Practical: Read Counting in RNA-seq

Hervé Pagès (hpages@fhcrc.org)

5 February 2014

Contents

1 Introduction

In the context of a high-throughput sequencing experiment, *counting the reads* means counting the number of reads per gene (or exon). It is usually the preliminary step to a *differential analysis* at the gene level (or exon level).

The result of this counting will typically be organized as a matrix where:

- each row represents a gene (or exon);
- each column represents a sequencing run (usually a given sample);
- and each value is the raw number of reads from the sequencing run that were *assigned* to the gene (or exon).

Different criteria can be used for assigning reads to genes. A common one is to assign a read to a gene if the aligned read overlaps with that gene and with that gene only.

In this practical we learn how aligned reads stored in BAM files can be counted with the `summarizeOverlaps()` function from the *GenomicRanges* package. With this function, the criteria used for assigning reads to genes is controlled via 2 arguments: the `mode` and `inter.feature` arguments. In addition to the man page for `summarizeOverlaps()`, the "Counting reads with summarizeOverlaps" vignette (located in the *GenomicRanges* package) is recommended reading if you're planning to use this function for your work.

The output of `summarizeOverlaps()` will be a *SummarizedExperiment* object containing the matrix of counts together with information about the genes (or exons) in the `rowData` component and about the samples (e.g. patient ID, treatment, etc...) in the `colData` component. This object will be suitable input to the *DESeq2* package for performing a *differential analysis*.

IMPORTANT NOTE: Starting with the upcoming version of *Bioconductor* (BioC 2.14, scheduled for April 2014), the `summarizeOverlaps()` function and its vignette will be located in the new *GenomicAlignments* package.

2 First look at some precomputed read counts

Before we do our own read counting, we start by having a quick look at some precomputed counts so we get an idea of what a *SummarizedExperiment* object looks like.

The *parathyroidSE* package contains RNA-seq data from the publication of Haglund et al. [1]. The *paired-end* sequencing was performed on primary cultures from parathyroid tumors of 4 patients at 2 time points over 3 conditions (control, treatment with diarylpropionitrile (DPN) and treatment with 4-hydroxytamoxifen (OHT)). DPN is a selective estrogen receptor β 1 agonist and OHT is a selective estrogen receptor modulator. One sample (patient 4, 24 hours, control) was omitted by the paper authors due to low quality.

The *parathyroidSE* package contains several data sets. One of them is the `parathyroidGenesSE` data set which contains the counts of reads per gene.

Exercise 1 *In this exercise, we load the `parathyroidGenesSE` data set from the parathyroidSE package and perform some basic manipulations on it.*

a. *Load the `parathyroidGenesSE` data set from the parathyroidSE package.*
   What's the class of this object? What are its dimensions?
b. *The information in a SummarizedExperiment object can be accessed with accessor functions. For example, to get the actual data (i.e., here, the read counts), we use the `assay()` function.*
   What's returned by `assay()`? What are its dimensions. Display the top left corner of it (e.g. first 8 rows and columns). Does it have row names? Column names? What are the row names?
c. *In this matrix of read counts, each row represents an Ensembl gene, each column a sequencing run, and the values are the raw numbers of reads in each sequencing run that were assigned to the respective gene.*
   How many reads were assigned to a gene in each sequencing run? How many genes have non-zero counts?
d. *Use `rowData()` on `parathyroidGenesSE`. What do you get? What's its length?*
e. *Use `colData()` on `parathyroidGenesSE`. What do you get? How many rows does it have?*
   Use `table()` to summarize the number of runs for each treatment (Control, DPN, and OHT).

