Practical 1: Reading and Manipulating Short Reads

Martin Morgan[*]

7-9 June, 2010

Contents

1 Introduction

This practical uses the *ShortRead* package to input aligned and other short read data files, and illustrates some of the available sequence manipulation and quality assessment tools. Activities during the lab are posed as exercises. So, as a first exercise:

Exercise 1
Start an R session, and load the ShortRead package.

[*]Fred Hutchinson Cancer Research Center, Seattle, WA 98008

```
> library(ShortRead)
> packageDescription("ShortRead")$Version

[1] "1.6.2"
```

Confirm that the version of your package is at least as recent as the version in this document. Seek assistance from one of the course assistants if you need help getting the current version of ShortRead.

The course also requires access to sample data.

Exercise 2
Obtain and load the EMBL2010 package. Use

```
> install.packages("path/to/EMBL2010.tar.gz", repos=NULL)
```

2 Aligned read input

This section illustrates input of aligned reads. It focuses on aligned reads produced by the Solexa Genome Analyzer *ELAND* software; reading data produced by software such as *MAQ* or *Bowtie*, or in *BAM* format is described in the *ShortRead* 'Overview' vignette and on the `readAligned` help page.

2.1 Navigating Solexa output

Vendor and third-party software is likely to process raw images, base calling (*Rolexa* is a *Bioconductor* package providing alternative base calling), and alignment to a reference genome (see *BSgenome* and the `matchPDict` function of *Biostrings*). *Bioconductor* packages might enter a typical work flow after alignment. The following starts with reads aligned with *ELAND*, an alignment program from the Illumina Genome Analyzer II (GAII) platform.

We'll start by creating a variable, `extdataDir`, containing the directory holding the sample GAII data, and check to make sure that we have defined the right path.

```
> extdataDir <-
+     system.file("extdata", "slx", package="EMBL2010")
> file.exists(extdataDir)

[1] TRUE

> stopifnot(file.exists(extdataDir))
```

A key functionality provided by *ShortRead* is input of a diversity of file types, both of files from the Illumina pipeline and from other software and in formats appropriate for other technologies. The interface to these input functions is meant to facilitate reading one or more files into a single object. The interface is like that for `list.files`: provide a directory path where relevant files are to be found, plus a regular expression to select the files you are interested in.

Exercise 3
Use `list.files` to display all files in the `extdataDir` directory.

```
> list.files(extdataDir)

[1] "s_1_0001_int.txt.gz"      "s_1_0001_nse.txt.gz"
[3] "s_1_1_export_head.txt"    "s_1_1_sequence_head.txt"
```

We'll select just one file to use in this analysis, based on files matching the regular expression `.*_export_head.txt`. These files are produced using *ELAND* software run in `eland-extended` mode. This mode produces files, one for each lane (and 'end' of paired end reads) that summarize diverse features of *all* reads, and is a very convenient starting point for analysis; other inputs (e.g., *MAQ*, *Bowtie*, *BWA*, or *BAM* files) are also supported.

Exercise 4
We'll use an abbreviated file for most parts of this lab. The abbreviated file contains the first 500,000 reads from a single lane of an Illumina paired-end run. It is from lane 1, and we will use the first end only. The name of the file is `s_1_1_export_head.txt`*. We will use this as the 'pattern' to match, and check that we have specified the pattern appropriately*

```
> pattern <- "s_1_1_export_head.txt"
> list.files(extdataDir, pattern)

[1] "s_1_1_export_head.txt"
```

Our success shouldn't be too surprising in this case, but it often pays to check. For instance, the pattern above would also match a file with the same name but with `.tar.gz` *appended!*

2.2 `readAligned` and the *AlignedRead* class

The `readAligned` function can be used to input aligned reads. The first argument is a directory path where alignment files are to be found. The second argument is the regular expression to select files to be read. An optional third argument allows the user to specify which type of file is to be read in.

Exercise 5
Use `readAligned` *to read in our abbreviated version of lane 1, specifying* `SolexaExport` *as the type of file.*

```
> aln <- readAligned(extdataDir, pattern, "SolexaExport")
```

See the help page for `readAligned` *for additional details about supported file types.*

What does `readAligned` input? It inputs the short read sequences and base call qualities, and the chromosome, position, and strand information associated with short read alignments. This information is expected to be provided by all short read alignemnt software.

Exercise 6
Display the object we have read in.

```
> aln
```

```
class: AlignedRead
length: 500000 reads; width: 35 cycles
chromosome: 255:255:255 255:255:255 ... QC QC
position: NA NA ... NA NA
strand: NA NA ... NA NA
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

There are 500000 reads in the object, each read consisting of 35 nucleotides. View the first several reads and query information about, e.g., the number of reads that align to each strand, or the number of positions recorded as `NA`.

