UCLA Department of Statistics
Statistical Consulting Center

Graphics for Exploratory Data Analysis in R: Part I

Irina Kukuyeva
ikukuyeva@stat.ucla.edu

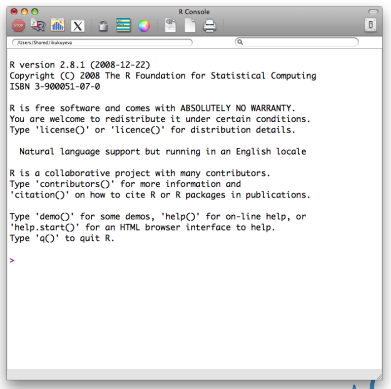
August 20, 2009

Outline

Software Installation

Installing R on a Mac

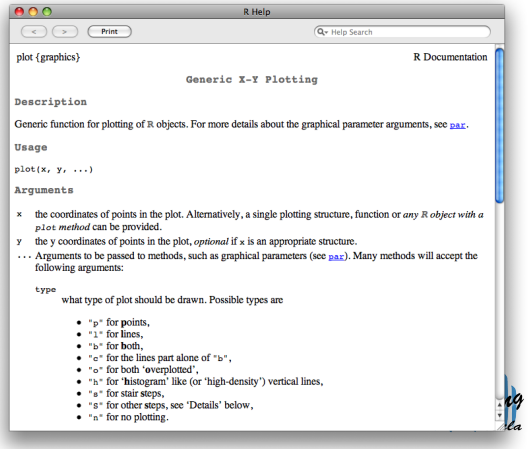
1. Go to `http://cran.r-project.org/` and select *MacOS X*
2. Select to download the latest version: 2.9.1 (2009-06-26)
3. Install and Open. The R window should look like this:



R Help

For help with any function in R, put a question mark before the function name to determine what arguments to use, examples and background information.

```
1 ?plot
```



Data from the Internet

When downloading data from the internet, use `read.table()`. In the arguments of the function:

- `header:` if TRUE, tells R to include variables names when importing
- `sep:` tells R how the entires in the data set are separated
  - `sep=",":` when entries are separated by COMMAS
  - `sep="\t":` when entries are separated by TAB
  - `sep=" ":` when entries are separated by SPACE

```
1 data<-read.table("http://www.stat.ucla.edu
2 /~vlew/stat130a/datasets/twins.csv",
3 header=TRUE, sep=",")
```


Importing Data from Your Computer

1. Check what folder R is working with now:

```
1 getwd()
```

2. Tell R in what folder the data set is stored (if different from (1)). Suppose your data set is on your desktop:

```
1 setwd("~/Desktop")
```

3. Now use the `read.table()` command to read in the data, substituting the name of the file for the website.

Using Data Available in R

1. To use a data set available in one of the R packages, install that package (if needed).
2. Load the package into R, using the `library()` function.

```
1 library(alr3)
```

3. Extract the data set you want from that package, using the `data()` function. In our case, the data set is called `UN2`.

```
1 data(UN2)
```

Loading data into R [1]

```
survey = read.table("http://www.stat.ucla.edu
    /~mine/students_survey_2008.txt", header =
     TRUE, sep = "\t")
# To see the variable names
names(survey)
# To refer to the variables by name:
attach(survey)
```

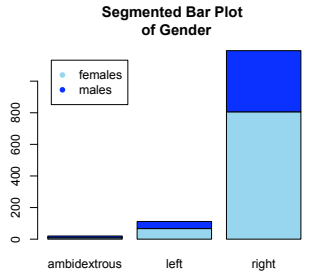
```
 [1] "gender"      "hand"        "eyecolor"    "glasses"     "california"
 [6] "birthmonth"  "birthday"    "birthyear"   "ageinmonths" "height"
[11] "graduate"    "oncampus"    "time"        "walk"        "hsclass"
[16] "HSCA"        "calculus"    "AP"          "cell"        "ipod"
[21] "sleep"       "alcohol"     "speed"       "UCLA"        "book"
[26] "relax"       "instructor"  "tire"        "Quarter"
```

