R Homework 7: Scaling Up: Lists, Apply, Numerical Integration, and Analysis

Marguerite A. Butler

March 10, 2018

DUE: Saturday March 17, Midnight on GitHub

Instructions

In this homework, we will be getting more practice at scaling up the workflow and some analaysis: file access with multiple files, processing of raw data (smoothing, subsetting, and getting area under the curve), practice with apply functions (sapply), and regression analysis.

We are going to analyze data from an experiment performed by Jeff Scales. It is commonly assumed that a cost of pregnancy is reduced locomotor performance resulting from weight gain associated with pregnancy. However, there are at least two effects during pregnancy, increased mass as well as increased volume. He performed an experiment simulating pregnancy in green iguanas (animals which normally run well and also lay a large number of eggs) in order to test whether an increase in mass or volume was the greater problem for females and how fast they can run. He simulated pregnancy by implanting iguanas and filled them with either air (a), saline (s), or leaving them empty (e).

These data are from iguanas running over a force plate. As the animal steps on the plate, the force plate records the ground reaction forces of the step in three dimensions with respect to time (Fx: (force in x-direction), lateral or side-to-side, Fy: propulsive, Fz: upward). From the force trace, we can calculate peak force (maximum force) as well as impulse (the sum of forces applied during the entire step, or area under the force trace), which are both important biomechanical quantities. Here we are just going to play with Fy, which is the force in the direction of movement.

Let's say we want to automate the analysis of multiple files, and obtain the values from our raw data traces that we can use to compare individuals or treatments. We'll also use numerical integration (the area under the curve defined by our datapoints if we were to connect-the-dots) to obtain the total force or impulse from each trace.

Therefore, the tasks we need to accomplish are summarized in flowchart 1.

The data are in a folder named "Iguanaforce_data". Place this entire folder (Iguanaforce_data) in the same directory as your script. We will also be using penalized splines to smooth data, which is in a new package. Please install the `pspline` package. From within R you can do at the command prompt:

```
> install.packages("pspline")
```

Or you can install it from the package installer (a menu option in your gui interface).

Some useful functions to help you

A function to read in one file and save it as a dataframe. Pretty self-evident at this point in the course.

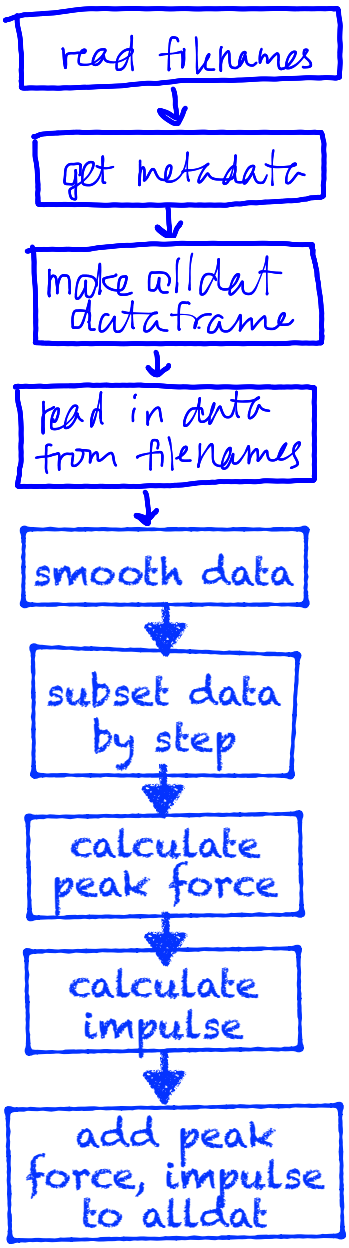


Figure 1: The basic steps in the program. This works fine if you only have to deal with one file.

```
> ##### reads in data  ######
> readdata <- function( filename, path="Iguanaforce_data")
+ {
+   myfile <- paste(path, filename, sep="/")
+   dat <- read.table(myfile, header=F, skip=13)
+   names(dat) <- c("time","Fx","Fy", "Fz", "Ft", "aux", "pc1","pc2","pc3","pc4")
+   return(dat)
+ }
```

The next function smooths data. Force trace data are often noisy, and we usually want to smooth the data. First load the pspline package using `require(pspline)`. Use the function below on `dat`. What it will output is a single column dataframe. By default, we've written the function to work on the column named "Fx", and we would typically save the smoothed Fx output as `Fx.hat` (see below in the problems).

