Exploring short read sequences – lab

Martin Morgan*

Fred Hutchinson Cancer Research Institute, Seattle, WA

June 27-July 1, 2011

These exercises introduce key *Bioconductor* packages and data structures for working with sequence data. For the first time only, install the package with

```
> pkg <- "http://10.0.1.2/ExploringSequences_0.1.0.tar.gz"
> install.packages(pkg, repos=NULL, type="source")
```

Use the file selection dialog box that appears to select the file ExploringSequences_0.1.0.tar.gz. We start by loading a package designed for the course.

```
> library(ExploringSequences)
```

The package contains data sets, helper functions, and the course material. In particular, the slides and labs are available by starting *R* and entering

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

This launches a web browser; the links labeled R contain scripts that can be run as a short-cut to completing exercises. Evaluate the following to set up paths that point to the data we will use:

```
> fastqFile <- system.file("data", "fq.Rda", package="ExploringSequences")
> bamFiles <- system.file("extdata", "bam", package="ExploringSequences")
> barFile <- system.file("data", "bar.Rda", package="ExploringSequences")
```

Do a 'sanity check' to ensure the path is set correctly

```
> stopifnot(file.exists(fastqFile))
> stopifnot(file.exists(bamFiles))
> stopifnot(file.exists(barFile))
```

1   RNA-seq

For these exercises we use a subset of the data from [1]. The experiment involved RNAi combined with mRNA-seq in *D. melanogaster*. The data itself is available in GEO as part of experiment GSE18508. We look at a subset of samples, summarized in Table 1. There are three biological replicates of untreated and RNAi. The data were collected over a period when there were very rapid changes in technology; the samples with multiple runs are single-end reads, with fewer reads per run and hence several runs per sample. Later samples were paired-end runs with more reads per sample.

---

*mtmorgan@fhcrc.org

Table 1: GEO records from GSE18508

Id	Sample	Run
708	S2_DRSC_Untreated-1	SRR0317086
709		SRR0317096
710		SRR0317106
711		SRR0317116
712		SRR0317126
713		SRR0317136
714	S2_DRSC_Untreated-3	SRR0317147
715		SRR0317157
716	S2_DRSC_Untreated-4	SRR0317168
717		SRR0317178
718	S2_DRSC_CG8144_RNAi-1	SRR0317189
719		SRR0317199
720		SRR0317209
721		SRR0317219
722		SRR0317229
723		SRR0317239
724	S2_DRSC_CG8144_RNAi-3	SRR0317240
725		SRR0317250
726	S2_DRSC_CG8144_RNAi-4	SRR0317261
727		SRR0317271

1.1 Reads

Exercise 1

This exercise reads in one fastq file. The file contains sequence and quality information from 1 million reads of a paired end run; the file name ending with '`_1.fastq`' is from the first mate pair. The reads have been sampled randomly from a larger file.

a. *Use* `load(fastqFile)` *to read the file in to your R session. Explore the object you have created.*

Solution: Read in the data with

```
> load(fastqFile)
> fq

class: ShortReadQ
length: 1000000 reads; width: 37 cycles
```

Normally, fastq files are input using the command `readFastq`. The result is a *ShortReadQ* object. Discover the class of an object with `class`. Find out information about the class with, e.g., `?ShortReadQ`.

```
> class(fq)        # class ShortReadQ

[1] "ShortReadQ"
attr(,"package")
[1] "ShortRead"
```

```
> sread(fq)        # short reads

  A DNAStringSet instance of length 1000000
          width seq
      [1]    37 GTTTTGTCCAAGTTCTGGTAGCTGAATCCTGGGGCGC
      [2]    37 GTTGTCGCATTCCTTACTCTCATTCGGGAATTCTGTT
      [3]    37 GAATTTTTTGAGAGCGAAATGATAGCCGATGCCCTGA
      [4]    37 CGCAAACGCCACGAGTCCACTCTGGGTATTTATGTGA
      [5]    37 GGAAAAAGTCCTGCTCGAGCATTTCAGGAATGGTGCA
      [6]    37 GCTTGCGGAAGTTCTTTACGCGCGCCTGCTCCTGGGG
      [7]    37 GTGGTGGCGCCTAGTTCCGTCCGCCCTTCTCGTCGCA
      [8]    37 GTTCACAGTGCTAGAAATGCGTGGGGTTATATAGAAA
      [9]    37 GTTCATATCGATAAAAAAGATTGCGACCTCGATGTTG
      ...    ... ...
  [999992]    37 GGCCAAACTAAAGGTGTTTTATTCAGACGATAAACAG
  [999993]    37 GGCGAGGGCAACAGATTGTTGGTCCATTTCTTGGTGT
  [999994]    37 GCGACGTCTGCTTCGCGGACACTTACAAGGATGGTTC
  [999995]    37 GTCGTCGATGTTTATCCCGTTTGAGAGAGCCATCTTC
  [999996]    37 CTCCAGGGCAGTCACGTTGAGGATACCGTTTTTATCT
  [999997]    37 CCTCGTTGATGTACATCATATACTTTGTTTTGGCCGC
  [999998]    37 GAGTTCGTAGCGGACCGTTTGCTCGTGGAGTTGGGGT
  [999999]    37 GTCTGGTCTACCAGGGCGTTAAGGTGATGCCCTATTC
 [1000000]    37 GAAGTCGGTACCCTCGAACAGAGAGTCGATCTCAATG

> quality(fq)      # quality scores

class: FastqQuality
quality:
  A BStringSet instance of length 1000000
          width seq
      [1]    37 IIIIIIIIIIIIIIIIIIIIIIIIIIII+HIIII<IE
      [2]    37 IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
      [3]    37 IIIIIIIIIIIIIIIIIIIIII'IIIIIGBIIII2I+
      [4]    37 IIIIIIIIIIIIIIIIIIIIIII,II*E,&4HI++B
      [5]    37 IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII&.$
      [6]    37 III.IIIIIIIIIIIIIIIIIII%IIE(-EIH<IIII
      [7]    37 IIIIIIIIIIIIIIIIIII:IIIIIIIIIIII?III)
      [8]    37 IIIIIIIIIIIIIIIIIIIII+II$IIIIIIIIIIII
      [9]    37 IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII?II?II
      ...    ... ...
  [999992]    37 IIIIIIIIIIIIII+IIIIIIIIHIIABI%III6CIH
  [999993]    37 .IIIIIIIIIIIIIIIIIIIIIIIII?2&%&3/$*6$
  [999994]    37 +IIIIIIIIIBIIIIIII=III8IIGAC5;D*-9I2&2<
  [999995]    37 IIIIIIIIIIIIIIIIIIIIIIIIIIIIICIIIII
  [999996]    37 IIIIIIIIIIIIIIII6IIIIIII.IH=$.1#'"#%%$
  [999997]    37 IIIIIIIIIIIIIIIIII*IIIIIIIIIIIIIIIII$
  [999998]    37 IIIIIIIIIIIIIIIIIIIIIIIIIIIIAIIIIDII/&6'
  [999999]    37 IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
 [1000000]    37 IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
```