[1] This and the next three slides are modified from the SCC Mini-Course "Introductory Statistics with R" by Mine Çetinkaya

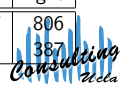
Segmented bar charts

Displays two categorical variables at a time:

```
barplot(table(gender, hand
    ), col = c("skyblue", "
    blue"), main = "
    Segmented Bar Plot \n
    of Gender")
legend("topleft", c("
    females","males"), col
    = c("skyblue", "blue"),
     pch = 16, inset =
    0.05)
```



	ambidextrous	left	right
female	9	67	806
male	11	45	387



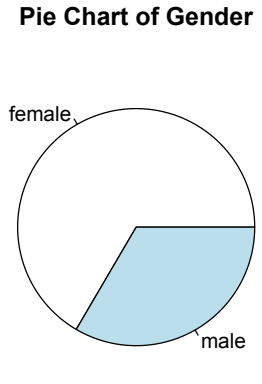
Pie charts

Pie charts display counts as percentages of individuals in each category:

```
table(gender)
```

	gender
female	882
male	443

```
pie(table(gender))
```

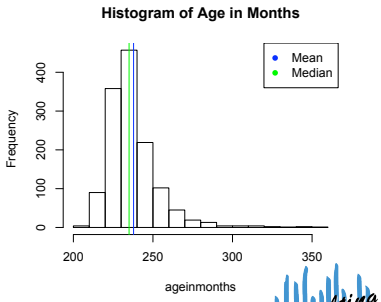


Histograms

Adding Summary Statistics to Plots

Add the mean and median to a histogram:

```
1 hist(ageinmonths, main="
    Histogram of Age (Mo)")
2 abline(v=mean(ageinmonths)
    , col = "blue")
3 abline(v=median(
    ageinmonths), col = "
    green")
4 legend("topright", c("Mean
    ", "Median"), pch = 16,
     col = c("blue", "green
    "))
```



Histograms I

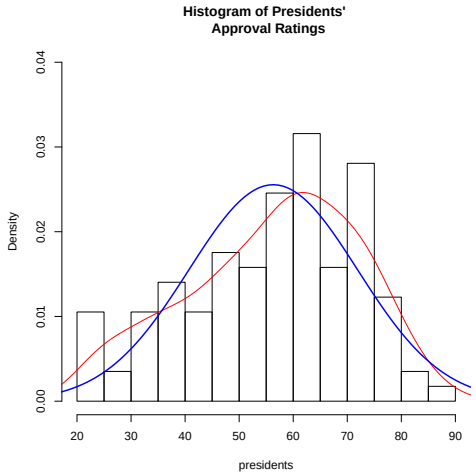
Checking Normality

One of the methods to test for normality of a variable is to look at the histogram (the sample density is in red, the theoretical normal density in blue):

```
1 data(presidents)
2 hist(presidents, prob=T, ylim=c(0, 0.04),
    breaks=20)
3 lines(density(presidents, na.rm=TRUE), col="
    red")
4 mu<-mean(presidents, na.rm=TRUE)
5 sigma<-sd(presidents, na.rm=TRUE)
6 x<-seq(10,100,length=100)
7 y<-dnorm(x,mu,sigma)
8 lines(x,y,lwd=2,col="blue")
```


Histograms II

Checking Normality

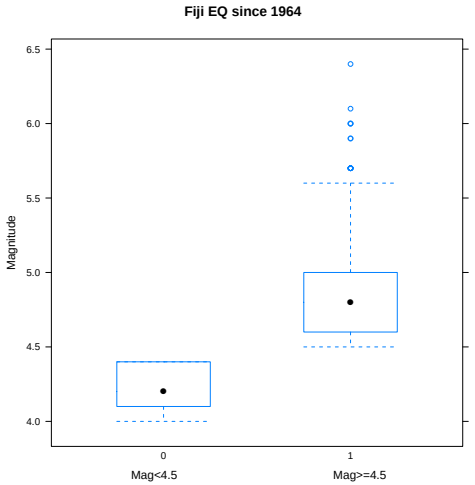


Box and Whisker Plot I

Another method of looking at the distribution of the data is via boxplot:

```
1 data(quakes)
2 # Subset the magnitude:
3 ind<-ifelse(quakes[, 4]<4.5, 0, 1)
4 ind<-as.factor(ind)
5 boxplot(quakes[, 4]~ind)
6 # Alternatively:
7 library(lattice)
8 bwplot(quakes[, 4]~ind, xlab=c("Mag<4.5", "Mag
    >=4.5"), ylab="Magnitude")
```

