# Counting and measuring aligned sequences

Martin Morgan*

July , 2011

**Abstract**

This workshop is about effectively using the *Rsamtools* *Bioconductor* package for exploration and initial analysis of high-throughput sequence experiments. We investigate features such as: input of aligned reads; querying regions of interest across many BAM files; extracting coverage and nucleotide-level summaries; and querying indexed FASTA reference sequences. The workshop touches on related infrastructure from *GenomicRanges*, *ShortRead*, and other packages, and uses as motivating examples subsets of in-house RNAseq, 1000 genomes, and Complete Genomics data sets. The emphasis is on leading-edge features; attendees should be prepared with the 'development' version of *R* and current *Bioconductor* packages.

## 1 Basics

**Exercise 1**
*This exercise is to ensure that software required for the workshop is installed. Start R and load the following packages:*

- *Rsamtools*
- *leeBamViews*
- *org.Sc.sgd*
- *TxDb.Scerevisiae.UCSC.sacCer2.ensGene*

**Solution:**

```
> library(Rsamtools)
> library(leeBamViews)
> library(org.Sc.sgd.db)
> library(TxDb.Scerevisiae.UCSC.sacCer2.ensGene)
```

*mtmorgan@fhcrc.org

**Exercise 2**

*This exercise introduces BAM files and the BamFile and BamFileList classes, including the use of element metadata to track information about samples.*

*Use the command* `system.file` *and* `dir` *to list all the BAM files in the* `bam` *directory of the leeBamViews package.*

*Create a character vector of the paths to the files, and name the character vector. One naming scheme uses* `basename` *to get the name of the file itself, and* `sub` *to remove non-discriminating trailing characters.*

*Organize the file names into a BamFileList instance.*

*Add element metadata (*`values`*) to the list of bam files. Do this by creating a DataFrame with columns corresponding to the genotype and lane (extracted from the file names) of each file. Can you create the data frame in a way that minimizes opportunities for mis-labeling lanes?*

*Finally,* `open` *the bam files. What is the rationale for opening the bam files once?*

**Solution:**

```
> fls <- dir(system.file("bam", package="leeBamViews"),
+            "bam$", full=TRUE)
> names(fls) <- sub("_13e.bam$", "", basename(fls))
> bam <- BamFileList(fls)
> regex <- "([[:alpha:]]+)([[:digit:]])"
> values(bam) <-
+     DataFrame(Genotype=factor(sub(regex, "\\1", names(bam))),
+               Lane=sub(regex, "\\2", names(bam)))
> open(bam)

BamFileList of length 8
names(8): isowt5 isowt6 rlp5 rlp6 ssr1 ssr2 xrn1 xrn2
```

**Exercise 3**

*This exercise introduces the GappedAlignments class, and a simple interface to bam files for retrieving information about aligned reads.*

*Extract the first element in the BamFileList created in the previous exercise. Use this element and the* `readBamGappedAlignments` *function to read in all reads in the BAM file. Explore the resulting object.*

*The bam files are a subset of the study; use* `table` *on the GappedAlignments class to determine which reference sequence names the reads we have access to came from. How many reads align to the positive and to the negative strand? What's a cigar? Are there any gaps in these aligned reads?*

*What does the* `keepSeqlevels` *function do? Why might it be useful?*

*The org.Sc.sgd.db annotation package contains objects that relate gene symbols to information about them, e.g., the* `org.Sc.sgd.CHRLOC` *object relates the SGD identifier (e.g.,* `YMR297W` *to the chromosome and location at which the*

*corresponding gene starts. Likewise,* `org.Sc.sgd.CHRLOCEND` *maps the SGD identifier to the genomic location where the gene ends. Use these objects and create a* `GRanges` *object that specifies the genomic coordinates of* `YMR297W`*. We'll call this range the 'region of interest'.*

*Use this the* `GRanges` *object created in the previous paragraph as the* `which` *argument to* `readBamGappedAlignments` *to read in just those reads in the region of interest.*

