A Little Book of R For Bioinformatics

Release 0.1

Avril Coghlan

June 10, 2011

CONTENTS

By Avril Coghlan, University College Cork, Cork, Ireland. Email: a.coghlan@ucc.ie

This is a simple introduction to bioinformatics, with a focus on genome analysis, using the R statistics software.

To encourage research into neglected tropical diseases such as leprosy, Chagas disease, trachoma, schistosomiasis etc., most of the examples in this booklet are for analysis of the genomes of the organisms that cause these diseases.

There is a pdf version of this booklet available at http://media.readthedocs.org/pdf/a-little-book-of-r-for-bioinformatics/latest/a-little-book-of-r-for-bioinformatics.pdf.

Contents:

CHAPTER

ONE

HOW TO INSTALL R

1.1 Introduction to R

This little booklet has some information on how to use R for bioinformatics.

R (www.r-project.org) is a commonly used free Statistics software. R allows you to carry out statistical analyses in an interactive mode, as well as allowing simple programming.

1.2 How to check if R is installed on a Windows PC

To use R, you first need to install the R program on your computer.

These instructions are for installing R on a Windows PC.

Before you install R on your computer, the first thing to do is to check whether R is already installed on your computer (for example, by a previous user). There are two ways you can do this:

1. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 2 instead.
2. Click on the "Start" menu at the bottom left of your Windows desktop, and then move your mouse over "All Programs" in the menu that pops up. See if "R" appears in the list of programs that pops up. If it does, it means that R is already installed on your computer, and you can start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the list.

If either (1) or (2) above does succeed in starting R, it means that R is already installed on the computer that you are using. (If neither succeeds, R is not installed yet). If there is an old version of R installed on the Windows PC that you are using, it is worth installing the latest version of R, to make sure that you have all the latest R functions available to you to use.

1.3 Finding out what is the latest version of R

To find out what is the latest version of R, you can look at the CRAN (Comprehensive R Network) website, http://cran.r-project.org/.

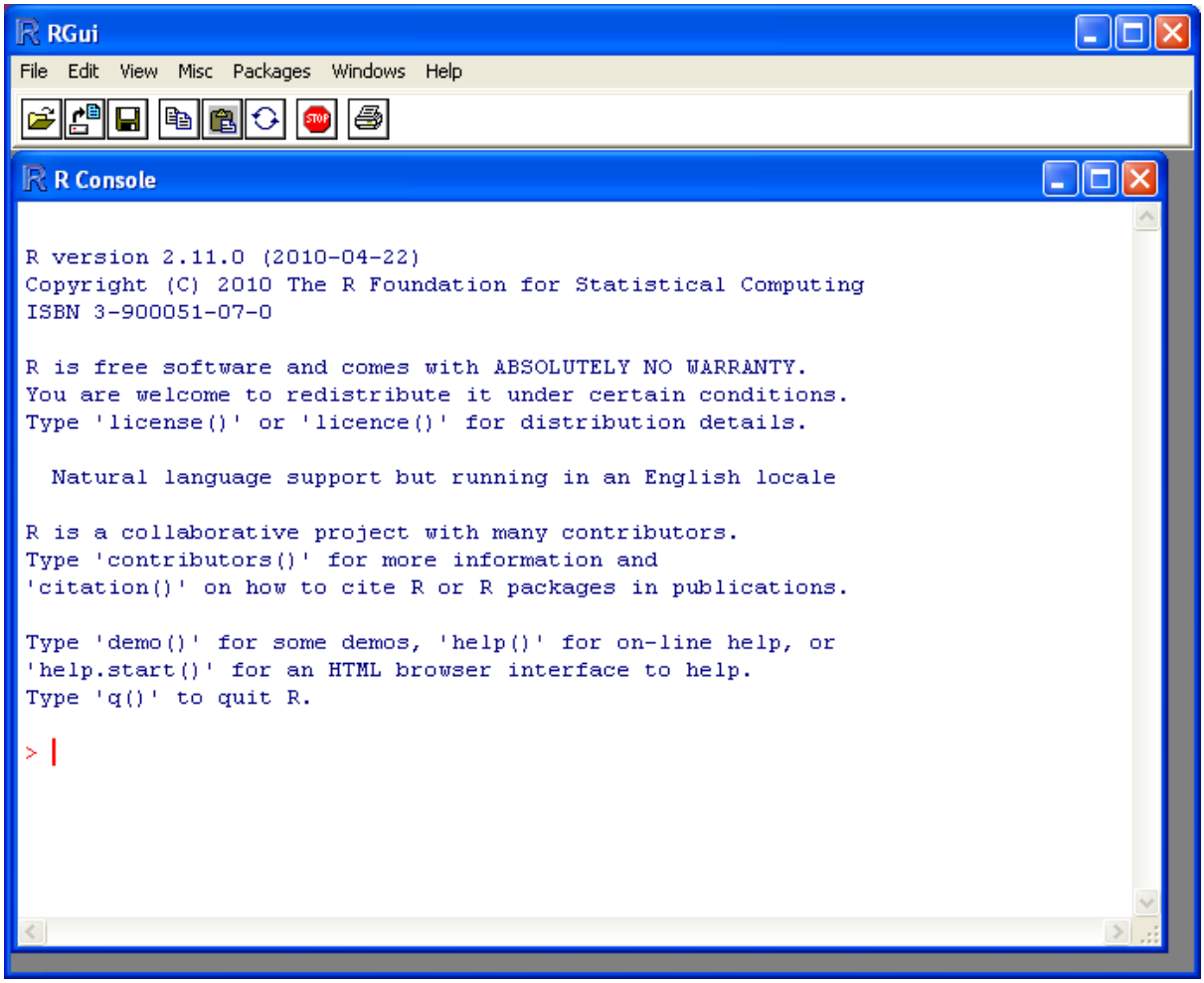
Beside "The latest release" (about half way down the page), it will say something like "R-X.X.X.tar.gz" (eg. "R-2.12.1.tar.gz"). This means that the latest release of R is X.X.X (for example, 2.12.1).

New releases of R are made very regularly (approximately once a month), as R is actively being improved all the time. It is worthwhile installing new versions of R regularly, to make sure that you have a recent version of R (to ensure compatibility with all the latest versions of the R packages that you have downloaded).

1.4 Installing R on a Windows PC

To install R on your Windows computer, follow these steps:

1. Go to http://ftp.heanet.ie/mirrors/cran.r-project.org.
2. Under "Download and Install R", click on the "Windows" link.
3. Under "Subdirectories", click on the "base" link.
4. On the next page, you should see a link saying something like "Download R 2.10.1 for Windows" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.11.1). Click on this link.
5. You may be asked if you want to save or run a file "R-2.10.1-win32.exe". Choose "Save" and save the file on the Desktop. Then double-click on the icon for the file to run it.
6. You will be asked what language to install it in - choose English.
7. The R Setup Wizard will appear in a window. Click "Next" at the bottom of the R Setup wizard window.
8. The next page says "Information" at the top. Click "Next" again.
9. The next page says "Information" at the top. Click "Next" again.
10. The next page says "Select Destination Location" at the top. By default, it will suggest to install R in "C:\Program Files" on your computer.
11. Click "Next" at the bottom of the R Setup wizard window.
12. The next page says "Select components" at the top. Click "Next" again.
13. The next page says "Startup options" at the top. Click "Next" again.
14. The next page says "Select start menu folder" at the top. Click "Next" again.
15. The next page says "Select additional tasks" at the top. Click "Next" again.
16. R should now be installed. This will take about a minute. When R has finished, you will see "Completing the R for Windows Setup Wizard" appear. Click "Finish".
17. To start R, you can either follow step 18, or 19:
18. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 19 instead.
19. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.
20. The R console (a rectangle) should pop up:



1.5 How to install R on non-Windows computers (eg. Macintosh or Linux computers)

The instructions above are for installing R on a Windows PC. If you want to install R on a computer that has a non-Windows operating system (for example, a Macintosh or computer running Linux, you should download the appropriate R installer for that operating system at http://ftp.heanet.ie/mirrors/cran.r-project.org and follow the R installation instructions for the appropriate operating system at http://ftp.heanet.ie/mirrors/cran.r-project.org/doc/FAQ/R-FAQ.html#How-can-R-be-installed_003f).

1.6 How to install an R package

R comes with some standard packages that are installed when you install R. However, in this booklet I will also tell you how to use some additional R packages that are useful, for example, the "seqinr" package. These additional packages do not come with the standard installation of R, so you need to install them yourself.

Once you have installed R on a Windows computer (following the steps above), you can install an additional package by following the steps below:

1. To start R, follow either step 2 or 3:
2. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 3 instead.

3. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.
4. The R console (a rectangle) should pop up.
5. Once you have started R, you can now install an R package (eg. the "seqinr" package) by choosing "Install package(s)" from the "Packages" menu at the top of the R console. This will ask you what website you want to download the package from, you should choose "Ireland" (or another country, if you prefer). It will also bring up a list of available packages that you can install, and you should choose the package that you want to install from that list (eg. "seqinr").
6. This will install the "seqinr" package.
7. The "seqinr" package is now installed. Whenever you want to use the "seqinr" package after this, after starting R, you first have to load the package by typing into the R console:

```
> library("seqinr")
```

Note that there are some additional R packages for bioinformatics that are part of a special set of R packages called Bioconductor (www.bioconductor.org) such as the "yeastExpData" R package, the "Biostrings" R package, etc.). These Bioconductor packages need to be installed using a different, Bioconductor-specific procedure (see How to install a Bioconductor R package below).

1.7 How to install a Bioconductor R package

The procedure above can be used to install the majority of R packages. However, the Bioconductor set of bioinformatics R packages need to be installed by a special procedure. Bioconductor (www.bioconductor.org) is a group of R packages that have been developed for bioinformatics. This includes R packages such as "yeastExpData", "Biostrings", etc.

To install the Bioconductor packages, follow these steps:

1. To start R, follow either step 2 or 3:
2. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 3 instead.
3. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.
4. The R console (a rectangle) should pop up.
5. Once you have started R, now type in the R console:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
```

6. This will install a core set of Bioconductor packages ("affy", "affydata", "affyPLM", "annaffy", "annotate", "Biobase", "Biostrings", "DynDoc", "gcrma", "genefilter", "geneplotter", "hgu95av2.db", "limma", "marray", "matchprobes", "multtest", "ROC", "vsn", "xtable", "affyQCReport"). This takes a few minutes (eg. 10 minutes).
7. At a later date, you may wish to install some extra Bioconductor packages that do not belong to the core set of Bioconductor packages. For example, to install the Bioconductor package called "yeastExpData", start R and type in the R console:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("yeastExpData")
```

8. Whenever you want to use a package after installing it, you need to load it into R by typing:

```
> library("yeastExpData")
```

1.8 Running R

To use R, you first need to start the R program on your computer. You should have already installed R on your computer (see above).

To start R, you can either follow step 1 or 2: 1. Check if there is an "R" icon on the desktop of the computer that you are using.

If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 2 instead.

2. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.

This should bring up a new window, which is the *R console*.

1.9 A brief introduction to R

You will type R commands into the R console in order to carry out analyses in R. In the R console you will see:

```
>
```

This is the R prompt. We type the commands needed for a particular task after this prompt. The command is carried out after you hit the Return key.

Once you have started R, you can start typing in commands, and the results will be calculated immediately, for example:

```
> 2*3
[1] 6
> 10-3
[1] 7
```

All variables (scalars, vectors, matrices, etc.) created by R are called *objects*. In R, we assign values to variables using an arrow. For example, we can assign the value 2*3 to the variable *x* using the command:

```
> x <- 2*3
```

To view the contents of any R object, just type its name, and the contents of that R object will be displayed:

```
> x
[1] 6
```

There are several possible different types of objects in R, including scalars, vectors, matrices, arrays, data frames, tables, and lists. The scalar variable *x* above is one example of an R object. While a scalar variable such as *x* has just one element, a vector consists of several elements. The elements in a vector are all of the same type (eg. numeric or characters), while lists may include elements such as characters as well as numeric quantities.

To create a vector, we can use the c() (combine) function. For example, to create a vector called *myvector* that has elements with values 8, 6, 9, 10, and 5, we type:

```
> myvector <- c(8, 6, 9, 10, 5)
```

To see the contents of the variable *myvector*, we can just type its name:

```
> myvector
[1]  8  6  9 10  5
```

The [1] is the index of the first element in the vector. We can extract any element of the vector by typing the vector name with the index of that element given in square brackets. For example, to get the value of the 4th element in the vector *myvector*, we type:

```
> myvector[4]
[1] 10
```

In contrast to a vector, a list can contain elements of different types, for example, both numeric and character elements. A list can also include other variables such as a vector. The list() function is used to create a list. For example, we could create a list *mylist* by typing:

```
> mylist <- list(name="Fred", wife="Mary", myvector)
```

We can then print out the contents of the list *mylist* by typing its name:

```
> mylist
$name
[1] "Fred"

$wife
[1] "Mary"

[[3]]
[1]  8  6  9 10  5
```

The elements in a list are numbered, and can be referred to using indices. We can extract an element of a list by typing the list name with the index of the element given in double square brackets (in contrast to a vector, where we only use single square brackets). Thus, we can extract the second and third elements from *mylist* by typing:

```
> mylist[[2]]
[1] "Mary"
> mylist[[3]]
[1]  8  6  9 10  5
```

Elements of lists may also be named, and in this case the elements may be referred to by giving the list name, followed by "$", followed by the element name. For example, *mylist$name* is the same as *mylist[[1]]* and *mylist$wife* is the same as *mylist[[2]]*:

```
> mylist$wife
[1] "Mary"
```

We can find out the names of the named elements in a list by using the attributes() function, for example:

```
> attributes(mylist)
$names
[1] "name" "wife" ""
```

When you use the attributes() function to find the named elements of a list variable, the named elements are always listed under a heading "$names". Therefore, we see that the named elements of the list variable *mylist* are called "name" and "wife", and we can retrieve their values by typing *mylist$name* and *mylist$wife*, respectively.

Another type of object that you will encounter in R is a *table* variable. For example, if we made a vector variable *mynames* containing the names of children in a class, we can use the table() function to produce a table variable that contains the number of children with each possible name:

```
> mynames <- c("Mary", "John", "Ann", "Sinead", "Joe", "Mary", "Jim", "John", "Simon")
> table(mynames)
mynames
   Ann    Jim    Joe   John   Mary  Simon Sinead
     1      1      1      2      2      1      1
```

We can store the table variable produced by the function table(), and call the stored table "mytable", by typing:

```
> mytable <- table(mynames)
```

To access elements in a table variable, you need to use double square brackets, just like accessing elements in a list. For example, to access the fourth element in the table *mytable* (the number of children called "John"), we type:

```
> mytable[[4]]
[1] 2
```

Alternatively, you can use the name of the fourth element in the table ("John") to find the value of that table element:

```
> mytable[["John"]]
[1] 2
```

Functions in R usually require *arguments*, which are input variables (ie. objects) that are passed to them, which they then carry out some operation on. For example, the log10() function is passed a number, and it then calculates the log to the base 10 of that number:

```
> log10(100)
2
```

In R, you can get help about a particular function by using the help() function. For example, if you want help about the log10() function, you can type:

```
> help("log10")
```

When you use the help() function, a box or webpage will pop up with information about the function that you asked for help with.

If you are not sure of the name of a function, but think you know part of its name, you can search for the function name using the help.search() and RSiteSearch() functions. The help.search() function searches to see if you already have a function installed (from one of the R packages that you have installed) that may be related to some topic you're interested in. The RSiteSearch() function searches all R functions (including those in packages that you haven't yet installed) for functions related to the topic you are interested in.

For example, if you want to know if there is a function to calculate the standard deviation of a set of numbers, you can search for the names of all installed functions containing the word "deviation" in their description by typing:

```
> help.search("deviation")
Help files with alias or concept or title matching
'deviation' using fuzzy matching:

genefilter::rowSds
                    Row variance and standard deviation of
                    a numeric array
nlme::pooledSD      Extract Pooled Standard Deviation
stats::mad          Median Absolute Deviation
stats::sd           Standard Deviation
vsn::meanSdPlot     Plot row standard deviations versus row
```

Among the functions that were found, is the function sd() in the "stats" package (an R package that comes with the standard R installation), which is used for calculating the standard deviation.

In the example above, the help.search() function found a relevant function (sd() here). However, if you did not find what you were looking for with help.search(), you could then use the RSiteSearch() function to see if a search of all functions described on the R website may find something relevant to the topic that you're interested in:

```
> RSiteSearch("deviation")
```

The results of the RSiteSearch() function will be hits to descriptions of R functions, as well as to R mailing list discussions of those functions.

We can perform computations with R using objects such as scalars and vectors. For example, to calculate the average of the values in the vector *myvector* (ie. the average of 8, 6, 9, 10 and 5), we can use the mean() function:

```
> mean(myvector)
[1] 7.6
```

We have been using built-in R functions such as mean(), length(), print(), plot(), etc. We can also create our own functions in R to do calculations that you want to carry out very often on different input data sets. For example, we can create a function to calculate the value of 20 plus square of some input number:

```
> myfunction <- function(x) { return(20 + (x*x)) }
```

This function will calculate the square of a number (x), and then add 20 to that value. The return() statement returns the calculated value. Once you have typed in this function, the function is then available for use. For example, we can use the function for different input numbers (eg. 10, 25):

```
> myfunction(10)
[1] 120
> myfunction(25)
[1] 645
```

To quit R, type:

```
> q()
```

1.10 Links and Further Reading

Some links are included here for further reading.

For a more in-depth introduction to R, a good online tutorial is available on the "Kickstarting R" website, cran.r-project.org/doc/contrib/Lemon-kickstart.

There is another nice (slightly more in-depth) tutorial to R available on the "Introduction to R" website, cran.r-project.org/doc/manuals/R-intro.html.

1.11 Acknowledgements

Thank you to Noel O'Boyle for helping in using Sphinx, http://sphinx.pocoo.org, to create this document, and github, https://github.com/, to store different versions of the document as I was writing it, and readthedocs, http://readthedocs.org/, to build and distribute this document.

1.12 Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

1.13 License

CHAPTER

TWO

NEGLECTED TROPICAL DISEASES

Neglected tropical diseases are serious diseases that affect many people in tropical countries and which have been relatively little studied. The World Health Organisation lists the following as neglected tropical diseases: trachoma, leprosy, schistosomiasis, soil transmitted helminths, lymphatic filariasis, onchocerciasis, Buruli ulcer, yaws, Chagas disease, African trypanosomiasis, leishmaniasis, Dengue fever, rabies, Dracunculiasis (guinea-worm disease), and Fascioliasis (see http://www.who.int/neglected_diseases/diseases/en/).

The genomes of many of the organisms that cause neglected tropical diseases have been fully sequenced, or are currently being sequenced, including:

- the bacterium Chlamydia trachomatis, which causes trachoma
- the bacterium Mycobacterium leprae, which causes leprosy
- the bacterium Mycobacterium ulcerans, which causes Buruli ulcer
- the bacterium Treponema pallidum subsp. pertenue, which causes yaws
- the protist Trypanosoma cruzi, which causes Chagas disease
- the protist Trypanosoma brucei, which causes African trypanosomiasis
- the protist Leishmania major, and related *Leishmania* species, which cause leishmaniasis
- the schistosome worm Schistosoma mansoni, which causes schistosomiasis
- the nematode worms Brugia malayi and Wuchereria bancrofti, which cause lymphatic filariasis
- the nematode worm Loa loa, which causes subcutaneous filariasis
- the nematode worm Onchocerca volvulus, which causes onchocerciasis
- the nematode worm Necator americanus, which causes soil-transmitted helminthiasis
- the Dengue virus, which causes Dengue fever
- the Rabies virus, which causes Rabies

To encourage research into these organisms, many of the examples in this booklet are based on analysing these genomes.

Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

2.1 License

CHAPTER

THREE

DNA SEQUENCE STATISTICS

3.1 Using R for Bioinformatics

This booklet tells you how to use the R software to carry out some simple analyses that are common in bioinformatics. In particular, the focus is on computational analysis of biological sequence data such as genome sequences and protein sequences.

This booklet assumes that the reader has some basic knowledge of biology, but not necessarily of bioinformatics. The focus of the booklet is to explain simple bioinformatics analysis, and to explain how to carry out these analyses using R.

To use R, you first need to start the R program on your computer. You should have already installed R on your computer (if not, for instructions on how to install R, see How to install R).

3.2 R packages for bioinformatics: Bioconductor and SeqinR

Many authors have written R packages for performing a wide variety of analyses. These do not come with the standard R installation, but must be installed and loaded as "add-ons".

Bioinformaticians have written several specialised *packages* for R. In this practical, you will learn to use the SeqinR package to retrieve sequences from a DNA sequence database, and to carry out simple analyses of DNA sequences.

Some well known bioinformatics packages for R are the Bioconductor set of R packages (www.bioconductor.org), which contains several packages with many R functions for analysing biological data sets such as microarray data; and the SeqinR package (pbil.univ-lyon1.fr/software/seqinr/home.php?lang=eng), which contains R functions for obtaining sequences from DNA and protein sequence databases, and for analysing DNA and protein sequences.

To use function from the SeqinR package, we first need to install the SeqinR package (for instructions on how to install an R package, see How to install an R package). Once you have installed the "SeqinR" R package, you can load the "SeqinR" R package by typing:

```
> library("seqinr")
```

Remember that you can ask for more information about a particular R command by using the help() function. For example, to ask for more information about the library() function, you can type:

```
> help("library")
```

3.3 FASTA format

The FASTA format is a simple and widely used format for storing biological (DNA or protein) sequences. It was first used by the FASTA program for sequence alignment. It begins with a single-line description starting with a

">" character, followed by lines of sequences. Here is an example of a FASTA file:

```
> A06852 183 residues
MPRLFSYLLGVWLLLSQLPREIPGQSTNDFIKACGRELVRLWVEICGSVSWGRTALSLEE
PQLETGPPAETMPSSITKDAEILKMMLEFVPNLPQELKATLSERQPSLRELQQSASKDSN
LNFEEFKKIILNRQNEAEDKSLLELKNLGLDKHSRKKRLFRMTLSEKCCQVGCIRKDIAR
LC
```

3.4 The NCBI sequence database

The National Centre for Biotechnology Information (NCBI) (www.ncbi.nlm.nih.gov) in the US maintains a huge database of all the DNA and protein sequence data that has been collected, the NCBI Sequence Database. This also a similar database in Europe, the European Molecular Biology Laboratory (EMBL) Sequence Database (www.ebi.ac.uk/embl), and also a similar database in Japan, the DNA Data Bank of Japan (DDBJ; www.ddbj.nig.ac.jp). These three databases exchange data every night, so at any one point in time, they contain almost identical data.

