Analysis of single-cell RNA-seq data

Vladimir Kiselev, Tallulah Andrews and Martin Hemberg

2016-03-24

Contents

Chapter 1

About the course

Recent technological advances have made it possible to obtain genome-wide transcriptome data from single cells using high-throughput sequencing (scRNA-seq). Even though scRNA-seq makes it possible to address problems that are intractable with bulk RNA-seq data, analysing scRNA-seq is also more challenging.

In this course we will be surveying the existing problems as well as the available computational and statistical frameworks available for the analysis of scRNA-seq.

1.1 Registration

http://training.csx.cam.ac.uk/bioinformatics/event/1626698

1.2 GitHub

https://github.com/hemberg-lab/scRNA.seq.course

1.3 License

1.4 Prerequisites

The course is intended for those who have basic familiarity with Unix and the R scripting language.

We will also assume that you are familiar with mapping and analysing bulk RNA-seq data as well as with the commonly available computational tools.

We recommend attending the Introduction to RNA-seq and ChIP-seq data analysis or the Analysis of high-throughput sequencing data with Bioconductor before attending this course.

Chapter 2

Technical requirements

We tried to make this course purely R-based. However, one of the methods that we describe (SNN-Cliq) is only partly R-based. It makes a simple *python* call from R and requires a user to have rights to write to the current directory. You also need to download this file and put it in `~/snn-cliq/` directory.

Other than that, before running the course exercises, all you need to do is to install the following packages in your R:

devtools for installing packages from GitHub:

```
install.packages("devtools")
```

scRNA.seq.funcs - R package containing our own functions used in this course:

```
devtools::install_github("hemberg-lab/scRNA.seq.funcs")
```

mvoutlier - for an automatic outlier detection in the scater package.

```
install.packages("mvoutlier")
```

M3D for identification of important and DE genes, developed by Tallulah Andrews:

```
devtools::install_github("tallulandrews/M3D", ref = "release")
```

RUVSeq for normalization using ERCC controls:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("RUVSeq")
```

ROCR for performance estimations:

```
install.packages("ROCR")
```

limma for plotting Venn diagrams:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("limma")
```

DESeq2 for calculation of DE genes:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("DESeq2")
```

scde for calculation of DE genes:

```
devtools::install_github("hms-dbmi/scde", build_vignettes = FALSE)
```

- Installation on Mac OS X may require this additional gfortran library:

  ```
  curl -O http://r.research.att.com/libs/gfortran-4.8.2-darwin13.tar.bz2
  sudo tar fvxz gfortran-4.8.2-darwin13.tar.bz2 -C /
  ```

- See the help page for additional support.

pheatmap for plotting good looking heatmaps:

```
install.packages("pheatmap")
```

pcaMethods required by **pcaReduce** package below for unsupervised clustering of scRNA-seq data:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("pcaMethods")
```

pcaReduce for unsupervised clustering of scRNA-seq data (bioRxiv):

```
devtools::install_github("JustinaZ/pcaReduce")
```

Rtsne for unsupervised clustering of scRNA-seq data:

```
install.packages("Rtsne")
```

SC3 for unsupervised clustering of scRNA-seq data (bioRxiv):

```
devtools::install_github("hemberg-lab/SC3", ref = "R-old")
```

- Before running **SC3** for the first time **only**, please start R and enter:

```
RSelenium::checkForServer()
```

Chapter 3

Introduction to single-cell RNA-seq

3.1 Bulk RNA-seq

- A major breakthrough (replaced microarrays) in the late 00's and has been widely used since
- Provides an **average expression level** for each gene from a large population of input cells
- Useful for comparative transcriptomics, e.g. samples of the same tissue from different species
- Useful for quantifying expression signatures from ensembles, e.g. in disease studies
- **Insufficient** for studying heterogeneous systems, e.g. early development studies, complex tissues (brain)
- Does **not** provide insights into the stochastic nature of gene expression

3.2 scRNA-seq

- A **new** technology, first publication by Tang et al in 2009
- Instead of providing an average of expression of a population of cells, scRNA-seq provides a **distribution of expression levels** from a population of cells
- Allows to study new biological questions in which **cell-specific changes in transcriptome are important**, e.g. cell type identification, inference of gene regulatory networks across the cells, stochastic component of transcription
- Datasets range **from** 10^2 **to** 10^4 **cells** and increase in size every year
- Currently there are several different protocols in use, e.g. SMART-seq2, CELL-seq and Drop-seq
- Several computational analysis methods from bulk RNA-seq **can** be used
- **In most cases** requires adaptation of the existing methods or development of new ones

3.3 Protocol

image from Wikipedia - Single cell sequencing

Overall, experimental scRNA-seq protocols are similar to the methods used for bulk RNA-seq. For a discussion on experimental methods, please see reviews by Saliba et al, Handley et al or Kolodziejczyk et al.

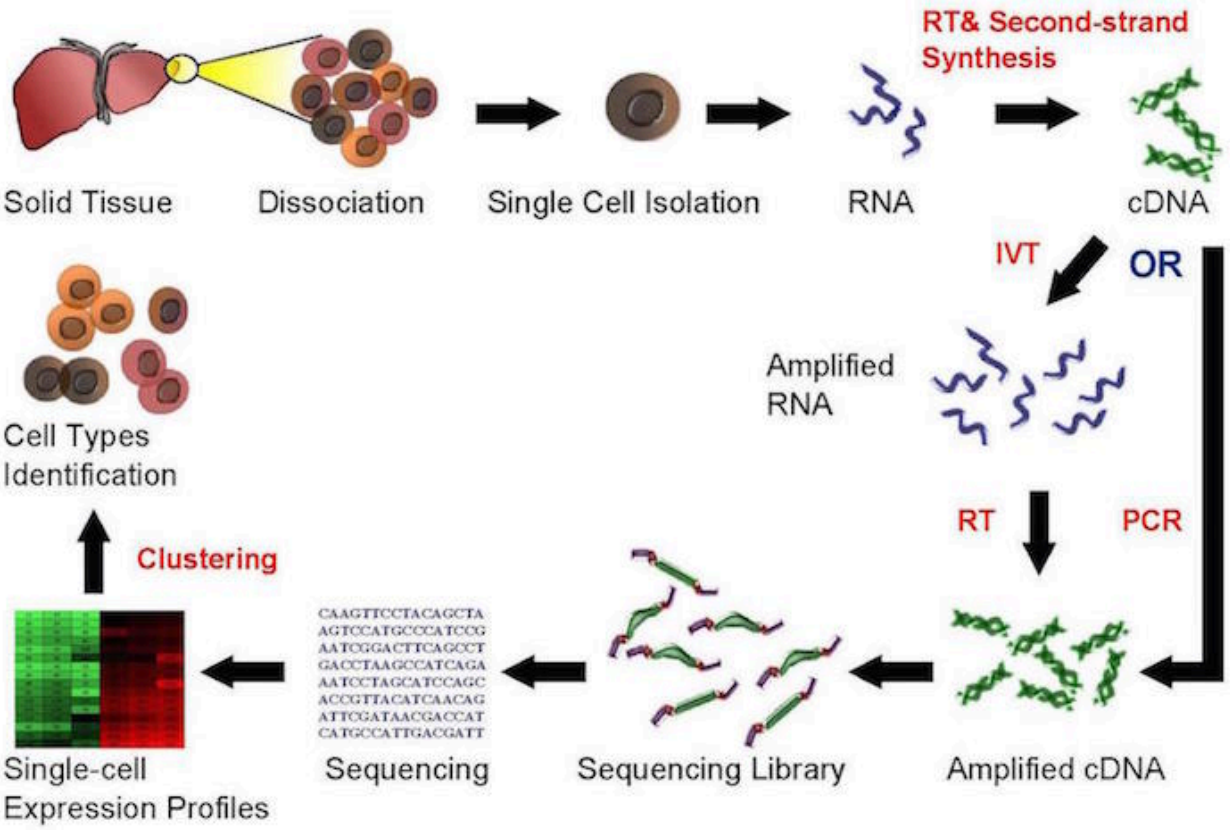
RT& Second-strand Synthesis
Solid Tissue
Dissociation
Single Cell Isolation
RNA
cDNA
IVT
OR
Amplified RNA
Cell Types Identification
RT
PCR
Clustering
CAAGTTCCTACAGCTA
AGTCCATGCCCATCCG
AATCGGACTTCAGCCT
GACCTAAGCCATCAGA
AATCCTAGCATCCAGC
ACCGTTACATCAACAG
ATTCGATAACGACCAT
CATGCCATTGACGATT
Single-cell Expression Profiles
Sequencing
Sequencing Library
Amplified cDNA

