Risk Analysis in the Earth Sciences: A Lab Manual with Exercises in R

7 September 2015

Edited by Patrick J. Applegate and Klaus Keller

Contributions by Patrick J. Applegate, Ryan L. Sriver, Gregory G. Garner, Richard B. Alley, and Klaus Keller

---

Summary

Patrick Applegate, patrick.applegate@psu.edu, and Klaus Keller, klaus@psu.edu

7 September 2015

Greenhouse gas emissions have caused considerable changes in climate, including increased surface air temperatures and rising sea levels. Rising sea levels increase the risks of flooding for people living near the world's coastlines. Managing such risks requires an understanding of many fields, including Earth science, statistics, and economics. At the same time, the free, open-source programming environment R is growing in popularity among statisticians and scientists due to its flexibility and graphics capabilities, as well as its large collection of existing software libraries.

This e-textbook presents a series of laboratory exercises in R that teach the Earth science and statistical concepts needed for assessing climate-related risks. These exercises are intended for upper-level undergraduates, beginning graduate students, and professionals in other areas who wish to gain insight into academic climate risk analysis.

Acknowledgements

7 September 2015

The exercises discussed in this lab manual were developed as part of a course titled Risk Analysis in the Earth Sciences taught in the Department of Geosciences at the Pennsylvania State University. This course was designed and originally taught by Klaus Keller. Many of these exercises were co-developed by a large group, particularly Patrick Applegate, Klaus Keller, and Ryan Sriver; see the list of contributors on the cover page.

The development of this e-textbook was partly supported by the US National Science Foundation through the Sustainable Climate Risk Management (SCRiM) network. Other support was provided by the US Department of Energy, the Center for Climate Risk Management, and the Rock Ethics Institute.

PJA thanks the students from the Risk Analysis in the Earth Sciences course, Spring 2013 and Spring 2014, for many helpful comments that improved the exercises. PJA also thanks AJR for support during the writing of this e-textbook.

KK thanks the students in the Risk Analysis in the Earth Sciences course, the members of the Keller research group, and his SCRiM collaborators for invaluable inputs. Special thanks go to his family, many mentors, and friends for support.

Introduction

Patrick Applegate, patrick.applegate@psu.edu, and Klaus Keller, klaus@psu.edu

7 September 2015

Climate science and climate risks

We begin with a very brief discussion of some key findings of climate science and the problems posed by climate change. There are many useful reviews of these subjects, in both textbook and popular formats (e.g., Kump et al., 2009; Alley, 2011). For high-level discussions of these topics, see the Intergovernmental Panel on Climate Change's recent reports, particularly the Technical Summaries of the IPCC's Working Groups I and II (Stocker et al., 2013; Field et al., 2014). We particularly recommend that you read chapters 1, 3, 6, 15, and 16 in Kump et al. (2009) before working on the exercises in this e-textbook.

Most scientists who study the climate system agree that human activities are causing surface air temperatures to increase (e.g. Oreskes, 2007). Combustion of fossil fuels and changes in land use lead to increases in the concentrations of carbon dioxide, methane, and nitrous oxide in the atmosphere. The increased concentrations of these gases trap heat near the earth's surface that would normally escape to outer space. Temperatures have risen measurably over the last century and will probably continue to increase in the future (Alexander et al., 2013; Collins et al., 2013).

This increase in surface air temperatures affects other parts of the Earth system, notably sea level. Rising surface air temperatures cause the water near the ocean's surface to expand, and also lead to increased rates of melting from land ice. Global mean sea level has risen by a few tens of centimeters over the past 1-2 centuries, and could go up by more than one meter by 2100 (e.g. NOAA, 2012; Alexander et al., 2013; Church et al., 2013).

Increases in sea level create risks for people living near present-day sea level. Sea level at a particular point along the world's coastline rises and falls over time scales of hours to days, in response to tides and storms. Stacked on top of the long-term rise in sea level, these short-period oscillations flood normally-dry land with increased frequency (e.g. Spanger-Siegfried et al., 2014).

Fossil fuel consumption also creates benefits for people, although these benefits are unevenly distributed. Countries with low per capita gross domestic products also tend to emit very little carbon dioxide per person. Among countries with high per capita gross domestic products, some emit large amounts of carbon dioxide per person each year, whereas others emit comparatively little. However, achieving good economic outcomes in today's energy economy seems to require at least some fossil fuel consumption. Moreover, the benefits of fossil fuel consumption occur now, whereas the negative effects of climate change will take place primarily in the future.

Thus, reducing greenhouse gas emissions creates costs in terms of lost economic productivity. Modern societies require large amounts of energy to manufacture goods, produce food, and move people from place to place. Today, most of that energy comes from fossil fuels (IEA, 2014). Reducing fossil fuel consumption likely means cuts to manufacturing, agriculture, or transportation, at least in the short term.

Some questions

The discussion above leads to several interrelated questions:

1. How much carbon dioxide, and other greenhouse gases, will human societies emit in the future (e.g. Sanford et al., 2014)?
2. How much will surface air temperatures rise in response to past and future greenhouse gas emissions?

3. How much will sea level rise as a consequence of surface air temperature increases?
4. What effects will increases in temperature and sea level, as well as changes in precipitation and other climatic variables, have on people?
5. Given the benefits associated with fossil fuel consumption and the risks presented by climate change, what is the appropriate balance between economic growth, cutting greenhouse gas emissions, and building protective measures to moderate the negative effects of climate change?
6. In choosing a strategy for addressing climate change, how do we take account of values like fairness? For example, how do we ensure that the negative impacts of climate change do not fall disproportionately on people who have not contributed to the problem?

Answering these questions requires contributions from several disciplines, including the social sciences (especially economics; questions 1, 4, and 5), meteorology (2), oceanography (3), glaciology (3), and philosophy (6). It also requires estimating future changes, often with computer models.

Statistics also plays an important role in answering these questions, because our answers always contain some level of uncertainty. Future changes in temperature and sea level depend on emissions (question 1), which are difficult to predict. Moreover, climate projections are made with imperfect computer models that depend on input parameters whose values we aren't sure of. When we make an estimate of future changes using a computer model, how sure should we be about the answer? How can we report our answers in a way that accurately reflects the state of our knowledge? Statistics is the discipline that provides answers to these questions.

Levels of (un)certainty

Suppose that we wish to characterize the state of our knowledge about a future event whose outcomes can take values on the real number line. Depending on the event in question, this quantity can be *certain*, *uncertain*, or *deeply uncertain*. These "levels of uncertainty" (Riesch, 2013) have different mathematical representations:

- If the outcome is certain, it can take just one value.
- If the outcome is uncertain, but not deeply uncertain, there is a well-defined and generally agreed-upon probability density function that relates different potential outcomes to the probability that they will occur (see Exercise 3 for a discussion of probability density functions). For example, rolling a die has six possible outcomes (the integers 1, 2,... 6), and each of these outcomes is equally likely to occur if the die is fair.
- If the outcome is deeply uncertain, there is no single, agreed-upon probability density function describing the chance of observing different outcomes.

Many important questions in climate science involve deep uncertainty. Published estimates of the climate sensitivity provide perhaps the clearest demonstration of this point. Climate sensitivity is the change in global mean temperature that would result if carbon dioxide concentrations in the atmosphere were to double. Many studies have estimated this quantity. Though these studies have obtained overlapping answers, their probability density functions differ (Collins et al., 2013, their Fig. 1 in Box 12.2; see also Meehl et al., 2007, their Fig. 9.20).

What is risk?

Answering many important questions requires careful consideration of *risk*. Risk is the harm associated with a particular bad outcome, multiplied by the chance of that bad outcome occurring. For example, answering questions 5 and 6, above, requires considering the harms associated with future climate change, given that we are not sure how severe it will be (questions 1-4). A possible future event can represent an important risk if

it is relatively benign, but very likely to occur, or if it is very unlikely, but extremely damaging if it does happen.

Classic risk analysis typically assumes that the probability density functions relating outcomes to the chances of their occuring are well-known. However, as noted above, many questions in climate science have deeply uncertain answers. In these cases, other methods have to be used to assess the merits of different approaches for managing climate risk. One such approach is robust decision making (RDM; Lempert et al., 2012; see also Hadka et al., 2015), which focuses on identifying the weaknesses of particular systems and then identifying solutions which produce good results over a wide range of potential outcomes.

Risk analysis in the Earth sciences often involves considering low-probability, high-impact events. For example, at least one study suggests that large temperature increases could make important parts of the Earth's surface metabolically uninhabitable (Sherwood and Huber, 2010). Animals, including humans, have to maintain a body temperature within a relatively restricted range in order to survive. Climate modeling indicates that future climate change could raise temperatures near the equator so much that human beings could not live there, except in air-conditioned buildings. This possibility is unlikely, but would have important effects on human societies.

"Uncertainty" vs. "risk"

The meanings that we assign to "uncertainty", "deep uncertainty", and "risk" seem intuitive to us; however, other researchers use these words to mean quite different things. The taxonomy we use here separates future events into groups depending on whether or not we have a good sense of the probabilities associated with different outcomes, and it distinguishes the chances of different outcomes from their consequences. However, an influential early taxonomy of uncertainty (Knight, 1921) used "risk" and "uncertainty" to represent the concepts that we term "uncertainty" and "deep uncertainty," respectively. We have also heard influential scientists use the word "uncertainty" to mean the same thing as what we term "risk." Readers of this e-textbook should be aware of these varying definitions when looking at other sources.

Computer programming in R

To help answer the questions above, it makes sense to use a computing environment that is flexible, designed to be used for statistics, and has good plotting capabilities. Although not the only possibility, R may be the best tool for the job. A *New York Times* article (Vance, 2009) lists some of R's advantages:

- it is widely used in industry and academia (Revolution Analytics maintains a list of companies that use R)
- it allows users to perform complex analyses without a deep knowledge of computer science
- it is free and open-source
- it is flexible (there are thousands of user-developed libraries that extend the language's core functionality)

The *Times* article goes on to explain that, as R's popularity increases, other closed-source statistics packages are becoming less commonly used. These trends have become more noticeable in the years between the *Times* article and the writing of this e-textbook; Robert Muenchen has written a detailed blog post tracking the popularity of different software packages for statistics and data analysis. According to this article, R is commonly mentioned as a desirable skill in online job postings, and is gaining in popularity among academic users relative to closed-source packages such as SAS and SPSS.

About this book

This e-textbook presents a series of laboratory exercises that introduce students to the statistical aspects of risk analysis in the Earth sciences using R. Most of these exercises have to do with sea level rise, which is a major focus of the authors' research activities.

The exercises are intended for upper-level undergraduates, beginning graduate students, and professionals in other areas who wish to gain insight into academic climate risk analysis. Previous programming experience is helpful, but not required; the first exercise explains how to learn the basics of R.

Each exercise begins with a description of the Earth science and/or statistical concepts that the exercise teaches. Next, each exercise presents a detailed explanation of a piece of existing R code. Finally, students are asked to either modify the existing code or write their own code to produce a well-defined outcome. Each exercise also includes questions to encourage classroom discussion and reflection on the part of individual students.

A few chapters contain appendices that present additional discussion of the topics raised in the preceding parts of those chapters. These appendices are intended for graduate students or researchers who have additional background in programming or the Earth sciences. They can be skipped by other readers with no loss in comprehension.

The book is designed to be modular. The first nine exercises (0-8) were written by Patrick Applegate, with contributions by others as noted in the individual chapters. Other exercises may be added by members of the Keller research group in the future.

An important note

Carrying out the exercises in this e-textbook will not teach you to perform publication-quality or consulting-grade risk analyses. This e-textbook is intended primarily for advanced undergraduates, and the material presented here should only be applied to "real" problems after additional training. The authors and editors of this e-textbook specifically disclaim any liability for damages associated with the use of the material presented in this e-textbook.

References

Alexander, L. V., et al., 2013. Summary for policymakers. In Stocker, T. F., et al., eds., Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press. Available online at http://www.climatechange2013.org/report/full-report/

Alley, R. B., 2011. Earth: The Operators' Manual. W. W. Norton and Company, 496 p.

Church, J. A., et al., 2013. Sea level change. In Stocker, T. F., et al., eds., Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press. Available online at http://www.climatechange2013.org/report/full-report/

Collins, M., et al., 2013. Long-term climate change: projections, commitments, and irreversibility. In Stocker, T. F., et al., eds., Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press. Available online at http://www.climatechange2013.org/report/full-report/

Field, C. B., et al., 2014. Technical summary. In Field, C. B., et al., eds., Climate Change 2014: Impacts, Adaptation, and Vulnerability. Part A: Global and Sectoral Aspects. Contribution of Working Group II to

the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press. Available online at http://ipcc-wg2.gov/AR5/report/full-report/

Hadka, D., Herman, J., Reed, P., and Keller, K., 2015. OpenMORDM: an open source framework for many-objective robust decision making. Environmental Modeling and Software. doi: 10.1016/j.envsoft.2015.07.014

IEA (International Energy Agency), 2014. Key world energy statistics. Available online at http://www.iea.org/publications/freepublications/publication/key-world-energy-statistics-2014.html

Knight, F. H., 1921. Risk, uncertainty, and profit. Library of Economics and Liberty. Available online at http://www.econlib.org/library/Knight/knRUP.html

Kump, L. R., Kasting, J. F., and Crane, R. G., 2009. The earth system (3rd ed). Prentice Hall, 432 p.

Lempert, R., Sriver, R. L., and Keller, K., 2012. Characterizing uncertain sea level rise projections to support investment decisions. California Energy Commission White Paper CEC-500-2012-056.

Meehl, G. A., et al., 2007. Global climate projections. In Solomon, S., et al. (eds), Climate Change 2007: The Physical Science Basis. Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press. Available online at https://www.ipcc.ch/publications_and_data/ar4/wg1/en/contents.html

NOAA (National Oceanic and Atmospheric Administration), 2012. Global sea level rise scenarios for the United States National Climate Assessment. NOAA Technical Report OAR CPO-1. Available online at http://cpo.noaa.gov/AboutCPO/AllNews/TabId/315/ArtMID/668/ArticleID/80/Global-Sea-Level-Rise-Scenarios-for-the-United-States-National-Climate-Assessment.aspx

Oreskes, N., 2007. The scientific consensus on climate change: how do we know we're not wrong? In DiMento, J. F. C., and Doughman, P., Climate Change: What It Means for Us, Our Children, and Our Grandchildren. MIT Press, 360 p.

Riesch, H., 2013. Levels of uncertainty. In Roeser, S., Hillerbrand, R., Sandin, P., Peterson, M. (eds.), Essentials of Risk Theory. SpringerBriefs in Philosophy, Springer, 148 p. Available online at http://www.springer.com/us/book/9789400754546

Sanford, T., Frumhoff, P. C., Luers, A., and Gulledge, J., 2014. The climate policy narrative for a dangerously warming world. Nature Climate Change 4, 164–166. Available online at http://www.nature.com/nclimate/journal/v4/n3/full/nclimate2148.html

Spanger-Siegfried, E., Fitzpatrick, M., and Dahl, K., 2014. Encroaching tides: how sea level rise and tidal flooding threaten U.S. East and Gulf Coast communities over the next 30 years. Union of Concerned Scientists, 66 p. Available online at http://www.ucsusa.org/global_warming/impacts/effects-of-tidal-flooding-and-sea-level-rise-east-coast-gulf-of-mexico#.VbJZGXj6QfM

Stocker, T. F., et al., 2013. Technical summary. In Stocker, T. F., et al., (eds.), Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press. Available online at http://www.climatechange2013.org/report/full-report/

Vance, A., 7 January 2009. Data analysts captivated by R's power. The New York Times, p. B6. Available online at http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?pagewanted=all&_r=1&

Exercise #0: Learning the basics of R

Patrick Applegate, patrick.applegate@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- install R, RStudio, and `swirl` on your computer
- use `swirl` to learn the basics of R, including
  - assigning values to variables in R
  - the differences between vectors, matrices, and data frames
  - using indexing to extract individual values, or ranges of values, from variables
- modify an R script to produce a desired result
- get help by
  - searching R's built-in documentation
  - searching the Web
  - asking questions in a way that increases your chance of receiving a helpful response

Introduction

The computing environment and programming language R is a powerful tool for risk analysis. This exercise will guide you through installing R and its integrated development environment RStudio. You will also learn the basics of using R at the command line using `swirl`, plus how to run scripts in R. Finally, you'll become familiar with different ways of getting help when you encounter problems with your R code.

Tutorial

Installing R and RStudio

To install R, navigate to https://cran.rstudio.com and look for the link that says, "Download R for *[name of your operating system]*," near the top of the page. Click on this link and follow the instructions for installing the software. If you run into problems, look at the FAQ, which is linked to from the same page.

R's basic development environment is functional, but somewhat crude. RStudio provides a much cleaner environment for working with R. There are presently two versions of RStudio, one for companies (which costs money) and one for private individuals and academics (which is free). There are also versions for servers, as well as desktop computers. The free, desktop version works perfectly well for our purposes. **To install RStudio**, navigate to https://www.rstudio.com/products/RStudio/ and download the Open Source Edition of RStudio Desktop.

The RStudio working environment

Open RStudio. You should see a window that looks something like Figure 1. The RStudio Web page provides documentation on how to use RStudio, but we provide a short description here for the sake of completeness.

RStudio divides its window up into four different sub-windows, most of which have tabs. These sub-windows, and their tabs, are as follows.

- The **upper left** sub-window is where you write R scripts (series of R commands that are stored in text files). You can have multiple scripts open at the same time, and clicking on the different tabs lets you switch between these scripts.
- The **lower left** sub-window is the Console window, where you enter R commands one at a time and the output appears immediately in the same window. Some output from running scripts appears here, too. (The other tab, R Markdown, isn't important for our purposes here.)
- The **upper right** sub-window usually shows the variables that presently exist in R's working memory (if the Environment tab is visible). The other tab, History, shows the commands that have been entered in the Console window.
- The **lower right** sub-window is usually set to show either Plots or Help. The other tabs, Files, Packages, and Viewer, are less important for our purposes.

Most often, you will work in the Script sub-window, with occasional jumps to the Console, Plots, and Help sub-windows.

