RnBeads – Comprehensive Analysis of DNA Methylation Data

Fabian Müller, Yassen Assenov, Pavlo Lutsik, Michael Scherer

Contact: team@rnbeads.org

Package version: 2.10.0

October 13, 2026

RnBeads is an R package for the comprehensive analysis of genome-wide DNA methylation data with single basepair resolution. Supported assays include the Infinium 450k microarray, whole genome bisulfite sequencing (WGBS), reduced representation bisulfite sequencing (RRBS), other forms of enrichment bisulfite sequencing, and any other large-scale method that can provide DNA methylation data at single basepair resolution (e.g. MeDIP-seq after suitable preprocessing). It builds upon a significant and ongoing community effort to devise effective algorithms for dealing with large-scale DNA methylation data and implements these methods in a user-friendly, high-throughput workflow, presenting the analysis results in a highly interpretable fashion.

Contents

1 Getting Started 2
- 1.1 Installation . . . 2
- 1.2 Overview of RnBeads Modules . . . 3
- 1.3 Datasets . . . 5
- 1.4 Setting Up the Analysis Environment . . . 5

2 Vanilla Analysis 5

3 Working with RnBSet Objects 7

4 Tailored Analysis: RnBeads Modules 13
- 4.1 Data Import . . . 13
- 4.2 Quality Control . . . 15
  - 4.2.1 Sex Prediction . . . 19
  - 4.2.2 CNV estimation . . . 19
- 4.3 Preprocessing . . . 22
  - 4.3.1 Filtering . . . 23
  - 4.3.2 Normalization . . . 23
- 4.4 Covariate Inference . . . 24
  - 4.4.1 Surrogate Variable Analysis . . . 25
  - 4.4.2 Inference of Cell Type Contributions . . . 25

4.4.3 Age Prediction . . . 26
4.4.4 Immune Cell Content Estimation . . . 27
4.5 Exploratory Analysis . . . 27
4.5.1 Low-dimensional Representation and Batch Effects . . . 28
4.5.2 Clustering . . . 29
4.6 Differential Methylation Analysis . . . 30
4.6.1 Differential Variability Analysis . . . 33
4.6.2 Paired Analysis . . . 35
4.6.3 Adjusting for Covariates in the Differential Analysis . . . 35
4.6.4 Enrichment Analysis of Differentially Methylated Regions . . . 36
4.7 Tracks and Tables . . . 37

5 Analyzing Bisulfite Sequencing Data **38**
5.1 Data Loading . . . 39
5.2 Quality Control . . . 41
5.3 Filtering . . . 41

6 Advanced Usage **43**
6.1 Analysis Parameter Overview . . . 43
6.2 Annotation Data . . . 43
6.2.1 Custom Annotation . . . 44
6.3 Parallel Processing . . . 45
6.4 Extra-Large Matrices, RAM and Disk Space Management . . . 46
6.5 Some Sugar and Recipes . . . 47
6.5.1 Plotting Methylation Value Distributions . . . 47
6.5.2 Plotting Low Dimensional representations . . . 49
6.5.3 Generating Locus Profile Plots (aka Genome-Browser-Like Views) . . . 49
6.5.4 Adjusting for Surrogate Variables in Differential Methylation Analysis . . . 50
6.5.5 Generating Density-Scatterplots . . . 51
6.6 Correcting for Cell Type Heterogeneity Effects . . . 51
6.7 Deploying RnBeads on a Scientific Compute Cluster . . . 52
6.8 Genome-wide segmentation of the methylome . . . 54

7 Beyond DNA Methylation Analysis **55**
7.1 HTML Reports . . . 55
7.2 Logging . . . 58

1 Getting Started

1.1 Installation

Automatic installation of `RnBeads` and one or more of its companion data packages is performed just like any other Bioconductor package. As an example, the following commands install RnBeads and its annotation package for the human genome annotation hg38:

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
+     install.packages("BiocManager")
> BiocManager::install(c("RnBeads", "RnBeads.hg38"))
```

Alternatively, we also provide a an easy-to-use installation script that ensures all dependencies are installed. This way, you can install RnBeads by running a single line of code in your R environment:

```
> source("http://rnbeads.org/data/install.R")
```

Note that you might have to install *Ghostscript* which `RnBeads` uses for generating plot files. *Ghostscript* is included in most Unix/Linux distributions by default, but may require installation on other operating systems. You can obtain the version corresponding to your operating system freely from the program's website[1]. Follow their installation instructions. You might have to set path variables as well. Additional hints on installation of Ghostscript can be found on the FAQ section of the RnBeads website[2]. The website also describes the installation process in detail.

1.2 Overview of `RnBeads` Modules

`RnBeads` implements a comprehensive analysis pipeline from data import via filtering, normalization and exploratory analyses to characterizing differential methylation. Its modularized design allows for conducting simple vanilla types of analysis (with the need to specify only a few parameters; `RnBeads` automatically takes care of the rest) as well as targeted, highly customizable analysis. Multiple input formats are supported. Moreover, advanced code logging capabilities are integrated into the package. Finally, the output can be presented as comprehensive, interpretable analysis reports in `html` format. Figure 1 visualizes the standard workflow through `RnBeads` modules. Analysis modules include:

Data Import Various input formats are supported both for Infinium and bisulfite sequencing data.

Quality Control If quality control data is available from the input (e.g. in `IDAT` format files for Infinium data), QC based on the various types of QC probes on the Infinium methylation chip is performed. For bisulfite sequencing experiments, QC analysis is based on coverage information.

Preprocessing Includes probe and sample filtering. This happens in an automated fashion after the user specifies filtering criteria in which he is interested. Furthermore, for Infinium 450k data a number of microarray normalization methods are implemented. The normalization report helps to assess the effects of the normalization procedure.

Tracks and Tables Methylation profiles can be exported to bed, bigBed and bigWig file formats and visualized in various genome browsers via the export to track hubs.

Covariate Inference In this optional module, sample covariates potentially confounding the analysis can be identified and quantified. In consecutive modules, e.g. they can be taken into account and corrected for during differential methylation analysis.

Exploratory Analysis Dimension reduction, statistical association tests and visualization techniques are applied in order to discover associations of the methylation data with various sample characteristics specified in the sample annotation sheet provided by the user. Methylation fingerprints of samples and sample groups are identified and presented. Unsupervised learning techniques (clustering) are applied.

[1] http://www.ghostscript.com/download/gsdnld.html
[2] https://rnbeads.org/

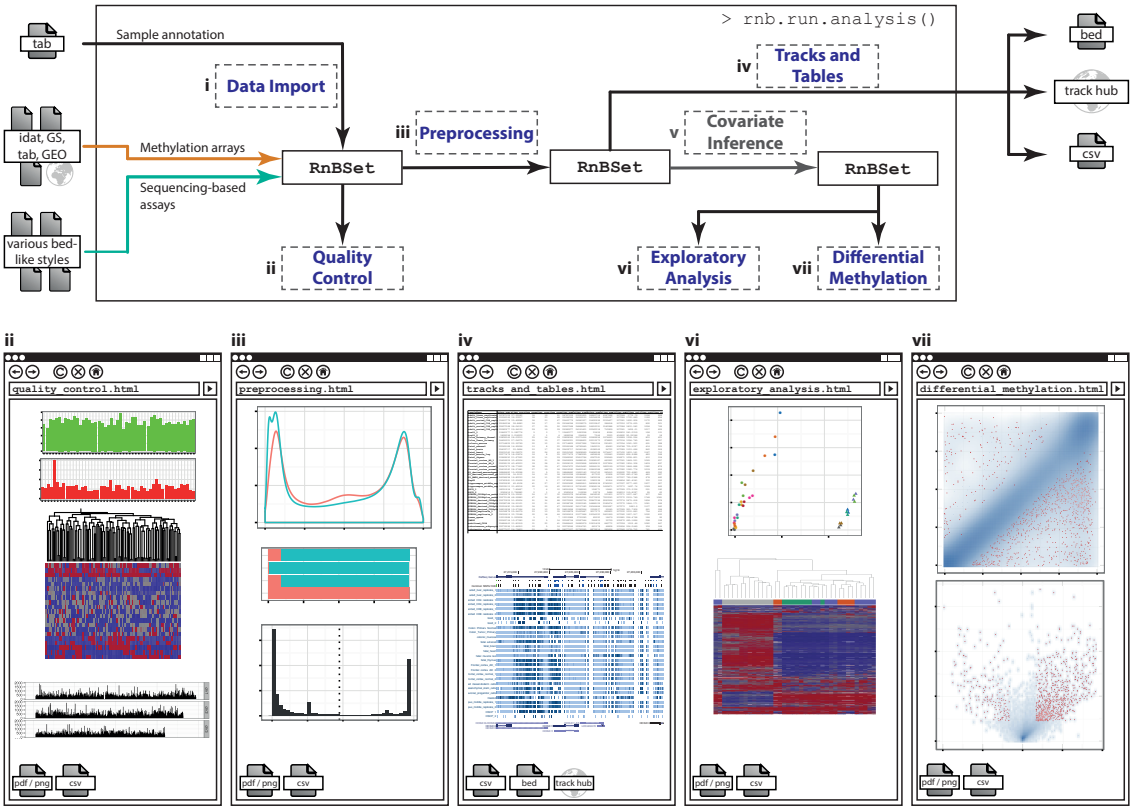

Figure 1: Workflow and modules in RnBeads

Differential Methylation Analysis Provided with categorical sample, differentially methylated sites and genomic regions are identified, their degree of differential methylation is quantified and visualized. Furthermore, differentially methylated regions (DMRs) can be annotated, e.g. with enrichment analysis.

1.3 Datasets

In this vignette, we first describe how to analyze data generated by the Infinium 450k methylation array. Section 5 describes how to handle bisulfite sequencing data. The general concepts of `RnBeads` are shared for array and sequencing data. In fact, for the most part, the same methods and functions are used. Therefore, the 450k examples will also be very instructive for the analysis of bisulfite sequencing data. We will introduce basic features of `RnBeads` using a dataset of Infinium 450k methylation data from multiple human embryonic stem cells (hESCs) and induced pluripotent stem cells (hiPSCs) lines. This dataset has been published as a supplement to a study on non-CpG methylation [1] and can be downloaded as a zip file from the `RnBeads` examples website[3]. Note that while the article [1] focuses on non-CpG methylation, we will only investigate methylation in the CpG context here. Before working with the data, unzip the archive to a directory of your choice.

1.4 Setting Up the Analysis Environment

Before we get started, we need to carry out some preparations, such as loading the package and specifying input and output directories:

```
> library(RnBeads)
> # Directory where your data is located
> data.dir <- "~/RnBeads/data/Ziller2011_PLoSGen_450K"
> idat.dir <- file.path(data.dir, "idat")
> sample.annotation <- file.path(data.dir, "sample_annotation.csv")
> # Directory where the output should be written to
> analysis.dir <- "~/RnBeads/analysis"
> # Directory where the report files should be written to
> report.dir <- file.path(analysis.dir, "reports")
```

We will use these variables throughout this vignette whenever we mean to specify input and output files. Of course, you have to adapt the directories corresponding to your own file system and operating system. The zip file you downloaded contains a directory that entails a sample annotation sheet (we store its location in the `sample.annotation` variable in the code above) and a directory containing the IDAT files that hold the methylation data (`idat.dir`).

2 Vanilla Analysis

So, let's get our hands dirty and perform an analysis. First we may want to specify a few analysis parameters:

```
> rnb.options(filtering.sex.chromosomes.removal=TRUE,
+     identifiers.column="Sample_ID")
```

[3] https://rnbeads.org/examples.html

This tells RnBeads to remove all probes on sex chromosomes for the analysis in the filtering step. Furthermore, RnBeads will look for a column called "Sample_Name" in the sample annotation sheet and use its contents as identifiers in the analysis. Feel free to set other options as well (you can inspect available analysis options using ?rnb.options). More details on analysis options can be found in Section 6.1 of this vignette.

The main function in RnBeads is rnb.run.analysis(). It executes all analysis modules, as specified in the package options and generates a dedicated report for each module. The method has multiple arguments, the most important of which are those specifying the type of the data to be analyzed, the location of the methylation data and sample annotation, and the location of the output directory. In this example we are dealing with data from the Infinium 450k microarray and thus, for a simple analysis run we execute rnb.run.analysis using four arguments. Specifically, data.dir defines the location of the input IDAT intensity files. A sample annotation sheet is essential for every analysis. It is a tabular file containing sample information such as sample identifiers, file locations, phenotypic information, batch processing information and potential confounding factors. Feel free to inspect the sample annotation sheet provided in the dataset you just downloaded to get an impression on how such annotation can be structured. rnb.run.analysis() finds the location of the annotation table using the sample.sheet parameter. report.dir specifies the location of the output directory. Make sure that this directory is non-existing, as otherwise RnBeads will not be able to create reports there. Finally, we have to specify the type of the input data which should be "idat.dir" for processing IDAT files. Now we can run our analysis with our defined options:

```
> rnb.run.analysis(dir.reports=report.dir, sample.sheet=sample.annotation,
+     data.dir=idat.dir, data.type="infinium.idat.dir")
```

Note that in the above code we specified the arguments explicitly, i.e. we used the syntax argument.name=argument.value. It is good practice to stick to this explicit argument statement.

Now, as running through the entire analysis pipeline takes a while, get a cup of coffee. You might also want to browse through some of the examples on the RnBeads website to get an idea of what kind of results you can expect once the analysis is done. Anyhow, it might take a while (on our testing machine, roughly four hours for the complete analysis).

After the analysis has finished, you can find a variety of files in the report directory you specified. Of particular interest are the html files containing the analysis results and links to tables and figures which are stored in the output directory structure. Furthermore, analysis.log is a log file containing status information on your analysis. Here, you can check for potential errors and warnings that occurred in the execution of the pipeline.

Now, feel free to inspect the actual results of our analysis. All of them are stored as html report files. A good starting point is the index.html in your reports directory which can be opened in your favorite web browser. This overview page provides the links to the analysis reports of the individual modules. You can also access the individual reports directly by opening the corresponding html files in the reports directory. These reports contain method descriptions of the conducted analyses as well as the results. The reports are designed to be self-contained and self-explanatory, therefore, we do not discuss them in detail here. Take your time browsing through the results of the analysis and getting an idea of what RnBeads can do for you.

The reports provide a convenient way of sharing results and re-inspecting them later. It is particularly easy to share results over the Internet if you have webspace available somewhere. Simply upload the entire report directory to a server and send a corresponding link to your collaborators. If you compress the the analysis report directories you can also share them via

cloud services such as Google Drive and Dropbox. You can find more example reports on the RnBeads website.

In your subsequent analysis runs you might not want to execute the full-scale RnBeads pipeline. You can deactivate individual modules by setting the corresponding global RnBeads options before running the analysis. For instance, you could deactivate the differential methylation module:

```
> rnb.options(differential=FALSE)
```

3 Working with RnBSet Objects

Every analysis workflow in RnBeads is centered around a dataset object, which is implemented as an R object of type RnBSet (an S4 class). This object contains the sample annotation table, methylation values for individual sites and regions, as well as additional, platform-specific data. The function rnb.run.analysis, introduced in the previous section, returns such an object (invisibly) which contains the processed dataset. RnBeads provides a multitude of functions operating on a dataset; some of these functions retrieve information from the dataset, others create plots summarizing some of the data contained, or modify the methylation values or sample annotation. In this section, we introduce the RnBSet class and show a few examples on how to work with these objects in an interactive R session. The following sections give more details by describing each of the RnBeads modules. With the exception of the import module, which prepares such objects from input data files, all of these modules operate directly on RnBSet objects.

RnBSet objects come in different flavors (subclasses), depending on the data type that was used to create them: RnBeadSet and RnBeadRawSet objects are derived from processed and unprocessed array data respectively, while RnBiseqSet objects result from processing bisulfite-sequencing data.

Here, we use a small example dataset which is contained in the RnBeads.hg19 package and comprises a subset of probes of the Infnium 450K dataset introduced earlier in this vignette. Readers are encouraged to try the functions presented here on the full dataset or on their own datasets. As a first example, by printing a dataset you can see a summary of its type and size.

```
> library(RnBeads.hg19)
> data(small.example.object)
> rnb.set.example

Object of class RnBeadRawSet
      12 samples
    1736 probes
        of which: 1730 CpG, 6 CpH, 0 rs and 0 nv
Region types:
             470 regions of type tiling
             184 regions of type genes
             184 regions of type promoters
             137 regions of type cpgislands
Intensity information is present
Detection p-values are present
Bead counts are present
```

```
Quality control information is present
Summary of normalization procedures:
        The methylation data was not normalized.
        No background correction was performed.
```

As shown in the code snippet above, the example dataset consists of 12 samples × 1,736 probes. In addition to measurements at individual CpGs, it has summarized the methylation state for 470 genomic tiling regions, 184 gene bodies, 184 gene promoters and 137 CpG islands. Notice the first line of printed for this dataset, it states that it is an object of type `RnBeadRawSet`. These objects denote array-based datasets containing array-specific measurements, such as probe intensity values, detection p-values, and others.

You can use the following functions in order to obtain the number of CpGs covered and the sample identifiers of the dataset:

```
> nsites(rnb.set.example)

[1] 1736

> samples(rnb.set.example)

 [1] "hES_HUES13_p47"    "hiPS_20b_p43"      "hES_HUES1_p29"
 [4] "hiPS_11c_p23"      "hES_HUES1_p28"     "hiPS_20b_p49_TeSR"
 [7] "hiPS_20b_p49_KOSR" "hiPS_17b_p35_TeSR" "hiPS_27b_p31"
[10] "hES_HUES64_p19"    "hiPS_17b_p35_KOSR" "hES_H9_p58"
```

The following command shows the the first 4 columns of the sample annotation table:

```
> pheno(rnb.set.example)[, 1:4]

           Sample_ID Sentrix_ID Sentrix_Position   Sample_Plate
1     hES_HUES13_p47 5815381013           R03C01 WG0001341-MSA4
2       hiPS_20b_p43 5815381013           R05C02 WG0001341-MSA4
3      hES_HUES1_p29 5815381013           R02C01 WG0001341-MSA4
4       hiPS_11c_p23 5815381013           R04C02 WG0001341-MSA4
5      hES_HUES1_p28 5815381013           R01C01 WG0001341-MSA4
6  hiPS_20b_p49_TeSR 5815381013           R03C02 WG0001341-MSA4
7  hiPS_20b_p49_KOSR 5815381013           R02C02 WG0001341-MSA4
8  hiPS_17b_p35_TeSR 5815381013           R01C02 WG0001341-MSA4
9       hiPS_27b_p31 5815381013           R06C02 WG0001341-MSA4
10    hES_HUES64_p19 5815381013           R04C01 WG0001341-MSA4
11 hiPS_17b_p35_KOSR 5815381013           R06C01 WG0001341-MSA4
12        hES_H9_p58 5815381013           R05C01 WG0001341-MSA4
```

Methylation values for individual CpGs can be obtained using the `meth` command:

```
> mm <- meth(rnb.set.example)
```

Let's take look at the distribution of methylation values in sample number 5 of our dataset:

```
> hist(mm[,5], col="steelblue", breaks=50)
```

RnBSet objects also contain summarized methylation levels for various region types. To inspect which region types are represented in an object use summarized.regions:

```
> summarized.regions(rnb.set.example)

[1] "tiling"     "genes"      "promoters"  "cpgislands"
```

The next commands instruct RnBeads that the first column in the annotation table denotes sample identifiers, and extract the methylation beta values at the first five gene promoters for the first three samples:

```
> rnb.options(identifiers.column="Sample_ID")
> meth(rnb.set.example, type="promoters", row.names=TRUE, i=1:5, j=1:3)

                hES_HUES13_p47 hiPS_20b_p43 hES_HUES1_p29
ENSG00000131591     0.05520151   0.06547974    0.05503473
ENSG00000117069     0.05308049   0.04594461    0.05725949
ENSG00000162694     0.06312758   0.06848181    0.07229479
ENSG00000273204     0.11724093   0.12067451    0.09609808
ENSG00000162695     0.07275482   0.07489990    0.07031157
```

Similarly, the method mval is used to extract M values:

```
> mval(rnb.set.example, type="promoters", row.names=TRUE)[1:5, 1:3]

                hES_HUES13_p47 hiPS_20b_p43 hES_HUES1_p29
ENSG00000131591      -4.097227    -3.835106     -4.101847
ENSG00000117069      -4.156988    -4.376105     -4.041274
ENSG00000162694      -3.891510    -3.765791     -3.681703
ENSG00000273204      -2.912543    -2.865276     -3.233587
ENSG00000162695      -3.671836    -3.626574     -3.724913
```

To get the coordinates and additional annotation for sites or regions contained in the dataset, use the annotation function:

```
> annot.sites <- annotation(rnb.set.example)
> head(annot.sites)

           Chromosome   Start     End Strand      names Strand.1 AddressA
cg24488772       chr1 1017318 1017319      - cg24488772        - 72623496
cg24378253       chr1 1017383 1017384      - cg24378253        - 48673390
cg04794393       chr1 1018118 1018119      - cg04794393        - 65630381
cg17840536       chr1 1018254 1018255      + cg17840536        + 67655486
cg19758750       chr1 1018479 1018480      + cg19758750        + 46688365
cg18024260       chr1 1019860 1019861      - cg18024260        - 59711429
           AddressB Design Color Context Random HumanMethylation27 Mismatches A
cg24488772 28718406      I   Red      CG  FALSE              FALSE            0
cg24378253 38614336      I   Red      CG  FALSE              FALSE            0
cg04794393 22788466      I   Red      CG  FALSE              FALSE            0
cg17840536 26703405      I   Grn      CG  FALSE              FALSE            0
```

```
cg19758750 46688365     II  Both      CG  FALSE              FALSE            0
cg18024260 42807375      I   Red      CG  FALSE              FALSE            0
           Mismatches B CGI Relation CpG GC SNPs 3 SNPs 5 SNPs Full
cg24488772            0       Island   4 64      0      0         0
cg24378253            0       Island  10 74      0      0         1
cg04794393            1       Island   8 72      0      0         0
cg17840536            0       Island   9 71      0      0         0
cg19758750            0  North Shore   4 66      0      0         0
cg18024260            0       Island   8 67      0      0         0
           Cross-reactive
cg24488772              0
cg24378253              0
cg04794393              0
cg17840536              0
cg19758750              0
cg18024260              0
```

```
> annot.promoters <- annotation(rnb.set.example, type="promoters")
> head(annot.promoters)
```

```
                Chromosome     Start       End Strand           names
ENSG00000131591       chr1   1051242   1053241      - ENSG00000131591
ENSG00000117069       chr1  77331626  77333625      + ENSG00000117069
ENSG00000162694       chr1 101361055 101363054      - ENSG00000162694
ENSG00000273204       chr1 101358984 101360983      + ENSG00000273204
ENSG00000162695       chr1 101360132 101362131      + ENSG00000162695
ENSG00000240386       chr1 152747348 152749347      + ENSG00000240386
                    symbol entrezID CpG   GC   C   G
ENSG00000131591   C1orf159    54991 187 1367 727 640
ENSG00000117069 ST6GALNAC5    81849  60  847 420 427
ENSG00000162694      EXTL2     2135  79 1043 519 524
ENSG00000273204       <NA>     <NA>  52  825 456 369
ENSG00000162695    SLC30A7   148867 113 1123 588 535
ENSG00000240386      LCE1F   353137  21  992 461 531
```

Furthermore, indices of overlapping regions can be added to each annotated methylation site.

```
> annot.sites.with.rgns <- annotation(rnb.set.example, include.regions=TRUE)
> head(annot.sites.with.rgns)
```

```
           Chromosome   Start     End Strand      names Strand.1 AddressA
cg24488772       chr1 1017318 1017319      - cg24488772        - 72623496
cg24378253       chr1 1017383 1017384      - cg24378253        - 48673390
cg04794393       chr1 1018118 1018119      - cg04794393        - 65630381
cg17840536       chr1 1018254 1018255      + cg17840536        + 67655486
cg19758750       chr1 1018479 1018480      + cg19758750        + 46688365
cg18024260       chr1 1019860 1019861      - cg18024260        - 59711429
           AddressB Design Color Context Random HumanMethylation27 Mismatches A
cg24488772 28718406      I   Red      CG  FALSE              FALSE            0
```

```
cg24378253 38614336      I   Red      CG  FALSE              FALSE            0
cg04794393 22788466      I   Red      CG  FALSE              FALSE            0
cg17840536 26703405      I   Grn      CG  FALSE              FALSE            0
cg19758750 46688365     II  Both      CG  FALSE              FALSE            0
cg18024260 42807375      I   Red      CG  FALSE              FALSE            0
           Mismatches B CGI Relation CpG GC SNPs 3 SNPs 5 SNPs Full
cg24488772           0       Island   4 64      0      0         0
cg24378253           0       Island  10 74      0      0         1
cg04794393           1       Island   8 72      0      0         0
cg17840536           0       Island   9 71      0      0         0
cg19758750           0  North Shore   4 66      0      0         0
cg18024260           0       Island   8 67      0      0         0
           Cross-reactive tiling genes promoters cpgislands
cg24488772              0      1     1         0          1
cg24378253              0      1     1         0          1
cg04794393              0      1     1         0          2
cg17840536              0      1     1         0          2
cg19758750              0      1     1         0          0
cg18024260              0      1     1         0          3
```

You can manipulate `RnBSet` objects using several methods. Here, we show you how to remove samples and sites from the dataset and how to add columns to the sample annotation table:

```
> # Remove samples
> rnb.set2 <- remove.samples(rnb.set.example, samples(rnb.set.example)[c(2, 6, 10)])
> setdiff(samples(rnb.set.example), samples(rnb.set2))

[1] "hiPS_20b_p43"      "hiPS_20b_p49_TeSR" "hES_HUES64_p19"

> # Remove probes which are not in CpG context
> notCpG <- annot.sites[,"Context"]!="CG"
> rnb.set.example <- remove.sites(rnb.set.example, notCpG)
> nsites(rnb.set.example)

[1] 1730

> # Add a sample annotation column indicating whether the given
> # sample represents an iPS cell line
> is.hiPSC <- pheno(rnb.set.example)[, "Sample_Group"]=="hiPSC"
> rnb.set.example <- addPheno(rnb.set.example, is.hiPSC, "is_hiPSC")
> pheno(rnb.set.example)[, c("Sample_ID", "Cell_Line", "is_hiPSC")]

           Sample_ID  Cell_Line is_hiPSC
1     hES_HUES13_p47 hES_HUES13    FALSE
2       hiPS_20b_p43   hiPS_20b     TRUE
3      hES_HUES1_p29  hES_HUES1    FALSE
4       hiPS_11c_p23   hiPS_11c     TRUE
5      hES_HUES1_p28  hES_HUES1    FALSE
6  hiPS_20b_p49_TeSR   hiPS_20b     TRUE
```

```
7  hiPS_20b_p49_KOSR   hiPS_20b     TRUE
8  hiPS_17b_p35_TeSR   hiPS_17b     TRUE
9       hiPS_27b_p31   hiPS_27b     TRUE
10    hES_HUES64_p19 hES_HUES64    FALSE
11 hiPS_17b_p35_KOSR   hiPS_17b     TRUE
12        hES_H9_p58     hES_H9    FALSE
```

Since the example object was derived from an Infinium 450k experiment it also stores low-level information about the corresponding experiment, such as measured intensities in each of the color channels, out-of-band intensities, number of beads, detection p-values etc:

```
> # Methylated ...
> Mint <- M(rnb.set.example, row.names=TRUE)
> Mint[1:5,1:3]

           hES_HUES13_p47 hiPS_20b_p43 hES_HUES1_p29
cg24488772          16781        17096         15753
cg24378253          20360        20248         18280
cg04794393          18826        19608         16931
cg17840536          21972        24487         20954
cg19758750          20029        24664         19166

> # ...and unmethylated probe intensities
> Uint <- U(rnb.set.example, row.names=TRUE)
> Uint[1:5,1:3]

           hES_HUES13_p47 hiPS_20b_p43 hES_HUES1_p29
cg24488772           2358         1579          2090
cg24378253            592          387           921
cg04794393            978          477          1045
cg17840536           3193         1441          3071
cg19758750           4271         2916          3872

> # Infinium bead counts
> nbead <- covg(rnb.set.example, row.names=TRUE)
> nbead[1:5,1:3]

           hES_HUES13_p47 hiPS_20b_p43 hES_HUES1_p29
cg24488772             10           14             8
cg24378253             14            8            13
cg04794393             14           10             6
cg17840536              6           12            13
cg19758750             16           15            11

> # detection P-values
> pvals <- dpval(rnb.set.example, row.names=TRUE)
> pvals[1:5,1:3]

           hES_HUES13_p47 hiPS_20b_p43 hES_HUES1_p29
cg24488772              0            0             0
```

```
cg24378253              0            0            0
cg04794393              0            0            0
cg17840536              0            0            0
cg19758750              0            0            0
```

Moreover, the object also contains quality control information. RnBeads offers a number of diagnostic plots visualizing this information:

```
> # boxplot of control probes
> rnb.plot.control.boxplot(rnb.set.example)
> # barplot of a selected control probe
> control.meta.data <- rnb.get.annotation("controls450")
> ctrl.probe<-paste0(unique(control.meta.data[["Target"]])[4], ".1")
> rnb.plot.control.barplot(rnb.set.example, ctrl.probe)
```

Finally, the full control probe intensities can be retrieved using method `qc`:

```
> qc_data<-qc(rnb.set.example)
```

`qc_data` is a list with elements `Cy3` and `Cy5` containing all Infinium control probe intensities for the green and red color channels respectively. It can be used for more advanced QC procedures.

4 Tailored Analysis: RnBeads Modules

In this section, we show how the package's modules can be invoked individually given suitable options and input data. Most modules operate on the `RnBSet` objects introduced in the previous section. Some modules depend on data generated by other modules (c.f. Figure 1). We also introduce useful parameter settings and report-independent stand-alone methods for methylation analysis.

Note that the examples in this section operate on the complete dataset and therefore potentially take longer to execute than the operations that were executed on the reduced dataset introduced in the previous section.

To get started, let us initialize a new report directory prior to our analysis. Keep in mind that `RnBeads` does not overwrite existing reports and thus the analysis fails when a report's HTML file or subdirectory already exists.

```
> report.dir <- file.path(analysis.dir, "reports_details")
> rnb.initialize.reports(report.dir)
```

Additionally, as we do not need to store the logging messages in a file and just want them printed, we restrict logging to the console:

```
> logger.start(fname=NA)
```

4.1 Data Import

RnBeads supports multiple input formats for microarray-based and bisulfite sequencing data. The following paragraphs focus on Infinium 450k methylation data sets. Section 5 describes in details how bisulfite sequencing data can be loaded.

Here, we use a more standardized way to specify the input location compared to the one in Section 2: the `data.source` argument. Its value is dependent on the nature of the input data. Table 4.1 lists all supported data types together with the recommended format for the `data.source` argument. Since the input data in our example consists of IDAT files, `data.source` should be a vector of two elements, containing (1) the directory the IDAT files are stored in and (2) the file containing the sample annotation table.

```
> data.source <- c(idat.dir, sample.annotation)
```

We can load the dataset into an `RnBeadSet` object:

```
> result <- rnb.run.import(data.source=data.source,
+     data.type="infinium.idat.dir", dir.reports=report.dir)
> rnb.set <- result$rnb.set
```

The new variable `result` is a list with two elements: the dataset (`rnb.set`) and a report object (see Section 7.1). Every `rnb.run.*()` method creates a report. The generated import report (`import.html`) contains information on the loaded dataset including data type, number of loaded samples and sites, description of the parsed annotation table etc. If you wish to parse a methylation dataset into an `RnBSet` object without producing an analysis report, you can use `rnb.execute.import`:

```
> rnb.set <- rnb.execute.import(data.source=data.source,
+         data.type="infinium.idat.dir")
```

`rnb.set` is an object of class `RnBeadRawSet`. This S4 class inherits from class `RnBSet` which is the main data container class in `RnBeads` and serves as input for many methods of the analysis pipeline. Just like other `RnBSet` objects, the `RnBeadRawSet` class contains slots for sample annotation (`pheno`) and methylation values of sites and predefined genomic regions (accessible by the `meth` function). In addition, it contains raw microarray intensities (`M` and `U` functions), detection p-values (`dpval`) and quality control information (`qc`):

```
> rnb.set
> summary(pheno(rnb.set))
> dim(M(rnb.set))
> summary(M(rnb.set))
> dim(meth(rnb.set))
> summary(meth(rnb.set))
> dim(meth(rnb.set, type="promoters"))
> summary(meth(rnb.set, type="promoters"))
> summary(dpval(rnb.set))
> str(qc(rnb.set))
```

Notably, `RnBeads` accepts GEO accession numbers as input. Here is an example of a somewhat larger dataset, from another study on pluripotent and differentiated cells [2]:

```
> rnb.set.geo <- rnb.execute.import(data.source="GSE31848",
+         data.type="infinium.GEO")
```

As the dataset is fairly large, downloading and import might take a while. Note that data sets loaded from GEO typically do not include detection p-values and QC probe information. Therefore, most of the normalization methods and some of the subsequent reports and analyses cannot be carried out. Examples include parts of the *Quality Control* module or probe filtering based on detection p-values as performed by the *Preprocessing* module.

Table 1: RnBeads input data types.

data.type	data.source recommended format[a]
Infinium 450k data	
infinium.idat.dir	A character vector of length 1 or 2 containing the directory in which the IDAT files are stored and (optionally) the filename of the sample annotation table (if the latter is located in a different directory)
infinium.GS.report	A character vector of length 1 containing the filename of a Genome Studio report
infinium.GEO	A character vector of length 1 containing the GEO accession number
infinium.data.dir	A character vector of length 1 giving the name of the target directory. The filenames in the directory should contain key words: sample for the sample information table, beta for the table with beta-values, pval for p-values table, and beads for bead counts
infinium.data.files	A character vector of length 2 to 4. At minimum paths to the sample annotation table and to a table containing beta-values should be given. Additionally, paths to a table with detection p-values, and bead counts can be specified
Bisulfite Sequencing data	
bs.bed.dir	A character vector of length 3. It contains [1] the directory where the methylation bed files can be found, [2] the filename of the sample annotation table, and optionally [3] the index of the sample annotation file column, which contains the file names

[a] Alternative formats of the data.source argument are possible; please refer to the R documentation of the rnb.execute.import function

4.2 Quality Control

RnBeads generates quality control plots both for Infinium 450k and bisulfite sequencing data. Experimental quality control for the Infinium 450k data is performed using the microarray control probe information which should be present in the input data. We recommend starting out from the IDAT files which contain this information. Table 4.2 gives short description of each control probe type represented on the array. The following paragraphs describe some QC plots targeting Infinium 450k datasets only. For the quality control of bisulfite sequencing data, sequencing coverage is taken into account and its distribution is visualized in a series of plots. For more details see Section 5.

```
> rnb.run.qc(rnb.set, report.dir)
```

is the command which generates the QC report (qc.html). For Infinium data, this report contains three major quality control plots: *control boxplot*, *control barplot* and *negative control boxplot* that are briefly explained below.

Control Boxplot is a diagnostic plot showing the distribution of signal intensity for each of the quality control probes in both of the channels (Figure 2). The expected intensity level (high, medium, low or background) is given in the plot labels. The boxplots provide a useful tool for

Table 2: Types of Infinium 450k control probes.

Control type	**Purpose**
Sample-independent controls	
STAINING	Monitor the efficiency of the staining step. These controls are independent of the hybridization and extension step.
HYBRIDIZATION	Monitor overall hybridization performance using synthetic targets as a reference. The targets are present in the hybridization buffer at three concentrations: low, medium and high, which should result in three well separable intensity intervals.
EXTENSION	Monitor the extension efficiency of A, T, C, and G nucleotides from a hairpin probe
TARGET REMOVAL	Monitor efficiency of the stripping step after the extension reaction
Sample-dependent controls	
BISULFITE CONVERSION I	Monitor bisulfite conversion efficiency as reported by Infinium I design probes. Converted (C) and unconverted (U) probes are present
BISULFITE CONVERSION II	Monitor bisulfite conversion efficiency as reported by the Infinium II design probes
SPECIFICITY I	Monitor allele-specific extension for Infinium I probes. PM probes should give high signal, while the MM probes should give background signal levels
SPECIFICITY II	Monitor allele-specific extension for Infinium II probes
NON-POLYMORPHIC	Monitor the efficiency of all steps of the procedure by querying a non-polymorphic base in the genome – one probe for each nucleotide
NEGATIVE	Monitor signal at bisulfite-converted sequences that do not contain CpGs. Used to estimate the overall background of the signal in both channels. Moreover, the detection p-value for each probe are estimated based on the intensities of the negative control probes
NORM	Normalization controls are used to calculate the scaling factor for the Illumina default normalization

detecting experimentally failed assays. Low quality samples, if any, will appear as outliers. When examining the boxplot pay attention to the intensity scale.

Control Barplot focuses on individual samples. In case problematic behavior has been spotted in the control boxplots, the low quality samples can be identified by inspecting the barplots. Here the intensity for each sample measured by each of the control probe types is shown (Figure 3). The problematic samples should be excluded from the analysis by modifying the supplied sample annotation sheet or RnBeadSet object (see Section 4.1).

Negative Control Boxplots show distributions of intensities for the approximately 600 negative control probes which are present on the Infinium 450k array. The negative control probes are specifically designed for estimating the background signal level in both channels. In both channels

the negative control probe intensities are expected to be normally distributed around a relatively low mean (as a simple rule, the 0.9 quantile should be below 1000). Any strong deviations from such a picture in one or more samples may indicate quality issues; discarding such samples could be beneficial for downstream analyses.

For a loaded dataset stored in variable `rnb.set`, one can generate quality control plots directly. For instance you can generate a boxplot specifying the type of quality control probe as the second argument:

```
> rnb.plot.control.boxplot(rnb.set, "BISULFITE CONVERSION I")
```

To get a quality barplot, the second argument should be the unique identifier of the quality control probe:

```
> rnb.plot.control.barplot(rnb.set, "BISULFITE CONVERSION I.2")
```

Negative control boxplots are generated with the following command:

```
> rnb.plot.negative.boxplot(rnb.set)
```

The quality control report of the example dataset indicates high experimental quality. The quality boxplots for most of the controls show very compact intensity distributions. Let us, however, consider an example of quality control output where the experimental quality of the assay is suboptimal. In Figure 2 a boxplot of intensity measured by bisulfite conversion control probes is given for a data set of 17 samples. It is easy to spot the outliers which show quite high intensity in the green channel at probes for which background level intensities are expected (probes 4, 5 and 6). In order to identify problematic samples, let us examine the control barplot for one of the problematic probes `BISULFITE COVERSION I.6` (Figure 3). Note that sample `CellLineA_1` has an intensity which is at least four times higher than that of any other sample. This might indicate problems with bisulfite conversion and the problematic sample should be discarded. Since the quality information-based filtering of the Infinium 450k samples is as a rule performed via the inspection by the wet-lab researcher, low-quality samples could be removed, for example, by manual editing of the sample annotation file and restarting the analysis on the high-quality samples only.

Sample mixups The Infinium 450k BeadChip also contains a small number (65) of genotyping probes. Their dbSNP identifiers and additional information can be obtained using the following commands:

```
> annot450 <- rnb.annotation2data.frame(rnb.get.annotation("probes450"))
> annot450[grep("rs", rownames(annot450)), ]
```

Despite a certain level of technical variation, each SNP takes one of the three possible β-value levels: low (homozygous state, first allele), high (homozygous state, second allele) or intermediate (heterozygous). Examining these values can be very useful for the identification of sample mixups, especially in case of studies with genetically matched design. In this case, samples with the same genetic background should have very similar values for each of the 65 SNPs. Clustered heatmaps for the SNP-values are useful to obtain a global overview of genotype-related sample grouping and similarities and thus provide an efficient way of detecting sample-mixups. Figure 4.2 gives an example of such a heatmap. Such heatmaps are included into the quality control report by default, but can also be individually generated with the following command:

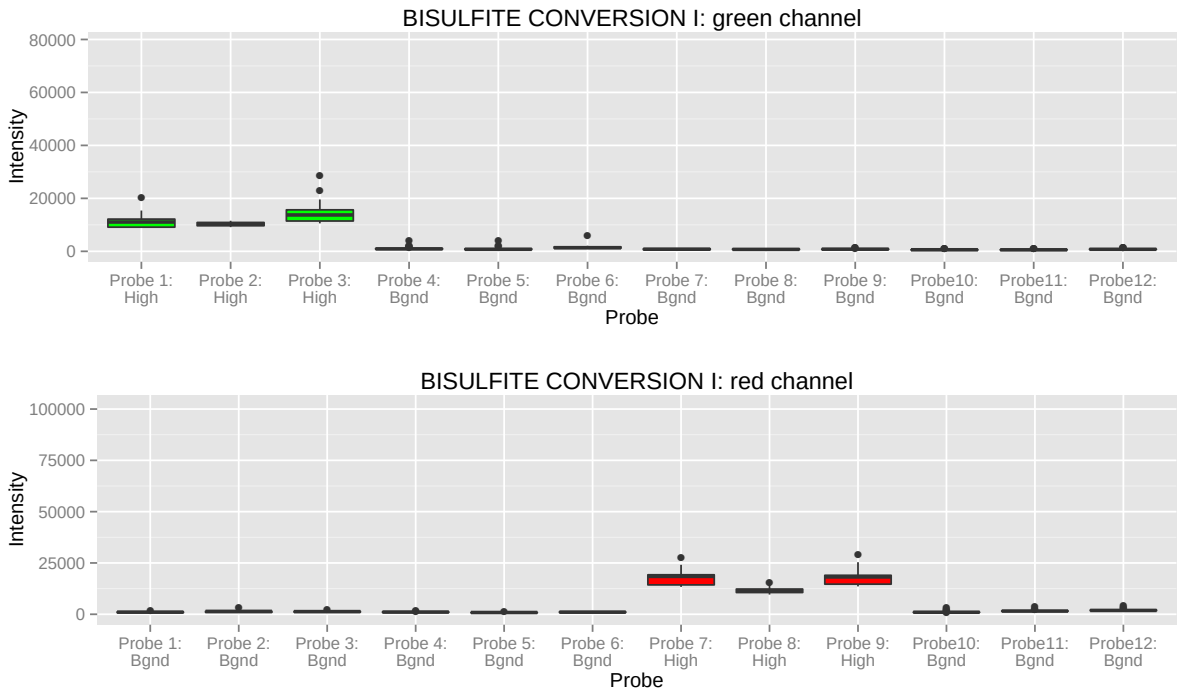

Figure 2: An example quality control boxplot.

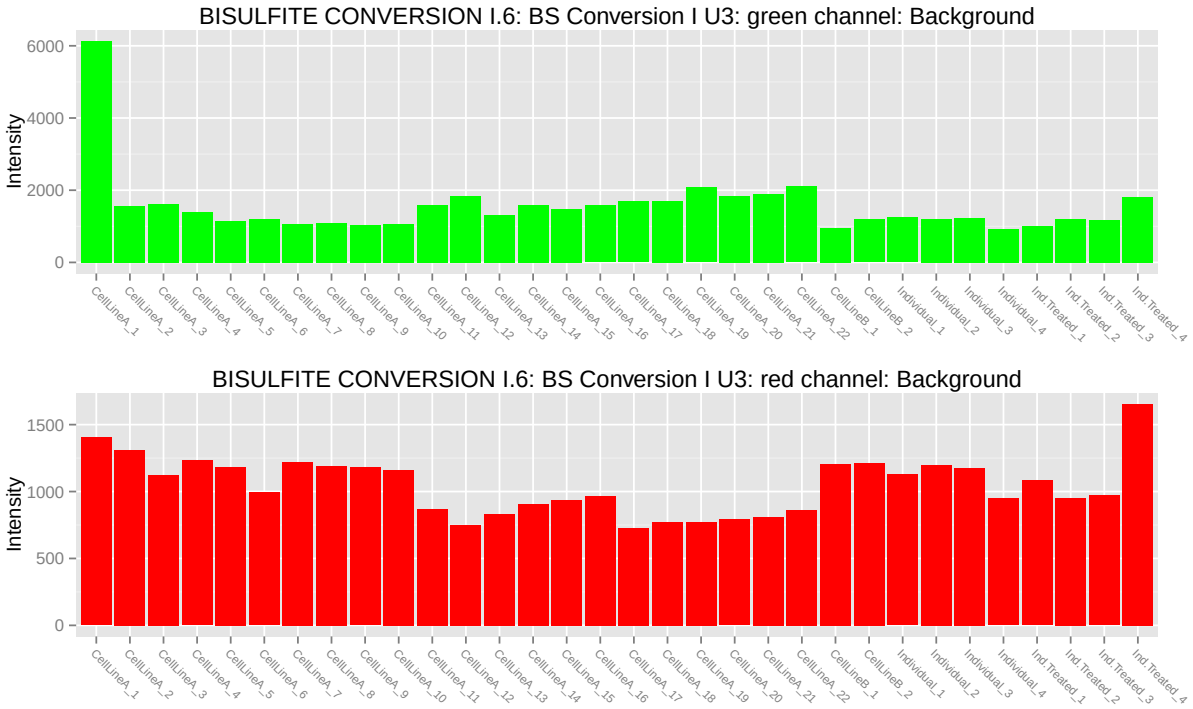

Figure 3: An example quality control barplot.

```
> rnb.plot.snp.heatmap(rnb.set.unfiltered)
```

SNP distance heatmap is an alternative way to get an overview of the SNP probes. The heatmap is a diagonal matrix visualizing coherence between each of the samples with respect to their SNP probe patterns.

SNP barplots visualize the SNP-probe beta-values directly:

```
> rnb.plot.snp.barplot(rnb.set.unfiltered, samples(rnb.set)[1])
```

Note that when executing `rnb.run.preprocessing` in the earlier example, we removed the SNP probe information from the `RnBSet` object. Hence, these plots have to be done based on the unfiltered object.

Finally, an optional *SNP boxplot* gives a global picture of the SNP probes and are helpful for the cases in which all the analyzed samples have the same genetic background (e.g. samples of of the same cell line). To include the SNP boxplot into the quality control report use the corresponding option:

```
> rnb.options(qc.snp.boxplot=TRUE)
```

4.2.1 Sex Prediction

`RnBeads` includes a method for inferring the sex of methylation samples. It is accessible using the function `rnb.execute.sex.prediction`. Currently, the classifier is applicable for Infinium 450k, EPIC and bisulfite sequencing datasets with either signal intensity or coverage information. Sex prediction is automatically executed when loading those datasets from IDAT/BED files; it can be disabled using the corresponding options, as shown below:

```
> rnb.options(import.sex.prediction = FALSE)
```

The methods for inferring sexes from Infinium datsets are different from those for bisulfite sequencing data sets. For Infinium datasets, the assumption made in RnBeads is that the quantity (number) of X and Y chromosomes in a sample is reflected in the signal intensities on the array. The measurements used in the prediction are MI_X: *mean increase in X* and MI_Y: *mean increaase in Y*. These metrics are defined as $\overline{I}_X - \overline{I}_a$ and $\overline{I}_Y - \overline{I}_a$, where $\overline{I}_X$, $\overline{I}_Y$ and $\overline{I}_a$ denote the average probe signal intensity on the X chromosome, the Y chromosome and all autosomes, respectively. Probes known to be cross-reactive, as well as those overlapping with known SNPs, are ignored in the calculations presented here. `RnBeads` uses these metrics in a logistic regression classifier to predict sex of a sample, $MI_Y > 3MI_X + 3$ indicates male sex. Figure 4.2.1 shows the *mean increase* values for over 9,200 samples with known sex, as well as the decision boundary of the classifier applied in `RnBeads`.
For bisulfite sequencing (RRBS or WGBS) datasets, `RnBeads` compares the sequencing coverages for the sex chromosomes to those of the autosomes. From these values, a logistic regression classifier is employed to predict class probabilities of each sample for the male and female category. Both the predicted probability and the predicted class are added to the `RnBSet's` sample annotation. Coefficients for the logistic regression classifier were trained on a large dataset (c.f. Figure 3) containing sex annotated samples.

4.2.2 CNV estimation

Copy number variation (CNV) can be estimated from signal intensity values of BeadArray data sets. RnBeads' CNV estimation module comes with two flavors: a reference-based and

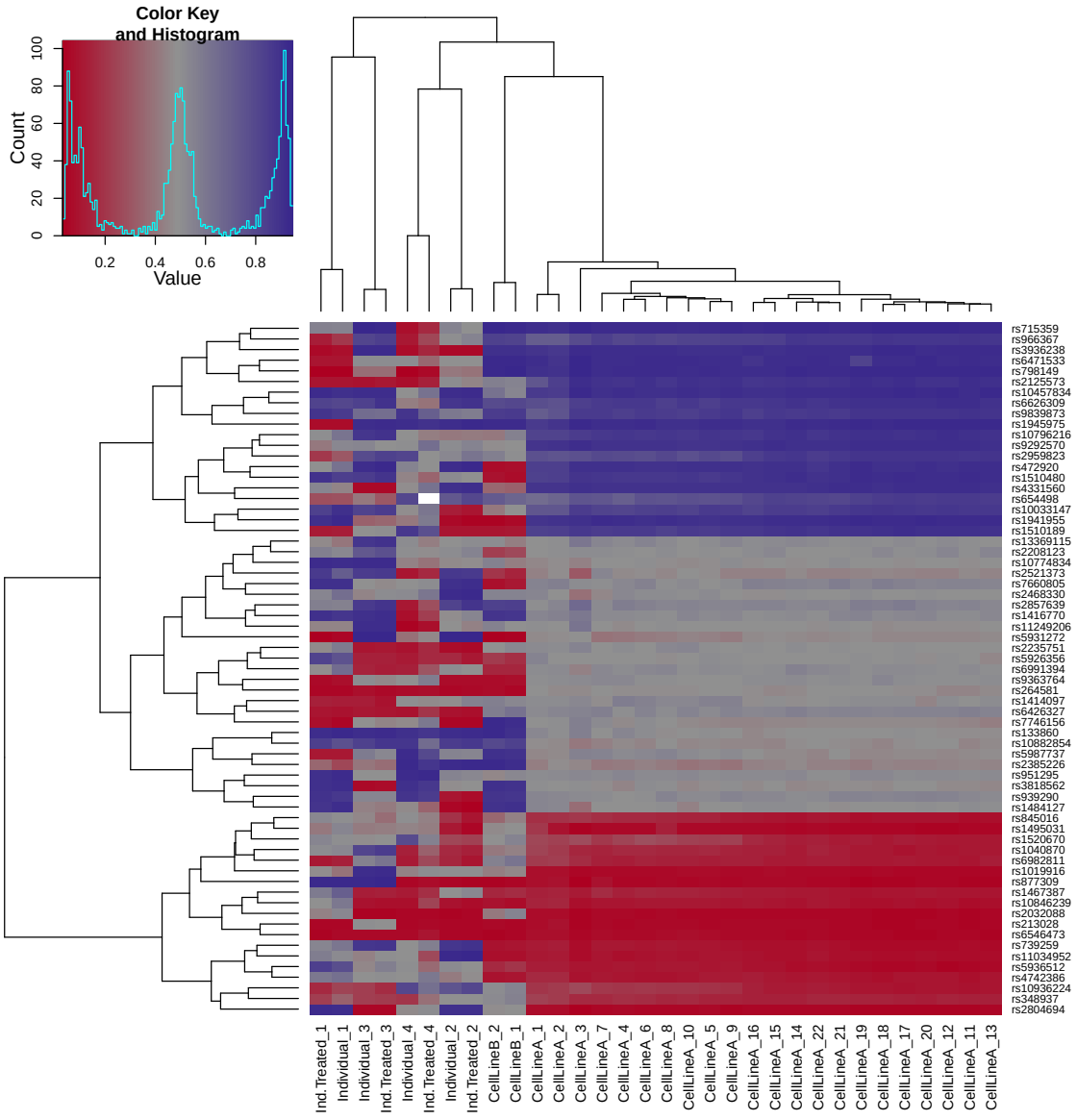

Figure 4: SNP heatmap

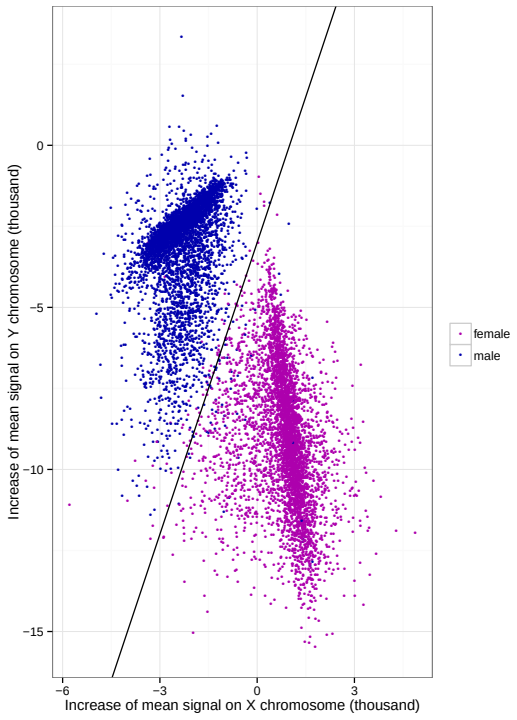

Figure 5: Mean signal increase in the X and Y chromosomes of 4587 female and 4633 male samples, represented by red and blue points, respectively. The presented dataset includes tumor and normal samples from 30 tumor types, downloaded from The Cancer Genome Atlas (TCGA). The solid black line shows the decision boundary in the classifier applied by RnBeads. Training a logistic regression model on the presented data produces an almost identical decision boundary.

Table 3: Dataset used to train coefficients of the sex prediction logistic regression model for bisulfite sequencing datasets.

Dataset	Source	Organism	Method	#total	#female	#male
BLUEPRINT (08/2016 release)	BLUEPRINT	human	WGBS	188	99	89
DEEP data (version 11/2016)	DEEP	human	WGBS	31	14	17
Kiel Cohort	IKMB Kiel	human	RRBS	239	106	133
Ewing sarcoma tumor samples	GEO GSE88826	human	RRBS	96	53	42
Reizel GEO	GEO GSE60012	mouse	RRBS	152	94	58
Complete				**706**	**366**	**339**

a reference-free approach. We recommend to use the reference-free approach, since validation of the reference data set is still to be conducted. CNV estimation is only to be executed within the QC module and not individually. It is activated by:

```
> rnb.options(qc.cnv=TRUE)
> result <- run.run.qc(rnb.set.unfiltered,dir.reports=report.dir)
```

The estimation is based on the GLAD package. We obtained the reference data set from two published twin studies (GSE85647 and GSE100940) as the median signal intensity for the probes. To activate the reference-based mode, you need to set:

```
> rnb.options(qc.cnv.refbased=TRUE)
```

In the default mode, RnBeads computes median signal intensities for all samples in the data set and uses this as the reference to detect potential outliers with respect to signal intensity.

4.3 Preprocessing

Several steps might be required to prepare the data for downstream analysis. Importantly, probe and sample filtering is carried out according to default or user specified criteria. Furthermore, for Infinium data, it is usually recommended to normalize the data. The *Preprocessing* module takes care of this. In a first filtering step, probes and samples that could bias the normalization procedure are removed. Subsequently, normalization is conducted for microarray datasets. Finally, additional filtering steps are potentially executed. The following commands run these steps and return a preprocessed `RnBSet` object:

```
> rnb.set.unfiltered <- rnb.set
> result <- rnb.run.preprocessing(rnb.set.unfiltered, dir.reports=report.dir)
> rnb.set <- result$rnb.set
```

The following two subsections contain details on the filtering and normalization steps, respectively.

4.3.1 Filtering

RnBeads facilitates filtering of sites and samples according to a variety of criteria. As noted above, the filtering steps are executed in two stages: one before and the other after data normalization[4]. In the default pipeline, these steps exclude Infinium probes overlapping with too many SNPs and sites with too many missing values. Additionally, if detection p-values are available, both sites and samples are filtered using a greedy approach (see the generated report for details).

You can also invoke individual filtering steps explicitly using the corresponding rnb.execute.* functions:

```
> nrow(meth(rnb.set.unfiltered)) # the number of sites in the unfiltered object
> # Remove probes outside of CpG context
> rnb.set.filtered <- rnb.execute.context.removal(rnb.set.unfiltered)$dataset
> nrow(meth(rnb.set.filtered)) # the number of CpG sites in the unfiltered object
> # SNP filtering allowing no SNPs in the probe sequence
> rnb.set.filtered <-
+     rnb.execute.snp.removal(rnb.set.filtered, snp="any")$dataset
> # the number of CpG sites in the unfiltered object
> # that do not contain a SNP
> nrow(meth(rnb.set.filtered))
> # Remove CpGs on sex chromosomes
> rnb.set.filtered <- rnb.execute.sex.removal(rnb.set.filtered)$dataset
> nrow(meth(rnb.set.filtered))
> # Remove probes and samples based on a greedy approach
> rnb.set.filtered <- rnb.execute.greedycut(rnb.set.filtered)$dataset
> nrow(meth(rnb.set.filtered))
> # Remove probes containing NA for beta values
> rnb.set.filtered <- rnb.execute.na.removal(rnb.set.filtered)$dataset
> nrow(meth(rnb.set.filtered))
> # Remove probes for which the beta values have low standard deviation
> rnb.set.filtered <-
+     rnb.execute.variability.removal(rnb.set.filtered, 0.005)$dataset
> nrow(meth(rnb.set.filtered))
```

Some of these filtering criteria do not apply to bisulfite sequencing data while others are added. See Section 5 for more details.

4.3.2 Normalization

For Infinium 450k data RnBeads includes possibilities for normalization of the microarray signals in order to decrease the level of technical noise and eliminate the systematic biases of the microarray procedure.

Currently, RnBeads supports several normalization methods:

- manufacturer-recommended scaling to the internal controls, implemented in the methylumi R-package [8],
- SWAN-normalization [4], as implemented in the minfi package [9],

[4] Note: for bisulfite sequencing date, no normalization is performed and hence the two filtering stages are executed consecutively.

- data-driven normalization methods [6] implemented in the `wateRmelon` package,
- the BMIQ procedure [10].

In addition, background subtraction methods implemented in the `methylumi` package – e.g. NOOB and Lumi – can be applied in combination with any of the above normalization methods [11]. Table 4.3.2 summarizes supported normalization and background correction methods.

Table 4: Infinium 450k normalization and background subtraction methods.

Method	**Backend package**	**Option**	**Reference**
Normalization (`normalization.method=`)			
Normalization to internal controls	`methylumi`	`"illumina"`	[11, 8]
SWAN	`minfi`	`"swan"`	[4, 9]
"Data-driven" normalization	`wateRmelon`	`"wm.*"` [a]	[6]
Functional normalization	`minfi`	`minfi.funnorm`	[21, 9]
BMIQ	`BMIQ`	`"bmiq"` [b]	[10]
Background correction (`normalization.background.method=`)			
NOOB	`methylumi`	`"methylumi.noob"`	[11, 8]
Lumi	`methylumi`	`"methylumi.lumi"`	[11, 8]

[a] * is a placeholder for the `wateRmelon` normalization methods, e.g. "dasen", "nanet" etc.
[b] unlike other normalization and background subtraction methods BMIQ is applied to the summarized methylation calls, so it can be used even if raw intensity data were not loaded, like in case of `GEO` or `data.dir` data types

The preprocessing report includes a number of diagnostic plots that visualize the effects of the normalization procedure: density plots of the methylation values prior and subsequent to the normalization procedure, histograms of the per-data point beta-value differences, and others.

Note that the data contained within a loaded `RnBeadRawSet` object can be normalized without generating a report:

```
> rnb.set.norm <- rnb.execute.normalization(rnb.set.unfiltered, method="illumina",
+     bgcorr.method="methylumi.noob")
```

4.4 Covariate Inference

This optional module facilitates the inference of additional dataset covariates. For instance, surrogate variables can be estimated to detect and adjust for possible hidden confounders in the dataset. Additionally, the effects of cell type contributions can be assessed.

Note that this module is disabled by default. You can enable it for the `rnb.run.analysis` command using the following option setting:

```
> rnb.options(inference=TRUE)
```

The annotation inference module can be executed individually using its dedicated function:

```
> rnb.set <- rnb.run.inference(rnb.set, report.dir)$rnb.set
```

4.4.1 Surrogate Variable Analysis

Surrogate Variable Analysis (SVA) [12] has become a popular tool to adjust for hidden confounders in expression microarray analysis and can also be employed in the DNA methylation setting. RnBeads uses the sva package [13] to infer surrogate variables which can be related to other sample annotation and can be used to adjust differential methylation analysis using the limma method. In order to compute SVs for given target variables, to add the covariates to an RnBSet and to generate a corresponding report, consider the following example:

```
> rnb.options(
+         inference.targets.sva=c("Sample_Group"),
+         inference.sva.num.method="be"
+ )
> rnb.set.new <- rnb.run.inference(rnb.set, report.dir)$rnb.set
```

If you just want to conduct the analysis without generating a report, you can compute the SVs and add the information to an RnBSet the following way:

```
> sva.obj <- rnb.execute.sva(rnb.set, cmp.cols="Sample_Group", numSVmethod="be")
> rnb.set <- set.covariates.sva(rnb.set, sva.obj)
```

Section 6.5.4 contains a recipe on how these covariates can be used for adjustment during differential methylation detection.

4.4.2 Inference of Cell Type Contributions

RnBeads is capable of tackling cell type heterogeneity of the analyzed samples. In case methylomes of reference cell types are available, the annotation inference module will try to estimate the relative contributions of underlying cell types in target data. RnBeads applies the method by Houseman et al. [14]. The following paragraphs give details on how this algorithm is incorporated in RnBeads. Section 6.6 describes the usage of an alternative reference-free approach.

Methylation profiles of reference cell types should be processed together with the target data. Assignment of the samples to cell types is done via a special column of the sample annotation file set by the inference.reference.methylome.column. The column should map each reference sample to one of the cell types, while the target samples should be assigned with missing (NA) values. As an example, let the reference methylome column be called "CellType". After setting the option:

```
> rnb.options(inference.reference.methylome.column="CellType")
```

RnBeads will automatically activate the cell type section of the annotation inference module. The estimation procedure from [14] consists of two major steps. First, the marker model is fit to the reference methylome data, to identify CpG sites, methylation level of which is associated with the cell type. To limit computational load at this step, RnBeads by default uses only 50,000 CpGs with highest variance across all samples. Alternative number can be set via the inference.max.cell.type.markers option. The final number of selected cell type markers is controlled by inference.top.cell.type.markers which is 500 by default. The report visualizes the selection by plotting the F-statitistics of the marker model fit. In the second step, the marker model coefficients for selected markers CpGs are used for estimation of cell type contributions by solving a (constrained) quadratic optimization problem for corresponding vectors of each of

the target methylomes a.k.a. the projection. The inferred contributions will be plotted in the annotation inference report and saved in the RnBSet object and later used as covariates in the association analysis to adjust for the variation in cell type composition. The latter option is only available in case "limma" method is used for the testing differential methylation.

Alternatively, the reference-based analysis of cell type contributions can be performed without report generation:

```
> ct.obj<-rnb.execute.ct.estimation(rnb.set, cell.type.column="CellType",
+     test.max.markers=10000, top.markers=500)
```

The returned S3 object of class CellTypeInferenceResult contains all information about the estimation procedure and can be supplied as an argument to functions rnb.plot.marker.fstat and rnb.plot.ct.heatmap. Cell type contributions are available by direct slot access:

```
> ct.obj$contributions
```

for raw coefficients of the projection model,

```
> ct.obj$contributions.constr
```

to get constrained (non-negative) coefficients which resemble cell type counts and sum up to a small value < 1 .

4.4.3 Age Prediction

In addition to the inference possibilities mentioned before, RnBeads is capable of predicting human age from DNA methylation data. The prediction was initially performed by building an elastic net regression model on the methylation values of the CpG sites. Performance was evaluated by testing the prediction on several independent test data sets.

Basically, the age prediction section of RnBeads is activated by setting:

```
> rnb.options(inference.age.prediction=TRUE)
```

Or by performing the command

```
> rnb.execute.age.prediction(rnb.set)
```

The difference between the first and the second option is that the first one creates a report, whereas the second modifies the rnb.set object by adding additional columns predicted_ages and age_increase. The first one corresponds to the predicted age by the prediction algorithm and the second to the difference between predicted and annotated age.

Then, the age prediction section is able to run in two different modes. The default mode runs the prediction by loading a predefined predictor in the form of a CSV-file, which then is used to predict human age from DNA methylation data. More specifically, the predefined predictor consists of coefficients of an elastic net regression run for 761 CpG sites used in the final age prediction. As a second possibility, the user can set the option inference.age.prediction.predictor to a path to a predictor that was previously trained in a RnBeads run. For instance, this predictor could be trained by running the second mode of the age prediction, which is the training mode. Training is performed by either one of these commands:

```
> rnb.execute.training(rnb.set,predictor.path)
> rnb.options(inference.age.prediction.training=TRUE)
```

In the first case, a new predictor is trained based on the methylation information in the `rnb.set` object and written as a CSV-file at the given `predictor.path`, which has to be a full path. The second option requires running `rnb.run.inference` to create a new predictor that then is written into the `RnBeads` data folder and sets the option `inference.age.prediction.predictor` to the corresponding one.

Additionally, by setting the option `inference.age.prediction.cv`, a 10-fold cross-validation is performed to serve as an accuracy measure for the training functionality. Setting the option `inference.age.column` enables `RnBeads` to read the corresponding age annotation column of the data set, which enables the creation of evaluation plots. These include scatterplots of the deviation between annotated and predicted age as well as density plots for the differences between predicted and annotated ages. On default, `inference.age.column` is set to `age`.

4.4.4 Immune Cell Content Estimation

The LUMP algorithm for estimating immune cell content was developed by Dvir Aran, Marina Sirota and Atul J. Buttea for Infinium 450k data [22]. It is implemented in `RnBeads` for all supported platforms in the genome assembly hg19: Infinium 27k, Infinium 450k, MethylationEPIC and sequencing-based data.

By default, immune cell content estimation is attempted when the Covariate Inference module is run. Upon successful completion of the LUMP algorithm, the sample annotation table receives a column named `Immune Cell Content (LUMP)` that lists the estimated immune cell fractions for the samples. If such a column already exists, its contents will be overwritten. LUMP can be disabled using the following option:

```
> rnb.options(inference.immune.cells=FALSE)
```

The dedicated function to obtain LUMP estimates expects a methylation dataset as a parameter:

```
> immune.content <- rnb.execute.lump(rnb.set)
```

If none of the CpGs used in LUMP is measured in the dataset of interest, the return value is `NULL`. Otherwise, the function returns a numeric vector with immune cell fractions for every sample, that is, values between 0 and 1. The vector also has an integer attribute named `"sites"` that stores the number of CpGs or probes used in estimating immume cell proportions for the specific dataset.

4.5 Exploratory Analysis

You guessed it: the command for running the exploratory analysis module is

```
> rnb.run.exploratory(rnb.set, report.dir)
```

Note that depending on the options settings and types of enabled plots, this might take a while. It creates a report that lets you inspect the association of various sample traits with each other, with quality control probes (if available), as well as with sparse representations of the methylation profiles. This module also contains sections dedicated to inspecting methylation profiles of sample groups, as well as unsupervised learning techniques to cluster samples. The corresponding report file is `exploratory.html`.

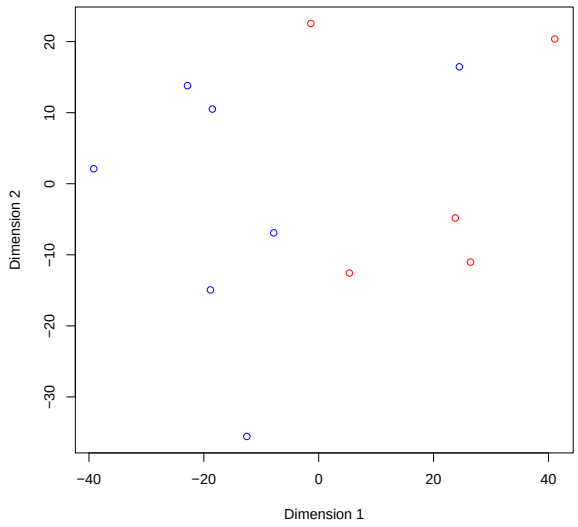

Figure 6: Multidimensional scaling applied to the example dataset. Points are colored according to cell type: Blue: iPSCs, Red: ESCs

4.5.1 Low-dimensional Representation and Batch Effects

The following series of commands calculates the sparse representations, i.e. by Principal Component Analysis (PCA) and Multidimensional Scaling (MDS) for single CpG sites and promoters and plots them:

```
> dred.sites <- rnb.execute.dreduction(rnb.set)
> dred.promoters <- rnb.execute.dreduction(rnb.set, target="promoters")
> dred <- list(sites=dred.sites, promoters=dred.promoters)
> sample.colors <- ifelse(pheno(rnb.set)$Sample_Group=="hESC", "red", "blue")
> plot(dred[[1]]$mds$euclidean, col=sample.colors,
+     xlab="Dimension 1", ylab="Dimension 2")
```

The resulting plot can be seen in Figure 6. Subsequently, let us investigate associations of sample annotations:

```
> assoc <- rnb.execute.batcheffects(rnb.set, pcoordinates=dred)
> str(assoc)
```

The initiated object contains slots for the association of sample annotations with each other (`assoc$traits`) and with the principal components calculated in the previous step (`assoc$pc`: a list containing elements for each region type; here: `sites` and `promoters`). Each of these slots summarizes whether a test could be conducted (`failures`), which kind of test was applied (`tests`), the resulting p-value (`pvalues`) and, if calculated, a table of `correlations`. The `batch.dreduction.columns` option determines which sample annotations are processed (see Section 6.2). By default, these are all columns in the table `pheno(rnb.set)` which have a sufficient number of samples per group and not too many resulting groups (controlled by the `min.group.size` and `max.group.count` parameters). However, you can restrict the analysis to certain columns via commands like:

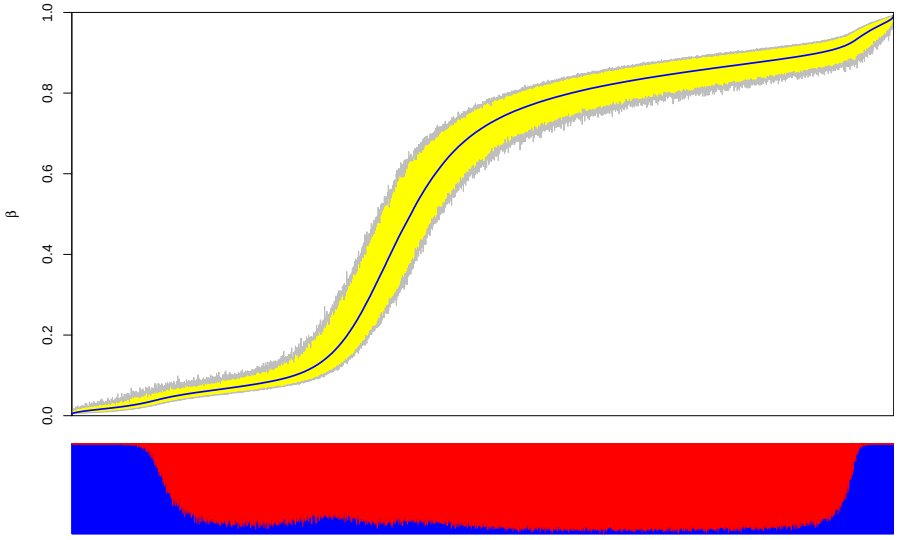

Figure 7: Deviation plot of the example dataset. The blue line corresponds to the median methylation value among all samples, the yellow bands to the 5-th and 95-th percentiles. Coloring below the horizontal axis indicates probe types (Blue: Infinium Design Type I, Red: Infinium Design Type II).

```
> rnb.options(exploratory.columns=c("Sample_Group", "Passage_No"))
```

Next, one can compute associations of the principal components with the Infinium control probes (if that information is available):

```
> assoc.qc <- rnb.execute.batch.qc(rnb.set, pcoordinates=dred)
```

A good example for useful visualizations is the so-called deviation plots. These plots show the median β values and their percentiles in a "snake-like" curve. The plot can be annotated with coloring according to site or region features. The code snippet below creates Figure 7 which illustrates the beta value distributions according to probe design type:

```
> probe.types <- as.character(annotation(rnb.set)[, "Design"])
> pdf(file="deviationPlot_probeType.pdf", width=11)
> deviation.plot.beta(meth(rnb.set), probe.types,
+     c.legend=c("I"="blue", "II"="red"))
> dev.off()
```

Note that creating such a plot requires a few minutes.

4.5.2 Clustering

Use the following command to apply all combinations of clustering parameter settings and conduct the corresponding clusterings on the DNA methylation levels of individual CpGs as well as promoter regions:

```
> clusterings.sites <- rnb.execute.clustering(rnb.set, region.type="sites")
> clusterings.promoters <-
+     rnb.execute.clustering(rnb.set, region.type="promoters")
```

The values returned by the commands above are lists of clustering results. Each element in the list is an object of type `RnBeadClustering`. Currently, `RnBeads` performs agglomerative hierarchical clustering in conjunction with three distance metrics and three linkage methods. The distance metrics are correlation-based ($1 - \rho(x, y)$), Manhattan and Euclidean distances. The agglomeration methods are average, complete and median-based linkages. Here's a little recipe which lets you create a clustered heatmap of the methylation values according to the first clustering of the first 100 promoters (Figure 8) using the following commands:

```
> # Get the methylation values
> X <- meth(rnb.set, type="promoters")[1:100, ]
> # Convert the clustering result to a dendrogram
> # Index 7 holds euclidean distance, average linkage clustering
> cresult <- clusterings.promoters[[7]]@result
> attr(cresult, "class") <- "hclust"
> cresult <- as.dendrogram(cresult)
> # Save the heatmap as pdf file
> pdf(file="promoter_heatmap.pdf")
> heatmap.2(X, Rowv=TRUE, Colv=cresult, dendrogram="both",
+     scale="none", trace="none")
> dev.off()
```

4.6 Differential Methylation Analysis

Differential methylation analysis in `RnBeads` can be conducted on the site level as well as on the genomic region level. Genomic regions are inferred from the annotation data (see Section 6.2). As seen before, a comprehensive analysis can be obtained by the corresponding `rnb.run.*.` command:

```
> rnb.run.differential(rnb.set, report.dir)
```

Note that this analysis module requires more runtime than other modules. Specifically computing differential methylation can be achieved using the following steps:

```
> # Specify the sample annotation table columns for which
> # differential methylation is to be computed
> cmp.cols <- "Sample_Group"
> # Specify the region types
> reg.types <- c("genes", "promoters")
> # Conduct the analysis
> diffmeth <- rnb.execute.computeDiffMeth(rnb.set, cmp.cols,
+     region.types=reg.types)
```

Differential methylation for each selected annotation column is performed in a one-vs-all manner, i.e. samples in each group are compared to all samples not in the group. This is done for each group and annotation column. Depending on the number of comparisons, this can take a while. The `differential.comparison.columns.all.pairwise` option facilitates conducting all pairwise group comparisons on the specified columns. Handle this option with care, as the number of comparisons can explode very quickly if the number of groups specified in an annotation column is large. In order to define custom comparisons between groups you can edit the sample annotation sheet before data loading. For instance, if you want to carry out a specific

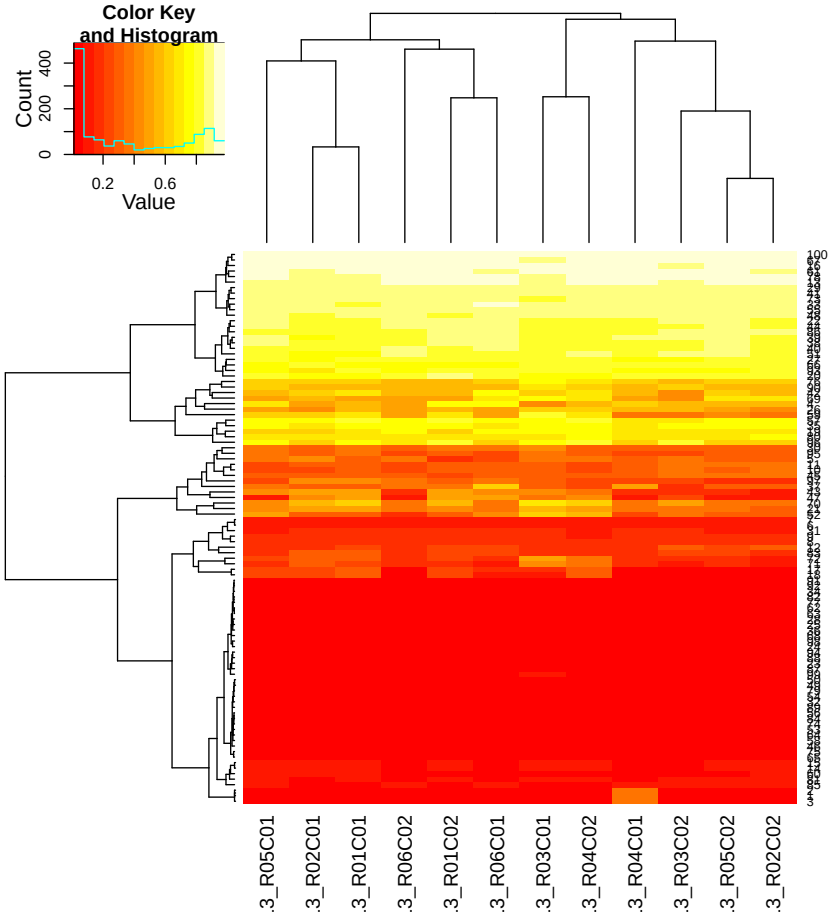

Figure 8: Clustered heatmap of 100 promoters.

comparison in addition to the already defined comparisons, consider adding a column containing the labels "Group1" and "Group2" for specific samples and "NA" for the others. Alternatively, you can use addPheno() to add sample annotation to the RnBSet object as seen before.

Let us inspect the resulting RnBDiffMeth object (an S4 class) more closely.