Box and Whisker Plot II

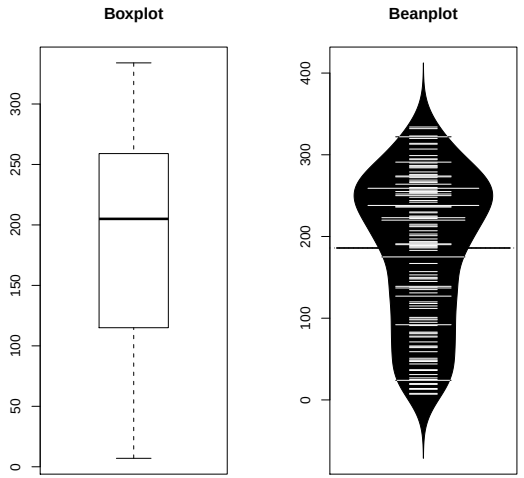


Beanplot I

An alternative to the boxplot is the `beanplot()`:

```
1 library(beanplot)
2 par(mfrow=c(1,2))
3 data(airquality)
4 boxplot(airquality[, 2], main="Boxplot", xlab=
    "Solar")
5 beanplot(airquality[, 2], main="Beanplot",
    xlab="Solar")
```


Beanplot II

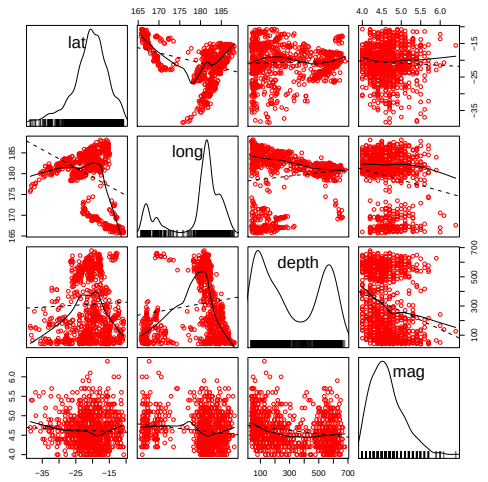


Scatterplots I

A method of looking at the distribution and correlation of the data is via `scatterplot.matrix()`:

```
1 data(quakes)
2 library(car)
3 scatterplot.matrix(quakes[, 1:4])
```

Scatterplots II

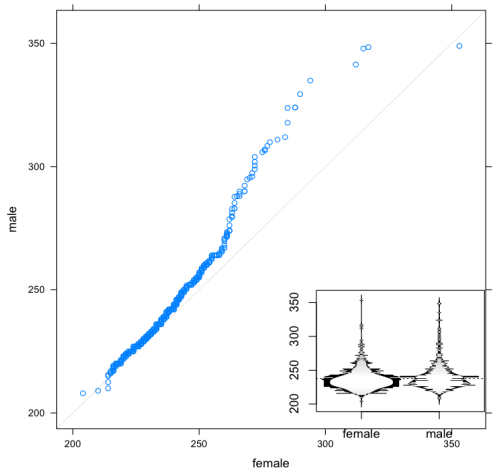


Checking Equality of Distributions I

A method of checking equality of distributions is via qq():

```
1 library(lattice)
2 survey = read.table("http://www.stat.ucla.edu
    /~mine/students_survey_2008.txt", header =
     TRUE, sep = "\t")
3 attach(survey)
4 qq(gender~ageinmonths)
```


