Item Response Theory with R

A Quick-Start Introduction to Modeling, Simulation, and Best Practices

David Fikis
dfikis1@gsu.edu
November 2014

Table Contents

Foreword........ 2

Preparation........ 3

System Requirements........ 3

Installation........ 3

Step 1: Install VirtualBox........ 3

Step 2: Acquire Debian's Install Disk Image........ 3

Step 3: Create a Debian Virtual Machine........ 3

Verifying Installation........ 11

Features and Notes........ 13

Saving a Session........ 13

Exporting a Virtual Machine........ 14

Finding Shared Folders........ 15

User Interface Details........ 17

Presentation and Practice........ 19

Review of Formulae........ 19

Best Practices........ 20

Session 0: Basic R Tasks........ 22

Session 1: Excel HW1........ 36

Session 2: Excel HW2........ 40

Session 3: Excel HW3........ 42

Session 4: Excel HW4........ 47

Session 5: Excel HW5........ 49

Session 6: Excel HW6........ 52

Evaluation and Expansion........ 57

Table of Figures........ 58

Table of Code........ 60

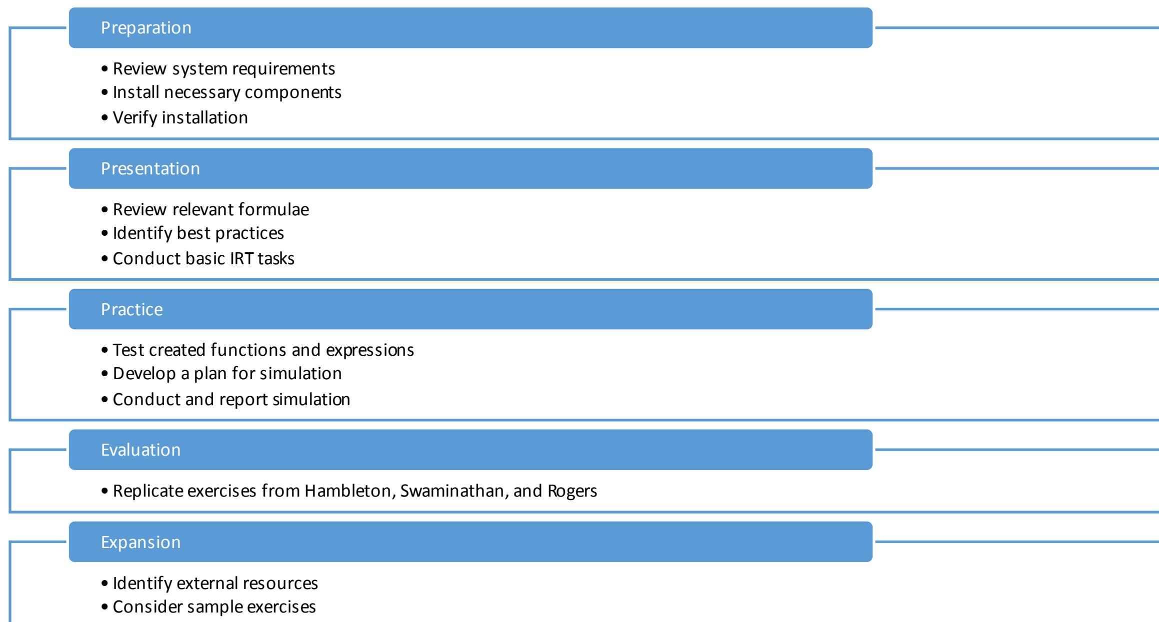
Foreword

This guide is intended to offer the student of Item Response Theory some basic instruction and operational knowledge sufficient to use the R statistical software package to complete introductory exercises. Like any statistical endeavor, this small guide operates under a number of assumptions. The validity of its operations and scope depend on the validity of said assumptions, and enumerating them briefly helps the reader determine whether this guide matches the needs of the end-user. They are:

- No single body of instruction can be both sound and complete
- A theoretical understanding of tools used improves their use
- Problem-solving perspectives aid in conceptualizing tasks

The pedagogical approach of the guide is based on the content matter. R, like most statistical software, has its own interface and syntax; interacting with this syntax forms the vast majority of an R experience. Programming "languages" are just that – languages. This guide assumes some basic knowledge of IRT and approaches the exercise of using R as that of a second language. Second-language acquisition, particularly academic languages, benefit from ordered approaches, because such languages are often *procedural* in addition to *declarative* exercises (Chamot and O'Malley 1987, 232). The following broad categories and objectives describe the path of activities to follow in this guide:

Figure 1: Objectives



Preparation

System Requirements

Before getting into the details of using R for IRT, R will first need to be installed on the computer to be used for the process. R will run on machines that wouldn't run SAS or SPSS. R also boasts a particular advantage among comparable software, detailed below:

Figure 2: Prices of Software

	SPSS	SAS	R
License Type	1-year	1-year	GPL-2
Price	>$5,000	>$5,000	$0

Considering that most student licenses under which we can acquire SPSS or SAS often forbid using them for publications, R stands out as a very advantageous choice in terms of price. This advantage, however, is not without opportunity costs: as you will see, the R experience is profoundly different from the typical statistical software package. These differences, once appreciated, add to the virtue of R in academic settings. For instance, the open source nature of R allows for publication of new libraries and inspection of existing ones. While the initial lack of a user interface is intimidating to those who are used to command-line interfaces, the resulting programmability and portability of code is very convenient.

Installation

One of the most important but least-considered properties of a statistical software package during learning time is portability. If you have to work on another computer, how long will it take? Are you able to count on consistent performance in six months? In a year? R is a command-line environment at its heart, and within that command-line paradigm are a plethora of free, high-quality pieces of software.

Although it takes a bit more work up-front, creating a *virtual machine* for your R work allows you to have a completely portable and self-contained environment. Much of the documentation for R is written from the perspective of a Linux user, and if you're not interested in reformatting your computer, rebooting often, and other fun tasks, a virtual machine is how you can have your cake and eat it, too. Cake isn't cheap, though, and virtual machines do have some overhead. I would not recommend this on sub-par laptops, but most desktop machines should be capable of handling the burden.

Step 1: Install VirtualBox

Head to https://www.virtualbox.org/ and visit the Downloads section. Download the "Virtualbox platform package" for your operating system and install it like you would any application.

Step 2: Acquire Debian's Install Disk Image

Now go to https://www.debian.org/ and visit the Net Install sub-section of the Getting Debian Section. Select the i386 architecture and wait for the .ISO file to download.

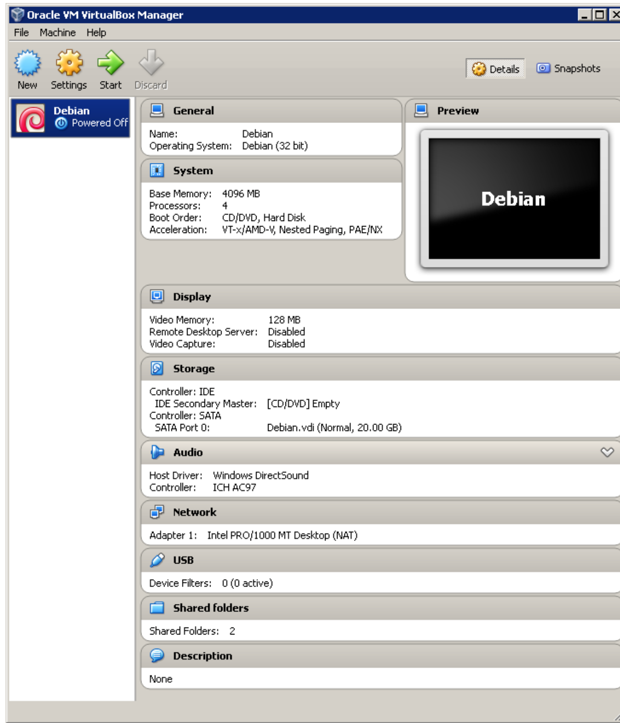
Step 3: Create a Debian Virtual Machine

This is one of the more complicated parts of the guide, but the results will be worth it. A virtual computer-within-a-computer will be created to house all of our work in a portable and consistent way; performance should be consistent over various operating systems, and the machine offers some nice conveniences for workflow that will be discussed later. We will make some decisions about configuration that will be briefly, but not fully, explained.

Create a New Virtual Machine

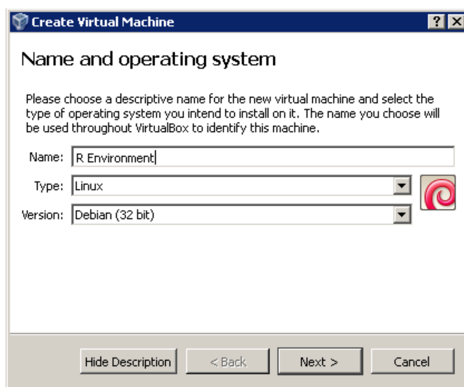
Begin by opening the VirtualBox program if you have not already. With different versions and operating systems, there's a chance that your screen may appear different, but the general preview of the main windows is depicted below. Click on the "New" button near the top left-hand portion of the screen or click on the "Machine" menu and select the "New" option there.

Figure 3: VirtualBox Main Window



The next steps involve proceeding through the new machine creation dialogs. The machine will need a name, in this case the name "R Environment" is chosen. Then, for aesthetic purposes, we can set basic properties of the machine: set the type to "Linux," then find "Debian (32-bit)" in the version drop-down menu. The result is shown below: note the red spiral, a Debian trademark. Click "Next" when done.

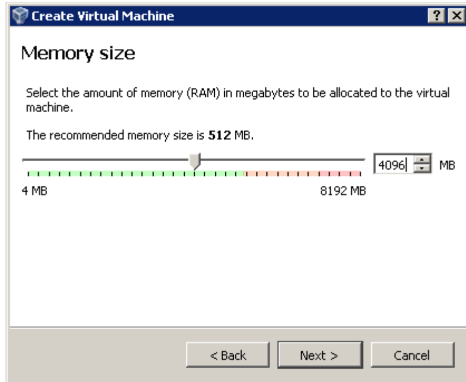
Figure 4: Naming the New Machine



Next, determine how much RAM the Virtual Machine will be told it has. More is never a bad thing, unless you jeopardize your host system's ability to function by allocating too much. For the purposes of

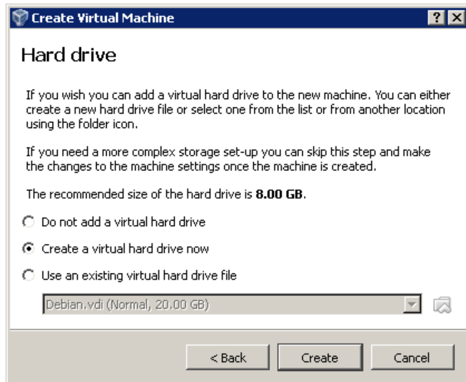
this guide, use a relatively safe amount of 4096MB, shown below. If this appears to be too much on your system, indicated by the red part of the slider bar, drop back to 2048MB. These sizes are 4 and 2 gigabytes, respectably. The result is shown below. Click “Next” when done.

Figure 5: Allocating RAM



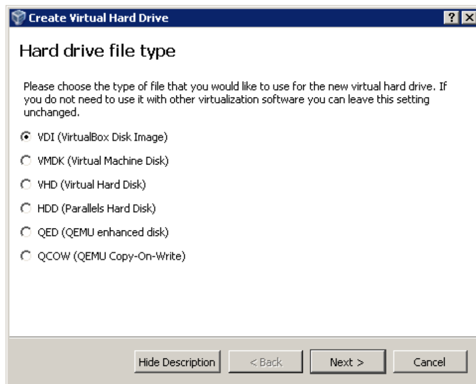
Next, a virtal hard drive will be made for the machine. The dialog has three options. Select “Create a virtual hard drive now” and click next:

Figure 6: Create a Virtual Hard Drive



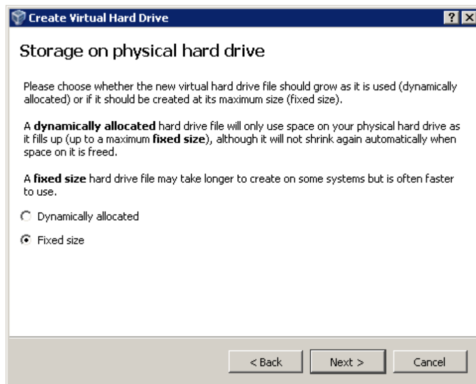
Select the first hard drive option, a “VDI (Virtual Disk Image)” and continue:

Figure 7: Virtual Disk Image Specification



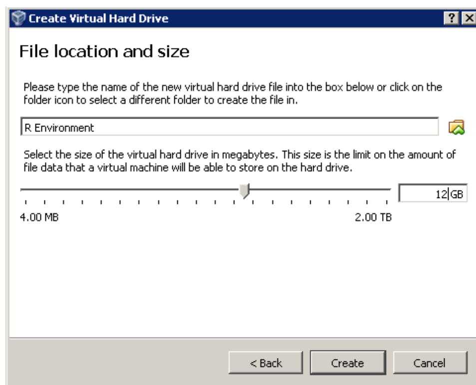
Select the "Fixed size" option. Although this will create more overhead on the host computer's hard drive at first, the performance benefits will help later:

Figure 8: Fixed Disk Size



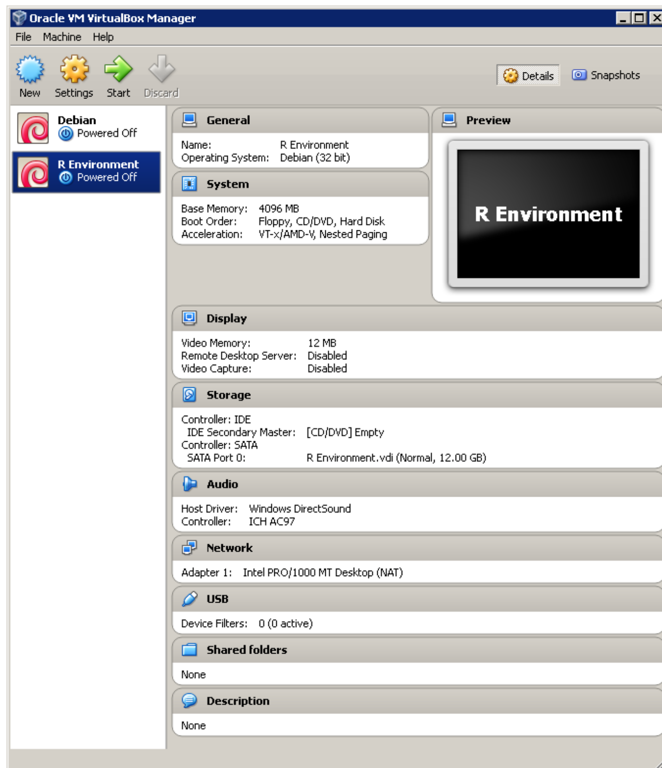
Set the size of the virtual disk. For this guide, 12 gigabytes will provide plenty of overhead for any additional software or file manipulation we might expect. Correcting for insufficient disk space is a headache, so it's generally a good idea to overestimate when it isn't costly to do so. If you know that your computer has two separate, physical hard drives, you may want to consider clicking on the folder icon and relocating the virtual disk to a secondary hard drive for significant performance enhancement:

Figure 9: Disk Size to 12GB



Once this process finishes – it may take a few minutes – your virtual machine is built! Of course, it has no software and will do nothing useful just yet, but you should be able to see it in the main window of the VirtualBox program now.

Figure 10: Confirm VM Creation

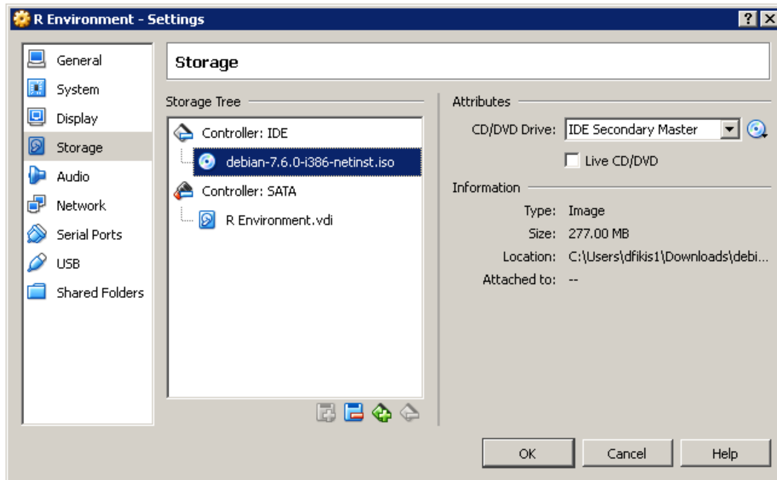


Install Software

Now that our machine is built, we need to install the base Operating System as well as R. We will also install a few utilities that will come in handy later on. Although some basic wizards exist, there's a chance that you might have accidentally already run your machine, so we'll go through it manually.

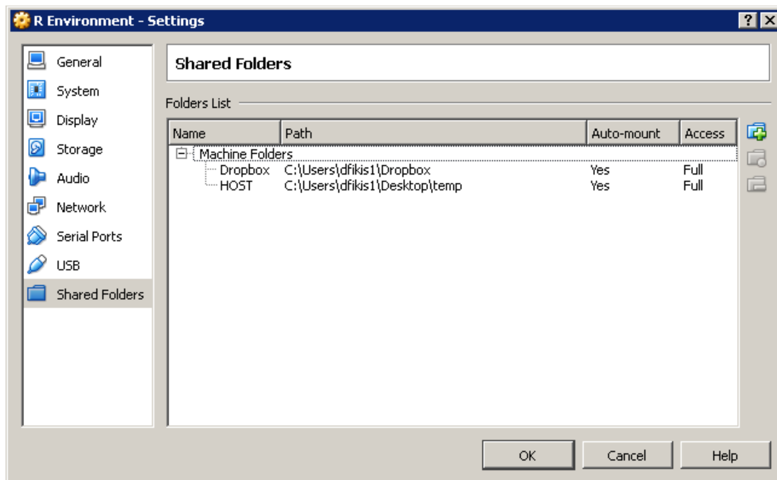
Right-click on the "R Environment" list-item and select the "Settings" option in the context menu. From there, select the "Storage" option. Select the CD-rom looking icon under the "Controller: IDE" list-item, then click on the CD and arrow icon on the right edge of the dialog window to select the "Choose a Virtual CD/DVD disk file" option. Find the Debian installation .ISO file downloaded earlier in this guide, and select it. The result is shown below.