Solution:

a. First we load the *parathyroidSE* package.

```
library(parathyroidSE)
```

Before we load the `parathyroidGenesSE` data set, we can check what data sets are contained in the *parathyroidSE* package with:

```
data(package="parathyroidSE")
```

Load the `parathyroidGenesSE` data set:

```
data(parathyroidGenesSE)
parathyroidGenesSE
## class: SummarizedExperiment
## dim: 63193 27
## exptData(1): MIAME
## assays(1): counts
## rownames(63193): ENSG00000000003 ENSG00000000005 ... LRG_98 LRG_99
## rowData metadata column names(0):
## colnames: NULL
## colData names(8): run experiment ... study sample
class(parathyroidGenesSE)
## [1] "SummarizedExperiment"
## attr(,"package")
## [1] "GenomicRanges"
dim(parathyroidGenesSE)
```

```
## [1] 63193    27
```

b.
```
class(assay(parathyroidGenesSE))
## [1] "matrix"
dim(assay(parathyroidGenesSE))
## [1] 63193    27
assay(parathyroidGenesSE)[1:8, 1:8]
##                 [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## ENSG00000000003  792 1064  444  953  519  855  413  365
## ENSG00000000005    4    1    2    3    3    1    0    1
## ENSG00000000419  294  282  164  263  179  217  277  204
## ENSG00000000457  156  184   93  145   75  122  228  171
## ENSG00000000460  396  207  210  212  221  173  611  199
## ENSG00000000938    3    8    2    5    0    4   13   22
## ENSG00000000971   12   23   10   12    4    7   12    8
## ENSG00000001036 2536 2349 1438 2307 1339 1677 1086  929
```

The row names are Ensembl gene ids. No column names:

```
colnames(parathyroidGenesSE)
## NULL
```

c. To compute the number of reads that were assigned to a gene in each sequencing run, we just need to sum all the counts that are in a column and do this for each column:

```
colSums(assay(parathyroidGenesSE))
##  [1]  9102683 10827109  5217761  9706035  5700022  7854568  8610014  6844144  5251911
## [10] 19332369  8267977  5620890 17969521  8247122  7341000  8064268 12481958 16310090
## [19] 23697329  7642648  7701432  7135899  4499893  9318500  6099942  5505205 10320006
```

Genes with non-zero counts:

```
sum(rowSums(assay(parathyroidGenesSE)) != 0)
## [1] 35415
```

d.
```
rowData(parathyroidGenesSE)
## GRangesList of length 63193:
## $ENSG00000000003
## GRanges with 17 ranges and 2 metadata columns:
##        seqnames              ranges strand   |   exon_id       exon_name
##           <Rle>           <IRanges>  <Rle>   | <integer>     <character>
##    [1]        X [99883667, 99884983]      -   |    664095 ENSE00001459322
##    [2]        X [99885756, 99885863]      -   |    664096 ENSE00000868868
##    [3]        X [99887482, 99887565]      -   |    664097 ENSE00000401072
##    [4]        X [99887538, 99887565]      -   |    664098 ENSE00001849132
##    [5]        X [99888402, 99888536]      -   |    664099 ENSE00003554016
##    ...      ...                 ...    ... ...       ...             ...
##   [13]        X [99890555, 99890743]      -   |    664106 ENSE00003512331
##   [14]        X [99891188, 99891686]      -   |    664108 ENSE00001886883
##   [15]        X [99891605, 99891803]      -   |    664109 ENSE00001855382
##   [16]        X [99891790, 99892101]      -   |    664110 ENSE00001863395
##   [17]        X [99894942, 99894988]      -   |    664111 ENSE00001828996
##
## ...
## <63192 more elements>
## ---
## seqlengths:
```

```
##                 1                 2 ...            LRG_98            LRG_99
##         249250621         243199373 ...             18750             13294
```

We get a *GRangesList* object with one list element per gene. Each list element is a *GRanges* object containing the exon ranges for the gene.