What do the letters in the quality score represent?

Exercise 2

The goal of the next several exercises is to summarize mono- and dinucleotide use in our reads. To start:

a. *Extract the reads from an instance of the ShortReadQ class, using* `sread`.

b. *Calculate the frequency of the letters used in each read, using* `alphabet-Frequency`. *Consult the help page for this function,* `?alphabetFrequency`, *and use the* `baseOnly` *and* `collapse=TRUE` *arguments to ignore IUPAC ambiguity letters (there are none in our reads) and to report the nucleotide counts over all reads (rather than for each read separately).*

Solution:

```
> mono <- alphabetFrequency(sread(fq), baseOnly=TRUE, collapse=TRUE)
> mono

       A        C        G        T    other
 8364351 10002656 10189120  8406472    37401
```

Exercise 3

Determine the frequency of di-nucleotides. Do this using the `dinucleotideFrequency` *function. The operation is a little trickier, because* `dinucleotideFrequency` *does not have an option to collapse counts over all reads. Instead...*

a. *Use* `dinucleotideFrequency` *to create a matrix, with each column representing a different combination of nucleotides and each row a different read.*

b. *Use* `colSums` *to sum up each column.*

Solution: The following takes a *ShortReadQ* instance and returns dinucleotide counts:

```
> di <- colSums(dinucleotideFrequency(sread(fq)))
```

Exercise 4

The following exercises asks about the distribution of 'GC' nucleotide content between reads.

a. *Use* `alphabetFrequency` *to determine nucleotide use, but omit the* `collapse` *function argument so that the result is a matrix summarizing use in each read.*

b. *Subsets the nucleotide use matrix to select the 'G' and 'C' columns, and uses* `rowSums` *to sum the GC content for each read (row of the matrix).*

c. *Uses* `tabulate` *to count how many times reads with 0, 1, . . . 37 'GC' occurs. A trick here is that* `tabulate` *ignores zeros; by adding 1 to each count, we get a vector where the first element is the number of reads with 0 'GC' nucleotides, the second element is the number of reads with 1 'GC' nucleotide, and so on.*

There are several steps involves, so formulate your solution as a function.

Solution: A function determining the distribution of GC content is:

```
> gcContent <- function(x)
+ {
+     abc <- alphabetFrequency(sread(x), baseOnly=TRUE)
+     gcPerRead <- rowSums(abc[,c("G", "C")])
+     wd <- unique(width(sread(x)))
+     tabulate(1 + gcPerRead, 1 + wd)
+ }
```

Applied to our data, we have

```
> gc <- gcContent(fq)
> gc

 [1]     91     29     31     88    155    412    913   1670   3070   5513
[11]   7137  10359  14422  19507  25642  33830  44586  58652  74725  90623
[21] 103081 109063 105138  92233  73137  52001  33855  19888  10288   5189
[31]   2416   1196    593    268    137     38     17      7
```

Exercise 5

Visualization is an important tool for gaining insight. R has flexible built-in graphics commands, but additional packages provide expressive (lattice) and pretty (ggplot2) alternatives. This lab uses lattice. Most lattice functions expect a data frame in 'long' format, where one or more columns contain the data to be plotted. Additional columns are indicator variables indicating which group the correspond rows belongs to.

Start by making a vector `bins` to indicate GC content (0, 1, . . .). Use this in creating a 'long' data frame with. . .

a. *A column `Count` containing the counts calculated previously.*

b. *A column `GC` containing our 'GC' content from `bins`, scaled to represent a proportion*

A powerful and expressive feature of lattice is the use of a formula to describe the relationship between plotted variables, in this case Count as a function of GC content. The `type` argument controls what gets plotted – a background grid and both lines and points. `pch` controls the type of point (try `example(points)`). `file` is used in this vignette; it is not part of lattice.

