**http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures.html**

**Lecture 2—Monday, August 30, 2010**

**Topics**

* [Description of the data](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#description)
* [Data entry](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#readdata)
  + [Reading data from a web site](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#web)
  + [Reading data from a local file](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#local)
    - [Specifying a file's location interactively](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#locating)
    - [Some ways to determine the full path name to a file](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#specifying)
    - [Path name conventions in Windows](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#pathwin)
    - [Path name conventions in Mac OS X](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#pathmac)
    - [File name extensions](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#extensions)
* [Fixing mistakes and retrieving the command history](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#fixing)
  + [Windows OS](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#win1)
  + [Mac OS X](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#mac1)
* [Getting information about data frames](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#getting)
* [Extracting the treatment and response values from tadpole data set with box plots](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#extracting)
* [Visualizing the results of an experiment with box plots](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#visualizing)
  + [R color codes and plot symbols](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#symbols)
* [Homogeneity of variance](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#homogeneity)
* [Coding of factors](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#coding)
* [Fitting the ANOVA model](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#fitting)
* [Graphical examination of the three-factor interaction](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#graphical)
  + [Downloading a new R package from the CRAN web site](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#downloading)
    - [Windows OS](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#win2)
    - [Mac OS X](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#mac2)
  + [Creating a three-factor interaction diagram using the effects package](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#drawing)
  + [Alternative displays of the same three-factor interaction](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#alternative)
* [Examining the two-factor interactions](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#twofactor)
  + [Testing the fac1 × fac3 interaction](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#int1)
  + [Testing the fac1 × fac2 interaction](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#int2)
  + [Testing the fac2 × fac3 interaction](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#int3)
* [The final model](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#final)
* [Closing the R session](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#closing)
* [Cited references](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#cited)
* [R code](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#Rcode)

**R functions and commands demonstrated**

* [anova](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#anova) when applied to a model object created by the **lm** function, produces an analysis of variance table with the usual F-statistic for testing various effects. The F-tests it produces are variables-added-in-order tests.
* [as.character](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#ascharacter) converts its argument to character data.
* [as.numeric](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#nchar) converts its argument to numeric data. We used it to obtain the numeric levels of a factor.
* [axis](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#axis) is a low-level graphics function for customizing the features of the axes of the currently displayed graph. Its first argument is one of the numbers 1, 2, 3, or 4, which specifies the bottom, left, top, and right sides of the graph, respectively.
* [box](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#box) is a low-level graphics function that draws a box around the currently displayed graph.
* [boxplot](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#boxplot) produces a single box plot of a continuous variable or side-by-side box plots of a continuous variable stratified by a grouping variable.
* [c](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#c) is the catenation function that converts its arguments into a single vector.
* [contrasts](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#contrasts) displays the coding scheme used by R for a factor.
* [data.frame](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#dataframe) constructs a data frame from its arguments. The arguments must all be vectors of the same length, but they can be of different types.
* [dim](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#dim) returns the number of rows and columns of a data frame.
* [factor](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#factor) declares a variable to be a factor. The unique values of the variable become the levels of the factor and by default they are ordered in numerical or alphabetical order. Dummy contrasts are set up for the categories.
* [file.choose](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#filechoose) can be used in the **read.table** function to permit files to be located interactively.
* [getwd](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#getwd) is short for "get working directory" and displays the path of the current working directory to which R will reads and writes files.
* [history](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#history) is a function that works on Windows operating systems to open up a history window that lists previously issued commands. For example, history(50) would display the last 50 commands issued. There is a 512 line default limit to what is saved, although this value can be changed.
* [jitter](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#points) randomly adds a small numeric value (noise) to each element of its argument.
* [letters](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#dataframe) is a constant vector in R that contains the 26 lower case letters of the English alphabet.
* [levels](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#levelsfunc) displays the unique values of a factor variable in their internal order.
* [library](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#library) loads an R package into memory.
* [lm](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#lm) is the linear model function in R for fitting ordinary regression models and analysis of variance.
* [ls](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#ls) is short for "list" and is used to list the objects in the current R workspace.
* [mean](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#mean) calculates the mean of a variable.
* [names](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#names) displays names of the variables in a data frame.
* [nchar](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#nchar) counts the number of characters in a character string.
* [points](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#points) is a low-level graphics command that adds individual points to the currently active plot.
* [read.table](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#readtable) reads in text data from an external file.
* [source](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#source) is used to load and execute a file of R commands from an external text file.
* [substr](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#substring) is used to extract a character string from portion of a character variable.
* [summary](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#summary) displays the parameter estimates and summary statistics of a linear model.
* [tapply](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#tapply) stands for "table apply". It is used to apply a function (3rd argument) to a variable (1st argument) separately for each group specified by the second argument.
* [update](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#update) is used to add or subtract terms from a previously fit model.
* [~](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#boxplot) is the formula symbol and is used in defining expressions in R for model fitting. We used it in **boxplot** and **lm**.
* [.~.](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#update) is used with the **update** function to denote the current model.
* [\*](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#star) in linear models is used as a shortcut notation to form all main effects and interactions involving a set of variables, e.g., var1\*var2 is a shortcut for writing var1 + var2 + var1:var2
* [:](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#colon) in linear models is used to denote an interaction between two variables, e.g., var1:var2
* [#](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#comment) indicates that a given line of code is a comment and should be ignored by the R compiler.
* [<-](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#assignment) is the assignment operator in R. It is the less than symbol followed by a dash, representing a left pointing arrow. The arrow points in the direction of assignment. One may also use the equals sign, =, for a left assignment.
* [->](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#assignment) is the alternate assignment operator in R. It is a dash followed by a greater than symbol, representing a right pointing arrow. The arrow points in the direction of assignment.
* [[ ]](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#brackets), a pair of brackets, is used for specifying elements of vectors or portions of data frames and matrices.
* [$](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#dollarsign) is the list notation symbol that can be used to reference columns of a data frame by the name assigned to that column, as in temp.dat$var

**R function options**

* [at](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#at)= (argument to **axis**) is used to specify the locations of tick marks.
* [axes](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#axes)= (argument to many high-level graphics functions) when set to FALSE suppresses the display of axes and tick marks.
* [cex](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#cex)= (argument to many graphics functions) specifies the amount of character expansion to be used for plotting symbols; cex=1 is the default.
* [col](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#col)= (argument to many graphics functions) specifies the color to use in plotting points and/or line segments.
* [header](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#header)= (argument to **read.table**) takes on values TRUE or FALSE, indicates whether the first line of a text file should be treated as containing variable names (TRUE) or not (FALSE).
* [horizontal](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#horizontal)= (argument to **boxplot**) when set to TRUE causes box plots to be drawn horizontally.
* [labels](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#labels)= (argument to **axis** and **factor**) specifies a vector of values to appear at the tick mark locations specified by **at=** in **axis** OR the [labels](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#labelsf) for the levels defined by the **factor** function.
* [las](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#las)= (argument for many graphics text functions) controls the orientation of printed text. We chose las=2 to display tick mark labels perpendicular to the axis on which they occur.
* [levels](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#levelsf)= (argument to **factor**) is used to specify the levels of a factor variable in the order desired. In dummy coding the first value specified in the levels argument becomes the reference level.
* [pch](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#pch)= stands for plot character and is used to designate the plotting symbol to be used in various plotting functions, such as **plot**, **points**, etc.
* [multiline](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#multiline)= (argument to **plot.eff** of **effects** package) is set to T or F. If TRUE then different profiles corresponding to the levels of the **z.var** factor are displayed in the same panel.
* [na.rm](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#narm)= (argument to **mean** and many other statistical functions) is set to TRUE or FALSE to indicate whether missing values should be removed (TRUE) before performing calculations. If set to FALSE and there are missing value then the function returns NA as its value.
* [outline](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#outline)= (argument to **boxplot**) when set to FALSE turns off the printing of outliers in a box plot.
* [sep](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#sep)= (argument to **read.table**) specifies the character that is used to separate fields in the text file. For example, **sep=','** indicates that the entries are separated by commas while **sep='\t'** indicates that the entries are separated by tabs.
* [xlab](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#xlab)= (argument of all plotting functions) a user-specified value to be used as the label for the *x*-axis, e.g., xlab="Mitotic activity"
* [x.var](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#xvar)= (argument to **plot.eff** of **effects** package) name of factor to display on *x*-axis, e.g., x.var="fac1"
* [z.var](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#zvar)= (argument to **plot.eff** of **effects** package) name of factor that identifies the different profiles, e.g., z.var="fac2"

**Additional R packages used**

* [effects](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#effects) for the **effect** function and its **plot.eff** method to generate plots of interaction effects.

