RNA-seq data analysis practical

Ângela Gonçalves, Nicolas Delhomme

September 8, 2012

Contents

1 Introduction

Note: This tutorial is partially based on materials developed by Patrick Aboyoun, Simon Anders and Martin Morgan, who we would like to thank and acknowledge.

This tutorial illustrates how to use R and Bioconductor for the analysis of data obtained from high-throughput sequencing (HTS) assays. Starting from raw files as they come from a sequencing machine, we will import the short read sequences and make diagnostic plots to assess the quality of the run.

Throughout this document you will find R commands that you should type or copy paste into the R command line. There are also exercises (some of them might be challenging) that you should try to solve (the solutions are provided at the back of this document). Whenever there is an output to your command please take some time to understand what it means.

This tutorial will be done in two steps. After the "Introduction to R and Bioconductor" lecture, the section 2 (page 4), 3 (page 7) and 4 (page 9) will be performed and after the lecture "Dealing with raw data: quality assessment and filtering", the remainder of the tutorial.

2 Warming up at the console

In the following, you will have to enter commands at the prompt in a console. The commands will be indicated as follow:

```
Here is the command to type in at the console prompt.
```

2.1 Servers

The practicals have been designed such that you can run them on your own laptop. If for any reason this would not work, as well as for some more challenging exercises, an access to a local powerful machine **hippu.csc.fi** and a cluster **vuori.csc.fi** have been set up. To access these, you need to open a terminal (easy on Mac and Linux). For windows, please see that link: http://the.earth.li/ sgtatham/putty/0.62/htmldoc/Chapter2.html#gs-insecure.

2.2 Users

There are in total 10 users called rnaseq1 to rnaseq10, meaning that every user will be shared by 4 of you. The following example use rnaseq as an example user, CHANGE IT to the one attributed to you when typing the commands!

2.3 Password

The password will be made available at the course.

2.4 Access

Now that you have the credentials, let's try to connect. On Mac and Linux, open a terminal and type in:

```
ssh rnaseq@vuori.csc.fi
```

On Windows, open `putty` and paste in the credentials.

2.5 Setting up

Every user has a directory called */wrk/rnaseq* (again use your user there...). Let's go to that directory:

```
cd /wrk/rnaseq
```

and list the content:

```
ls -l
```

Well it's empty so let's create something in there. As you are sharing your account with 3 other people, create your own directory, just to avoid conflicts :-) And do not call it like the one in the example below, that belongs to me!

```
mkdir delhomme
```

Did that work?

```
ls -l
```

Now, let's go into that directory

```
cd delhomme
```

And let's set up some work for later. We will need a large file, so rather than having everyone downloading it, I've made a copy there: */wrk/rnaseq11/delhomme*. This command you do not need to edit. Why?

```
cp /wrk/rnaseq11/delhomme/SRAmetadb.sqlite.gz .
```

Now as you can see from the file extension - .gz - is zipped (compressed), so let's decompress it.

```
gunzip SRAmetadb.sqlite.gz
```

Good, we are set for now.

2.6 The queueing system

As you can imagine, working on a cluster implies a number of rules. And you might wonder why you have to learn about using a cluster at all. The answer is that even with a great laptop, you probably won't be able to manipulate the amount of raw data generated by an RNA-Seq / ChIP-Seq study. Only once these have been pre-processed, that it becomes possible to perform analyses on a laptop.

Back to the cluster rules. To make sure the cluster is used fairly by every user, a queueing strategy is used. When you submit a job to the cluster, it will first be put on hold in a queue and will only get to run when the requested resources (CPU, RAM) are available. To check the status of the queue, simply type:

```
squeue
```

Later, when we will perform some exercise, we will use "interactive" jobs. To launch one, we will execute the following command:

```
salloc -n 2 --mem-per-cpu=1000 -t 03:00:00 -p interactive
```

It will require 2 CPU, 2GB of RAM for 3 hours. Of course, it might take a little time before we get the requested resources, which is why we start it now. It does not matter if we get the allocated resource earlier as 3 hours from now should be amply sufficient.