Solution: The following creates a data frame. . .

```
> bins <- seq_along(gc) - 1
> df <- data.frame(Count=unlist(gc, use.names=FALSE),
+                   GC=bins / bins[length(bins)])
```

. . . and plots the result, Figure 1

```
> xyplot(Count~GC, df, type=c("g", "b"), pch=20, file="gc")
```
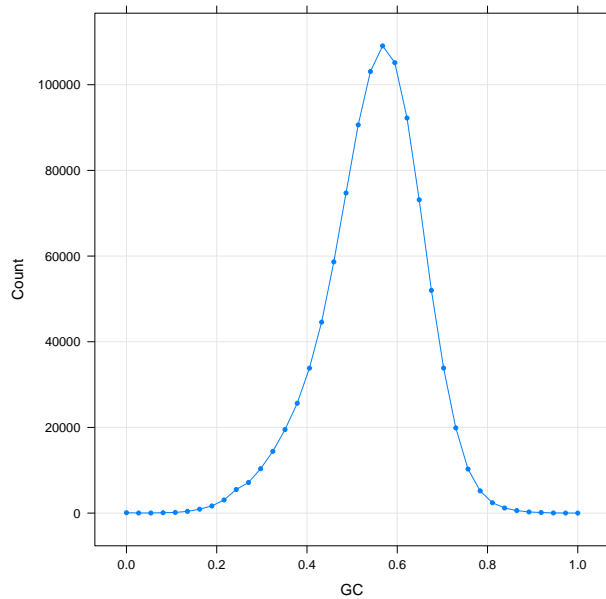
Figure 1: Read GC content.

Exercise 6

A simple null expectation is that dinucleotide frequencies are the product of the mononucleotide frequencies.

  a. *Calculate the frequency of mononucleotides (ignoring 'N' for simplicity), and use the `outer` product to determine the expected dinucleotide frequency.*

  b. *Express the expected dinucleotide frequency in the sample as counts by multiplying by the sample total dinucleotide count.*

  c. *Use `chisq.test` to calculate a χ^2 test.*

Create a graphical display, and comment on the results.

Solution:

```
> ## outer product of mononucleotides frequencies
> f <- mono[-5] / sum(mono[-5])
> exp0 <- as.vector(outer(f, f))
> ## expected values, as counts
> n <- sum(di)
> exp <- exp0 * n
> mode(exp) <- "integer"
> head(exp, 3)

[1] 1840839 2201400 2242437

> ## Chi-squared test
> chisq.test(cbind(di, exp))
```
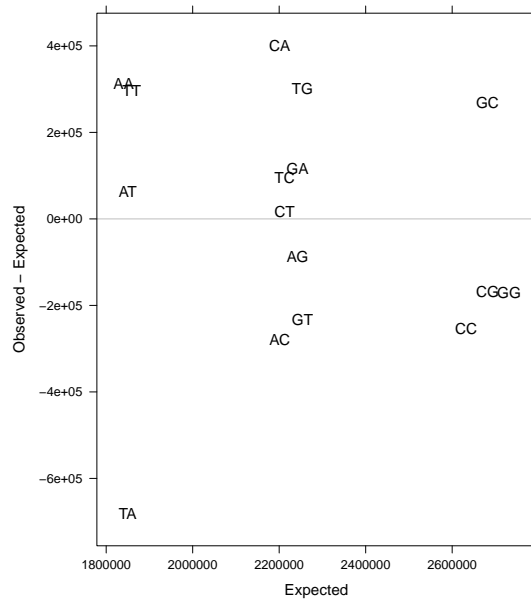
6

Figure 2: Deviation of observed from expected dinucleotide counts.

```
        Pearson's Chi-squared test

data:  cbind(di, exp)
X-squared = 327580.7, df = 15, p-value < 2.2e-16
```

To display the data using *lattice*, create a data frame:

a. Coerce columns `Observed` and `Expected`, containing the corresponding vectors.

b. Use the `names` of `di` to create a column `Label`.

The `panel` argument describes how each panel will be created – drawing a horizontal line of color gray (`panel.abline`) followed by placement of text at particular coordinates (`panel.text`; the 'usual' xy plot panel function is `panel.xyplot`). Note the use of ... to forward arguments not directly used by our custom panel function.

```
> ## display
> df <- data.frame(Observed=as.vector(di), Expected=as.vector(exp),
+                  Label=names(di), stringsAsFactors=FALSE)
> xyplot(Observed - Expected ~ Expected, df, aspect="iso",
+        panel=function(...) {
+            panel.abline(h=0, col="gray")
+            panel.text(..., labels=df$Label)
+        }, file="dinuc")
```

The result is in Figure 2, with the following interesting points:

7

a. The χ^2 test indicates departure from the null.

b. There is a strong bias against TA dinucleotides; knowledge of TA use in the reference genome would help us to understand whether this is a biological result or a technical artifact.

c. Similarly, G, C nucleotide pairs are most common.

d. The χ^2 test is not really satisfactory, because each nucleotide in a read is counted twice (as the first and then second nucleotide in the pair). Any suggestions for improving the analysis?

Exercise 7
Nucleotide counts in each cycle of a read provide important insight into sample preparation and technological artifacts.

a. *Write a short function that extracts reads from a ShortReadQ object and uses* `alphabetByCycle` *to summarize nucleotide use by cycle. Subset the value returned by* `alphabetByCycle` *to include only the nucleotides A, C, G, T.*

b. *Apply this function to* `fq`*, and explore the result.*

c. *Display the result using* `lattice`*. To do this, cast the results into a long data frame, and then use* `xyplot`*. It is easiest to start with a simple plot, and to subsequently adjust display and formatting.*

Solution: Here we calculate and plot DNA alphabet use as a function of cycle.

```
> nuc <- c("A", "C", "G", "T", "N")
> abc <- alphabetByCycle(sread(fq))[nuc,]
> abc[,1:5]

        cycle
alphabet    [,1]    [,2]    [,3]    [,4]    [,5]
       A   78194  153156  200468  230120  283083
       C  439302  265338  362839  251434  203787
       G  397671  270342  258739  356003  301640
       T   84833  311164  177954  162443  211490
       N       0       0       0       0       0
```