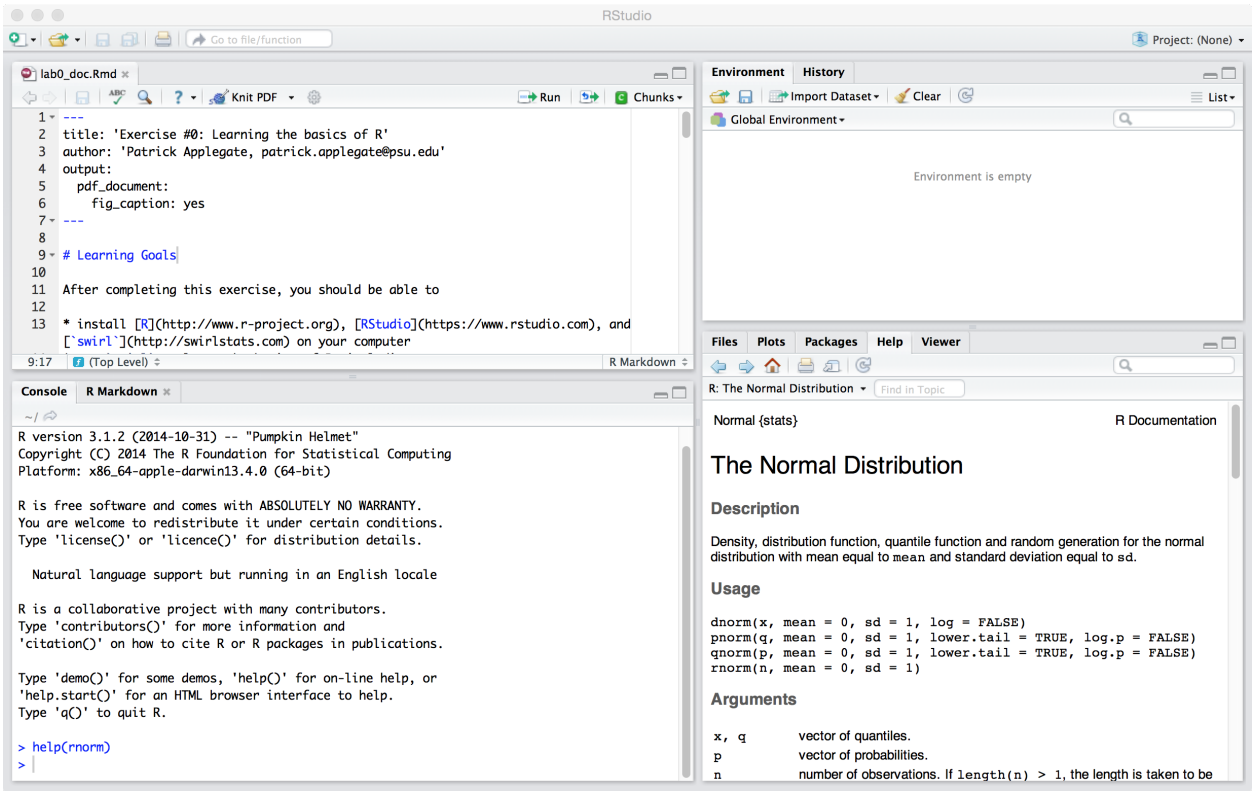

Figure 1: A representative screen shot from RStudio on a Mac. See text for discussion.

Installing `swirl`

`swirl` is a software library that will teach you to use R. The installation process for `swirl` is quite different

from the process you followed for installing R or RStudio. **To install `swirl`**,

- start RStudio, if you haven't already
- click in the Console sub-window in the lower left-hand corner of the RStudio window so that the cursor appears next to the prompt (a greater-than sign, `>`)
- carefully enter (or copy-paste) the following commands in the Console sub-window, pressing Enter after each one:

```
install.packages('swirl')
require('swirl')
install_from_swirl('R Programming Alt')
```

In the block of commands above, the first command installs the `swirl` package, the second loads the package into memory, and the third command installs materials for a basic course in using R.

When you enter the second command above, you should see a message that reads, `| Hi! Type swirl() when you are ready to begin.` Ignore this message. You'll need to enter this command each time you open RStudio if you plan to use `swirl` in that RStudio session.

When you enter the third command above, a small, red STOP sign should appear in the upper right-hand corner of the Console window. If all goes well, you should receive a message saying, `| Course installed successfully!` after a few minutes.

Learning the basics of R with `swirl`

Close and reopen RStudio. Enter each of the following commands, again pressing Enter after each one.

```
require('swirl')
swirl()
```

Follow the prompts and read the messages that appear. When you're given a choice of courses, enter the number corresponding to `R Programming Alt`, the course you installed previously. You'll be offered a list of 12 lessons.

Work through lessons 1-6 of the `R Programming Alt` course in order. Take notes on the material presented during each lesson. For example, Lesson 1 (`Basic Building Blocks`) introduces you to the commands `c`, `?`, `sqrt`, and `abs`. Note what each new command does.

Each `swirl` lesson should take you 10-20 minutes to work through, meaning that you'll need one to two hours (possibly more) to work through the first six exercises in the `R Programming Alt` course. If you stop working and come back, you'll need to enter both commands above each time you open RStudio and want to use `swirl`.

Scripting in RStudio

In the `swirl` tutorial, you worked mostly in the Console sub-window in RStudio, entering commands one at a time. However, real scientific analyses should be *reproducible*; that is, there should be a list of steps that anyone can follow to get the same results as the person who originally did the analysis. R provides the capability of saving a group of commands as a script, which can then be run later just as if the commands had been typed into the Console sub-window in RStudio.

Open the script `lab0_sample.R` in RStudio by clicking on File > Open File... and then navigating to the location of the file on your hard drive. The file should open in the Script sub-window in the upper left-hand corner of the RStudio window.

Take a few moments to look at this file. RStudio automatically colors parts of the code depending on their function.

- Black text represents commands.
- Blue text represents numerical values.
- Green text that begins with a pound sign (`#`) represent comments, notes to the programmer that aren't used by the computer when making calculations.
- Green text surrounded by quotation marks are strings (for example, `'string'` or `"string"`), groups of non-numeric characters that (unlike comments) *are* part of the calculation being performed.

Click on Session > Set Working Directory > To Source File Location and then click on the Source button in the upper right-hand corner of the Script window. When you do this, several things happen:

- A new line appears in the Console window, something like `source('some_path/lab0_sample.R')`, indicating that the script has been executed.
- Several new lines appear in the Environment window, showing the values of variables `mu`, `num`, `sigma`, and `x`.
- A new file, `lab0_fig1.pdf`, appears in the same directory as `lab0_sample.R`.

Open the new `.pdf` file. It should look like Figure 2, which shows a histogram and a red, vertical line near the apex of the histogram. The histogram represents 1000 random samples from a normal distribution with a mean of 1 and a standard deviation of 1.

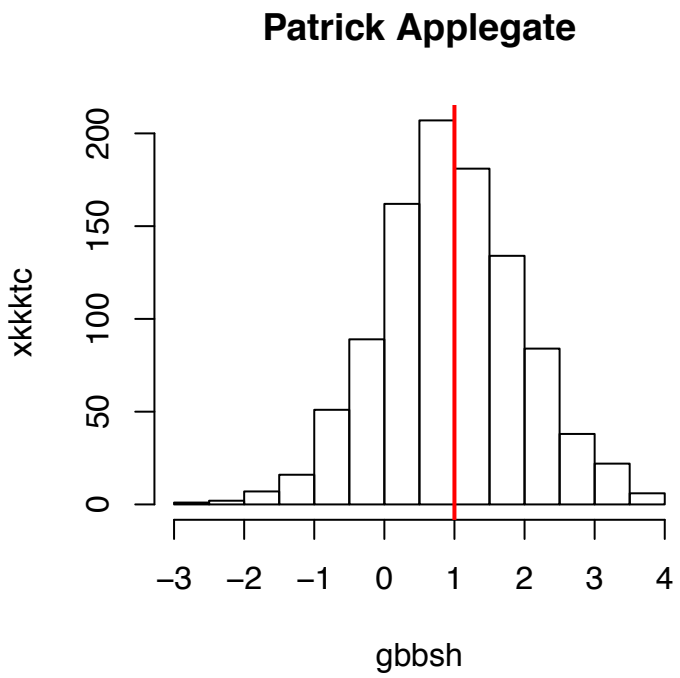

Figure 2: Contents of `lab0_fig1.pdf` after executing `lab0_sample.R`. See text for discussion.

In general, you will want to use the Source button to execute your scripts, rather than Run. To see what the Run button does, place the cursor at the top of `lab0_sample.R` and repeatedly click the Run button. The commands in the script will appear in the Console window. As you might expect, R executes the commands in a script one after another, in order from the top of the script file to the bottom.

Getting `help()`

You might be wondering what the commands in `lab0_sample.R` do. Looking at lines 14-16 in this file, we have

```
# Sample randomly from a normal distribution and assign these values to a
# variable called x.
x <- rnorm(n = num, mean = mu, sd = sigma)
```

The first two lines are comments, but the third line is a piece of code that starts with `x <-`. Based on your experiences with `swirl`, you probably realize that the results of any calculations on the right-hand side of the assignment symbol `<-` will be stored in the variable `x`. The right-hand side of this third line starts with `rnorm` and then has some parentheses surrounding some odd groups of characters separated by equals signs and commas. What is that all about?

As you know from your experiences with `swirl`, R functions are generally a string of characters followed by parentheses that contain the function's arguments. In the third line in the code block above, the function is `rnorm()`, which takes the arguments `n`, `mean`, and `sd`. These arguments are passed values stored in variables that were defined near the top of the script, `num`, `mu`, and `sigma`:

```
# Set some values.
num <- 10^3    # number of random draws
mu <- 1        # mean of normal distribution to draw from
sigma <- 1     # standard deviation of normal distribution
```

Try searching for the `rnorm` command with `help(rnorm)`. The help file that comes up is titled, "The Normal Distribution." It describes not just `rnorm()`, but also three other commands. Careful inspection of the help file suggests that `rnorm()` handles "random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`." Now we know what two of the arguments we mentioned above do, but what about the third? Looking at the list of arguments under **Arguments**, we see that `n` is the "number of observations," or the number of random values to generate.

You'll follow this procedure often. When you see a function that you don't understand, try searching for it in R's documentation using the `help()` command. Careful inspection of the relevant help files will usually give you important clues about what an unknown function does and how its arguments affect the output from the function.

However, sometimes R's documentation can be cryptic. Further down in the script, you'll see the line

```
hist(x, main = 'Patrick Applegate', xlab = 'gbbsh', ylab = 'xkkktc')
```

What do the arguments `main`, `xlab`, and `ylab` do? Here, `help(hist)` is unhelpful; the help file says only that "these arguments to `title` have useful defaults here," which conveys nothing to a beginning user of R. In this case, a Google search on "r hist" may be more useful; the fourth hit leads to an article on the "Basics of Histograms."

When the documentation and searching the Internet don't help

When you encounter problems in writing R code that you can't solve by looking at the documentation or searching the Internet, you'll want to ask someone who knows more about R than you do. When you do, you should try to ask your questions in a way that maximizes your chances of getting the help you need.

A document called "How to Ask Questions the Smart Way" suggests a strategy for asking questions. This document is long and its tone is curt, so we don't recommend that you read it, at least not right away.

Consider coming back to it after finishing the exercises in this lab manual. A shorter document is online here on the Stack Overflow Web site.

Before asking any questions, you should use the resources available to you: search the documentation and the Web. If you still need to ask someone, your question should reflect

1. what you are trying to do (the goal of your program)
2. what your program does that addresses the problem
3. what you were expecting to have happen when you ran your program
4. what actually happened (did you get an error message? if so, what did it say?)
5. what you did to try and solve your problem before asking your question

For an example of a well-written question, have a look at this post on Stack Overflow. Note that the original poster (user tjnel) addressed all of the five points above in his/her question; he/she

- says explicitly that he/she is trying to reproduce the results from a tutorial (question #1, above),
- gives the code that is being used, allowing others to fully understand the problem (#2)
- provides the expected output from running the code in the tutorial (#3),
- shows the error message that he/she obtained when running the tutorial code (#4),
- shows that he/she tried to reproduce a different part of the tutorial exercise before asking a question (#5).

Exercise

Modify `lab0_sample.R` so that it produces a .pdf file that contains a histogram of 10^4 random samples from a normal distribution with mean = 5 and standard deviation = 2. Indicate the mean of the distribution with a vertical, red line. Show the mean minus the standard deviation and the mean plus the standard deviation with two vertical, blue lines. The x-axis of the histogram should be labeled "Variable," and the y-axis should be labeled "Frequency (n = 10^4)." The title of the plot should be your name.

Exercise #1: downloading and plotting time series data

Patrick Applegate, patrick.applegate@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- explain the relationship between atmospheric carbon dioxide concentration, surface air temperature, and sea level
- download data from the Internet in R
- read data files into R
- make simple plots in R, including
  - plots with multiple panels
  - plots with multiple curves in the same panel

Introduction

This textbook on climate risk management will discuss sea level rise extensively. Many people live near present-day sea level (e.g., Nicholls et al., 2008), and rises in sea level expose these people to the possibility of flooding. For example, the ongoing increase in sea level will likely cause communities on the United States' eastern coast to experience frequent flooding within the next few decades (Spanger-Siegfried et al., 2014).

Sea level rise is caused by temperature increases, which in turn are driven by increases in carbon dioxide concentrations in the atmosphere. Carbon dioxide is produced by human activities and natural processes. Increases in carbon dioxide concentrations in the atmosphere enhance the trapping of infrared radiation near the Earth's surface and contribute to rises in surface air temperatures. As the ocean absorbs some of the excess heat from the atmosphere, its temperature increases, causing it to expand and causing sea level rise. Temperature increases cause melt of glaciers and ice sheets, which leads to sea level rise by adding mass to the oceans.

Data covering the last century support this relationship between atmospheric carbon dioxide concentrations, temperature, and sea level (Fig. 1). The curves in the three panels of Figure 1 rise together, suggesting that these variables are related. Although correlation does not prove causation, the combination of a clear relationship between variables with a plausible explanation for why they should be related is a strong argument for causation.

Frequent, accurate measurements of carbon dioxide in the atmosphere began in the late 1950s at Mauna Loa in Hawaii (Keeling et al., 1976; blue curve in Fig. 1, top panel). These measurements show an annual cycle that represents Northern Hemisphere seasons. Plants lose their leaves or die during the winter, releasing carbon dioxide to the atmosphere. The Northern Hemisphere has much more land, and therefore more plants, than the Southern Hemisphere. Thus, the Northern Hemisphere's seasons largely control the variability in atmospheric carbon dioxide concentrations within any individual year. However, there is a definite upward trend in this curve that is larger than the amplitude of the annual cycle.

Measurements of former atmospheric compositions from bubbles trapped in ice cores allow us to extend the observational record of carbon dioxide concentrations farther back in time (MacFarling Meure et al., 2006; black curve in Fig. 1, top panel). As snow falls on the Greenland and Antarctic ice sheets, it traps samples of the atmosphere. Because new snow buries and compresses old snow, the time at which different snow samples fell can be estimated by counting the layers in an ice sheet. The ice core measurements of atmospheric carbon

dioxide concentrations are less finely resolved in time than the direct measurements, and therefore don't reflect the annual cycle of CO_2 in the atmosphere. However, the trend of the ice core data is similar to that of the direct observations during the period when they overlap, suggesting that the ice core data are reliable.

Because carbon dioxide mixes readily in the atmosphere, measurements of atmospheric carbon dioxide concentrations at most places on the Earth's surface are relatively representative of the globe as a whole. In contrast, both surface air temperatures and sea levels are measured at widely dispersed stations and must be aggregated to give a global mean value. Global mean temperatures must be estimated from individual weather stations with long records (Hansen et al., 2010); past sea levels are estimated using data from tide gages (Jevrejeva et al., 2014). As one might expect, there are various methods for performing this aggregation, and the different methods give somewhat different answers. However, it seems clear that both global mean surface air temperature and global mean sea level are rising.

In this lab exercise, you'll examine an R script that downloads the data files needed to make the top two panels of Figure 1 from the Internet and plots them. You'll then modify the script so that it plots all three panels of Figure 1.

Tutorial

We'll start by looking at several Web pages that describe the data we'll be using in this exercise. As of May 2015, the various data sets displayed in Figure 1 are archived in the following places:

Table 1: Internet sources of data used in Exercise #1 and associated references. NOAA, National Oceanic and Atmospheric Administration; DOE CDIAC, United States Department of Energy – Carbon Dioxide Analysis Center; NASA GISS, National Aeronautics and Space Administration – Goddard Institute for Space Studies; PSMSL, Permanent Service for Mean Sea Level.

Data type	Reference	Location on the Web
CO_2, direct measurements	Keeling et al. (1976)	NOAA
CO_2, Law Dome ice core	MacFarling Meure et al. (2006)	DOE CDIAC
Surface air temperature change	Hansen et al. (2010)	NASA GISS
Sea level anomaly	Jevrejeva et al. (2014)	PSMSL

Click on the links in the right-hand column of Table 1 and look at the descriptions of the data stored there. Also look for links to the particular data sets mentioned in the Introduction. Some Web pages contain links to multiple data sets; we want the "Mauna Loa CO_2 monthly mean data," the Law Dome data (scroll down to near the bottom of the page), and the "global-mean monthly, seasonal, and annual means, 1880-present, updated through most recent month" of the "Land-Ocean Temperature Index, LOTI."

Executing a sample R script, `lab1_sample.R`

Put the file `lab1_sample.R` in an empty directory. Open this file in RStudio and look at the contents. This script carries out the following tasks:

- Clears the workspace by deleting any variables already in R and closing any open figures
- Downloads data files describing atmospheric carbon dioxide concentrations and surface air temperature anomalies
- Reads the data from these files into R
- Makes plots of these time series

With `lab1_sample.R` open in RStudio, choose `Session > Set Working Directory > To Source File Location` in the menu bar near the top of the screen. This step ensures that R knows where to put the files

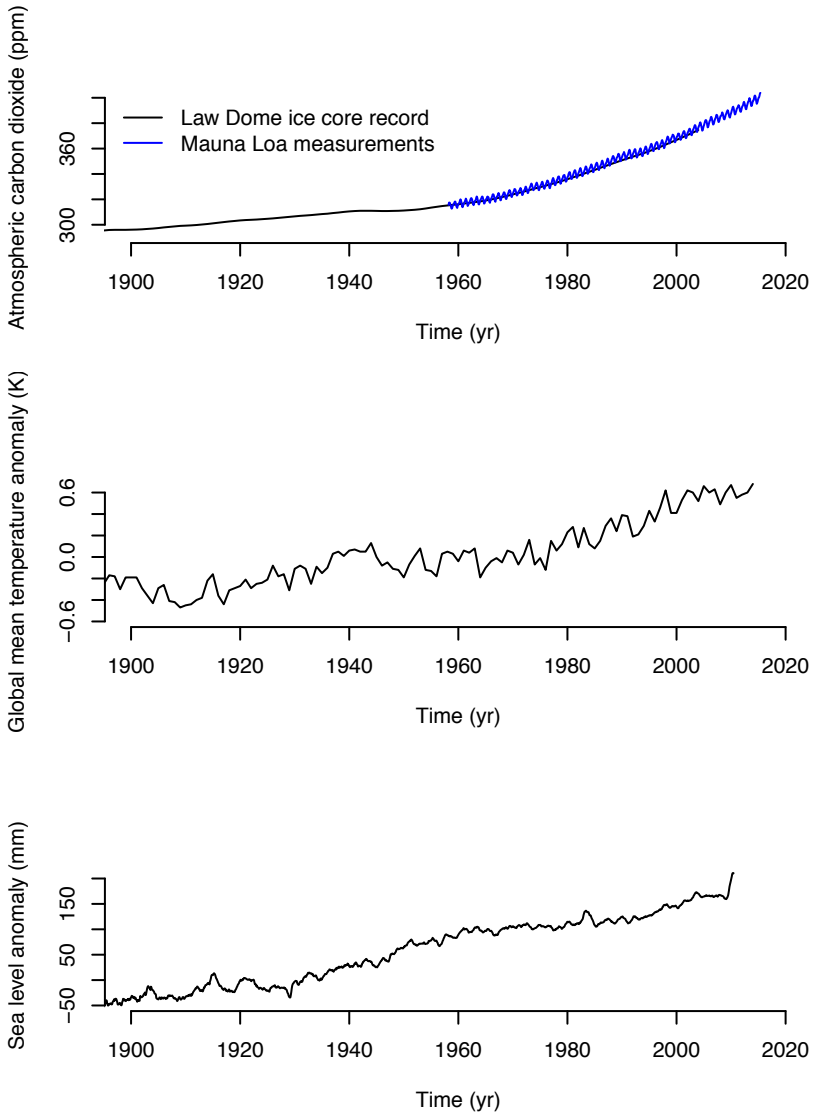

Figure 1: Atmospheric carbon dioxide concentrations (top panel), surface air temperature change (middle panel), and sea level change (bottom panel), between 1900 and ~2015. All three quantities rise over this period, suggesting a causal relationship between them. See text for discussion.

that will be downloaded when we run the script.

