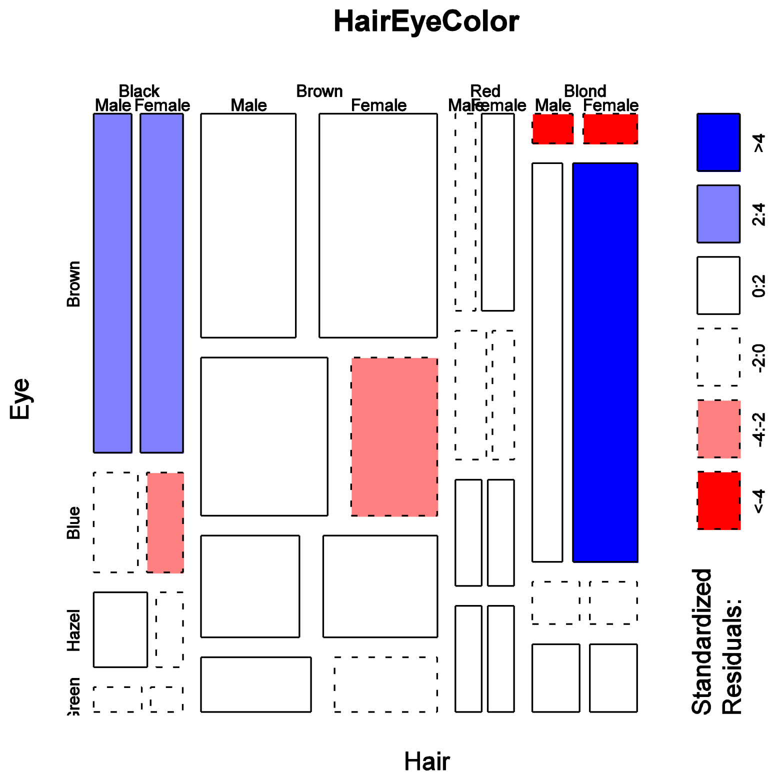


High level plots (2/3)

Code: [Source: R documentation]

```
> data(HairEyeColor)
> mosaicplot(HairEyeColor, shade = TRUE)
```

High level plots (2/3)

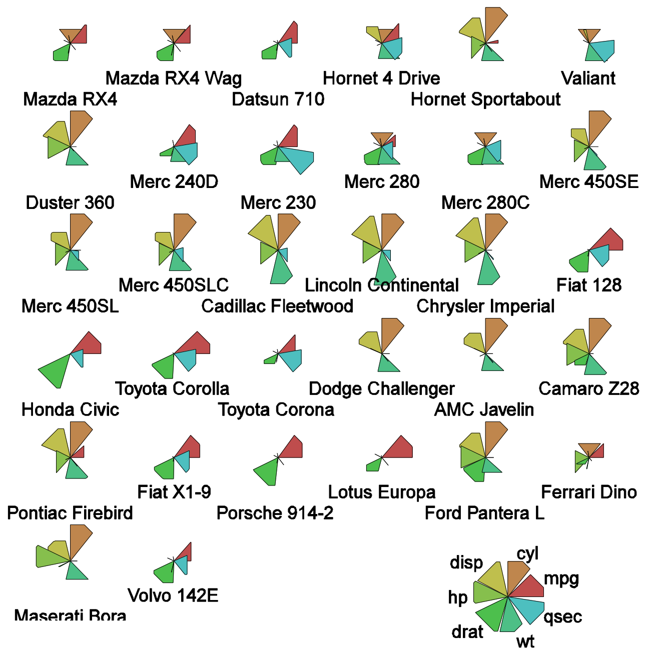


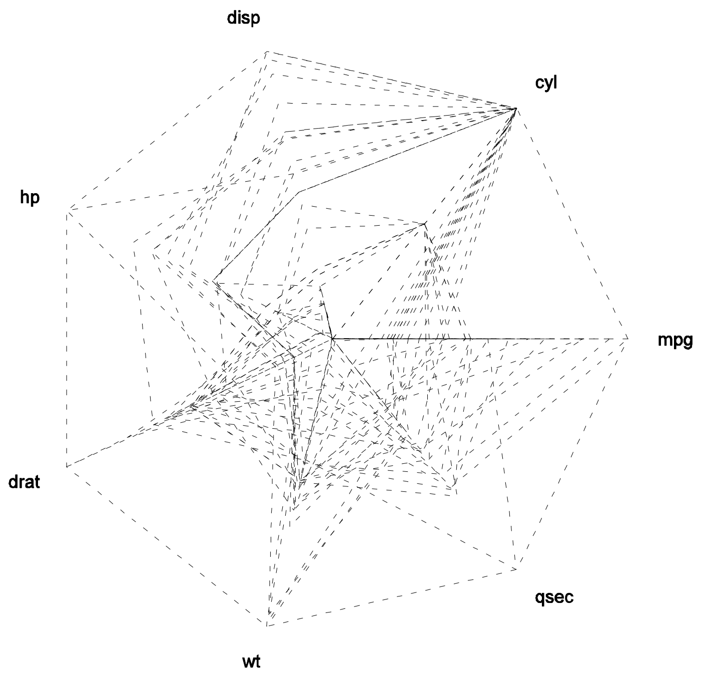
High level plots (2/3)

Code: [Source: R documentation]

```
> par(mfrow = c(2, 1))
> palette(rainbow(12, s = 0.6, v = 0.75))
> stars(mtcars[, 1:7], len = 0.8, key.loc = c(12, 1.5), draw.segments = TRUE)
> stars(mtcars[, 1:7], locations = c(0, 0), radius = FALSE, key.loc = c(0, 0),
+ lty = 2)
```

High level plots (2/3)





High level plots (3/3)

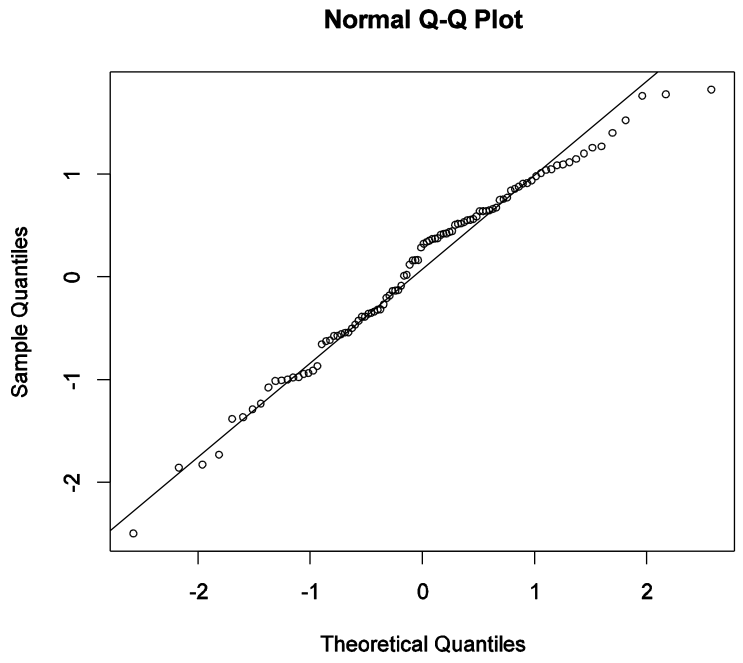
qqnorm()	Plots a qq-plot that compares quantiles to the normal-distrib. (the ‚qq.plot' function from ‚car' package can compare to any other distrib.)
qqline()	Inserts a line though 1st and 3rd quantile (low level)
qqplot()	Analog to qqnorm, compares to any other data set
contour()	Plots a contour plot, visualises the level curves of a data set.
image()	Plots a contour plot in a ‚heat map' style
persp()	Plots a surface on the x-y plane, only built in 3D-function, can be rotated via ‚theta' and ‚phi' parameters

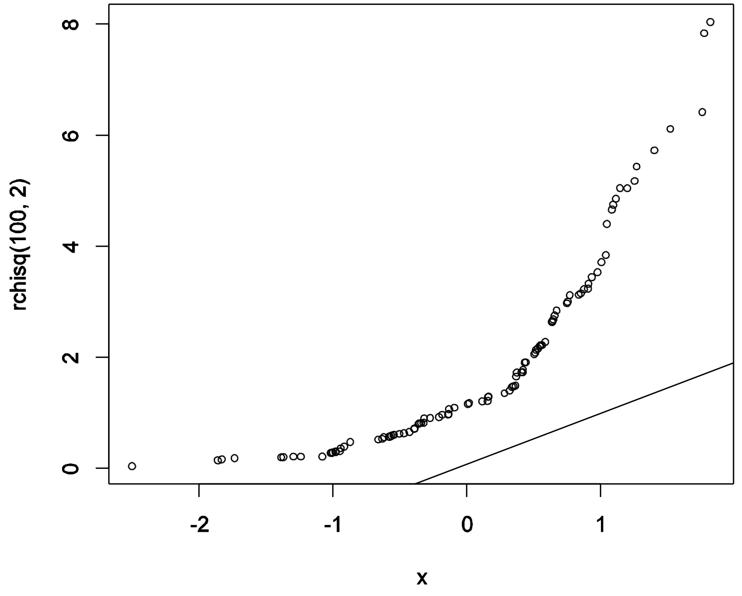
High level plots (3/3)

Code

```
> par(mfrow = c(2, 1))
> qqnorm(x <- rnorm(100), cex = 0.7)
> qqline(x)
> qqplot(x, rchisq(100, 2), cex = 0.7)
> qqline(x)
```

High level plots (3/3)



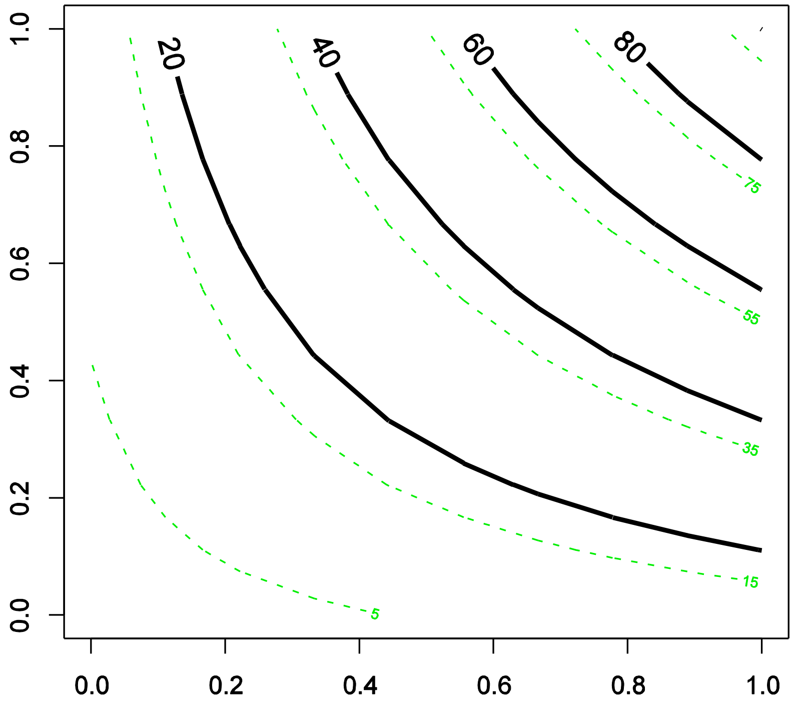


High level plots (3/3)

Code

```
> z <- (matrix(c(1:10, 2 * 1:10, 3 * 1:10, 4 * 1:10, 5 * 1:10, 6 * 1:10, 7 * 1:10,
+ 8 * 1:10, 9 * 1:10, 10 * 1:10), 10, 10))
> contour(z, nlevels = 4, labcex = 1.2, method = "edge", lwd = 3)
> contour(z, nlevels = 4, levels = c(5, 15, 35, 55, 75, 95), col = "green2", add =
T, lty = 2)
```

High level plots (3/3)

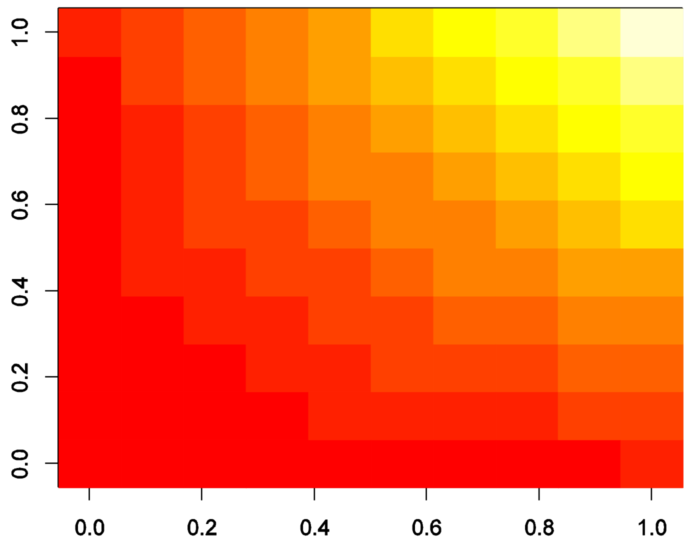


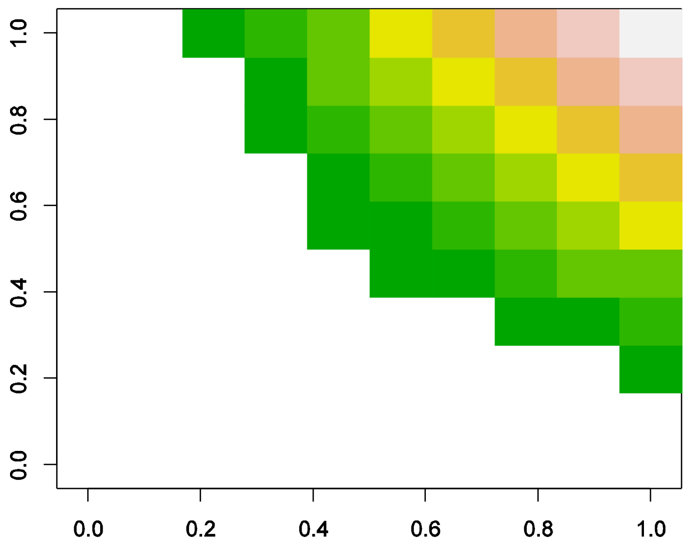
High level plots (3/3)

Code

```
> par(mfrow = c(2, 1))
> z <- (matrix(c(1:10, 2 * 1:10, 3 * 1:10, 4 * 1:10, 5 * 1:10, 6 * 1:10, 7 * 1:10,
+ 8 * 1:10, 9 * 1:10, 10 * 1:10), 10, 10))
> image(z)
> image(z, col = terrain.colors(10), zlim = c(30, 100))
```

High level plots (3/3)



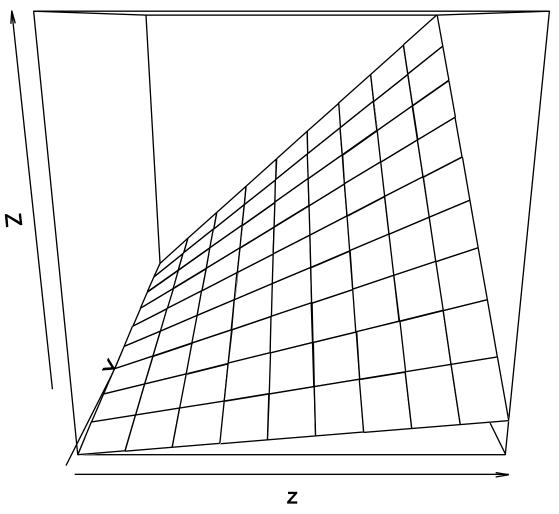


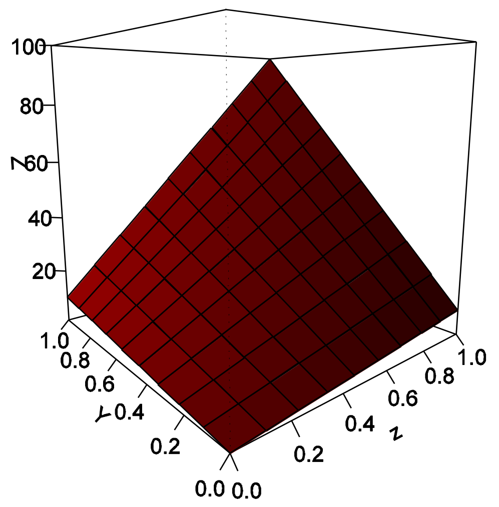
High level plots (3/3)

Code

```
> par(mfrow = c(2, 1))
> z <- (matrix(c(1:10, 2 * 1:10, 3 * 1:10, 4 * 1:10, 5 * 1:10, 6 * 1:10, 7 * 1:10,
+ 8 * 1:10, 9 * 1:10, 10 * 1:10), 10, 10))
> persp(z)
> persp(z, theta = -40, phi = 10, col = "red", shade = 1.5, ticktype = "detailed")
```

High level plots (3/3)





I. Graphics in R

iii. Common parameters

‚Mother‘s little helper‘

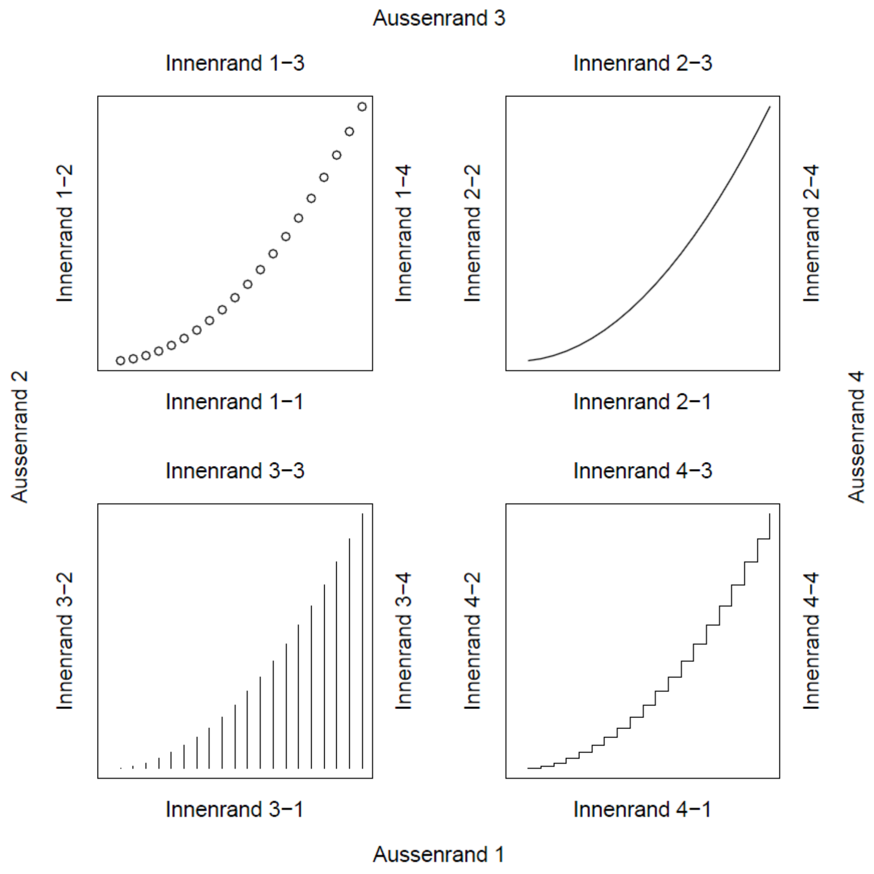
Functional parameters (1/2)

axes	Determines whether axes are drawn or not
las	Determines alignment of axes‘ labels
main / sub	Inserts a sub-/ heading
xlab / ylab	Inserts the x- and y-axis‘ labels
xlim / ylim	Sets the x- and y-axis‘ domain of definition (2-dim. vectors)
bg	Sets the background colour

Functional parameters (2/2)

font	Sets fonttype such as bold or italic. (There are sub.paramters such as font.lab, font.axis, font.main…)
lab	Sets the number of tickmarks on the axes. (Note that it needs a 3-dim. Vector input but the 3rd component is ignored.)
add	Surpresses the plot.new() call of a high level function. Causes the plot to be drawn in the same window.
oma, omi, omd	Controls the size of outer margins.
xlog / ylog	Sets the x- or y-axis to logarithmic scale

Margins of a plot window [Source: Alexander Bauer]



Graphical parameters

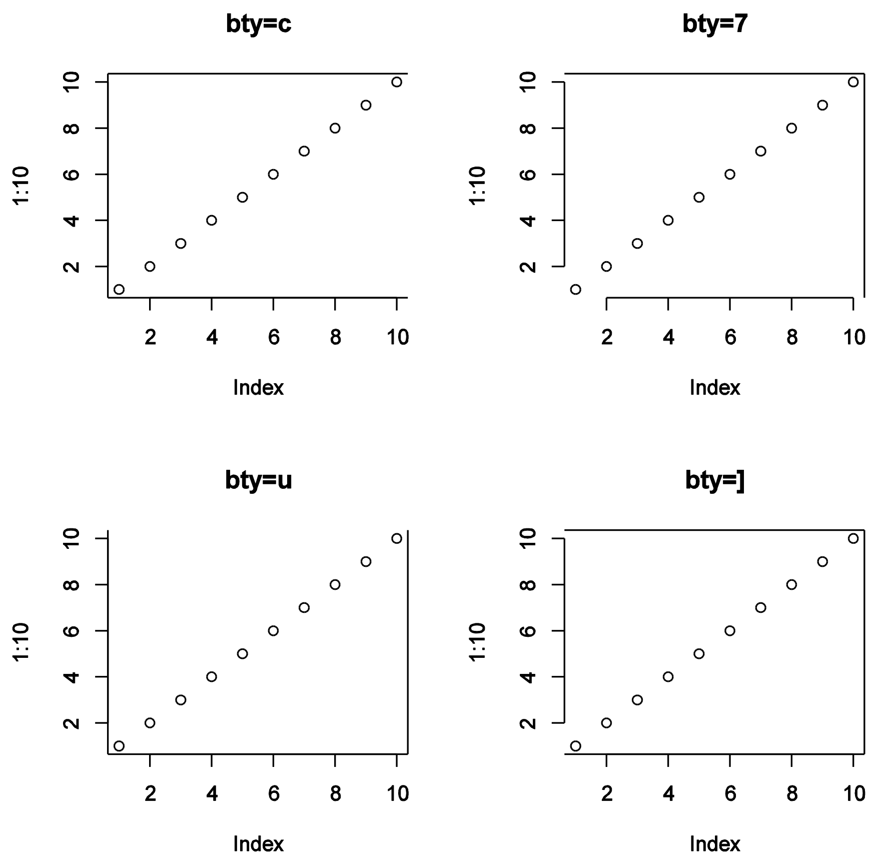
col	Sets colour of curves, symbols, … Vector input possible.
bty	Sets the style of the boundary box (U-shaped, L-shaped,...)
lwd	Sets the line width.
lty	Sets the line type, such as points, lines, both,…
pch	Sets the symbol. Numeric value, Vector input possible. 1-25: Standard R symbols 32-127: ASCII characters NA: blank
cex	Sets the symbols‘ size. Vector input possible.

Graphical parameters

Code

```
> par(mfrow = c(2, 2))
> plot(1:10, bty = "c", main = "bty=c")
> plot(1:10, bty = "7", main = "bty=7")
> plot(1:10, bty = "u", main = "bty=u")
> plot(1:10, bty = "]", main = "bty=]")
```

Graphical parameters

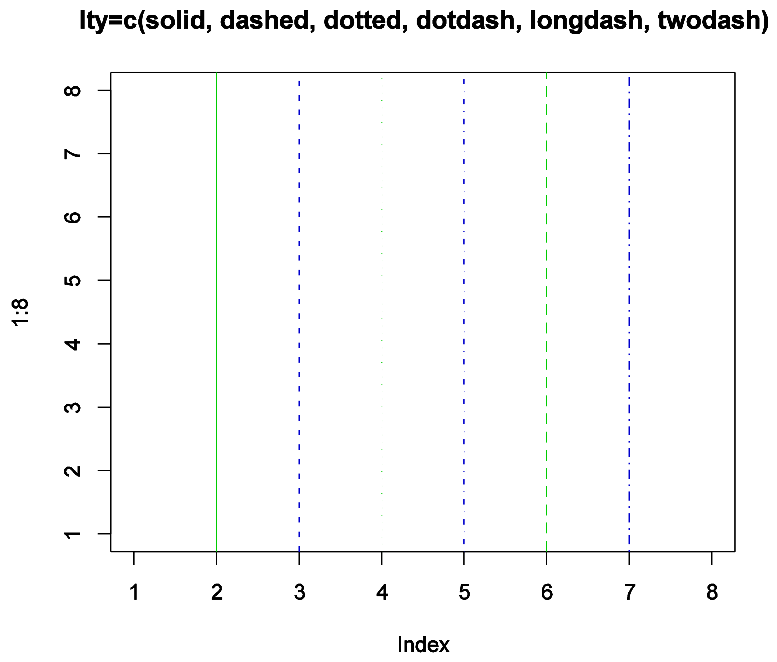


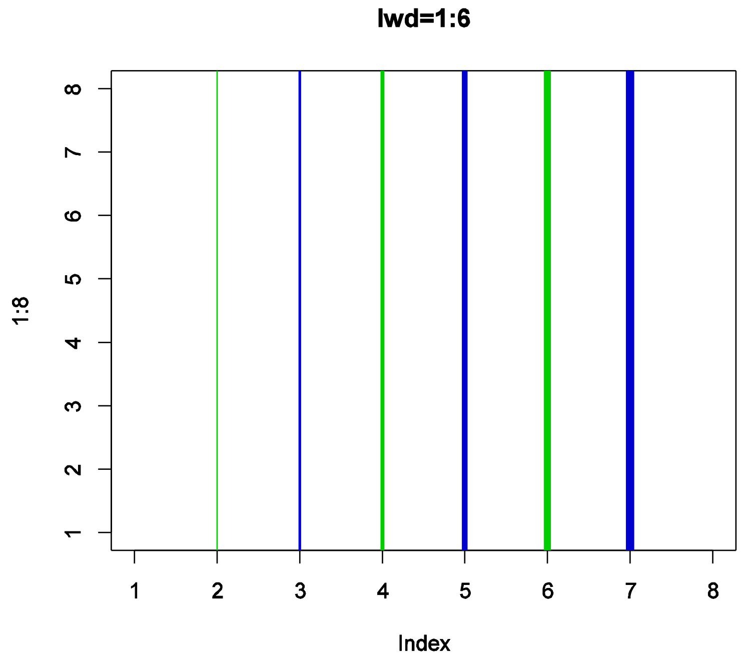
Graphical parameters

Code

```
> par(mfrow = c(2, 1))
> plot(1:8, cex = 0, main = "lty=c(solid, dashed, dotted, dotdash, longdash,
twodash)")
> abline(v = 2:7, lty = 1:6, col = c("green3", "blue3"))
> plot(1:8, cex = 0, main = "lwd=1:6")
> abline(v = 2:7, lwd = 1:6, col = c("green3", "blue3"))
```