```
> head(sread(aln))

  A DNAStringSet instance of length 6
    width seq
[1]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[2]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[3]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[4]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[5]    35 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[6]    35 GCAAGTTAAGAAGAGAGCAGAGAAGAACGTTTTTA

> table(strand(aln), useNA="ifany")

     +      -      *   <NA> 
119685 119313      0 261002 

> sum(is.na(position(aln)))

[1] 261002
```

Notice how elements of the `aln` object are extracted using accessors *such as `sread` and `strand`; these are described on the help page for the class of the `aln` object (indicated in the display of `aln`, above, as class AlignedRead); note that the help page refers to the help page for `accessors` to enumerate additional ways of accessing the data.*

What are all the `NA` values returned by `strand` and `position`? These correspond to reads that did not align to the reference genome used by *ELAND*; that about 1/2 the reads do not align is below normal, making this an interesting opportunity for quality assessment. The `strand` function returns a factor with three levels. The first two describe reads aligned to the plus and minus strands, the third (*) is available for successful alignments where strand information is irrelevant.

Aligned reads contain several different kinds of information about 'quality'. Individual bases are assessed for quality during base calling. These 'raw' base qualities are 'calibrated' during *ELAND* alignment; details of calibration are to be found in Illumina documentation. The alignments themselves also have qualities associated with them, with the details of alignment quality differing between alignment algorithms.

Exercise 7
Retrieve calibrated base quality from `aln`*.*

```
> head(quality(aln))

class: SFastqQuality
quality:
  A BStringSet instance of length 6
    width seq
[1]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[2]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[3]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[4]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[5]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[6]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
```

These qualities are string-encoded $-10\log_{10}$ probabilities. The encoding in this case follows a convention established by Illumina. The details of the encoding can be obtained by querying `quality(aln)` *for its* `alphabet`*; the letter* `A` *corresponds to a $-10log_{10}$ score of 1.*

Numeric values are readily retrieved as a matrix, with rows corresponding to reads and columns to cycles. Computations can then be performed on them, e.g., to determine average calibrated quality scores as a function of cycle.

```
> alf <- alphabet(quality(aln))
> m <- as(quality(aln), "matrix")
> colMeans(m)

 [1] 21.45995 19.91676 17.98661 19.67178 18.27315 18.61814
 [7] 18.99522 18.81180 18.69941 18.47509 19.12563 17.39689
[13] 18.26270 18.85640 17.82326 17.43106 17.95288 17.01137
[19] 17.56018 17.73439 16.51199 16.99086 17.80355 16.38983
[25] 17.35305 17.15449 15.99980 16.38890 16.59765 16.18294
[31] 13.48905 13.03266 13.00785 12.75729 11.96461
```

Alignment qualities are accessible with `alignQuality`. This returns an object that can contain quality scores in different formats; to extract the actual quality scores, use `quality`. Reads failing to align or to align in multiple locations have an alignment quality of 0.

Exercise 8
Retrieve the alignment quality scores, determine how many align poorly, and visualize the distribution (Figure 1) of scores.

```
> alignQuality(aln)

class: NumericQuality
quality: 0 0 ... 0 0 (500000 total)

> q <- quality(alignQuality(aln))
> sum(q==0)

[1] 295387
```

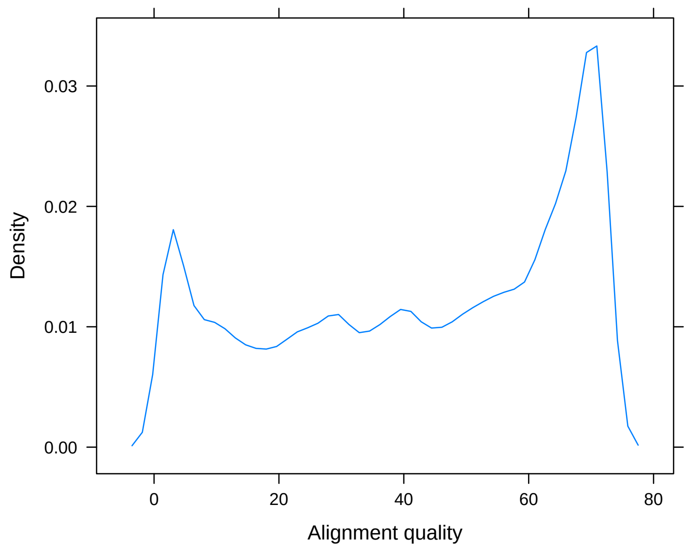

Figure 1: Alignment quality

```
> print(densityplot(q[q>1], plot.points=FALSE,
+                   xlab="Alignment quality"))
```

Alignment algorithms produce information in addition to basic data about chromosome, position, and strand alignment. The exact content varies between algorithms, and is available with `alignData`. `alignData` returns an *AlignedDataFrame* object that contains these data and a metadata description of them. For instance, *ELAND* includes information about whether the read passed a base-calling filter (based on strength and consistency of early bases in the read), in addition to the lane, tile, x and y coordinate of each read.

Exercise 9
Use the `alignData` *function to extract the additional information in the ELAND alignment file. The underlying data in this object can be accessed as though it were a data frame, for instance to tally the number of reads passing Illumina base calling filter.*

```
> alignData(aln)

An object of class "AlignedDataFrame"
  readName: 1, 2, ..., 500000  (500000 total)
  varLabels and varMetadata description:
    run: Analysis pipeline run
    lane: Flow cell lane
    ...: ...
    contig: Contig
    (7 total)
```

```
> table(alignData(aln)$filtering)

     Y      N
287222 212778
```

2.3 Subsets and filters

A very common operation is to reduce the number of reads used for subsequent analysis. This can be done in a coordinated fashion by creating a subset of `aln`.

Exercise 10
Select just the aligned reads passing Illumina filtering, and aligning to the reference genome.

