Plotting in R

A Quick Guide

Stilianos Louca

December 3, 2008

Contents

0.1 General

0.1.1 Loading packages

```
require("scatterplot3d")
```

or

```
library("scatterplot3d")
```

0.1.2 Current working directory

Setting and getting current working directory:

```
setwd("Desktop/myDirectory")
getwd()
```

0.1.3 Colors

Getting a color from RGB:

```
rgb(r,g,b)
```

where **r,g,b** range within [0,1].

0.1.4 Defining & creating data

- Creating a vector (list) of numbers:

  ```
  foo = c(1,2,3,4,5)
  ```

- Creating a **data.frame** from vectors $a, b, c, ..$ and column-titles $A, B, C, ..$:

  ```
  foo = data.frame(A=a, B=b, C=c,..)
  ```

- Reading in a CSV file of values:

  ```
  foo = read.csv(file="Desktop/data.csv", head=FALSE, sep=",")
  ```

  Reading in a CSV file of values, where the 1st row contains the column-titles:

  ```
  foo = read.csv(file="Desktop/data.csv", head=TRUE, sep=",")
  ```

  Note: **foo** will be of type **data.frame**

- Writing list or data frame to file:

  ```
  write.table(mylist, file="output.txt")
  write.table(mylist,
    file="output.txt",
    sep=", ", quote=FALSE,
    row.names=FALSE, col.names=FALSE)
  ```

- Displaying basic attributes, a summary and the column-names of a **data.frame**:

  ```
  attributes(foo)
  summary(foo)
  names(foo)
  ```

- Accessing single column (as vector) of **mydataframe**:

  ```
  foo = mydataframe$columnname
  ```

- Getting items count of list **bar**:

```
foo = length(bar)
```

- Converting number-vector, into factors-vector:

```
foo = factor(foo)
```

Example:

```
myfactors=factor(read.csv(file="in.csv", head=TRUE, sep=",")$mycolumn)
```

- Creating list (vector) of **N** (or **d**-distanced) values from **minvalue** to **maxvalue**:

```
foo = seq(minvalue,maxvalue,d)
foo = seq(minvalue,maxvalue,length=N)
```

Alternatively

```
foo = 1:100
```

creates a list of numbers 1,2,3,..,100.

- Creating a **data.frame** from all combinations of the vectors supplied:

```
foo = expand.grid(values1,values2,..)
foo = expand.grid(columnTitle1=values1,columnTitle2=values2,..)
```

Each row of **foo** represents one combination.

- Outer product: Creating a matrix of function values for all combinations of the 2 input values:

```
f = function(x,y){sin(x) + exp(y*x)}
foo = outer(x,y,f)
```

where **x,y** are lists of numbers.

- Creating list normally distributed random numbers:

```
foo = rnorm(10000, 0, 1)
```

0.2 Interpolations

0.2.1 Splinefun

Given vectors **x,y**, the function **splinefun(x,y)** returns an interpolation function of the points defined by **x,y**.
Example: For given points defined by vectors **x,y** and x-values **xBetween**, the code

```
foo = splinefun(x,y)
yBetween = foo(xBetween)
dydxBetween = foo(xBetween, deriv=1)
```

spline-fits the new values to the old points and calculates the corresponding 1st derivatives of the splined function.

0.3 1D-Plots

0.3.1 Stripchart

Displaying a vector x of numbers as boxes on $\mathbb{R}$ axis:

```
stripchart(x, method="stack", main="Graph title", xlab="x-axis title")
```

Other **method** options include "**jitter**" and none at all.
Adding a plot over an existing one:

```
stripchart(x,method="stack",add=TRUE,at=15)
```

