Lecture 10:

Last time

We closed our treatment of Perl by considering various ways to extend the language; we introduced the structure of subroutines and described how make use of external modules, collections of subroutines and other code that are written by others.

We also looked at our ability to program “on” the language with the function `eval`; this turns out to be extremely powerful highlights an important aspect of the languages we have seen so far (including R).

Today

We take a bottom-up approach to working with data in R; we will start with matrices and vectors and slowly add detail.

Today we will cover some basic subsetting operations and a bread-n-butter plot command.

Our data come from an actual experiment conducted by a robotic mapping platform.

Thermal mapper

To characterize the thermal properties of alpine plants, a robotic scanning device was designed to take measure the surface temperature over a 3m transect every 1cm.

A complete pass of the transect took about 10 minutes and the robotic device ran semi-autonomously from 8:30 am August 1 to 8:40 am August 2 (of this year).

The robot carried with it an infrared temperature sensor as well as a camera to examine the ground cover.
Thermal mapper

The experiment took place near the White Mountains Research Station (elevation 12,800’)

The site was chosen because it exhibits extreme temperature variations; air temperature could be as low as 10°C, while the surface could be closer to 50°C.

The data

Given the regular sampling plan employed by the robotic sensor node, we can arrange the observational data in the form of a rectangle.

Today we will consider the simplest possible data structures R knows about; over the course of the next two lectures, we will gradually move up to higher components of the language.
Simple, but...

Don't be fooled by the certainty implicit in an Excel spreadsheet; this regular grid fails to expose a significant normalization process that took place.

It turns out that the robot's position was subject to drift; with each pass the recorded position was off by a little bit.

The engineers corrected for this effect in the field, but what are we to make of the changes?

**Moral:** Be involved in the data collection; be facile enough with computing that you can evaluate the corrections on your own.

---

**Getting started**

You start an R process in a Unix terminal window simply typing the command `R`; this will eventually give you a **prompt**.

R has a pretty simple user interface; the most basic form of interaction involves entering expressions, having the system evaluate them and then print a result (sound familiar?)

The prompt indicates that R is waiting for input.

---

**Loading data**

There are many paths into R; we can scan various kinds of files, even those used by different languages.

But, in the spirit of keeping things simple, we will load data that I've dumped* into a file and placed on my website; you can bring it into R with the following command.

```
source("http://www.stat.ucla.edu/~cocteau/stat202a/whitemountains.R")
```

This command will create a series of objects in your **workspace**; you can examine them with the commands `objects()` or `ls()`.

* Remember Perl's Data::Dumper? The `dump()` command in R is similar.
Objects

R thinks in objects; everything in your workspace is some kind of object.

Among the objects we have loaded, three are vectors and one is a matrix.

Unlike Perl, vectors and matrices can hold only one type or mode of data; numeric, complex, logical, character, or raw.

Objects

For a vector or a matrix, the mode of an object represents the basic type of data it contains.

Since everything in R is an object, R knows about more than the five modes available for vectors and matrices; later we will see objects of mode list and function.

The mode of an object is one of its basic “properties”; another basic property of any object is its length.

Note that what we might think of as a scalar value (a single thing) like the number 5, is really a vector of length 1 with mode numeric.
Objects

The length of a vector is what you would expect; the number of basic items it contains.

All objects have a length, but it might not be as useful a number as the length for a vector.

In addition to mode and length, all objects in R have a class; for vectors the class is just its mode, the type of data it contains.

---

Ok, enough chatter...

Let’s consider some of the data; the objects in our directory can be described in words as follows:

- `groundcover`: A vector of length 307 indicating either a kind of plant, a patch of sand (small rocks) or a boulder (um, a big rock).

- `positions`: Also a vector of length 307, describing the distance of the spot in cm from a fixed reference.

- `scantimes`: A vector of length 148 representing the times each scan ended.

- `temp`: A 307x148 matrix representing the temperatures collected by the robotic node.
Having a look

R provides simple printing of objects; if you enter the object's name it will print (in some form)

For an array, this is usually a formatted display that begins with a column of indices (not part of the object; these are added for readability)

Here's what we get when we enter groundcover

Ok, not very helpful

For character vectors, we might want to have a look at the number of different kinds of ground cover we have in the data

We can use the built-in function table to give us these values; we will also store the output from this command and have a look at its mode, length and class
Subsetting

Notice that the first time we called `table`, the result was just printed on the screen; the second time we captured the output and stored it in a new object (called `x`)

One important feature of these data is that they occur in “clumps”; patches of sand are followed by patches of one kind of plant and so on

We can explore some of that by printing out pieces or subsets of the vector `groundcover`; in our example, we looked at the first ten entries (note that unlike Perl, R indexes its arrays starting with 1)

Subsetting

This is an example of subsetting with an index; here are a few more

```
groundcover[6]  # the sixth element
groundcover[6:10]  # the 6th through the 10th elements
groundcover[c(1:5,8,11:15)]  # a group
```

Generating sequences

The construction `n:m` is shorthand in R; it creates a vector holding the integers between `n` and `m` (ascending or descending depending on whether `m` is larger than `n`)

There are several other convenience functions that help with this; for example

```
rep(9:11,3)         # 9 10 11 9 10 11 9 10 11
rep(9:11,rep(3,3))  # 9 9 9 10 10 10 11 11 11
seq(1,10,by=1)      # 1 through 10
seq(0,1,len=21)     # 0.00 0.05 0.1 ... 0.90 0.95 1.00
```

Concatenating vectors

Finally, the function `c(...)` takes any number of vector objects and returns the vector formed by concatenating them end to end

Because vectors can only hold one basic type of data, if you try to combine vectors of different types, R will coerce the data into a common form

You can do this on your own with built-in functions like `as.character`, `as.logical` and `as.numeric`
Ok, enough chatter

The real meat of the problem lies with our temperature data; we'd like to have a look at subsets in either space or time.