Figure 3.1:

3.4 Analysis

This course is concerned with the computational analysis of the data obtained from scRNA-seq experiments. Even though the format of the data is the same as for bulk RNA-seq, the fact that individual cells are sampled means that the data **should** be analyzed differently. We will provide an overview of how to process a scRNA-seq sample and how to analyze the data to provide biological insights.

Flowchart for analyzing scRNA-seq data.

3.5 Challenges

The main difference between bulk and single cell RNA-seq is that each sequencing library represents a single cell, instead of a population of cells. Therefore, significant attention has to be paid to comparison of the results from different cells (sequencing libraries). The main sources of discrepancy between the libraries are:

- **Amplification** (up to 1 million fold)
- **Gene 'dropouts'** in which a gene is observed at a moderate expression level in one cell but is not detected in another cell (Kharchenko et al).

In both cases the discrepancies are introduced due to low starting transcript amount (from one cell only) and have yet to be addressed by improving the experimental protocol.

3.6 Controls

To address challenges in technical variation between scRNA sequencing libraries two quantitative standards were introduced. They allow to normalize gene expression levels across different cells.

3.6.1 ERCC *spike-ins*

One standard, strongly recommended for all scRNA-seq experiments is to use extrinsic *spike-in* molecules. These molecules are added to the lysate before the reverse transcription. The most popular and widely used artificial spike-ins are from External RNA Control Consortium (ERCC). It contains 92 synthetic spikes based on bacterial sequences. Normalization using *spike-ins* is based on the fact that the number of molecules of each *spike-in* RNA species should be the same across all single-cell libraries.

3.6.2 UMIs

Another method of standartisation is to use Unique Molecular Identifiers (UMIs). Instead of sequencing the amplified reads from a cell library, it allows for sequencing reads derived solely from 3' or 5' end of the amplified transcript. UMIs are added as barcodes to the individual RNA molecules. This approach provides an estimate of the number of transcripts that is independent of amplification biases.

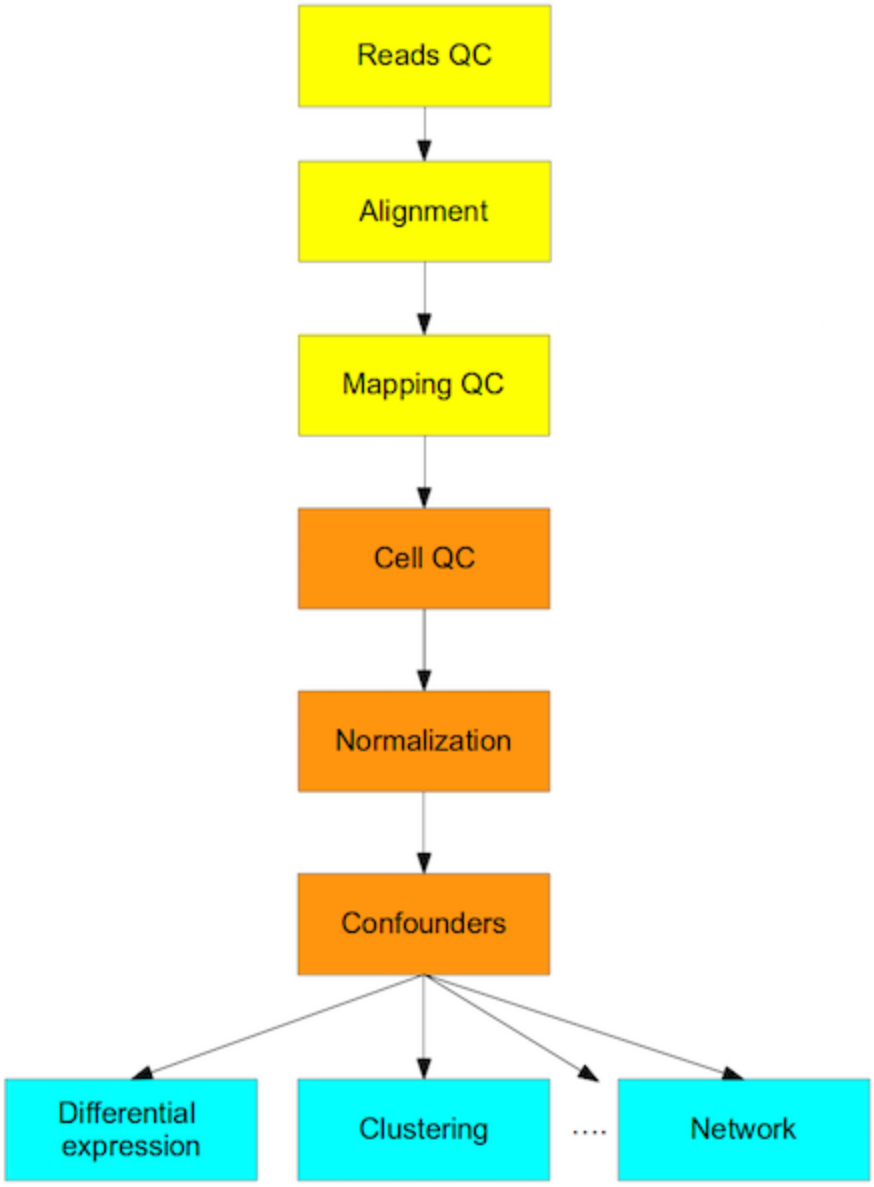
Reads QC
Alignment
Mapping QC
Cell QC
Normalization
Confounders
Differential expression
Clustering
....
Network

Figure 3.2:

Chapter 4

Construction of expression matrix

4.1 Reads QC

The output from a scRNA-seq experiment is a large collection of short cDNA reads. The first step is to ensure that the reads are of high quality. In application to scRNA-seq, this can be performed by using standard bulk RNA-seq QC tools, such as FastQC or Kraken. Currently, there are no specialized tools for scRNA-seq available, so we use FastQC.

Assuming that our reads are in experiment.bam, we run FastQC as

```
$<path_to_fastQC>/fastQC experiment.bam
```

Below is an example of the output from FastQC for a dataset of 125 bp reads. Here, the quality of the reads is overall high, so we can proceed the analysis with confidence.

