An Introduction to QuasR

ATCGGCCATAC
GGCGAACCGCGGA
ACCGTACAGCCA

0111011110110
01011110010
10101101101

Anita Lerch, Dimos Gaidatzis and Michael Stadler

Modified: January 17 2014. Compiled: May 8, 2014

Contents

1 Introduction

The *QuasR* package (short for Quantify and annotate short reads in R) integrates the functionality of several *R* packages (such as *IRanges* [1] and *Rsamtools*) and external software (e.g. *bowtie*, through the *Rbowtie* package). The package aims to cover the whole analysis workflow of typical ultra-high throughput sequencing experiments, starting from the raw sequence reads, over pre-processing and alignment, up to quantification. A single *R* script can contain all steps of a complete analysis, making it simple to document, reproduce or share the workflow containing all relevant details.

The current *QuasR* release supports the analysis of single read and paired-end ChIP-seq (chromatin immuno-precipitation combined with sequencing), RNA-seq (gene expression profiling by sequencing of RNA) and Bis-seq (measurement of DNA methylation by sequencing of bisulfite-converted genomic DNA) experiments.

2 Preliminaries

2.1 Citing QuasR

If you use *QuasR* [2] in your work, you can cite it as follows:

```
> citation("QuasR")
```

Please use the QuasR reference below to cite the software itself. If
you were using qAlign with Rbowtie as aligner, it can be cited as
Langmead et al. (2009) (unspliced alignments) or Au et al. (2010)
(spliced alignments).

  Anita Lerch, Dimos Gaidatzis and Michael Stadler (2012). QuasR:
  Quantify and Annotate Short Reads in R. R package version 1.4.2
  (unpublished)

  Langmead B, Trapnell C, Pop M, Salzberg SL. Ultrafast and
  memory-efficient alignment of short DNA sequences to the human
  genome. Genome Biology 10(3):R25 (2009).

  Au KF, Jiang H, Lin L, Xing Y, Wong WH. Detection of splice junctions
  from paired-end RNA-seq data by SpliceMap. Nucleic Acids Research,
  38(14):4570-8 (2010).

This free open-source software implements academic research by the
authors and co-workers. If you use it, please support the project by
citing the appropriate journal articles.

2.2   Installation

QuasR is a package for the *R* computing environment and it is assumed that you have already installed *R*. See the *R* project at http://www.r-project.org. To install the latest version of *QuasR*, you will need to be using the latest version of *R*. *QuasR* is part of the Bioconductor project at http://www.bioconductor.org. To get *QuasR* together with its dependencies you can use

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("QuasR")
```

Bioconductor works on a 6-monthly official release cycle. As with other Bioconductor packages, there are always two versions of *QuasR*. Most users will use the current official release version, which will be installed by biocLite if you are using the current version of *R*. There is also a developmental version of *QuasR* that includes new features due for the next official release. The developmental version will be installed if you are using the developmental version of *R*. The official release version always has an even second number (for example 0.2.1), whereas the developmental version has an odd second number (for example 0.3.6).

2.3   Loading of QuasR and other required libraries

In order to run the code examples in this vignette, the *QuasR* library and a few additional libraries need to be loaded:

```
> library(QuasR)
> library(BSgenome)
> library(Rsamtools)
> library(rtracklayer)
> library(GenomicFeatures)
> library(Gviz)
```

2.4   How to get help

Most questions about *QuasR* will hopefully be answered by the documentation or references. If you've run into a question which isn't addressed by the documentation, or you've found a conflict between the documentation and software itself, then there is an active support community which can offer help. The authors of the package (maintainer: Michael Stadler <michael.stadler@fmi.ch>) always appreciate receiving reports of bugs in the package functions or in the documentation. The same goes for well-considered suggestions for improvements. Any other questions or problems concerning *QuasR* should be sent to the Bioconductor mailing list bioconductor@stat.math.ethz.ch. To subscribe to the mailing list, see https://stat.ethz.ch/mailman/listinfo/bioconductor. Please send requests for general assistance and advice to the mailing list rather than to the individual authors. Users posting to the mailing list for the first time should read the helpful posting guide at http://www.bioconductor.org/doc/postingGuide.html. Note that each function in *QuasR* has it's own help page, as described in

the section 3.1. Mailing list etiquette requires that you read the relevant help page carefully before posting a problem to the list.

3 Quick Start

3.1 A brief introduction to R

If you already use *R* and know about its command line interface, just skip this section and continue with section 3.2 on page 7.

The structure of this vignette and in particular this section is based on the excellent user guide of the *limma* package, which we would like to hereby acknowledge. *R* is a program for statistical computing. It is a command-driven language meaning that you have to type commands into it rather than pointing and clicking using a mouse. In this guide it will be assumed that you have successfully downloaded and installed *R* from http://www.r-project.org as well as *QuasR* (see section 2.2). A good way to get started is to type

```
> help.start()
```

at the *R* prompt or, if you're using *R* for Windows, to follow the drop-down menu items *Help* ≻ *Html help*. Following the links *Packages* ≻ *QuasR* from the html help page will lead you to the contents page of help topics for functions in *QuasR*. Before you can use any *QuasR* commands you have to load the package by typing

```
> library(QuasR)
```

at the *R* prompt. You can get help on any function in any loaded package by typing ? and the function name at the *R* prompt, for example

```
> ?preprocessReads
```

or equivalently

```
> help("preprocessReads")
```

for detailed help on the preprocessReads function. The individual function help pages are especially important for listing all the arguments which a function will accept and what values the arguments can take.

A key to understanding *R* is to appreciate that anything that you create in *R* is an *object*. Objects might include data sets, variables, functions, anything at all. For example

```
> x <- 2
```

will create a variable x and will assign it the value 2. At any stage of your *R* session you can type

```
> ls()
```

to get a list of all the objects you have created. You can see the contents of any object by typing the name of the object at the prompt. The following command will print out the contents of x:

```
> x
```

We hope that you can use *QuasR* without having to spend a lot of time learning about the *R* language itself but a little knowledge in this direction will be very helpful, especially when you want to do

something not explicitly provided for in *QuasR* or in the other Bioconductor packages. For more details about the *R* language see *An Introduction to R* which is available from the online help. For more background on using *R* for statistical analysis see [3].

3.2 Sample QuasR session

This is a quick overview of what an analysis could look like for users preferring to jump right into an analysis. The example uses data that is provided with the *QuasR* package, which is first copied to the current working directory, into a subfolder called "extdata":

```
> file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)
```

```
[1] TRUE
```

The sequence files to be analyzed are listed in sampleFile (see section 5.1 on page 12 for details). The sequence reads will be aligned using *bowtie* [4] (from the *Rbowtie* package [5]) to a small reference genome (consisting of three short segments from the hg19 human genome assembly, available in full for example in the *BSgenome.Hsapiens.UCSC.hg19* package). Make sure that you have sufficient disk space, both in your *R* temporary directory (tempdir) as well as to store the resulting alignments (see section 7.2).

```
> sampleFile <- "extdata/samples_chip_single.txt"
> genomeFile <- "extdata/hg19sub.fa"
> proj <- qAlign(sampleFile, genomeFile)
```

```
nodeNames
zin2
   1
```

```
> proj
```

```
Project: qProject
 Options   : maxHits         : 1
             paired          : no
             splicedAlignment: FALSE
             bisulfite       : no
             snpFile         : none
 Aligner   : Rbowtie v1.4.1 (parameters: -m 1 --best --strata)
 Genome    : /tmp/Rtmp6EN4qA/Rbuild71d34031696c/QuasR/vigne.../hg19sub.fa (file)

 Reads     : 2 files, 2 samples (fastq format):
   1. chip_1_1.fq.bz2  Sample1 (phred33)
   2. chip_2_1.fq.bz2  Sample2 (phred33)

 Genome alignments: directory: same as reads
   1. chip_1_1_76fb7d404822.bam
   2. chip_2_1_76fb48423632.bam
```

```
 Aux. alignments: none
```

The `proj` object keeps track of all the information of a sequencing experiment, for example where sequence and alignment files are stored, and what aligner and reference genome was used to generate the alignments.

Now that the alignments have been generated, further analyses can be performed. A quality control report is saved to the "extdata/qc_report.pdf" file using the qQCReport function.

```
> qQCReport(proj, "extdata/qc_report.pdf")
```

The number of alignments per promoter region is quantified using qCount. Genomic coordinates for promoter regions are imported from a gtf file (annotFile) into the GRanges-object with the name promReg:

```
> library(rtracklayer)
> library(GenomicFeatures)
> annotFile <- "extdata/hg19sub_annotation.gtf"
> txStart <- import.gff(annotFile, format="gtf", asRangedData=FALSE,
                        feature.type="start_codon")
> promReg <- promoters(txStart, upstream=500, downstream=500)
> names(promReg) <- mcols(promReg)$transcript_name
> promCounts <- qCount(proj, query=promReg)
> promCounts
```

	width	Sample1	Sample2
TNFRSF18-003	1000	20	4
TNFRSF18-002	1000	20	4
TNFRSF18-001	1000	20	4
TNFRSF4-001	1000	5	2
SDF4-007	1000	8	2
SDF4-001	1000	8	2
SDF4-002	1000	8	2
SDF4-201	1000	8	2
B3GALT6-001	1000	25	274
RPS7-001	1000	121	731
RPS7-008	1000	121	731
RPS7-009	1000	121	731
RPS7-005	1000	121	731
C3orf10-201	1000	176	496
C3orf10-001	1000	176	496
AC034193.1-201	1000	5	2
VHL-001	1000	61	336
VHL-002	1000	61	336
VHL-201	1000	61	336

4   QuasR Overview

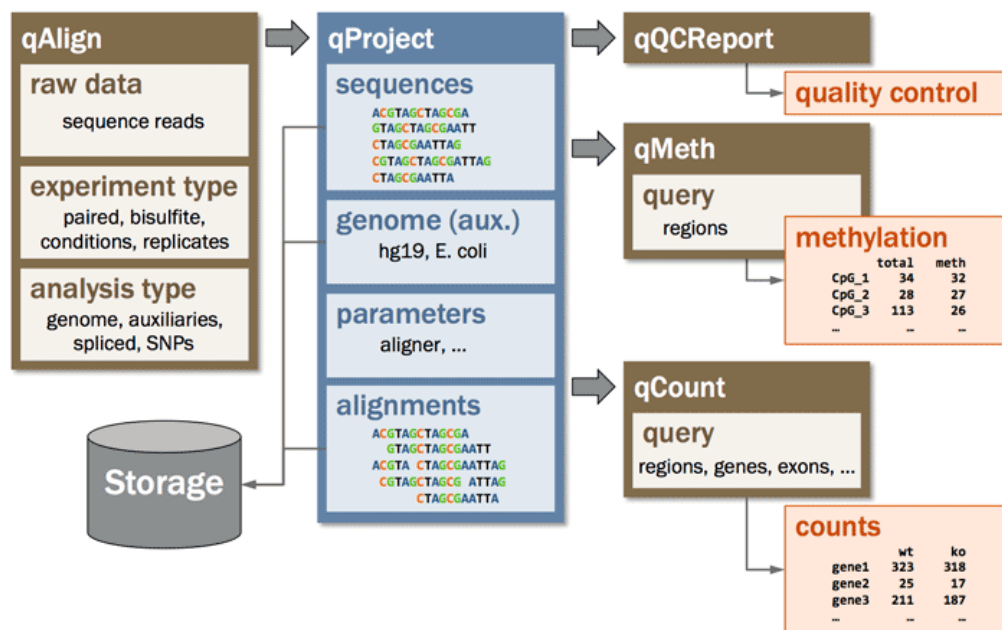The following scheme shows the major components of *QuasR* and their relationships:

Figure 1: QuasR package overview

QuasR works with data (sequences and alignments, reference genome, etc.) that are stored as files on your storage (the gray cylinder on the lower left of Figure 1, see section 4.1 for details on storage locations). *QuasR* does not need a database management system, or these files to be named and organized according to a specific scheme.

In order to keep track of directory paths during an analysis, *QuasR* makes use of a qProject object that is returned by the qAlign function, which at the minimum requires two inputs: the name of a samples text file (see section 5.1 for details), and the reference genome for the alignments (see section 5.3).

The qProject object is the main argument passed to subsequent functions such as qQCReport and qCount. The qProject object contains all necessary information on the current project and eliminates the need to repeatedly enter the same information. All functions that work on qProject objects can be recognized by their names starting with the letter *q*.

Read quantification (apart from quantification of methylation which has its own function qMeth) is done using the qCount function: It counts the alignments in regions of interest (e.g. promoters, genes, exons, etc.) and produces a count table (regions in rows, samples in columns) for further visualization and analysis. The count table can also be used as input to a statistical analysis using packages such as *edgeR* [6], *DESeq* [7], *DESeq2*, *TCC* [8], *DEXSeq* [9] or *baySeq* [10].

In summary, a typical *QuasR* analysis consists of the following steps (some of them are optional):

- preprocessReads (optional): Remove adapters from start or end of reads, filter out reads of low quality, short length or low complexity (section 5.4 on page 16).
- Prepare *samples file*: List sequence files or alignments, provide sample names (section 5.1 on page 12).
- Prepare *auxiliary file* (optional): List additional reference sequences for alignment of reads not matching the reference genome (section 5.2 on page 14).
- qAlign: Create qProject object and specify project parameters. Also download BSgenome package, create aligner indices and align reads if not already existing (7.2 on page 42).
- qQCReport (optional): Create quality control report with plots on sequence qualities and alignment statistics (section 7.4 on page 43).
- qExportWig (optional): Export genomic alignments as wiggle tracks for genome browser visualization (section 7.6 on page 47).
- qCount: Quantify alignments in regions of interest (section 7.7 on page 48).

Recurrent example tasks that may be part of any typical analysis are described in section 5 starting on page 12. Example workflows for specific experiment types (ChIP-seq, RNA-seq and Bis-seq) are described in section 6 starting on page 18.

4.1   File storage locations

Appart from qExportWig and qQCReport, which generate wig files and pdf reports, qAlign is the only function in *QuasR* that stores files on the disk (see section 7.2 for details). All files generated by qAlign are listed here by type, together with their default location and how locations can be changed.

- *Temporary files* (default: tempdir()): Temporary files include reference genomes in FASTA format, decompressed input sequence files, and temporary alignments in text format, and can require a large amount of disk space. By default, these files will be written to the temporary directory of the *R* process (as reported by tempdir()). If using clObj for parallel processing, this may be the tempdir() from the cluster node(s). An alternative location can be set using the TMPDIR environment variable (see ?tempdir).
- *Alignment files (*BAM *format)* (default: same directory as the input sequence files): Alignments against reference genome and auxiliary targets are stored in BAM format in the same directory that also contains the input sequence file (listed in sampleFile). Please note that if the input sequence file correspond to a symbolic link, *QuasR* will follow the link and use the directory of the orignial file instead. An alternative directory can be specified with the alignmentsDir argument from qAlign, which will store all BAM in that directory even if the input sequence files are located in different directories.
- *Alignment index files* (default: depends on genome and snpFile arguments): Many alignment tools including *bowtie* require an index of the reference sequence to perform alignments. If necessary, qAlign will build this index automatically and store it in a default location that depends on the genome argument:
  - BSgenome: If genome is the name of a *BSgenome* package (such as "BSgenome.Hsapiens.

UCSC.hg19"), the index will be stored as a new *R* package in the default library path (as reported by .libPaths()[1], see ?install.packages for details). The name of this index package will be the name of the original [*BSgenome*] package with a suffix for the index type, for example "BSgenome.Hsapiens.UCSC.hg19.Rbowtie".

– FASTA: If genome refers to a reference genome file in FASTA format, the index will be stored in a subdirectory at the same location. Similarly, the indices for files listed in auxiliaryFile are store at the location of these files. For example, the Rbowtie index for the genome at "./genome/mm9.fa" is stored in "./genome/mm9.fa.Rbowtie".

– *Allele-specific analysis*: A special case is the allele-specific analysis, where reference and alternative alleles listed in snpFile (e.g. "./mySNPs.tab") are injected into the genome (e.g. "BSgenome.Mmusculus.UCSC.mm9") to create two variant genomes to be indexed. These indices are saved at the location of the snpFile in a directory named after snpFile, genome and the index type (e.g. "./mySNPs.tab.BSgenome.Mmusculus.UCSC.mm9.A.fa. Rbowtie").

5 Example tasks

5.1 Create a sample file

The sample file is a tab-delimited text file with two or three columns. The first row contains the column names: For a single read experiment, these are 'FileName' and 'SampleName'; for a paired-end experiment, these are 'FileName1', 'FileName2' and 'SampleName'. If the first row does not contain the correctly spelled column names, *QuasR* will not accept the samples file. Subsequent rows contain the input sequence files.

Here are examples of such sample files for a single read experiment:

```
FileName         SampleName
chip_1_1.fq.bz2  Sample1
chip_2_1.fq.bz2  Sample2
```

and for a paired-end experiment:

```
FileName1        FileName2        SampleName
rna_1_1.fq.bz2   rna_1_2.fq.bz2   Sample1
rna_2_1.fq.bz2   rna_2_2.fq.bz2   Sample2
```

These example files are also contained in the *QuasR* package and may be used as templates. The path of the files can be determined using:

```
> sampleFile1 <- system.file(package="QuasR", "extdata",
                        "samples_chip_single.txt")
> sampleFile2 <- system.file(package="QuasR", "extdata",
                        "samples_rna_paired.txt")
```

The columns *FileName* for single-read, or *FileName1* and *FileName2* for paired-end experiments contain paths and names to files containing the sequence data. The paths can be absolute or relative to the location of the sample file. This allows combining files from different directories in a single analysis. For each input sequence file, qAlign will create one alignment file and by default store it in the same directory as the sequence file. Already existing alignment files with identical parameters will not be re-created, so that it is easy to reuse the same sequence files in multiple projects without unnecessarily copying sequence files or recreating alignments.

The *SampleName* column contains sample names for each sequence file. The same name can be used on several lines to indicate multiple sequence files that belong to the same sample (qCount can use this information to automatically combine counts for one sample from multiple files).

Three file formats are supported for input files (but cannot be mixed within a single sample file):

- FASTA files have names that end with '.fa', '.fna' or '.fasta'. They contain only sequences (and no base qualities) and will thus by default be aligned on the basis of mismatches (the best alignment is the one with fewest mismatches).

- FASTQ files have names that end with '.fq' or '.fastq'. They contain sequences and corresponding base qualities and will be aligned by default using these qualities.
- BAM files have names that end with '.bam'. They can be used if the sequence reads have already been aligned outside of *QuasR*, and *QuasR* will only be used for downstream analysis based on the alignments contained in the BAM files. This makes it possible to use alignment tools that are not available within *QuasR*, but making use of this option comes with a risk and should only be used by experienced users. For example, it cannot be guaranteed any more that certain assumptions made by qCount are fulfilled by the external aligner. In addition, since the data provided to *QuasR* has already been processed, quality control plots that are based on unprocessed raw data will be missing from the output of qQCReport.

FASTA and FASTQ files can be compressed with gzip, bzip2 or xz (file extensions '.gz', '.bz2' or 'xz', respectively) and will automatically decompressed when necessary.

Working only with BAM files after performing alignments

Once alignments have been created, most analyses will only require the BAM files and will not access the original raw sequence files anymore. However, re-creating a qProject object by a later identical call to qAlign will still need access to the raw sequences to verify consistency between raw data and alignments. It may be desirable to remove this dependency, for example to archive or move away the raw sequence files and to reclaim used disk space.

This can be achieved using the following procedure involving two sequential calls to qAlign. First, qAlign is called with the orignial sample file (sampleFile1) that lists the raw sequence files, and subsequently with a second sample file (sampleFile2) that lists the BAM files generated in the first call. Such a second sample file can be easily generated given the qProject object (proj1) returned by the first call:

```
> sampleFile1 <- "samples_fastq.txt"
> sampleFile2 <- "samples_bam.txt"
> proj1 <- qAlign(sampleFile1, genomeFile)
> write.table(alignments(proj1)$genome, sampleFile2, sep="\t", row.names=FALSE)
> proj2 <- qAlign(sampleFile2, genomeFile)
```

The analysis can now be exclusively based on the BAM files using sampleFile2 and proj2.

Consistency of samples within a project

The sample file implicitly defines the type of samples contained in the project: *single read* or *paired-end read*, sequences *with* or *without* qualities. This type will have a profound impact on the downstream analysis. For example, it controls whether alignments will be performed in single or paired-end mode, either with or without base qualities. That will also determine availability of certain options for quality control and quantification in qQCReport and qCount. For consistency, it is therefore required that all samples within a project have the same type; it is not possible to mix both single and paired-end read

samples, or FASTA and FASTQ files in a single project (sample file). If necessary, it may be possible to analyse different types of files in separate *QuasR* projects and combine the derived results at the end.

5.2 Create an auxiliary file (optional)

By default *QuasR* aligns reads only to the reference genome. However, it may be interesting to align non-matching reads to further targets, for example to identify contamination from vectors or a different species, or in order to quantify spike-in material not contained in the reference genome. In *QuasR*, such supplementary reference files are called *auxiliary* references and can be specified to qAlign using the auxiliaryFile argument (see section 7.2 on page 42 for details). The format of the auxiliary file is similar to the one of the sample file described in section 5.1: It contains two columns with column names 'FileName' and 'AuxName' in the first row. Additional rows contain names and files of one or several auxiliary references in FASTA format.

An example auxiliary file looks like this:

FileName	AuxName
NC_001422.1.fa	phiX174

and is available from your *QuasR* installation at

```
> auxFile <- system.file(package="QuasR", "extdata", "auxiliaries.txt")
```

5.3 Select the reference genome

Sequence reads are primarily aligned against the reference genome. If necessary, *QuasR* will create an aligner index for the genome. The reference genome can be provided in one of two different formats:

- **a** string, referring to the name of a *BSgenome* package:
  ```
  > available.genomes()
   [1] "BSgenome.Alyrata.JGI.v1"
   [2] "BSgenome.Amellifera.BeeBase.assembly4"
   [3] "BSgenome.Amellifera.UCSC.apiMel2"
   [4] "BSgenome.Amellifera.UCSC.apiMel2.masked"
   [5] "BSgenome.Athaliana.TAIR.04232008"
   [6] "BSgenome.Athaliana.TAIR.TAIR9"
   [7] "BSgenome.Btaurus.UCSC.bosTau3"
   [8] "BSgenome.Btaurus.UCSC.bosTau3.masked"
   [9] "BSgenome.Btaurus.UCSC.bosTau4"
  [10] "BSgenome.Btaurus.UCSC.bosTau4.masked"
  [11] "BSgenome.Btaurus.UCSC.bosTau6"
  [12] "BSgenome.Btaurus.UCSC.bosTau6.masked"
  [13] "BSgenome.Celegans.UCSC.ce10"
  [14] "BSgenome.Celegans.UCSC.ce2"
  [15] "BSgenome.Celegans.UCSC.ce6"
  ```

```
[16] "BSgenome.Cfamiliaris.UCSC.canFam2"
[17] "BSgenome.Cfamiliaris.UCSC.canFam2.masked"
[18] "BSgenome.Cfamiliaris.UCSC.canFam3"
[19] "BSgenome.Cfamiliaris.UCSC.canFam3.masked"
[20] "BSgenome.Dmelanogaster.UCSC.dm2"
[21] "BSgenome.Dmelanogaster.UCSC.dm2.masked"
[22] "BSgenome.Dmelanogaster.UCSC.dm3"
[23] "BSgenome.Dmelanogaster.UCSC.dm3.masked"
[24] "BSgenome.Drerio.UCSC.danRer5"
[25] "BSgenome.Drerio.UCSC.danRer5.masked"
[26] "BSgenome.Drerio.UCSC.danRer6"
[27] "BSgenome.Drerio.UCSC.danRer6.masked"
[28] "BSgenome.Drerio.UCSC.danRer7"
[29] "BSgenome.Drerio.UCSC.danRer7.masked"
[30] "BSgenome.Ecoli.NCBI.20080805"
[31] "BSgenome.Gaculeatus.UCSC.gasAcu1"
[32] "BSgenome.Gaculeatus.UCSC.gasAcu1.masked"
[33] "BSgenome.Ggallus.UCSC.galGal3"
[34] "BSgenome.Ggallus.UCSC.galGal3.masked"
[35] "BSgenome.Ggallus.UCSC.galGal4"
[36] "BSgenome.Ggallus.UCSC.galGal4.masked"
[37] "BSgenome.Hsapiens.NCBI.GRCh38"
[38] "BSgenome.Hsapiens.UCSC.hg17"
[39] "BSgenome.Hsapiens.UCSC.hg17.masked"
[40] "BSgenome.Hsapiens.UCSC.hg18"
[41] "BSgenome.Hsapiens.UCSC.hg18.masked"
[42] "BSgenome.Hsapiens.UCSC.hg19"
[43] "BSgenome.Hsapiens.UCSC.hg19.masked"
[44] "BSgenome.Mmulatta.UCSC.rheMac2"
[45] "BSgenome.Mmulatta.UCSC.rheMac2.masked"
[46] "BSgenome.Mmulatta.UCSC.rheMac3"
[47] "BSgenome.Mmulatta.UCSC.rheMac3.masked"
[48] "BSgenome.Mmusculus.UCSC.mm10"
[49] "BSgenome.Mmusculus.UCSC.mm10.masked"
[50] "BSgenome.Mmusculus.UCSC.mm8"
[51] "BSgenome.Mmusculus.UCSC.mm8.masked"
[52] "BSgenome.Mmusculus.UCSC.mm9"
[53] "BSgenome.Mmusculus.UCSC.mm9.masked"
[54] "BSgenome.Osativa.MSU.MSU7"
[55] "BSgenome.Ptroglodytes.UCSC.panTro2"
[56] "BSgenome.Ptroglodytes.UCSC.panTro2.masked"
[57] "BSgenome.Ptroglodytes.UCSC.panTro3"
[58] "BSgenome.Ptroglodytes.UCSC.panTro3.masked"
[59] "BSgenome.Rnorvegicus.UCSC.rn4"
[60] "BSgenome.Rnorvegicus.UCSC.rn4.masked"
```

```
[61] "BSgenome.Rnorvegicus.UCSC.rn5"
[62] "BSgenome.Rnorvegicus.UCSC.rn5.masked"
[63] "BSgenome.Scerevisiae.UCSC.sacCer1"
[64] "BSgenome.Scerevisiae.UCSC.sacCer2"
[65] "BSgenome.Scerevisiae.UCSC.sacCer3"
[66] "BSgenome.Sscrofa.UCSC.susScr3"
[67] "BSgenome.Sscrofa.UCSC.susScr3.masked"
[68] "BSgenome.Tgondii.ToxoDB.7.0"
[69] "BSgenome.Tguttata.UCSC.taeGut1"
[70] "BSgenome.Tguttata.UCSC.taeGut1.masked"
> genomeName <- "BSgenome.Hsapiens.UCSC.hg19"
```
In this example, the BSgenome package "BSgenome.Hsapiens.UCSC.hg19" refers to an un-masked genome; alignment index and alignments will be performed on the full unmasked genome sequence (recommended). If using a masked genome (e.g. "BSgenome.Hsapiens.UCSC.hg19.masked"), masked regions will be replaced with "N" bases, and this hard-masked version of the genome will be used for creating the alignment index and further alignments.
- **a** file name, referring to a sequence file containing one or several reference sequences (e.g. chromosomes) in FASTA format:
```
> genomeFile <- system.file(package="QuasR", "extdata", "hg19sub.fa")
```

5.4 Sequence data pre-processing

The preprocessReads function can be used to prepare the input sequence files prior to alignment. The function takes one or several sequence files (or pairs of files for a paired-end experiment) in FASTA or FASTQ format as input and produces the same number of output files with the processed reads.

In the following example, we truncate the reads by removing the three bases from the 3'-end (the right side), remove the adapter sequence AAAAAAAAAA from the 5'-end (the left side) and filter out reads that, after truncation and adapter removal, are shorter than 14 bases or contain more than 2 N bases:

```
> td <- tempdir()
> infiles <- system.file(package="QuasR", "extdata",
                         c("rna_1_1.fq.bz2","rna_2_1.fq.bz2"))
> outfiles <- file.path(td, basename(infiles))
> res <- preprocessReads(filename = infiles,
                         outputFilename = outfiles,
                         truncateEndBases = 3,
                         Lpattern = "AAAAAAAAAA",
                         minLength = 14,
                         nBases = 2)
> res

                rna_1_1.fq.bz2 rna_2_1.fq.bz2
totalSequences            3002           3000
matchTo5pAdapter           466            463
```

```
matchTo3pAdapter              0              0
tooShort                    107             91
tooManyN                      0              0
lowComplexity                 0              0
totalPassed                2895           2909
```

```
> unlink(outfiles)
```

preprocessReads returns a matrix with a summary of the pre-processing. The matrix contains one column per (pair of) input sequence files, and contains the total number of reads (totalSequences), the number of reads that matched to the five prime or three prime adapters (matchTo5pAdapter and matchTo3pAdapter), the number of reads that were too short (tooShort), contained too many non-base characters (tooManyN) or were of low sequence complexity (lowComplexity, deactivated by default). Finally, the number of reads that passed the filtering steps is reported in the last row (totalPassed).

In the example below we process paired-end reads, removing all pairs with one or several N bases. Even if only one sequence in a pair fulfills the filtering criteria, both reads in the pair are removed, thereby preserving the matching order of the sequences in the two files:

```
> td <- tempdir()
> infiles1 <- system.file(package="QuasR", "extdata", "rna_1_1.fq.bz2")
> infiles2 <- system.file(package="QuasR", "extdata", "rna_1_2.fq.bz2")
> outfiles1 <- file.path(td, basename(infiles1))
> outfiles2 <- file.path(td, basename(infiles2))
> res <- preprocessReads(filename=infiles1,
                         filenameMate=infiles2,
                         outputFilename=outfiles1,
                         outputFilenameMate=outfiles2,
                         nBases=0)
> res
```

```
                  rna_1_1.fq.bz2:rna_1_2.fq.bz2
totalSequences                             3002
matchTo5pAdapter                             NA
matchTo3pAdapter                             NA
tooShort                                      0
tooManyN                                      3
lowComplexity                                 0
totalPassed                                2999
```

```
> unlink(c(outfiles1,outfiles2))
```

More details on the preprocessReads function can be found in the function documentation (see ?preprocessReads) or in the section 7.1 on page 42.

6   Example workflows

6.1   ChIP-seq: Measuring protein-DNA binding and chromatin modifications

Here we show an exemplary single-end ChIP-seq workflow using a small number of reads from a histone 3 lysine 4 trimethyl (H3K4me3) ChIP-seq experiment. This histone modification is known to locate to genomic regions with a high density of CpG dinucleotides (so called CpG islands); about 60% of mammalian genes have such a CpG island close to their transcript start site. All necessary files are included in the *QuasR* package, and we start the example workflow by copying those files into the current working directly, into a subfolder called "extdata":

```
> file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)
```

```
[1] TRUE
```

6.1.1   Align reads using the `qAlign` function

We assume that the sequence reads have already been pre-processed as described in section 5.4. Also, a sample file (section 5.1) that lists all sequence files to be analyzed has been prepared. A FASTA file with the reference genome sequence(s) is also available (section 5.3), as well as a auxiliary file for alignment of reads that failed to match the reference genome (section 5.2).

By default, newly generated BAM files will be stored at the location of the input sequence files, which should be writable and have sufficient capacity (an alternative location can be specified using the `alignmentsDir` argument). Make also sure that you have sufficient temporary disk space for intermediate files in `tempdir()` (see section 7.2). We start by aligning the reads using qAlign:

```
> sampleFile <- "extdata/samples_chip_single.txt"
> auxFile <- "extdata/auxiliaries.txt"
> genomeFile <- "extdata/hg19sub.fa"
> proj1 <- qAlign(sampleFile, genome=genomeFile, auxiliaryFile=auxFile)

nodeNames
zin2
   1

> proj1

Project: qProject
 Options    : maxHits         : 1
              paired          : no
              splicedAlignment: FALSE
              bisulfite       : no
              snpFile         : none
 Aligner    : Rbowtie v1.4.1 (parameters: -m 1 --best --strata)
 Genome     : /tmp/Rtmp6EN4qA/Rbuild71d34031696c/QuasR/vigne.../hg19sub.fa (file)
```

```
Reads      : 2 files, 2 samples (fastq format):
  1. chip_1_1.fq.bz2  Sample1 (phred33)
  2. chip_2_1.fq.bz2  Sample2 (phred33)

Genome alignments: directory: same as reads
  1. chip_1_1_76fb7d404822.bam
  2. chip_2_1_76fb48423632.bam

Aux. alignments: 1 file, directory: same as reads
  a. /tmp/Rtmp6EN4qA/Rbuild71d34031696c/QuasR/vignet.../NC_001422.1.fa  phiX174
     1. chip_1_1_76fb723f15b9.bam
     2. chip_2_1_76fb1cd44b73.bam
```

qAlign will build alignment indices if they do not yet exist (by default, if the genome and auxiliary sequences are given in the form of FASTA files, they will be stored in the same folder). The qProject object (proj1) returned by qAlign now contains all information about the ChIP-seq experiment: the (optional) project name, the project options, aligner package, reference genome, and at the bottom the sequence and alignment files. For each input sequence file, there will be one BAM file with alignments against the reference genome, and one for each auxiliary target sequence with alignments of reads without genome hits. Our auxFile contains a single auxiliary target sequence, so we expect two BAM files per input sequence file:

```
> list.files("extdata", pattern=".bam$")
```

```
[1] "chip_1_1_76fb723f15b9.bam" "chip_1_1_76fb7d404822.bam"
[3] "chip_2_1_76fb1cd44b73.bam" "chip_2_1_76fb48423632.bam"
```

The BAM file names consist of the base name of the sequence file with an added random string. The random suffix makes sure that newly generated alignment files do not overwrite existing ones, for example of the same reads aligned against an alternative reference genome. Each alignment file is accompanied by two additional files with suffixes ".bai" and ".txt":

```
> list.files("extdata", pattern="^chip_1_1_")[1:3]
```

```
[1] "chip_1_1_76fb723f15b9.bam"     "chip_1_1_76fb723f15b9.bam.bai"
[3] "chip_1_1_76fb723f15b9.bam.txt"
```

The ".bai" file is the BAM index used for fast access by genomic coordinate. The ".txt" file contains all the parameters used to generate the corresponding BAM file. Before new alignments are generated, qAlign will look for available ".txt" files in default locations (the directory containing the input sequence file, or the value of alignmentsDir), and read their contents to determine if a compatible BAM file already exists. A compatible BAM file is one with the same reads and genome, aligned using the same aligner and identical alignment parameters. If a compatible BAM file is not found, or the ".txt" file is missing, qAlign will generate a new BAM file. It is therefore recommended not to delete the ".txt" file - without it, the corresponding BAM file will become unusable for *QuasR*.

6.1.2 Create a quality control report

QuasR can produce a quality control report in the form of a series of diagnostic plots with details on sequences and alignments (see Figure 1 on page 9). The plots are generated by calling the qQCReport function with the qProject object as argument. qQCReport uses *ShortRead* [11] internally to obtain some of the quality metrics, and some of the plots are inspired by the FastQC quality control tool by Simon Andrews (http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/). The plots will be stored into a multipage PDF document defined by the pdfFilename argument, or else shown as individual plot windows on the current graphics device. In order to keep the running time reasonably short, some quality metrics are obtained from a random sub-sample of the sequences or alignments.

```
> qQCReport(proj1, pdfFilename="extdata/qc_report.pdf")
```

Currently available plots are described in section 7.4 on page 43 and following.

6.1.3 Alignment statistics

The alignmentStats gets the number of (un-)mapped reads for each sequence file in a qProject object, by reading the BAM file indices, and returns them as a data.frame. The function also works for arguments of type character with one or several BAM file names (for details see section 7.5 on page 47).

```
> alignmentStats(proj1)
```

	seqlength	mapped	unmapped
Sample1:genome	95000	2339	258
Sample2:genome	95000	3609	505
Sample1:phiX174	5386	251	7
Sample2:phiX174	5386	493	12

6.1.4 Export genome wig file from alignments

For visualization in a genome browser, alignment coverage along the genome can be exported to a (compressed) wig file using the qExportWig function. The created fixedStep wig file (see http://genome.ucsc.edu/goldenPath/help/wiggle.html for details on the wig format) will contain one track per sample in the qProject object. The resolution is defined using the binsize argument, and if scaling is set to TRUE, read counts per bin are scaled by the total number of aligned reads in each sample to improve comparability:

```
> qExportWig(proj1, binsize=100L, scaling=TRUE, collapseBySample=TRUE)
```

6.1.5 Count alignments using qCount

Alignments are quantified using qCount, for example using a GRanges object as a query. In our H3K4me3 ChIP-seq example, we expect the reads to occur around the transcript start site of genes. We can

therefore construct suitable query regions using genomic intervals around the start sites of known genes. In the code below, this is achieved with help from the *GenomicFeatures* package: We first create a TranscriptDb object from a ".gtf" file with gene annotation. With the promoters function, we can then create the GRanges object with regions to be quantified. Finally, because most genes consist of multiple overlapping transcripts, we select the first transcript for each gene:

```
> library(GenomicFeatures)
> annotFile <- "extdata/hg19sub_annotation.gtf"
> chrLen <- scanFaIndex(genomeFile)
> chrominfo <- data.frame(chrom=as.character(seqnames(chrLen)),
                          length=width(chrLen),
                          is_circular=rep(FALSE, length(chrLen)))
> txdb <- makeTranscriptDbFromGFF(file=annotFile, format="gtf",
                                  exonRankAttributeName="exon_number",
                                  gffGeneIdAttributeName="gene_name",
                                  chrominfo=chrominfo,
                                  dataSource="Ensembl",
                                  species="Homo sapiens")
> promReg <- promoters(txdb, upstream=1000, downstream=500,
                       columns=c("gene_id","tx_id"))
> gnId <- sapply(mcols(promReg)$gene_id, paste, collapse=",")
> promRegSel <- promReg[ match(unique(gnId), gnId) ]
> names(promRegSel) <- unique(gnId)
> head(promRegSel)

GRanges with 6 ranges and 2 metadata columns:
                  seqnames            ranges strand |           gene_id       tx_id
                     <Rle>         <IRanges>  <Rle> | <CharacterList> <integer>
  ENSG00000176022     chr1 [31629, 33128]        + | ENSG00000176022         1
  ENSG00000186891     chr1 [ 6452,  7951]        - | ENSG00000186891         2
  ENSG00000186827     chr1 [14013, 15512]        - | ENSG00000186827         6
  ENSG00000078808     chr1 [31882, 33381]        - | ENSG00000078808         9
  ENSG00000171863     chr2 [ 1795,  3294]        + | ENSG00000171863        17
  ENSG00000252531     chr2 [ 7160,  8659]        + | ENSG00000252531        26
  ---
  seqlengths:
    chr1  chr2  chr3
   40000 10000 45000
```

Using promRegSel object as query, we can now count the alignment per sample in each of the promoter windows.

```
> cnt <- qCount(proj1, promRegSel)
> cnt

                  width Sample1 Sample2
ENSG00000176022   1500     157     701
```

```
ENSG00000186891   1500      22      5
ENSG00000186827   1500      10      3
ENSG00000078808   1500      73    558
ENSG00000171863   1500      94    339
ENSG00000252531   1500      59      9
ENSG00000247886   1500     172    971
ENSG00000254999   1500     137    389
ENSG00000238642   1500       8      3
ENSG00000134086   1500       9     18
ENSG00000238345   1500      13     25
ENSG00000134075   1500       7      3
```

The counts returned by qCount are the raw number of alignments per sample and region, without any normalization for the query region length, or the total number of aligned reads in a sample. As expected, we can find H3K4me3 signal at promoters of a subset of the genes with CpG island promoters, which we can visualize with help of the *Gviz* package:

```
> gr1 <- import("Sample1.wig.gz", asRangedData=FALSE)
> gr2 <- import("Sample2.wig.gz", asRangedData=FALSE)
> library(Gviz)
> axisTrack <- GenomeAxisTrack()
> dTrack1 <- DataTrack(range=gr1, name="Sample 1", type="h")
> dTrack2 <- DataTrack(range=gr2, name="Sample 2", type="h")
> txTrack <- GeneRegionTrack(txdb, name="Transcripts", showId=TRUE)
> plotTracks(list(axisTrack, dTrack1, dTrack2, txTrack),
              chromosome="chr3", extend.left=1000)
```
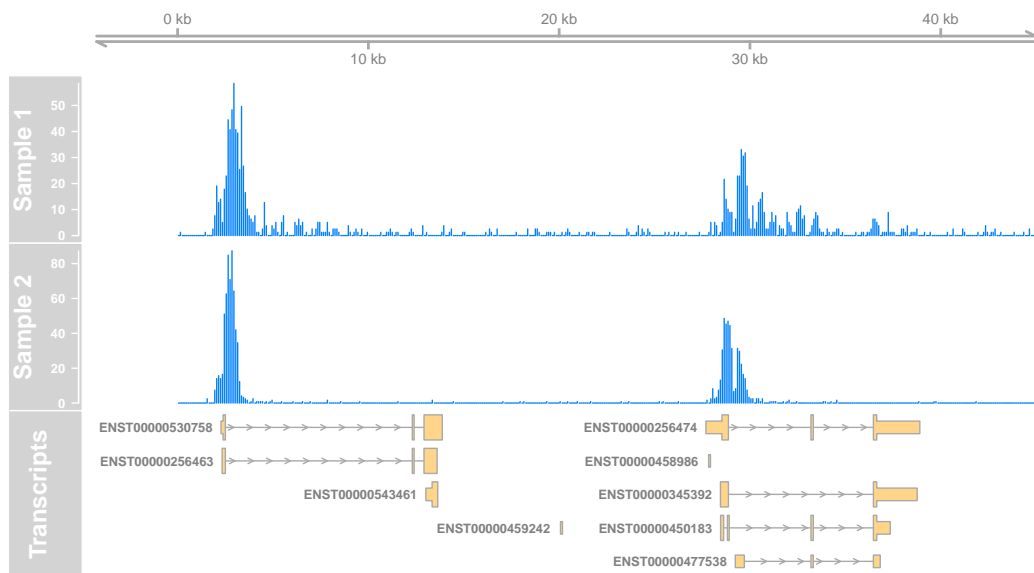
6.1.6   Create a genomic profile for a set of regions using `qProfile`

Given a set of anchor positions in the genome, qProfile calculates the number of nearby alignments relative to the anchor position, for example to generate a average profile. The neighborhood around anchor positions can be specified by the upstream and downstream argument. Alignments that are upstream of an anchor position will have a negative relative position, and downstream alignments a positive. The anchor positions are all aligned at position zero in the return value.

Anchor positions will be provided to qProfile using the query argument, which takes a GRanges object. The anchor positions correspond to start() for regions on "+" or "*" strands, and to end() for regions on the "-" strand. As mentioned above, we expect H3K4me3 ChIP-seq alignments to be enriched around the transcript start site of genes. We can therefore construct a suitable query object from the start sites of known genes. In the code below, start sites ('start_codon') are imported from a ".gtf" file with the help of the *rtracklayer* package. In addition, 'strand' and 'gene_name' are also selected for import. Duplicated start sites, e.g. from genes with multiple transcripts, are removed. Finally, all regions are given the name "TSS", because qProfile combines regions with identical names into a single profile.