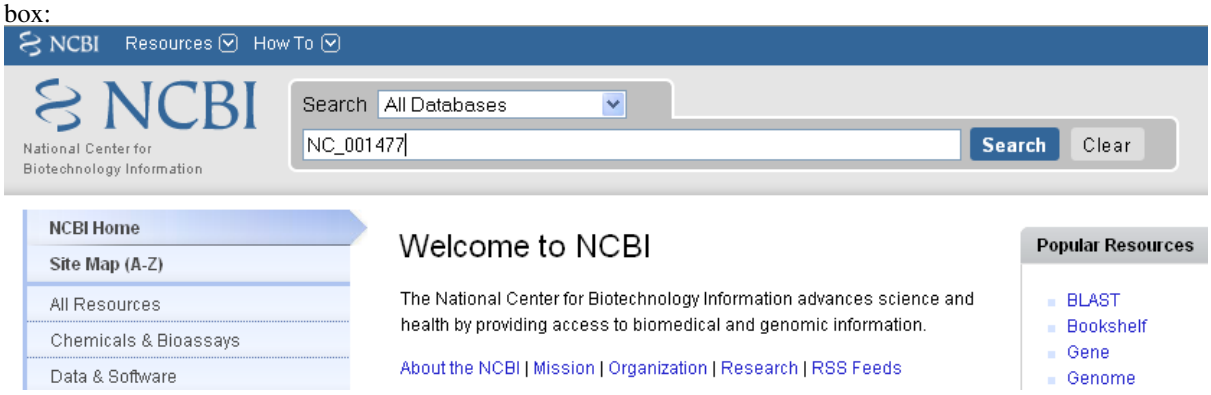
Each sequence in the NCBI Sequence Database is stored in a separate *record*, and is assigned a unique identifier that can be used to refer to that sequence record. The identifier is known as an *accession*, and consists of a mixture of numbers and letters. For example, Dengue virus causes Dengue fever, which is classified as a neglected tropical disease by the WHO. by any one of four types of Dengue virus: DEN-1, DEN-2, DEN-3, and DEN-4. The NCBI accessions for the DNA sequences of the DEN-1, DEN-2, DEN-3, and DEN-4 Dengue viruses are NC_001477, NC_001474, NC_001475 and NC_002640, respectively.

Note that because the NCBI Sequence Database, the EMBL Sequence Database, and DDBJ exchange data every night, the DEN-1 (and DEN-2, DEN-3, DEN-4) Dengue virus sequence will be present in all three databases, but it will have different accessions in each database, as they each use their own numbering systems for referring to their own sequence records.

3.5 Retrieving genome sequence data from NCBI

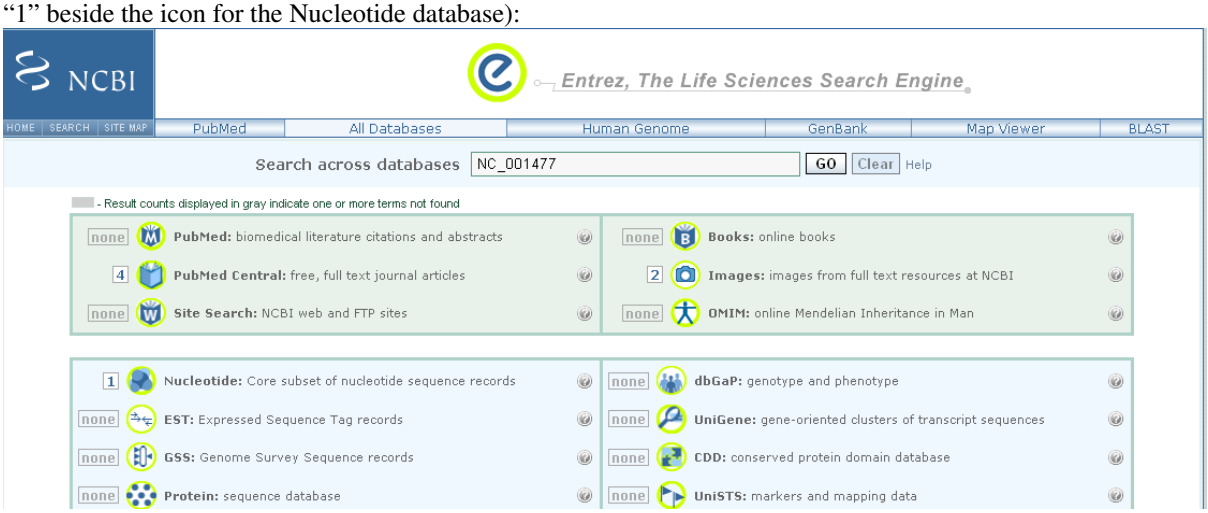
You can easily retrieve DNA or protein sequence data from the NCBI Sequence Database via its website www.ncbi.nlm.nih.gov.

The Dengue DEN-1 DNA sequence is a viral DNA sequence, and as mentioned above, its NCBI accession is NC_001477. To retrieve the DNA sequence for the Dengue DEN-1 virus from NCBI, go to the NCBI website, type "NC_001477" in the Search box at the top of the webpage, and press the "Search" button beside the Search box:

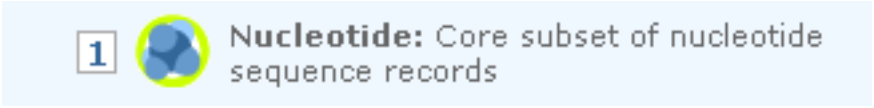


On the results page you will see the number of hits to "NC_001477" in each of the NCBI databases on the NCBI website. There are many databases on the NCBI website, for example, the "PubMed" data contains abstracts from scientific papers, the "Nucleotide" database contains DNA and RNA sequence data, the "Protein" data contains protein sequence data, and so on. The picture below shows what the results page should look like for your NC_001477 search. As you are looking for the DNA sequence of the Dengue DEN-1 virus genome, you expect

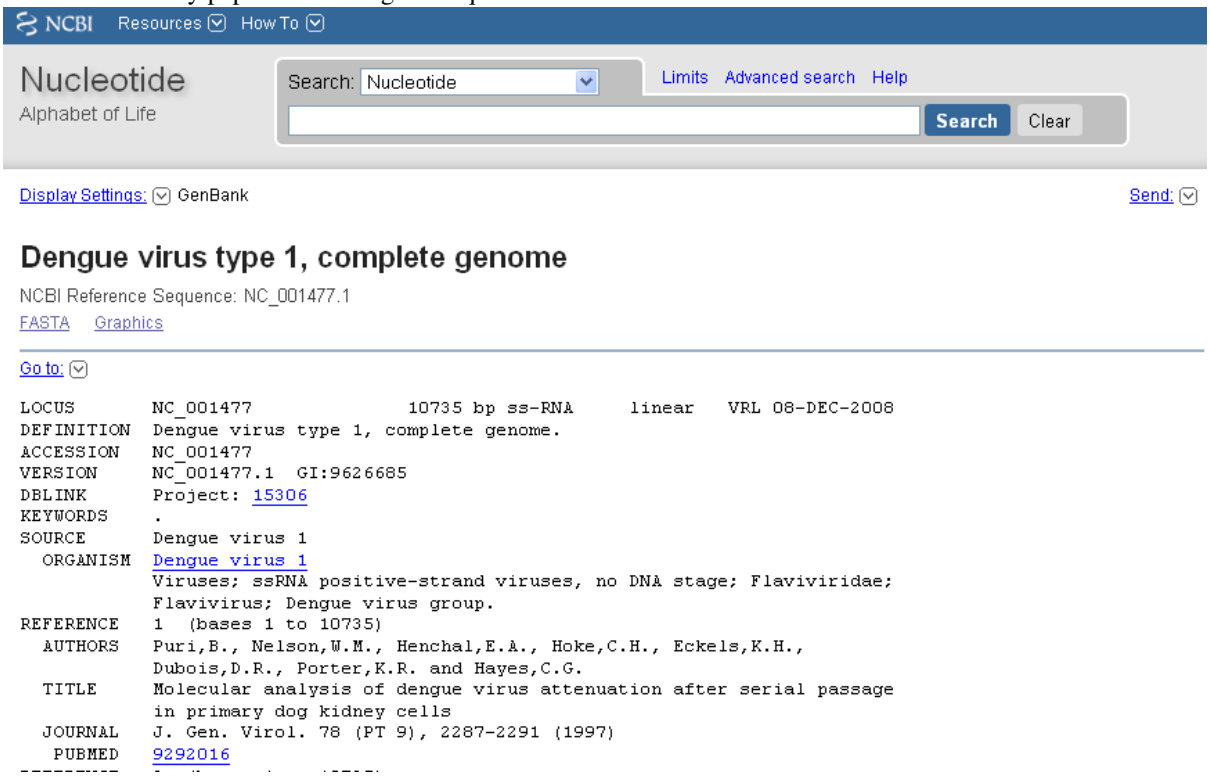
to see a hit in the NCBI Nucleotide database, and indeed there is hit in the Nucleotide database (indicated by the "1" beside the icon for the Nucleotide database):



To look at the one sequence found in the Nucleotide database, you need to click on the icon for the NCBI Nucleotide database on the results page for the search:



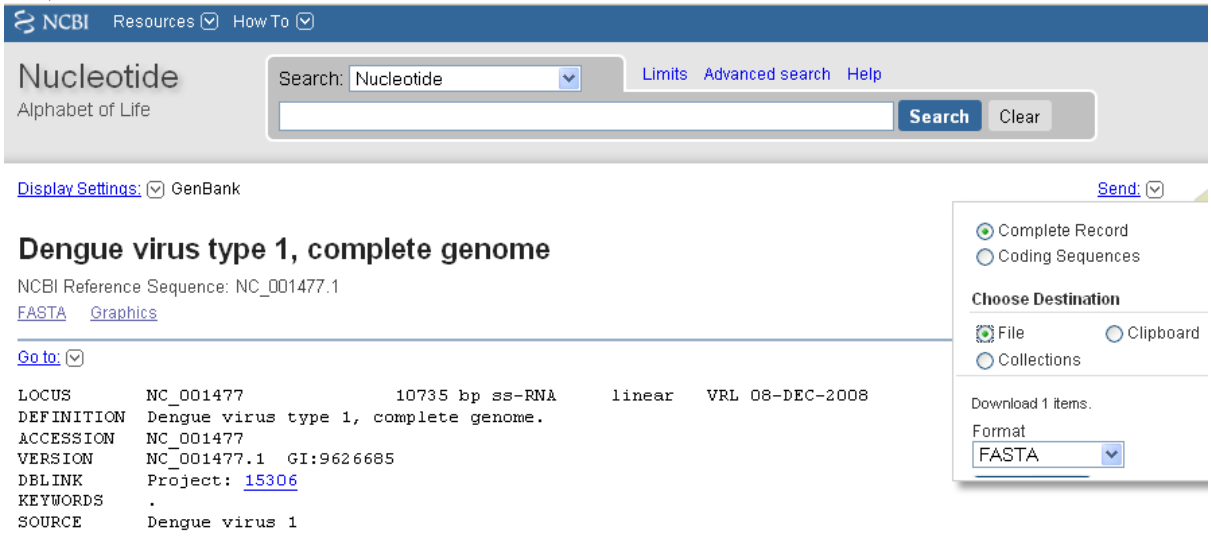
When you click on the icon for the NCBI Nucleotide database, it will bring you to the record for NC_001477 in the NCBI Nucleotide database. This will contain the name and NCBI accession of the sequence, as well as other details such as any papers describing the sequence:



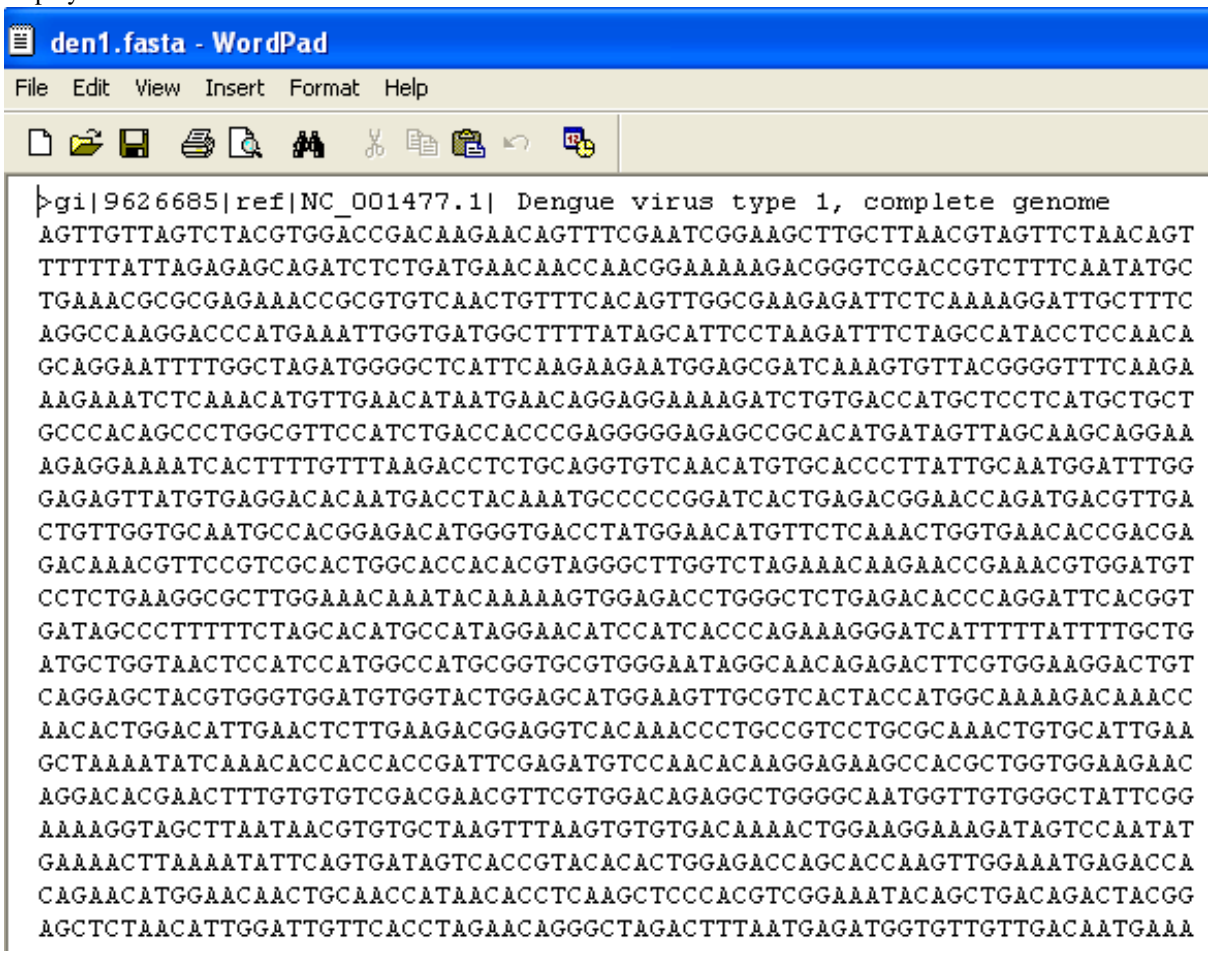
To retrieve the DNA sequence for the DEN-1 Dengue virus genome sequence as a FASTA format sequence file, click on "Send" at the top right of the NC_001477 sequence record webpage, and then choose "File" in the pop-up menu that appears, and then choose FASTA from the "Format" menu that appears, and click on "Create file".

click on the "Download" link at the top right of the NC_001477 sequence record webpage, and choose "FASTA" from the list that appears. A box will pop up asking you what to name the file. You should give it a sensible name (eg. "den1.fasta") and save it in a place where you will remember (eg. in the "My Documents" folder is a good

idea):



You can now open the FASTA file containing the DEN-1 Dengue virus genome sequence using WordPad on your computer. To open WordPad, click on "Start" on the bottom left of your screen, click on "All Programs" in the menu that appears, and then select "Accessories" from the menu that appears next, and then select "WordPad" from the menu that appears next. WordPad should start up. In Wordpad, choose "Open" from the "File" menu. The WordPad "Open" dialog will appear. Set "Files of type" to "All Documents" at the bottom of the WordPad "Open" dialog. You should see a list of files, now select the file that contains the DEN-1 Dengue virus sequence (eg. "den1.fasta"). The contents of the FASTA format file containing the Dengue DEN-1 sequence should now be displayed in WordPad:



3.6 Reading genome sequence data into SeqinR

Using the SeqinR package in R, you can easily read a DNA sequence from a FASTA file into R. For example, we described above how to retrieve the DEN-1 Dengue virus genome sequence from the NCBI database and save it in a FASTA format file (eg. "den1.fasta"). You can read this FASTA format file into R using the read.fasta() function from the SeqinR R package:

```
> library("seqinr")
> dengue <- read.fasta(file = "den1.fasta")
```

Note that R expects the files that you read in (eg. "den1.fasta") to be in the "My Documents" folder on your computer, so if you stored "den1.fasta" somewhere else, you will have to move or copy it into "My Documents".

The command above reads the contents of the fasta format file den1.fasta into an R object called *dengue*. The variable *dengue* is an R list object. As explained above, a list is an R object that is like a vector, but can contain elements that are numeric and/or contain characters. In this case, the list *dengue* contains information from the FASTA file that you have read in (ie. the NCBI accession for the dengue sequence, and the DNA sequence itself). In fact, the first element of the list object *dengue* contains the the DNA sequence. As described above, we can access the elements of an R list object using double square brackets. Thus, we can store the DNA sequence for DEN-1 Dengue virus in a variable *dengueseq* by typing:

```
> dengueseq <- dengue[[1]]
```

The variable *dengueseq* is a vector containing the nucleotide sequence. Each element of the vector contains one nucleotide of the sequence. Therefore, to print out a certain subsequence of the sequence, we just need to type the name of the vector *dengueseq* followed by the square brackets containing the indices for those nucleotides. For example, the following command prints out the first 50 nucleotides of the DEN-1 Dengue virus genome sequence:

```
> dengueseq[1:50]
[1] "a" "g" "t" "t" "g" "t" "t" "a" "g" "t" "c" "t" "a" "c" "g" "t" "g" "g" "a"
[20] "c" "c" "g" "a" "c" "a" "a" "g" "a" "a" "c" "a" "g" "t" "t" "t" "c" "g" "a"
[39] "a" "t" "c" "g" "g" "a" "a" "g" "c" "t" "t" "g"
```

Note that *dengueseq[1:50]* refers to the elements of the vector *dengueseq* with indices from 1-50. These elements contain the first 50 nucleotides of the DEN-1 Dengue virus sequence.

3.7 Length of a DNA sequence

Once you have retrieved a DNA sequence, we can obtain some simple statistics to describe that sequence, such as the sequence's total length in nucleotides. In the above example, we retrieved the DEN-1 Dengue virus genome sequence, and stored it in the vector variable *dengueseq* To subsequently obtain the length of the genome sequence, we would use the length() function, typing:

```
> length(dengueseq)
[1] 10735
```

The length() function will give you back the length of the sequence stored in variable *dengueseq*, in nucleotides. The length() function actually gives the number of elements in the input vector that you pass to it, which in this case in the number of elements in the vector *dengueseq*. Since each element of the vector *dengueseq* contains one nucleotide of the DEN-1 Dengue virus sequence, the result for the DEN-1 Dengue virus genome tells us the length of its genome sequence (ie. 10735 nucleotides long).

3.8 Base composition of a DNA sequence

An obvious first analysis of any DNA sequence is to count the number of occurrences of the four different nucleotides ("A", "C", "G", and "T") in the sequence. This can be done using the the table() function. For example, to find the number of As, Cs, Gs, and Ts in the DEN-1 Dengue virus sequence (which you have put into vector variable *dengueseq*, using the commands above), you would type:

```
> table(dengueseq)
dengueseq
   a    c    g    t
3426 2240 2770 2299
```

This means that the DEN-1 Dengue virus genome sequence has 3426 As, 2240 Cs, 2770 Gs and 2299 Ts.

3.9 GC Content of DNA

One of the most fundamental properties of a genome sequence is its GC content, the fraction of the sequence that consists of Gs and Cs, ie. the %(G+C).

The GC content can be calculated as the percentage of the bases in the genome that are Gs or Cs. That is, GC content = (number of Gs + number of Cs)*100/(genome length). For example, if the genome is 100 bp, and 20 bases are Gs and 21 bases are Cs, then the GC content is (20 + 21)*100/100 = 41%.

You can easily calculate the GC content based on the number of As, Gs, Cs, and Ts in the genome sequence. For example, for the DEN-1 Dengue virus genome sequence, we know from using the table() function above that the genome contains 3426 As, 2240 Cs, 2770 Gs and 2299 Ts. Therefore, we can calculate the GC content using the command:

```
> (2240+2770)*100/(3426+2240+2770+2299)
[1] 46.66977
```

Alternatively, if you are feeling lazy, you can use the GC() function in the SeqinR package, which gives the fraction of bases in the sequence that are Gs or Cs.

```
> GC(dengueseq)
[1] 0.4666977
```

The result above means that the fraction of bases in the DEN-1 Dengue virus genome that are Gs or Cs is 0.4666977. To convert the fraction to a percentage, we have to multiply by 100, so the GC content as a percentage is 46.66977%.