The function call to `smooth.Pspline` smooths data according to provided parameters, creating a smoothing spline (this is similar in concept to the numerical derivative function you coded previously, but here we are interested in smoothing the data by taking basically running averages), and the call to `predict` then generates the smoothed (predicted) data values according to the intervals specified by the input data and the spline function (it is also possible to make predictions at different intervals). The beauty of smoothing splines is that it is also possible to estimate curves of the first and second derivative. Although we don't use these for force trace data, we do when we digitize video of locomotion, where the original data is position, the first derivative = velocity, and the second derivative = acceleration.

Note that the function is set up so that we pass it two columns, `yy` is the response variable that we're trying to smooth, for example force Fy, and `xx` is the independent variable, typically time in this experiment. What is returned is a single column dataframe of smoothed data (i.e., Fyhat), which can be added on as an additional column to the original dataframe.

```
> require(pspline)
> ##### smooths data  ######
> smdat <-function (yy, xx, sspar=10^-12,order=3, j=1)
+ {
+   sp <- smooth.Pspline(xx, yy, spar=sspar, norder=order, method=j)  ## the spline
+   smootheddata <- predict(sp, xx, nderiv=0)     ## returns smoothed data points
+   return(smootheddata)   ## as single-column data frame
+ }
>
```

When you have an experiment with many datafiles, it is convenient to have them all in the same directory, and named so that each filename is uniquely identified. Furthermore, if you include metadata in the filenames, you can use the filenames to grab just the files that you want. Just think what a headache it would be if you had separate directories for each individual, for example, but you had to put them together by treatment.

These data files are named with a code which uniquely identifies each trial. For example, in the file "071106B502s.txt": The first six digits (071106) are the date, B5 is the animal's ID, 02 is the trial number for that day (multiple trials are run per day), and "s" is the experimental treatment ("a" and "e" are the other treatments). You can populate the output dataframe with this metadata.

Problems (What you turn in)

1. Produce a script for the workflow for one file to obtain peak force and impulse (area under the curve) for Fy, along with date, trial, id and treatment. (Workflow shown in flowchart 1). You may want to use one of the the numerical integration functions you wrote in the last homework. Make sure you check your smoothing by creating pdfs of your raw data with your smoothed data as a line overlay (see below).