```
> library(rtracklayer)
> annotationFile <- "extdata/hg19sub_annotation.gtf"
> tssRegions <- import.gff(annotationFile, format="gtf",
                            asRangedData=FALSE,
                            feature.type="start_codon",
                            colnames=c("strand", "gene_name"))
> tssRegions <- tssRegions[!duplicated(tssRegions)]
> names(tssRegions) <- rep("TSS", length(tssRegions))
> head(tssRegions)

GRanges with 6 ranges and 1 metadata column:
      seqnames            ranges strand |   gene_name
         <Rle>         <IRanges>  <Rle> | <character>
  TSS     chr1 [ 6949,  6951]        - |    TNFRSF18
  TSS     chr1 [14505, 14507]        - |     TNFRSF4
  TSS     chr1 [29171, 29173]        - |        SDF4
  TSS     chr1 [32659, 32661]        + |     B3GALT6
  TSS     chr2 [ 3200,  3202]        + |        RPS7
  TSS     chr3 [ 2386,  2388]        + |     C3orf10
  ---
  seqlengths:
   chr1 chr2 chr3
     NA   NA   NA
```

Alignments around the tssRegions coordinates are counted in a window defined by the upstream and downstream arguments, which specify the number of bases to include around each anchor position. For query regions on "+" or "*" strands, upstream refers to the left side of the anchor position (lower coordinates), while for regions on the "-" strand, upstream refers to the right side (higher coordinates). The following example creates separate profiles for alignments on the *same* and on the *opposite* strand

of the regions in query.

```
> prS <- qProfile(proj1, tssRegions, upstream=3000, downstream=3000,
                  orientation="same")
> prO <- qProfile(proj1, tssRegions, upstream=3000, downstream=3000,
                  orientation="opposite")
> lapply(prS, "[", , 1:10)

$coverage
-3000 -2999 -2998 -2997 -2996 -2995 -2994 -2993 -2992 -2991
    8     8     8     8     8     8     8     8     8     8

$Sample1
-3000 -2999 -2998 -2997 -2996 -2995 -2994 -2993 -2992 -2991
    1     0     0     0     0     0     0     0     0     0

$Sample2
-3000 -2999 -2998 -2997 -2996 -2995 -2994 -2993 -2992 -2991
    0     0     0     2     0     0     1     1     1     0
```

The counts returned by qProfile are the raw number of alignments per sample and position, without any normalization for the number of query regions or the total number of alignments in a sample per position. To obtain the average number of alignments, we divide the alignment counts by the number of query regions that covered a given relative position around the anchor sites. This coverage is available as the first element in the return value. The shift between *same* and *opposite* strand alignments is indicative for the average length of the sequenced ChIP fragments.

```
> prCombS <- do.call("+", prS[-1]) /prS[[1]]
> prCombO <- do.call("+", prO[-1]) /prO[[1]]
> plot(as.numeric(colnames(prCombS)), filter(prCombS[1,], rep(1/100,100)),
       type=l, xlab="Position relative to TSS", ylab="Mean no. of alignments")
> lines(as.numeric(colnames(prCombO)), filter(prCombO[1,], rep(1/100,100)),
        type=l, col="red")
> legend(title="strand", legend=c("same as query","opposite of query"),
         x="topleft", col=c("black","red"), lwd=1.5, bty="n", title.adj=0.1)
```
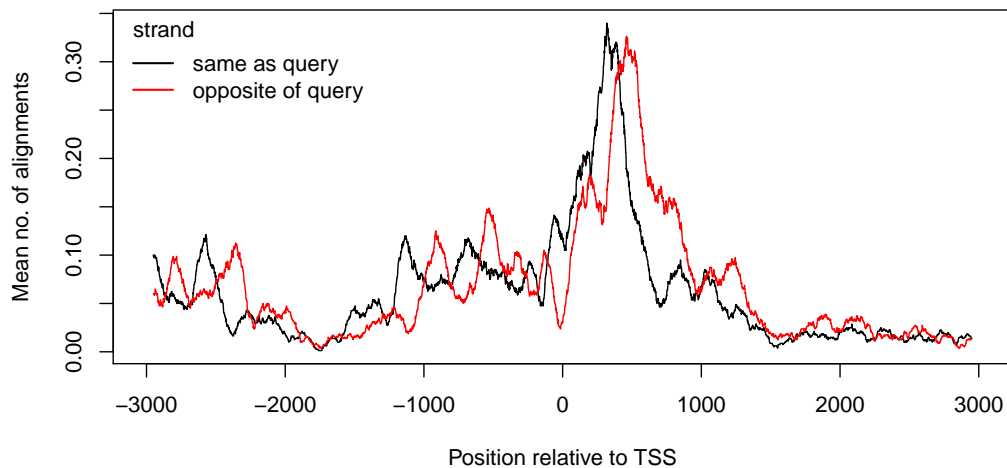
6.1.7 Using a BSgenome package as reference genome

QuasR also allows using of *BSgenome* packages instead of a FASTA file as reference genome (see section 5.3). To use a *BSgenome*, the genome argument of qAlign is set to a string matching the name of a *BSgenome* package, for example "BSgenome.Hsapiens.UCSC.hg19". If that package is not already installed, qAlign will check if it is available from bioconductor.org and download it automatically. The corresponding alignment index will be saved as a new package, named after the original *BSgenome* package and the aligner used to build the index, for example BSgenome.Hsapiens.UCSC.hg19.Rbowtie.

The code example below illustrates the use of a *BSgenome* reference genome for the same example data as above. Running it for the first time will take several hours in order to build the aligner index:

```
> file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)
> sampleFile <- "extdata/samples_chip_single.txt"
> auxFile <- "extdata/auxiliaries.txt"
> available.genomes() # list available genomes
> genomeName <- "BSgenome.Hsapiens.UCSC.hg19"
> proj1 <- qAlign(sampleFile, genome=genomeName, auxiliaryFile=auxFile)
> proj1
```

6.2 RNA-seq: Gene expression profiling

In *QuasR*, an analysis workflow for an RNA-seq dataset is very similar to the one described above for a ChIP-seq experiment. The major difference is that here reads are aligned using splicedAlignment=TRUE, which will cause qAlign to align reads with SpliceMap[12], rather than bowtie[4] (both are contained in the *Rbowtie* package). SpliceMap and *QuasR* also support spliced paired-end alignments; the splicedAlignment argument can be freely combined with the paired argument.

6.2.1 Spliced-alignment of RNA-seq reads

We start the example workflow by copying the example data files into the current working directly, into a subfolder called "extdata", and then create spliced alignments using qAlign:

```
> file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)

[1] TRUE

> sampleFile <- "extdata/samples_rna_paired.txt"
> genomeFile <- "extdata/hg19sub.fa"
> proj2 <- qAlign(sampleFile, genome=genomeFile, splicedAlignment=TRUE)

nodeNames
zin2
   1

> proj2

Project: qProject
 Options    : maxHits         : 1
              paired          : fr
              splicedAlignment: TRUE
              bisulfite       : no
              snpFile         : none
 Aligner    : Rbowtie v1.4.1 (parameters: -max_intron 400000 -min_intron 20000 -max_multi_hit
 Genome     : /tmp/Rtmp6EN4qA/Rbuild71d34031696c/QuasR/vigne.../hg19sub.fa (file)

 Reads      : 2 pairs of files, 2 samples (fastq format):
   1. rna_1_1.fq.bz2  rna_1_2.fq.bz2  Sample1 (phred33)
   2. rna_2_1.fq.bz2  rna_2_2.fq.bz2  Sample2 (phred33)

 Genome alignments: directory: same as reads
   1. rna_1_1_76fb43133028.bam
   2. rna_2_1_76fb3ec7e58b.bam

 Aux. alignments: none
```

Aligning the reads with splicedAlignment=TRUE is much slower than the default, but will allow to also align reads that cross one or two exon junctions, and thus have a large deletion (the intron) relative to the reference genome.

```
> proj2unspl <- qAlign(sampleFile, genome=genomeFile,
                       splicedAlignment=FALSE)

nodeNames
zin2
   1

> alignmentStats(proj2)
```

```
           seqlength mapped unmapped
Sample1:genome    95000   6002        2
Sample2:genome    95000   6000        0

> alignmentStats(proj2unspl)

           seqlength mapped unmapped
Sample1:genome    95000   2258     3746
Sample2:genome    95000   2652     3348
```

6.2.2 Quantification of gene and exon expression

As with ChIP-seq experiments, qCount is used to quantify alignments. For quantification of gene or exon expression levels, qCount can be called with a query of type TranscriptDB, such as the one we constructed in the ChIP-seq workflow above from a ".gtf" file. The argument reportLevel can be used to control if annotated exonic regions should be quantified independently (reportLevel="exon") or non-redundantly combined per gene (reportLevel="gene"):

```
> geneLevels <- qCount(proj2, txdb, reportLevel="gene")
> exonLevels <- qCount(proj2, txdb, reportLevel="exon")
> head(geneLevels)

                  width Sample1 Sample2
ENSG00000078808    4697     708    1076
ENSG00000134075     589    1201    1322
ENSG00000134086    4213     282     295
ENSG00000171863    5583    2922    2224
ENSG00000176022    2793      62     344
ENSG00000186827    1721      37       8

> head(exonLevels)

     width Sample1 Sample2
1     2793      62     344
10     187       1       0
11     307       1       0
12     300       9       2
13     493      18       2
14     129       5       0
```

6.2.3 Calculation of RPKM expression values

The values returned by qCount are the number of alignments. Sometimes it is required to normalize for the length of query regions, or the size of the libraries. For example, gene expression levels in the form of *RPKM* values (reads per kilobase of transcript and million mapped reads) can be obtained as follows:

```
> geneRPKM <- t(t(geneLevels[,-1] /geneLevels[,1] *1000)
              /colSums(geneLevels[,-1]) *1e6)
> geneRPKM
```

```
                Sample1 Sample2
ENSG00000078808   21281   31628
ENSG00000134075  287879  309883
ENSG00000134086    9450    9667
ENSG00000171863   73892   54998
ENSG00000176022    3134   17005
ENSG00000186827    3035     642
ENSG00000186891    2679     202
ENSG00000238345       0       0
ENSG00000238642       0       0
ENSG00000247886       0       0
ENSG00000252531    5197    1694
ENSG00000254999  210572  220589
```

Please note the RPKM values in our example are higher than what you would usually get for a real RNA-seq dataset. The values here are artificially scaled up because our example data contains reads only for a small number of genes.

6.2.4   Analysis of alternative splicing: Quantification of exon-exon junctions

Exon-exon junctions can be quantified by setting reportLevel="junction". In this case, qCount will ignore the query argument and scan all alignments for any detected splices, which are returned as a GRanges object: The region start and end coordinates correspond to the first and last bases of the intron, and the counts are returned in the mcols() of the GRanges object. Alignments that are identically spliced but reside on opposite strands will be quantified separately. In an unstranded RNA-seq experiment, this may give rise to two separate counts for the same intron, one each for the supporting alignments on plus and minus strands.

```
> exonJunctions <- qCount(proj2, NULL, reportLevel="junction")
> exonJunctions
```

```
GRanges with 52 ranges and 2 metadata columns:
      seqnames            ranges strand  |   Sample1    Sample2
         <Rle>         <IRanges>  <Rle>  | <numeric>  <numeric>
  [1]     chr1 [12213, 12321]       +  |         1          0
  [2]     chr1 [12213, 12321]       -  |         2          0
  [3]     chr1 [13085, 13371]       -  |         1          0
  [4]     chr1 [18069, 18837]       +  |         8          7
  [5]     chr1 [18069, 18837]       -  |         6         12
  ...      ...               ...    ... ...       ...        ...
 [48]     chr1 [14166, 14362]       -  |         0          1
 [49]     chr1 [19009, 19148]       -  |         0          2
```

```
  [50]     chr1 [29327, 32271]      -   |            0            2
  [51]     chr1 [ 4617,  4778]      -   |            0            1
  [52]     chr3 [ 2504,  5589]      +   |            0            2
  ---
  seqlengths:
   chr1 chr2 chr3
     NA   NA   NA
```

About half of the exon-exon junctions detected in this sample datset correspond to known introns; they tend to be the ones with higher coverage:

```
> knownIntrons <- unlist(intronsByTranscript(txdb))
> isKnown <- overlapsAny(exonJunctions, knownIntrons, type="equal")
> table(isKnown)

isKnown
FALSE  TRUE
   28    24

> tapply(rowSums(as.matrix(mcols(exonJunctions))),
        isKnown, summary)

$FALSE
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0     2.0     5.0    32.2    47.8   177.0


$TRUE
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0     1.0    10.5    42.7    60.0   209.0
```

When quantifying exon junctions, only spliced alignments will be included in the quantification. It is also possible to only include unspliced alignments in the quantification, for example when counting exon body alignments that complement the exon junction alignments. This can be done using the includeSpliced argument from qCount:

```
> exonBodyLevels <- qCount(proj2, txdb, reportLevel="exon", includeSpliced=FALSE)
> summary(exonLevels - exonBodyLevels)

     width        Sample1         Sample2
 Min.   :0   Min.   :  0    Min.   :  0
 1st Qu.:0   1st Qu.:  0    1st Qu.:  0
 Median :0   Median :  3    Median :  3
 Mean   :0   Mean   : 37    Mean   : 29
 3rd Qu.:0   3rd Qu.: 26    3rd Qu.: 31
 Max.   :0   Max.   :735    Max.   :564
```

6.3   smRNA-seq: small RNA and miRNA expression profiling

Expression profiling of miRNAs differs only slightly from the profiling of mRNAs. There are a few details that need special care, which are outlined in this section.

6.3.1   Preprocessing of small RNA (miRNA) reads

Again, we start the example workflow by copying the example data files into the current working directly, into a subfolder called "extdata".