Figure 11: Mounting a CD-ROM Image



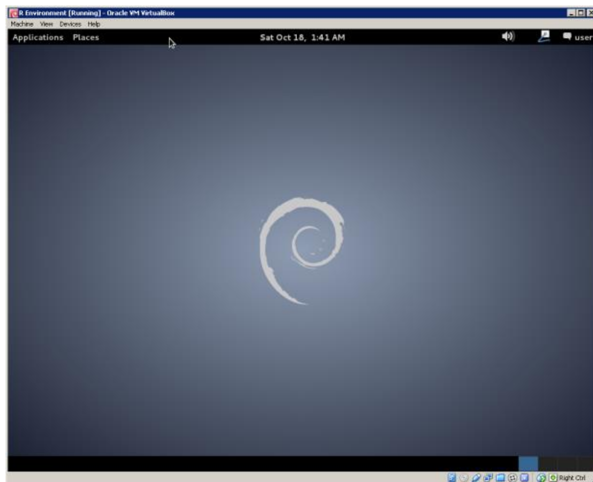
While we're in the settings menu, let's configure a Shared Folder. This will make moving files around much easier between our Virtual Machine and our real computer. Select the "Shared Folder" item on the left list, and click the little icon with a folder and green plus sign to add a shared folder. Select the folder path like you would when selecting any directory: I usually go for a sub-folder off of the Desktop for convenience. Give it an easy to remember name. In the example, we'll call it "HOST." Leave the read only option unchecked. Check the auto-mount option. At this time, you may also want to add a shared folder for Dropbox if you are a Dropbox user. Although we could install Dropbox on the virtual machine, it is more efficient to access the files via your real computer's directories. Click "Ok" when done to close the settings dialog entirely. The results are shown below.

Figure 12: Shared Folders



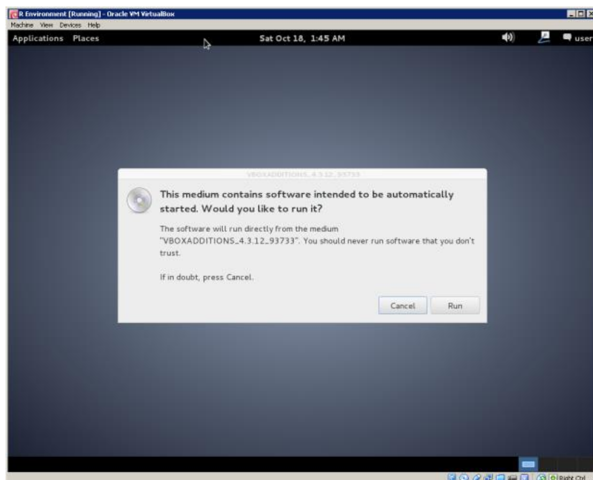
We're now ready to turn on the Virtual Machine. This will boot the machine through the operating system installation, which may change over time. Generally speaking, the options can be left with all their default values. Choose a hostname, account name, and password that are unique, but unused elsewhere. Once the installation completes, you will be able to login. It will look not unlike the figure below. More guidance on Operating System installation can be found at http://www.debian.org/.

Figure 13: First Login



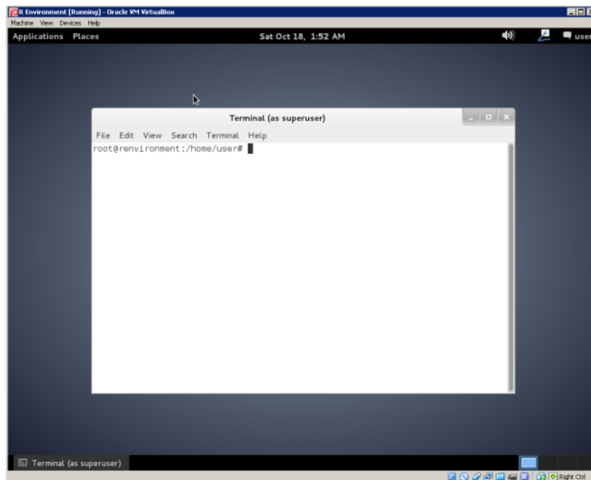
The first task now is to prepare the machine to with some add-ons to make it run better. First, let's attach the Guest Additions CD-ROM by selecting the last option, "Install Guest Additions Image…" from the Device menu. A pop-up dialog to confirm using Auto-Run will appear. Click cancel – we are not going to do this the automatic way because of a bug in the existing setup. In the future, this bug might get patched and this section of instructions may be obsolete. The dialog is pictured below.

Figure 14: The Guest Additions Auto-Run Dialog



From the Applications menu, select the Accessories option, then the Root Terminal application. You will need to enter your password. From there, you will be in an administrator command prompt where we can install all the software we will need. The result will look like the figure below.

Figure 15: Root Command Prompt



Within this command prompt, enter the following command (note that if you're using a different architecture than what we chose earlier, you may have to specify a different package) :

Code 1: Installing Kernel Headers for the i486 kernel

```
apt-get install linux-headers-i486
```

Choose "Y" or "yes" for a prompt if it confirms installing options. Then wait a while. Some errors about "DKMS" and "modules" and versions may display. These are expected. Now we will run the Guest Additions Setup from the "CD-ROM" we used earlier. This will take more than one line of code, and you may have to confirm installation over existing versions.

Code 2: Installing VirtualBox Guest Additions

```
cd /media/cdrom
sh VBoxLinuxAdditions.run
```

While in this root prompt, let's install the R components we will need to get right into things alongside a handy text editor called geany. For this line, you will probably have to confirm the installation of additional, required packages. That is okay. Enter the code below to get everything started:

Code 3: Install the R Base System, development files, and a text editor

```
apt-get install r-base r-base-dev geany
```

We also need to patch just a couple things pertaining to shared folders, assuming a username of "user":

Code 4: Adding User to the Shared Folders groups

```
usermod -a -G fuse user
usermod -a -G vboxsf user
```

When that's done, the safest thing to do now is to reboot the system to confirm that the bug is fixed:

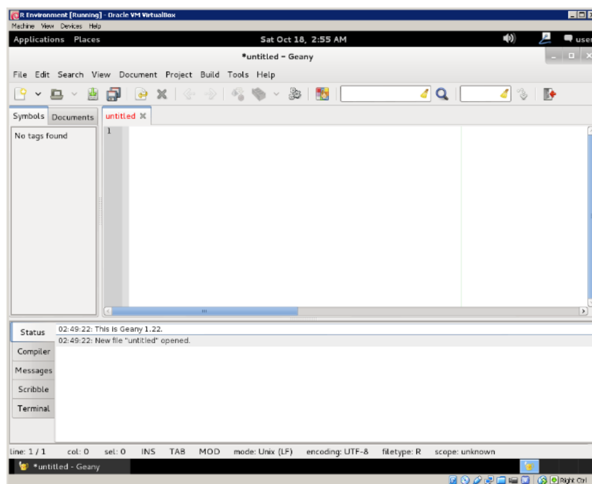
Code 5: Restart the Machine

```
reboot
```

Verifying Installation

When the machine reboots and you log in, click the Applications menu. Under the "Programming" sub-menu, select "geany" to open our IDE (Integrated Development Environment.) Geany is one of countless programs whose mission statement is to bundle together many tasks programmers (such as you are about to be) need to work efficiently. We will be using the IDE approach to honor one of the best practices in our field: saving our syntax! The application will look like the figure below, but be warned that future versions may have slightly different defaults and themes.

Figure 16: The geany IDE Initial View



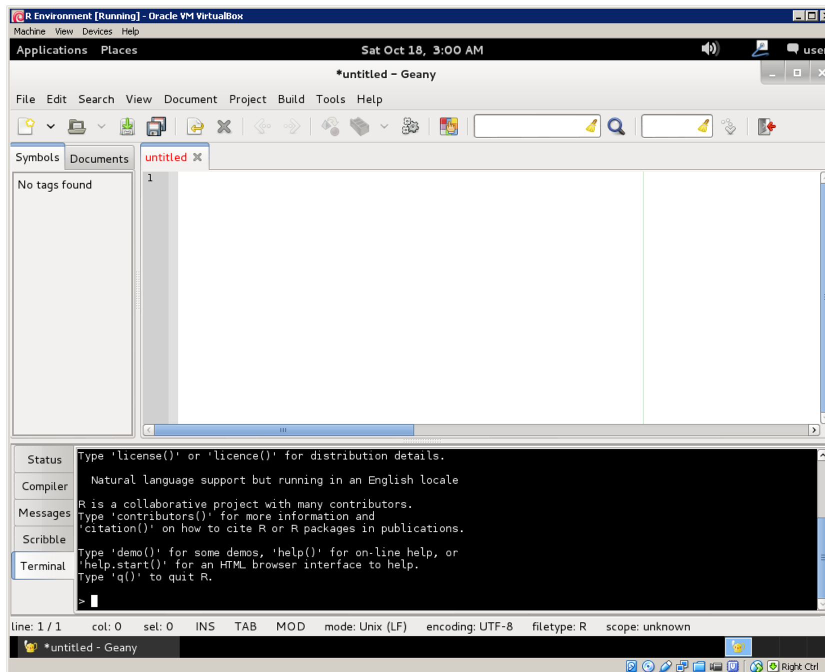
We're going to change a couple things about this initial view to better suit our purpose. First, near the bottom-left hand screen is a "Terminal" tab. Click it. You should recognize what happens – we have already been inside a terminal. Within this terminal, enter the following code:

Code 6: Starting R inside a Terminal

```
R --vanilla
```

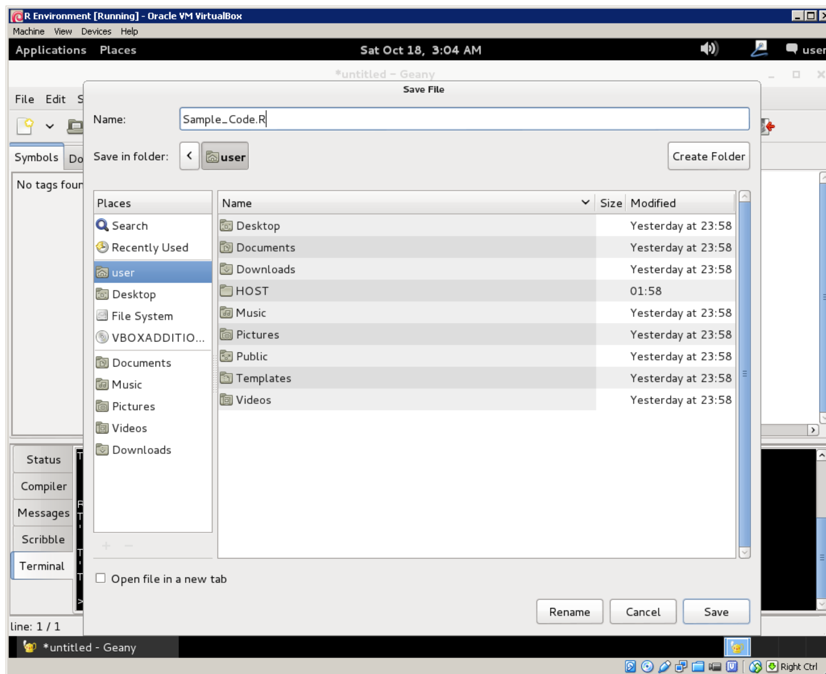
The "--vanilla" option tells R to open a fresh, plain instance and to not save any changes made to the workspace. This will help maintain consistency, but more advanced users may eventually find that the vanilla option no longer satisfies their needs. If the command succeeds, you will see a new kind of prompt along with some of the basic R startup information. It will look like the figure below.

Figure 17: Geany IDE with R



If that worked, we can continue to implement some best practices. The next two steps relate to the text document portion of our IDE. First, click on the Document menu. Then select the Set Filetype sub-menu. Select the Programming Languages sub-menu. Finish by selecting the "R Source" option. This will help add a bit of color to our code to make it easier on the eyes. Now we need to save our file. Just click the File menu, then the "Save As" option like normal. Select the Home option on the left to save in your home directory. Give the file whatever name you like – conventionally, these types of files end in the .R extension. The resulting dialog window is pictured blow. Confirm saving when done and remember, going forward, to select File -> Save frequently.

Figure 18: Saving R Code

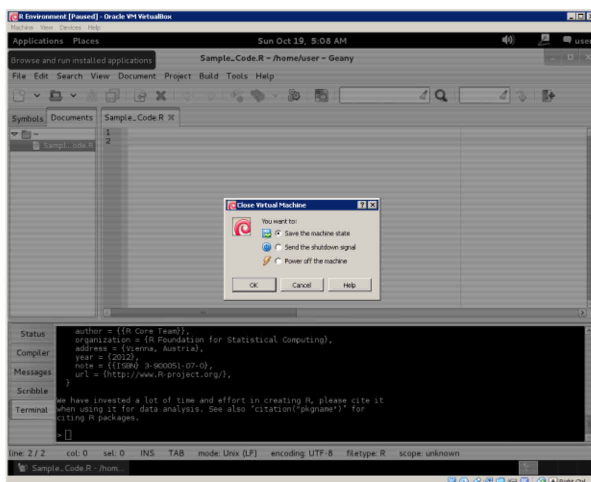


Features and Notes

Saving a Session

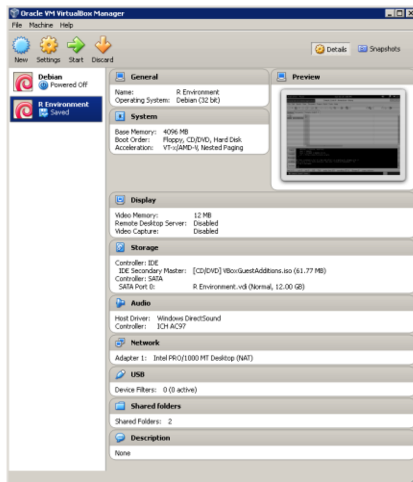
VirtualBox offers a tool that makes it easy to stop, set aside, and resume work within your IDE. From the Machine menu in VirtualBox, select the Close option. Choices will appear. If you select "save the machine state," as shown below, your session will be saved not unlike if you were to hibernate your computer. You will be able to resume working without having to restart applications.

Figure 19: Saving the Machine State



Once the state is saved, the machine will boot up right where it was left off as if one had suspended it. In the main VirtualBox window, the option to start the machine will be present alongside the option to "discard" the saved state and treat the machine as if it had been unplugged and whatever suspended information had been retained would then be cleared from the machine.

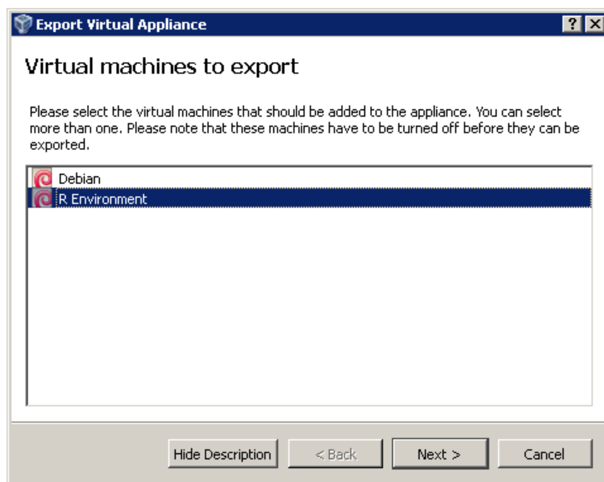
Figure 20: Saved Machine State



Exporting a Virtual Machine

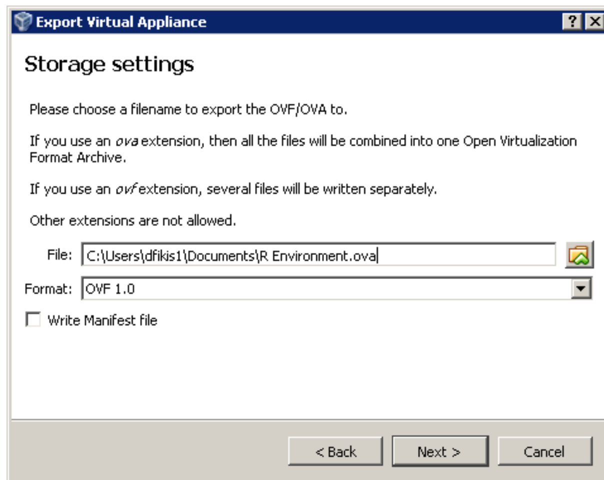
One of the fundamental advantages of using a virtual machine for statistics work is the portability of the environment. Suppose a workstation was getting upgraded. Under the usual operating conditions, this would mean hours of re-installing and re-licensing statistical software packages. With a virtual machine and R, all we need to do is export the virtual machine and import it into a reinstalled VirtualBox program. This can work between computers as well. From the main VirtualBox window, click on the File menu, then select the “Export Appliance” option. A dialog will open to allow you to select which machine to export.

Figure 21: Selecting a Machine for Export



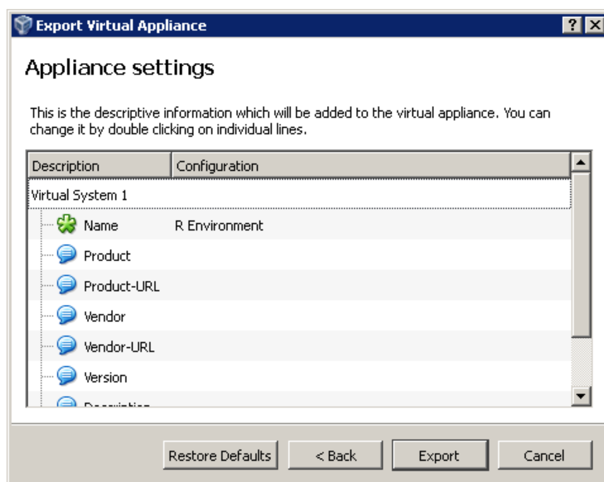
Afterwards, confirmation of the location and format for the exported file will be displayed. The options can be left at their defaults: using a .OVA extension, as the dialog details, keeps the exported file simple to manage.

Figure 22: Selecting Export Options



An additional window concerning metadata will appear. All values can be left at their defaults:

Figure 23: Reviewing Export Metadata

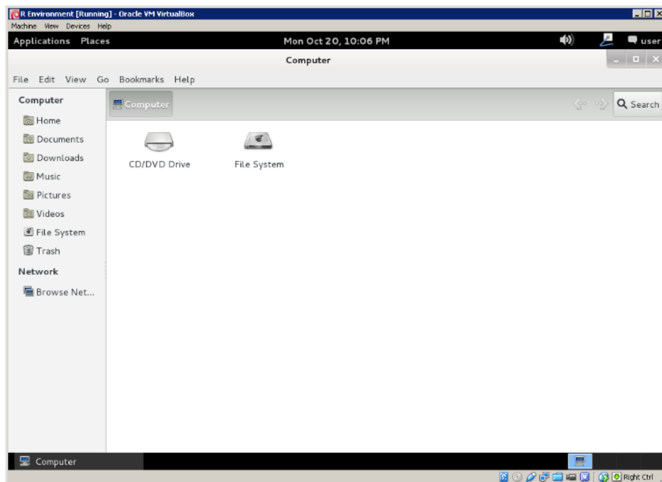


A progress bar will be displayed as the export procedure executes. It will take several minutes as the virtual hard disk is compressed to save on file size. The resulting file can be saved on any medium as a backup of the virtual machine. To restore, select the “Import Appliance” option from the file menu.