Try executing `lab1_sample.R` by clicking on the `Source` button in the upper right-hand corner of the script window in RStudio. You should see two new folders in the directory where `lab1_sample.R` is. One of these new folders, `data`, should contain some text files; the other, `figures`, will contain a `.pdf` file (Fig. 2).

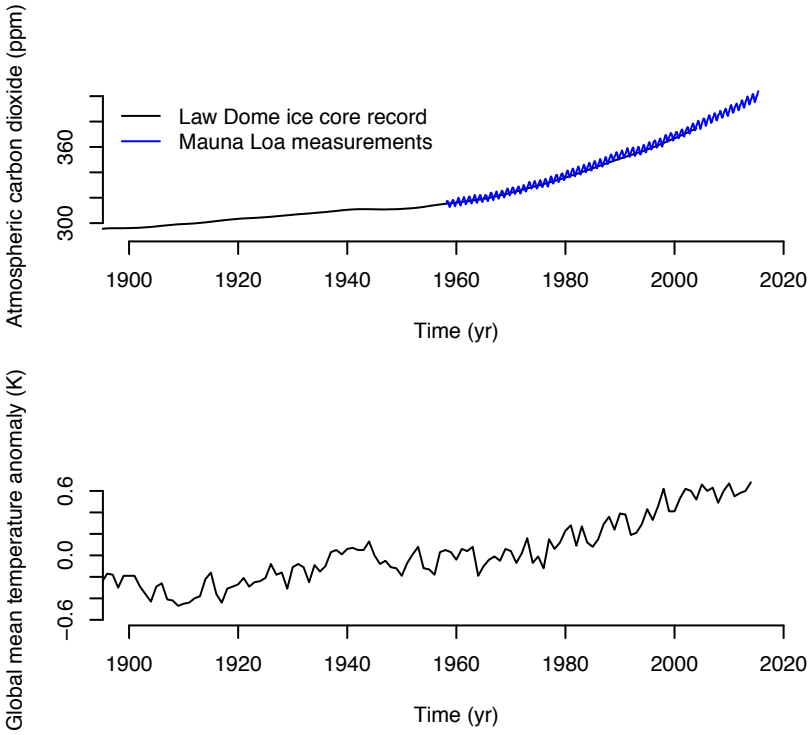

Figure 2: Contents of `data/lab1_sample_plot1.pdf` after running `lab1_sample.R`. See text for discussion.

Looking in detail at `lab1_sample.R`

So, now we know what `lab1_sample.R` does: it makes two directories and puts some files in each of them. In particular, the `figures` directory contains a `.pdf` file with a figure. But, how does it work, and where is the third panel of Figure 1? In this section, we'll go through `lab1_sample.R` in detail. In the Exercise, you'll modify `lab1_sample.R` so that the figure it produces contains all three panels of Figure 1.

The top few lines of `lab1_sample.R` are comments:

```
# lab1_sample.R
# Patrick Applegate, patrick.applegate@psu.edu
#
# Downloads data on atmospheric carbon dioxide concentration and global mean
# temperature and makes plots of these quantities.
```

It's a good idea to begin your R scripts with comments like this one.

The next few lines read

```
# Clear away any existing variables or figures.
rm(list = ls())
graphics.off()
```

These lines clear any existing variables from memory and close any figures that you may have open in the Figure window in RStudio. Including these lines at the beginnings of your R scripts can help you avoid mistakes.

The next block of lines makes a new directory called `data` and downloads some files into it.

```
dir.create('data')
download.file('ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt',
              'data/co2_mm_mlo.txt', method = 'curl')
download.file('ftp://ftp.ncdc.noaa.gov/pub/data/paleo/icecore/antarctica/law/law2006.txt',
              'data/law2006.txt', method = 'curl')
download.file('http://data.giss.nasa.gov/gistemp/tabledata_v3/GLB.Ts+dSST.txt',
              'data/GLB.Ts+dSST.txt', method = 'curl')
```

The command `dir.create('data')` creates a directory with a particular name, in this case `data`, in the present working directory. Note that the directory name needs to be enclosed in quotation marks, `' '`.

For details on the `download.file()` command, type `help(download.file)` in the Console window and press Enter. The help file gives a short description of what the command does, then gives its syntax (under "Usage"), and then a description of the arguments. In this case, we see

```
download.file(url, destfile, method, quiet = FALSE, mode = "w", cacheOK = TRUE,
              extra = getOption("download.file.extra"))
```

Only the first few "words" in the parentheses (`url`, `destfile`, and `method`) will matter in this exercise. These words are arguments of the `download.file()` function. Look at the explanations of what these arguments mean in the help file. Now look back at the code block above. Can you see why the code looks the way it does?

In the line

```
download.file('ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt',
              'data/co2_mm_mlo.txt', method = 'curl')
```

the first group of characters in quotation marks is the Internet location of the file we want to download (the `url` argument for `download.file()`), and the second group of characters in quotation marks is where we want to put the downloaded file (`destfile`). The `method` argument should be set to `curl` on Macintosh machines; other types of computers may need different values. Check the `download.file()` help file if you run into problems.

Now that we have the files, we need to read the information that they contain into R using the `read.table()` command. Open each of the data files using TextWrangler or a similar text editor and examine them. Which files contain "extra" information at the top or bottom of the file? In each file, which columns contain the information we need?

In `lab1_sample.R`, the following lines read in the atmospheric CO_2 concentration data from the Mauna Loa and Law Dome records.

```
# Read in the information from the downloaded files.
loa.co2.data <- read.table('data/co2_mm_mlo.txt', skip = 57, header = FALSE)
law.co2.data <- read.table('data/law2006.txt', skip = 183, nrows = 2004,
                           header = FALSE)
```

In each of the `read.table()` commands, the first argument is the file to open. Look at the help file for `read.table()`. What do the `skip` and `nrows` arguments do? Look back at the corresponding files in your text editor. Can you see why these arguments have the values that they do?

The temperature data file is formatted in blocks, each of which contains about 20 yr of data. For that reason, it is much more complicated to read into R. The code block below accomplishes this task. We won't discuss how this code block works here; it requires the use of `for` loops, which we'll cover in a future lesson.

```
# Reading in the GISS temperature data is somewhat harder.
begin.rows <- c(9, 31, 53, 75, 97, 119, 141)
num.rows <- c(19, 20, 20, 20, 20, 20, 14)
temp.data <- matrix(NA, nrow = sum(num.rows), ncol = 20)
temp.data[1: num.rows[1], ] <- as.matrix(read.table('data/GLB.Ts+dSST.txt',
                    skip = begin.rows[1], nrows = num.rows[1], header = FALSE))
for (i in 2: length(begin.rows)) {
  temp.data[(sum(num.rows[1: i- 1])+ 1): sum(num.rows[1: i]), ] <-
    as.matrix(read.table('data/GLB.Ts+dSST.txt', skip = begin.rows[i],
                         nrows = num.rows[i], header = FALSE))
}
```

Finally, we make a figure and send it to a .pdf file.

```
# Make a plot of the results.
dir.create('figures')
pdf('figures/lab1_sample_plot1.pdf', width = 4.5, height = 4)
par(mfrow = c(2, 1), cex = 0.66)
plot(law.co2.data[, 1], law.co2.data[, 6], type = 'l', xlim = c(1900, 2020),
     ylim = c(290, 400), bty = 'n', xlab = 'Time (yr)',
     ylab = 'Atmospheric carbon dioxide (ppm)')
lines(loa.co2.data[, 3], loa.co2.data[, 5], type = 'l', col = 'blue')
legend('topleft', c('Law Dome ice core record', 'Mauna Loa measurements'),
       col = c('black', 'blue'), lwd = 1, bty = 'n')
plot(temp.data[, 1], temp.data[, 14]/ 100, type = 'l', xlim = c(1900, 2020),
     ylim = c(-0.6, 0.7), bty = 'n', xlab = 'Time (yr)',
     ylab = 'Global mean temperature anomaly (K)')
dev.off()
```

As before, `dir.create()` makes a directory, this time called `figures`. The `pdf()` and `dev.off()` commands open a new `.pdf` file and close this file once we are done writing to it. The `width` and `height` arguments of the `pdf()` function determine the size of the resulting `.pdf` file. If you forget to include `dev.off()` at the end of your plotting commands, your `.pdf` file may not open properly.

The `plot()` command is the main plotting command in R. Now that you've run `lab1_sample.R`, try pasting the following line of R code into the Console window in RStudio and press Enter:

```
plot(law.co2.data[, 1], law.co2.data[, 6], type = 'l', xlim = c(1900, 2020),
     ylim = c(290, 400), bty = 'n', xlab = 'Time (yr)',
     ylab = 'Atmospheric carbon dioxide (ppm)')
```

A figure should appear in the Plots window in RStudio. The curve in this plot represents the Law Dome CO_2 record. The first two arguments represent the independent variable (in this case, time, as contained in the first column of `law.co2.data`), and the dependent variable (CO_2 concentrations, as contained in the sixth column of `law.co2.data`).

To see why the figure looks the way it does, look at `help(plot)` and `help(par)`. What do the other arguments in the call to `plot()`, above, do? How does the plot change if you adjust the values of those arguments?

If you get an error instead of a figure, you may not have run `lab1_sample.R` properly. Make sure that

1. `lab1_sample.R` is open and visible in RStudio's script window
2. you have clicked `Session > Set Working Directory > To Source File Location` in the menu bar near the top of the screen
3. you have clicked the Source button in the upper right-hand corner of the script window

Good scientific graphics help the reader to compare different pieces of data (Tufte, 2001). To achieve this goal, we often need to make a figure with two or more panels, or to draw two data series on the same set of axes. A quick summary of how to carry out these tasks might look like this:

- **Make a plot with two or more panels:** Before issuing your first `plot()` command, use the `par()` command with arguments `mfcol` or `mfrow` to determine how many panels the figure will have.

- **Plot two data series on the same set of axes:** Issue the `plot()` command to plot the first data series, then use the `lines()` or `points()` commands to plot subsequent data series. Note that the data series that is displayed first will be under the second, and so forth.

`lab1_sample.R` applies both of these methods. In the code block below, we first issue the `par(mfrow = c(2, 1))` command to tell R that we want to make a plot with two panels, one on top of the other. We then issue the `plot()` command to display the Law Dome CO_2 record in the top panel. The subsequent call to `lines()` displays the Mauna Loa CO_2 measurements in the same panel as the Law Dome record.

```
par(mfrow = c(2, 1), cex = 0.66)
plot(law.co2.data[, 1], law.co2.data[, 6], type = 'l', xlim = c(1900, 2020),
     ylim = c(290, 400), bty = 'n', xlab = 'Time (yr)',
     ylab = 'Atmospheric carbon dioxide (ppm)')
lines(loa.co2.data[, 3], loa.co2.data[, 5], type = 'l', col = 'blue')
```

Issuing a second `plot()` commands would cause its output to appear in the lower panel. If we then used a third `plot()` command, the figure we were making would be erased, and the output of the third `plot` command would appear in the top panel of the new figure.

Finally, we add a legend to the top panel of our figure with the following line of code.

```
legend('topleft', c('Law Dome ice core record', 'Mauna Loa measurements'),
       col = c('black', 'blue'), lwd = 1, bty = 'n')
```

`help(legend)` explains how this command works. Scroll down to `Details` and note that the first argument `x` can be set to any of various words that tell R where to put the legend.

Exercise

Save `lab1_sample.R` to a new file. Now, using what you've learned from following the detailed discussion of `lab1_sample.R` above, modify the new file so that it produces a `.pdf` file like Figure 1, with three panels. The bottom panel should show the sea level anomalies from Jevrejeva et al. (2014). Specifically, you'll need to add commands to your new file that

1. download the sea level anomaly data from Jevrejeva et al. (2014) to your `data` folder (have a look at Table 1)
2. read in this data to a new variable (you might name your new variable `sl.data`)
3. causes the figure to have three panels instead of two
4. adjusts the size of the resulting `.pdf` file
5. plots the sea level anomaly data in the bottom panel

Make sure that the lower panel of your plot has sensible axis limits in the x and y directions, as well as descriptive labels for the axes (see Fig. 1 for an example). Also be sure to update the comments at the top of your file.

Questions

1. By how much have atmospheric carbon dioxide concentrations, global mean temperatures, and sea levels changed between 1900 and the early part of the present century? Hint: Check Figure 1 and/or the figure produced by your updated R script.
2. How do the changes in atmospheric carbon dioxide concentrations, global mean temperatures, and sea levels between 1900 and 2015 compare to the changes in the same variables between the last ice age (~20,000 years ago) and preindustrial times (around 1850)? Give your results as the changes from question #1, above, divided by the changes between the last ice age and the preindustrial period. Note that changes in temperature and sea level were small between 1850 and 1900. You'll need to search the Web for values of these quantities during the last ice age and the preindustrial period. If this exercise is for a class, be sure to give the Internet sources you used in this part of the assignment.
3. How do the *rates* of change in atmospheric carbon dioxide concentrations, global mean temperatures, and sea levels between 1900 and 2015 compare to the rates of change in these quantities between the last ice age and preindustrial times? Recall that a rate of change is the change divided by the amount of time over which the change happens.

References

Hansen, J., Ruedy, R., Sato, M., and Lo, K., 2010. Global surface temperature change. Reviews of Geophysics 48, RG4004.

Jevrejeva, S., Moore, J. C., Grinsted, A., Matthews, A., Spada, G., 2014. Trends and acceleration in global and regional sea levels since 1807. Global and Planetary Change 113, 11-22.

Keeling, C. D., Bacastow, R. B., Bainbridge, A. E., Ekdahl, C. A., Guenther, P. R., and Waterman, L. S., 1976. Atmospheric carbon dioxide variations at Mauna Loa Observatory, Hawaii. Tellus 28, 538-551.

MacFarling Meure, C., Etheridge, C., Trudinger, C., Steele, P., Langenfelds, R., van Ommen, T., Smith, A., and Elkins, J., 2006. The Law Dome CO2, CH4 and N2O ice core records extended to 2000 years BP. Geophysical Research Letters 33, L14810.

Nicholls, R. J., Tol, R. S. J., and Vafeidis, A. T., 2008. Global estimates of the impact of a collapse of the West Antarctic ice sheet: an application of FUND. Climatic Change 91, 171-191.

Tufte, E. R., 2001. The Visual Display of Quantitative Information. Graphics Press, 197.

Spanger-Siegfried, E., Fitzpatrick, M., and Dahl, K., 2014. Encroaching tides: how sea level rise and tidal flooding threaten U.S. East and Gulf Coast communities over the next 30 years. Union of Concerned Scientists, 66.

Exercise #2: Normal distributions and the Galton board

Patrick Applegate, patrick.applegate@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- describe under what circumstances normal distributions arise
- use quantile-quantile plots to determine whether individual data sets are normal or not
- install and load R packages from CRAN, the Comprehensive R Archive Network
- write simple `for` loops in R

Introduction

Suppose that we ask a class of students to walk individually from one end of a field to the other and count the number of steps that they take in order to cover the distance. More than likely, each student will get a somewhat different answer. After each student has paced off the field, we then make a histogram of the number of steps taken by the individual students.

If we were to perform this experiment with a large number of students, the histogram would likely resemble a normal distribution. Normal distributions arise when a large number of measurements are taken of a single quantity, and the individual measurements are affected by random processes that are additive and that can be either positive or negative. In our example, the random processes have to do with the varying stride lengths of the students (some take longer strides and some shorter), and counting errors (which are likely to be small and can cause any given student to either over- or underestimate the actual number of paces they took).

A Galton board is a physical device that demonstrates this concept (Fig. 1). A Galton board has rows of pins arranged in a triangular shape, with each row of pins offset horizontally relative to the rows above and below it. The top row has one pin, the second row has two pins, and so forth.

If a ball is dropped into the Galton board, it falls either to the right or the left when it bounces off the top pin. The ball then falls all the way to the bottom of the board, moving slightly to the right or left as it passes through each row of pins. Bins at the bottom of the board capture the ball and record its final position. If this experiment is repeated with many balls, and if the Galton board includes many rows of pins, the number of balls in each bin resembles a normal distribution (Fig. 1).

In this exercise, you'll experiment with a simple representation of a Galton board in R and examine the distributions of final ball positions that it produces. You'll also examine two data sets that are "built in" to R and see whether they are well described by a normal distribution or not.

Tutorial

There is a ready-made, animated representation of the Galton board in R's `animation` package. An R package is a set of functions or commands that have been written by R users and posted to CRAN, the Comprehensive R Archive Network. R packages have to meet some minimum standards of quality in order to be included on CRAN, and the packages are open-source and free to use.

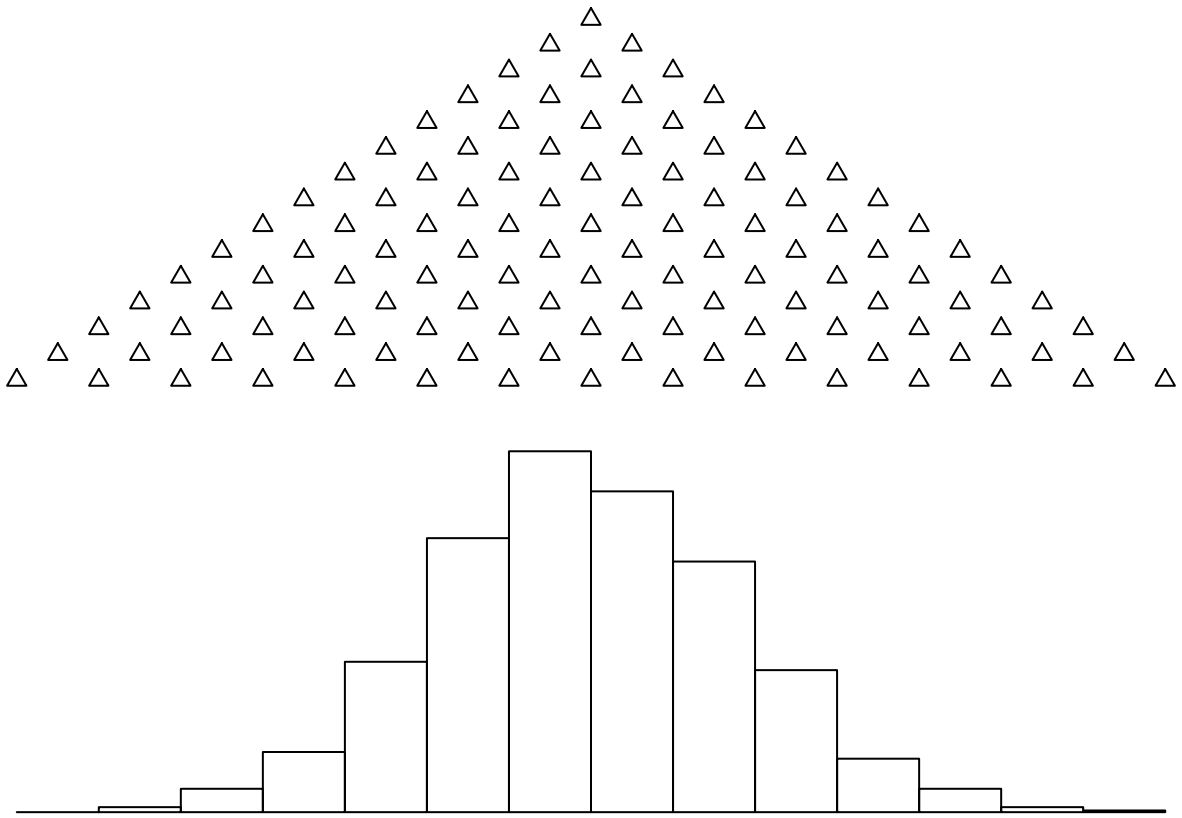

Figure 1: A Galton board with 15 rows of pins (top), and one possible distribution of final ball positions resulting from dropping 1000 balls through the board (bottom). Triangles, pins in the Galton board; bars, heights of balls in each bin.