```
> file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)
```

[1] TRUE

As a next step, we need to remove library adapter sequences from short RNA reads. Most sequencing experiments generate reads that are longer than the average length of a miRNA (22nt). Therefore, the read sequence will run through the miRNA into the library adapter sequence and would not match when aligned in full to the reference genome.

We can remove those adapter sequences using preprocessReads (see section 7.1 on page 42 for more details), which for each input sequence file will generate an output sequence file containing appropriately truncated sequences. In the example below, we get the input sequence filenames from sampleFile, and also prepare an updated sampleFile2 that refers to newly generated processed sequence files:

```
> # prepare sample file with processed reads filenames
> sampleFile <- file.path("extdata", "samples_mirna.txt")
> sampleFile
```

[1] "extdata/samples_mirna.txt"

```
> sampleFile2 <- sub(".txt", "_processed.txt", sampleFile)
> sampleFile2
```

[1] "extdata/samples_mirna_processed.txt"

```
> tab <- read.delim(sampleFile, header=TRUE, as.is=TRUE)
> tab
```

```
    FileName SampleName
1 mirna_1.fa     miRNAs
```

```
> tab2 <- tab
> tab2$FileName <- sub(".fa", "_processed.fa", tab$FileName)
> write.table(tab2, sampleFile2, sep="\t", quote=FALSE, row.names=FALSE)
> tab2
```

```
            FileName SampleName
1 mirna_1_processed.fa     miRNAs
```

```
> # remove adapters
> oldwd <- setwd(dirname(sampleFile))
> res <- preprocessReads(tab$FileName,
                         tab2$FileName,
                         Rpattern="TGGAATTCTCGGGTGCCAAGG")
> res
```

```
                    mirna_1.fa
totalSequences            1000
matchTo5pAdapter             0
matchTo3pAdapter          1000
tooShort                     0
tooManyN                     0
lowComplexity                0
totalPassed               1000
```

```
> setwd(oldwd)
```

The miRNA reads in `mirna_1.fa` are by the way synthetic sequences and do not correspond to any existing miRNAs. As you can see above from the return value of preprocessReads, all reads matched to the 3'-adapter and were therefore truncated, reducing their length to roughly the expected 22nt:

```
> # get read lengthes
> library(Biostrings)
> oldwd <- setwd(dirname(sampleFile))
> lens <- fasta.info(tab$FileName, nrec=1e5)
> lens2 <- fasta.info(tab2$FileName, nrec=1e5)
> setwd(oldwd)
> # plot length distribution
> lensTab <- rbind(raw=tabulate(lens,50),
                   processed=tabulate(lens2,50))
> colnames(lensTab) <- 1:50
> barplot(lensTab/rowSums(lensTab)*100,
        xlab="Read length (nt)", ylab="Percent of reads")
> legend(x="topleft", bty="n", fill=gray.colors(2), legend=rownames(lensTab))
```
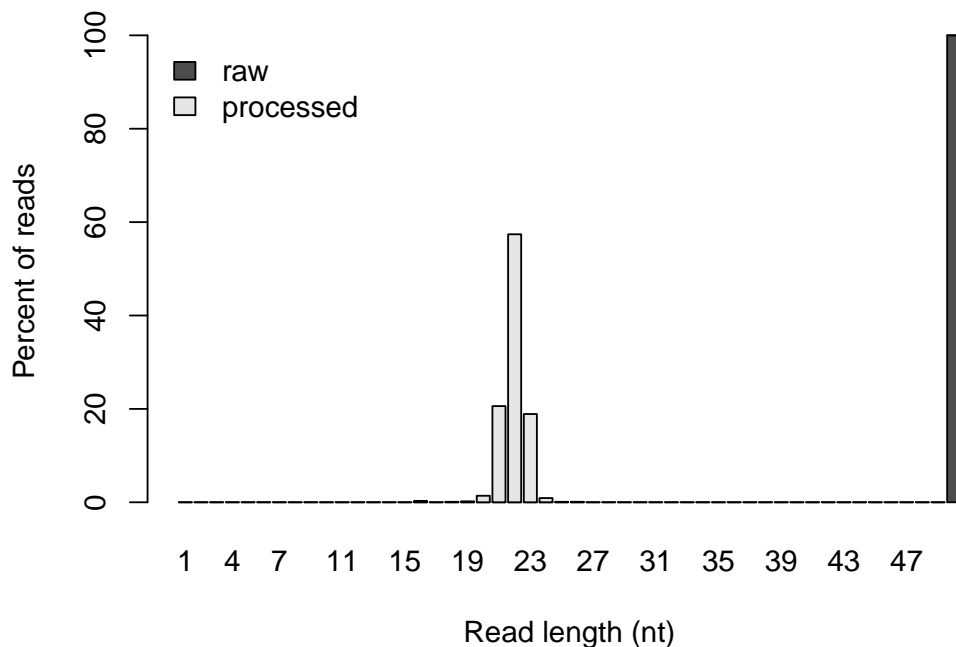
6.3.2 Alignment of small RNA (miRNA) reads

Next, we create alignments using qAlign. In contrast to the general RNA-seq workflow (section 6.2), alignment time can be reduced by using the default unspliced alignment (splicedAlignment=FALSE). Importantly, we need to set maxHits=50 or similar to also align reads that perfectly match the genome multiple times. This is required because of the miRNAs that are encoded by multiple genes. Reads from such miRNAs would not be aligned and thus their expression would be underestimated if using the default maxHits=1. An example of such a multiply-encoded miRNA is mmu-miR-669a-5p, which has twelve exact copies in the mm10 genome assembly according to mirBase19.

```
> proj3 <- qAlign(sampleFile2, genomeFile, maxHits=50)

nodeNames
zin2
   1

> alignmentStats(proj3)

               seqlength mapped unmapped
miRNAs:genome      95000   1000        0
```

A more detailed picture of the experiments' quality can be obtained using qQCReport(proj3, "qcreport.pdf") or similar (see also section 7.4 on page 43).

6.3.3 Quantification of small RNA (miRNA) reads

As with other experiment types, miRNAs are quantified using qCount. For this purpose, we first construct a query GRanges object with the genomic locations of mature miRNAs. The locations can be obtained from the *mirbase.db* package, or directly from the species-specific gff files provided by the mirBase database (e.g. `ftp://mirbase.org/pub/mirbase/19/genomes/mmu.gff3`). For the purpose of this example, the *QuasR* package provides a small gff file (″mirbaseXX_qsr.gff3″) that is formatted as the ones available from mirBase. The gff file contains both the locations of pre-miRNAs (hairpin precursors), as well as mature miRNAs. The two can be discriminated by their ″type″:

```
> mirs <- import("extdata/mirbaseXX_qsr.gff3", asRangedData=FALSE)
> names(mirs) <- mirs$Name
> preMirs <- mirs[ mirs$type=="miRNA_primary_transcript" ]
> matureMirs <- mirs[ mirs$type=="miRNA" ]
```

Please note that the name attribute of the GRanges object must be set appropriately, so that qCount can identify a single mature miRNA sequence that is encoded by multiple loci (see below) by their identical names. In this example, there are no multiply-encoded mature miRNAs, but in a real sample, you can detect them for example with `table(names(mirs))`.

The `preMirs` and `matureMirs` could now be used as query in qCount. In practise however, miRNA seem to not always be processed with high accuracy. Many miRNA reads that start one or two bases earlier or later can be observed in real data, and also their length may vary for a few bases. This is the case for the synthetic miRNAs used in this example, whose lengthes and start positions have been sampled from a read data set:

```
> library(Rsamtools)
> alns <- scanBam(alignments(proj3)$genome$FileName,
                  param=ScanBamParam(what=scanBamWhat(), which=preMirs[1]))[[1]]
> alnsIR <- IRanges(start=alns$pos - start(preMirs), width=alns$qwidth)
> mp <- barplot(as.vector(coverage(alnsIR)), names.arg=1:max(max(alnsIR)),
                xlab="Relative position in pre-miRNA",
                ylab="Alignment coverage")
> rect(xleft=mp[start(matureMirs)-start(preMirs)+1,1], ybottom=-par(cxy)[2],
       xright=mp[end(matureMirs)-start(preMirs)+1,1], ytop=0,
       col="#CCAA0088", border=NA, xpd=NA)
```
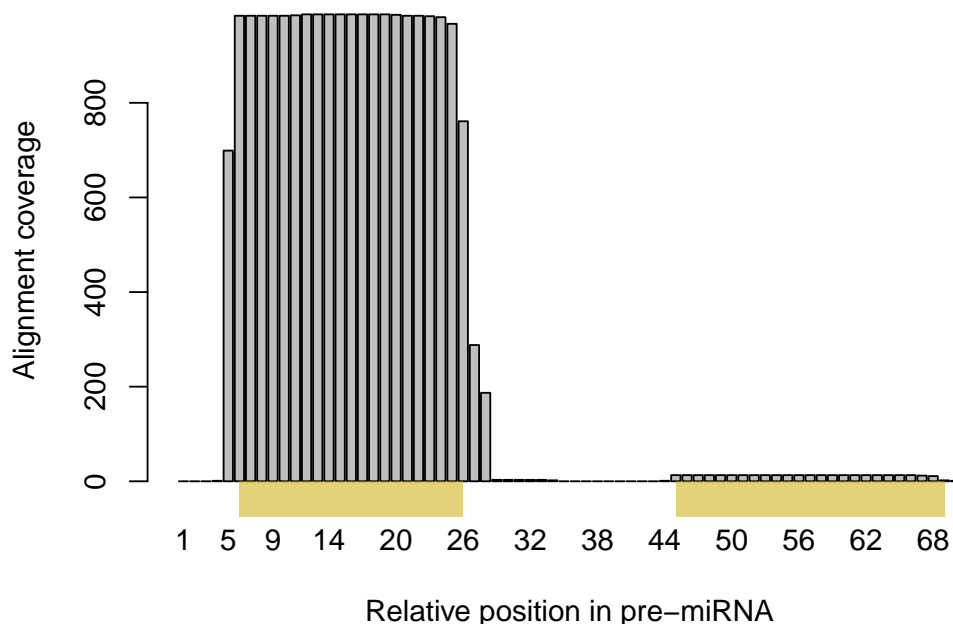
By default, qCount will count alignments that have their 5'-end within the query region (see `selectReadPosition` argument). The 5'-end correspond to the lower (left) coordinate for alignments on the plus strand, and to the higher (right) coordinate for alignments on the minus strand. In order not to miss miRNAs that have a couple of extra or missing bases, we therefore construct a query window around the 5'-end of each mature miRNA, by adding three bases up- and downstream:

```
> matureMirsExtended <- matureMirs
> start(matureMirsExtended) <- ifelse(as.logical(strand(matureMirs)=="+"),
                                       start(matureMirs) - 3,
                                       end(matureMirs) - 3)
> end(matureMirsExtended)   <- ifelse(as.logical(strand(matureMirs)=="+"),
                                       start(matureMirs) + 3,
                                       end(matureMirs) + 3)
```

The resulting extended query is then used to quantify mature miRNAs. Multyply-encoded miRNAs will be represented by multiple ranges in matureMirs and matureMirsExtended, which have identical names. qCount will automatically sum all alignments from any of those regions and return a single number per sample and unique miRNA name.