**Description of the data**

The data are drawn from an experiment in which a continuous response was measured on twelve treatment groups that were constructed by combining the levels of three categorical factors. Typically such an experiment is analyzed as a three-factor analysis of variance (ANOVA). Here's a description of the variables as supplied to me by the researcher.

**Response**: The response variable is mitotic activity in the intestine of my tadpoles after I fed them certain diets. I measured it using an antibody against phosphorylated histone (DNA histones are phosphorylated during mitosis), and basically it gives me a measure of how well the tadpoles are assimilating the diet that I feed them.

**Factor 1** (Co, No and Ru) is a hormonal manipulation. In a previous experiment I found that these tadpoles exhibited more mitotic activity when fed shrimp (their native diet) and less when fed detritus. I was trying to figure out whether the reduction in mitotic activity on detritus was related to stress hormones. Co = CORT, or corticosterone, No = None, or control and Ru = Ru486, which is a CORT antagonist. So the thought was that if the depression of mitotic activity in detritus-fed tadpoles was alleviated by Ru486, or the elevated mitotic activity in shrimp-fed tadpoles was depressed by CORT, then stress hormones may be the proximate cause of diet-induced differences in these tadpoles.

**Factor 2** is the diet. D = Detritus and S = Shrimp. Most tadpoles eat detritus, this species has evolved a novel diet of shrimp and other tadpoles.

**Factor 3** is family. I used two sibships for this experiment, and, as I unfortunately found out, one of the families performed about as well on detritus as it did on shrimp (contrary to all other experiments I've done). So, there appears to be family variation for shrimp assimilation.

**Data entry**

When R is launched the R Console window appears in which R commands can be entered. In the Windows operating system to move around within a command line to correct mistakes you need to use the left and right arrow keys on the keyboard rather than the mouse. The home and end keys can be used to move to the beginning and end of the current line. Previously issued commands can be recalled with the up arrow key. Choose Help > Console from the menu for additional details. These same keyboard manipulations work with R for Mac OS X but in addition the mouse can be used to move around within the same command line.

**Reading data from a web site**

Data files can be read directly from web sites. The data for today's class can be found at the location <http://www.unc.edu/courses/2010fall/ecol/563/001/data/lecture2/tadpoles.txt>. The basic way to read data into R from a text file is to use the **read.table** function. R uses standard mathematical notation *f*(*x*,*y*,*z*) to specify functions and their arguments, so parentheses are always required with functions but for some functions it is not always necessary to specify arguments.

The structure of the current data file is particularly simple so we only need to specify two arguments.

1. The first argument is the location of this file on the web with the complete path enclosed in quotes. You may use single or double quotes for this, but you must not mix the two types as part of the same argument.
2. The argument **header=TRUE** is needed to indicate that the names of variables appear in the first line of the file.

The **read.table** function processes the external data file and creates an R object called a data frame which by default is then printed to the screen. To be able to make use of function output we need to explicitly assign the output to a named R object using the assignment operator (also called the assignment arrow). The assignment arrow is constructed from two keyboard characters and is either a left-pointing arrow, **<-** , or a right-pointing arrow, **->**. The arrow points in the direction of the assignment. whose purpose is to assign the output of a function to a named object in R.

* If we use **<-** then the named object appears first, followed by the arrow, followed by the calculation that produces the output. The syntax is named\_object <- calculation.
* If we use **->** then the calculation should appear first, then **->**, followed by the variable that is to get assigned the result of the calculation. The syntax is calculation -> named\_object.

Note: The **<-** can be replaced with an **=** sign. The equal sign always assigns the result of the calculation on the right to the variable on the left of the equal sign. Thus it cannot be used to replace **->**.

As an illustration I read in the file tadpoles.txt from the web with **read.table** and store the result in a variable I call temp.dat.

**#read in data from Web**

temp.dat <- read.table('http://www.unc.edu/courses/2010fall/ecol/563/001/data/lecture2/tadpoles.txt', header=TRUE)

R treats any line that begins with a **#** symbol as a comment and ignores it. I will occasionally mix comments in with the code to annotate what's being done. They will be colored **green** with a shaded background.

Unlike the current data set, typically the data file you'll read into R will contain more than one variable with some kind of delimiter between the values. The use of delimiters between the fields is especially important if there are any missing data. The **sep** argument of **read.table** can be used to tell R what the delimiter is. (The default is for R to treat either spaces of tabs as field delimiters.) For instance, use **sep=','** to indicate a comma-delimited file or **sep='\t'** to explicitly indicate a tab-delimited file. With the current file we can use either one of these because there is only one column of data and so a delimiter is not present.

**#assumes entries in different fields are separated by commas**

temp.dat <- read.table('http://www.unc.edu/courses/2010fall/ecol/563/001/data/lecture2/tadpoles.txt', header=TRUE, sep=',')

**#assumes entries in different fields are separated by tabs**

temp.dat <- read.table('http://www.unc.edu/courses/2010fall/ecol/563/001/data/lecture2/tadpoles.txt', header=TRUE, sep='\t')

**Reading data from a local file into R**

To read your own data into R from Excel, say, you would first need to create an appropriate text file. While there exist R packages for reading Excel files directly (on Windows machines), for most purposes it is preferable to use Excel to create a text file of your data and read in the text file instead. To create a text file from an Excel worksheet you would first choose File > Save As... from the file menu. In the Save As window that appears change the Save as type option at the bottom of the window to Text (tab delimited) (\*txt) or CSV (Comma delimited) (\*.csv). Fig. 3 shows both of these options with the comma-delimited option highlighted.



**Fig. 1** Choosing the file format when saving files in Excel

To read this file into R would proceed the same as with web-based files except the file location would change. The web address is replaced with the full path name to the file on your computer. A problem with some operating systems is that the details of the file hierarchy are concealed from the user so that a file's exact address may be unclear. Here are a couple of work-arounds if that is the case.

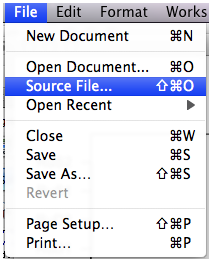
**Specifying a file's location interactively**

You don't have to know the location of the file to read it into R. You can use the **file.choose** function for the file location in **read.table**. This will bring up a "File Choose" window within which you can then navigate to the file.

**#locate file interactively**

temp.dat<-read.table(file.choose(), header=T)

I don't recommend this as a regular option. Ideally you want to save all of the R commands used for an analysis in a text file so that you can easily repeat the analysis when needed. Having to negotiate through a file selection window each time is silly, particularly when at a future date you may not even remember the name of the data file you used in the analysis. I recommend hard-wiring the file location in.



**Fig. 2** Source file… in File menu

**Some ways to determine the full path name to a file**

Depending on the operating system you're using the full path to files in special folders may not be explicitly shown (e.g., in older versions of Windows). In this case you can use R to show you the path name of your file. Use Notepad to create a text file with a single calculation line in it, say the mathematical expression 2+2. Save this file in the same folder containing your data. For instance, I save it as the file arithmetic.txt.

Next within R choose File > Source File … (Fig. 4) from the menu, navigate to the folder that contains the file you just created, and select it. R will display the entire path name of the file it is sourcing in the console window.

source("C:\\Users\\jmweiss\\Documents\\ecol563\\arithmetic.txt")

Now all that's left to do is to copy this line (or use the up arrow on the keyboard to navigate to it), change the function name from **source** to **read.table**, change arithmetic.txt to the file name of the data set, and add the additional argument **header = T**.

Another thing that may help is to determine the folder that R is using as the current working directory. The working directory is the folder that R uses by default for reading and writing files. The **getwd** function of R returns the location of the working directory.

getwd()  
"C:\\Users\\jmweiss\\Documents"

If the data file of interest happens to lie inside the Documents folder then I can easily complete the rest of the path name. In fact because the Documents folder is the default working directory, I don't really need to give the full path to the file. I can start instead with the first folder inside the Documents folder that contains my file.

temp.dat<-read.table("ecol563\\tadpoles.txt", header=T)

**Path names for files in Windows**

Windows uses a single back slash, \, to separate the folders in the path name of a file. This representation is not allowed in R. You should replace each back slash with a forward slash, as in the URLs of web sites, or with double back slashes. For example, if the file tadpoles.txt is located on the C: drive in the folder ecol563 of my hard drive, I should specify the path name to this file using forward slashes as in

'C:/Users/jmweiss/Documents/ecol563/tadpoles.txt'

or using double back slashes as in

'C:\\Users/jmweiss\\Documents\\ecol563\\tadpoles.txt',

but not the single backslashes that Windows conventionally uses 'C:\Users\jmweiss\Documents\ecol563\tadpoles.txt'. The single backslash within character strings to denote special characters in R that allows you to modify how text enclosed in quotes will print to the screen. We have already seen an example of this, the use of the single backslash followed by the letter t to indicate a tab.