3.10 DNA words

As well as the frequency of each of the individual nucleotides ("A", "G", "T", "C") in a DNA sequence, it is also interesting to know the frequency of longer DNA "words". The individual nucleotides are DNA words that are 1 nucleotide long, but we may also want to find out the frequency of DNA words that are 2 nucleotides long (ie. "AA", "AG", "AC", "AT", "CA", "CG", "CC", "CT", "GA", "GG", "GC", "GT", "TA", "TG", "TC", and "TT"), 3 nucleotides long (eg. "AAA", "AAT", "ACG", etc.), 4 nucleotides long, etc.

To find the number of occurrences of DNA words of a particular length, we can use the count() function from the R SeqinR package. For example, to find the number of occurrences of DNA words that are 1 nucleotide long in the sequence *dengueseq*, we type:

```
> count(dengueseq, 1)
   a    c    g    t
 3426 2240 2770 2299
```

As expected, this gives us the number of occurrences of the individual nucleotides. To find the number of occurrences of DNA words that are 2 nucleotides long, we type:

```
> count(dengueseq, 2)
  aa   ac   ag   at   ca   cc   cg   ct   ga   gc   gg   gt   ta   tc   tg   tt
1108  720  890  708  901  523  261  555  976  500  787  507  440  497  832  529
```

Note that by default the count() function includes all overlapping DNA words in a sequence. Therefore, for example, the sequence "ATG" is considered to contain two words that are two nucleotides long: "AT" and "TG".

If you type help('count'), you will see that the result (output) of the function count() is a *table* object. This means that you can use double square brackets to extract the values of elements from the table. For example, to extract the value of the third element (the number of Gs in the DEN-1 Dengue virus sequence), you can type:

```
> denguetable <- count(dengueseq,1)
> denguetable[[3]]
[1] 2770
```

The command above extracts the third element of the table produced by count(dengueseq,1), which we have stored in the table variable *denguetable*.

Alternatively, you can find the value of the element of the table in column "g" by typing:

```
> denguetable[["g"]]
[1] 2770
```

3.11 Summary

In this practical, you will have learnt to use the following R functions:

1. length() for finding the length of a vector or list
2. table() for printing out a table of the number of occurrences of each type of item in a vector or list.

These functions belong to the standard installation of R.

You have also learnt the following R functions that belong to the SeqinR package:

1. GC() for calculating the GC content for a DNA sequence
2. count() for calculating the number of occurrences of DNA words of a particular length in a DNA sequence

3.12 Links and Further Reading

Some links are included here for further reading.

For background reading on DNA sequence statistics, it is recommended to read Chapter 1 of *Introduction to Computational Genomics: a case studies approach* by Cristianini and Hahn (Cambridge University Press; www.computational-genomics.net/book/).

For more in-depth information and more examples on using the SeqinR package for sequence analysis, look at the SeqinR documentation, http://pbil.univ-lyon1.fr/software/seqinr/doc.php?lang=eng.

There is also a very nice chapter on "Analyzing Sequences", which includes examples of using SeqinR for sequence analysis, in the book *Applied statistics for bioinformatics using R* by Krijnen (available online at cran.r-project.org/doc/contrib/Krijnen-IntroBioInfStatistics.pdf).

For a more in-depth introduction to R, a good online tutorial is available on the "Kickstarting R" website, cran.r-project.org/doc/contrib/Lemon-kickstart.

There is another nice (slightly more in-depth) tutorial to R available on the "Introduction to R" website, cran.r-project.org/doc/manuals/R-intro.html.

3.13 Acknowledgements

Thank you to Noel O'Boyle for helping in using Sphinx, http://sphinx.pocoo.org, to create this document, and github, https://github.com/, to store different versions of the document as I was writing it, and readthedocs, http://readthedocs.org/, to build and distribute this document.

Many of the ideas for the examples and exercises for this chapter were inspired by the Matlab case studies on *Haemophilus influenzae* (www.computational-genomics.net/case_studies/haemophilus_demo.html) and Bacteriophage lambda (http://www.computational-genomics.net/case_studies/lambdaphage_demo.html) from the website that accompanies the book *Introduction to Computational Genomics: a case studies approach* by Cristianini and Hahn (Cambridge University Press; www.computational-genomics.net/book/).

Thank you to Jean Lobry and Simon Penel for helpful advice on using the SeqinR package.

3.14 Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

3.15 License

3.16 Exercises

Answer the following questions, using the R package. For each question, please record your answer, and what you typed into R to get this answer.

Model answers to the exercises are given in the chapter entitled Answers to the exercises on DNA Sequence Statistics.

Q1. What are the last twenty nucleotides of the Dengue virus genome sequence?

Q2. What is the length in nucleotides of the genome sequence for the bacterium *Mycobacterium leprae* strain TN (accession
Note: *Mycobacterium leprae* is a bacterium that is responsible for causing leprosy, which is classified by the WHO as a neglected tropical disease. As the genome sequence is a DNA sequence, you need to look for it in the NCBI Nucleotide database.

Q3. How many of each of the four nucleotides A, C, T and G, and any other symbols, are there in the *Mycobacterium leprae*
Note: other symbols apart from the four nucleotides A/C/T/G may appear in a sequence. They correspond to positions in the sequence that are are not clearly one base or another and they are due, for example, to sequencing uncertainties. or example, the symbol 'N' means 'aNy base', while 'R' means 'A or G' (puRine). There is a table of symbols at www.bioinformatics.org/sms/iupac.html.

Q4. What is the GC content of the *Mycobacterium leprae* TN genome sequence, when (i) all non-A/C/T/G nucleotides are inc
Hint: look at the help page for the GC() function to find out how it deals with non-A/C/T/G nucleotides.

Q5. How many of each of the four nucleotides A, C, T and G are there in the complement of the *Mycobacterium leprae* TN g
Hint: you will first need to search for a function to calculate the complement of a sequence. Once you have found out what function to use, remember to use the help() function to find out what are the arguments (inputs) and results (outputs) of that function. How does the function deal with symbols other than the four nucleotides A, C, T and G? Are the numbers of As, Cs, Ts, and Gs in the complementary sequence what you would expect?

Q6. How many occurrences of the DNA words CC, CG and GC occur in the *Mycobacterium leprae* TN genome sequence?

Q7. How many occurrences of the DNA words CC, CG and GC occur in the (i) first 1000 and (ii) last 1000 nucleotides of the
How can you check that the subsequence that you have looked at is 1000 nucleotides long?

CHAPTER

FOUR

ANSWERS TO THE EXERCISES ON DNA SEQUENCE STATISTICS

Q1. *What are the last twenty nucleotides of the DEN-1 Dengue virus genome sequence?*

To answer this, you first need to install the "SeqinR" R package, and download the DEN-1 Dengue genome sequence from the NCBI database and save it as a file "den1.fasta" in the "My Documents" folder.

Then to find the length of the DEN-1 Dengue virus genome sequence, type in the R console:

```
> library("seqinr")
> dengue <- read.fasta(file="den1.fasta")
> denguеseq <- dengue[[1]]
> length(denguеseq)
[1] 10735
```

This tells us that the length of the sequence is 10735 nucleotides. Therefore, the last 20 nucleotides are from 10716 to 10735. You can extract the sequence of these nucleotides by typing:

```
> denguеseq[10716:10735]
[1] "c" "t" "g" "t" "t" "g" "a" "a" "t" "c" "a" "a" "c" "a" "g" "g" "t" "t" "c"
[20] "t"
```

Q2. *What is the length in nucleotides of the genome sequence for the bacterium Mycobacterium leprae strain TN (accession NC_002677)?*

To get the *Mycobacterium leprae* TN genome, and find out its length, it?s necessary to first go to the NCBI website (www.ncbi.nlm.nih.gov) and search for NC_002677 and download it as a fasta format file (eg. "leprae.fasta") and save it in the "My Documents" folder. Then in R type:

```
> leprae <- read.fasta(file="leprae.fasta")
> lepraeseq <- leprae[[1]]
> length(lepraeseq)
[1] 3268203
```

Q3. *How many of each of the four nucleotides A, C, T and G, and any other symbols, are there in the Mycobacterium leprae TN genome sequence?*

Type:

```
> table(lepraeseq)
lepraeseq
     a      c      g      t
687041 938713 950202 692247
```

Q4. *What is the GC content of the Mycobacterium leprae TN genome sequence, when (i) all non-A/C/T/G nucleotides are included, (ii) non-A/C/T/G nucleotides are discarded?*

Find out how the GC function deals with non-A/C/T/G nucleotides, type:

```
> help("GC")
```

Type:

```
> GC(lepraeseq)
[1] 0.5779675
> GC(lepraeseq, exact=FALSE)
[1] 0.5779675
```

This gives 0.5779675 or 57.79675%. This is the GC content when non-A/C/T/G nucleotides are not taken into account.

The length of the *M. leprae* sequence is 3268203 bp, and it has 938713 Cs and 950202 Gs, and 687041 As and 692247 Ts. So to calculating the GC content when only considering As, Cs, Ts and Gs, we can also type:

```
> (938713+950202)/(938713+950202+687041+692247)
[1] 0.5779675
```

To take non-A/C/T/G nucleotides into account when calculating GC, type:

```
> GC(lepraeseq, exact=TRUE)
[1] 0.5779675
```

We get the same answer as when we ignored non-A/C/G/T nucleotides. This is actually because the *M. leprae* TN sequence does not have any non-A/C/G/T nucleotides.

However, many other genome sequences do contain non-A/C/G/T nucleotides. Note that under ?Details? in the box that appears when you type ?help(?GC?)?, it says : "When exact is set to TRUE the G+C content is estimated with ambiguous bases taken into account. Note that this is time expensive. A first pass is made on non-ambiguous bases to estimate the probabilities of the four bases in the sequence. They are then used to weight the contributions of ambiguous bases to the G+C content."

Q5. *How many of each of the four nucleotides A, C, T and G are there in the complement of the Mycobacterium leprae TN genome sequence?*

First you need to search for a function to calculate reverse complement, eg. by typing:

```
> help.search("complement")
```

You will find that there is a function seqinr::comp that complements a nucleic acid sequence. This means it is a function in the SeqinR package.

Find out how to use this function by typing:

```
> help("comp")
```

The help says "Undefined values are returned as NA". This means that the complement of non-A/C/T/G symbols will be returned as NA.

To find the number of A, C, T, and G in the reverse complement type:

```
> complepraeseq <- comp(lepraeseq)
> table(complepraeseq)
 complepraeseq
     a      c      g      t
692247 950202 938713 687041
```

Note that in the *M. leprae* sequence we had 687041 As, in the complement have 687041 Ts. In the *M. leprae* sequence we had 938713 Cs, in the complement have 938713 Gs. In the *M. leprae* sequence we had 950202 Gs, in the complement have 950202 Cs. In the *M. leprae* sequence we had 692247 Ts, in the complement have 692247 As.

Q6. *How many occurrences of the DNA words CC, CG and GC occur in the Mycobacterium leprae TN genome sequence?*

```
> count(lepraeseq, 2)
    aa     ac     ag     at     ca     cc     cg     ct     ga     gc     gg
149718 206961 170846 159516 224666 236971 306986 170089 203397 293261 243071
    gt     ta     tc     tg     tt
210473 109259 201520 229299 152169
```

Get count for CC is 236,971; count for CG is 306,986; count for GC is 293,261.

Q7. *How many occurrences of the DNA words CC, CG and GC occur in the (i) first 1000 and (ii) last 1000 nucleotides of the Mycobacterium leprae TN genome sequence?*

Type:

```
> length(lepraeseq)
[1] 3268203
```

to find the length of the *M. leprae* genome sequence. It is 3,268,203 bp. Therefore the first 1000 nucleotides will have indices 1-1000, and the last thousand nucleotides will have indices 3267204-3268203. We find the count of DNA words of length 2 by typing:

```
> count(lepraeseq[1:1000],2)
 aa ac ag at ca cc cg ct ga gc gg gt ta tc tg tt
 78 95 51 49 85 82 92 54 68 63 39 43 42 73 31 54
> count(lepraeseq[3267204:3268203],2)
 aa ac ag at ca cc cg ct ga gc gg gt ta tc tg tt
 70 85 44 55 94 81 87 50 53 75 49 51 36 72 48 49
```

To check that the subsequences that you looked at are 1000 nucleotides long, you can type:

```
> length(lepraeseq[1:1000])
[1] 1000
> length(lepraeseq[3267204:3268203])
[1] 1000
```

4.1 Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

4.2 License

CHAPTER

FIVE

DNA SEQUENCE STATISTICS (2)

5.1 A little more introduction to R

In the previous chapters, you learnt about variables in R, such as scalars, vectors, and lists. You also learnt how to use functions to carry out operations on variables, for example, using the log10() function to calculate the log to the base 10 of a scalar variable *x*, or using the mean() function to calculate the average of the values in a vector variable *myvector*:

```
> x <- 100
> log10(x)
[1] 2
> myvector <- c(30,16,303,99,11,111)
> mean(myvector)
[1] 95
```

You also learnt that you can extract an element of a vector by typing the vector name with the index of that element given in square brackets. For example, to get the value of the 3rd element in the vector *myvector*, we type:

```
> myvector[3]
[1] 303
```

A useful function in R is the seq() function, which can be used to create a sequence of numbers that run from a particular number to another particular number. For example, if we want to create the sequence of numbers from 1-100 in steps of 1 (ie. 1, 2, 3, 4, ... 97, 98, 99, 100), we can type:

```
> seq(1, 100, by = 1)
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
```

We can change the step size by altering the value of the "by" argument given to the function seq(). For example, if we want to create a sequence of numbers from 1-100 in steps of 2 (ie. 1, 3, 5, 7, ... 97, 99), we can type:

```
> seq(1, 100, by = 2)
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
[26] 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

In R, just as in programming languages such as Python, it is possible to write a *for loop* to carry out the same command several times. For example, if we want to print out the square of each number between 1 and 10, we can write the following for loop:

```
> for (i in 1:10) { print (i*i) }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

In the *for loop* above, the variable *i* is a counter for the number of cycles through the loop. In the first cycle through the loop, the value of *i* is 1, and so *i* * *i* =1 is printed out. In the second cycle through the loop, the value of *i* is 2, and so *i* * *i* =4 is printed out. In the third cycle through the loop, the value of *i* is 3, and so *i* * *i* =9 is printed out. The loop continues until the value of *i* is 10. In the tenth cycle through the loop, the value of *i* is 10, and so *i* * *i* =100 is printed out.

Note that the commands that are to be carried out at each cycle of the *for loop* must be enclosed within curly brackets ("{" and "}").

You can also give a *for loop* a vector of numbers containing the values that you want the counter *i* to take in subsequent cycles. For example, you can make a vector *avector* containing the numbers 2, 9, 100, and 133, and write a *for loop* to print out the square of each number in vector *avector*:

```
> avector <- c(2, 9, 100, 133)
> for (i in avector) { print (i*i) }
[1] 4
[1] 81
[1] 10000
[1] 17689
```

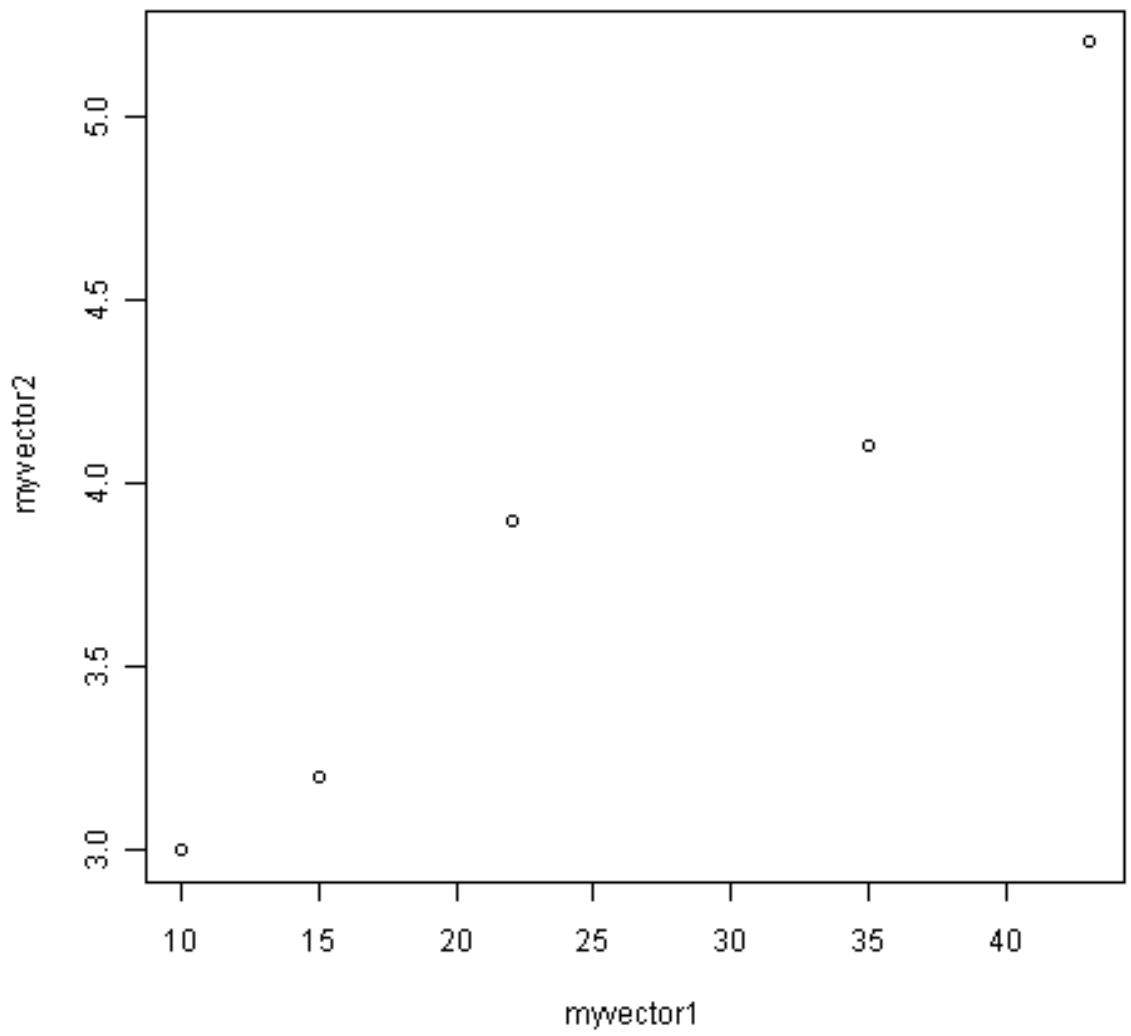
How can we use a *for loop* to print out the square of every second number between 1 and 10? The answer is to use the seq() function to tell the *for loop* to take every second number between 1 and 10:

```
> for (i in seq(1, 10, by = 2)) { print (i*i) }
[1] 1
[1] 9
[1] 25
[1] 49
[1] 81
```

In the first cycle of this loop, the value of *i* is 1, and so *i* * *i* =1 is printed out. In the second cycle through the loop, the value of *i* is 3, and so *i* * *i* =9 is printed out. The loop continues until the value of *i* is 9. In the fifth cycle through the loop, the value of *i* is 9, and so *i* * *i* =81 is printed out.