Checking Equality of Distributions II



Identify-ing Observations I

Preliminaries

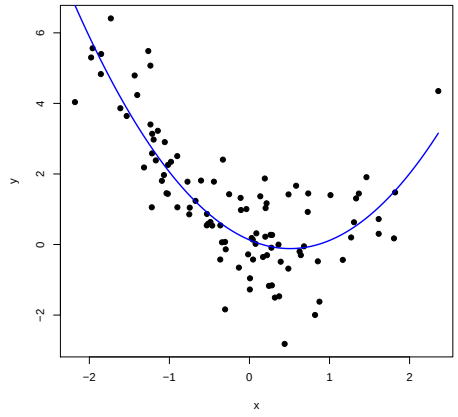
```
1 # Generate data and fit a regression curve:
2 set.seed(3012008)
3 x=rnorm(100); y=-x+I(x^2) +rnorm(100)
4 fit<-lm(y~x+I(x^2)); fit
```

Intercept	x	x^2
0.1307	-0.9701	0.9549

```
1 # Plot the resulting regression curve:
2 plot(y~x, pch=19)
3 curve(expr=fit[[1]][1]+fit[[1]][2]*x+fit
    [[1]][3]*I(x^2), from=range(x)[1], to=
    range(x)[2], add=TRUE, col="blue", lwd=2)
```

Identify-ing Observations II

Preliminaries

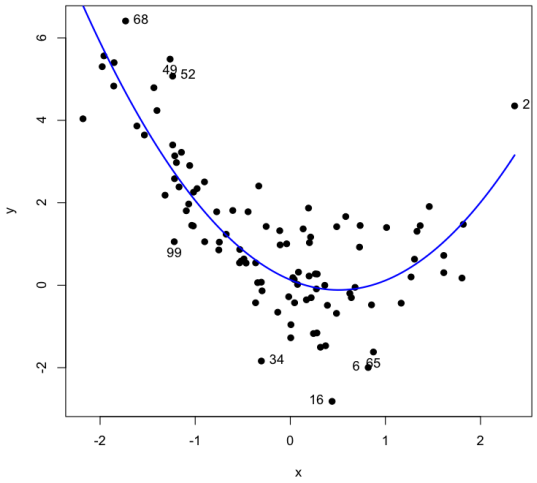


Identify-ing Observations I

Left-click on the observations in the graphics window to see the row number.
Right-click on the observation to exit the function.

```
# Plot the data and fit the regression curve:
plot(y~x, pch=19)
curve(expr=fit[[1]][1]+fit[[1]][2]*x+fit
    [[1]][3]*I(x^2), from=range(x)[1], to=
    range(x)[2], add=TRUE, col="blue", lwd=2)
# Identify the "outlying" observations:
index<-identify(y~x); index
```

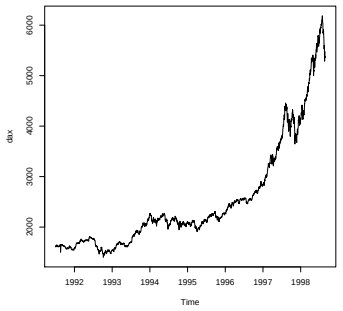

Identify-ing Observations II



Univariate Time Series Plot [2]

To plot one variables one at a time, use `plot()`:

```
1 data(EuStockMarkets)
2 dax<-EuStockMarkets[, 1]
3 plot(dax)
```



[2] This section is from the SCC Mini-Course "Introductory Time Series with R" by Irina Kukuyeva

Multivariate Time Series Plots I

Approach 1

To plot more than variables one at a time, use `plot()`:

```
1  # Convert data to a time series via ts() or
       zoo():
2  data(airquality)
3  a<-airquality[, 1:3]
4  time<-ts(1:nrow(a), start=c(1973, 5),
       frequency=365)
5  # If your data is stored as a data frame,
6  # coerce it to be a matrix via as.matrix()
7  class(a)
8  a.mat<-as.matrix(a)
9
10
```