The following creates a data frame and displays the result using *lattice*.

```
> ## create a 'long' data frame
> ncycle <- ncol(abc)
> cycle <- rep(seq_len(ncycle), each=nrow(abc))
> df <- data.frame(Count=as.vector(abc),
+                  Nucleotide=factor(nuc, levels=nuc),
+                  Cycle=cycle)
> ## plot Count as a function of cycle, with Nucleotide used to group lines
> ## within a panel.
```
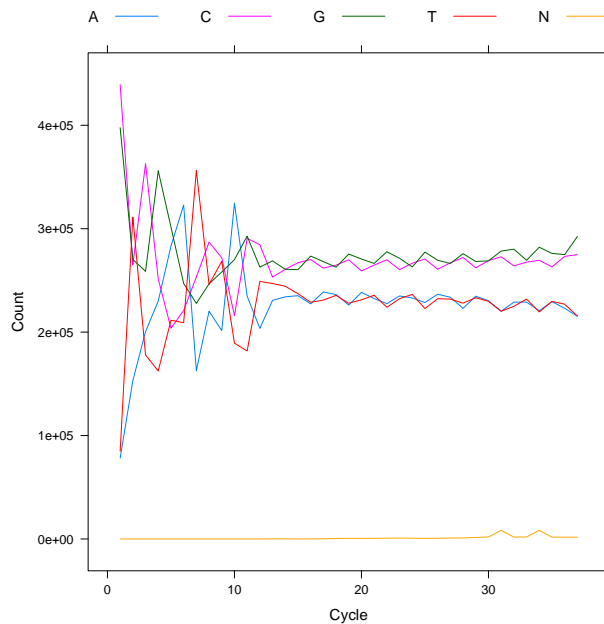
Figure 3: Alphabet-by-cycle.

```
> xyplot(Count ~ Cycle, group=Nucleotide, df, type="l",
+        key=simpleKey(lines=TRUE, points=FALSE, columns=5, text=nuc),
+        file="abc-read")
```

Interesting points (some well-known) from Figure 3 include:

a. Primers introduce initial bias, through cycle ≈ 12.

b. Under a null of uniform coverage, lines should be horizontal; but the plots suggest that A, T decrease in frequency as cycles progress. This is much more pronounced in early (GAI) Solexa / Illumina runs.

c. There is a weak but consistent periodic bias – e.g., every second cycle has relatively more A, T.

Exercise 8
Base quality decreases as cycle number increases. Calculate the average quality per cycle, and display the result. To calculate quality per cycle. . .

a. *Extract the quality score from a ShortReadQ instance using the* `quality` *function.*

b. *Coerces the quality score to a numerical matrix representation using the* `as` *function. This matrix has as many rows as there are reads, and as many columns as there are cycles.*

c. *Calculate the average quality score of each cycle using* `colMeans`.
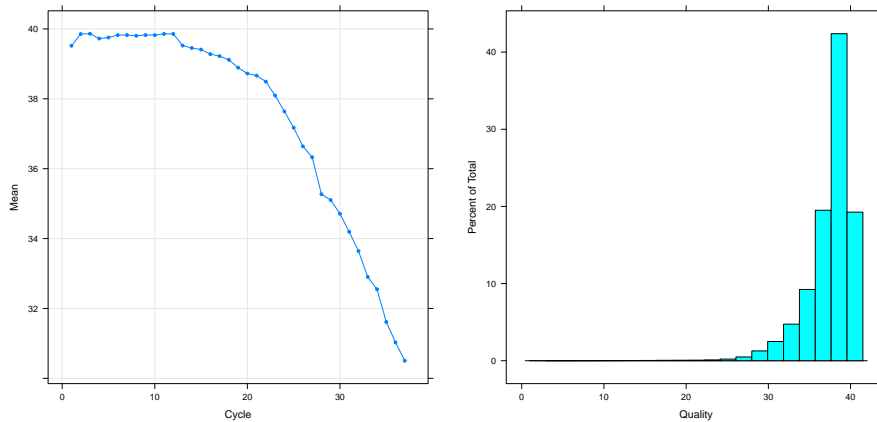
9

Figure 4: Quality-by-cycle (left) and distribution of read quality (right).

    d. Develop a second function that calculates an overall read quality using `rowMeans`.

Visualize the result using lattice.

    Solution: Calculating quality by cycle requires translating character encodings to their numeric representation.

```
> abc <- colMeans(as(quality(fq), "matrix"))
```

The following creates a data frame and displays the result.

```
> df <- data.frame(Mean=abc, Cycle=seq_along(abc))
> xyplot(Mean ~ Cycle, df, type=c("g", "b"), pch=20,
+        file="abc-quality")
```

Using the average quality of each read as a measure of 'overall quality', we have

```
> qual <- rowMeans(as(quality(fq), "matrix"))
> df <- data.frame(Quality=qual)
> histogram(~Quality, df, file="quality-by-read")
```

Interesting, mostly well-known, points from Figure 4 include:

    a. Average quality declines with cycle.

    b. The second mate of paired reads is consistently lower quality.

    c. Average quality and error are related (larger deviation from the diagonal at higher quality scores).