e.
```
colData(parathyroidGenesSE)
## DataFrame with 27 rows and 8 columns
##             run experiment  patient treatment     time submission     study    sample
##     <character>   <factor> <factor>  <factor> <factor>   <factor>  <factor>  <factor>
## 1     SRR479052  SRX140503        1   Control      24h  SRA051611 SRP012167 SRS308865
## 2     SRR479053  SRX140504        1   Control      48h  SRA051611 SRP012167 SRS308866
## 3     SRR479054  SRX140505        1       DPN      24h  SRA051611 SRP012167 SRS308867
## 4     SRR479055  SRX140506        1       DPN      48h  SRA051611 SRP012167 SRS308868
## 5     SRR479056  SRX140507        1       OHT      24h  SRA051611 SRP012167 SRS308869
## ...         ...        ...      ...       ...      ...        ...       ...       ...
## 23    SRR479074  SRX140523        4       DPN      48h  SRA051611 SRP012167 SRS308885
## 24    SRR479075  SRX140523        4       DPN      48h  SRA051611 SRP012167 SRS308885
## 25    SRR479076  SRX140524        4       OHT      24h  SRA051611 SRP012167 SRS308886
## 26    SRR479077  SRX140525        4       OHT      48h  SRA051611 SRP012167 SRS308887
## 27    SRR479078  SRX140525        4       OHT      48h  SRA051611 SRP012167 SRS308887
```

We get a *DataFrame* object with one row per sequencing run.

```
table(colData(parathyroidGenesSE)$treatment)
##
## Control     DPN     OHT
##       7      10      10
```

3 Aligned reads and BAM files

To operate, the `summarizeOverlaps()` function needs 2 data objects:

1. one representing the genomic ranges of the genes (or exons);
2. one representing the aligned reads.

The aligned reads are typically stored in one BAM file per sequencing run. In the next exercise we will have a quick look at the BAM files included in the *parathyroidSE* package. The reads in these files are *paired-end reads* that were aligned using the TopHat aligner. To keep the package to a reasonable size, only a subset of all the aligned reads from the experiment have been placed in these files. More information on how these BAM files were obtained can be found in the vignette located in the *parathyroidSE* package.

To get the paths to these files, do:

```
bamdir <- system.file("extdata", package="parathyroidSE")
bampaths <- list.files(bamdir, pattern="bam$", full.names=TRUE)
bampaths
```

```
## [1] "/home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479052.bam"
## [2] "/home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479053.bam"
## [3] "/home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479054.bam"
```

To load *single-end reads* from a BAM file, we can use the `readGAlignmentsFromBam()` function from the *Rsamtools* package. If the reads are *paired-end* and we want to preserve the pairing, the `readGAlignmentPairsFromBam()` function can be used. However, for downstream analyses where the pairing doesn't need to be preserved (e.g. if

we're only going to compute the coverage of the reads), the reads can be loaded with `readGAlignmentsFromBam()`, which is faster and returns an object that is simpler and easier to manipulate.

One last thing before we start the exercise. By default `readGAlignmentsFromBam()` and `readGAlignmentPairsFromBam()` load PCR or optical duplicates as well as secondary alignments. These alignments are generally discarded from the read counting step. We can discard them up-front by filtering them out when we load the alignments from the BAM files. This is done by creating and passing a *ScanBamParam* object to the `param` argument of `readGAlignmentsFromBam()` or `readGAlignmentPairsFromBam()`.

Exercise 2 *In this exercise, we learn how to load paired-end reads and filter out the alignments that are not suitable for read counting.*

a. *Load the SRR479052.bam file included in the parathyroidSE package, first with `readGAlignmentsFromBam()`, then with `readGAlignmentPairsFromBam()`.*
   What are the classes of the returned objects?
   How many pairs are there in the 2nd object?
b. *The first and last mate for each pair can be extracted from the GAlignmentPairs object with the `first()` and `last()` accessor functions.*
   Extract the first mates. Extract the last mates.
c. *See the man page for the `ScanBamParam()` constructor in the Rsamtools package.*
   Construct a ScanBamParam object (that you will pass to `readGAlignmentPairsFromBam()`) that will filter out PCR or optical duplicates as well as secondary alignments.
   Use it to load the pairs again.