**Solution:**

```
> aln <- readBamGappedAlignments(bam[[1]])
> aln <- keepSeqlevels(aln, "Scchr13")
> seqlevels(bam[[1]])

 [1] "Scchr01" "Scchr02" "Scchr03" "Scchr04" "Scchr05" "Scchr06"
 [7] "Scchr07" "Scchr08" "Scchr09" "Scchr10" "Scchr11" "Scchr12"
[13] "Scchr13" "Scchr14" "Scchr15" "Scchr16" "Scmito"

> start <- org.Sc.sgdCHRLOC[["YMR297W"]]
> end <- org.Sc.sgdCHRLOCEND[["YMR297W"]]
> which <- GRanges("Scchr13", IRanges(start, end))
> aln1 <- readBamGappedAlignments(bam[[1]], which=which)
```

**Exercise 4**

*The package* TxDb.Scerevisiae.UCSC.sacCer2.ensGene *contains structural information about genes (e.g., exon boundaries, organization of exons into transcripts and genes). Create a convenient reference* `txdb` *to the data base* `Scerevisiae_UCSC_sacCer2_ensGene_TxDb` *present in this package.*

*Use the* `exonsBy` *function to extract the exons of in yeast, organized by gene. In particular, select the exons corresponding to our region of interest,* `YMR297W`*.*

*The* `countOverlaps` *function is an efficient way to count the number of times a* `query` *(e.g., the aligned reads) overlap one or more* `subjects` *(e.g., the regions of interest). Use* `countOverlaps` *function to determine the number of times each alignment overlaps exons in the region of interest.*

*There are several issues you'll need to overcome: use* `renameSeqlevels` *to make the sequence names of the alignments appropriate for the sequence names used to denote exons (why are these names different?). The protocol used in this experiment created reads that align to either strand, regardless of the strand from which the gene that they are derived was coded on. Use the* `ignore.strand` *argument to allow alignments to either strand.*

*Use* `table` *to summarize the number of times reads align to the region of interest.*

**Solution:**

```
> txdb <- Scerevisiae_UCSC_sacCer2_ensGene_TxDb
> ex <- exonsBy(txdb, "gene")
> ex1 <- ex[["YMR297W"]]
> aln2 <- renameSeqlevels(aln, list(Scchr13="chrXIII"))
> cnt <- countOverlaps(aln2, ex1, ignore.strand=TRUE)
> table(cnt)

cnt
    0     1
20369 14180
```

**Exercise 5**
*Explore the `coverage` function.*

*Start by restricting attention to our region of interest, using `keepSeqlevels` and `renameSeqlevels` to bring the sequence names in the gene model and the alignments into line with the sequence names in the BAM files.*

*Use the `coverage` function on the alignments to summarize how many reads cover each nucleotide. Use the `shift` and `width` arguments to restrict attention to our region of interest. What is a run-length encoding? Why might it be used? to represent coverage? What kinds of operations can be performed on a run length encoding?*

*Now, write a function that takes as arguments a bam file ana an argument `which` specifying the region of interest. The function should (a) input a `GappedAlignments` object; (b) keep only the sequence levels corresponding to chromosome 13; (c) calculate coverage in the region of interest; and (d) converts the run-length encoding of coverage to an integer vector.*

*Use `sapply` to apply the function you've just written to each bam file, for our region of interest.*

*The result is a matrix (what are its dimensions?); plot it using `matplot`, with `type="l"`. Can you arrange for each genotype to be plotted in the same color?*