Graphical parameters



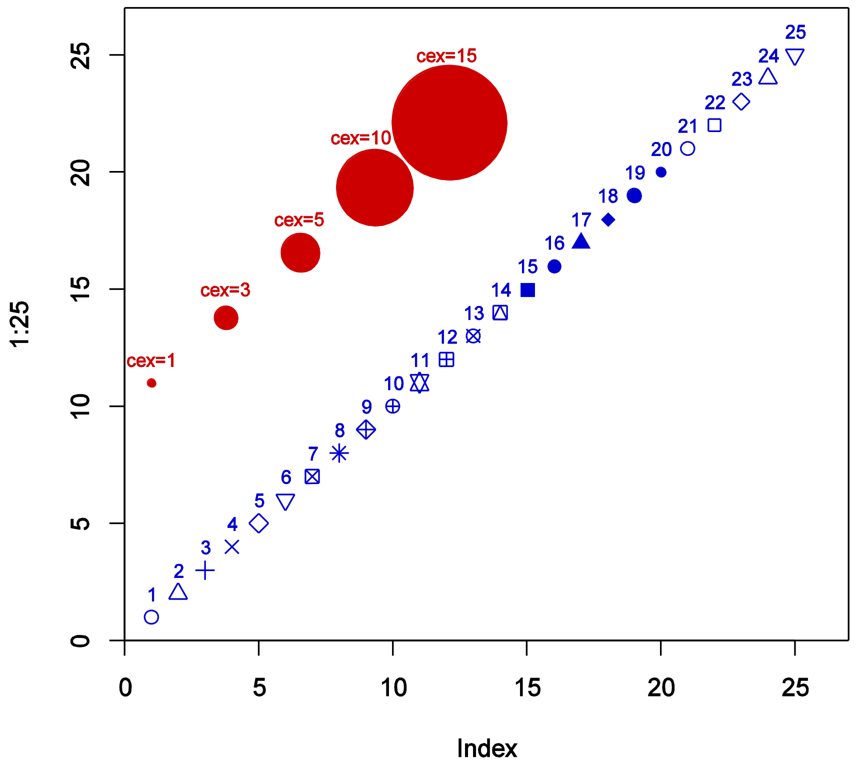


Graphical parameters

Code

```
> plot(1:25, xlim = c(1, 26), ylim = c(1, 26), pch = 1:25, cex = 1.2, col =
"blue3")
> text(1:25, 2:26, cex = 0.75, col = "blue3")
> curve(x + 10, add = T, type = "p", n = 10, col = "red3", cex = c(1, 3, 5, 10,
+ 15, 0, 0), pch = 20)
> text(c(1, 3.75, 6.5, 8.8, 12), c(12, 15, 18, 21.5, 25), cex = 0.75, col = "red3",
+ label = c("cex=1", "cex=3", "cex=5", "cex=10", "cex=15"))
```

Graphical parameters



Colours (1/2)

colors()	Returns a list of prebuilt colours in R. Note: Many standard colours come in shaded variations, e.g. „red2“, „red3“,…
rgb()	Creates a colour from the RGB scheme. (red, green, blue, intesity, alpha-value (transparency))
hsv()	Creates a colour from the HSV scheme. (hue, saturation, value, alpha-value (transparency))
hcl()	Creates a colour from the HCL scheme. (hue, chroma, luminance, alpha-value (transparency))
gray()	Creates a shade of gray from a level

Colours (2/2)

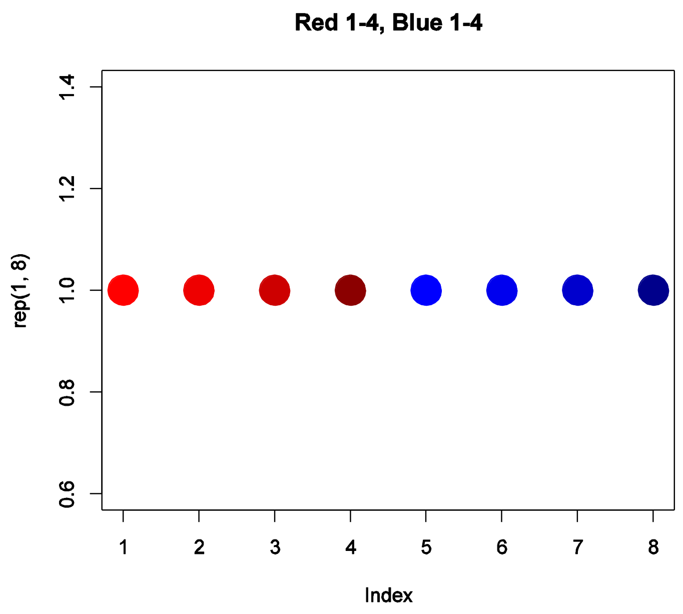
rainbow()	Creates a gradient of colour in a rainbow-style.
diverge_hcl()	Creates a gradient of colour from two hcl-colours.
heat.colors()	Creates a gradient of colour in a ‘heatmap’-style
terrain.colors()	Creates a gradient of colour in green-brown style
topo.colors()	Creates a gradient of colour in green-brown-blue style.
cm.colors()	Creates a gradient of colour from light blue to pink.

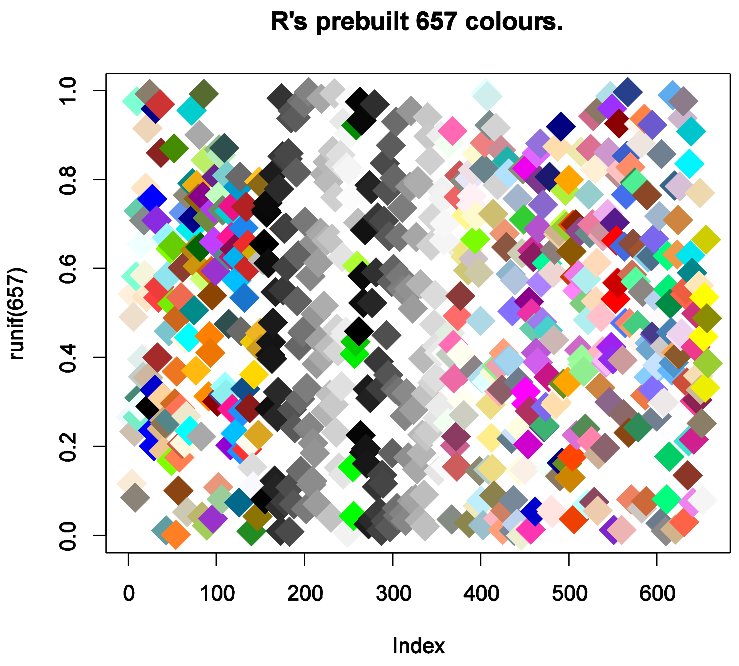
Colours

Code

```
> par(mfrow = c(2, 1))
> plot(rep(1, 8), col = c(paste("red", 1:4), paste("blue", 1:4)), pch = 20, cex = 5,
+ main = "Red 1-4, Blue 1-4")
> plot(runif(657), col = colors(), pch = 18, cex = 3)
```

Colours


Red 1-4, Blue 1-4
rep(1, 8)
Index
0.6
0.8
1.0
1.2
1.4
1
2
3
4
5
6
7
8

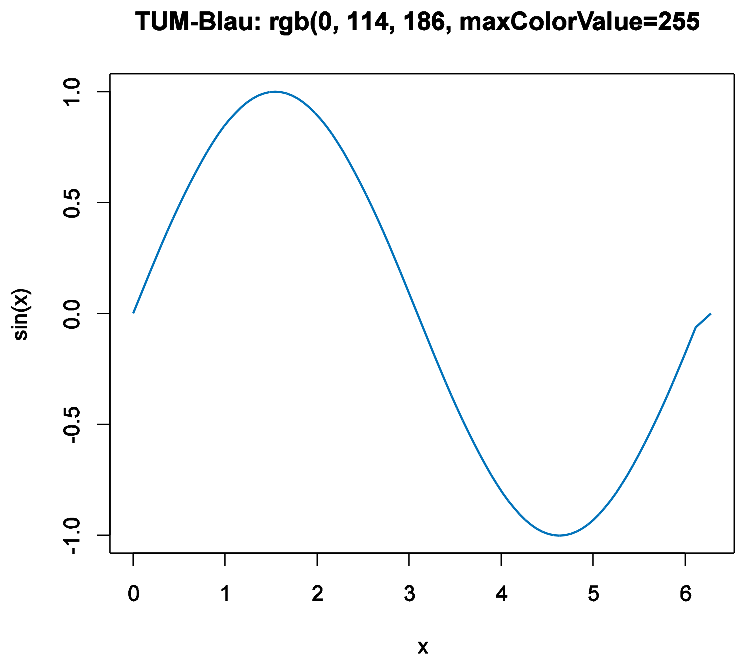

R's prebuilt 657 colours.
runif(657)
Index
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500
600

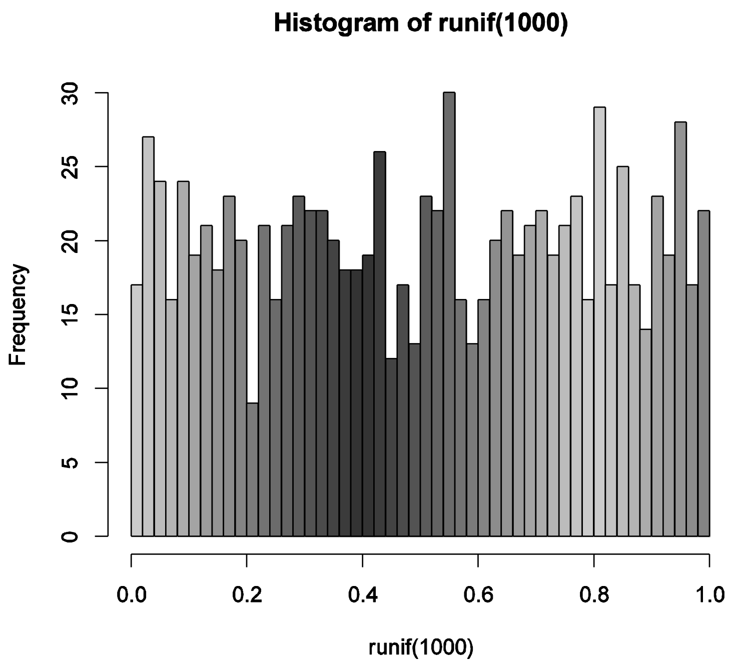
Colours

Code

```
> par(mfrow = c(2, 1))
> blau <- rgb(0, 114, 186, maxColorValue = 255)
> curve(sin(x), xlim = c(0, 2 * pi), lwd = 1.5, col = blau, main = "TUM-Blau:
rgb(0, 114, 186, maxCol=255)“
> hist(runif(1000), breaks = 40, col = gray(c(seq(0.8, + 0.2, length = 20),
seq(0.2, 0.8, length = 20))))
```

Colours


TUM-Blau: rgb(0, 114, 186, maxColorValue=255
sin(x)
x
1.0
0.5
0.0
-0.5
-1.0
0
1
2
3
4
5
6

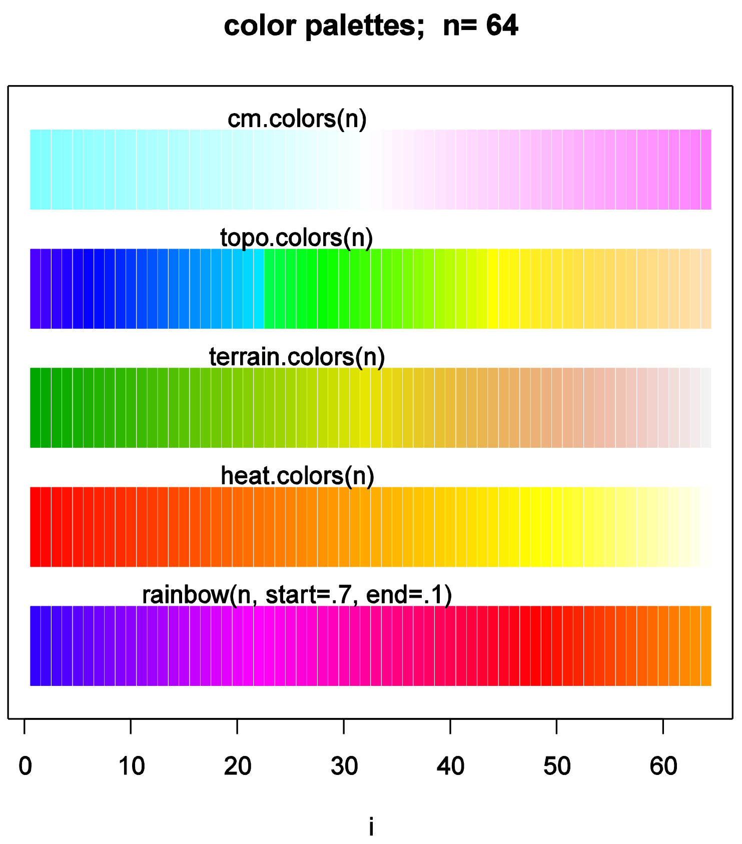

Histogram of runif(1000)
Frequency
runif(1000)
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0

Colours

Code: [Source: R documentation]

```
> demo.pal <- function(n, border = if (n < 32) "light gray" else NA, main = paste("color palettes; n+ n),
ch.col = c("rainbow(n, start=.7, end=.1)", "heat.colors(n)", "terrain.colors(n)",
+ "topo.colors(n)", "cm.colors(n)")) {
+ nt <- length(ch.col)
+ i <- 1:n
+ j <- n/nt
+ d <- j/6
+ dy <- 2 * d
+ plot(i, i + d, type = "n", yaxt = "n", ylab = "", main = main)
+ for (k in 1:nt) {
+ rect(i - 0.5, (k - 1) * j + dy, i + 0.4, k * j, col = eval(parse(text = ch.col[k])),
+ border = border)
+ text(2 * j, k * j + dy/4, ch.col[k])
+ }
+ }
> n <- if (.Device == "postscript") 64 else 16
> demo.pal(n)
```

Colours


color palettes; n= 64
cm.colors(n)
topo.colors(n)
terrain.colors(n)
heat.colors(n)
rainbow(n, start=.7, end=.1)
0
10
20
30
40
50
60
i

I. Graphics in R

i. Basic plotting
ii. High level plots
iii. Common parameters
iv. Additional packages

iv. Additional packages

This is where the fun begins…

Additional packages

- Installation via ‚utils:::menuInstallPkgs()‘ or windows setup
- CRAN, R-project homepage:
  http://cran.r-project.org
- Overview of graphical packages
  http://addictedtor.free.fr/graphiques
- Another site dedicated to graphics in R
  http://bm2.genes.nig.ac.jp/RGM2/
- Good overview of some packages with examples.
  http://www.stat.ucl.ac.be/ISpersonnel/lecoutre/stats/chiers/_gallery.pdf

Additional packages

- Some important packages
  - Lattice
  - Vcd
  - Rgraphivz

Lattice package

- Quite popular
- Comes with an own plot device, started with 'trellis.device(),
- Powerful tool for multiple plots in one window and complex data
- Most function are called with a formular like , x   y | z‘
- Re-does some built in function…
- …but provides new functions too
- Lattice / Trellis homepage:

  ttp://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/index.html

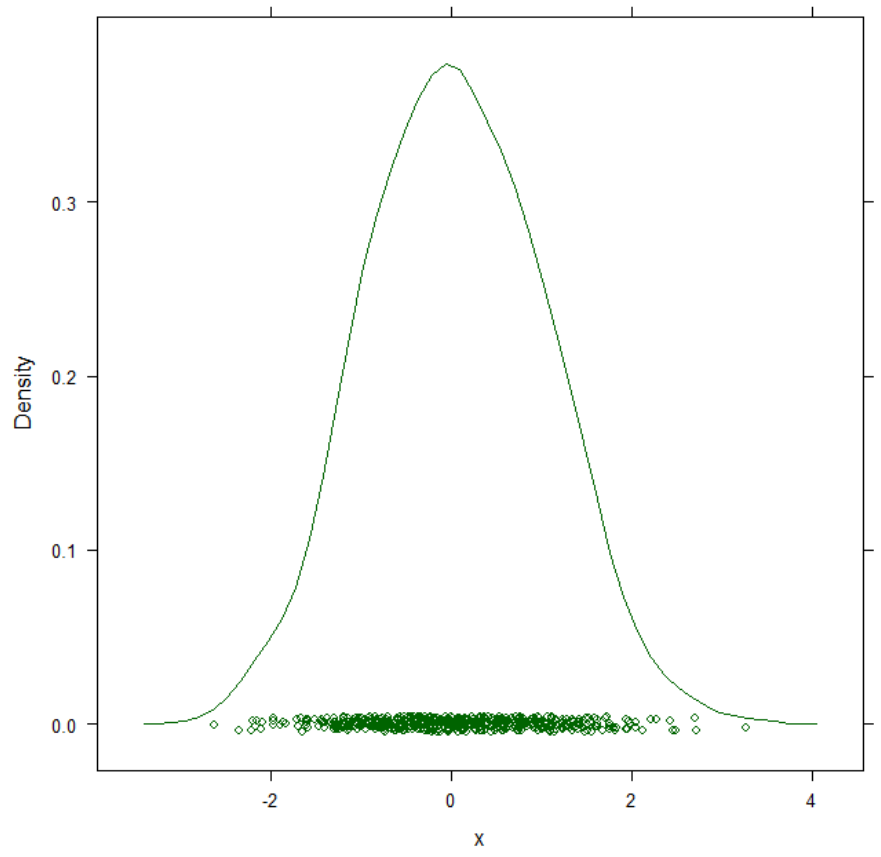
Lattice package functions (1/2)

barchart()	barplot()	Barplot
bwplot()	boxplot()	Boxplot
densityplot()	- / rug() + hist()	Estimates density.
dotplot()	dotchart()	Dotplot
histogram()	hist()	Histogram
qqmath()	qqnorm()	QQ-plot: Data vs. distribution
qq()	qqplot()	QQ-plot: Data vs. data
stripplot()	stripchart()	1-dim. Scatterplot

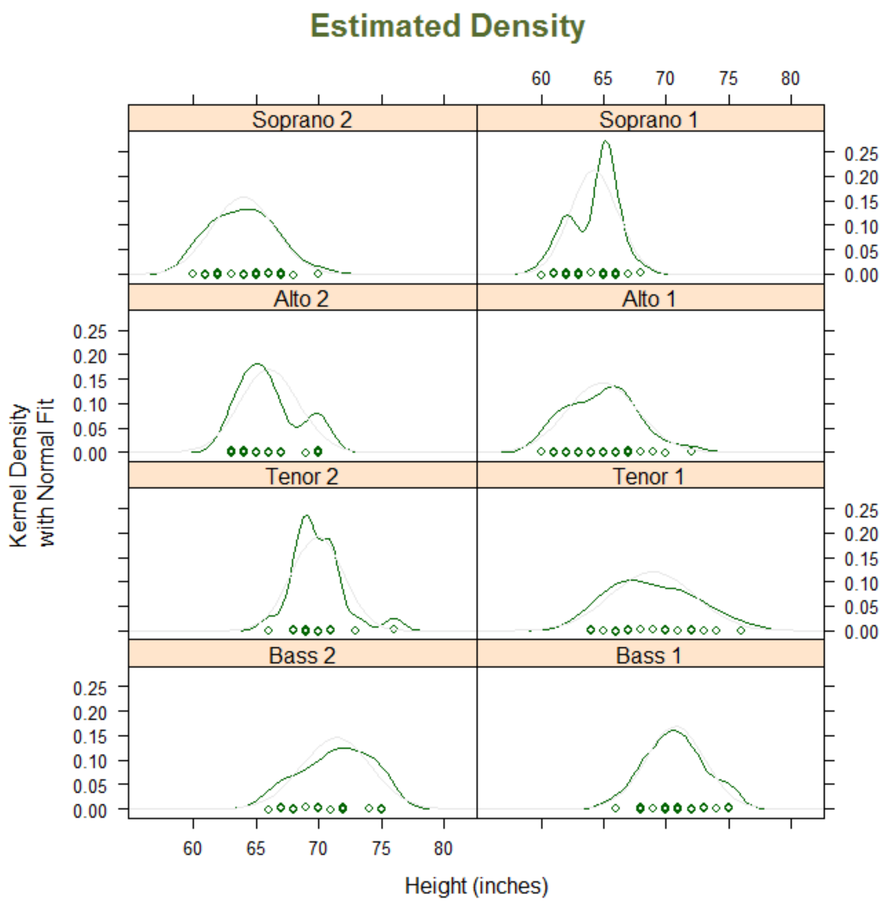
Lattice package functions (2/2)

xyplot()	plot()	2-dim. Scatterplot
contourplot()	contour()	Countourplot
levelplot()	image()	Sort of discrete countourplot
cloud()	-	3-dim. Scatterplot
wireframe()	persp()	3-dim. surface
splom()	pairs()	Several Scatterplots
parallel()	-	Several conditioning plots

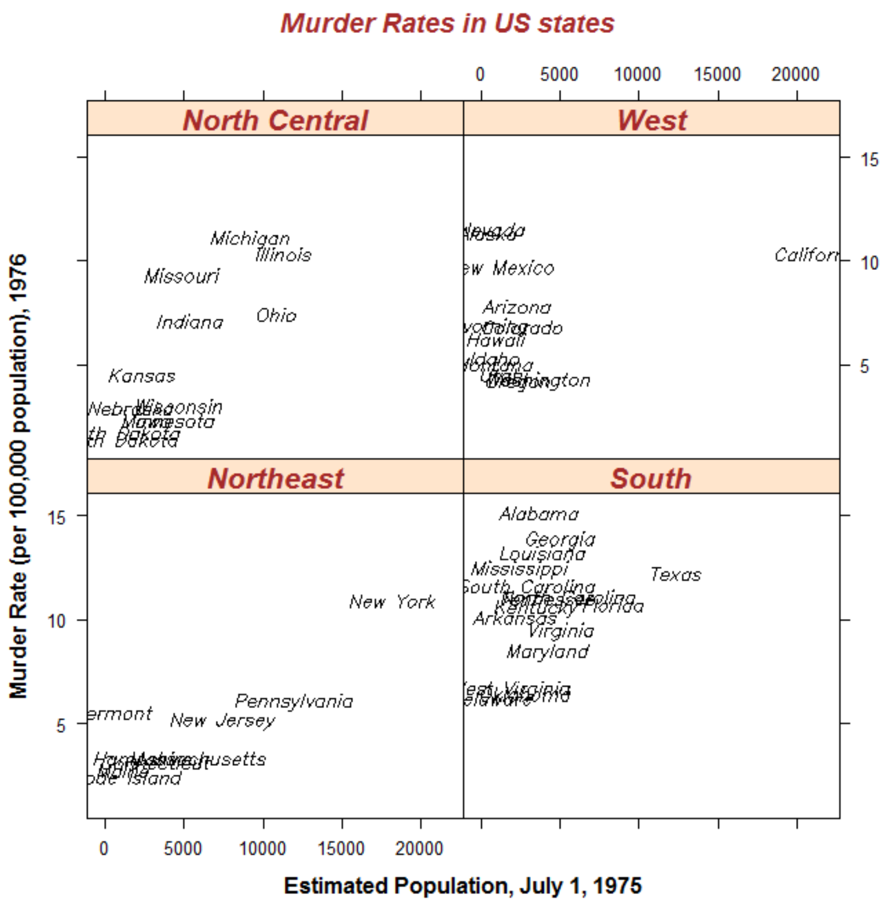
Lattice package: densityplot() [Source: demo(lattice)]



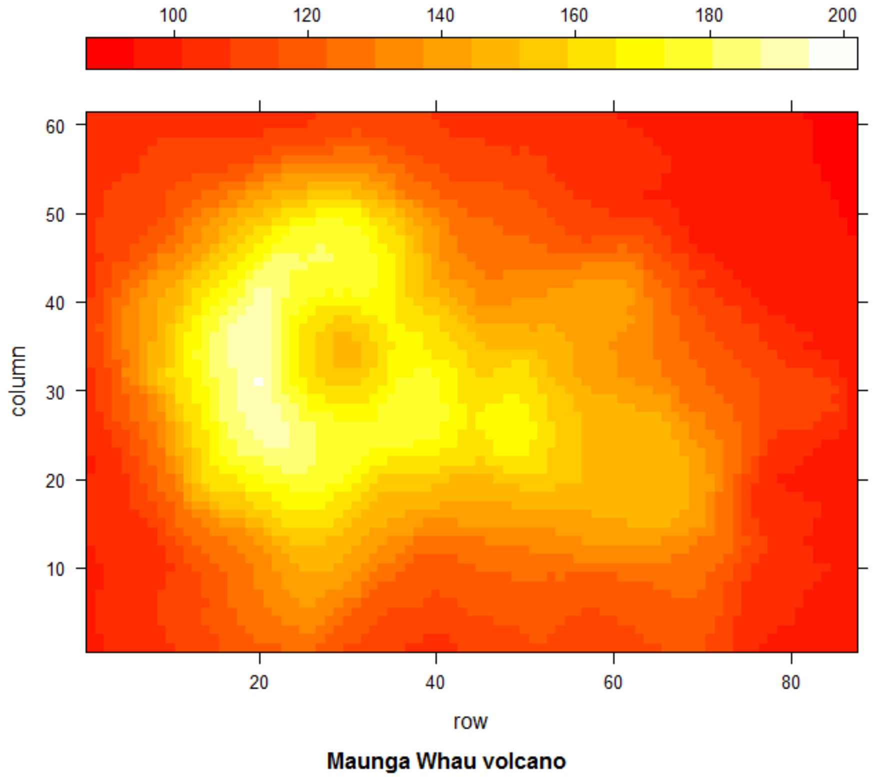
Lattice package: densityplot() [Source: demo(lattice)]



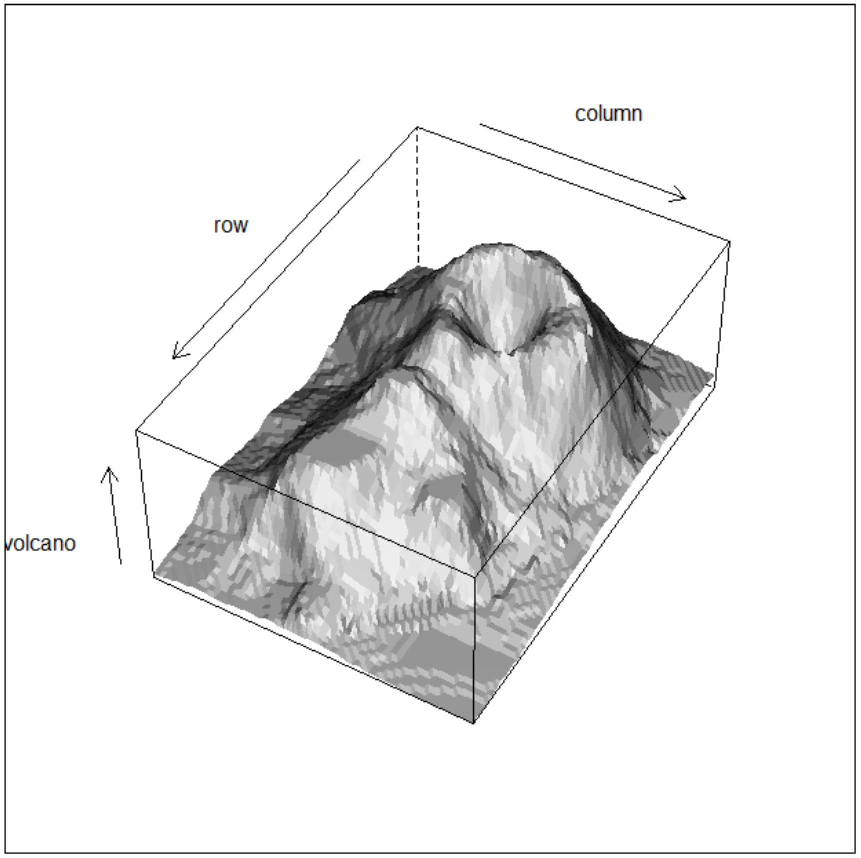
Lattice package: xyplot() [Source: demo(lattice)]