```
> str(diffmeth)
```

It contains tables storing the differential methylation information for each comparison for both site and region level. You can see the comparisons that were conducted using get.comparisons(diffmeth). The regions that were compared can be seen with get.region.types(diffmeth). Information on which groups have been compared in each comparison can be obtained by get.comparison.grouplabels(diffmeth). For a given comparison and region type you can get get a table containing information on differential methylation using get.table():

```
> comparison <- get.comparisons(diffmeth)[1]
> tab.sites <- get.table(diffmeth, comparison, "sites", return.data.frame=TRUE)
> str(tab.sites)
> tab.promoters <- get.table(diffmeth, comparison, "promoters",
+         return.data.frame=TRUE)
> str(tab.promoters)
```

Below you can find the most important columns of the differential methylation tables on the site level
(tab.sites from the code above):

mean.g1, mean.g2: mean methylation in each of the two groups

mean.diff: difference in methylation means between the two groups: mean.g1-mean.g2. In case of paired analysis, it is the mean of the pairwise differences.

mean.quot.log2: log2 of the quotient in methylation: $\log_2 \frac{\texttt{mean.g1}+\varepsilon}{\texttt{mean.g2}+\varepsilon}$, where $\varepsilon = 0.01$. In case of paired analysis, it is the mean of the pairwise quotients

diffmeth.p.val: p-value obtained from a two-sided Welch t-test or alternatively from linear models employed in the limma package (which type of p-value is computed is specified in the differential.site.test.method option). In case of paired analysis, the paired Student's t-test is applied.

max.g1, max.g2: maximum methylation level in group 1 and 2 respectively

min.g1, min.g2: minimum methylation level in group 1 and 2 respectively

sd.g1, sd.g2: standard deviation of methylation levels

min.diff: minimum of 0 and the smallest pairwise difference between samples of the two groups

diffmeth.p.adj.fdr: FDR adjusted p-values of all the sites

combinedRank: mean.diff, mean.quot and t.test.p.val are ranked for all sites. This aggregates them using the maximum, i.e. worst rank of a probe among the three measures

and the region level (e.g. tab.promoters):

`mean.mean.g1, mean.mean.g2:` mean of mean methylation levels for group 1 and 2 across all probes in a region

`mean.mean.diff:` mean difference in means across all sites in a region

`mean.mean.quot.log2:` log2 of the mean quotient in means across all sites in a region

`comb.p.val:` combined p-value using a generalization of Fisher's method [3]

`comb.p.adj.fdr` FDR adjusted combined p-value

`num.sites:` number of sites associated with the region

`combinedRank:` `mean.mean.diff`, `mean.mean.quot.log2` and `comb.p.val` are ranked for all regions. This column aggregates them using the maximum, i.e. worst rank of a probe among the three measures

For explanations on additional columns and more details be referred to the corresponding report (`differential.html`) and the help pages (`?computeDiffTab.site,?computeDiffTab.region`). In an `RnBeads` analysis, we recommend to conduct differential methylation analysis based on the combined ranking score. It combines absolute and relative effect sizes as well as p-values from statistical modeling into a single score. The lower the combined rank, the higher is the degree of differential methylation. I.e. when you sort the list according to their combined rank, the best ranking sites will be at the top of the list, and when you work your way towards the bottom of the list, false positives become more likely.

```
> dmps <- tab.sites[order(tab.sites[, "combinedRank"]), ]
> summary(dmps[1:100, ])
> summary(dmps[1:1000, ])
```

However, traditional, cutoff-based approaches are also possible:

```
> which.diffmeth <- abs(dmps[, "mean.diff"])>0.2 & dmps$diffmeth.p.adj.fdr<0.05
> dmps.cutoff <- dmps[which.diffmeth, ]
```

Note that you can specify how p-values are computed by using `RnBeads`' `'differential.site.test.method'` option. Currently supported are linear modeling using the `limma` package and regular t-tests.

The following code creates a volcano plot comparing effect size and significance for all promoter regions (see Figure 9):

```
> dmrs <- get.table(diffmeth, comparison, "promoters")
> plot(dmrs[, "mean.mean.diff"], -log10(dmrs[, "comb.p.val"]),
+     xlab="mean difference", ylab="-log10(combined p-value)", col="blue")
```

4.6.1 Differential Variability Analysis

In addition to comparing mean methylation levels inside groups with each other, the variances inside each group can be used to detect differences among them. For this purpose, `RnBeads` employs a similar approach to the one used for differential methylation. To conduct differential variability (together with differential methylation) analysis, execute the following lines of code:

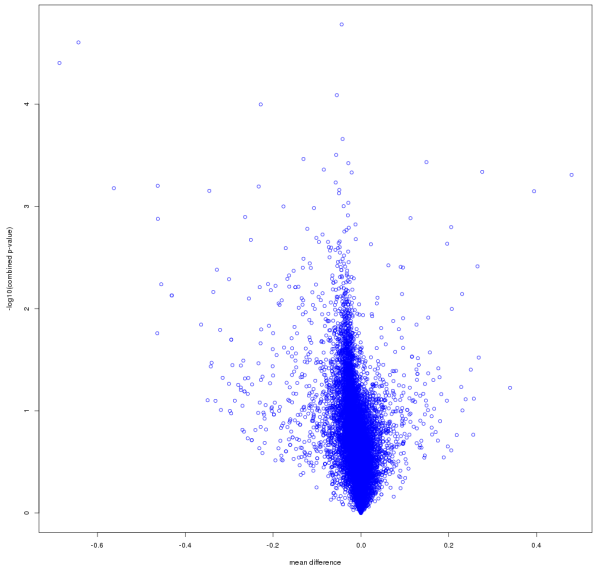

Figure 9: Volcano plots for promoter differential methylation comparing hESCs against hiPSCs.

```
> rnb.options("differential.variability"=TRUE)
> rnb.run.differential(rnb.set,report.dir)
```

To conduct solely differential variability analysis (without report generation), one can call:

```
> cmp.cols <- "Sample_Group"
> reg.types <- c("genes","promoters")
> diff.var <- rnb.execute.diffVar(rnb.set,cmp.cols,region.types=reg.types)
> comparison <- get.comparisons(diff.var)[1]
> tab.sites <- get.table(diff.var,comparison,"sites",return.data.frame=TRUE)
> tab.genes <- get.table(diff.var,comparison,"genes",return.data.frame=TRUE)
```

After this operation, the tables contained in the `diff.var` object (of class `RnBDiffMeth`) possess different columns in comparison to differential methylation. Those columns are (from `tab.sites` above):

`var.g1, var.g2:` variances in each of the two groups

`var.diff:` differences between the variances found is the two groups: `var.g1-var.g2`.

`var.log.ratio:` log2 of the quotient of variances: $\log_2 \frac{\texttt{var.g1}+\epsilon}{\texttt{var.g2}+\epsilon}$, with $\epsilon = 0.01$.

`diffVar.p.val:` p-value resulting from employing the differential variability test specified in the option differential.variability.method. Those are either 'diffVar' from the 'missMethyl' package or the iEVORA algorithm.

`diffVar.p.adj.fdr:` FDR adjustment of the above p-value.

`combinedRank.var:` `var.diff`, `var.log.ratio` and `diffVar.p.val` for all sites. The rank is computed as the maximum, i.e. worst rank among these values.

For the table `tab.genes` (i.e. the region level), the columns are:

`mean.var.g1, mean.var.g2:` mean of variances for both groups across all sites in the region.

`mean.var.diff:` mean difference between the probe variances in the two groups.

`mean.var.log.ratio:` mean difference between the probe quotients in the two groups.

`comb.p.val.var:` combined p-value for the region, combined as for differential methylation.

`comb.p.adj.var.fdr:` FDR adjustment of the above p-value.

`combinedRank.var:` `mean.var.diff, mean.var.log.ratio` and `comb.p.val.var` for all regions. The rank is computed as the worst rank among those.

Similar to differential methylation, the differential variability module adds diagnostic plots to the `RnBeads` report in `differential.html`. Enrichment analysis is performed analogously to differential methylation (see below).

4.6.2 Paired Analysis

Upon specifying the corresponding options, `RnBeads` can also take into account sample pairing information when computing p-values. Suppose you have a sample annotation table like this one[5]:

sample	individual	diseaseState
sample_1	Bruce	normal
sample_2	Bruce	tumor
sample_3	Clark	normal
sample_4	Clark	tumor
sample_5	Peter	normal
sample_6	Peter	tumor

Further suppose, you want to compare tumor vs normal but with the pairing information by the patient/individual. Then you would apply the following option setting before running the analysis:

```
> rnb.options("differential.comparison.columns"=c("diseaseState"),
+     "columns.pairing"=c("diseaseState"="individual"))
```

Note, that since the dataset used in this vignette has no column called `diseaseState`, the option settings above will cause some the examples in the next sections to fail. So, revert the option setting before continuing:

```
> rnb.options("differential.comparison.columns"=NULL,
+     "columns.pairing"=NULL)
```

4.6.3 Adjusting for Covariates in the Differential Analysis

When the `'differential.site.test.method'` option is set to `'limma'` (default), you can specify columns of potential confounding factors that should be taken into account when computing the site specific p-values. Again, setting the corresponding option prior to analysis is the key here:

[5] Note: while we use the first names of super-heroes in this table for instructive purposes, you should never reveal patient names that have not been pseudonymized in your analysis

```
> rnb.options("covariate.adjustment.columns"=c("Passage_No"))
> diffmeth.adj <- rnb.execute.computeDiffMeth(rnb.set, cmp.cols,
+     region.types=reg.types)
```

4.6.4 Enrichment Analysis of Differentially Methylated Regions

RnBeads allows you to characterize differentially methylated regions by computing enrichments for Gene Ontology (GO) [16] terms and for custom genomic annotations using the LOLA tool [17]. You can active these analyses for your RnBeads runs using the corresponding options:

```
> rnb.options("differential.enrichment.go"=TRUE)
> rnb.options("differential.enrichment.lola"=TRUE)
```

If you want to manually extract the enrichment results, you can use RnBeads' build-in functions. To compute GO enrichments from the computed differential methylation results you can use the following commands (takes a while because an online GO database is queried for each comparison, region type and for different differential methylation rank cutoffs):

```
> enrich.go <- performGoEnrichment.diffMeth(rnb.set, diffmeth, verbose=TRUE)
```

The result is a nested list storing enrichment tables for each comparison, ontology, differential methylation rank cutoff and direction of differential methylation. Each element in this nested list corresponds to a result computed using the GOStats package [18]. It contains enrichment odds ratios and p-values. For instance, let us inspect the enrichment of the 500 highest ranking hypomethylated promoters (in ESCs compared to iPSCs) for biological process (BP) ontology terms:

```
> enrich.table.go <-
+     enrich.go[["region"]][[comparison]][["BP"]][["promoters"]][["rankCut_500"]][["hypo"]]
> class(enrich.table.go)
> summary(enrich.table.go)
```

In addition, RnBeads supports enrichment analyses of differential methylation results using the LOLA tool [17]. This tool facilitates enrichment analyses of arbitrary genomic and epigenomic annotations using Fisher's exact test. To run a LOLA analysis manually, you first need to obtain a corresponding database. For instance, to download the core database from the tool website to a temporary destination, execute the following code:

```
> lolaDest <- tempfile()
> dir.create(lolaDest)
> lolaDirs <- downloadLolaDbs(lolaDest, dbs="LOLACore")
```

This database currently contains annotations for transcription factor binding sites and motifs, DNase accessible regions and genomic features. For more detailed information on the LOLA tool, available precomputed databases and instructions on how to assemble your own database, please be referred to the LOLA documentation[6]. You can now compute the enrichment of differentially methylated regions to reference region sets in the database:

```
> enrich.lola <-
+     performLolaEnrichment.diffMeth(rnb.set, diffmeth, lolaDirs[["hg19"]])
```

[6] http://databio.org/lola/

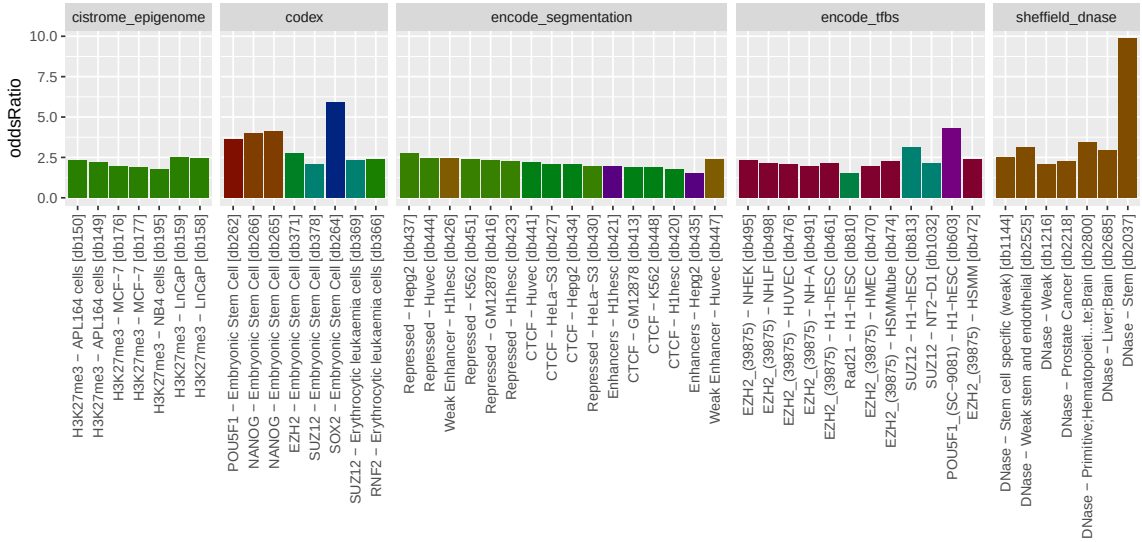

Figure 10: Bar plot of odds ratios for `LOLA` region sets significantly enriched in promoters hypermethylated in ESCs compared to iPSCs. The bars are grouped by category in the `LOLA` core database. The coloring indicates different features (chromatin marks, states, transcription factors, etc.).

The result is a structured object containing enrichment tables for different comparisons, region types and rank cutoffs as well as information on the reference database. These objects are best explored using `RnBeads`' utility functions. Here we plot the enrichment results of the 500 highest ranking hypermethylated promoter regions in ESCs compared to iPSCs.

```
> enrich.table.lola <- enrich.lola$region[[comparison]][["promoters"]]
> enrich.table.lola <-
+     enrich.table.lola[enrich.table.lola$userSet=="rankCut_500_hyper",]
> lolaBarPlot(enrich.lola$lolaDb, enrich.table.lola, scoreCol="oddsRatio",
+     orderCol="maxRnk", pvalCut=0.05)
> lolaBoxPlotPerTarget(enrich.lola$lolaDb, enrich.table.lola, scoreCol="pValueLog",
+     orderCol="maxRnk", pvalCut=0.05)
```

Figures 10 and 11 show the results as bar and boxplots. Interpreting the results, we find patterns associated with pluripotency hypermethylated in ESCs compared to iPSCs: enriched region sets are associated with reprogramming factors (POU5F1, SOX2, NANOG) and accessible chromatin (DNase) in stem cells. Additionally, signatures of gene repression (repressed state in the ENCODE segmentation and the associated H3K27me3 histone mark) are present in more differentiated cells.

4.7 Tracks and Tables

By default, methylation tracks are exported in `bed` format. They contain genomic locations and beta values for each site in the filtered `RnBSet` for each sample in the dataset. Additionally, track hubs[7] for the UCSC or Ensembl Genome browsers can be generated. For using the track hubs

[7] http://genome.ucsc.edu/goldenPath/help/hgTrackHubHelp.html

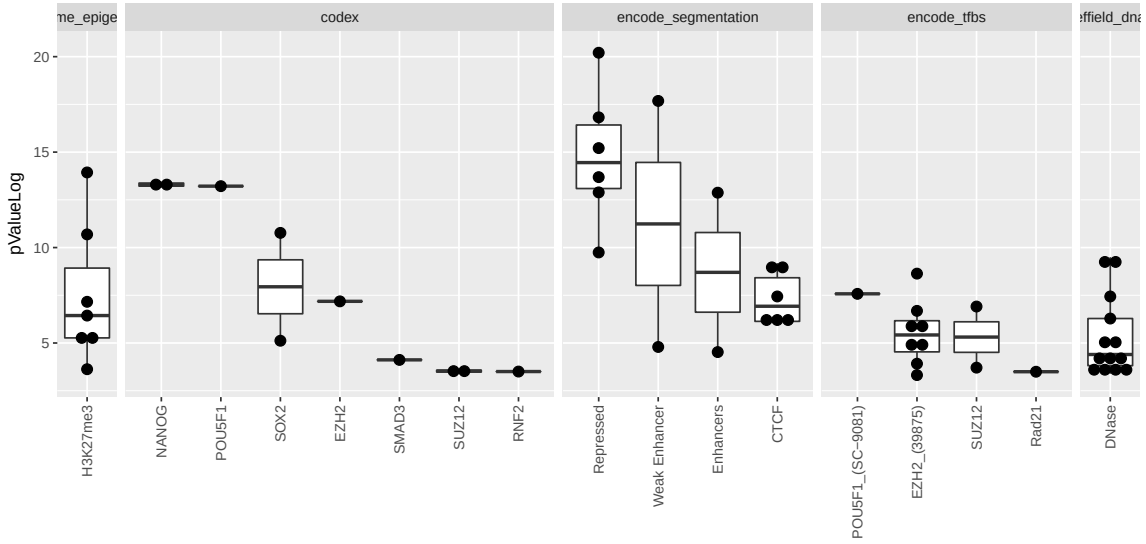

Figure 11: Box plot of log p-values for `LOLA` region sets significantly enriched in promoters hypermethylated in ESCs compared to iPSCs. Each bar summarizes a given feature across experiments in the `LOLA` core database.

in conjunction with a genome browser you will need a server that can be reached via a URL to provide the data. The command to convert the `RnBSet` to the desired output and to generate the corresponding report is

```
> rnb.options(export.to.csv=TRUE)
> rnb.run.tnt(rnb.set, report.dir)
```

The report contains specific instructions on how to visualize the data in the genome browser. Additionally, by default the Tracks and Tables report will also contain a sample summary table. In this particular example, prior to starting the module, we enable the option `export.to.csv`. Hence, the generated report contains also single-site methylation value matrices, exported to comma-separated value files.

5 Analyzing Bisulfite Sequencing Data

`RnBeads` also facilitates the analysis of bisulfite data obtained from experiments such as whole genome, reduced representation or locus specific bisulfite sequencing. The general structure for analyzing bisulfite sequencing data with `RnBeads` is the same as for Infinium data. However, the devil is in the details. Here are the key differences:

- In addition to human methylation data, `RnBeads` also supports mouse and rat methylome analysis through its companion packages (`RnBeads.mm9`,`RnBeads.mm10`,`RnBeads.rn5`).
- Loading is performed with `bed` file input.
- No normalization will be conducted during preprocessing.
- There are no control probes in bisulfite sequencing that are incorporated in the QC and batch effect reports. Instead, coverage information is used to assess experimental quality.

- Certain types of analysis would take significantly longer when using bisulfite sequencing data and are therefore disabled in standard analysis. Among them are the greedycut algorithm for filtering unreliable samples and sites and deviation plots as obtained in the profiles module for Infinium datasets.

5.1 Data Loading

RnBeads expects sequencing data to be bed-formatted or have a similar coordinate file format. The data.source argument of the rnb.run.import() and rnb.run.analysis() functions for analyzing bisulfite sequencing data requires a vector or list of length 1, 2 or 3 containing:

1. the directory where the methylation bed files are located;
2. the sample annotation sheet
3. the index of the column of the sample annotation sheet that contains the names or full paths to the bed files

In case only the first element of data.source is given, RnBeads expects the bed files and sample annotation to be located in the same directory. The sample annotation file should contain token "sample" in the file name. The sample annotation should contain the names of the bed files relative to the specified directory. There should be exactly one bed file for each sample in the directory. One of the first two elements have to be present. In case only the sample sheet is specified, one column should be giving full absolute paths of the BED-like files with sequencing information. If both elements (1) and (2) are specified, the files should reside in the directory, specified as element (1). If the third element is missing, RnBeads attempts to find the file name-containing column automatically.

The bed files are expected to contain methylation and localization information for each individual site (CpG), where information includes the chromosome, coordinates (start and end coordinates) and genomic strand. Methylation information should contain any two numbers that indicate how many methylation events (#M – number of cytosines sequenced for a particular position), unmethylation events (#U – number of thymines sequenced) and total events (#T = #U + #M) were measured for a particular location. This can be specified in any of the following forms:

- any two measurements among #M, #U and #T;
- #T and the methylation percentage: $\frac{\#\mathrm{M}}{\#\mathrm{U}+\#\mathrm{M}} \times 100$

The package option 'import.bed.columns' controls how RnBeads interprets the bed files. Since there is no clear standard available for this type of methylation data, specification can be tricky. Examples for possible uncertainties are: Are the coordinates 0-based or 1-based? What chromosome and strand notation is used? Are the CpGs listed for both strands or just one? Are the coordinates on the reverse strand shifted? Is the methylation percentage specified in the interval [0,100] or [0,1]? Many more questions may arise. Hence, there is no guarantee that RnBeads correctly loads each and every format. Probably the most convenient and safe way to load bisulfite sequencing data is to use the format presets which are tailored to particular tools and pipelines. These presets are specified in the package option 'import.bed.style'. Here are examples of the presets currently available:

'BisSNP' The `bed` files are assumed to have been generated by the methylation calling tool `BisSNP` [5]. A tab-separated file contains the chromosome name, start coordinate, end coordinate, methylation value in percent, the coverage, the strand, as well as additional information not taken into account by `RnBeads`. The file should contain a header line. Coordinates are 0-based, spanning the first and the last coordinate in a site (i.e. end-start = 1 for a CpG). Sites on the negative strand are shifted by +1. Here are some example lines (genome assembly `hg19`):