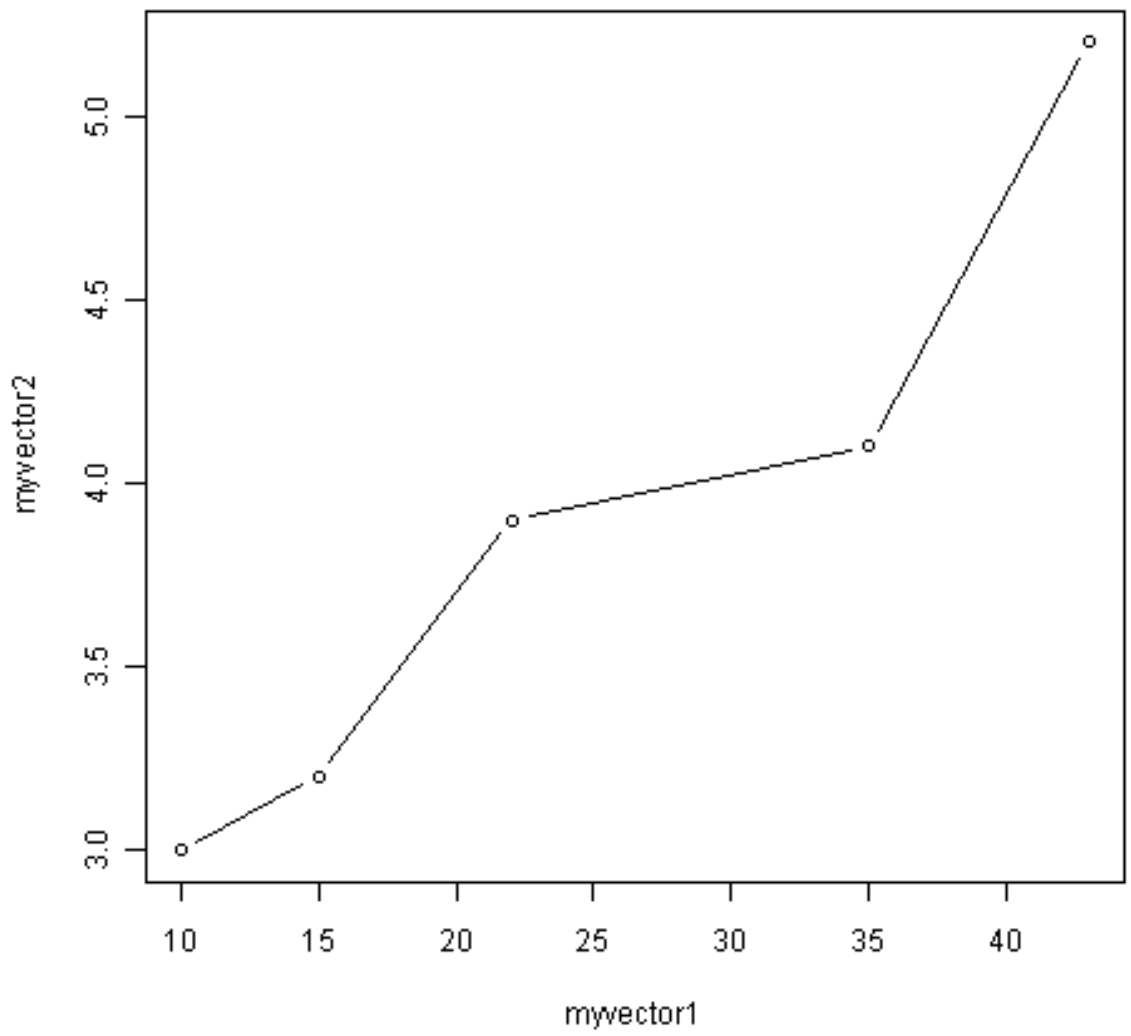
R allows the production of a variety of plots, including scatterplots, histograms, piecharts, and boxplots. For example, if you have two vectors of numbers *myvector1* and *myvector2*, you can plot a scatterplot of the values in *myvector1* against the values in *myvector2* using the plot() function. If you want to label the axes on the plot, you can do this by giving the plot() function values for its optional arguments *xlab* and *ylab*:

```
> myvector1 <- c(10, 15, 22, 35, 43)
> myvector2 <- c(3, 3.2, 3.9, 4.1, 5.2)
> plot(myvector1, myvector2, xlab="myvector1", ylab="myvector2")
```



If you look at the help page for the plot() function, you will see that there are lots of optional arguments (inputs) that it can take that. For example, one optional argument is the *type* argument, that determines the type of the plot. By default, plot() will draw a dot at each data point, but if we set *type* to be "b", then it will also draw a line between each subsequent data point:

```
> plot(myvector1, myvector2, xlab="myvector1", ylab="myvector2", type="b")
```



We have been using built-in R functions such as mean(), length(), print(), plot(), etc. We can also create our own functions in R to do calculations that you want to carry out very often on different input data sets. For example, we can create a function to calculate the value of 20 plus square of some input number:

```
> myfunction <- function(x) { return(20 + (x*x)) }
```

This function will calculate the square of a number (*x*), and then add 20 to that value. The return() statement returns the calculated value. Once you have typed in this function, the function is then available for use. For example, we can use the function for different input numbers (eg. 10, 25):

```
> myfunction(10)
[1] 120
> myfunction(25)
[1] 645
```

You can view the code that makes up a function by typing its name (without any parentheses). For example, we can try this by typing "myfunction":

```
> myfunction
function(x) { return(20 + (x*x)) }
```

When you are typing R, if you want to, you can write comments by writing the comment text after the "#" sign. This can be useful if you want to write some R commands that other people need to read and understand. R will ignore the comments when it is executing the commands. For example, you may want to write a comment to explain what the function log10() does:

```
> x <- 100
> log10(x) # Finds the log to the base 10 of variable x.
[1] 2
```

5.2 Reading sequence data with SeqinR

In the previous chapter you learnt how to use to search for and download the sequence data for a given NCBI accession from the NCBI Sequence Database. For example, you could have downloaded the sequence data for a the DEN-1 Dengue virus sequence (NCBI accession NC_001477), and stored it on a file on your computer (eg. "den1.fasta").

Once you have downloaded the sequence data for a particular NCBI accession, and stored it on a file on your computer, you can then read it into R by using read.fasta function from the SeqinR R package. For example, if you have stored the DEN-1 Dengue virus sequence in a file "den1.fasta", you can type:

```
> library("seqinr")                          # Load the SeqinR package.
> dengue <- read.fasta(file = "den1.fasta")  # Read in the file "den1.fasta".
> dengueseq <- dengue[[1]]                   # Put the sequence in a vector called "dengueseq".
```

Once you have retrieved a sequence from the NCBI Sequence Database and stored it in a vector variable such as *dengueseq* in the example above, it is possible to extract subsequence of the sequence by type the name of the vector (eg. *dengueseq*) followed by the square brackets containing the indices for those nucleotides. For example, to obtain nucleotides 452-535 of the DEN-1 Dengue virus genome, we can type:

```
> dengueseq[452:535]
 [1] "c" "g" "a" "g" "g" "g" "g" "g" "a" "g" "a" "g" "c" "c" "g" "c" "a" "c" "a"
[20] "t" "g" "a" "t" "a" "g" "t" "t" "a" "g" "c" "a" "a" "g" "c" "a" "g" "g" "a"
[39] "a" "a" "g" "a" "g" "g" "a" "a" "a" "a" "t" "c" "a" "c" "t" "t" "t" "t" "g"
[58] "t" "t" "t" "a" "a" "g" "a" "c" "c" "t" "c" "t" "g" "c" "a" "g" "g" "t" "g"
[77] "t" "c" "a" "a" "c" "a" "t" "g"
```

5.3 Local variation in GC content

In the previous chapter, you learnt that to find out the GC content of a genome sequence (percentage of nucleotides in a genome sequence that are Gs or Cs), you can use the GC() function in the SeqinR package. For example, to find the GC content of the DEN-1 Dengue virus sequence that we have stored in the vector *dengueseq*, we can type:

```
> GC(dengueseq)
[1] 0.4666977
```

The output of the GC() is the fraction of nucleotides in a sequence that are Gs or Cs, so to convert it to a percentage we need to multiply by 100. Thus, the GC content of the DEN-1 Dengue virus genome is about 0.467 or 46.7%.

Although the GC content of the whole DEN-1 Dengue virus genome sequence is about 46.7%, there is probably local variation in GC content within the genome. That is, some regions of the genome sequence may have GC contents quite a bit higher than 46.7%, while some regions of the genome sequence may have GC contents that are quite a big lower than 46.7%. Local fluctuations in GC content within the genome sequence can provide different interesting information, for example, they may reveal cases of horizontal transfer or reveal biases in mutation.

If a chunk of DNA has moved by horizontal transfer from the genome of a species with low GC content to a species with high GC content, the chunk of horizontally transferred DNA could be detected as a region of unusually low GC content in the high-GC recipient genome.

On the other hand, a region unusually low GC content in an otherwise high-GC content genome could also arise due to biases in mutation in that region of the genome, for example, if mutations from Gs/Cs to Ts/As are more common for some reason in that region of the genome than in the rest of the genome.

5.4 A sliding window analysis of GC content

In order to study local variation in GC content within a genome sequence, we could calculate the GC content for small chunks of the genome sequence. The DEN-1 Dengue virus genome sequence is 10735 nucleotides long. To study variation in GC content within the genome sequence, we could calculate the GC content of chunks of the DEN-1 Dengue virus genome, for example, for each 2000-nucleotide chunk of the genome sequence:

```
> GC(dengueseq[1:2000])       # Calculate the GC content of nucleotides 1-2000 of the Dengue genom
[1] 0.465
> GC(dengueseq[2001:4000])    # Calculate the GC content of nucleotides 2001-4000 of the Dengue ge
[1] 0.4525
> GC(dengueseq[4001:6000])    # Calculate the GC content of nucleotides 4001-6000 of the Dengue ge
[1] 0.4705
> GC(dengueseq[6001:8000])    # Calculate the GC content of nucleotides 6001-8000 of the Dengue ge
[1] 0.479
> GC(dengueseq[8001:10000])   # Calculate the GC content of nucleotides 8001-10000 of the Dengue g
[1] 0.4545
> GC(dengueseq[10001:10735]) # Calculate the GC content of nucleotides 10001-10735 of the Dengue
[1] 0.4993197
```

From the output of the above calculations, we see that the region of the DEN-1 Dengue virus genome from nucleotides 1-2000 has a GC content of 46.5%, while the region of the Dengue genome from nucleotides 2001-4000 has a GC content of about 45.3%. Thus, there seems to be some local variation in GC content within the Dengue genome sequence.

Instead of typing in the commands above to tell R to calculate the GC content for each 2000-nucleotide chunk of the DEN-1 Dengue genome, we can use a *for loop* to carry out the same calculations, but by typing far fewer commands. That is, we can use a *for loop* to take each 2000-nucleotide chunk of the DEN-1 Dengue virus genome, and to calculate the GC content of each 2000-nucleotide chunk. Below we will explain the following *for loop* that has been written for this purpose:

```
> starts <- seq(1, length(dengueseq)-2000, by = 2000)
> starts
[1]    1 2001 4001 6001 8001
> n <- length(starts)    # Find the length of the vector "starts"
> for (i in 1:n) {
        chunk <- dengueseq[starts[i]:(starts[i]+1999)]
        chunkGC <- GC(chunk)
        print (chunkGC)
    }
[1] 0.465
[1] 0.4525
[1] 0.4705
[1] 0.479
[1] 0.4545
```

The command starts <- seq(1, length(dengueseq)-2000, by = 2000) stores the result of the seq() command in the vector *starts*, which contains the values 1, 2001, 4001, 6001, and 8001.

We set the variable *n* to be equal to the number of elements in the vector *starts*, so it will be 5 here, since the vector *starts* contains the five elements 1, 2001, 4001, 6001 and 8001. The line for (i in 1:n) means that the counter *i* will take values of 1-5 in subsequent cycles of the *for loop*. The *for loop* above is spread over several lines. However,

R will not execute the commands within the *for loop* until you have typed the final "}" at the end of the *for loop* and pressed "Return".

Each of the three commands within the *for loop* are carried out in each cycle of the loop. In the first cycle of the loop, *i* is 1, the vector variable *chunk* is used to store the region from nucleotides 1-2000 of the Dengue virus sequence, the GC content of that region is calculated and stored in the variable *chunkGC*, and the value of *chunkGC* is printed out.

In the second cycle of the loop, *i* is 2, the vector variable *chunk* is used to store the region from nucleotides 2001-4000 of the Dengue virus sequence, the GC content of that region is calculated and stored in the variable *chunkGC*, and the value of *chunkGC* is printed out. The loop continues until the value of *i* is 5. In the fifth cycle through the loop, the value of *i* is 5, and so the GC content of the region from nucleotides 8001-10000 is printed out.

Note that we stop the loop when we are looking at the region from nucleotides 8001-10000, instead of continuing to another cycle of the loop where the region under examiniation would be from nucleotides 10001-12000. The reason for this is because the length of the Dengue virus genome sequence is just 10735 nucleotides, so there is not a full 2000-nucleotide region from nucleotide 10001 to the end of the sequence at nucleotide 10735.

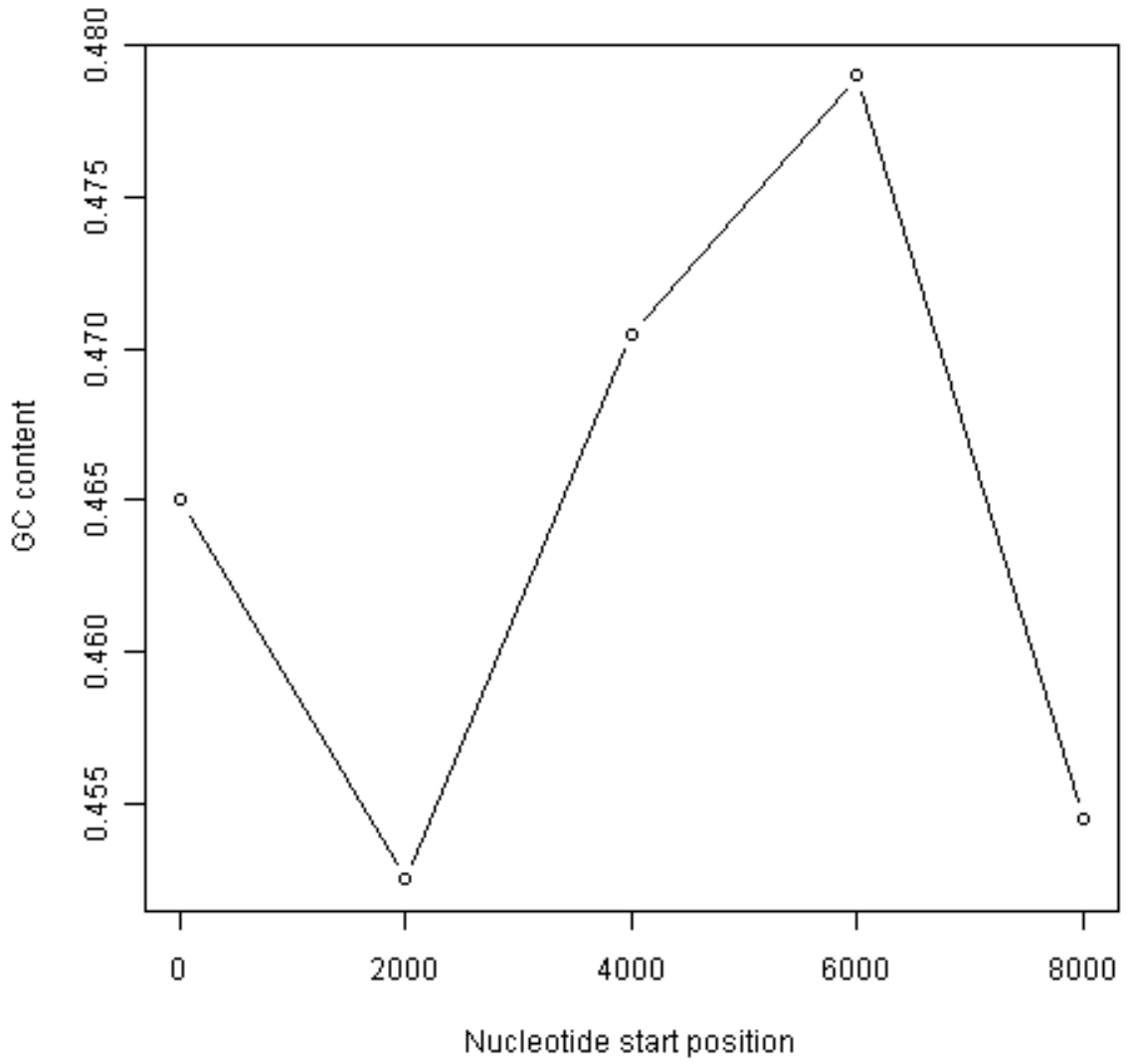
The above analysis of local variation in GC content is what is known as a *sliding window analysis of GC content*. By calculating the GC content in each 2000-nucleotide chunk of the Dengue virus genome, you are effectively sliding a 2000-nucleotide *window* along the DNA sequence from start to end, and calculating the GC content in each non-overlapping window (chunk of DNA).

Note that this sliding window analysis of GC content is a slightly simplified version of the method usually carried out by bioinformaticians. In this simplified version, we have calculated the GC content in non-overlapping windows along a DNA sequence. However, it is more usual to calculate GC content in overlapping windows along a sequence, althought that makes the code slightly more complicated.

5.5 A sliding window plot of GC content

It is common to use the data generated from a sliding window analysis to create a *sliding window plot of GC content*. To create a sliding window plot of GC content, you plot the local GC content in each window of the genome, versus the nucleotide position of the start of each window. We can create a sliding window plot of GC content by typing:

```
> starts <- seq(1, length(dengueseq)-2000, by = 2000)
> n <- length(starts)     # Find the length of the vector "starts"
> chunkGCs <- numeric(n) # Make a vector of the same length as vector "starts", but just containi
> for (i in 1:n) {
        chunk <- dengueseq[starts[i]:(starts[i]+1999)]
        chunkGC <- GC(chunk)
        print(chunkGC)
        chunkGCs[i] <- chunkGC
    }
> plot(starts,chunkGCs,type="b",xlab="Nucleotide start position",ylab="GC content")
```



In the code above, the line chunkGCs <- numeric(n) makes a new vector *chunkGCs* which has the same number of elements as the vector *starts* (5 elements here). This vector *chunkGCs* is then used within the *for loop* for storing the GC content of each chunk of DNA.

After the loop, the vector *starts* can be plotted against the vector *chunkGCs* using the plot() function, to get a plot of GC content against nucleotide position in the genome sequence. This is a *sliding window plot of GC content*.

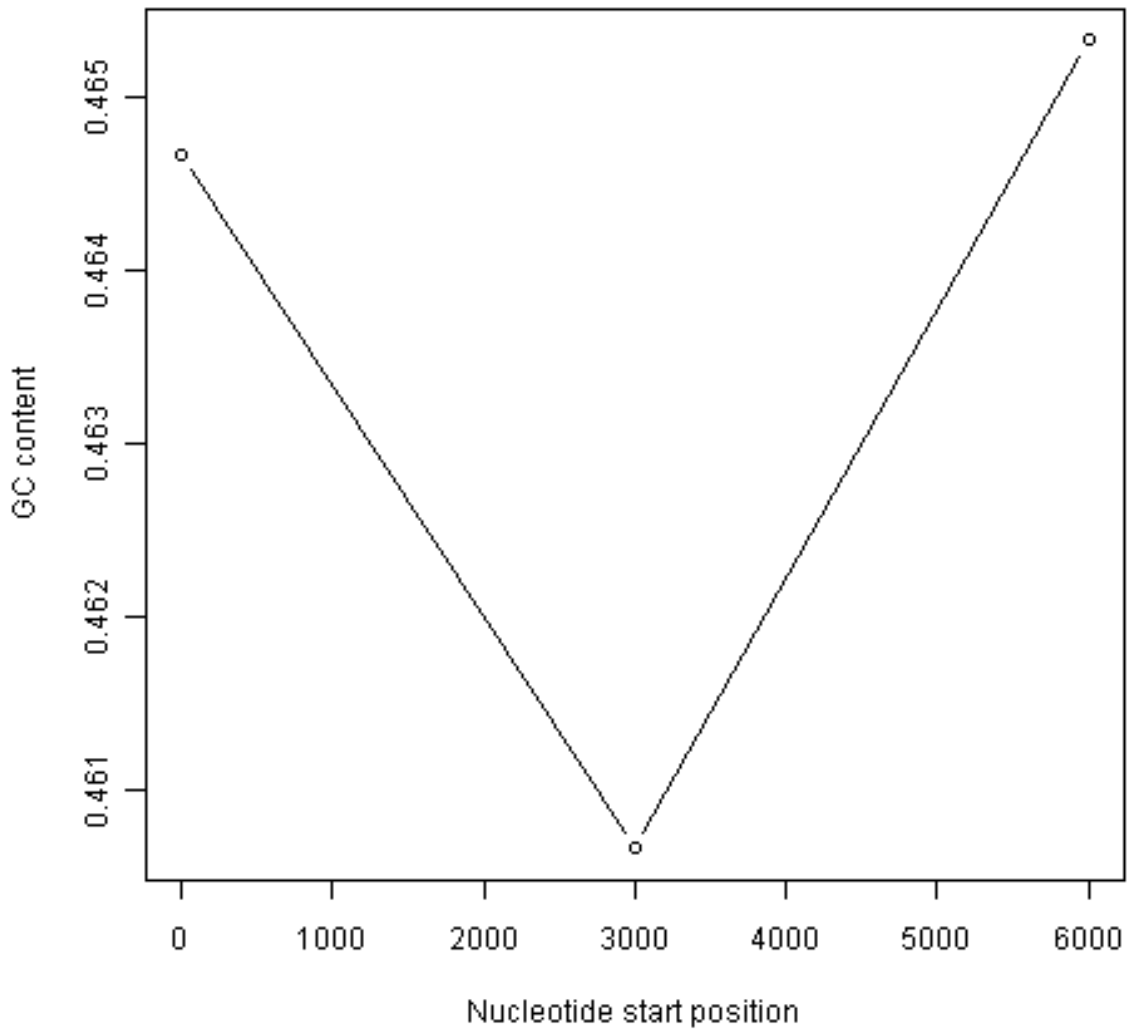
You may want to use the code above to create sliding window plots of GC content of different species' genomes, using different windowsizes. Therefore, it makes sense to write a function to do the sliding window plot, that can take the windowsize that the user wants to use and the sequence that the user wants to study as arguments (inputs):

```
> slidingwindowplot <- function(windowsize, inputseq)
{
   starts <- seq(1, length(inputseq)-windowsize, by = windowsize)
   n <- length(starts)    # Find the length of the vector "starts"
   chunkGCs <- numeric(n) # Make a vector of the same length as vector "starts", but just contain
   for (i in 1:n) {
        chunk <- inputseq[starts[i]:(starts[i]+windowsize-1)]
        chunkGC <- GC(chunk)
        print(chunkGC)
        chunkGCs[i] <- chunkGC
```