0.4 2D - Plotting

0.4.1 Plot

Simple 2D plotting points, described by vectors **xvalues, yvalues**:

```
plot(xvalues,yvalues)
plot(xvalues,yvalues,main="Title",xlab="x-label",ylab="y-label", sub="subtitle", type="p")
plot(xvalues,yvalues,
  cex.axis=0.7,cex.lab=0.9, #scales labels
  cex=0.8, #scales point size
  pch=16) #draws full  circles as points
```

type can be "p" for points, "l" for lines, "b" for both, "s" for stair steps etc. To add error bars of length specified in vector **errors**, use

```
arrows(xvalues,yvalues+errors,xvalues,yvalues-errors,
  length=0.02,angle=90,code=3)
```

0.4.2 Dnorm

Test-plot values against normal distribution:

```
qqnorm(myvalues, main="Normal distro test") #plot quantiles
qqline(myvalues) #draw theoretical line
```

0.4.3 Hist

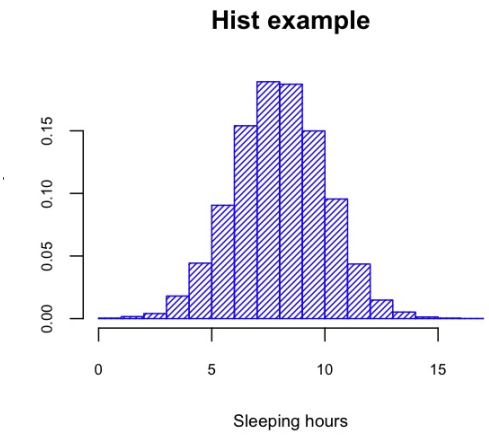
Displaying a histogram of the values in vector x:

```
hist(x)
hist(x,brakes=10)
hist(x,brakes=10,xlim=c(minValue,maxValue),main="Graph title",xlab="x-axis title")
```

brakes is the number of intervals, **xlim** the x-axis limits (as 2-value vector).
The following code

```
hist(rnorm(1000,8,2),
  xlab="Sleeping hours",main="Hist example",
  cex.axis=0.7,cex.lab=0.8,
  freq=FALSE,
  density=30, #density of lines drawn to create "shade" within boxes
  col="blue")
```

produces the histogram



0.4.4 Histogram & Densitplot

Displays histogram of measured values passed within a vector or **data.frame**.

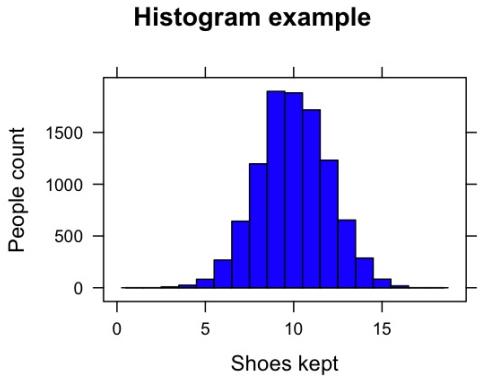
```
counts=rnorm(1000,0,1) #create random test data
histogram(~counts)
```

The code

```
data=data.frame(shoes=factor(round(rnorm(10000,4,2)))) #create random test data

histogram(~shoes , data=data,
  type="count", #other options are "density" and "percent"
  main="Histogram example",xlab="Shoes kept",ylab="People count",
  col="blue")
```