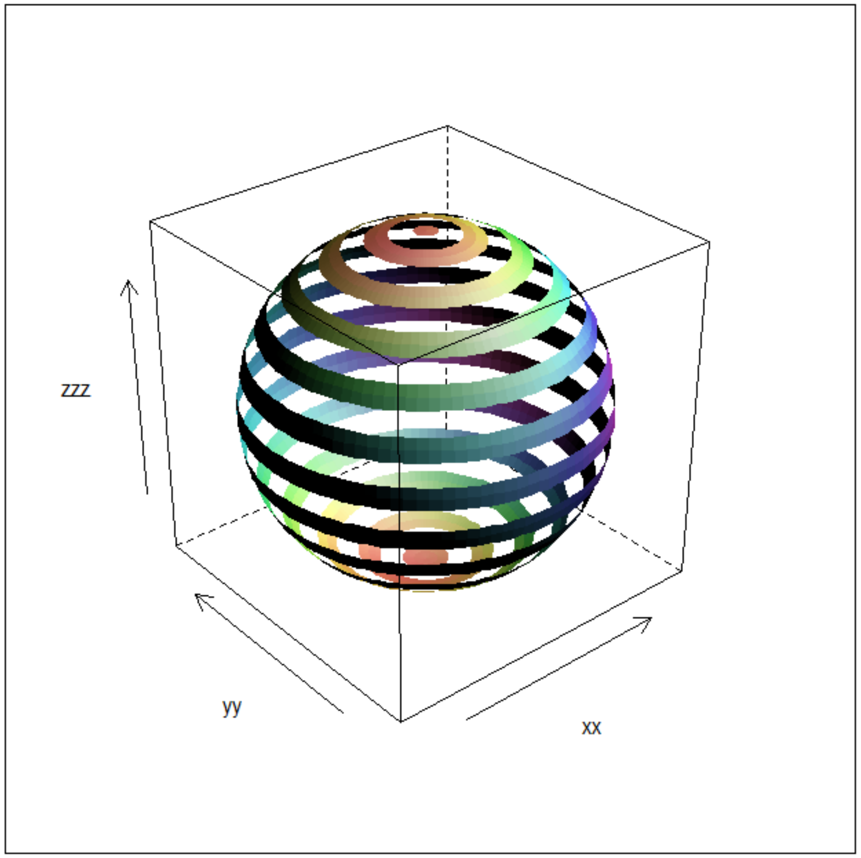
Lattice package: levelplot() [Source: demo(lattice)]



Lattice package: wireframe() [Source: demo(lattice)]



Lattice package: wireframe() [Source: demo(lattice)]

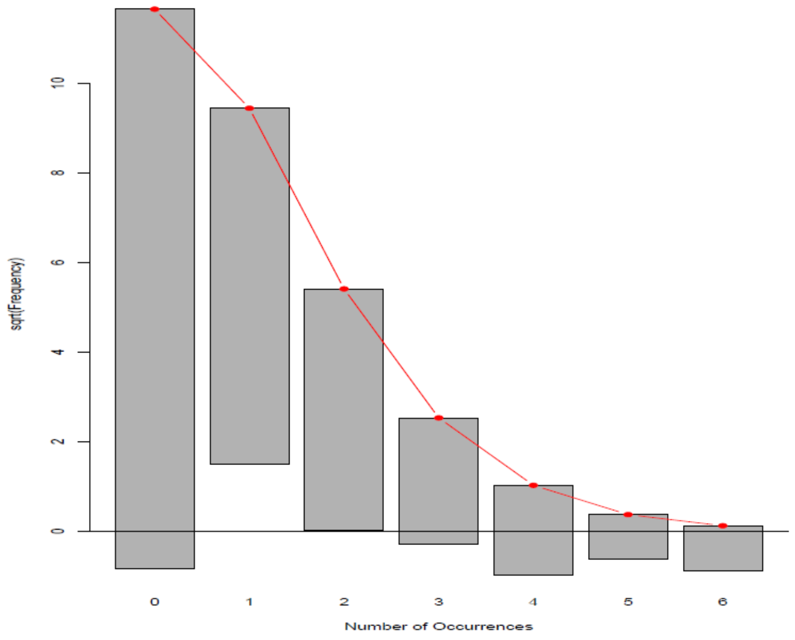


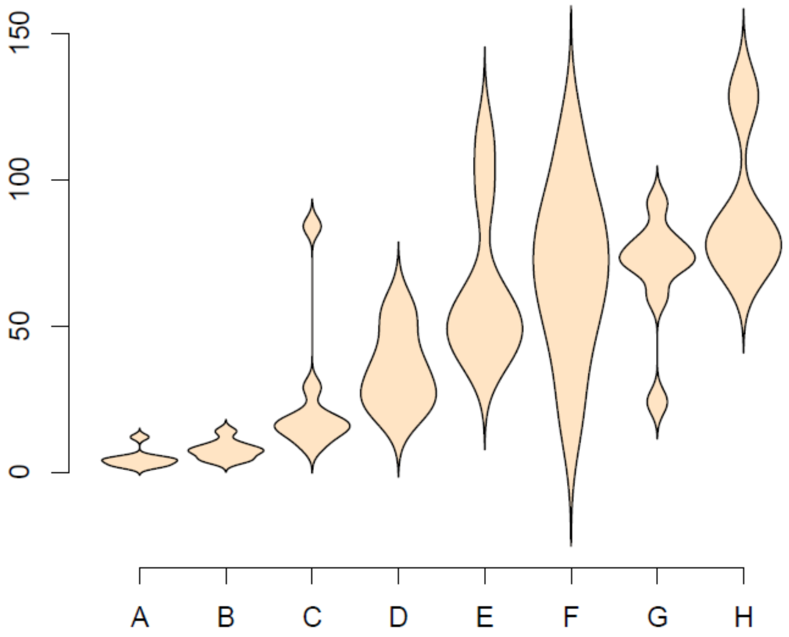
vcd-package

- Several new plots
- Documentation of the vcd package at CRAN R-Project
  http://cran.r-project.org/web/packages/vcd/index.html

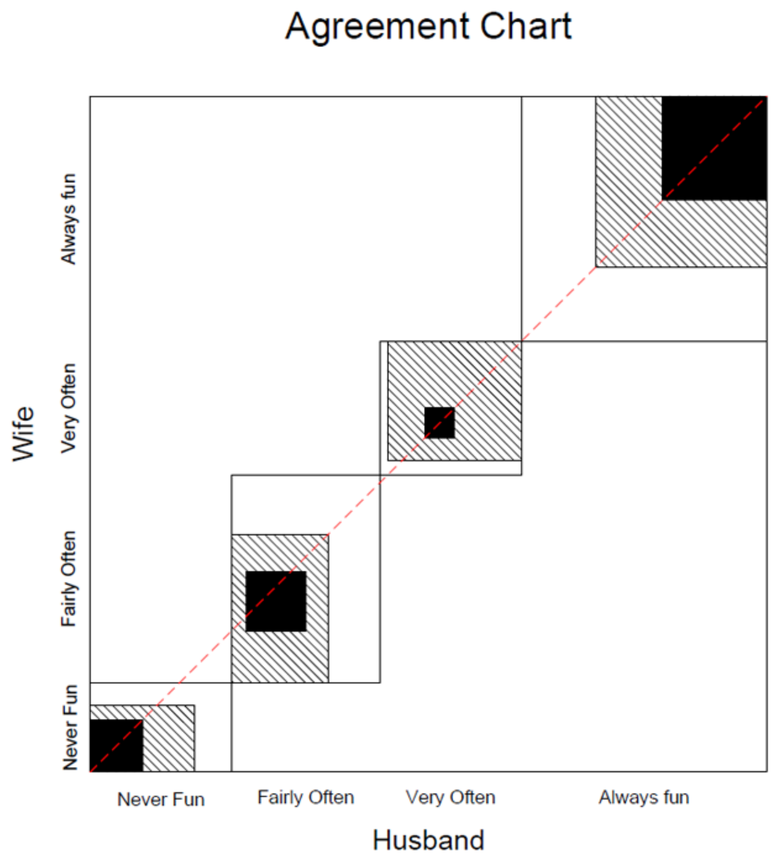
- Images source:
  http://www.stat.ucl.ac.be/ISpersonnel/lecoutre/stats/chiers/_gallery.pdf

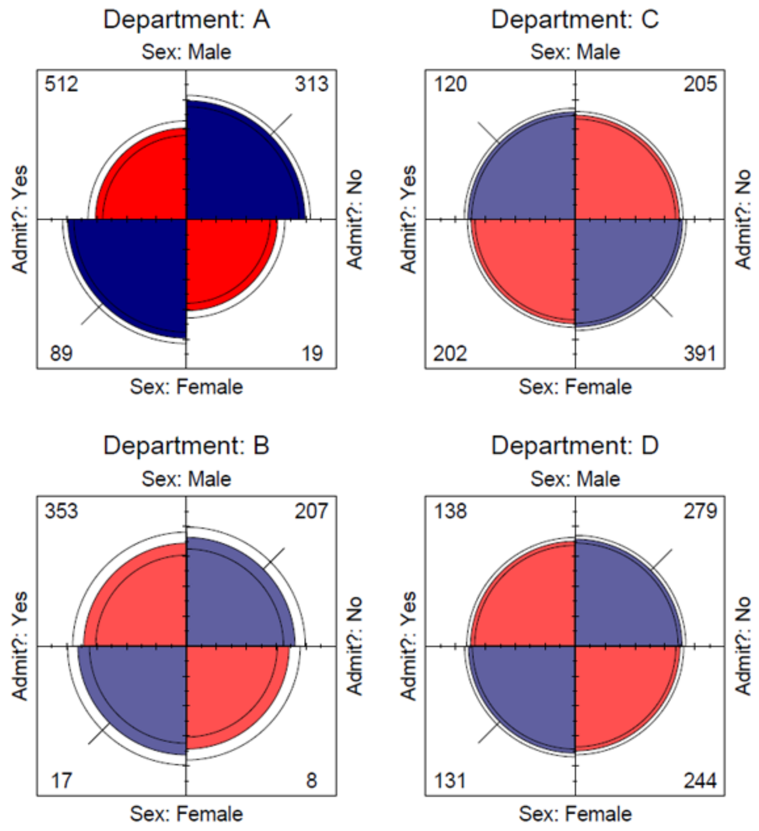
vcd package: rootogram() & simple.violinplot()



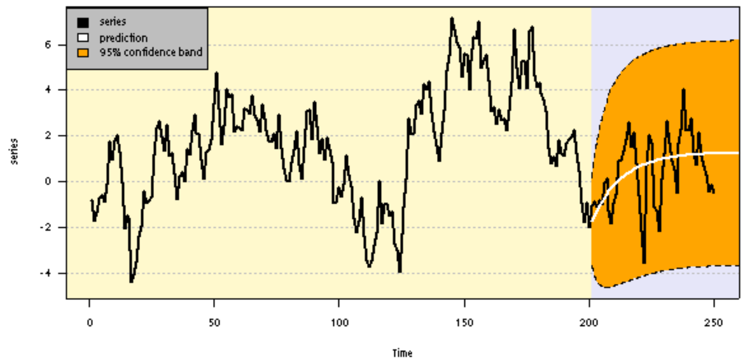


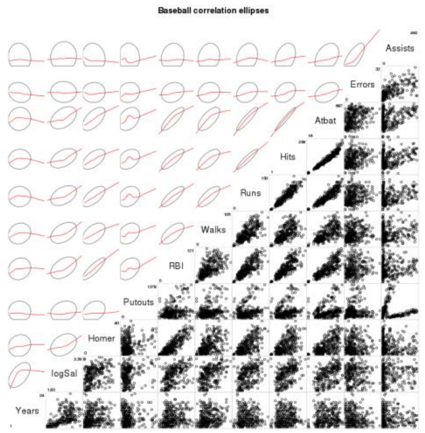
vcd package: agreementplot() & fourfoldplot()

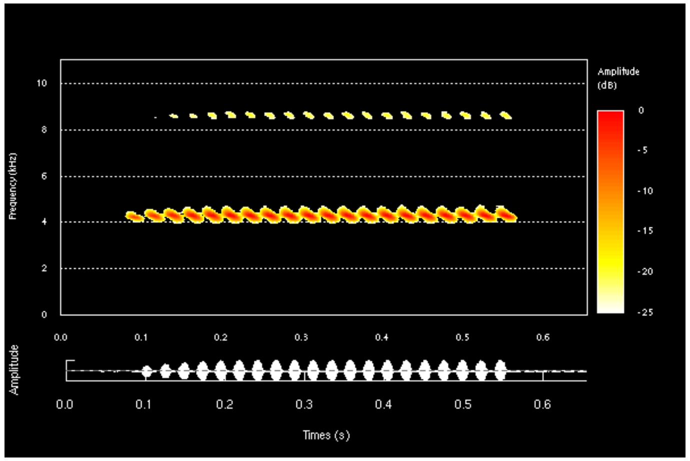


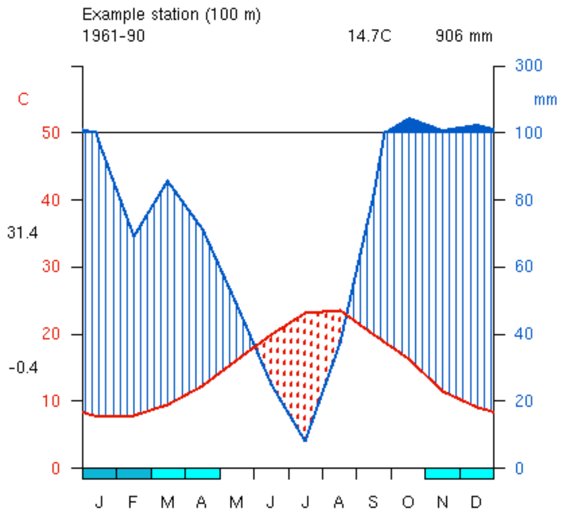


Other packages: [Source: http://addictedtor.free.fr/graphiques]

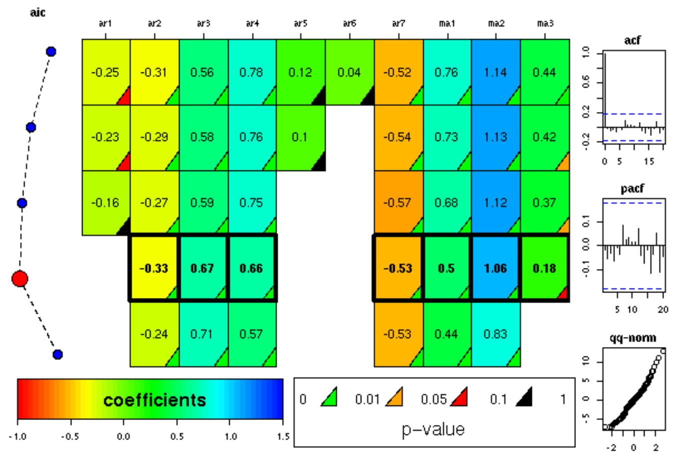


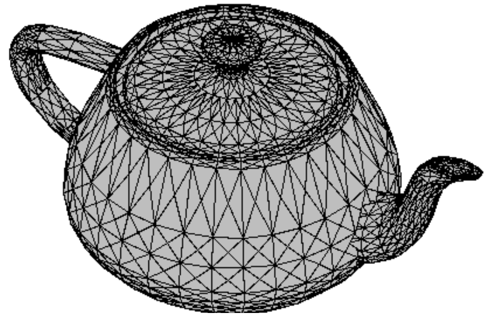


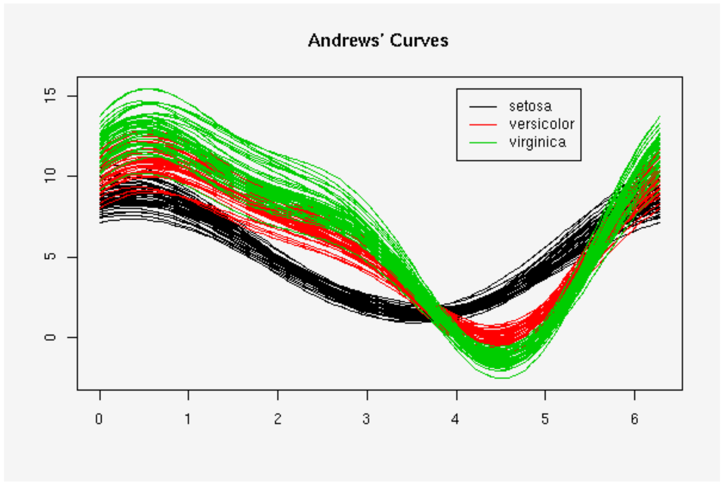


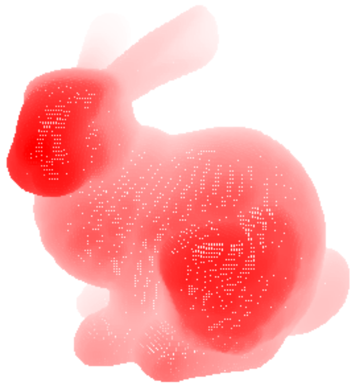


Other packages: [Source: http://addictedtor.free.fr/graphiques]







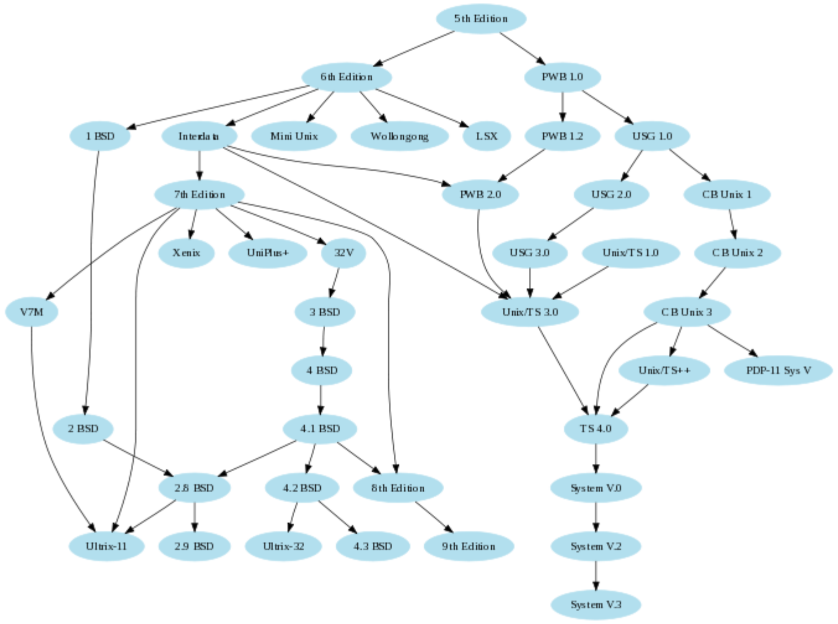


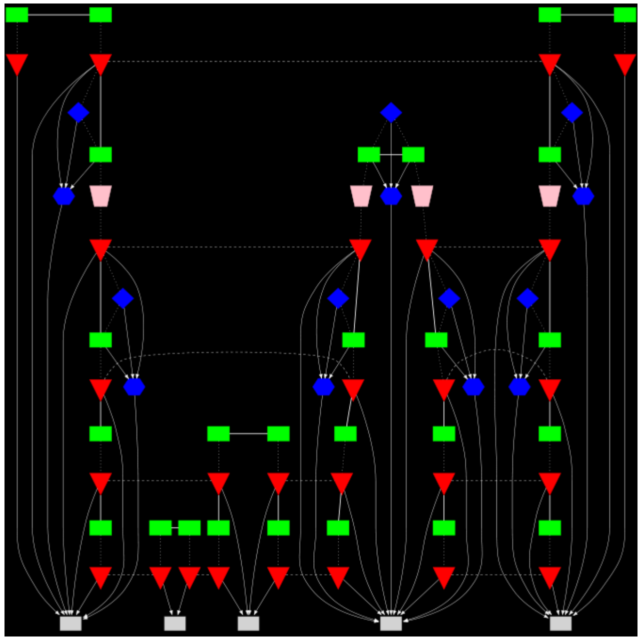
Drawing graphs with R

- There actually is a way to draw graphs in R!
- …with 3rd party Software though.
- Requires BioConductor and Graphviz software
- …and the Rgraphivz package
- Tutorial to drawing graphs with R by Peter Cock

  http://www2.warwick.ac.uk/fac/sci/moac/currentstudents/peter_cock/r/rgraphviz/

Drawing graphs with R





II. 3-dimensional barplots: hist3d

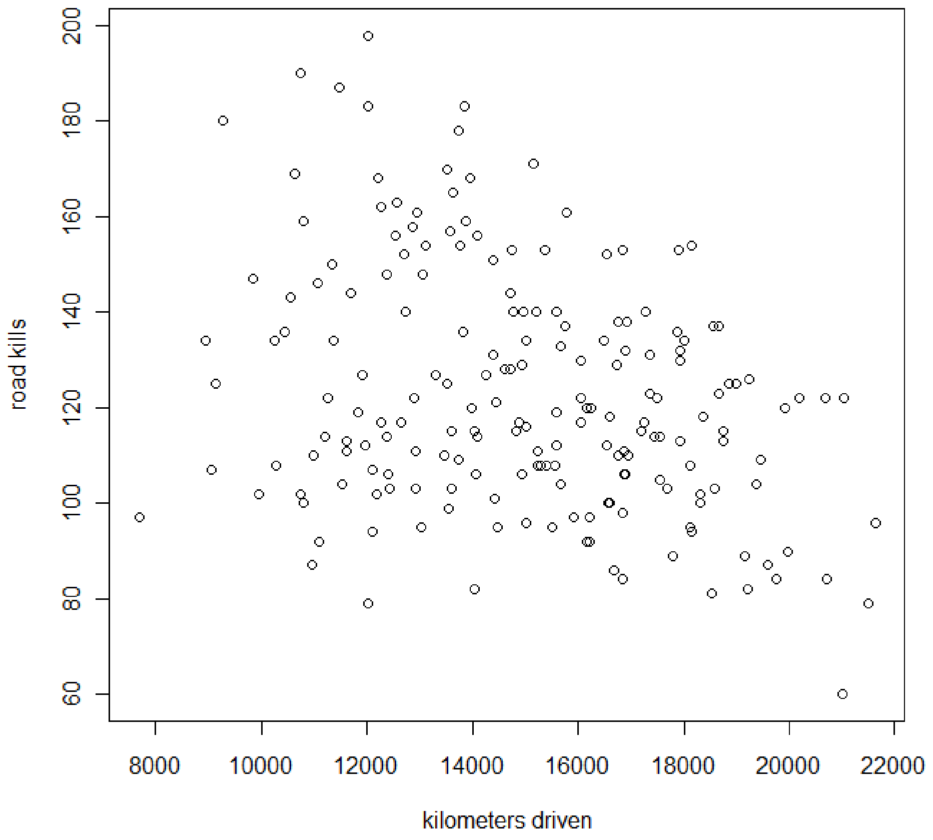
i. Motivation

What about 3-dimensional barplots?

Motivation

- „barplot()“ function is 2-dimensional
- There is no built-in 3-dimensional barplot function
- There seems to be no package to do the job
- There may however be the need
  - Example: ‚Seatbelts‘ data
  - Want to visualise: # road kills   vs. # kilometeres driven
  - Frequency / Sort of 3-dimensional histogram

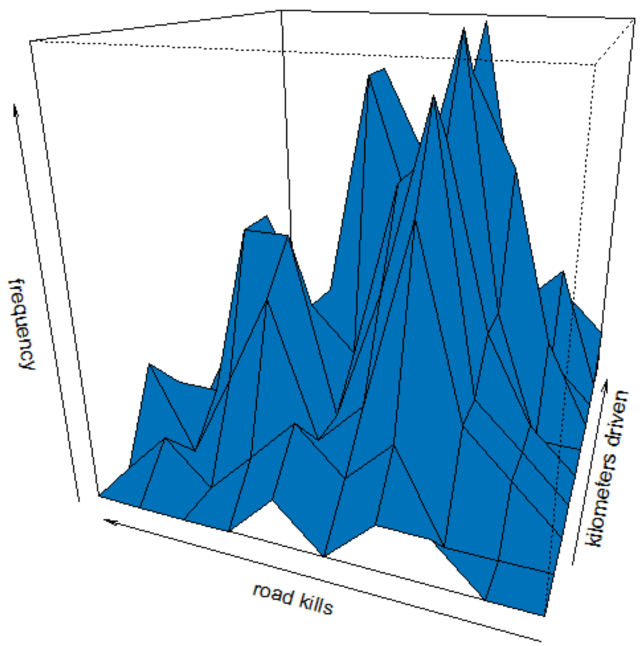
Motivation


road kills
200
180
160
140
120
100
80
60
8000
10000
12000
14000
16000
18000
20000
22000
kilometers driven

Motivation

- Plotting the „pure“ data with persp:

persp-plot of Haufigkeit



II. 3-dimensional barplots: hist3d

ii. Implementation

Basic ideas

Implementation

Prepare data:

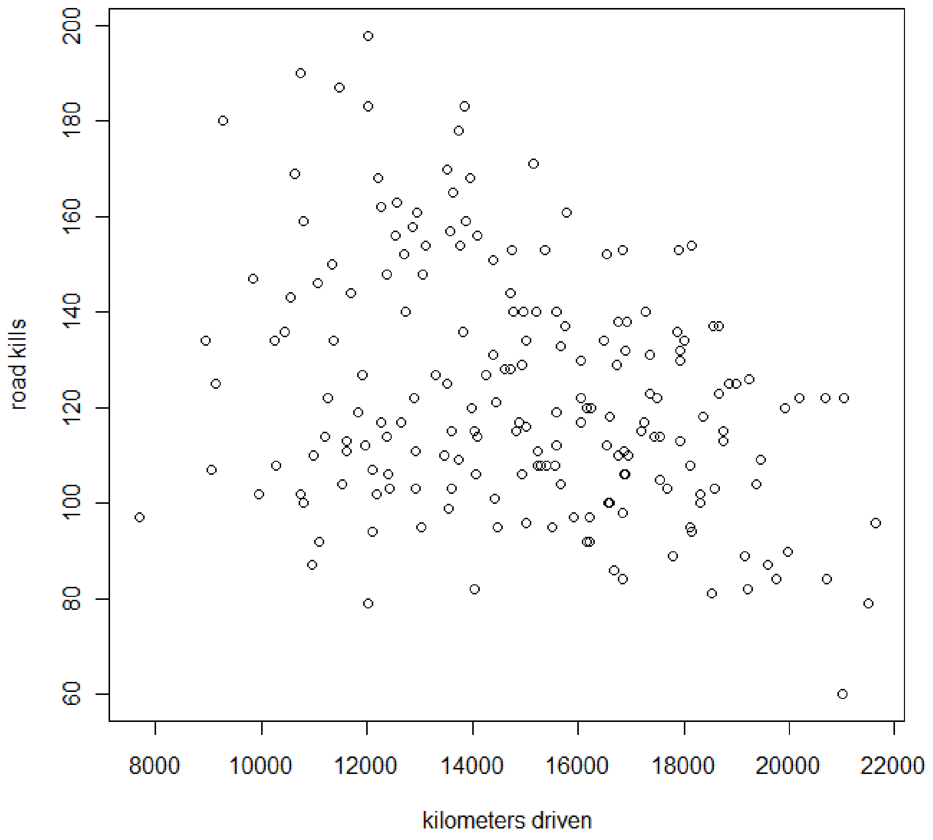
1. Divide the domain of definition into squares
2. Count the datapoints in the squares (=frequency)

