ECON 41701: Homework 1

Instructor: Benjamin S. Skrainka

Due: 9 April 2012 at the start of class

These problems are due in class at the start of lecture. For Unix exercises, you may submit a hardcopy or PDF electronically. Make sure that you show both the answer and how you computed it. Do not show results from commands which produce voluminous output, i.e., do not show listings for directories with thousands of files. For programming exercises, you should submit your source code and/or scripts – including a script to build your software or a Makefile – as a gzipped-tar file via email to Mathilde (malmlund@gmail.com).

Problem 1: Unix Essentials

This problem tests basic Unix knowledge and will walk you through some of the problems you may encounter when working with large scale simulations or data. We will examine actual log files produced by solving the Berry, Levinsohn, and Pakes (1995) model. (These experiments are described at length in Skrainka (2012). You do not need to read either of these papers to understand this problem.) Note: this homework will be easier if you have listened to or read all of the Software Carpentry lectures on Unix and looked at their example exercises. You will need to look up some commands in Sobell or with the `man` command. These problems are examples of what you will encounter every day in research and can solve quickly once you know a little Unix. If you want Unix to generate a log of your session, use `script`.

1. **Determining File Types.** The files for this exercise are in the file `/home/skrainka/econ41701/hw0/hw.log.tar.gz`. How big is this file? What kind of file is it? What command can you use to find out?

2. **Checksums.** When copying or downloading files, you can verify their integrity by computing a checksum or secure hash. MD5 and SHA-1 are two popular algorithms, though SHA-1 is more secure. Check that the hashes for `hw.log.tar.gz` match those in `md5.txt` and `sha1`.txt. The commands to use are `md5sum` (`md5` on OS/X) and `sha1sum`. When downloading files or software from the Internet, it is good to verify the hash if possible.

3. **Tar & Counting Files.** Create a directory to work in. Extract the files from `hw.log.tar.gz`. The format of this file – a gzipped tarfile – uses two standard Unix methods for grouping and compressing

files (`tar` and `gzip`, respectively). This is analagous to zipped files on Windows. The suffix `.tar.gz` indicates that the file is in this format.

To see how the files are laid out, execute:

```
tree | less
```

`GenData.T1.J12` contains subdirectory, `log`, and Portable Batch System (PBS) job files to run simulations for the BLP model for T=1 market and J=12 products. (You will learn how to write PBS files to run jobs in parallel later in the course. They are just bash shell scripts with some extra directives for the job manager on the cluster. Hence, the need to master Unix basics...) Cd to the `log` subdirectory. This directory contains the log files for runs to generate synthetic data and estimate the BLP model.

- What are the different types of log files?
  [Hint: you can strip the terminal '***-n***' from filenames with `sed 's/-.*//'`. By default, sed reads one line at a time from standard input into its edit buffer and performs whatever editing commands you specified. In this case, 's/*pattern*/*replacement*/' is like Find & Replace in an editor: it searches for *pattern* and replaces it with *replacement*. Patterns can use regular expressions, hence the power.]
- How many of each file type are there?
- Advanced optional question: compute this using a loop so you don't need to enter the command(s) for each file type. [Hint: man bash and read about 'for'.]

4. **Solver Exit codes.** These log files are just a small subset of those created in my paper. Files which end with suffixes like '.o1234567-89' are output files from the job and contain whatever was written to standard output. Those which end with '.e1234567-89' are capture what was written to standard error. Each job has both a .o and a .e file. The root of the name describes the type of job:

   *BLP.char.T1.J12.pbs.** Estimate BLP using characteristics of rival goods for IV

   *BLP.cost.T1.J12.pbs.** Estimate BLP using exogenous cost shifters for IV

   *Job.GenData.T1.J12.pbs.** Generate synthetic data sets.

   *Re.char.pbs.** Estimate BLP using the optima found for BLP.char.T1.J12.pbs.* (i.e., restart)

*Re.cost.pbs.** Estimate BLP using the optima found for BLP.cost.T1.J12.pbs.* (i.e., restart)

Each file contains results from estimating the model several times (It is important to group fast jobs together to minimize scheduler overhead.) We need to figure out which jobs failed and how much time they used. Look at one of the BLP.char.T1.J12.pbs.o* output files. When the solver completes, it prints the exit code as 'SNOPTA EXIT' followed by the exit code.

