Introduction to the R language and environment for data analysis

Barry J. Grant
University of Michigan, Ann Arbor

R is a system for statistical computation and graphics. We use R for several reasons:

- R is open-source and freely available for Mac, PC, and Linux machines. This means that there is no restriction on having to license a particular software program, or have students work in a specific lab that has been outfitted with the technology of choice.
- R is user-extensible and user extensions can easily be made available to others.
- R is commercial quality. It is the package of choice for many statisticians and those who use statistics frequently.
- R is becoming very popular in the field of bioinformatics, especially in certain sub-disciplines, like genomics.
- R is very powerful. Furthermore, it is gaining new features every day. New statistical methods are often available first in R.

Using RStudio

RStudio is an alternative interface to R and can be installed as either a desktop/laptop application or as a server application that is accessible to others via the Internet. RStudio is available from http://www.rstudio.org

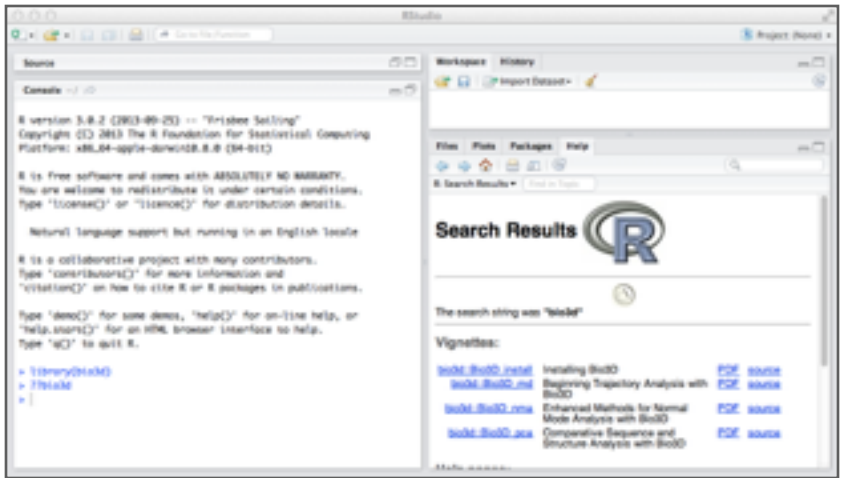

Figure 1: The RStudio interface. Notice that RStudio divides its world into a number of panels. Several of the panels are further subdivided into multiple tabs. RStudio offers the user some control over which panels are located where and which tabs are in which panels, so you initial configuration might not be exactly like the one illustrated here. The *Console* panel (shown here on the left) is where we type commands that R will execute.

1.1 Using R as a Calculator

Lets begin our introduction by using R as a simple calculator. Try typing the following commands (in blue font) in the *console panel* (see figure 1). Note that throughout this document all commands are rendered in blue font and R output in green. The hash character (#) is used for comments in R.

```
> 5 + 3                      ## Basic math with + - * / etc…
[1] 8
> 15.3 * 23.4
[1] 358
> p = 15.3 * 23.4            ## Save result
> p                          ## Show the result
[1] 358

> log( (product/2) )         ## log of half of the product
[1] 5.187442                 ## etc…
```

1.2 Five Key Things to Know About R

1. Functions in R use the following syntax:

   ```
   > functionname( argument1, argument2, ... )
   ```

   - The arguments are always surrounded by (round) parentheses and separated by commas.
   - Some functions (like `data()`) have no required arguments, but you still need the parentheses.
   - If you type a function name without the parentheses, you will see the code for that function. This probably isn't what you want at this point but will be useful later when you want to see how a function works in detail.

2. TAB key completion and arrows can improve typing speed and accuracy.
   If you begin a command and hit the TAB key, RStudio will show you a list of possible ways to complete the command. If you hit TAB after the opening parenthesis of a function, it will show you the list of arguments it expects. The up and down arrows can be used to retrieve past commands.

