htSeqTools: High-Throughput Sequencing Quality Control, Processing and Visualization in R

Evarist Planet\textsuperscript{1}, Camille Stephan-Otto Attolini\textsuperscript{1}, Oscar Reina\textsuperscript{1}, Oscar Flores\textsuperscript{2} and David Rossell\textsuperscript{1*}

\textsuperscript{1}Biostatistics & Bioinformatics Unit, IRB Barcelona, Spain
\textsuperscript{2}IRB-BSC Joint Research Program on Computational Biology, IRB Barcelona, Spain

Contents

1 Package functionality 2
  1.1 Data import and storage ............................................ 2
  1.2 Quality control ....................................................... 3
  1.3 ChIP-seq workflow .................................................. 4
  1.4 RNA-seq workflow ................................................... 5

2 Detailed method description 6
  2.1 Detecting over-amplification artifacts ............................ 6
  2.2 Derivation of SD\textsubscript{n} and G\textsubscript{n} ....................... 7
  2.3 Determining rich-read regions and differential expression .... 9
  2.4 Detecting the accumulation of significant hits ................ 9

3 Examples 10
  3.1 Example 1: GSE25836 .................................................. 10
  3.2 Example 2: GSE16926 .................................................. 20
  3.3 Example 3: Histone methylation data .............................. 24
  3.4 Example 4: Yeast RNA-seq .......................................... 28

*to whom correspondence should be addressed
1 Package functionality

htSeqTools relies on the general infra-structure set up by Bioconductor (Gentleman et al., 2004). Bioconductor provides access to a wide variety of data import, processing and analysis tools, and allows for an efficient implementation of common tasks. Most functions in htSeqTools allow for parallel computing by using the multicore package (Urbanek, 2011). Table 1 provides running times for several functions when applied to the example in Section 3.1. For instance, computing coverage and 28 pairwise correlations with cmds required 274.8 seconds on a single core. The use of 4 cores reduced the time to 114.4 seconds.

We now describe the main functions provided in htSeqTools. As much of the emphasis is on quality control and pre-processing, in Section 1.2 we provide a comparison with similar software. We then detail typical workflows for ChIP-seq (Section 1.3) and RNA-seq (Section 1.4) studies. For some detailed examples with R code see Section 3.

1.1 Data import and storage

Several packages are available for importing aligned reads into Bioconductor, e.g. Rsamtools for BAM (Morgan and Pagés, 2011a), ShortRead for BAM, Bowtie, ELAND, MAQ and SOAP (Morgan et al., 2011b), and rtracklayer for BED, GFF and WIG (Lawrence et al., 2011). The input format for htSeqTools is based on the data classes defined in the IRanges package (Pagés et al., 2011). Data from multiple samples are assumed to be stored either in a RangedDataList or a GRangesList object (RangedData or GRanges for a single sample). The RangedData format is convenient in that a number of Bioconductor packages provide tools to efficiently manipulate and analyze it (e.g. coverage computation, overlaps with pre-specified genomic regions etc.). Additionally, the format allows efficient data compression, which greatly reduces the required disk space and time for read/write operations. As an illustration, roughly 15 million reads occupying 800Mb of disk space after Bowtie alignment and bzip2 compression required only 45Mb in the RangedData format.

Part of the data reduction in RangedData objects is achieved by discarding base calls and quality scores. The information by most htSeqTools functions is the chromosome, start and end position that each read was aligned to, and in some cases the strand, which is usually sufficient in studies like ChIP-Seq or RNA-Seq. Notice however that some studies may require additional information, e.g. single nucleotide polymorphism (SNP) analysis strategies require base calls and qualities.
<table>
<thead>
<tr>
<th></th>
<th>1 core</th>
<th>2 cores</th>
<th>4 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmds</td>
<td>274.8 (100%)</td>
<td>191.2 (69.6%)</td>
<td>114.4 (41.6%)</td>
</tr>
<tr>
<td>tabDuplReads</td>
<td>118.3 (100%)</td>
<td>52.8 (44.6%)</td>
<td>38.4 (32.5%)</td>
</tr>
<tr>
<td>filterDuplReads</td>
<td>287.5 (100%)</td>
<td>183.9 (64.0%)</td>
<td>161.5 (56.2%)</td>
</tr>
<tr>
<td>islandCounts</td>
<td>213.7 (100%)</td>
<td>181.6 (85.0%)</td>
<td>157.8 (73.4%)</td>
</tr>
<tr>
<td>enrichedPeaks</td>
<td>102.2 (100%)</td>
<td>56.1 (54.9%)</td>
<td>36.7 (35.9%)</td>
</tr>
</tbody>
</table>

Table 1: Execution time (seconds) for an increasing number of processors. Obtained on an Apple running OS X 10.6.7 with 2.8GHz processors and 16Gb DDR2 RAM.

### 1.2 Quality control

We describe the quality control functionality of `htSeqTools`, as well as the following four free softwares: **ShortRead** package in Bioconductor [Morgan et al. (2011b)](http://www.bioinformatics.bbsrc.ac.uk/projects); FastQC version 0.9.3; SeqMonk version 0.16.0 (both at [http://www.bioinformatics.bbsrc.ac.uk/projects](http://www.bioinformatics.bbsrc.ac.uk/projects)); and UCSC Table browser version Oct 15, 2011 ([Karolchik et al. (2004)](http://genome.ucsc.edu/cgi-bin/hgText)). Our aim is not to provide a comprehensive list of all available software and their options, but to illustrate the potential usefulness of `htSeqTools`. In fact, we believe that other software can and should be used in conjunction with `htSeqTools` to better assess data quality.

Table 2 summarizes the software comparison. We consider a block of features related to read quality assessment, which includes inspecting intensities and base-calling accuracy, GC content and detecting adapter sequences. As a second block, we consider the detection of PCR over-amplification artifacts. We evaluate the capacity to detect, statistically assess and remove such artifacts. The third block is based on computing and visualizing correlations between samples, which provides insight into the samples that most closely resemble each other and points to sources of technical bias as well as possible outliers. Finally, we assess the capability to detect problems in sample preparation steps aimed to enrich certain genomic regions (e.g. immuno-precipitation) as well as other data pre-processing steps.

After the sequence assessment steps available in `ShortRead`, `htSeqTools` provides several functions that extend the Bioconductor flow. To our knowledge, it is the only software measuring enrichment efficiency and providing a statistical control on the number of read repeats. This procedure allows the user to retain sequences with a few repeats, in situations where a certain number of naturally occurring repeats are expected (e.g. short genomes, deeply sequenced samples, experiments targeting a small subset of the genome).
Table 2: Comparison of quality control software functionality. 

<table>
<thead>
<tr>
<th>Function</th>
<th>htSeqTools</th>
<th>ShortRead</th>
<th>FastQC</th>
<th>SeqMonk</th>
<th>UCSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence assessment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base call qualities</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>GC content</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Adapter detection</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Over-amplification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detection</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Statistical assessment</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Removal</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>R²</td>
<td>No</td>
</tr>
<tr>
<td>Pre-processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correct strand bias</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Extend read length</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Find read-reach regions</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sample correlation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>R¹</td>
</tr>
<tr>
<td>Visualize</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Enrichment efficiency</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

1.3 ChIP-seq workflow

Below we indicate the steps in a typical ChIP-seq workflow using **htSeqTools**, as well as the functions needed for each step. Detailed ChIP-seq examples are provided in Sections 3.1-3.2.