2.7 Disconnect

(Do NOT run it now), but it's as simple as:

```
logout
```

Now we are done with the console setup, let's move to R. You can either perform it on your own computer, or if the interactive job is already available, as well on the cluster. If you want to do so, you need to start R using the following command:

```
Rd
```

3 Warming up for R

Note: Credit for this section goes to Martin Morgan. This section is for those who are not really comfortable with R; others may skip it and go directly to the next section.

R is a computer programming language. It has some familiar - to programmers - data types and instructions, but it has many unique features too; in many ways it is easier to learn R as a non-programmer than as someone with experience in another programming language. Welcome!

A key concept in R is the variable. Variables are vector-valued and can be of different types.

```
> # define an integer vector, 1, ..., 10
> x <- 1:10
> # print the contents of the vector
> x

 [1]  1  2  3  4  5  6  7  8  9 10

> # logical vector
> const <- c(TRUE, TRUE, FALSE)
> # named numeric vector
> const <- c(pi=3.14159, e=2.718282)
> # list of heterogeneous types
> misc <- list(x=1:10, alph=c("a", "b", "c"))
```

Vectors can be subset by logical value, name, or integer index. An important distinction is between subsetting with a single square bracket, which returns an instance of the original object, versus element selection with a double square bracket, which returns a specific element.

```
> # select elements 4, 3 and 2 of vector x
> x[4:2]

[1] 4 3 2

> # a list of length 1, containing an element pi
> const["pi"]

     pi
3.14159

> # not a list; the element pi
> const[["pi"]]
```

```
[1] 3.14159
```

Vectors are a basic R type, but R itself and many additional packages define classes representing complicated data structures that the user is expected to manipulate.

```
> library(Biostrings)
> dna <- DNAString("AAACTCTAT")
> translate(dna)

  3-letter "AAString" instance
seq: KLY
```

The R help system is essential to discovering how to work with classes. The elements of this system include interactive use...

```
> class(dna)

[1] "DNAString"
attr(,"package")
[1] "Biostrings"
```

... coupled with 'man' pages accessible within R or via a web browser.

```
> # see the manual page for the DNAString class
> ?DNAString
> # or start the web-based help system
> help.start()
```

Bioconductor packages include vignettes that provide a rich textual description of how the package is supposed to be used.

```
> browseVignettes(package="ShortRead")
```

The package vignette page also lists .R files. These contain the script used in the vignette. Cut-and-paste from the script into an R session to speed through an exercise.

4 First application

Two libraries we will need for tomorrow's session have been overlooked in the installation script.

Exercise 1: Please install the libraries: *EatonEtAlChIPseq* and *leeBamViews*.

Have a look at that webpage for help: http://www.bioconductor.org/install/. The solution is at the end of this document in appendix A (page 19).

5 Getting the Data

First, we need to get the data. For that we will use the *SRAdb* package. This package lists the content of the Short Read Archive (SRA) and European Nucleotide Archive (ENA) and offers the possibility to search and download datasets. The information is stored in an SQL database that has to be downloaded locally. The database is rather large so to avoid downloading it, it is available from the *Practicals/BiocPractical* directory as a zipped archive: SRAmetadb.sqlite.gz. To proceed, you can either work on your computer or use the interactive session open on **vuori**. The examples below are meant for that last one; to use your laptop adapt the path to the file.

First, we connect to the database:

```
> library(SRAdb)
> sqlfile <- "/wrk/rnaseq11/Course/data/SRAmetadb.sqlite"
> sra_con <- dbConnect(SQLite(),sqlfile)
```

Then list the files we are interested in.

```
> acc <- c("SRR490224","SRR490225")
> getFASTQinfo( in_acc = acc, srcType = 'ftp' )
```

Then download them

```
> getSRAfile( in_acc=acc, sra_con, destDir = getwd(),
+             fileType = 'fastq', srcType = 'ftp' )
```

Clean up

```
> dbDisconnect( sra_con )
```

We have now downloaded in the current directory two files from the Jääger et al. [2012] study. They were selected just for their size (1MB) - not the relevance of the study. The study aim was the dynamics of differentiation of human dermis - and adipose - derived stromal stem cells.

```
> dir()
```

Note: If some of the previous steps (database connection, download, etc.) would not work, the data are available in the *Practicals/BiocPractical* directory.

