R : A *damn* small intro

Dionyssios Mintzopoulos

Why N How 2011

Overview

General features and pros: The Good

- Powerful programming platform, similar to Matlab in certain respects
- Interpreted language
- Can be used for a lot of general data analysis functions (eg fits to data etc), but “forte” is statistical analysis
- R is the free version of a similar (proprietary) language, S (manuals in Chessler library!)
- Contributed add-on packages, analogous to Matlab toolboxes, offer extensive capabilities

Some minuses: The Bad

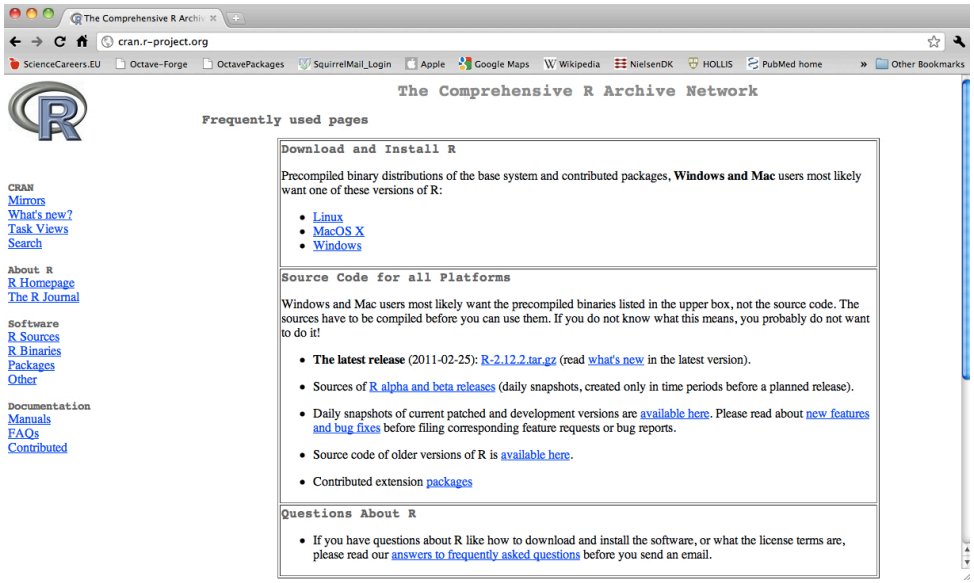
- Can be hard to find resources; need to look around.
- Plenty of books for *$*, *$$*, or for *$$$*, but Google is still one of the best options - plenty of manuals and tutorials!

Some good starting references

The official site: http://cran.r-project.org/

http://cran.r-project.org/doc/contrib/

The official site is also great to check contributed packages etc.



How-to for graphs and plots: “R Graph Gallery,”

http://addictedtor.free.fr/graphiques/

Misc

Mac:

- nice autocomplete and parenthesis-complete features
- easy to uninstall (trash) / install new versions

PC (XP):

- autocomplete not exactly the same, uninstall must be done correctly through control panel

- ! caution with uninstalling current version some package (“toolbox”) may not be updated. As a result, be conservative so as not to lose existing capabilities.

Misc (2)

Everything in R is essentially an object, and mathematical operations (such as regression) are operations on objects.

In a way, we can think of model-building as of building a hiearchy of objects.

Much like Matlab, one can write loops, if-then and other logical structures in R, but this can be slow. Optimum speed is when using the pre-compiled functions.

Hands-on example

needs a *little bit* of

The Ugly: command-line typing

some more pointers

Linear mixed models

- packages **lme** & **lme4** (Douglas Bates)
- eg start from

http://cran.r-project.org/doc/contrib/Fox-Companion/appendix-mixed-models.pdf

Logistic regression and generalized models:

- package **glm**

Bayesian computations

- package **bayes**

finding your way

```
# this is a comment in R

# your current pwd                                                    getwd()

# use ? for help, eg.                                                 ?getwd()

# cd to some other directory = “set working directory                 setwd()

# directory listing:                                                  dir()

#variables-in-memory listing, analogous to matlab ‘who’:             objects()
#or                                                                  ls()

#analog to matlab ‘clear’                                            rm()

# example of rm()
objects()
 [1] "age"    "data"   "fit"    "group"  "height" "m"      "q"      "r"
> ls()
 [1] "age"    "data"   "fit"    "group"  "height" "m"      "q"      "r"
> rm(age)
> ls()
 [1] "data"   "fit"    "group"  "height" "m"      "q"      "r”
```

packages

Packages are installed from the Net using the install.packages() program

install packages (analogs to matlab toolboxes)