Finding Shared Folders

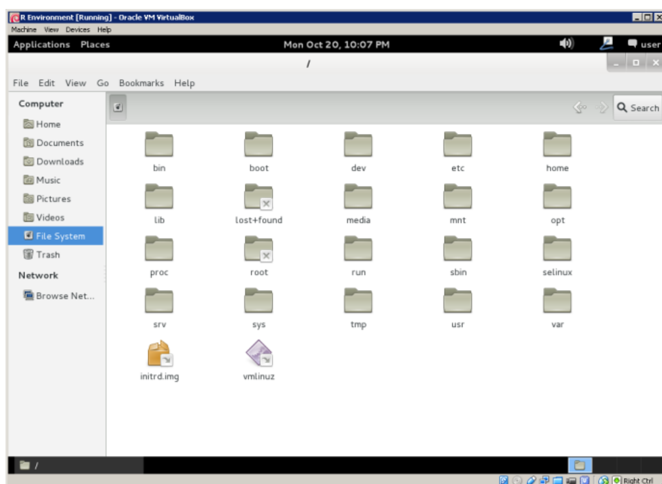
If the “Auto-Mount” option was enabled as earlier and the proper user group fixes were applied, then shared folders can be found on the virtual machine inside the “/media” directory. To find these files, from the desktop select the “Places” menu, then the “Computer” option to open the file browser.

Figure 24: File Browser Main Window



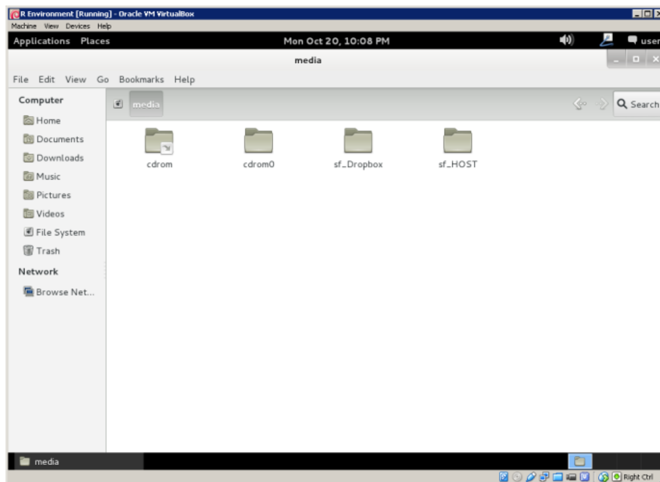
Select the “File System” option on the left to view the root filesystem:

Figure 25: Root FileSystem



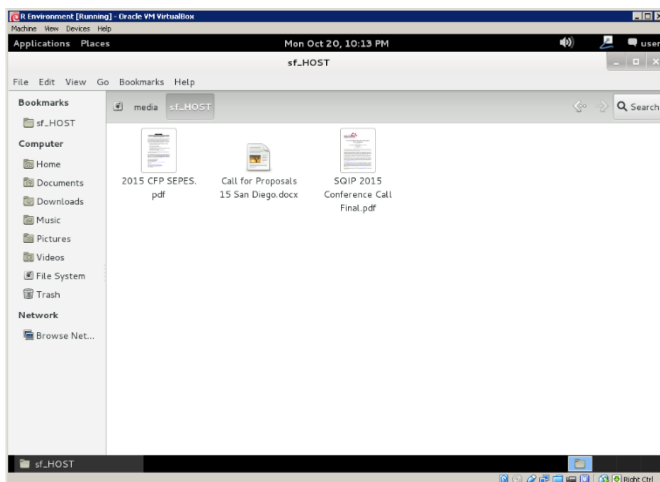
Then double-click on the “Media” file-folder icon to open the media folder. Inside, you will be able to see the automatically attached shared folders leading to your host computer. In the example below, our “Host” and “Dropbox” folders are visible.

Figure 26: Media Folder



This might not be convenient to do every time, so let's create a "bookmark" to this location. After opening the folder you would like to create a link to, click on the "Bookmark" menu, then select "Add Bookmark." The current directory will be added to the convenient list of clickable tabs on the left.

Figure 27: Bookmarked Directory



User Interface Details

Before closing out this section, there are a few interesting properties of the user interface to note. Some readers may be used to using Control+S to save files and the Control+C, Control+V method of cut and paste. In our machine as configured, the right control key is the "host" key. This key is used to send commands to VirtualBox itself rather than to the machine it is emulating. For example, HOSTKEY+F will make the virtual machine toggle between fullscreen modes. Remember to use the *left* control key for copy/pasting inside VirtualBox.

Copying text to and from windows inside the Debian environment is extremely easy if your mouse has a middle (scrolling) button. All you have to do is click and drag to select text from one location, left-click once at the destination, and click the middle mouse button. This is called the *primary buffer* and is incredibly convenient once you get the hang of it.

We will be going forward with the concept of constantly saving syntax in executable files. Possibilities abound, however, so do not feel obligated to use this method forever. More advanced users may be interested in learning about a concept called *version control*.[1]

Before continuing, take some time to play with the IDE a bit. Take risks and enter some commands into the R terminal. Try changing the sizes of various parts of the window to suit your taste. Explore some of the features by creating new documents and trying out the “Documents” tab next to the “Symbols” tab near the top-left part of the screen. Every programmer has their own tastes for an IDE. There are cult followings, jokes, and even intense newsgroup arguments on the subject! Don’t be afraid to experiment and find your own visually appealing style before diving in further.

[1] For a great overview, see Hartl, M. (2013). *Ruby on Rails tutorial: Learn Web developments with Rails*. Upper Saddle River, NJ: Addison-Wesley. pp. 27-34

Presentation and Practice

Review of Formulae

In Item Response Theory, normally-distributed latent traits often referred to as *abilities* influence the probability of correctly responding to an item according to the following 3-parameter formula.

Equation 1: 3-Parameter Item Response Model

$$P(\theta) = c + \frac{1 - c}{1 + e^{-1.7a(\theta - b)}}$$

The equation represents the probability of any one dichotomous item being answered correctly, and the graph appears as a logistic (or sigmoid) function with ranges asymptotic of c and 1. Details of the role of each variable are better discussed in course texts, but in short: *a* is a scaled discrimination factor, *b* is a difficulty parameter, and *c* is a *guessing* parameter. The 1.7 is sometimes written as *D*, and it is a scaling factor meant to make the model more closely represent the cumulative distribution function. To reduce the model to the 2-parameter model, fix *c* at zero. To further reduce the 2-parameter to the 1-parameter model, fix *a* at one.

Other equations will be used in this guide to fit models. The most basic is Q, the probability of one dichotomous item being answered incorrectly:

Equation 2: Probability of Incorrect Response Q

$$Q(\theta) = 1 - P(\theta)$$

Assuming local independence, probabilities are cumulative. The probability of any one set of responses is the product of all component scores. In plain language, if correct use P, but if false use Q, and find the total product of these components. The likelihood function can be used to estimate θ for a given set of responses when the item parameters are known. Often, the logistic function is maximized: this is sometimes called *log-likelihood* estimation. Although logic can be used, the formula can also be expressed algebraically as below, assuming *U* is the response of a dichotomous item.

Equation 3: Likelihood Function

$$L(\theta) = \prod_{i=1}^{n} P_i^{U_i} Q_i^{(1-U_i)}$$

When estimating parameters and abilities at the same time, the calculations are more complicated:

Equation 4: Likelihood Function with Unknown Abilities

$$L(u_N | \theta, a, b, c) = \prod_{i=1}^{N} \prod_{j=1}^{n} P_{ij}^{u_{ij}} Q_{ij}^{(1-u_{ij})}$$

This method of estimation would be indeterminate if ability levels were not fixed in some way: the number of estimated parameters is equal to three times the number of items (each parameter for the item) plus the number of respondents (abilities.) Usually, abilities are scaled to a standard normal distribution to avoid indeterminacy, creating a two-stage estimation process. In the first stage, abilities are estimated initially based on score transformations, and then item parameters are estimated based on the said ability estimates. In the second stage, abilities are estimated using the item parameters derived from the first stage. These steps are repeated until changes in estimates are negligible. This

method is referred to as *joint maximum likelihood estimation*. The join maximum likelihood estimation method has several disadvantages discussed in more detail in the literature alongside other alternatives such as Bayesian estimation methods and the use of integration.[2]

Best Practices

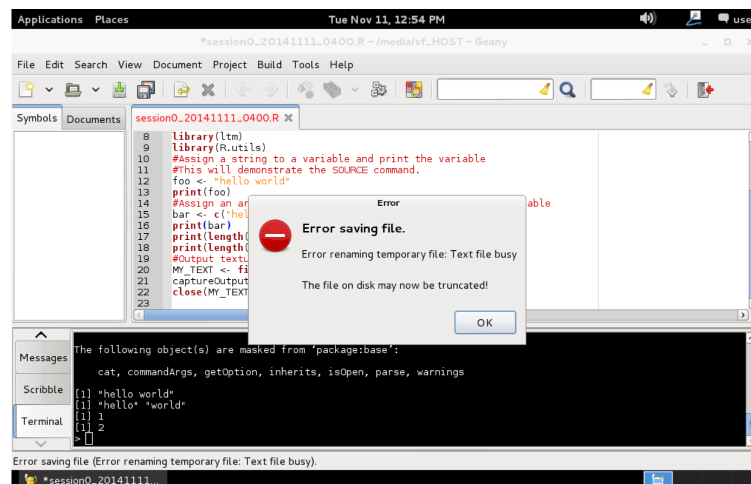
Save Syntax: Stay in Syntax

Retaining and operating from a syntax file is a best practice whenever it is available, although basic data exploration and learning exercises may not immediately benefit from the practice. Programs such as IRTPRO and SAS have their own proprietary file formats and specifications to save analyses, and R is no different. One advantage with R is that syntax is saved in plaintext documents requiring no special formatting; these files are highly portable and can aid in adjusting or replicating work later.

In this guide, syntax will be prepared in a separate file that we reference in R: rather than directly entering a series of commands, we will instead build a "script" that will be evaluated all at once. This will more closely allow us to practice using R as we might find ourselves using it in the field. We will use the *source* command in R, but there are multiple ways to execute pre-written R code, such as the *RScript* utility: more advanced users interested in automating code may be interested in that.

In practice, you may get an error when trying to save your R syntax to a shared folder reading something like, "error renaming temporary file: Text file busy." If this is the case, then the shared folder mechanism is not functioning at optimal speed to keep up with the file edits, like below:

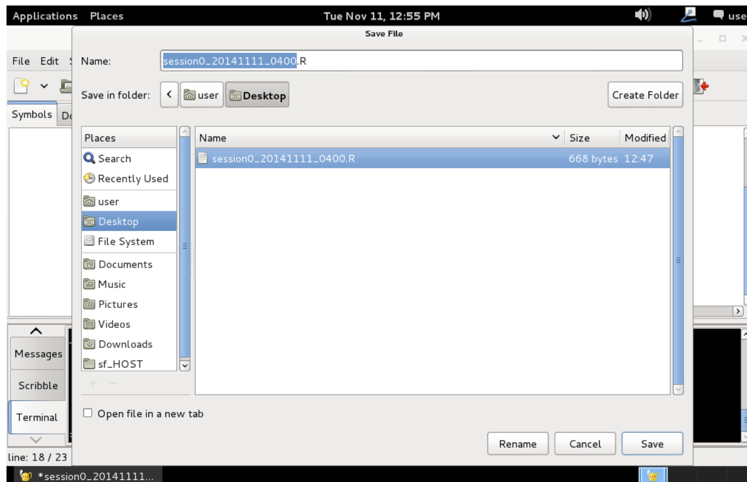
Figure 28: Text file busy error



The solution is to just re-save your files to the desktop folder of your virtual machine:

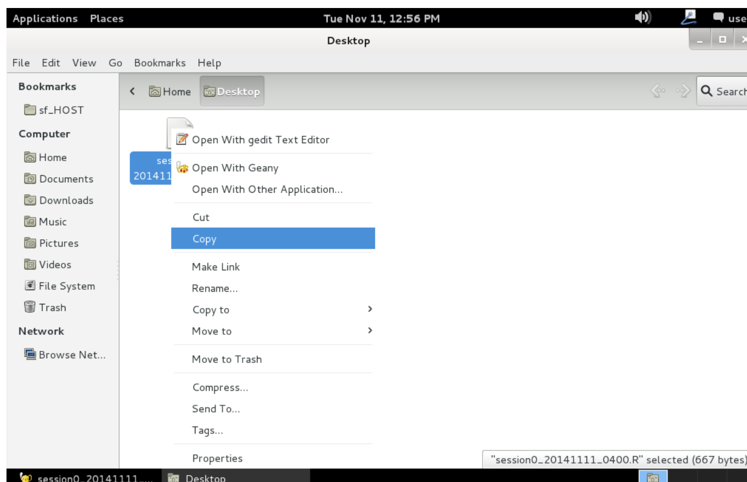
[2] See Hambleton, Swaminathan, and Rogers, Chapter 3, for further discussion

Figure 29: Saving to virtual machine desktop



You can then copy-paste from the desktop to the shared folder using the Places menu at the very top of the screen, much like you might do in your normal computing environment, to select the desktop and then right click to copy the file:

Figure 30: Copying a file



R Prefers Vectors

R is built on a framework that offers great speed and readability of code based on a vector approach. In traditional program environments used in statistical analysis, we often make use of loops to iterate through replications or values in arrays. In R, however, we can often reference these arrays directly to take advantage of speed and flexibility. This is probably best explained with some illustrations. We will explore this during our first R practice session.

Style of Content

The main material will be presented in a series of independent *sessions* suitable to following along with the Excel and book-based work in the introductory item response theory course. By completing these sessions, the first major portion of coursework can be completed in R with no other required software.

Session 0: Basic R Tasks

Objectives

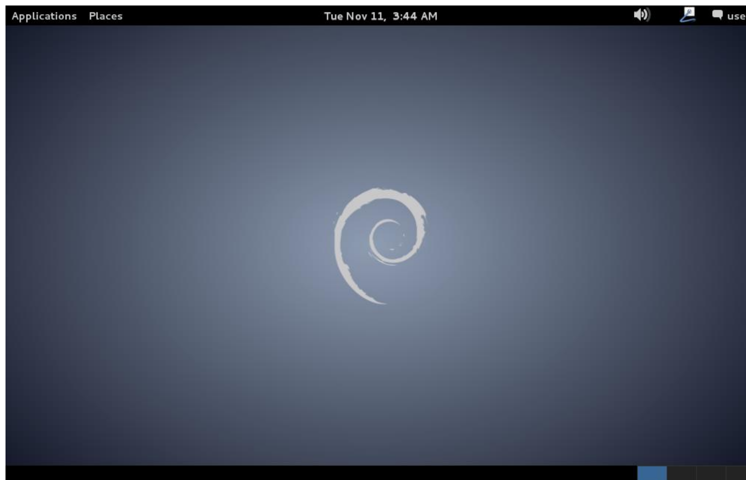
In this session, we will:

- Confirm our IDE functions
- Briefly examine variables and functions
- Install and load an R add-on package
- Execute basic descriptive statistical operations
- Import some data
- Create a graph

Procedures

Let's boot up our Virtual Machine and log into our linux desktop. It should look something like this:

Figure 31: Debian Desktop



Then we open geany, our IDE, by selecting it from the Applications -> Programming menu:

Figure 32: Starting geany IDE

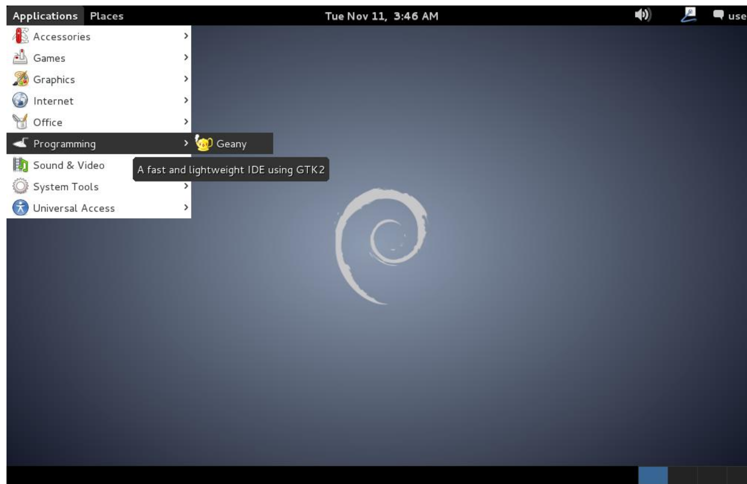
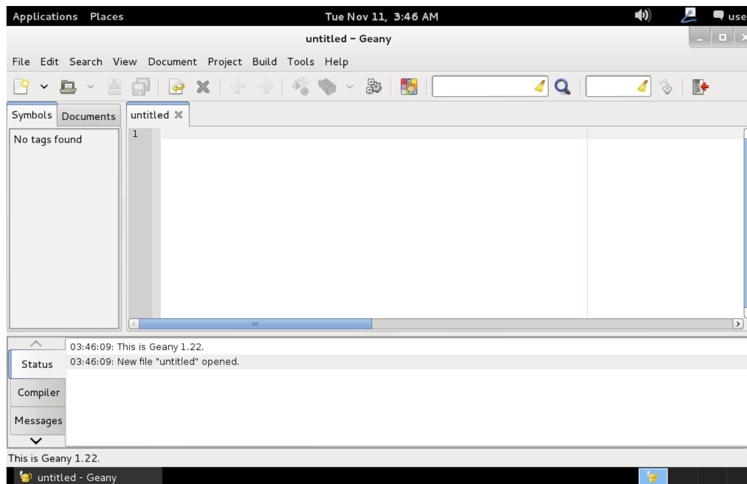
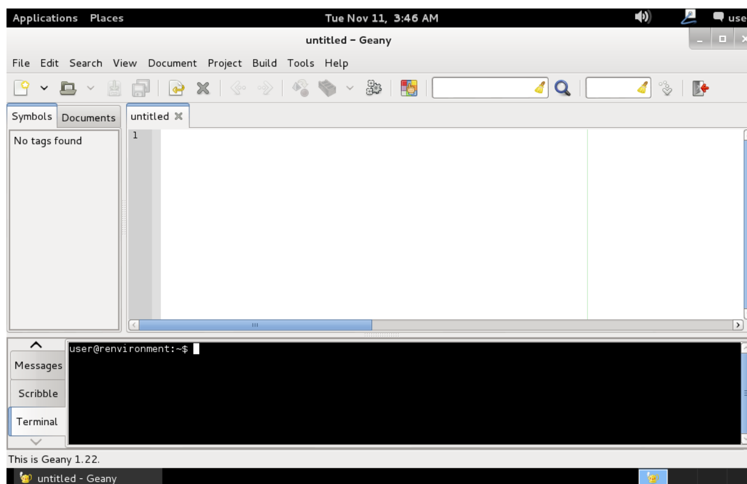


Figure 33: IDE Loaded



Using the down arrow at the bottom left of the interface, select the Terminal option:

Figure 34: Terminal View

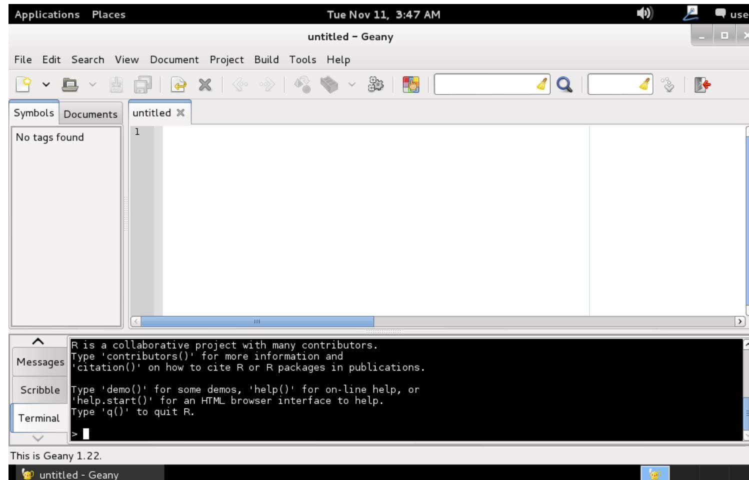


Being an R instance by issuing the command to open a “vanilla” R interface: using a vanilla interface will help us ensure that we develop code that is more likely to work from place-to-place.