**Path names for files in Mac OS X**

In Mac OS X the file name syntax is only a little different. Suppose I have the file in a folder I call ecol563 in the Documents folder on my hard disk. The correct file designation for me would be:

'/Users/jackweiss/Documents/ecol563/tadpoles.txt'.

You can see the path name to a file by clicking on the folder name at the top of its window while holding down the command (apple) key. In specifying the path in R it is not necessary to include any details about the path for any folder in the hierarchy that is above the Users folder.

**File name extensions**

Remember that the files created by programs such as Excel, Notepad, Textedit, etc. have extensions associated with them by default. You will not see those extensions if your operating system is configured so that file extensions are made invisible, but they are there. Typically they will be .txt (for space-delimited or tab-delimited files), .dat (sometimes for tab-delimited files), and .csv (for comma-delimited files). If you get an error message such as 'No such file or directory' with the **read.table** function it may because you did not specify the extension in the file name.

**# I forget to include the file extension .txt**

temp.dat<-read.table('/Users/jackweiss/Documents/ecol563/tadpoles', header=T)

Error in file(file, "rt") : cannot open the connection  
In addition: Warning message:  
In file(file, "rt") :  
cannot open file '/Users/jackweiss/Documents/ecol563/tadpoles': No such file or directory

Anytime you get an error message resembling the one shown above, it means you've made an error in specifying the name or location of the file.

**Fixing mistakes and retrieving the command history**

If you accidentally hit the return key before you finished typing a line, R will present you with a **+** prompt indicating that it is waiting for more information. At this point you can just finish the line and then hit return again. In line below I pressed the return key before closing the parenthesis on the **read.table** function.

**>** temp.dat<-read.table('/Users/jackweiss/Documents/ecol563/tadpoles.txt', header=T  
**+** )

Alternatively you can press the esc key to cancel the command altogether. Then you can then use the up arrow key, ↑, to move back to that line and finish it or fix any errors that you made.

Previously issued commands are stored in a history buffer. Accessing it is done differently in Windows and Mac OS X.

**Windows**

Old commands can be viewed by opening the history window. The command to do this is **history(*n*)** where *n* is the number of old command lines you wish to see. The following will retrieve the last 50 lines and display them in a separate window.

history(50)

Eventually the history buffer gets filled and the old commands are lost, so you should issue this command frequently if you want to record your work. I usually paste the commands from the history window into a Notepad or Word document where I remove the mistakes and add comments. The contents of the history window can also be saved to a file by choosing File > Save to File ... from the menu when the R History window is the active window.

**Mac OS X**

In Mac OS X the command history is displayed by clicking the history icon historythat appears in the tool bar at the top of the console window. This causes the list of old commands to appear on the right side of the console. A button at the bottom of this window can be used to save the entire history to a file.

**Getting information about data frames**

The object temp.dat created above with the **read.table** function is called a data frame. Data frames look like matrices but the elements of data frames can be mixtures of character and numeric data. More formally data frames are tightly coupled collections of variables that share many of the properties of matrices and of lists (another R data structure). Data frames are the fundamental data structures for most of **R**'s modeling functions.

The **dim** function is used to determine the dimensions of a data frame.

dim(temp.dat)

[1] 270 1

The displayed output from **dim** is a vector with two components (the number of rows followed by the number of columns). From the output we see that there are 270 rows and 1 column. We can access the individual entries of the output using R's vector notation, a pair of brackets containing the numbers of the entries we wish to access.

dim(temp.dat)[1]

[1] 270

Because a data frame is both a matrix and a list, we can use either notation to access its elements. The data frame we're working with is too simple to illustrate these ideas well, so I create an artificial data frame for this purpose. The **data.frame** function allows one to collect separate vectors of the same length into a data frame.

my.example <- data.frame(var1=1:8, var2=c(0,1,1,1,0,1,1,0), var3=letters[1:8])

* var1, var2, and var3 are the names I've chosen for the variables in this data frame.
* 1:8 is shortcut notation for generating the integers 1 through 8.
* The **c** function in R is used to create vectors. It concatenates individual values into a vector.
* **letters** is an example of a pre-defined constant in R. It's a vector consisting of the 26 lower-case letters of the Roman alphabet. The notation [1:8] accesses the first 8 elements of this vector.

Typing the name of a data frame causes its contents to be printed to the screen.

my.example

var1 var2 var3  
1 1 0 a  
2 2 1 b  
3 3 1 c  
4 4 1 d  
5 5 0 e  
6 6 1 f  
7 7 1 g  
8 8 0 h

Note: The numbers appearing on the left side of the displayed output are just the labels for the rows. In this case the labels are the numbers 1 through 8. It is possible to create your own names for the rows but this is rarely useful.

Elements of a data frame can be accessed with matrix notation by specifying the desired row and column numbers separated by a comma and enclosed in brackets. For instance my.example [2,1] returns the element in row 2 and column 1 of the data frame my.example.

my.example [2,1]

[1] 2

Specifying a row number followed by a comma and then no column number inside the brackets returns the entire row. Below I get the third row.

my.example[3,]

var1 var2 var3  
3 3 1 c

Specifying a comma followed by a number inside the brackets returns an entire column. Below I get the second column.

my.example[,2]

[1] 0 1 1 1 0 1 1 0

The colon notation is used to generate a sequence of numbers. Thus 2:3 yields the vector c(2,3). We can use this to extract two adjacent rows from a data frame.

my.example[2:3,]

var1 var2 var3  
2 2 1 b  
3 3 1 c

To get 3 rows that are not all adjacent we need to use the **c** function of R to concatenate the desired row numbers into a vector. Below I extract rows 2, 3, and 5 of my.example.

my.example[c(2:3,5),]

var1 var2 var3  
2 2 1 b  
3 3 1 c  
5 5 0 e

We can also specify column elements by name (row elements too if they have names). Here I select the values of the variable called var1. Notice that the variable name appears in quotes and is used instead of the column number.

my.example[,"var1"]

[1] 1 2 3 4 5 6 7 8

An alternative way of obtaining the same column is with list notation. In list notation we specify the data frame name followed by a **$** sign followed by the variable name (unquoted this time).

my.example$var1

[1] 1 2 3 4 5 6 7 8

**Extracting the treatment and response values from the tadpole data**

The **names** function displays the names of the variables contained in a data frame. The tadpoles data set consists of a single variable of character data called var.

names(temp.dat)

[1] "var"

When we extract a single variable from a data frame it gets treated as a vector and we must access its elements using vector notation. Vectors in R have a length, but not a dimension.

length(temp.dat$var)

[1] 270

dim(temp.dat$var)

NULL

Although mathematically a vector can be thought of as a column matrix (a matrix with a single column) or a row matrix (a matrix with a single row), that's not the case in R. Vectors in R are not considered to have either rows or columns and hence are not matrices. So although we use bracket notation with a list of the element numbers that we want indicated between the brackets, we don't put a comma before the list of numbers (to get columns) or after (to get rows). We just place the list of numbers between the brackets as is. Below I look at just ten elements, element numbers 31 through 40.

temp.dat$var[31:40]

[1] CouchCoD14.165999889 CouchRuS14.428999901 CouchRuS13.654999971 CouchRuS14.137000084  
[5] CouchRuS14.079999924 CouchRuS14.528999805 CouchRuS14.395999908 CouchRuS14.09100008   
[9] CouchRuS14.282000065 CouchRuS14.028999805  
240 Levels: CouchCoD13.119 CouchCoD13.14 CouchCoD13.155 CouchCoD13.18 ... CouchRuS24.567

The notation at the bottom, "240 Levels: ..." indicates that R has converted test.dat$var to a factor, although the individual entries still appear on the screen as character data. The data are in packed format. From the output we see that every value begins with "Couch". The next four characters after "Couch" record the treatment levels of the experiment. The values of the response variable begin at position 10 and continue to the end.

There are a number of ways to obtain the output shown above. All of the following are equivalent here.

* We can pull out a variable and work with the result as a vector, temp.dat$var[31:40].
* We can treat the data frame as a matrix and extract a single column using temp.dat[31:40,1] or temp.dat[31:40,"var"]. Because temp.dat has only one column we could also write this as temp.dat[31:40,] and essentially select all of the columns.