Plot:

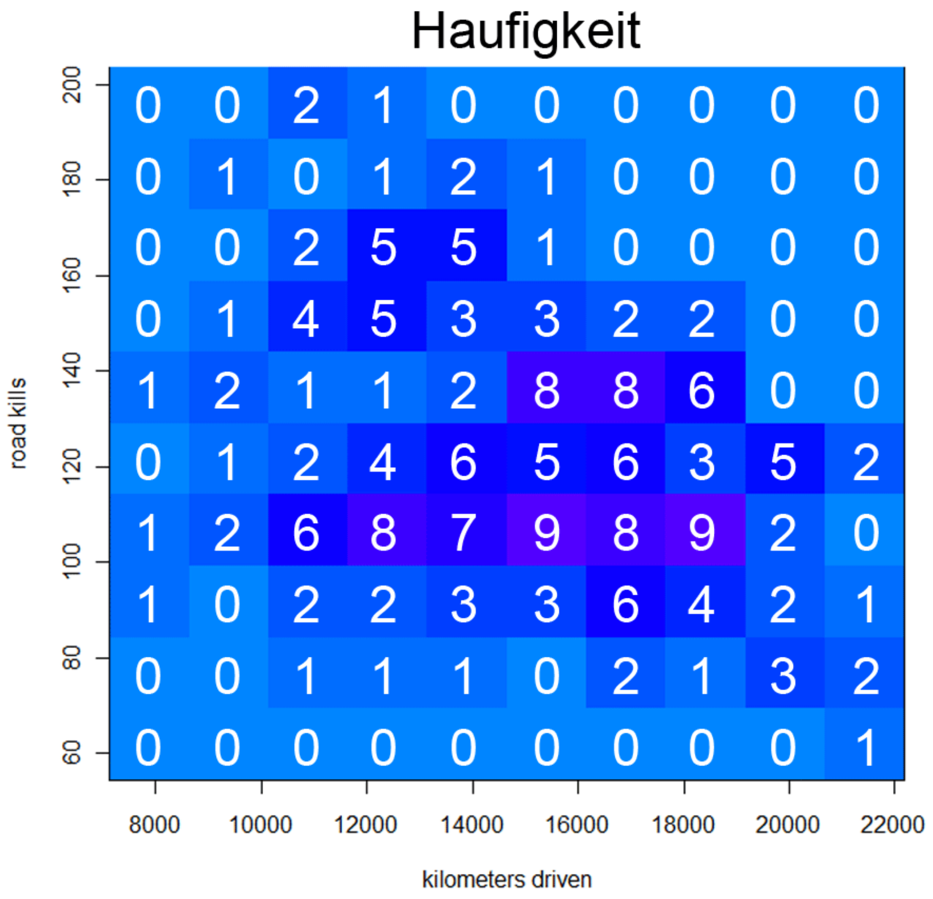
3. Divide the existing squares into smaller squares (=finer grid)
4. Fill this fine grid with „old“ values from the rough grid
5. Plot a persp() function

=> The fine grid will force persp() to plot something like a 3-dim. bar

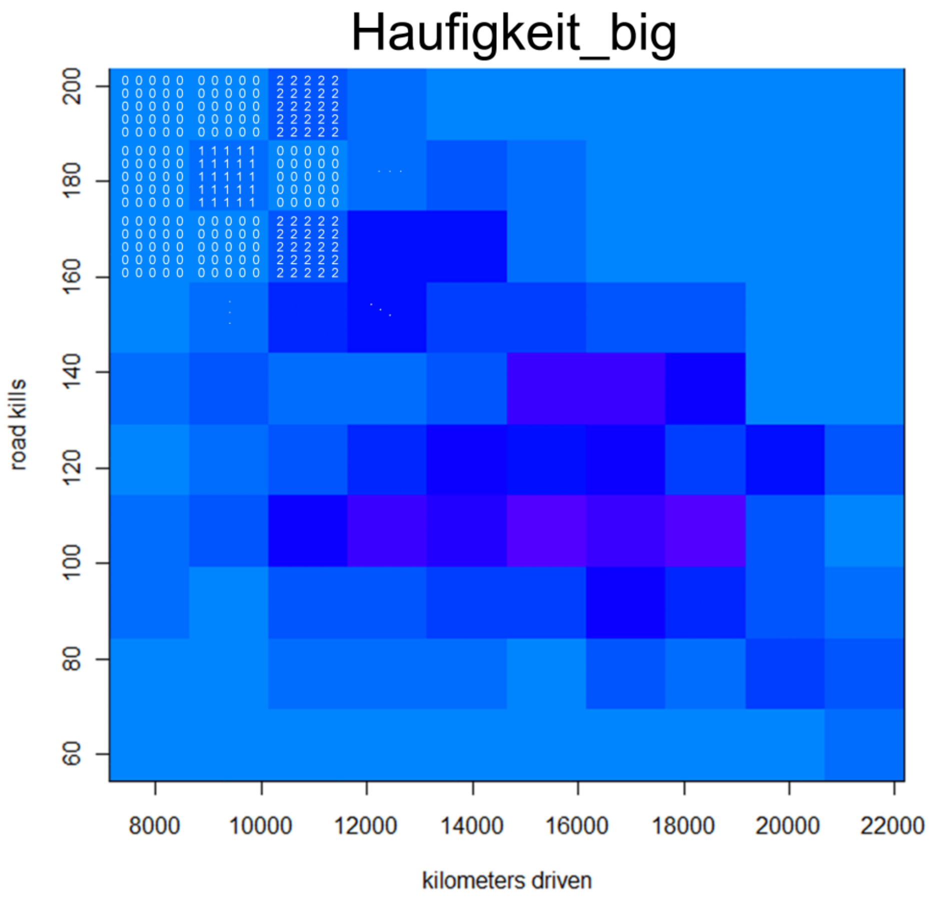
Implementation

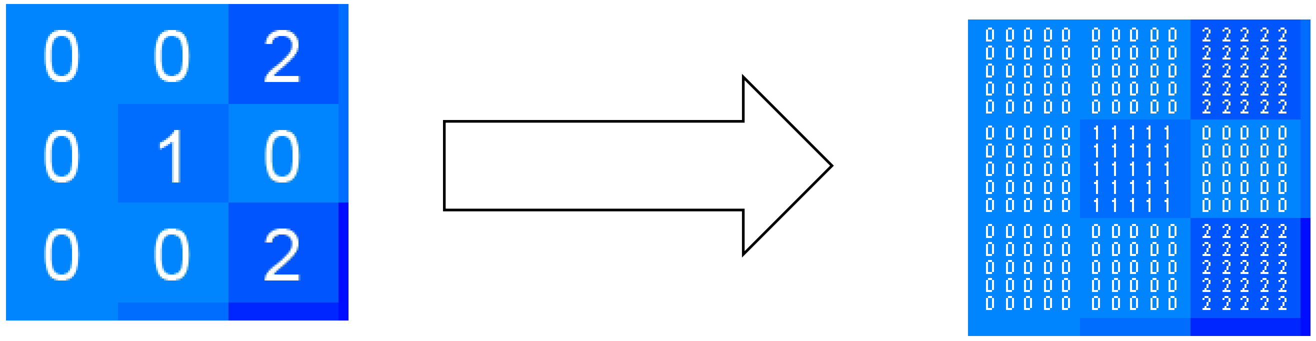


Implementation

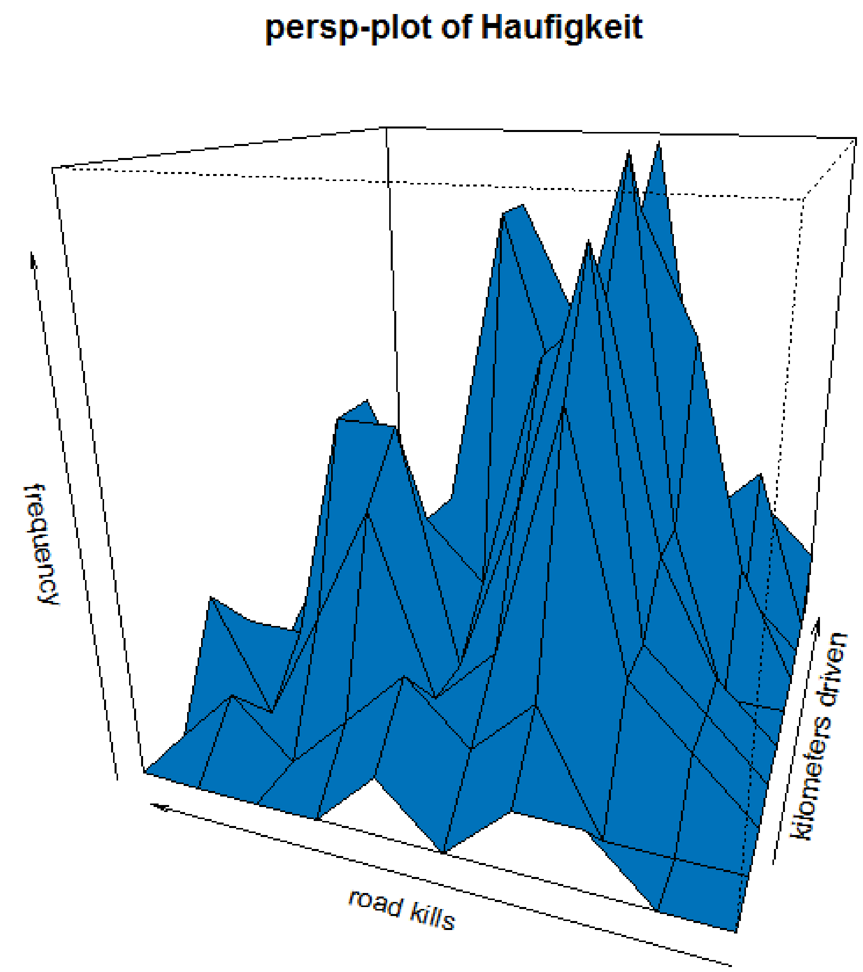


Implementation

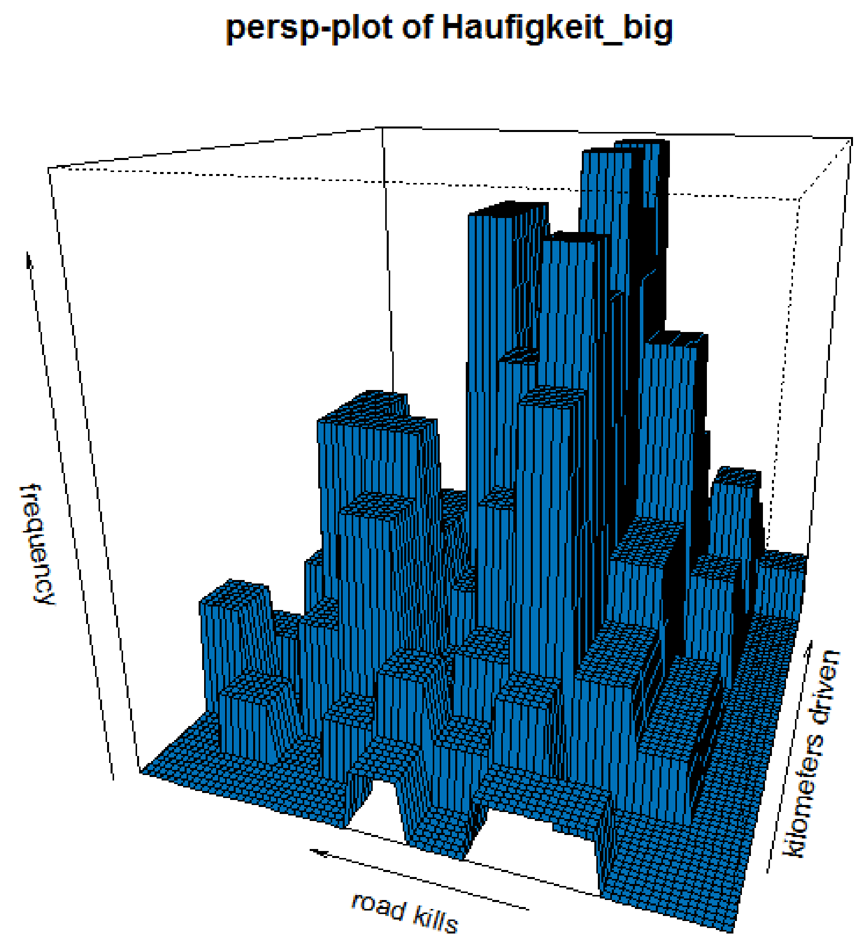




Implementation



Implementation

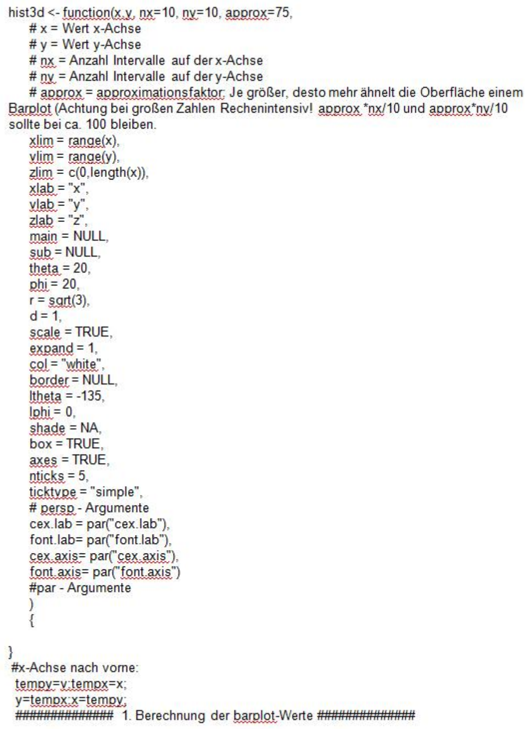


II. 3-dimensional barplots: hist3d

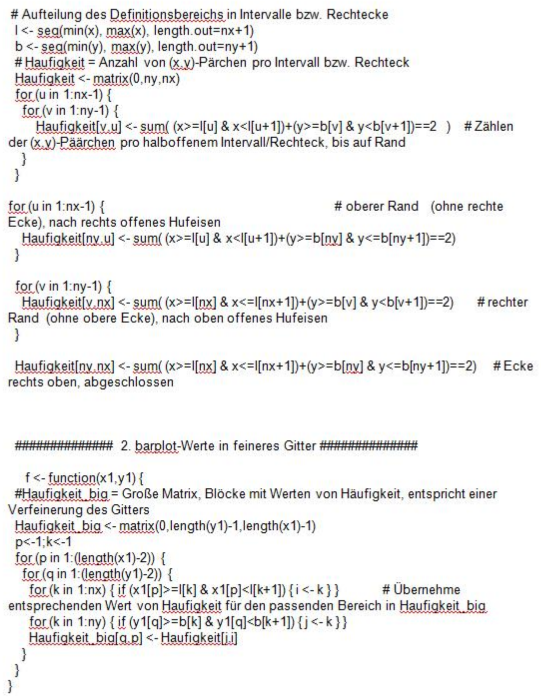
iii. The Code – step by step

Take a deep breath…

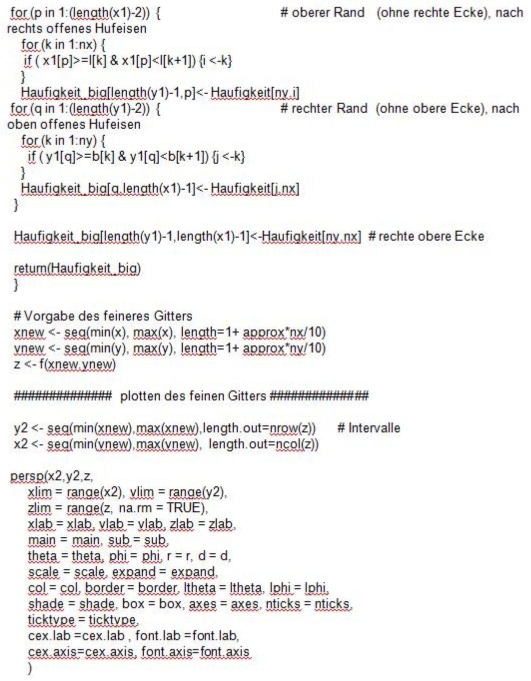
The Code – step by step



```
hist3d <- function(x,y, nx=10, ny=10, approx=75,
      # x = Wert x-Achse
      # y = Wert y-Achse
      # nx = Anzahl Intervalle auf der x-Achse
      # ny = Anzahl Intervalle auf der y-Achse
      # approx = approximationsfaktor: Je größer, desto mehr ähnelt die Oberfläche einem
Barplot (Achtung bei großen Zahlen Rechenintensiv! approx *nx/10 und approx*ny/10
sollte bei ca. 100 bleiben.
      xlim = range(x),
      ylim = range(y),
      zlim = c(0,length(x)),
      xlab = "x",
      ylab = "y",
      zlab = "z",
      main = NULL,
      sub = NULL,
      theta = 20,
      phi = 20,
      r = sqrt(3),
      d = 1,
      scale = TRUE,
      expand = 1,
      col = "white",
      border = NULL,
      ltheta = -135,
      lphi = 0,
      shade = NA,
      box = TRUE,
      axes = TRUE,
      nticks = 5,
      ticktype = "simple",
      # persp - Argumente
      cex.lab = par("cex.lab"),
      font.lab= par("font.lab"),
      cex.axis= par("cex.axis"),
      font.axis= par("font.axis")
      #par - Argumente
      )
      {

}
 #x-Achse nach vorne:
  tempy=y;tempx=x;
  y=tempx;x=tempy;
  ##############  1. Berechnung der barplot-Werte ##############
```

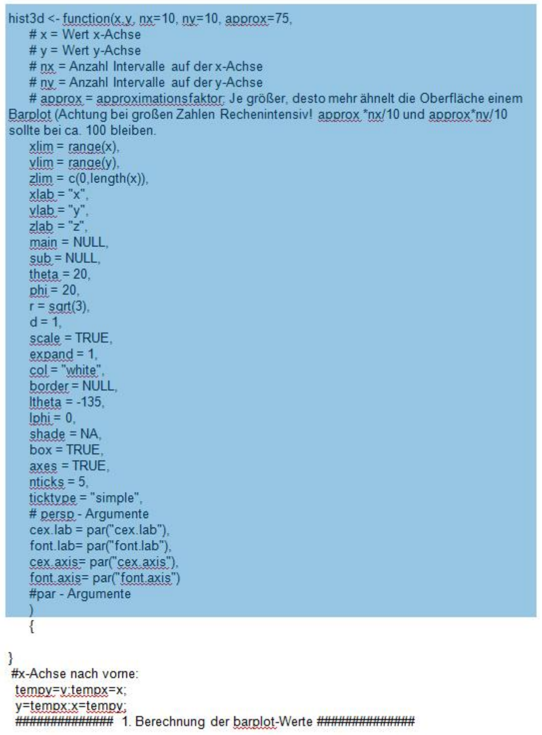


```
 # Aufteilung des Definitionsbereichs in Intervalle bzw. Rechtecke
  l <- seq(min(x), max(x), length.out=nx+1)
  b <- seq(min(y), max(y), length.out=ny+1)
  # Haufigkeit = Anzahl von (x,y)-Pärchen pro Intervall bzw. Rechteck
  Haufigkeit <- matrix(0,ny,nx)
  for (u in 1:nx-1) {
    for (v in 1:ny-1) {
       Haufigkeit[v,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[v] & y<b[v+1])==2  )   # Zählen
der (x,y)-Päärchen pro halboffenem Intervall/Rechteck, bis auf Rand
    }
  }

for (u in 1:nx-1) {                                              # oberer Rand   (ohne rechte
Ecke), nach rechts offenes Hufeisen
    Haufigkeit[ny,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[ny] & y<=b[ny+1])==2)
  }

  for (v in 1:ny-1) {
    Haufigkeit[v,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[v] & y<b[v+1])==2)      # rechter
Rand  (ohne obere Ecke), nach oben offenes Hufeisen
  }

  Haufigkeit[ny,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[ny] & y<=b[ny+1])==2)    # Ecke
rechts oben, abgeschlossen



  ##############  2. barplot-Werte in feineres Gitter ##############

     f <- function(x1,y1) {
  #Haufigkeit_big = Große Matrix, Blöcke mit Werten von Häufigkeit, entspricht einer
Verfeinerung des Gitters
  Haufigkeit_big <- matrix(0,length(y1)-1,length(x1)-1)
  p<-1;k<-1
  for (p in 1:(length(x1)-2)) {
    for (q in 1:(length(y1)-2)) {
      for (k in 1:nx) { if (x1[p]>=l[k] & x1[p]<l[k+1]) { i <- k } }          # Übernehme
entsprechenden Wert von Haufigkeit für den passenden Bereich in Haufigkeit_big
      for (k in 1:ny) { if (y1[q]>=b[k] & y1[q]<b[k+1]) { j <- k } }
      Haufigkeit_big[q,p] <- Haufigkeit[j,i]
    }
  }
}
```

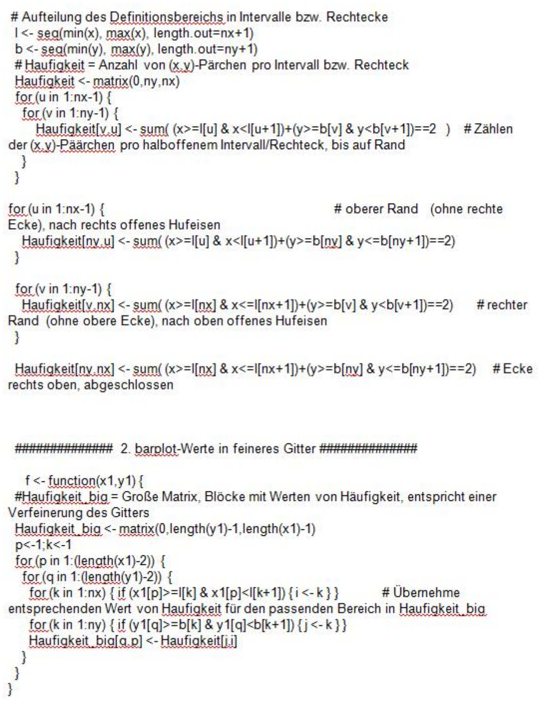


```
 for (p in 1:(length(x1)-2)) {                        # oberer Rand   (ohne rechte Ecke), nach
rechts offenes Hufeisen
    for (k in 1:nx) {
     if ( x1[p]>=l[k] & x1[p]<l[k+1]) {i <-k}
    }
    Haufigkeit_big[length(y1)-1,p]<- Haufigkeit[ny,i]
 for (q in 1:(length(y1)-2)) {                        # rechter Rand  (ohne obere Ecke), nach
oben offenes Hufeisen
    for (k in 1:ny) {
      if ( y1[q]>=b[k] & y1[q]<b[k+1]) {j <-k}
    }
    Haufigkeit_big[q,length(x1)-1]<- Haufigkeit[j,nx]
  }

  Haufigkeit_big[length(y1)-1,length(x1)-1]<-Haufigkeit[ny,nx]  # rechte obere Ecke

  return(Haufigkeit_big)
  }

  # Vorgabe des feineres Gitters
  xnew <- seq(min(x), max(x), length=1+ approx*nx/10)
  ynew <- seq(min(y), max(y), length=1+ approx*ny/10)
  z <- f(xnew,ynew)

  ##############  plotten des feinen Gitters ##############

  y2 <- seq(min(xnew),max(xnew),length.out=nrow(z))      # Intervalle
  x2 <- seq(min(ynew),max(ynew),  length.out=ncol(z))

 persp(x2,y2,z,
     xlim = range(x2), ylim = range(y2),
     zlim = range(z, na.rm = TRUE),
     xlab = xlab, ylab = ylab, zlab = zlab,
     main = main, sub = sub,
     theta = theta, phi = phi, r = r, d = d,
     scale = scale, expand = expand,
     col = col, border = border, ltheta = ltheta, lphi = lphi,
     shade = shade, box = box, axes = axes, nticks = nticks,
     ticktype = ticktype,
     cex.lab =cex.lab , font.lab =font.lab,
     cex.axis=cex.axis, font.axis=font.axis
     )
```

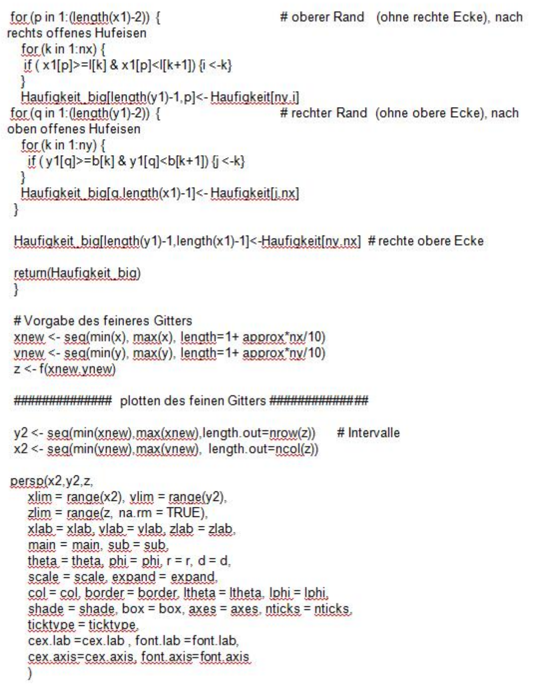
The Code – step by step



```
hist3d <- function(x,y, nx=10, ny=10, approx=75,
     # x = Wert x-Achse
     # y = Wert y-Achse
     # nx = Anzahl Intervalle auf der x-Achse
     # ny = Anzahl Intervalle auf der y-Achse
     # approx = approximationsfaktor; Je größer, desto mehr ähnelt die Oberfläche einem
Barplot (Achtung bei großen Zahlen Rechenintensiv! approx *nx/10 und approx*ny/10
sollte bei ca. 100 bleiben.
     xlim = range(x),
     ylim = range(y),
     zlim = c(0,length(x)),
     xlab = "x",
     ylab = "y",
     zlab = "z",
     main = NULL,
     sub = NULL,
     theta = 20,
     phi = 20,
     r = sqrt(3),
     d = 1,
     scale = TRUE,
     expand = 1,
     col = "white",
     border = NULL,
     ltheta = -135,
     lphi = 0,
     shade = NA,
     box = TRUE,
     axes = TRUE,
     nticks = 5,
     ticktype = "simple",
     # persp - Argumente
     cex.lab = par("cex.lab"),
     font.lab= par("font.lab"),
     cex.axis= par("cex.axis"),
     font.axis= par("font.axis")
     #par - Argumente
     )
     {

}
 #x-Achse nach vorne:
  tempy=y;tempx=x;
  y=tempx;x=tempy;
  ###############  1. Berechnung der barplot-Werte ###############
```