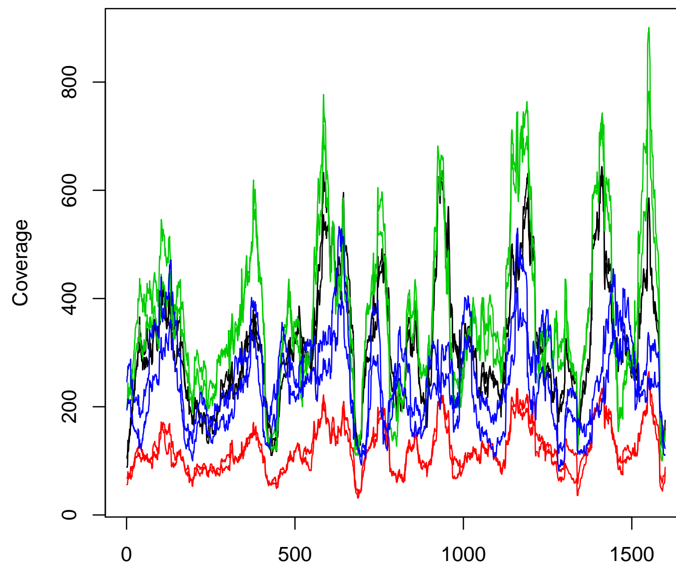
**Solution:**

```
> ex2 <- renameSeqlevels(keepSeqlevels(ex1, "chrXIII"),
+                        list(chrXIII="Scchr13"))
> aln1 <- keepSeqlevels(aln1, "Scchr13")
> coverage(aln1, shift=-start(ex2) + 1L, width(ex2))

SimpleRleList of length 1
$Scchr13
'integer' Rle of length 1599 with 1503 runs
  Lengths:   2   1   1   1   1   2   2 ...   1   1   1   2   1   1
  Values :  88  93 121 117 119 134 147 ... 141 142 144 145 149 159

> cvg <- function(bam, which) {
+     aln <- readBamGappedAlignments(bam, which)
```

```
+      aln <- keepSeqlevels(aln, seqlevels(which))
+      cvg <- coverage(aln, shift=-start(which) + 1L, width(which))
+      as.integer(cvg)
+ }
> Coverage <- sapply(open(bam), cvg, ex2)
> matplot(Coverage, type="l", col=values(bam)$Genotype, lty=1)
```




## 2 GC Content

**NOTE**: This section requires files that are not available during the workshop; it is meant as a demo rather than series of exercises.

The *Rsamtools* contains several useful functions, in addition to `readBamGappedAlignments`. Some of these (e.g., `scanBam` are described in detail in the vignette (try `browseVignettes('Rsamtools')`) that comes with the package.

The `indexFa` function allows one to create an index into a fasta file containing multiple sequences. These sequences, including arbitrary ranges within the sequences, can the be input using `scanFa`. Here we get the coordinates of genes (ignoring for simplicity the exon structure of the gene) on human chromosomes 8 and 9, using the hg18 build.

```
> ## genes on chr 8 or 9
> library(Rsamtools)
> library(TxDb.Hsapiens.UCSC.hg18.knownGene)
> txdb <- Hsapiens_UCSC_hg18_knownGene_TxDb
> ex <- exonsBy(txdb, "gene")
> ex89 <- range(keepSeqlevels(ex, c("chr8", "chr9")))
> gn <- unlist(ex89[elementLengths(ex89) == 1], use.names=FALSE)
> names(gn) <- names(ex89[elementLengths(ex89) == 1])
```

Next we create an index into a file containing the complete sequences of chromosomes 8 and 9 from the same build (the index only needs to be created once), and query the index for our regions of interest. From this we obtain a `DNAStringSet` of sequences in our regions of interest.

```
> ## seq and GC content
> ## preparation
> indexFa("chr8-9.fa")
> seq <- scanFa("chr8-9.fa", gn)
```

Calculate the GC content of each reference sequence using `alphabetFrequency`.

```
> alph <- alphabetFrequency(seq, as.prob=TRUE)
> gc <- rowSums(alph[,c("G", "C")])
```

Now, create a list of BAM files referencing 64 different samples, adding metadata to describe the samples in a way analogous to that done above.

```
> ## bam files
> dir0 <- "/mnt/cpl/data/Solexa/SOC/101101/Samples/"
> dir <- dir(dir0, "Benign|Normal|SOC", full=TRUE)
> fls <- list.files(dir, "*bam$", recursive=TRUE, full=TRUE)
> names(fls) <- sub(".*tophat_([^/]+).*", "\\1", fls)
> bam <- BamFileList(fls)
> re <- "^([[:alnum:]]+)_+([[:alnum:]]+).*"
> values(bam) <- DataFrame(Trt=sub(re, "\\1", names(bam)),
+                          Id=sub(re, "\\2", names(bam)))
```