Installing and loading packages from CRAN in R

Making a particular package available for use in R is a two-step process: first we download the package, and then we load it into R's working environment. To download and load the `animation` package, open RStudio and type the following commands into the Console window (pressing Enter after each command). You can ignore the text after the pound signs (`#`): these text fragments are *comments* that are not processed by R.

```
install.packages('animation') # downloads the animation package from CRAN
require('animation') # loads the animation package
```

Once the package has been loaded, we can experiment with the Galton board animation. Typing `help(quincunx)` in the Console window in RStudio and pressing Enter lets us look at the arguments of the `quincunx()` function. ("Quincunx" is another name for a Galton board.)

```
# from help(quincunx)
quincunx(balls = 200, layers = 15, pch.layers = 2, pch.balls = 19,
    col.balls = sample(colors(), balls, TRUE), cex.balls = 2)
```

Here, the `balls` argument represents the number of balls to drop through the Galton board, and `layers` indicates the number of rows of pins in the board. The numbers after each of these arguments (`balls = 200, layers = 15`) indicate the arguments' default values.

Let's try just one ball at first. Type the following two commands into the Console window in RStudio and press Enter after each one.

```
ani.options(nmax = 14) # sets the number of animation frames
quincunx(balls = 1) # runs the animation
```

You'll see the ball move downward through the rows of pins and land in a bin, where it is represented with a tall bar.

Try entering the second command above into RStudio repeatedly. With your cursor at the prompt in the Console window, press the Up arrow and then Enter to reevaluate the last command you entered. How does the path of the ball differ among evaluations of the `quincunx()` function? Which bins receive balls most often?

Once you've examined the single-ball case, let's move on to a case with many balls. Enter the following commands into RStudio's Console window, pressing Enter after each command.

```
n.balls <- 200 # sets the number of balls to drop
n.rows <- 15 # sets the number of rows of pins in the Galton board
ani.options(nmax = n.balls+ n.rows- 2)
quincunx(balls = n.balls, layers = n.rows)
```

This example will take several minutes to run. If you get tired of watching the animation, you can stop RStudio by holding down Control and pressing C on your keyboard, or by clicking on the little STOP sign that appears in the upper right-hand corner of the Console window while R is working.

This time, we see many balls of different colors fall from the top of the Galton board to the bottom. The heights of the bars in the different bins grow as the balls reach the bottom of the board. How well does the final histogram agree with Figure 1?

Writing our own simple Galton board script

Building computer models involves thinking about how to represent the behavior of physical (or chemical, or social) systems in a form that can be understood by a computer. In thinking about how we could represent the Galton board in terms of R code, we might start by looking at the diagram of the Galton board in the top part of Figure 1. The horizontal spacing between the pins is always the same, but the rows are offset relative to one another; that is, none of the pins in any given row are immediately above or below the pins in the adjacent rows. This offset is exactly half of the horizontal spacing between pins in the same row. Thus, we can imagine that the ball bounces 0.5 distance units to one side whenever it hits a pin.

We can then represent the path that a single ball takes through the Galton board by

```
path <- sample(x = c(-0.5, 0.5), size = (n.rows- 1), replace = TRUE)
print(path)
```

```
##  [1] -0.5 -0.5  0.5  0.5 -0.5  0.5  0.5  0.5  0.5 -0.5 -0.5 -0.5  0.5 -0.5
```

Positive values indicates that the ball bounces to the right; negative values indicate that the ball bounces to the left. So, in this example, the ball bounces twice to the left (-0.5, -0.5), twice to the right (0.5, 0.5), and so on.

The `sample()` command generates sequences of values that come from a vector of possibilities supplied to the argument `x`. Have a look at `help(sample)` to see what the other arguments mean. We set the `size` argument set to `(n.rows- 1)`, rather than just `n.rows`, for consistency with the `quincunx()` function, which yields a number of bins equal to `n.rows -1`. To check this statement, count the number of bins in the lower part of Figure 1.

The bin that the ball lands in is then just

```
bin <- sum(path)
print(bin)
```

```
## [1] 0
```

In this case, the ball lands immediately below the top pin of the Galton board, in the middle bin (0).

Doing things over and over again: `for` loops

The commands above tell us about the path that any single ball takes through the Galton board, as well as the bin it finally lands in. But, suppose we want to write a piece of R code that performs this calculation for many balls and gives us a histogram of the bins that they land in. How could we do that?

Most, perhaps all, programming languages include a method for carrying out a set of instructions a fixed number of times. In R, this method is called a `for` loop. The syntax of `for` loops looks like this:

```
n.times <- 3 # not part of the for loop syntax
for (i in 1: n.times) { # note the open curly brace!
  print(i) # instructions to be repeated go here
} # note the close curly brace!
```

```
## [1] 1
## [1] 2
## [1] 3
```

This sample `for` loop is not very exciting: it simply `print()`s the contents of the *index variable* `i` during each iteration of the commands within the curly braces (`{}`).

To see why this sample `for` loop `print()`s the values that it does, we can examine the vector of values taken by the index variable `i`:

```
1: n.times # equivalent to seq(1, n.times, by = 1)
```

```
## [1] 1 2 3
```

So, this vector contains the values 1, 2, 3. As the `for` loop runs through these values, it `print()`s them to the screen successively.

In "real" `for` loops, the name of the index variable, the vector of values that the index variable can take (here `1: n.times`), and the instructions in the curly braces can all change. In particular, there can be more than one line of instructions between the curly braces.

In general, we will want to create a variable that will store the results from the commands carried out within the `for` loop. The following block of code generates and `print()`s the first five Fibonacci numbers:

```
n.times <- 5 # not part of the for loop syntax
output <- rep(1, n.times) # makes a vector of five 1's
for (i in 3: n.times) {
  # stores the sum of the preceding two elements of output in the ith element
  # of output
  output[i] <- sum(output[(i- 2): (i- 1)])
}
print(output)
```

```
## [1] 1 1 2 3 5
```

Here, `output` is a vector with `n.times` elements that receives the results of the calculations performed within the `for` loop.

In the Exercise, below, you'll put together the information above to write your own script for making histograms of the outcomes from a Galton board. First, though, let's examine how we determine whether a particular data set is normal or not.

Normal or not normal?: Generating and interpreting quantile-quantile plots

It can be tempting to assume that a particular process gives normally-distributed output, or that data are normally distributed; however, assuming normality when it isn't present can lead to large errors. How can we check individual data sets for normality?

One robust and simple, but approximate, test for normality involves making a quantile-quantile (Q-Q) plot. The details of how Q-Q plots work aren't important for our purposes here. However, if data are drawn from a normal distribution, they will lie on a line connecting the first and third quartiles of the data when drawn on an appropriately-constructed Q-Q plot (see `help(qqline)`). R makes drawing this type of Q-Q plot easy.

For example, the following block of code generates 100 random numbers from a normal distribution, makes a Q-Q plot from them, and draws a 1:1 line on the plot:

```
norm.vals <- rnorm(100, mean = 5, sd = 3) # generates normally-distributed values
qqnorm(norm.vals) # makes the q-q plot
qqline(norm.vals) # adds a line; do the points lie on this line?
```

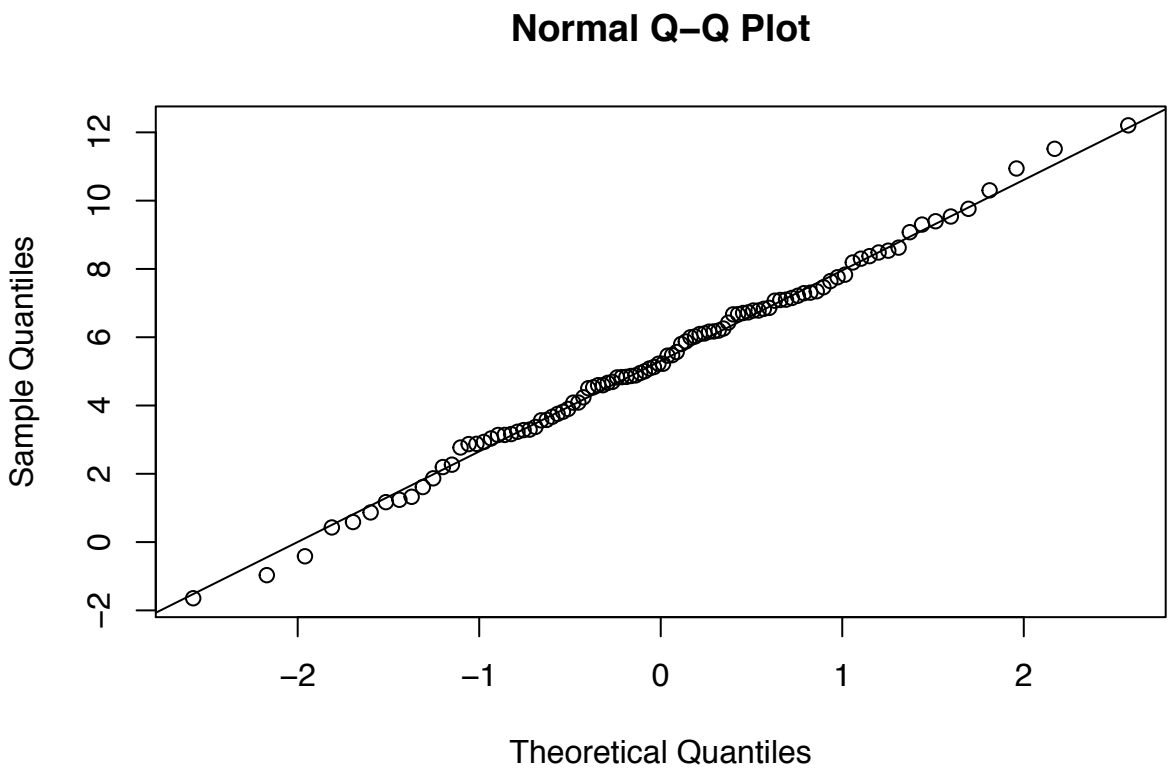

Figure 2: A Q-Q plot showing 100 normally-distributed random values. The points, which represent the random values, lie approximately on the line, confirming that they come from a normal distribution.

Exercise

Part 1. Building on your R script from Exercise 1 and the information presented in this document, write an R script that

1. has comments at the top that explain what the script does and who wrote it
2. clears existing variables from memory and closes open figures
3. sets values for the number of balls to drop and the number of rows of pins in the Galton board (you might use the variable names `n.balls` and `n.rows`)
4. creates a vector `output` that contains a number of elements equal to `n.balls`; these elements should be populated with `NA`s to start (hint: see `help(rep)`)
5. uses a `for` loop that

- runs through values of the index variable `i` from `1: n.balls`,
- determines which bin of the Galton board each ball lands in, and
- stores these values in the different elements of `output`

6. makes a histogram of `output`. The code block below shows one way of carrying out this step.

```
# Make a histogram of the results.
bin_edges <- seq(-n_rows/ 2, n_rows/ 2, by = 1)
hist(output, bin_edges, xlab = 'Bin', ylab = 'Frequency')
```

Part 2. Add commands to your R script so that it also generates a Q-Q plot, with a line, for the results in `output`.

Part 3. Generate Q-Q plots for two built-in data sets in R,

1. the waiting time between Old Faithful eruptions (`faithful[, 2]`), and
2. the sepal length of setosa irises on the Gaspe Peninsula (`iris3[, 1, 1]`).

You can learn about the sources of these data with `help(faithful)` and `help(iris3)`.

Questions

1. Set the number of rows of pins in your Galton board R script to 10. Also set the number of balls to drop to a low value, say 10 to begin.

- How does the histogram change as you repeatedly `source()` the script?
- What is the smallest number of balls that you can drop that gives more or less the same histogram each time you run the script, for 10 rows of pins?

2. Recall from the discussion above that a Galton board only gives an approximately-normal distribution if both the number of balls that are dropped and the number of bins are large. Leaving the number of rows of pins at 10, increase the number of balls to a large value like 10^4 (in R code, that would be just `10^4`). Confirm that the `output` under these circumstances is not normally distributed. How large does the number of rows of pins have to be to get approximately-normal results?
3. Which of the built-in data sets listed above is approximately normally distributed, and which is not? How can you tell?

Exercise #3: Other probability distributions and random sampling

Patrick Applegate, patrick.applegate@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- generate random numbers from various probability distributions in R
- construct histograms and empirical cumulative distribution functions from sets of random numbers
- describe how changing the parameters of various probability distributions affects the shape of the distributions

Introduction

Risk analysis often involves simulating processes that have stochastic (that is, random) components. Many of the random elements within computer models can be represented by normal distributions; however, as we saw in Part 2 of the last exercise, not all data sets are normally distributed. We need a set of methods for simulating random effects within numerical models, while accounting for the possibility that these random effects may not obey normal distributions.

But, what is a probability distribution? Loosely speaking, a probability distribution function (pdf) is a mathematical relationship that ties values of a quantity x to the chance of observing particular values of that quantity. These chances (probabilities) range from 0, indicating that the associated value of x is impossible, to 1, indicating that the associated value of x must occur on any given trial. Assuming that *something* occurs on any given trial, integrating these probabilities over all possible values of x must yield 1. Probability distributions have parameters that change the distribution's shape, sometimes drastically.

In this exercise, you'll perform random sampling from several probability distributions and examine two ways of plotting the results from this random sampling. You'll also see how changing the parameters of statistical distributions change the distributions' shape.

Tutorial

In RStudio, open the file `lab3_sample.R` and change R's working directory to the location of that file on your hard drive. Next, execute the script (look back at Exercise 1 if you need help with changing R's working directory or with executing the script). A directory called `figures` should appear that contains a file called `lab3_sample_plot1.pdf`. Open this file and look at the contents. This file should contain a pair of plots that resembles Figure 1.

Histograms and empirical cumulative distribution functions

In Figure 1, the top panel shows a histogram, and the lower panel shows an empirical cumulative distribution function (ecdf). These plots both display the same set of randomly-generated synthetic data, in this case $n = 10^4$ random draws from a normal distribution with mean μ of 5 and a standard deviation σ of 3.

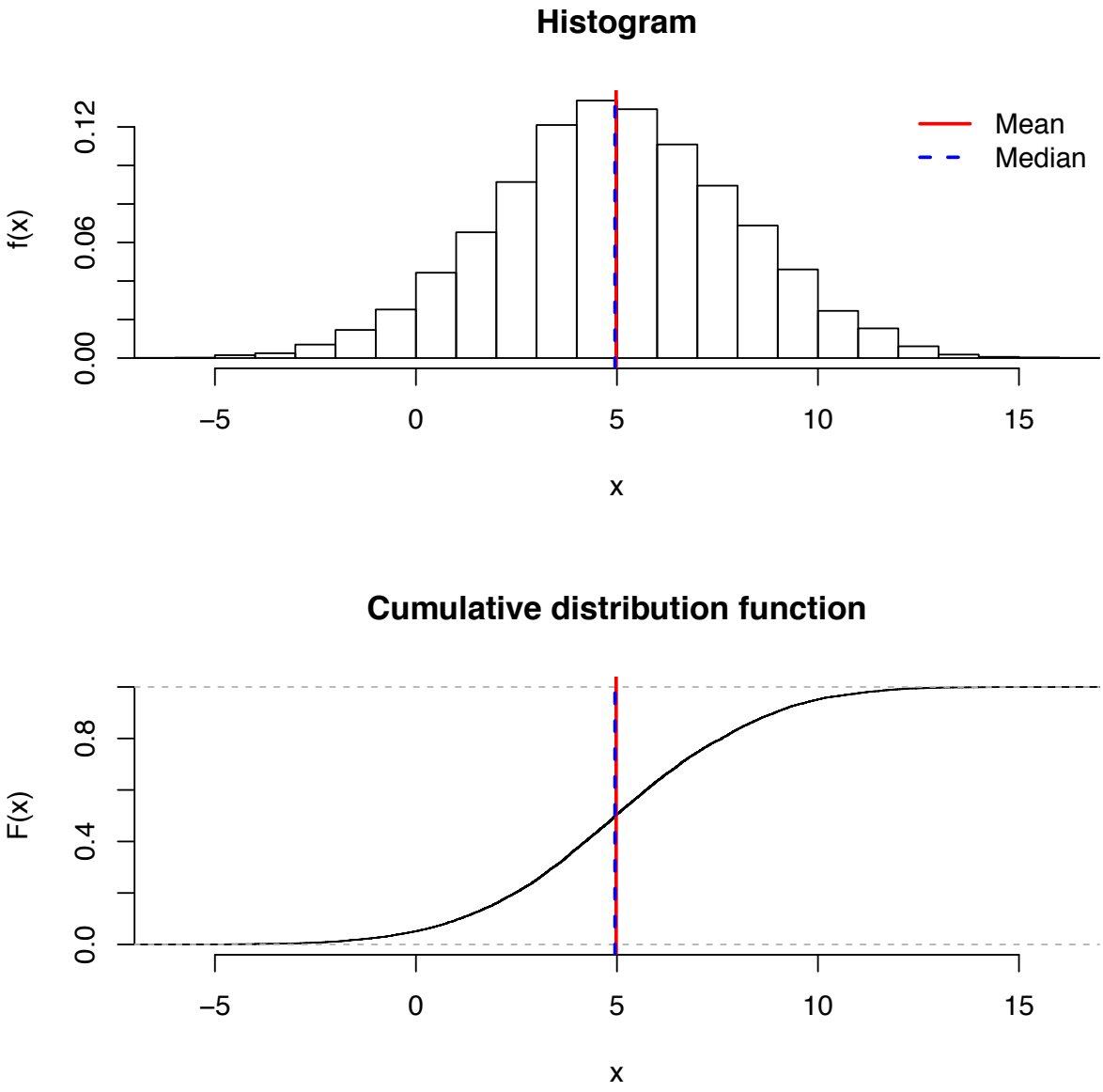

Figure 1: Contents of `figures/lab3_sample_plot1.pdf` after running `lab3_sample.R`. See text for discussion.

In a histogram, a group of observations is sorted into bins that usually have evenly-spaced edges, and the plot is then drawn as a series of rectangles with heights that are proportional to the number of observations in each bin. In general, then, bins with taller rectangles contain more observations than those with shorter rectangles.