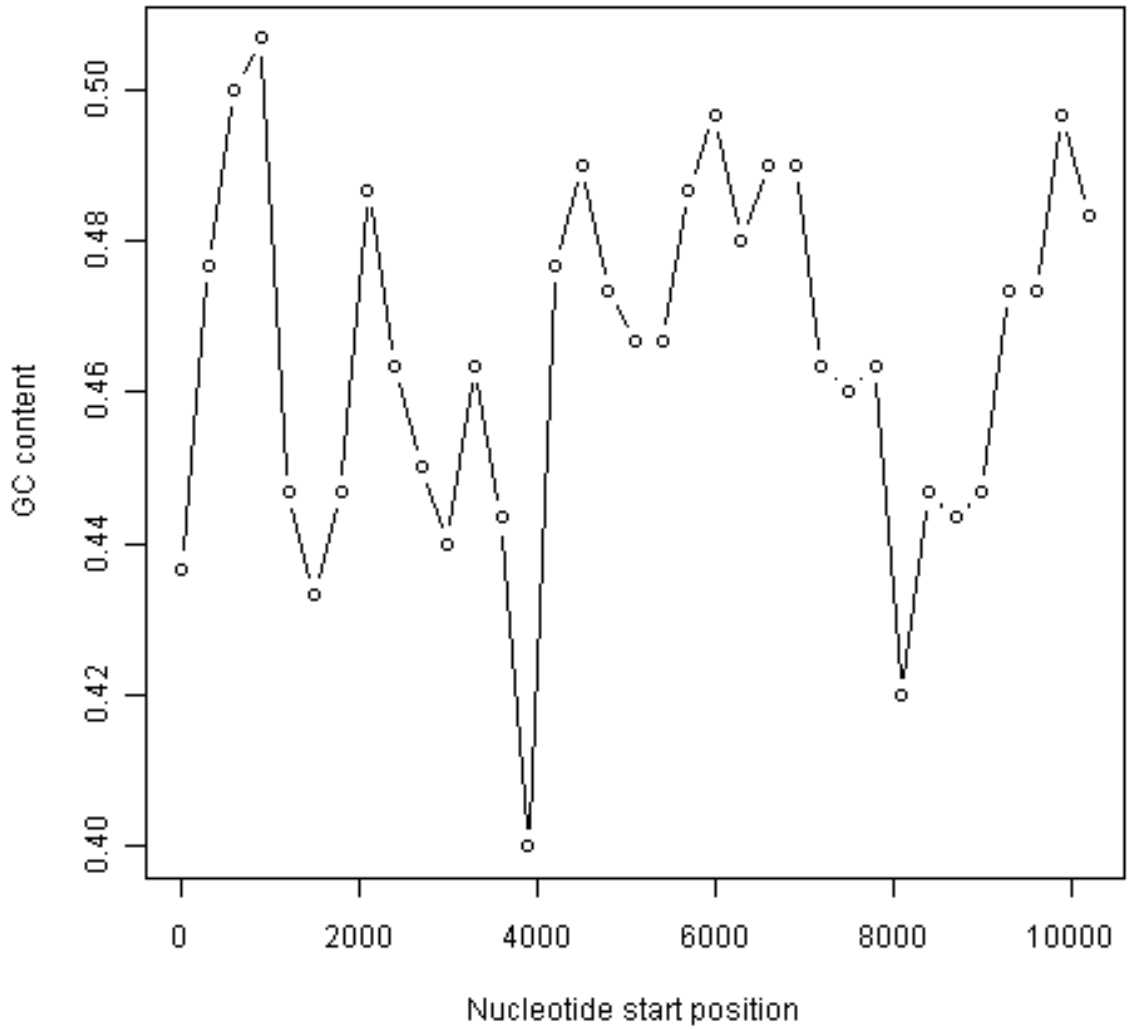
```
    }
    plot(starts,chunkGCs,type="b",xlab="Nucleotide start position",ylab="GC content")
}
```

This function will make a sliding window plot of GC content for a particular input sequence *inputseq* specified by the user, using a particular windowsize *windowsize* specified by the user. Once you have typed in this function once, you can use it again and again to make sliding window plots of GC contents for different input DNA sequences, with different windowsizes. For example, you could create two different sliding window plots of the DEN-1 Dengue virus genome sequence, using windowsizes of 3000 and 300 nucleotides, respectively:

```
> slidingwindowplot(3000, denguseq)
```




```
> slidingwindowplot(300, denguseq)
```



5.6 Over-represented and under-represented DNA words

In the previous chapter, you learnt that the count() function in the SeqinR R package can calculate the frequency of all DNA words of a certain length in a DNA sequence. For example, if you want to know the frequency of all DNA words that are 2 nucleotides long in the Dengue virus genome sequence, you can type:

```
> count(dengueseq, 2)
  aa   ac   ag   at   ca   cc   cg   ct   ga   gc   gg   gt   ta   tc   tg   tt
1108  720  890  708  901  523  261  555  976  500  787  507  440  497  832  529
```

It is interesting to identify DNA words that are two nucleotides long ("dinucleotides", ie. "AT", "AC", etc.) that are over-represented or under-represented in a DNA sequence. If a particular DNA word is *over-represented* in a sequence, it means that it occurs many more times in the sequence than you would have expected by chance. Similarly, if a particular DNA word is *under-represented* in a sequence, it means it occurs far fewer times in the sequence than you would have expected.

A statistic called ρ (Rho) is used to measure how over- or under-represented a particular DNA word is. For a 2-nucleotide (dinucleotide) DNA word ρ is calculated as:

$\rho(xy) = f_{xy}/(f_x * f_y)$,

where f_{xy} and f_x are the frequencies of the DNA words *xy* and *x* in the DNA sequence under study. For example, the value of ρ for the DNA word "TA" can be calculated as: $\rho(TA) = f_{TA}/(f_T * f_A)$, where f_{TA}, f_T and f_A are the frequencies of the DNA words "TA", "T" and "A" in the DNA sequence.

The idea behind the ρ statistic is that, if a DNA sequence had a frequency f_x of a 1-nucleotide DNA word *x*, and a frequency f_y of a 1-nucleotide DNA word *y*, then we expect the frequency of the 2-nucleotide DNA word *xy* to be $f_x * f_y$. That is, the frequencies of the 2-nucleotide DNA words in a sequence are expected to be equal the products of the specific frequencies of the two nucleotides that compose them. If this were true, then ρ would be equal to 1. If we find that ρ is much greater than 1 for a particular 2-nucleotide word in a sequence, it indicates that that 2-nucleotide word is much more common in that sequence than expected (ie. it is *over-represented*).

For example, say that your input sequence has only 5% Ts (ie. $f_T = 0.05$). In a random DNA sequence with 5% Ts, you would expect to see the word "TT" very infrequently. In fact, we would only expect 0.05 * 0.05=0.0025 (0.25%) of 2-nucleotide words to be TTs (ie. we expect $f_{TT} = f_T * f_T$). This is because Ts are rare, so they are expected to be adjacent to each other very infrequently if the few Ts are randomly scattered throughout the DNA. Therefore, if you see lots of TT 2-nucleotide words in your real input sequence (eg. $f_{TT} = 0.3$, so $\rho = 0.3/0.0025 = 120$), you would suspect that natural selection has acted to increase the number of occurrences of the TT word in the sequence (presumably because it has some beneficial biological function).

To find over-represented and under-represented DNA words that are 2 nucleotides long in the DEN-1 Dengue virus sequence, we can calculate the ρ statistic for each 2-nucleotide word in the sequence. For example, given the number of occurrences of the individual nucleotides A, C, G and T in the Dengue sequence, and the number of occurrences of the DNA word GC in the sequence (500, from above), we can calculate the value of ρ for the 2-nucleotide DNA word "GC", using the formula $\rho(GC) = f_{GC}/(f_G * f_C)$, where f_{GC}, f_G and f_C are the frequencies of the DNA words "GC", "G" and "C" in the DNA sequence:

```
> count(dengueseq, 1) # Get the number of occurrences of 1-nucleotide DNA words
    a     c     g     t
 3426 2240 2770 2299
> 2770/(3426+2240+2770+2299) # Get fG
[1] 0.2580345
> 2240/(3426+2240+2770+2299) # Get fC
[1] 0.2086633
> count(dengueseq, 2) # Get the number of occurrences of 2-nucleotide DNA words
  aa   ac   ag   at   ca   cc   cg   ct   ga   gc   gg   gt   ta   tc   tg   tt
1108  720  890  708  901  523  261  555  976  500  787  507  440  497  832  529
> 500/(1108+720+890+708+901+523+261+555+976+500+787+507+440+497+832+529) # Get fGC
[1] 0.04658096
> 0.04658096/(0.2580345*0.2086633) # Get rho(GC)
[1] 0.8651364
```

We calculate a value of ρ(GC) of approximately 0.82. This means that the DNA word "GC" is about 0.82 times as common in the DEN-1 Dengue virus sequence than expected. That is, it seems to be slightly under-represented.

Note that if the ratio of the observed to expected frequency of a particular DNA word is very low or very high, then we would suspect that there is a statistical under-representation or over-representation of that DNA word. However, to be sure that this over- or under-representation is statistically significant, we would need to do a statistical test. We will not deal with the topic of how to carry out the statistical test here.

5.7 Summary

In this chapter, you will have learnt to use the following R functions:

1. seq() for creating a sequence of numbers
2. print() for printing out the value of a variable
3. plot() for making a plot (eg. a scatterplot)
4. numeric() for making a numeric vector of a particular length

5. function() for making a function

All of these functions belong to the standard installation of R. You also learnt how to use *for loops* to carry out the same operation again and again, each time on different inputs.

5.8 Links and Further Reading

Some links are included here for further reading.

For background reading on DNA sequence statistics, it is recommended to read Chapter 1 of *Introduction to Computational Genomics: a case studies approach* by Cristianini and Hahn (Cambridge University Press; www.computational-genomics.net/book/).

For more in-depth information and more examples on using the SeqinR package for sequence analysis, look at the SeqinR documentation, http://pbil.univ-lyon1.fr/software/seqinr/doc.php?lang=eng.

There is also a very nice chapter on "Analyzing Sequences", which includes examples of using SeqinR for sequence analysis, in the book *Applied statistics for bioinformatics using R* by Krijnen (available online at cran.r-project.org/doc/contrib/Krijnen-IntroBioInfStatistics.pdf).

For a more in-depth introduction to R, a good online tutorial is available on the "Kickstarting R" website, cran.r-project.org/doc/contrib/Lemon-kickstart.

There is another nice (slightly more in-depth) tutorial to R available on the "Introduction to R" website, cran.r-project.org/doc/manuals/R-intro.html.

5.9 Acknowledgements

Thank you to Noel O'Boyle for helping in using Sphinx, http://sphinx.pocoo.org, to create this document, and github, https://github.com/, to store different versions of the document as I was writing it, and readthedocs, http://readthedocs.org/, to build and distribute this document.

Many of the ideas for the examples and exercises for this chapter were inspired by the Matlab case studies on *Haemophilus influenzae* (www.computational-genomics.net/case_studies/haemophilus_demo.html) and Bacteriophage lambda (http://www.computational-genomics.net/case_studies/lambdaphage_demo.html) from the website that accompanies the book *Introduction to Computational Genomics: a case studies approach* by Cristianini and Hahn (Cambridge University Press; www.computational-genomics.net/book/).

Thank you to Jean Lobry and Simon Penel for helpful advice on using the SeqinR package.

5.10 Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

5.11 License

5.12 Exercises

Answer the following questions, using the R package. For each question, please record your answer, and what you typed into R to get this answer.

Q1. Draw a sliding window plot of GC content in the DEN-1 Dengue virus genome, using a window size of 200 nucleotides.
Make a sketch of each plot that you draw. At what position (in base-pairs) in the genome is there the largest change in local GC content (approximate position is fine here)? Compare the sliding window plots of GC content created using window sizes of 200 and 2000 nucleotides. How does window size affect your ability to detect differences within the Dengue virus genome?

Q2. Draw a sliding window plot of GC content in the genome sequence for the bacterium *Mycobacterium leprae* strain TN (a
Make a sketch of each plot that you draw. Write down the approximate nucleotide position of the highest peak or lowest trough that you see. Why do you think a window size of 20000 nucleotides was chosen? What do you see if you use a much smaller windowsize (eg. 200 nucleotides) or a much larger windowsize (eg. 200,000 nucleotides)?

Q3. Write a function to calculate the AT content of a DNA sequence (ie. the fraction of the nucleotides in the sequence that a
Hint: use the function count() to make a table containing the number of As, Gs, Ts and Cs in the sequence. Remember that function count() produces a table object, and you can access the elements of a table object using double square brackets. Do you notice a relationship between the AT content of the *Mycobacterium leprae* TN genome, and its GC content?

Q4. Write a function to draw a sliding window plot of AT content. Use it to make a sliding window plot of AT content along
Make a sketch of the plot that you draw.

Q5. Is the 3-nucleotide word GAC GC over-represented or under-represented in the *Mycobacterium leprae* TN genome sequ
What is the frequency of this word in the sequence? What is the expected frequency of this word in the sequence? What is the ρ (Rho) value for this word? How would you figure out whether there is already an R function to calculate ρ (Rho)? Is there one that you could use?

CHAPTER

SIX

ANSWERS TO THE EXERCISES ON DNA SEQUENCE STATISTICS (2)

Q1. *Draw a sliding window plot of GC content in the DEN-1 Dengue virus genome, using a window size of 200 nucleotides. Do you see any regions of unusual DNA content in the genome (eg. a high peak or low trough)?*

To do this, you first need to download the DEN-1 Dengue virus sequence from the NCBI website http://www.ncbi.nlm.nih.gov. To do this follow the steps in the chapter DNA Sequence Statistics.

Then read the sequence into R using the SeqinR library:

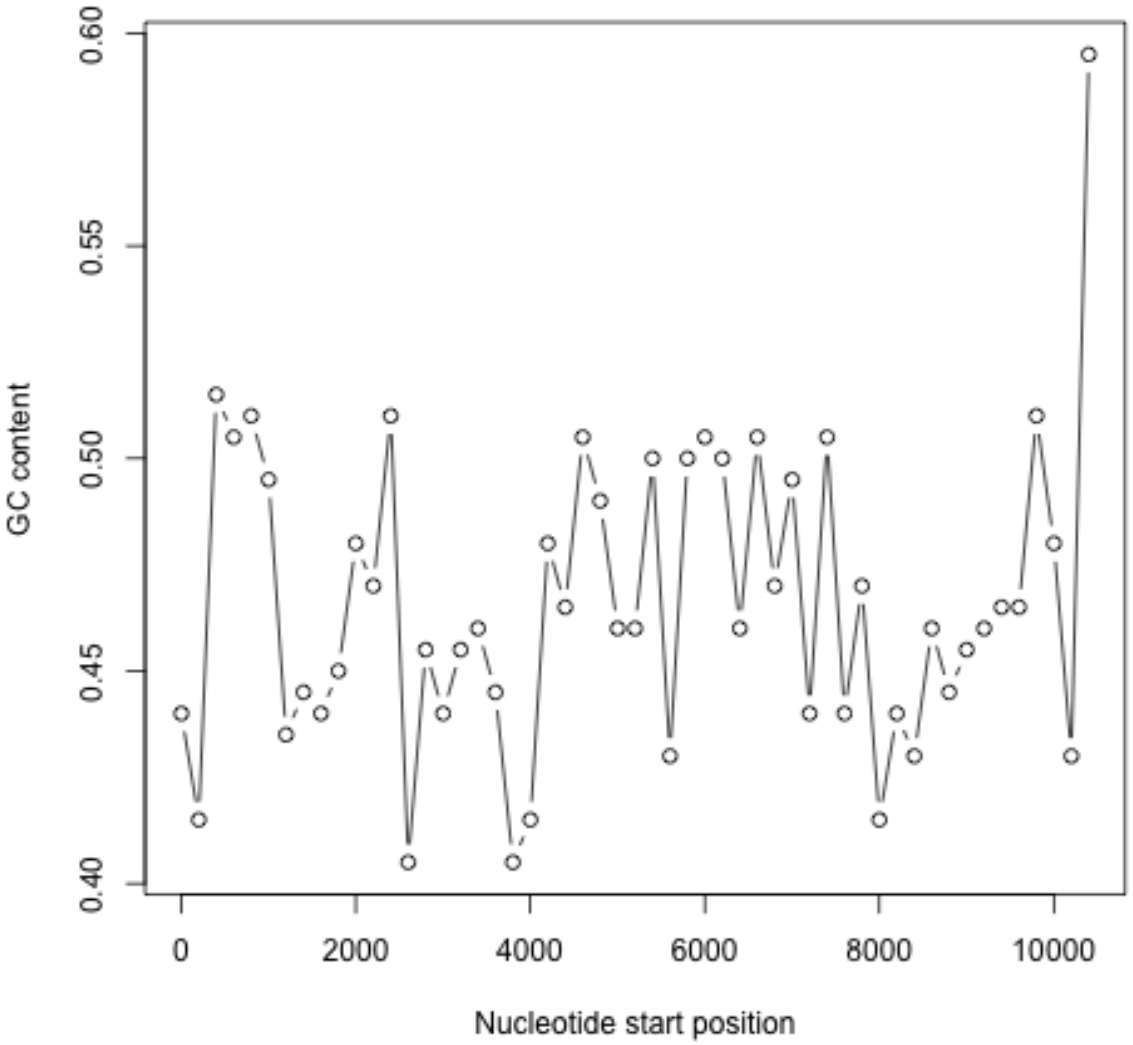
```
> library("seqinr")
> dengue <- read.fasta(file = "den1.fasta")
> denguеseq <- dengue[[1]]
```

Then write a function to make a sliding window plot:

```
> slidingwindowplot <- function(windowsize, inputseq)
  {
     starts <- seq(1, length(inputseq)-windowsize, by = windowsize)
     n <- length(starts)
     chunkGCs <- numeric(n)
     for (i in 1:n) {
        chunk <- inputseq[starts[i]:(starts[i]+windowsize-1)]
        chunkGC <- GC(chunk)
        chunkGCs[i] <- chunkGC
     }
     plot(starts,chunkGCs,type="b",xlab="Nucleotide start position",ylab="GC content")
  }
```

Then make a sliding window plot with a window size of 200 nucleotides:

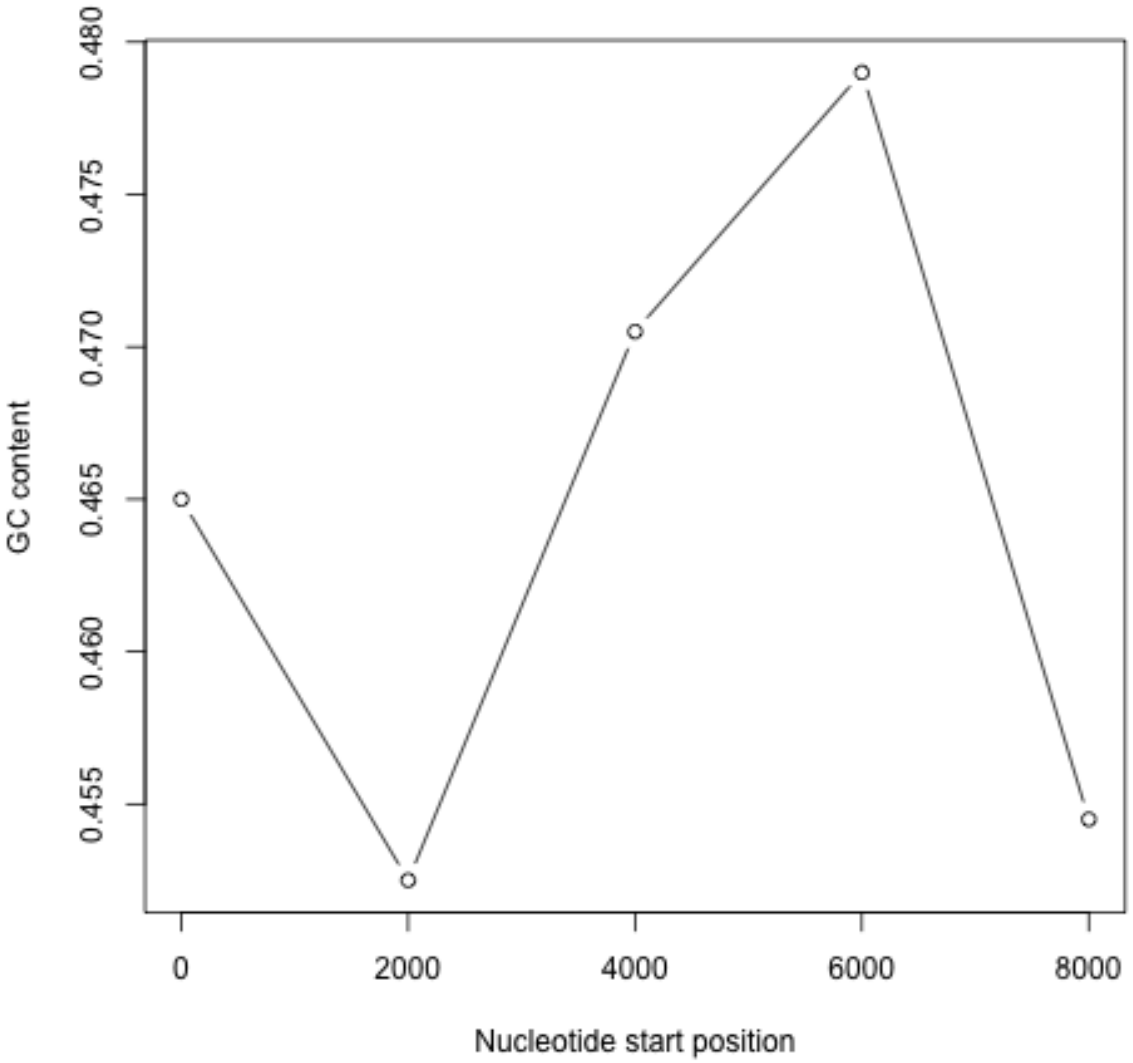
```
> slidingwindowplot(200, denguеseq)
```



The GC content varies from about 45% to about 50% throughout the DEN-1 Dengue virus genome, with some noticeable troughs at about 2500 bases and at about 4000 bases along the sequence, where the GC content drops to about 40%. There is no strong difference between the start and end of the genome, although from around bases 4000-7000 the GC content is quite high (about 50%), and from about 2500-3500 and 7000-9000 bases the GC content is relatively low (about 43-47%).

We can also make a sliding window plot of GC content using a window size of 2000 nucleotides:

```
> slidingwindowplot(2000, dengueseq)
```



In this picture it is much more noticeable that the GC content is relatively high from around 4000-7000 bases, and lower on either side (from 2500-3500 and 7000-9000 bases).