1. Detect and remove over-amplification artifacts (*filterDuplReads, tabDuplReads, fdrEnrichedCounts*)
2. Assess similarity between sample via correlation plots (*cmds*)
3. Extend reads (optional, *extendRanges*)
4. Align peaks (*alignPeaks*)
5. Assess enrichment efficiency (*giniCoverage, ssdCoverage*)
6. Find read-rich regions (*enrichedRegions*)
7. Find peaks within regions (*enrichedPeaks*)
8. Annotate peaks (package **ChIPpeakAnno**)
9. Visualize peak location (*PeakLocation, stdPeakLocation, gridCoverage*)
10. Find regions with accumulation of peaks (enrichedChrRegions)

Although here we only include them in Step 2, MDS correlation plots can be used again after read extension, peak alignment or any other desired pre-processing in order to assess their effect on the data. Regarding Step 3, in practice it may be difficult to establish the number of bases that reads should be extended to. In some cases, the extension may actually cause a loss of resolution in peak calling. Hence, we consider Step 3 as optional. We also note that Step 10 is not the primary interest in many ChIP-seq experiments, but included it here for completeness.

1.4 RNA-seq workflow

Here we indicate the basic steps in an RNA-seq workflow. See Section 3.4 for a yeast RNA-seq worked example.

1. Detect and remove over-amplification artifacts (filterDuplReads, tabDuplReads, fdrEnrichedCounts)
2. Assess similarity between sample via correlation plots (cmds)
3. Identify expressed RNA de novo (islandCounts)
4. Compare RNA expression across samples (enrichedRegions)
5. Annotate differentially expressed RNA (package ChIPpeakAnno)
6. Find regions with accumulation of differential expression (enrichedChrRegions)

Similar to the ChIP-seq workflow in Section 1.3, we start by removing over-amplification artifacts. As an important remark, when sequencing short RNAs the same fragment will typically be sequenced many times. That is, one expects a large number of repeats and therefore step 1 (removal of over-amplification artifacts) should be skipped. Next, one inspects if samples cluster appropriately via MDS coverage correlation plots (step 2). Further pre-processing steps are possible. Repeating the MDS correlation plot after pre-processing can help assess whether the processing helped improve the separation between groups.

Steps 3 and 5 are most useful when the genome is poorly annotated, or the user wishes to refine the available annotation, e.g. find previously unknown exons. Regarding step 4, many alternatives are available within R for determining differential expression, e.g. edgeR (Robinson et al., 2010),
DESeq \cite{Anders2010}, and DEGseq \cite{Wang2011}. The approach implemented in enrichedRegions (details in Section 2.3) is specially suited for comparing expression between individual samples. Although it remains useful in datasets with multiple samples per group (see examples in Section 3), in such setups we view it mainly as a quick screening tool and recommend using approaches that formally incorporate the within-group variability (e.g. edgeR). Finally, step 6 allows the user to find spatial trends in differential expression, which can suggest common regulatory mechanisms.

2 Detailed method description

2.1 Detecting over-amplification artifacts

High-throughput sequencing requires a PCR amplification step to enrich for adapter-ligated fragments. This step can induce biases as some DNA regions amplify more efficiently than others (e.g. depending on GC content). These PCR artifacts, caused by over-amplification or primer dimers, affect the accuracy of the coverage and can produce biases in downstream analyses. The function filterDuplReads aims to automatically detect and remove these artifacts. The basic rationale is that, by counting the number of times that each read is repeated, we can detect the reads that repeat an unusually large number of times. Ideally, the threshold determining the maximum number of allowable repeats should be determined for each sample separately. The expected number of naturally occurring repeats depends on the genome length, the sequencing depth and the characteristics of each sample. For instance, sequences from IP samples in ChIP-seq experiments focus on a relatively small genomic region while those from controls are distributed along most of the genome, and therefore a higher number of repeats is expected in the former. filterDuplReads determines the threshold in a data-adaptive manner by controlling the False Discovery Rate (FDR). For an example where over-amplified reads were artificially added to a sample see the htSeqTools vignette. For experimental data examples see Section 3.

The basic rationale is that only reads repeating a large number of times are likely to be artifacts. Hence, the null distribution can be estimated by modelling the reads with few repeats. More precisely, let $p_j$ be the proportion of observed reads with $j$ repeats for $j \in \mathbb{N} \setminus \{0\}$, $\alpha$ be a lower bound for the proportion of non over-amplified reads, and $q_\alpha$ be the $\alpha$ quantile associated to $p_j$. By default we set $\alpha = 0.999$, i.e. at most 1/1000 reads are affected by over-amplification. In statistical terms, our basic assumption is that reads with $1, \ldots, q_\alpha$ repeats are not over-amplification artifacts. We model the
number of repeats for read $i$, which we denote as $X_i$ for $i = 1, \ldots, n$, as independent realizations from a mixture of Negative Binomial distributions truncated at $1 \leq X_i \leq q_0$. More formally, $X_i \sim \sum_{k=1}^{K} \pi_k \text{TNegBin}_{[1,q_0]}(r_k, \theta_k)$, \hspace{1cm} (1)

where $K$ is the number of components in the mixture, and $(r_k, \theta_k)$ are the number of failures and success probability in component $k$. We fit the model via Maximum Likelihood and choose $K$ according to the Bayesian Information Criterion (Schwarz, 1978). Overall, (1) is a flexible model which we observed to fit experimental data reasonably well in a number of scenarios.

In order to estimate the FDR for a given threshold $t$, we use an empirical Bayes approach similar to that in Efron et al (2001). Let $f_0(x)$ be the null distribution for the number of read repeats, $f_1(x)$ the distribution for over-amplified reads and $w$ the proportion of non over-amplified reads. The distribution of $X_i$ can be written as $f(x_i) = wf_0(x_i) + (1 - w)f_1(x_i)$, with independence across $i$. We estimate $f(x)$ with the observed frequencies, imposing that $\hat{f}(x)$ must be monotone decreasing after its mode (using isoreg from package base) to prevent random fluctuations in $\hat{f}(x)$ for large $x$. $f_0(x)$ is estimated with the truncated Negative Binomial mixture (1). Setting $w$ to its upper bound 1, an estimated upper bound for the FDR($t$) associated to the threshold $t$ is $\sum_{i \geq t} \hat{f}(i) \sum_{i \geq t} f(i)$. We enforce that the estimated FDR is monotone decreasing with $t$ via the monotone regression in isoreg. By default we set FDR($t$) < 0.01.

### 2.2 Derivation of SD$_n$ and G$_n$

Denote the number of observed sequences by $n$. Intuitively, a lack of uniformity in the distribution of such sequences along the genome indicates that some regions were selectively sequenced, i.e. enriched in the sample preparation. We measure lack of uniformity by assessing variability in the coverage, i.e. the number of sequences covering each base in the genome. In particular, ssdCoverage computes the coverage standard deviation (SD) and giniCoverage the Gini index (Gini, 1912), which is a classical econometrics measure of wealth inequality. Because coverage is never uniform in practice (e.g. due to sequencing preferences or sequence-dependent biases in sample preparation), strong departures from uniformity are expected in all samples. For this reason, the main usefulness of these indexes lies in comparing variability between a sample of interest and its control.
We first show that, assuming uniformity in the read distribution, the expected value of the coverage \( \text{SD} \) is proportional to \( \sqrt{n} \). Hence, it is preferable to use \( \text{SD}_n = \text{SD}/\sqrt{n} \) as a measure that can be compared across samples with different sequencing depths. Let \( m \) be the genome length, \( r \) the read length (which we assume constant for all reads), and \( y_k \) the coverage at position \( k \in \{1, \ldots, m\} \). Since each read covers \( r \) bases, the probability that an individual read overlaps position \( k \) is \( \frac{r}{m} \) and the mean coverage is equal to \( \bar{y} = \frac{1}{m} \sum_{k=1}^{m} y_k = \frac{nr}{m} \). Assuming independence between reads, \( E(y_k) = \frac{nr}{m} \) and \( V(y_k) = \frac{n}{m} \left( 1 - \frac{r}{m} \right) \) for all \( k \). The expected value of the coverage variance can thus be computed as

\[
E(\text{SD}^2) = E \left( \frac{1}{m} \sum_{k=1}^{m} (y_k - \bar{y})^2 \right) = \frac{1}{m} \sum_{k=1}^{m} E \left[ (y_k - \frac{nr}{m})^2 \right] = \frac{1}{m} \sum_{k=1}^{m} V(y_k) = \frac{m}{m} V(y_1) = n \frac{r}{m} \left( 1 - \frac{r}{m} \right).
\] (2)