Solution:

a. Loading the BAM file first with `readGAlignmentsFromBam()`:

```
library(Rsamtools)
gal0 <- readGAlignmentsFromBam(bampaths[1])
gal0
## GAlignments with 9973 alignments and 0 metadata columns:
##          seqnames strand       cigar    qwidth     start       end     width      ngap
##             <Rle>  <Rle> <character> <integer> <integer> <integer> <integer> <integer>
##      [1]       10      +        101M       101  59953037  59953137       101         0
##      [2]       10      -        101M       101  59953061  59953161       101         0
##      [3]       11      +        101M       101    209517    209617       101         0
##      [4]       10      +        101M       101 121691690 121691790       101         0
##      [5]       10      -        101M       101 121691702 121691802       101         0
##      ...      ...    ...         ...       ...       ...       ...       ...       ...
##   [9969]       14      -  71M117N30M       101  24035299  24035516       218         1
##   [9970]       16      +        101M       101  56466390  56466490       101         0
##   [9971]       16      -        101M       101  56466533  56466633       101         0
##   [9972]       MT      +        101M       101      6194      6294       101         0
##   [9973]       MT      -        101M       101      6316      6416       101         0
##   ---
##   seqlengths:
##             1        10        11        12 ...         9        MT         X         Y
##    249250621 135534747 135006516 133851895 ... 141213431     16569 155270560  59373566
```

then with `readGAlignmentPairsFromBam()`:

```
galp0 <- readGAlignmentPairsFromBam(bampaths[1])
## Warning:  8 alignments with ambiguous pairing were dumped.
##    Use 'getDumpedAlignments()' to retrieve them from the dump environment.
## Warning:  12.987012987013% of the pairs with discordant seqnames or strand were flagged
##  as proper pairs by the aligner.  Dropping them anyway.
```

```
galp0
## GAlignmentPairs with 4435 alignment pairs and 0 metadata columns:
##        seqnames strand   :                ranges  --                ranges
##           <Rle>  <Rle>   :             <IRanges>  --             <IRanges>
##    [1]       10      +   : [ 59953037,  59953137]  -- [ 59953061,  59953161]
##    [2]       10      -   : [121691702, 121691802]  -- [121691690, 121691790]
##    [3]        2      +   : [123166234, 123166334]  -- [123166269, 123166369]
##    [4]        6      +   : [ 36945908,  36946357]  -- [ 36953742,  36953842]
##    [5]        3      -   : [ 15112192,  15112292]  -- [ 15112134,  15112234]
##    ...      ...    ... ...                    ... ...                   ...
## [4431]        8      +   : [ 97621642,  97621742]  -- [ 97621646,  97621746]
## [4432]        6      -   : [149730898, 149730998]  -- [149720309, 149730801]
## [4433]       14      +   : [ 24033812,  24034370]  -- [ 24035299,  24035516]
## [4434]       16      +   : [ 56466390,  56466490]  -- [ 56466533,  56466633]
## [4435]       MT      -   : [     6316,      6416]  -- [     6194,      6294]
## ---
## seqlengths:
##          1        10        11        12 ...         9        MT         X         Y
##  249250621 135534747 135006516 133851895 ... 141213431     16569 155270560  59373566
```

`gal` is a *GAlignments* object. `galp` is a *GAlignmentPairs* object. A *GAlignmentPairs* object is also vector-like object where each element represents an aligned *paired-end read*. So the number of pairs in it is just:

```
length(galp0)
## [1] 4435
```