As was noted the variable temp.dat$var is a factor. For what I wish to do next it's more convenient to treat it as a pure character variable. I carry out this conversion with the **as.character** function. Notice how the printed output for a character variable differs from that of a factor.

prelim <- as.character(temp.dat$var)

prelim[31:40]

[1] "CouchCoD14.165999889" "CouchRuS14.428999901" "CouchRuS13.654999971"  
[4] "CouchRuS14.137000084" "CouchRuS14.079999924" "CouchRuS14.528999805"  
[7] "CouchRuS14.395999908" "CouchRuS14.09100008" "CouchRuS14.282000065"  
[10] "CouchRuS14.028999805"

To extract the values of the treatment and response from each of the packed values, I use the **substr** function, which is short for substring. Like many R functions **substr** is vectorized which means that if you give it a vector it works on each individual entry of the vector separately. The **substr** function needs three arguments: a character value to substring, a starting position, and a stopping position. To pull out the values corresponding to the treatment, I start at position 6 and stop at position 9.

treatment<-substr(prelim,6,9)

treatment[15:24]

[1] "RuD1" "RuD1" "RuD1" "RuD1" "RuD1" "RuD1" "CoD1" "CoD1" "CoD1" "CoD1"

Extracting the response variable is a bit more complicated because different numbers of digits were stored for some of the cases for others so we can't use a single number as the stopping value of **substr**. We can get around this with the **nchar** function which counts the number of elements in a character string and thus allows each observation to have a different length. Finally I apply the **as.numeric** function to the result to force the character string of numbers we obtain to be treated as actual numbers.

response<-as.numeric(substr(prelim, 10, nchar(prelim)))

I print out a few of the entries at different locations in the resulting vector.

response[37:40]

[1] 4.396 4.091 4.282 4.029

response[1:8]

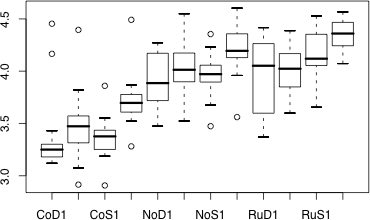
[1] NA NA NA NA NA NA NA NA

The NA value is the missing code in R. The acronym means "not available." So, we see that there are some missing values for the response.

**Visualizing the results of an experiment with box plots**

A convenient graphical way to compare the distributions of groups side by side is the box plot. We look first at the default display and then modify it incrementally to increase its information content. Here's the default box plot display we get. The basic syntax for the **boxplot** function in R is to specify the response variable followed by a tilde, **~**, which is the formula symbol in R, followed by the grouping variable.

boxplot(response~treatment)



**Fig. 3** Box plot of experiment results

The boxes themselves display the locations of the middle 50% of the data. The top and bottom edges are the first and third quartiles, Q1 and Q3, respectively. The horizontal line in each box is the location of the median. The whiskers on the boxes extend out to the smallest and largest values within the inner fences. The inner fences are given by Q1 – 1.5 × IQ, Q3 + 1.5 × IQ, where IQ is the interquartile width, Q3 – Q1. The small circles denote individual observations that lie beyond the inner fences.

One problem with this display is that there is not enough room to show all of the treatment labels on the *x*-axis. By default R suppresses the printing of some of the labels if they interfere with each other. One solution is to switch the *x*- and *y*-axes here so that the treatments appear on the *y*-axis, and then rotate the labels so that they are horizontal.

In addition to fixing the label problem I want to make a few additions to the plot.

1. Given that our final analysis will be a comparison of treatment means it would useful to add the mean of each group to the boxplot. With both the mean and median of each group visible we can also examine each distribution for skewness by seeing whether or not the median and mean occur at roughly the same place.
2. It's been argued in ecological journals that it is always preferable when possible to display the raw data data rather than just summaries of the data (Magnusson 2000). With 24 or fewer observations in each group we are in a situation here where we might try to display the individual observations. Still there is enough data that including the boxes too might be useful for assessing the characteristics of the underlying distributions. So, for the current problem I would recommend generating the box plots and then superimposing the raw data on top of the boxes, with the data jittered if necessary to prevent overlap.

The **boxplot** function is an example of a higher level graphics function in R. By default, each call to **boxplot** opens up a new graphics window and erases the current display. R also provides lower level graphics functions that can be used to add information to plots that are currently being displayed. Lower level functions include **axis** (to define characteristics of the axes), **box**, and **points** (to add additional plotted points to a display). I illustrate the use of these functions in what follows.

First we need to obtain the mean response of each treatment group. This can be obtained with the **tapply** function.

treat.mean <- tapply(response, treatment, mean, na.rm=T)

treat.mean

CoD1 CoD2 CoS1 CoS2 NoD1 NoD2 NoS1 NoS2   
3.398462 3.479176 3.366042 3.705542 3.910263 4.054167 3.973682 4.220375   
RuD1 RuD2 RuS1 RuS2   
3.935833 4.020000 4.146500 4.348875

* The first argument to **tapply** is the variable whose mean we want, in this case response.
* The second argument is the stratification variable, in this case treatment. It identifies the groups whose different response means we want.
* The third argument is the function to be applied to the first argument, **mean** in this case.
* Following the function we can include any additional arguments that need to be passed to the function. If there are any missing values in a group, **mean** by default returns a value of missing, NA, for that group. To avoid this I add the argument **na.rm=T** which causes R to remove the missing values before calculating the mean.

If we leave off the **na.rm=T** argument we get the missing code NA for some of the treatment groups.

tapply(response, treatment, mean)

CoD1 CoD2 CoS1 CoS2 NoD1 NoD2 NoS1 NoS2 RuD1   
NA NA 3.366042 3.705542 3.910263 4.054167 3.973682 4.220375 NA   
RuD2 RuS1 RuS2   
4.020000 NA 4.348875

Next I produce the boxplot, but I suppress the display of the axes and transpose the boxes.

boxplot(response~treatment, horizontal=T, axes=F, outline=F, xlab='Mitotic activity')

* The option **horizontal=T** rotates the boxes
* The option **axes=F** suppresses display of the axes
* The option **outline=F** suppresses display of the outlier points. We'll add the points back again later.
* The **xlab=** argument adds a label for the *x*-axis.

Next I use the **axis** function to add an *x*-axis, axis(1), and a customized *y*-axis, axis(2).

axis(1)

axis(2, las=2, at=1:12, labels=names(treat.mean))

* The number 1 denotes the *x*-axis while 2 denotes the *y*-axis. By entering just axis(1) I accept all the default settings for the *x*-axis.
* For the *y*-axis, **las=2** causes the labels to appear perpendicular to the axis, in this case horizontal. The acronym **las** is an abbreviation for "label style".
* The **at=** argument specifies the location of the tick marks. The treatment variable has 12 unique values and when viewed as a numerical factor these values are numbered 1 through 12. These are also the *y*-locations at which R has plotted the boxes.
* The **labels=** argument gives the text to place at each of the identified tick marks. I use the **names** function here to pull off the labels from the treat.mean vector we created above. Notice that the order of these labels is the same order that the **boxplot** function used in plotting the boxes.

Next I draw a box around the graph with the **box** function.

box()

I use the **points** function to overlay the raw data on top of the boxes. Unlike **boxplot**, **points** does not support the tilde notation. Instead we have to enter the *x*-coordinates followed by the *y*-coordinates of the points separated by commas. The **jitter** function adds noise to its argument. In this instance I use it to cause the plotted points to be randomly displaced up or down.

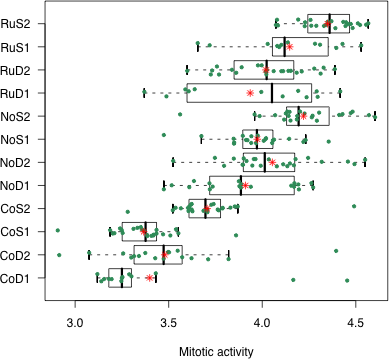
points(response, jitter(as.numeric(factor(treatment))), pch=16, col='seagreen', cex=.6)

* The first two arguments of **points** are the *x*-coordinates and the *y*-coordinates of the points.
* The *y*-coordinates are the different treatments which technically are just character values. The **jitter** function requires a numeric argument. So I use the **factor** function to convert the treatment variable to a factor that I then convert to numbers with the **as.numeric** function. A factor is a hybrid of numeric and character data. It possesses character attributes, the labels, but also numeric attributes in that the categories are numbered from 1 to however many there are. So as.numeric(factor(treatment)) converts the treatment labels to the numbers 1 through 12 corresponding to the character values in alphabetical order.
* The **pch** argument stands for "plot character" and defines the plotting symbol. Symbol type 16 is a filled circle. Fig 4a shows the print characters that are available in R.
* The **col** argument defines the color of the plotting symbols. R has over 600 named colors as well as 8 colors that can be specified with the numbers 1 through 8, Fig 4b. To see a full list of colors go to <http://research.stowers-institute.org/efg/R/Color/Chart/index.htm>.
* The **cex** argument, for "character expansion", defines the plot symbol size. Here **cex=.6** means 60% of normal size.

|  |  |
| --- | --- |
| (a) http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture2/symbols.png | (b) http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture2/colors.png |
| **Fig. 4** Plotting characters and numeric color codes in R | |

Next I use the **points** function again to add the treatment means. This time I use an asterisk for the plot symbol, **pch=8**, and red as the color, **col=2**. Fig. 5 shows the finished graph.

points(treat.mean, 1:12, pch=8, col=2)



**Fig. 5** Graphical summary of the results of the tadpole experiment

For convenience all of the R code used to create Fig. 5 is repeated below.