6 Dealing with raw sequence data

The downloaded FASTQ files (the one with the .fastq extension) contains the sequences and qualities of the short reads produced for two of the samples in the Jääger et al. [2012] experiment. Open this file in the console to get an idea of how this looks like. The files are compressed, so you need to decompress before opening them. To make this easier, you can decompress them on the fly and "pipe" - *i.e.* transmit the output of the gunzip command as an input to the next command - it into the "head" command that lists only the top first lines of a file. The -12 argument here precise that we want to look at the first 12 lines.

```
gunzip -c SRR490224.fastq.gz | head -12
```

The FASTQ format is commonly used to store the nucleotide sequence of reads and their associated qualities.

Exercise 2: Each read is represented by 4 lines in the FASTQ file. The 3rd one is just a separator, can you tell what the other ones are?

6.1 Importing FASTQ files

Using R, we start by importing our FASTQ data file with the *ShortRead* package. This package provides useful functionality for input, quality assessment, and basic manipulation of "short read" DNA sequences. It provides flexible import functions of common short read data formats such as those produced by Solexa, 454, and related technologies.
Load the *ShortRead* library and read the file into a `ShortReadQ` object:

```
> library(ShortRead)
> # the first argument of the readFastq function
> # is the directory containing the data, and the
> # second a pattern describing the filename
> sr <- readFastq(".", pattern="SRR490224.fastq.gz")
```

Note that "." means the current directory. And that `readFastq` can read gzipped files; handy isn't it?

The function loads the raw data into an object of class *ShortReadQ*. This class provides a way to store and manipulate the reads, identifiers, and quality scores of uniform-length short reads. You can use the following functions (or accessors) to look into this object:

```
> # print the read sequences
> sread(sr)
> # print the base qualities
> quality(sr)
> # print the names of the reads
> id(sr)
> # print the run lengths of the first few reads
> head(width(sr))
> # print a summary of the object
> sr
```

6.2 Quality assessment (QA)

There are also several functions defined in the *ShortRead* package that help you assess the quality of the data. We can, for example, plot the mean base quality per cycle:

```
> # convert the encoded quality scores to a numeric value
> nqual <- as(quality(sr), "matrix")
> # find the mean quality for each cycle
> meanqual <- colMeans( nqual )
> # plot the mean quality per cycle
> plot( meanqual, xlab="Cycle", ylab="Average cycle quality" )
```

In a similar way we can use the function `alphabetByCycle` to plot the number of each base per cycle and check if the result is uniform across all cycles as expected:

```
> # count the number of bases of each kind per cycle
> abc <- alphabetByCycle(sread(sr))
> # have a look at the first 5 cycles
> abc[,1:5]
> # there are more letters in the table than we are interested
> # in, subset the matrix to rows 1-4 and 15,  corresponding to
> # bases 'A', 'T', 'C', 'G' and 'N', where 'N' represents
> # uncalled bases
> abc <- abc[c(1:4,15),]
> abc
> # set some colour options for the next plot
> cols <- seq(1:5)
> # the matplot takes a matrix and plots each column as a
> # separate set of points
> matplot(t(abc), type="l", lty=rep(1,4), col=cols,
+ xlab="Cycle", ylab="Number of bases" )
```

```
> # plot a legend
> legend( "Topright", legend=rownames(abc), fill=cols )
```

You can use `dev.off()` to close the plotting window.

Exercise 3: We would like to interrogate the distribution of GC content between reads. How many reads have 50% GC content? Can you plot a histogram of reads per GC content?
Start by:

- using `alphabetFrequency` to determine nucleotide counts per read and storing it in an object called `abr`
- subsetting the nucleotide count matrix `abr` to contain only the G and C columns and use `rowSums` to sum the GC content for each read, store the result in an object called `gc`
- use the function *table* on `gc` to count how many times reads with 0, 1, ...,58 GCs occur, this allows you to see how many reads have 29 bases that are either Gs or Cs
- use the function *hist* on `gc` to plot a histogram.