3. If you see a `+` prompt, it means R is waiting for more input.
   Often this means that you have forgotten a closing parenthesis or made some other syntax error. If you have messed up and just want to get back to the normal plot, hit the ESC key and start the command fresh. If you are using regular R in the terminal Control-C will do the same.

4. In addition to its core features, R provides many more features through a large number of **packages**. Bio3D is one such package for structural bioinformatics. To

use the Bio3D package, it must be installed (one time only[1]), and loaded at the start of each session with the command:

```
> library(bio3d)
```

Once loaded all the features of Bio3D will be available, for details see the output of:

```
> help(package="bio3d")
```

and visit the package homepage at http://thegrantlab.org/bio3d/

5. R has a variety of data types (known as "object types" in R parlance) for storing data. The major ones we will be using when working with Bio3D including ***vectors***, ***matrices***, ***data frames***, and ***lists***. (see **Table 1** for an overview).

Table 1. Overview of major R object types.

	vectors	matrices	data frames	lists

Description	Combines entries of the *same mode* (e.g. numeric, character, logical, etc.).
	`> c(1, 2, 3)`
	`[1] 1 2 3`
	`> c("a", "b", "c")`
	`[1] "a" "b" "c"`
	`> c(T, F, T)`
	`[1] TRUE FALSE TRUE`
		Combines vectors of the *same mode*.
		`> matrix(1:12, ncol=3, byrow=T)`
		`[,1] [,2] [,3]`
		`[1,] 1 2 3`
		`[2,] 4 5 6`
		`[3,] 7 8 9`
		`[4,] 10 11 12`
			Combines vectors of *different mode*.
			`> data.frame(c(1, 2, 3), c("a", "b", "c"), c(T, F, T))`
			`1 a TRUE`
			`2 b FALSE`
			`3 c TRUE`
				Combines vectors, matrices, data frames and lists into a single object.
				`> list(c("a","b","c"), matrix(1:4, nc=2))`
				`[[1]]`
				`[1] "a" "b" "c"`
				`[[2]]`
				`[,1] [,2]`
				`[1,] 1 3`
				`[2,] 2 4`

Usage	Data access is via *indices* (subscripts referring to the position within the vector) placed inside *square parentheses* [ ]
	`> x = c("a", "b", "c")`
	`> x[2]`
	`[1] "b"`
	`> x[c(1,3)]`
	`[1] "a" "c"`
	`> x[c(T, F, T)]`
	`[1] "a" "c"`
		Use *two coma separated indices* inside *square parentheses* to access particular row and column positions.
		`> y = matrix(1:12, ncol=3, byrow=T)`
		`> y[2,3]`
		`[1] 6`
		`> y[2, ]`
		`[1] 4 5 6`
		`> y[ ,3]`
		`[1] 3 6 9 12`
			Use either pairs of indices inside square parentheses or the *dollar name attribute* ($name) if available.
			`> w = data.frame(x=c(1:2), y=c("a","b"))`
			`> w$y`
			`[1] a b`
			`> w[2,2]`
			`[1] b`
			`> w$y[2]`
			`[1] b`
				Use either indices inside *pairs of square parentheses* or, more commonly, the *dollar name attribute* $name.
				`> z = list("v"=c(1,2,3), "m"=matrix(1:4, nc=2))`
				`> z[[1]]`
				`[1] 1 2 3`
				`> z$v`
				`[1] 1 2 3`
				`> z$m[2,2]`
				`[1] 4`

Notes	You can think of vectors as a *one row table of data entries* where the values must be all numeric, or all character, or all logical (i.e. of the *same mode*).	You can think of matrices as a *multi row and column table* where again all the values must be of the *same mode*.	These *more general spreadsheet-like table objects* are useful for storing data where the columns can be of different mode.	A very common output object from Bio3D functions that usefully *combines vectors, matrices and lists into one named object*.

1.3 Getting Help

[1] To install Bio3D from within R use the command: `install.packages("bio3d")`

If something doesn't go quite right, or if you can't remember something, it's a good to know where to turn for help. In addition to asking your friends and neighbors, you can use the R help system.