Regarding the Gini coefficient \( G \), to our knowledge there is no closed-form expression for its expected value \( E(G|n) \) given a number of reads \( n \). Therefore, we resort to a simulation scheme. We repeatedly generate \( n \) reads uniformly distributed along the genome and compute the average of the observed \( G \) in each simulated dataset. Typically, as the number of reads \( n \) is large (tens or hundreds of millions), 1 or 2 simulations are enough to ensure highly precise \( E(G|n) \) estimates, which renders the approach computationally feasible. In order to define the \( n \)-adjusted Gini \( (G_n) \), we notice that the Gini coefficient can be interpreted as the difference between two integrals which compare the observed vs. a uniform cumulative distribution function (cdf). Therefore, we define \( G_n = G - E(G|n) \), so that \( G_n \) is the difference between the observed and the expected cdf under a sample of \( n \) uniformly distributed reads. We note that, although the expected value of \( \text{SD}_n \) and \( G_n \) does not depend on \( n \), their variability will typically increase when \( n \) is small. Hence, for small \( n \) these indexes are less reliable. A rigorous variability assessment can be performed via Bootstrap.

\( \text{SD}_n \) is preferable to \( G_n \) in terms of computational speed, as it does not require any simulations. The examples in Section 3 explore the performance of both indexes in transcription factor ChIP-seq and chromatin mark ChIP-seq studies. Transcription factor ChIP-seq experiments commonly show a relatively reduced number of strong peaks. While some chromatin marks behave in the same manner, some marks are widely spread and show smaller peaks. The results in Section 3 suggest that, while both indexes perform well for transcription factor ChIP-seq, \( \text{SD}_n \) is preferable for chromatin mark ChIP-seq. Hence, we recommend \( \text{SD}_n \) as a default choice.
2.3 Determining rich-read regions and differential expression

In many sequencing experiments it is of interest to screen the genome for regions accumulating a large number of reads. In ChIP-seq studies the screen helps identify and focus on regions with peaks. In RNA-seq studies it may point to previously unknown transcripts, exons or short RNAs. The function `enrichedRegions` implements a computationally efficient scheme to achieve this goal.

We consider setups where the interest lies in a single sample, as well as setups with two or more samples. The basic rationale is to compute the overall read coverage (i.e. across all samples under consideration) and select regions with coverage above a user-specified threshold. We refer to such regions as *islands* and denote them as $i = 1, \ldots, I$. Further, let $J$ be the number of samples, $x_{ij}$ the number of reads in sample $j$ overlapping island $i$, $n_j = \sum_{i=1}^{I} x_{ij}$ the number of reads overlapping some island and $p_{ij} = E\left(\frac{x_{ij}}{n_j}\right)$. Conditioning on $n_j$ and assuming that reads are independent, the marginal distribution of $x_{ij}$ is Bin($n_j, p_{ij}$). This observation suggests a simple approach to test for a significant accumulation of reads in any given island.

In a single sample setup (i.e. $J = 1$), we aim to detect islands with a proportion of reads above average, which can be formalized as testing the null hypothesis $H_0: p_{i1} \leq \frac{1}{J}$ versus the alternative $H_1: p_{i1} > \frac{1}{J}$. A straightforward binomial test is applied to obtain an exact one-sided $P$-value (`enrichedRegions` also allows the user to choose a two-sided version of the test).

In a setup with $\geq 2$ samples, the null hypothesis can be formulated as $H_0: p_{i1} = \ldots = p_{iJ}$. A $P$-value is obtained via a likelihood ratio test comparing the null hypothesis with the unrestricted model $p_{i1} \neq \ldots \neq p_{iJ}$. The likelihood-ratio test may be inexact when the expected number of reads in a sample $\sum_{j=1}^{J} x_{ij}$ under the null hypothesis is low (as a rule of thumb, below 5). Although such cases are rare, because the islands were defined as regions with high coverage to start with, when they do occur `enrichedRegions` provides the option of using permutation-based chi-square tests.

Finally, the user can select several $P$-value adjustment procedures to control for multiple testing (e.g. [Benjamini and Yekutieli (2001)]).

2.4 Detecting the accumulation of significant hits

The function `enrichedChrRegions` looks for chromosome regions accumulating a large number of hits, e.g. peaks in ChIP-seq or differential expression.
in RNA-seq experiments.

Let \( n \) be the number of hits and \( l_i \) for \( i = 1, \ldots, n \) be the location of hit \( i \), given by its chromosome and midpoint between hit start and end. We compute a smoothed number of hits \( s(l) \) by counting the number of hits in a moving window of user-specified size (10kb by default). We then report regions with \( s(l) > t \), where \( t \) is a threshold to control the FDR below a user-specified level (0.05 by default). Similar to the procedure in Section 2.1, an upper bound for FDR(\( t \)) is estimated as

\[
\hat{P}(s(l) \geq t | H_0) \approx \frac{P(s(l) \geq t)}{P(s(l) \geq t)},
\]

where \( \hat{P}(s(l) \geq t) \) is the observed number of regions with \( s(l) \geq t \). For \( P(s(l) \geq t | H_0) \) we simulate \( n \) hits \( l_1^*, \ldots, l_n^* \) uniformly distributed along the genome and compute \( s(l^*) \).

\( \hat{P}(s(l) \geq t | H_0) \) is the mean number of regions with \( s(l^*) \geq t \) averaged across several independent simulations.

3 Examples

3.1 Example 1: GSE25836

The aligned reads of the human ChIP-sequencing experiment GSE25836 were downloaded from [www.ncbi.nlm.nih.gov/geo](http://www.ncbi.nlm.nih.gov/geo). For each sample we read the chromosome, strand and start/end positions of the alignments into R via `read.table` and structured them as a `RangedData` objects. Information from all samples was then combined in a `RangedDataList` object named `gse25836` and saved to an .RData file. We start an R session, load the `htSeqTools` and `multicore` packages and the data.

```r
> library(htSeqTools)
> library(multicore)
> load('gse25836.RData')
> gse25836
RangedDataList of length 7
names(7): GSM634613 GSM634614 GSM634615 GSM634616 GSM634617 GSM634618 GSM634619
> head(gse25836[[1]])
RangedData with 6 rows and 1 value column across 25 spaces

<table>
<thead>
<tr>
<th>space</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>[222562220, 222562251]</td>
<td>-</td>
</tr>
<tr>
<td>chr1</td>
<td>[117797271, 117797302]</td>
<td>+</td>
</tr>
<tr>
<td>chr1</td>
<td>[113171797, 113171828]</td>
<td>+</td>
</tr>
<tr>
<td>chr1</td>
<td>[229997262, 229997293]</td>
<td>-</td>
</tr>
<tr>
<td>chr1</td>
<td>[108697607, 108697638]</td>
<td>+</td>
</tr>
<tr>
<td>chr1</td>
<td>[232944304, 232944335]</td>
<td>+</td>
</tr>
</tbody>
</table>
```
The experiment has 7 samples. Following the workflow in Section 1.3, we start by removing over-amplification artifacts with `filterDuplReads`. We set the FDR at 0.01 and use 2 cores to speed up processing, and report the number of reads before and after filtering.

```r
> gse25836Fil <- filterDuplReads(gse25836, fdrOverAmp=0.01, mc.cores=2)
> sapply(gse25836,nrow)
GSM634613 GSM634614 GSM634615 GSM634616 GSM634617 GSM634618 GSM634619
3081174 5772530 3049264 5623018 2365943 4262704 2243477
> sapply(gse25836Fil,nrow)
GSM634613 GSM634614 GSM634615 GSM634616 GSM634617 GSM634618 GSM634619
3079527 5763439 3047288 5615025 2365943 4261657 2241978
```