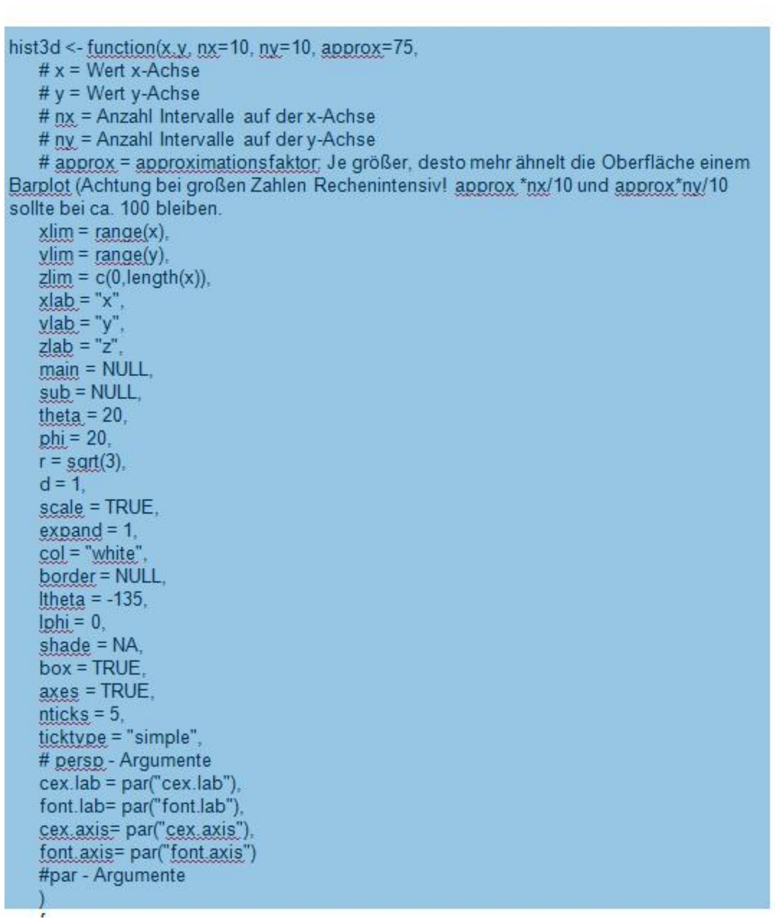
```
 # Aufteilung des Definitionsbereichs in Intervalle bzw. Rechtecke
  l <- seq(min(x), max(x), length.out=nx+1)
  b <- seq(min(y), max(y), length.out=ny+1)
  # Haufigkeit = Anzahl von (x,y)-Pärchen pro Intervall bzw. Rechteck
  Haufigkeit <- matrix(0,ny,nx)
  for (u in 1:nx-1) {
   for (v in 1:ny-1) {
      Haufigkeit[v,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[v] & y<b[v+1])==2  )   # Zählen
der (x,y)-Päärchen pro halboffenem Intervall/Rechteck, bis auf Rand
   }
  }

for (u in 1:nx-1) {                                        # oberer Rand   (ohne rechte
Ecke), nach rechts offenes Hufeisen
   Haufigkeit[ny,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[ny] & y<=b[ny+1])==2)
  }

  for (v in 1:ny-1) {
   Haufigkeit[v,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[v] & y<b[v+1])==2)      # rechter
Rand  (ohne obere Ecke), nach oben offenes Hufeisen
  }

  Haufigkeit[ny,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[ny] & y<=b[ny+1])==2)    # Ecke
rechts oben, abgeschlossen



  ###############  2. barplot-Werte in feineres Gitter ###############

     f <- function(x1,y1) {
  #Haufigkeit_big = Große Matrix, Blöcke mit Werten von Häufigkeit, entspricht einer
Verfeinerung des Gitters
  Haufigkeit_big <- matrix(0,length(y1)-1,length(x1)-1)
  p<-1;k<-1
  for (p in 1:(length(x1)-2)) {
   for (q in 1:(length(y1)-2)) {
     for (k in 1:nx) { if (x1[p]>=l[k] & x1[p]<l[k+1]) { i <- k } }           # Übernehme
entsprechenden Wert von Haufigkeit für den passenden Bereich in Haufigkeit_big
     for (k in 1:ny) { if (y1[q]>=b[k] & y1[q]<b[k+1]) { j <- k } }
     Haufigkeit_big[q,p] <- Haufigkeit[j,i]
    }
  }
}
```



```
 for (p in 1:(length(x1)-2)) {                    # oberer Rand   (ohne rechte Ecke), nach
rechts offenes Hufeisen
   for (k in 1:nx) {
    if ( x1[p]>=l[k] & x1[p]<l[k+1]) {i <-k}
   }
   Haufigkeit_big[length(y1)-1,p]<- Haufigkeit[ny,i]
 for (q in 1:(length(y1)-2)) {                    # rechter Rand  (ohne obere Ecke), nach
oben offenes Hufeisen
   for (k in 1:ny) {
     if ( y1[q]>=b[k] & y1[q]<b[k+1]) {j <-k}
   }
   Haufigkeit_big[q,length(x1)-1]<- Haufigkeit[j,nx]
  }

  Haufigkeit_big[length(y1)-1,length(x1)-1]<-Haufigkeit[ny,nx]  # rechte obere Ecke

  return(Haufigkeit_big)
  }

  # Vorgabe des feineres Gitters
  xnew <- seq(min(x), max(x), length=1+ approx*nx/10)
  ynew <- seq(min(y), max(y), length=1+ approx*ny/10)
  z <- f(xnew,ynew)

  ###############  plotten des feinen Gitters ###############

  y2 <- seq(min(xnew),max(xnew),length.out=nrow(z))      # Intervalle
  x2 <- seq(min(ynew),max(ynew),  length.out=ncol(z))

 persp(x2,y2,z,
     xlim = range(x2), ylim = range(y2),
     zlim = range(z, na.rm = TRUE),
     xlab = xlab, ylab = ylab, zlab = zlab,
     main = main, sub = sub,
     theta = theta, phi = phi, r = r, d = d,
     scale = scale, expand = expand,
     col = col, border = border, ltheta = ltheta, lphi = lphi,
     shade = shade, box = box, axes = axes, nticks = nticks,
     ticktype = ticktype,
     cex.lab =cex.lab , font.lab =font.lab,
     cex.axis=cex.axis, font.axis=font.axis
     )
```

Calling the function

Calling the function



```
hist3d <- function(x,y, nx=10, ny=10, approx=75,
     # x = Wert x-Achse
     # y = Wert y-Achse
     # nx = Anzahl Intervalle auf der x-Achse
     # ny = Anzahl Intervalle auf der y-Achse
     # approx = approximationsfaktor; Je größer, desto mehr ähnelt die Oberfläche einem
Barplot (Achtung bei großen Zahlen Rechenintensiv! approx *nx/10 und approx*ny/10
sollte bei ca. 100 bleiben.
     xlim = range(x),
     ylim = range(y),
     zlim = c(0,length(x)),
     xlab = "x",
     ylab = "y",
     zlab = "z",
     main = NULL,
     sub = NULL,
     theta = 20,
     phi = 20,
     r = sqrt(3),
     d = 1,
     scale = TRUE,
     expand = 1,
     col = "white",
     border = NULL,
     ltheta = -135,
     lphi = 0,
     shade = NA,
     box = TRUE,
     axes = TRUE,
     nticks = 5,
     ticktype = "simple",
     # persp - Argumente
     cex.lab = par("cex.lab"),
     font.lab= par("font.lab"),
     cex.axis= par("cex.axis"),
     font.axis= par("font.axis")
     #par - Argumente
     )
```

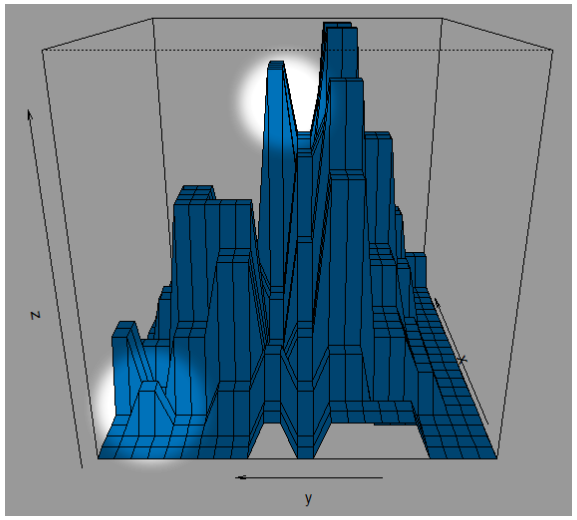
Calling the function

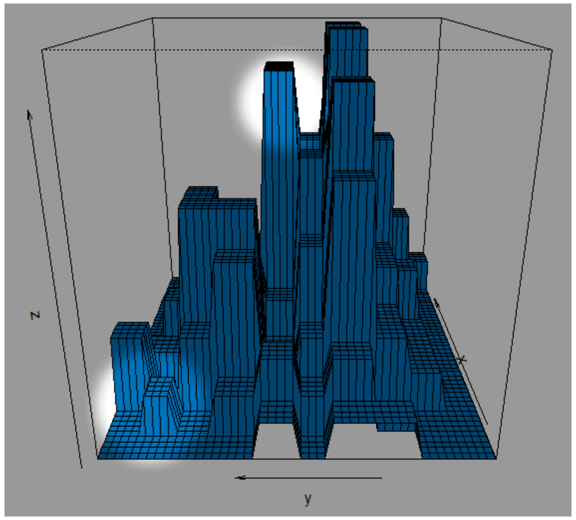
- There are several parameters when calling the new „hist3d“ fct.:

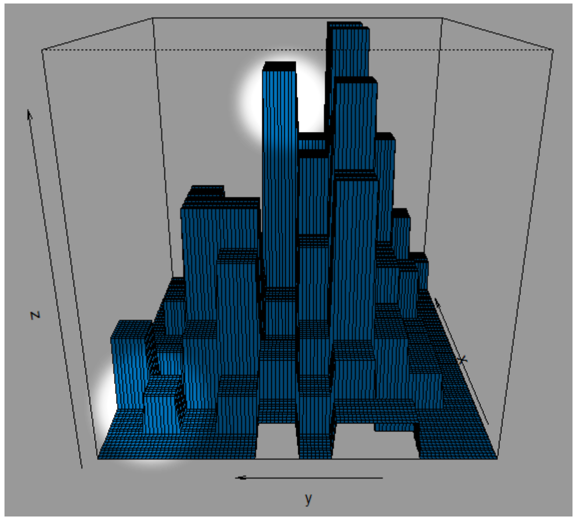
x / y	The data you want to plot
nx / ny	The number of bars you want on the axes Right now: Use nx = ny to avoid malfunction
approx	Sets the size of the fine grid. The higher the more the fct. Will look like a barplot, the lower, the more like a surface
persp	All persp() parameters such as „xlim“ or „theta“ are handed through the function. All defaults are persp‘s defaults.
par	You may use cex.lab, font.lab,cex.axis and font.axis

Calling the function: The approx-parameter

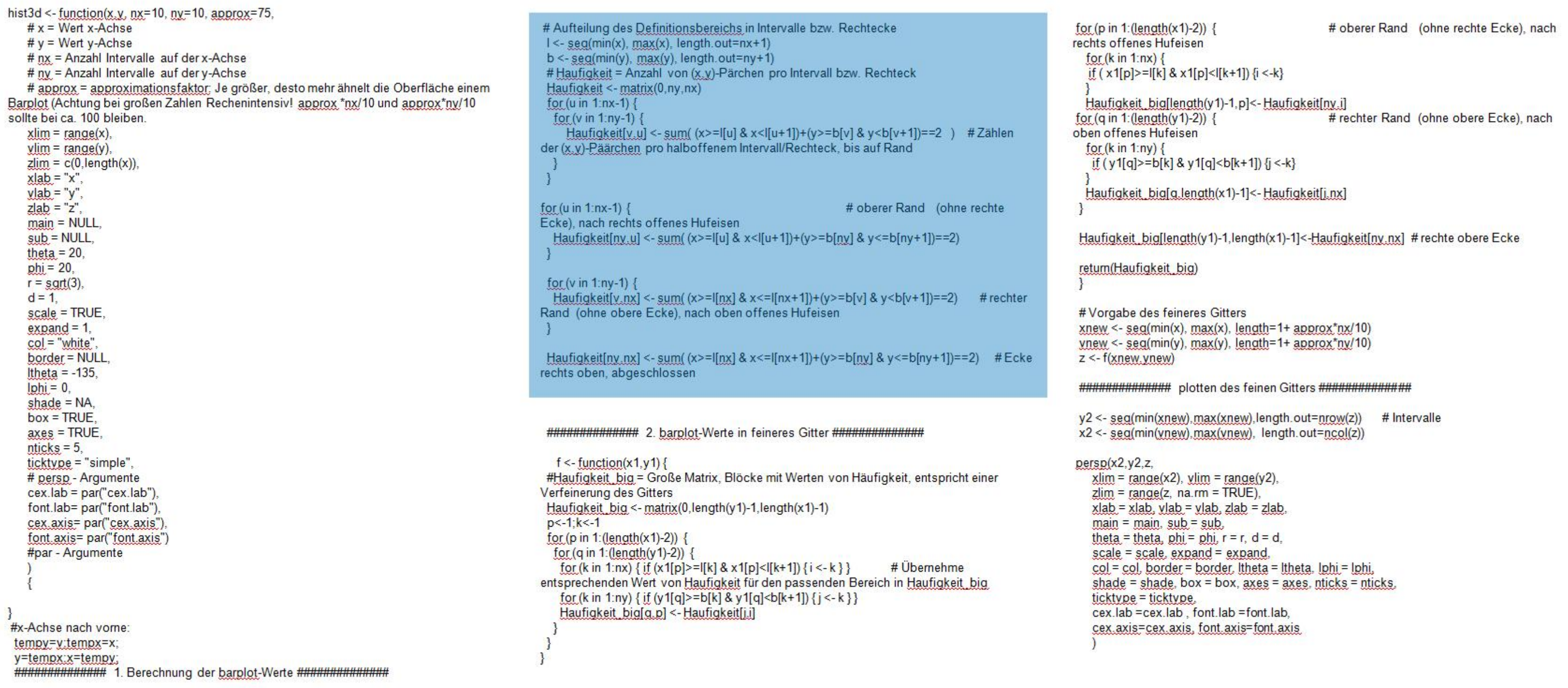
approx = 25 $\Rightarrow$ approx = 50 $\Rightarrow$ approx= 75







The Code – step by step



```
hist3d <- function(x,y, nx=10, ny=10, approx=75,
      # x = Wert x-Achse
      # y = Wert y-Achse
      # nx = Anzahl Intervalle  auf der x-Achse
      # ny = Anzahl Intervalle  auf der y-Achse
      # approx = approximationsfaktor: Je größer, desto mehr ähnelt die Oberfläche einem
Barplot (Achtung bei großen Zahlen Rechenintensiv! approx *nx/10 und approx*ny/10
sollte bei ca. 100 bleiben.
      xlim = range(x),
      ylim = range(y),
      zlim = c(0,length(x)),
      xlab = "x",
      ylab = "y",
      zlab = "z",
      main = NULL,
      sub = NULL,
      theta = 20,
      phi = 20,
      r = sqrt(3),
      d = 1,
      scale = TRUE,
      expand = 1,
      col = "white",
      border = NULL,
      ltheta = -135,
      lphi = 0,
      shade = NA,
      box = TRUE,
      axes = TRUE,
      nticks = 5,
      ticktype = "simple",
      # persp - Argumente
      cex.lab = par("cex.lab"),
      font.lab= par("font.lab"),
      cex.axis= par("cex.axis"),
      font.axis= par("font.axis")
      #par - Argumente
      )
      {

}
 #x-Achse nach vorne:
  tempy=y;tempx=x;
  y=tempx;x=tempy;
  ###############  1. Berechnung der barplot-Werte ###############
```

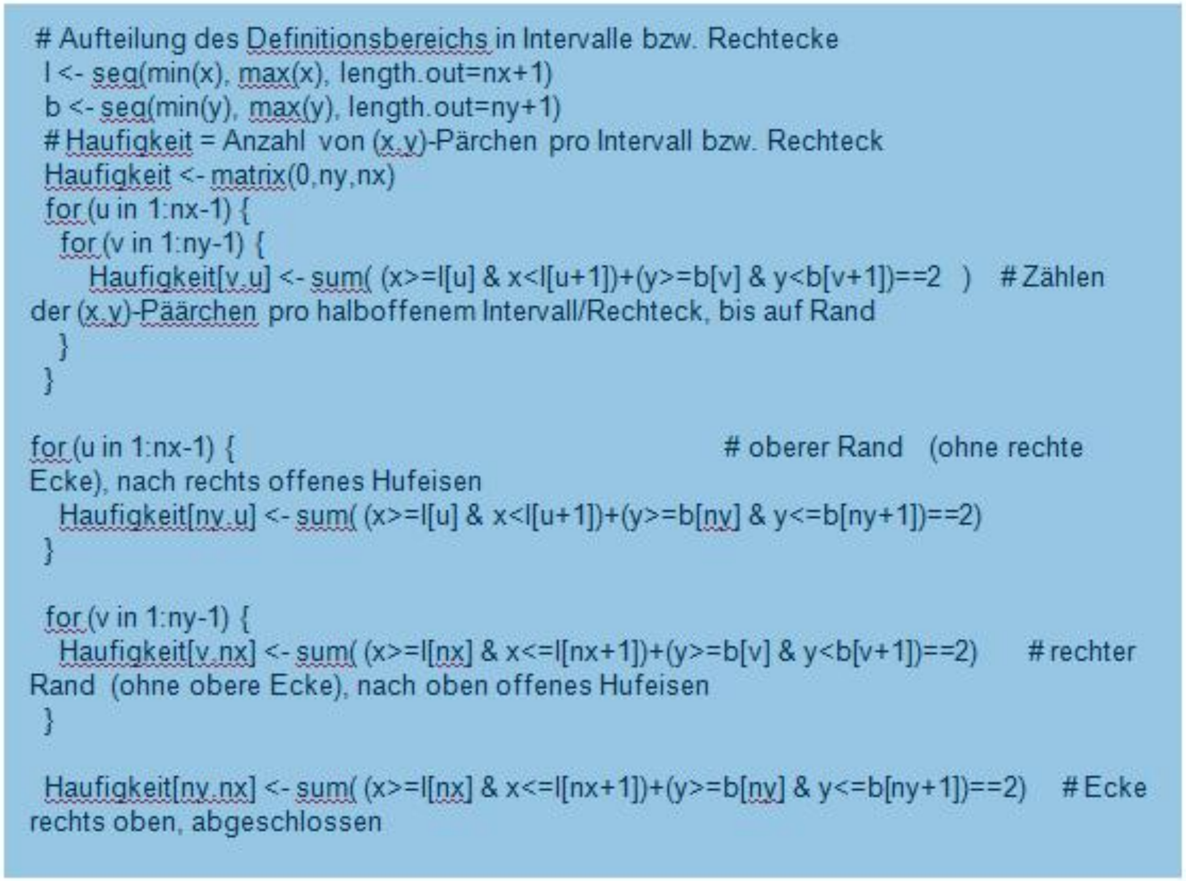
```
 # Aufteilung des Definitionsbereichs in Intervalle bzw. Rechtecke
  l <- seq(min(x), max(x), length.out=nx+1)
  b <- seq(min(y), max(y), length.out=ny+1)
  # Haufigkeit = Anzahl  von (x,y)-Pärchen  pro Intervall bzw. Rechteck
  Haufigkeit <- matrix(0,ny,nx)
  for (u in 1:nx-1) {
   for (v in 1:ny-1) {
      Haufigkeit[v,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[v] & y<b[v+1])==2   )    # Zählen
der (x,y)-Päärchen  pro halboffenem Intervall/Rechteck, bis auf Rand
   }
  }

for (u in 1:nx-1) {                                                    # oberer Rand    (ohne rechte
Ecke), nach rechts offenes Hufeisen
   Haufigkeit[ny,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[ny] & y<=b[ny+1])==2)
  }

  for (v in 1:ny-1) {
   Haufigkeit[v,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[v] & y<b[v+1])==2)      # rechter
Rand  (ohne obere Ecke), nach oben offenes Hufeisen
  }

  Haufigkeit[ny,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[ny] & y<=b[ny+1])==2)    # Ecke
rechts oben, abgeschlossen
```

```
  ###############  2. barplot-Werte in feineres Gitter ###############

     f <- function(x1,y1) {
  #Haufigkeit_big = Große Matrix, Blöcke mit Werten  von Häufigkeit, entspricht einer
Verfeinerung des Gitters
  Haufigkeit_big <- matrix(0,length(y1)-1,length(x1)-1)
  p<-1;k<-1
  for (p in 1:(length(x1)-2))  {
   for (q in 1:(length(y1)-2))  {
     for (k in 1:nx) { if (x1[p]>=l[k] & x1[p]<l[k+1]) { i <- k } }           # Übernehme
entsprechenden Wert  von Haufigkeit für den passenden Bereich in Haufigkeit_big
     for (k in 1:ny) { if (y1[q]>=b[k] & y1[q]<b[k+1]) { j <- k } }
     Haufigkeit_big[q,p] <- Haufigkeit[j,i]
   }
  }
}
```

```
 for (p in 1:(length(x1)-2))  {                          # oberer Rand   (ohne rechte Ecke), nach
rechts offenes Hufeisen
   for (k in 1:nx) {
    if ( x1[p]>=l[k] & x1[p]<l[k+1]) {i <-k}
   }
   Haufigkeit_big[length(y1)-1,p]<- Haufigkeit[ny,i]
 for (q in 1:(length(y1)-2))  {                          # rechter Rand  (ohne obere Ecke), nach
oben offenes Hufeisen
   for (k in 1:ny) {
     if ( y1[q]>=b[k] & y1[q]<b[k+1]) {j <-k}
   }
   Haufigkeit_big[q,length(x1)-1]<- Haufigkeit[j,nx]
  }

  Haufigkeit_big[length(y1)-1,length(x1)-1]<-Haufigkeit[ny,nx]  # rechte obere Ecke

  return(Haufigkeit_big)
  }

  # Vorgabe des feineres Gitters
  xnew <- seq(min(x), max(x), length=1+ approx*nx/10)
  ynew <- seq(min(y), max(y), length=1+ approx*ny/10)
  z <- f(xnew,ynew)

  ###############  plotten des feinen Gitters ###############

  y2 <- seq(min(xnew),max(xnew),length.out=nrow(z))      # Intervalle
  x2 <- seq(min(ynew),max(ynew),  length.out=ncol(z))

 persp(x2,y2,z,
     xlim = range(x2), ylim = range(y2),
     zlim = range(z,  na.rm = TRUE),
     xlab = xlab, ylab = ylab, zlab = zlab,
     main = main, sub = sub,
     theta = theta, phi = phi, r = r, d = d,
     scale = scale, expand = expand,
     col = col, border = border, ltheta = ltheta, lphi = lphi,
     shade = shade, box = box, axes = axes, nticks = nticks,
     ticktype = ticktype,
     cex.lab =cex.lab , font.lab =font.lab,
     cex.axis=cex.axis, font.axis=font.axis
     )
```

Computing the barplot values

Computing the barplot values



```
# Aufteilung des Definitionsbereichs in Intervalle bzw. Rechtecke
 l <- seq(min(x), max(x), length.out=nx+1)
 b <- seq(min(y), max(y), length.out=ny+1)
 # Haufigkeit = Anzahl von (x,y)-Pärchen pro Intervall bzw. Rechteck
 Haufigkeit <- matrix(0,ny,nx)
 for (u in 1:nx-1) {
   for (v in 1:ny-1) {
      Haufigkeit[v,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[v] & y<b[v+1])==2   )    # Zählen
der (x,y)-Päärchen pro halboffenem Intervall/Rechteck, bis auf Rand
   }
 }

for (u in 1:nx-1) {                                          # oberer Rand   (ohne rechte
Ecke), nach rechts offenes Hufeisen
   Haufigkeit[ny,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[ny] & y<=b[ny+1])==2)
 }

 for (v in 1:ny-1) {
   Haufigkeit[v,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[v] & y<b[v+1])==2)      # rechter
Rand  (ohne obere Ecke), nach oben offenes Hufeisen
 }

 Haufigkeit[ny,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[ny] & y<=b[ny+1])==2)     # Ecke
rechts oben, abgeschlossen
```

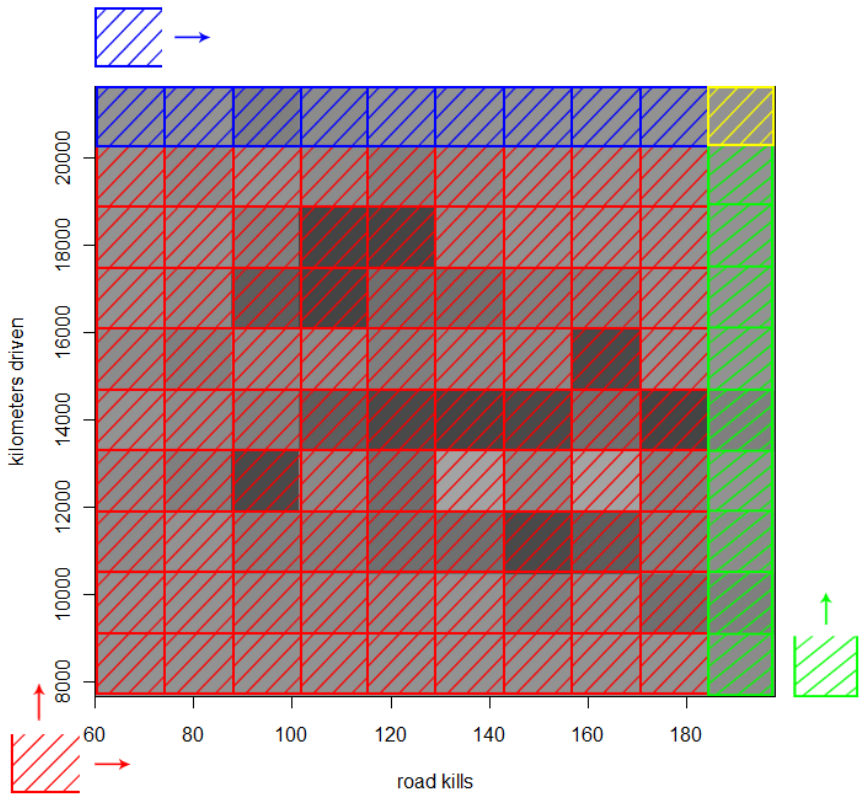
Computing the barplot values

1. Search breakpoints for intervalls (from min(x) to max(x))
2. Initialise matrix of #intervals x #intervals
3. Fill „inner“ with L-shaped squares
4. Fill upper margin with C-shaped squares
5. Fill right margin with U-shaped squares
6. Fill upper right corner with closed box

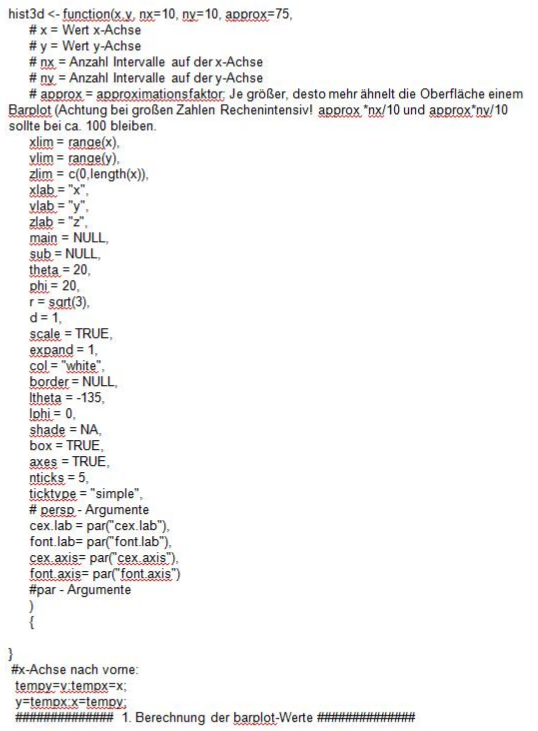
$\Rightarrow$ No datapoint is counted twice!