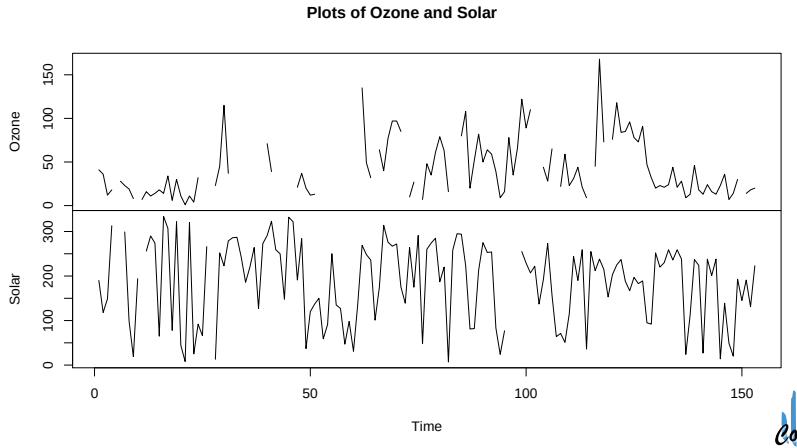

Multivariate Time Series Plots II

Approach 1

```
11 # Make a time series of the two (or more
       variables)
12 library(zoo)
13 name.zoo<-zoo(cbind(a.mat[, 1], a.mat[, 2]))
14 colnames(name.zoo)<-c("Ozone", "Solar")
15 #### Plot the variables
16 plot(name.zoo)
```


Multivariate Time Series Plots III

Approach 1



Multivariate Time Series Plots I

Approach 2

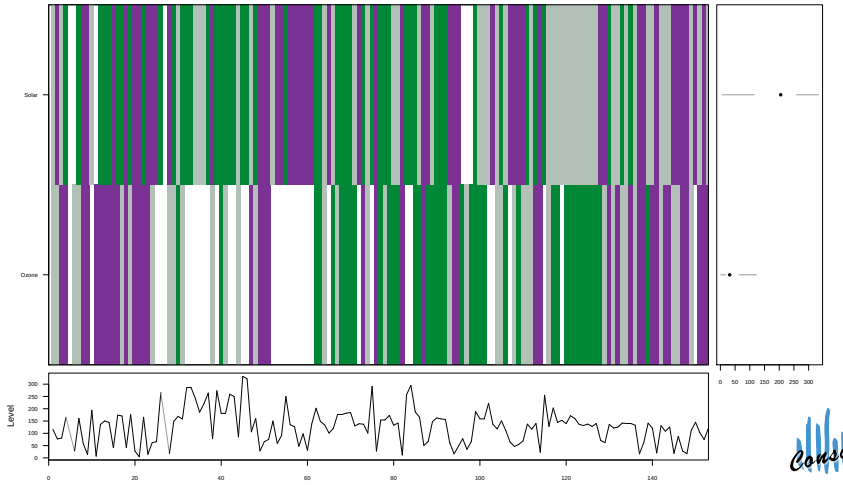
To plot more than variables one at a time, use `mvtsplot()`:

- For documentation, go to: www.jstatsoft.org/v25/c01/paper
- Go to: http://www.biostat.jhsph.edu/~rpeng/RR/mvtsplot/
- Copy the relevant R Code and paste it into the R Console. Press ENTER.
- Plot your data

```
# After processing data as in Approach 1
# Plot the variables
mvtsplot(name.zoo)
# Purple=low, grey=medium, green=high,
    white=missing values
```


Multivariate Time Series Plots II

Approach 2



Multivariate Time Series Plots I

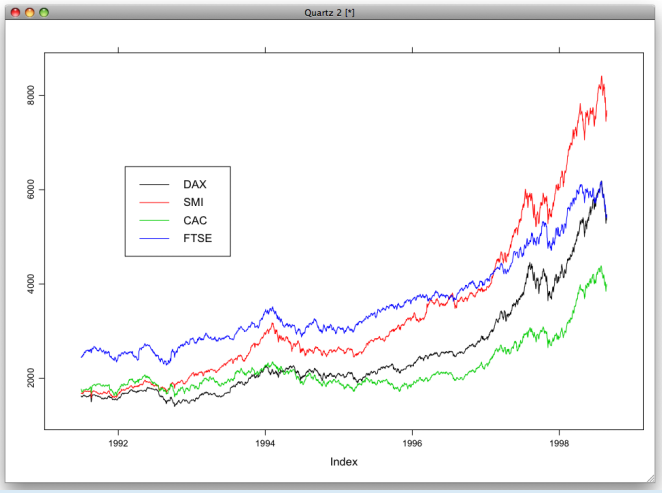
Approach 3

To plot more than variables one at a time, use `xyplot()`:

```
# After processing data as in Approach 1
# load both libraries:
library(lattice)
library(zoo)
data(EuStockMarkets)
z<-EuStockMarkets
xyplot(z, screen = c(1,1,1,1), col = 1:4,
    strip = FALSE)
legend(1992, 5000, colnames(z), lty = 1, col =
    1:4)
```