Q2. *Draw a sliding window plot of GC content in the genome sequence for the bacterium Mycobacterium leprae strain TN (accession NC_002677) using a window size of 20000 nucleotides. Do you see any regions of unusual DNA content in the genome (eg. a high peak or low trough)?*

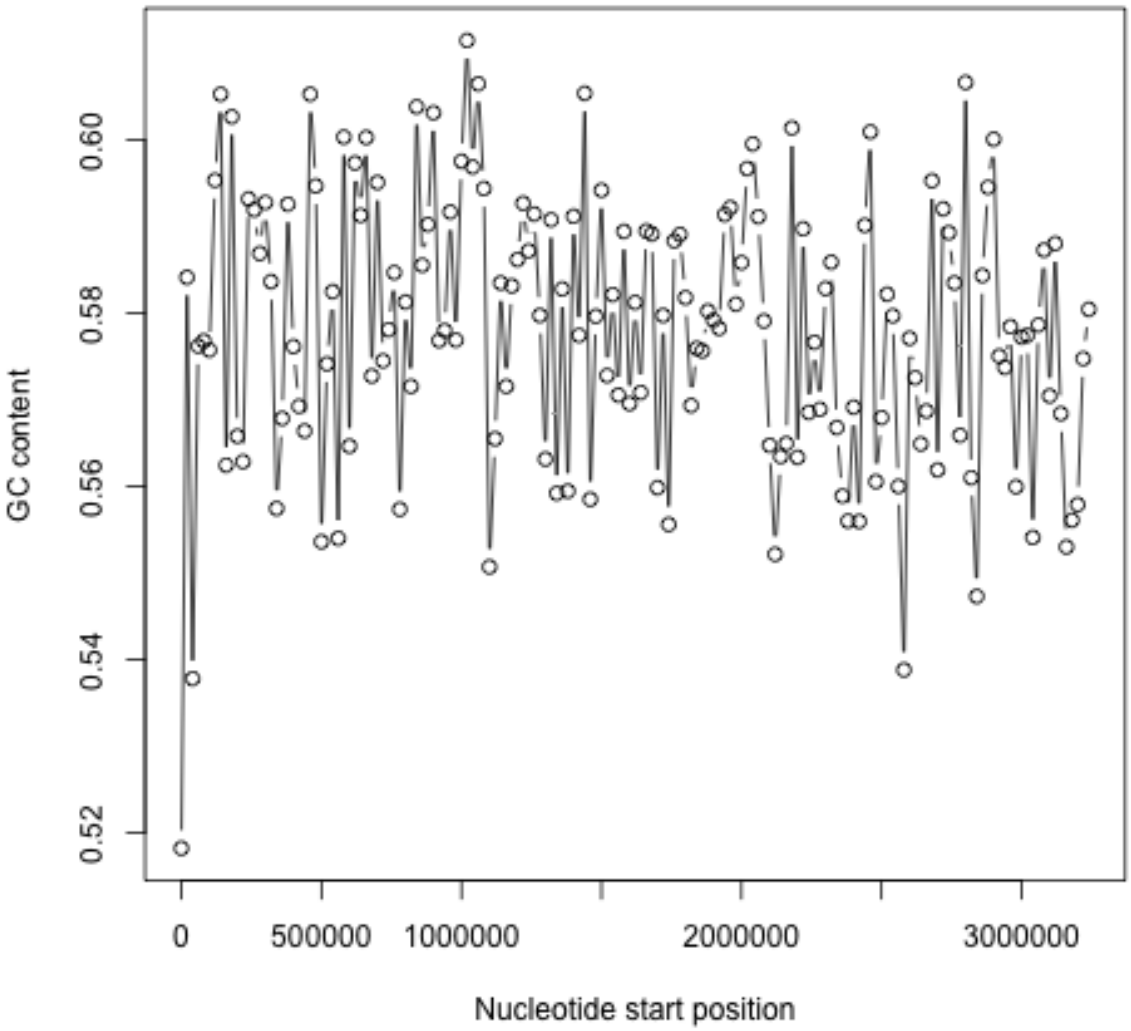
To do this, you first need to download the *Mycobacterium leprae* sequence from the NCBI website http://www.ncbi.nlm.nih.gov. To do this follow the steps in the chapter DNA Sequence Statistics.

Then read the sequence into R using the SeqinR library:

```
> leprae <- read.fasta(file = "leprae.fasta")
> lepraeseq <- leprae[[1]]
```

Then make a sliding window plot with a window size of 20000 nucleotides:

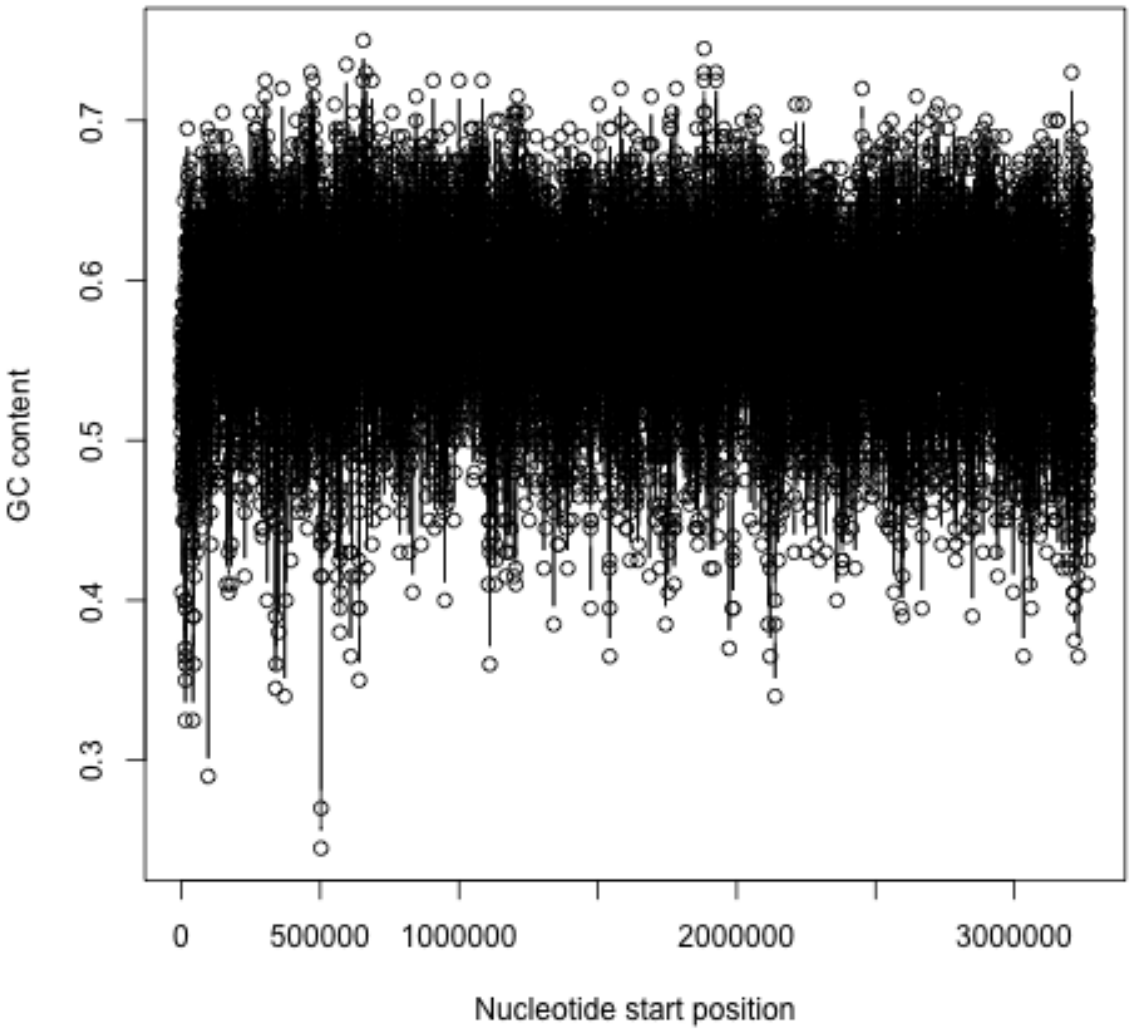
```
> slidingwindowplot(20000, lepraeseq)
```



We see the highest peak in GC content at about 1 Mb into the *M. leprae* genome. We also see troughs in GC content at about 1.1 Mb, and at about 2.6 Mb.

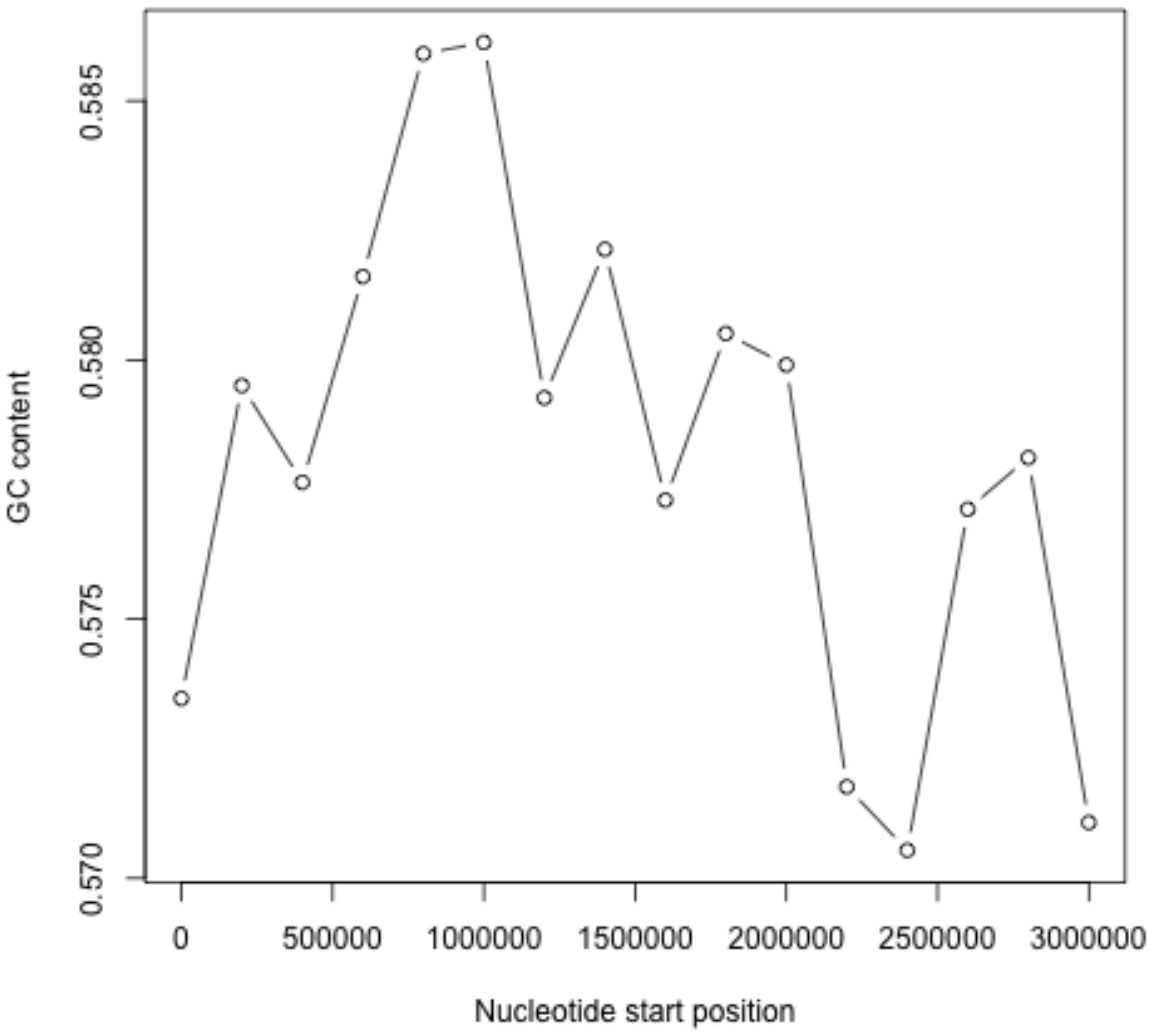
With a window size of 200 nucleotides, the plot is very messy, and we cannot see the peaks and troughs in GC content so easily:

```
> slidingwindowplot(200, lepraeseq)
```



With a window size of 200,000 nucleotides, the plot is very smooth, and we cannot see the peaks and troughs in GC content very easily:

```
> slidingwindowplot(200000, lepraeseq)
```



Q3. *Write a function to calculate the AT content of a DNA sequence (ie. the fraction of the nucleotides in the sequence that are As or Ts). What is the AT content of the Mycobacterium leprae TN genome?*

Here is a function to calculate the AT content of a genome sequence:

```
> AT <- function(inputseq)
  {
     mytable <- count(inputseq, 1) # make a table with the count of As, Cs, Ts, and Gs
     mylength <- length(inputseq) # find the length of the whole sequence
     myAs <- mytable[[1]] # number of As in the sequence
     myTs <- mytable[[4]] # number of Ts in the sequence
     myAT <- (myAs + myTs)/mylength
     return(myAT)
  }
```

We can then use the function to calculate the AT content of the *M. leprae* genome:

```
> AT(lepraeseq)
[1] 0.4220325
```

You should notice that the AT content is (1 minus GC content), ie. (AT content + GC content = 1):

```
> GC(lepraeseq)
[1] 0.5779675
> 0.4220325 + 0.5779675
[1] 1
```

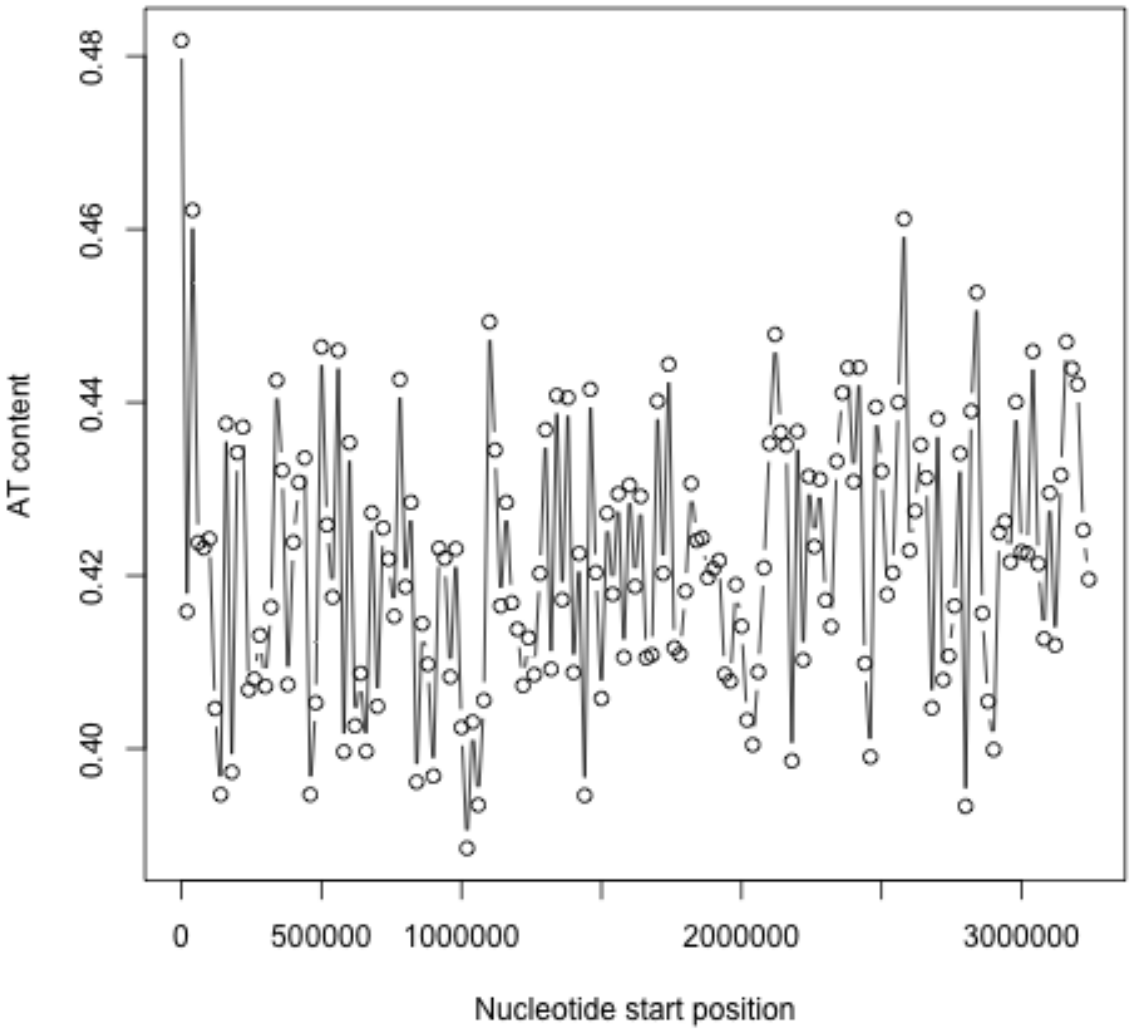
Q4. *Write a function to draw a sliding window plot of AT content. Use it to make a sliding window plot of AT content along the Mycobacterium leprae TN genome, using a windowsize of 20000 nucleotides. Do you notice any relationship between the sliding window plot of GC content along the Mycobacterium leprae genome, and the sliding window plot of AT content?*

We can write a function to write a sliding window plot of AT content:

```
> slidingwindowplotAT <- function(windowsize, inputseq)
  {
     starts <- seq(1, length(inputseq)-windowsize, by = windowsize)
     n <- length(starts)
     chunkATs <- numeric(n)
     for (i in 1:n) {
        chunk <- inputseq[starts[i]:(starts[i]+windowsize-1)]
        chunkAT <- AT(chunk)
        chunkATs[i] <- chunkAT
     }
     plot(starts,chunkATs,type="b",xlab="Nucleotide start position",ylab="AT content")
 }
```

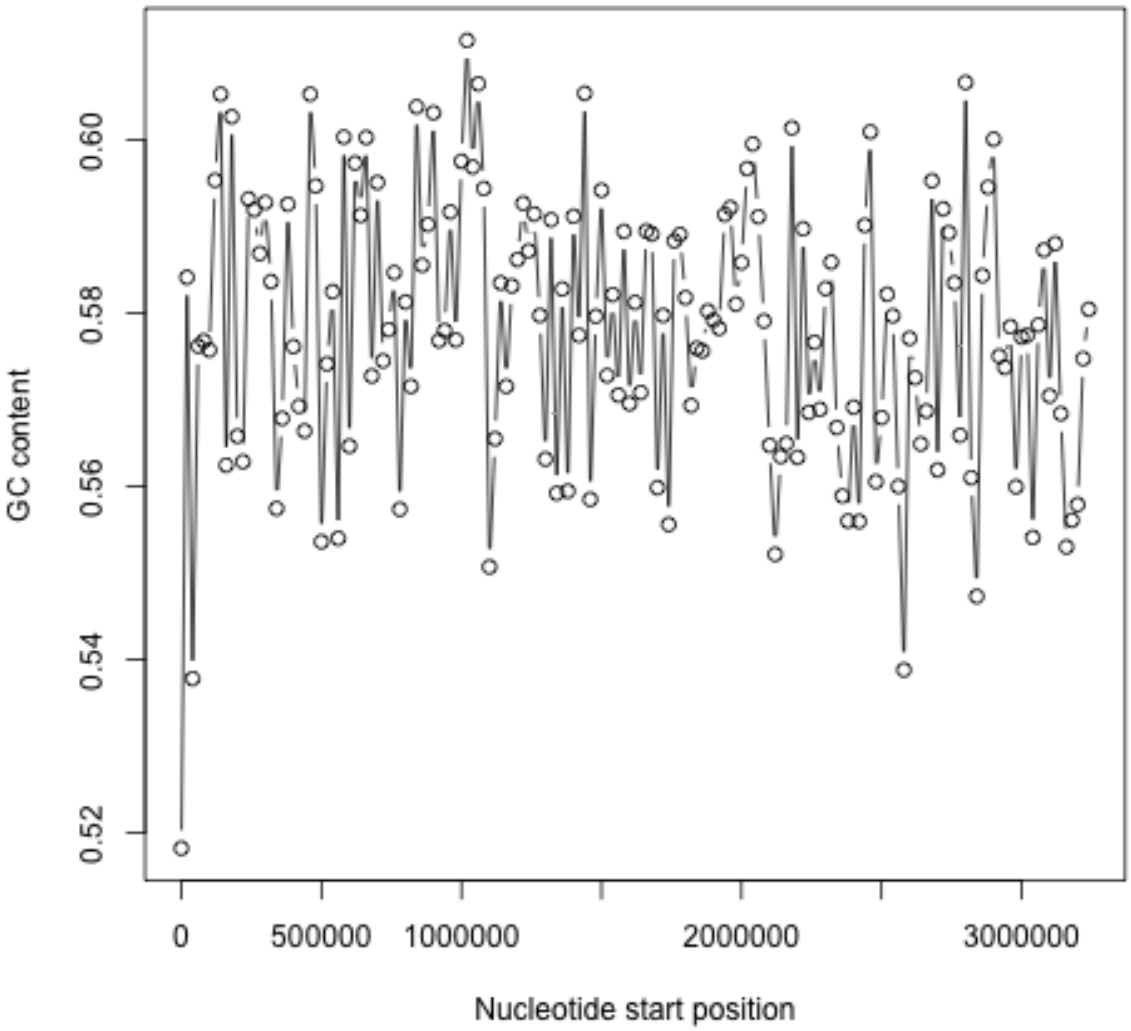
We can then use this function to make a sliding window plot with a window size of 20000 nucleotides:

```
> slidingwindowplotAT(20000, lepraeseq)
```



This is the mirror image of the plot of GC content (because AT equals 1 minus GC):

```
> slidingwindowplot(20000, lepraeseq)
```



Q5. *Is the 3-nucleotide word GAC GC over-represented or under-represented in the Mycobacterium leprae TN genome sequence?*

We can get the number of counts of each of the 3-nucleotide words by typing:

```
> count(lepraeseq, 3)
   aaa   aac   aag   aat   aca   acc   acg   act   aga   agc   agg   agt   ata   atc   atg
 32093 48714 36319 32592 44777 67449 57326 37409 31957 62473 38946 37470 25030 57245 44268
   att   caa   cac   cag   cat   cca   ccc   ccg   cct   cga   cgc   cgg   cgt   cta   ctc
 32973 52381 64102 64345 43838 64869 46037 87560 38504 78120 82057 89358 57451 29004 39954
   ctg   ctt   gaa   gac   gag   gat   gca   gcc   gcg   gct   gga   ggc   ggg   ggt   gta
 64730 36401 43486 61174 40728 58009 66775 80319 83415 62752 44002 81461 47651 69957 33139
   gtc   gtg   gtt   taa   tac   tag   tat   tca   tcc   tcg   tct   tga   tgc   tgg   tgt
 60958 65955 50421 21758 32971 29454 25076 48245 43166 78685 31424 49318 67270 67116 45595
   tta   ttc   ttg   ttt
 22086 43363 54346 32374
```

There are 61,174 GACs in the sequence.