b.

```
first(galp0)
## GAlignments with 4435 alignments and 0 metadata columns:
##          seqnames strand       cigar    qwidth     start       end     width      ngap
##             <Rle>  <Rle> <character> <integer> <integer> <integer> <integer> <integer>
##      [1]       10      +        101M       101  59953037  59953137       101         0
##      [2]       10      -        101M       101 121691702 121691802       101         0
##      [3]        2      +        101M       101 123166234 123166334       101         0
##      [4]        6      +  24M349N77M       101  36945908  36946357       450         1
##      [5]        3      -        101M       101  15112192  15112292       101         0
##      ...      ...    ...         ...       ...       ...       ...       ...       ...
##   [4431]        8      +        101M       101  97621642  97621742       101         0
##   [4432]        6      -        101M       101 149730898 149730998       101         0
##   [4433]       14      +  58M458N43M       101  24033812  24034370       559         1
##   [4434]       16      +        101M       101  56466390  56466490       101         0
##   [4435]       MT      -        101M       101      6316      6416       101         0
##   ---
##   seqlengths:
##            1        10        11        12 ...         9        MT         X         Y
##    249250621 135534747 135006516 133851895 ... 141213431     16569 155270560  59373566
last(galp0)
## GAlignments with 4435 alignments and 0 metadata columns:
##          seqnames strand       cigar    qwidth     start       end     width      ngap
##             <Rle>  <Rle> <character> <integer> <integer> <integer> <integer> <integer>
##      [1]       10      -        101M       101  59953061  59953161       101         0
##      [2]       10      +        101M       101 121691690 121691790       101         0
##      [3]        2      -        101M       101 123166269 123166369       101         0
##      [4]        6      -        101M       101  36953742  36953842       101         0
##      [5]        3      +        101M       101  15112134  15112234       101         0
##      ...      ...    ...         ...       ...       ...       ...       ...       ...
```

```
##   [4431]        8      -         101M       101  97621646  97621746       101         0
##   [4432]        6      + 12M10392N89M       101 149720309 149730801     10493         1
##   [4433]       14      -   71M117N30M       101  24035299  24035516       218         1
##   [4434]       16      -         101M       101  56466533  56466633       101         0
##   [4435]       MT      +         101M       101      6194      6294       101         0
##   ---
##   seqlengths:
##           1        10        11        12 ...         9        MT         X         Y
##    249250621 135534747 135006516 133851895 ... 141213431     16569 155270560  59373566
```

c.
```
param <- ScanBamParam(flag=scanBamFlag(isDuplicate=FALSE,
                                       isNotPrimaryRead=FALSE))
readGAlignmentPairsFromBam(bampaths[1], param=param)
## Warning:  1.51515151515152% of the pairs with discordant seqnames or strand were flagged
##  as proper pairs by the aligner.  Dropping them anyway.
## GAlignmentPairs with 4268 alignment pairs and 0 metadata columns:
##        seqnames strand   :               ranges  --               ranges
##           <Rle>  <Rle>   :            <IRanges>  --            <IRanges>
##    [1]       10      +   : [ 59953037,  59953137]  -- [ 59953061,  59953161]
##    [2]       10      -   : [121691702, 121691802]  -- [121691690, 121691790]
##    [3]        2      +   : [123166234, 123166334]  -- [123166269, 123166369]
##    [4]        6      +   : [ 36945908,  36946357]  -- [ 36953742,  36953842]
##    [5]        3      -   : [ 15112192,  15112292]  -- [ 15112134,  15112234]
##    ...      ...    ... ...                   ... ...                   ...
## [4264]        8      +   : [ 97621642,  97621742]  -- [ 97621646,  97621746]
## [4265]        6      -   : [149730898, 149730998]  -- [149720309, 149730801]
## [4266]       14      +   : [ 24033812,  24034370]  -- [ 24035299,  24035516]
## [4267]       16      +   : [ 56466390,  56466490]  -- [ 56466533,  56466633]
## [4268]       MT      -   : [     6316,      6416]  -- [     6194,      6294]
## ---
## seqlengths:
##          1        10        11        12 ...         9        MT         X         Y
##  249250621 135534747 135006516 133851895 ... 141213431     16569 155270560  59373566
```

4 Choosing and loading a gene model

To operate, `summarizeOverlaps()` needs access to the genomic ranges of the genes (or exons). This information can be extracted from what we call a *gene model*. Gene models for various organisms are provided by many annotation providers on the internet (UCSC, Ensembl, NCBI, TAIR, FlyBase, WormBase, etc...) In *Bioconductor* a gene model is typically represented as a *TranscriptDb* object. The *GenomicFeatures* package contains tools for obtaining a gene model from these providers and store it in a *TranscriptDb* object (the container for gene models). For convenience, the most commonly used gene models are available as *Bioconductor* data packages (called TxDb packages). Each TxDb package contains a *TranscriptDb* object ready to use.