Multivariate Time Series Plots II

Approach 3

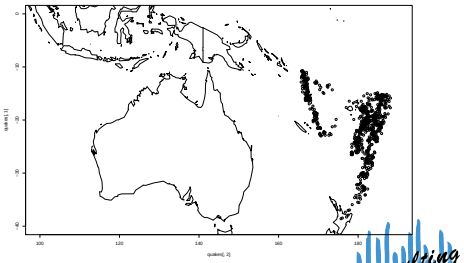


Geographic Maps

Map of Fiji Earthquakes Since 1964

To overlay a map to a plot containing latitude and longitude, load the package `maps`:

```
1 data(quakes)
2 library(maps)
3 plot(quakes[, 2],
    quakes[, 1], xlim=
    c(100, 190), ylim=
    c(-40, 0))
4 map("world", add=T)
```

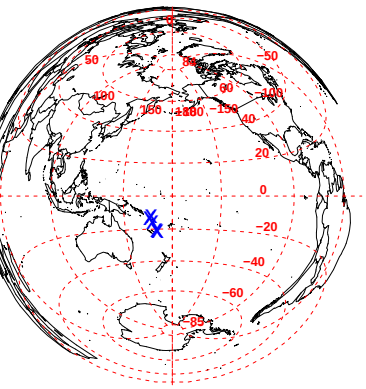


Projection Maps I

Map of Fiji Earthquakes Since 1964

For a different perspective of a map, use `mapproject()`:

```
1 library(mapproj)
2 library(maps)
3 m <- map('world',plot=FALSE)
4 # Projection is Azimuthal with equal-area
5 map('world',proj='azequalarea',orient=c(
    longitude=0,latitude=180,rotation=0))
6 map.grid(m,col=2)
7 points(mapproject(list(y=quakes[which(quakes[,
    4]>=6), 1],x=quakes[which(quakes[, 4]>=6)
    , 2])),col="blue",pch="x",cex=2)
```



Bonus Feature of the `maps` package:

To determine in which part of the world the observations are (based on latitude and longitude), use `map.where()`:

```
in.what.country<-map.where(database="world",
    quakes[, 2], quakes[, 1])
```

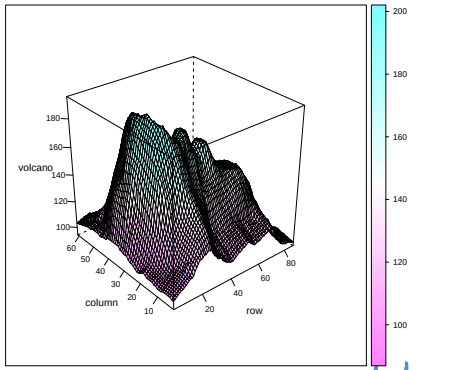
To determine which observations are in the ocean:

```
# Number of points in ocean after filtering:
ind<-sum(is.na(in.what.country)); ind
# Number of observations: 1000
# Number in Ocean: 993
```

One way to create 3D images is with the package `lattice`:

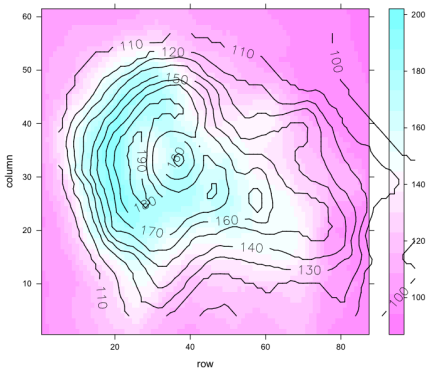
Method 1: Using `wireframe()`:

```
library(lattice)
wireframe(volcano,
    color.palette =
    terrain.colors,
    asp = 1, color.key
    =TRUE, drape=TRUE,
     scales = list(
    arrows = FALSE))
```