For illustration, we show the operations applied by `filterDuplReads` step-by-step. First we count the number of read repetitions, find the 0.999 quantile of the read repeat distribution and estimate the FDR for a series of cutoffs.

```r
> tdr <- tabDuplReads(gse25836,mc.cores=2)
> head(tdr[['GSM634613']])
ans
 1 2 3 4 5 6
2488473 257510 1868 261 101
> q <- sapply(tdr,function(z) which(cumsum(z/sum(z))>.999)[1])
> q
GSM634613.3 GSM634614.4 GSM634615.3 GSM634617.2 GSM634618.2
 3 4 3 4 2 2
GSM634619.3
3
> fdrest <- vector("list",length(tdr)); names(fdrest) <- names(tdr)
> for (i in 1:length(fdrest)) fdrest[[i]] <- fdrEnrichedCounts(tdr[[i]],
use=1:q[i], components=0, mc.cores=2)
> fdrest[[1]][1:7,]
pdfH0 pdfOverall fdrEnriched
1 8.984324e-01 8.982263e-01 1.000000000
2 9.297078e-02 9.294947e-02 0.997974636
3 7.943553e-03 7.941727e-03 0.974225969
4 6.070586e-04 6.742636e-04 0.740222673
5 4.310700e-05 9.420920e-05 0.221891062
6 2.905485e-06 3.645643e-05 0.027348257
7 1.883469e-07 2.129633e-05 0.002589592
> cutoff <- sapply(fdrest,function(z) rownames(z)[z$fdrEnriched<.01][1])
> cutoff
GSM634613 GSM634614 GSM634615 GSM634617 GSM634618 GSM634619
"7" "7" "7" "9" "6" "4"
```

The first sample has 2,488,473 reads appearing only once, 257,510 appearing twice, etc. Its 0.999 quantile for the read repeat distribution is \( q_\alpha = 3 \).
Further, for this sample the removal of reads with ≥ 7 repeats is estimated to have an FDR ≤ 0.0026.

Figure 1 shows the estimated FDR as a function of the number of read repeats for all samples. It is important to note that read repeats are more frequent in IP samples, which is to be expected as they focus on a relatively small part of the genome. That is, using the same cutoff for control and IP samples might be too stringent and result in a loss of precision in the subsequent peak calling. This observation illustrates the advantages of using a data-adaptive cutoff. The code required to produce Figure 1 is shown below.

```r
> lty <- rep(1:2,c(4,3))
> col <- rep(c(1,'gray'),c(4,3))
> plot(fdrest[[1]]$fdrEnriched,type='l',xlab='Number of repeats',
  ylab='Estimated FDR',log='x')
> for (i in 2:length(fdrest)) lines(fdrest[[i]]$fdrEnriched,lty=lty[i],
  col=col[i])
> legend('topright',c('IP','Input'),lty=1:2,col=c(1,'gray'))
```
The next step in the workflow is to assess the similarity between samples via correlation plots. This is achieved with `cmds`, which also allows for parallel computing. Pearson, Spearman and Kendall correlation coefficients are available. In principle, Spearman is more general as it captures non-linear associations, but in practice all options typically produce very similar results.

```r
> cmds1 <- cmds(gse25836Fil, mc.cores=2)
> cmds1Spear <- cmds(gse25836Fil, cor.method='spearman', mc.cores=2)
> dpearson <- cmds1@d[upper.tri(cmds1@d)]
> dspear <- cmds1Spear@d[upper.tri(cmds1Spear@d)]
> cor(dpearson,dspear)
[1] 0.9999551
> col <- rep(c(1,'gray'),c(4,3))
> pch <- c(1,2,1,2,1,2,1)
> n <- c('IP,GAI','IP,GAII','IP,GAI','IP,GAII','IP,GAI','IP,GAII','Input,GAI','Input,GAII')
> plot(cmds1,col=col,pch=pch,labels=n)
```

The visual representation of the correlation distances can be achieved via classical Multi-Dimensional Scaling (MDS) and is shown in Figure 2(a). For comparison, Figure 2(b) shows a hierarchical clustering (complete linkage) dendrogram based on the SeqMonk correlations, which use RPKM in 1,000bp windows (necessary to avoid memory saturation problems in our 8Gb RAM desktop). Although the main features in the two plots are the same, panel (a) reveals some finer details, e.g. the relative position of GAI IP and input samples with respect to the GAII samples.

In order to remove potential strand-specific biases we use `alignPeaks`, which implements a procedure similar to that used by MACS ([Zhang et al., 2008](#)). Again, we make use of parallel computing by setting the argument `mc.cores`. A suitable alternative is provided in the package `chipseq` function `estimate.mean.fraglen`.

```r
> gse25836 <- alignPeaks(gse25836Fil, strand='strand', mc.cores=4)
Estimated shift size is 7.389731
Estimated shift size is 7.99554
Estimated shift size is 0
Estimated shift size is 18.33544
Estimated shift size is 6.199713
Estimated shift size is 0.3745177
Estimated shift size is 0.9554126
```
Figure 2: Sample correlations in GSE25836. (a) htSeqTools MDS plot; (b) SeqMonk hierarchical clustering (1,000bp window RPKM correlation)

In these data most samples require a minor adjustment only. Note that the adjustment tends to be slightly larger for the IP samples (the first 4 samples). In our experience, it is often the case that the stronger peaks observed in IP samples are more prone to strand-specific biases. See Section 3.2 or the htSeqTools vignette for examples where these biases are more evident.