$\Rightarrow$ Matrix „Haufigkeit“ is filled with frequencie-values of the intervals

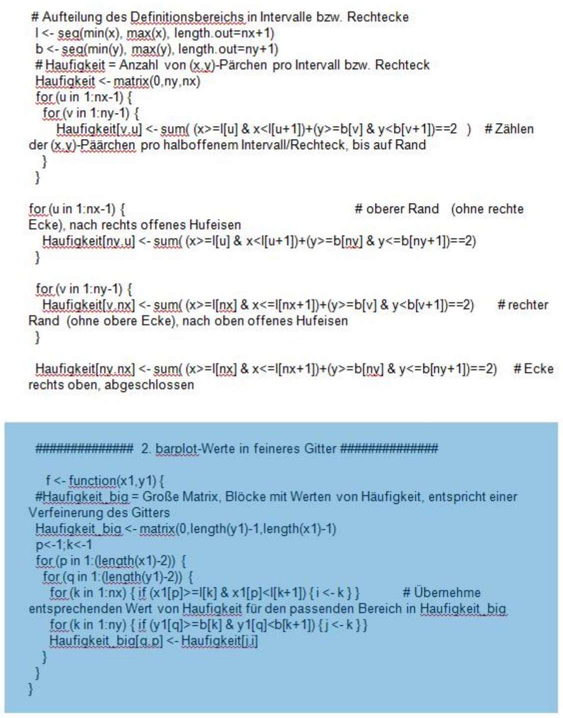
Computing the barplot values



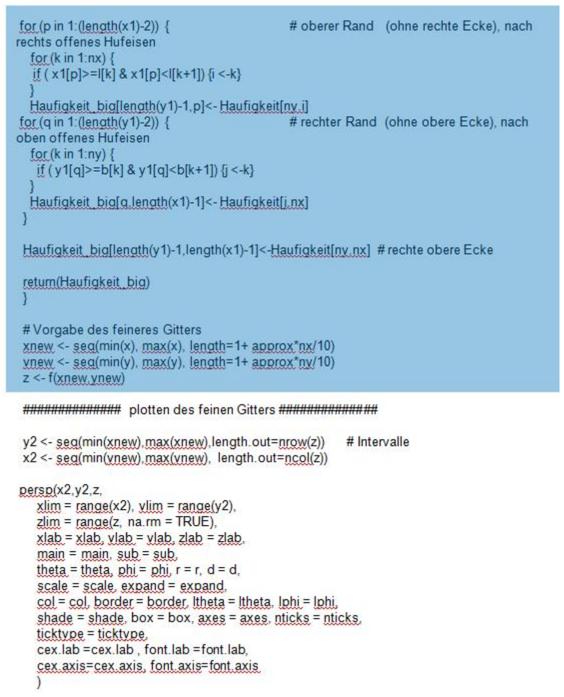
The Code – step by step



```
hist3d <- function(x,y, nx=10, ny=10, approx=75,
     # x = Wert x-Achse
     # y = Wert y-Achse
     # nx = Anzahl Intervalle auf der x-Achse
     # ny = Anzahl Intervalle auf der y-Achse
     # approx = approximationsfaktor: Je größer, desto mehr ähnelt die Oberfläche einem
Barplot (Achtung bei großen Zahlen Rechenintensiv! approx *nx/10 und approx*ny/10
sollte bei ca. 100 bleiben.
     xlim = range(x),
     ylim = range(y),
     zlim = c(0,length(x)),
     xlab = "x",
     ylab = "y",
     zlab = "z",
     main = NULL,
     sub = NULL,
     theta = 20,
     phi = 20,
     r = sqrt(3),
     d = 1,
     scale = TRUE,
     expand = 1,
     col = "white",
     border = NULL,
     ltheta = -135,
     lphi = 0,
     shade = NA,
     box = TRUE,
     axes = TRUE,
     nticks = 5,
     ticktype = "simple",
     # persp - Argumente
     cex.lab = par("cex.lab"),
     font.lab= par("font.lab"),
     cex.axis= par("cex.axis"),
     font.axis= par("font.axis")
     #par - Argumente
     )
     {

}
 #x-Achse nach vorne:
  tempy=y;tempx=x;
  y=tempx;x=tempy;
  ##############  1. Berechnung der barplot-Werte ##############
```



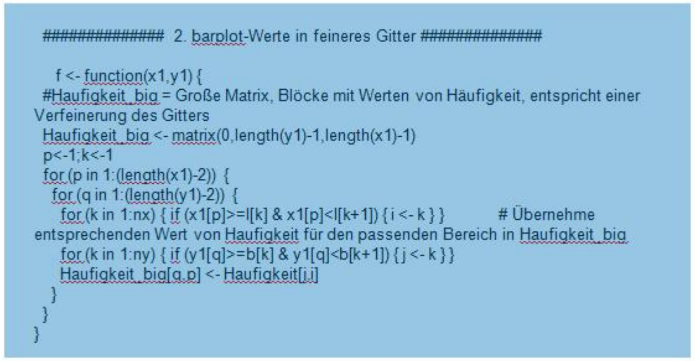
```
 # Aufteilung des Definitionsbereichs in Intervalle bzw. Rechtecke
  l <- seq(min(x), max(x), length.out=nx+1)
  b <- seq(min(y), max(y), length.out=ny+1)
  # Haufigkeit = Anzahl von (x,y)-Pärchen pro Intervall bzw. Rechteck
  Haufigkeit <- matrix(0,ny,nx)
  for (u in 1:nx-1) {
    for (v in 1:ny-1) {
       Haufigkeit[v,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[v] & y<b[v+1])==2  )   # Zählen
der (x,y)-Päärchen pro halboffenem Intervall/Rechteck, bis auf Rand
    }
  }

for (u in 1:nx-1) {                                        # oberer Rand   (ohne rechte
Ecke), nach rechts offenes Hufeisen
    Haufigkeit[ny,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[ny] & y<=b[ny+1])==2)
  }

  for (v in 1:ny-1) {
    Haufigkeit[v,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[v] & y<b[v+1])==2)      # rechter
Rand  (ohne obere Ecke), nach oben offenes Hufeisen
  }

  Haufigkeit[ny,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[ny] & y<=b[ny+1])==2)    # Ecke
rechts oben, abgeschlossen

  ##############  2. barplot-Werte in feineres Gitter ##############

     f <- function(x1,y1) {
  #Haufigkeit_big = Große Matrix, Blöcke mit Werten von Häufigkeit, entspricht einer
Verfeinerung des Gitters
  Haufigkeit_big <- matrix(0,length(y1)-1,length(x1)-1)
  p<-1;k<-1
  for (p in 1:(length(x1)-2)) {
    for (q in 1:(length(y1)-2)) {
      for (k in 1:nx) { if (x1[p]>=l[k] & x1[p]<l[k+1]) { i <- k } }          # Übernehme
entsprechenden Wert von Haufigkeit für den passenden Bereich in Haufigkeit_big
      for (k in 1:ny) { if (y1[q]>=b[k] & y1[q]<b[k+1]) { j <- k } }
      Haufigkeit_big[q,p] <- Haufigkeit[j,i]
    }
  }
}
```



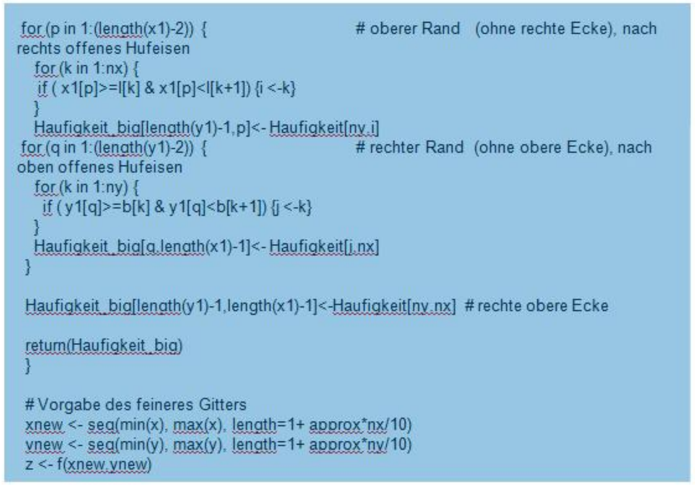
```
 for (p in 1:(length(x1)-2)) {                       # oberer Rand   (ohne rechte Ecke), nach
rechts offenes Hufeisen
   for (k in 1:nx) {
    if ( x1[p]>=l[k] & x1[p]<l[k+1]) {i <-k}
   }
   Haufigkeit_big[length(y1)-1,p]<- Haufigkeit[ny,i]
 for (q in 1:(length(y1)-2)) {                       # rechter Rand  (ohne obere Ecke), nach
oben offenes Hufeisen
   for (k in 1:ny) {
     if ( y1[q]>=b[k] & y1[q]<b[k+1]) {j <-k}
   }
   Haufigkeit_big[q,length(x1)-1]<- Haufigkeit[j,nx]
  }

  Haufigkeit_big[length(y1)-1,length(x1)-1]<-Haufigkeit[ny,nx]  # rechte obere Ecke

  return(Haufigkeit_big)
  }

  # Vorgabe des feineres Gitters
  xnew <- seq(min(x), max(x), length=1+ approx*nx/10)
  ynew <- seq(min(y), max(y), length=1+ approx*ny/10)
  z <- f(xnew,ynew)

  ##############  plotten des feinen Gitters ##############

  y2 <- seq(min(xnew),max(xnew),length.out=nrow(z))      # Intervalle
  x2 <- seq(min(ynew),max(ynew),  length.out=ncol(z))

 persp(x2,y2,z,
     xlim = range(x2), ylim = range(y2),
     zlim = range(z,  na.rm = TRUE),
     xlab = xlab, ylab = ylab, zlab = zlab,
     main = main, sub = sub,
     theta = theta, phi = phi, r = r, d = d,
     scale = scale, expand = expand,
     col = col, border = border, ltheta = ltheta, lphi = lphi,
     shade = shade, box = box, axes = axes, nticks = nticks,
     ticktype = ticktype,
     cex.lab =cex.lab , font.lab =font.lab,
     cex.axis=cex.axis, font.axis=font.axis
     )
```

Computing the finer grid

Computing the finer grid



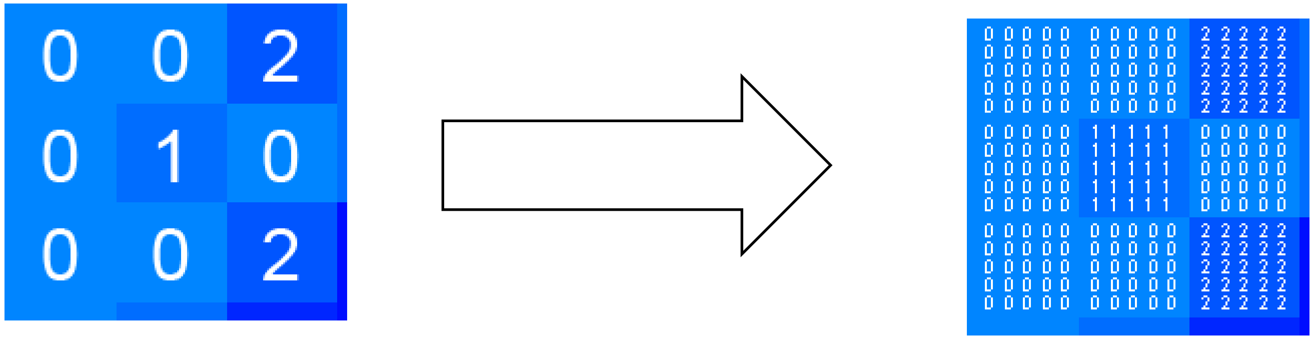
```
   ##############  2. barplot-Werte in feineres Gitter ##############

     f <- function(x1,y1) {
   #Haufigkeit_big = Große Matrix, Blöcke mit Werten von Häufigkeit, entspricht einer
  Verfeinerung des Gitters
   Haufigkeit_big <- matrix(0,length(y1)-1,length(x1)-1)
   p<-1;k<-1
   for (p in 1:(length(x1)-2)) {
    for (q in 1:(length(y1)-2)) {
     for (k in 1:nx) { if (x1[p]>=l[k] & x1[p]<l[k+1]) { i <- k } }          # Übernehme
  entsprechenden Wert von Haufigkeit für den passenden Bereich in Haufigkeit_big
     for (k in 1:ny) { if (y1[q]>=b[k] & y1[q]<b[k+1]) { j <- k } }
     Haufigkeit_big[q,p] <- Haufigkeit[j,i]
    }
   }
  }
```



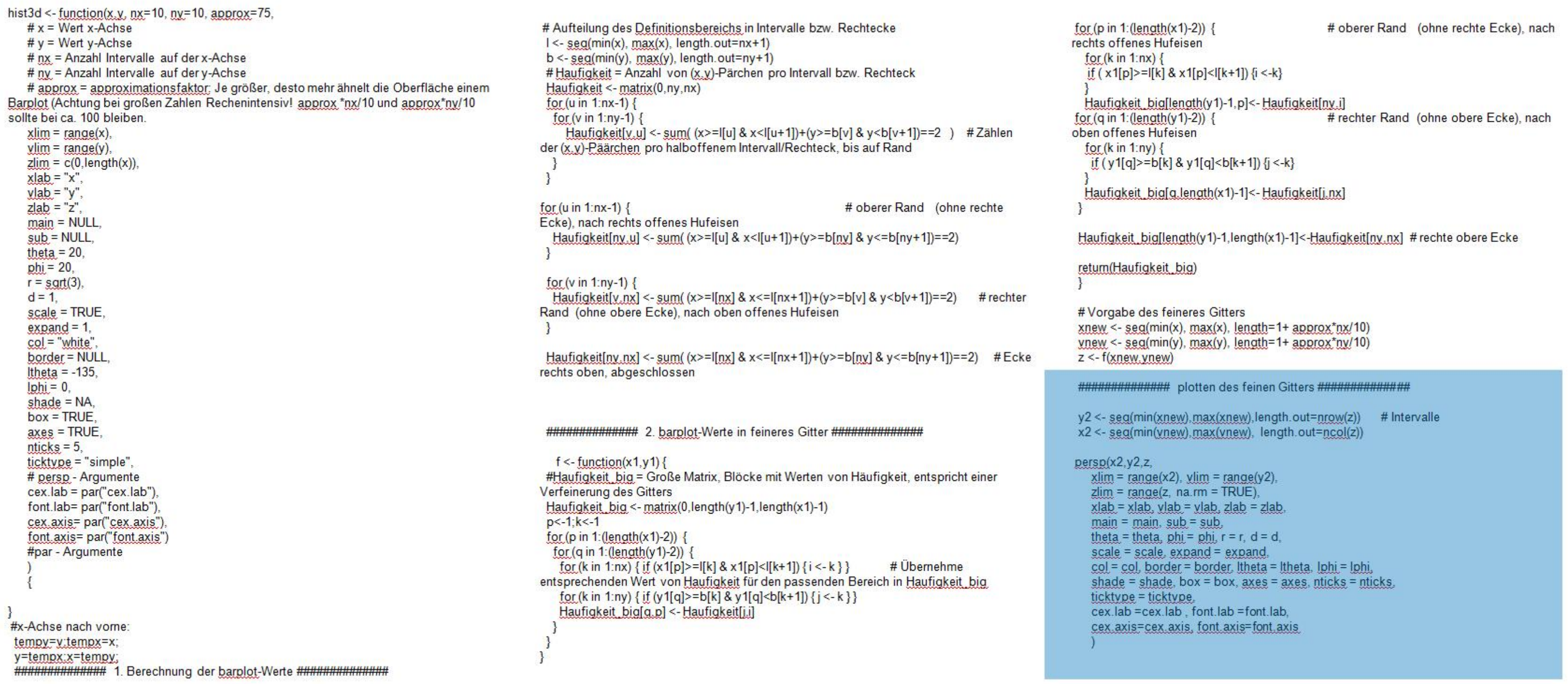
```
 for (p in 1:(length(x1)-2)) {                          # oberer Rand   (ohne rechte Ecke), nach
rechts offenes Hufeisen
   for (k in 1:nx) {
    if ( x1[p]>=l[k] & x1[p]<l[k+1]) {i <-k}
   }
   Haufigkeit_big[length(y1)-1,p]<- Haufigkeit[ny,i]
 for (q in 1:(length(y1)-2)) {                          # rechter Rand  (ohne obere Ecke), nach
oben offenes Hufeisen
   for (k in 1:ny) {
    if ( y1[q]>=b[k] & y1[q]<b[k+1]) {j <-k}
   }
   Haufigkeit_big[q,length(x1)-1]<- Haufigkeit[j,nx]
  }

  Haufigkeit_big[length(y1)-1,length(x1)-1]<-Haufigkeit[ny,nx]  # rechte obere Ecke

  return(Haufigkeit_big)
  }

  # Vorgabe des feineres Gitters
  xnew <- seq(min(x), max(x), length=1+ approx*nx/10)
  ynew <- seq(min(y), max(y), length=1+ approx*ny/10)
  z <- f(xnew,ynew)
```

Computing the finer grid

1. Create more breakpoints (length.out= 1 * approx/10 * nx)
2. Start same procedure…
3. …but fill new, larger matrix with values from old, smaller matrix



The Code – step by step



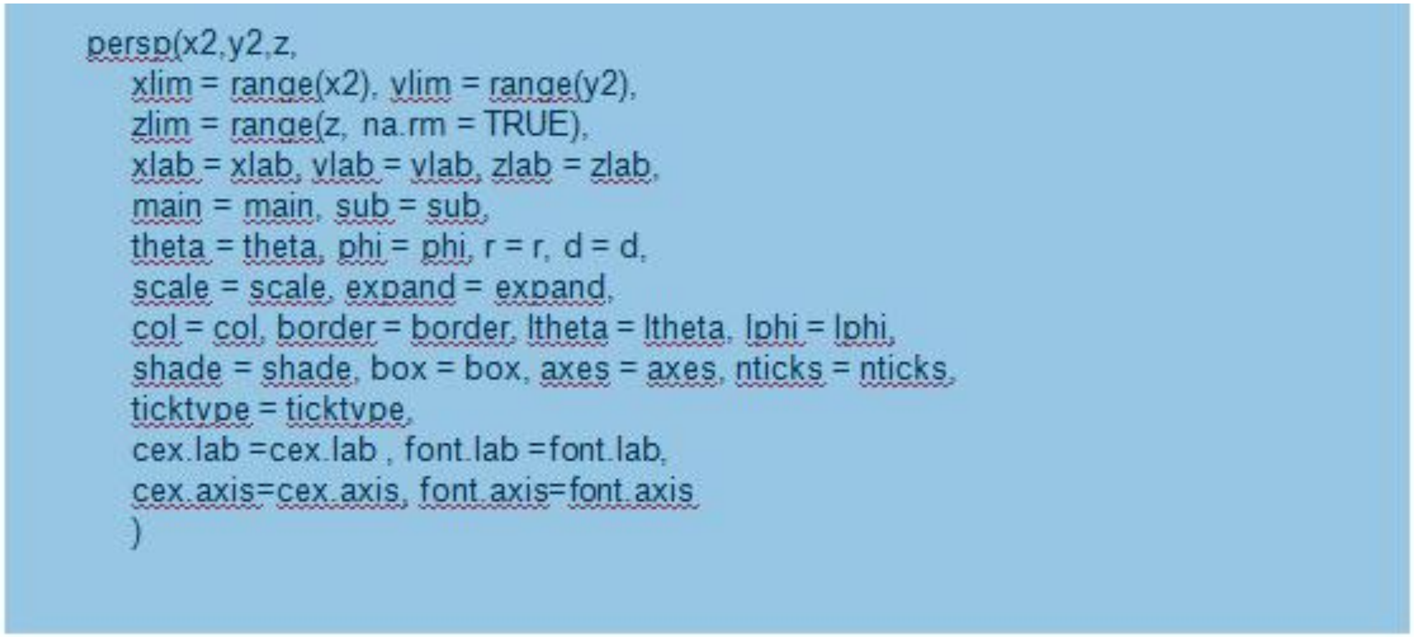
```
hist3d <- function(x,y, nx=10, ny=10, approx=75,
     # x = Wert x-Achse
     # y = Wert y-Achse
     # nx = Anzahl Intervalle  auf der x-Achse
     # ny = Anzahl Intervalle  auf der y-Achse
     # approx = approximationsfaktor: Je größer, desto mehr ähnelt die Oberfläche einem
Barplot (Achtung bei großen Zahlen Rechenintensiv! approx *nx/10 und approx*ny/10
sollte bei ca. 100 bleiben.
     xlim = range(x),
     ylim = range(y),
     zlim = c(0,length(x)),
     xlab = "x",
     ylab = "y",
     zlab = "z",
     main = NULL,
     sub = NULL,
     theta = 20,
     phi = 20,
     r = sqrt(3),
     d = 1,
     scale = TRUE,
     expand = 1,
     col = "white",
     border = NULL,
     ltheta = -135,
     lphi = 0,
     shade = NA,
     box = TRUE,
     axes = TRUE,
     nticks = 5,
     ticktype = "simple",
     # persp - Argumente
     cex.lab = par("cex.lab"),
     font.lab= par("font.lab"),
     cex.axis= par("cex.axis"),
     font.axis= par("font.axis")
     #par - Argumente
     )
     {

}
 #x-Achse nach vorne:
  tempy=y;tempx=x;
  y=tempx;x=tempy;
  ###############  1. Berechnung der barplot-Werte ###############
```

```
 # Aufteilung des Definitionsbereichs in Intervalle bzw. Rechtecke
  l <- seq(min(x), max(x), length.out=nx+1)
  b <- seq(min(y), max(y), length.out=ny+1)
  # Haufigkeit = Anzahl  von (x,y)-Pärchen  pro Intervall bzw. Rechteck
  Haufigkeit <- matrix(0,ny,nx)
  for (u in 1:nx-1) {
   for (v in 1:ny-1) {
      Haufigkeit[v,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[v] & y<b[v+1])==2  )   # Zählen
der (x,y)-Päärchen pro halboffenem Intervall/Rechteck, bis auf Rand
   }
  }

for (u in 1:nx-1) {                                        # oberer Rand   (ohne rechte
Ecke), nach rechts offenes Hufeisen
   Haufigkeit[ny,u] <- sum( (x>=l[u] & x<l[u+1])+(y>=b[ny] & y<=b[ny+1])==2)
  }

  for (v in 1:ny-1) {
   Haufigkeit[v,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[v] & y<b[v+1])==2)      # rechter
Rand  (ohne obere Ecke), nach oben offenes Hufeisen
  }

  Haufigkeit[ny,nx] <- sum( (x>=l[nx] & x<=l[nx+1])+(y>=b[ny] & y<=b[ny+1])==2)    # Ecke
rechts oben, abgeschlossen



  ###############  2. barplot-Werte in feineres Gitter ###############

     f <- function(x1,y1) {
  #Haufigkeit_big = Große Matrix, Blöcke mit Werten von Häufigkeit, entspricht einer
Verfeinerung des Gitters
  Haufigkeit_big <- matrix(0,length(y1)-1,length(x1)-1)
  p<-1;k<-1
  for (p in 1:(length(x1)-2)) {
   for (q in 1:(length(y1)-2)) {
     for (k in 1:nx) { if (x1[p]>=l[k] & x1[p]<l[k+1]) { i <- k } }           # Übernehme
entsprechenden Wert von Haufigkeit für den passenden Bereich in Haufigkeit_big
     for (k in 1:ny) { if (y1[q]>=b[k] & y1[q]<b[k+1]) { j <- k } }
     Haufigkeit_big[q,p] <- Haufigkeit[j,i]
   }
  }
}
```

```
 for (p in 1:(length(x1)-2)) {                     # oberer Rand   (ohne rechte Ecke), nach
rechts offenes Hufeisen
    for (k in 1:nx) {
     if ( x1[p]>=l[k] & x1[p]<l[k+1]) {i <-k}
    }
    Haufigkeit_big[length(y1)-1,p]<- Haufigkeit[ny,i]
 for (q in 1:(length(y1)-2)) {                     # rechter Rand  (ohne obere Ecke), nach
oben offenes Hufeisen
    for (k in 1:ny) {
      if ( y1[q]>=b[k] & y1[q]<b[k+1]) {j <-k}
    }
    Haufigkeit_big[q,length(x1)-1]<- Haufigkeit[j,nx]
  }

  Haufigkeit_big[length(y1)-1,length(x1)-1]<-Haufigkeit[ny,nx]  # rechte obere Ecke

  return(Haufigkeit_big)
  }

  # Vorgabe des feineres Gitters
  xnew <- seq(min(x), max(x), length=1+ approx*nx/10)
  ynew <- seq(min(y), max(y), length=1+ approx*ny/10)
  z <- f(xnew,ynew)

  ###############  plotten des feinen Gitters ###############

  y2 <- seq(min(xnew),max(xnew),length.out=nrow(z))      # Intervalle
  x2 <- seq(min(ynew),max(ynew),  length.out=ncol(z))

 persp(x2,y2,z,
     xlim = range(x2), ylim = range(y2),
     zlim = range(z, na.rm = TRUE),
     xlab = xlab, ylab = ylab, zlab = zlab,
     main = main, sub = sub,
     theta = theta, phi = phi, r = r, d = d,
     scale = scale, expand = expand,
     col = col, border = border, ltheta = ltheta, lphi = lphi,
     shade = shade, box = box, axes = axes, nticks = nticks,
     ticktype = ticktype,
     cex.lab =cex.lab , font.lab =font.lab,
     cex.axis=cex.axis, font.axis=font.axis
     )
```

Plotting with persp()

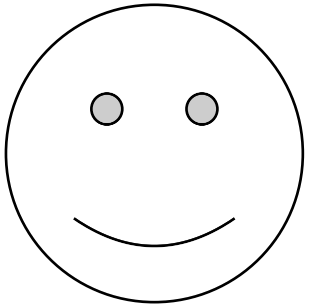
Plotting with persp()



```
persp(x2,y2,z,
    xlim = range(x2), ylim = range(y2),
    zlim = range(z, na.rm = TRUE),
    xlab = xlab, ylab = ylab, zlab = zlab,
    main = main, sub = sub,
    theta = theta, phi = phi, r = r, d = d,
    scale = scale, expand = expand,
    col = col, border = border, ltheta = ltheta, lphi = lphi,
    shade = shade, box = box, axes = axes, nticks = nticks,
    ticktype = ticktype,
    cex.lab =cex.lab , font.lab =font.lab,
    cex.axis=cex.axis, font.axis=font.axis
    )
```

Plotting with persp()

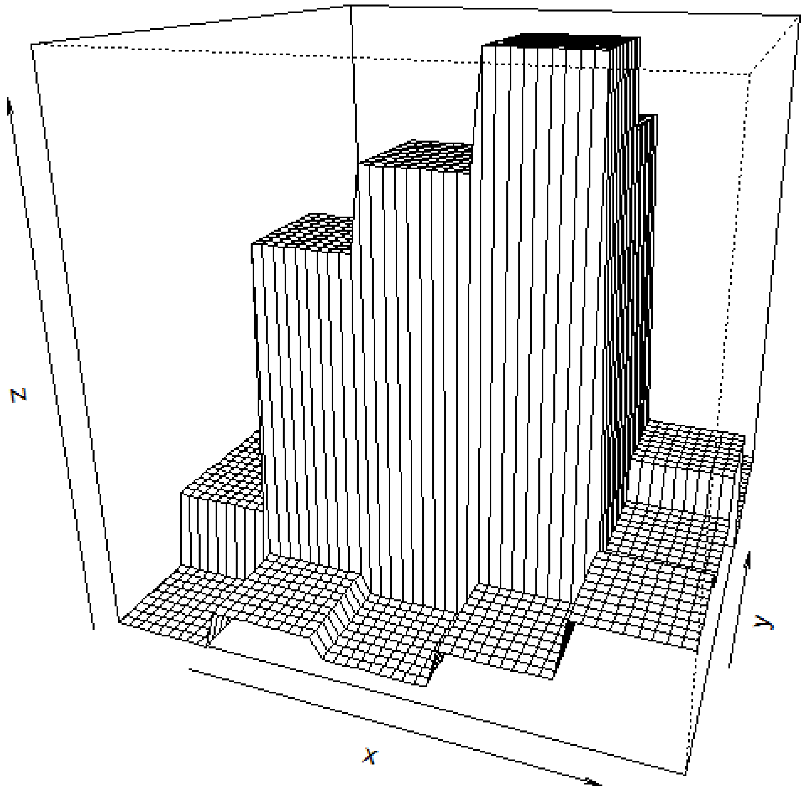
- Plot the fine grid, respectively the new, large matrix
- …and use all handed-through parameters