```
> filtIdx <- alignData(aln)$filtering=="Y"
> alignedIdx <- !is.na(strand(aln))
> aln[filtIdx & alignedIdx]

class: AlignedRead
length: 197432 reads; width: 35 cycles
chromosome: chr11.fa chr9.fa ... chr8.fa chr4.fa
position: 104853312 3036336 ... 44295163 47191474
strand: - - ... - -
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

A different approach to subsetting is to use objects of class *SRFilter*. These can be particularly useful as an argument to `readAligned`, in addition to use in interactive sessions.

Exercise 11
Construct instances of built-in filters to select reads passing the Illumina filtering criterion, and uniquely aligning to a fully assembled chromosomes. These can be 'composed' into a single overall filter, and applied to restrict available reads.

```
> filt1 <- alignDataFilter(expression(filtering=="Y"))
> filt2 <- chromosomeFilter("chr[0-9XYM]+.fa")
> filt <- compose(filt1, filt2)
> caln <- aln[filt(aln)]
> caln

class: AlignedRead
length: 195719 reads; width: 35 cycles
chromosome: chr11.fa chr9.fa ... chr8.fa chr4.fa
position: 104853312 3036336 ... 44295163 47191474
strand: - - ... - -
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

The filters developed above could be used to filter reads while being read in to *R*, e.g,. with

```
> readAligned(extdataDir, pattern, filter=filt)
```

The `srFilter` function can be used to create custom filters. The idea is that filter functions accept a single argument `x` that is an object to be filtered, and returns a logical vector that can be used to select elements of the object.

Exercise 12
As a first example, write and use a filter to select only a single read from all that align to a particular chromosome, position, and strand.

```
> ualignFilter <- srFilter(function(x) {
+     ## create a numerical index of reads. Divide the index,
+     ## position, and strand information between
+     ## chromosomes. Select the index of a single read at each
+     ## unique position and strand. Return the selected index as a
+     ## logical vector with the same length as x
+     oindex <- seq_len(length(x))
+     index <- tapply(oindex, chromosome(x), c)
+     pdup <- tapply(position(x), chromosome(x), duplicated)
+     sdup <- tapply(strand(x), chromosome(x), duplicated)
+     keep <- oindex  %in% unlist(mapply(function(i, p, s) {
+         i[!(p & s)]
+     }, index, pdup, sdup))
+ }, name="select only one read per position & strand ")
> caln[ualignFilter(caln)]

class: AlignedRead
length: 188219 reads; width: 35 cycles
chromosome: chr11.fa chr9.fa ... chr8.fa chr4.fa
position: 104853312 3036336 ... 44295163 47191474
strand: - - ... - -
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

The filter functions built-in to *ShortRead* use a 'factory' pattern to create instances of each filter that 'remember' how the filters were created. For instance, `chromosomeFilter("chr2.fa")` creates an instance of the chromosome filter to select only chromosomes matching `chr2.fa`.