?install.packages()
install.packages()

After having run install.packages("Design") type to load into memory, for example:

linbrary(package name)

```
> library(Design) # load into memory
#Loading required package: Hmisc
#Loading required package: survival
#Loading required package: splines
#
#Attaching package: 'Hmisc’
```

... verbose output continues until package is loaded into memory, then it’s ready to use

what’s in the working memory

```
# check what's loaded in memory

> search()
 [1] ".GlobalEnv"        "tools:RGUI"        "package:stats"     "package:graphics"
 [5] "package:grDevices" "package:utils"     "package:datasets"  "package:methods"
 [9] "Autoloads"         "package:base"


# after having run instal.packages("Design")


> search()
[1] ".GlobalEnv"        "package:Design"    "package:Hmisc"     "package:survival"
[5] "package:splines"   "tools:RGUI"        "package:stats"     "package:graphics"
[9] "package:grDevices" "package:utils"     "package:datasets"  "package:methods"
[13] "Autoloads"        "package:base"
```

some useful things

```
# how to do a t-test
# the ? gives the help page; or, equivalently help(t.test)
?t.test

# how to do an F-test
?var.test

# kolmogorov-smirnov, shapiro test for normality,
# and Mann-Whitney (which is non-parametric analog to t-test)
help(ks.test)
?shapiro.test
?wilcox.test  (Mann-Whitney U-test)

# distributions, uniform and normal
?dunif
?rnorm
rmultnorm(n, mu, vmat, tol = 1e-10)
rnorm(n, mean = 0, sd = 1)
```

some useful things

Gettting the t-statistic and the F-statistic for given alpha (α) and degrees of freedom (dof)

	Matlab	Excel	R
Compute threshold statistic, given p-value and dof			
F-statistic	finv(1-alpha , dof1, dof2) finv(.99, 1, 200)	finv(alpha , dof1, dof2)	qf(1-alpha, dof1, dof2) qf(.99, 1, 200)
t-statistic	tinv(1-alpha/2, n-p)	TINV(alpha,n-p)	qt(1-alpha/2, n-p) qt(1-0.05/2, 10-3)
Compute p-value, given statistic and dof			
Given F-statistic	1-fcdf(f-statistic, dof1, dof2) 1-fcdf(6.7633, 1, 200)	fdist(f-statistic, dof1, dof2)	1-pf(1-alpha, dof1, dof2) 1-pf(6.7633, 1, 200)
Given t-Statistic	alpha/2 = 1-tcdf(t-statistic,n-p) >> x=tinv(.975, 7) % x = 2.3646 >> 1-tcdf(x,7) = 0.0250 % alpha/2	TDIST(t-statistc, dof, tails)	1-alpha/2 = pt(t-statistic,n-p) # alpha = 2-2*pt(t-statistic,n-p) # .975 = 1-0.05/2 qt(1-0.05/2, 10-3) # = 2.364624, the t-statistic

some useful things

Equivalent notation (R/Matlab) among functions relating to statistical distributions

	R	Matlab
Relating to F-distribution	pf	fcdf
	qf	finv
	df	fpdf
	rf	frnd
Relating to F-distribution	pt	tcdf
	qt	tinv
	dt	tpdf
	rt	trnd
	(I am not aware of equivalent)	fstat

basic vector and matrix manipulation

```
#make a vector
m<-c(1:12)

#get transpose
t(m)

# show w/out element #6
m[-6]
# and save that way
m<-m[-6]

# more vector manipulation (run these commands to see output)
> m <- cbind(1, 1:7) # for example, this is useful for adding columns to make a GLM by hand
> m
> m <- cbind(m, 8:14)[, c(1, 3, 2)]
> m
> m <- cbind(m, 8:14)[, c(1, 2, 3, 4)]
> m

#analogous operation for matrix elements:  MM[-27,] = skip row 27 (etc)
```

a (silly) linear model

```
# make a silly example of a linear model w/ two factors
r<-c(11.9044, 9.0088, 11.8362, 10.7099)

#              Shapiro-Wilk normality test
shapiro.test(r)
##W = 0.8631, p-value = 0.2714  ## this p-statistic shows that
# the elements of the vector r do not violate normality

# add four more elements – let each one of the four equal  the mean of r
# mean(r) = 10.86482
yy<-c(r,rep(mean(r),4))
# yy = 11.90440  9.00880 11.83620 10.70990  //  10.86482 10.86482 10.86482 10.86482

# make the factor categories
group<-c(rep(1,4), rep(2,4))

data = data.frame(r = r, group = factor(group)) # create a data set for command lm()
fit<-lm(r~group,data) # estimate the lm() object on model r~group using data

# print summaries on-screen
anova(fit)
summary.aov(fit)
summary.lm(fit)
```