To get help on a specific function or data set, simply precede its name with a ? which is a shortcut for the help() function:

```
> ?read.pdb          ## both ? and help() can be used
> help(read.pdb)
```

If you don't know the exact name of a function, you can give part of the name and R will find all functions that match. Quotation marks are mandatory here.

```
> apropos("dcd")    ## must include quotes (single or double).
```

If that fails, you can do a broader search using ?? or help.search(), which will find matches not only in the names of functions and data sets, but also in the documentation for them.

```
> ??"mode"          ## both ?? and help.search() can be used
> help.search("mode")
```

1.4 Examples and Demos

Most Bio3D functions and data sets include example code demonstrating typical uses. This example code is included at the end of each functions help page and can be executed by using the example(functionname), command, for example:

```
> example(read.pdb)
```

This will generate all the output for the example code, including plots, and print out the commands used to create them. Examples such as this are intended to help you learn how specific R functions work.

The Bio3D package (and some other packages as well) also includes demos. Demos are bits of R code that can be executed using the demo() command with the name of the demo. You can get a list of available demos using

```
> demo()                    ## list all demos
> demo(package='bio3d')     ## just bio3d demos
```

Demos are intended to illustrate a concept, a method, or more typically a collection of related functions. We will run some demos in **section 2** of the workshop.

We also maintain full online documentation of all Bio3D functions. These have the advantage of detailing the output of all example code including various plots. These online docs are also searchable. See: http://thegrantlab.org/bio3d/html/index.html

1.5 Bio3D Tutorials (a.k.a Bio3D vignettes)

We also distribute a number of extended Bio3D vignettes that provide worked examples of using Bio3D to perform a particular type of analysis. Currently available vignettes include:

- Installing Bio3D (PDF | HTML)
- Getting started with Bio3D (PDF | HTML)
- PDB structure manipulation and analysis with Bio3D (PDF | HTML)
- Beginning trajectory analysis with Bio3D (PDF | HTML)
- Enhanced methods for Normal Mode Analysis with Bio3D (PDF | HTML)
- Comparative sequence and structure analysis with Bio3D (PDF | HTML)
- Correlation network analysis with Bio3D (PDF | HTML)
- Protein structure network analysis with Bio3D (PDF | HTML)

There is also a full package manual (in PDF format) that is a concatenation of each functions documentation.

Note that for information on Bio3D development status or to report a bug, please refer to: https://bitbucket.org/Grantlab/bio3d

1.6 NDS Bio3D Introduction (i.e. No Deep 'Stuff')

Start RStudio and load the Bio3D package by typing **library(bio3d)** at the R console prompt.

```
> library(bio3d)
```

Protein Data Bank files (or PDB files) are the most common format for the distribution and storage of high-resolution biomolecular coordinate data. To read a single PDB file with Bio3D we can use the **read.pdb()** function. The minimal input required for this function is a specification of the file to be read. This can be either:
(1) the file name of a local file on disc or
(2) the RCSB PDB identifier of a file to read directly from the on-line PDB repository.

For example to read and inspect the on-line file with PDB ID **4q21**:

```
> pdb = read.pdb("4q21")
```

To get a quick summary of the contents of the pdb object you just created you can issue the command **print(pdb)** or simply type **pdb** (which is equivalent in this case):

```
> pdb
```

Note from the printed output that the attributes (**+ attr:**) of this object are listed on the last couple of lines. To find the attributes of any such object you can use:

```
> attributes(pdb)
```

To get a simple 3D structure display of your input object try the **view()** family if functions.

```
> view(pdb)
```

To access these individual attributes we use the dollar-attribute name convention that is common with R list objects. For example, to access the atom attribute or component use **pdb$atom**, for example we can use a call to the **head()** function to view just the first few lines of our **pdb$atom**:

```
> head(pdb$atom)
```

Print a subset of $atom data for the first two atoms

```
> pdb$atom[1:2, c("eleno", "elety", "x","y","z")]
```

Side-note: **The 'pdb' class**
Objects created by the **read.pdb()** function are of class "pdb". This is recognized by other so called generic Bio3D functions (for example **atom.select()**, **nma()**, **print()**, **summary()** etc.). A generic function is a function that examines the class of its first argument, and then decides what type of operation to perform (more specifically it decides which specific method to dispatch to). So for example, the generic **atom.select()** function knows that the input is of class "pdb", rather than for example an AMBER parameter and topology file, and will act accordingly.