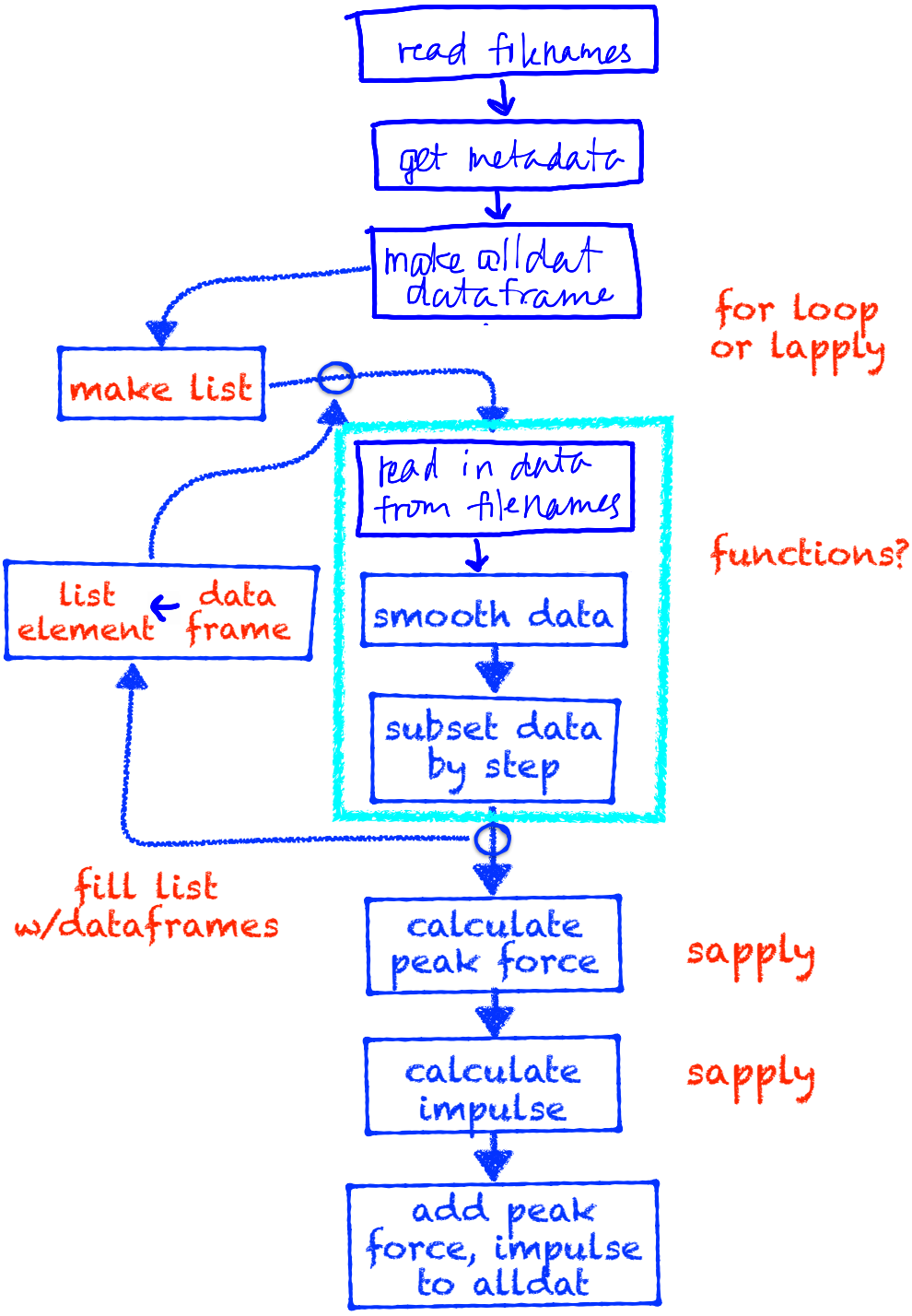


Figure 2: If we have 16 files, or 1000 files, we don't want to repeat the same lines of code over and over! We want to write functions so that we can put the steps that produce each data frame into a function that can be used in a loop or in an apply method to automate the work.

2. Write a script that performs the same workflow over all the files (workflow shown in flowchart 2). Use `apply` functions (if you like, you can practice using `for loops` but turn in the version with apply functions). Save the output to `.csv`. It should look like the dataframe below. Save the list of data to an `.Rdata` file (you may want to reanalyze the data later). What is the difference in your code using `apply` functions versus what it would look like if you used `for loop`s?

3. Perform a linear regression of impulse (y-axis) as a function of peak force (x-axis) using your own matrix math computations. What is the estimate of the slope and intercept? Compare your results in the last part with results from built-in R functions. Make a plot and make sure that your results look reasonable. What parts of the regression results are "significant" and what does that mean in real life?

4. Try out some other things to explore the data.

Hints/Useful info

- **Getting trial data from filenames** The first 6 characters of the filename is the date, the next two is the individual id, the next two is the trial number, and the next character is the treatment (air, empty, or saline). Extract the `date`, `trial`, and `direction` from the filenames and make separate objects for each one. You may find the following functions helpful. Please take a look at the help pages for each.

  strsplit String split splits a character string at whatever character you specify