Additionally, the data can be visualized using the Integrative Genomics Browser (IGV).

4.2 Reads alignment

Once the low-quality reads have been removed, the remaining reads can be mapped to a reference genome. Again, there are no special purpose methods for this, so we can use the STAR or the TopHat aligner.

An example of how to map reads.bam to using STAR hg38 is

```
$<path_to_STAR>/STAR --runThreadN 1 --runMode alignReads
--readFilesIn reads1.fq.gz reads2.fq.gz --readFilesCommand zcat --genomeDir <path>
--parametersFiles FileOfMoreParameters.txt --outFileNamePrefix <outpath>/output
```

Note, if the *spike-ins* are used, the reference sequence should be augmented with the DNA sequence of the *spike-in* molecules before mapping.

Note, when UMIs are used, their barcodes should be removed from every read.

Once we have mapped the reads for each cell to the reference genome, we need to make sure that a sufficient number of reads from each cell could be mapped to the reference genome. In our experience, the fraction of mappable reads is 60-70%. However, this result may vary depending on protocol, read length and settings for the read alignment. As a general rule, we expect all cells to have a similar fraction of mapped reads, so any outliers should be inspected and possibly removed.

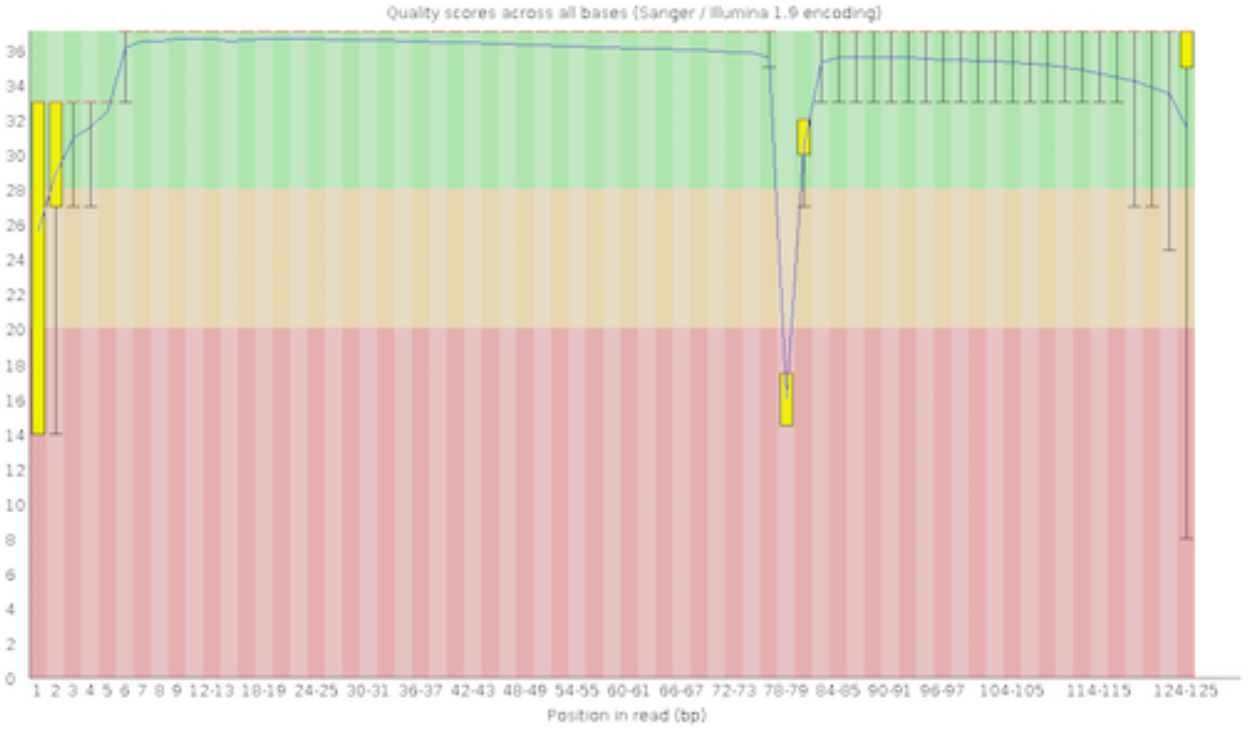

Figure 4.1:

4.3 Alignment example

The histogram below shows the total number of reads mapped to each cell for an scRNA-seq experiment. Each bar represents one cell, and they have been sorted in ascending order by the total number of reads per cell. The three red arrows indicate cells that are outliers in terms of their coverage and they should be removed from further analysis. The two yellow arrows point to cells with a surprisingly large number of unmapped reads. However, we deem the discrepancy as small and we retain the cells for now.

4.4 Mapping QC

After mapping the raw sequencing to the genome we should evaluate the quality of the mapping. There are many ways to measure this including: amount of reads mapping to rRNA/tRNAs, proportion of uniquely mapping reads, reads mapping across splice junctions, read depth along the transcripts. Methods developed for bulk RNA-seq, such as RSeQC, are applicable to single-cell data:

```
python <RSeQCpath>/geneBody_coverage.py -i input.bam -r genome.bed -o output.txt
python <RSeQCpath>/bam_stat.py -i input.bam -r genome.bed -o output.txt
python <RSeQCpath>/split_bam.py -i input.bam -r rRNAmask.bed -o output.txt
```

However the expected results will depend on the experimental protocol, e.g. many scRNA-seq methods use poly-A selection to avoid sequencing rRNAs which results in a 3' bias in the read coverage across the genes (aka gene body coverage). The figure below shows this 3' bias as well as three cells which were outliers and removed from the dataset:

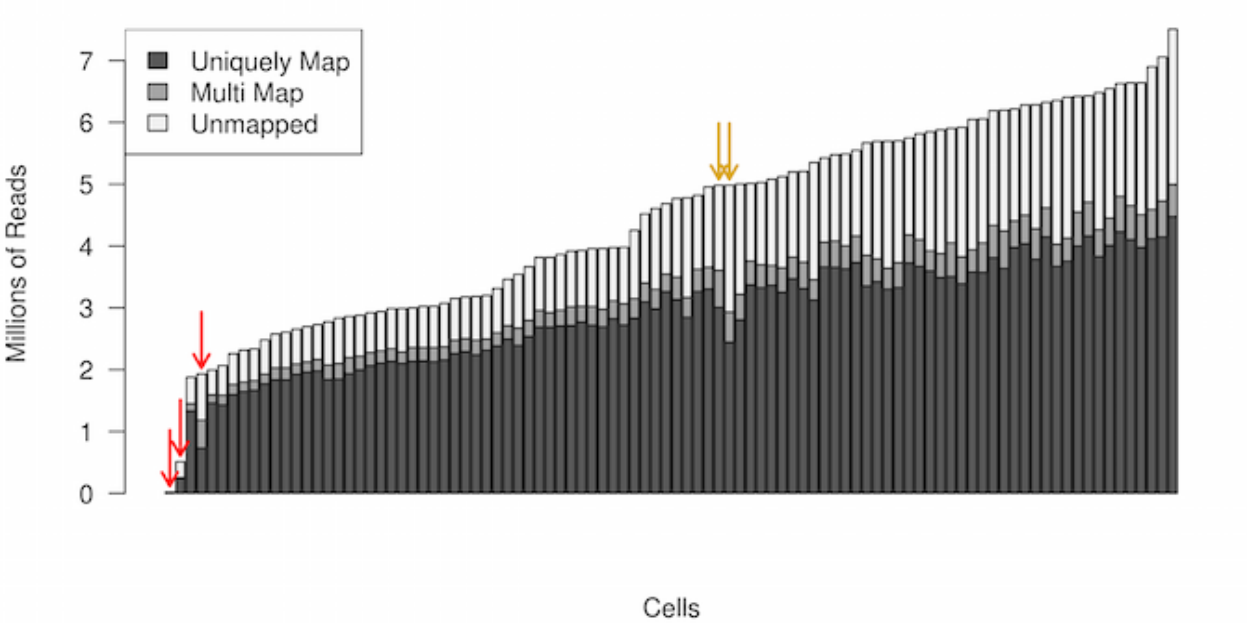

Figure 4.2:

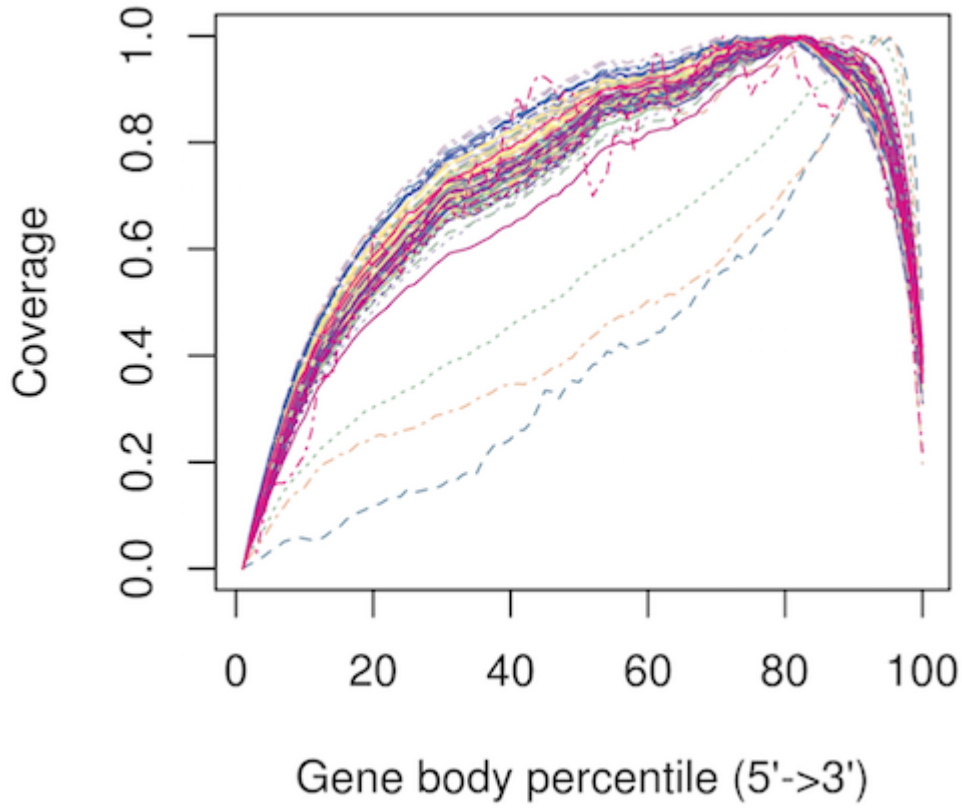

Figure 4.3:

4.5 Reads quantification

The next step is to quantify the expression level of each gene for each cell. For mRNA data, we can use one of the tools which has been developed for bulk RNA-seq data, e.g. HT-seq or FeatureCounts

```
# include multimapping
<featureCounts_path>/featureCounts -O -M -Q 30 -p -a genome.gtf -o outputfile input.bam
# exclude multimapping
<featureCounts_path>/featureCounts -Q 30 -p -a genome.gtf -o outputfile input.bam
```

Note, when UMIs are used, the expression counts can be collapsed by summing the number of unique barcodes associated with all reads mapped to a given gene.

Chapter 5

scater package

5.1 Introduction

scater is a single-cell analysis toolkit for expression with R. This package contains useful tools for the analysis of single-cell gene expression data using the statistical software R. The package places an emphasis on tools for quality control, visualisation and pre-processing of data before further downstream analysis.

scater enables the following:

- Automated computation of QC metrics
- Transcript quantification from read data with pseudo-alignment
- Data format standardisation
- Rich visualisations for exploratory analysis
- Seamless integration into the Bioconductor universe
- Simple normalisation methods

We highly recommend to use scater for any single-cell RNA-seq analysis you will be doing in the future. scater is the basis of the first part of the course and therefore we will spend some time explaining its details.

5.2 scater workflow

5.3 Terminology

(this chapter is taken from the scater vignette)

- The capabilities of scater are built on top of Bioconductor's Biobase.
- In Bioconductor terminology we assay numerous **"features"** for a number of **"samples"**.
- Features, in the context of scater, correspond most commonly to genes or transcripts, but could be any general genomic or transcriptomic regions (e.g. exon) of interest for which we take measurements.
- In the following chapters it may be more intuitive to mentally replace **"feature"** with **"gene"** or **"transcript"** (depending on the context of the study) wherever **"feature"** appears.
- In the scater context, **"samples"** refer to individual cells that we have assayed.