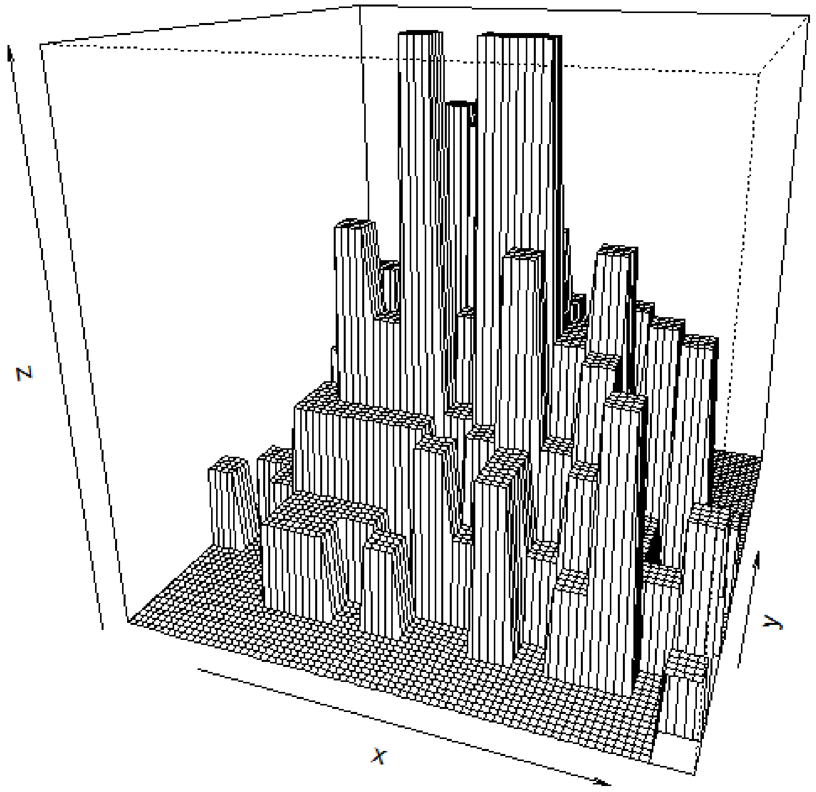
II. 3-dimensional barplots: hist3d

iv. Results

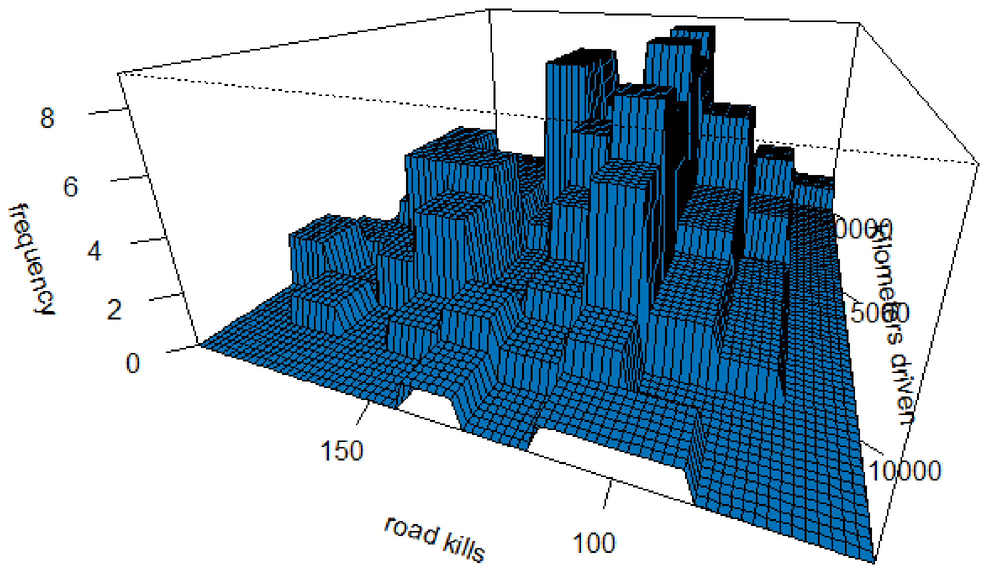
Results



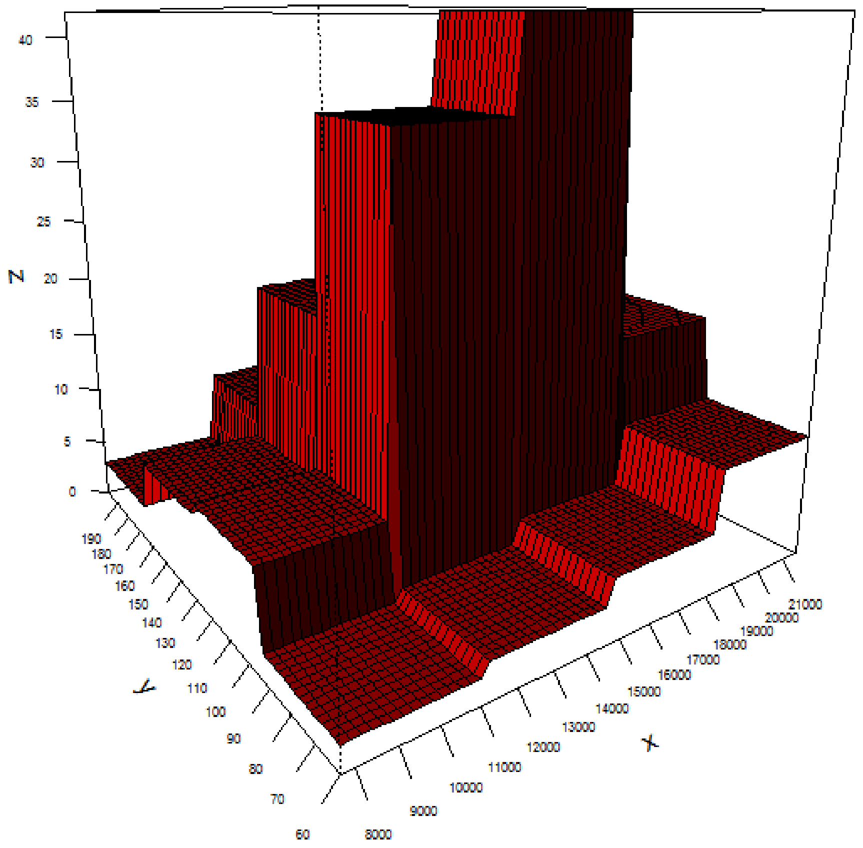
Results



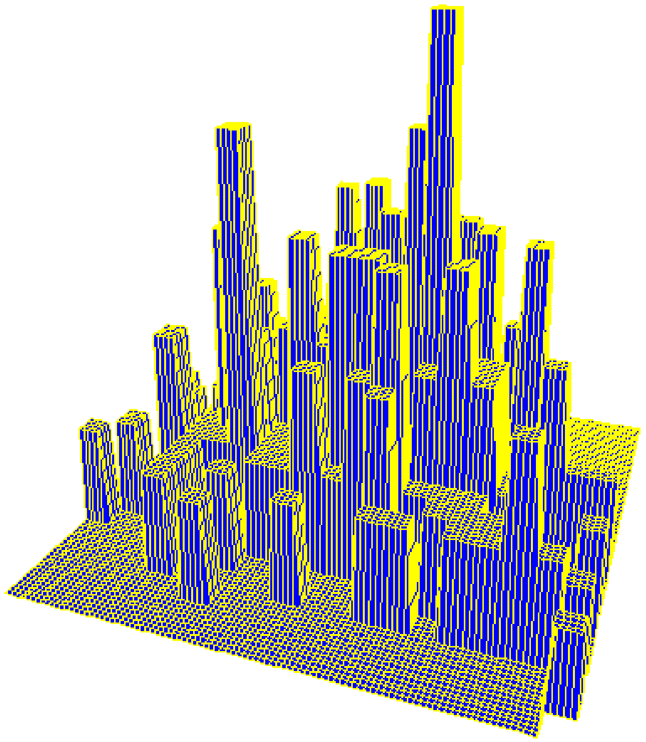
Results


8
6
4
2
0
frequency
150
100
road kills
5000
10000
kilometers driven

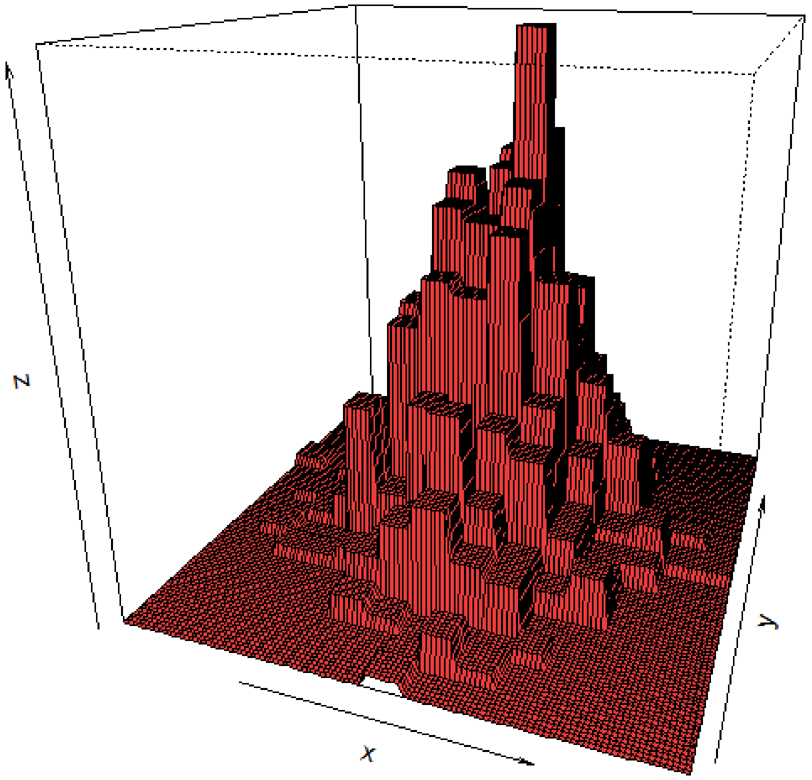
Results


Z
0
5
10
15
20
25
30
35
40
Y
60
70
80
90
100
110
120
130
140
150
160
170
180
190
X
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000
20000
21000

Results



Results



II. 3-dimensional barplots: hist3d

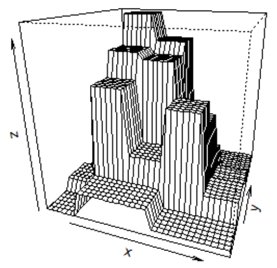
v. hist3drot

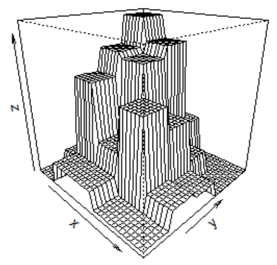
hist3d‘s little sister

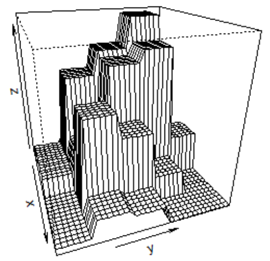
hist3drot

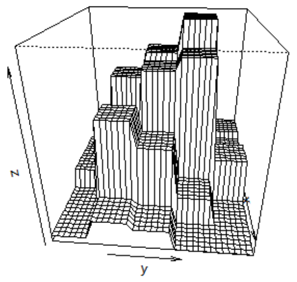
- No rotation in R graphic-window possible

=> hist3drot does that for you









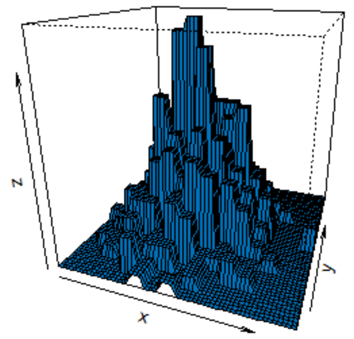
hist3drot

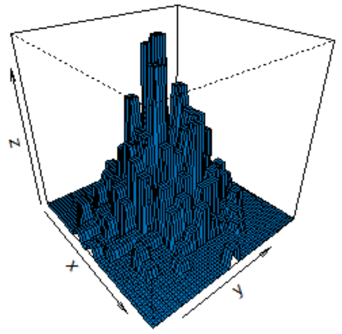
- Two additional parameters:
  - dphi
  - dtheta
- …specify phi and theta rotation in each new plot
- Four plots alltogether (par(mfrow=c(2,2)))

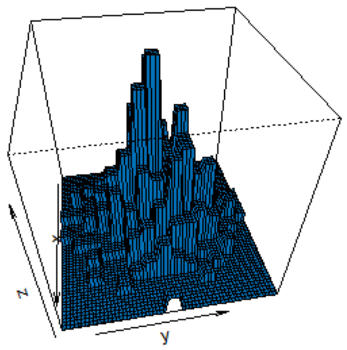
Code

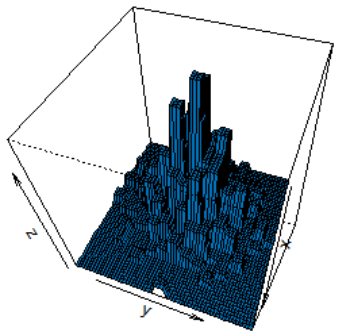
```
par(mfrow=c(2,2))
persp(x2,y2,z,theta = theta, phi = phi,     …
persp(x2,y2,z,theta = theta+dtheta, phi = phi+dphi,   …
persp(x2,y2,z,theta = theta+2*dtheta, phi = phi+2*dphi,    …
persp(x2,y2,z,theta = theta+3*dtheta, phi = phi+3*dphi,     …
par(mfrow=c(1,1))
```

hist3drot









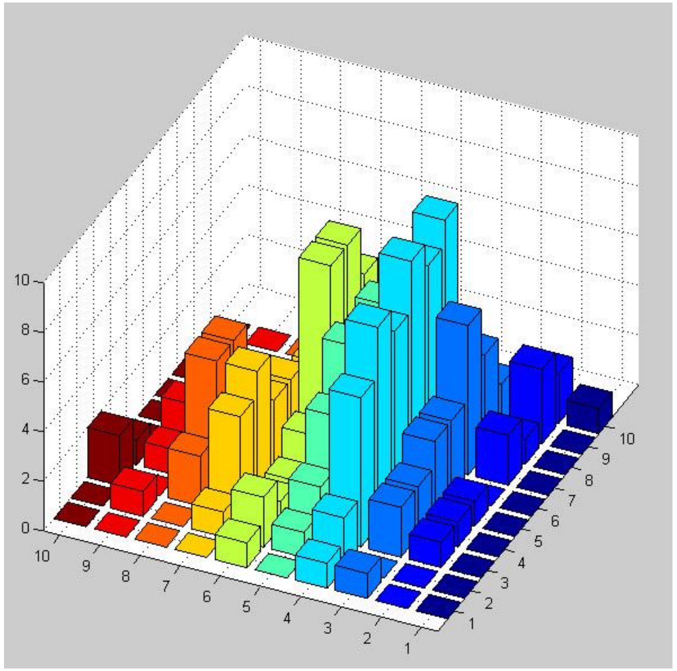
II. 3-dimensional barplots: hist3d

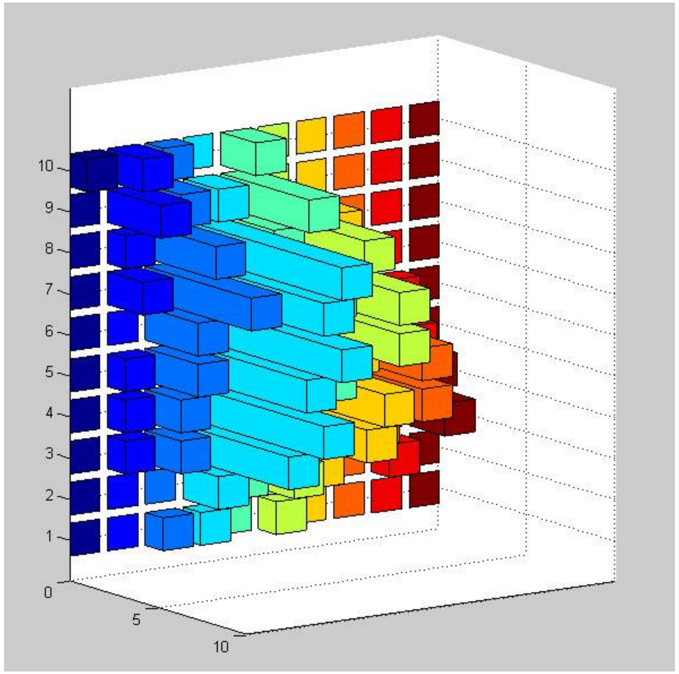
vi. Alternatives

Plotting 3-dimensional bars

Alternatives: Matlab

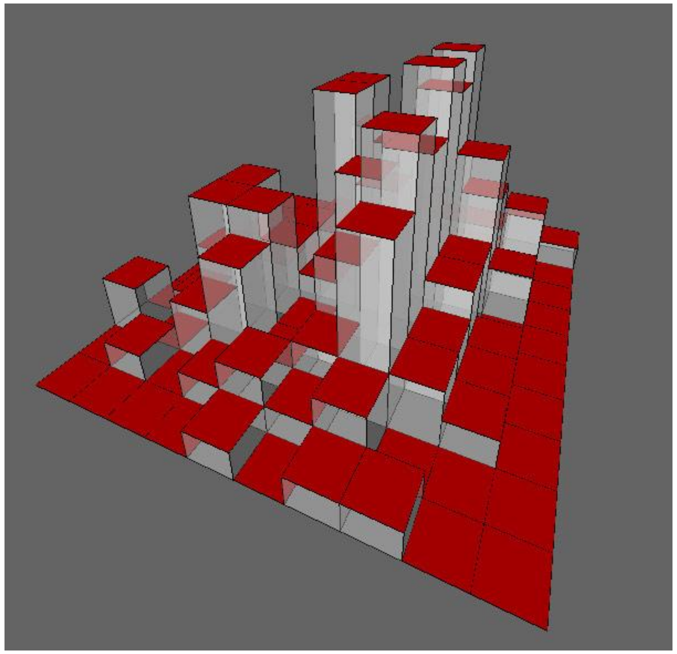
- Matlab does provide 3-dimensional barplots
- „bar3“ function

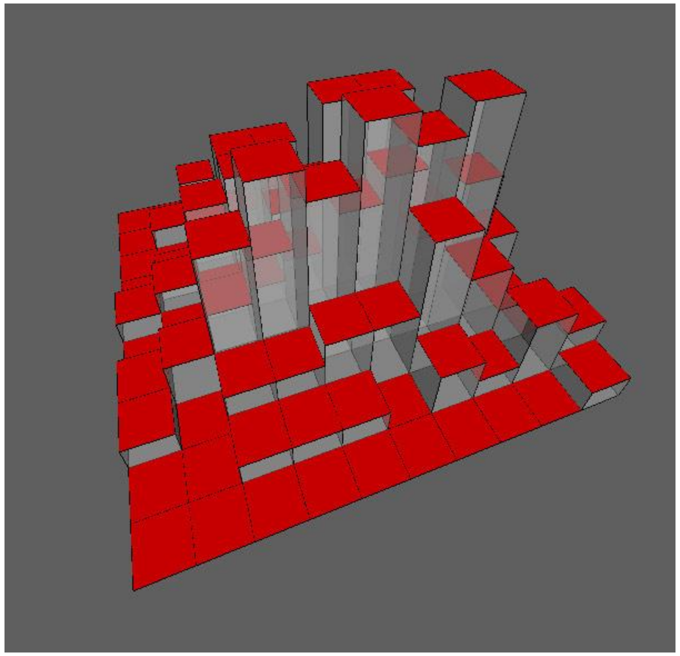




Alternatives: RGL package

- Rgl package provides a sophisticated plotting engine
- „hist3d“ function





III. Plotting maps with R

i. The „map“ and „mapproj“ package

Plotting maps

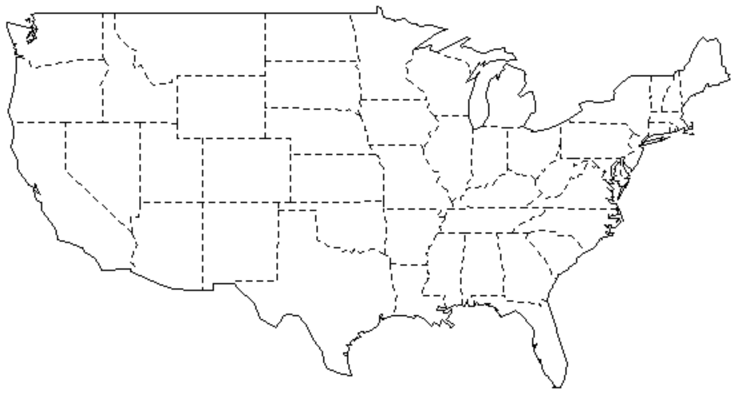
The „map“ and „mapproj“ package

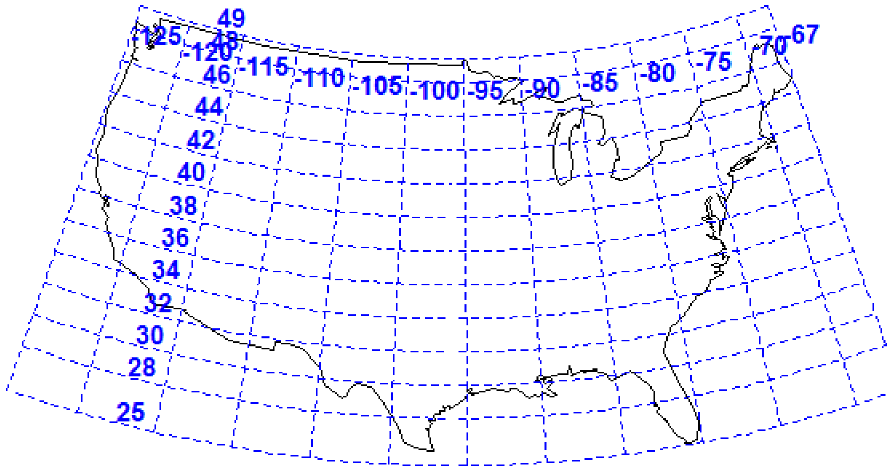
- Includes „map()“ function that plots maps
- Includes „mapproj()“ function that projects maps on spheres etc.

- Documentation of maps package:
  http://cran.r-project.org/web/packages/maps/maps.pdf
- Documentation of mapproj package:
  http://cran.r-project.org/web/packages/mapproj/mapproj.pdf

Example plots with „maps“ and „mapproj“

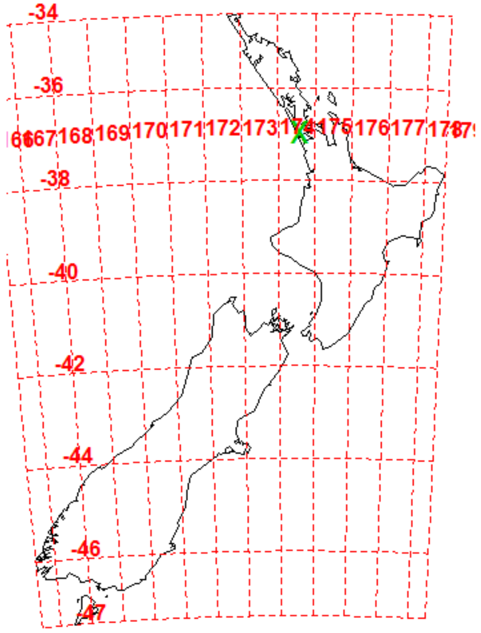
[Source: R documentation]

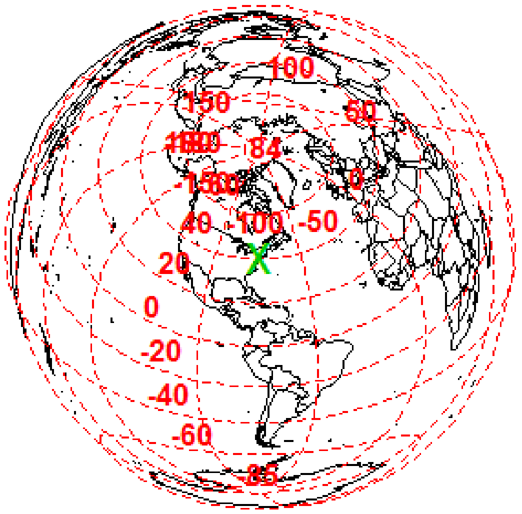




Example plots with „maps“ and „mapproj“

[Source: R documentation]

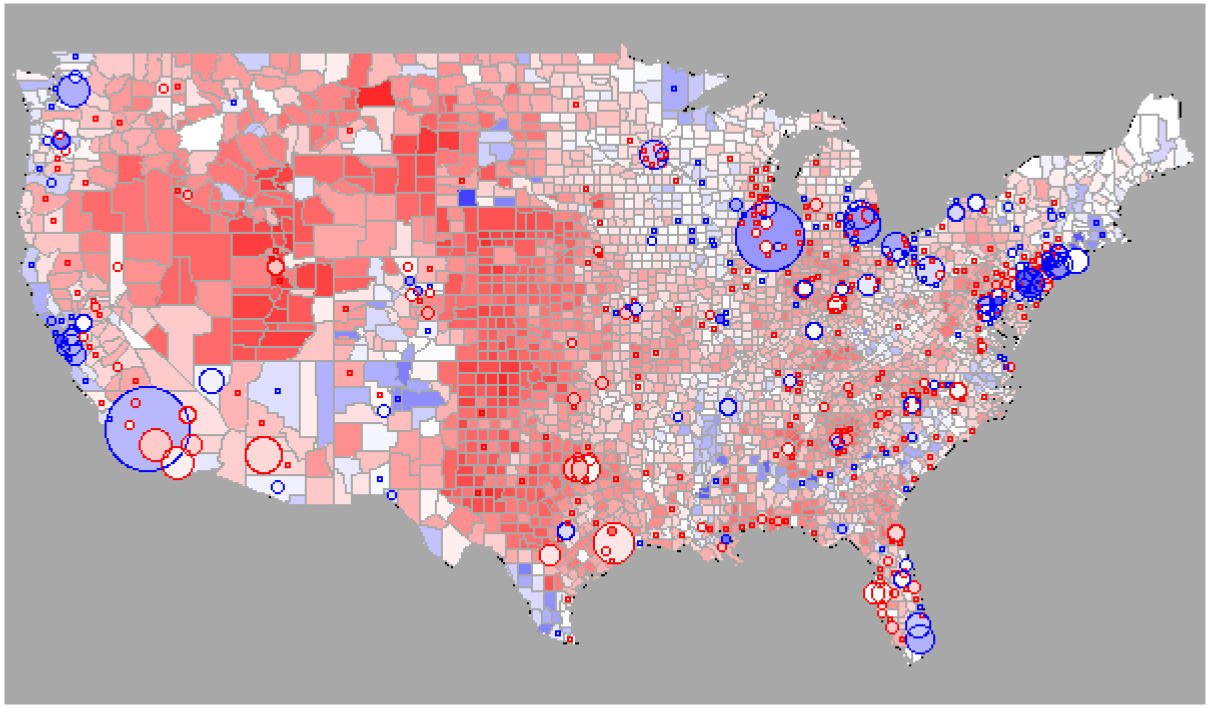




Example plots with „maps“ and „mapproj“

[Source: http://www.ai.rug.nl/~hedderik/R/US2004/map.r]

