Introduction to R for Economists

Basic Programming Principles and Data Analysis (Part 1)

June 28, 2019

Outline

Administration I

- Part 1 learning objectives:
  - Basic universal programming skills.
  - Getting started with RStudio and coding in the R language.
    - Using the command line, defining variables, importing data, plotting, matrix operations, saving and running your code, regressions...
  - Some tips & tricks for your econ homework, RA work, and future research projects.
- Required software:

  R and RStudio Can be downloaded free of charge from `http://www.rstudio.com/`
- I will show you example code, switching between slides and RStudio. We will go slowly, so that you can follow along on your own laptops.
- Afterwards, I will distribute these slides, along with example code and practice problems that you can try at home.
- There's a list of additional resources at the end of this slide deck, in case you want to learn more.

Good programming practice I

Regardless of your choice of language, there are general practices that will help streamline the programming process and facilitate collaboration with others.

1. Use a consistent **coding style**: choose conventions for structuring your code and naming objects (variables, functions, etc.), and stick to them.
   - For example, Google's R Style guide recommends:
     - `variable.name` or `variableName`, `FunctionName`.
     - Line length should not exceed 80 characters.
     - Curly braces (`{`, `}`) are used for control statements: first on same line, last on own line.
     - Use `<-` for assignment, instead of `=`.
     - Comments start with `#` followed by a space; two spaces before the `#` for inline comments.
   - More detailed style guide: `http://style.tidyverse.org/index.html`.
   - Some people indicate the object type in variable names, e.g. `vX` for vectors and `mX` for matrices. This can get complicated; R has many object types.
   - Conventions may differ by language. E.g., periods in names of variables are common in R, but cannot be used in Matlab, Java, or C++.
   - RStudio has tools to help keep your coding style consistent:
     `Tools > Global options > Code > Diagnostics`.

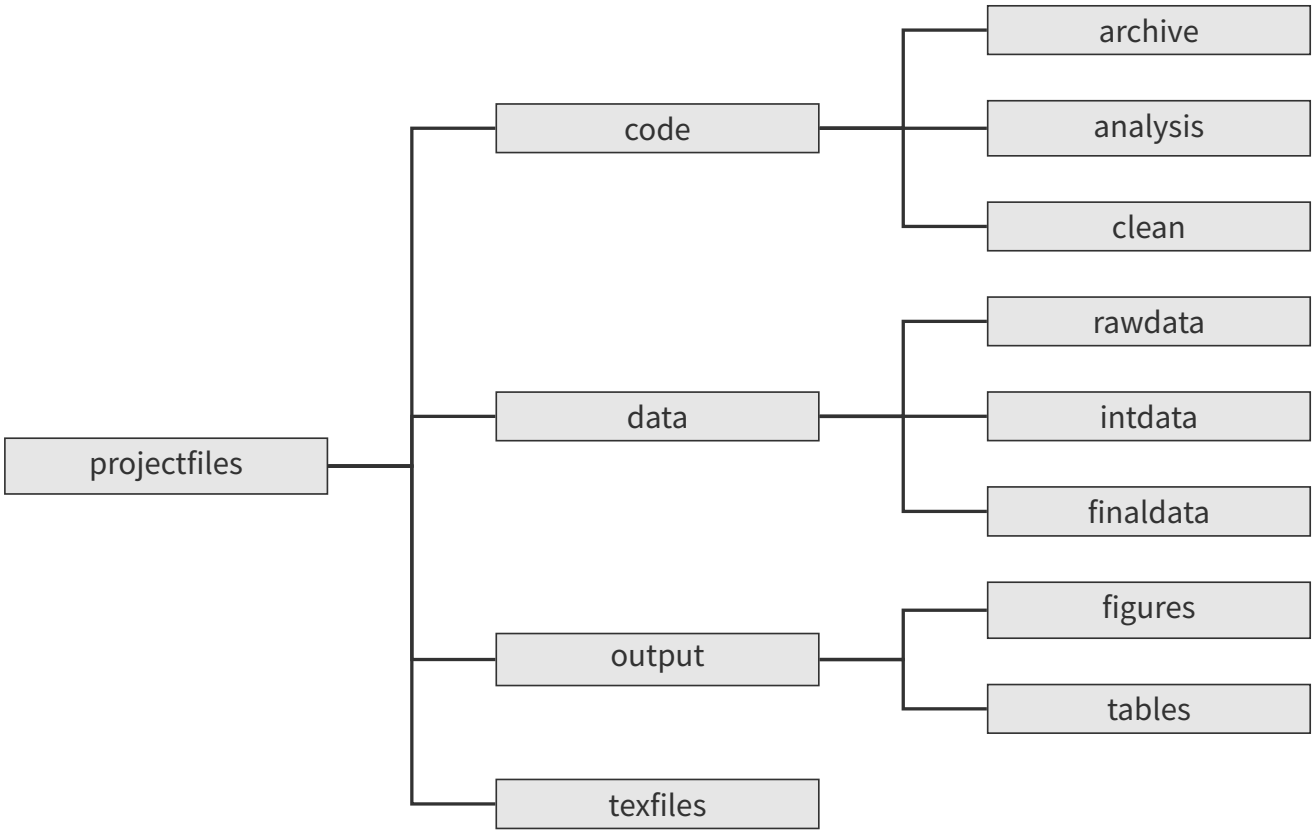
Good programming practice II

2. Include many detailed **comments** in your code files. Use comments to add structure, and to explain what lines or chunks of code are intended to do.
   - It's important that your code is *very* easy to understand, for other people and for your future self. You will feel like you are exaggerating, but you are almost certainly not overdoing it.
   - It also helps to give your variables clear and descriptive names, e.g. `educ` and `GDP` instead of `X` and `Y`.