linear models (somewhat of a cookbook) (1)

```
# remove all variables (cleanup) , but be careful to use it.
rm(list=ls(all=TRUE))

# read data in numeric format from a .csv (comma-delimited) file :
X=read.table("mydata.csv",header=TRUE,sep=',') ;
## attach() means you can use the header in the csv file as a variable name
## (I believe you can also create a string for variable names and attach)
## w/out attach you'd need to call a variable something like X$myvariable
##    or, do something lm(Y~b1+b2, data="X")
attach(X)
# caveat w/ attach() - It's possible to get error messages w/ attach(X)
# if the variable name exists in memory
# do search() and ls() to check local variables

## some useful packages
library(Design) # for robust covariates
library(systemfit) # for seemingly unrelated regression
library(lmtest) # bptest =  Breusch-Pagan test for heteroscedasticity
library(moments) # skewness, kurtosis
library(faraway) # more stuff with Cook's D and other things
```

linear models (somewhat of a cookbook) (2)

```
#take a look at the correlation matrix
cor(X[sapply(X, is.numeric)])

#make an object which is a linear regression model

# assume a linear regression Y  ~ b1 + b2 + b3 + b4
# lin.model.object<- Y  ~ b1 + b2 + b3 + b4


# add an interaction by hand to a general linear model
model.name.object <- Y ~ b0 + b1 + … + bn + b1*b2

# the following is the same as b0+b1+b1*b2
model.name.object <- Y ~ b1*b2

# make another object which is the linear model run on object model.name
# the command lm works with regressors and factors and so can do ANOVA, regression, and
ANCOVA
# package (current version of) lme for linear mixed models
# also check package for generalized linear models (extension to logistic regression)

lin.model <-lm(model.name.object)
```

linear models (somewhat of a cookbook) (3)

```
# some diagnostics

## normality test on residuals
shapiro.test(  rstandard(lin.model)  )



## collinearity
# get the condition number of the design matrix; a diagnostic of collinearity
# Not sure how this best handles missing data

XX<-cbind(b1 ,   b2  ,  b3  ,  b4 )


# run kappa with exact T, this is the same as running condition number in matlab
XX.cond.number<-kappa(XX[-27,],exact=T) ## 158.484
```

linear models (somewhat of a cookbook) (4)

```
##  Heteroscedasticity
# formal testing :  Studentized Breusch-Pagan test for heteroscedasticity
# It tests whether the estimated variance of the residuals from a regression
#    are dependent on the values of the independent variables.

# library(lmtest)
bptest(lin.model)

# hetetoscedasticity by eye
# Look at the scatters of the residuals (this is same as /scatterplot(*zresid *pred) in SPSS):
plot(fitted(lin.model), residuals(lin.model), xlab="Fitted", ylab="Residuals")

# Skewness and kurtosis of residuals
library(moments)

#Un-standardized, I believe, residuals
skewness(residuals(lin.model))

kurtosis(residuals(lin.model))
```

linear models (somewhat of a cookbook) (5)

Choose data “Y” such that a condition is satisfied on an attribute of Y (in SPSS syntax you’d be running SELECT IF commands:

```
> mydata<-read.csv(”mydata.csv",h=TRUE);
> attach(mydata)
> names(mydata)  ## show me the names of the variables

# make two subsets for each of two locations
> hippdata<-subset(mydata,subset=(LOCATION=="hippocampus"))
> cortdata<-subset(mydata,subset=(LOCATION=="cortex"))

# choose Alanine spectra from cortex, such that standard dev. of Alanine estimate < 20%
s1<-subset(cortdata,subset=(Ala_SD<20))

> s1$FACTOR
## s1$FACTOR is a “vector” of my factors, for each and all spectra in my chosen location
##                  which survived the threshold on the standard deviation.
# output is something like:
# [1] treated untreated untreated treated treated treated untreated
# Levels: treated untreated

# OK, now run a t-test between treated and untreated:
t.test( subset(s1$Ala_normalized, subset=(s1$FACTOR==“treated")) , subset
(s1$Ala_normalized,subset=(s1$FACTOR==“untreated")) )
```

Another example: nonlinear fit to data

Control data were synthesized using the three-component formula

$$y = a_1\exp(-t/b_1) + a_2\exp(-t/b_2) + a_3\exp(-t/b_3).$$

Seven time points were used. Additive noise was drawn from a zero-centered Gaussian to mimic typical χ^2 values of actual experimental data.