Exercise 4: One issue that frequently arises in the analysis of RNA-seq data is what to do when several reads have the exact same sequence. Which are the top 10 sequences that occur the most frequently and how many of the total sequences show up more than 1 time? Try searching for more information in the man page for `tables` (`?tables`) and `quantile` (`?quantile`).

Most of the sequences occur only one time, but some occur truly many times. The presence of such duplicate reads could have several explanations:

- by chance many RNA/cDNA fragments could come from the exact same place, just because a locus is very highly expressed
- they could also come from repetitive regions, hence from different fragments that happen to have the same sequence
- they could correspond to only one initial fragment, which was amplified more efficiently than other fragments in the PCR, or more efficiently transcribed into cDNA

The most widely adopted explanation is the last one. If you consider this option to be applicable to your data, then in the worst case, you should count these reads only once (see **Section 6.3**), keeping in mind that the dynamic range is being limited by doing so. There are other, less stringent

approaches, especially if you are using PE data (*i.e.* if the reads are genuine, then it is unlikely they have the same mates), see the `rmdup` function of the *samtools* utility tool. And for ChIP-seq, it is common to estimate the expected coverage and using this based on a Poisson distribution, estimate the maximal number of copy of a read to be retained. Nevertheless, whatever your approach, it implies that it is essential to verify the reads alignment once you have boiled your data down to your candidate list of interest.

What we have done in the 2 previous exercises can be done in one step using the function `qa` from the `ShortRead` package:

```
> # create a qa object from the fastq files
> qa <- qa(".",pattern="*\\.fastq\\.gz",
+          type="fastq")
> # create an html report in the qa directory
> report(qa, dest="my_qa")
> browseURL(file.path(getwd(),"my_qa","index.html"))
```

The QA report provides summary statistics about the numbers of reads, base calls and qualities, characteristics of per-lane quality and other information. The QA report is self-documenting and provides a narrative description of each section. Feel free to explore the function documentation for other input formats, such as Solexa Export (the export files of Illumina's pipeline) or BAM files.

6.3 Filtering the FASTQ files

After analysing the quality assessment, one might want to discard some of the reads based on quality, nucleotide composition or duplication criteria. Here are some examples:

```
> # check the initial number of reads
> sr
> # trim the last 18 bases of all reads (for example because you believe they
> # are of bad quality)
> subsr <- narrow( sr, start=1, end=40 )
> # create a custom filter to remove reads with mean base call quality < 20
> qFilt <- srFilter(function(x) {
+     apply(as(quality(x), "matrix"), 1, mean) > 20
+     }, name="GoodQualityBases")
> subsr <- subsr[qFilt(subsr)]
> # filter out reads with more than 2 N (i.e. more than 2 invalid base calls)
> nFilt <- nFilter(threshold=2)
> subsr <- subsr[nFilt(subsr)]
> # remove duplicate reads
> subsr <- subsr[!srduplicated(sread(subsr))]
```

Other approaches of trimming reads is to use the so called dynamic trimming where the 3' end of the reads are chopped based on thei quality value. This yields reads of variable size, but this is not a drawback anymore, as most tools (*e.g.* aligners) deal smoothly with that.

Exercise 5: Write a filter to discard reads that contain the same nucleotide more than 30 times (e.g. polyA runs "AAAAAAAA...."). Try using the function `polynFilter`.

```
> # check number of reads after the filtering steps above
> subsr
> # save this filtered version of the object into a new fastq file in the
> # data directory
> writeFastq( subsr, "SRR490224_filtered.fastq" )
```

Note: In this practical some of these filters might not affect the number of reads. This is because we are working with a subset of the original data.

6.4 De-multiplexing samples

Note: This section does not apply to all datasets but only to multiplexed ones. Since the data we loaded so far into R was not multiplexed we will use a different dataset here.