**#obtain treatment means**

treat.mean <- tapply(response, treatment, mean, na.rm=T)

**#produce plot with customized y-axis labels**

boxplot(response~treatment, horizontal=T, axes=F, outline=F, xlab='Mitotic activity')

axis(1)

axis(2, las=2, at=1:12, labels=names(treat.mean))

box()

**#overlay raw data**

points(response, jitter(as.numeric(factor(treatment))), pch=16, col='seagreen', cex=.6)

**#add group means**

points(treat.mean,1:12,pch=8,col=2)

**Homogeneity of variance**

In the basic analysis of variance we assume that

anova distribution

where *i* denotes the treatment, *j* an experimental unit assigned to that treatment, and *y* is the response. So each observation has a normal distribution in which the mean, μi, varies by treatment *i*, but the variance, σ2, is the same for each observation. This is called the constant variance or homogeneity of variance assumption.

Fig. 6 highlights a potential problem with these data. Notice how the variability across the groups varies. In particular look at the RuD1 group. Its interquartile width is two to five times greater than that of any other group. This could mean that the treatment is affecting the variance in addition to the mean. It could also mean that a number of confounding factors have not been properly controlled for in this experiment thus introducing heterogeneity in the groups. In either case the observed heterogeneity of group variances calls into question the constant variance assumption.

Solutions to this problem are to use a distribution (other than the normal) in which the variance is allowed to vary. Another solution is keep the normal distribution but to use generalized least squares (GLS) rather than ordinary least squares (OLS) to estimate the model. In GLS we can include an additional model for the variance. If we have time we'll revisit this issue later in the course.

**Coding of factors**

To carry out analysis of variance we need to split the treatment labels back into the original component factors. Once again we can use the **substr** function for that.

fac1<-factor(substr(treatment,1,2))

fac2<-factor(substr(treatment,3,3))

fac3<-factor(substr(treatment,4,4))

The **factor** function is used to create a factor variable and if desired can be used to set the order for the levels of the treatment in the dummy coding scheme. By default, the levels are ordered alphabetically making the first level alphabetically the reference level. We can see that this has happened with the **levels** function.

levels(fac1)

[1] "Co" "No" "Ru"

To see the dummy coding scheme that R has generated use the **contrasts** function. Here we see that the columns labeled No and Ru below are the dummy variables that indicate the No and Ru categories respectively.

contrasts(fac1)

No Ru  
Co 0 0  
No 1 0  
Ru 0 1

Recall from the researcher's description of the experiment that "No" is the control level of factor 1. Given this we might prefer to have "No" as the reference level. To accomplish this we would need to use the **factor** function with the **levels** argument in which we list the levels in the order we want specifying the reference level first.

fac1a<-factor(substring(treatment,1,2), levels=c('No','Co','Ru'))

contrasts(fac1a)

Co Ru  
No 0 0  
Co 1 0  
Ru 0 1

Now we see that "No" has been made the reference level in the dummy coding. If we want, we can make the labels more informative with the **labels=** argument of **factor**.

fac1a<-factor(substring(treatment,1,2), levels=c('No','Co','Ru'), labels=c('Control', 'Corticosterone', 'Ru486'))

contrasts(fac1a)

Corticosterone Ru486  
Control 0 0  
Corticosterone 1 0  
Ru486 0 1

**Fitting the ANOVA model**

One way we can analyze these data is as a one-way analysis variance in which we parameterize the 12-category treatment variable using 11 dummy regressors. This is called the cell means model. Because analysis of variance is simply regression with dummy variables we can fit this model with **lm**, the ordinary linear regression function of R. **lm** stands for "linear model". The model can be fit as follows.

lm(response~factor(treatment)) -> out0

Alternatively we can make use of the fact that the 12 treatments really are made up of three factors combined in all possible ways and carry out what's called a three-factor analysis of variance. With three factors the full three-factor analysis of variance includes the three main effects, the three two-factor interactions, and the single three-factor interaction.

lm(response~fac1\*fac2\*fac3) -> out1

All regression models in R are fit using formula notation. To the left of the ~ is the response variable. To the right of the ~ is the linear model containing the predictors. The notation fac1\*fac2\*fac3 is shortcut notation for a three-factor interaction plus all the terms that are marginal to it. The same model can be fit without shortcut notation as follows.

lm(response~fac1 + fac2 + fac3+ fac1:fac2 + fac1:fac3 + fac2:fac3 + fac1:fac2:fac3) -> out1

The notation in this last run should be self-explanatory. The terms separated by colons are interactions. Thus, e.g., fac1:fac2 is the two-factor interaction between fac1 and fac2, and fac1:fac2:fac3 is the three-factor interaction.

The cell means model and the full factorial model are identical except for their parameterization. This is apparent when we look at the **summary** output from these two models.

summary(out0)

Call:  
lm(formula = response ~ factor(treatment))

Residuals:  
Min 1Q Median 3Q Max   
-0.6594 -0.1382 -0.0095 0.1240 1.0555

Coefficients:  
Estimate Std. Error t value Pr(>|t|)   
(Intercept) 3.39846 0.06834 49.726 < 2e-16 \*\*\*  
factor(treatment)CoD2 0.08071 0.09079 0.889 0.374927   
factor(treatment)CoS1 -0.03242 0.08486 -0.382 0.702785   
factor(treatment)CoS2 0.30708 0.08486 3.619 0.000365 \*\*\*  
factor(treatment)NoD1 0.51180 0.08869 5.770 2.58e-08 \*\*\*  
factor(treatment)NoD2 0.65571 0.08486 7.727 3.52e-13 \*\*\*  
factor(treatment)NoS1 0.57522 0.08620 6.673 1.90e-10 \*\*\*  
factor(treatment)NoS2 0.82191 0.08486 9.686 < 2e-16 \*\*\*  
factor(treatment)RuD1 0.53737 0.09865 5.447 1.33e-07 \*\*\*  
factor(treatment)RuD2 0.62154 0.08486 7.324 4.15e-12 \*\*\*  
factor(treatment)RuS1 0.74804 0.09865 7.583 8.58e-13 \*\*\*  
factor(treatment)RuS2 0.95041 0.08486 11.200 < 2e-16 \*\*\*  
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2464 on 227 degrees of freedom  
(31 observations deleted due to missingness)  
Multiple R-squared: 0.6264, Adjusted R-squared: 0.6083   
F-statistic: 34.6 on 11 and 227 DF, p-value: < 2.2e-16

summary(out1)

Call:  
lm(formula = response ~ fac1 \* fac2 \* fac3)

Residuals:  
Min 1Q Median 3Q Max   
-0.6594 -0.1382 -0.0095 0.1240 1.0555

Coefficients:  
Estimate Std. Error t value Pr(>|t|)   
(Intercept) 3.398462 0.068344 49.726 < 2e-16 \*\*\*  
fac1No 0.511802 0.088695 5.770 2.58e-08 \*\*\*  
fac1Ru 0.537372 0.098646 5.447 1.33e-07 \*\*\*  
fac2S -0.032420 0.084859 -0.382 0.7028   
fac32 0.080715 0.090790 0.889 0.3749   
fac1No:fac2S 0.095839 0.114704 0.836 0.4043   
fac1Ru:fac2S 0.243087 0.131610 1.847 0.0660 .   
fac1No:fac32 0.063189 0.118189 0.535 0.5934   
fac1Ru:fac32 0.003452 0.125829 0.027 0.9781   
fac2S:fac32 0.258785 0.115338 2.244 0.0258 \*   
fac1No:fac2S:fac32 -0.155995 0.155945 -1.000 0.3182   
fac1Ru:fac2S:fac32 -0.140577 0.168770 -0.833 0.4057   
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2464 on 227 degrees of freedom  
(31 observations deleted due to missingness)  
Multiple R-squared: 0.6264, Adjusted R-squared: 0.6083   
F-statistic: 34.6 on 11 and 227 DF, p-value: < 2.2e-16

Observe that all of the summary statistics reported at the bottom of the output are the same for the two models. Although each uses 12 parameters to describe the outcome, the parameter estimates are different.