Exercise 13
As an advanced example, the following filter subsamples a (user-specified) number of reads. The `samplingFilter` *function uses the factory pattern, so filters created with it remember how many reads to sample.*

```
> samplingFilter <- function(sampleSize) {
+     srFilter(function(x) {
+         idx <- seq_len(length(x))
+         idx %in% sample(idx, sampleSize)
+     }, name="Demo sampling filter")
+ }
> sample100 <- samplingFilter(100)
> caln[sample100(caln)]
```

```
class: AlignedRead
length: 100 reads; width: 35 cycles
chromosome: chr18.fa chr12.fa ... chr10.fa chr9.fa
position: 17466268 4595481 ... 16042762 46614794
strand: - + ... + -
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

2.4 Cautions

There are several confusing areas associated with reading data aligned with various software packages. (1) Some alignment programs and genome resources start numbering nucleotides of the subject sequence at 0, whereas others start at 1. (2) Some alignment programs report matches on the minus strand in terms of the 'left-most' position of the read (i.e., the location of the 3' end of the aligned read), whereas other report 'five-prime' matches (i.e., in terms of the 5' end of the read), regardless of whether the alignment is on the plus or minus strand. (3) Some alignment programs reverse complement the sequence of reads aligned to the minus strand. (4) Base qualities are sometimes encoded as character strings, but the encoding differs between 'fastq' and 'solexa fastq'. It seems that all combinations of these choices are common 'in the wild'.

The help page for `readAligned` attempts to be explicit about how reads are formatted. Briefly:

- Subject sequence nucleotides are numbered starting at 1, rather than zero. `readAligned` adjusts the coordinate system of input reads if necessary (e.g., when reading MAQ alignments).
- ELAND and Bowtie alignments on the minus strand are reported in 'left-most' coordinates systems.
- ELAND and Bowtie alignments on the minus strand are not reverse complemented.
- Character-encoded base quality scores are interpreted as the default for the software package whose output is being parsed, e.g., as 'Solexa fastq' for *ELAND*. The object returned by `quality` applied to an *AlignedRead* object is either *FastqQuality* or *SFastqQuality*.

Alignment programs sometimes offer the opportunity to customize output; such customization needs to be accommodated when reads are input using *ShortRead*.

3 Additional input functions

Many sequence alignment programs are producing 'bam' (Binary AlignMent) files. These can be read in to *R* using `readAligned` with `type="BAM"`. `readAligned` filters reads to include only those reads aligned without indels; more flexible representation is available with `readGappedAlignments`, which inputs the reads and their 'cigar' representation. A cigar is a short run length encoding providing information about how the sequence of nucleotides in the read align to the reference, including information about indels. Even greater flexiblity is provided by `scanBam` function in the *Rsamtools* package. This will parse *BAM* files that are either local or remote (e.g., from the 1000 genomes project), and can be used to efficiently select reads satisfying particular criteria, e.g., overlapping a particular range or aligned to the minus strand. See the help page for `scanBam` and `ScanBamParam` for additional information.

ShortRead, *Biostrings*, and the standard input functions from *R* provide additional tools for reading Illumina and other alignment formats. The `readXStringColumns` function provides a convenient way to read DNA and quality sequences into compact data structures. `readFasta` and its counterpart `readFastq` provide tools for reading FASTA- or FASTQ- (i.e., including quality annotation) formatted files.

Exercise 14
Files `_sequence.txt` contain fastq-formatted sequence and quality scores. A sample of this file type is available. Read these in to data structured defined in ShortRead. The content of `reads` can be retrieved with the accessor functions `id`, `sread`, and `quality`.

```
> reads <- readFastq(extdataDir, "s_1_1_sequence_head.txt$")
> head(id(reads))

  A BStringSet instance of length 6
    width seq
[1]    24 HWI-EAS88_3:1:1:33:484/1
[2]    24 HWI-EAS88_3:1:1:33:272/1
[3]    24 HWI-EAS88_3:1:1:31:594/1
[4]    24 HWI-EAS88_3:1:1:33:383/1
[5]    24 HWI-EAS88_3:1:1:35:216/1
[6]    25 HWI-EAS88_3:1:1:883:458/1
```

The `readPrb` functions reads 'raw' base call quality scores from `_prb` files in the `baseCallPath` directory; the result is a *BStringSet* object that compactly represents the quality scores in a way analogous to the results of `quality` applied to `aln`.

Exercise 15
The `_export.txt` files read by `readAligned` are tab-delimited text files. The goal of this exercise is to read the DNA sequence and quality score columns in to R as DNAStringSet and BStringSet objects. The DNAStringSet and BStringSet classes represent DNA or 'biological' strings; they extend the base class XString.

Start by parsing the first line of a `_prb` file to get a sense of its content. Specify the `colClasses` to be imported, using `NULL` to indicate that a column should be skipped, and `DNAString` or `BString` to indicate columns that are to be read as the corresponding data types (classes). Read the file with `readXStringColumns`.

```
> fl <- list.files(extdataDir, pattern, full=TRUE)
> cols <- strsplit(readLines(fl, 1), "\t")[[1]]
> length(cols)

[1] 22
```

```
> cols[9:10]

[1] "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
[2] "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU"

> colClasses <- rep(list(NULL), 22)
> colClasses[9:10] <- c("DNAString", "BString")
> strings <-
+     readXStringColumns(extdataDir, pattern, colClasses=colClasses)
> head(strings[[2]])

  A BStringSet instance of length 6
    width seq
[1]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[2]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[3]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[4]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[5]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
[6]    35 ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZUUUUU
```

For each column in `colClasses`, `readXStringColumns` *parses the corresponding column in all files matching the function argument* `pattern` *into a single XStringSet, i.e., concatenating the content of all files into a single object.*

4 Quality assessment

This part of the course addresses *ShortRead* facilities for assessing quality, primarily of Solexa data. The *ShortRead* functionality is mean to complement rather than replace QA tools provided by the *ELAND* pipeline.

4.1 Generating a QA report

.

Creating a QA report is a two-step process. The first step is to visit necessary files to collate information in a compact representation. The second step is to present the information in a useful format.

The `qa` function collates information for the QA report. It visits each `_export.txt` file, and extracts information on reads and their qualities. Evaluation of the function is straight-forward, e.g., `qa <- qa(extdataDir, ".*_export.txt", type="SolexaExport")`. The process of collating files can be time consuming (each export file must be parsed, taking 3-4 minutes per file) and memory intensive (lanes are processed independently of one another, but processing a full lane consumes 2-3 GB of memory). The return value of the `qa` function is actually quite compact, and easy to work with.

Exercise 16
Rather than collating information during the lab, we load the data from a previously stored instance.