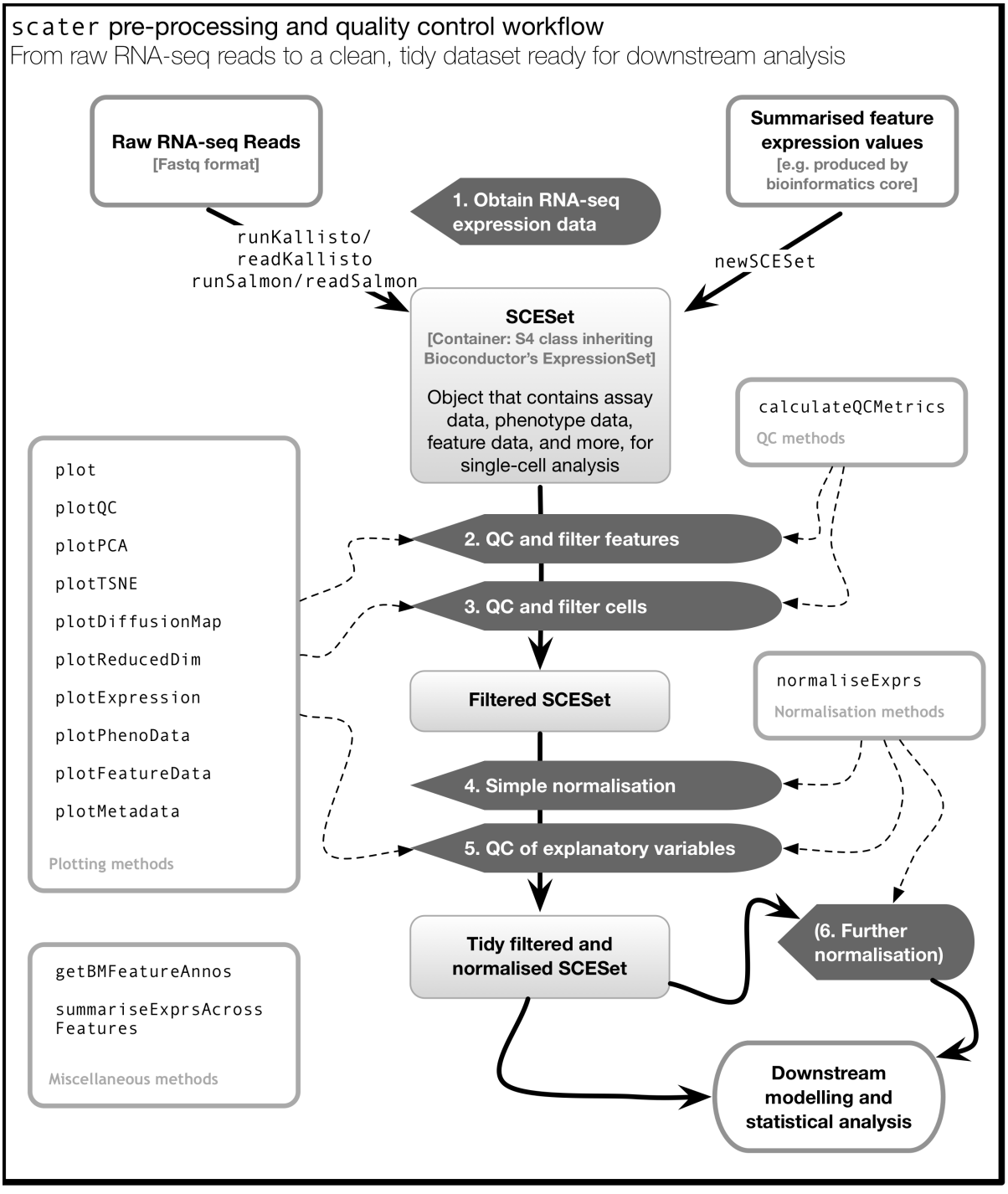
scater pre-processing and quality control workflow
From raw RNA-seq reads to a clean, tidy dataset ready for downstream analysis
Raw RNA-seq Reads
[Fastq format]
Summarised feature expression values
[e.g. produced by bioinformatics core]
1. Obtain RNA-seq expression data
runKallisto/
readKallisto
runSalmon/readSalmon
newSCESet
SCESet
[Container: S4 class inheriting Bioconductor's ExpressionSet]
Object that contains assay data, phenotype data, feature data, and more, for single-cell analysis
calculateQCMetrics
QC methods
plot
plotQC
plotPCA
plotTSNE
plotDiffusionMap
plotReducedDim
plotExpression
plotPhenoData
plotFeatureData
plotMetadata
Plotting methods
2. QC and filter features
3. QC and filter cells
Filtered SCESet
normaliseExprs
Normalisation methods
4. Simple normalisation
5. QC of explanatory variables
Tidy filtered and normalised SCESet
(6. Further normalisation)
getBMFeatureAnnos
summariseExprsAcross Features
Miscellaneous methods
Downstream modelling and statistical analysis

Figure 5.1:

5.4 SCESet class

(this chapter is taken from the scater vignette)

In scater we organise single-cell expression data in objects of the **SCESet** class. The class inherits the Bioconductor ExpressionSet class, which provides a common interface familiar to those who have analyzed microarray experiments with Bioconductor. The class requires three input files:

- **exprs**, a numeric matrix of expression values, where rows are features, and columns are cells
- **phenoData**, an *AnnotatedDataFrame* object, where rows are cells, and columns are cell attributes (such as cell type, culture condition, day captured, etc.)
- **featureData**, an *AnnotatedDataFrame* object, where rows are features (e.g. genes), and columns are feature attributes, such as biotype, gc content, etc.

For more details about other features inherited from Bioconductor's **ExpressionSet** class, type `?ExpressionSet` at the R prompt.

When your data is wrapped in the **SCESet** class, scater will do the dirty job of calculating different properties of the data automatically. You will see how it works in the next chapters.

Chapter 6

Expression QC (UMI)

6.1 Introduction

Once gene expression has been quantified the resulting expression matrix must be examined for poor quality cells which were not detected from QC of the raw reads. Failure to remove low quality cells at this stage will add a large amount of technical noise which will obscure the biological signals of interest in downstream analysis. We will demonstrate some simple filters that can be applied directly to the gene expression matrix with minimal extrinsic information.

Typically, the results are summarized using an **expression matrix**. In the expression matrix, each row represents a gene and each column represents a cell.

To illustrate cell QC, we consider a dataset from lymphoblast and induced pluripotent stem cells generated by John Blischak in Yoav Gilad's lab at the University of Chicago. The experiments were carried out on the Fluidigm C1 platform and to facilitate the quantification both unique molecular identifiers (UMIs) and ERCC *spike-ins* were used. For our purposes you need to download the files `annotation.txt`, `molecules.txt`, and `reads.txt` from here into the `blischak` folder in your working directory. These files are the copies of the original files made on the 15/03/16. We will use these copies for reproducibility purposes.

To illustrate how one can standardise the analysis of scRNA-seq data we will also be using the scater package, described in the previous chapter.

6.2 Blischak dataset

```
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
```

Load the data and annotations:

```
molecules <- read.table("blischak/molecules.txt", sep = "\t")
anno <- read.table("blischak/annotation.txt", sep = "\t", header = TRUE)
```

How does the data look like?

```
knitr::kable(
    head(molecules[ , 1:3]), booktabs = TRUE,
    caption = 'A table of the first 6 rows and 3 columns of the molecules table.'
)
```

Table 6.1: A table of the first 6 rows and 3 columns of the molecules table.

	NA19098.r1.A01	NA19098.r1.A02	NA19098.r1.A03
ENSG00000237683	0	0	0
ENSG00000187634	0	0	0
ENSG00000188976	3	6	1
ENSG00000187961	0	0	0
ENSG00000187583	0	0	0
ENSG00000187642	0	0	0

```
knitr::kable(
    head(anno), booktabs = TRUE,
    caption = 'A table of the first 6 rows of the anno table.'
)
```

Table 6.2: A table of the first 6 rows of the anno table.

individual	replicate	well	batch	sample_id
NA19098	r1	A01	NA19098.r1	NA19098.r1.A01
NA19098	r1	A02	NA19098.r1	NA19098.r1.A02
NA19098	r1	A03	NA19098.r1	NA19098.r1.A03
NA19098	r1	A04	NA19098.r1	NA19098.r1.A04
NA19098	r1	A05	NA19098.r1	NA19098.r1.A05
NA19098	r1	A06	NA19098.r1	NA19098.r1.A06

The data consists of 3 individuals and 3 replicates and therefore has 9 batches in total.

Let's standardise the analysis by using the scater package described above. First, make scater SCESet classes:

```
pheno_data <- new("AnnotatedDataFrame", anno)
rownames(pheno_data) <- pheno_data$sample_id
umi <- scater::newSCESet(
    countData = molecules,
    phenoData = pheno_data
)
```

Remove genes that are not expressed in any cell:

```
keep_feature <- rowSums(is_exprs(umi)) > 0
umi <- umi[keep_feature, ]
```

Define control features (genes) - ERCC spike-ins and mitochondrial genes (provided by the authors):

```
ercc <- featureNames(umi)[grepl("ERCC-", featureNames(umi))]
mt <- c("ENSG00000198899", "ENSG00000198727", "ENSG00000198888",
        "ENSG00000198886", "ENSG00000212907", "ENSG00000198786",
        "ENSG00000198695", "ENSG00000198712", "ENSG00000198804",
        "ENSG00000198763", "ENSG00000228253", "ENSG00000198938",
        "ENSG00000198840")
```

Calculate the quality metrics:

```
umi <- scater::calculateQCMetrics(
    umi,
    feature_controls = list(ERCC = ercc, MT = mt)
)
```

6.3 Gene QC

6.3.1 Gene dropouts

First, one can look at the gene expression frequency versus mean expression level to assess the effects of technical dropout in the dataset.

```
scater::plotQC(umi, type = "exprs-freq-vs-mean")
```

Dropout rate vs mean expression

Interestingly, only less than half of the genes are expressed in more than 50% of the cells.