Next we assess the efficiency of the immuno-precipitation with the n-adjusted coverage variability measures $SD_n$ and $G_n$ (Section 2.2). Both indexes are larger in the IP samples than in their respective controls, which suggests an efficient immuno-precipitation.

```r
> sdn <- ssdCoverage(gse25836, mc.cores=6)
> gn <- giniCoverage(gse25836, mc.cores=6, mk.plot=FALSE)
> ans <- data.frame(n,round(cbind(sdn,gn),3))
> ans[c(1,3,5,7,2,4,6),]
n  sdn  gini gini.adjust
GSM634613  IP,GAI 0.115  0.975   0.006
GSM634615  IP,GAI 0.114  0.975   0.006
GSM634617  Input,GAI 0.110  0.978   0.002
GSM634619  Input,GAI 0.106  0.979   0.002
GSM634614  IP,GAII 0.124  0.951   0.014
GSM634616  IP,GAII 0.135  0.960   0.021
GSM634618  Input,GAII 0.120  0.955   0.002
```
After assessing that the data quality is satisfactory, the next step is to find peaks, i.e. chromosomal regions with large concentration of IP reads with respect to the input sample. To this end, we merge reads from all IP samples into a single RangedData, and similarly for the input samples. We use enrichedRegions to find regions where the coverage is greater than minReads=10 (the default) and the proportion of reads in the IP sample falling in the region is significantly higher than in the input sample (Benjamini-Yekutieli adjusted P-value <0.05, see Section 2.3).

```r
> ip <- rbind(gse25836[[1]], gse25836[[2]], gse25836[[3]], gse25836[[4]])
> input <- rbind(gse25836[[5]], gse25836[[6]], gse25836[[7]])
> regions <- enrichedRegions(ip, input, minReads=10, pvalFilter=.05, p.adjust.method='BY', mc.cores=2)
> nrow(regions)
[1] 493
> head(regions)
RangedData with 6 rows and 5 value columns across 25 spaces
<table>
<thead>
<tr>
<th>space ranges</th>
<th>sample1</th>
<th>sample2</th>
<th>pvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>[7634643, 7634691]</td>
<td>30</td>
<td>0.0037205770</td>
</tr>
<tr>
<td>chr1</td>
<td>[8230352, 8230440]</td>
<td>44</td>
<td>0.0000907842</td>
</tr>
<tr>
<td>chr1</td>
<td>[8319253, 8319289]</td>
<td>21</td>
<td>0.0369954205</td>
</tr>
<tr>
<td>chr1</td>
<td>[8319784, 8319877]</td>
<td>52</td>
<td>0.0002467893</td>
</tr>
<tr>
<td>chr1</td>
<td>[8323426, 8323483]</td>
<td>31</td>
<td>0.0369954205</td>
</tr>
<tr>
<td>rpkm1</td>
<td>34.97487</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>rpkm2</td>
<td>28.24188</td>
<td>0.000000</td>
<td></td>
</tr>
</tbody>
</table>
```

We find 493 enriched regions. For each region, the function reports the number of reads, RPKM \cite{Mortazavi2008}, and P-value. We visually inspect the first four detected peaks (Figure 3). The coverage in IP samples (black line) is above that for the controls (grey line) in all selected regions, as expected.

```r
> par(mfrow=c(2,2))
> cov1 <- coverage(ip['chr1']); cov2 <- coverage(input['chr1'])
> for (i in 1:4) {
>   st <- start(regions['chr1'])[i]; en <- end(regions['chr1'])[i]
>   xlim <- c(st-100, en+100)
>   cov1sel <- seqselect(cov1[['chr1']], xlim[1], xlim[2])
>   cov2sel <- seqselect(cov2[['chr1']], xlim[1], xlim[2])
> }
```
Figure 3: Four regions (±100bp) enriched in IP samples for the GSE25836 dataset. The vertical dashed lines indicate the enriched region limits. The lower panels show two detected peaks that were next to each other.
Given that GAI and GAII data appear so separated in the MDS plot (Figure 2), another sensible strategy is to analyze GAI and GAII samples separately and report common findings.

Most of the regions identified with the GAI and GAII samples individually are also found in the combined samples. There are 61 regions identified in both platforms. In order to keep the illustration simple, here we proceed with the results obtained from the merged samples.

Two useful functions related to `enrichedRegions` are `enrichedPeaks` and `islandCounts`. The former selects the highest coverage areas within the enriched regions, which can be useful to define peaks according to more stringent criteria. Below we define peaks as sub-regions with coverage above 25.

```
> peaks <- enrichedPeaks(regions, ip, input, minHeight=25, mc.cores=2)
> nrow(peaks)
[1] 314
> head(peaks)
RangedData with 6 rows and 2 value columns across 25 spaces
   space ranges | height region.pvalue
   <character> <IRanges> | <integer> <numeric>
```
islandCounts finds all regions with overall coverage (i.e. across all samples) above a user-specified threshold. Next it computes the number of reads overlapping with each region. This function is useful to obtain read counts, which can then be analyzed with a number of Bioconductor packages, e.g. BayesPeak [Spyrou et al., 2009], DESeq [Anders and Huber, 2010] and edgeR [Robinson et al., 2010]. Here we show how to combine islandCounts with package edgeR.

```r
> counts <- islandCounts(RangedDataList(ip,input),minReads=10,mc.cores=2)
> head(counts)
RangedData with 6 rows and 2 value columns across 25 spaces
   space ranges | counts1 counts2
<character> <IRanges> | <integer> <integer>
1 chr1 [1336314, 1336315] | 9 1
2 chr1 [1336318, 1336329] | 10 1
3 chr1 [2507532, 2507532] | 10 0
4 chr1 [2507549, 2507551] | 10 0
5 chr1 [5732170, 5732193] | 7 6
6 chr1 [5734147, 5734172] | 13 3
> nrow(counts)
[1] 6572
> countsTable <- cbind(counts[['counts1']],counts[['counts2']])
> library(edgeR)
> d <- DGEList(countsTable,lib.size=c(nrow(ip),nrow(input)),
group=c('ip','input'))
> d <- estimateCommonDisp(d)
Warning message:
In estimateCommonDisp(d) :
  There is no replication. Setting common dispersion to 0.
> de.com <- exactTest(d)
> padj <- p.adjust(de.com$table$p.value,method='BY')
> sel <- (padj<.05) & (de.com$table$logFC>0)
> sum(sel)
[1] 413
> tab <- table(counts[sel,] %in% regions)
> sum(tab[,2])
[1] 380
```

islandCounts finds 6572 regions. The exact test in edgeR determined that 413 are significantly enriched in the IP samples. Of these, 380 (92%)
overlap with the regions found by `enrichedRegions`. Since no technical replicates are available, `edgeR` estimates the variability based on a Poisson model. Below we illustrate how to obtain more precise variance estimates by using the technical replicates obtained with the GAII platform. Note that due to having a smaller number of reads, in this analysis we choose not to adjust P-values in order to preserve statistical power. We identify 61 regions, all of which were also detected in our previous analysis.

```r
> counts.ga2 <- islandCounts(gse25836[c(2,4,6)],minReads=10,mc.cores=4)
> lsize <- sapply(gse25836,nrow)[c(2,4,6)]
> countsTable.ga2 <- cbind(counts.ga2[['GSM634614']],
                         counts.ga2[['GSM634616']],counts.ga2[['GSM634618']])
> d <- DGEList(countsTable.ga2,lib.size=lsize,group=c('ip','ip','input'))
> d <- estimateCommonDisp(d)
> de.com.ga2 <- exactTest(d)
> Comparison of groups: ip - input
> sel <- (de.com.ga2$table$p.value<.05) & (de.com.ga2$table$logFC>0)
> tab <- table(counts.ga2[sel,,] %in% regions)
> sum(tab[,,'TRUE'])
[1] 61
> sum(tab[,,'TRUE'])/sum(tab)
[1] 1
```

We show how to use packages `biomaRt` (Durinck and Huber 2011) and `ChIPpeakAnno` to annotate the regions by finding the closest transcription start site (TSS). The peak distribution with respect to the closest TSS can be visualized with `PeakLocation` and `stdPeakLocation`. Figure 4(a) shows that although most peaks occur close to the TSS, a non-negligible proportion locate further downstream. Since transcripts have different lengths, panel (a) does not reveal their exact location. The distance relative to the transcript length in panel (b) reveals that most downstream peaks locate close to the transcript end.

```r
> library(biomaRt)
> library(ChIPpeakAnno)
> mart <- useMart("ensembl", "hsapiens_gene_ensembl")
> hsanno <- getAnnotation(mart, featureType='TSS')
> peaksanno <- annotatePeakInBatch(peaks, AnnotationData=hsanno,
                                  PeakLocForDistance="middle")
> PeakLocation(peaksanno,peakDistance=10^4)
> stdPeakLocation(peaksanno)
```

As a final step in our proposed ChIP-seq workflow, we look for chromosome regions with a large number of peaks.