To construct an empirical cumulative distribution function, the data are first sorted so that the data are in order from the smallest value to the largest. The plot is drawn as a series of "stairsteps" that connect $y = 0$ on the side of the plot corresponding to small values of x, to $y = 1$ on the right-hand side of the plot. In between, the value of the quantity plotted on the y-axis increases by $1/n$ at each x-value corresponding to one of the observations. On Figure 1, these "stairsteps" are too small to see individually, because each one corresponds to a change in y of $\Delta y = 1/n = 10^{-4}$. As a consequence of how they are constructed, empirical cumulative distribution functions always increase from small values of x to larger ones; that is, they are monotonically increasing.

The quantity plotted on the y-axis differs between histograms and empirical cumulative distribution functions. On a histogram, the y-axis represents the number of observations in each bin or (as in Figure 1) this number divided by the total number of observations. On an empirical cumulative distribution function, the y-axis represents the running total of the observations, going from small values of x to large ones, again divided by the total number of observations.

Measures of central tendency: the mean vs. the median

On Figure 1, the colored, vertical lines indicate the mean (specifically, the arithmetic mean) and median of the data. These quantities are both measures of the central tendency of a data set. The arithmetic mean is just the average of a group of numbers; the median represents the value that separates a sorted data set into two groups that contain the same number of observations.

In Figure 1, the lines representing the mean and the median lie on top of one another, indicating that these metrics are very close to each other. This result is expected for normally-distributed data sets that contain many observations, like the one shown in Figure 1. However, this result is often *not* true for data sets containing only a few values, or for those that are drawn from distributions that are not symmetrical.

Picking apart `lab3_sample.R`

Let's examine the R code in `lab3_sample.R`. The top few lines should look familiar; they tell us what the code does and delete any existing variables and figures from the workspace.

```
# lab3_sample.R
# Patrick Applegate, patrick.applegate@psu.edu
#
# Produces (pseudo-)random numbers from a normal distribution and plots a
# histogram and empirical cumulative density function using these numbers.

# Clear away any existing variables or figures.
rm(list = ls())
graphics.off()
```

The next block of code sets some values that we will use further down in the script. We set these values here so that they will be easy to find and change in the future. Also, it's good programming practice to avoid *magic numbers* (in this case, unnamed numerical constants). Note the comments that explain what each value represents.

```
# Set some values.
mu <- 5          # mean of normal distribution
sigma <- 3       # standard deviation of normal distribution
n.obs <- 10^ 4   # number of random values to generate
n.bins <- 25     # number of bins in the histogram
```

Next, we set the *seed* for random number generation. This command ensures that the code will give the same answers each time it is run, which is helpful for debugging. In general, any piece of code that depends on random number generation should include this command (though the value in parentheses can be changed).

```
# Set the seed for random sampling.
set.seed(1)
```

The following code block generates the random values. Look carefully at `help(rnorm)` to see what the arguments of the `rnorm()` command do. How do these arguments relate to the code block that starts, "`# Set some values`," above?

```
# Generate the random values.
data <- rnorm(n = n.obs, mean = mu, sd = sigma)
```

Next, we calculate the mean and the median of the values in `data`.

```
# Extract some relevant quantities from the random values.
mean.data <- mean(data)
median.data <- median(data)
```

The last, somewhat long, code block creates a `.pdf` file in a new directory called `figures` and plots the values in `data` in this file. The new commands in this code block include

- `hist` (plots histograms)
- `plot.ecdf` (plots empirical cumulative distribution functions)
- `abline` (overlays straight lines on an already-existing plot)
- `legend` (puts a legend on an already-existing plot)

```
# Make a plot.
dir.create('figures')
pdf('figures/lab3_sample_plot1.pdf')
par(mfrow = c(2, 1))

# Plot the histogram.
hist(data, breaks = n.bins, freq = FALSE, xaxs = 'i', main = 'Histogram',
     xlab = 'x', ylab = 'f(x)')

# Show the mean and the median on the histogram.
abline(v = mean.data, lty = 1, lwd = 2, col = 'red')
abline(v = median.data, lty = 2, lwd = 2, col = 'blue')

# Put a legend on the histogram.
legend('topright', legend = c('mean', 'median', '0.025 and 0.975'),
       lty = c(1, 2), lwd = 2, bty = 'n', col = c('red', 'blue'))
```

```
# Find the extents of the histogram's axes.
axis.lims <- par('usr')

# Now, plot the empirical cumulative distribution function.
plot.ecdf(data, xlim = axis.lims[1: 2], bty = 'n', xaxs = 'i',
          main = 'Cumulative distribution function', xlab = 'x', ylab = 'F(x)')

# Plot the mean and the median on the ecdf.
abline(v = mean.data, lty = 1, lwd = 2, col = 'red')
abline(v = median.data, lty = 2, lwd = 2, col = 'blue')
dev.off()
```

To see what the arguments of the `hist()` and `plot.ecdf()` commands mean, look at `help(hist)`, `help(plot.ecdf)`, and `help(par)`. Note that the `help(par)` page in particular is very long; it can be helpful to search within the help page for the names of particular arguments. To search within a help page, click on the Help window in RStudio, and (on a Mac) press and hold Command and then press F.

Exercise

Part 1. Adjust one of the variables near the top of `lab3_sample.R` (immediately after "`# Set some values`") and rerun the code. Repeat this process for each of these variables. How do each of these variables affect the figure that appears in `figures/lab3_sample_plot1.pdf`? You might want to change the file name in the `pdf()` command so that the new figures don't overwrite the old ones.

Part 2. Next, we'll experiment with other probability distributions. Change `n.obs` and `n.bins` back to their original values (see above). Create a copy of `lab3_sample.R` by saving the file under a different name (click on `File > Save As...` in RStudio).

Change your new script so that it generates random draws from each the following probability distributions. You'll need to make a new copy of the script each time you move to a new distribution.

- (continuous) uniform distribution
- lognormal distribution
- triangular distribution (you'll need the `triangle` package to work with this distribution; check Lesson 2 for a reminder about how to install and load packages)
- exponential distribution

To carry out this task for one of the distributions above, you'll need to

1. find out what the parameters of the new distribution are by reading the appropriate Wikipedia article
2. look at the help page associated with the new distribution in R using, for example, `help.search('uniform distribution')` – in particular, note which command on that help page generates random numbers, and what the arguments of that command are. How do these arguments match up with the parameters of the distribution from the Wikipedia article?
3. change the R script so that it sets the parameters of the new distribution under "`# Set some values`," and populates the vector `data` using the random number generation command for the new distribution.

Questions

1. How does changing the values of each of the variables near the top of `lab3_sample.R` adjust the appearance of the plots? Write a one-sentence description of the effects of each variable on how the plots look.

2. Make a table with the name of each distribution, the parameters of the distributions, and a short description of the plots associated with each histogram.
3. Which of the distributions place the mean and the median close to one another, and which have them far apart? Be sure to use a large value of `n.obs`, say `10^4`, for all the distributions when answering this question.

Exercise #4: What is the economically "optimal" height of flood protection structures?: The Van Dantzig (1956) example

Patrick Applegate, patrick.applegate@psu.edu, and Klaus Keller, klaus@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- describe the Van Dantzig (1956) approach to estimating the optimal height of flood protection structures (dikes)
- explain how discounting works and describe the effects of discounting rates on the results of cost-benefit analyses
- perform a simple Monte Carlo analysis based on Van Dantzig (1956)

Introduction

Suppose that we are aware of a potential future danger. We don't know the magnitude of this danger in advance, but we do know the risks it poses. That is, we know the probabilities and economic damages associated with different levels of danger. We also know the costs of putting protection in place. These protective measures are imperfect, and we may want to avoid spending more money on protection than is necessary. How do we decide how much protection to institute?

Van Dantzig (1956) considered this problem in the context of avoiding damaging floods from storm surges in the Netherlands. The Netherlands is a densely-populated and low-lying country in Europe. Storm surges occur when low-pressure regions associated with large storms cause local sea level to rise temporarily. These storm surges can be several meters high. A particularly large and damaging flood occurred in the Netherlands in 1953.

After the flood of 1953, a team of engineers and mathematicians set to work to design a set of dikes (levees) to protect the Netherlands from similar future events (Van Dantzig, 1956). The dikes could not be made tall enough to close out all potential future floods. The floods that had already happened suggested that even bigger ones could occur. As dikes are built higher, they also become wider and take up increasing amounts of valuable real estate. Thus, the cost of building very tall dikes becomes unreasonably large.

Traditional risk analyses, like the one carried out by Van Dantzig (1956), require some criterion for evaluating possible solutions. Van Dantzig (1956) suggested that one way of identifying an economically-optimal dike height is to evaluate the sum of the cost of dike construction and the expected damages due to future floods, for various potential dike heights. The damages are weighted (discounted) according to how far in the future they occur. The dike height that gives the lowest sum of construction costs and discounted flood damages is the economically "optimal" solution, in the Van Dantzig (1956) framework.

The minimum-total-cost criterion makes a number of assumptions, some of which we will discuss below. Despite these limitations, Van Dantzig (1956) presents a valuable example of traditional risk analysis as applied to flood protection, one that still provides a foundation for modern studies (see, for example, Lempert et al., 2012). This exercise follows sections 1-4 of Van Dantzig (1956), which present a simplified version of his full analysis. The appendix (see below) mentions some additional considerations that may lead to different answers.

An important note

Carrying out this exercise will not prepare you to design flood protection structures. The presentation of the material in this exercise has been simplified for pedagogical reasons, and this material should only be applied to "real" problems after additional training. The authors and editors of this e-textbook specifically disclaim any liability for damages associated with the use of the material presented in this exercise.

Dike construction costs

Van Dantzig (1956) argued that the cost of building a dike I is a linear function of height H,

$I(H) = I_0 + kH,$

for small values of H. Strictly speaking, Van Dantzig (1956) analyzed the case in which dikes with an initial height H_0 are increased to a final height H, with the height increase $X = H - H_0$. Here, we assume that there are no existing protective structures, so that $H_0 = 0$ and $H = X$.

Expected damages and the role of discounting

The question is then, how much damage should we expect from floods if we build the dikes to a particular height? Van Dantzig (1956) approached this question by

1. calculating the expected damage in a single year and
2. integrating these damages over all future times,
3. while assigning a reduced importance to floods that happen far in the future.

The **expected damage in a single year** is the losses due to a single flood, multiplied by the chances of a flood in a given year. Van Dantzig (1956) assumed that, in the case of a flood, the value V of all goods (buildings, livestock, and other property) within the area protected by a dike would be lost. This assumption makes sense because, in the Netherlands, the areas protected by individual dikes (polders) fill with water like a bathtub if the dikes are overtopped or fail. Van Dantzig (1956; see also Wemelsfelder, 1939) noted that the annual maximum sea level values from the central Netherlands followed an exponential distribution,

$P(H) = p_0 e^{-\alpha H}$

Recall from Exercise #3 that an exponential distribution assigns the largest probabilities to small values, but has a long "tail" that can extend to large positive values. Multiplying the damage from one flood by the probability of flooding in one year gives

$VP(H) = V p_0 e^{-\alpha H}$

If we were to integrate these expected damages from flooding in a single year over all future times, we would arrive at an infintely large total future damages value, which would imply an unreasonably tall optimal dike height. However, the use of *discounting* in the Van Dantzig (1956) framework causes the integrated future flood damages to tend to a single, finite value.

Discounting **assigns a reduced weight to floods that happen far in the future**, and it is a common feature of cost-benefit analyses. In the context of flood protection, discounting is reasonable because spending money on flood protection now helps us avoid future flood damages. These avoided damages represent benefits, but we value future benefits less than the money we might spend on flood protection now. We define a *discount factor* F_d that assigns a weight to future potential floods depending on how many years t have gone by when they happen,

$F_\mathrm{d} = (1 + \delta)^{-t}$

Putting the last two equations together, Van Dantzig (1956) arrived at an expression for the total future losses L as a function of dike height H. The summation (Σ) in this expression **integrates flood damages over all future times**.

$$L(H) = VP(H) \sum_{t=0}^{\infty} F_d$$

Substituting for $P(H)$ and F_{d} and simplifying, Van Dantzig (1956) obtained

$$L(H) \approx (Vp_0 e^{-\alpha H})/\delta$$

How could Van Dantzig (1956) perform this simplification? Performing the summation $\sum_{t=0}^{\infty}(1+\delta)^{-t}$, we see that it tends to $(1+\delta)/\delta$. For small values of δ, this factor is close to $1/\delta$. The code block below generates a figure (Fig. 1) that demonstrates this point.

```
# Demonstration that the cumulative sum of the discount factor tends to
# (1+ delta)/ delta as t becomes large, and that this value is close to
# 1/ delta for small values of delta.
delta <- 0.04              # discounting rate
t <- seq(0, 200, by = 1)   # vector of time values (years)
F.d <- (1+ delta)^ -t      # discount factor as a function of time
plot(t, cumsum(F.d), type = 'l', bty = 'n', lwd = 2, xlab = 'Time (yr)',
     ylab = 'Running total, discount factor', col = 'blue')
abline(h = (1+ delta)/ delta, lty = 2, lwd = 2, col = 'red')
abline(h = 1/ delta, lty = 2, lwd = 2, col = 'gray')
```

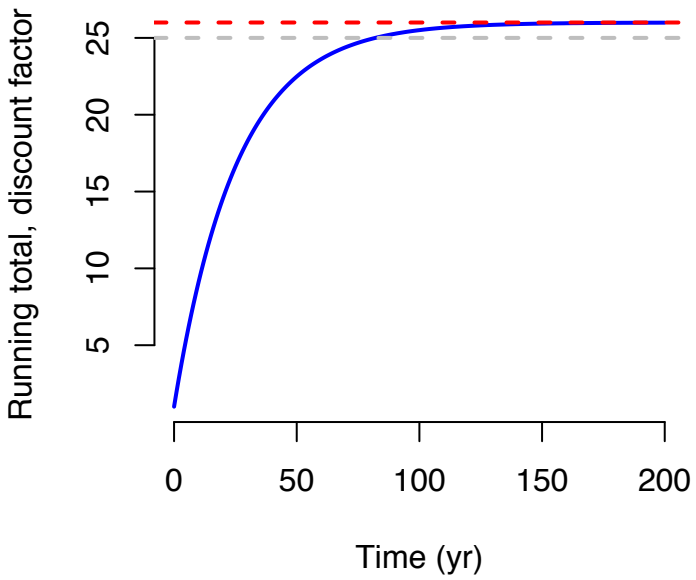

Figure 1: Cumulative sum of the discount factor as a function of time, for a discounting rate $\delta = 0.04$. This cumulative sum tends toward $(1+\delta)/\delta$ (red, dashed line) as t becomes large. This value is close to $1/\delta$ for small values of δ (gray, dashed line).

Finding the optimal dike height

Recall that Van Dantzig (1956) suggested that the height of the dikes should be increased until the sum of their construction cost and the damages expected from future floods reaches a minimum. We now have

an equation that describes the cost of building dikes as a function of their height, $I(H) = I_0 + kH$, and another equation that describes the expected future damages due to floods, also as a function of dike height, $L(H) \approx (V p_0 e^{-\alpha H})/\delta$.

The following code block plots $I(H)$, $L(H)$, and their sum $I(H) + L(H)$, and finds the minimum on the total curve. This point corresponds to the optimal dike height, in the case where this analysis' assumptions are satisfied.

```
# Constants from Van Dantzig (1956) -- read the whole paper carefully to see
# where these values come from
p_0 = 0.0038    # unitless; probability of flooding in a given year if the dikes
                # aren't built
alpha = 2.6     # unitless; constant associated with flooding probabilities
V = 10^ 10      # guilders; value of goods threatened by flooding
delta = 0.04    # unitless; discount rate
I_0 = 0         # guilders; initial cost of building dikes
k = 42* 10^ 6   # guilders/m; cost of raising the dikes by 1 m

# Make a vector of possible dike height increases in meters.
H = seq(0, 3, by = 0.001)

# Calculate the expected losses due to floods L, the cost of increasing the
# dikes I, and the total of L and I, all as a function of H.
L = p_0* exp(-alpha* H)* V/ (delta)
I = I_0+ k* H
Total = L+ I

# Make a plot.  The lowest point on the Total curve
# is the best value for H.
plot(H, L, type = 'l', col = 'red', xlab = 'Dike height increase H (m)',
     ylab = 'Cost (guilders)', bty = 'n')
lines(H, I, col = 'blue')
lines(H, Total, col = 'black', lwd = 2)

# Add a point indicating the minimum.
min_Total = Total[rank(Total) == 1]
min_H = H[rank(Total) == 1]
points(min_H, min_Total, pch = 16)
```

Monte Carlo simulation

The method proposed by Van Dantzig (1956) for estimating optimal dike heights includes a number of parameters whose values have to be estimated (see the list in the code block above). Suppose that the probability distribution of floods of different heights is well-known and the initial cost of building higher dikes is 0 (that is, `p_0`, `alpha`, and `I_0` are fixed). In that case, we still have three uncertain parameters, the value of goods protected by flooding `V`, the discount rate `delta`, and the cost of raising the dikes by 1 m `k`. How can we assess the uncertainty in the optimal dike height, given that these parameters aren't known perfectly?

Monte Carlo simulation (e.g. Bevington and Robinson, 2002, their ch. 5) provides a method for estimating the uncertainty in a calculated output, given probability distributions of the inputs. To carry out Monte Carlo simulation, we

1. generate groups of input parameter values randomly (see Exercise #3),

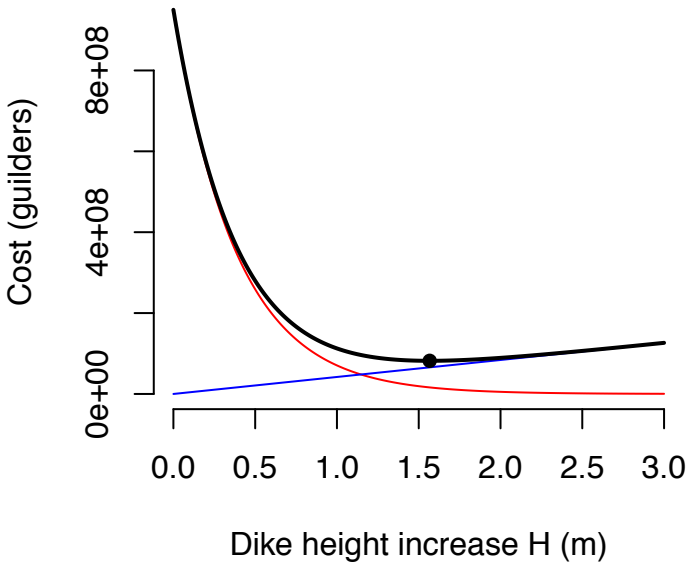

Figure 2: Expected total future damages due to flooding at different dike heights $I(H)$ (blue curve), the costs of building dikes of different heights $L(H)$ (red curve), and their sum (black curve). The minimum point on the black curve indicates the optimal dike height.

2. feed each group of input parameters into the model and record the outputs, and
3. plot the distributions of the output.

As a simple example, suppose we want to estimate the area of a circle, but we only know that its radius is somewhere between 0 and 1 units. We assume that the distribution of radius values is uniform, make a vector of possible values, feed them into the equation for a circle, and histogram the results. As shown in Figure 3, the distribution of outputs looks very different from the distribution of inputs.

```
# Creates randomly-sampled values of a circle's radius, calculates the area of
# the circle from each of the radius values, and makes histograms of the
# radius and area values.
n.trials <- 10^5    # number of Monte Carlo trials to attempt
radius <- runif(n.trials, min = 0, max = 1)
area <- pi* radius^2
par(mfrow = c(1, 2))
hist(radius, main = '', xlab = 'Radius (length)')
hist(area, main = '', xlab = 'Area (length^2)')
```

In the code block above, note that both `radius` and `area` are vectors that each contain `n.trials` values.

