R and RStudio tutorial - Math 4450/5450 - Spring 2015 - Zirbel

R is a programming language that is used for statistics. RStudio is an Integrated Development Environment (IDE), a place where you can edit R programs, run them, and keep track of the variables in the programs and the graphics windows that R produces.

Installing R and RStudio

If you installed R a long time ago, you might want to uninstall it and install the most recent version. I just installed version 3.1.2.

To install R, go to `http://cran.rstudio.com/`, find your platform, and pick your way through the links that get R installed. Generally, you would rather use compiled binaries than source code, tarballs, or anything like that.

To install RStudio, go here: `http://www.rstudio.com/products/rstudio/download/` and look for the section called **Download RStudio Desktop v0.98.1091** or some higher-numbered version. I suggest that you find the Installer for your platform, click that, wait patiently, and install it.

When you start RStudio, the window will be divided into sub-windows. One of these is the Console window, with the prompt >. The commands below should be typed at the prompt. Don't type the line numbers or the comments at the ends of the lines.

R basics

1:	`2*9`	R can be used as a calculator.
2:	`exp(1)`	The exponential function.
3:	`2^999`	
4:	`x=exp(1)`	You can store values with variable names. This sets the value of `x` equal to `exp(1)`.
5:	`x`	This displays the value of `x`
6:	`x <- exp(1)`	Alternate way to assign a value to `x`, often used by people writing about R. But why use two characters when one will do?
7:	`x=runif(10)`	`x` is a collection of 10 random numbers, uniformly distributed between 0 and 1. `x` is like a row of cells in a spreadsheet. It is called a **vector**.
8:	`x+5`	Add 5 to each entry of `x`.
9:	`1000*x`	Multiply each entry of `x` by 100.
10:	`round(150*x)`	Round to the nearest integer.
11:	`ceiling(100*x)`	Round up to the next integer. This is how you generate random numbers uniformly distributed from 1 to 100.
12:	`x=runif(100)`	Go ahead, try something larger than 100.
13:	`max(x)`	
14:	`mean(x)`	
15:	`sum(x)`	
16:	`median(x)`	
17:	`sort(x)`	Sort x in increasing order.
18:	`cumsum(x)`	A vector of cumulative sums of entries of `x`.
19:	`P = matrix(0,ncol=20,nrow=20)`	Examine `P` to see what you get.
20:	`rowSums(P)`	Calculate the sums of the rows of `P`.
21:	`P[1,2] = 0.5`	Set one value in the matrix `P`.

Plotting points and histograms

22:	`plot(x,pch=16)`	Plot the values of `x` against numbers 1, 2, ... as filled dots.
23:	`plot(cumsum(x),type="l")`	Plot the cumulative sum against the numbers 1, 2, ..., connecting points with lines.
24:	`dev.off()`	Clear the plotting window.

25: `hist(x)` Make a histogram of the values of `x`.
26: `hist(x,breaks=20)` Use 20 bins instead of the default 10.
27: `y=-log(x)` Take the natural logarithm of the entries of `x`.
28: `hist(y,breaks=20)` Looks like an exponential distribution. It will look better with more data points, so re-define `x` and do it again.
29: `?hist` To get help, type ? followed by the name of a command. Sometimes you can guess what you want and then read about it. Or google your question.
30: `up arrow` Press the up arrow to return to previous commands, hit enter to run them again.
31: `control-L)` On a PC, clear all the old commands in the console window.
32: `cat("\014")` On a Mac, clear all the old commands in the console window.

Downloading R programs for random processes from Github

33: I have posted a number of R programs on Github. To find them, google "Github zirbel random processes" or go to `https://github.com/clzirbel/Random_Processes` and find the Download Zip button or go straight to `https://github.com/clzirbel/Random_Processes/archive/master.zip` to get the .zip file. Unzip the file and save it in an accessible place.
34: In RStudio, use the File, Open File menu to navigate to where you unzipped the programs from Github, then find the R folder within it, and open the file `gambler_outcome.R`. It will open in an editor window. It is a good idea to read the commands in it and start to understand what they do. I have added many comments to the commands in the file, separated from the commands by the # character.
35: `getwd()` Figure out what R considers to be the working directory. This is where it looks for programs and where it writes out files. I suggest that you use the directory where you have downloaded and unzipped files from Github. The easiest way to change the working directory is to go to the Session menu, Set Working Directory, To Source File Location if you have already opened a file. Or, use Choose Directory.
36: `source("gambler_outcome.R")` The source function will load the file and run the commands in it. Make sure you have the working directory set right. Instead of typing this, you can click the button that says Source in the upper right of the editor window. The program simulates gambler's wealth for 80 steps. If the game goes longer than that, it does all the remaining steps at once. (I could not figure out how to get R to extend the axes dynamically.)
37: Run the program `gambler_outcome.R` multiple times, then quit. In the program, find the line that pauses after each step, set the pause time to 0.0, and then source it again. Now each game will take just a moment to generate and draw, and you can get a better sense of what they look like.

R functions

38: In RStudio, use File, Open File to open `source("gambler_functions.R")`. Use the Source button on the upper right of the editor window to have RStudio load the file and run the commands in it.
39: `P = gambler_transition_matrix()` Set up the transition matrix for Gambler's wealth.
40: `P` Show the matrix.
41: `P = gambler_transition_matrix(10,20,0.5)` Specifically tell R what initial wealth, opponent wealth, and win probability to use. Check that `P` looks the same.
42: `P = gambler_transition_matrix(5,5,0.7)` Set up a transition matrix for a different gambling situation. Look at `P` now.
43: Read through the file `gambler_functions.R`. There are several function definitions, for example, this one: `gambler_transition_matrix = function(m=10,n=20,p=0.5)`. It says that there are three inputs, which will be called `m, n,` and `p` within the function. If the user does not give values for them, the default values will be 10, 20, and 0.5, respectively.

Matrix operations and plotting

44: `install.packages("expm")` Do this once to install the package on your computer.
45: `library(expm)` Do this once per R session to use the expm library, which allows R to calculate matrix powers. If you source matrix_functions.R (see below), then this is done for you.

46: `install.packages("gplots")` Do this once to install the package on your computer.

47: `source("matrix_functions.R")` Load functions for matrix calculations.

48: `print_matrix(P)` Show the matrix so that it can be copied and pasted into another program easily.

49: `transition_matrix_powers(P,0)` Show six powers of P. If you get "Error in plot.new() : figure margins too large" try making the figure window larger.

50: `P %^% 20` Calculate the 20th power of P. This uses the expm package.

51: `pdf("filename.pdf",width=8.5,height=11)` If you want to save a plot as a PDF, type this command, then make the plot, and then do the next command to actually write the file.

52: `dev.off()` Do this after plotting, to save the file.