The total number of 3-nucleotide words is calculated by typing:

```
> sum(count(lepraeseq,3))
[1] 3268201
```

Therefore, the observed frequency of GAC is 61174/3268201 = 0.01871794.

To calculate the expected frequency of GAC, first we need to get the number of counts of 1-nucleotide words by typing:

```
> count(lepraeseq, 1)
    a      c      g      t
 687041 938713 950202 692247
```

The sequence length is 3268203 bp. The frequency of G is 950202/3268203 = 0.2907414. The frequency of A is 687041/3268203 = 0.2102198. The frequency of C is 938713/3268203 = 0.2872260. The expected frequency of GAC is therefore 0.2907414*0.2102198*0.2872260 = 0.01755514.

The value of Rho is therefore the observed frequency/expected frequency = 0.01871794/0.01755514 = 1.066237. That, is there are about 1.1 times as many GACs as expected. This means that GAC is slightly over-represented in this sequence. The difference from 1 is so little however that it might not be statistically significant.

We can search for a function to calculate rho by typing:

```
> help.search("rho")
  base::getHook                          Functions to Get and Set Hooks for Load, Attach, Detach a
  seqinr::rho                            Statistical over- and under- representation of dinucleoti
  stats::cor.test                        Test for Association/Correlation Between Paired Samples
  survival::pbc                          Mayo Clinic Primary Biliary Cirrhosis Dat
```

There is a function rho in the SeqinR library. For example, we can use it to calculate Rho for words of length 3 in the *M. leprae* genome by typing:

```
> rho(lepraeseq, wordsize=3)
       aaa       aac       aag       aat       aca       acc       acg       act       aga
1.0570138 1.1742862 0.8649101 1.0653761 1.0793820 1.1899960 0.9991680 0.8949893 0.7610323
       agc       agg       agt       ata       atc       atg       att       caa       cac
1.0888781 0.6706048 0.8856096 0.8181874 1.3695545 1.0462815 1.0697245 1.2626819 1.1309452
       cag       cat       cca       ccc       ccg       cct       cga       cgc       cgg
1.1215062 1.0487995 1.1444773 0.5944657 1.1169725 0.6742135 1.3615987 1.0467726 1.1261261
       cgt       cta       ctc       ctg       ctt       gaa       gac       gag       gat
0.9938162 0.6939044 0.6996033 1.1197319 0.8643241 1.0355868 1.0662370 0.7012887 1.3710523
       gca       gcc       gcg       gct       gga       ggc       ggg       ggt       gta
1.1638601 1.0246015 1.0512300 1.0855155 0.7576632 1.0266049 0.5932565 1.1955191 0.7832457
       gtc       gtg       gtt       taa       tac       tag       tat       tca       tcc
1.0544820 1.1271276 1.1827465 0.7112314 0.7888126 0.6961501 0.8135266 1.1542345 0.7558461
       tcg       tct       tga       tgc       tgg       tgt       tta       ttc       ttg
1.3611325 0.7461477 1.1656391 1.1636701 1.1469683 1.0695410 0.7165237 1.0296334 1.2748168
       ttt
1.0423929
```

The Rho value for GAC is given as 1.0662370, in agreement with our calculation above.

6.1 Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

6.2 License

CHAPTER

SEVEN

SEQUENCE DATABASES

7.1 The NCBI Sequence Database

All published genome sequences are available over the internet, as it is a requirement of every scientific journal that any published DNA or RNA or protein sequence must be deposited in a public database. The main resources for storing and distributing sequence data are three large databases: the NCBI database (www.ncbi.nlm.nih.gov/), the European Molecular Biology Laboratory (EMBL) database (www.ebi.ac.uk/embl/, and the DNA Database of Japan (DDBJ) database (www.ddbj.nig.ac.jp/). These databases collect all publicly available DNA, RNA and protein sequence data and make it available for free. They exchange data nightly, so contain essentially the same data.

In this chapter we will discuss the NCBI database. Note however that it contains essentially the same data as in the EMBL/DDBJ databases. The NCBI database contains several sub-databases, including the NCBI Nucleotide database and the NCBI Protein database. These are both sequence databases. The NCBI database also contains sub-databases that contain non-sequence data, such as PubMed, which contains data on scientific publications.

Sequences in the NCBI Sequence Database (or EMBL/DDBJ) are identified by an accession number. This is a unique number that is only associated with one sequence. For example, the accession number NC_001477 is for the DEN-1 Dengue virus genome sequence. The accession number is what identifies the sequence. It is reported in scientific papers describing that sequence.

As well as the sequence itself, for each sequence the NCBI database (or EMBL/DDBJ databases) also stores some additional *annotation* data, such as the name of the species it comes from, references to publications describing that sequence, etc. Some of this annotation data was added by the person who sequenced a sequence and submitted it to the NCBI database, while some may have been added later by a human curator working for NCBI.

7.2 Searching for an accession number in the NCBI database

In the DNA Sequence Statistics chapter, you learnt how to obtain a FASTA file containing the DNA sequence corresponding to a particular accession number, eg. accession number NC_001477 (the DEN-1 Dengue virus genome sequence).

Briefly, if you want to obtain a FASTA file containing the DNA or protein sequence corresponding to a particular NCBI accession, follow these steps:

1. Go to the NCBI website (www.ncbi.nlm.nih.gov)
2. Type the accession number in the search box on the top of the NCBI homepage, under 'Search All Databases for', and press 'Search'. This searches the NCBI sequence databases (nucleotide and protein sequence databases) for the accession number of interest.
3. The results from your search will appear in a page saying how many hits you found in each NCBI database. The number of hits to a particular database (eg. the NCBI Nucleotide database or the NCBI Protein database) appears to the left of the database name in a square box. If the accession that you searched for corresponds to a DNA sequence, you should get a hit to a nucleotide sequence record. Click on the word 'Nucleotide'

to go to that record. Similarly, if the accession corresponds to a protein sequence, you should get a hit to a protein sequence record, and you should click on the word 'Protein' to go to that record.

4. Once you are looking at the NCBI record for the sequence, to download the sequence, click on the 'Send' link at the top right of the page, and select 'File' under 'Choose Destination' in the box that pops up, and then choose 'FASTA' as the format that you want from the drop-down menu that appears. Then click 'Create File'. Choose a sensible name for the file, for example 'den1.fasta' for the DEN-1 Dengue virus genome sequence, and save the file in an easy-to-find place so that you will be able to find it again later (for example, make a folder in the 'My Documents' folder with your name (eg. folder 'JoeBloggs') and save it there).

As explained in the DNA Sequence Statistics chapter, the FASTA format is a file format commonly used to store sequence information. The first line starts with the character '>' followed by a name and/or description for the sequence. Subsequent lines contain the sequence itself.

```
>mysequence1
ACATGAGACAGACAGACCCCCAGAGACAGACCCCTAGACACAGAGAGAG
TATGCAGGACAGGGTTTTTGCCCAGGGTGGCAGTATG
```

A FASTA file can contain more than one sequence. If a FASTA file contains many sequences, then for each sequence it will have a header line starting with '>' followed by the sequence itself.

```
>mysequence1
ACATGAGACAGACAGACCCCCAGAGACAGACCCCTAGACACAGAGAGAG
TATGCAGGACAGGGTTTTTGCCCAGGGTGGCAGTATG
>mysequence2
AGGATTGAGGTATGGGTATGTTCCCGATTGAGTAGCCAGTATGAGCCAG
AGTTTTTTACAAGTATTTTTCCCAGTAGCCAGAGAGAGAGTCACCCAGT
ACAGAGAGC
```

7.3 NCBI Sequence Format (NCBI Format)

As mentioned above, for each sequence the NCBI database stores some extra information such as the species that it came from, publications describing the sequence, etc. This information is stored in the NCBI entry or NCBI record for the sequence. The NCBI entry for a sequence can be viewed by searching the NCBI database for the accession number for that sequence. The NCBI entries for sequences are stored in a particular format, known as NCBI format.

To view the NCBI entry for the DEN-1 Dengue virus (which has accession NC_001477), follow these steps:

1. Go to the NCBI website (www.ncbi.nlm.nih.gov).
2. Search for the accession number.
3. On the results page, if your sequence corresponds to a nucleotide (DNA or RNA) sequence, you should see a hit in the Nucleotide database, and you should click on the word 'Nucleotide' to view the NCBI entry for the hit. Likewise, if your sequence corresponds to a protein sequence, you should see a hit in the Protein database, and you should click on the word 'Protein' to view the NCBI entry for the hit.
4. After you click on 'Nucleotide' or 'Protein' in the previous step, the NCBI entry for the accession will appear.

The NCBI entry for an accession contains a lot of information about the sequence, such as papers describing it, features in the sequence, etc. The 'DEFINITION' field gives a short description for the sequence. The 'ORGANISM' field in the NCBI entry identifies the species that the sequence came from. The 'REFERENCE' field contains scientific publications describing the sequence. The 'FEATURES' field contains information about the location of features of interest inside the sequence, such as regulatory sequences or genes that lie inside the sequence. The 'ORIGIN' field gives the sequence itself.

7.4 RefSeq

When carrying out searches of the NCBI database, it is important to bear in mind that the database may contain redundant sequences for the same gene that were sequenced by different laboratories. Some of these sequences may be better quality than others.

There are also many different types of nucleotide sequences and protein sequences in the NCBI database. With respect to nucleotide sequences, some many be entire genomic DNA sequences, some may be mRNAs, and some may be lower quality sequences such as expressed sequence tags (ESTs, which are derived from parts of mRNAs), or DNA sequences of contigs from genome projects. Furthermore, some sequences may be manually curated so that the associated entries contain extra information, but the majority of sequences are uncurated.

As mentioned above, the NCBI database often contains redundant information for a gene (because many different labs have sequenced the gene, and submitted their sequences to the NCBI database), some of which may be low quality. As a result, NCBI has made a special database called RefSeq (reference sequence database), which is a subset of the NCBI database. The data in RefSeq is manually curated, is high quality sequence data, and is non-redundant; this means that each gene (or splice-form of a gene, in the case of eukaryotes), protein, or genome sequence is only represented once.

The data in RefSeq is of much higher quality than the rest of the NCBI Sequence Database. However, unfortunately, because of the high level of manual curation required, RefSeq does not cover all species, and is not comprehensive for the species that are covered so far.

You can easily tell that a sequence comes from RefSeq because its accession number starts with particular sequence of letters. That is, RefSeq sequences corresponding to protein records usually start with 'NP_', and RefSeq curated complete genome sequences usually start with 'NC_' or 'NS_'.

7.5 Querying the NCBI Database

As a bioinformatician you may need to interrogate the NCBI Database to find particular sequences or a set of sequences matching given criteria, such as:

- All human nucleotide sequences associated with malaria
- The sequence published in *Nature* **460**:352-358
- All sequences from *Chlamydia trachomatis*
- Sequences submitted by Matthew Berriman
- Flagellin or fibrinogen sequences
- The glutamine synthetase gene from *Mycobacteriuma leprae*
- The upstream control region of the *Mycobacterium leprae dnaA* gene
- The sequence of the *Mycobacterium leprae* DnaA protein
- The genome sequence of *Trypanosoma cruzi*

Say for example that you want to find all high-quality nucleotide sequences associated with malaria. Firstly, to find all nucleotide sequences associated with malaria, follow these steps:

1. Go to the NCBI website (www.ncbi.nlm.nih.gov).
2. As you want to search for nucleotide sequences, select 'Nucleotide' from the drop-down list above the search box at the top of the NCBI homepage.
3. Type **malaria** in the search box. (Note that if you are searching for a phrase such as 'colon cancer', you would need to include the inverted commas, ie. type **"colon cancer"** and not **colon cancer**. This is because if you type just **colon cancer**, the search will be for records that contain the words 'colon' or 'cancer' (not necessarily both words), while you want records that contain the phrase 'colon cancer'.) Press 'Search'.

4. The search results will include all nucleotide sequences for which the phrase 'malaria' appears somewhere in their NCBI records. The phrase may appear in the 'DEFINITION' field of the NCBI record (which gives a short description), in the title of a journal article on the nucleotide sequence, or elsewhere in the NCBI record.

The search above should have identified thousands of sequences from many different species. Some of these may be of low quality. To limit your search to high quality sequences, you may decide to restrict your search to RefSeq sequences. You can do this using NCBI search tags. NCBI search tags allow you to limit your restrict your search to a specific data set, such as the RefSeq data set. It also allows us to limit searches to retrieve records with certain attributes, such as molecule type (eg. mRNAs) or species.

The NCBI search tag "[PROP]" allows you to restrict your search to sequences form a particular subset of the NCBI Sequence Database, such as RefSeq. To use NCBI search tags to restrict your search to nucleotide sequences from RefSeq that are associated with malaria, follow these steps:

1. Go to the NCBI website, and select 'Nucleotide' from the drop-down list above the search box.
2. In the search box, type **malaria AND srcdb_refseq[PROP]**, and press 'Search'.

This should give you all RefSeq nucleotide sequences for which the phrase malaria appears somehwere in the NCBI record.

Note that you should find fewer sequences than when you just searched for **malaria**, but these should be higher quality sequences (since they are RefSeq sequences), and their NCBI entries will contain manually curated information about the sequences (eg. details of publications about the sequences and features in them).

The search above should have identified RefSeq sequences from several species (eg. malaria itself, human, mouse, etc.) that are associated with malaria (or more precisely, where the word 'malaria' appears somewhere in the NCBI records). What if you are only interested in human sequences associated with malaria?

One way to solve this problem is to use NCBI search tags to restrict your search to human sequences. The "[ORGN]" search tag allows you to restrict your search to sequences from a particular species (eg. *Mycobacteriuma leprae*, the bacterium that causes leprosy, or set of species (eg. Bacteria). To use NCBI search tags to retrieve human RefSeq sequences associated with malaria, follow these steps:

1. Go to the NCBI website, and select 'Nucleotide' from the drop-down list above the search box.
2. In the search box, type **malaria AND srcdb_refseq[PROP] AND "Homo sapiens"[ORGN]**, and press 'Search'.

This will give you a list of all human nucleotide sequences from RefSeq that are associated with malaria (or more precisely, all the human nucleotide sequences from Refseq for which the word 'malaria' appears somewhere in the NCBI record).

In the searches above you used the "[PROP]" and "[ORGN]" NCBI sequence tags to retrict your search to a specific subset of the NCBI Sequence Database, or to sequences from a particular taxon, respectively. Other useful NCBI sequence tags are:

- "[JOUR]": to restrict your search to sequences described in a paper published in a particular journal
- "[VOL]": to restrict your search to sequences described in a paper published in a particular volume of a journal
- "[PAGE]": to restrict your search to sequences described in a paper with a particular start-page in a journal
- "[AU]": to restrict your search to sequences submitted to the NCBI Database by a particular person, or described in a journal paper by a particular person. The person's name should be in the form: surname first-initial (eg. Bloggs J[AU])
- "[ORGN]": to restrict your search to sequences from a particular species or taxon (eg. *Mycobacterium leprae* or *Mycobacterium* or Bacteria or Archaea)
- "[PROP]": to restrict your search to a particular subset of the NCBI database (eg. "srcdb_refseq[PROP]" restricts your search to RefSeq) or to a particular type of molecule (eg. "biomol mrna[PROP]" restrict your search to mRNA sequences).

7.6 Finding the genome sequence for a particular species

Microbial genomes are generally smaller than eukaryotic genomes (*Escherichia coli* has about 5 million base pair in its genome, while the human genome is about 3 billion base pairs). Because they are considerably less expensive to sequence, many microbial genome sequencing projects have been completed.

If you don't know the accession number for a genome sequence (eg. for *Mycobacterium leprae*, the bacterium that causes leprosy), how can you find it out? One way to do this is to look at the NCBI Genome website, which lists all fully sequenced genomes and gives the accession numbers for the corresponding DNA sequences.

If you didn't know the accession number for the *Mycobacterium leprae* genome, you could find it on the NCBI Genome website by following these steps:

1. Go to the NCBI Genome website (http://www.ncbi.nlm.nih.gov/sites/entrez?db=Genome)
2. On the homepage of the NCBI Genome website, it gives links to the major subdivisions of the Genome database, which include Eukaryota, Prokaryota (Bacteria and Archaea), and Viruses. Click on 'Prokaryota', since *Mycobacterium leprae* is a bacterium. This will bring up a list of all fully sequenced bacterial genomes, with the corresponding accession numbers. Note that more than one genome (from various strains) may have been sequenced for a particular species.
3. Use 'Find' in the 'Edit' menu of your web browser to search for 'Mycobacterium leprae' on the webpage. You should find that the genomes of several different *M. leprae* strains have been sequenced. One of these is *M. leprae* TN, which has accession number NC_002677.

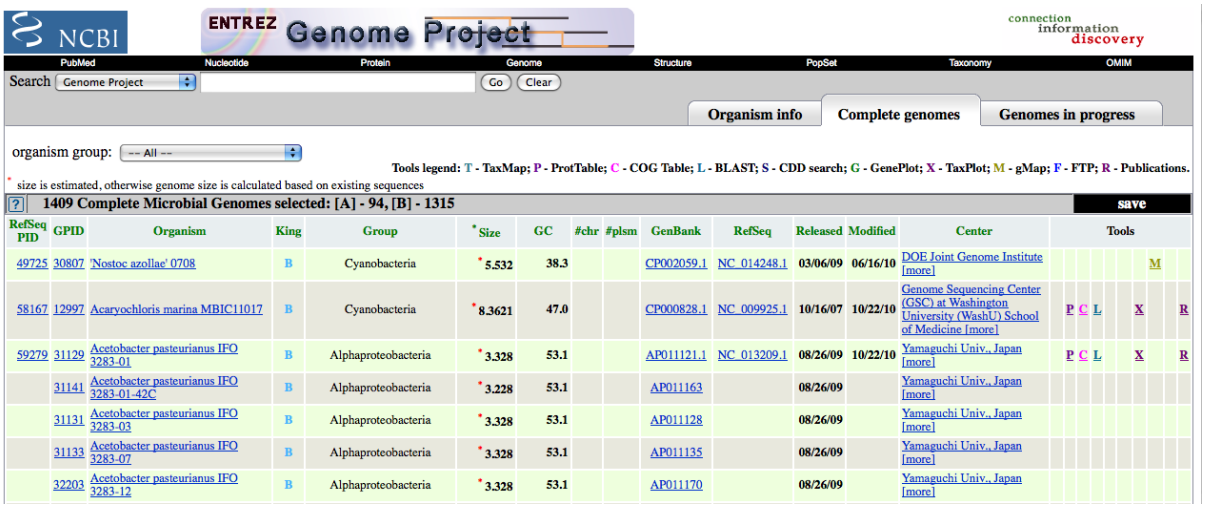
The list of sequenced genomes on the NCBI Genomes website is not a definitive list; that is, some sequenced genomes may be missing from this list. If you want to find out whether a particular genome has been sequenced, but you don't find it NCBI Genomes website's list, you should search for it by following these steps:

1. Go to the NCBI website (www.ncbi.nlm.nih.gov).
2. Select 'Genome' from the drop-down list above the search box.
3. Type the name of the species you are interested in in the search box (eg. **"Mycobacterium leprae"[ORGN]**). Press 'Search'.

Note that you could also have found the *Mycobacterium leprae* genome sequence by searching the NCBI Nucleotide database, as the NCBI Genome database is just a subset of the NCBI Nucleotide database.

7.7 How many genomes have been sequenced, or are being sequenced now?

On the NCBI Genome website (http://www.ncbi.nlm.nih.gov/sites/entrez?db=Genome), the front page gives a link to a list of all sequenced genomes in the groups Eukaryota, Prokaryota (Bacteria and Archaea) and Viruses. If you click on one of these links (eg. Prokaryota), at the top of the page it will give the number of sequenced genomes in that group (eg. number of sequenced prokaryotic genomes). For example, in this screenshot (from January 2011), we see that there were 1409 complete prokaryotic genomes (94 archaeal, 1315 bacterial):



NCBI ENTREZ Genome Project connection information discovery

PubMed Nucleotide Protein Genome Structure PopSet Taxonomy OMIM

Search Genome Project Go Clear

Organism info | Complete genomes | Genomes in progress

organism group: -- All --

Tools legend: T - TaxMap; P - ProtTable; C - COG Table; L - BLAST; S - CDD search; G - GenePlot; X - TaxPlot; M - gMap; F - FTP; R - Publications.