3. **Organization (1)**: for larger projects, have a single piece of code that generates all of the results.
   - Makes it easy to replicate your results.
   - You can store auxiliary code in separate files to increase readability.
   - Call this file `main.R` (or `main.do` or `main.m`).

Good programming practice III

4. **Organization (2)**: for large projects that involve data, organize your code and folders in a consistent and transparent way.
   - For example, I like using the following structure:



   - Include a `readme.txt` file in the main folder to explain the structure to other users.

Good programming practice IV

5. **Automation**: If you want your code to repeat the same action, you should write it in a way that allows you to re-use the relevant chunk of code by calling it (do not copy-paste chunks of code!).
   - Defining your own functions is very useful.
   - Allows you to fix errors much faster by making changes only in one place.

- More good practices: *Code and Data for the Social Sciences: A Practitioner's Guide* (by Matt Gentzkow and Jesse Shapiro),

  `http://web.stanford.edu/~gentzkow/research/CodeAndData.pdf`.

  - This guide details best practices for managing research projects. You will certainly benefit from reading it in your own future projects or when doing RA work. It is mostly geared towards Stata users, but many of the principles apply to R users as well.

About R I

- R is a programming **language and environment** that is developed specifically for implementing statistical methods.
  - It is designed for matrix handling, statistical analysis and data visualization.
  - By contrast, Matlab was created with engineering applications in mind.
- Importantly, R is **open source**, and it is used by most statisticians, which makes R highly responsive to new developments in methodology.
  - There are thousands of free extension **packages** available online, and there is a large **community** of R users who are willing to share their knowledge and code.
  - A (somewhat outdated) New York Times article on the growing popularity of R: http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?pagewanted=1&_r=0.
  - By contrast, Stata and Matlab require paid licenses.
- In my opinion, it's best to have at least basic knowledge of several programming languages, so that you can always choose the most appropriate tool for the task at hand... But if you are an economics major and you were going to learn only one language, then R is currently the best available option.

About R II

The R packages ecosystem:

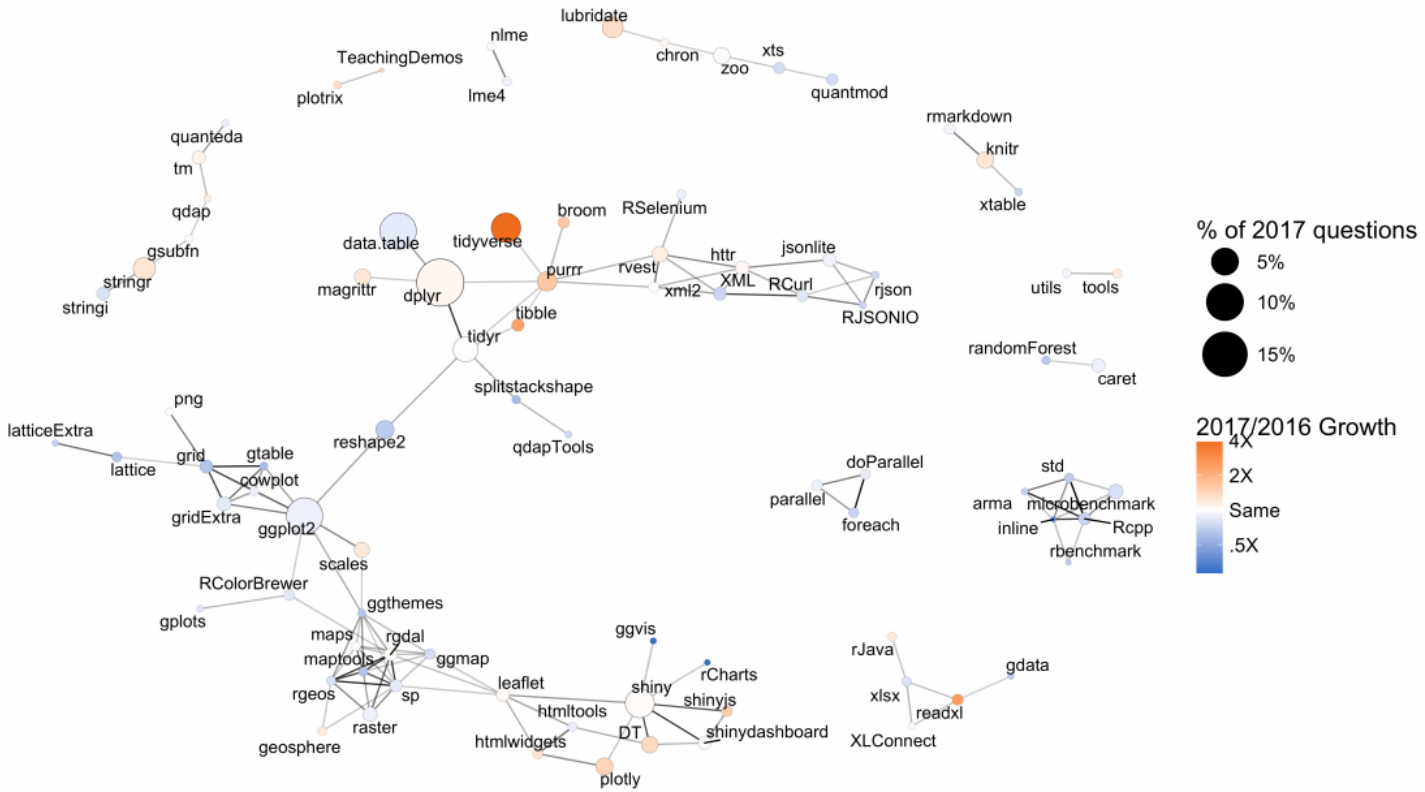


Figure 1.1: The 'Ecosystem of R packages': correlations are based on packages mentioned in Stack Overflow answers on the same question. This image is taken from https://stackoverflow.blog/2017/10/10/impressive-growth-r.