The synthetic data were fitted to

- $a_1\exp(-t/b_1)$;
- $a_0+a_1\exp(-t/b_1)$;
- $a_1\exp(-t/b_1)+a_2\exp(-t/b_2)$; and
- $a_1\exp(-t/b_1)+a_2\exp(-t/b_2)+a_3\exp(-t/b_3)$.

Numerical fit was performed using two standard nonlinear-square fitting algorithms (*nls* in R and *lsqnonlin* in MATLAB), both employing the Levenberg-Marquardt algorithm.

nonlinear fit to data

- Fitting sums of decaying exponentials is a notoriously ill-posed inverse problem.
- Here is the outcome of fitting the sum of N=3 exponential decays w/ R and Matlab, using a single exponential, single exponential + noise floor, two exponentials, and three exponentials.
- The outcome can also be sensitive to the starting initial conditions, even in the absence of additive noise to the synthetic model.

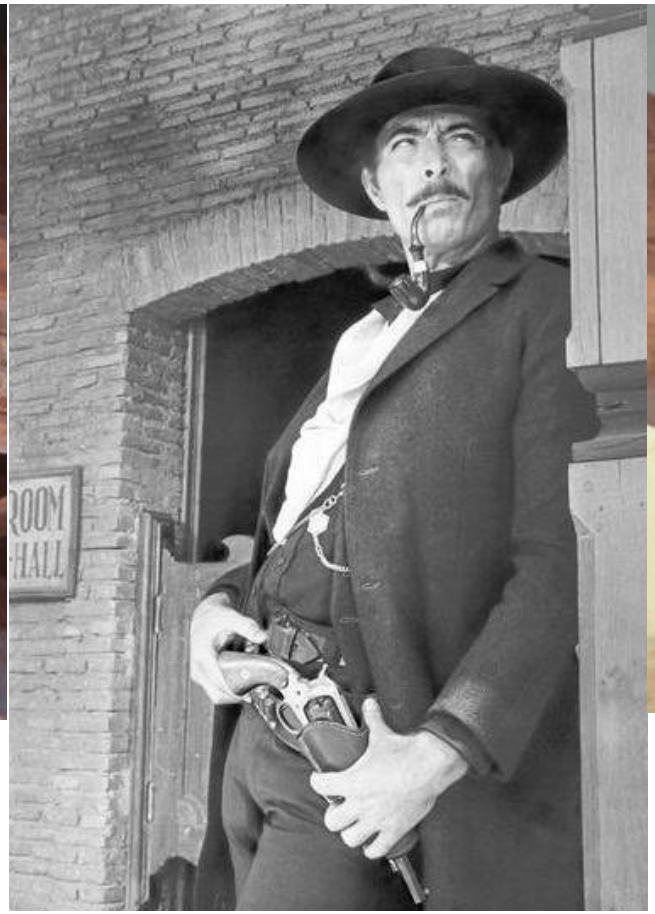
A priori parameters		R[a]					MATLAB[b]							
		A	B	C	D(¶)	%Δ(§)	A	B(†)	B	C	D	%Δ(§)	D(⋆)	%Δ(§)
a_1	0.4	0.8	0.2	0.5	-	-	0.8	0.2	0.2	0.5	0.5	20.6	0.4	
a_2	0.3		0.6	0.7	-	-		0.6	0.6	0.7	0.0	99.8	0.3	
a_3	0.3				-	-					0.7	120.5	0.3	
b_1	25.5	596.0	322.1	20.2	-	-	596.0	327.7	311.7	19.8	20.1	21.1	38.5	51.0
b_2	338.3			750.8	-	-				746.6	555.5	64.2	743.4	119.7
b_3	2728.7				-	-					750.2	72.5	1660.5	39.1

(a) R version 2.11.1, function *nls*.
(b) MATLAB, function *lsqnonlin*.
(†) Fit algorithm preconditioned using regression on the log data to determine a starting point for the Levenberg-Marquardt algorithm.
(§) Percent difference between the estimated parameter p_i^* $\left(p_i^* = a_i^*, b_i^*\right)$ and the corresponding *a priori* parameter p_i.
(¶) The three-component *nls* fit resulted in singular gradient and did not evaluate fitting parameters.
(⋆) Parameter estimation for noiseless data $(\eta = 0)$.

The Cast

The Good

The Bad



The Ugly

The guy who made the slides