```
track name=file_sorted.realign.recal.cpg.filtered.sort.CG.bed type=bedDetail description="CG methylation
chr1    10496   10497   79.69   64      +       10496   10497   180,60,0        0       0
chr1    10524   10525   90.62   64      +       10524   10525   210,0,0 0       0
chr1    864802  864803  58.70   46      +       864802  864803  120,120,0       0       5
chr1    864803  864804  50.00   4       -       864803  864804  90,150,0        1       45
...
```

'EPP' The `bed` files are assumed to have the format as output files from the Epigenome Processing Pipeline developed by Fabian Müller and Christoph Bock. A tab-separated file contains: the chromosome name, start coordinate, end coordinate, methylation value and coverage as a string ('#M/#T'), methylation level scaled to the interval 0 to 1000, strand, and additional information not taken into account by `RnBeads`. The file should not contain a header line. Coordinates are 0-based, spanning the first coordinate in a site and the first coordinate outside the site (i.e. end-start = 2 for a CpG). Here are some example lines (genome assembly `mm9`):

```
chr1    3010957 3010959 '27/27' 1000    +
chr1    3010971 3010973 '10/20' 500     +
chr1    3011025 3011027 '57/70' 814     -
...
```

'bismarkCov' The `cov` files are assumed to have the format as defined by Bismark's coverage file output converted from its bedGraph output (Bismark's `bismark2bedGraph` module; see the section "Optional bedGraph output" in the Bismark User Guide). A tab-separated file contains: the chromosome name, cytosine coordinate, cytosine coordinate (again), methylation value in percent, number of methylated reads (#M) and the number of unmethylated reads (#U). The file should not contain a header line. Coordinates are 1-based. Strand information does not need to be provided, but is inferred from the coordinates: Coordinates on the negative strand specify the cytosine base position (G on the positive strand). Coordinates referring to cytosines not in CpG content are automatically discarded. Here are some example lines (genome assembly `hg19`):

```
...
chr9    73252   73252   100     1       0
chr9    73253   73253   0       0       1
chr9    73256   73256   100     1       0
chr9    73260   73260   0       0       1
chr9    73262   73262   100     1       0
chr9    73269   73269   100     1       0
...
```

'bismarkCytosine' The `bed` files are assumed to have the format as defined by Bismark's cytosine report output (Bismark's `coverage2cytosine` module; see the section "Optional genome-wide cytosine report output" in the Bismark User Guide). A tab-separated file contains: the chromosome name, cytosine coordinate, strand, number of methylated reads (#M), number of unmethylated reads (#U), and additional information not taken into account by `RnBeads`. The file should not contain a header line. Coordinates are 1-based. Coordinates on the negative strand specify the cytosine position (G on the positive strand). CpG without coverage are allowed, but not required. Here are some example lines (genome assembly `hg19`):

```
...
chr22   16050097        +       0       0       CG      CGG
chr22   16050098        -       0       0       CG      CGA
chr22   16050114        +       0       0       CG      CGG
chr22   16050115        -       0       0       CG      CGT
...
chr22   16115591        +       1       1       CG      CGC
chr22   16117938        -       0       2       CG      CGT
chr22   16122790        +       0       1       CG      CGC
...
```

'Encode' The bed files are assumed to have the format as the ones that can be downloaded from UCSC's ENCODE data portal. A tab-separated file contains: the chromosome name, start coordinate, end coordinate, some identifier, read coverage (#T), strand, start and end coordinates again (RnBeads discards this duplicated information), some color value, read coverage (#T) and the methylation percentage. The file should contain a header line. Coordinates are 0-based. Note that this file format is very similar but not identical to the 'BisSNP' one. Here are some example lines (genome assembly hg19):

```
track name="SL1815 MspIRRBS" description="HepG2_B1__GC_" visibility=2 itemRgb="On"
chr1    1000170 1000171 HepG2_B1__GC_   62      +       1000170 1000171 55,255,0    62      6
chr1    1000190 1000191 HepG2_B1__GC_   62      +       1000190 1000191 0,255,0     62      3
chr1    1000191 1000192 HepG2_B1__GC_   31      -       1000191 1000192 0,255,0     31      0
chr1    1000198 1000199 HepG2_B1__GC_   62      +       1000198 1000199 55,255,0    62      10
chr1    1000199 1000200 HepG2_B1__GC_   31      -       1000199 1000200 0,255,0     31      0
chr1    1000206 1000207 HepG2_B1__GC_   31      -       1000206 1000207 55,255,0    31      10
...
```

Note, that in all of the formats above, CpGs may be listed on both strands individually to indicate strand-specific methylation. However, by default, RnBeads will merge methylation information for both strands for each CpG.

If you have a format that you feel should be supported by a preset in RnBeads or you have trouble loading a certain format, do not hesitate to contact the RnBeads developers.

Assuming input file locations and annotation stored in the variable data.source, and report.dir specifies a valid report directory, the entire analysis pipeline can be conducted analogously to Infinium data:

```
> rnb.run.analysis(dir.reports=report.dir, data.source=data.source,
+     data.type="bs.bed.dir")
```

Modules can also be started individually. After executing rnb.run.import with "bed.dir" as data type, the resulting object is of class RnBSet. This object is very similar to objects of its child class (RnBeadSet) but does not contain Infinium-specific information.

5.2 Quality Control

Sequencing depth/coverage is currently the only metric used by RnBeads' quality control reports for bisulfite sequencing. Other quality control steps such as estimating the bisulfite conversion rate using spike-in controls, assessing the alignment rate, etc. are typically done as part of the preprocessing prior to an RnBeads analysis. Several control plots are available. For example, the *histogram of coverage*, *genomic coverage plot* (see example in Figure 12) and *violin plot*, each visualizes coverage information for individual samples. The *coverage thresholds plot* gives an overview of how many sites meet the criteria for a deep coverage in a minimal fraction of the samples in the whole dataset.

5.3 Filtering

The code snippet below shows an example sequence of filtering steps on a bisulfite sequencing dataset.

```
> # Remove CpGs on sex chromosomes
> rnb.set.filtered <- rnb.execute.sex.removal(rnb.set.unfiltered)$dataset
> # Remove sites that have an exceptionally high coverage
> rnb.set.filtered <-
+     rnb.execute.highCoverage.removal(rnb.set.filtered)$dataset
> # Remove sites containing NA for beta values
```

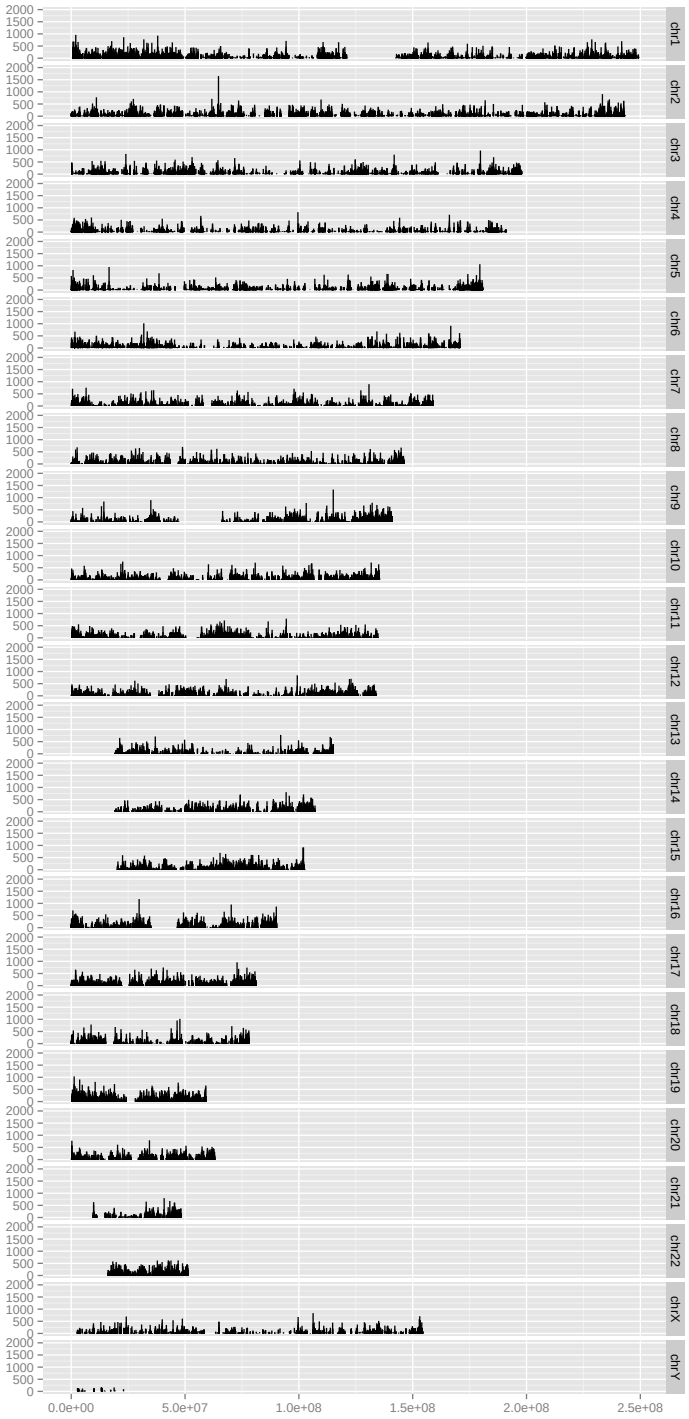

Figure 12: Sequencing coverage along the genome position

```
> rnb.set.filtered <- rnb.execute.na.removal(rnb.set.filtered)$dataset
> # Remove sites for which the beta values have low standard deviation
> rnb.set.filtered <-
+     rnb.execute.variability.removal(rnb.set.filtered, 0.005)$dataset
```

Executing Greedycut on large bisulfite sequencing datasets is not recommended because of its long running time.

6 Advanced Usage

RnBeads contains a variety of other useful methods and annotations that are used throughout the standard pipeline run, but that can be executed individually as well. We also present methods and functionality beyond the standard analysis pipeline. This section provides instructive code examples for RnBeads functions that you might consider helpful. If you are curious concerning the functions used, don't hesitate to use R's help functionality (?functionName). In addition, RnBeads's annotation data is briefly described here. For more information refer to the *RnBeads – Annotations* document.

6.1 Analysis Parameter Overview

RnBeads offers a multitude of options that let you configure your analysis. They are introduced in this section. You can obtain an overview on the option values by the following command:

```
> rnb.options()
```

If you want to inquire about a specific option, you can also use:

```
> rnb.getOption("analyze.sites")
```

To set options to specific values use:

```
> rnb.options(filtering.sex.chromosomes.removal=TRUE,
+     colors.category=rainbow(8))
```

A complete list of available options is contained in the help pages for rnb.options:

```
> ?rnb.options
```

6.2 Annotation Data

There are four companion data packages of RnBeads, named RnBeads.hg19, RnBeads.mm10, RnBeads.mm9 and RnBeads.rn5. They contain annotations for CpG sites, array probes and predefined genomic elements. These annotation tables are used throughout the analysis. User-defined annotations can be added to focus the analysis on specific genomic regions.

In this section, we show how to access the available annotation tables. Note that the first time annotations are requested (e.g., by calling one of the functions presented here), a set of tables is loaded. This inevitably leads to a slight delay. Subsequent calls to these functions are faster.

Annotations used by RnBeads (provided by separate packages listed above) can be accessed using the rnb.get.annotation() command:

```
> library(RnBeads)
> rnb.get.annotation(type="CpG") # all CpGs in the human genome
> rnb.get.annotation(type="probes450") # all Infinium 450k probes
```

By default they are provided as GRangesList objects (see the GenomicRanges package for more details). However, you can convert them to regular data frames:

```
> probes.450k <- rnb.annotation2data.frame(rnb.get.annotation(type="probes450"))
> head(probes.450k)
```

Annotation is also available on the genomic region level. Available region types are revealed by:

```
> rnb.region.types()
```

```
[1] "tiling"     "genes"      "promoters"  "cpgislands"
```

The default gene-related regions are of Ensembl genes (defined as the whole locus from transcription start site to transcription end site) and promoters (defined as the regions 1.5 kb upstream and 0.5 kb downstream of transcription start sites). Furthermore, CpG island and tiling regions (5kb windows) are supported. Region annotation tables can be accessed similarly to the probe annotation data:

```
> rnb.get.annotation("promoters",assembly="mm9")
```

Annotation of sites and regions represented by an rnb.set object can be accessed via the annotation() function:

```
> head(annotation(rnb.set, type="sites"))
> head(annotation(rnb.set, type="genes"))
```

Here we use our example RnBeadSet from the previous sections. Note that the rows in the resulting dataframe are in the same order as the rows returned by the meth() and rnb.execute.computeDiffMeth() functions. Hence you can use the annotation command in order to annotate obtained methylation or differential methylation values:

```
> aa <- annotation(rnb.set, type="promoters")
> annotated.dmrs <- data.frame(aa, dmrs)
> head(annotated.dmrs)
```

The code above uses the rnb.set object and the differential methylation table (dmrs) defined in Section 4.6.

6.2.1 Custom Annotation

Custom annotations can be included using the function rnb.set.annotation. For instance, the code below retrieves annotation data for genomic enhancers as specified by a chromatin state segmentation approach employed in the ENCODE project. We retrieve the state segmentation annotations for the H1 embryonic stem cell line from the UCSC Table Browser using the rtracklayer package. Then we restrict the resulting table to enhancer states and set the annotation for RnBeads:

```
> # Retrieve the chromHMM state segmentation from UCSC
> library(rtracklayer)
> mySession <- browserSession()
> genome(mySession) <- "hg19"
```

```
> tab.chromHMM.h1 <- getTable(ucscTableQuery(mySession,
+     track="wgEncodeBroadHmm", table="wgEncodeBroadHmmH1hescHMM"))
> # Filter for enhancer states
> tab.enhancers <- tab.chromHMM.h1[grep("Enhancer", tab.chromHMM.h1$name), ]
> # Select the interesting parts of the table and rename columns
> tab.enhancers <- tab.enhancers[, c("chrom", "chromStart", "chromEnd", "name")]
> colnames(tab.enhancers) <- c("Chromosome", "Start", "End", "name")
> # Create RnBeads annotation by providing a data.frame
> rnb.set.annotation("enhancersH1EscChromHMM", tab.enhancers, assembly="hg19")
> # Set the options to include the enhancer annotation
> rnb.options(region.types=c(rnb.getOption("region.types"),
+     "enhancersH1EscChromHMM"))
> # Parse the input again, this time with the enhancer annotation added
> rnb.set.enh <-
+     rnb.execute.import(data.source=data.source, data.type="idat.dir")
> rnb.set.enh
> # Annotation and methylation levels of enhancer regions in this object
> annot.enh <- annotation(rnb.set.enh, "enhancersH1EscChromHMM")
> head(annot.enh)
> meth.enh <- meth(rnb.set.enh, "enhancersH1EscChromHMM")
> head(meth.enh)
```

Note that the included genomic regions remain available to `RnBeads` in the current R session only. If you later want to reuse custom annotation data, use the `rnb.save.annotation()` and `rnb.load.annotation()` functions:

```
> annotation.filename <- file.path(analysis.dir,
+     "annotation_hg19_enhancersH1EscChromHMM.RData")
> # Save the enhancer annotation to disk
> rnb.save.annotation(annotation.filename, "enhancersH1EscChromHMM",
+     assembly="hg19")
> # Load the enhancer annotation as a duplicate
> rnb.load.annotation(annotation.filename, "enhancersH1EscChromHMM_duplicate")
> # Check that the annotation has been successfully loaded
> rnb.region.types()
```

On the `RnBeads` website[8], we also provide a resource of region sets that are of general use. These include tiling regions of various window sizes, alternative gene models, variably methylated regions and putative regulatory regions derived from epigenome segmentation data. The code below shows how tiling regions of size 1kb can be loaded.