We can also subset matrices by index:

```
temp[1,]  # the first row of temp
temp[,1]  # the first column of temp
temp[1:5,1:5]  # the upper lefthand corner
```

Having a look

Notice that when we extract a single column or row from a matrix, R converts the result to a vector.

In our plot command, we have used a simple index (1:148) to represent time; recall that it takes the robot roughly 10 minutes for a pass.

Technically, if we just call `plot(temp[1,])` we get the same figure; in general, if `x` is a vector, in `plot(x)` R assumes that you want to treat the data like a time series and uses `1:length(x)` for the `x`-axis.
Having a look

With subsetting and the plot function, we can start to explore the basic hypothesis that different ground covers behave differently over the course of the day.

We note that the points in black correspond to a grassy plant (poa) while the cyan dots are classified as sand.

What can we say about sand and poa in general?
Having a look

• The function apply allows us to operate on rows or columns of a matrix.

In this case, we have taken the data in each column and formed an average with the built-in function mean.

The "2" in this call tells R to operate on the data in a column; if we had used "1", it would apply the mean function to each row.

What do we notice?

• It would seem that one difference between a plant’s response to changing ambient temperature is different than that of the sand.

We might consider modeling the data around the “time” 25 as a quadratic function and examine the curvature at 25.

Let’s see how to do this...

Old school: For illustrative purposes ONLY

What we’d like to do is solve a small least squares problem involving just the first 50 points.

We’d like to fit a quadratic to minimize

\[ \sum_{i=1}^{50} (y_i - \beta_0 - \beta_1 i - \beta_2 i^2)^2 \]

If we let \( y = (y_1, y_2, \ldots, y_{50})^T \) and

\[
X = \begin{pmatrix}
1 & 1 & 1 \\
1 & 2 & 4 \\
1 & 3 & 9 \\
\vdots & \vdots & \vdots \\
1 & 50 & 2500
\end{pmatrix}
\]

then \( \hat{\beta} = (X^TX)^{-1}X^Ty \)
Some explanation

Just as \( c(...) \) concatenates data into a vector, \( \text{cbind}(...) \) will try to combine its arguments into a matrix

If the vectors are all the same length, it works out great; if not, R invokes something called the recycle rule

If R needs vectors of a common length, but is given vectors of different lengths, it will recycle data (sometimes fractionally, but with a warning) of the shorter vectors to bring them to the length of the longest

Some explanation

In our first command, notice \( x^2 \); this is the first example of what we refer to in R as a vectorized operation

R will apply operations to entire vectors or matrices; we don’t have to write loops to create a vector representing the squares of the elements of \( x \)

\[
x^2 \quad \text{# another vector, elements are squares}
3 \times x + 5 \quad \text{# elementwise, multiply by 3, add 5}
2 + x + 3 \times x^3 + \sqrt{x} + \exp(x) \quad \text{# yikes!}
\]
Some explanation

So this means `cbind(1,x,x^2)` will create a matrix with three columns; it recycles the value 1 50 times to match the length of `x` and `x^2`.

Next, we encounter operators that work on matrices; `t()` returns the transpose of a matrix, `solve()` yields its inverse; and `%*%` represents matrix multiplication.

Do we see our normal equations now? Notice that the result is a matrix with just one column.

A more modern, less grubby approach

R has a leg up on Perl and the other languages you’ve met so far precisely because it has features to model data easily.

Let’s use a built-in function called `lm()` together with specialized extractors that help us pull out the interesting parts of the fitted model.
What kind of thing is fit? It's an object (everything in R is an object, and this is something in R, so it must be an object) but what kind?

We can use `class()` to figure it out; in addition to `print()` we can often use the `summary()` command to learn a little about an object.

And at the risk of getting a little ahead of ourselves, most of the objects you will encounter that represent "fitted" statistical models allow us to make predictions.

This is done with the command `predict()`; next time we will talk about what's going on here really, that is, why we can call `predict()` on a linear model object, a generalized linear model object, or even a tree object and get something sensible.

Here, if we call `predict(fit)`, we get the predictions from the model at the original design points (the points in $x$).

```r
> print(fit)
> summary(fit)
> class(fit)
[1] "lm"
> summary(fit)
Call:
  lm(formula = y ~ x)

Residuals:
    Min     1Q   Median     3Q    Max
-0.431  -0.150  -0.032  -0.009  0.267

Coefficients:    Estimate Std. Error t value Pr(>|t|)    
(Intercept) 22.964132   1.238702  18.39  < 2e-16 ***
   x          1.419938   0.115271  12.37  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.237 on 47 degrees of freedom
Multiple R-squared:  0.969,  Adjusted R-squared: 0.9684
F-statistic: 149.7 on 2 and 47 DF,  p-value: < 2.2e-16
```
Having a look

This kind of simple modeling has allowed us to (possibly) capture a feature of the temperature response that could help us differentiate between plant (poa) and sand.

To give you some experience with matrices and subsetting, here’s a brief assignment for next time.

Task

For Wednesday, consider the data at the end of the time period.

Rather than fit a quadratic, consider a line fit to the data from 125 on.

Do this for the two samples we considered today; the poa and the sand sample.

Turn in the slope associated with the two fits.