Code 7: Starting R in vanilla mode

```
R --vanilla
```

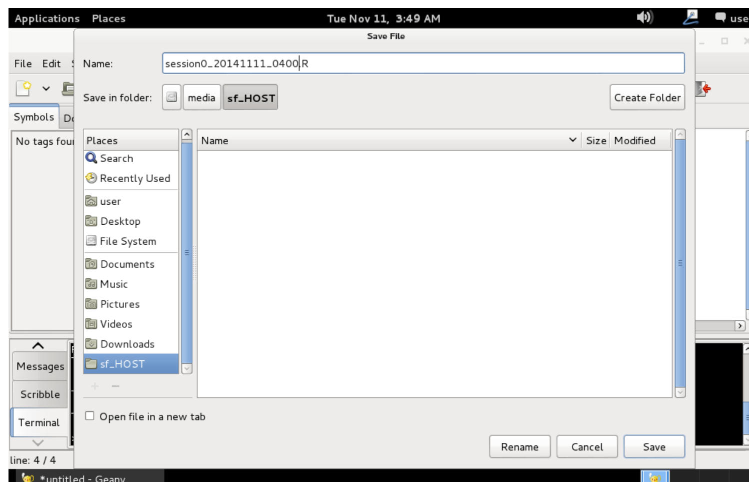
Figure 35: IDE with R Terminal



Now we can immediately begin some best practices: let’s save the blank text file so that we can start to keep track of our work. Generally speaking, there are many ways to name files. Personally, I find adding the date and time to files aids in tracking versions when other options aren’t available. More advanced users may wish to read about using a utility called *git* to keep track of code.

Below, we save the file to our “sf_HOST” folder we configured earlier. This will allow our syntax file to be accessed and maintained on our main computer, outside of the virtual machine. This can be useful for sharing and editing without having to boot up the machine; remember that this means the file will not be included in exports of the virtual machine, though. R syntax generally uses a .R extension, and using that will help our IDE interpret and highlight our code.

Figure 36: Saving a Syntax File



Let’s get out first lines written. In R syntax, the hash tag (#) is used to mark comments. Because we might be sharing this file, or we might simply forget what we were doing, some of the best things to include at the top of the file are a name, a last updated date, and a brief description of what the syntax does. For example, in our session 0 file, we could begin with:

Code 8: Session 0 Syntax

```
#R IRT Tutorial Session 0
#Last updated 11/11/2014
#Purpose: This syntax will complete the basic introductory tasks
```

As you type, you may notice that having saved the file with a .R extension enables syntax coloring. That’s normal and one of the handy features of a robust IDE. There are many features that more advanced users may enjoy exploring.

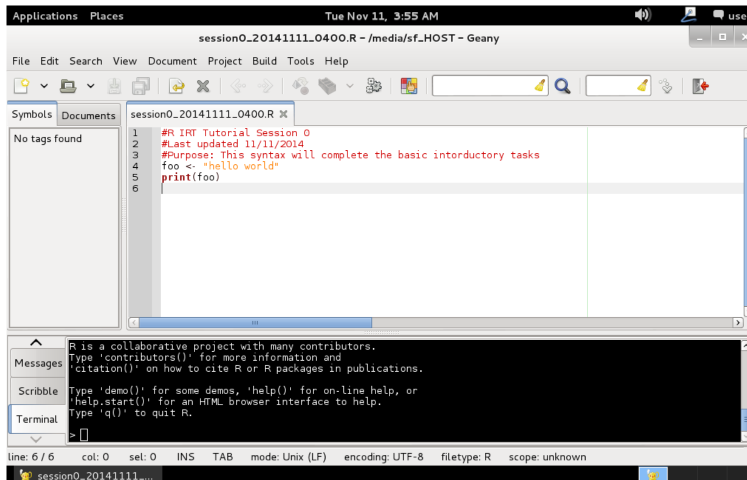
For now, let’s enter our first evaluable lines of code. R operates with named, case sensitive variables and function calls (that also, technically, are variables.) Although variable typing is a complex topic, let’s just dive right in with our first variable assignment and function calls:

Code 9: Session 0 Syntax

```
foo <- “hello world”
print(foo)
```

Before doing anything else, save your progress. Now is the time to build this good habit. In the first line, our variable name is on the left-hand side of a “<-“ operator, which can be thought of as an arrow. On the right-hand side, the quote-offset “hello world” acts as a string of characters. This string is then directed into the foo variable. Variable assignment can become very complex. The second line calls the “print” function: know functions by their use of parentheses. Inside parentheses, we are able to specify function parameters. Here, we just tell “print” what to print – the foo variable.

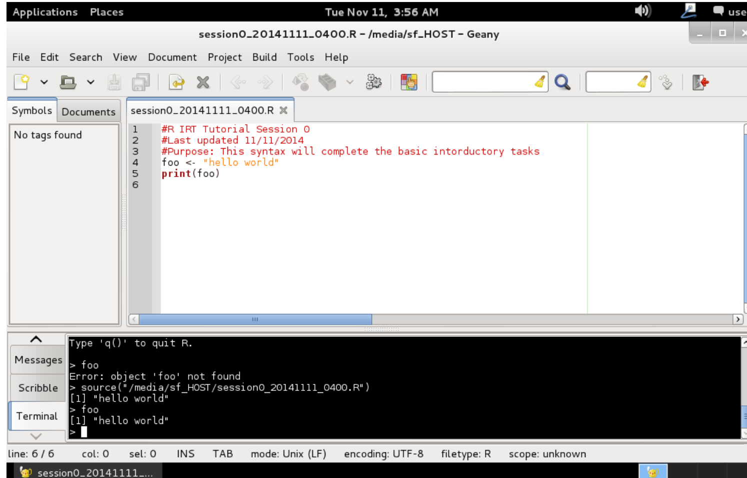
Figure 37: Code Highlighting after File Save



Let’s run our code. We’ll do that by clicking on the terminal window. Type “foo” to see what the “foo” variable currently is set to: you should see that it has no value. Then, call the “source” function. This source function will be how we call up our typed syntax. Inside the source function, we need to specify the file. We know the file name and location based on the title bar of our IDE. In the case of the example, it is “/media/sf_HOST/session0_20141111_0400.R” and, when entered, will evaluate. R has tab completion: you may want to try pressing tab while typing the file name to see it in action. After we

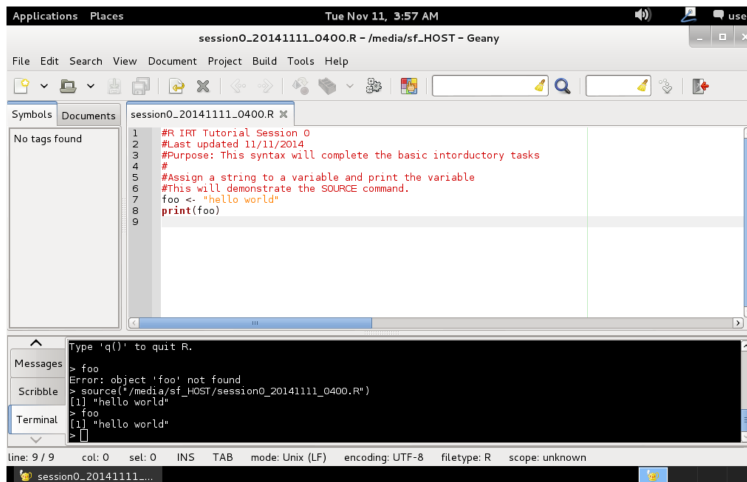
call our source function, we'll see the results – the print command printing the foo variable. Let's conclude this batch of code by entering "foo" again in the terminal: as we can see, we have made changes to the R workspace with our syntax.

Figure 38: Session 0 Code Output



Let's add a few more comments to explain what's happening. In complex, multi-step syntax such as simulations, it is important to document the code both for one's own memory and for sharing:

Figure 39: Adding some comments

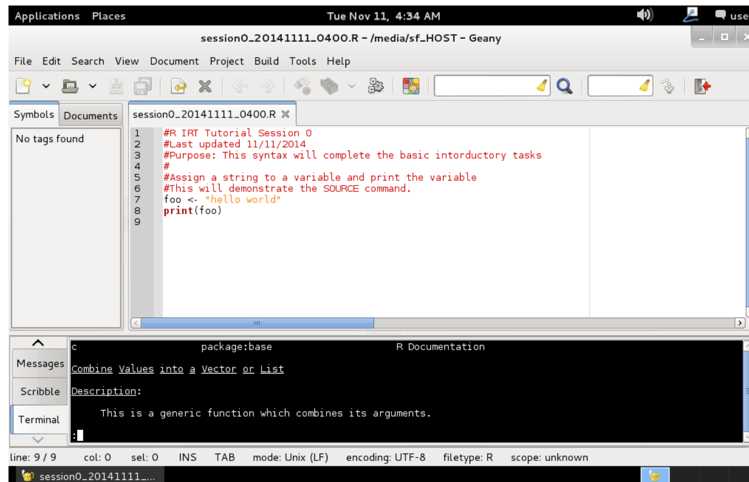


This is a good point to highlight a critical paradigm in R: within the R environment, nearly everything is conceived as a vector. When we view our "foo" variable, a "[1]" is visible. This is R telling us that foo is a n array with a length of 1 whose content happens to be "hello world." We can learn more about this at the same time as we can review one of the most important commands in R. Within the terminal, issue the help command by looking up the "c" function as follows:

Code 10: R help file access

```
?c
```

Figure 40: An R help file



You can use the up and down arrows to move around in the help file, and you can click and drag the divider to resize the terminal window. To exit the help file and return to the R terminal, just press the "q" key to "quit" the help file view. In the help file for the "c" function, we learn that it pastes all its arguments together: this function is one of the most convenient ways to enter arrays of information.

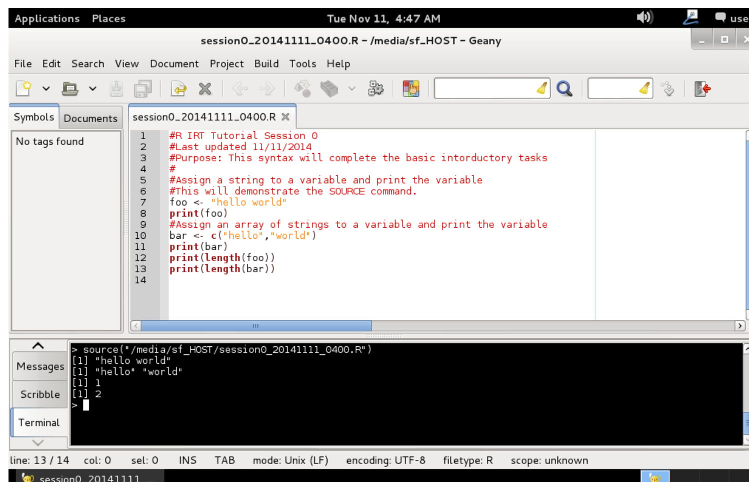
Let's edit our syntax a little to see what this means and looks like. Add the following lines:

Code 11: Session 0 Syntax

```
#Assign an array of strings to a variable and print the variable
#Then, examine the lengths of two variables
bar <- c("hello","world")
print(bar)
print(length(foo))
print(length(bar))
```

In the R terminal, you can press the left control key and L to "clear" the screen. You can also press the up arrow to scroll through the command history. Try using this to issue the "source" command again and observe the output:

Figure 41: Syntax output



We can see that the second variable, entered as a vector, has a length of two. We can also see that the length command's output itself is a vector with a length of one. Variables in R are most like the columns of a spreadsheet. If we cluster a number of these columns together, we can use a structure in R known as a *data frame*. A data frame is extremely useful and will be how we convey most information to and from R for IRT tasks. You can issue the "?data.frame" command to learn more from R's internal help files, but these discuss the topic in more depth that practical knowledge may require. We will discuss data frames more in a little while.

Before that, however, let's implement another set of best practices. While typing the "source" function in R, you may have noticed that the path to our file is typed out in full. In more advanced syntax, this can sometimes create a problem. What if, for example, someone is running a different R environment where the path to their shared folder is different? We can solve this problem before it starts by taking advantage of variables and functions to implement a best programming practice. Add the following line of code near the top of our syntax, just below the first batch of comments:

Code 12: Relative folder location setup

```
#Save our preferred path location to a variable
MY_PATH <- "/media/sf_HOST/"
```

To go more deeply into R and use if efficiently, we will need to install some add-on packages. R has a network of archives and packages that we can take advantage of called CRAN (Comprehensive R Archive Network.) There are indexes of packages, some 6000+ currently, sorted by both name and publication date. If you need something fancy, browsing this list is one of the ways to find it. Another is to use "Task Views" which contain digests and some annotations on packages. For example, the psychometrics task view contains a section specifically discussing item response theory packages.[3] Using that information alongside some of the knowledge we already know, let's install a few packages. Issue the following code inside the R terminal:

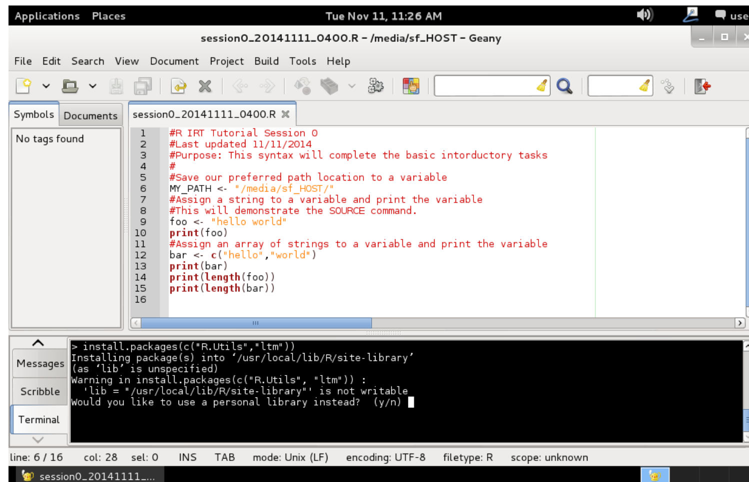
Code 13: Installing R add-ons via CRAN

```
install.packages(c("R.Utils","ltm"))
```

You might recognize the "c" function: as you might guess, we're using it to install two packages at once. If this is the first time we've added packages, we might be prompted to use a personal library and to select a mirror. Using a personal library is fine, and for mirror selection pick whatever's geographically closest to you.

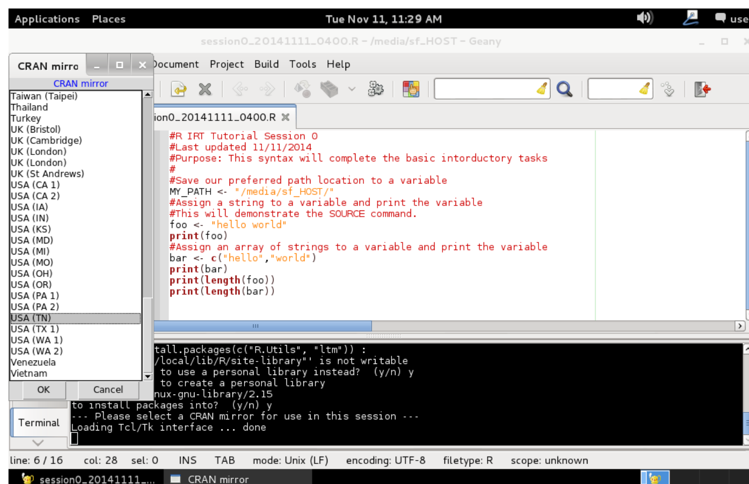
[3] http://cran.r-project.org/web/views/Psychometrics.html

Figure 42: Beginning package installation



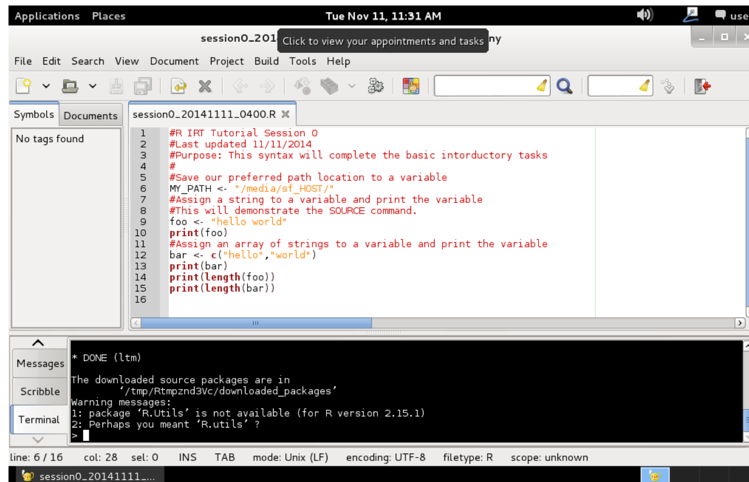
It is often necessary to use personal libraries because of the security settings within a linux environment. More advanced users who want to know more may wish to research the "root" account and the "sudo" linux command for more information on these subtleties. Confirm creating a directory, if necessary, and select an appropriate mirror:

Figure 43: Confirming personal directories, selecting a mirror



Surprise! If you followed the instructions to the letter, you will have just encountered an error:

Figure 44: R is case sensitive!



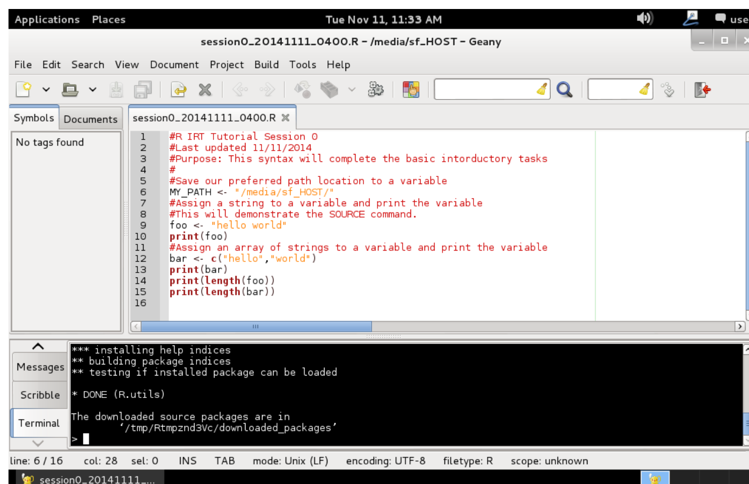
This encounter was purposefully engineered to give you your first, relatively harmless encounter with just how picky R syntax can be when it comes to capitalization. Issue the command again differently:

Code 14: Package installation, continued

```
install.packages(c(“R.utils”,”ltm”))
```

This time, the process should complete without any errors. We now have the packages installed:

Figure 45: Successful package installation

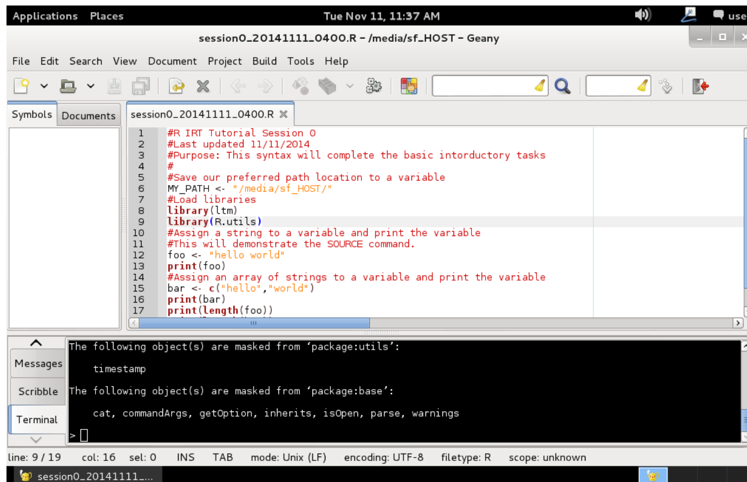


In order to use these packages, we need to load them into the R environment. Execute the following code in the R terminal, but also be sure to add it to our syntax, just below declaring “MY_PATH.” Now is also a good time to remember to save your syntax after editing it.