About R III

The R packages ecosystem:

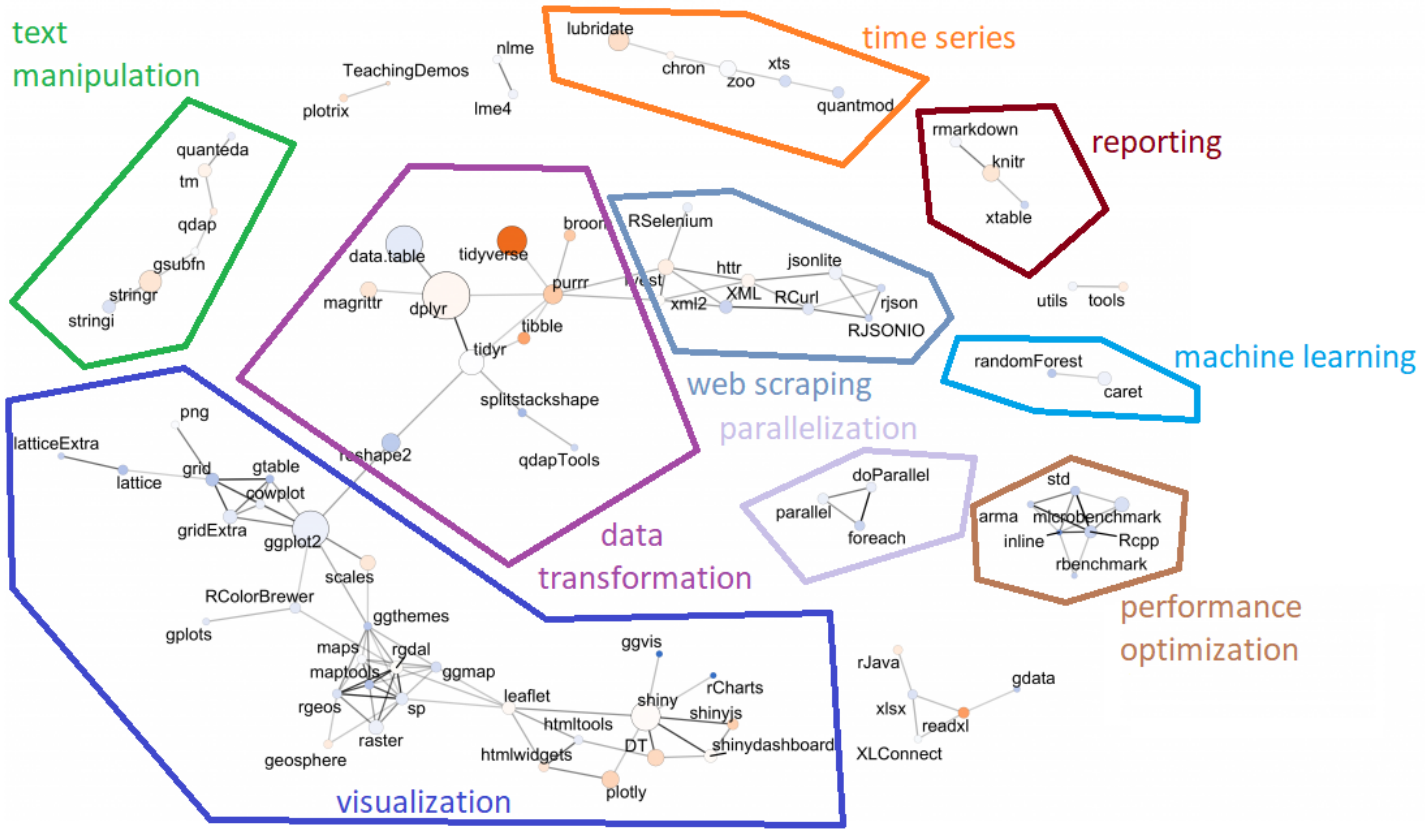


Figure 1.2: The 'Ecosystem of R packages': correlations are based on packages mentioned in Stack Overflow answers on the same question. This image is taken from https://stackoverflow.blog/2017/10/10/impressive-growth-r.

About R IV

The number of packages published on CRAN (=Comprehensive R Archive Network):