```
> rnb.load.annotation.from.db("tiling1kb", assembly="hg19")
> rnb.region.types()
```

6.3 Parallel Processing

If you run `RnBeads` on a computational environment that supports parallel processing (e.g. a computer using multiple cores), enabling parallel computation can speed up some of the analysis

[8] http://rnbeads.mpi-inf.mpg.de/regions.html

modules quite significantly. Particularly the exploratory analysis and differential methylation modules profit from this as they operate on large tables and generate a large number of plots. Prior to any analysis, parallel processing can be enabled by the following command:

```
> logger.start(fname=NA)
> parallel.isEnabled()
```

```
[1] FALSE
```

```
> num.cores <- 2
> parallel.setup(num.cores)
```

```
2026-10-13 20:01:37     2.2  STATUS STARTED Setting up Multicore
2026-10-13 20:01:37     2.2    INFO     Using 2 cores
2026-10-13 20:01:37     2.2  STATUS COMPLETED Setting up Multicore
```

```
> parallel.isEnabled()
```

```
[1] TRUE
```

where `num.cores` specifies the number of compute cores you would like to use. If successful, `parallel.getNumWorkers()` returns the number of cores used:

```
> if (parallel.isEnabled()) parallel.getNumWorkers()
```

```
[1] 2
```

You can disable parallel processing again using

```
> parallel.teardown()
```

Note, that it is good programming practice to always disable parallel processing again, after the computation is done.

6.4 Extra-Large Matrices, RAM and Disk Space Management

`RnBeads` can perform analysis on much larger data sets than those presented in the examples above. Starting from several hundred of Infinium450k samples or already a few dozen WGBS profiles the data matrices in the `RnBeads` objects become too large to keep them in main memory (RAM). To handle large amounts of data `RnBeads` makes use of the disk-backed objects provided by the `ff` package [7]. In other words, rather than carrying around the huge matrices in main memory, they are stored on the hard drive. This enables large analysis also on machines with an intermediate amount of RAM, but comes at the cost of slightly increased runtimes (accessing matrices on the hard drive is not as fast as accessing them from the main memory). In order to enable this feature use:

```
> rnb.options(disk.dump.big.matrices=TRUE, disk.dump.bigff=TRUE)
```

The file system location which will be used by the `ff` objects is controlled by a global R option (notice that it is not an `RnBeads` option!):

```
> options(fftempdir="/path/to/a/very/large/directory/")
```

By default, the `ff` files will be saved to the current R temporary directory, which is usually a subdirectory of the systems global temporary directory and can be found by executing:

```
> tempdir()
```

In case of Infinium 450k data sets the package will need roughly 10 GB of hard drive space per 100 samples. So make sure that you have enough space for your analysis. You can monitor the amount of hard drive space occupied by the `RnBeads` objects in the logger statements by setting:

```
> rnb.options(logging.disk=TRUE)
```

In order to ensure that the disk-backed objects are effectively cleaned up during the pipeline execution use the following option:

```
> rnb.options(enforce.memory.management=TRUE)
```

This will force `RnBeads` to use garbage collection so that no obsolete `ff` files remain on the hard drive.

After setting these memory tweaking options, you can load data into `RnBSet` objects in the same ways that you already know:

```
> rnb.set.disk <- rnb.execute.import(data.source=data.source, data.type="idat.dir")
```

Data handling is exactly the same way as before. You can execute all functions that also work non-disk-backed `RnBSet` objects.

The `RnBSet` objects created while the `disk.dump.big.matrices` option is enabled cannot be saved using standard `save` and `load` methods in R. Saving should be performed as follows:

```
> save.dir <- file.path(report.dir, "analysis")
> save.rnb.set(rnb.set.disk, path=save.dir, archive=TRUE)
```

After that you should be able to find a file `analysis15.zip` in the `report.dir` directory. This object can be reloaded into another R session, possibly on a different machine:

```
> rns <- load.rnb.set(path=paste0(save.dir, ".zip"))
```

The `ff` files behind an `RnBeads` object can be deleted completely from the hard disk by executing the destructor method:

```
> destroy(rnb.set.disk)
```

Overall, when applied to data sets with a large number of samples or to whole-genome bisulfite sequencing data, `RnBeads` can become computationally demanding, so one should consider running the package in an high-performance environment, e.g. on dedicated compute servers or clusters.

6.5 Some Sugar and Recipes

This section introduces a few examples on how to work with `RnBSet` objects once they have been computed. `RnBeads` contains various functions that cater to downstream analysis.

6.5.1 Plotting Methylation Value Distributions

This example produces beta density distributions for groups of samples as in Figure 13

```
> rnb.plot.betadistribution.sampleGroups(meth(rnb.set),
+     rnb.sample.groups(rnb.set)[["Sample_Group"]], "ESC or iPSC")
```

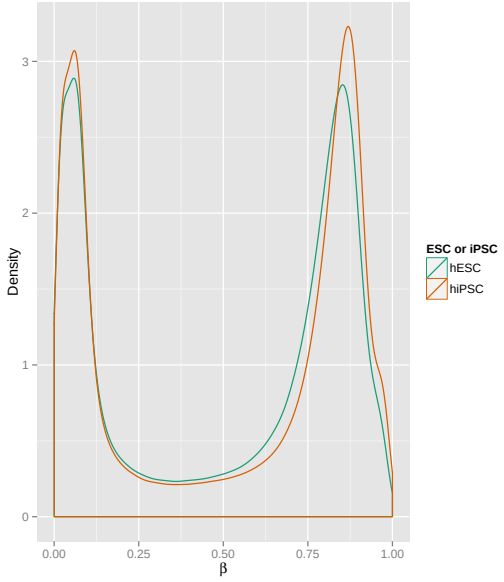

Figure 13: Density plots of probe beta values comparing hESCs to hiPSCs.

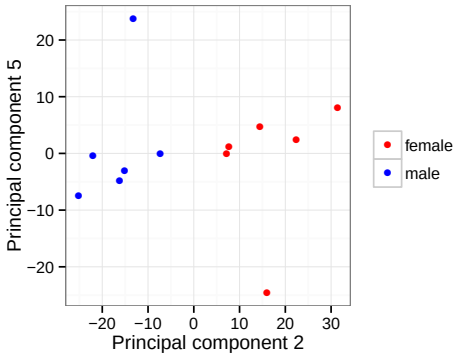

Figure 14: The second and fifth principal components of the (unnormalized) dataset. Point colors denote sex information.

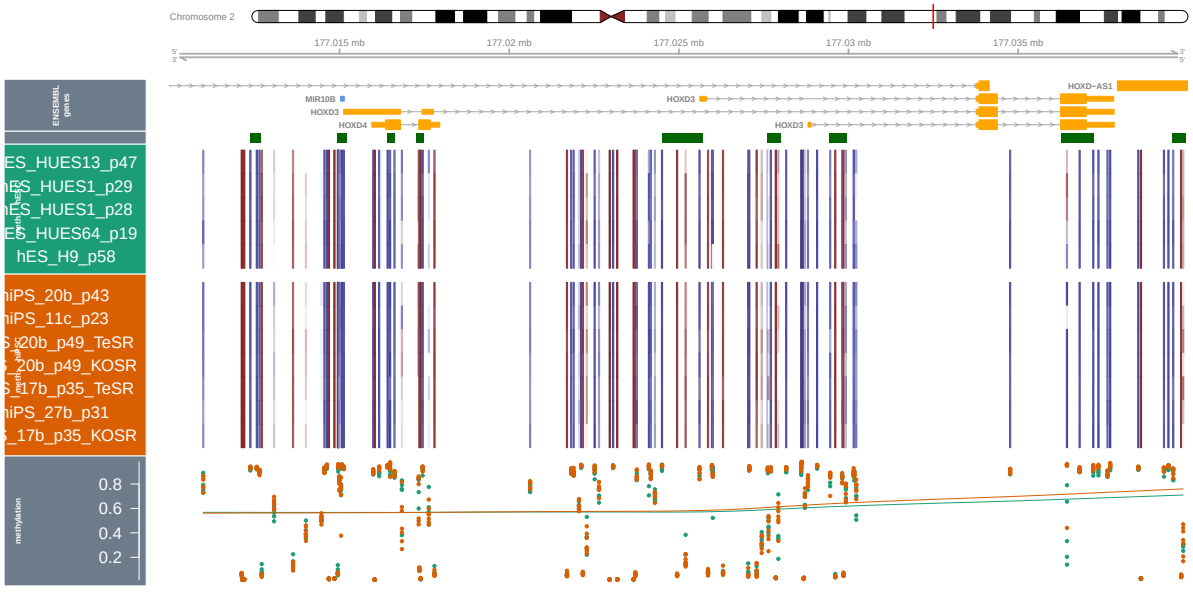

Figure 15: Methylation profile for the HOXB3 locus

6.5.2 Plotting Low Dimensional representations

As mentioned in Section 4.5, RnBeads creates figures that show the high-dimensional methylation data transformed to two dimensions only, by applying the techniques MDS and PCA. Such plots can be created using the function `rnb.plot.dreduction`. This function provides even greater flexibility in the visualization than what can be seen in the reports. For example, the code snippet below modifies the default colors denoting different categories and then displays the second and fifth principal components of the example dataset. Sex information is mapped to colors. The resulting plot is shown in Figure 14.

```
> theme_set(theme_bw())
> rnb.options(colors.category = c("red", "blue", "grey"))
> print(rnb.plot.dreduction(rnb.set, dimensions = c(2, 5),
+     point.colors="Predicted Sex"))
```

6.5.3 Generating Locus Profile Plots (aka Genome-Browser-Like Views)

`RnBeads` provides functionality for generating plots of methylation levels in custom genomic intervals similarly as in current genome browsers. It makes use of the popular `ggbio` package for this. The following code produces the plot depicted in Figure 15:

```
> # Coordinates around the HOXD3 locus
> chrom <- "chr2"
> start <- 177010000
> end <- 177040000
> sample.grouping <- rnb.sample.groups(rnb.set)[[1]]
> rnb.plot.locus.profile(rnb.set, chrom, start, end, grps=sample.grouping)
```

The `rnb.section.locus.profiles()` function is very useful if you are interested in a whole set of genomic regions and want to create a corresponding `RnBeads` report for easy browsing of multiple genomic locations.

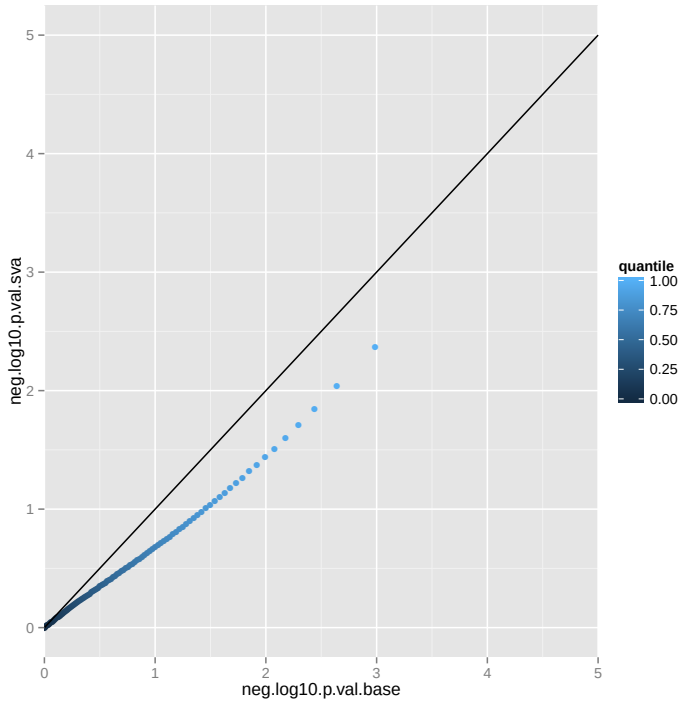

Figure 16: QQ-plot comparing the negative $\log_{10}$ p-values for the unadjusted and the SV-adjusted analysis of differential methylation.

6.5.4 Adjusting for Surrogate Variables in Differential Methylation Analysis

In section 4.4.1 you learned how you can use `RnBeads` to infer Surrogate Variables (SVs). Here, we see an example on how these inferred SVs can be adjusted for in differential methylation analysis and how the results can differ compared to an unadjusted analysis. In the code below, we first infer the SVs. Then we compute tables containing metrics of differential methylation for the unadjusted and the SVA-adjusted case. Finally, we compare the p-values obtained from the `limma` method using a qq-plot (see Figure 16).

```
> # set the target comparison column and region types for
> # differential methylation analysis
> cmp.cols <- "Sample_Group"
> cmp.name <- "hESC vs. hiPSC (based on Sample_Group)"
> reg.types <- c("cpgislands", "promoters")
> # if you have not done so yet, compute the SVs and add them to the RnBSet object
> sva.obj <- rnb.execute.sva(rnb.set,cmp.cols=cmp.cols,numSVmethod="be")
> rnb.set.sva <- set.covariates.sva(rnb.set, sva.obj)
> # compute differential methylation tables: unadjusted and SVA adjusted
> diffmeth.base <- rnb.execute.computeDiffMeth(
+         rnb.set.sva, cmp.cols, region.types=reg.types,
+         adjust.sva=FALSE
+ )
> diffmeth.sva <- rnb.execute.computeDiffMeth(
+         rnb.set.sva, cmp.cols, region.types=reg.types,
```

```
+         adjust.sva=TRUE, pheno.cols.adjust.sva=cmp.cols
+ )
> dm.tab.base <- get.table(diffmeth.base, cmp.name, "sites",
+         return.data.frame=TRUE)
> dm.tab.sva  <- get.table(diffmeth.sva,  cmp.name, "sites",
+         return.data.frame=TRUE)
> # compute quantiles of -log10 p-values and prepare a data.frame for plotting
> p.val.perc.base <- quantile(-log10(dm.tab.base$diffmeth.p.val),
+         probs = seq(0, 1, 0.01))
> p.val.perc.sva  <- quantile(-log10(dm.tab.sva$diffmeth.p.val),
+         probs = seq(0, 1, 0.01))
> df <- data.frame(
+         neg.log10.p.val.base=p.val.perc.base,
+         neg.log10.p.val.sva=p.val.perc.sva,
+         quantile=seq(0, 1, 0.01)
+ )
> # plot using ggplot2
> library(ggplot2)
> ggplot(df,aes(x=neg.log10.p.val.base,y=neg.log10.p.val.sva,color=quantile)) +
+ geom_point() + geom_abline(intercept=0, slope=1) + xlim(0, 5) + ylim(0, 5)
```

The resulting Figure 16 shows a slight inflation of the unadjusted p-values when compared to the SVA-adjusted case.

6.5.5 Generating Density-Scatterplots

The Differential Methylation reports generated by RnBeads contain scatterplots indicating point density and highlight the most likely candidates for differential methylation. You can easily produce these plots on your own using `RnBeads`' `create.densityScatter` function. The following example visualizes the differences between ESCs and iPSCs you have seen in section 4.6 assuming you have already computed the `diffmeth` object (see Figure 17).

```
> comparison <- "hESC vs. hiPSC (based on Sample_Group)"
> dframe <- get.table(diffmeth, comparison, "sites", return.data.frame=TRUE)
> # define the probes with an FDR corrected p-value below 0.05
> # as differentially methylated
> isDMP <- dframe[,"diffmeth.p.adj.fdr"] < 0.05
> create.densityScatter(dframe, is.special=isDMP, sparse.points=0.001,
+         add.text.cor=TRUE) + labs(x="mean beta: hESC", y="mean beta: hiPSC") +
+         coord_fixed()
```

6.6 Correcting for Cell Type Heterogeneity Effects

Cell type heterogeneity of the profiled samples is a widely recognized source of confounding in DNA methylation profiling studies. Several methods have been proposed to tackle it in large data sets, some of which rely upon availability of reference methylomes [14, 15], while the other can be applied directly [19, 20]. `RnBeads` supports several approaches for cell type heterogeneity inference and adjustment. In case the reference methylomes are available, they can be used to

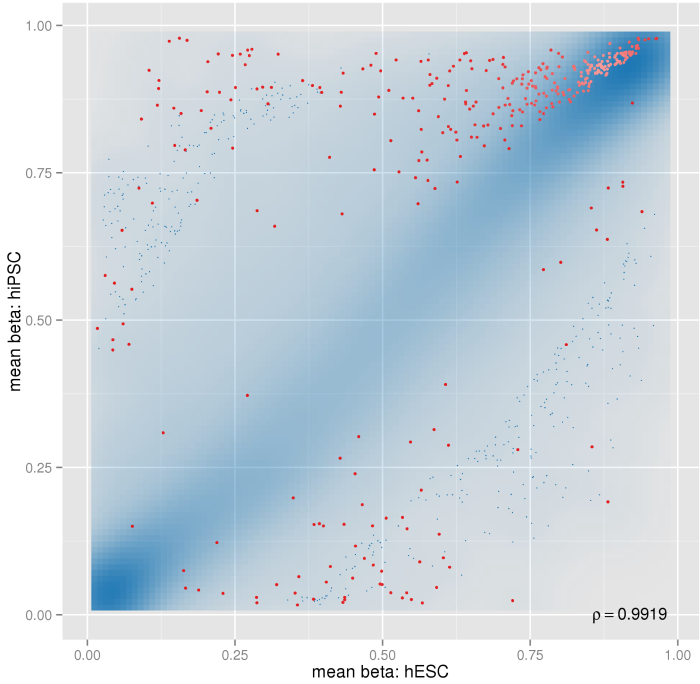

Figure 17: Density-scatterplot showing probes differentially methylated between ESCs and iPSCs.

estimate the relative contributions of each cell type (see Section 4.4.2). Next, the contribution estimates can be used as covariates in the limma-based differential methylation analysis.

For the case when reference methylomes are absent, differential methylation analysis in RnBeads can be corrected using the reference-free method by Houseman *et al.* [19]. RefFreeEWAS analysis is fully integrated with differential methylation module of RnBeads, and, similarly to limma-based analysis, can adjust for covariates. It can be activated using a single option:

```
> rnb.options(differential.site.test.method="refFreeEWAS")
```

Keep in mind that RefFreeEWAS uses bootstrapping to estimate the variance of the model coefficients, and thus may take a lot of computational time already on moderate-sized data sets.

Finally, RnBeads also provides an option to export the data ready to be processed by the EWASHER tool [20]. The input files as well as a brief guidance on how to start the EWASHER, are prepared in a specialized section of the inference module report. The EWASHER export can is controlled via a dedicated option:

```
> rnb.options(export.to.ewasher=TRUE)
```

6.7 Deploying RnBeads on a Scientific Compute Cluster

This section is intended for advanced users only

RnBeads can be deployed on a scientific compute cluster. We have implemented the necessary configurations for a Sun Grid Engine (SGE) environment. However, the required classes can be easily extended to fit any cluster environment. To run an RnBeads analysis on the cluster, it is necessary that the analysis options and data source are specified in an XML file. You can

find examples for such files in the "Methylome Resource" section of the RnBeads website. The following example illustrates how you can submit the RnBeads jobs for an analysis to the cluster. Log on to a machine which runs R and which can submit jobs to the cluster. Run R and then submit an analysis using the following commands:

```
> library(RnBeads)
> # specify the xml file for your analysis
> xml.file <- "MY_ANALYSIS_SETTINGS.XML"
> # set the cluster architecture specific to your environment. We use SGE here
> arch <- new("ClusterArchitectureSGE")
> # initialize the object containing the RnBeads specific cluster configuration
> rnb.cr <- new("RnBClusterRun",arch)
> # set up the cluster so that 32GB of memory are required
> # (SGE resource is called "mem_free")
> rnb.cr <- setModuleResourceRequirements(rnb.cr,c(mem_free="32G"),"all")
> # set up the cluster to use 4 cores on each node for all modules
> rnb.cr <- setModuleNumCores(rnb.cr,4L,"all")
> # set up the cluster to use 2 cores for the exploratory analysis module
> rnb.cr <- setModuleNumCores(rnb.cr,2L,"exploratory")
> # run the actual analysis
> run(rnb.cr, "rnbeads_analysis", xml.file)
```

If you want to configure your own cluster environment, all you have to do is to extend `RnBeads`' `ClusterArchitecture` S4 class. Here, all that you really need to define is the `getSubCmdTokens` method, which returns a vector of command line tokens required for submitting a cluster job in your environment. The following example illustrates the specification of a new class for a hypothetical environment which submits jobs to the cluster using the submission syntax

```
iSubmit -myParameter 5 -waitForJobs [job1,job2] -jobName [MY_JOB_NAME] -logFile
[LOGFILE] '[COMMAND_TO_RUN]'
```

```
> # Define the new class, extending RnBeads ClusterArchitecture class
> setClass("ClusterArchitectureMyEnv",
+         contains = "ClusterArchitecture"
+ )
> #define the getSubCmdTokens method
> setMethod("getSubCmdTokens",
+   signature(
+     object="ClusterArchitectureMyEnv"
+   ),
+   function(
+     object,
+     cmd.tokens,
+     log,
+     job.name = "",
+     res.req = character(0),
+     depend.jobs = character(0)
+   ) {
+     dependency.token <- NULL
+     if (length(depend.jobs)>0){
```

```
+       dependency.token <- c( "--waitForJobs", paste(depend.jobs,collapse=","))
+     }
+     job.name.token <- NULL
+     if (nchar(job.name)>0) {
+       job.name.token <- c("--jobName",job.name)
+     }
+     result <- c(
+       "iSubmit", "--myParamter", "5",
+       dependency.token, job.name.token,
+       "--logFile",log,
+       paste0("'",paste(cmd.tokens,collapse=" "),"'")
+     )
+     return(result)
+   }
+ )
```

If you now want to submit an RnBeads analysis using your brand new environment all you have to do is to to replace `"ClusterArchitectureSGE"` by `"ClusterArchitectureMyEnv"` in the previous example.

For the SGE example, see

```
> ?`getSubCmdTokens,ClusterArchitectureSGE-method`
```

It might also be helpful to have a look at the source code for the SGE class. You can find it in the source files of RnBeads (location: `R/clusterArchitectureSGE.R`).

6.8 Genome-wide segmentation of the methylome

Global DNA methylation has been shown to be organized into larger domains using whole-genome bisulfite sequencing data [23]. The strategy is based on a hidden-markov model that first classifies the regions into either partially methylated domains (PMDs) and into the remaining. The latter is then further classified into fully methylated domains (FMDs), lowly methylated regions (LMRs), and unmethylated regions (UMRs) according to overall methylation state and CpG content [24]. Each of these categories is annotated to a distinct biological function. `RnBeads` support segmentation according to this proposed `MethylSeekR` approach.

MethylSeekR is only applicable to larger WGBS datasets and produces a segmentation for each sample individually. Such a dataset is too large to be shiped together with `RnBeads`, thus we assume we loaded a larger, preprocessed `RnBiseqSet` object obtained from WGBS data called `rnb.set`.