It's almost never a good strategy to fit the cell means model first. If the treatments possess a structure then it makes sense to take advantage of it. When treatments are constructed by combining factors in all possible ways, analyzing it as a factorial design allows us to tease apart the independent effects of the individual factors (main effects) as well as the synergistic effects (interactions) if present. The advantage of the factorial design is that it helps us understand the reasons why the means show the pattern they do. Furthermore the individual factors affect the response in a simple fashion, then we will be able to parsimoniously describe the response with fewer than twelve parameters. Finally in the end it will still be possible to obtain the individual treatment means if so desired.

When analyzing categorical regression models it is necessary to distinguish the categorical variables themselves (the predictors) from the dummy variables (regressors) that we use to represent them in the regression model. The regressors that correspond to a single categorical variable or effect are a unit that is sometimes referred to as a construct (Polissar and Diehr 1982). To simplify the regression model, we try to drop constructs, not individual dummy regressors. R's **anova** function allows us to examine the statistical significance of the individual model constructs. When dealing with an experiment in which the experimental units have been randomly assigned to treatments the full interaction model is the right place to begin the analysis. (See the discussion on this point in [lecture 1](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture1.htm#when).)

anova(out1)  
Analysis of Variance Table

Response: response  
Df Sum Sq Mean Sq F value Pr(>F)   
fac1 2 18.4339 9.2169 151.7899 < 2.2e-16 \*\*\*  
fac2 1 1.5013 1.5013 24.7238 1.304e-06 \*\*\*  
fac3 1 2.2771 2.2771 37.5007 3.984e-09 \*\*\*  
fac1:fac2 2 0.3926 0.1963 3.2328 0.04127 \*   
fac1:fac3 2 0.0838 0.0419 0.6900 0.50263   
fac2:fac3 1 0.3503 0.3503 5.7693 0.01711 \*   
fac1:fac2:fac3 2 0.0695 0.0347 0.5723 0.56505   
Residuals 227 13.7838 0.0607   
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

The last column reports *p*-values for the F-statistics (labeled F value in the output). Each test is a sequential test. It tests whether the effect in a given row is statistically significant when added to a model that already contains all the terms listed above it in the table. For the test to make sense it must adhere to the [principle of marginality](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture1.htm#interactions), something that the **anova** function does. The important conclusion that we can draw from the **anova** output above is that the three-factor interaction is not statistically significant (as seen from the p-value in the row labeled fac1:fac2:fac3).

**Graphical examination of the three-factor interaction**

The substantive meaning of a three-factor interaction can be assessed by examining plots of the two-factor interaction for two of the variables contained in the interaction for fixed values of the third variable. The **effects** package of R makes the generation of such a plot fairly easy. The **effects** package is not part of the standard R installation and it must first be downloaded from the CRAN site on the web. This process needs to be done only once.

**Downloading a new package from the CRAN site**

**Windows OS**

On Windows this is a three-step process.

1. Choose Install Packages from the Packages menu (Fig. 6a).
2. Choose a mirror site for the download (Fig. 6b). The North Carolina site is not always the best choice.
3. Select the desired package from the list of packages that appears and click the OK button (Fig. 6c).

|  |  |  |
| --- | --- | --- |
| (a) http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture2/pkg1.png | (b) http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture2/pkg2.png | (c) effects |
| **Fig. 6** Windows OS: Installing an R package from the CRAN site on the web | | |

The desired package will be downloaded along with any additional packages it requires.

**Mac OS X**

Choose Package Installer from the Packages & Data menu (Fig. 7a). In the window that appears (Fig. 7b) you can click Get List to get a list of all available packages or enter effects in the Package Search box and press the return key to locate just the **effects** package. Click the Install dependencies check box so that if the package depends on any other packages these will be downloaded also. Finally, highlight the desired package in the list and click Install Selected to download the package to the R libraries folder.

|  |  |
| --- | --- |
| (a) mac1 | (b) mac2 |
| **Fig. 7** Mac OS X: Installing an R package from the CRAN site. | |

**Creating a three-factor interaction diagram using the effects package**

Before using the **effects** package the first time in a session it needs to be loaded into memory with the **library** function.

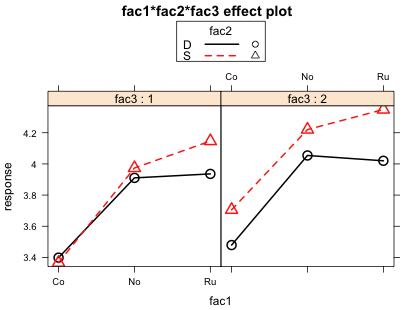
library(effects)

The function we need is called **effect**. It takes two required arguments:

* the desired effect to plot in quotes,
* followed by the name of the model in which the effect was estimated.

We then plot the output from **effect** using the R **plot** function. I add the option **multiline=T** to get separate mean profiles displayed in the same panel.

plot(effect('fac1:fac2:fac3', out1), multiline=T)



**Fig. 8** Plot of the 3-factor interaction

Each panel is a plot of the same two-factor interaction plot, in this case the interaction between fac1 and fac2. The left panel shows this two-factor interaction when fac3 = 1 while the right panel shows the same two-factor interaction when fac3 = 2. If the nature of the two-factor interaction is different in these two panels then we have graphical evidence for the presence of a three-factor interaction. For example, if we saw evidence of a two-factor interaction in one panel but not in the other, then the nature of the two-factor interaction changes with the level of fac3 indicating that a three-factor interaction is probably present. Note that it's the pattern of the two-factor interaction that we're concerned with, not whether the two graphs are identical. If there are other two-factor interactions occurring between the variables, then it is unlikely that the two graphs will look the same.

So what do we learn from Fig. 8? I would argue that we get essentially the same pattern in both panels. If fac1 consisted only of levels Co and No we would fail to find evidence of a fac1 × fac2 interaction in either of the panels. In both panels the red and black profiles are roughly parallel. But when we add level Ru to the mix the profiles are no longer parallel. In both panels we see that the mean profiles diverge as we move from level No to level Ru. Both panels show roughly the same pattern so fac3 has failed to modify the nature of the two-factor interaction we're observing. Graphically then we do not find evidence of a three-factor interaction.

**Alternative displays of the same three-factor interaction**

This is not the only three-factor interaction plot we could construct from these data. Currently fac1 is plotted on the *x*-axis, fac2 identifies the profiles, and fac3 defines the panels. These three assignments can be permuted in 2 × 2 × 2 = 6 different ways yielding six different ways of displaying the same three-factor interaction. Each version shows the same three-factor interaction but from a different perspective. All six plots should agree with each other but sometimes it is easier to draw conclusions from one plot than from another.

The **plot** method for an **effect** object allows the use of two additional keywords, **x.var=** and **z.var=**, that alter the factors displayed on the *x*-axis and the factor used to define the different profiles, respectively. Using these arguments I create two additional views of the three-factor interaction, one that uses the fac1:fac3 two-factor interaction (Fig. 9a) and the other the fac2:fac3 two-factor interaction (Fig. 9b). I start by creating the **effect** object and saving it as out2.eff, then I plot it using different settings for **x.var** and **z.var**.

out1.eff<-effect('fac1:fac2:fac3', out1)

plot(out1.eff, multiline=T, x.var="fac1", z.var="fac3")

plot(out1.eff, multiline=T, x.var="fac2", z.var="fac3")

|  |  |
| --- | --- |
| (a) three factor 2 | (b) three factor 3 |
| **Fig. 9** Two additional graphical displays of the same three-factor interaction shown in Fig. 8 | |

Fig 9a is more ambiguous than Fig. 8 was. If we look at the right panel when fac2 = 'S', we see that the two fac3 profiles are perfectly parallel. So there is clearly no evidence of a fac1 × fac3 interaction here. If we look at the left panel when fac2 = 'D', the fac3 profiles are not perfectly parallel (at least in the second half), but they're darn close. So this one is a tougher call but there is certainly no dramatic evidence of a fac1 × fac3 interaction. Because fac2 does not markedly affect the nature of the fac1 × fac3 interaction, we fail to find strong evidence of a three-factor interaction.