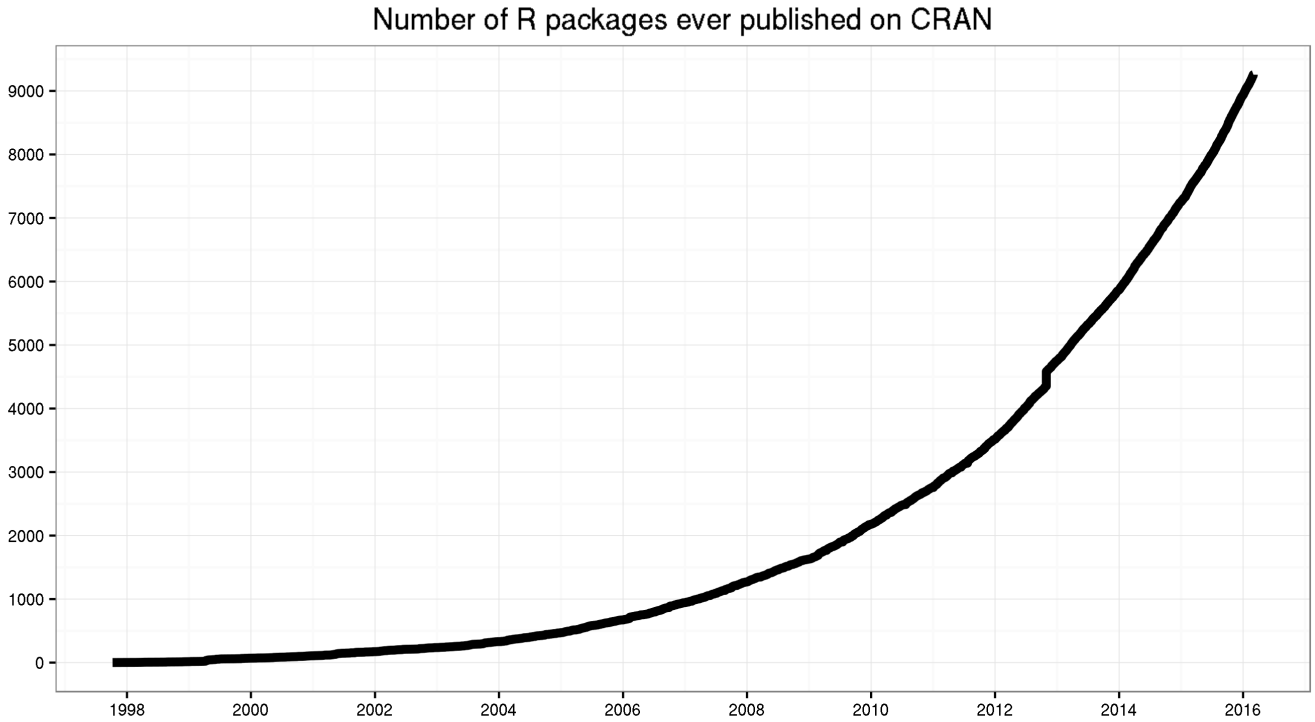


Figure 1.3: Number of published R packages over time. This image is taken from `https://gist.github.com/daroczig/3cf06d6db4be2bbe3368`, which also contains an instructive example of R code for extracting data from web pages.

About R V

- R vs. Python, what's the difference?

	R:	Python:
Intended users	Statisticians	General Purpose
Within econ	Micro, metrics	Machine Learning, Macro
Cost	Free	Free

- So what should you use?
  - For small projects and simple tasks, it doesn't matter much.
  - The choice between Matlab and Python (and Stata) becomes important when you're doing very specialized tasks; having a package or a toolbox available that implements the complicated estimation method you want to use could save you lots of time!
    - To explore a data set, try out a recently developed machine learning technique, or do data analysis using methods not built into Stata $\longrightarrow$ use R or Python.
    - For projects involving more traditional econometrics, use R. For projects using newer methods, e.g. techniques within machine learning, Python might be a better choice.

About R VI

- For projects involving more general purpose programming (e.g. web scraping), use Python.

RStudio I

- The standard, built-in R GUI looks like this:

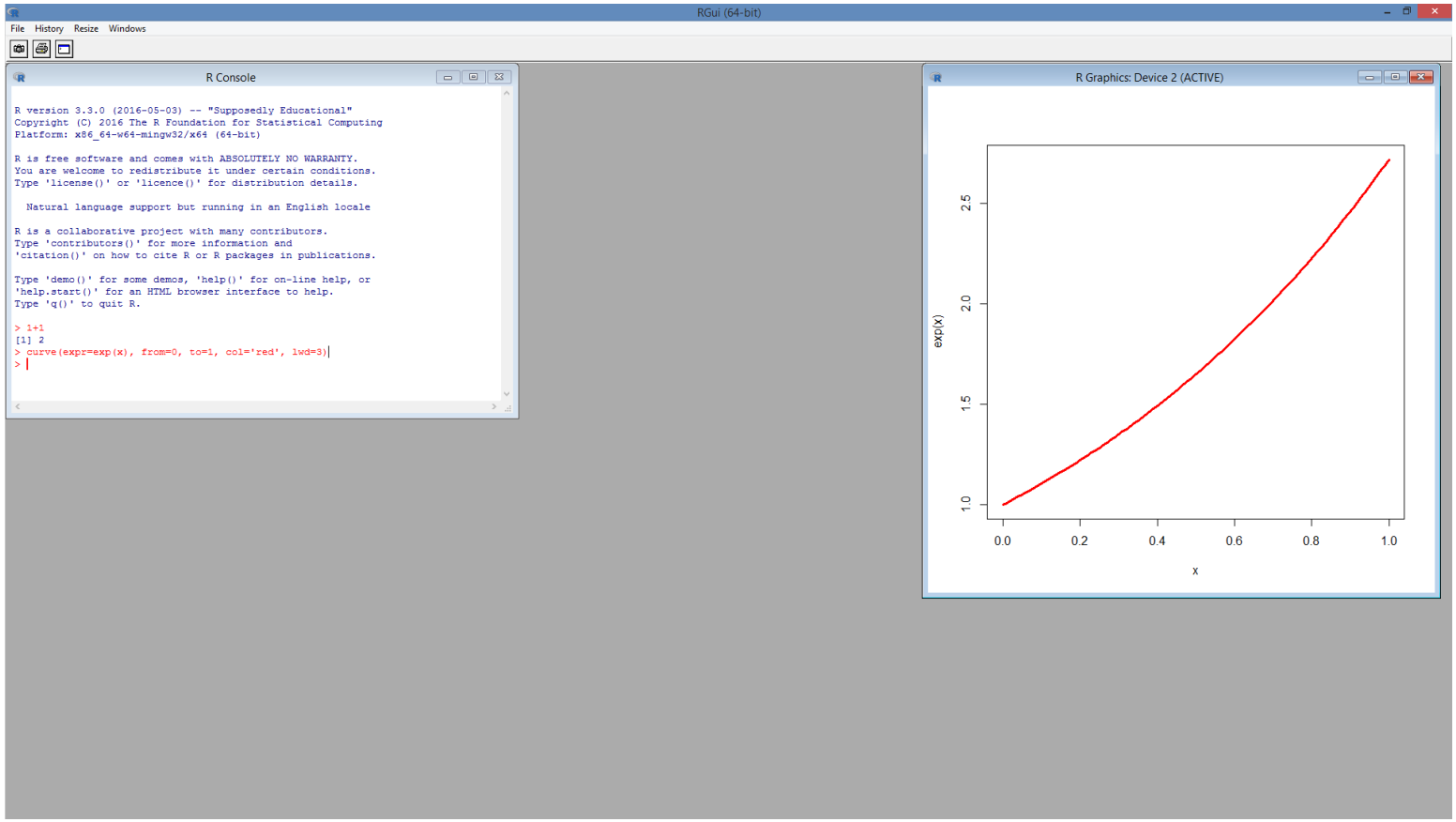


Figure 1.4: R environment.

RStudio II

- Rather than use the standard R environment as a graphical interface for R, I recommend that you use **RStudio**.
- Both R and RStudio can be downloaded for free and are available for Windows, Mac, and Linux/UNIX:
  - Download R: `http://cran.r-project.org/`.
  - Download RStudio Desktop: `https://www.rstudio.com/products/rstudio/`.

RStudio III

- The RStudio interface looks something like this:

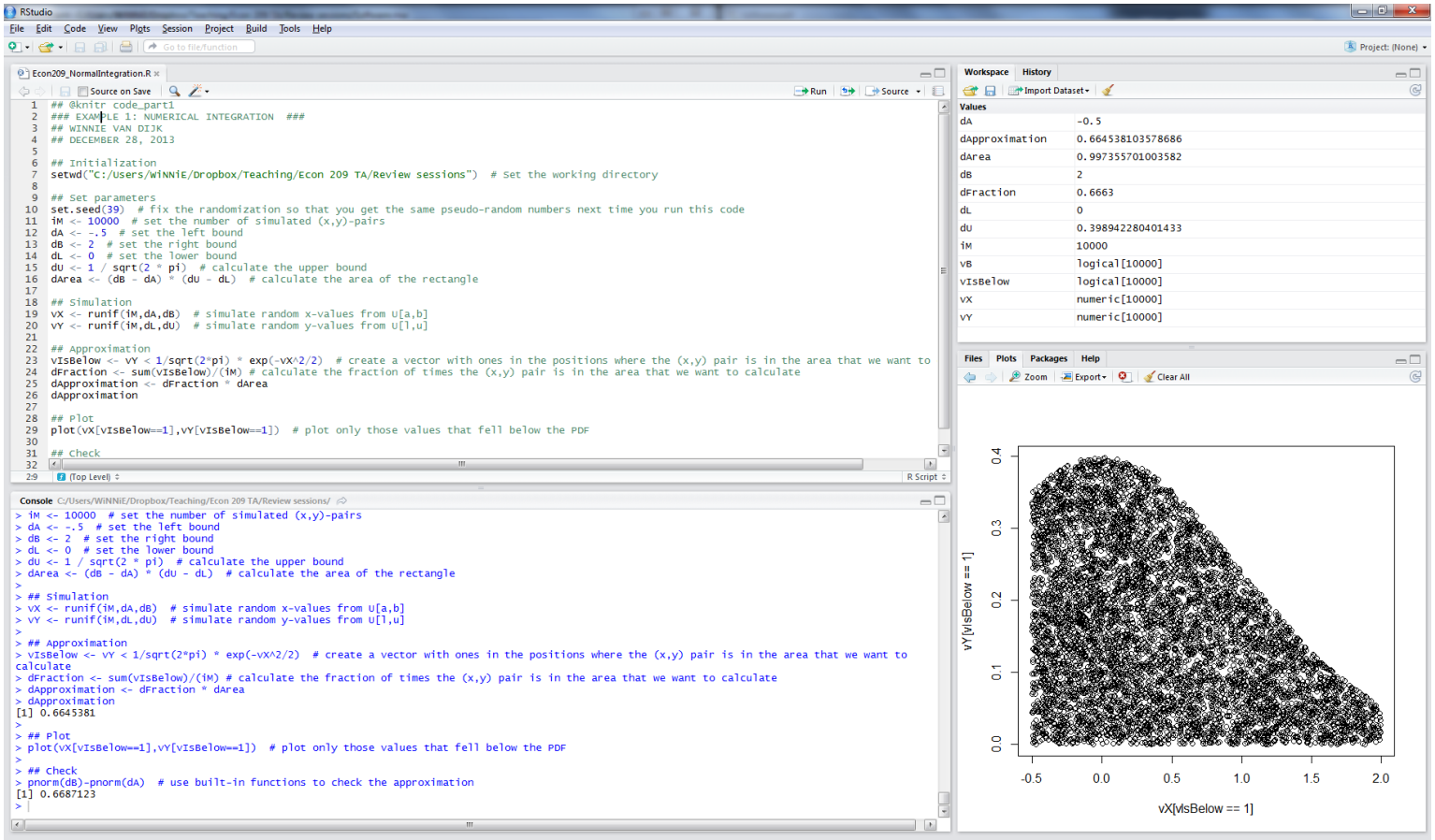


Figure 1.5: RStudio.

RStudio IV

- You can change the interface (arrangement of the panels, color scheme, ..) by selecting 'Tools' and then 'Global Options' in the menu above.
- Let's try it out...
  - Commands from the demonstration are in the file `REU_commands.R` on the website.
  - Solutions to exercises are also in that file.
- A useful shortcut in RStudio is `alt+shift+k`: it shows an overview of all shortcuts.