Code 15: Loading packages

```
#Load libraries
library(ltm)
library(R.utils)
```

Figure 46: Libraries loaded

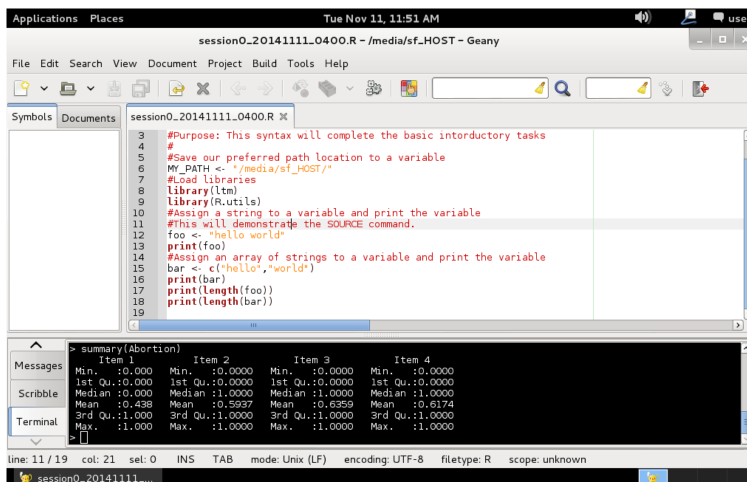


Let's return to the subject of data frames before continuing. A data frame has a name and sub-columns combined with the scalar "$" character, such as Attendance$Name. Let's examine the "summary" command as well as briefly touch on R's powerful vector-based notation to do some rudimentary comparisons. When we loaded the ltm library, a special data frame, Abortion, as added. This data frame is provided with the ltm package to allow for replication and testing of code. Such publication of datasets is a common occurrence in the R community: support forums often use these published sets of data to generate example code. Let's take a look at it by entering some code into the R terminal window:

Code 16: Summary command

```
summary(Abortion)
```

Figure 47: Summary command output

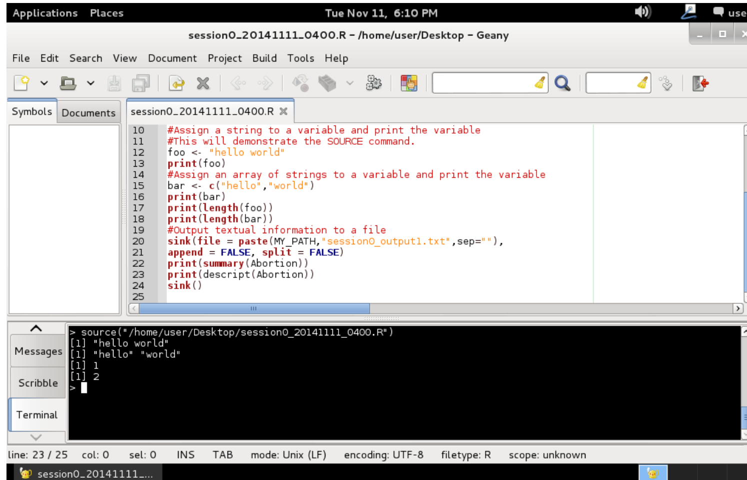


Let's see how this can be more useful by introducing the concept of saving R output to files. Right now we know how to import pre-written code, but the output is stuck inside an R terminal window. How do we get that information into files we can share and archive? Let's try adding the following line to our code just after we declare MY_PATH:

Code 17: Saving textual output to a file

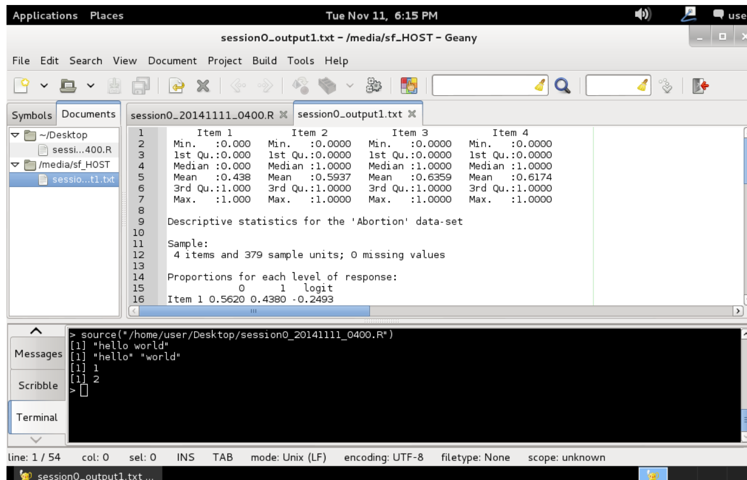
```
#Output textual information to a file
sink(file = paste(MY_PATH,"session0_output1.txt",sep=""),
append = FALSE, split = FALSE)
print(summary(Abortion))
print(descript(Abortion))
sink()
```

Figure 48: Capturing output



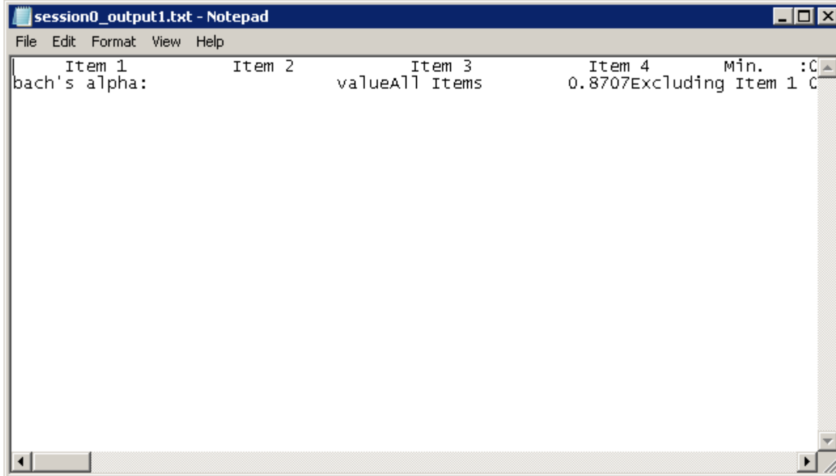
This code also shows an important side note: R does not always mind if code stretches over multiple lines. The "sink" function allows output to be sunk to a different channel than the default output, and in this instance we instruct it to be saved to a file. The "print" command is necessary around our code to make sure it is properly displayed and rerouted. The final "sink" command closes the connection to the file. We can open the output file in our IDE to see what it looks like, and switch the left sidebar from the "Symbols" to the "Documents" view to be more useful to us:

Figure 49: Session 0 Output File



There is an opportunity for further refinement here. If this file were to be opened in Windows, the default application would be notepad, and a peculiar Windows-only standard would result in it looking like this:

Figure 50: Default syntax output appearance in windows notepad



If you want, this is an easy issue to fix inside our IDE. In the Document menu, select the "Set Line Endings" sub-menu, then the "Convert and Set to CR/LF (Win)" and save the document.

Figure 51: Fixing line endings

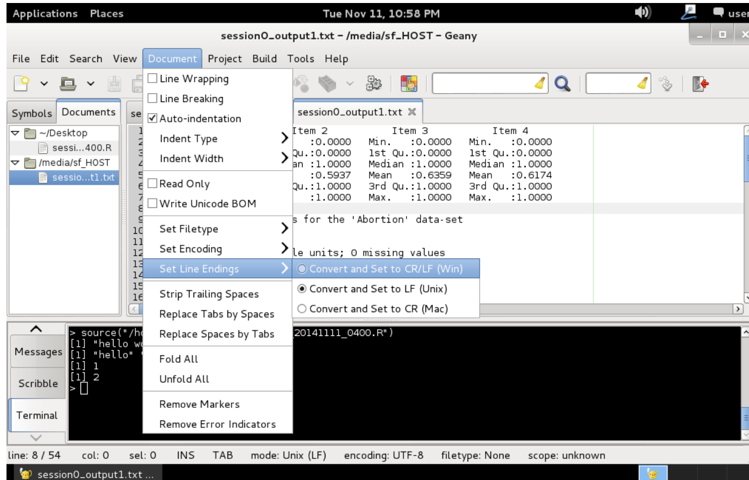
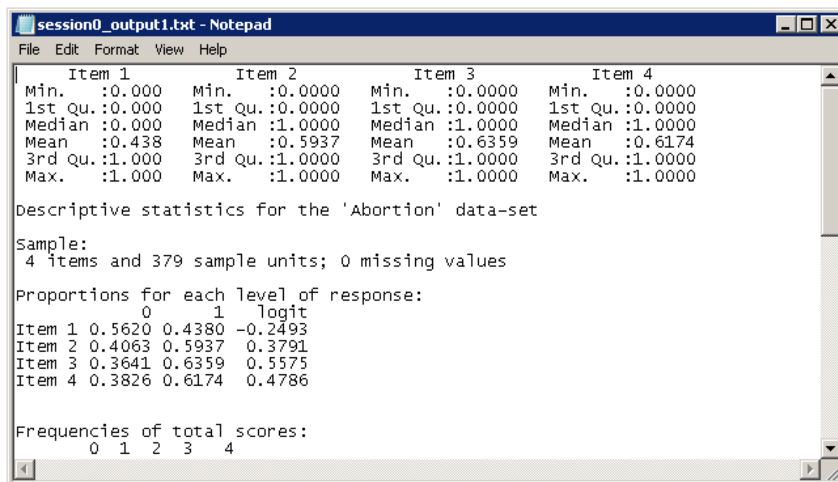


Figure 52: Fixed line endings


```
session0_output1.txt - Notepad
File Edit Format View Help
     Item 1          Item 2          Item 3          Item 4
 Min.   :0.000  Min.   :0.0000  Min.   :0.0000  Min.   :0.0000
 1st Qu.:0.000  1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:0.0000
 Median :0.000  Median :1.0000  Median :1.0000  Median :1.0000
 Mean   :0.438  Mean   :0.5937  Mean   :0.6359  Mean   :0.6174
 3rd Qu.:1.000  3rd Qu.:1.0000  3rd Qu.:1.0000  3rd Qu.:1.0000
 Max.   :1.000  Max.   :1.0000  Max.   :1.0000  Max.   :1.0000

Descriptive statistics for the 'Abortion' data-set

Sample:
 4 items and 379 sample units; 0 missing values

Proportions for each level of response:
            0      1   logit
Item 1 0.5620 0.4380 -0.2493
Item 2 0.4063 0.5937  0.3791
Item 3 0.3641 0.6359  0.5575
Item 4 0.3826 0.6174  0.4786


Frequencies of total scores:
      0  1  2  3   4
```

We're all set for basic R work except for two more important features. Let's start by looking at a convenient way to get external data into R. Suppose we have an online source data, such as: http://coeweb.gsu.edu/coshima/EPRS8410/Class10.csv. We can easily bring that into R.

Code 18: Downloading a file conveniently

```
download.file("http://coeweb.gsu.edu/coshima/EPRS8410/Class10.csv", destfile="Class10.csv")
```

Once the file is on our system, we can read it into an R data frame. There are many different ways to do this depending on the filetype: R can read from all kinds of things, including .CSV and .XLSX files. Check out the "read.table" function's help files for more advanced information. For now, we'll load the Class10 data. There's an interesting option we'll be setting: because the Class10.csv file's first column is the ID of the participant, we can use that ID to set the row name. Doing so will make it more convenient later.

Code 19: Loading a CSV into an R dataframe

```
Class10 <- read.csv("Class10.csv", row.names = 1)
```

Setting "row.names" to 1 allows for using the first column of the file to be used to name the rows. That'll keep later graphs from charting up the ID field. Speaking of charts, let's look at that now. R can make some really crisp-looking graphics, but if you can't get them into your report or manuscript, what use are they? Let's take a look, first, at how to get a graph, then let's save it to a file that can be taken into other programs. Before we do this, make sure to load up the "ltm" library if you haven't already:

Code 20: Loading the ltm library

```
library(ltm)
```

Now let's jump right into this from an IRT perspective. We'll talk more about these functions later, but let's plot the item characteristic curves for the Class 10 data using the Birnbaum 3-parameter model estimated with likelihood-maximization and standardization of ability parameters:

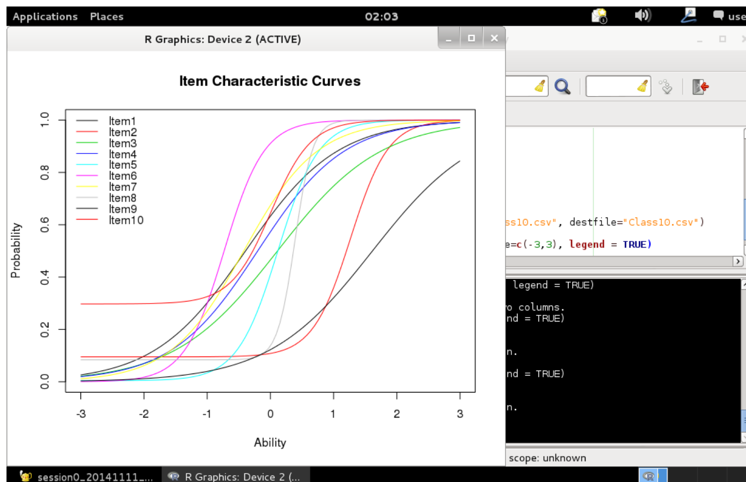
Code 21: Generating 3PL ICCs

```
plot(tpm(Class10,type="latent.trait"), type ="ICC", zrange=c(-3,3), legend = TRUE)
```

This is a *nested* function: without the "plot" function, the first argument is a call to the "tpm" function. Within the tpm function we reference the Class10 object and set an option, type, to "latent.trait." You

can always check the help pages for a function to reference the arguments and their uses: in this case, we choose "latent.trait" in order to get a proper 3PL as we expect it. The plot function has several options, too. We can get different types of plots from an IRT model function, and we select the ICC. We set the range from the traditional -3 to 3, and for legibility we let the plot know we want to see a legend.

Figure 53: ICCs for the Class10 3PL



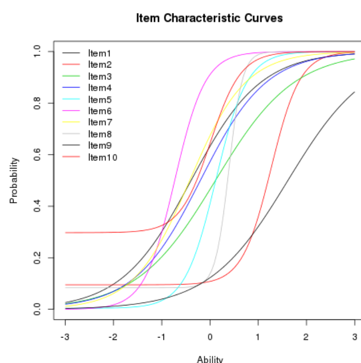
To save it to a file, we use a set of functions not unlike the "sink" function we used earlier to save text output. Many options are available, but we'll use the "png" function for flexibility.

Code 22: Saving Class10 3PL ICC to a file

```
png(file=”Class10ICC.png”)
plot(tpm(Class10,type="latent.trait"), type ="ICC", zrange=c(-3,3), legend = TRUE)
dev.off()
```

Remember to make use of your virtual shared folders to conveniently position files.

Figure 54: Exported image



That was a lot of information. Remember, you can always use the "?" tool inside R to reference a function, and the online communities supporting R are incredibly diverse and in-depth.[4]

---

[4] See http://stats.stackexchange.com/questions/tagged/r for a great example of the active R support community

Session 1: Excel HW1

Source

Classical Item Analysis

For those who would really like to understand CTT, the best way is to actually calculate indices yourself. Calculate p-index, D-index, point biserial, coefficient alpha, KR20, and KR21 for data shown in Table 1 in Harris' article (http://coeweb.gsu.edu/coshima/EPRS8410/1p2p3p.pdf) using Excel. You also get to see how indices based on CTT and those based on IRT are different or similar.

TABLE 1

Item Responses for 10 Examinees to 14 Items

Examinee	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Total raw score
AA	1	0	1	0	0	1	1	0	1	0	0	1	1	0	7
BB	1	1	1	1	1	1	1	0	1	1	0	0	1	0	10
CC	1	0	0	0	0	0	1	0	1	0	1	1	0	0	5
DD	1	0	1	0	0	0	0	0	0	0	0	1	0	0	3
EE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14
FF	1	0	1	0	1	1	1	1	1	0	0	1	0	1	9
GG	1	1	0	0	1	1	1	0	1	0	0	0	0	0	6
HH	1	0	0	0	0	1	1	0	1	0	0	0	0	0	4
II	1	0	0	0	0	1	0	0	1	0	0	0	0	1	4
JJ	1	1	1	0	1	0	0	1	1	0	0	1	0	0	7

Objectives

In Excel HW1, we must:

- Construct a response data frame
- Evaluate the p-indices
- Evaluate the D-indices
- Calculate point-biserial correlations
- Evaluate coefficient alpha
- Find the KR20 and KR21 statistics

Procedures

First, because the source data is within a PDF, let's create the dataframe manually:

Code 23: Excel HW1 Data Frame

```
HW1Test <- data.frame(
Item1 = c(1,1,1,1,1,1,1,1,1,1),
Item2 = c(0,1,0,0,1,0,1,0,0,1),
Item3 = c(1,1,0,1,1,1,0,0,0,1),
Item4 = c(0,1,0,0,1,0,0,0,0,0),
Item5 = c(0,1,0,0,1,1,1,0,0,1),
Item6 = c(1,1,0,0,1,1,1,1,1,0),
Item7 = c(1,1,1,0,1,1,1,1,0,0),
Item8 = c(0,0,0,0,1,1,0,0,0,1),
Item9 = c(1,1,1,0,1,1,1,1,1,1),
Item10 = c(0,1,0,0,1,0,0,0,0,0),
Item11 = c(0,0,1,0,1,0,0,0,0,0),
Item12 = c(1,0,1,1,1,1,0,0,0,1),
Item13 = c(1,1,0,0,1,0,0,0,0,0),
Item14=c(0,0,0,0,1,1,0,0,1,0))
```

Fortunately, because this is dichotomous data, the p-values are simply the means of each item. We can make use of R's vast libraries to make these next steps much easier. Using the knowledge you gained

earlier, install the “CTT” package. We will load it and use some of its functions, and we will make sure the output is sent to a text file we could, for example, submit for credit. Setting the “width” to 1000 via the “options” function makes it so that wide lists, such as the P-values, don’t line-break in our output.