* size is estimated, otherwise genome size is calculated based on existing sequences

1409 Complete Microbial Genomes selected: [A] - 94, [B] - 1315 save

RefSeq PID	GPID	Organism	King	Group	*Size	GC	#chr	#plsm	GenBank	RefSeq	Released	Modified	Center	Tools
49725	30807	'Nostoc azollae' 0708	B	Cyanobacteria	*5.532	38.3			CP002059.1	NC_014248.1	03/06/09	06/16/10	DOE Joint Genome Institute [more]	M
58167	12997	Acaryochloris marina MBIC11017	B	Cyanobacteria	*8.3621	47.0			CP000828.1	NC_009925.1	10/16/07	10/22/10	Genome Sequencing Center (GSC) at Washington University (WashU) School of Medicine [more]	P C L X R
59279	31129	Acetobacter pasteurianus IFO 3283-01	B	Alphaproteobacteria	*3.328	53.1			AP011121.1	NC_013209.1	08/26/09	10/22/10	Yamaguchi Univ., Japan [more]	P C L X R
	31141	Acetobacter pasteurianus IFO 3283-01-42C	B	Alphaproteobacteria	*3.228	53.1			AP011163		08/26/09		Yamaguchi Univ., Japan [more]	
	31131	Acetobacter pasteurianus IFO 3283-03	B	Alphaproteobacteria	*3.328	53.1			AP011128		08/26/09		Yamaguchi Univ., Japan [more]	
	31133	Acetobacter pasteurianus IFO 3283-07	B	Alphaproteobacteria	*3.328	53.1			AP011135		08/26/09		Yamaguchi Univ., Japan [more]	
	32203	Acetobacter pasteurianus IFO 3283-12	B	Alphaproteobacteria	*3.328	53.1			AP011170		08/26/09		Yamaguchi Univ., Japan [more]	

Another useful website that lists genome sequencing projects is the Genomes OnLine Database (GOLD), which lists genomes that have been completely sequenced, or are currently being sequenced. To find the number of complete or ongoing bacterial sequencing projects, follow these steps:

1. Go to the GOLD website (http://genomesonline.org/).
2. Click on the yellow 'Enter GOLD' button in the centre of the webpage. On the subsequent page, it will give the number of ongoing bacterial, archaeal and eukaryotic genome sequencing projects.
3. Click on the 'Bacterial Ongoing' link to see the list of ongoing bacterial genome sequencing projects. By default, just the first 100 projects are listed, and the rest are listed on subsequent pages. In one of the columns of the page, this gives the university or institute that the genome was sequenced in. Other columns give the taxonomic information for the organism, and links to the sequence data.
4. Find the number of published genome sequencing projects. Go back one page, to the page with the 'Bacterial Ongoing' link. You will see that this page also lists the number of complete published genomes. To see a list of these genomes, click on 'Complete Published'. This will bring up a page that gives the number of published genomes at the top of the page. In one column of the page, this gives the university or institute that the genome was sequenced in.

As explained above, it is possible to identify genome sequence data in the NCBI Genome database. The GOLD database also gives some information about ongoing genome projects. Often, the GOLD database lists some ongoing projects that are not yet present in the NCBI Genome Database, because the sequence data has not yet been submitted to the NCBI Database. If you are interested in finding out how many genomes have been sequenced or are currently being sequenced for a particular species (eg. *Mycobacterium leprae*), it is a good idea to look at both the NCBI Genome database and at GOLD.

7.8 Summary

In this chapter, you have learnt how to retrieve sequences from the NCBI Sequence database, as well as to find out how many genomes have been sequenced or are currently being sequenced for a particular species.

7.9 Links and Further Reading

There is detailed information on how to search the NCBI database on the NCBI Help website at http://www.ncbi.nlm.nih.gov/bookshelf/br.fcgi?book=helpentrez?part=EntrezHelp.

There is more information about the GOLD database in the paper describing GOLD by Liolios *et al*, which is available at http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2808860/?tool=pubmed.

7.10 Acknowledgements

Thank you to Noel O'Boyle for helping in using Sphinx, http://sphinx.pocoo.org, to create this document, and github, https://github.com/, to store different versions of the document as I was writing it, and readthedocs, http://readthedocs.org/, to build and distribute this document.

Thank you to Andrew Lloyd and David Lynn, who generously shared their practical on sequence databases with me, which inspired many of the examples in this practical.

7.11 Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

7.12 License

7.13 Exercises

Answer the following questions. For each question, please record your answer, and what you did/typed to get this answer.

Q1. What information about the DEN-1 Dengue virus sequence (NCBI accession NC_001477) can you obtain from its annot What does it say in the DEFINITION and ORGANISM fields of its NCBI record?

Q2. What were the nucleotide sequences published in *Nature* volume 460, page 352? What are their accession numbers in the NCBI Sequence Database?

Q3. How many nucleotide sequences are there from the bacterium *Chlamydia trachomatis* in the NCBI Sequence Database? Remember to type **"Chlamydia trachomatis"** including the inverted commas.

Q4. How many nucleotide sequences are there from the bacterium *Chlamydia trachomatis* in the *RefSeq* part of the NCBI Sequence Database?

Q5. How many nucleotide sequences were submitted to NCBI by Matthew Berriman? Note that the name of the person who submitted a sequence is stored in the author field of the NCBI record, as is the name of people who published papers on the sequence. There may be more than one author fields in the NCBI record for a sequence, corresponding to the person who submitted the sequence and/or people who published papers on the sequence.

Q6. How many nucleotide sequences from nematode worms are there in the NCBI Database?

Q7. How many nucleotide sequences for collagen genes from nematode worms are there in the NCBI Database? Hint: look at the examples above for malaria-related genes.

Q8. How many *mRNA sequences* for collagen genes from nematode worms are there in the NCBI Database? Hint: look at the notes about the "[PROP]" search tag above.

Q9. How many *protein sequences* for collagen proteins from nematode worms are there in the NCBI database?

Q10. What is the accession number for the *Trypanosoma cruzi* genome in NCBI? Do you see genome sequences for more than one strain of *Trypanosoma cruzi*?

Q11. How many fully sequenced nematode worm species are represented in the NCBI Genome database?

xxx

Q12. How many ongoing genome sequencing projects are there for Bacteria, Archaea, and Eukarotes, respectively, in the G
Hint: Use the 'Find' option in the 'Edit' menu of your web browser to search for 'Cork' in the GOLD database's webpage listing ongoing genome sequencing projects.

Q14. How many genome sequences are there for *Lactobacillus salivarius* in the NCBI Genomes database?
Why are there more than one?

Q15. How many complete or ongoing genome sequencing projects for *Lactobacillus salivarius* are listed in GOLD?
Does GOLD or NCBI Genomes have more sequencing projects for this species? If not, can you suggest an explanation why?

CHAPTER

EIGHT

ANSWERS TO THE EXERCISES ON SEQUENCE DATABASES

Q1. *What information about the DEN-1 Dengue virus sequence (NCBI accession NC_001477) can you obtain from its annotations in the NCBI Sequence Database?*

To do this, you need to go to the www.ncbi.nlm.nih.gov website and type the DEN-1 Dengue virus genome sequence accession (NC_001477) in the search box, and press 'Search'.

On the search results page, you should see '1' beside the word 'Nucleotide', meaning that there was one hit to a sequence record in the NCBI Nucleotide database, which contains DNA and RNA sequences. If you click on the word 'Nucleotide', it will bring you to the sequence record, which should be the NCBI sequence record for the DEN-1 Dengue virus' genome (ie. for accession NC_001477).

On the webpage for the NCBI sequence record for the DEN-1 Dengue virus' genome (webpage http://www.ncbi.nlm.nih.gov/nuccore/NC_001477) , you will see in the DEFINITION, ORGANISM and REFERENCE fields of its NCBI record:

DEFINITION: Dengue virus type 1, complete genome. ORGANISM: Dengue virus 1 REFERENCE: There are several papers (the first is): AUTHORS: Puri,B., Nelson,W.M., Henchal,E.A., Hoke,C.H., Eckels,K.H., Dubois,D.R., Porter,K.R. and Hayes,C.G. TITLE: Molecular analysis of dengue virus attenuation after serial passage in primary dog kidney cells JOURNAL: J. Gen. Virol. 78 (PT 9), 2287-2291 (1997)

There are also some other references, for papers published about the Dengue virus genome sequence.

Q2. *What were the nucleotide sequences published in Nature volume 460, page 352?*

To do this, you need to go to the NCBI website at www.ncbi.nlm.nih.gov and type in the search box on the top: "Nature"[JOUR] AND 460[VOL] AND 352[PAGE]

Here [JOUR] specifies the journal name, [VOL] the volume of the journal the paper is in, and [PAGE] the page number.

This should bring up a results page with "50890" beside the word "Nucleotide", and "1" beside the word "Genome", and "25700" beside the word "Protein", indicating that there were 50890 hits to sequence records in the Nucleotide database, which contains DNA and RNA sequences, and 1 hit to the Genome database, which contains genome sequences, and 25700 hits to the Protein database, which contains protein sequences.

If you click on the word "Nucleotide", it will bring up a webpage with a list of links to the NCBI sequence records for those 50890 hits. The 50890 hits are all contigs from the schistosome worm *Schistosoma mansoni*.

Likewise, if you click on the word "Protein", it will bring up a webpage with a list of links to the NCBI sequence records for the 25700 hits, and you will see that the hits are all predicted proteins for *Schistosoma mansoni*.

If you click on the word "Genome", it will bring you to the NCBI record for the *Schistosoma mansoni* genome sequence, which has NCBI accession NS_00200. Note that the accession starts with "NS_", which indicates that it is a RefSeq accession.

Therefore, in *Nature* volume 460, page 352, the *Schistosoma mansoni* genome sequence was published, along with all the DNA sequence contigs that were sequenced for the genome project, and all the predicted proteins for

the gene predictions made in the genome sequence. You can view the original paper on the *Nature* website at http://www.nature.com/nature/journal/v460/n7253/abs/nature08160.html.

Q3. *How many nucleotide sequences are there from the bacterium Chlamydia trachomatis in the NCBI Sequence Database?*

To answer this, you need to go to www.ncbi.nlm.nih.gov, and select "Nucleotide" from the drop-down list at the top of the webpage, as you want to search for nucleotide (DNA or RNA) sequences.

Then in the search box, type "Chlamydia trachomatis"[ORGN] and press 'Search'.

Here [ORGN] specifies the organism you are interested in, that is, the species name in Latin.

The results page should give you a list of the hits to sequence records in the NCBI Nucleotide database. It will say "Found 35385 nucleotide sequences. Nucleotide (35237) GSS (148)". This means that 35,385 sequences were found, of which 35237 are DNA or RNA sequences, and 148 are DNA sequences from the Genome Sequence Surveys (GSS), that is, from genome sequencing projects [as of 19-Feb-2011]. Note that there are new sequences being added to the database continuously, so if you check this again in a couple of months, you will probably find a higher number of sequences (eg. 36,000 sequences).

Note: if you just go to the www.ncbi.nlm.nih.gov database, and search for "Chlamydia trachomatis"[ORGN] (without choosing "Nucleotide" from the drop-down list), you will see 35237 hits to the Nucleotide database and 148 to the GSS (Genome Sequence Survey) database.

Note also that if you search for "Chlamydia trachomatis", without using [ORGN] to specify the organism, you will get 51046 hits to the Nucleotide database and 149 to the GSS database, but some of these might not be *Chlamydia trachomatis* sequences – they could just be sequences for which the NCBI sequence record contains the phrase "Chlamydia trachomatis" somewhere.

Q4. *How many nucleotide sequences are there from the bacterium Chlamydia trachomatis in the RefSeq part of the NCBI Sequence Database?*

To answer this, you need to go to www.ncbi.nlm.nih.gov and select "Nucleotide" from the drop-down list at the top of the webpage, as you want to search for nucleotide sequences.

Then in the search box, type "Chlamydia trachomatis"[ORGN] AND srcdb_refseq[PROP] and press 'Search'.

Here [ORGN] specifies the organism, and [PROP] specifies a property of the sequences (in this case that they belong to the RefSeq subsection of the NCBI database).

At the top of the results page, it should say "Results: 1 to 20 of 29 sequences" [as of 19-Feb-2011]. As for Q3, if you try this again in a couple of months, the number will probably be higher, due to extra sequences added to the database.

Note that the sequences in Q3 are all *Chlamydia trachomatis* DNA and RNA sequences in the NCBI database. The sequences in Q4 gives the *Chlamydia trachomatis* DNA and RNA sequences in the RefSeq part of the NCBI database, which is a subsection of the database for high-quality manually-curated data.

The number of sequences in RefSeq is much fewer than the total number of *C. trachomatis* sequences, partly because low quality sequences are never added to RefSeq, but also because RefSeq curators have probably not had time to add all high-quality sequences to RefSeq (this is a time-consuming process, as the curators add additional information to the NCBI Sequence records in RefSeq, such as references to papers that discuss a particular sequence).

Q5. *How many nucleotide sequences were submitted to NCBI by Matthew Berriman?*

To answer this, you need to go to www.ncbi.nlm.nih.gov, and select "Nucleotide" from the drop-down list, as you want to search for nucleotide sequences.

Then in the search box, type "Berriman M"[AU] and press 'Search'.

Here [AU] specifies the name of the person who either submitted the sequence to the NCBI database, or wrote a paper describing the sequence.

On the results page, it should say at the top: "Found 460052 nucleotide sequences. Nucleotide (250328) EST (121075) GSS (88649)". This means that 460052 DNA/RNA sequences were either submitted to the NCBI database by someone called M. Berriman, or were described in a paper by someone called M. Berriman. Of

these, 250328 were DNA/RNA sequences, 121075 were EST sequences (part of mRNAs), and 88649 were DNA sequences from genome sequencing projects (GSS or Genome Sequence Survey sequences).

Note that unfortunately the NCBI website does not allow us to search for "Berriman Matthew"[AU] so we cannot be sure that all of these sequences were submitted by Matthew Berriman.

Q6. *How many nucleotide sequences from nematode worms are there in the NCBI Database?*

To answer this, you need to go to www.ncbi.nlm.nih.gov and select "Nucleotide" from the drop-down list, as you want to search for nucleotide sequences.

Then in the search box, type Nematoda[ORGN] and press 'Search'.

Here [ORGN] specifies the group of species that you want to search for sequences from. In Q4, [ORGN] was used to specify the name of one organism (*Chlamydia trachomatis*). However, you can also use [ORGN] to specify the name of a group of organisms, for example, Fungi[ORGN] would search for fungal sequences or Mammalia[ORGN] would search for mammalian sequences. The name of the group of species that you want to search for must be given in Latin, so to search for sequences from nematode worms we use the Latin name Nematoda.

The search page should say at the top 'Found 2202458 nucleotide sequences. Nucleotide (378255) EST (1140454) GSS (683749)' [as of 19-Feb-2011]. This means that 2,202,458 DNA or RNA sequences were found from nematode worm species in the database, of which 378,255 are DNA/RNA sequences, 1,140,454 are ESTs, and 683,749 sequences are DNA sequences from genome sequencing projects. These sequences are probably from a wide range of nematode worm species, including the model nematode worm *Caenorhabditis elegans*.

Q7. *How many nucleotide sequences for collagen genes from nematode worms are there in the NCBI Database?*

To answer this, you need to go to www.ncbi.nlm.nih.gov and select "Nucleotide" from the drop-down list, as you want to search for nucleotide sequences.

Then in the search box, type Nematoda[ORGN] AND collagen.

Here [ORGN] specifies that we want sequences from nematode worms. The phrase "AND collagen" means that the word collagen must appear somewhere in the NCBI entries for those sequences, for example, in the sequence name, or in a description of the sequence, or in the title of a paper describing the sequence, etc.

On the results page, you should see 'Found 8341 nucleotide sequences. Nucleotide (1546) EST (6795)' [as of 19-Feb-2011]. This means that 8341 DNA or RNA sequences for collagen genes from nematode worms were found, of which 6795 are EST sequences (parts of mRNAs). Note that these 8341 nucleotide sequences may not all necessarily be for collagen genes, as some of the NCBI records found may be for other genes but contain the word 'collagen' somewhere in the NCBI record (for example, in the title of a cited paper).

Q8. *How many mRNA sequences for collagen genes from nematode worms are there in the NCBI Database?*

To answer this, you need to go to www.ncbi.nlm.nih.gov, and select "Nucleotide" from the drop-down sequences, as you want to search for nucleotide sequences (nucleotide sequences include DNA sequences and RNA sequences, such as mRNAs).

Then in the search box, type Nematoda[ORGN] AND collagen AND "biomol mRNA"[PROP].

Here [ORGN] specifies the name of the group of species, collagen specifies that we want to find NCBI entries that include the word collagen, and [PROP] specifies a property of those sequences (that they are mRNAs, in this case).

The search page should say 'Found 7656 nucleotide sequences. Nucleotide (861) EST (6795)' [as of 19-Feb-2011]. This means that 7656 mRNA sequences were found that contain the word 'collagen' in the NCBI record. Of the 7656, 6795 are EST sequences (parts of mRNAs).

Note that in Q7 we found 8341 nucleotide (DNA or RNA) sequences from nematode worms. In this question, we found out that only 7656 of those sequences are mRNA sequences. This means that the other (8341-7656=) 685 sequences must be DNA sequences, or other types of RNA sequences (not mRNAs) such as tRNAs or rRNAs.

Q9. *How many protein sequences for collagen proteins from nematode worms are there in the NCBI database?*

To answer this, you need to go to www.ncbi.nlm.nih.gov, and select "Protein" from the drop-down list, as you want to search for protein sequences.

Then type in the search box: Nematoda[ORGN] AND collagen and press 'Search'.

On the results page, you should see '1 to 20 of 1886'. This means that 1886 protein sequences from nematode worms were found that include the word collagen in the NCBI sequence entries [as of 19-Feb-2011].

Q10. *What is the accession number for the Trypanosoma cruzi genome in NCBI?*

There are two ways that you can answer this.

The first method is to go to www.ncbi.nlm.nih.gov and select "Genome" from the drop-down list, as you want to search for genome sequences.

Then type in the search box: "Trypanosoma cruzi"[ORGN] and press 'Search'.

The results page says 'All:1', and lists just one NCBI record, the genome sequence for *Trypanosoma cruzi* strain CL Brener, which has accession NZ_AAHK00000000.

The second method of answering the question is to go to the NCBI Genomes webpage http://www.ncbi.nlm.nih.gov/sites/entrez?db=Genome.

Click on the 'Eukaryota' link at the middle the page, as *Trypanosoma cruzi* is a eukaryotic species.

This will give you a complete list of all the eukaryotic genomes that have been sequenced.

Go to the 'Edit' menu of your web browser, and choose 'Find', and search for 'Trypanosoma cruzi'.

You should find *Trypanosoma cruzi* strain CL Brener. You will also find that there are several ongoing genome sequencing projects listed for other strains of *Trypanosoma cruzi*: strains JR cl. 4, Sylvio X10/1, Y, and Esmeraldo Esmeraldo cl. 3.

The link 'GB' (in green) at the far right of the webpage gives a link to the NCBI record for the sequence. In this case, the link for *Trypanosoma cruzi* strain CL Brener leads us to the NCBI record for accession AAHK01000000. This is actually an accession for the *T. cruzi* strain CL Brener sequencing project, rather than for the genome sequence itself. On the top right of the page, you will see a link "Genome", and if you click on it, it will bring you to the NCBI accession NZ_AAHK00000000, the genome sequence for *Trypanosoma cruzi* strain CL Brener.

Of the other *T. cruzi* strains listed, there is only a 'GB' link for one other strain, Sylvio X10/1. Presumably there are no links for the other *Trypanosoma cruzi* strains, because the sequencing projects are still in progress. If you click on the link for *Trypanosoma cruzi* strain Sylvio X10/1, it will bring you to the NCBI record for accession ADWP01000000, the accession for the *T. cruzi* strain Sylvio X10/1 sequencing project. At the top right of that page, there is no "Genome" link, which tells you that there is not yet a genome assembly available for this strain.

Note that the answer is slightly different for the answer from the first method above, which did not find the information on the genome projects for strains JR cl. 4, Sylvio X10/1, Y, and Esmeraldo Esmeraldo cl. 3, because genome assemblies are not yet available for those strains.

Q11. *How many fully sequenced nematode worm species are represented in the NCBI Genome database?*

To answer this question, you need to go to the NCBI Genome webpage http://www.ncbi.nlm.nih.gov/sites/entrez?db=Genome.

In the search box at the top of the page, type Nematoda[ORGN] to search for genome sequences from nematode worms, using the Latin name for the nematode worms.

On the results page, you will see 'Items 1 - 20 of 62', indicating that 62 genome sequences from nematode worms have been found. If you look down the page, you will see however that many of these are mitochondrial genome sequences, rather than chromosomal genome sequences.

If you are just interested in chromosomal genome sequences, you can type 'Nematoda[ORGN] NOT mitochondrion' in the search box, to search for non-mitochondrial sequences. This should give you 16 sequences, which are all chromosomal genome sequences for nematode worms, including the species *Caenorhabditis elegans*, *Caenorhabditis remanei*, *Caenorhabditis briggsae*, *Loa loa* (which causes subcutaneous filariasis), and *Brugia malayi* (which causes lymphatic filariasis). Thus, there are nematode genome sequences from five different species that have been fully sequenced (as of 19-Feb-2011). Because nematode worms are multi-chromosomal species, there may be several chromosomal sequences for each species.

Note that when you search the NCBI Genome database at http://www.ncbi.nlm.nih.gov/sites/entrez?db=Genome, you will find the NCBI records for completely sequenced genomes (completely sequenced nematode genomes in this case).

If you are interested in partially sequenced genomes, that is sequences from genome sequencing projects that are still in progress, you can go to the NCBI Genome Projects website at http://www.ncbi.nlm.nih.gov/genomeprj. If you search the NCBI Genome Projects database for Nematoda[ORGN], you will find that genome sequencing projects for many other nematode species are ongoing, including for the species *Onchocerca volvulus* (which causes onchocerciasis), *Wuchereria bancrofti* (which causes lymphatic filariasis), and *Necator americanus* (which causes soil-transmitted helminthiasis).

8.1 Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

8.2 License

CHAPTER

NINE

ACKNOWLEDGEMENTS

Thank you to Noel O'Boyle for helping in using Sphinx, http://sphinx.pocoo.org, to create this document, and github, https://github.com/, to store different versions of the document as I was writing it, and readthedocs, http://readthedocs.org/, to build and distribute this document.

CHAPTER

TEN

CONTACT

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address a.coghlan@ucc.ie

CHAPTER

ELEVEN

LICENSE