Exercise 9
Individual read sequences can be represented 1, 2, … times in a sample. The overall extent to which reads are duplicated can indicate depth of coverage. Singleton sequences (represented exactly once) and sequences with very high
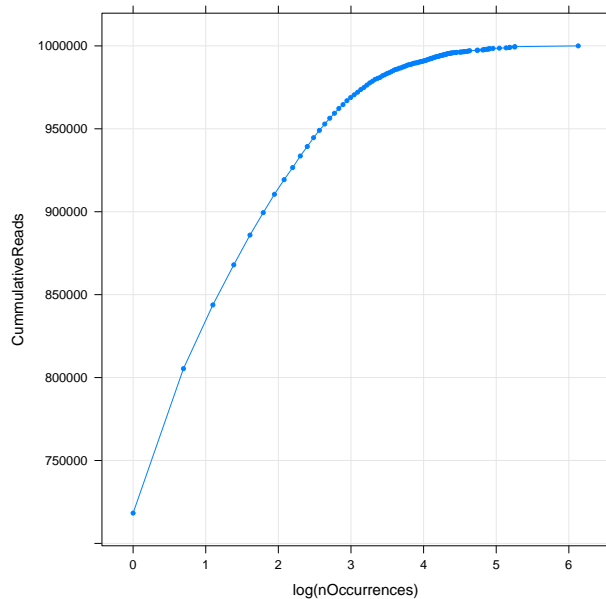
10

Figure 5: Cumulative frequency of reads occurring 1, 2, ... times.

representation often reflect limitations of technology. Use the `tables` function, and the element `distribution` of the return value, to create and interpret a figure that plots the cumulative number of reads as a function of the number of times a read occurs in a sample.

Solution:

```
> t <- tables(sread(fq), n=0)[["distribution"]]
> df <- data.frame(nOccurrences=t$nOccurrences,
+                  CummulativeReads=cumsum(t$nOccurrences * t$nReads))
> ## display
> xyplot(CummulativeReads ~ log(nOccurrences), df,
+        type=c("b", "g"), pch=20, file="freqseq")
```

Interesting points (Figure 5):

a. The intercept (singleton reads) will decrease as sample size increases – repeated observation of the same reads. The large value here implies limited coverage.

b. That the second of the read pair is above the first might partly reflect elevated error rates.

c. The most abundant reads in the second of the read pair reflects bias – particular sequences (degenerate?) become super-abundant.

Exercise 10

ShortRead can create a report summarizing several `fastq` files. A short script (see the appendix) was used to summarize all fastq files; here we read in the summary and generate the report.

    a. *Load the data R object `qa_GSM461176_81`, representing the qa summary of GSM records 461176 through 461181. The path to the object is*

```
> qaFile <- system.file("data", "qa_GSM461176_81.rda",
+                         package="ExploringSequences")
```

    b. *Use the `report` function to generate a report, by default in a temporary directory.*

    c. *Browse the report using `browseURL`.*

The 'Id' column in Table 1 is used to identify samples in the QA report; `_1` or `_2` refers to the first and second mate in paired end runs. Reflect on the quality of the reads in this experiment.

    Solution:

```
> load(qaFile)
> rpt <- report(qa_GSM461176_81)
> browseURL(rpt)
```

If for some reason the report generation fails, a copy is available at

```
> rpt <- system.file("GSM461176_81_qa_report", "index.html",
+                      package="ExploringSequences")
> browseURL(rpt)
```

Interesting points:

    a. Substantial variation in number of reads per sample; low read counts in samples 709, 719, 723.

    b. GC content of 718-723 is unusual – all from `S2_DRSC_CG8144_RNAi-1`.

    c. Samples are quite heterogeneous in distribution of read qualities.

    d. Read distributions are not super-saturated – opportunity for greater sequencing? Sample 719 is unusual – disproportionate representation of very common sequences.

    e. All samples show initial primer bias; some samples, e.g., 708, 713, 717_2 show cycle-specific base trends. Sample 723 has unusual final nucleotides.

    f. Samples differ in read length, maximum quality – likely different machines and chemistry. 708-713, 718-723 are longer with low-quality tails.

1.2 Alignments

The primary goal of this exercise is to become familiar with the *Rsamtools* package for querying BAM alignment files

Exercise 11
The goal of this exercise is to open several BAM files as a BamFileList, and to query one BAM file for the information about the reference sequences to which reads have been aligned (using the `seqinfo` function) and the software tools used to perform the alignment (using `scanBamHeader` and parsing the return value).

Solution: Create a list of `BamFile` objects, each pointing to a file and its index. Using a `BamFile` avoids loading the index each time the file is used.

```
> fls <- list.files(bamFiles, "bam$", full=TRUE)
> stopifnot(length(fls) == 7)
> bam <- open(BamFileList(fls))
> names(bam) <- sub("_subset.bam", "", basename(fls))
```

Summary information is available in the header of each file, e.g., querying for a summary of (reference) sequences and their lengths, or as tags available in the `text` element of the header (see the samtools web site for details of header content, including meaning of tags). For instance, our BAM files record the command line used to generate them.

```
> seqinfo(bam[[1]])
> h <- scanBamHeader(bam[[1]])[["text"]]
> noquote(unname(sapply(h[["@PG"]], strwrap)))
```

Exercise 12
Common operations on BAM files are available as functions. For instance, `countBam` counts the number of reads in each BAM file. A more interesting use is to specify the `param` argument to this and other Rsamtools functions.

  a. *Create a GRanges object containing coordinates of four Drosophila genes; see the solution for one such set.*

  b. *Create a ScanBamParam object to specify the regions on which BAM file operations are to be performed.*

  c. *Use `countBam` to count reads in each of the regions.*

  d. *Aligned reads have various flags that summarize the status of the read or alignment. Use `countFlags` to summarize flags of reads in one of the genes.*

