

This is a PARTIAL list of techniques that you'll want to be familiar with. If you're not, please experiment in R, use the help commands, and ASK the course instructors in office hours or class lab sessions. We're here to help, and all of us have questions about these things.

In addition, please make sure that you're familiar with the R functions and techniques that are listed in the sections of the Lab Manual that we have covered so far in class, and the commands used in class (please carefully look at all codes from class, which are posted online with class notes).

- Making .R files using the text editor. Say you make a code called "myprogram.R" – you save it in a directory, make sure that this is your working directory (see below), and type `source("myprogram")` at the command line to run this code. You can also give the absolute working directory, e.g. `source("/your/filepath/to/your/file/myprogram")`.
- Changing the working directory in R. The `getwd()` function gets you the current work directory. To set it to a different directory, use `setwd()`. If you're really lazy to type the directory or you're not sure what it is, you can drag and drop the file into the R command line in Windows. This will try and load the file (but fails); however, the location of the file will be given. For *nix operating systems, it is good to change to the work directory using the shell before starting R.
- Use `?functionname` to get some help from a function. Since R is open source, the documentation can be very terse (and it's reference-based). So it is common to use Google and the R help page to get more info. Go to <http://www.r-project.org/>, then click on **Search** then click on **Searchable mail archives** and there you will find the R help archives. You can also set up your browser (Firefox, Opera) to create a custom search by right clicking the search box.
- Use `ls()` shows all the commands you have in your current session. Use `search()` to show what packages are there. Use `install.packages()` to install a package, and `library()` to run installed packages. You can search for commands within a package by running `ls(package:stats)` (this is just for the stats package).
- Editing .R program files: just type `edit()`, and your default text editor comes up.
- To load data one line at a time, use `scan()`, to read a table that is comma delimited, use `read.table("mydata.txt", head = T, sep = ",")`, where the `head` option reads in the first line of the data as the column names. These are read in as data frames. To read a text file one line at a time while getting it as a character vector, use `readLines()`. You can use this to parse text data (say email messages or HTML code).
- Of course we can write data as well. `write()` is the opposite of `scan()`. Use the `append` option to append an existing text file. `write.table()` exists as well.
- R is an **object oriented language**. This means that each data structure (i.e. object in the R session) has some "stuff" affiliated with it. For example, we can create an lm "object" by doing `lm.object <- lm(y ~ x)`. This `lm.object` has a bunch of attributes used for different functions. Try stuff like `coeff()` and `summary()` to see what it does. The `names()` function gives you names of the stuff that you can extract from the object using the dollar sign, e.g. `lm.object$coeff` gives you the coefficients of the parameters. These structures make R very flexible and intuitive, but computations become slower.
- Prominent data types (you can check what data type by using `class()`). These are important to know the distinction to make certain functions work the way it's supposed to work.:
 - numeric vector: this is just a vector of numbers e.g. `c(1,2,5,3)`
 - matrix: this is a matrix like in MATLAB. You can do matrix multiplications using `%*%`. Multiplying a vector with a matrix automatically makes a vector a matrix (nx1). e.g. `matrix(rnorm(10,0,1),5,2)`. You can name matrices using `rownames()` and `colnames()`.
 - array: this is just a generalization of a matrix to 3 or more dimensions.
 - integer: use it for `for` loops and `while` loops. it automatically changes to a numeric when you put arithmetic operators e.g. `1:5`.