Code 24: Excel HW1 P-Indices

```
library(CTT)
options(width=1000)
sink(file = "ExcelHW1.txt",append = FALSE, split = TRUE)
cat("P Values:\n")
print(reliability(HW1Test,itemal=TRUE)$itemMean)
sink()
```

Issue “?reliability” to get information on the function and what “values” we can get by using the scalar (dollar symbol) not unlike a dataframe. We use the “cat” function with a line break (\n) character to make nice-looking text output, while we use “print” for certain values based on default formatting. When in doubt, test both and use what looks most suitable. You’ll find that we can also get our point-biserial correlations and co-efficient alpha this way. Add the following to our code before the “sink()” closes the output file. Remember “\n” for line breaks.

Code 25: Excel HW1, continued

```
cat("Point-Biserial Correlations:\n")
cat(round(reliability(HW1Test,itemal=TRUE)$pBis,2),"\n")
cat("Coefficient alpha:\n")
cat("Coefficient alpha:",round(reliability(HW1Test,itemal=TRUE)$alpha,2),"\n")
```

The rest is just a little trickier, because it is found in different sources. What remains are the KR-20 and KR-21 evaluations as well as the D indices. This is a good opportunity to introduce R’s ability to computer numbers like a calculator in addition to performing functions. Let’s build our own KR-20 function using the common formula. Since R uses vectors, this is a much more fluid and convenient process than Excel. Instead of defining a cell area such as “A2:A10,” we can instead just refer to the vector. The code below may appear more intimidating than it would inside our IDE due to parenthetical highlighting.

Equation 5: KR-20, KR21, and D-index

$$r_{KR20} = \frac{k}{k-1}\left(1 - \frac{\sum_{i=1}^{k} p_i q_i}{\sigma_x^2}\right), D = \frac{p_{upper} - p_{lower}}{n_{upper} + n_{lower}}$$

Code 26: KR-20, KR-21

```
cat("KR-
20:",round((length(reliability(HW1Test,itemal=TRUE)$itemMean)/(length(reliability(HW1Test,itemal=T
RUE)$itemMean)-1))*(1-(sum(reliability(HW1Test,itemal=TRUE)$itemMean*(1-
reliability(HW1Test,itemal=TRUE)$itemMean))/reliability(HW1Test,itemal=TRUE)$scaleSD^2)),2),"\n")
cat("KR-
21:",round((length(reliability(HW1Test,itemal=TRUE)$itemMean)/(length(reliability(HW1Test,itemal=T
RUE)$itemMean)-1))*(1-
((reliability(HW1Test,itemal=TRUE)$scaleMean*(length(reliability(HW1Test,itemal=TRUE)$itemMean)-
reliability(HW1Test,itemal=TRUE)$scaleMean))/(length(reliability(HW1Test,itemal=TRUE)$itemMean)*re
liability(HW1Test,itemal=TRUE)$scaleSD^2))),2),"\n")
```

The above code is hard to read for humans. Commonly, scalars that are frequently used over and over can often be set to temporary variables. While this is a practical tool that you will see and use, there are risks of changing the data structure type or inadvertently referring to an old or unwanted piece of information. It’s safer to observe where and how the source data is being retrieved at this stage, but more advanced users concerned with the speedy execution of code might be interested in rewriting the above equations using the following hint:

Code 27: Advanced Coding Hint

```
P <- reliability(HW1Test,itemal=TRUE)$itemMean
q<- 1-p
```

The last part, calculating D-indices, introduces us to one of the most powerful and attractive features of R: vector searching. Often, tutorials may recommend using the "subset" function, but it might help demonstrate R's power by using the following code:

Code 28: D-Index

```
upper <- HW1Test[score(HW1Test)$score >= quantile(score(HW1Test)$score,2/3),]
lower <- HW1Test[score(HW1Test)$score <= quantile(score(HW1Test)$score,1/3),]
cat("D Indices:\n")
print(round((reliability(upper,itemal=TRUE)$itemMean -
reliability(lower,itemal=TRUE)$itemMean)/(length(upper$Item1)+length(lower$Item1)),digits=2))
```

What's new here is the use of a pair of brackets with text inside them. Take note of the comma: when we use brackets after a data frame (like HW1Test) we pass two arguments inside them. The first can be thought of as a *row* selector and the second can be thought of as a *column* selector. Try executing the first argument in the selector:

Code 29: D-Index Upper Group Selector

```
score(HW1Test)$score >= quantile(score(HW1Test)$score,2/3
```

You will notice the result is a vector of 10 true/false (or *logical*) values. Where the value is true, the score has been evaluated to be at or above the upper third of the distribution based on the median. Where these values are true, then, the corresponding *rows* of our data frame will be returned. To select columns, we could construct a selector such as "c("Item1","Item2" …)" but it is more convenient to leave the field blank: when blank, all columns in the dataframe will be returned.

Final Results

Code 30: Excel Homework 1 Input

```
#Excel HW 1
HW1Test <- data.frame(
Item1 = c(1,1,1,1,1,1,1,1,1,1),
Item2 = c(0,1,0,0,1,0,1,0,0,1),
Item3 = c(1,1,0,1,1,1,0,0,0,1),
Item4 = c(0,1,0,0,1,0,0,0,0,0),
Item5 = c(0,1,0,0,1,1,1,0,0,1),
Item6 = c(1,1,0,0,1,1,1,1,1,0),
Item7 = c(1,1,1,0,1,1,1,1,0,0),
Item8 = c(0,0,0,0,1,1,0,0,0,1),
Item9 = c(1,1,1,0,1,1,1,1,1,1),
Item10 = c(0,1,0,0,1,0,0,0,0,0),
Item11 = c(0,0,1,0,1,0,0,0,0,0),
Item12 = c(1,0,1,1,1,1,0,0,0,1),
Item13 = c(1,1,0,0,1,0,0,0,0,0),
Item14=c(0,0,0,0,1,1,0,0,1,0))
library(CTT)
options(width=1000)
sink(file = "ExcelHW1.txt",append = FALSE, split = TRUE)
cat("P Values:\n")
print(reliability(HW1Test,itemal=TRUE)$itemMean)
cat("Point-Biserial Correlations:\n")
cat(round(reliability(HW1Test,itemal=TRUE)$pBis,2),"\n")
cat("Coefficient alpha:",round(reliability(HW1Test,itemal=TRUE)$alpha,2),"\n")
cat("KR-
20:",round((length(reliability(HW1Test,itemal=TRUE)$itemMean)/(length(reliability(HW1Test,itemal=T
```

```
RUE)$itemMean)-1))*(1-(sum(reliability(HW1Test,itemal=TRUE)$itemMean*(1-
reliability(HW1Test,itemal=TRUE)$itemMean))/reliability(HW1Test,itemal=TRUE)$scaleSD^2)),2)," \n")
cat("KR-
21:",round((length(reliability(HW1Test,itemal=TRUE)$itemMean)/(length(reliability(HW1Test,itemal=T
RUE)$itemMean)-1))*(1-
((reliability(HW1Test,itemal=TRUE)$scaleMean*(length(reliability(HW1Test,itemal=TRUE)$itemMean)-
reliability(HW1Test,itemal=TRUE)$scaleMean))/(length(reliability(HW1Test,itemal=TRUE)$itemMean)*re
liability(HW1Test,itemal=TRUE)$scaleSD^2))),2),"\n")
upper <- HW1Test[score(HW1Test)$score >= quantile(score(HW1Test)$score,2/3),]
lower <- HW1Test[score(HW1Test)$score <= quantile(score(HW1Test)$score,1/3),]
cat("D Indices:\n")
print(round((reliability(upper,itemal=TRUE)$itemMean -
reliability(lower,itemal=TRUE)$itemMean)/(length(upper$Item1)+length(lower$Item1)),digits=2))
sink()
```

Figure 55: Excel Homework 1 Output

```
P Values:
 Item1  Item2  Item3  Item4  Item5  Item6  Item7  Item8  Item9 Item10 Item11 Item12 Item13 Item14
   1.0    0.4    0.6    0.2    0.5    0.7    0.7    0.3    0.9    0.2    0.2    0.6    0.3    0.3
Point-Biserial Correlations:
NA 0.49 0.43 0.75 0.63 0.26 0.34 0.54 0.33 0.75 0.3 0.08 0.62 0.3
Coefficient alpha: 0.8
KR-20: 0.83
KR-21: 0.74
D Indices:
 Item1  Item2  Item3  Item4  Item5  Item6  Item7  Item8  Item9 Item10 Item11 Item12 Item13 Item14
  0.00   0.07   0.08   0.04   0.09   0.03   0.03   0.07   0.03   0.04  -0.01   0.03   0.07   0.02
```

Extension Material: Fizz Buzz

Statisticians often wind up being, in some part, programmers. Within the programming world, credentials are often diverse and hard to assess. One of the common challenges faced in filling a position is determining whether or not the candidate, after having been in a computer science classroom for a few semesters, is actually able to problem solve with programs. In the industry, one of the common tasks is to give applicants a test or to ask for sample code. One of these tests borrows from a classic educational activity: the fizzbuzz game. As a certain web source suggests, “the ‘Fizz-Buzz test’ is an interview question designed to help filter out the 99.5% of programming job candidates who can't seem to program their way out of a wet paper bag.” Trying out the Fizz-Buzz test in your favorite statistical package, in addition to R, is an extremely worthwhile exercise.

FizzBuzz is a game used to teach about common multiples. The game is played by counting from 1 upwards, and the rules are to say “fizz” when the number is divisible by 3 and “buzz” when the number is divisible by 5. For numbers divisible by both 3 and 5, the correct response is both “fizz” and “buzz.” A typical sequence would sounds like, “1 2 fizz 4 buzz fizz 7 8 buzz fizzbuzz.” The bonus task for this session is to create a fizz buzz routine in R. Here are two hints:

Code 31: Session 1 Bonus Hint 1

```
fizzbuzz = data.frame(input = seq(1,10))
fizzbuzz$output[fizzbuzz$input > 5] <- “foo”
print(fizzbuzz)
```

Code 32: Session 1 Bonus Hint 2

```
?”%%”
?”&”
```

After completing this session, you should be well on your way to a confident R proficiency. Remember, there are lots of tutorials and help files available both inside R (each help file classically ends with example code) and on the internet.

Session 2: Excel HW2

Source

```
Excel HW 2
Chapter 2 Homework: ICCs

Consider the following three items with given item parameters:

Item 1: a = 1.0 b = -.5 c = 0
Item 2: a = 1.2 b = 0 c = 0
Item 3: a = 1.5 b = 1 c = .20

1. Draw item characteristic curves for those three items using a computer program such as Excel.

2. By looking at the graph, for someone with theta = 1, which item has the highest probability of
being answered correctly by this person?

3. Again, by looking at the graph, which item appears to be most discriminating for someone with
theta = -1?
```

Objectives

This exercise is fairly straightforward. In R, we just need to plot Item Characteristic Curves.

Procedures

This is a very appropriate time to discuss one of the features of R more akin to traditional programming languages than what some statistical software packages make easily available. In R, we can define and call our own functions just like we've been using those available in the core and supplementary packages. A function, as you have already seen, can have arguments and returns values. Let's create our own custom function to replicate Birnbaum's 3-parameter model:

Equation 6: The 3-Parameter Model

$$P(\theta)=c+\frac{1-c}{1+e^{-1.7a(\theta-b)}}$$

Code 33: A Custom 3PL Function

```
P <- function(difficulty,discrimination,pseudoguessing,ability) {
return (pseudoguessing+(1-pseudoguessing)/(1+exp(-1.7*discrimination*(ability-difficulty))))
}
P(0,1,0,0)
```

As you can see in the second line, we can issue the new "P" function to calculate a probability. Using this functionality, we can now write some code to make the ICCs that looks easy to read. We'll break the graphing process into separate steps to have more control over the output.

Code 34: Generating ICCs

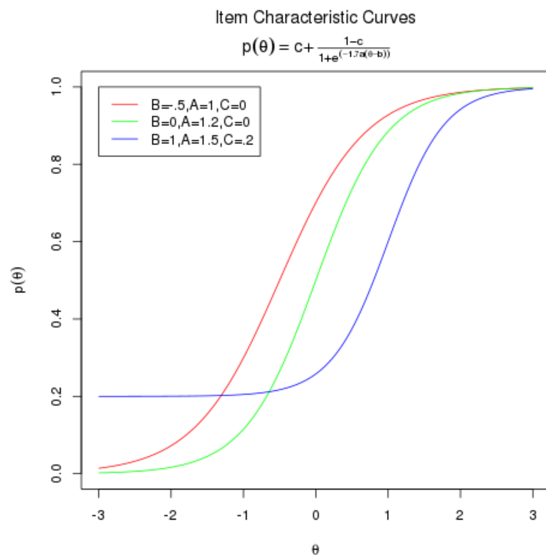
```
png(file="Excel-HW2.png")
plot(0,0,xlim=c(-
3,3),ylim=c(0,1),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=expression(atop("I
tem Characteristic Curves",p(theta)==c+over(1-c,1+e^(-1.7*a*(theta-b))))))
lines(seq(-3,3,.001),P(-.5,1,0,seq(-3,3,.001)),type="l",col="red")
lines(seq(-3,3,.001),P(0,1.2,0,seq(-3,3,.001)),type="l",col="green")
lines(seq(-3,3,.001),P(1,1.5,.2,seq(-3,3,.001)),type="l",col="blue")
legend(x = -3, y = 1, lwd = 1, legend=c("B=-
.5,A=1,C=0","B=0,A=1.2,C=0","B=1,A=1.5,C=.2"),col=c("red","green","blue"))
dev.off()
```

Final Results

Code 35: Excel HW2 Input

```
P <- function(difficulty,discrimination,pseudoguessing,ability) {
return (pseudoguessing+(1-pseudoguessing)/(1+exp(-1.7*discrimination*(ability-difficulty))))
}
png(file="Excel-HW2.png")
plot(0,0,xlim=c(-
3,3),ylim=c(0,1),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=expression(atop("I
tem Characteristic Curves",p(theta)==c+over(1-c,1+e^(-1.7*a*(theta-b))))))
lines(seq(-3,3,.001),P(-.5,1,0,seq(-3,3,.001)),type="l",col="red")
lines(seq(-3,3,.001),P(0,1.2,0,seq(-3,3,.001)),type="l",col="green")
lines(seq(-3,3,.001),P(1,1.5,.2,seq(-3,3,.001)),type="l",col="blue")
legend(x = -3, y = 1, lwd = 1, legend=c("B=-
.5,A=1,C=0","B=0,A=1.2,C=0","B=1,A=1.5,C=.2"),col=c("red","green","blue"))
dev.off()
```

Figure 56: Excel HW2 Output



Session 3: Excel HW3

Source

```
Use the three items from Chapter 2 Homework.
Item 1: a = 1.0 b = -.5 c = 0
Item 2: a = 1.2 b = 0 c = 0
Item 3: a = 1.5 b = 1 c = .20

1. Estimate theta for Person A whose answer pattern is {1 1 0}.
Note {1 1 0} indicates Person A answered Items 1 and 2 correctly, and Item 3 wrong.

2. Estimate theta for Person B whose answer pattern is {1 0 0}.

3. Estimate theta for Person C whose answer pattern is {0 0 0}.
```

Objectives

There are three very general estimation styles for item response functions: parameter, ability, and joint estimation of both. In this exercise, we conduct ability estimation with a set of given parameters and responses: this will be done via examining the likelihood function of responses given parameters for each ability. Although we could write our own, custom function for likelihood as we did previously for item characteristic curves, it will be easier and more robust to call in a new R library.

Extension Material: Updating the Virtual Machine's Operating System and R Software

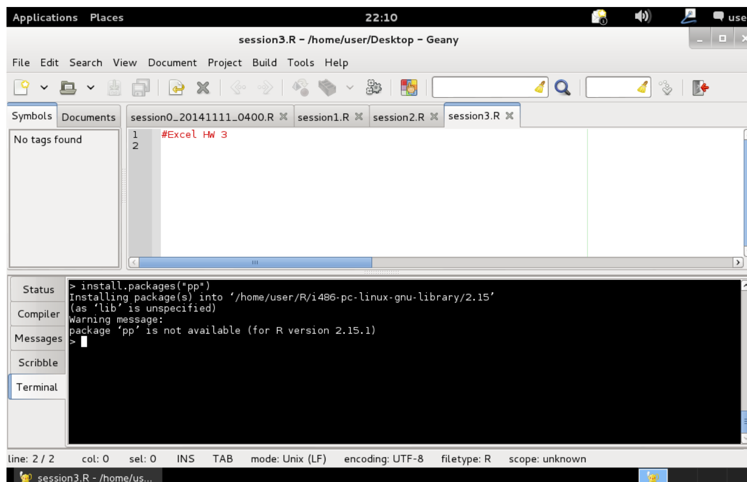
After examining the CRAN repository and checking some articles for citations, it can be found that the person parameter library "PP" will fit out needs. Installing packages is something we've already done, but for reference:

Code 36: Installing and Loading the Person Parameter Library

```
install.packages("PP")
library(PP)
```

Surprise! You are very likely to encounter an error.