```
> # quantify mature miRNAs
> cnt <- qCount(proj3, matureMirsExtended, orientation="same")
> cnt
                 width miRNAs
qsr-miR-9876-5p      7     13
```

```
qsr-miR-9876-3p       7     984

> # quantify pre-miRNAs
> cnt <- qCount(proj3, preMirs, orientation="same")
> cnt

            width miRNAs
qsr-mir-9876     75    1000
```

6.4   Bis-seq: Measuring DNA methylation

Sequencing of bisulfite-converted genomic DNA allows detection of methylated cytosines, which in mammalian genomes typically occur in the context of CpG dinucleotides. The treatment of DNA with bisulfite induces deamination of non-methylated cytosines, converting them to uracils. Sequencing and aligning of such bisulfite-converted DNA results in C-to-T mismatches. Both alignment of converted reads, as well as the interpretation of the alignments for calculation of methylation levels require specific approaches and are supported in *QuasR* by qAlign (bisulfite argument, section 7.2) and qMeth (section 7.9), respectively.

We start the analysis by copying the example data files into the current working directly, into a subfolder called "extdata". Then, bisulfite-specific alignment is selected in qAlign by setting bisulfite to "dir" for a directional experiment, or to "undir" for an undirectional Bis-seq experiment:

```
> file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)

[1] TRUE

> sampleFile <- "extdata/samples_bis_single.txt"
> genomeFile <- "extdata/hg19sub.fa"
> proj4 <- qAlign(sampleFile, genomeFile, bisulfite="dir")

nodeNames
zin2
   1

> proj4

Project: qProject
 Options    : maxHits          : 1
             paired           : no
             splicedAlignment: FALSE
             bisulfite        : dir
             snpFile          : none
 Aligner    : Rbowtie v1.4.1 (parameters: -k 2 --best --strata -v 2)
 Genome     : /tmp/Rtmp6EN4qA/Rbuild71d34031696c/QuasR/vigne.../hg19sub.fa (file)

 Reads      : 1 file, 1 sample (fasta format):
   1. bis_1_1.fa.bz2  Sample1
```

```
 Genome alignments: directory: same as reads
   1. bis_1_1_76fb508a2a42.bam


 Aux. alignments: none
```

The resulting alignments are not different from those of non-Bis-seq experiments, apart from the fact that they may contain many C-to-T (or A-to-G) mismatches that are not counted as mismatches when aligning the reads. The number of alignments in specific genomic regions could be quantified using qCount as with ChIP-seq or RNA-seq experiments. For quantification of methylation the qMeth function is used:

```
> meth <- qMeth(proj4, mode="CpGcomb", collapseBySample=TRUE)
> meth

GRanges with 3110 ranges and 2 metadata columns:
          seqnames           ranges strand | Sample1_T Sample1_M
             <Rle>        <IRanges>  <Rle>  | <integer> <integer>
     [1]      chr1        [19,  20]      * |         1         1
     [2]      chr1        [21,  22]      * |         1         1
     [3]      chr1        [54,  55]      * |         3         1
     [4]      chr1        [57,  58]      * |         3         0
     [5]      chr1        [80,  81]      * |         6         5
     ...       ...              ...    ... ...       ...       ...
  [3106]      chr3 [44957, 44958]      * |         8         7
  [3107]      chr3 [44977, 44978]      * |         5         3
  [3108]      chr3 [44981, 44982]      * |         4         3
  [3109]      chr3 [44989, 44990]      * |         1         1
  [3110]      chr3 [44993, 44994]      * |         1         1
  ---
  seqlengths:
    chr1  chr2  chr3
   40000 10000 45000
```

By default, qMeth quantifies methylation for all cytosines in CpG contexts, combining the data from plus and minus strands (mode="CpGcomb"). The results are returned as a GRanges object with coordinates of each CpG, and two metadata columns for each input sequence file in the qProject object. These two columns contain the total number of aligned reads that overlap a given CpG (C-to-C matches or C-to-T mismatches, suffix _T in the column name), and the number of read alignments that had a C-to-C match at that position (methylated events, suffix _M).

Independent of the number of alignments, the returned object will list all CpGs in the target genome including the ones that have zero coverage, unless you set keepZero=FALSE:

```
> chrs <- readDNAStringSet(genomeFile)
> sum(vcountPattern("CG",chrs))

[1] 3110
```

```
> length(qMeth(proj4))
```

```
[1] 3110
```

```
> length(qMeth(proj4, keepZero=FALSE))
```

```
[1] 2929
```

The fraction methylation can easily be obtained as the ratio between _M and _T columns:

```
> percMeth <- mcols(meth)[,2] *100 /mcols(meth)[,1]
> summary(percMeth)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NAs
    0.0    75.0    90.9    75.4   100.0   100.0    181
```
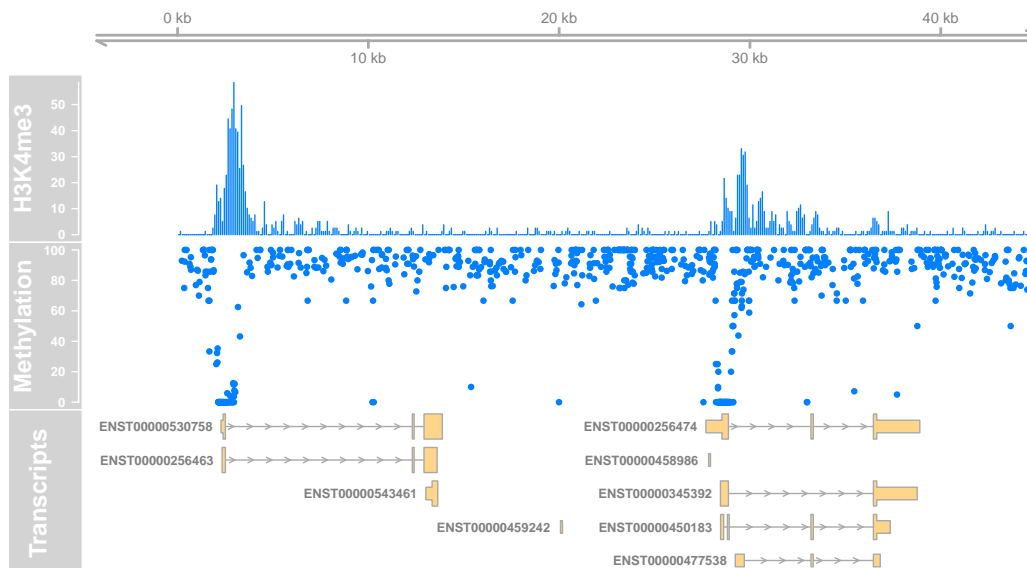
```
> axisTrack <- GenomeAxisTrack()
> dTrack1 <- DataTrack(range=gr1, name="H3K4me3", type="h")
> dTrack2 <- DataTrack(range=meth, data=percMeth,
                     name="Methylation", type="p")
> txTrack <- GeneRegionTrack(txdb, name="Transcripts", showId=TRUE)
> plotTracks(list(axisTrack, dTrack1, dTrack2, txTrack),
           chromosome="chr3", extend.left=1000)
```


If qMeth is called without a query argument, it will by default return methylation states for each C or CpG in the genome. Using a query argument it is possible to restrict the analysis to specific genomic regions, and if using in addition collapseByQueryRegion=TRUE, the single base methylation states will further be combined for all C's that are contained in the same query region:

```
> qMeth(proj4, query=GRanges("chr1",IRanges(start=31633,width=2)),
       collapseBySample=TRUE)
```

```
GRanges with 1 range and 2 metadata columns:
```

```
          seqnames            ranges strand | Sample1_T Sample1_M
             <Rle>         <IRanges>  <Rle> | <integer> <integer>
  CpGcomb      chr1 [31633, 31634]       * |        10         2
  ---
  seqlengths:
    chr1  chr2  chr3
   40000 10000 45000
```

```
> qMeth(proj4, query=promRegSel, collapseByQueryRegion=TRUE,
       collapseBySample=TRUE)
```

```
GRanges with 12 ranges and 2 metadata columns:
        seqnames            ranges strand   | Sample1_T Sample1_M
           <Rle>         <IRanges>  <Rle>   | <numeric> <numeric>
   [1]      chr1 [31629, 33128]        +    |       426        74
   [2]      chr1 [ 6452,  7951]        -    |       388       244
   [3]      chr1 [14013, 15512]        -    |       627       560
   [4]      chr1 [31882, 33381]        -    |       522       232
   [5]      chr2 [ 1795,  3294]        +    |       997       539
   ...       ...              ...    ... ...        ...       ...
   [8]      chr3 [ 1276,  2775]        +    |       715       253
   [9]      chr3 [19069, 20568]        +    |       253       204
  [10]      chr3 [26692, 28191]        +    |       934       818
  [11]      chr3 [26834, 28333]        +    |       934       777
  [12]      chr3 [13102, 14601]        -    |       307       287
  ---
  seqlengths:
    chr1  chr2  chr3
   40000 10000 45000
```

Finally, qMeth allows the retrieval of methylation states for individual molecules (per alignment). This is done by using a query containing a single genomic region (typically small, such as a PCR amplicon) and setting reportLevel="alignment". In that case, the return value of qMeth will be a list (over samples) of lists (with four elements giving the identities of alignment, C nucleotide, strand and the methylation state). See the documentation of qMeth for more details.

6.5 Allele-specific analysis

All experiment types supported by *QuasR* (ChIP-seq, RNA-seq and Bis-seq; only alignments to the genome, but not to auxiliaries) can also be analyzed in an allele-specific manner. For this, a file containing genomic location and the two alleles of known sequence polymorphisms has to be provided to the snpFile argument of qAlign. The file is in tab-delimited text format without a header and contains four columns with chromosome name, position, reference allele and alternative allele.

Below is an example of a SNP file, also available from system.file(package="QuasR", "extdata", "hg19sub_snp.txt"):

```
chr1   8596   G   A
chr1   18443  G   A
chr1   18981  C   T
chr1   19341  G   A
...
```

For a given locus, either reference or alternative allele may but does not have to be identical to the sequence of the reference genome (genomeFile in this case). To avoid an alignment bias, all reads are aligned separately to each of the two new genomes, which *QuasR* generates by *injecting* the SNPs listed in snpFile into the reference genome. Finally, the two alignment files are combined, only retaining the best alignment for each read. While this procedure takes more than twice as long as aligning against a single genome, it has the advantage to correctly align reads even in regions of high SNP density and has been shown to produce more accurate results.

While combining alignments, each read is classified into one of three groups (stored in the BAM files under the XV tag):

- **R**: the read aligned better to the **reference** genome
- **U**: the read aligned equally well to both genomes (**unknown** origin)
- **A**: the read aligned better to the **alternative** genome

Using these alignment classifications, the qCount and qMeth functions will produce three counts instead of a single count; one for each class. The column names will be suffixed by _R, _U and _A.

The examples below use data provided with the *QuasR* package, which is first copied to the current working directory, into a subfolder called "extdata":

```
> file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)
```

```
[1] TRUE
```

The example below aligns the same reads that were also used in the ChIP-seq workflow (section 6.1), but this time using a snpFile:

```
> sampleFile <- "extdata/samples_chip_single.txt"
> genomeFile <- "extdata/hg19sub.fa"
> snpFile <- "extdata/hg19sub_snp.txt"
> proj1SNP <- qAlign(sampleFile, genome=genomeFile, snpFile=snpFile)
```

```
nodeNames
zin2
   1
```

```
> proj1SNP
```

```
Project: qProject
 Options   : maxHits         : 1
             paired          : no
             splicedAlignment: FALSE
             bisulfite       : no
```

```
            snpFile          : /tmp/Rtmp6EN4qA/Rbuild71d3403.../hg19sub_snp.txt
 Aligner    : Rbowtie v1.4.1 (parameters: -k 2 --best --strata -v 2)
 Genome     : /tmp/Rtmp6EN4qA/Rbuild71d34031696c/QuasR/vigne.../hg19sub.fa (file)

 Reads      : 2 files, 2 samples (fastq format):
   1. chip_1_1.fq.bz2  Sample1 (phred33)
   2. chip_2_1.fq.bz2  Sample2 (phred33)

 Genome alignments: directory: same as reads
   1. chip_1_1_76fb307c55fa.bam
   2. chip_2_1_76fb6bfd700f.bam

 Aux. alignments: none
```

Instead of one count per promoter region and sample, qCount now returns three (promRegSel was generated in the ChIP-seq example workflow on page):

```
> head(qCount(proj1, promRegSel))

                width Sample1 Sample2
ENSG00000176022  1500     157     701
ENSG00000186891  1500      22       5
ENSG00000186827  1500      10       3
ENSG00000078808  1500      73     558
ENSG00000171863  1500      94     339
ENSG00000252531  1500      59       9
```

```
> head(qCount(proj1SNP, promRegSel))

                width Sample1_R Sample1_U Sample1_A Sample2_R Sample2_U
ENSG00000176022  1500         0       133         0         0       559
ENSG00000186891  1500         4        16         0         0         5
ENSG00000186827  1500         2         8         0         0         2
ENSG00000078808  1500         0        59         0         0       432
ENSG00000171863  1500         4        78         0         8       263
ENSG00000252531  1500         3        50         2         0         6
                Sample2_A
ENSG00000176022         0
ENSG00000186891         0
ENSG00000186827         0
ENSG00000078808         0
ENSG00000171863         0
ENSG00000252531         0
```

The example below illustrates use of a snpFile for Bis-seq experiments. Similarly as for qCount, the count types are labeled by R, U and A. These labels are added to the total and methylated column suffixes _T and _M, resulting in a total of six instead of two counts per feature and sample:

```
> sampleFile <- "extdata/samples_bis_single.txt"
> genomeFile <- "extdata/hg19sub.fa"
> proj4SNP <- qAlign(sampleFile, genomeFile,
                     snpFile=snpFile, bisulfite="dir")

nodeNames
zin2
   1

> head(qMeth(proj4SNP, mode="CpGcomb", collapseBySample=TRUE))