We indicate the chromosome lengths in a named vector, use nSims=100 simulations to estimate the FDR and set the mc.cores argument to speed up
computations. We find 32 chromosomal areas where peaks tend to cluster. Figure 5 shows their location.

```r
> library(BSgenome.Hsapiens.UCSC.hg19)
> chrLength <- seqlengths(Hsapiens)[names(ip)]
> chrRegions <- enrichedChrRegions(regions, chrLength=chrLength, fdr=0.05, nSims=100, mc.cores=6)
> nrow(chrRegions)
[1] 32
> plotChrRegions(chrRegions, chrLength=chrLength)
```

### 3.2 Example 2: GSE16926

Here we illustrate the quality control features in htSeqTools with the *S. cerevisiae* ChIP-seq data GSE16926 (www.ncbi.nlm.nih.gov/geo). We read the data into R via `read.table` and stored it into a `RangedDataList` object `gse16926Raw`. We load htSeqTools, multicore and the data. We remove over-amplification artifacts with `filterDuplReads` (using 6 cores to speed up computations).

```r
> library(htSeqTools)
> library(multicore)
```
Similar to the example in Section 3.1, we show the estimated FDR for each possible threshold to declare over-amplification, i.e. maximum number of allowable repeats (Figure 6).

```r
> tdr <- tabDuplReads(gse16926Raw,mc.cores=6)
> q <- sapply(tdr,function(z) which(cumsum(z/sum(z))>.999)[1])
> q
GSM424491.35 GSM424492.rep1.47 GSM424492.rep2.55 GSM424493.rep1.38 35 47 55 38
GSM424493.rep2.35 GSM424494.rep1.91 GSM424494.rep2.36 GSM442537.40 35 91 36 40
> fdrest <- vector("list",length(tdr)); names(fdrest) <- names(tdr)
> for (i in 1:length(fdrest)) fdrest[[i]] <- fdrEnrichedCounts(tdr[[i]], use=1:q[i], components=0, mc.cores=6)
> 
```
The first replicate of sample GSM424494 shows an unusually large number of highly repetitive reads (0.999 quantile = 91 repeats). The sample also stands out in Figure 6. The observation that GSM424494 was immunoprecipitated whereas the rest of the samples were MNase-digested does not explain this finding, as this behavior is not observed in the second replicate. This finding suggests that the first replicate was particularly prone to over-amplification artifacts. By default, `filterDuplReads` adopts a conservative approach and allows for a larger number of read repetitions in this sample. A more aggressive repeat removal can be forced by setting the argument `maxRepeats`.

```r
> gsm424494 <- filterDuplReads(gse16926Raw[[6]], maxRepeats=500, mc.cores=6)
```

Figure 6: Estimated FDR vs. number of read repeats for GSE16926
Figure 7: Sample correlations in GSE16926. (a) htSeqTools MDS plot; (b) SeqMonk hierarchical clustering.

We produce an MDS plot to visualize the correlations between samples (Figure 7(a)). The plot reveals several interesting features. First, IP and MNase-digested samples are clearly separated. Second, the mutant G2 samples are most similar to the wild-type G2 samples. In fact, mutant G2 samples present coverage profiles which are somewhere in between wild-type G2 and wild-type G1 and asynchronous samples. The latter observation is not obvious from the dendrogram in Figure 7(b), which is based on correlations computed with the SeqMonk software. This example illustrates the potential advantages in visualizing similarity (or dissimilarity) matrices via MDS plots.

```R
lab <- c("wt_asyn_23c","wt_G2_37c_mnase","wt_G2_37c_mnase", "mt_G2_37c_mnase", "mt_G2_37c_mnase","wt_G2_orc_chip", "wt_G2_orc_chip","wt_G1_37c_mnase")
plot(cmds1,labels=lab,cex.text=.8,xlim=c(-.2,.4))
```

Next we use alignPeaks to remove strand-specific biases and assess the sample enrichment efficiency via the indexes SD_n and G_n (Section 2.2).

```R
> gse16926 <- alignPeaks(gse16926, strand='strand', mc.cores=6)
Estimated shift size is 10.71125
Estimated shift size is 32.54875
Estimated shift size is 39.20407
Estimated shift size is 22.65083
```
Estimated shift size is 30.42685
Estimated shift size is 41.10548
Estimated shift size is 45.36787
Estimated shift size is 18.31953

```r
> sdn <- ssdCoverage(gse16926, mc.cores=6)
> gn <- giniCoverage(gse16926, mc.cores=6, mk.plot=FALSE)
> data.frame(lab,round(cbind(sdn,gn),3))
```

<table>
<thead>
<tr>
<th>lab</th>
<th>sdn</th>
<th>gini</th>
<th>gini.adjust</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM424491</td>
<td>9.300</td>
<td>0.499</td>
<td>0.360</td>
</tr>
<tr>
<td>GSM424492.rep1</td>
<td>9.388</td>
<td>0.512</td>
<td>0.362</td>
</tr>
<tr>
<td>GSM424492.rep2</td>
<td>9.768</td>
<td>0.530</td>
<td>0.353</td>
</tr>
<tr>
<td>GSM424493.rep1</td>
<td>9.855</td>
<td>0.540</td>
<td>0.396</td>
</tr>
<tr>
<td>GSM424493.rep2</td>
<td>8.378</td>
<td>0.520</td>
<td>0.349</td>
</tr>
<tr>
<td>GSM424494.rep1</td>
<td>24.163</td>
<td>0.655</td>
<td>0.458</td>
</tr>
<tr>
<td>GSM424494.rep2</td>
<td>13.661</td>
<td>0.594</td>
<td>0.346</td>
</tr>
<tr>
<td>GSM442537</td>
<td>9.604</td>
<td>0.490</td>
<td>0.317</td>
</tr>
</tbody>
</table>

Most samples require a strand-bias correction ranging between 20-50 bases, which is non-negligible. Both SD\(_n\) and G\(_n\) suggest that the IP sample GSM424494 (replicate 1) presents the clearest peaks. According to SD\(_n\), GSM424494 replicate 2 is the second sample presenting clearest peaks. These findings suggest that immuno-precipitation of sonicated DNA was more efficient than MNase digestion in isolating the target DNA. This would be consistent with its capacity to produce shorter DNA fragments. It should also be noted that SD\(_n\) gives more consistent results across replicates than G\(_n\), thereby suggesting SD\(_n\) as the default metric of choice.

### 3.3 Example 3: Histone methylation data

In Section 3.1 and 3.2 we showed that MDS correlation plots and the indexes SD\(_n\) and G\(_n\) can help assess inefficient immuno-precipitation of DNA-binding transcription factors. We now explore the case of histone methylation marks, which in some cases may show less pronounced coverage profiles than transcription factors and therefore represent a more challenging scenario.

We obtained ChIP-seq data assessing the genome-wide distribution of histones H3K9Me3 and H3K4Me3 during the \(D.\ melanogaster\) development. For H3K9Me3, aligned reads in BAM format were downloaded from the modEncode project (modEncode 801-809) and imported into R using `scanBAM` from package `Rsamtools` (Morgan and Páges, 2011a). The H3K4Me3 raw data were obtained in FASTQ format from GEO (GSE15292). Reads uniquely aligned with Bowtie to the dm3 genome with up to two mismatches were read into R with the `readAligned` function in package `ShortRead`.