Exercise 1.1 (Simple calculations)

Using the command line, compute the following:

(a) $\sqrt{e^2}$

(b) $\Phi(.5)$, where $\Phi(\cdot)$ is the standard Normal CDF (Hint: Look up the correct syntax for `pnorm()` in the 'Help' tab).

(c) $\max\{1, [\sin(\pi/2)]^2\}$

Vectors and matrices I

- Before we go through a more challenging example, let's briefly practice working with vectors and matrices in R.

Exercise 1.2 (Vectors)

(a) Define the vectors

$$\texttt{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \qquad \texttt{y} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}.$$

Do this once using the `c()` command and once using `seq()`. Also try `1:3`.

(b) What is the difference between `x*y` and `x%*%y`?

(c) Using `x` and `y` as in the previous exercise, try `cbind(x,y)` and `rbind(x,y)`.

(d) Using `x` and `y` as before, what is the difference between `max(x,y)`, `pmax(x,y)`, `apply(cbind(x,y),1,max)`, and `apply(cbind(x,y),2,max)`?

Vectors and matrices II

- **Indexing**: use square brackets to extract one or several elements of the vector.
  - For example, `x[1]` extracts the first element of vector `x`.

Exercise 1.3 (Vectors, continued)

Define $\mathtt{x} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$.

(a) Sum all the elements.

(b) Calculate the mean.

(c) Replace the first element by 0.

(d) Add 1 to all elements and save the result in a vector `y`.

(e) What is the result of `x[c(1,3)]`? `x[2:4]`? `x[-2]`? `x[x<4]`?

(f) Add 1 to the first, third, and fifth element of `x`. (Using a single statement).

(g) Add 1 to all elements of `x` with a value strictly less than 4. (Using a single statement).

(h) What happens if we type `x[7] <- 7`? And `x[9] <- 9`?

Vectors and matrices III

Exercise 1.4 (Matrices)

Create the following matrices:

$$\texttt{A} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}, \qquad \texttt{B} = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 6 \end{bmatrix}.$$

To create *A*, use the command `matrix()`. To create *B*, use the command `diag()`.

(a) What is the difference between `A*B` and `A%*%B`?

(b) What does `A[1,]` do? And `A[,1]`?

(c) What does `t(A)` do? And `solve(A)`? Why doesn't this command work? Does it work on `B`?

(d) Replace the $(2, 2)$ element of `A` by 1.

(e) Replace the diagonal of `A` by ones.

(f) Store the first two rows of `A` in a new matrix `C` (so that `C` is a 2×3 matrix).

(g) What does `dim(C)` do? And `nrow(C)`?

Example: numerical integration I

- Let's go over some basic R commands by walking through a longer example. We'll try to approximate an integral.
- Recall the example of calculating π by throwing darts at a board. Suppose we take draws from a distribution uniformly distributed over the unit square $[0, 1] \times [0, 1]$. Take *N* draws. Calculate the fraction of draws that land within the circle with center at the origin and radius equal to one: $y^2 + x^2 = 1$. This is equivalent to computing the integral

$$\int_0^1 \sqrt[+]{1 - x^2} \mathrm{d}x.$$

- We don't want to calculate the integral analytically. Instead, we can use *simulation* to approximate it.
  - In R, it is easy to generate 'pseudo-random' values, and to evaluate the function $\phi(x)$.

Example: numerical integration II

- The idea is to "throw darts" at a rectangle of known area that encloses the area under the graph. In R, we can easily assess what fraction of the darts has landed in the area that we want to approximate (the fraction that has landed below the PDF), and take this fraction of the total known area as our approximation.

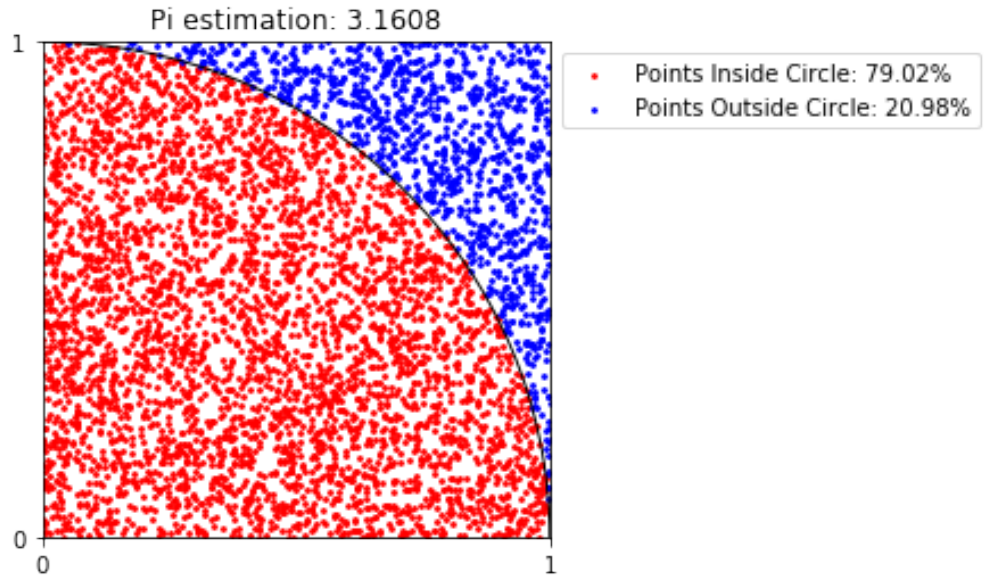


Figure 1.6: Numerical integration example.

Example: numerical integration III

- The code is in the file `REU_integration.R`.