How could we apply Monte Carlo simulation to the Van Dantzig (1956) analysis? In addition to the equations above, Van Dantzig (1956) presents a separate equation that gives the optimal dike height directly,

$H_{best} = \alpha^{-1} \ln[(V p_0 \alpha)/(\delta k)]$

We can imagine generating vectors of the uncertain parameters, feeding them into the equation above, and making a histogram of the optimal dike height values for all the parameter groups. That histogram would give us some idea of how sure we can be about the optimal dike height.

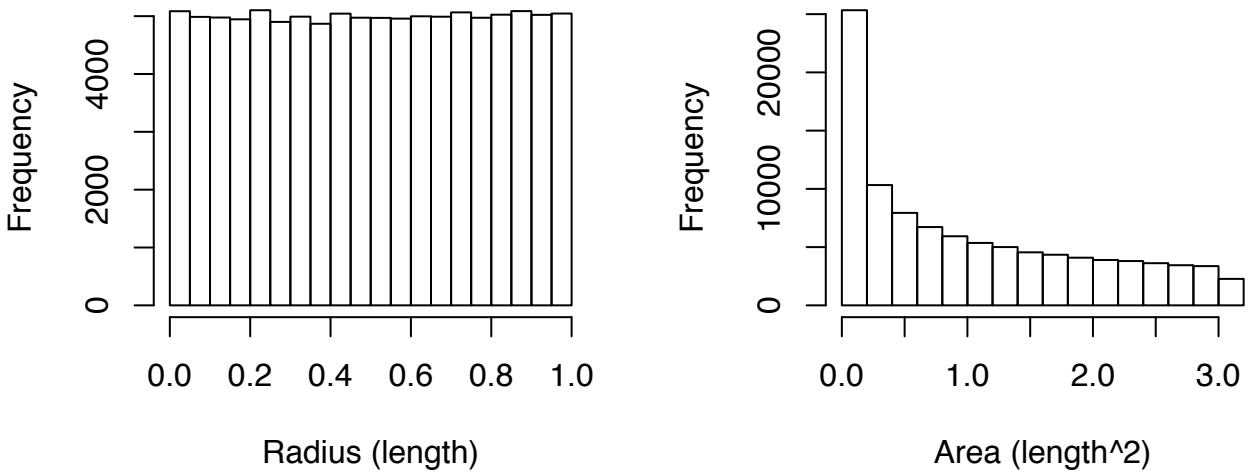

Figure 3: Randomly-sampled values of a circle's radius (input parameter; left panel) and the resulting circle areas (output value; right panel). Note that the distributions of the input and the output do not have the same shape.

Tutorial

Open the R script `lab4_sample.R` and inspect the contents. This script performs a simple Monte Carlo analysis with Van Dantzig (1956)'s equation for the optimal dike height, varying the parameter δ between 0.95 and 1.05 of its base value.

The first part of the code is straightforward; it provides an explanation of what the script does, clears the workspace, and sets the values for different parameters in the equation above.

```
# lab4_sample.R
# Patrick Applegate, patrick.applegate@psu.edu
#
# Performs a simple Monte Carlo analysis with the optimal dike height
# equation from Van Dantzig (1956).

# Clear any existing variables and plots.
rm(list = ls())
graphics.off()

# Constants from Van Dantzig (1956)
p_0 = 0.0038      # unitless; probability of flooding in 1 yr if the dikes
                  # aren't built
alpha = 2.6       # unitless; constant associated with flooding probabilities
V = 10^ 10        # guilders; value of goods threatened by flooding
delta = 0.04      # unitless; discount rate
I_0 = 0           # guilders; initial cost of building dikes
k = 42* 10^ 6     # guilders/m; cost of raising the dikes by 1 m
```

The next group of commands determines how many Monte Carlo calculations to perform (`n.trials`), the range of each parameter to search over (`range`), and how wide a range of values will be reported by the script when it's run (`probs`). `probs <- c(0.025, 0.5, 0.975)` tells the script to report the 95% range of the results, plus the median.

```
# Set some other values.
n.trials <- 10^5  # number of Monte Carlo trials to do
range <- 0.1      # fractional range of each parameter to test
probs <- c(0.025, 0.5, 0.975)
                  # which quantiles to report
```

The random sampling is handled in the next block of code. Note the use of the `set.seed()` command to ensure that the script will give reproducible results.

```
# Set the seed for random sampling.
set.seed(1)

# Perform the random sampling.
facs <- c((1- 0.5* range), (1+ 0.5* range))
delta.vals <- runif(n.trials, min = facs[1]* delta,
                max = facs[2]* delta)
```

Finally, the code calculates the optimal dike height for each value of `delta.vals` and makes a histogram of the `best.heights`, with a vertical red line to indicate the height obtained using the best estimate of each uncertain parameter. The quantiles of the values in `best.heights` are also written to the screen.

```
# Calculate the optimal dike heights.
best.heights <- alpha^-1* log((V* p_0* alpha)/ (delta.vals* k))

# Make a histogram and print the quantiles to the screen.
hist(best.heights, main = '', xlab = 'Optimal dike heights (m)')
abline(v = alpha^-1* log((V* p_0* alpha)/ (delta* k)), lwd = 2, col = 'red')
print(round(quantile(best.heights, probs = probs), 3))
```

Exercise

Part 1. Make a plot of the discount factor F_{d} for `delta = seq(0, 0.1, by = 0.2)` over the time interval 0-200 yr. (Make sure you are plotting the discount factor, not its cumulative sum as shown in Fig. 1.)
Part 2. Execute `lab4_sample.R` and take note of the quantiles that the script produces. Save a copy of the histogram produced by the script.
Part 3. Make a copy of `lab4_sample.R` by saving it with a different file name. Modify this copied file so that it incorporates randomly-selected `V` and `k` values into the Monte Carlo simulation. You'll need to create vectors `V.vals` and `k.vals` and populate them with random values, using code similar to that for `delta.vals`, above. You'll also need to change the line in which `best.heights` is calculated, to incorporate these values into calculation of the optimal dike heights. Execute this new script, take note of the quantiles, and save a copy of the histogram it produces.

Questions

1. For each of the discount factors you investigated in Part 1, how much weight do losses at the end of 100 years have relative to losses now?
2. Compare the distribution of `best.heights` from Part 2 to the distribution of values in `delta.vals`. Do these distributions look like one another?
3. Now compare the histograms and quantiles from Parts 2 and 3 to one another. How does the distribution of `best.heights` change as more free parameters are added to the calculation?

Appendix

After reading Van Dantzig (1956), address some or all of the following questions. These questions are intended for students with advanced backgrounds in the Earth sciences and R programming.

1. What dike height does Van Dantzig (1956) identify as a "reasonable estimate of a sufficiently safe height"? How does this value compare to that marked by the black dot in Figure 2? What are the reasons for this difference?
2. Make a list of the assumptions that Van Dantzig (1956) explicitly mentions in the paper. Can you think of any other assumptions that he does not discuss? How does each of these assumptions affect the answer? That is, if each of these assumptions were relaxed, would the optimal dike height be greater or smaller than the value identified by Van Dantzig (1956)?
3. Examine Morgan and Henrion (1990, their section 3.4.1). Which of the decision criteria from Morgan and Henrion (1990, their Table 3.2) did we apply in Figure 2? Which decision criterion does Van Dantzig (1956) use in determining his "reasonable estimate of a sufficiently safe height"? Are any of the other decision criteria from Morgan and Henrion (1990, their Section 3.4.1) potentially relevant for building flood protection structures?
4. Make a list of the parameters that go into the Van Dantzig (1956) analysis. How did Van Dantzig (1956) decide which value of each parameter to use? Do you agree with these parameter values? Why or why not? What determines whether a parameter combination is "appropriate" or "good" in the context of this problem?
5. The exercise above assumes that relative sea level is constant over time. However, as Van Dantzig (1956) points out, relative sea level is actually increasing over time due to sinking of the land (plus increases in global mean sea level, as discussed elsewhere in this e-textbook). How would such increases in relative sea level affect the simplified analysis presented above?
6. What does Van Dantzig (1956) mean by his reference to a "utility function"? How could a utility function be built into this analysis? How might the answer change if the analysis were redone in this way?
7. How could this analysis be applied to another place with a fundamentally different distribution of annual maximum flood heights?
8. Suppose that sea level is rising (as in question #5, above), but we can't be sure about the trajectory of future sea level change. How could we design a strategy for incrementally raising the dikes based on observations of yearly sea level?
9. In the original Van Dantzig (1956) analysis, the probability distribution of annual-maximum flood levels is constant over time. What would happen if higher floods, relative to mean sea level, became more probable over time? How could this change be built into this type of analysis?
10. Identify one other potential weakness in the Van Dantzig (1956) analysis and describe a way in which the analysis could be changed or updated to address this problem. If you were to carry out this improved analysis, would the optimal dike height be greater or smaller?

References

Bevington, P. R., and Robinson, D. K., 2002. Data Reduction and Error Analysis for the Physical Sciences. McGraw-Hill, 320 p.

Lempert, R., Sriver, R. L., and Keller, K., 2012. Characterizing uncertain sea level rise projections to support investment decisions. California Energy Commission White Paper CEC-500-2012-056.

Morgan, M. G., and Henrion, M., 1990. Uncertainty: A Guide to Dealing with Uncertainty in Quantitative Risk and Policy Analysis. Cambridge University Press, 332 p.

van Dantzig, D., 1956. Economic decision problems for flood prevention. Econometrica 24, 276-287.

Wemelsfelder, P. J., 1939. Wetmatigheden in het optreden van stormvloeden. De Ingenieur 54(9), 31-35.

Exercise #5: Fitting a second-order polynomial to sea level data

Patrick Applegate, patrick.applegate@psu.edu, and Ryan Sriver, rsriver@illinois.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- fit a second-order polynomial to sea level data
- write your own functions in R
- use `optim()` to perform function minimization

Introduction

In Exercise #4, we examined the question of how high to build structures for flood protection, based on the analysis of van Dantzig (1956). In that exercise, we identified the dike height that minimizes the total costs from dike construction and future flooding. We also examined how changing some of the input parameters within reasonable ranges affects the estimated optimal dike height.

This analysis is appealingly simple; however, it assumes no change in sea level, whereas tide gauge data shows that sea level is rising (see Exercise #1). Sea level has risen by a few tens of centimeters over the last two centuries (Jevrejeva et al. 2008, 2014), and could rise by a meter or more by the end of the century (e.g. Pfeffer et al., 2008). If we did not take this rise into account when building flood protection structures, we would build the dikes too low and be flooded much more often than expected as a result (Sriver et al., 2012).

If we wanted to incorporate future sea level change into the van Dantzig (1956) analysis, we would first need to characterize past sea level change. We can think of a time series, such as the sea level data, as a smooth curve with some superimposed "wiggles." For example, Jevrejeva et al. (2008) noted that the sea level data show a decline between ~1700 and ~1850, then an accelerating rise. These data have the general shape of a second-order polynomial (a parabola). The sea level data contains some short-period noise, though, so there will be some remaining differences between the data and any smooth curve drawn through the data. These remaining differences are called *residuals*.

Fitting a curve to data is a very common operation in risk analysis and scientific computing in general. As pointed out in the Wikipedia article, curve fitting allows us to separate the trend in a data set from its "wiggles" (smoothing), estimate a quantity between existing data points (interpolation), or estimate a quantity outside of the range of our data (extrapolation).

In this exercise, we'll fit a second-order polynomial to the sea level data from Jevrejeva et al. (2008), both manually and using automated methods. We'll also examine the residuals that are left over after performing this curve fit.

Tutorial

We'll use a somewhat older version of the sea-level data that we saw in Exercise #1, from Jevrejeva et al. (2008). The following code block creates a new directory and downloads the data into it:

```
# Create a directory called data and download a file into it.
dir.create('data')
download.file('http://www.psmsl.org/products/reconstructions/gslGRL2008.txt',
              'data/gslGRL2008.txt', method = 'curl')
```

Once the data are in hand, we can plot them with the following commands.

```
# Read in the information from the downloaded file.
sl.data <- read.table('data/gslGRL2008.txt', skip = 14, header = FALSE)

# Extract two key vectors from sl.data.
# t, time in years
# obs.sl, global mean sea level in mm
t <- sl.data[, 1]
obs.sl <- sl.data[, 2]

# Make a plot.
plot(t, obs.sl, type = 'l', xlab = 'Time (yr)', ylab = 'Sea level anomaly (mm)')
```

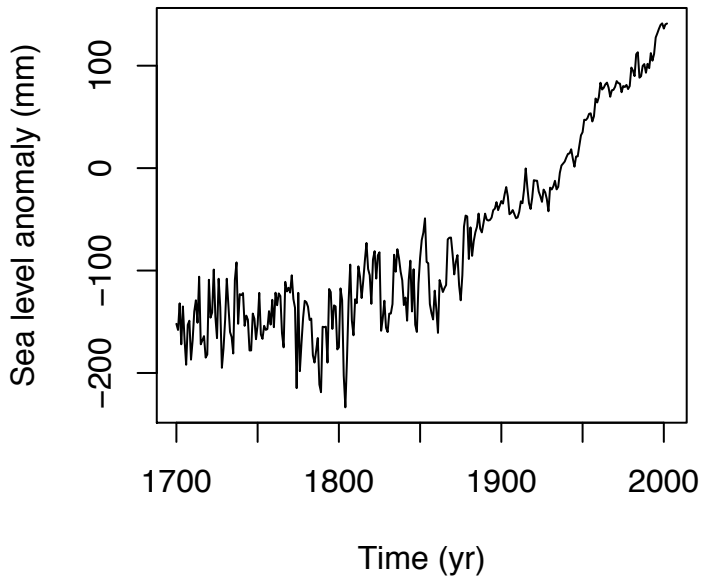

Figure 1: Global mean sea level anomaly as a function of time, from Jevrejeva et al. (2008).

How do we fit a curve to these data? Recall that a second-order polynomial has the equation

$y = ax^2 + bx + c$

Translating this equation to the sea level context, we have

$L_S = a(t - t_0)^2 + b(t - t_0) + c,$

where L_S is the sea level anomaly in millimeters, t is time in years, and a, b, and c are coefficients of the polynomial. The subtraction $t - t_0$ shifts the lowest point of the parabola to the right (otherwise it would be near $t = 0$).

What units do each of the free parameters have? Only quantities with the same units can be added or subtracted, and therefore each term in this equation must have the same units. It follows that

- t_0 must have the same units as t (years),

- a must have units of mm/yr^2,
- b must have units of mm/yr, and
- c must have units of mm.

Note that the units of a and b correspond to acceleration and velocity, respectively.

Fitting a second-order polynomial to the sea level data involves adjusting the free parameters a, b, c, and t_0 until the curve passes through the data. Open the R script `lab5_sample.R` and examine its contents. This script takes values of `a`, `b`, `c`, and `t.0` (set near the top of the script), plots the data and a second-order polynomial based on the free parameter values, and then displays the residuals (`resids`) between the observed and calculated sea level anomalies (`obs.sl` and `est.sl`, respectively).

The root mean squared error

The script also calculates the *root mean squared error* and prints it to the screen using the lines

```
# Find the root mean squared error between the observed and estimated sea level
# anomaly values.
rmse <- sqrt(mean((obs.sl- est.sl)^ 2))
print(rmse)
```

The root mean squared error (RMSE) is a metric for how well a given curve matches a particular data set. An RMSE of 0 indicates a perfect match, whereas large values of the RMSE indicate a poor match between a curve and the data. Because the errors are *squared*, the RMSE penalizes outliers heavily. That is, parameter combinations that leave large residuals will receive a high (bad) RMSE score compared to parameter combinations that give smaller residuals between the calculated and observed values. The RMSE has the same units as the differences being analyzed (in the case of the sea level data, mm).

Optimization and functions

R contains a built-in function, `optim()`, that can fit the second-order polynomial to the sea level data for us. `optim()` is a function minimization algorithm: it adjusts the parameters of a function to find lower and lower function values, reflecting increasingly better fits.

If we write a function that accepts

- the free parameters,
- a vector of time values, and
- a vector of observed sea level anomalies as inputs,

and returns the root mean squared error as an output, we can feed that function into `optim()`. An outline for this function might look like this:

```
# Sample function for calculating the root mean squared error given a set of
# parameters, a vector of time values, and a vector of sea level values.
sl.rmse <- function(params, time, sea.levels) { # don't forget the curly braces

  # Step 1: Pick apart the vector params into its individual values.
  a <- params[1]
  b <- params[2]
  c <- params[3]
  t.0 <- params[4]
```

```
  # Step 2: Calculate the estimated sea level anomalies based on a, b, c, t.0,
  # and time.

  # Step 3: Calculate the rmse based on the estimated sea level anomalies
  # calculated in Step 2 and the observed values passed into the function
  # in sea.levels.

  # Step 4: Return the rmse value calculated in Step 3.
  return(rmse)

} # don't forget the curly braces
```

In the function outline above, Steps 1 and 4 are done for you, but you'll need to fill in Steps 2 and 3 on your own (see the Exercise, below).

Once you have your function, you can incorporate it into `optim()` with the code

```
# Optimize the sl.rmse function.
start <- c(0, 0, -100, 1800)
optim.fit <- optim(start, sl.rmse, gr = NULL, time = t, sea.levels = obs.sl)

# Extract the parameters of the best-fit polynomial, and the root mean squared
# error, from optim_fit.
best.a <- optim.fit$par[1]
best.b <- optim.fit$par[2]
best.c <- optim.fit$par[3]
best.t.0 <- optim.fit$par[4]
best.rmse <- optim.fit$value
```

This code specifies a starting group of parameter values (`start`), applies `optim()` to find the best parameter combination, and then picks apart `optim.fit` to find the best parameter combination and RMSE value.

Exercise

Part 1. In the script `lab5_sample.R`, experiment with the free parameters `a`, `b`, `c`, and `t.0` (set near the top of the script) to find a good match to the data. To do this, open `lab5_sample.R`, set the working directory to be the same as that where the script resides, and `source` the script repeatedly, changing the parameter values each time. Make a note of your best-fit parameter combination and its associated root mean squared error value, and save off the plot associated with this parameter combination.
Part 2. Save a copy of `lab5_sample.R` with a different file name. Write a function `sl.rmse()` that accepts a vector of parameter values, a vector of time values, and a vector of observed sea level values as inputs, and returns their RMSE value. The general format of this function is given above. Embed this function near the top of your R script, but *below* the line `rm(list = ls())`. If you insert the following lines near the bottom of your modified R script and `source()` the script, you should get `[1] 96.34322`.

```
# check on function sl.rmse
test.rmse <- sl.rmse(params = c(0, 0, -100, 1800), time = t,
                     sea.levels = obs.sl)
print(test.rmse)
```

Part 3. Save a copy of your R script from Part 2 under a different file name. Modify this script so that it uses `optim()` to find the best-fitting parameter values, plots the best-fit second-order polynomial on top of

the data, and displays the resulting residuals. Your final script should *not* include the lines that set values of `a`, `b`, `c`, and `t.0` near the top of `lab5_sample.R`. Again, note the best-fit parameter combination, and save off the plot produced by the script.