Solution: `countBam` can be used to retrieve the number of reads aligned in each BAM file. N.B., `countBam` is not meant to be used for counting reads in complex regions as in an RNAseq analysis; see `?countGenomicOverlaps`.

```
> cnt <- countBam(bam)
> cnt
```

```
          space start  end width                   file records nucleotides
treated1     NA    NA   NA    NA    treated1_subset.bam   15937      709566
treated2     NA    NA   NA    NA    treated2_subset.bam   16413      607281
treated3     NA    NA   NA    NA    treated3_subset.bam   21364      790468
untreated1   NA    NA   NA    NA  untreated1_subset.bam   18929     1419675
untreated2   NA    NA   NA    NA  untreated2_subset.bam   28279     1272555
untreated3   NA    NA   NA    NA  untreated3_subset.bam   19164      709068
untreated4   NA    NA   NA    NA  untreated4_subset.bam   20245      749065
```

The `ScanBamParam` function creates an object that allows easy access to particular portions of the file. For instance, the `which` argument can be used to select which genomic regions are accessed. The `which` argument could be one of several different objects, for instance a `GRanges` containing chromosomes and the regions of interest.

```
> which <- GRanges(c("chr3L", "chrX", "chr3L"),
+                IRanges(c(1871574, 10675019, 14769596),
+                        c(1876336, 10680978, 14779523)))
> param <- ScanBamParam(which=which)
> head(cnts <- countBam(bam, param=param))
```

```
            space      start       end width                file records
treated1.1  chr3L   1871574   1876336  4763 treated1_subset.bam    8464
treated1.2  chr3L  14769596  14779523  9928 treated1_subset.bam    2912
treated1.3   chrX  10675019  10680978  5960 treated1_subset.bam    1389
treated2.1  chr3L   1871574   1876336  4763 treated2_subset.bam    9280
treated2.2  chr3L  14769596  14779523  9928 treated2_subset.bam    2202
treated2.3   chrX  10675019  10680978  5960 treated2_subset.bam    1175
           nucleotides
treated1.1      376766
treated1.2      129703
treated1.3       61846
treated2.1      343360
treated2.2       81474
treated2.3       43475
```

```
> ## create a label, e.g., chr3L:1871574-1876336
> cnts$rgn <- with(cnts, sprintf("%s:%d-%d", space, start, end))
> xyplot(file~records, group=rgn, cnts, type="b", pch=20,
+        auto.key=list(lines=TRUE, points=FALSE, columns=2),
+        file="bam-regions")
```

The results are shown in Figure 6.

BAM files contain a `flag` field encoding information about the aligned reads, e.g., the strand to which they aligned, whether they are mate paired, whether the pairs are 'proper' in the eyes of the alignment tool producing the BAM file. Here we summarize reads satisfying various flags (note that a read may be tallied under more than one category, e.g., is a proper pair and is on the negative strand).

```
> param1 <- ScanBamParam(which=which[3])
> countFlags(bam, param=param1)
```
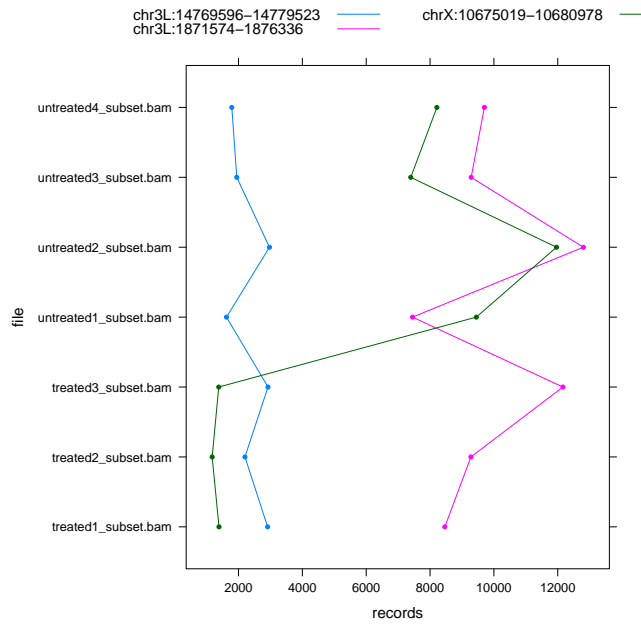
Figure 6: Number of reads aligning to regions of interest.

	File	Range	isPaired	isProperPair
treated1	treated1_subset.bam	chr3L:14769596-14779523	0	0
treated2	treated2_subset.bam	chr3L:14769596-14779523	2202	994
treated3	treated3_subset.bam	chr3L:14769596-14779523	2924	1323
untreated1	untreated1_subset.bam	chr3L:14769596-14779523	0	0
untreated2	untreated2_subset.bam	chr3L:14769596-14779523	0	0
untreated3	untreated3_subset.bam	chr3L:14769596-14779523	1946	948
untreated4	untreated4_subset.bam	chr3L:14769596-14779523	1791	944

	isUnmappedQuery	hasUnmappedMate	isMinusStrand	isMateMinusStrand
treated1	0	0	1516	0
treated2	0	598	1110	802
treated3	0	282	1462	1324
untreated1	0	0	774	0
untreated2	0	0	1586	0
untreated3	0	141	963	901
untreated4	0	406	892	693

	isFirstMateRead	isSecondMateRead	isPrimaryRead	isValidVendorRead
treated1	0	0	0	0
treated2	1275	927	0	0
treated3	1500	1424	0	0
untreated1	0	0	0	0
untreated2	0	0	0	0
untreated3	978	968	0	0
untreated4	1013	778	0	0

	isDuplicate
treated1	0

```
treated2            0
treated3            0
untreated1          0
untreated2          0
untreated3          0
untreated4          0

> bam2 <- bam[-c(1, 4, 5)]
```

Exercise 13

The previous exercise introduced BAM files, some aspects of the ScanBamParam class, and built-in functions for retrieving information from the file.