GRanges with 6 ranges and 6 metadata columns:
      seqnames        ranges strand | Sample1_TR Sample1_MR Sample1_TU Sample1_MU
         <Rle>     <IRanges>  <Rle> |  <integer>  <integer>  <integer>  <integer>
  [1]     chr1 [ 19,  20]       * |          0          0          1          1
  [2]     chr1 [ 21,  22]       * |          0          0          1          1
  [3]     chr1 [ 54,  55]       * |          0          0          3          1
  [4]     chr1 [ 57,  58]       * |          0          0          3          0
  [5]     chr1 [ 80,  81]       * |          0          0          6          5
  [6]     chr1 [103, 104]       * |          0          0          6          5
      Sample1_TA Sample1_MA
       <integer>  <integer>
  [1]           0          0
  [2]           0          0
  [3]           0          0
  [4]           0          0
  [5]           0          0
  [6]           0          0
  ---
  seqlengths:
    chr1  chr2  chr3
   40000 10000 45000
```

7   Description of Individual QuasR Functions

Please refer to the *QuasR* reference manual or the function documentation (e.g. using ?qAlign) for a complete description of *QuasR* functions. The descriptions provided below are meant to give and overview over all functions and summarize the purpose of each one.

7.1   preprocessReads

The preprocessReads function can be used to prepare the input sequences before alignment to the reference genome, for example to filter out low quality reads unlikely to produce informative alignments. When working with paired-end experiments, the paired reads are expected to be contained in identical order in two separate files. For this reason, both reads of a pair are filtered out if any of the two reads fulfills the filtering criteria. The following types of filtering tasks can be performed (in the order as listed):

1. **Truncate reads**: remove nucleotides from the start and/or end of each read.
2. **Trim adapters**: remove nucleotides at the beginning and/or end of each read that match to a defined (adapter) sequence. The adapter trimming is done by calling trimLRPatterns from the *Biostrings* package [13].
3. **Filter out low quality reads**: Filter out reads that fulfill any of the filtering criteria (contain more than nBases N bases, are shorter than minLength or have a dinucleotide complexity of less than complexity-times the average complexity of the human genome sequence).

The dinucleotide complexity is calculated in bits as Shannon entropy using the following formula $-\sum_i f_i \cdot \log_2 f_i$, where f_i is the frequency of dinucleotide i $(i = 1, 2, ..., 16)$.

7.2   qAlign

qAlign is the function that generates alignment files in BAM format, for all input sequence files listed in sampleFile (see section 5.1), against the reference genome (genome argument), and for reads that to not match to the reference genome, against one or several auxiliary target sequences (auxiliaryFile, see section 5.2).

The reference genome can be provided either as a FASTA sequence file or as a BSgenome package name (see section 5.3). If a BSgenome package is not found in the installed packages but available from Bioconductor, it will be automatically downloaded.

The alignment program is set by aligner, and parameters by maxHits, paired, splicedAlignment and alignmentParameter. Currently, aligner can only be set to "Rbowtie", which is a wrapper for *bowtie* [4] and *SpliceMap* [12]. *SpliceMap* will be used if splicedAlignment=TRUE. The alignment strategy is affected by the parameters snpFile (alignments to variant genomes containing sequence polymorphisms) and bisulfite (alignment of bisulfite-converted reads). Finally, clObj can be used to enable parallelized alignment, sorting and conversion to BAM format.

For each input sequence file listed in `sampleFile`, one BAM file with alignments to the reference genome will be generated, and an additional one for each auxiliary sequence file listed in `auxiliaryFile`. By default, these BAM files are stored at the same location as the sequence files, unless a different location is specified under `alignmentsDir`. If compatible alignment files are found at this location, they will not be regenerated, which allows re-use of the same sequencing samples in multiple analysis projects by listing them in several project-specific `sampleFiles`.

The alignment process produces temporary files, such as decompressed input sequence files or raw alignment files before conversion to BAM format, which can be several times the size of the input sequence files. These temporary files are stored in the directory specified by `cacheDir`, which defaults to the *R* process temporary directory returned by `tempdir()`. The location of `tempdir()` can be set using environment variables (see `?tempdir`).

`qAlign` returns a `qProject` object that contains all file names and paths, as well as all alignment parameters necessary for further analysis (see section 7.3 for methods to access the information contained in a `qProject` object).

7.3   qProject **class**

The `qProject` objects are returned by `qAlign` and contain all information about a sequencing experiment needed for further analysis. It is the main argument passed to the functions that start with a *q* letter, such as `qCount`, `qQCReport` and `qExportWig`. Some information inside of a `qProject` object can be accessed by specific methods (in the examples below, `x` is a `qProject` object):

- `length(x)` gets the number of input files.
- `genome(x)` gets the reference genome as a `character(1)`. The type of genome is stored as an attribute in `attr(genome(x),"genomeFormat")`: `"BSgenome"` indicates that `genome(x)` refers to the name of a *BSgenome* package, `"file"` indicates that it contains the path and file name of a genome in FASTA format.
- `auxiliaries(x)` gets a `data.frame` with auxiliary target sequences. The `data.frame` has one row per auxiliary target file, and two columns "FileName" and "AuxName".
- `alignments(x)` gets a list with two elements "genome" and "aux". "genome" contains a `data.frame` with `length(x)` rows and two columns "FileName" (containing the path to bam files with genomic alignments) and "SampleName". "aux" contains a `data.frame` with one row per auxiliary target file (with auxiliary names as row names), and `length(x)` columns (one per input sequence file).
- `x[i]` returns a `qProject` object instance with `i` input files, where `i` can be an NA-free logical, numeric, or character vector.

7.4   qQCReport

The `qQCReport` function samples a random subset of sequences and alignments from each sample or input file and generates a series of diagnostic plots for estimating data quality. The plots below show the

currently available plots as produced by the ChIP-seq example in section 6.1 (except for the fragment size distributions which are based on an unspliced alignment of paired-end RNA seq reads):

- **Quality score boxplot** shows the distribution of base quality values as a box plot for each position in the input sequence. The background color (green, orange or red) indicates ranges of high, intermediate and low qualities. The plot is available for fastq only (BAM files may contain base quality information, which is however not used here because reads contained in the BAM file, e.g. aligned reads, may not be a representative sub-sample of all sequenced reads).
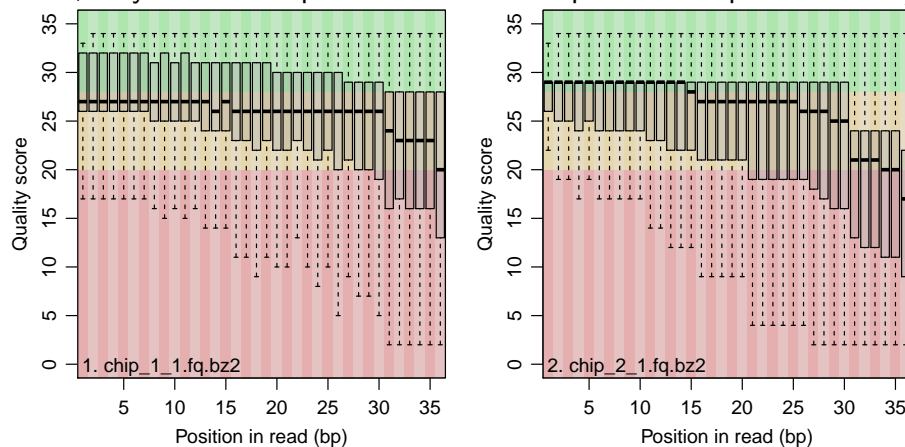
- **Nucleotide frequency** plot shows the frequency of A, C, G, T and N bases by position in the read. The plot is always available.
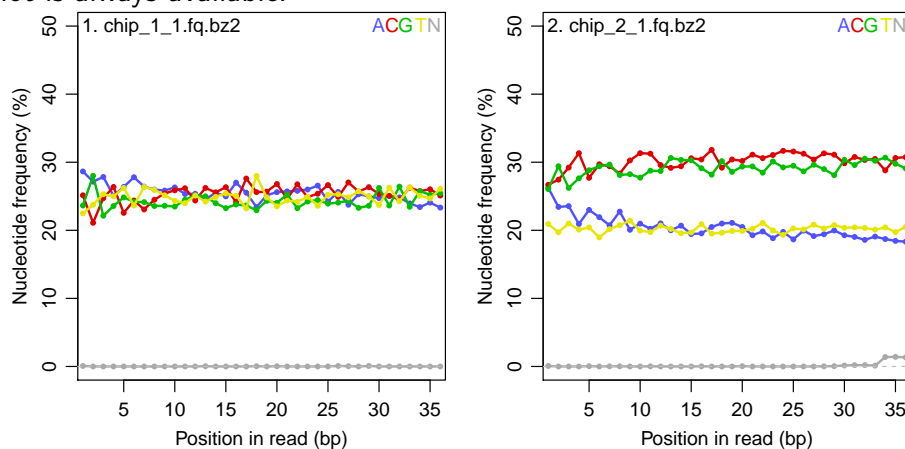

- **Duplication level** plot shows for each sample the fraction of reads observed at different duplication levels (e.g. once, two-times, three-times, etc.). In addition, the most frequent sequences are listed. The plot is available for fasta or fastq files, but not for bam files, again because contained reads may not be representative for the experiment.
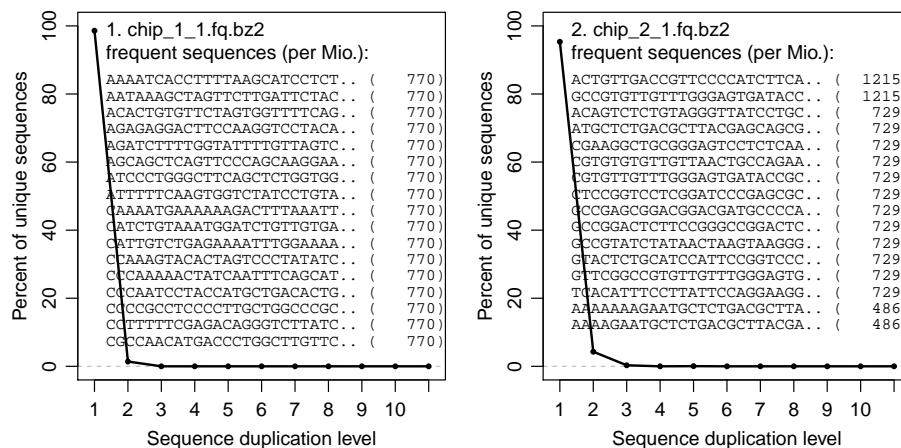
- **Mapping statistics** shows fractions of reads that were (un)mappable to the reference genome. This plot is available for bam input, i.e. if input is a vector of BAM files, or a qProject with alignment files as returned by qAlign.
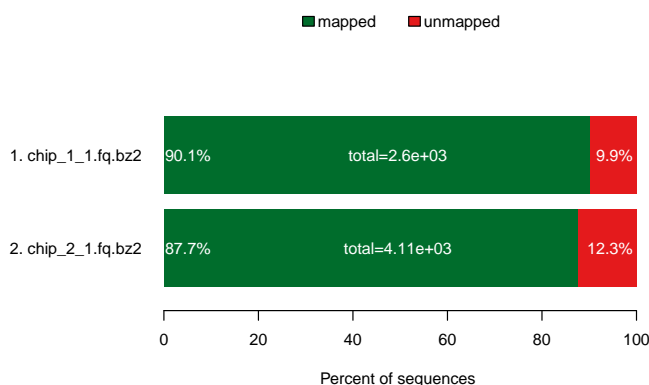

- **Library complexity** shows fractions of unique read(-pair) alignment positions. Please note that this measure is not independent from the total number of reads in a library, and is best compared between libraries of similar sizes. This plot is available for bam input.
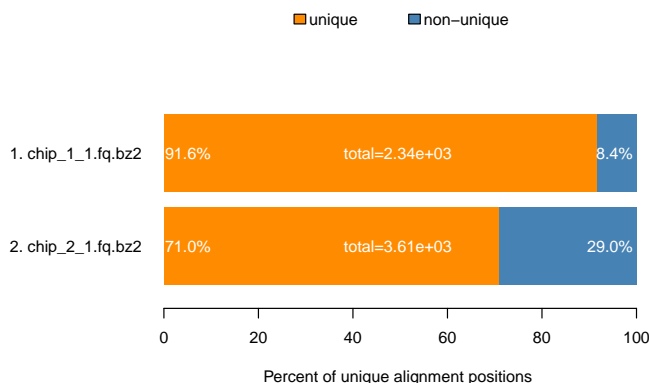

- **Mismatch frequency** shows the frequency and position (relative to the read sequence) of mismatches in the alignments against the reference genome. The plot is available for bam input.
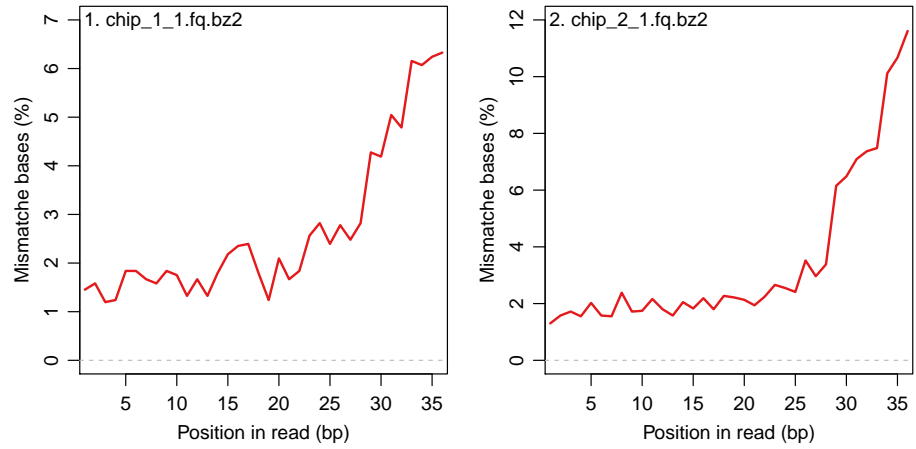
- **Mismatch types** shows the frequency of read bases that caused mismatches in the alignments to the reference genome, separately for each genome base. This plot is available for bam input.
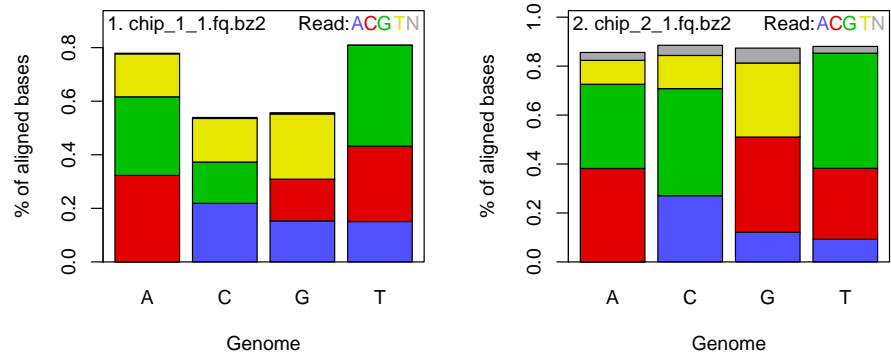
- **Fragment size** shows the distribution of fragment sizes inferred from aligned read pairs. This plot is available for paired-end bam input.
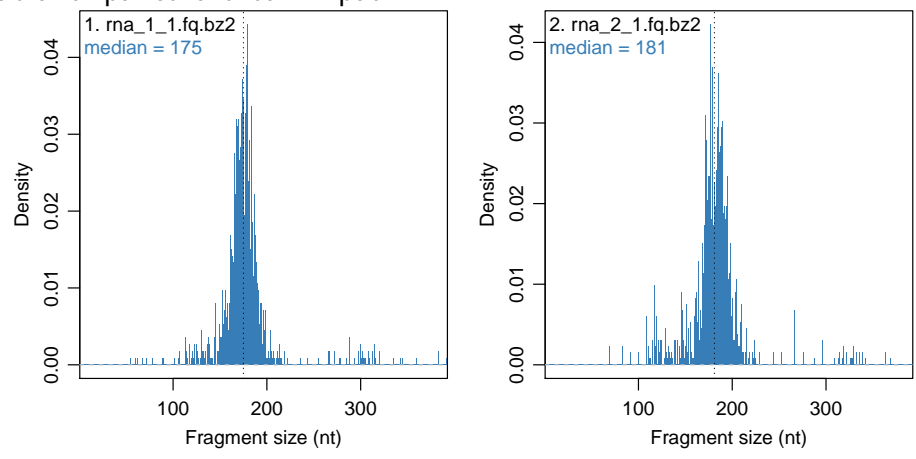
7.5 `alignmentStats`

alignmentStats is comparable to the "idxstats" function from Samtools; it returns the size of the target sequence, as well as the number of mapped and unmapped reads that are contained in an indexed BAM file. The function works for arguments of type qProject, as well as on a string with one or several BAM file names. There is however a small difference in the two that is illustrated in the following example, which uses the qProject object from the ChIP-seq workflow created on page 25:

```
> # using bam files
> alignmentStats(alignments(proj1)$genome$FileName)

                            seqlength mapped unmapped
chip_1_1_76fb7d404822.bam       95000   2339      258
chip_2_1_76fb48423632.bam       95000   3609      505

> alignmentStats(unlist(alignments(proj1)$aux))

                            seqlength mapped unmapped
chip_1_1_76fb723f15b9.bam        5386    251        0
chip_2_1_76fb1cd44b73.bam        5386    493        0

> # using a qProject object
> alignmentStats(proj1)

                  seqlength mapped unmapped
Sample1:genome        95000   2339      258
Sample2:genome        95000   3609      505
Sample1:phiX174        5386    251        7
Sample2:phiX174        5386    493       12
```

If calling `alignmentStats` on the bam files directly as in the first two expressions of the above example, the returned numbers correspond exactly to what you would obtain by the "idxstats" function from Samtools, only that the latter would report them separately for each target sequence, while `alignmentStats` sums them for each BAM file. These numbers correctly state that there are zero unmapped reads in the auxiliary BAM files. However, if calling `alignmentStats` on a qProject object, it will report 7 and 12 unmapped reads in the auxiliary BAM files. This is because `alignmentStats` is aware that unmapped reads are removed from auxiliary BAM files by *QuasR*, but can be calculated from the total number of reads to be aligned to the auxiliary target, which equals the number of unmapped reads in the corresponding genomic BAM file.

7.6 qExportWig

qExportWig creates fixed-step wig files (see http://genome.ucsc.edu/goldenPath/help/wiggle. html for format definition) from the genomic alignments contained in a qProject object. The combine argument controls if several input files are combined into a single multi-track wig file, or if they are exported as individual wig files. Alignments of single read experiments can be shifted towards there 3'-end using `shift`; paired-end alignments are automatically shifted by half the insert size. The resolution

of the created wig file is defines by the `binsize` argument, and if `scaling=TRUE`, multiple alignment files in the qProject object are scaled by their total number of reads.

7.7 qCount

qCount is the workhorse for counting alignments that overlap query regions. Usage and details on parameters can be obtained from the qCount function documentation. Two aspects that are of special importance are also discussed here:

7.7.1 Determination of overlap

How an alignment overlap with a query region is defined can be controlled by the following arguments of qCount:

- `selectReadPosition` specifies the read base that serves as a reference for overlaps with query regions. The alignment position of that base, eventually after shifting (see below), needs to be contained in the query region for an overlap. `selectReadPosition` can be set to "start" (the default) or "end", which refer to the biological start (5'-end) and end (3'-end) of the read. For example, the "start" of a read aligned to the plus strand is the leftmost base in the alignment (the one with the lowest coordinate), and the "end" of a read aligned to the minus strand is also its leftmost base in the alignment.
- `shift` allows shifting of alignments towards their 3'-end prior to overlap determination and counting. This can be helpful to increase resolution of ChIP-seq experiments by moving alignments by half the immuno-precipitated fragment size towards the middle of fragments. `shift` can either contain "integer" values that specify the shift size, or for paired-end experiments, it can be set to the keyword "halfInsert", which will estimate the true fragment size from the distance between aligned read pairs and shift the alignments accordingly.
- `orientation` controls the interpretation of alignment strand relative to the strand of the query region. The default value "any" will count all overlapping alignments, irrespective of the strand. This setting is for example used in an unstranded RNA-seq experiment where both sense and antisense reads are generated from an mRNA. A value of "same" will only count the alignments on the same strand as the query region (e.g. in a stranded RNA-seq experiment), and "opposite" will only count the alignments on the opposite strand from the query region (e.g. to quantify anti-sense transcription in a stranded RNA-seq experiment).
- `useRead` only applies to paired-end experiments and allows to quantify either all alignments (useRead="any"), or only the first (useRead="first") or last (useRead="last") read from each read pair or read group. Note that for useRead="any" (the default), an alignment pair that is fully contained within a query region will contribute two counts to the value of that region.
- `includeSpliced`: When set to `FALSE`, spliced alignments will be excluded from the quantification. This could be useful for example to avoid redundant counting of reads when the spliced alignments are quantified separately using `reportLevel="junction"`.

7.7.2   Running modes of qCount

The features to be quantified are specified by the query argument. At the same time, the type of query selects the mode of quantification. qCount supports three different types of query arguments and implements three corresponding quantification types, which primarily differ in the way they deal with redundancy, such as query bases that are contained in more than one query region. A fourth quantification mode allows counting of alignments supporting exon-exon juctions:

- GRanges query: Overlapping alignments are counted separately for each coordinate region in the query object. If multiple regions have identical names, their counts will be summed, counting each alignment only once even if it overlaps more than one of these regions. Alignments may however be counted more than once if they overlap multiple regions with different names. This mode is for example used to quantify ChIP-seq alignments in promoter regions (see section 6.1 on page 21), or gene expression levels in an RNA-seq experiment (using a 'query' with exon regions named by gene).
- GRangesList query: Alignments are counted and summed for each list element in the query object if they overlap with any of the regions contained in the list element. The order of the list elements defines a hierarchy for quantification: Alignment will only be counted for the first element (the one with the lowest index in the query) that they overlap, but not for any potential further list elements containing overlapping regions. This mode can be used to hierarchically and uniquely count (assign) each alignment to a one of several groups of regions (the elements in the query), for example to estimate the fractions of different classes of RNA in an RNA-seq experiment (rRNA, tRNA, snRNA, snoRNA, mRNA, etc.)
- TranscriptDb query: Used to extract regions from annotation and report alignment counts depending on the value of the reportLevel argument. If reportLevel="exon", alignments overlapping each exon in the query are counted. If reportLevel="gene", alignment counts for all exons of a gene will be summed, counting each alignment only once even if it overlaps multiple annotated exons of a gene. These are useful to calculate exon or gene expression levels in RNA-seq experiments based on the annotation in a TranscriptDB object. If reportLevel="promoter", the promoters function from package *GenomicFeatures* is used with default arguments to extract promoter regions around transcript start sites, e.g. to quantify alignments inf a ChIP-seq experiment.
- any of the above or NULL for reportLevel="junction": The query argument is ignored if reportLevel is set to "junction", and qCount will count the number of alignments supporting each exon-exon junction detected in any of the samples in proj. The arguments selectReadPosition, shift, orientation, useRead and mask will have no effect in this quantification mode.

7.8   qProfile

The qProfile function differs from qCount in that it returns alignments counts relative to their position in the query region. Except for upstream and downstream, the arguments of qProfile and qCount are the same. This section will describe these two additional arguments; more details on the other arguments are available in section 7.7 and from the qProfile function documentation.

The regions to be profiled are anchored by the biological start position, which are aligned at position

zero in the return value. The biological start position is defined as `start(query)` for regions on the plus strand and `end(query)` for regions on the minus strand. The anchor positions are extended to the left and right sides by the number of bases indicated in the `upstream` and `downstream` arguments.

- `upstream` indicates the number of bases upstream of the anchor position, which is on the left side of the anchor point for regions on the plus strand and on the right side for regions on the minus strand.
- `downstream` indicates the number of bases downstream of the anchor position, which is on the left side of the anchor point for regions on the plus strand and on the left side for regions on the minus strand.

Be aware that query regions with a "*" strand are handled the same way as regions on the plus strand.

7.9 qMeth

qMeth is used exclusively for Bis-seq experiments. In contrast to qCount, which counts the number of read alignments per query region, qMeth quantifies the number of C and T bases per cytosine in query regions, in order to determine methylation status.

qMeth can be run in one of four modes, controlled by the mode argument:

- `CpGcomb`: Only C's in CpG context are considered. It is assumed that methylation status of the CpG base-pair on both strands is identical. Therefore, the total and methylated counts obtained for the C at position i and the C on the opposite strand at position $i + 1$ are summed.
- `CpG`: As with `CpGcomb`, only C's in CpG context are quantified. However, counts from opposite strand are not summed, resulting in separate output values for C's on both strands.
- `allC`: All C's contained in query regions are quantified, keeping C's from either strand separate. While this mode allows quantification of non-CpG methylation, it should be used with care, as the large result could use up available memory. In that case, a possible work-around is to divide the region of interest (e.g. the genome) into several regions (e.g. chromosomes) and call qMeth separately for each region.
- `var`: In this mode, only alignments on the opposite strand from C's are analysed in order to collect evidence for sequence polymorphisms. Methylated C's are hot-spots for C-to-T transitions, which in a Bis-seq experiment cannot be discriminated from completely unmethylated C's. The information is however contained in alignments to the G from the opposite strand: Reads containing a G are consistent with a non-mutated C, and reads with an A support the presence of a sequence polymorphism. qMeth(..., mode="var") returns counts for total and matching bases for all C's on both strands. A low fraction of matching bases is an indication of a mutation and can be used as a basis to identify mutated positions in the studied genome relative to the reference genome. Such positions should not be included in the quantification of methylation.

When using qMeth in a allele-specific quantification (see also section 6.5), cytosines (or CpGs) that overlap a sequence polymorphism will not be quantified.

8 Session information

The output in this vignette was produced under:

- R version 3.1.0 (2014-04-10), x86_64-unknown-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, utils
- Other packages: AnnotationDbi 1.26.0, BSgenome 1.32.0, Biobase 2.24.0, BiocGenerics 0.10.0, BiocInstaller 1.14.2, Biostrings 2.32.0, GenomeInfoDb 1.0.2, GenomicFeatures 1.16.0, GenomicRanges 1.16.3, Gviz 1.8.0, IRanges 1.22.6, QuasR 1.4.2, Rbowtie 1.4.1, Rsamtools 1.16.0, XVector 0.4.0, rtracklayer 1.24.0
- Loaded via a namespace (and not attached): BBmisc 1.6, BatchJobs 1.2, BiocParallel 0.6.0, BiocStyle 1.2.0, DBI 0.2-7, Formula 1.1-1, GenomicAlignments 1.0.1, Hmisc 3.14-4, R.methodsS3 1.6.1, RColorBrewer 1.0-5, RCurl 1.95-4.1, RSQLite 0.11.4, Rcpp 0.11.1, ShortRead 1.22.0, VariantAnnotation 1.10.1, XML 3.98-1.1, biomaRt 2.20.0, biovizBase 1.12.1, bitops 1.0-6, brew 1.0-6, cluster 1.15.2, codetools 0.2-8, colorspace 1.2-4, dichromat 2.0-0, digest 0.6.4, fail 1.2, foreach 1.4.2, hwriter 1.3, iterators 1.0.7, lattice 0.20-29, latticeExtra 0.6-26, matrixStats 0.8.14, munsell 0.4.2, plyr 1.8.1, scales 0.2.4, sendmailR 1.1-2, splines 3.1.0, stats4 3.1.0, stringr 0.6.2, survival 2.37-7, tools 3.1.0, zlibbioc 1.10.0

References

[1] Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T. Morgan, and Vincent J. Carey. Software for computing and annotating genomic ranges. *PLoS Comput Biol*, 9:e1003118, 2013. URL: http://www.ploscompbiol.org/article/info:doi/10.1371/journal.pcbi.1003118, doi:doi:10.1371/journal.pcbi.1003118.

[2] A. Lerch, D. Gaidatzis, F. Hahne, and M.B. Stadler. Quasr: Quantify and annotate short reads in r. unpublished, 2012.

[3] P. Dalgaard. *Introductory Statistics with R*. Springer, 2002.

[4] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

[5] F. Hahne, A. Lerch, and M.B. Stadler. Rbowtie: A r wrapper for bowtie and splicemap short read aligners. unpublished, 2012.

[6] Mark D Robinson, Davis J McCarthy, and Gordon K Smyth. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26:139–140, 2010.

[7] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome Biology*, 11:R106, 2010. URL: http://genomebiology.com/2010/11/10/R106/, doi:10.1186/gb-2010-11-10-r106.

[8] Jianqiang Sun, Tomoaki Nishiyama, Kentaro Shimizu1, and Koji Kadota. Tcc: an r package for comparing tag count data with robust normalization strategies. *BMC Bioinformatics*, 14:219, 2013. URL: http://www.biomedcentral.com/1471-2105/14/219, doi:doi:10.1186/1471-2105-14-219.

[9] Simon Anders, Alejandro Reyes, and Wolfgang Huber. Detecting differential usage of exons from rna-seq data. *Genome Research*, 22:2008–2017, 2012. URL: http://genome.cshlp.org/content/early/2012/06/21/gr.133744.111.full.pdf+html, doi:10.1101/gr.133744.111.

[10] Thomas J Hardcastle and Krystyna A Kelly. bayseq: empirical bayesian methods for identifying differential expression in sequence count data. *BMC Bioinformatics*, 11:422, 2010. URL: http://www.biomedcentral.com/1471-2105/11/422, doi:10.1186/1471-2105-11-422.

[11] Martin Morgan, Simon Anders, Michael Lawrence, Patrick Aboyoun, Hervé Pagès, and Robert Gentleman. Shortread: a bioconductor package for input, quality assessment and exploration of high-throughput sequence data. *Bioinformatics*, 25:2607–2608, 2009. URL: http://dx.doi.org10.1093/bioinformatics/btp450, doi:10.1093/bioinformatics/btp450.

[12] K.F. Au, H. Jiang, L. Lin, Y. Xing, and W.H. Wong. Detection of splice junctions from paired-end rna-seq data by splicemap. *Nucleic Acids Research*, 38(14):4570–4578, 2010.

[13] H. Pages, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: String objects representing biological sequences, and matching algorithms*. R package version 2.28.0.