Questions

1. In Part 1, above, how does each parameter change the appearance of the second-order polynomial and the magnitude of the residuals? Relate your answer to the general equation for a second-order polynomial in x-y space, given above.
2. Compare your results from Parts 1 and 3, above. How do the parameter combinations that you identified by manual and automated methods differ? Compare the RMSE values and the graphs that you obtained by these two methods.

Appendix

This question is intended for students with advanced backgrounds in the Earth sciences and R programming.

Jaynes (2003) provides some interesting historical and statistical background on why the RMSE is so commonly used as a metric for describing the agreement between a given curve and a particular data set. However, the RMSE is just one of many possible metrics that could be used for this purpose. What are other metrics than could be used to quantify how well a model fits observations? How does using one of these alternative metrics, instead of the RMSE, change the results of this exercise? How might the use of these alternative metrics affect the outcomes of risk analyses?

References

Jaynes, E. T., 2003. Probability Theory: The Logic of Science. Cambridge University Press, 727 pp.

Jevrejeva, S., Moore, J. C., Grinsted, A., and Woodworth, P. L., 2008. Recent global sea level acceleration started over 200 years ago? Geophysical Research Letters 35, L08715.

Jevrejeva, S., Moore, J. C., Grinsted, A., Matthews, A., and Spada, G., 2014. Trends and acceleration in global and regional sea levels since 1807. Global and Planetary Change 113, 11-22.

Pfeffer, W. T., Harper, J. T., and O'Neel, S., 2008. Kinematic constraints on glacier contributions to 21st-century sea-level rise. Science 321, 1340-1343.

Sriver, R. L., Urban, N. M., Olson, R., and Keller, K., 2012. Toward a physically plausible upper bound of sea-level rise projections. Climatic Change 115, 893-902.

van Dantzig, D., 1956. Economic decision problems for flood prevention. Econometrica 24, 276-287.

Exercise #6: Coin flipping and the bootstrap

Patrick Applegate, patrick.applegate@psu.edu, and Klaus Keller, klaus@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- perform a simple bootstrap analysis
- use the `sample()` command to both generate series of coin flips and bootstrap replicates from existing data sets

Introduction

In the earth sciences, we often have to make inferences with limited data. This problem can be especially acute in climate science, where observational records of temperature and sea level rise (for example) go back a few hundred years at most. Given these limited data, how can we make inferences about the past and future of the climate system, while accurately representing our uncertainties?

A statistical procedure called the *bootstrap* provides a means of assessing how sure we can be about our answers given available data. The bootstrap involves generating "replicates" of the original data set by sampling, with replacement, from the original data set, and calculating the statistic of interest from each of the replicates. The distribution of this statistic is then a guide to our uncertainty in the statistic as estimated from the original data set.

In this exercise, we'll examine the application of the bootstrap to coin flipping. Coin flips provide a simple, easily-simulated system for thinking about probability; we will generalize the lessons from coin flips to sea level data in the next exercise.

Tutorial

Suppose we have a fair coin, one in which the ratio of heads to the total number of flips $c = 0.5$ over the long term. If we flip this coin many times and calculate our *estimate* of c based on the data we have after each flip, we will get something like Figure 1.

```
# Perform a simulation of successive estimates of c, the ratio of heads to the
# total number of coin flips, for a fair coin.

# Set some values.
true.c <- 0.5          # the true ratio of heads to total flips (0.5 indicates
                       # a fair coin)
n.flips <- 5* 10^ 2    # how many coin flips to perform

# Perform the random sampling and calculate the estimates of c.  1, heads;
# 0, tails.
set.seed(1)
ht <- sample(x = c(0, 1), size = n.flips, replace = TRUE,
             prob = c(true.c, 1- true.c))
```

```
flips <- seq(from = 1, to = n.flips, by = 1)
c.ests <- cumsum(ht)/ flips

# Plot up the results.
plot(flips, c.ests, type = 'l', col = 'blue', log = 'x',
     xlab = 'Number of flips carried out (log scale)', ylab = 'Estimates of c')
abline(h = true.c, lty = 2)
```

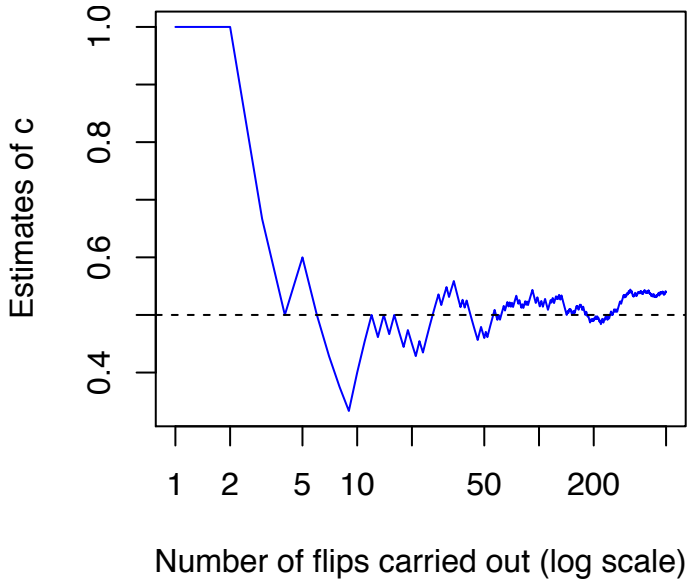

Figure 1: Successive estimates of c (blue curve) after varying numbers of flips with a fair coin that has a true value of $c = 0.5$ (black, dashed line).

Looking at Figure 1, we can see that our c estimates only settle down close to the true value after a few tens of flips. Estimates of c based on smaller numbers of coin flips can be quite inaccurate.

Generating bootstrap replicates from a vector of coin flips

Now assume that we have just ten coin flips. If this data set consists of five heads and five tails, our best estimate of c is 0.5:

$c_{est} = n_{heads}/n_{flips} = 5/10 = 0.5$

The following code block stores the data in `ht.data`, generates just one bootstrap replicate and stores it in `ht.boot`, and calculates the c values associated with the data and the bootstrap replicate. It then prints the bootstrap replicate and its associated c value to the screen.

```
# Set up a vector of coin flips that represents the data.  1, heads; 0, tails.
ht.data <- c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1)

# Calculate the value of c based on ht.data.
c.data <- sum(ht.data)/ length(ht.data)

# Generate one bootstrap replicate based on ht.data and calculate the
# associated estimate of c.
set.seed(1)
```

```
ht.boot <- sample(x = ht.data, size = length(ht.data), replace = TRUE,
                  prob = NULL)
c.boot <- sum(ht.boot)/ length(ht.boot)

# Print the bootstrap replicate and the c estimate to the screen.
paste('ht.boot =', paste(ht.boot, collapse = ' '))
paste('c.boot =', c.boot)
```

```
## [1] "ht.boot = 0 0 1 1 0 1 1 1 1 0"
## [1] "c.boot = 0.6"
```

In this case, we get a bootstrap replicate in `ht.boot` that contains six heads (`1`) and four tails (`0`), and therefore has a `c.boot` of 0.6.

Examine the `help()` file for `sample()`; can you see why the arguments `x`, `size` and `prob` differ between the two code blocks above? Note that we used `sample()` to accomplish two different tasks:

- In the first code block, we used `sample()` to generate a sequence of coin flips from a coin with a known c.
- In the second code block, we used the same function to generate a single bootstrap realization by random sampling, with replacement, from a data vector.

Exercise

Part 1. Using the second code block above and what you learned about `for` loops in Exercise #2, write an R script that

- generates `n.boot <- 10^3` bootstrap replicates of the ten flips stored in `ht.data`, above
- calculates a c value for each bootstrap replicate, and stores these c values in a vector `c.vals` (you should define this vector *before* the `for` loop and use the loop's counter variable to save each new value of c in a different element of `c.vals`)

```
c.vals <- rep(NA, n.boot)
for (i in 1: n.boot) {
  # one or more commands for generating the bootstrap replicates go here
  c.vals[i] <- # calculate your c values on the right-hand side here
}
```

- makes a histogram of these c values and plots `c.data` on top of the histogram as a vertical, red line (this line should coincide with the peak of the histogram)
- calculates the 95% range of `c.vals` using `quantile(c.vals, probs = c(0.025, 0.975))`.

Part 2. Save a copy of your script from Part 1 using a different file name. Incorporate code from the first code block above into this new script so that it generates a synthetic data set of a given length and `true.c`, and then performs the bootstrap analysis from Part 1.

Questions

1. What is the 95% range of `c.vals` in Part 1?
2. `source()` your code from Part 2 repeatedly using a true c value of 0.5 and a length for the original data set of 10 flips. How does the histogram change depending on the number of heads in the original data set?
3. Again using your code from Part 2, generate synthetic data sets with lengths of 100 and 1000 coin flips and repeat the bootstrap analysis each time. Use a true c value of 0.5. How do the histograms and 95% ranges in these experiments compare to that from Question 1?
4. Set your code from Part 2 to use 1000 coin flips and change the true c value to 0.3 and 0.8. Compare the histograms and 95% ranges in these experiments to those from Question 3.

Exercise #7: Performing a simple bootstrap with the sea level data

Patrick Applegate, patrick.applegate@psu.edu, Ryan Sriver, rsriver@illinois.edu, and Klaus Keller, klaus@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- perform a very simple bootstrap of time series data by resampling residuals
- identify potential problems in applying this approach to "real" data, such as the sea level data from Jevrejeva et al. (2008)

Introduction

Making estimates of potential future changes, with uncertainties, is a key aspect of environmental risk analysis. For example, the amount by which sea level will rise in the future is clearly an important input for the design of flood protection structures; if sea levels are expected to rise by several meters over the next few centuries, then perhaps dikes must be built higher than if they are expected to rise by a few tens of centimeters.

In this exercise, we'll combine techniques from the two preceding chapters to demonstrate techniques that could be used to estimate future sea level rise. In Exercise #5, we fitted a second-order polynomial to sea level data covering the last ~300 yr. In Exercise #6, we learned how to perform a simple bootstrap analysis using coin flips. Here, we'll apply the bootstrap to estimate our uncertainties in future sea level based on curve fitting.

It's important to note that the sea level rise projections you will produce in this exercise are probably too low, and the associated uncertainties are also incorrect. The approach used in this exercise assumes that

- adjacent residuals aren't correlated with one another
- the scatter in the residuals about the "true," underlying curve doesn't change over time
- the "true" curve on which the "wiggles" are superimposed really is a second-order polynomial

In fact, all three of these assumptions are violated by the sea level data. The analysis presented in this exercise mostly ignores sea level contributions from the ice sheets, and neglects autocorrelation and heteroscedasticity in the residuals. However, this simplified analysis is an important stepping stone to methods that do account for ice sheets and provide better estimates of the actual uncertainty.

Tutorial

In Exercise #6, we performed the bootstrap by resampling our *observations*, but we'll need a different approach for the sea level data. What would happen if we were to simply resample the actual sea level observations from Exercise #5? The following code block reads in the data, plots the original data, and then plots a single bootstrap realization of the data. As you can see from the plots, resampling the sea level data destroys any trend that they contain.

```
# Read in the information from the downloaded file.
sl.data <- read.table('data/gslGRL2008.txt', skip = 14, header = FALSE)

# Extract two key vectors from sl.data.
# t, time in years
# obs.sl, global mean sea level in mm
t <- sl.data[, 1]
obs.sl <- sl.data[, 2]

# Resample the observations in obs.sl.
re.sl <- sample(obs.sl, length(obs.sl), replace = TRUE)

# Make some plots.
par(mfcol = c(1, 2))
plot(t, obs.sl, type = 'l', xlab = 'Time (yr)', ylab = 'Sea level anomaly (mm)')
plot(t, re.sl, type = 'l', xlab = 'Time (yr)', ylab = 'Resampled anomalies (mm)')
```

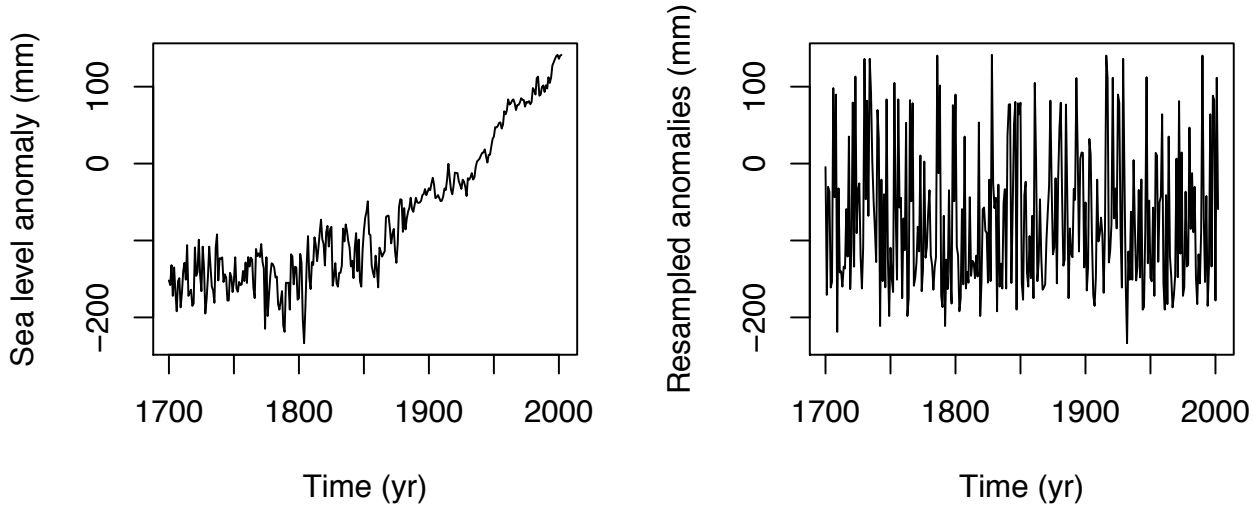

Figure 1: Sea level anomaly data from Jevrejeva et al. (2008; left panel), and the anomalies resampled with replacement (right panel). Simple resampling of the anomaly data destroys any trend or signal in the original data set.

Instead, we'll resample the *residuals* between our best-fit curve from Exercise #5 and the data, then add these resampled residuals back on to the best-fit curve. This approach will preserve the underlying trend in the data and let us learn something about the uncertainties in our estimates of a, b, c, t_0, and future sea level.

Let's examine how to generate one bootstrap replicate using this method and how we would make a projection of future sea level based on that replicate. Your code from Exercise #5 should include a vector that contains the differences between the best-fit second-order polynomial and the data. Assuming that this vector is called `resids`, the following command will create a vector of bootstrapped residuals:

```
# Generate a bootstrap replicate of the residuals.
boot.resids <- sample(resids, length(resids), replace = TRUE)
```

But these residuals only represent the "wiggles," not the long-term trend in the data. Your code should include a vector that contains the past long-term change in sea level based on the best-fit second-order polynomial. Let's assume that this vector is called `best.sl`. We can get a plausible estimate of past sea levels, including both the underlying trend and noise, by adding our resampled residuals onto this vector.

```
# Add the resampled residuals to the best-fit second-order polynomial.
boot.sl <- best.sl+ boot.resids
```

To see why the code block above works, recall that R adds vectors of the same length element-by-element.

Now we fit a curve to the new sea level "data" and extract the best-fit parameters from it, using code like that we saw in Exercise #5:

```
# Fit a second-order polynomial to the residuals+ trend.
boot.start <- c(best.a, best.b, best.c, best.t.0)
boot.fit <- optim(boot.start, sl.rmse, gr = NULL, time = t,
                  sea.levels = boot.sl)

# Extract the new parameter values from boot.fit.
boot.a <- boot.fit$par[1]
boot.b <- boot.fit$par[2]
boot.c <- boot.fit$par[3]
boot.t.0 <- boot.fit$par[4]
```

Note that the code block above assumes that you have a function `sl.rmse` and best-fit parameter values stored in `best.a`, `best.b`, `best.c`, and `best.t.0`.

We can make an estimate of past and future trends in sea level by defining a new, longer, vector of time values and using `boot.a`, `boot.b`, and so on to calculate values of a second-order polynomial:

```
# Define a vector of years extending from 1700 to 2100.
proj.t <- seq(1700, 2100)

# Estimate the trends in past and future sea levels based on boot.a, etc.
proj.sl <- boot.a* (proj.t- boot.t.0)^ 2+ boot.b* (proj.t- boot.t.0)+ boot.c
```

Exercise

Before you proceed, **make sure that you have a working version of the script from Exercise #5.** Open your script in RStudio and `source()` it. Examine the value in `best.rmse`, perhaps by typing `best.rmse` in the Console sub-window and pressing Enter. You should get a value of `[1] 24.4101`. If not, check your script against the instructions in Exercise #5. Once you're satisfied that your script from Exercise #5 is working properly, **make a copy** by saving it under a different file name.

Part 1: Carrying out the bootstrap. Modify your new script so that it generates `n.boot <- 10^3` bootstrap realizations of past sea levels and fits a second-order polynomial to each realization, using a `for` loop. As in Exercise #6, you'll need to create vectors *before* the `for` loop to contain the results of your calculations, and then store the results in different elements of these vectors using the `for` loop's index variable. You'll also need to create a `matrix()`, perhaps called `proj.sl`, for storing the trends in past and future sea levels based on `boot.a`, `boot.b`, and so forth.

Your script should make the following plots:

1. a two-panel plot with the top panel showing the original sea level data, displayed as `type = 'l'`, with the best-fit second-order polynomial on the same set of axes and a separate panel showing the residuals (as in Exercise #5)
2. a plot showing the second-order polynomials from each of the bootstrap realizations as gray curves (use `matplot()`), the original sea level data in black, and the best-fit second-order polynomial to the original data as a red curve

3. histograms of `boot.a`, `boot.b`, `boot.c`, and `boot.t.0`, with the values from the original data shown as vertical, red lines – make sure to label the axes of your plots in a sensible way and include the units of each quantity
4. a histogram of your sea level estimates in 2100 (these values are stored in `boot.sl`), with the estimate from the original data shown as a vertical, red line

Part 2: Identifying problems with this approach. Apply the `acf()` function to both the original residuals and one bootstrap replicate of the residuals. Plot these autocorrelation functions in two panels, one on top of the other, with sensible labels. On these plots, any bars that extend above or below the blue, dashed lines indicate that there is significant autocorrelation between residuals that have spacings shown on the x-axis.

Questions

1. Compare your sea level rise estimates in 2100 to the four scenarios in NOAA (2012). Which scenario do your sea level rise estimates match most closely? Does this scenario represent an optimistic or a high-end estimate of future sea level rise, according to the NOAA report? How was this scenario constructed?
2. Examine the residuals (Part 1) and the autocorrelation functions (Part 2) for evidence of heteroscedasticity and autocorrelation. Do the residuals, based on the original data and the best fit to them, show evidence of these problems? What about the resampled residuals?

References

Jevrejeva, S., Moore, J. C., Grinsted, A., and Woodworth, P. L., 2008. Recent global sea level acceleration started over 200 years ago? Geophysical Research Letters 35, L08715.