generates the histogram

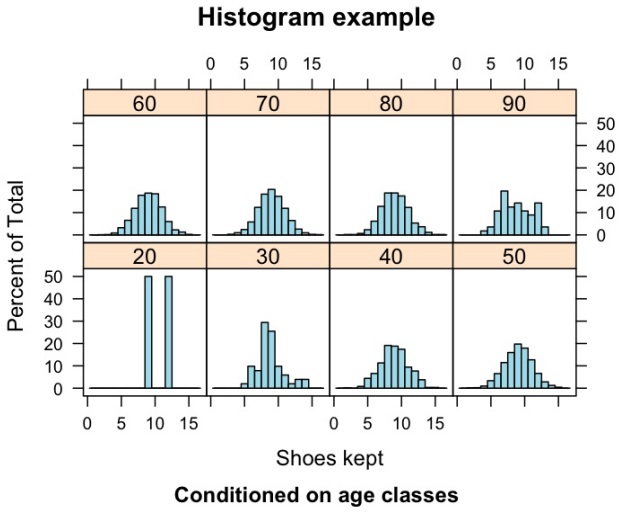


Conditional displays are possible, and each value produces a different histogram. The code

```
experiment=data.frame(shoes=factor(round(rnorm(10000,4,2))),
    weight=factor(10*round(rnorm(10000,6,1)))) #generate random shoes counts & weight classes

histogram(~shoes | weight, data=experiment,
  type="percent",
  main="Histogram example",xlab="Shoes kept",sub="Conditioned on age classes",
  col="lightblue")
```

creates histograms of **shoes**-count conditioned on **weight**-class and generates the histogram



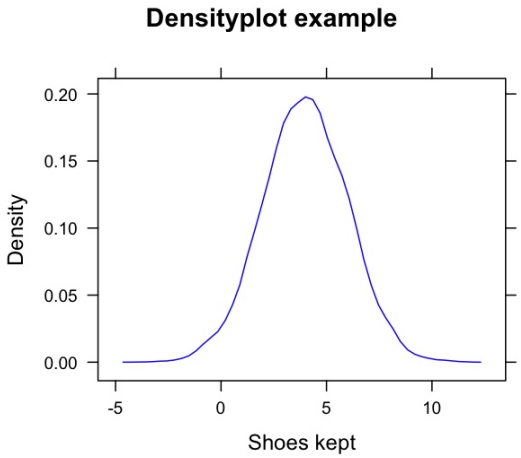
Densityplot works the exact same way.

```
densityplot(~myvalues)
densityplot(~rnorm(10000,4,2),
  type="density",col="blue",
  main="Gauss")
```

The code

```
densityplot(~rnorm(10000,4,2),
  main="Densityplot example",xlab="Shoes kept",
  col="blue",type="density")
```

generates the histogram



0.4.5 Contourplot & Levelplot

Draw level plots of z-values over an x,y-grid:

```
yvalues=xvalues=seq(0,1,length=100) #define grid lines
grid=expand.grid(xtitle=xvalues, ytitle=yvalues) #create grid as a frame. Each row is a
    combination of x & y
grid$ztitle=cos(grid$xtitle)+sin(grid$ytitle) #evaluate z on each grid-point
levelplot(ztitle~xtitle*ytitle, grid,cuts=50,labels=list(cex=0.7,col="green"))
```

Alternatively (minimalistic):

```
levelplot(grid$ztitle~grid$xtitle*grid$ytitle)
```

First (and required) argument is of format

```
responce~coordinate1*coordinate2*...
```

while **coordinates** are lists of numbers, where **coordinateN[i]** is the N-th coordinate of the i-th grid point. So **responce[i]** is the field-value on that grid point.

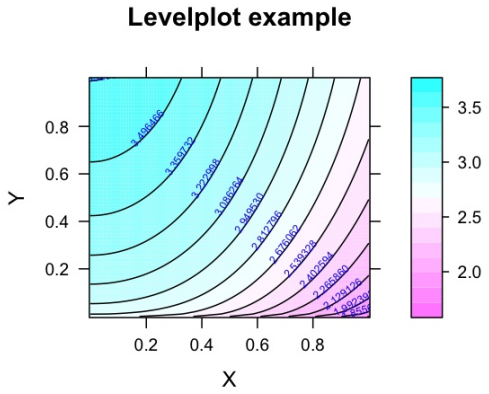
Notes:

- Similar to **contourplot**, only that in the later, just contours are drawn.
- Requires **lattice** package.