Figure 57: Error: R version insufficient for library



This is a normal problem for popular linux software distributions: the "package" maintained by the operating system is not the most up-to-date version of software published by the author. Fortunately,

this is easy to fix in the Debian environment we have installed.[5] To patch the issue, open up a root terminal and issue the following command:

Code 37: Adding a Package Repository to the Operating System

```
add-apt-repository 'deb http://mirrors.nics.utk.edu/cran/bin/linux/debian wheezy-cran3/'
apt-key adv --keyserver keys.gnupg.net --recv-key 381BA480
apt-get update
apt-get dist-upgrade
```

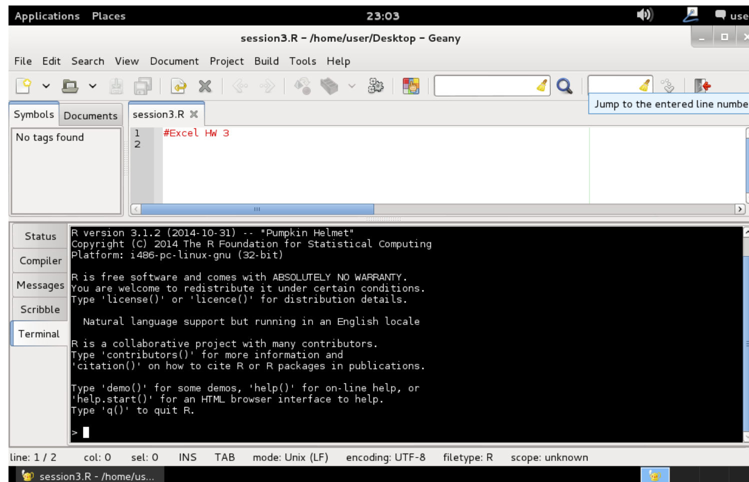
The above commands will add CRAN's repository to what our virtual machine's operating system checks for updates. Then, we add an encryption verification key to our operating system for secure updates. We refresh our local database of software with the most updated version, and then we install any necessary updates. You might see something like:

Figure 58: Software Upgrade Messages

You can verify these updates. They might take a few minutes. Sometimes, rebooting is necessary depending on what all gets upgraded in the system. Since virtual machines are fairly efficient, it doesn't hurt to just reboot on principle. After re-opening our IDE, we can see that R has indeed been upgraded:

[5] See http://cran.r-project.org/bin/linux/debian/README for a detailed discussion

Figure 59: R Upgraded



The first thing we should do, now, is to upgrade our packages:

Code 38: Upgrading Internal R Packages

```
update.packages()
```

Procedures

After that's done, we can go back to installing the Person Parameter package:

Code 39: Installing the Person Parameter Library

```
install.packages(“PP”)
library(PP)
```

From here, it's just a matter of calling up the appropriate functions provided by this library and sending the appropriate arguments to these functions. We can record our output using our separate syntax and output approach so far. The documentation for this package is available online, and like many pieces of R documentation, it is both in-depth and a little challenging to get into at first for newcomers. R writers usually use conventions for naming and terms with a different, programmatic perspective than many psychometric articles and texts sometimes do; developing the understanding that a "hierarchical linear model" and a "mixed-effects model" are referring, in general, to similar concepts is part of developing an ear for the literature.[6]

If we just fed a dataframe containing responses to the person parameter function, we wouldn't be giving it everything it needs: it requires knowing the difficulty (b, we call it difficulty, too) and slope (a, we call it discrimination) parameters. If we include these in the same dataframe as extra columns, though, the function will get confused. This is a great time to introduce the concept of *attributes* and to showcase one of their applications. As you have seen, a data frame object contains vectors of data of varying types referenced with the scalar ("$") operator, such as "foo$bar." Applying a function to a dataframe often propagates that function throughout its columns. Attributes are a way of attaching data without putting it in that propagating schematic structure. Let's create our responses dataframe with columns representing responses to each item, and then we will set attributes representing properties of each item suitable for the Person Parameter function:

[6] See http://cran.r-project.org/web/packages/PP/vignettes/intro_pp.html for an in-depth estimation guide

Code 40: Creating the appropriate data objects

```
hw3test <- data.frame(
item1 = c(1,1,0),
item2 = c(1,0,0),
item3 = c(0,0,0)
)
attr(hw3test,difficulty_parameter) <- c(-.5,0,1)
attr(hw3test,discrimination_parameter) <- c(1,1.2,1.5)
attr(hw3test,guessing_parameter <- c(0,0,.2)
```

Although it's possible to keep the parameters in their own variables, it's quite cleaner (and therefore safer) to keep them associated with the data. For more information, issue the following commands:

Code 41: Examining the workspace

```
ls()
str(hw3test)
```

The "ls" command lists objects in the current environment. In the linux environment, "ls" is an essential command to check the file contents of a directory. The "str" command gives a string-like output of the object's information. In the case of our hw3test data frame, this includes the columns of item data and attributes representing item parameters. In both cases, the row-column orientation is vector-oriented: the first position of each item vector refers to the first respondent, and the first position of each parameter vector refers to the first question. It can be easy to forget this and accidentally transpose.

Now that we've taken a little bit of time to get the right, add-on packages and build our data, all we need to do is call the functions. In traditional software packages, this inevitably involves clicking around and having additional windows appear. In R, all it involves is calling a function with some arguments:

Code 42: Ability Estimation

```
PP_4pl(
respm = as.matrix(hw3test),
thres = attr(hw3test,"difficulty_parameter"),
slopes = attr(hw3test,"discrimination_parameter"),
lowerA = attr(hw3test,"guessing_parameter"),
type = "mle"
)
```

In the above example, we used line-breaks to make the list of arguments more readable as we have been for dataframes. In the help files for this function (try "?PP_4pl" to see them yourself) the "respm" argument must be a matrix, so we use the "as.matrix()" function to *coerce* the data.frame into the appropriate object type. For the item parameters, we use the "attr()" function to access the data we stored as attributes of the hw3test object. Finally, the "type" argument is referenced in the help file. We use maximum likelihood estimation in this instance. This showcases some of R's potential: we can just change that argument from "mle" to "wle" to switch to weighted maximum likelihood.

Remember to save syntax and to use the "sink()" function to structure our code to place its output in a file suitable for submission and archival. It can be useful to add comments to functions to reference their arguments if you do not plan on committing them to memory and want to avoid continually opening up the help files. In addition, although the help files are accessible inside R, it's also possible to have a dual setup browsing the online help files via the CRAN repository.[7] More advanced users may want to experiment with using the desktop workspace features to have more than one "desktop."

---

[7] http://cran.r-project.org/web/packages/

Final Results

Code 43: Excel HW3 Input

```
#Excel HW 3
library(PP)
hw3test <- data.frame(
item1 = c(1,1,0),
item2 = c(1,0,0),
item3 = c(0,0,0)
)
attr(hw3test,"difficulty_parameter") <- c(-.5,0,1)
attr(hw3test,"discrimination_parameter") <- c(1,1.2,1.5)
attr(hw3test,"guessing_parameter") <- c(0,0,.2)
options(width=1000)
sink(file = "ExcelHW3.txt",append = FALSE, split = TRUE)
print(str(hw3test))
print(PP_4pl(
respm = as.matrix(hw3test),
thres = attr(hw3test,"difficulty_parameter"),
slopes = attr(hw3test,"discrimination_parameter"),
lowerA = attr(hw3test,"guessing_parameter"),
type = "mle"
))
sink()
```

Code 44: Excel HW3 Output

```
'data.frame':        3 obs. of  3 variables:
 $ item1: num  1 1 0
 $ item2: num  1 0 0
 $ item3: num  0 0 0
 - attr(*, "difficulty_parameter")= num  -0.5 0 1
 - attr(*, "discrimination_parameter")= num  1 1.2 1.5
 - attr(*, "guessing_parameter")= num  0 0 0.2
NULL
Estimating:  3pl model ...
type = mle
Estimation finished!
     estimate     SE
[1,]   0.7092 1.1133
[2,]  -0.5901 1.2786
[3,]     -Inf     NA
```

Session 4: Excel HW4

Source

```
Using data from Table 4.3 (p. 74), create a graph like Figure 4.5.
Use three models:
One-parameter model: b = 0.17
Two-parameter model: b = 0.18; a = 0.56
Three-parameter model: b = 0.76; a = 1.23; c = .25
```

Table 1: HSR Table 4.3

θ	p	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
-2	20%	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1
-1	25%	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0
0	40%	1	0	0	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0	1	1
1	75%	1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1	0	1	0	1
2	90%	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1

Table 2: Model Parameters

Model	a	b	c
1P	1	0.17	0
2P	0.56	0.18	0
3P	1.23	0.76	0.25

Objectives

Because we already know how to plot item-characteristic curves, the only additional work required in this task is to plot observed points for comparison against each of the three models.

Procedures

First, let's create a data frame representing the observed P values and ability levels:

Code 45: Creating a dataframe for graphing

```
hw4test <- data.frame(
ability = c(-2,-1,0,1,2),
p_obs = c(.2,.25,.4,.75,.9)
)
```

Let's then pull in code we developed for drawing ICCs in HW2:

Code 46: Drawing ICCs

```
P <- function(difficulty,discrimination,pseudoguessing,ability) {
return (pseudoguessing+(1-pseudoguessing)/(1+exp(-1.7*discrimination*(ability-difficulty))))
}
plot(0,0,xlim=c(-
3,3),ylim=c(0,1),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=expression(atop("I
tem Characteristic Curves",p(theta)==c+over(1-c,1+e^(-1.7*a*(theta-b))))))
lines(seq(-3,3,.001),P(.17,1,0,seq(-3,3,.001)),type="l",col="red")
lines(seq(-3,3,.001),P(.18,.56,0,seq(-3,3,.001)),type="l",col="green")
lines(seq(-3,3,.001),P(1.23,.76,.25,seq(-3,3,.001)),type="l",col="blue")
legend(x = -3, y = 1, lwd = 1, legend=c("B=-
.17,A=1,C=0","B=.18,A=.56,C=0","B=.76,A=1.23,C=.25"),col=c("red","green","blue"))
dev.off()
```

Above, we've changed the arguments sent to the custom "P()" function to fit our new curves. We also need to add the observed data as points without a line, and we should add it to our legend. The two changed lines will look like so:

Code 47: New line drawing, modified legend

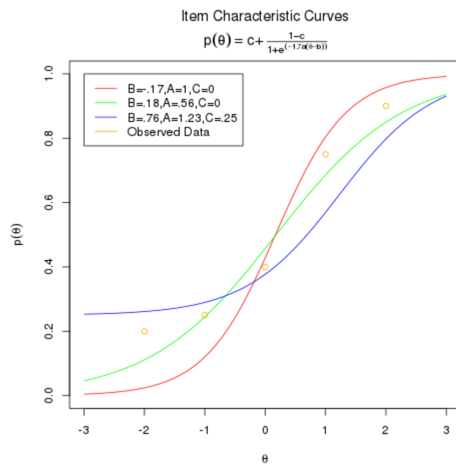
```
lines(hw4test$ability,hw4test$p_obs,type="p",col="orange")
legend(x = -3, y = 1, lwd = 1, legend=c("B=-
.17,A=1,C=0","B=.18,A=.56,C=0","B=.76,A=1.23,C=.25",”Observed
Data”),col=c("red","green","blue",”orange”))
```

Final Results

Code 48: Excel HW4 Input

```
#Excel HW 4
P <- function(difficulty,discrimination,pseudoguessing,ability) {
return (pseudoguessing+(1-pseudoguessing)/(1+exp(-1.7*discrimination*(ability-difficulty))))
}
hw4test <- data.frame(
ability = c(-2,-1,0,1,2),
p_obs = c(.2,.25,.4,.75,.9)
)
png(file="Excel-HW4.png")
plot(0,0,xlim=c(-
3,3),ylim=c(0,1),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=expression(atop("I
tem Characteristic Curves",p(theta)==c+over(1-c,1+e^(-1.7*a*(theta-b))))))
lines(seq(-3,3,.001),P(.17,1,0,seq(-3,3,.001)),type="l",col="red")
lines(seq(-3,3,.001),P(.18,.56,0,seq(-3,3,.001)),type="l",col="green")
lines(seq(-3,3,.001),P(1.23,.76,.25,seq(-3,3,.001)),type="l",col="blue")
lines(hw4test$ability,hw4test$p_obs,type="p",col="orange")
legend(x = -3, y = 1, lwd = 1, legend=c("B=-
.17,A=1,C=0","B=.18,A=.56,C=0","B=.76,A=1.23,C=.25","Observed
Data"),col=c("red","green","blue","orange"))
dev.off()
```

Figure 60: Excel HW4 Output



Session 5: Excel HW5

Source

```
Use the three items from Chapter 2 Homework.
Item 1: a = 1.0 b = -.5 c = 0
Item 2: a = 1.2 b = 0 c = 0
Item 3: a = 1.5 b = 1 c = .20

1. Draw a test characteristic curve for those three items using a computer program such as Excel.

2. Draw item information curves for those three items using a computer program such as Excel.

3. By looking at the graph, for someone with theta = 1, which item has the highest information?

4. Draw a test information curve for this three-item test.  For what range of thetas dose this
test offer most information?
```

Objectives

New types of graphs must be made in this assignment. We will create functions for the test characteristic function, item information curves, and test information curves. We will prepare graphs.

Procedures

Since the test characteristic curve is just an amalgamation of item characteristic curves, it's easy to add together values using item characteristic curve functions we've already made to produce the output:

Code 49: Making a TCC function

```
P <- function(difficulty,discrimination,pseudoguessing,ability) {
return (pseudoguessing+(1-pseudoguessing)/(1+exp(-1.7*discrimination*(ability-difficulty))))
}
test5tcc <- function(ability) {
return (P(-.5,1,0,ability)+P(0,1.2,0,ability)+P(1,1.5,.2,ability))
}
```

Graphing the result is easy, if we re-use what we've done before:

Code 50: Graphing a TCC

```
plot(0,0,xlim=c(-3,3),
ylim=c(0,3),type="l",xlab=expression(theta),ylab=expression(p(theta)),
main=expression("Test Characteristic Curve"))
lines(seq(-3,3,.001),test5tcc(seq(-3,3,.001)),type="l",col="red")
legend(x = -3, y = 3, lwd = 1, legend=c("Test 1 (Items 1-3)"),col=c("red"))
```

Note that we change a few parameters, such as the "ylim" argument to the "plot()" function and the "y" argument to the "legend" function to better graph out the different domain of this function. The next part gets interesting. Information functions can be expressed precisely using derivatives. R isn't meant to be a computational algebra environment – programs like Mathematica and open source languages like Maple or Julia do that – but R can handle simple derivations well enough to justify their use.

In order to take advantage of R's ability to calculate derivatives, we need to supply it with clean, simple mathematical formulae in the form of objects called "expressions." We've already seen these before when dealing with making our ICC formula look nice in a plot. There's more than one way to proceed, but let's use a somewhat easy-to-read approach for now. We will define the item characteristic curves as mathematical expressions, and then we will define the item information curves as functions making use of the derivative to replicate the following formula.

Equation 7: Item Information Function

$$I(\theta) = \frac{P'(\theta)^2}{P(\theta)Q(\theta)}$$

Code 51: Creating the Item Information Functions

```
item1icc <- expression(0+(1-0)/(1+exp(-1.7*1*(ability+.5))))
item2icc <- expression(0+(1-0)/(1+exp(-1.7*1.2*(ability-0))))
item3icc <- expression(.2+(1-.2)/(1+exp(-1.7*1.5*(ability-1))))
item1iic <- function(ability) {eval(D(item1icc, "ability"))^2/(eval(item1icc)*(1-eval(item1icc)))}
item2iic <- function(ability) {eval(D(item2icc, "ability"))^2/(eval(item2icc)*(1-eval(item2icc)))}
item3iic <- function(ability) {eval(D(item3icc, "ability"))^2/(eval(item3icc)*(1-eval(item3icc)))}
```

The amount of parentheses may be daunting at first. Feel free to use line-breaks and even tab notation if that helps.[8] Now that we have "function" type objects returning the item information statistic for a particular set of items, we can go ahead and graph them using the same techniques we've already used:

Code 52: Graphing some IICs

```
plot(0,0,xlim=c(-
3,3),ylim=c(0,1.5),type="l",xlab=expression(theta),ylab=expression(I(theta)),main=expression("Item
Information Curves"))
lines(seq(-3,3,.001),item1iic(seq(-3,3,.001)),type="l",col="red")
lines(seq(-3,3,.001),item2iic(seq(-3,3,.001)),type="l",col="green")
lines(seq(-3,3,.001),item3iic(seq(-3,3,.001)),type="l",col="blue")
legend(x = -3, y = 1.5, lwd = 1, legend=c("Item 1","Item 2","Item 3"),col=c("red","green","blue"))
```

The last portion, test information graphing, is simple: we've already combined item probabilities, so combining item information shouldn't seem too novel as far as the R code is concerned:

Code 53: Making the Test Information Curve

```
test5tic <- function(ability) {
return (item1iic(ability)+item2iic(ability)+item3iic(ability))
}
plot(0,0,xlim=c(-
3,3),ylim=c(0,3),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=expression("Test
Information Curve"))
lines(seq(-3,3,.001),test5tic(seq(-3,3,.001)),type="l",col="red")
legend(x = -3, y = 3, lwd = 1, legend=c("Test 1 (Items 1-3)"),col=c("red"))
```

All that remains is to tie these all together with "png()" and "dev.off()" function calls to ensure our output is appropriately separate from our syntax and in a format we could turn in or use elsewhere.

---

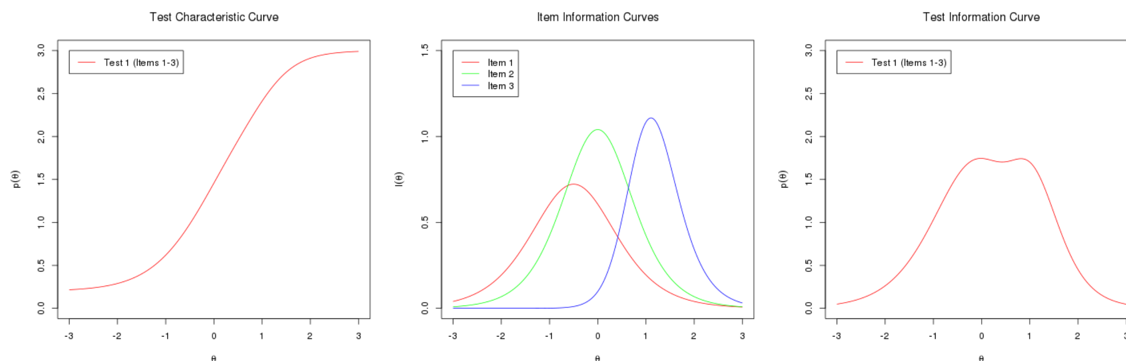
[8] See https://google-styleguide.googlecode.com/svn/trunk/Rguide.xml for an in-depth discussion

Final Results

Code 54: Excel HW5 Input

```
#Excel HW 5
P <- function(difficulty,discrimination,pseudoguessing,ability) {
return (pseudoguessing+(1-pseudoguessing)/(1+exp(-1.7*discrimination*(ability-difficulty))))
}
test5tcc <- function(ability) {
return (P(-.5,1,0,ability)+P(0,1.2,0,ability)+P(1,1.5,.2,ability))
}
png(file="Excel-HW5A.png")
plot(0,0,xlim=c(-
3,3),ylim=c(0,3),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=expression("Test
Characteristic Curve"))
lines(seq(-3,3,.001),test5tcc(seq(-3,3,.001)),type="l",col="red")
legend(x = -3, y = 3, lwd = 1, legend=c("Test 1 (Items 1-3)"),col=c("red"))
dev.off()
item1icc <- expression(0+(1-0)/(1+exp(-1.7*1*(ability+.5))))
item2icc <- expression(0+(1-0)/(1+exp(-1.7*1.2*(ability-0))))
item3icc <- expression(.2+(1-.2)/(1+exp(-1.7*1.5*(ability-1))))
item1iic <- function(ability) {eval(D(item1icc, "ability"))^2/(eval(item1icc)*(1-eval(item1icc)))}
item2iic <- function(ability) {eval(D(item2icc, "ability"))^2/(eval(item2icc)*(1-eval(item2icc)))}
item3iic <- function(ability) {eval(D(item3icc, "ability"))^2/(eval(item3icc)*(1-eval(item3icc)))}
png(file="Excel-HW5B.png")
plot(0,0,xlim=c(-
3,3),ylim=c(0,1.5),type="l",xlab=expression(theta),ylab=expression(I(theta)),main=expression("Item
Information Curves"))
lines(seq(-3,3,.001),item1iic(seq(-3,3,.001)),type="l",col="red")
lines(seq(-3,3,.001),item2iic(seq(-3,3,.001)),type="l",col="green")
lines(seq(-3,3,.001),item3iic(seq(-3,3,.001)),type="l",col="blue")
legend(x = -3, y = 1.5, lwd = 1, legend=c("Item 1","Item 2","Item 3"),col=c("red","green","blue"))
dev.off()
test5tic <- function(ability) {
return (item1iic(ability)+item2iic(ability)+item3iic(ability))
}
png(file="Excel-HW5C.png")
plot(0,0,xlim=c(-
3,3),ylim=c(0,3),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=expression("Test
Information Curve"))
lines(seq(-3,3,.001),test5tic(seq(-3,3,.001)),type="l",col="red")
legend(x = -3, y = 3, lwd = 1, legend=c("Test 1 (Items 1-3)"),col=c("red"))
dev.off()
```

Figure 61: Excel HW6 Output



Session 6: Excel HW6

Source

```
DIF Analysis
Conduct a DIF analysis using data "dn211dif.csv" (a 10-item test with N = 200) between Group 1 and
Group 2. Use the 2PL model. Identify DIF items.
```

Objectives

We need to conduct a DIF analysis.