To start, use `readGappedReads` to input aligned reads from a single bam file in our region of interest. `readGappedReads` includes the sequences, as well as the coordinates of the alignments. Use `countOverlaps` to count the number of single-hit alignments overlapping each gene. Plot the relationship between gc content and counts per gene, e.g., as a box-and-whiskers plot.

```
> ## reads overlapping genes
> library(ShortRead)
> aln <- readGappedReads(path(bam[[1]]), which=gn)
> hits <- countOverlaps(aln, gn, ignore.strand=TRUE)
> cnt <- countOverlaps(gn, aln[hits==1], ignore.strand=TRUE)
> (bwplot(log(1+cnt)~cut(gc, 30), scales=list(x=list(rot=90))))
```

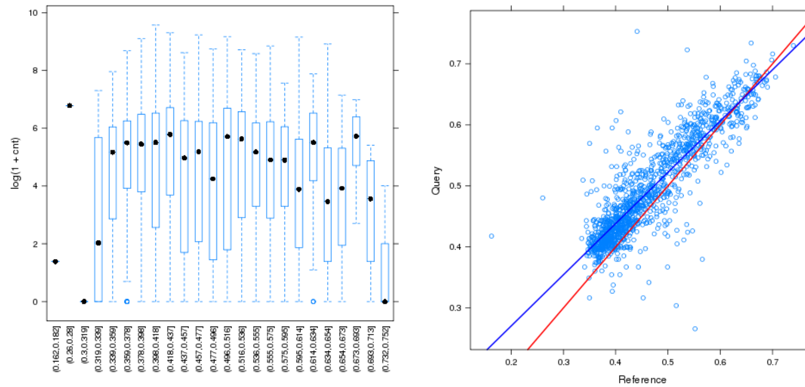



Figure 1: GC content and read counts. Left: reads aligning to genes as a function of reference sequence GC content; Right: GC content of reads versus corresponding reference gene sequence

Finally, use `findOverlaps` to identify alignments that overlap genic regions; the code leading up to `keep` uses the overlaps to find reads hitting a single gene, and is computationally more efficient that calling `countOverlaps` twice, as we did in previous code chunks. Using the unique hits to retrieve the sequences of individual reads, calculate the GC content of each read, and relate the mean GC content of reads in each gene to the GC content of the gene itself.

```
> ## more detail -- gc content of aligned reads
> olap <- findOverlaps(aln, gn, ignore.strand=TRUE)
> hits <- tabulate(queryHits(olap), length(aln))
> keep <- queryHits(olap) %in% which(hits == 1)
> reads <- qseq(aln)[ queryHits(olap)[keep] ]
> readGc <- rowSums(alphabetFrequency(reads, as.prob=TRUE)[,c("G", "C")])
> readGcPerGene <- tapply(readGc, subjectHits(olap)[keep], mean)
> idx <- as.integer(names(readGcPerGene))
> xyplot(readGcPerGene ~ gc[idx],
+        panel=function(...) {
+            panel.xyplot(...)
+            panel.abline(0, 1, col="red", lwd=2)
+            panel.lmline(..., col="blue", lwd=2)
+        }, xlab="Reference", ylab="Query")
```

As an advanced exercise, generalize the preceeding work flow to work on all of our bam files. Do this by encapsulating our work flow into a function on which we can use `lapply`, `sapply`, or a multi-processor variant like `multicore::mclapply`.

```
> sessionInfo()
```

```
R Under development (unstable) (2011-07-26 r56509)
```

```
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=C                 LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
 [1] TxDb.Scerevisiae.UCSC.sacCer2.ensGene_2.5.0
 [2] GenomicFeatures_1.5.16
 [3] org.Sc.sgd.db_2.5.0
 [4] RSQLite_0.9-4
 [5] DBI_0.2-5
 [6] AnnotationDbi_1.15.9
 [7] leeBamViews_0.99.11
 [8] BSgenome_1.21.3
 [9] Biobase_2.13.7
[10] Rsamtools_1.5.43
[11] Biostrings_2.21.7
[12] GenomicRanges_1.5.21
[13] IRanges_1.11.15

loaded via a namespace (and not attached):
[1] biomaRt_2.9.2       RCurl_1.6-6         rtracklayer_1.13.10
[4] tools_2.14.0        XML_3.4-0           zlibbioc_0.1.7
```