```
> data("qa_080828_081110", package="EMBL2010")
> qa
```

```
class: SolexaExportQA(9)
QA elements (access with qa[["elt"]]):
  readCounts: data.frame(8 3)
  baseCalls: data.frame(8 5)
  readQualityScore: data.frame(12288 4)
  baseQuality: data.frame(752 3)
  alignQuality: data.frame(629 3)
  frequentSequences: data.frame(1200 4)
  sequenceDistribution: data.frame(4540 4)
  perCycle: list(2)
    baseCall: data.frame(1400 4)
    quality: data.frame(6664 5)
  perTile: list(2)
    readCounts: data.frame(7200 4)
    medianReadQualityScore: data.frame(7200 4)
```

One feature of *ShortRead* that can speed up this stage of the operation is the use of clustered computer resources and the *Rmpi* or *mullticore* package; `qa` uses the `srapply` function to automatically detect and distribute collation tasks across pre-established nodes. This is outlined in more detail in a subsequent section.

The QA information collated from the `_export.txt` files is summarized into an html report using the `report` function. The command to create the report is `rpt <- report(qa, dest=tempfile())`. This creates an HTML file at the location specified by the argument `dest`.

Exercise 17
Create a report from the `qa` *object loaded in the previous exercise. Do this with the command*

```
> rpt <- report(qa, dest=tempfile())
```

View the report in your browser with

```
> browseURL(rpt)
```

The QA report provides summary statistics about the numbers of reads and alignments, base calls and qualities, characteristics of per-lane and per-tile read quality, and other information. The QA report is self-documenting, providing a narrative description of each section.

4.2 Exploring `qa`

The `qa` object is a list-like structure with several entities.

Exercise 18
The `readCounts` *element of* `qa` *is a simple data frame summarizing, on a per-lane bases, the total number of reads, the number of reads passing Solexa internal filtering, and the number of aligned reads.*

```
> qa[["readCounts"]]
```

```
                     read filtered aligned
s_1_1_export.txt 3668433  2278945 1910666
s_2_1_export.txt 4230424  3239956 2771169
s_3_1_export.txt 4003465  3089375 1720396
s_4_1_export.txt 4521919  3446177 2571235
s_6_1_export.txt 4004807  3127297 1985855
s_7_1_export.txt 3546869  2732974 1368590
s_8_1_export.txt 4232977  3291627 2379709
s_5_1_export.txt 2842633  2399387 2320140
```

Lanes 1-4 and 6-8 correspond to biologically interesting samples; lane 5 is the Solexa φX-174 control lane. The number of reads (between 2.8 and 4.5 million) is low for a typical experiment (official guidelines are provided in Solexa documentation).

It can be difficult to scan large numbers, so the QA report template defines functions that help to display the information in a more comprehensible fashion. Source these files into your current R session, and read the second and third columns as a proportion of the first.

```
> ShortRead:::.ppnCount(qa[["readCounts"]])
```

```
                     read filtered aligned
s_1_1_export.txt 3668433    0.621   0.521
s_2_1_export.txt 4230424    0.766   0.655
s_3_1_export.txt 4003465    0.772   0.430
s_4_1_export.txt 4521919    0.762   0.569
s_6_1_export.txt 4004807    0.781   0.496
s_7_1_export.txt 3546869    0.771   0.386
s_8_1_export.txt 4232977    0.778   0.562
s_5_1_export.txt 2842633    0.844   0.816
```

Refer to the QA report for further commentary on this and other aspects of the report

4.3 Frequent sequences

A feature of raw reads, and of many subsequent stages of short read analysis, is a power law-like relationship between the number of times a read occurs, and the number of occurrences of a particular sequence in the sample. This information is contained in the `sequenceDistribution` element of `qa`, and is the result of running the `tables` command (defined in ShortRead) on a *DNAStringSet* object.

Exercise 19
Retrieve the `sequenceDistribution` *element from* `qa`*; it is a simple data frame. Look at the contents of the data frame using* `head`*, and select just lane 5 raw reads. plot the power-law relationship between the number of reads and number of times reads occur. As an alternative display of the same information, plot the cumulative number of reads as a function of the number of times a read occurs.*