Procedures

Things are starting to get more complex, but that's actually a good thing. The more practical the problem, the more likely it is that someone has already written an R package to tackle it. As we've seen, some tasks such as ability estimation are made very easy when existing packages already supply functions, but we can also approach things by making functions of our own.

We'll start by reading in the source data file. Assuming you already have it in a particular location – in the case of the example below, it's been placed in the /media/sf_HOST shared folder – bringing the data into R is easy! There are many advanced configuration options that can be browsed in the help files.

Code 55: Importing Data

```
hw6data <- read.csv("/media/sf_HOST/dn211dif.csv")
summary(hw6data)
```

There are some challenges with this data from its source that are revealed when we use the "summary()" function. First, the dataset contains ID and group assignments: while this isn't directly a problem, as we've seen before, it can sometimes fool add-on packages into treating these columns like data responses. Second, items 7-10 use lower-case while items 1-6 use uppercase letters at the start of each name. We can clean both of these issues with just a few commands.

Code 56: Some data cleaning

```
hw6test <- hw6data[TRUE,c(3,4,5,6,7,8,9,10,11,12)]
colnames(hw6test) <-
c("Item1","Item2","Item3","Item4","Item5","Item6","Item7","Item8","Item9","Item10")
attr(hw6test,"group") <- hw6data$Group
```

In the first line, we use the selector brackets we've seen before. Remember, the first argument tells R which rows to select: by giving it one TRUE value, we quickly tell R that every row is okay for selection. The second argument tells R which columns to select. You can refer to them by name with quotes or just select the number, starting with 1. In this case, we use numbers for efficiency. The second command just assigns new names to the resulting columns that use a consistent capitalization scheme. Now we have a "clean" data frame, hw6test, and the original source unmodified in hw6data.

One of the most amazing features of R is the limitless, free potential of its package system. This is also one of its greatest challenges for users looking to perform some advanced statistical analysis. Rather than re-invent the wheel itself, which should we use? This is where the CRAN Task View and other help file browsing comes in handy. If we use SPSS, IRTPRO, or other software packages, we are essentially trusting the developer to have understood and appropriately programmed the mathematics. With R packages, help files usually cite sources and the original programming language is freely available for inspection and validation. Let's make use of the "difR" package and go forward by using Raju's area.[9]

---

[9] See http://cran.r-project.org/web/packages/difR/difR.pdf for the relevant help files.

Remember, first we must load the “difR” package. That might require installing the package first.

Code 57: Install difR package

```
install.packages("difR")
library(difR)
```

This might take a little longer than previous installations because of the number of co-dependencies. The help file for difR gives a lot of information. The “difRaju” function has the following arguments:

Figure 62: difRaju documentation excerpt

```
difRaju(Data, group, focal.name, model, c=NULL, engine="ltm", discr=1,
irtParam=NULL, same.scale=TRUE, alpha=0.05, signed=FALSE, purify=FALSE,
nrIter=10, save.output=FALSE, output=c("out","default"))
```

Let’s build our function call to handle these arguments. In help files, generally, an argument listed such as “purify=FALSE” implies that the default value for the argument is FALSE: if we do not set it, it’s false. This means we could avoid setting it if we wanted it to be false, but for this example we’ll be redundant:

Code 58: Calling the difRaju function

```
difRaju(
    Data = as.matrix(hw6test),
    group = attr(hw6test,"group"),
    focal.name = 1,
    same.scale = FALSE,
    model = "2PL"
)
```

Notice that we did not request purification and specified that items aren’t yet scaled between groups. This is the “anchor all items” step in IRTPRO translated into R. The call will give the following output:

Figure 63: difRaju initial output

```
Detection of Differential Item Functioning using Raju's method
with 2PL model and without item purification

Type of Raju's Z statistic: based on unsigned area

Engine 'ltm' for item parameter estimation

Raju's statistic:

       Stat.   P-value
Item1  -1.1545  0.2483
Item2  -1.4217  0.1551
Item3   2.2838  0.0224 *
Item4   1.3486  0.1775
Item5   0.8661  0.3865
Item6  -0.0554  0.9558
Item7  -1.5553  0.1199
Item8   2.9140  0.0036 **
Item9  -0.9817  0.3263
Item10  0.6810  0.4959

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Detection thresholds: -1.96 and 1.96 (significance level: 0.05)

Items detected as DIF items:

 Item3
 Item8

Output was not captured!
```

Extension Materials: DRY

In programming, a common maxim is "DRY" or "Don't Repeat Yourself." We're about to make a lot of graphs – why copy-paste code that might need changing and create a big headache? Let's build some functionality to replicate work without having to replicate effort.

Code 59: Replicating Plots

```
makeplot <- function(item) {
x1 <- plot(group_1, type = "ICC", item = item)[,1]
y1 <- plot(group_1, type = "ICC", item = item)[,2]
x2 <- plot(group_2, type = "ICC", item = item)[,1]
y2 <- plot(group_2, type = "ICC", item = item)[,2]
png(file=paste("Excel-HW56-Item",item,".png",sep=""))
plot(0,0,xlim=c(-
3,3),ylim=c(0,3),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=paste("Differentia
l Item Functioning\nItem",item))
lines(x1,y1,type="l",col="red")
lines(x2,y2,type="l",col="blue")
legend(x = -3, y = 3, lwd = 1, legend=c("Group 1","Group 2"),col=c("red","blue"))
dev.off()
}
sapply(seq(1,10),makeplot)
```

The above code will have a few odd outputs in the console, but that's not important at this stage and level of work. What we do here is take advantage of R's data scoping to extract the X- and Y- coordinates of each ICC, then display them in a new, advantageous way based on our specifications. We build a function that can do this for any given item, then send instructions to do so with items 1-10.

Final Results

Code 60: Excel HW6 Input

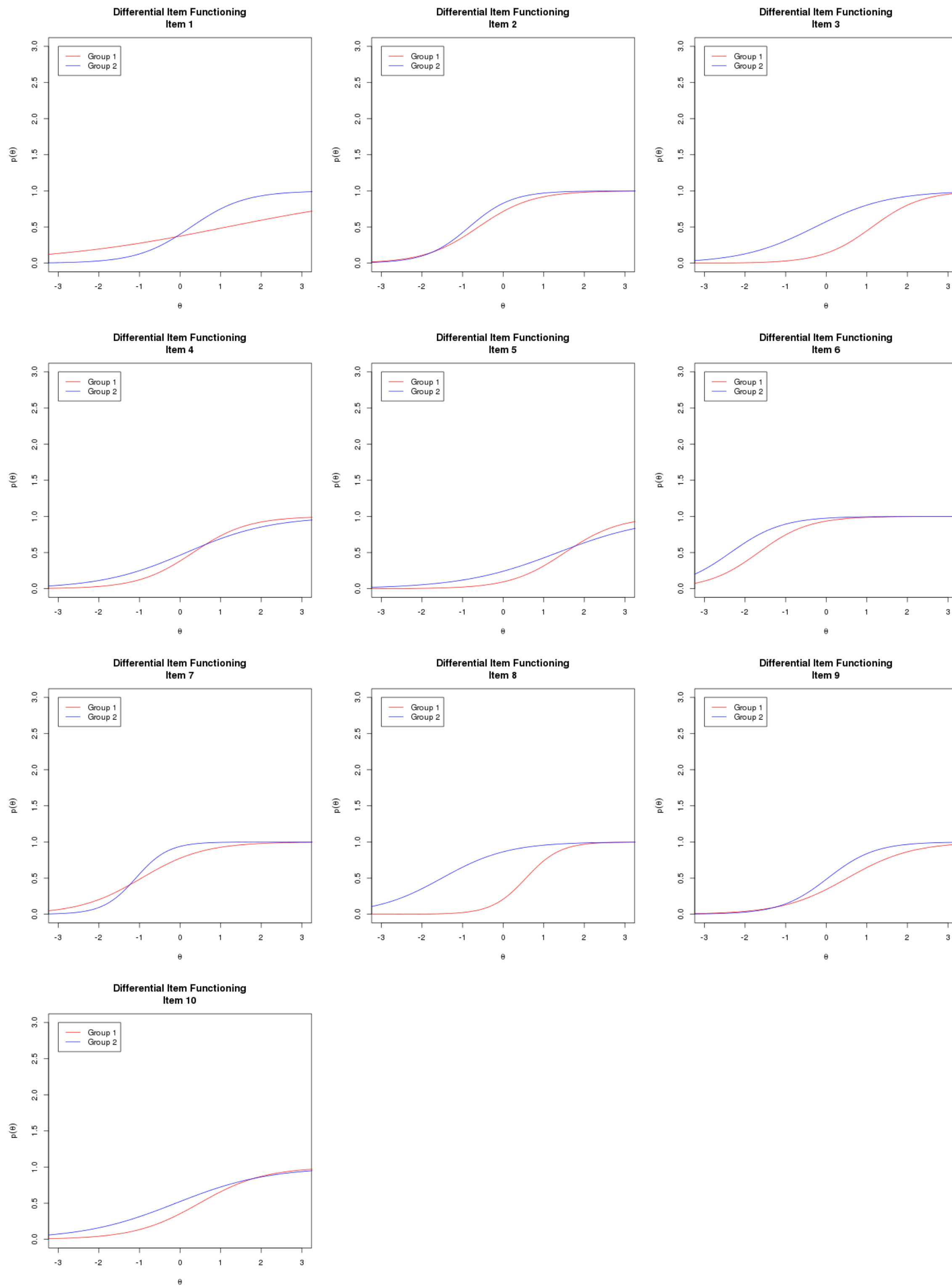
```
#Excel HW6
library(difR)
library(ggplot2)
hw6data <- read.csv("/media/sf_HOST/dn211dif.csv")
hw6test <- hw6data[TRUE,c(3,4,5,6,7,8,9,10,11,12)]
colnames(hw6test) <-
c("Item1","Item2","Item3","Item4","Item5","Item6","Item7","Item8","Item9","Item10")
attr(hw6test,"group") <- hw6data$Group
hw6dif <- difRaju(
    Data = as.matrix(hw6test),
    group = attr(hw6test,"group"),
    focal.name = 1,
    same.scale = FALSE,
    model = "2PL"
)
sink(file = "ExcelHW6.txt",append = FALSE, split = TRUE)
print(hw6dif)
sink()
group_1 <- tpm(
    data = hw6test[attr(hw6test,"group") == 1,],
    type = "latent.trait",
    constraint = cbind(seq(1:10),1,0),
)
group_2 <- tpm(
    data = hw6test[attr(hw6test,"group") == 2,],
    type = "latent.trait",
    constraint = cbind(seq(1:10),1,0),
)
makeplot <- function(item) {
    x1 <- plot(group_1, type = "ICC", item = item)[,1]
    y1 <- plot(group_1, type = "ICC", item = item)[,2]
    x2 <- plot(group_2, type = "ICC", item = item)[,1]
    y2 <- plot(group_2, type = "ICC", item = item)[,2]
```

```
    png(file=paste("Excel-HW6-Item",item,".png",sep=""))
    plot(0,0,xlim=c(-
3,3),ylim=c(0,3),type="l",xlab=expression(theta),ylab=expression(p(theta)),main=paste("Differentia
l Item Functioning\nItem",item))
    lines(x1,y1,type="l",col="red")
    lines(x2,y2,type="l",col="blue")
    legend(x = -3, y = 3, lwd = 1, legend=c("Group 1","Group 2"),col=c("red","blue"))
    dev.off()
}
sapply(seq(1,10),makeplot)
```

Figure 64: Excel HW6 Output

```
Detection of Differential Item Functioning using Raju's method
with 2PL model and without item purification

Type of Raju's Z statistic: based on unsigned area

Engine 'ltm' for item parameter estimation

Raju's statistic:

       Stat.   P-value
Item1  -1.1545  0.2483
Item2  -1.4217  0.1551
Item3   2.2838  0.0224 *
Item4   1.3486  0.1775
Item5   0.8661  0.3865
Item6  -0.0554  0.9558
Item7  -1.5553  0.1199
Item8   2.9140  0.0036 **
Item9  -0.9817  0.3263
Item10  0.6810  0.4959

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Detection thresholds: -1.96 and 1.96 (significance level: 0.05)

Items detected as DIF items:

 Item3
 Item8

Output was not captured!
```

Figure 65: Excel HW6 Output, Continued



Evaluation and Expansion

The scope of this guide is somewhat ambitious. We've covered no small amount of material and touched on some valuable programming paradigms. However, it remains true that we haven't set many open-ended challenges. These can easily be found for the motivated learner by using the questions from the end-of-chapter exercises in the Hambleton, Swaminathan, and Rogers text. This document is probably best used as a springboard for further, independent exploration to the degree desired by the reader.

The free and open nature of R aligns with the virtues of scholarly research. If you've found R to be effective or interesting, consider trying to use it to replicate previous work you've done in other statistical software packages. During these exercises, as well as any imported from the IRT textbooks available, it can be valuable to have an overarching paradigm to approach what is, for all intents and purposes, statistical programming. R provides an interface powerful and flexible enough to meet the needs of the theoretical as well as the practical researcher.

This guide will conclude with a simple piece of expansion material. Much of statistics and programming can be thought of as algorithmic work, and no small amount of pedagogy has been dedicated to helping achieve mastery of this developmental framework. As your adventures in statistics and R continue, consider the advice of George Pólya in "How to Solve It:"

1. Understand the problem
2. Devise a plan
3. Carry out the plan
4. Look back on your work

Table of Figures

Figure 1: Objectives 2
Figure 2: Prices of Software 3
Figure 3: VirtualBox Main Window 4
Figure 4: Naming the New Machine 4
Figure 5: Allocating RAM 5
Figure 6: Create a Virtual Hard Drive 5
Figure 7: Virtual Disk Image Specification 5
Figure 8: Fixed Disk Size 6
Figure 9: Disk Size to 12GB 6
Figure 10: Confirm VM Creation 7
Figure 11: Mounting a CD-ROM Image 8
Figure 12: Shared Folders 8
Figure 13: First Login 9
Figure 14: The Guest Additions Auto-Run Dialog 9
Figure 15: Root Command Prompt 10
Figure 16: The geany IDE Initial View 11
Figure 17: Geany IDE with R 12
Figure 18: Saving R Code 13
Figure 19: Saving the Machine State 13
Figure 20: Saved Machine State 14
Figure 21: Selecting a Machine for Export 14
Figure 22: Selecting Export Options 15
Figure 23: Reviewing Export Metadata 15
Figure 24: File Browser Main Window 16
Figure 25: Root FileSystem 16
Figure 26: Media Folder 17
Figure 27: Bookmarked Directory 17
Figure 28: Text file busy error 20
Figure 29: Saving to virtual machine desktop 21
Figure 30: Copying a file 21
Figure 31: Debian Desktop 22
Figure 32: Starting geany IDE 23
Figure 33: IDE Loaded 23

Figure 34: Terminal View 23
Figure 35: IDE with R Terminal..24
Figure 36: Saving a Syntax File 24
Figure 37: Code Highlighting after File Save 25
Figure 38: Session 0 Code Output..26
Figure 39: Adding some comments..26
Figure 40: An R help file 27
Figure 41: Syntax output..27
Figure 42: Beginning package installation 29
Figure 43: Confirming personal directories, selecting a mirror 29
Figure 44: R is case sensitive!..30
Figure 45: Successful package installation..30
Figure 46: Libraries loaded 31
Figure 47: Summary command output 31
Figure 48: Capturing output..32
Figure 49: Session 0 Output File 32
Figure 50: Default syntax output appearance in windows notepad 33
Figure 51: Fixing line endings 33
Figure 52: Fixed line endings..34
Figure 53: ICCs for the Class10 3PL 35
Figure 54: Exported image..35
Figure 55: Excel Homework 1 Output..39
Figure 56: Excel HW2 Output..41
Figure 57: Error: R version insufficient for library 42
Figure 58: Software Upgrade Messages..43
Figure 59: R Upgraded..44
Figure 60: Excel HW4 Output..48
Figure 61: Excel HW6 Output..51
Figure 62: difRaju documentation excerpt..53
Figure 63: difRaju initial output..53
Figure 64: Excel HW6 Output..55
Figure 65: Excel HW6 Output, Continued..56

Table of Code

Code 1: Installing Kernel Headers for the i486 kernel 10
Code 2: Installing VirtualBox Guest Additions 10
Code 3: Install the R Base System, development files, and a text editor 10
Code 4: Adding User to the Shared Folders groups 10
Code 5: Restart the Machine 11
Code 6: Starting R inside a Terminal 11
Code 7: Starting R in vanilla mode 24
Code 8: Session 0 Syntax 25
Code 9: Session 0 Syntax 25
Code 10: R help file access 26
Code 11: Session 0 Syntax 27
Code 12: Relative folder location setup 28
Code 13: Installing R add-ons via CRAN 28
Code 14: Package installation, continued 30
Code 15: Loading packages 30
Code 16: Summary command 31
Code 17: Saving textual output to a file 32
Code 18: Downloading a file conveniently 34
Code 19: Loading a CSV into an R dataframe 34
Code 20: Loading the ltm library 34
Code 21: Generating 3PL ICCs 34
Code 22: Saving Class10 3PL ICC to a file 35
Code 23: Excel HW1 Data Frame 36
Code 24: Excel HW1 P-Indices 37
Code 25: Excel HW1, continued 37
Code 26: KR-20, KR-21 37
Code 27: Advanced Coding Hint 38
Code 28: D-Index 38
Code 29: D-Index Upper Group Selector 38
Code 30: Excel Homework 1 Input 38
Code 31: Session 1 Bonus Hint 1 39
Code 32: Session 1 Bonus Hint 2 39
$P\theta=c+1-c1+e-1.7a\theta-b$ Code 33: A Custom 3PL Function 40

Code 34: Generating ICCs 40
Code 35: Excel HW2 Input 41
Code 36: Installing and Loading the Person Parameter Library 42
Code 37: Adding a Package Repository to the Operating System 43
Code 38: Upgrading Internal R Packages 44
Code 39: Installing the Person Parameter Library 44
Code 40: Creating the appropriate data objects 45
Code 41: Examining the workspace 45
Code 42: Ability Estimation 45
Code 43: Excel HW3 Input 46
Code 44: Excel HW3 Output 46
Code 45: Creating a dataframe for graphing 47
Code 46: Drawing ICCs 47
Code 47: New line drawing, modified legend 48
Code 48: Excel HW4 Input 48
Code 49: Making a TCC function 49
Code 50: Graphing a TCC 49
Code 51: Creating the Item Information Functions 50
Code 52: Graphing some IICs 50
Code 53: Making the Test Information Curve 50
Code 54: Excel HW5 Input 51
Code 55: Importing Data 52
Code 56: Some data cleaning 52
Code 57: Install difR package 53
Code 58: Calling the difRaju function 53
Code 59: Replicating Plots 54
Code 60: Excel HW6 Input 54