6.3.2 Gene expression

One can also look at the number of reads consumed by the top 50 expressed genes.

```
scater::plotQC(umi, type = "highest-expression")
```

Number of total counts consumed by the top 50 expressed genes

Not surprisingly, most of the very top expressed genes are either ERCCs or MT genes (Feature controls).

6.4 Gene filtering

We can remove genes whose expression level is considered **"undetectable"**. We define a detectable gene expression level, in which at least 2 cells contain more than 1 read mapping to the gene.

```
filter_genes <- apply(counts(umi), 1, function(x) length(x[x > 1]) >= 2)
fData(umi)$use <- filter_genes
```

```
knitr::kable(
    as.data.frame(table(filter_genes)),
    booktabs = TRUE,
    row.names = FALSE,
    caption = 'The number of genes removed by gene filter (FALSE)'
)
```

Table 6.3: The number of genes removed by gene filter (FALSE)

filter_genes	Freq
FALSE	4512
TRUE	14214

Depending on the cell-type, protocol and sequencing depth, other cut-offs may be appropriate.

6.5 Cell QC

6.5.1 Library size

Next we will look at the total number of RNA molecules detected per sample (if we were using read counts rather than UMI counts this would be total reads). Wells with few reads/molecules are likely to have been broken or failed to capture a cell, thus should be removed.

```
hist(
    umi$total_counts,
    breaks = 100
)
```

Histogram of library sizes for all cells

Exercise

Apply a suitable filter to remove the cells that contain too few molecules. What distribution do you expect that the total number of molecules for each cell should follow?

Answer

```
filter_by_total_counts <- (umi$total_counts > 10000)
```

```
knitr::kable(
    as.data.frame(table(filter_by_total_counts)),
    booktabs = TRUE,
    row.names = FALSE,
    caption = 'The number of cells removed by total counts filter (FALSE)'
)
```

Table 6.4: The number of cells removed by total counts filter (FALSE)

filter_by_total_counts	Freq
FALSE	2
TRUE	862

6.5.2 Detected genes (1)

One of the simplest measures of cell quality is the number of genes that were detected. Cells with few detected genes may have been broken or dead prior to capture.

```
hist(
    umi$total_features,
    breaks = 100
)
```

Histogram of the number of detected genes in all cells

Here we see that most cells have between 5,000-11,000 detected genes, which is normal for high-depth scRNA-seq. However this varies by experimental protocol and sequencing depth with droplet-based methods or lower sequencing-depth detecting fewer genes per cell. The feature to note is the **"heavy tail"** of the left hand side of the distribution. If detection rates were equal across the cells then the distribution should be approximately normal. Thus we remove those cells in the tail of the distribution (<5,000 detected genes).

```
filter_by_expr_features <- (umi$total_features > 5000)
```

```
knitr::kable(
    as.data.frame(table(filter_by_expr_features)),
    booktabs = TRUE,
    row.names = FALSE,
    caption = 'The number of cells removed by total features filter (FALSE)'
)
```

Table 6.5: The number of cells removed by total features filter (FALSE)

filter_by_expr_features	Freq
FALSE	16
TRUE	848

6.5.3 Detected genes (2)

Additionally, we can plot the library size as a function of the number of the detected genes. This will also allow us to find other cell outliers.

```
scater::plotPhenoData(
    umi,
    aes(x = total_features, y = log10(total_counts), colour = batch)
)
```

Library size vs number of detected genes

6.5.4 ERCCs and MTs

Another measures of cell quality is the ration between ERCC *spike-in* RNAs and endogenous RNAs. This can be used to estimate the total amount of RNA in the captured cells. Cells with a high level of *spike-in*

RNAs had low starting amounts of RNA, likely due to the cell being dead or stressed, or the RNA being degraded.

```
scater::plotPhenoData(
    umi,
    aes_string(x = "total_features",
               y = "pct_counts_feature_controls_MT",
               colour = "batch")
)
```

Percentage of counts in MT genes

```
scater::plotPhenoData(
    umi,
    aes_string(x = "total_features",
               y = "pct_counts_feature_controls_ERCC",
               colour = "batch")
)
```

Percentage of counts in ERCCs

This analysis shows that majority of the cells from NA19098.r2 batch have a very high ERCC/Endo ratio. Indeed, it has been shown by the authors that this batch contains cells of smaller size.

Exercise

Create filters for removing batch NA19098.r2 and cells with high expression of mitochondrial genes (>10% of total counts in a cell).

Answer

```
filter_by_ERCC <- umi$batch != "NA19098.r2"
```

```
knitr::kable(
  as.data.frame(table(filter_by_ERCC)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by ERCC filter (FALSE)'
)
```

Table 6.6: The number of cells removed by ERCC filter (FALSE)

filter_by_ERCC	Freq
FALSE	96
TRUE	768

```
filter_by_MT <- umi$pct_counts_feature_controls_MT < 10
```

```
knitr::kable(
  as.data.frame(table(filter_by_MT)),
  booktabs = TRUE,
  row.names = FALSE,
```

```
  caption = 'The number of cells removed by MT filter (FALSE)'
)
```

Table 6.7: The number of cells removed by MT filter (FALSE)

filter_by_MT	Freq
FALSE	31
TRUE	833

6.6 Cell filtering

6.6.1 Default

To filter out outlier cells one could use the defaults provided by the scater package:

- **filter_on_total_features**: filter out cells based on their total_features being (by default) more than 5 median absolute deviations from the median total_features for the dataset
- **filter_on_total_counts**: filter out cells based on their log10-total_counts being (by default) more than 5 median absolute deviations from the median log10-total_counts for the dataset?
- **filter_on_pct_counts_feature_controls**: filter out cells on the basis that the percentage of counts from feature controls is higher than a defined threhold (default is 80%)?
- **is_cell_control**: filter out cells that were defined as controls

```
umi$use_default <- (
    # remove cells with unusual numbers of genes
    !umi$filter_on_total_features &
    # sufficient molecules counted
    !umi$filter_on_total_counts &
    # sufficient endogenous RNA
    !umi$filter_on_pct_counts_feature_controls_ERCC &
    # remove cells with unusual number of reads in MT genes
    !umi$filter_on_pct_counts_feature_controls_MT &
    # controls shouldn't be used in downstream analysis
    !umi$is_cell_control
)
```

```
knitr::kable(
  as.data.frame(table(umi$use_default)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by default filter (FALSE)'
)
```

Table 6.8: The number of cells removed by default filter (FALSE)

Var1	Freq
FALSE	6
TRUE	858

This filtering can be performed even without looking at the plots above.

6.6.2 Automatic

Another option available in **scater** is to conduct PCA on a set of QC metrics. The advantage of doing this is that the QC metrics focus on technical aspects of the libraries that are likely to distinguish problematics cells. Automatic outlier detection on PCA plots using QC metrics is available to help identify potentially problematic cells.