```
> df <- qa[["sequenceDistribution"]]
> df5raw <- df[df$lane=="s_5_1_export.txt" & df$type=="read",]
> head(df5raw)
```

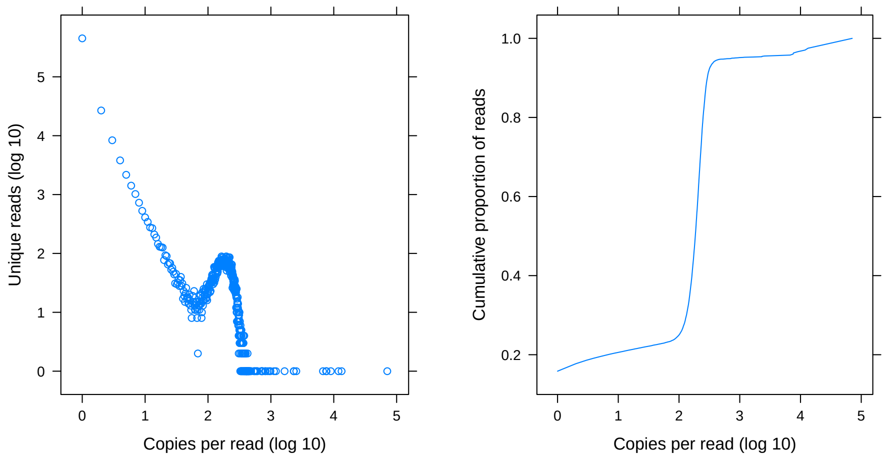

Figure 2: Number of copies of unique reads

```
     nOccurrences nReads type             lane
3337            1 450124 read s_5_1_export.txt
3338            2  26810 read s_5_1_export.txt
3339            3   8366 read s_5_1_export.txt
3340            4   3806 read s_5_1_export.txt
3341            5   2166 read s_5_1_export.txt
3342            6   1417 read s_5_1_export.txt
```

```
> print(xyplot(log10(nReads)~log10(nOccurrences), df5raw,
+              xlab="Copies per read (log 10)",
+              ylab="Unique reads (log 10)"))
```

```
> csum <- with(df5raw, cumsum(nReads * nOccurrences))
> csum <- csum / csum[length(csum)]
> print(xyplot(csum ~log10(nOccurrences), df5raw,
+              xlab="Copies per read (log 10)",
+              ylab="Cumulative proportion of reads",
+              type="l"))
```

Results appear in Figure 2. The cumulative form of this figure appears in the QA report.

The power-law relationship between copies per read and number of unique reads in the control lane consists of three components. At the left of the graph are $> 10^5$ reads that are each represented by only one copy. These likely correspond to sequencing, base calling, or other errors associated with the technology. At the right of the figure are a small number of reads represented many times. These 'frequent' sequences are summarized in the `frequentSequences` element of `qa`; frequent sequences are also reported by the `tables` function of *ShortRead*.

Exercise 20
Discover the frequent sequences amongst the raw and aligned reads of lane 5.

```
> df <- qa[["frequentSequences"]]
> head(df[df$lane=="s_5_1_export.txt" & df$type=="read",1:2])

                                 sequence count
1051 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 70947
1052 ANNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 13320
1053 TNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 11892
1054 CNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  8978
1055 GNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  7670
1056 GATCTTTGGCGGCACGGAGCCGCGCATCACCTGTA  7561
```

Frequent sequences include poly-A reads, reads where only a few bases were called, and reads with close similarity to the Solexa primer or adapter sequence used in sample preparation.

Exercise 21
Many of the primer sequences are filtered out by Solexa criteria, but it is worth discovering how many reads are 'similar' to this sequence. Use the `srdistance` *function to identify such reads, e.g., amongst those reads that contain no* `N` *nucleotides.*

```
> seq <- "CGGTTCAGCAGGAATGCCGAGATCGGAAGAGCGGT"
> dist <- srdistance(clean(aln), seq)[[1]]
> head(table(dist))

dist
  0   1   2   3   4   5
218 107  59  31  25  17
```

`srdistance` *returns the edit distance between each read and the reference sequence, where the edit distance is defined so that each base mismatch represents an additional increment of 1. There are 384 reads that differ at 2 or fewer locations, from amongst the 370049 reads in the cleaned sample used in this exercise.*

Reads represented many times may be problematic for downstream analysis. For instance, sample preparation protocols may involve a PCR step that results in differential amplification, whereas the analysis assumes reads are represented in proportion to their occurrence. The `ualignFilter` function defined above, and the `srduplicated` function in *ShortRead* represent two approaches to dealing with this problem by ensuring that reads are represented exactly once (by some definition of 'once'!).

Sequences in the middle portion of the graph in Figure 2 will often, depending on the nature of the investigation, represent the sequences of main biological interest. These are sequences represented an intermediate number of times, as might be required for reasonable coverage in a SNP discovery or ChIP-seq experiment. The right-hand graph in Figure 2 shows a relatively abrupt transition between reads represented rarely and those represented many times.

The sample QA report shows that the non-control lanes show much broader transitions from rarely to frequently represented reads. This could represent technical shortcomings of this run (e.g., inadequate enrichment of sample DNA) or features of intrinsic biological interest (e.g., wide variability in ChIP abundance between binding sites). Regardless of ultimate source, the broad distribution of read occurrences implies some effort may be required to distinguish 'noise' (reads corresponding to those in the left and right portions of the control lane graph) from signal.

Finally, while the control lane shows a relatively abrupt transition between reads that occur rarely and those that are common (Figure 2), the distribution is in fact 3- or 4-fold broader than expected under a naive model of random read starts along the φX-174 genome. This is reinforced by alignments to the reference genome, where clear patterns (e.g., unequal representation on plus and minus strands; non-uniform coverage) are apparent.

Exercise 22
As an advanced exercise, simulate reads selected uniformly along both strands of the 5200bp long φX-174 genome. Compare the times each read is represented in your sample with those from the actual control lane. Hint: use `sample` *with* `replace=TRUE` *to generate the reads, and* `table(table(reads))` *to summarize their occurrence; this should take less than 5 lines of R code.*

4.4 Cycle-specific qualities and base calls

As a final foray into the details of quality assessment, consider base calls and quality, and how these change across cycles (see the second table and Section 4 of the QA report).

The table in the QA report suggests that the control lane (lane 5) is enriched for A and T; this is confirmed by the figure in section 4. This likely reflects underlying differences in the genomic regions represented in each lane.

Exercise 23
Use `alphabetFrequency` *to summarize nucleotide use in the short reads in* `aln`*, from the first part of this lab.*