Figure 8 shows an MDS plot to visualize the Pearson correlation between log-coverages for H3K9Me3. Except in Pupae, the plot shows a large separa-
Figure 8: MDS plot for coverage correlation in H3K9Me3 data (modEncode 801-809).
Table 3: SD\textsubscript{n} and G\textsubscript{n} for H3K9Me3 (modEncode 801-809) and H3K4Me3 (GSE15292) ChIP-seq data

<table>
<thead>
<tr>
<th></th>
<th>H3K9Me3</th>
<th></th>
<th>H3K4Me3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SD\textsubscript{n}</td>
<td>G\textsubscript{n}</td>
<td>SD\textsubscript{n}</td>
<td>G\textsubscript{n}</td>
</tr>
<tr>
<td>E0-4, Input</td>
<td>0.631 0.079</td>
<td>0.640 0.098</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E0-4, IP</td>
<td>0.749 0.131</td>
<td>1.157 0.252</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E4-8, Input</td>
<td>0.636 0.195</td>
<td>0.638 0.225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E4-8, IP</td>
<td>0.712 0.147</td>
<td>2.169 0.380</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8-12, Input</td>
<td>0.520 0.112</td>
<td>0.516 0.138</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8-12, IP</td>
<td>0.714 0.190</td>
<td>2.505 0.478</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E12-16, Input</td>
<td>0.774 0.226</td>
<td>0.824 0.265</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E12-16, IP</td>
<td>0.714 0.176</td>
<td>2.520 0.427</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E16-20, Input</td>
<td>0.612 0.115</td>
<td>0.620 0.140</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E16-20, IP</td>
<td>0.764 0.200</td>
<td>1.825 0.411</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E20-24, Input</td>
<td>0.511 0.043</td>
<td>0.509 0.043</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E20-24, IP</td>
<td>0.578 0.078</td>
<td>1.255 0.337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1, Input</td>
<td>0.598 0.138</td>
<td>0.604 0.169</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1, IP</td>
<td>0.508 0.043</td>
<td>0.636 0.179</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2, Input</td>
<td>0.489 0.033</td>
<td>0.483 0.033</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2, IP</td>
<td>0.704 0.211</td>
<td>1.219 0.345</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L3, Input</td>
<td>-</td>
<td>-</td>
<td>0.501 0.084</td>
<td></td>
</tr>
<tr>
<td>L3, IP</td>
<td>-</td>
<td>-</td>
<td>0.858 0.250</td>
<td></td>
</tr>
<tr>
<td>Pupae, Input</td>
<td>0.514 0.130</td>
<td>0.510 0.130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pupae, IP</td>
<td>0.537 0.089</td>
<td>0.602 0.133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adult female, Input</td>
<td>-</td>
<td>-</td>
<td>0.615 0.0734</td>
<td></td>
</tr>
<tr>
<td>Adult female, IP</td>
<td>-</td>
<td>-</td>
<td>1.257 0.300</td>
<td></td>
</tr>
<tr>
<td>Adult male, Input</td>
<td>-</td>
<td>-</td>
<td>0.509 0.121</td>
<td></td>
</tr>
<tr>
<td>Adult male, IP</td>
<td>-</td>
<td>-</td>
<td>0.513 0.072</td>
<td></td>
</tr>
</tbody>
</table>
tion between IP and input samples. That is, the immuno-precipitation was efficient enough to give samples a distinctive coverage profile. The two samples with the lowest number of aligned reads are the E20-24 and L2 inputs, at 1.05 and 0.67 million reads (respectively). These two samples appear as outliers in Figure 8. Regarding H3K4Me3, Figure 9 also shows a clear separation between IP and control samples.

Table 3 shows $SD_n$ and $G_n$ for the H3K9Me3 and H3K4Me3 data. In most developmental stages, the H3K9Me3 IP samples present a larger $SD_n$ than their controls. The IP Pupae sample presents only a slightly larger $SD_n$ than its control, which is consistent with Figure 8 in signaling poor sample separation. The poor separation might be due to the sequencing depth in the Pupae samples being fairly low. $SD_n$ and $G_n$ are designed not to be biased by the sequencing depth, but their variability can be relatively large when few reads are available. In this example, the MDS plot also suggests poor separation and we therefore deem it likely that there were some problems in the Pupae sample preparation. For embryo 12-16 hours (E12-16) and Larva 1 (L1), $SD_n$ is larger in the controls. Figure 8 shows a clear separation between E12-16/L1 IP and control samples. $SD_n$ is useful in signaling that downstream peak calling algorithms may need to be adjusted.
to take into consideration that E12-16/L1 peaks are not as pronounced as in the other samples. The index $G_n$ does not discriminate between IP and input samples as well as $SD_n$. Similar to $SD_n$, $G_n$ flags the Pupae, E12-16 and L1 samples, and the E4-8 sample as well. Regarding H3K4Me3, both $SD_n$ and $G_n$ indicate the presence of stronger peaks than for H3K9Me3. $SD_n$ consistently presents larger values in the IP samples, which agrees with Figure 9 in signaling efficient immuno-precipitation. $G_n$ points in the same direction for all samples, with the exception of E4-8 and the adult male.

All together, these results suggest that the combined use of MDS and our proposed indexes can be a useful diagnosis for histone methylation data, signaling either problems in the sample preparation or the absence of pronounced peaks. Also, $SD_n$ appears to be more sensitive than $G_n$, and we therefore recommend it as a default choice.

3.4 Example 4: Yeast RNA-seq

We illustrate the RNA-seq workflow (Section 1.4) with the Yeast RNA-seq data in the Bioconductor package yeastRNASeq [Lee et al., 2010]. Following the yeastRNASeq vignette, we import the Bowtie aligned reads and format the data as a RangedDataList.

```r
> library(yeastRNASeq)
> library(ShortRead)
> f <- list.files(file.path(system.file(package = "yeastRNASeq"),"reads"),
  pattern="bowtie", full.names=TRUE)
> names(f) <- gsub("\..bowtie.*", ", basename(f))
> names(f)
[1] "mut_1_f" "mut_2_f" "wt_1_f" "wt_2_f"
> aligned <- lapply(f, readAligned, type = "Bowtie")
> aligned
$mut_1_f
class: AlignedRead
length: 423318 reads; width: 26 cycles
chromosome: Scchr05 Scchr15 ... Scchr08 Scchr13
position: 541317 885627 ... 488228 667296
strand: - + ... - +
alignQuality: NumericQuality
alignData varLabels: similar mismatch

$mut_2_f
class: AlignedRead
length: 420848 reads; width: 26 cycles
chromosome: Scchr02 Scchr13 ... Scchr16 Scchr04
position: 787251 120582 ... 719121 790753
strand: + + ... - -
```
alignQuality: NumericQuality
alignData varLabels: similar mismatch

$wt_1_f$
class: AlignedRead
length: 410349 reads; width: 26 cycles
chromosome: Scchr11 Scchr04 ... Scchr05 Scchr10
position: 283246 961713 ... 195595 218358
strand: + - ... + +
alignQuality: NumericQuality
alignData varLabels: similar mismatch

$wt_2_f$
class: AlignedRead
length: 430264 reads; width: 26 cycles
chromosome: Scchr14 Scchr04 ... Scchr13 Scchr06
position: 705342 1283088 ... 17545 254886
strand: - - ... - +
alignQuality: NumericQuality
alignData varLabels: similar mismatch

> al2rd <- function(z) RangedData(IRanges(position(z),position(z)+26),
space=chromosome(z),strand=strand(z))
> seqs <- lapply(aligned,al2rd)
> seqs <- RangedDataList(lapply(aligned,al2rd))
> seqs
RangedDataList of length 4
names(4): mut_1_f mut_2_f wt_1_f wt_2_f

We remove over-amplification artifacts with filterDuplReads, and display the estimated FDR for each threshold on the maximum number of allowable repeats (Figure 10(a)). The estimated FDR is similar across all samples. The MDS coverage correlation plot (Figure 10(b)) reveals a clear separation between wild-type and mutant samples.