According to the vignette located in the *parathyroidSE* package, the reads in the BAM files were aligned to the GRCh37 human reference genome. If we wanted to use the gene model for Human provided by Ensembl, we could do:

```
## Requires INTERNET ACCESS and takes about 6 min. Please don't try to run this!
library(GenomicFeatures)
txdb <- makeTranscriptDbFromBiomart(biomart="ensembl",
```

```
                                    dataset="hsapiens_gene_ensembl")
```

This would return a *TranscriptDb* object containing the Ensembl gene model for Human.

IMPORTANT NOTE: One must be careful to choose a gene model based on the same reference genome that was used to align the reads. The annotations provided by Ensembl are updated at each new Ensembl release, which typically happens 2 or 3 times per year (current release is Ensembl 73). The `"hsapiens_gene_ensembl"` dataset is usually based on the most recent version of the Human reference genome (currently GRCh37). So before we proceed with this *TranscriptDb* object, we would need to make sure that it's compatible with our BAM files, that is, we would need to check that the `"hsapiens_gene_ensembl"` dataset was based on GRCh37 human at the time the *TranscriptDb* object was made.

Because our goal is to use the counts to perform a *differential analysis* at the gene level, we will need to feed `summarizeOverlaps()` with a *GRangesList* object containing the exon ranges grouped by gene. This can be extracted from the *TranscriptDb* object with the `exonsBy()` function:

```
ex_by_gene <- exonsBy(txdb, by="gene")  # GRangesList object
```

For the purpose of this practical, we'll use a subset of the Ensembl genes. This subset is stored in the *parathyroidSE* package and is based on the GRCh37 human reference genome.

Exercise 3 *In this exercise, we have a quick look at the `exonsByGene` data set included in the parathyroidSE package.*

a. *Load the `exonsByGene` data set from the parathyroidSE package. What is it?*
b. *How many genes are represented in this object?*

Solution:

a.
```
data(exonsByGene)
exonsByGene
## GRangesList of length 100:
## $ENSG00000000003
## GRanges with 17 ranges and 2 metadata columns:
##        seqnames              ranges strand   |   exon_id       exon_name
##           <Rle>           <IRanges>  <Rle>   | <integer>     <character>
##    [1]        X [99883667, 99884983]      -   |    664095 ENSE00001459322
##    [2]        X [99885756, 99885863]      -   |    664096 ENSE00000868868
##    [3]        X [99887482, 99887565]      -   |    664097 ENSE00000401072
##    [4]        X [99887538, 99887565]      -   |    664098 ENSE00001849132
##    [5]        X [99888402, 99888536]      -   |    664099 ENSE00003554016
##    ...      ...                 ...    ... ...       ...             ...
##   [13]        X [99890555, 99890743]      -   |    664106 ENSE00003512331
##   [14]        X [99891188, 99891686]      -   |    664108 ENSE00001886883
##   [15]        X [99891605, 99891803]      -   |    664109 ENSE00001855382
##   [16]        X [99891790, 99892101]      -   |    664110 ENSE00001863395
##   [17]        X [99894942, 99894988]      -   |    664111 ENSE00001828996
##
## ...
## <99 more elements>
## ---
## seqlengths:
##                   1                 2 ...            LRG_98            LRG_99
##           249250621         243199373 ...             18750             13294
```

b. Number of genes in this object:

```
length(exonsByGene)
## [1] 100
```

5 Count the reads

To count the reads, we use the `summarizeOverlaps()` function defined and documented in the *GenomicRanges* package.