Nowadays, NGS machines produces so many reads (e.g. 40M for Illumina GAIIx, 100M for ABI SOLiD4 and 160M for an Illumina HiSeq), that the coverage obtained per lane for the transcriptome of organisms with small genomes, is very high. Sometimes it's more valuable to sequence more samples with lower coverage than sequencing only one to very high coverage, so techniques have been optimised for sequencing several samples in a single lane using 4-6bp barcodes to uniquely identify the sample within the libraryLefrançois et al. [2009]. This is called multiplexing and one can on average sequence 12 yeast samples at 30X coverage in a single lane of an Illumina GenomeAnalyzer GAIIx (100bp read, single end). This approach is very advantageous for researchers, especially in term of costs, but it adds an additional layer of pre-processing that is not as trivial as one would think. Extracting the barcodes would be fairly straightforward, but for the average 0.1-1 percent sequencing error rate that introduces a lot of multiplicity in the actual barcodes present in the samples. A proper design of the barcodes, maximising the Hamming distance Hamming [1950] is an essential step for proper de-multiplexing.

The data we loaded into R in the previous section was not mutiplexed, so we now load a different FASTQ file where the 4 different samples sequenced

were identified by the barcodes "ATGGCT", "TTGCGA", "ACACTG" and "ACTAGC". The data is located in *Practicals/QAPractical/data*. To check if your R session is in the correct directory, you can use the `getwd`. If it is not, use the `setwd` to change it.

```
> reads <- readFastq("multiplex.fq")
> # filter out reads with more than 2 Ns
> filter <- nFilter(threshold=2)
> reads <- reads[filter(reads)]
> # access the read sequences
> seqs <- sread(reads)
> # this is the length of your adapters
> barcodeLength <- 6
> # get the first 6 bases of each read
> seqs <- narrow(seqs, end=barcodeLength)
> seqs
```

Exercise 6: Which barcode is most represented in this library? Plot the relative frequency of the top 20 barcodes. Try:

- using the function *table* to count how many times each barcode occurs in the library, you can't apply this function to `seqs` directly you must convert it first to a character vector with the *as.character* function
- sort the counts object you just created with the function *sort*, use `decreasing=TRUE` as an argument for *sort* so that the elements are sorted from high to low (use `sort( ..., decreasing=TRUE )`)
- look at the first element of your sorted counts object to find out with barcode is most represented
- find out what the relative frequency of each barcode is by dividing your counts object by the total number of reads (the function *sum* might be useful)
- plot the relative frequency of the top 20 barcodes by adapting these function calls:

```
> # set up larger margins for the plot so we can read the barcode names
> par(mar=c(5, 5, 4, 2))
> barplot(..., horiz=T, las=1, col="orange" )
```

Exercise 7: The designed barcodes ("ATGGCT", "TTGCGA", "ACACTG" and "ACTAGC") seem to be equally distributed, what is the percentage of reads that cannot be assigned to a barcode?

We will now iterate over the 4 barcodes, split the reads between them and save a new fastq file for each:

```
> barcodes = c("ATGGCT", "TTGCGA", "ACACTG", "ACTAGC")
> # iterate through each of these top 10 adapters and write
> # output to fastq files
> for(barcode in barcodes) {
+     seqs <- sread(reads) # get sequence list
+     qual <- quality(reads) # get quality score list
+     qual <- quality(qual) # strip quality score type
+     mismatchVector <- 0 # allow no mismatches
+
+     # trim sequences looking for a 5' pattern
+     # gets IRanges object with trimmed coordinates
+     trimCoords <- trimLRPatterns(Lpattern=barcode,
+     subject=seqs, max.Lmismatch=mismatchVector, ranges=T)
+
+     # generate trimmed ShortReadQ object
+     seqs <- DNAStringSet(seqs, start=start(trimCoords),
+                          end=end(trimCoords))
+     qual <- BStringSet(qual, start=start(trimCoords),
+                        end=end(trimCoords))
+
+     # use IRanges coordinates to trim sequences and quality scores
+     qual <- SFastqQuality(qual) # reapply quality score type
+     trimmed <- ShortReadQ(sread=seqs, quality=qual, id=id(reads))
+
+     # rebuild reads object with trimmed sequences and quality scores
+     # keep only reads which trimmed the full barcode
+     trimmed <- trimmed[start(trimCoords) == barcodeLength + 1]
+
+     # write reads to Fastq file
+     outputFileName <- paste(barcode, ".fq", sep="")
+     writeFastq(trimmed, outputFileName)
+
+     # export filtered and trimmed reads to fastq file
+     print(paste("wrote", length(trimmed),
+                 "reads to file", outputFileName))
+ }
```

You should have four new FASTQ files: ACACTG.fq, ACTAGC.fq ATGGCT.fq and TTGCGA.fq with the reads (the barcodes have been trimmed) corresponding to each mutiplexed sampled.