In Fig. 9b we see that the fac2 profiles are not perfectly parallel in any of the panels. While it is the case that the lack of parallelism is most pronounced when fac1 = Co, it never goes away. Thus there is evidence of the same fac2 × fac3 interaction in each panel. Because this interaction is not markedly affected by fac1, we fail to find evidence of a three-factor interaction using this plot either.

**Examining the two-factor interactions**

Because the three-factor interaction is correlated with any term with which it shares a factor, it's not possible to assess the statistical significance of any other term as long as the three-factor interaction is also in the model. Given that it is not statistically significant, we need to drop it and then refit the model. One way to do this would be write out the model statement again listing all of the terms but leaving out the three-factor interaction as shown below.

out2 <- lm(response~fac1 + fac2 + fac3 + fac1:fac2 + fac1:fac3 + fac2:fac3)

An easier way to do this is with the **update** function.

update(out1,.~.-fac1:fac2:fac3)->out2

The first argument is the model we wish to update. The notation, **.~.** , is short-cut notation for the terms in the current model. The periods represent "the previous values", in this case both the left and right hand sides of the model out1. From that model we then subtract off the three-factor interaction.

anova(out2)  
Analysis of Variance Table

Response: response  
Df Sum Sq Mean Sq F value Pr(>F)   
fac1 2 18.4339 9.2169 152.3590 < 2.2e-16 \*\*\*  
fac2 1 1.5013 1.5013 24.8165 1.242e-06 \*\*\*  
fac3 1 2.2771 2.2771 37.6414 3.701e-09 \*\*\*  
fac1:fac2 2 0.3926 0.1963 3.2449 0.04077 \*   
fac1:fac3 2 0.0838 0.0419 0.6926 0.50133   
fac2:fac3 1 0.3503 0.3503 5.7909 0.01690 \*   
Residuals 229 13.8533 0.0605   
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

The next step is to determine if any of the two-factor interactions are superfluous. From the ANOVA table it would appear that fac1:fac3 is a candidate for dropping. Unfortunately because these are sequential sums-of-squares tests, the current test for fac1:fac3 is for a model that does not include fac2:fac3. What we really need is a test of fac1:fac3 in the context of everything else.

**Testing the fac1 × fac3 interaction**

One way to obtain a variables-added-last test of fac1:fac3 is to refit the model with the terms placed in a different order, with the term we want to test occurring last. This has to be done explicitly by listing the terms in the model statement in the order we want.

lm(response~fac1 + fac1 + fac3 + fac1:fac2 + fac2:fac3 + fac1:fac3)->out2a

anova(out2a)

Analysis of Variance Table

Response: response  
Df Sum Sq Mean Sq F value Pr(>F)   
fac1 2 18.4339 9.2169 152.3590 < 2.2e-16 \*\*\*  
fac3 1 2.1465 2.1465 35.4816 9.621e-09 \*\*\*  
fac1:fac2 3 2.0245 0.6748 11.1553 7.340e-07 \*\*\*  
fac3:fac2 1 0.3792 0.3792 6.2676 0.01299 \*   
fac1:fac3 2 0.0550 0.0275 0.4542 0.63551   
Residuals 229 13.8533 0.0605   
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

A more elegant approach is to drop the term of interest and then use the **anova** function to compare the new model with the old model. The **anova** function compares the incremental sum-of-squares for the two nested models using a partial *F*-test.

update(out2,.~.-fac1:fac3)->out3

anova(out3, out2)

Analysis of Variance Table

Model 1: response ~ fac1 + fac2 + fac3 + fac1:fac2 + fac2:fac3  
Model 2: response ~ fac1 + fac2 + fac3 + fac1:fac2 + fac1:fac3 + fac2:fac3  
Res.Df RSS Df Sum of Sq F Pr(>F)  
1 231 13.908   
2 229 13.853 2 0.055 0.4542 0.6355

The reported partial *F*-test is a test of terms contained in the model out2 that are not also contained in out3, in this case the terms associated with the two-factor interaction fac1:fac3. For this test to make sense the two models must be nested, i.e., all the terms of the "smaller" model must be contained in the "bigger" model.

Observe that the reported *p*-value for a test of fac1:fac3, 0.6355, is the same *p*-value we obtained when we changed the order of the terms and refit the model. Because the *p*-value is large we can drop fac1:fac3 from the model. Let's confirm this non-significant result by examining the two-factor interaction graphically. Because there are two ways to display this interaction, I generate both (Fig. 10).

plot(effect('fac1:fac3',out2), multiline=T)

plot(effect('fac1:fac3',out2), multiline=T, x.var='fac3', z.var='fac1')

|  |  |
| --- | --- |
| (a) fac1 x fac2 | (b) fac1 x fac2 |
| **Fig. 10** The two ways to graphically display the fac1 × fac3 two-factor interaction | |

In both graphs the profiles are almost perfectly parallel, so graphically we have no evidence for a two-factor interaction. The numerical and graphical analyses agree. We can drop the fac1 × fac3 interaction.

**Testing the fac1 × fac2 interaction**

I drop the fac1:fac3 interaction from the last model and examine the remaining terms.

update(out2,.~.-fac1:fac3)->out3

anova(out3)

Analysis of Variance Table

Response: response  
Df Sum Sq Mean Sq F value Pr(>F)   
fac1 2 18.4339 9.2169 153.0824 < 2.2e-16 \*\*\*  
fac2 1 1.5013 1.5013 24.9343 1.169e-06 \*\*\*  
fac3 1 2.2771 2.2771 37.8201 3.382e-09 \*\*\*  
fac1:fac2 2 0.3926 0.1963 3.2603 0.04015 \*   
fac2:fac3 1 0.3792 0.3792 6.2973 0.01278 \*   
Residuals 231 13.9083 0.0602   
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

The significance testing paradigm doesn't provide any guidelines on how to proceed in this situation. We've essentially been searching for the least significant two-factor interaction (based on *p*-values) and then picking it as the next term to be dropped. Because the ANOVA table produced by R displays sequential sums-of-squares, the two tests of two-factor interactions that were conducted by R are not equivalent. For instance the test of fac1:fac2 in the above table is for a model that includes only the three main effects. The test of fac2:fac3 on the other hand is for a model that includes the three main effects as well as the fac1:fac2 interaction.

The comparable test for fac1:fac2 would be one that included the three main effects as well as the fac2:fac3 in the baseline model. As before we can obtain this test by refitting the model and specifying the terms so that fac1:fac2 is the last one. Alternatively we can fit a new model in which fac1:fac2 has been dropped from model out3 above and then compare this new model with model out3 using the partial *F*-test as carried out by the **anova** function. I illustrate both approaches below.

**#Method 1: obtain a test of fac1:fac2 from the sequential sum-of-squares table**

out3a<-lm(response~fac1 + fac2 + fac3 + fac2:fac3 + fac1:fac2)

anova(out3a)

Analysis of Variance Table

Response: response  
Df Sum Sq Mean Sq F value Pr(>F)   
fac1 2 18.4339 9.2169 153.0824 < 2.2e-16 \*\*\*  
fac2 1 1.5013 1.5013 24.9343 1.169e-06 \*\*\*  
fac3 1 2.2771 2.2771 37.8201 3.382e-09 \*\*\*  
fac2:fac3 1 0.4700 0.4700 7.8066 0.005643 \*\*   
fac1:fac2 2 0.3017 0.1509 2.5057 0.083837 .   
Residuals 231 13.9083 0.0602   
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

**#obtain a test of fac1:fac2 with a partial F-test**

update(out3, .~.-fac1:fac2) -> out4

anova(out4,out3)

Analysis of Variance Table

Model 1: response ~ fac1 + fac2 + fac3 + fac2:fac3  
Model 2: response ~ fac1 + fac2 + fac3 + fac1:fac2 + fac2:fac3  
Res.Df RSS Df Sum of Sq F Pr(>F)   
1 233 14.2100   
2 231 13.9083 2 0.3017 2.5057 0.08384 .  
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Using either approach if we add the fac1:fac2 interaction last, we would fail to reject the null hypothesis (at α =.05) and conclude that a fac1:fac2 interaction is not needed (*p* = 0.084).

This contradicts the conclusion we might have drawn from the sequential sums-of-squares tests in the ANOVA table of model out3. There we added the fac1:fac2 interaction first among the two-factor interactions and found that we could reject the null hypothesis (at α = .05) thereby concluding that the fac1:fac2 interaction should be retained (*p* = 0.04). While these two tests are not the same, this does highlight the inherent arbitrariness of the α = .05 cut-off used in significance testing. In truth there is not much difference between a *p*-value of 0.04 and a *p*-value of 0.08 except the arbitrary cut-off value of α = 0.05.