1.7 Atom Selection

The Bio3D **atom.select()** function is arguably one of the most challenging for newcomers to master. It is however central to PDB structure manipulation and analysis. At its most basic, this function operates on PDB structure objects (as created by **read.pdb()** for example) and returns the **numeric indices** of a selected atom subset.

These indices can then be used to access the **$atom** and **$xyz** attributes of PDB structure related objects. For example to select the indices for all C-alpha atoms we can use the following command:

```
> ca.inds <- atom.select(pdb, "calpha")
> ca.inds
```

Note that the attributes of the returned **ca.inds** from **atom.select()** include both **atom** and **xyz** components. These are numeric vectors that can be used as indices to access the corresponding **atom** and **xyz** components of the input PDB structure object. For example:

Lets print details of the first few selected atoms

```
> head( pdb$atom[ca.inds$atom, ] )
```

In addition to the common selection strings (such as 'calpha' 'cbeta' 'backbone' 'protein' 'notprotein' 'ligand' 'water' 'notwater' 'h' and 'noh') various individual atom properties can be used for selection.

```
# Select chain A
> a.inds <- atom.select(pdb, chain="A")

# Select C-alphas of chain A
> ca.inds <- atom.select(pdb, "calpha", chain="A")


# We can combine multiple selection criteria
> cab.inds <- atom.select(pdb, elety="CA", resno=10:20)
```

Question: Using atom.select how would you extract the amino acid sequence of your structure in 3-letter and 1-letter forms?

1.8 Manipulate a PDB object

Basic functions for concatenating, trimming, splitting, converting, rotating, translating and super-posing PDB files are available but often you will want to manipulate PDB objects in your own custom way.

Below we provide a basic example of such a manipulation process where we read in a multi-chained PDB structure, select a subset of chains, reassign chain identifiers, and renumber selected residues before outputting a new PDB file.

```
# Read a PDB structure from online
> pdb <- read.pdb("4lhy")

# Select chains A and E
> inds <- atom.select(pdb, chain=c("A", "E"))

# Trim to selected atoms
> pdb2 <- trim.pdb(pdb, inds)

# Change chain E values to B
> pdb2$atom$chain[ pdb2$atom$chain=="E" ] <- "B"
```

```
# Lets renumber chain B
> nums <- pdb2$atom$resno[ pdb2$atom$chain=="B" ] - 156
> pdb2$atom$resno[ pdb2$atom$chain=="B" ] <- nums


# Center to coordinate origin and orient by principal axes
> xyz <- orient.pdb(pdb2)

# Write the new pdb object to file
> write.pdb(pdb2, xyz=xyz, file="4LHY_AE-oriented.pdb")
```

1.9 Next steps - analysis

We have covered the basics of PDB structure reading and manipulation with Bio3D. Now we be will begin to do more fun things with our PDB structure objects. Lets begin with a simple plot of B-factors, similar to what you saw in the online WebApps from Day 2.

```
# Read a PDB structure from online
> pdb <- read.pdb("4q21")

# Select Calpha atoms
> inds <- atom.select(pdb, "calpha")
> plot.bio3d(pdb$atom$b[inds$atom], sse=pdb, typ="l")
```

Lets perform a normal mode analysis (NMA) on our structure and compare to the B-factors from above.

```
> modes <- nma(pdb)
> plot(modes)
```

Plot B-factors with NMA derived fluctuations

```
> plot.bio3d(pdb$atom$b[inds$atom], sse=pdb, typ="l")
> par(new=TRUE)
> plot(modes$fluctuations, axes=FALSE, typ="l", col="blue")
```