The aligned reads must be passed to the `reads` argument of the function (the 2nd argument). They can be represented in different ways, including as a *BamFile*, a *GAlignments*, a *GAlignmentPairs*, or a *BamFileList* object. The first 3 types of objects only allow passing the reads from a single sequencing run at a time. Using a *BamFileList* object allows us to pass the reads from all the sequencing runs at once. To create such an object, we use the `BamFileList()` constructor function from the *Rsamtools* package:

```
library(Rsamtools)
bamfile_list <- BamFileList(bampaths, index=character())
```

Note that we need to use `index=character()` here because there are no BAM index files (`.bam.bai` extension) associated with our BAM files.

Exercise 4 *Let's do the read counting.*

a. *Use `summarizeOverlaps()` on `exonsByGene` and `bamfile_list` to count the reads. Check the man page for the details. Note that because the RNA-seq protocol was not strand specific, you need to specify `ignore.strand=TRUE`. Also because the reads are* paired-end*, you need to specify `singleEnd=FALSE`. This will tell `summarizeOverlaps()` to use `readGAlignmentPairsFromBam()` instead of `readGAlignmentsFromBam()` internally to read the BAM files.*
b. *When `summarizeOverlaps()` calls the reading function internally on each BAM file, it does so without specifying any particular `param` value, so, by default, PCR or optical duplicates and secondary alignments are loaded. However, if a `param` argument is passed to `summarizeOverlaps()`, it will be passed along to the reading function.*
Count the reads again but discard PCR or optical duplicates as well as secondary alignments.

Solution:

a.
```
read_count0 <- summarizeOverlaps(exonsByGene, bamfile_list,
                                 ignore.strand=TRUE,
                                 singleEnd=FALSE)
read_count0
## class: SummarizedExperiment
## dim: 100 3
## exptData(0):
## assays(1): counts
## rownames(100): ENSG00000000003 ENSG00000000005 ... ENSG00000005469
##   ENSG00000005471
## rowData metadata column names(0):
## colnames(3):
##   /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479052.bam
##   /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479053.bam
##   /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479054.bam
## colData names(0):
```

b. To discard PCR or optical duplicates as well as secondary alignments, we re-use the *ScanBamParam* object we prepared earlier:

```
read_count <- summarizeOverlaps(exonsByGene, bamfile_list,
                                ignore.strand=TRUE,
                                singleEnd=FALSE,
                                param=param)
read_count
## class: SummarizedExperiment
## dim: 100 3
## exptData(0):
## assays(1): counts
## rownames(100): ENSG00000000003 ENSG00000000005 ... ENSG00000005469
##   ENSG00000005471
## rowData metadata column names(0):
## colnames(3):
##   /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479052.bam
##   /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479053.bam
##   /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479054.bam
## colData names(0):
```

Let's do a quick comparison between the 2 counts:

```
colSums(assay(read_count0))
## /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479052.bam
##                                                                                            27
## /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479053.bam
##                                                                                            17
## /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479054.bam
##                                                                                            26
colSums(assay(read_count))
## /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479052.bam
##                                                                                            27
## /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479053.bam
##                                                                                            17
## /home/mtmorgan/R/x86_64-unknown-linux-gnu-library/3.0-2.13/parathyroidSE/extdata/SRR479054.bam
##                                                                                            26
```

No difference here in the final count. This means the reads we discarded didn't get assigned to any gene the first time we counted (but this wouldn't necessarily be the case with a bigger data set).

6 Conclusion

Now that we have our read counts, we're ready to perform a *differential analysis* with the *DESeq2* package.

THANKS!

References

[1] Felix Haglund, Ran Ma, Mikael Huss, Luqman Sulaiman, Ming Lu, Inga-Lena Nilsson, Anders Höög, Christofer˜C. Juhlin, Johan Hartman, and Catharina Larsson. Evidence of a Functional Estrogen Recep-

tor in Parathyroid Adenomas. *Journal of Clinical Endocrinology & Metabolism*, September 2012. URL: http://dx.doi.org/10.1210/jc.2012-2484, doi:10.1210/jc.2012-2484.