```
> alphabetFrequency(sread(aln), collapse=TRUE, baseOnly=TRUE,
+                   as.prob=TRUE)

         A          C          G          T      other
0.25588029 0.22092086 0.20225720 0.22800937 0.09293229
```

The frequency of 'other' (i.e., uncalled) nucleotides ($> 9\%$) is very high; typical runs are $< 1\%$.

An unexpected aspect of the figure in Section 4 of the QA report is apparent trends in nucleotide frequency with cycle. For instance, all lanes show a marked decrease in A and increase in C across cycles. This is unexpected in this experiment, where the *a priori* expectation is that sequences start at essentially arbitrary locations in the sequenced DNA: an A is expected as frequently at the beginning of the sequence as at the end.

Exercise 24
Use `alphabetByCycle` *to extract the number of each nucleotide A, C, T, G, sequenced at each cycle. Plot the transpose of the matrix using* `matplot`*, to see how nucleotide counts change across cycles.*

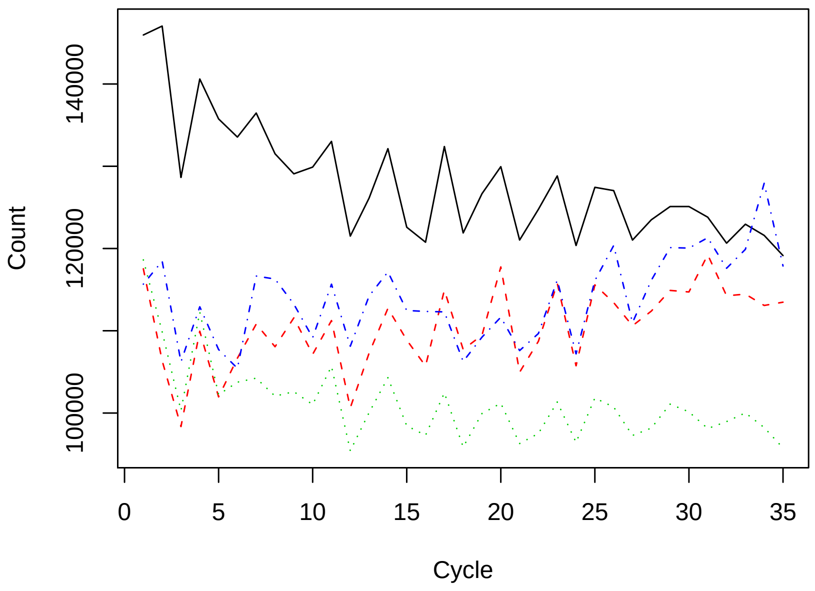

Figure 3: Nucleotide frequency per cycle, lane 1

```
> abc <- alphabetByCycle(sread(aln))
> dim(abc)

[1] 17 35

> abc[1:4,1:5]

        cycle
alphabet   [,1]   [,2]   [,3]   [,4]   [,5]
       A 145955 147044 128640 140609 135751
       C 117538 106357  98356 109948 101961
       G 118619 109879 100303 112267 102233
       T 115666 118470 106223 112917 107716

> matplot(t(abc[1:4,]), type="l", xlab="Cycle", ylab="Count")
```

There are a number of possible contributors to cycle-dependent nucleotide frequencies, including inadequate reagent volume, and nucleotide-specific differential accumulation of fluorescent dyes. Figure 3 and the figure in Section 4 of the report contain additional features that are moderately unexpected, and unexplained. For instance, the frequency of a nucleotide such as A changes very systematically across cycles, first increasing and then decreasing; this seems more regular than expected. Patterns of nucleotide change also seem to echo one another at similar cycles but in different lanes, even when the lanes have different biological material. This occurs for instance in lanes 1-4 of the QA report, where the last 5 cycles of the C nucleotide seem to change in (comparative!) unison.

5 Coverage

Many work flows invovle some aspect of coverage, the depth to which individual nucleotides in a reference sequence are covered by nucleotides in reads.

Exercise 25
Select just those reads that aligned to the reference genome (e.g., using `!is.na(position(aln))`). Use the `coverage` function to summarize how reads are aligned to chromosomes

```
> cvg <- coverage(aln[!is.na(position(aln))])
```

The `coverage` function returns a list-like structure, where each element of the list represents a chromosome. The data are represented as a 'run length encoded' (*Rle*) instance. *Rle* instances are readily interogatted for a variety of useful insights.

Exercise 26
Select an element of `cvg` for further investigation. Use (slice) to identify all regions where there are at least 2 reads. Use the `width` accessor and `densityplot` function to display the distribution of island widths. Use `viewMaxs` and `desnityplot` to display the distribution of maximum island depths.