This and the next exercise illustrates how Rsamtools *can be used more creatively. As a first step, suppose one is interested in the 'insert' distance between mapped reads of a mate pair. Construct a* ScanBamParam *object with:*

- a. *The* flag *argument, determined by the* scanBamFlags *function, to select the first read of each 'proper' mate pair.*

- b. *The* which *argument and GRanges class created above, to restrict the query to a particular region of the genome.*

- c. *The* what *argument to return the* pos *(alignment position) and* mpos *(mate read alignment position; arguments are described on* ?scanBam *help page).*

Use this, the BamFileList *object, and the* scanBam *function to query the BAM files for the relevant information. The insert width is the absolute value of the difference between the* pos *and* mpos *locations; calculate this for each sample, and summarize as cumulative distribution.*

Solution: Here is the ScanBamParam object:

```
> param2 <- ScanBamParam(
+     what=c("pos", "mpos"), which=which[3],
+     flag=scanBamFlag(isProperPair=TRUE, isFirstMateRead=TRUE))
```

Now query the BAM files, and take a look, using str, at the list-of-list-of-lists returned:

```
> iwd0 <- lapply(bam2, scanBam, param=param2)
> str(iwd0)

List of 4
 $ treated2  :List of 1
  ..$ chr3L:14769596-14779523:List of 2
  .. ..$ pos : int [1:497] 14769636 14769636 14769649 14769649 14769649 14769650 14769650
  .. ..$ mpos: int [1:497] 14769751 14769768 14769788 14769794 14769758 14769779 14769786
 $ treated3  :List of 1
  ..$ chr3L:14769596-14779523:List of 2
  .. ..$ pos : int [1:662] 14768575 14768576 14768577 14768577 14768578 14769599 14769599
  .. ..$ mpos: int [1:662] 14769114 14769129 14769137 14769116 14769112 14769744 14769746
 $ untreated3:List of 1
```

```
  ..$ chr3L:14769596-14779523:List of 2
  .. ..$ pos : int [1:474] 14769649 14769649 14769649 14769649 14769649 14769650 14769650
  .. ..$ mpos: int [1:474] 14769806 14769766 14769793 14769800 14769794 14769778 14769795
 $ untreated4:List of 1
  ..$ chr3L:14769596-14779523:List of 2
  .. ..$ pos : int [1:470] 14769605 14769642 14769649 14769649 14769650 14769650 14769650
  .. ..$ mpos: int [1:470] 14769749 14769766 14769781 14769806 14769768 14769779 14769795
```

Starting at the innermost list, the absolute value of the difference between each
element of the `pos` and `mpos` vectors represents the insert width (not quite –
what distance does this actually measure? What additional information would
be required to measure the insert distance? Is this information available from
the BAM file?). This information is itself represented in a list, so write a function
that extracts the first element of a list, and then calculates the distance between
mate pairs. Use `lapply` to calculate this distribution for each sample.

```
> fun <- function(elt)
+     with(elt[[1]], abs(pos - mpos))
> str(iwd <- lapply(iwd0, fun))

List of 4
 $ treated2  : int [1:497] 115 132 139 145 109 129 136 147 135 128 ...
 $ treated3  : int [1:662] 539 553 560 539 534 145 147 142 139 130 ...
 $ untreated3: int [1:474] 157 117 144 151 145 128 145 144 147 146 ...
 $ untreated4: int [1:470] 144 124 132 157 118 129 145 138 146 150 ...
```

To visualize the distribution, use (from the *lattice* package) the `make.groups`
function, invoked with `do.call`, to create a data frame with two columns, and
suitable for use in `densityplot`.

```
> df <- do.call(make.groups, iwd)
> densityplot(~data, group=which, df, plot.points=FALSE,
+             auto.key=list(lines=TRUE, points=FALSE, columns=2),
+             file="insert")
```

2   Microbiome / 454 sequences

Microbiome sequencing projects characterize the representation of organisms in
well-defined communities (e.g., flora of the human gut). Metagenomics aims
one step further, asking about the biological functions provided by the suite of
organisms found in a community. Both endeavors rely on assignment of longer
(200-400bp) sequence reads to a reference database of taxa.

A typical microbiome work flow involves collection of DNA samples from a
large number (10's to 100's) of individuals. Targeted PCR enriches samples for
one or a few phylogenetically informative markers, e.g., specific regions of 16s
RNA. A bar code is added to PCR-enriched DNA to allow sample multiplex-
ing. Sequencing uses a technology, e.g., 454, that produces reads that are long
enough to provide phylogenetic signal. Sequence processing requires reads to
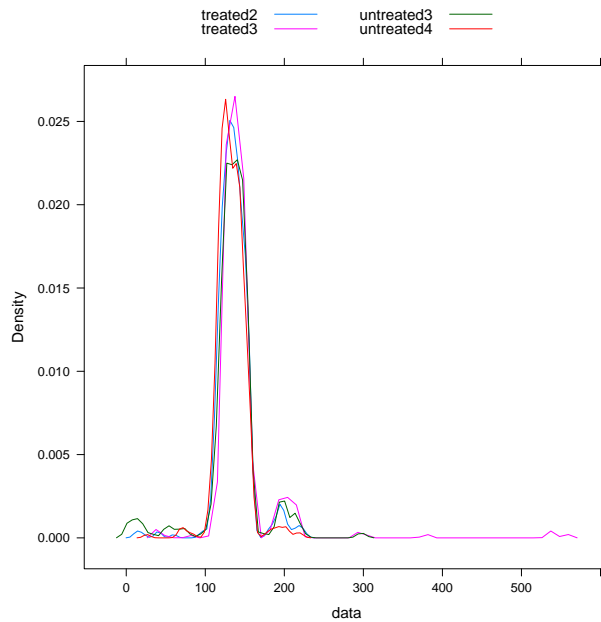be de-multiplexed. There are typically PCR artifacts (e.g., primer sequence) to

Figure 7: Insert size of proper mate pairs at `chr3L:1871574-1876336`.

be cleaned. Pre-processed reads are aligned to curated collections of reference genomes, with representation of taxa in the sample inferred from reads aligned to reference genome.