- Plotting a colour-based „histogram“ with „map()“ is possible…



Example plots with „maps“ and „mapproj“

[Source: http://www.ai.rug.nl/~hedderik/R/US2004/map.r]

- …however code-intensive

```
## ---------------------------------------------------------------------------- ## ## Hedderik van
Rijn, 041105 ## hedderik@ai.rug.nl, Artificial Intelligence Groningen ## ##
http://www.ai.rug.nl/~hedderik/R/election2004map ## ## Written for R 2.0.0+
(because of use of transparancy in plotting) ## ## ----------------------------------------
---------------------------------- ## ## Code to plot a map of the US elections with,
apart from just coloring ## almost each country red, colored circles reflecting the
number of voters ## in each county. (I *do* know about the perceptual problems
with circles - ## but this way it at least looks like Kerry was close :-)). ## ## Plot is
based on the excellent plots found on the New York Times election ## website.
## ## ------------------------------------------------------------------------ ## Load map
plotting code and databases. Library "maps" is not installed by ## default in R 2+,
so, if necessary, install maps first (using something ## like
install.packages("maps",lib="~/Library/R/R-2.0.0") and setting ##
R_LIBS="$HOME/Library/R" in .Renviron. library("maps") ## -------------------------
------------------------------------------------ ## bluered constructs a color based on the
number of votes for B(ush), ## K(erry). If graded==F, color is either blue or red,
otherwise color ## ranges from blue to red, reflecting the proportion of votes for
either ## candidate. S (originally for Size) is transparancy of the color. Only ##
useful when plotting to an OS X quartz device or to ## pdf(...,version="1.4").
Values other than 1 won't show on X11 or other ## devices. bluered <-
function(B,K,S=1,graded=F) { if (graded) { ## Purple plot
rgb(B/(B+K),0,K/(B+K),S); } else { ## All or none
ifelse(B>K,rgb(1,0,0,S),rgb(0,0,1,S)) } } ## map.center returns a dataframe with
the name and coordinates of the ## center of each map region. Code is based on
map.text, can probably be ## improved as often the map database is already
available. map.center <- function (database, regions = ".") { cc =
match.call(expand.dots = TRUE) cc[[1]] = as.name("map") cc$fill = TRUE cc$plot
= FALSE cc$move = cc$add = cc$cex = cc$labels = NULL cc$resolution = 0 m <-
eval(cc) x <- apply.polygon(m, centroid.polygon) x2 <- t(array(unlist(x), c(2,
length(x)))) x <- data.frame(name=names(x),x=x2[,1],y=x2[,2]) } ## -----------------
--------------------------------------------------------- ## Create a map "database".
counties <- map('county',plot=F) ## List of states and abbreviations states <-
c("ALABAMA","AL","ALASKA","AK","AMERICAN
SAMOA","AS","ARIZONA","AZ","ARKANSAS","AR","CALIFORNIA","CA","COLOR
ADO","CO","CONNECTICUT","CT","DELAWARE","DE","DISTRICT OF
COLUMBIA","DC", "FEDERATED STATES OF
MICRONESIA","FM","FLORIDA","FL","GEORGIA","GA","GUAM","GU","HAWAII","
HI","IDAHO","ID","ILLINOIS","IL","INDIANA","IN","IOWA","IA","KANSAS","KS","KE
NTUCKY","KY", "LOUISIANA","LA","MAINE","ME","MARSHALL
ISLANDS","MH","MARYLAND","MD","MASSACHUSETTS","MA","MICHIGAN","MI
","MINNESOTA","MN","MISSISSIPPI","MS","MISSOURI","MO","MONTANA","MT",
"NEBRASKA","NE","NEVADA","NV","NEW HAMPSHIRE","NH","NEW
JERSEY","NJ","NEW MEXICO","NM","NEW YORK","NY","NORTH
CAROLINA","NC","NORTH DAKOTA","ND","NORTHERN MARIANA
ISLANDS","MP",
```

```
"OHIO","OH","OKLAHOMA","OK","OREGON","OR","PALAU","PW","PENNSYLVA
NIA","PA","PUERTO RICO","PR","RHODE ISLAND","RI","SOUTH
CAROLINA","SC","SOUTH DAKOTA","SD","TENNESSEE","TN",
"TEXAS","TX","UTAH","UT","VERMONT","VT","VIRGIN
ISLANDS","VI","VIRGINIA","VA","WASHINGTON","WA","WEST
VIRGINIA","WV","WISCONSIN","WI","WYOMING","WY"); states <-
data.frame(name=states[(1:(length(states)/2))*2-
1],abbrev=states[(1:(length(states)/2))*2]) states$name <-
tolower(as.character(states$name)) states$abbrev <-
as.character(states$abbrev) ## Code used to download data files, files needed
some tweaking. Data files are ## available separately. if (F) { URL <-
"http://www.usatoday.com/news/politicselections/vote2004/PresidentialByCounty.
aspx?oi=P&rti=G&sp=XX&tf=l" for (S in states$abbrev) { URL.s <-
sub("XX",S,URL) cmdLine <- sprintf("/sw/bin/lynx -dump \"%s\" | /sw/bin/gawk '{if
($1==\"County\") output=1; if ($1==\"Updated:\") output=0; if (output) print $0; }' >
%s.dat",URL.s,S) cat("Working on ",S,"\n"); system(cmdLine) } ## Removed '
from .dat files ## Removed extra "County" names from NV.dat } ## Remove
states for which we don't have any data states <-
states[sapply(states$abbrev,function(X){file.exists(sprintf("%s.dat",X))}),] ##
Prepare and read in data election <- NULL for (S in states$abbrev) { ## calls to
gawk not necessary if csv files are downloaded. if (F) { cmdLine <-
sprintf("/sw/bin/gawk -f prep.awk %s.dat > %s.csv",S,S); system(cmdLine) } tmp
<- read.table(sprintf("%s.csv",S),header=T,sep=";") tmp$State <-
states$name[states$abbrev==S] tmp$X <- NULL if (is.null(election)) { election <-
tmp; } else { election <- rbind(election,tmp) } } ## Create a column in which state
and county is combined, as in the map database election$stcounty <-
tolower(paste(election$State,election$County,sep=",")) ## Add a column (order)
to the election df representing the order of ## counties in the map database
countyOrder <- data.frame(stcounty =
counties$names,order=1:length(counties$names)) election <-
merge(election,countyOrder) ## Determine size of county. (N.B., size is
determined by number of voters, ## not by real number of inhabitants.)
election$size <- election$Bush+election$Kerry+election$Nader election$sizeR <-
election$size/max(election$size) ## Create a vector of colors, counties for which
we don't have any data will ## be colored grey, others blue or red depending on
who "won" that county. col <- rep("grey",length(counties$names)) col[tmp$order]
<- ifelse((tmp$Bush>tmp$Kerry),"red","blue") ## Add the county center
coordinates to the election data. county.coord <- map.center("county")
names(county.coord)[1] <- "stcounty"; election <- merge(election,county.coord)
## ---------------------------------------------------------------------------
```

```
## ## And finally, the actual plotting ## ## ---------------------------------------------------
------------------------ ## Real plot, only works when exporting to a recent enough
version of PDF or ## Quartz. If you're using a device that can't handle
transparancy, change ## the .4 and the .6 in the bluered function to 1.
pdf("US04Election-PopGraded.pdf",version="1.4",width=10,height=6.5) ## Draw
the USA outline map('usa',fill=T,col="white",bg="darkgray") ## Create the colors
for the states col <- rep(rgb(.1,.1,.1,.2),length(counties$names))
col[election$order] <- bluered(election$Bush,election$Kerry,.4,graded=T) ## Plot
states without borders (should work with map, see help for fill ## argument, but
cannot get it working) m <- map('county',fill=T,plot=F)
polygon(m$x,m$y,col=col,border=NA) ## Draw county borders
map('county',col="darkgrey",add=T) ## Draw state borders
map('state',col="black",add=T,lwd=1) ## Color for circles col <-
bluered(election$Bush,election$Kerry,.6) ## Symbols is the easiest way to draw
circles that have the right aspect ratio symbols(election$x,
election$y,circles=log(election$sizeR+1)*3,fg=col,bg=col,add=T,inches=F)
',col="darkgrey",add=T) ## Color for circles col <-
bluered(election$Bush,election$Kerry,1,graded=T) ## Symbols is the easiest way
to draw circles that have the right aspect ratio symbols(election$x,
election$y,circles=log(election$sizeR+1)*3,fg=col,bg=col,add=T,inches=F) ## col
<- bluered(election$Bush,election$Kerry,1,graded=F) ## Symbols is the easiest
way to draw circles that have the right aspect ratio symbols(election$x,
election$y,circles=log(election$sizeR+1)*3,fg=col,bg=NA,add=T,inches=F)
dev.off() ## ------------------------------------------------------------------------ dev.off()
## Not sure which version I like more, the previous (with purple counties) ## or
the next one, with blue/red counties. ## Real plot, only works when exporting to a
recent enough version of PDF or Quartz. pdf("US04Election-
PopBin.pdf",version="1.4",width=10,height=6.5) ## Draw the USA outline
map('usa',fill=T,col="white",bg="darkgray") ## Create the colors for the states col
<- rep(rgb(.1,.1,.1,.2),length(counties$names)) col[election$order] <-
bluered(election$Bush,election$Kerry,.4,graded=F) ## Plot states without
borders (should work with map, see help for fill ## argument, but cannot get it
working) m <- map('county',fill=T,plot=F) polygon(m$x,m$y,col=col,border=NA)
## Draw county borders map('county',col="darkgrey",add=T) ## Draw state
borders map('state',col="black",add=T,lwd=2) ## Color for circles col <-
bluered(election$Bush,election$Kerry,.6) ## Symbols is the easiest way to draw
circles that have the right aspect ratio symbols(election$x,
election$y,circles=log(election$sizeR+1)*3,fg=col,bg=col,add=T,inches=F)
dev.off() ## Plot without transparancy, but neutral states colored white. Made
with ## help from Gregoire Thomas. bluered <- function(B,K,S=1,graded=F) { if
(graded) { ## Suggested by Gregoire Thomas: Kn <- K/(B+K) red <-
ifelse(Kn<0.5, 1, 2-2*Kn) blue <- ifelse(Kn<0.5, 2*Kn, 1) green <- ifelse(Kn<0.5,
blue, red) rgb(red, green, blue, S) } else { ## All or none
ifelse(B>K,rgb(1,0,0,S),rgb(0,0,1,S)) } } png("map3.png",width=600,height=400)
map('usa',fill=T,col="white",bg="darkgray") ## Create the colors for the states col
<- rep(rgb(.1,.1,.1,.2),length(counties$names)) col[election$order] <-
bluered(election$Bush,election$Kerry,1,graded=T) ## Plot states without borders
(should work with map, see help for fill ## argument, but cannot get it working) m
<- map('county',fill=T,plot=F) polygon(m$x,m$y,col=col,border=NA) ## Draw
county borders map('county
```

III. Plotting maps with R

i. The „map“ and „mapproj“ package

ii. The „drawmap()“ function

iii. Alternatives

ii. The „drawmap()“ function

Plotting maps with colour-based histograms

Motivation

- We want to plot a map with coloured areas…
- …where the colour is correspondent with the value of an area
- Example:
  - Election map
  - Number of doctors in an area
- Example:
  - The higher the number, the more red the colour is
  - The lower the number, the more green it is

Solution

- The „drawmap()“ function
- By Thomas Kneib
- Available here:

  http://www.statistik.lmu.de/~semwiso/raeumliche-statistik0809/uebungen/rcode/blatt6/drawmap.r

- By not it is included in a software package called „BayesX“:
  http://www.stat.uni-muenchen.de/~bayesx/bayesx.html

Input data

- „drawmap()“ basically builds on the „polygon()“ function
- Map input therefore is list of coordinates…
- …where each list-item is named
- Names may occur twice

Thus:

$\Rightarrow$ Map consists of polygons

$\Rightarrow$ Multiple polygons are grouped to so called „regions“ by their name

$\Rightarrow$ „regions“ may be coloured or their name may be drawn

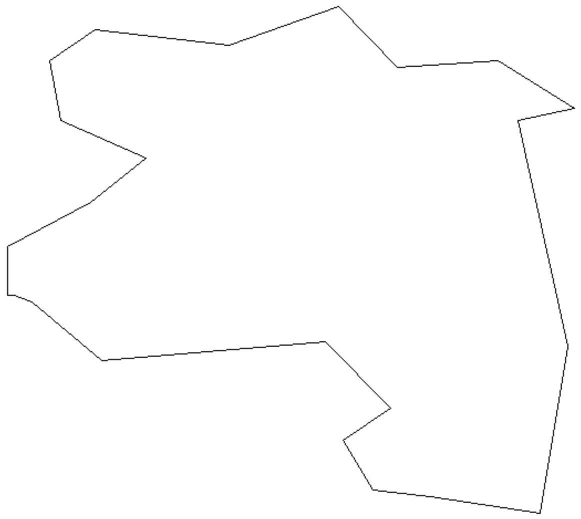
Input data: Example file

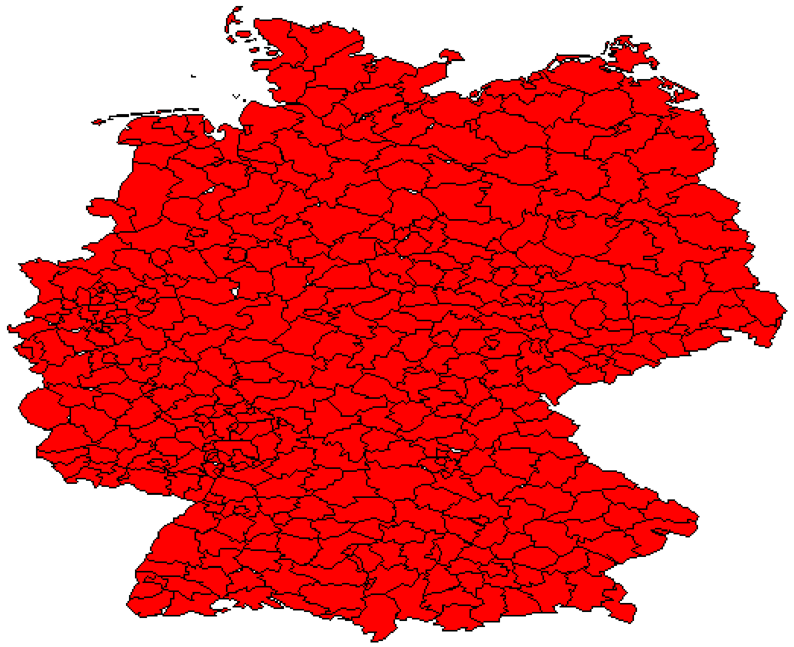
- $`island1`
- [,1] [,2]
- [1,] 3574 3623
- [2,] 3634 3614
- [3,] 3683 3637
- [4,] 3709 3601
- [5,] 3754 3605
- [6,] 3788 3577
- [7,] 3788 3577
- [8,] 3763 3570
- [9,] 3785 3437
- [10,] 3773 3339
- [11,] 3724 3349
- [12,] 3724 3349
- [13,] 3698 3353
- [14,] 3685 3382
- [15,] 3706 3401
- [16,] 3677 3440
- [17,] 3577 3429
- [18,] 3546 3463

- One polygon
- Part of the region „island1“

- These are the x-,y-coordinates

Input data





Data file

- „drawmap()“ supports an input file-path…
- …to a (txt-)file that maps each region-name to a value

- Note: This file is later processed with:

  „read.table(FILEPATH, header = T, row.names = NULL)
- …so be sure to include a header

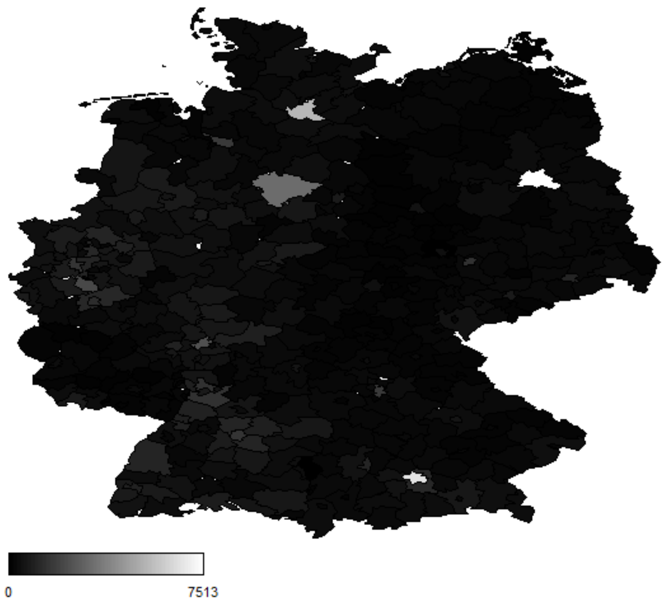
- Example:

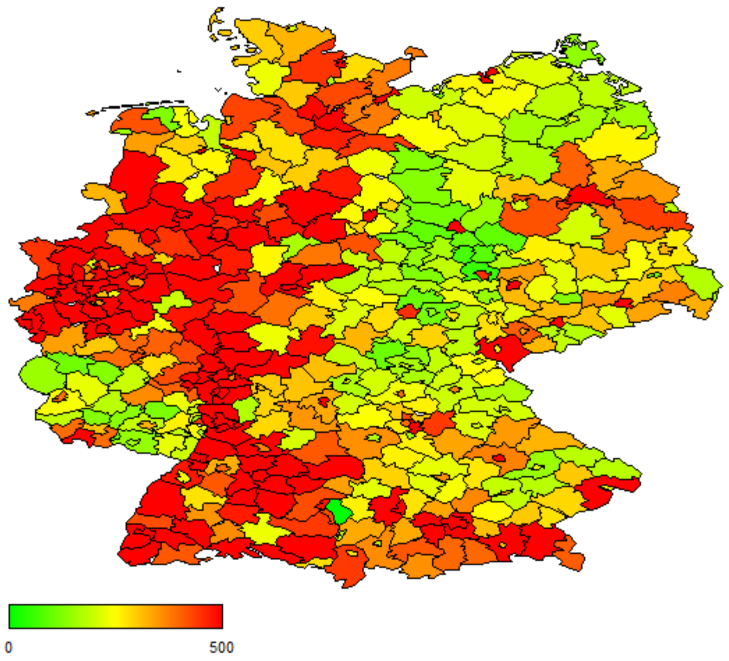
```
"Group.1" "x"
1001 218
1002 703
1003 545
1004 173
```

- …maps the region „1001“ to the value 218 and so on

Data file

- These values are then computed into a gradient of colour or grays
- …and the polygons filled with the corresponding colour





„drawmap()“ parameters

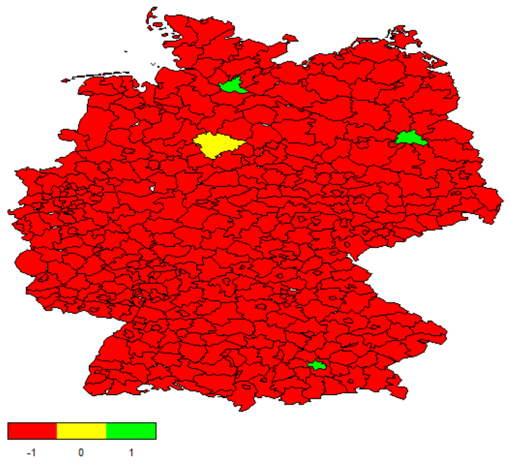
map	The map data needed to plot the map. Expects a list.
dfile	The file that links regions‘ names with values. Expects a path.
regionvar	Specifies which column in the dfile contains the regions‘ names
plotvar	Specifies which columns in the „dfile“ contains the values.
outfile	Specifies the file the map is saved to (in .ps-format). If left blank, the map is drawn to a window.
lowerlimit	Lower limit where to start color-scaling (to avoid outliers)
upperlimit	Upper limit where to start color-scaling (to avoid outliers)

„drawmap()“ parameters

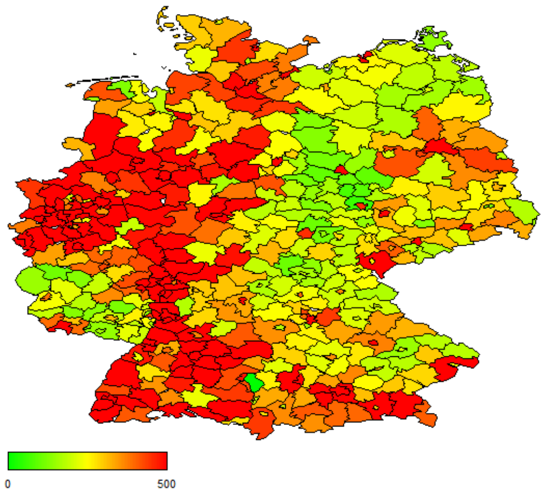
nrcolors	Sets the number of colors for scaling. The more the smoother.
swapcolors	Invert the color scaling. Expects logical value.
pcat	Simplifies values to -1,0,1 (nrcolors=3). Expects a logical value.
hcl	Makes use of the hcl color scaling. Expects a logical value.
h / c / l	Sets start and end color for colorscaling if hcl =T
legend	Specifies whether to plot legend or not. Expects a logical value.
cex.legend	Scales legend‘s size.
drawnames	Draws the regions names. Expects a logical value.
pstitle	Sets the title (equivalent to „main“ parameter)

Some examples

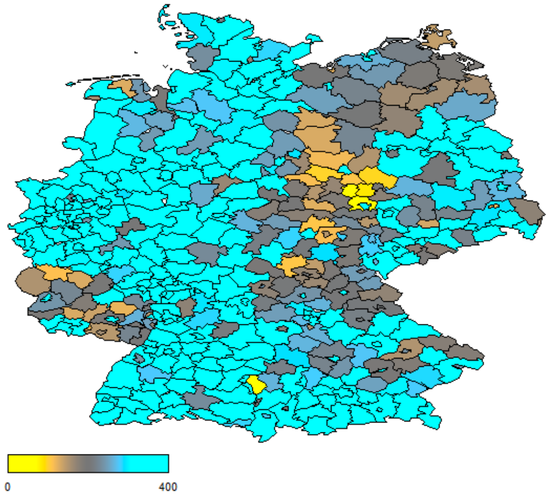
pcat = T



upperlimit=500, color=T



color=T,hcl=T



III. Plotting maps with R

i. The „map“ and „mapproj“ package

ii. The „drawmap()“ function

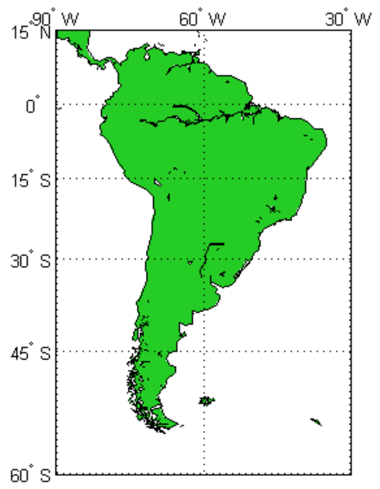
iii. Alternatives

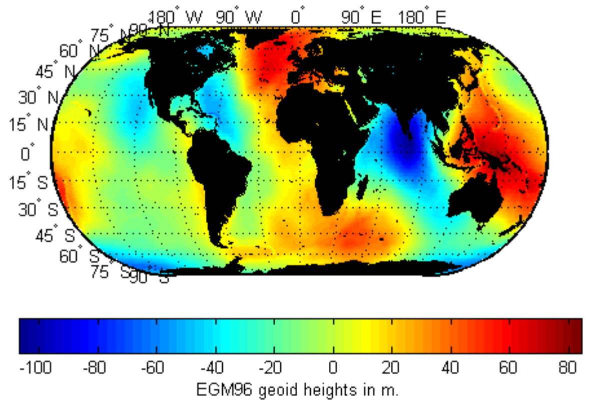
iii. Alternatives

What else is there?

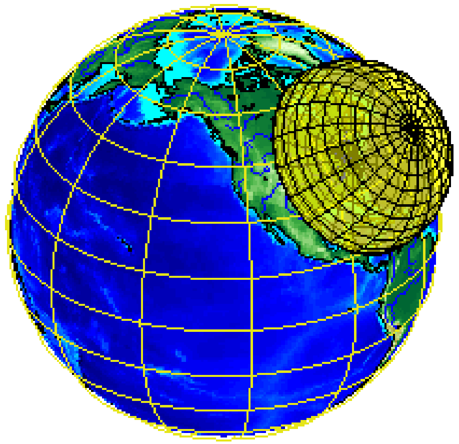
Maps in Matlab: Mapping Toolbox

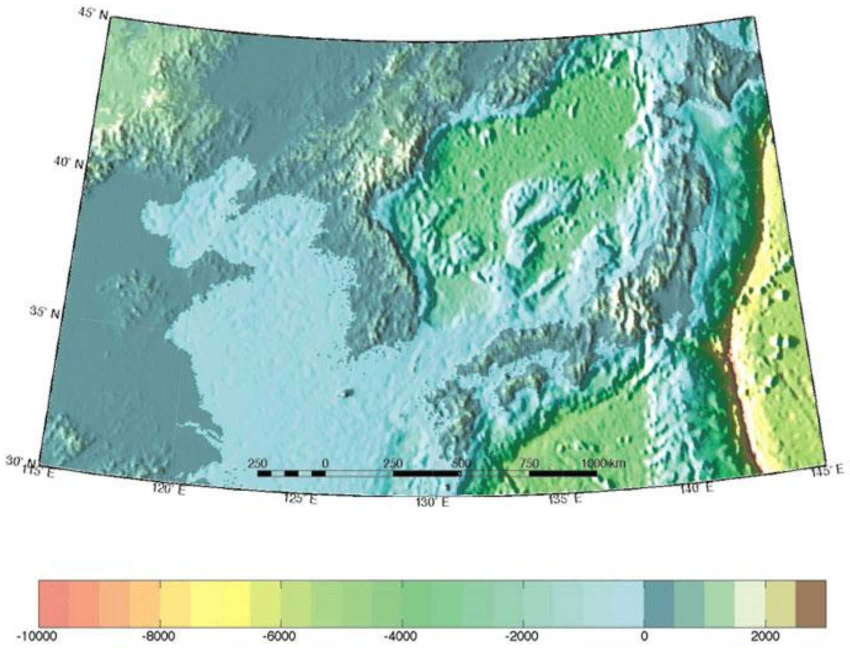
- Commercial addin for Matlab called „Mapping Toolbox“
- Distributed by „MathWorks“
- Homepage:
  http://www.mathworks.com/products/mapping/





Maps in Matlab: Mapping Toolbox





Thanks to:

Vizenz Erhard - for support!

Prof. Czado - for Proseminar!

You all - for your attention!