```
> slc <- slice(cvg[["chr1.fa"]], 2)
> densityplot(log10(width(slc)))
> densityplot(viewMaxs(slc))
```

There are many additional exploratory opportunities associated with coverage vectors; we will encounter some of these in ChIP-seq processing. As a final exercise. . .

Exercise 27
The rtracklayer package can export `Rle` and other objects to a format such as gff suitable for viewing in genome browsers. Use rtracklayer and its `export` function to export a description of the peaks.

```
> library(rtracklayer)
> fl <- tempfile()
> export(cvg[["chr1.fa"]], fl, "gff3")
```

6 Summary

This portion of the course has provided you with an overview of the input, manipulation, and quality assessment functionality available in the *ShortRead* and other packages. A summary of the insights learned might be reflected in simple, and somewhat naive, work flows.

The first work flow simply performs quality assessment on *ELAND* aligned data.

```
> rpt <- report(qa(extdataDir, "_export.txt"),
+               dest="reports/my_report.pdf")
```

Performing QA on *ELAND* data does not commit us to using *ELAND* alignments in subsequent steps.

The second work flow reads aligned data in to *R*. The work flow might start with aligned reads created by one of many aligners; we start with *ELAND* `_export.txt` files. There are many possible issues highlighted in the forgoing discussion. We choose to establish a series of filters to eliminate some reads at the very start of our work flow. We eliminate reads with ambiguous base calls and failing Solexa's internal filtering criteria. Reads aligning to multiple locations in the genome are not straight-forward to deal with, and are not essential for ChIP-seq style experiments (this is not necessarily the case for expression or RNA-seq experiments), so we remove these. Most close matches to the Solexa primer sequence are flagged as not passing Solexa base call filters, but we eliminate reads near to this as well. Finally, we restrict our attention to those reads that align to assembled nuclear chromosomes, putting aside for the moment those reads aligning to organelle genomes (the X and Y chromosomes also require special consideration, and we might often eliminate these from a first-pass work flow, too). Our second work flow is thus:

```
> filt1 <- nFilter()
> filt2 <- alignDataFilter(expression(filtering=="Y"))
> filt3 <- alignQualityFilter(threshold=1)
> filt4 <- srdistanceFilter("CGGTTCAGCAGGAATGCCGAGATCGGAAGAGCGGT", 4)
> filt5 <- chromosomeFilter("chr[0-9]+.fa")
> filt <- compose(filt1, filt2, filt3, filt4, filt5)
> aln <- readAligned(extdataDir, pattern, filter=filt)
```

This work flow applies equally to MAQ aligned data, with the exception that Solexa filtering criteria are not available and the chromosome naming convention in the MAQ-aligned reads in our sample are different:

```
> maqDir <- "path/to/maq"
> filt5 <- chromosomeFilter("chr[0-9XY]+$")
> filt <- compose(filt1, filt3, filt4, filt5)
> maq <- readAligned(maqDir, "s_8.map", "MAQMap", filter=filt)
```

Each of the filters represents a decision. The decision may be inappropriate for particular analyses, and may be revisited as understanding of the data matures.

The *GappedAlignments* class and the *GenomicRanges* and *Rsamtools* packages provide comparable facilities for input and manipulation of reads aligned with indels.

7 Session information

- R version 2.11.1 Patched (2010-05-31 r52167), `i386-apple-darwin9.8.0`
- Locale: `C/C/C/C/C/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, stats, tools, utils

- Other packages: AnnotationDbi 1.10.1, BSgenome 1.16.5, BSgenome.Hsapiens.UCSC.hg19 1.3.16, BSgenome.Scerevisiae.UCSC.sacCer2 1.3.16, Biobase 2.8.0, Biostrings 2.16.6, EMBL2010 0.0.11, EatonEtAlChIPseq 0.0.1, GenomicFeatures 1.0.3, GenomicRanges 1.0.5, IRanges 1.6.8, RCurl 1.4-2, Rsamtools 1.0.5, SNPlocs.Hsapiens.dbSNP.20090506 0.99.1, ShortRead 1.6.2, bitops 1.0-4.1, chipseq 0.4.0, hgu95av2probe 2.6.0, lattice 0.18-8, rtracklayer 1.8.1
- Loaded via a namespace (and not attached): DBI 0.2-5, RSQLite 0.9-1, XML 3.1-0, biomaRt 2.4.0, grid 2.11.1, hwriter 1.2