By default, the following metrics are used for PCA-based outlier detection:

- **pct_counts_top_100_features**
- **total_features**
- **pct_counts_feature_controls**
- **n_detected_feature_controls**
- **log10_counts_endogenous_features**
- **log10_counts_feature_controls**

A particular set of variables to be used can be specified with the selected_variables argument as shown in the example below.

```
umi <-
scater::plotPCA(umi,
                size_by = "total_features",
                shape_by = "filter_on_total_features",
                pca_data_input = "pdata",
                detect_outliers = TRUE,
                return_SCESet = TRUE)
```

```
## The following cells/samples are detected as outliers:
## NA19098.r2.A01
## NA19098.r2.A02
## NA19098.r2.A06
## NA19098.r2.A09
## NA19098.r2.A10
## NA19098.r2.A12
## NA19098.r2.B01
## NA19098.r2.B03
## NA19098.r2.B04
## NA19098.r2.B05
## NA19098.r2.B07
## NA19098.r2.B11
## NA19098.r2.B12
## NA19098.r2.C01
## NA19098.r2.C02
## NA19098.r2.C03
## NA19098.r2.C04
## NA19098.r2.C05
## NA19098.r2.C06
## NA19098.r2.C07
## NA19098.r2.C08
## NA19098.r2.C09
## NA19098.r2.C10
```

```
## NA19098.r2.C11
## NA19098.r2.C12
## NA19098.r2.D01
## NA19098.r2.D02
## NA19098.r2.D03
## NA19098.r2.D04
## NA19098.r2.D07
## NA19098.r2.D08
## NA19098.r2.D09
## NA19098.r2.D10
## NA19098.r2.D12
## NA19098.r2.E01
## NA19098.r2.E02
## NA19098.r2.E03
## NA19098.r2.E04
## NA19098.r2.E05
## NA19098.r2.E06
## NA19098.r2.E07
## NA19098.r2.E12
## NA19098.r2.F01
## NA19098.r2.F02
## NA19098.r2.F07
## NA19098.r2.F08
## NA19098.r2.F09
## NA19098.r2.F10
## NA19098.r2.F11
## NA19098.r2.F12
## NA19098.r2.G01
## NA19098.r2.G02
## NA19098.r2.G03
## NA19098.r2.G05
## NA19098.r2.G06
## NA19098.r2.G08
## NA19098.r2.G09
## NA19098.r2.G10
## NA19098.r2.G11
## NA19098.r2.H01
## NA19098.r2.H02
## NA19098.r2.H03
## NA19098.r2.H04
## NA19098.r2.H05
## NA19098.r2.H06
## NA19098.r2.H07
## NA19098.r2.H08
## NA19098.r2.H10
## NA19098.r2.H12
## NA19101.r3.A02
## NA19101.r3.C12
## NA19101.r3.D01
## NA19101.r3.E08
## Variables with highest loadings for PC1 and PC2:
##
##                                          PC1         PC2
## ---------------------------------  -----------  ----------
```

```
## pct_counts_top_100_features          0.4771343   0.3009332
## pct_counts_feature_controls          0.4735839   0.3309562
## n_detected_feature_controls          0.1332811   0.5367629
## log10_counts_feature_controls       -0.1427373   0.5911762
## total_features                      -0.5016681   0.2936705
## log10_counts_endogenous_features    -0.5081855   0.2757918
```

PCA plot used for automatic detection of cell outliers

```
knitr::kable(
  as.data.frame(table(umi$outlier)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by automatic filter (FALSE)'
)
```

Table 6.9: The number of cells removed by automatic filter (FALSE)

Var1	Freq
FALSE	791
TRUE	73

6.6.3 Manual

However, since we performed a more detailed analysis, visualized different features and defined our own filters we can use them instead of the default ones:

```
umi$use <- (
    # sufficient features (genes)
    filter_by_expr_features &
    # sufficient molecules counted
    filter_by_total_counts &
    # sufficient endogenous RNA
    filter_by_ERCC &
    # remove cells with unusual number of reads in MT genes
    filter_by_MT
)
```

```
knitr::kable(
  as.data.frame(table(umi$use)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by manual filter (FALSE)'
)
```

Table 6.10: The number of cells removed by manual filter (FALSE)

Var1	Freq
FALSE	141

Var1	Freq
TRUE	723

6.7 Compare filterings

Exercise

Compare the default, automatic and manual cell filters. Plot a Venn diagram of the outlier cells from these filterings.

Hint: Use `limma::vennCounts` and `limma::vennDiagram` functions from the limma package to make a Venn diagram.

Answer

```
def <- colnames(umi)[!umi$use_default]
auto <- colnames(umi)[umi$outlier]
man <- colnames(umi)[!umi$use]
venn.diag <- limma::vennCounts(cbind(colnames(umi) %in% def,
                                     colnames(umi) %in% auto,
                                     colnames(umi) %in% man))
limma::vennDiagram(venn.diag,
                   names = c("Default", "Automatic", "Manual"),
                   circle.col = c("magenta", "blue", "green"))
```

Comparison of the default, automatic and manual cell filters

6.8 Save the data

Dimensions of the QCed dataset (do not forget about the gene filter we defined above):

```
dim(umi[fData(umi)$use, pData(umi)$use])
```

```
## Features  Samples
##    14214      723
```

Save the data:

```
saveRDS(umi, file = "blischak/umi.rds")
```

6.9 Exercise

Perform exactly the same QC analysis with read counts of the same Blischak data. Use `blischak/reads.txt` file to load the reads. Once you have finished please compare your results to ours (next chapter).

Chapter 7

Expression QC (Reads)

Table 7.1: A table of the first 6 rows and 3 columns of the molecules table.

	NA19098.r1.A01	NA19098.r1.A02	NA19098.r1.A03
ENSG00000237683	0	0	0
ENSG00000187634	0	0	0
ENSG00000188976	57	140	1
ENSG00000187961	0	0	0
ENSG00000187583	0	0	0
ENSG00000187642	0	0	0

Table 7.2: A table of the first 6 rows of the anno table.

individual	replicate	well	batch	sample_id
NA19098	r1	A01	NA19098.r1	NA19098.r1.A01
NA19098	r1	A02	NA19098.r1	NA19098.r1.A02
NA19098	r1	A03	NA19098.r1	NA19098.r1.A03
NA19098	r1	A04	NA19098.r1	NA19098.r1.A04
NA19098	r1	A05	NA19098.r1	NA19098.r1.A05
NA19098	r1	A06	NA19098.r1	NA19098.r1.A06

Dropout rate vs mean expression

Number of total counts consumed by the top 50 expressed genes

Table 7.3: The number of genes removed by gene filter (FALSE)

filter_genes	Freq
FALSE	2445
TRUE	16281

Histogram of library sizes for all cells

Table 7.4: The number of cells removed by total counts filter (FALSE)

filter_by_total_counts	Freq
FALSE	180
TRUE	684

Histogram of the number of detected genes in all cells

Table 7.5: The number of cells removed by total features filter (FALSE)

filter_by_expr_features	Freq
FALSE	16
TRUE	848

Library size vs number of detected genes

Percentage of counts in MT genes

Percentage of counts in ERCCs

Table 7.6: The number of cells removed by ERCC filter (FALSE)

filter_by_ERCC	Freq
FALSE	103
TRUE	761

Table 7.7: The number of cells removed by MT filter (FALSE)

filter_by_MT	Freq
FALSE	18
TRUE	846

Table 7.8: The number of cells removed by default filter (FALSE)