```
> ?rnb.execute.segmentation
> rnb.set <- rnb.execute.segmentation(rnb.set=rnb.set,
+         sample.name=samples(rnb.set)[1])
> rnb.set
```

The dataset now contains further regions types representing the segmentation described above. This segmentation can now be exported to BED files or the methylation values of the CpGs in the different segments can be plotted.

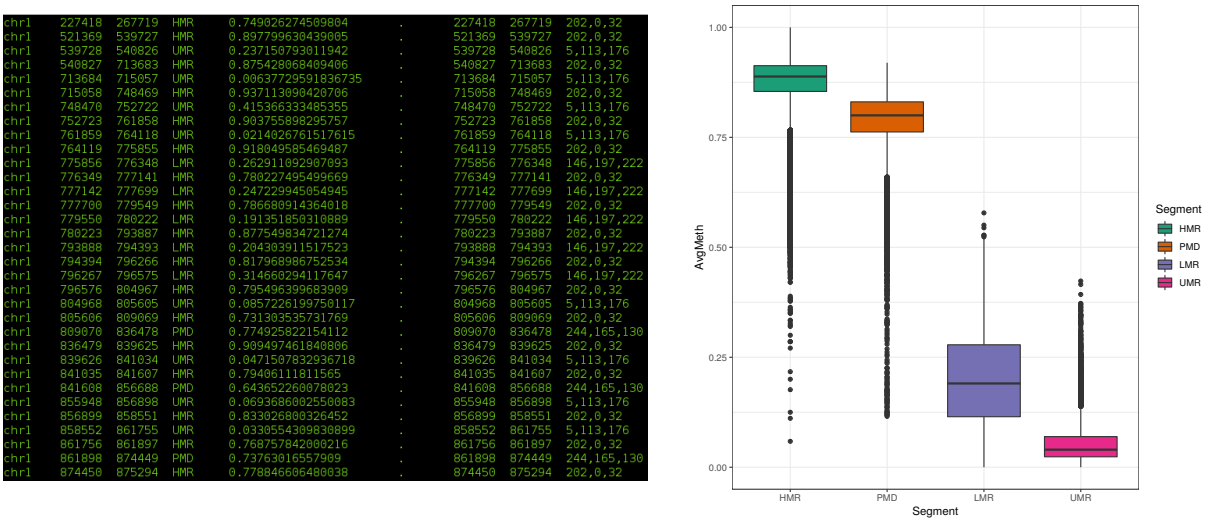

Figure 18: An example BED files (left) and methylation distribution (right) according to the different segments defined by the MethylSeekR approach.

```
> segmentation.results <- "~/segmentation"
> rnb.bed.from.segmentation(rnb.set = rnb.set,
+          sample.name = samples(rnb.set)[1], store.path = segmentation.results)
> plot <- rnb.boxplot.from.segmentation(rnb.set = rnb.set,
+           sample.name = samples(rnb.set)[1])+theme_bw()
```

The resulting BED file and plots will look as shown in Figure 18, but might vary from sample to sample.

7 Beyond DNA Methylation Analysis

RnBeads' extensive logging features and functions which let you generate flexible hypertext reports extend the package's usefulness beyond DNA methylation analysis.

7.1 HTML Reports

You can easily create entire report websites containing analysis results from within R using RnBeads. The reports' format is `html` and they thus provide a useful way of keeping track of your research results and share them with collaborators.

Reports need to be initialized, i.e. the directory for the report needs to be specified and stylesheets and other configuration settings need to be copied to the target directory.

```
> report.dir <- "RnBeads_report"
> rnb.initialize.reports(report.dir)
> report <- createReport(file.path(report.dir, "myreport.html"), "An Eye Opener",
+     page.title="Fascinating Report", authors=c("Me", "You"))
```

The last command in the above example creates an object of class `Report`:

```
> str(report)
```

The command `rnb.initialize.reports` creates the `configuration` directory (usually shared among several reports), containing common icons, as well as files that define the behaviour, visual style and layout of the report pages. Also, notice that the `RnBeads_report` directory now contains other subdirectories according to the following slots of the `report` object:

`dir.data` Directory in which data is placed that is linked to in the report. This can be any data.

`dir.pngs` Directory in which image files in `png` format are placed. These are files that are generated by methods like `createReportPlot()`. They should be relatively small in file size such that they can easily displayed on the web.

`dir.pdf` Directory in which pdf versions of plots will be stored.

`dir.high` Directory in which high resolution versions of plots will be stored. They can be larger in terms of file sizes.

If you view `myreport.html` in your favorite browser you will notice that it contains titles, but no real content as of yet. So, let us change that by adding a section with some paragraphs:

```
> stext <- "Here is some text for our awesome report"
> report <- rnb.add.section(report, "Adding a text section", stext)
> lorem1 <- c("Lorem ipsum dolor sit amet, consectetur adipiscing elit.
+             Maecenas vestibulum placerat lobortis. Ut viverra fringilla
+             urna at rutrum. In hac habitasse platea dictumst. Vestibulum
+             adipiscing rutrum libero et interdum. Etiam sed odio ac nibh
+             ullamcorper congue. Proin ac ipsum elit. Ut porta lorem sed
+             lorem molestie pharetra.",
+             "Vestibulum ante ipsum primis in faucibus orci luctus et
+             ultrices posuere cubilia Curae; Cras ac augue eu turpis
+             dignissim aliquam. Vivamus at arcu ligula, vel scelerisque nisi.
+             Vivamus ac lorem libero, quis venenatis metus. Fusce et lectus
+             at lectus vestibulum faucibus ac id sem.")
> report <- rnb.add.section(report, "A subsection", lorem1, level=2)
> lorem2 <- "Nunc congue molestie fringilla. Aliquam erat volutpat.
+            Integer consequat turpis nec dolor pulvinar et vulputate magna
+            adipiscing. Curabitur purus dolor, porttitor vel dapibus quis,
+            eleifend at lacus. Cras at mauris est, quis aliquam libero.
+            Nulla facilisi. Nam facilisis placerat aliquam. Morbi in odio
+            non ligula mollis rhoncus et et erat. Maecenas ut dui nisl.
+            Mauris consequat cursus augue quis euismod."
> rnb.add.paragraph(report, lorem2)
> rnb.add.paragraph(report, "TODO: Add content here", paragraph.class="task")
> rnb.add.paragraph(report, "To be or not to be, that is the question",
+     paragraph.class="note")
```

Again, view `myreport.html` and notice that it has been filled with content. Other objects, like lists and tables can be added as well:

```
> report <- rnb.add.section(report, "Lists and Tables", "")
> ll <- lapply(LETTERS[1:10], function(x) { paste(rep(x, 3), collapse="") })
```

```
> rnb.add.list(report, ll, type="o")
> tt <- matrix(sapply(LETTERS[1:24],
+     function(x) { paste(rep(x, 3), collapse="") }), ncol=4)
> colnames(tt) <- paste("Col", 1:4)
> rownames(tt) <- paste("Row", 1:6)
> rnb.add.table(report, tt, row.names=TRUE, first.col.header=TRUE,
+     tcaption="A table")
```

Note that text will be placed in the report as `html` text. Hence you can format text and add it to the report:

```
> stext <- c('<p>Some German umlauts:
+                 <ul>
+                     <li>&Auml;</li>
+                     <li>&Ouml;</li>
+                     <li>&Uuml;</li>
+                 </ul>
+             </p>',
+             '<p> A link: <a href="https://rnbeads.org/">
+             RnBeads website</a></p>')
> report <- rnb.add.section(report, "HTML code", stext)
```

Now, only text is boring. So, let us add some eye candy, i.e. some plots:

```
> report <- rnb.add.section(report, "Plots", "")
> report.plot <- createReportPlot("hist_normal", report, create.pdf=TRUE,
+     high.png=200)
> hist(rnorm(1000), breaks=50, col="blue")
> off(report.plot)
> desc <- 'A histogramm of samples drawn from the normal distribution
+          Click <a href="myreport_pdfs/hist_normal.pdf">
+          here</a> for the pdf version.'
> report <- rnb.add.figure(report, desc, report.plot)
```

Now, `myreport.html` will contain a figure. If created as in the case above, the high resolution version of it is accessible by clicking on the figure or in the `myreport_images` directory. A `pdf` version will be available in the `myreport_pdfs` directory.

In bioinformatics analysis, inspecting plots with multiple parameter settings is a common task. `RnBeads`' reports can handle these using dropdown menus for possible values of different parameters and showing the corresponding plot file. Here's an example:

```
> # Plot a sine curve
> plotSine <- function(a, b, fname) {
+     report.plot <- createReportPlot(fname, report, create.pdf=FALSE,
+         high.png=200)
+     curve(sin(a*x)+b, -2*pi, 2*pi, ylim=c(-2, 2), col="blue", lwd=2)
+     return(off(report.plot))
+ }
> # Set parameters for different sine curves
```

```
> period.params <- c(a05=0.5, a1=1, a2=2)
> intercept.params <- c(im1=-1, i0=0, i1=1)
> plot.list <- list()
> for (aa in names(period.params)){
+     for (bb in names(intercept.params)){
+         fname <- paste("sinePlot", aa, bb, sep="_")
+         current.plot <- plotSine(period.params[aa], intercept.params[bb],
+             fname)
+         plot.list <- c(plot.list, current.plot)
+     }
+ }
> setting.names <- list('period parameter'=period.params,
+     'intercept parameter'=intercept.params)
> description <- 'Sine for various parameters'
> report <- rnb.add.figure(report, description, plot.list, setting.names,
+      selected.image=5)
```

Finally, close the report, which adds the footer to the page:

```
> off(report)
```

7.2 Logging

In order to keep track of what your program is doing, logging is often essential. `RnBeads`' logging functionality is implemented via a few easy-to-handle methods that can be integrated in any code (not just bioinformatics). In this section, we introduce a few examples on how to use the logging capabilities.

First, a logger needs to be initialized:

```
> logger.start("Logging tutorial", fname=NA)
```

```
2026-10-13 20:01:37     2.2  STATUS STARTED Logging tutorial
```

will initialize a logger with its standard output being the console. You can also specify a file name for `fname` to redirect the logging output to a file.

```
> logger.isinitialized()
```

```
[1] TRUE
```

will tell you whether a logger has been initialized. Whenever you start a particular task, you can tell the logger to initialize a new section:

```
> logger.start("Some tedious task")
```

```
2026-10-13 20:01:37     2.2  STATUS     STARTED Some tedious task
```

By default, the logging message is automatically appended to the initialized logger. When you complete your task, just tell the logger:

```
> logger.completed()
```

```
2026-10-13 20:01:37     2.2  STATUS     COMPLETED Some tedious task
```

Notice the indentation of "Some tedious task". You can nest as many subtasks as you like:

```
> f <- function(){
+     logger.start("Another tedious task")
+     Sys.sleep(2)
+         logger.start("Some tedious subtask")
+         Sys.sleep(2)
+         logger.completed()
+         logger.start("Another, more tedious subtask")
+         Sys.sleep(2)
+             logger.start("Some tedious subsubtask")
+             Sys.sleep(2)
+             logger.completed()
+             logger.start("Some even more tedious subsubtask")
+             Sys.sleep(3)
+             logger.completed()
+         logger.completed()
+     logger.completed()
+ }
> f()
```

```
2026-10-13 20:01:37     2.2  STATUS     STARTED Another tedious task
2026-10-13 20:01:39     2.2  STATUS         STARTED Some tedious subtask
2026-10-13 20:01:41     2.2  STATUS         COMPLETED Some tedious subtask
2026-10-13 20:01:41     2.2  STATUS         STARTED Another, more tedious subtask
2026-10-13 20:01:43     2.2  STATUS             STARTED Some tedious subsubtask
2026-10-13 20:01:45     2.2  STATUS             COMPLETED Some tedious subsubtask
2026-10-13 20:01:45     2.2  STATUS             STARTED Some even more tedious subsubtask
2026-10-13 20:01:48     2.2  STATUS             COMPLETED Some even more tedious subsubtask
2026-10-13 20:01:48     2.2  STATUS         COMPLETED Another, more tedious subtask
2026-10-13 20:01:48     2.2  STATUS     COMPLETED Another tedious task
```

The logging capabilities also include status messages in various flavors:

info informs the user of a particular program setting, e.g. the current value of a particular variable. The information in the message is typically dependent on the program input.

status informs the user of which part of a program has been reached. These messages are typically fixed strings.

warning warns the user that something did not go as expected.

error a more severe form of a warning: something went terribly wrong. Causes an error to be thrown which terminates the program if not caught.

Here is an example function that uses the logging functionality

```
> draw.lotto <- function(count=6) {
+         urn <- 1:49
```

```
+         primes <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47)
+         logger.start(c("Drawing", count, "numbers from", length(urn)))
+         for (i in 1:count) {
+                 if (i > 9) {
+                         msg <- "Too many numbers drawn"
+                         logger.error(msg, terminate=FALSE)
+                         stop(msg)
+                 }
+                 drawn <- sample(urn, 1)
+                 urn <- setdiff(urn, drawn)
+                 logger.info(c("The next number is", drawn))
+                 if (drawn %in% primes) {
+                         logger.warning("A prime number was drawn")
+                 }
+         }
+         logger.completed()
+ }
> draw.lotto()
```

```
2026-10-13 20:01:48     2.2  STATUS     STARTED Drawing 6 numbers from 49
2026-10-13 20:01:48     2.2    INFO         The next number is 42
2026-10-13 20:01:48     2.2    INFO         The next number is 44
2026-10-13 20:01:48     2.2    INFO         The next number is 6
2026-10-13 20:01:48     2.2    INFO         The next number is 18
2026-10-13 20:01:48     2.2    INFO         The next number is 43
2026-10-13 20:01:48     2.2 WARNING         A prime number was drawn
2026-10-13 20:01:48     2.2    INFO         The next number is 2
2026-10-13 20:01:48     2.2 WARNING         A prime number was drawn
2026-10-13 20:01:48     2.2  STATUS     COMPLETED Drawing 6 numbers from 49
```

```
> #draw.lotto(15)
```

Note that setting the parameter `terminate` in `logger.error()` to `TRUE` will cause R to terminate. The logger does not automatically generate R warnings and errors.

References

[1] Ziller, M. J., Müller, F., Liao, J., Zhang, Y., Gu, H., Bock, C., Boyle, P., *et al.* (2011). *Genomic Distribution and Inter-Sample Variation of Non-CpG Methylation across Human Cell Types.* PLoS Genetics, 7(12)

[2] Nazor, K.L., Altun, G., Lynch, C., Tran, H., Harness, J.V., Slavin, I., Garitaonandia, I. *et al.* (2012), *Recurrent Variations in DNA Methylation in Human Pluripotent Stem Cells and Their Differentiated Derivatives.* Cell Stem Cell, 10(5),620-634

[3] Makambi, K. (2003). *Weighted inverse chi-square method for correlated significance tests.* Journal of Applied Statistics, 30(2), 225-234

[4] Maksimovic, J., Gordon, L., Oshlack, A. (2012). *SWAN: Subset quantile Within-Array Normalization for Illumina Infinium Infinium 450k BeadChips.* Genome Biology, 13(6), R44

[5] Liu, J., Siegmund, K., Laird, P., Berman, B. (2012). *Bis-SNP: combined DNA methylation and SNP calling for Bisulfite-seq data* Genome Biology, 13(7), R44

[6] Pidsley, R., Wong, C., Volta, M., Lunnon, K., Mill, J., and Schalkwyk, L. (2013) *A data-driven approach to preprocessing Illumina 450K methylation array data. BMC Genomics*,14(1), 293

[7] Adler, D., Gläser, C., Nenadic, O., Oehlschlägel, J. and Zucchini, W. (2012) *ff: memory-efficient storage of large data on disk and fast access functions. R package version 2.2-12.* http://ff.r-forge.r-project.org

[8] Davis, S., Du, P., Bilke, S., Triche, T., Bootwalla, M. (2013) *methylumi: Handle Illumina methylation data. R package version 2.6.1.*

[9] Hansen, K.D., Aryee, M *minfi: Analyze Illumina's 450k methylation arrays. R package version 1.6.0.*

[10] Teschendorff, A.,Marabita, F., Lechner, M., Bartlett, T., Tegner, J., Gomez-Cabrero, D., and Beck, S. (2013).*A beta-mixture quantile normalization method for correcting probe design bias in Illumina Infinium 450 k DNA methylation data.* Bioinformatics, 29(2),189-196

[11] Triche, T.J., Jr., Weisenberger, D.J., Van Den Berg, D., Laird, P.W., and Siegmund, K.D. (2013). *Low-level processing of Illumina Infinium DNA Methylation BeadArrays.* Nucleic Acids Res, 41(7), e90

[12] Leek, J. T., Storey, J. D. (2007). *Capturing Heterogeneity in Gene Expression Studies by Surrogate Variable Analysis.* PLoS Genetics, 3(9), 1724-1735.

[13] Leek, J. T., Johnson, W. E., Parker, H. S., Jaffe, A. E., Storey, J. D. (2012). *The sva package for removing batch effects and other unwanted variation in high-throughput experiments.* Bioinformatics, 28(6), 882-883.

[14] Houseman, E.A., Accomando W.P., Koestler D.C.,Christensen B.C., Marsit C.J., Nelson H.H., Wiencke J.K., Kelsey K.T. (2012). *DNA methylation arrays as surrogate measures of cell mixture distribution.* BMC Bioinformatics 13:86.

[15] Guintivano J., Aryee M.J., Kaminsky Z.A. (2013). *A cell epigenotype specific model for the correction of brain cellular heterogeneity bias and its application to age, brain region and major depression.* Epigenetics, 8(3), 290-302.

[16] Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., *et al.* (2000). *Gene ontology: tool for the unification of biology.* Nature Genetics, 25(1), 25-29

[17] Sheffield, N. C., and Bock, C. (2016). *LOLA: enrichment analysis for genomic region sets and regulatory elements in R and Bioconductor.* Bioinformatics, 32(4), 587-589

[18] Falcon, S., and Gentleman, R. (2007). *Using GOstats to test gene lists for GO term association.* Bioinformatics, 23(2), 257-258

[19] Houseman, E. A., Molitor, J., and Marsit, C. J. (2014). *Reference-Free Cell Mixture Adjustments in Analysis of DNA Methylation Data.* Bioinformatics, btu029.

[20] Zou, J., Lippert, C., Heckerman, D., Aryee, M., and Listgarten, J. (2014). *Epigenome-wide association studies without the need for cell-type composition.* Nature Methods. 11, 309-311.

[21] Fortin, J.-P., Labbe, A., Lemire, M., Zanke, B. W., Hudson, T. J., Fertig, E. J., *et al.* (2014). *Functional normalization of 450k methylation array data improves replication in large cancer studies.* Genome Biology, 15(12), 503

[22] Aran, D., Sirota, M., Butte, A.J. (2014). *Systematic pan-cancer analysis of tumour purity.* Nature Communications, 6, 8971

[23] Salhab, A., Nordström, K., Gasparoni, G., Kattler, K., Ebert, P., et.al. (2018) *A comprehensive analysis of 195 DNA methylomes reveals shared and cell-specific features of partially methylated domains.* Genome Biology, 19(1), 9-11.

[24] Burger, L., Gaidatzis, D., Schübeler, D., and Stadler, M. B. (2013). *Identification of active regulatory regions from DNA methylation data.* Nucleic Acids Research, 41(16), e155. https://doi.org/10.1093/nar/gkt599