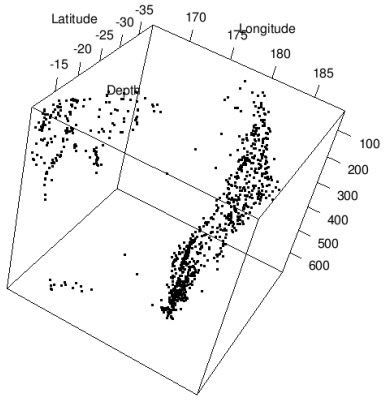
Method 2: Same image with the `levelplot()`:

```
library(lattice)
levelplot(volcano,
    color.palette =
    terrain.colors,
    asp = 1, color.key
    =TRUE, drape=TRUE,
     scales = list(
    arrows = FALSE))
contour(volcano, add=
    TRUE, lwd=1.3,
    labcex=1.3)
```



Another way to create 3D images is with the package `rgl`:

```
library(rgl)
data(quakes)
plot3d(x=quakes[, 2], y=quakes[, 1], z=quakes[, 3], xlab="Longitude", ylab="Latitude", zlab="Depth")
```



Simulations I

Preliminaries: The function `outer()`

```
x=5:6; y=1:3
outer(x,y)
```

```
     [,1] [,2] [,3]
[1,]    5   10   15
[2,]    6   12   18
```

```
fcn<-function(x,y){z=x+y}
outer(x,y,fcn)
```

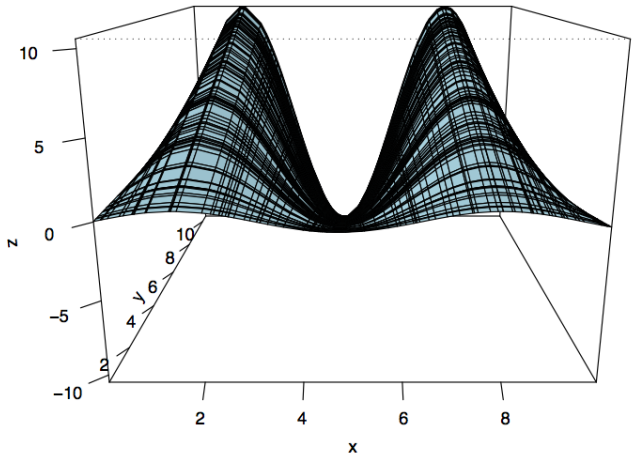
```
     [,1] [,2] [,3]
[1,]    6    7    8
[2,]    7    8    9
```


Simulations II

Suppose we want to know what the function $y \times sin(x)$ looks like:

```
# Sample from the random uniform:
x <- sort(runif(100, min=0, max=10))
y <- x+runif(1)
f <- function(x,y) { r <- y*sin(x)}
z <- outer(x,y,f)
persp(x, y, z, col = "lightblue", shade = 0.1,
      ticktype = "detailed", expand=0.7)
```

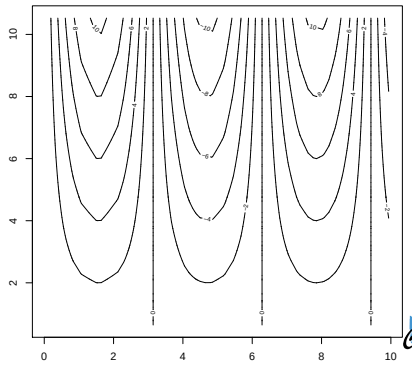
Simulations III



Simulations IV

To visually see its maximum and minimum values, look at the contours of the function:

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



Online Resources for R

Download R: *http://cran.stat.ucla.edu/*

Search Engine for R: *http://rseek.org*

R Reference Card:
http://cran.r-project.org/doc/contrib/Short-refcard.pdf

R Graphics Gallery:
http://addictedtor.free.fr/graphiques/

UCLA Statistics Information Portal: *http://info.stat.ucla.edu/grad/*

UCLA Statistical Consulting Center: *http://scc.stat.ucla.edu*