Functions I

- Suppose that we want to re-use the code from the previous example to repeatedly approximate

$$\int_0^1 4\sqrt[+]{1-x^2}\mathrm{d}x.$$

for some user-specified value of N. Then it is convenient to write a function that takes as input the **parameter** N.

- The syntax for defining a function is

```
<FunName> <- function(<par1>,<par2>) {
                 <...commands...>
                 return(<output>)
             }
```

Functions II

- The code in the file `REU_function.R` defines such a function, building on the numerical integration example.
- You can copy-paste the contents of this file to the command line, or you can save the file as an `.R` file in your working directory so that it can be sourced from the command line by typing

```
source("REU_function.R").
```

- After sourcing this code (look for `NumInt` in the workspace to see if this was successful), we are able to call the function from the command line, or from another `.R` file.
  - E.g., type `NumInt(1000)` on the command line.

Functions III

- Things to notice in the function code:
  - It is good practice to include **error handling**. For example, the user might specify $N > 0$, for which this code will return the error message that we specified.
  - **Comments** add structure, make it easy to read the code.
  - R suppresses **intermediate output** in a function. Using `print()` for printing intermediate output in a function can help you when testing/debugging your code.
  - An `if` statement was used for handling errors. The syntax for such a statement is:

```
if(<logical statement>) {
                    <...commands...>
                }
```

    - A logical statement can be either true or false. In our example, `N > 0`.
    - We can combine logical statements, using

```
<logical statement1> & <logical statement2>
<logical statement1> | <logical statement2>
```

Functions IV

Exercise 1.5 (Logical statements)

Using the command line, type

```
a <- 1;   b <- 0.
```

Which relationships are tested by the following commands?

(a) `a == b`

(b) `a >= b`

(c) `a < 0`

(d) `a != b`

(e) `a | b`

(f) `a & b`

(g) `!a`

Control statements: if, for, while I

- A control statement is a statement that determines whether other statements will be executed.
  - We just saw the syntax for an `if` statement. If the logical statement inside parentheses is true, the commands between the curly braces are executed.
  - We will see two other examples: `for` loops and `while` loops.
- `for` loop syntax:

```
for(i in seq) {
    <...commands...>
}
```

`seq` stores the values that `i` can take. It is usually a range of consecutive integers, but it can also be a set of strings.

- `while` loop syntax:

```
while(<logical statement>) {
    <...commands...>
}
```

Control statements: if, for, while II

- The code in `REU_control.R` simulates a data set using a `for` loop. The data-generating process is

$$Y_i = a + bX_i + \varepsilon_i, \qquad i = 1, .., N,$$

where $X_i \sim (3, 1), \varepsilon_i \sim (0, 1), a = 2$ and $b = .5, N = 200$.

  - Data is simulated for X_i and ε_i, $i = 1, .., N$, and stored in vectors

$$\mathtt{X} = \begin{bmatrix} X_1 & \cdots & X_N \end{bmatrix}'; \mathtt{E} = \begin{bmatrix} \varepsilon_1 & \cdots & \varepsilon_N \end{bmatrix}'.$$

  - We can use a `for` loop to generate the Y_i values. We first generate a vector of NA's (NA stands for"Not Available"), and then we 'walk through' the vector using a **counter** or **index** `i`. In every step of the loop, we replace the NA by a value.
    - At the end of each step, the counter is automatically updated.

Control statements: if, for, while III

- This is not the fastest way to generate the simulated data... It would have been faster to just write

```
Y <- a + X * b + E
```

  - Note that R adapts to the object it is asked to work on: R understands the $+$ and $*$ operations as operations on vectors rather than scalars.
  - This is both an advantage and a disadvantage of R: sometimes it's convenient, and sometimes it produces unexpected output. You should always check whether R is doing what you want it to do when vectors and matrices are involved.
  - For example, when you add two vectors of unequal length, R will recycle elements of the shorter vector to match the longest vector, and this may not be what you want. Matlab would give you an error message.

- A `while` loop can be used to do the exact same thing as the `for` loop from the previous example. Things to pay attention to in the code:
  - A `while` loop can have more flexible stopping criteria than a `for` loop. But you should make sure the stopping criterion will be satisfied before you run out of memory...
  - The counter is not automatically updated. To update in every step, 'self referencing' is used: `i <- i+1` adds 1 to `i`.

Regression I

- We can use the data that we have just simulated to run a regression.
  - The syntax for **simple regression** is

```
<output.name> <- lm(Y ~ X, data = <data.name>)
```

  - The syntax for **multiple regression** is

```
<output.name> <- lm(Y ~ X1 + X2, data = <data.name>)
```

  - These statements produce objects that can be read by `summary()` to produce a quick overview of the regression results in RStudio.
  - You will often want to produce **LaTeX tables**. Two packages that will be useful are `xtable` and `stargazer`.
  - A demonstration of how to produce summary statistics, load and save data, and running regressions is in `REU_regression.R`.

Regression II

Table 1.1: Produced with the `xtable` package.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.5505	0.2027	7.65	0.0000
X	0.6421	0.0632	10.16	0.0000

Regression III

Table 1.2: Produced with the `stargazer` package.

	Dependent variable:
	Y
X	0.642^{***}
	(0.063)
Constant	1.550^{***}
	(0.203)
Observations	200
R^2	0.343
Adjusted R^2	0.339
Residual Std. Error	0.906 (df = 198)
F Statistic	103.218^{***} (df = 1; 198)
Note:	*p<0.1; **p<0.05; ***p<0.01

Regression IV

Table 1.3: Produced with the `stargazer` package, incorporating output from two regression specifications.