> seqsFilt <- filterDuplReads(seqs,mc.cores=4)
> sapply(seqs,nrow)
mut_1_f mut_2_f wt_1_f wt_2_f
423318 420848 410349 430264
> sapply(seqsFilt,nrow)
mut_1_f mut_2_f wt_1_f wt_2_f
418885 416768 409773 429763
> tdr <- tabDuplReads(seqs,mc.cores=4)
> q <- sapply(tdr,function(z) which(cumsum(z/sum(z))>.999)[1])
> fdrest <- vector("list",length(tdr)); names(fdrest) <- names(tdr)
> for (i in 1:length(fdrest)) fdrest[[i]] <- fdrEnrichedCounts(tdr[[i]],
use=1:q[i], components=0, mc.cores=6)
> col <- c(1,1,'grey','grey')

29
Figure 10: Quality control for Yeast RNA-seq data. (a) Estimated FDR vs. number of read repeats (b) MDS plot to visualize correlation between coverages.

As the dataset focuses on novel low abundance and transient RNAs, the goal is to detect RNAs de novo and compare their expression between wild-type and mutant. We use islandCounts to find all genomic regions with \( \geq 10 \) reads (across all samples) and compute the number of reads in each sample.

```r
> islands1 <- islandCounts(seqs, minReads=10, mc.cores=4)
> nrow(islands1)
[1] 29934
> head(islands1)
RangedData with 6 rows and 4 value columns across 17 spaces
  space ranges | mut_1_f mut_2_f wt_1_f wt_2_f
  <character>  <IRanges> | <integer> <integer> <integer> <integer>
```
As shown below, it is possible to compare the total read count in each group using the function `enrichedRegions`. We also illustrate how to analyze the count data with the `edgeR` package. In our opinion, `edgeR` is preferable whenever replicates are available, as it estimates the within-group variability. In this particular example both approaches give similar results. Indeed, all regions found to be statistically significant by `edgeR` are also detected by `enrichedRegions`.

```r
> regions <- RangedData(ranges(islands1))
> regions$sample1 <- islands1$mut_1_f+islands1$mut_2_f
> regions$sample2 <- islands1$wt_1_f+islands1$wt_2_f
> regions <- enrichedRegions(regions=regions,pvalFilter=.05,
p.adjust.method='BY',twoTailed=TRUE)
> head(regions)
RangedData with 6 rows and 5 value columns across 17 spaces
   space ranges                               sample1 sample2 pvalue
  <character> <IRanges> | <integer> <integer> <numeric>
 1 Scchr01 [33456, 34741] | 199 1530 0.000000e+00
 2 Scchr01 [34785, 34935] | 37 144 5.248181e-11
 3 Scchr01 [35204, 35210] | 0 11 3.805142e-02
 4 Scchr01 [35270, 35363] | 11 56 5.213262e-05
 5 Scchr01 [35371, 35508] | 14 112 0.000000e+00
 6 Scchr01 [35530, 35702] | 31 104 3.222074e-06
   rpkm.sample1 rpkm.sample2
  <numeric> <numeric>
 1 267.3110 1840.8981
 2 423.2817 1475.5870
 3 0.0000 2431.4981
 4 202.1480 921.8058
 5 175.2482 1255.7935
 6 309.5425 930.1790
```

> library(edgeR)
> group <- c('mut','mut','wt','wt')
> d <- DGEList(countsTable,lib.size=colSums(countsTable),group=group)
> d <- estimateCommonDisp(d)
> de.com <- exactTest(d)
Comparison of groups: wt - mut
> padj <- p.adjust(de.com$table$p.value,method='BY')
> deislands <- islands1[padj<.05,]

31
The next step in the workflow is to identify genomic features close to the differentially expressed RNAs. We perform this step with the `biomaRt` (Durinck and Huber, 2011) and `ChIPpeakAnno` (Zhu et al., 2011) packages. For illustration purposes, we find the closest transcription start site. This requires adjusting the chromosome names so that they match those provided by `biomaRt`.

```
> library(biomaRt)
> library(ChIPpeakAnno)
> mart <- useMart("ensembl", "scerevisiae_gene_ensembl")
> yeastanno <- getAnnotation(mart, featureType='TSS')
> newchr <- as.character(deIslands$space)
> newchr[newchr=='Scchr01'] <- 'I'
> newchr[newchr=='Scchr02'] <- 'II'
> newchr[newchr=='Scchr03'] <- 'III'
> newchr[newchr=='Scchr04'] <- 'IV'
> newchr[newchr=='Scchr05'] <- 'V'
> newchr[newchr=='Scchr06'] <- 'VI'
> newchr[newchr=='Scchr07'] <- 'VII'
> newchr[newchr=='Scchr08'] <- 'VIII'
> newchr[newchr=='Scchr09'] <- 'IX'
> newchr[newchr=='Scchr10'] <- 'X'
> newchr[newchr=='Scchr11'] <- 'XI'
> newchr[newchr=='Scchr12'] <- 'XII'
> newchr[newchr=='Scchr13'] <- 'XIII'
> newchr[newchr=='Scchr14'] <- 'XIV'
> newchr[newchr=='Scchr15'] <- 'XV'
> newchr[newchr=='Scchr16'] <- 'XVI'
> > newchr[newchr=='Scmito'] <- 'Mito'
> deIslands <- RangedData(IRanges(start(deIslands),end(deIslands)), values=values(deIslands),space=newchr)
> islandsAnno <- annotatePeakInBatch(deIslands, AnnotationData=yeastanno, PeakLocForDistance="middle")
```

The final step in our basic workflow is to find chromosomal regions accumulating differentially expressed RNAs using `enrichedChrRegions`. We identify 62 significantly enriched regions (FDR ≤ 0.05), shown in Figure 11.
Figure 11: Yeast RNA-seq data: genomic regions with high concentration of differentially expressed RNAs
> newchr <- paste('chr',deIslands$space,sep='')
> newchr[newchr=='chrMito'] <- 'chrM'
> deIslands <- RangedData(IRanges(start(deIslands),end(deIslands)),
values=values(deIslands),space=newchr)
> chrLength <- seqlengths(Scerevisiae)[names(deIslands)]
> chrRegions <- enrichedChrRegions(deIslands,chrLength=chrLength,
fdr=0.05,nSims=100,mc.cores=6)
> nrow(chrRegions)
[1] 62
> plotChrRegions(chrRegions,chrLength=chrLength)

References


Durinck,S., Huber,W. biomart: Interface to BioMart databases (e.g. Ensembl, COSMIC, Wormbase and Gramene). R package version 2.6.0.


Lawrence,M., Carey,V., Gentleman,R. rtracklayer: R interface to genome browsers and their annotation tracks, R package version 1.10.6 [http://www.bioconductor.org/packages/2.8/bioc/html/rtracklayer.html]


Urbanek, S. *multicore*: Parallel processing of R code on machines with multiple cores or CPUs, R package version 0.1-5 [http://CRAN.R-project.org/package=multicore](http://CRAN.R-project.org/package=multicore)