Given the ambiguity of the results from these two different significance tests it makes sense to examine the graphical evidence of a two-factor interaction between fac1 and fac2. There are two different ways that the two-factor interaction can be displayed and I present both.

plot(effect('fac1:fac2',out3), multiline=T)

plot(effect('fac1:fac2',out3), multiline=T, x.var='fac2', z.var='fac1')

|  |  |
| --- | --- |
| (a) fac1 x fac2 | (b) fac1 x fac2 |
| **Fig. 11** Two alternate graphical displays of the fac1 × fac2 two-factor interaction | |

What both graphs reveal is that if fac1 only consisted of the first two levels "Co" and "No", then there would clearly be no two-factor interaction to speak of. When restricted to just these two levels, the profiles are parallel. It's only when the third level "Ru" is include does the lack of parallelism become an issue.

The magnitudes of the effects shown in these plots are worth considering. Recall that "No" is the control group. So when the treatment "Co" is applied, the mean response decreases by 0.5 units regardless of the level of fac2 (Fig. 11b). On the other hand when "Ru" is applied , the response remains unchanged when fac2 is at level "D". But if fac2 is at level "S" the response increases by about 0.15 units. Observe that this "S" effect for "Ru" is about one third the magnitude of the "Co" effect. Given the large size difference in the effects, this raises the question of whether an effect so small is of much consequence.

We can get additional insight by looking at the **summary** table of a model containing the fac1:fac2 interaction.

summary(out3)

Call:  
lm(formula = response ~ fac1 + fac2 + fac3 + fac1:fac2 + fac2:fac3)

Residuals:  
Min 1Q Median 3Q Max   
-0.67091 -0.13746 -0.01539 0.11680 1.07076

Coefficients:  
Estimate Std. Error t value Pr(>|t|)   
(Intercept) 3.38324 0.05245 64.499 < 2e-16 \*\*\*  
fac1No 0.54730 0.05837 9.376 < 2e-16 \*\*\*  
fac1Ru 0.53699 0.06085 8.825 2.76e-16 \*\*\*  
fac2S 0.01715 0.06696 0.256 0.7981   
fac32 0.10758 0.04815 2.234 0.0264 \*   
fac1No:fac2S 0.01341 0.07728 0.174 0.8623   
fac1Ru:fac2S 0.16350 0.08175 2.000 0.0467 \*   
fac2S:fac32 0.16323 0.06505 2.509 0.0128 \*   
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2454 on 231 degrees of freedom  
Multiple R-Squared: 0.623, Adjusted R-squared: 0.6116   
F-statistic: 54.53 on 7 and 231 DF, p-value: < 2.2e-16

The two lines I've colored in red contain estimates of the two dummy variables associated with the fac1:fac2 interaction. Both of these dummy variable estimates are interpretable in terms of what is shown in Fig. 2.

* The dummy variable coefficient labeled fac1No:fac2S estimates the difference in distance between the two profile lines of Fig. 11a when fac1 is at level "Co" compared with when fac1 is at level "No". From Fig. 11a the profiles are parallel here and the two distances are about the same making their difference approximately 0. The hypothesis test in the table for fac1No:fac2S is a test of whether the difference in distances between the profiles for these two levels is equal to zero. The reported *p*-value is large so we do not reject this hypothesis, a result that is consistent with what we see in the graph.
* The dummy variable coefficient labeled fac1Ru:fac2S estimates the difference in distance between the two profile lines of Fig. 11a when fac1 is at level "Co" versus when fac1 is at level "Ru". From Fig. 11a the profiles are not parallel here and the two distances appear to be different. The hypothesis test in the table for fac1No:fac2S is a test of whether the difference in distances is equal to zero. The estimated difference in the table is 0.16 (close to our eyeball graphical estimate of 0.15) and the reported *p*-value is relatively small (*p* = 0.047). We would just be able to reject this hypothesis at α = .05.

**Testing for a fac2 × fac3 interaction**

If we choose to not drop the fac1:fac2 interaction then steamboat output above for model out3 tells us that we should not drop the fac2:fac3 interaction either. (It was the last term added and it is significant at the α = .05 level, *p* = 0.013.) On the other hand what if we did choose to drop the fac1:fac2 interaction? This is model out4. The ANOVA table below finds the fac1:fac2 interaction to be highly significant.

anova(out4)

Analysis of Variance Table

Response: response  
Df Sum Sq Mean Sq F value Pr(>F)   
fac1 2 18.4339 9.2169 151.1291 < 2.2e-16 \*\*\*  
fac2 1 1.5013 1.5013 24.6162 1.350e-06 \*\*\*  
fac3 1 2.2771 2.2771 37.3375 4.137e-09 \*\*\*  
fac2:fac3 1 0.4700 0.4700 7.7069 0.005948 \*\*   
Residuals 233 14.2100 0.0610   
---  
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

So regardless of our decision about fac1:fac2, we should definitely retain the two-factor interaction fac2:fac3. I examine this interaction graphically using both possible ways of displaying it.

plot(effect('fac2:fac3',out4), multiline=T)

plot(effect('fac2:fac3',out4), multiline=T, x.var='fac3', z.var='fac2')

|  |  |
| --- | --- |
| (a) fac1 x fac2 | (b) fac1 x fac2 |
| **Fig. 12** The two graphical displays of the fac2 × fac3 two-factor interaction | |

The lack of parallelism of the mean profiles is apparent in both plots.

**The final model**

So we have two legitimate choices for the final model.

**Choice 1** is a model that includes a two-factor interaction between factors 2 and 3 and only a main effect for factor 1.

**Rationale**: We tested for a three-factor interaction, fac1:fac2:fac3, and found it to be not statistically significant (*p* = 0.56). We then examined a model with all three two-factor interactions and sequentially dropped any two-factor interaction not significant at α = .05, starting with the least significant. Thus we dropped the fac1:fac2 interaction *p* = 0.635, followed by the fac1:fac3 interaction (*p* = 0.084). Examination of the fac1:fac3 interaction revealed that the effect was entirely due to level "Ru" of factor 1 having a larger effect than expected when factor 3 was at level "S". This effect was much smaller though than the main effect of level "No" and thus was not considered large enough to be of interest. The remaining two-factor interaction was highly significant (*p* = 0.006) as was the main effect of factor 1 (*p* < 0.001).

**Choice 2** is a model that includes a two-factor interaction between factors 2 and 3 and a two-factor interaction between factors 1 and 3.

**Rationale**: We tested for a three-factor interaction, fac1:fac2:fac3, and found it to be not statistically significant (*p* = 0.56). We then switched to a main effects model and found that when we sequentially added two-factor interactions in the order fac1:fac3, fac2:fac3, and fac1:fac2 that fac1:fac3 was statistically significant (*p* = 0.040), fac2:fac3 was statistically significant (*p* = 0.013), but that fac1:fac2 was not (*p* = 0.635). I would argue that the rationale given here for this choice is pretty weak but the choice could be supported if the fac1:fac2 interaction had some special biological interest.

**Ending the R session**

The collection of objects created during an R session can be preserved by saving the R workspace. Then at a later time this workspace can be reloaded and the objects restored for use. The workspace is defined to be the set of objects that were created in a session (including previous sessions if an old workspace was used at start-up). You can save the current objects to the default workspace, to a workspace of your own choosing, or not at all.

To see the objects currently in the workspace, use the **ls** function with no arguments.

ls()

To remove all the objects in the workspace, use the **rm** function as follows.

rm(list=ls())

When quitting R you will be prompted to save the workspace. The code used to create the objects is not part of the workspace and hence is not saved when a workspace is saved. My preference is to save the code I used to create the objects, rather than the objects themselves. As a result I try to keep the workspace empty so that I can start new projects with a clean slate. So when I close down R I generally do not save the workspace.

**Cited references**

Magnusson, W. E. 2000. Error bars: are they the king's clothes? *Bulletin of the Ecological Society of America* **81**: 147–150.  
Polissar, Lincoln and Paula Diehr. 1982. Regression analysis in health services research: the use of dummy variables. *Medical Care* **20**(9): 959–966.

**R Code**

A compact collection of all the R code displayed in this document appears [here](http://www.unc.edu/courses/2010fall/ecol/563/001/notes/lecture2%20Rcode.txt).

[Course Home Page](http://www.unc.edu/courses/2010fall/ecol/563/001/index.html)

|  |
| --- |
| Jack Weiss *Phone:* (919) 962-5930 *E-Mail:* jack\_weiss@unc.edu *Address:* Curriculum in Ecology, Box 3275, University of North Carolina, Chapel Hill, 27516 Copyright © 2010 Last Revised--September 4, 2010 URL: [http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture2.htm#lecture2) |