- How many times was the model estimated for these runs?
- How many times did the solver find an optimum?
- What were the exit codes when the solver failed?

5. **Job Runtimes.** Look at the prologue (header) and epilogue (end) of a BLP.char.T1.J12.pbs.o* file. [Hint: in less or vi, what command will take you to the end of the file?] The prologue and epilogue are generated by the job manager and provide basic information about a job. In particular, the lines labeled 'Limits' and 'Resources' in the epilogue show, respectively, the resources limits which were set for the job when it was run and the resources which were used. On the Resources line, the element 'walltime=' shows how much time the job used in hh:mm:ss format (h = hours, m = minutes, s = seconds).

   - How much time was used by the fastest job?
   - How much time was used by the slowest job?
   - What was the median execution time for a job?

6. **Aliases & Functions & Shell Scripts.** You can now see the power of Unix for quickly figuring out which jobs ran successfully and how long they took without reading millions of log files. To automate this, it is common to setup shell aliases, functions, and scripts.

   - Setup create a function to find the slowest job

```
$ slow()
{
fgrep Resource $* | sed 's/.*walltime=//' | sort | tail -1
}
$ # Use it!
$ slow BLP.char.T1.J12.pbs.o1392086-*
00:02:53
```

Normally, you would put this in your .bashrc so it would be automatically created every time you log in. The shell variable $* expands to be everything after the command slow, i.e. all the BLP.char.T1.J12.pbs output files.

- Aliases should also be defined in your .bashrc. The don't take arguments so it is better to use a function or shell script. If you are sick of typing long commands, you can setup aliases either at the command line or .bashrc. Here are some aliases I use:

```
alias athens='ssh YourUsername@athensx.uchicago.edu'    # On my MacBook
alias h='history'
alias l='ls -F'
alias ll='ls -lFh'
```

- Shell scripts are text files which run as programs. First create a file fast.sh to find the fastest job which contains the following lines:

```
#!/bin/bash
fgrep Resource $* | sed 's/.*walltime=//' | sort | head -1
Then set permissions so that it is executable:
$ ls -l fast.sh
-rw-r--r-- 1 skrainka skrainka 72 Mar 31 13:56 fast.sh
$ # Make it executable
$ chmod +x fast.sh
$ ls -l fast.sh
-rwxr-xr-x 1 skrainka skrainka 72 Mar 31 13:55 fast.sh
$ # Execute it
$ ./fast.sh BLP.char.T1.J12.pbs.o1392086-*
00:00:07
```

Typically, you create a directory ~/bin to contain shell scripts which you use regularly and add ~/bin to your PATH.

7. Here is a bit of Unix fun: there is a command finger which lets you check basic data on other users. The syntax is:

```
finger user
```

user can be a username, first name, or surname. Run the command on yourself. You can tell other users what you are doing by creating a .plan file in your home directory. Create a .plan file using your favorite editor. Now if anyone fingers you, they will be impressed by your Unix knowledge.

Problem 2: Basic Programming: Rabbits and Coyotes

In this problem, you need to compute the number of rabbits and coyotes which exist on an island. Let $r(t)$ and $c(t)$ be the number of rabbits and coyotes at time t. Their population is governed by the system of equations

- $\frac{dr(t)}{dt} = \alpha r(t) - \beta r(t) \times c(t)$
- $\frac{dc(t)}{dt} = -\gamma c(t) + \pi r(t) \times c(t)$

where $\theta = (\alpha, \beta, \gamma, \pi)$ are non-negative constants and the number of of rabbit and coyotes is non-negative. I.e., $\theta \geq 0$, $r(t) \geq 0$, and $c(t) \geq 0 \; \forall t$. Your program should:

1. Read configuration parameters, starting population levels, number of periods, and step size from standard input in the following order: α, β, γ, π, r, c, N, and h (See Table 1 for some possible starting values). Then, you should calculate population levels for each period and write out each period's results t, r, and c on a new line using spaces as a delimiter. For example, period 10's results might be:

```
10 0.9 0.3
```

Parameter	Value
α	1
β	1
γ	1
π	1
$r(0)$	0.7
$c(0)$	0.1
N	10000
h	0.001

Table 1: Initial Values

2. Plot your results using R, MATLAB, gnuplot, or equivalent.
3. How would you solve this system of differential equations for the phase space trajectories?