	Dependent variable:	
	Y	
	(1)	(2)
X	0.642***	0.639***
	(0.063)	(0.063)
Z		0.084
		(0.065)
Constant	1.550***	1.385***
	(0.203)	(0.240)
Observations	200	200
R^2	0.343	0.348
Adjusted R^2	0.339	0.342
Residual Std. Error	0.906 (df = 198)	0.904 (df = 197)
F Statistic	103.218*** (df = 1; 198)	52.602*** (df = 2; 197)
Note:		*p<0.1; **p<0.05; ***p<0.01

Syntax: differences between R and Matlab I

- The table below is based on David Hiebeler's Matlab/R reference, which you can find here: `http://www.math.umaine.edu/~hiebeler/comp/matlabR.pdf`.

Syntax: differences between R and Matlab II

	R:	Matlab:
Computations	`a+b, a-b, a*b, a/b`	`a+b, a-b, a*b, a/b`
	`sqrt(a), exp(a), sin(a)`	`sqrt(a), exp(a), sin(a)`
	`â b`	`â b`
	`a&b, a|b, a==b, a!=b`	`a&b, a|b, a==b, a~=b`
Vectors	`x <- c(1,2,3)`	`x = [1 2 3]` (row vector)
		`x = [1; 2; 3]` (column vector) ← !
	`x[2]`	`x(2)`
	`x[2:length(x)]`	`x(2:end)`
	`x[1:length(x)-1]`	`x(1:end-1)`
	`1:6`	`1:6`
	`seq(1,6,by=2)`	`1:2:6`
	`seq(1,6,len=100)`	`linspace(1,6,100)`
	`t(x)`	`x'`
Matrices	`A <- matrix(1:6, nrow=2)`	`A = [1 3 5; 2 4 6]` ← !
	`A[1,2]`	`A(1,2)`
	`A[1,]`	`A(1,:)`
	`cbind(x,y)`	`[x y]`
	`rbind(x,y)`	`[x; y]`
	`A %*% B`	`A*B`
	`A * B`	`A.*B`
	`solve(A), det(A)`	`inv(A), det(A)`
	`solve(A,b)`	`A \ b` ← !
	`mean(A)`	`mean(A(:))`
	`colMeans(A), rowMeans(A)`	`mean(A), mean(A,2)`

Syntax: differences between R and Matlab III

Exercise 1.6 (System of linear equations)

Solve the following system of linear equations, using Matlab:

$$\begin{aligned} x_1 + 2x_2 - 2x_2 &= 1 \\ 3x_1 + 5x_2 + 6x_3 &= 2 \\ 7x_1 + 8x_2 + 9x_3 &= 3 \end{aligned}$$

Hint: Recall `A \ b` from the previous slide.

Syntax: differences between R and Matlab IV

	R:	Matlab:

if	`if(<logical statement>) {`
	`  <...commands...>`
	`}`
		`if <logical statement>`
		`  <...commands...>`
		`end`

if/else	`if(<logical statement>) {`
	`  <...commands...>`
	`} else{`
	`  <...commands...>`
	`}`
		`if <logical statement>`
		`  <...commands...>`
		`else`
		`  <...commands...>`
		`end`

for	`for(i in seq) {`
	`  <...commands...>`
	`}`
		`for i=seq`
		`  <...commands...>`
		`end`

while	`while(<logical statement>) {`
	`  <...commands...>`
	`}`
		`while (<logical statement>)`
		`  <...commands...>`
		`end`

Syntax: differences between R and Matlab V

R:

function
```
<FnName> <- function(<x1>,<x2>) {
   <...commands...>
   return(<output>)
}
```

Matlab:

```
function out.name=<FnName>(<x1>,<x2>)
   <...commands...>
out.name = <output>
```

Returning multiple values, objects that may have different types:

R:

function
```
<FnName> <- function(<x1>,<x2>) {
   <...commands...>
   return(list(<out1>,<out2>))
}
```

Matlab:

```
function [o1, o2]=<FnName>(<x1>,<x2>)
   <...commands...>
[o1, o2] = [<out1>, <out2>]
```

Recommended resources I

R:

W.N. Venables, D.M. Smith and the R Core Team - An Introduction to R

2017 version

`http://cran.r-project.org/doc/manuals/R-intro.pdf`

RStudio: Tips for learning R

`http://www.rstudio.com/online-learning/`

Google's R Style Guide

`https://google.github.io/styleguide/Rguide.xml`

The tidyverse style guide

`http://style.tidyverse.org/index.html`

TryR

Online interactive tutorial for first steps in R

`http://tryr.codeschool.com/`

Recommended resources II

Stack Overflow, Cross Validated

Forums for programming and statistics questions

`http://stackoverflow.com/` and `http://stats.stackexchange.com/`

Google Developers' Intro to R

Youtube tutorial videos – geared towards Mac users

`http://www.youtube.com/playlist?list=PLOU2XLYxmsIK9qQfztXeybpHvru-TrqAP`

Hadley Wickham - Advanced R

Companion website for Advanced R book

`http://adv-r.had.co.nz/`

Garrett Grolemund & Hadley Wickham - R for Data Science

Companion website for R for Data Science book

`http://r4ds.had.co.nz/`

Recommended resources III

Tom Short - R Reference Card

A useful cheat sheet, though somewhat outdated

`http://cran.r-project.org/doc/contrib/Short-refcard.pdf`

Knitr page, by Yihui Xie

Instructions for installing and using Knitr, a tool for integrating R code into LaTeX documents

`http://yihui.name/knitr/`

Recommended resources IV

Matlab:

MathWorks video tutorials

`http://www.mathworks.com/support/learn-with-matlab-tutorials.html?requestedDomain=www.mathworks.com`

David Hiebeler - MATLAB / R Reference

`http://www.math.umaine.edu/~hiebeler/comp/matlabR.pdf`

James P. LeSage - Applied Econometrics using MATLAB

`http://www.spatial-econometrics.com/html/mbook.pdf`