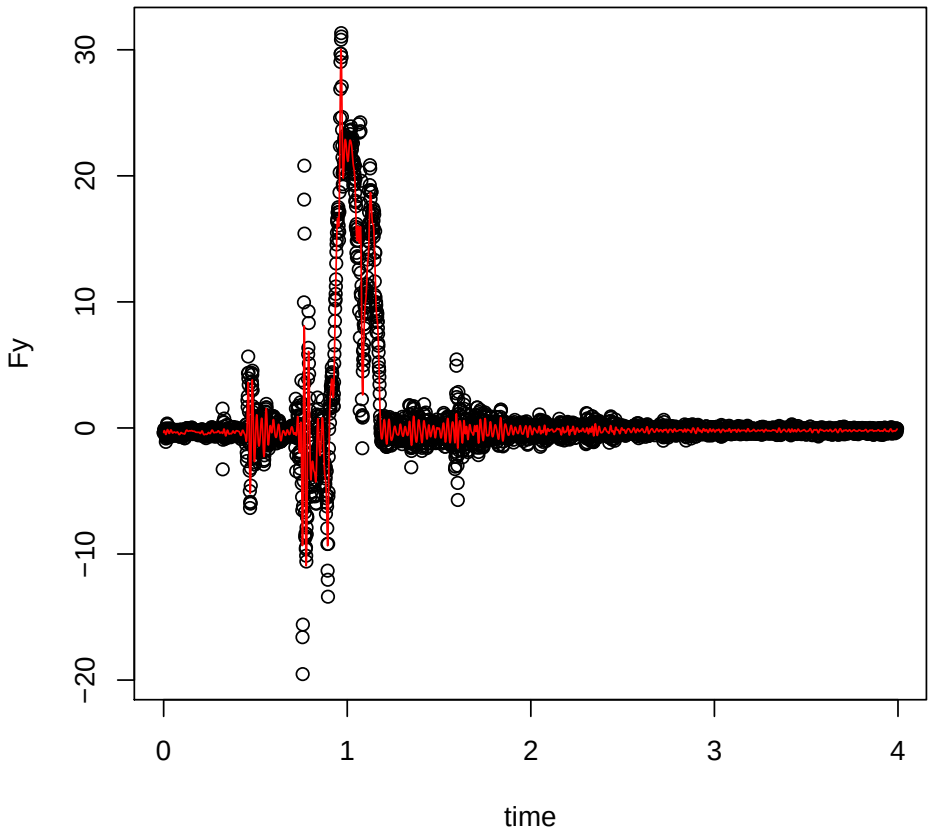
  substr Substring selects a portion of a character string, from the first to last argument. (i.e., It counts the characters).

  gsub Gsub is useful for replacing part of a character string, based on matching whatever characters you specify. You can replace it with nothing `''`

  Look at the help page for `strsplit()`. Below is an example of how you could use this function. Can you figure out what each one does?

  ```
  > dd <- strsplit(files, split="_", fixed=T)   # what does this do?
  > date <- sapply(dd, function(x) x[1])   # what does this do?
  ```

- Check how much your data are being smoothed. Plot the original data as points and the smoothed data as a red line overlay. It should look something like this:



- When you write functions in your code, you want to put the function definitions at the top of the script. Try to think about writing modular code. In general, it's one task, one function. You can combine multiple tasks by putting several of your simple functions inside of a larger function to mimic the hierarchy of your program flow. In any case, your function definitions should be at the top, and once the functions are perfected (meaning you have the inputs and outputs that you want), you shouldn't mess with them anymore. Each function should do one task, and each function should be defined only once. In the body of the script, you will use the functions to generate your objects. You can put your function calls inside loops, inside other functions, or just on simple lines of code. They are like lego pieces – you put them together in creative ways to build your fabulous code.
- You can he call to source loads the script into R, just as if you typed it in at the console. It is good programming style to define functions once and only once. This makes code maintenance much easier. If you want to edit your function, you just change it in the function script, and it will be updated for all other scripts that depend on those functions too.

```
> require(pspline)   # top of your script file
> source("MYFUNCTIONS.R")   # replace with your function script name
>
> # lines of code to do the tasks
```

- Clean up the script, putting all functions at the top, with comments to help us understand the

code.

- Lists are useful to hold multiples of the same kind of data. Seting up an empty list to hold your data frames, allows you to then use the functions to fill the elements of the list with the processed data frames. You can then use `sapply()` functions to operate over your data list and grab the information that you need.
- **Master Script** For larger projects, you can write a main script that will run your smaller scripts. This is done by sourcing your files within your script. Create a master script that sources the functions or other scripts that you need to read in the data, create the data files, and analyze peak force and impulse by treatment. Make sure you comment it with a header block explaining what the code does in general, and then a line of explanation for each script. What does it do, what input is required and what is the output (especially if it creates any files)?