After assessing the quality of the raw data, filtering and de-multiplexing you are ready to map your reads to a reference genome which is the subject

of the afternoon session.

7 If you were quick

I've installed on the servers **hippu** and **vuori**, the fastqc tool[1]. To start it, just run

```
/wrk/rnaseq11/FastQC/fastqc
```

directly on hippu or after having started an interactive job on vuori. It will open a window in which you can select your fastq files - which you have to gunzip first - and perform the QC.

8 Additional reading

If you are interested in reading more about the data processing in R, you can find more information and practical there:
http://www.bioconductor.org/help/workflows/high-throughput-sequencing/
And you might want to have a look at the vignettes of the *RnaSeqTutorial* and *easyRNASeq*.

```
> vignette("RnaSeqTutorial")
> vignette("easyRNASeq")
```

[1] from http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

9 sessionInfo

The version number of R[R Development Core Team, 2009] and Bioconductor [Gentleman et al., 2004] packages loaded for generating the tutorial were:

```
R version 2.15.1 (2012-06-22)
Platform: x86_64-apple-darwin10.8.0 (64-bit)

locale:
[1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] Biostrings_2.25.12 IRanges_1.15.42    BiocGenerics_0.3.1

loaded via a namespace (and not attached):
[1] stats4_2.15.1 tools_2.15.1
```

A Solutions to the exercises

Exercise 1:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite(c("EatonEtAlChIPseq","leeBamViews"))
```

Exercise 2: The 1st line is the read ID, the second is the read sequence, the 3rd is a separator that can be ignored and the 4th is the quality of the read sequence where each character represents the quality of the base in the corresponding position.

Exercise 3:

```
> abr <- alphabetFrequency(sread(sr))
> gc <- rowSums(abr[,c("G","C")])
> hgc <- table(gc)
> # 21512 reads have 50% of their sequence comprised of Gs or Cs
> sum(gc>29)
> hist(gc/width(sr)[1],xlab="%GC content",main="GC percent histogram")
```

Exercise 4:

```
> # summarise the number of times each sequence occurs
> tab <- tables( sread(sr), n=length(sr) )
> # show the top 10 sequences that occur the most frequently
> tab$top[1:10]
> # how many of the sequences show up more than 1 time?
> quantile(tab$top, probs=seq(0,1,0.1))
```

Exercise 5:

```
> # discard reads with more than 30 copies of any nucleotide (eg polyA tails)
> pFilt <- polynFilter(threshold=30, nuc=c("A", "C", "T", "G"))
> subsr <- subsr[pFilt(subsr)]
```

Exercise 6:

```
> barcount = sort(table(as.character(seqs)), decreasing=TRUE)
> barcount[1:10]   # TTGCGA
> barcount = barcount/sum(barcount)
> par( mar=c(5, 5, 4, 2))
> barplot(barcount[1:20], horiz=T, las=1, col="orange" )
```

Exercise 7:

```
> (1-sum(barcount[1:4]))*100   # ~6%
```

References

Robert C Gentleman et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology 2010 11:202*, 5(10):R80, Jan 2004.

R W Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, XXIX(2):1–14, Nov 1950.

Kersti Jääger, Saiful Islam, Pawel Zajac, Sten Linnarsson, and Toomas Neuman. Rna-seq analysis reveals different dynamics of differentiation of human dermis- and adipose-derived stromal stem cells. *PLoS ONE*, 7(6): e38833, Jan 2012.

Philippe Lefrançois, Ghia M Euskirchen, Raymond K Auerbach, Joel Rozowsky, Theodore Gibson, Christopher M Yellman, Mark Gerstein, and Michael Snyder. Efficient yeast chip-seq using multiplex short-read dna sequencing. *BMC genomics*, 10(1):37, Jan 2009. doi: 10.1186/1471-2164-10-37.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL `http://www.R-project.org`. ISBN 3-900051-07-0.