NOAA (National Oceanic and Atmospheric Administration), 2012. Global sea level rise scenarios for the United States National Climate Assessment. NOAA Technical Report OAR CPO-1. Available online at http://cpo.noaa.gov/AboutCPO/AllNews/TabId/315/ArtMID/668/ArticleID/80/Global-Sea-Level-Rise-Scenarios-for-the-United-States-National-Climate-Assessment.aspx

Exercise #8: Climate policy and the DICE model

Patrick Applegate, patrick.applegate@psu.edu, Greg Garner, ggg121@psu.edu, Richard Alley, rba6@psu.edu, and Klaus Keller, klaus@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- describe what the social cost of carbon (SCC) is
- explain in broad terms what the DICE model is and how it works
- perform simple calculations with the DICE model
- describe how uncertainties in the climate sensitivity affect the present-day social cost of carbon

Introduction

As noted in the Introduction to this e-textbook, fossil fuel use creates benefits for people, but also imposes costs in the form of climate change. Burning fossil fuels releases energy and waste products including carbon dioxide and water. The energy leads to economic productivity. However, the carbon dioxide causes long-lasting temperature increases, which lead to increased risks for people in the future. Most of the impacts associated with climate change will probably be harmful for people.

Because fossil fuel users pay less than the full cost associated with their actions, emissions of carbon dioxide are larger than the economically-optimal amount. An individual fossil fuel user pays an amount of money per unit of energy that represents the cost of extraction, processing (for example, gasoline is a highly processed derivative of oil), transportation, and profits, plus taxes and minus any subsidies. This cost does not reflect the increased risks to future people associated with climate change. Because people respond to price signals, they tend to consume more fossil fuel than they would if the price reflected the full cost to society of fossil fuel consumption (and were therefore higher).

The social cost of carbon (SCC) is the increase in climate-related damages caused by emitting an additional ton of CO_2 to the atmosphere (Nordhaus, 2013). In other words, if we were to emit one ton of CO_2 today, the SCC would be the sum of the negative impacts caused by that additional ton of CO_2, from now into the distant future. The future damages from carbon dioxide emissions are typically discounted in calculating the SCC; see Exercise 4 for a brief discussion of discounting. In 2010, the US Environmental Protection Agency estimated a value for the SCC of about \$21/t CO_2 (Interagency Working Group on Social Cost of Carbon, 2010, 2013; Johnson and Hope, 2012).

The Dynamic Integrated model of Climate and the Economy (DICE; Nordhaus, 2013; Nordhaus and Sztorc, 2013) includes the feedbacks between the climate system and the economy that are needed for estimating the SCC. As discussed previously, greenhouse gas emissions cause temperature increases, which cause climate-related damages to the economy. These damages can then motivate the development of policies and technologies that reduce emissions and possibly remove greenhouse gases from the atmosphere. A model like DICE that "closes the loop" between climate change and further emissions is necessary for accurate estimation of future damages and therefore the SCC. DICE is one of three models used by the EPA to estimate the social cost of carbon.

DICE is commonly used to identify optimal climate policies, sometimes subject to different constraints. When run in optimization mode, DICE maximizes a "utility function" that includes discounting as well as the

declining marginal benefit of additional consumption. That is, rich societies benefit less from an additional dollar of income than poor ones.

However, before DICE can be optimized, some assumptions have to be made about the climate sensitivity, among other parameters. As we saw in the Introduction, the climate sensitivity represents the amount by which global mean air temperatures would increase if carbon dioxide concentrations in the atmosphere were to double. Although many studies have estimated the climate sensitivity, this parameter remains deeply uncertain.

In this exercise, we use DICE to investigate how different climate and economic variables might change in the future under an optimized climate policy, plus how different values of the climate sensitivity affect the present value of the SCC.

An important note

Carrying out this exercise will not prepare you to estimate the social cost of carbon for publication in scientific journals or for use in policy applications. The presentation of the material in this exercise has been simplified for pedagogical reasons, and this material should only be applied to "real" problems after additional training. The authors and editors of this e-textbook specifically disclaim any liability for damages associated with the use of the material presented in this exercise.

Tutorial

The version of the DICE model that we use here was translated to R by Greg Garner from an original version in the GAMS language provided by Bill Nordhaus. The `dice.R` file contains the following useful functions (among others):

- `dice.new()`: creating a new instance of the DICE model with a prechosen set of parameters
- `dice.modify()`: changing the value of model parameters in DICE
- `dice.run()`: running the DICE model into the future using a given set of parameters
- `dice.solve()`: identifying an optimal trajectory using the utility function sketched above

Open `lab8_sample.R` in RStudio and examine the code that it contains. The following lines of code explain who wrote the code, describe what the code does, `source()`s the file containing the DICE model, and loads the DICE model into memory using the `dice.new()` function.

```
# lab8_done.R
# Patrick Applegate, patrick.applegate@psu.edu; Greg Garner, ggg121@psu.edu
#
# Optimizes the DICE model to produce a plausible climate-economic trajectory
# and performs a Monte Carlo experiment to evaluate the effects of uncertainty
# in the climate sensitivity on the present-day social cost of carbon.

# Mise en place.
rm(list = ls())
graphics.off()

# Load the DICE model.
source('dice.R')

# Create a new DICE object.
my.dice <- dice.new()
```

Next, we optimize the DICE model using the utility function described in the Introduction and pull some time-dependent output out of the object that results.

```
# Solve the DICE object for the optimal control policy.
# NOTE: THIS STEP MAY TAKE A COUPLE OF MINUTES!
dice.solve(my.dice)

# Set sensible names for the time-dependent variables to extract from DICE.
names <- c('Time (yr)',
           'Emissions (Gt CO_2/yr)',
           'Atmospheric [CO_2] (ppm)',
           'Global mean T anomaly (C)',
           'Climate damages (10^12 $)',
           'Social cost of carbon ($/t CO_2)')

# Make a place to store time-dependent output from optimized DICE.
opt.output <- matrix(data = NA, nrow = length(my.dice$year), ncol = 6)
colnames(opt.output) <- names

# Put the time.dependent output from optimized DICE into opt.output.
opt.output[, 1] <- my.dice$year
opt.output[, 2] <- my.dice$e
opt.output[, 3] <- my.dice$mat
opt.output[, 4] <- my.dice$tatm
opt.output[, 5] <- my.dice$damages
opt.output[, 6] <- my.dice$scc
```

The script also makes a plot of the time-dependent output.

```
# Plot the time-dependent output from the optimized DICE object.
pdf('lab8_plot1.pdf', width = 5, height = 8.5)
par(mfrow = c(5, 1))
plot(opt.output[, 1], opt.output[, 2], type = 'l', bty = 'n', xlab = names[1],
     ylab = names[2])
plot(opt.output[, 1], opt.output[, 3], type = 'l', bty = 'n', xlab = names[1],
     ylab = names[3])
plot(opt.output[, 1], opt.output[, 4], type = 'l', bty = 'n', xlab = names[1],
     ylab = names[4])
plot(opt.output[, 1], opt.output[, 5], type = 'l', bty = 'n', xlab = names[1],
     ylab = names[5])
plot(opt.output[, 1], opt.output[, 6], type = 'l', bty = 'n', xlab = names[1],
     ylab = names[6])
dev.off()
```

Running DICE in optimization mode and examining its time-varying output

Execute `lab8_sample.R` using the `source()` command or button (remember to set your working directory first), and look at the `.pdf` file that results. It should look like Figure 1.

The script also prints out the assumed climate sensitivity value and the present-day social cost of carbon (`opt.scc <- opt.output[1, 6]`) obtained using the optimization.

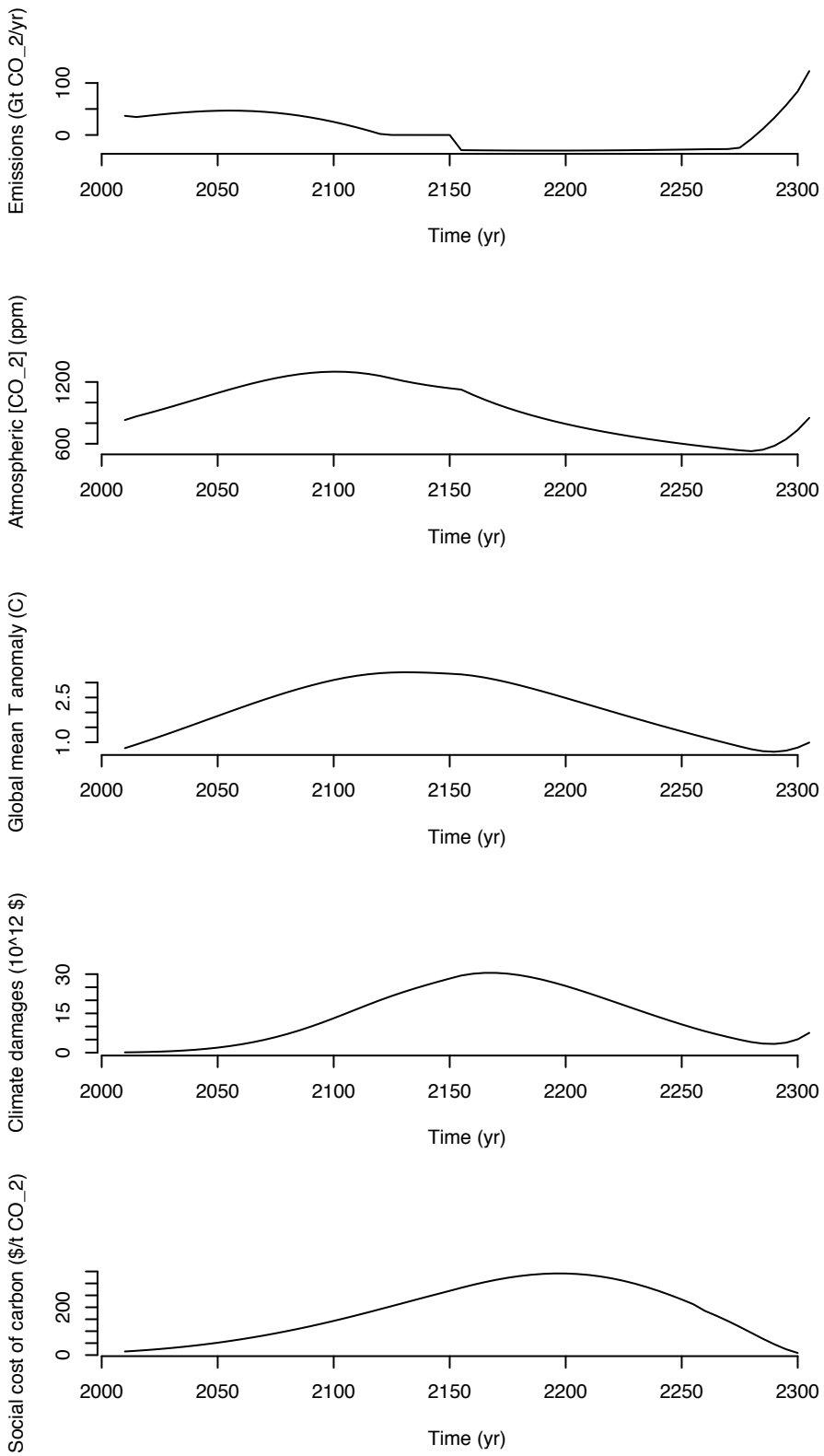

Figure 1: Time-dependent output from the optimized DICE model using the built-in parameter set. See text for discussion.

```
# Also extract the preset climate sensitivity and social cost of carbon
# from the DICE object.
opt.t2xCO$_2$ <- my.dice$t2xCO$_2$
opt.scc <- opt.output[1, 6]

# Print some key quantities.
print(sprintf('The assumed climate sensitivity for optimization is %2.2f C/doubling',
              opt.t2xCO$_2$))
print(sprintf('The optimized social cost of carbon in %d is $%4.2f/t CO_2',
              opt.output[1, 1], opt.scc))
```

Several interesting observations come out of Figure 1:

1. Each model curve bends sharply upward in the last few decades.
2. Ignoring the last few decades of model results, the peak of each curve is lagged in time relative to the peak of the preceding curve; so, peak temperatures occur after peak carbon dioxide concentrations, which occur after peak emissions.
3. The social cost of carbon is not constant; it starts out relatively small and grows over time until about 2200, when it begins to decline again.

We can ignore the last few decades of each time series shown in Figure 1. The DICE model assumes, in effect, that the world ends shortly after 2300 and that this apocalypse is known in advance, so that the people of the world start burning fossil fuels again.

The lags between the peaks in each curve happen for the following reasons.

- The atmospheric CO_2 concentration continues to increase after maximum net CO_2 emissions because the rate at which CO_2 is removed from the atmosphere is still smaller than the rate of release. Once net CO_2 emissions balance with the rate of CO_2 removal from the atmosphere, the atmospheric CO_2 peaks and subsequently declines as the net CO_2 emissions continue to drop.
- The atmospheric temperature peaks later than the atmospheric CO_2 due to the way energy (in the form of heat) is moved between the atmosphere and the ocean. Although the atmosphere responds quickly to additional forcing, the oceans are slow to remove the heat from the atmosphere. The peak in atmospheric temperature occurs when the ocean heat uptake balances the radiative forcing produced by the level of CO_2 in the atmosphere.
- The climate damages are a function of atmospheric temperature and gross world product. Once atmospheric temperatures begin to decrease, the damages as a fraction of the gross world product decreases; however, the gross world product is constantly increasing over time, and this continued increase results in the delayed peak in climate damages.
- As discussed earlier, the social cost of carbon represents the sum of future climate damages from the emission of a single ton of CO_2. Because the model assumes that the world ends shortly after 2300, the future damages due to a unit increase in CO_2 emissions decline as the model simulation approaches the end of the simulation period. As a result, the social cost of carbon first increases, then decreases.

Accounting for uncertainty in the climate sensitivity

DICE assumes a climate sensitivity value of 2.9 C per doubling of CO_2 concentration in the atmosphere. Although this value is reasonable, it doesn't capture our uncertainty in the actual value of the climate sensitivity. If we were to change this parameter within reasonable limits, we would get different climate and economic trajectories (Fig. 1), and a different present-day value for the social cost of carbon.

We could account for the effects of uncertainty in the climate sensitivity on present-day social cost of carbon values by performing a Monte Carlo experiment; however, we would first need a well-defined distribution for

the climate sensitivity. We could then run DICE repeatedly with climate sensitivity values sampled from this distribution and examine the distribution of present-day social cost of carbon values that came out.

One *approximate* method involves matching a lognormal distribution to the probabilistic statements of the Intergovernmental Panel on Climate Change on climate sensitivity. The lognormal distribution is only defined for positive values, and is therefore appropriate for variables like climate sensitivity that can't be negative. The latest IPCC report says, "Equilibrium climate sensitivity is... extremely unlikely less than 1°C... and very unlikely greater than 6°C..." (Alexander et al., 2013). Mastrandrea et al. (2010, their Table 1) suggest that "extremely unlikely" corresponds to a likelihood of 0-5%, and "very unlikely" corresponds to a likelihood of 0-10%.

The following code block accomplishes this matching, assuming the largest likelihood values for "extremely unlikely" and "very unlikely" (5% and 10%, respectively). It also generates a vector of random values from the resulting distribution.

```
# Set some values.
xs <- c(1, 6)       # climate sensitivities corresponding to the probabilities
                    # in ps
ps <- c(0.05, (1- 0.1))
                    # (approximate) probabilities of the climate sensitivity
                    # being less than xs[1] or greater than xs[2], according
                    # to IPCC AR5 WG1
n.trials <- 300     # number of Monte Carlo trials

# Define a function for matching the lognormal distribution to two (or more)
# tie points.
lnorm.rmse <- function(dist.params, xs, ps) {
  # dist.params, vector of meanlog and sdlog values;
  # see help(dlnorm)
  # xs, vector of values of the distributed variable to match
  # ps, probabilities of the values in xs
  logmu <- dist.params[1]
  logsigma <- dist.params[2]
  trial.xs <- qlnorm(ps, logmu, logsigma)
  rmse <- sqrt(mean((xs- trial.xs)^ 2))
  return(rmse)
}

# Identify the parameters of the lognormal distribution that produce the best
# match to the IPCC's statements about climate sensitivity.
lnorm.optim <- optim(log(c(2.9, 1.5)), lnorm.rmse, gr = NULL, xs = xs, ps = ps,
                     method = 'L-BFGS-B', lower = c(0, 0), upper = c(Inf, Inf))

# Generate a vector of climate sensitivity values.
set.seed(1)
tx2CO$_2$s <- rlnorm(n.trials, meanlog = lnorm.optim$par[1],
                   sdlog = lnorm.optim$par[2])
```

Exercise

Make a copy of `lab8_sample.R` by saving it under a different file name. Modify this copy so that it performs the Monte Carlo experiment described above using a `for` loop, with `n.trials <- 300` samples. For each value in `tx2CO$_2$s`, you'll need to

1. set the value of climate sensitivity in DICE using `dice.modify(my.dice, "t2xCO$_2$", tx2CO$_2$s[i])`
2. run the DICE model using `dice.run(my.dice)`
3. extract the 2010 value of the social cost of carbon from the DICE object and store it in the `i`th element of a vector `sccs`

Your modified script should make a plot with two panels. The top panel should show the distribution of climate sensitivity values based on fitting a lognormal distribution to the IPCC's probabilistic statement, with a vertical line to show DICE's default climate sensitivity value. The bottom panel should show the distribution of present-day social cost of carbon values that you obtained from your Monte Carlo experiment, with vertical lines showing the mean of these values and the social cost of carbon value from optimizing DICE with the default parameter values.

Questions

1. The mean social cost of carbon from your Monte Carlo experiment reflects an estimate of the social cost of carbon given our remaining uncertainty in climate sensitivity. Is this value higher or lower than the social cost of carbon value from optimizing DICE with the base parameters? How much higher or lower is it? Express your answer as a percentage.
2. How sensitive is your answer to question 1 to the likelihoods you assign to "extremely unlikely" and "very unlikely," given the likelihood ranges specified for these terms by Mastrandrea et al (2010, their Table 1)? Does one bound have more of an effect on the mean social cost of carbon from your Monte Carlo experiments than the other?
3. If we are uncertain about the actual value of climate sensitivity, should we spend more or less money on reducing carbon dioxide emissions now, compared to a case in which we are sure about the value of climate sensitivity? Justify your answer based on your responses to questions 1 and 2.

References

Alexander, L. V., et al., 2013. Summary for policymakers. In Stocker, T. F., et al., eds., Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press. Available online at http://www.climatechange2013.org/report/full-report/

Interagency Working Group on Social Cost of Carbon, 2010. Technical Support Document: Social Cost of Carbon for Regulatory Impact Analysis Under Executive Order 12866. Available online at www.epa.gov/otaq/climate/regulations/scc-tsd.pdf

Interagency Working Group on Social Cost of Carbon, 2013. Technical Support Document: Technical Update of the Social Cost of Carbon for Regulatory Impact Analysis Under Executive Order 12866. Available online at https://www.whitehouse.gov/sites/default/files/omb/inforeg/social_cost_of_carbon_for_ria_2013_update.pdf

Johnson, L. T., and Hope, C., 2012. The social cost of carbon in U.S. regulatory impact analyses: an introduction and critique. Journal of Environmental Studies and Science 2, 205-221.

Mastrandrea, M. D., et al., 2010. Guidance Note for Lead Authors of the IPCC Fifth Assessment Report on Consistent Treatment of Uncertainties. Available online at https://www.ipcc.ch/pdf/supporting-material/uncertainty-guidance-note.pdf

Nordhaus, W., 2013. The Climate Casino: Risk, Uncertainty, and Economics for a Warming World. Yale University Press, 378 p.

Nordhaus, W., and Sztorc, P., 2013. DICE 2013R: Introduction and User's Manual (2nd ed). Available online at http://www.econ.yale.edu/~nordhaus/homepage/documents/DICE_Manual_103113r2.pdf