These exercises focus on read manipulation prior to classification, with emphasis on

1. Data input.

2. Sub-sequence extraction and manipulation

3. Pattern matching.

The data are a subset of bacterial 16s sequences sampled from a human body cavity. Down-stream analysis (beyond the scope of this tutorial) can use excellent R packages for community and phylogenetic analysis (e.g., ape, vegan).

Exercise 14
454 technology is different from the Illumina platform. Reads are initially made available as 'flows' in the `sff` *file format. Unfortunately, there are no Bioconductor packages that work directly with* `sff` *files. Instead, analysis begins with fastq-like data, typically presented as pairs of fasta sequence (`.seq`) and quality (`.qual`) files.*

Input the sample data using the `load(barFile)` *function. The data is represented as a ShortReadQ object, the same as seen earlier. Note that the read widths are longer (up to 342 cycles) and variable. Quality scores also follow a different pattern. Subset the reads to contain only those with width 250 or more, then use the* `narrow` *function to look at the average quality of the trailing 250 nucleotides.*

18

Solution: Here we read in the data.

```
> load(barFile)
> bar

class: ShortReadQ
length: 262746 reads; width: 35..342 cycles

> summary(width(bar))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   35.0   248.0   251.0   248.2   254.0   342.0
```

Usually, 454 data is read in using the function `read454`.

Here we select reads with width 254 and look at the alphabet cycle at positions 100-110. Reads with this width

```
> bar254 <- bar[width(bar) == 254]
> alphabetByCycle(narrow(sread(bar254), 101, 110))[1:4,]

         cycle
alphabet  [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
       A     0     1     1     1     2     5  1204 61901   269 63108
       C    10    30 63084   264    40 63109 62132   239    23     5
       G   238    13    12  1180    19     3     4     9  1183    12
       T 63135 63339   286 61938 63322   266    43  1234 61907   258
```

Exercise 15
Microbiome studies generally benefit from many individuals and relatively fewer sequences. Samples are therefore multiplexed by preceding the target sequence with a bar code, in this case associated with the first 8 nucleotides of each read.

  a. Use `narrow` *function to isolate the bar codes.*

  b. Use `table` *to determine the occurrence of each code.*

  c. Create a subset of reads corresponding to the most common bar code, and

  d. Use `narrow` *again, this time trimming the bar code and two adapter nucleotides (nucleotides 1-10) from the sequences.*

Solution: The following narrows the reads to positions 1-8 (containing the bar code), and then tabulates and sorts, using standard R functions, the number of times each bar code occurs. We then focus on the most abundant bar code.

```
> codes0 <- narrow(sread(bar), 1, 8)
> codes <- as.character(codes0)
> cnt <- sort(table(codes), decreasing=TRUE)[1:5]
> cnt

codes
AAGCGCTT AAGCTTGC AAGCGGTA AAGCTTCG AAGCTAGG
   23387    20743    20602    18740    17567
```

```
> aBar <- bar[codes==names(cnt)[[1]]]
> aBar

class: ShortReadQ
length: 23387 reads; width: 47..307 cycles
```

Now that `aBar` consists of reads from a single bar code, we can remove those nucleotides using `narrow` to focus on the 11th through final nucleotides

```
> noBar <- narrow(aBar, 11, width(aBar))
> noBar

class: ShortReadQ
length: 23387 reads; width: 37..297 cycles
```

Exercise 16
A typical microbiome sample preparation involve targeted PCR amplification. In this case the (redundant) PCR primer is present in the read. Further, the primer is not always present fully intact. The primer sequence is

```
> pcrPrimer <- "GGACTACCVGGGTATCTAAT"
```

Trim the primer using the pattern-matching function `trimLRPatterns`. *Summarize the amount of trimming that has occurred by tabulating the difference between the width of the sequences with the bar codes removed, and the same sequences with the primer trimmed.*

   Solution:

```
> trimmed <-
+     trimLRPatterns(pcrPrimer, subject=noBar, Lfixed=FALSE)
> trimmed

class: ShortReadQ
length: 23387 reads; width: 26..291 cycles

> table(width(noBar) - width(trimmed))

    0     1    11    12    17    18    19    20
  843     3     1     1     1     3    26 22509
```

References

[1] A. N. Brooks, L. Yang, M. O. Duff, K. D. Hansen, J. W. Park, S. Dudoit, S. E. Brenner, and B. R. Graveley. Conservation of an RNA regulatory map between Drosophila and mammals. *Genome Res.*, 21:193–202, Feb 2011.

A Appendix

The QA report data was collated with the following script. For each fastq file, we create an identifier `id` from its file name, and read the file in with `readFastq`, and calculate summary statistics using `qa`. `doit` will perform an `lapply` operation on each file, but if the *multicore* package is available the operation will be in parallel. The result `qas` is a list of qa summaries; these are bound together into a single object using `rbind`.

```
> fun <- function(fl) {
+     id <- sub(".fastq$", "", basename(fl))
+     qa(readFastq(fl), id)
+ }
> fls <- list.files(pattern=".fastq$", full=TRUE)
> doit <-
+     if (suppressWarnings(require("multicore"))) {
+         mclapply
+     } else lapply
> qas <- doit(fls, fun)
> qa_GSM461176_81 <- do.call(rbind, qas)
```