- character: each element inside a character vector is a character object. You cannot combine character and numeric vector because they are stored in memory in different ways. e.g. `c("doink", "dink", "dung", "deng")`.
 - factor: used in modeling, we can change both numeric and character objects into a factor, as they will behave differently. It basically organizes the data into discrete parts with no ordinality.
 - data frame: this is usually the type used for data. It's usually a bunch of numeric vectors of *same length* with different column names. Extract column names by using `names()`. It *cannot* combine two vectors of different types. e.g. character with a numeric.
 - list: A list puts *different* data types into one. So you can say `list(a,b,c)`, where `a` is a character vector, `b` is an array, and `c` is a data frame. Extract each components using the `$` sign. You can also have iterated lists. A list inside a list inside a list inside a list is also a list. Used often as output of a complicated function like `lm()`.
 - logical: R can have boolean variables and hence, vectors. Used inside `if` statements and `for` loops. So `x == 2` is a logical. It outputs either `TRUE` or `FALSE` to pass through a conditional statement. The shorthand of `T` and `F` also works.
 - other model objects:
 - or create your own!
- we can turn objects into different data types. For example, we get some data as a character vector, e.g. `c("heavy", "heavy", "light", "heavy", "medium"...)` . If we want to use it inside model functions such as `lm` we need to turn into a factor. So we can either use `factor()` or `as.factor()`. To check, use `is.factor()`.
 - `length()` shows the length of a vector. `dim()` shows the column length and row length of matrices/data frames.
 - `rep(0, 10)` gives a vector of ten zeros.
 - `seq(0,10,length=10)` gives a vector starting at 0, ending at 10, with a vector length of 10. We can also say `seq(0,10,by=1)` for the same thing.
 - reference to the various special characters (non-arithmetic) in R
 - `[` use it to **index** variables. e.g. `x[1]`
 - `[[` use it to **index** lists. e.g. `x[[1]]` gives the first element of the list. You can also combine the above two, e.g. `x[[1]][1]` gives the first element in the first item in the list.
 - `(` used as **inputs** into a function. e.g. `max(c(1,2,3))`. note.. a `for` loop (and the like) is also a function.
 - `{` use to wrap around the "meat" of the function that you create. This also applies when using `for` loops (and the like). You don't need these curly brackets when the "meat" is one line long.
 - `$` use it extract stuff from lists and also data frames (if it has a names,, check with `names()`). e.g. `Loblolly$height`.

- General syntax for `for()` loops

```
for (condition) {
  do this
}
```

- Using the `print()`, `cat()` to display output. Use `format()` for pretty printing.
- R function general syntax

```
function.name <- function(inputs=defaults) {
  do this
  return(output)
}
```

- Use of anonymous functions and function handles
- Use `plot()` to plot a new plot on a graphics device. Use `points()` to use overlay more stuff on the current plot. To create a "null" plot, use `plot(0, 0, pch = "")` that shows nothing except the frame and axes.
- Use `par(mfrow=c(i,j))` for multiple plots of `i` rows and `j` columns on one graphics device.
- Use `dev.new()` to create a new graphing device. Then `dev.set(i)` to "activate" the plot so you can draw stuff on graphing device `i`.
- Use of the percent sign, `#`, to start a comment
- Element-wise multiplication via `*` and `^`
- Use of "vectorized operations `%%`" to speed up calculations. There are vectorized operations that are important to know.. `apply(x, 1, function)` applies a general function to each row of `x`. `apply(x, 2, function)` for each column. `lapply(x, function)` uses a function for each element of a list. `replicate(n, function)` for repeated evaluation of a function `n` times. `tapply(x, index, function)` uses a function for each factor denoted in `index`.
- `options(digits=3)` to control significant digits. `options()$digits` to see what the current value is. Check `options()` for other amiable parameters.

*** And, in week 3 and later weeks (as this material is covered): ***

- Using eigenvalue / eigenvector commands via `eigen()`
- Use of `which()` command to identify, extract, or modify target entries in a matrix or vector
- Using `max()` and `abs()` – e.g. to find dominant eigenvalues / eigenvectors (sec. 8 of lab manual)
- Linear solve of matrix equations $W*c = n$, where `W` is matrix, `n` is *given* vector, and `c` unknown vector that are solving for. Syntax: `c <- solve(crossprod(W)) %% crossprod(W,n)`
- Using `runif(n)` for `n` random variables from 0 to 1.
- Using the `mean` and `var` commands. Check out `summary()` as well
- Plotting and properly scaling a histogram.. `prob=T` option in `plot()`
- Simulating a Markov chain – using logical operators and repeated calls to `runif(n)`