The following code

```
grid=expand.grid(x=seq(0,1,length=100),y=seq(0,1,length=100)) #create grid
z=exp(cos(grid$x))+sqrt(sin(grid$y)) #evaluate z at grid points
levelplot(z~grid$x*grid$y,
  main="Levelplot example",xlab="X",ylab="Y",
  cut=15, #number of contours
  contour=TRUE,
  labels=list(cex=0.5,col="blue"))
```

produces



0.4.6 Contour

Simple contour display. Parameters are vectors **x,y** of grid cuts and a 2D-matrix of evaluation values (**NA** is possible):

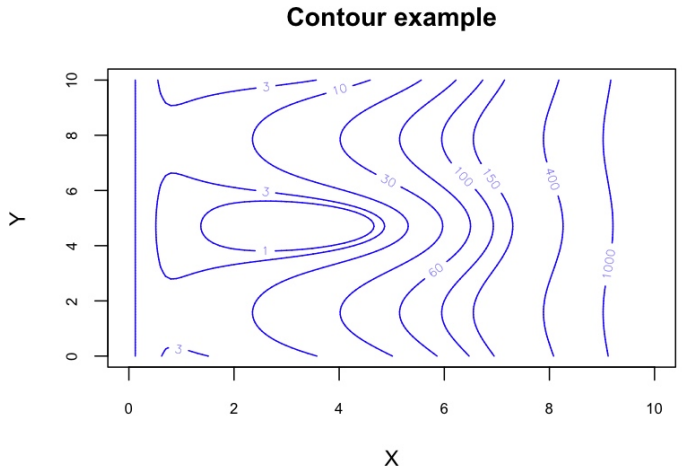
```
x = y = seq(0,10,0.1)
z=outer(x,y,function(x,y){x^2*sin(y)+exp(x)/x})
contour(x,y,z)
contour(x,y,z,nlevels=10,add=TRUE) #the last one adds the contour to an existing plot
```

Note: Requires package **graphics**

Example: The following code

```
contour(x,y,z,
  levels=c(1,3,10,30,60,100,150,400,1000,10000),,col="blue"
  labcex=0.6,cex.axis=0.7,axes=TRUE,
  main="Contour example",xlab="X",ylab="Y")
```

produces the plot



0.5 3D - Plots

0.5.1 Cloud & Wireframe

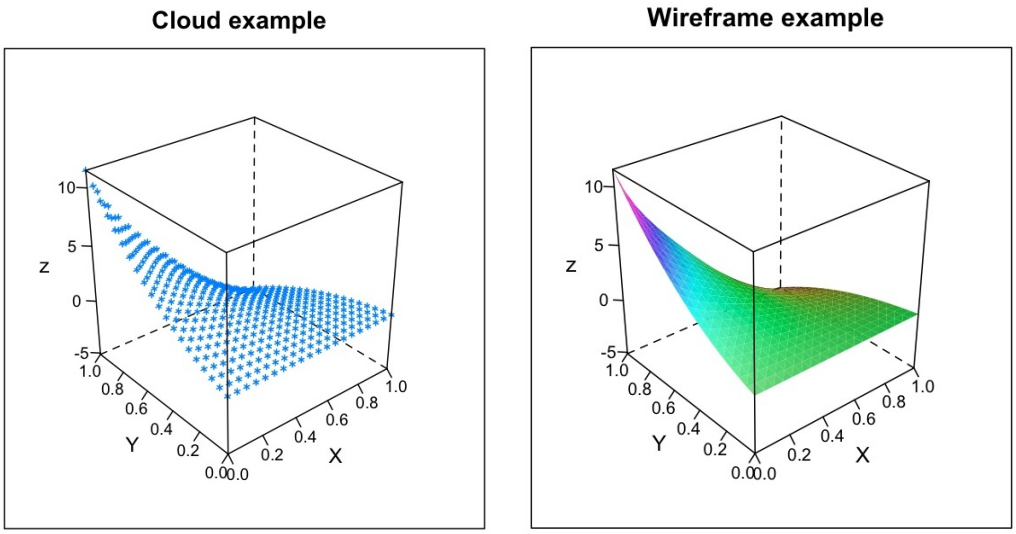
Similar to **levelplot**, these draw scatter plots & surfaces into a 3D box. The following codes

```
grid=expand.grid(x=seq(0,1,length=20),y=seq(0,1,length=20)) #create grid
z=exp(x+y)*sin(y)*cos(x*3)/(x+0.2) #evaluate z at grid points
cloud(z~x*y,
  main="Cloud example",xlab="X",ylab="Y",
  scales=list(arrows=FALSE))
```

and alternatively

```
wireframe(z~x*y,
  main="Wireframe example",sub="z=\exp(x+y)",xlab="X",ylab="Y",
  scales=list(arrows=FALSE),
  shade=TRUE,
  light.source=c(2,1,2))
```

produce the plots



0.5.2 Scatterplot3d

Plot points defined by 3 vectors **x, y, z**:

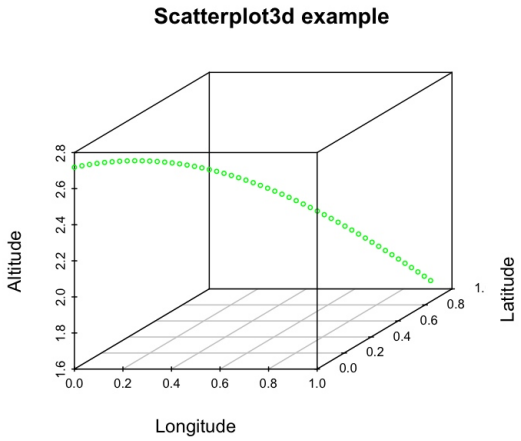
```
scatterplot3d(x,y,z)
scatterplot3d(x,y,z,main="Title",sub="Subtitle",box=TRUE, angle=40, highlight.3d=TRUE)
scatterplot3d(x,y,z,cex.symbols=0.3,cex.axis=0.7,cex.lab=1) #last 3 arguments change relative
    size of attributes (symbols, axis-annotation, labels)
```

Other options include **xlab, ylab, zlab, scale.y, col.axis, tick.marks, type, color** etc. Points style is defined by **pch** (i.e 1 for circles, 4 for cross, 16 for full dots)
Note: Requires package **scatterplot3d**.

Example: The code

```
x = seq(0,1,0.02) #create 50 equidistant numbers
y = sin(x)
z = exp(cos(x))
scatterplot3d(x,y,z,
  cex.symbols=0.5,
  cex.axis=0.7,
  color="green",
  main="Scatterplot3d example",
  grid=TRUE,
  type="l"
  xlab="Longitude", ylab="Latitude",zlab="Altitude")
```

produced the Plot



0.5.3 Persp

Display perspective view of surfaces over the x-y Plane. Input consists of two vectors **x,y** of accenting numbers (defining the grid cuts on each axis, not the points as in wireframe!) and a 2D-matrix z of dim=dim(x)*dim(y):

```
x = y = seq(0,10,0.1)
z = outer(x,y,"*")
persp(x,y,z)
persp(x,y,z,theta=90,phi=180,r=10)#Note: theta is the viewing azimuth, phi the colatitude
persp(shade=TRUE, border=NA, col="lightblue",box=FALSE) #the last one hides all numbering, labels
    and axes
persp(x,y,z,
  axes=TRUE, ticktype="detailed",
  cex.axis=0.7,cex.lab=0.8,
  xlab="Age",ylab="Weight",zlab="Sleep time")
```

Note: Requires **graphics** package.

Example: The following code

```
x = y = seq(0,10,0.1)
z = outer(x,y,function(x,y){(x^2+y^2)*sin(x*y/10)})
persp(x,y,z
  theta=180,phi=20,
  shade=TRUE, col="lightblue", border=NA,
  axes=TRUE, ticktype="detailed",
  cex.axis=0.6,cex.lab=0.8,
  main="Persp example",xlab="Age",ylab="Weight",zlab="Sleep time")
```

produced the Plot

Persp example