Var1	Freq
FALSE	37
TRUE	827

```
## The following cells/samples are detected as outliers:
## NA19098.r1.B10
## NA19098.r1.D07
## NA19098.r1.E04
## NA19098.r1.F06
## NA19098.r1.H08
## NA19098.r1.H09
```

```
## NA19098.r2.A01
## NA19098.r2.A06
## NA19098.r2.A09
## NA19098.r2.A12
## NA19098.r2.B01
## NA19098.r2.B11
## NA19098.r2.B12
## NA19098.r2.C04
## NA19098.r2.C09
## NA19098.r2.D02
## NA19098.r2.D03
## NA19098.r2.D09
## NA19098.r2.E04
## NA19098.r2.E07
## NA19098.r2.F01
## NA19098.r2.F11
## NA19098.r2.G01
## NA19098.r2.G05
## NA19098.r2.G10
## NA19098.r2.H01
## NA19098.r2.H07
## NA19098.r2.H08
## NA19098.r2.H12
## NA19098.r3.A05
## NA19098.r3.A07
## NA19098.r3.B02
## NA19098.r3.C07
## NA19098.r3.E05
## NA19098.r3.E08
## NA19098.r3.E09
## NA19098.r3.F11
## NA19098.r3.F12
## NA19098.r3.G02
## NA19098.r3.G03
## NA19098.r3.G04
## NA19098.r3.G11
## NA19098.r3.G12
## NA19098.r3.H08
## NA19101.r1.A01
## NA19101.r1.A12
## NA19101.r1.B01
## NA19101.r1.B06
## NA19101.r1.E09
## NA19101.r1.E11
## NA19101.r1.F05
## NA19101.r1.F10
## NA19101.r1.G01
## NA19101.r1.G06
## NA19101.r1.H04
## NA19101.r1.H09
## NA19101.r2.A03
## NA19101.r2.C10
## NA19101.r2.E05
## NA19101.r2.F02
```

```
## NA19101.r2.H04
## NA19101.r2.H10
## NA19101.r3.A02
## NA19101.r3.A03
## NA19101.r3.A05
## NA19101.r3.A09
## NA19101.r3.B05
## NA19101.r3.C01
## NA19101.r3.C09
## NA19101.r3.C12
## NA19101.r3.D01
## NA19101.r3.D04
## NA19101.r3.D07
## NA19101.r3.D09
## NA19101.r3.E08
## NA19101.r3.F09
## NA19101.r3.G09
## NA19101.r3.H01
## NA19101.r3.H03
## NA19101.r3.H07
## NA19101.r3.H09
## NA19239.r1.F05
## NA19239.r1.G05
## NA19239.r2.B01
## NA19239.r2.B03
## NA19239.r2.B10
## NA19239.r2.B11
## NA19239.r2.C03
## NA19239.r2.C06
## NA19239.r2.C08
## NA19239.r2.D07
## NA19239.r2.D09
## NA19239.r2.E09
## NA19239.r2.F04
## NA19239.r2.F06
## NA19239.r2.F07
## NA19239.r2.F12
## NA19239.r2.G03
## NA19239.r2.G08
## NA19239.r2.H02
## NA19239.r2.H03
## NA19239.r2.H07
## NA19239.r3.A01
## NA19239.r3.B09
## NA19239.r3.C04
## NA19239.r3.C07
## NA19239.r3.E01
## NA19239.r3.E03
## NA19239.r3.E12
## NA19239.r3.H02
## NA19239.r3.H10
## Variables with highest loadings for PC1 and PC2:
##
##                                          PC1         PC2
```

```
## --------------------------------  -----------  ----------
## pct_counts_feature_controls          0.5057646   0.2473134
## pct_counts_top_100_features          0.4888852   0.2277068
## n_detected_feature_controls          0.0231277   0.6235516
## log10_counts_feature_controls       -0.1226860   0.6576822
## total_features                      -0.4655518   0.2219694
## log10_counts_endogenous_features    -0.5223679   0.1278782
```

PCA plot used for automatic detection of cell outliers

Table 7.9: The number of cells removed by automatic filter (FALSE)

Var1	Freq
FALSE	753
TRUE	111

Table 7.10: The number of cells removed by manual filter (FALSE)

Var1	Freq
FALSE	254
TRUE	610

Comparison of the default, automatic and manual cell filters

```
## Features  Samples
##    16281      610
```

Chapter 8

Expression overview

8.1 Introduction

Here we will continue to work with the filtered **blischak** dataset produced in the previous chapter. We will look at what happened to the expression matrix after the quality control step.

```
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
umi <- readRDS("blischak/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

8.2 Top 500 genes

With scater one can look at the proportion of reads accounted by the top 500 expressed genes.

8.2.1 Before QC

```
scater::plot(umi[endog_genes, ],
             block1 = "individual",
             block2 = "replicate",
             colour_by = "use",
             exprs_values = "counts")
```

Proportion of reads accounted by the top 500

8.2.2 After QC

```
scater::plot(umi.qc[endog_genes, ],
             block1 = "individual",
             block2 = "replicate",
             colour_by = "use",
             exprs_values = "counts")
```

Proportion of reads accounted by the top 500

8.3 PCA plot

The easiest thing to overview the data is to transform it using the principal component analysis and then visualize the first two principal components. Again the scater package provides several very useful functions to make this analysis straightforward.

8.3.1 Before QC

```
scater::plotPCA(umi[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                exprs_values = "counts")
```

PCA plot of the blischak data

8.3.2 After QC

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                exprs_values = "counts")
```

PCA plot of the blischak data

8.4 Diffusion map

Another way of representing the data is a diffusion map. It is very useful if the cells represent a continuous process (e.g. development).

8.4.1 Before QC

```
scater::plotDiffusionMap(umi[endog_genes, ],
                         colour_by = "batch",
                         size_by = "total_features",
                         exprs_values = "counts",
                         rand_seed = 123456)
```

Diffusion map of the blischak data

8.4.2 After QC

```
scater::plotDiffusionMap(umi.qc[endog_genes, ],
                         colour_by = "batch",
                         size_by = "total_features",
                         exprs_values = "counts",
                         rand_seed = 123456)
```

Diffusion map of the blischak data

8.5 tSNE map

Another way of representing the data is a tSNE map.

8.5.1 Before QC

```
scater::plotTSNE(umi[endog_genes, ],
                 colour_by = "batch",
                 size_by = "total_features",
                 exprs_values = "counts",
                 rand_seed = 123456)
```

tSNE map of the blischak data

8.5.2 After QC

```
scater::plotTSNE(umi.qc[endog_genes, ],
                 colour_by = "batch",
                 size_by = "total_features",
                 exprs_values = "counts",
                 rand_seed = 123456)
```

tSNE map of the blischak data

8.6 Exercise

Perform the same analysis with read counts of the Blischak data. Use `blischak/reads.rds` file to load the reads SCESet object. Once you have finished please compare your results to ours (next chapter).

Chapter 9

Expression overview (Reads)

Proportion of reads accounted by the top 500

Proportion of reads accounted by the top 500

PCA plot of the blischak data

PCA plot of the blischak data

Diffusion map of the blischak data

Diffusion map of the blischak data

tSNE map of the blischak data

tSNE map of the blischak data

Chapter 10

Confounding factors

10.1 Introduction

There are a very large number of potential confounders, artifacts and biases in sc-RNA-seq data One of the main challenges in analyzing scRNA-seq data stems from the fact that it is difficult to carry out a true technical replicate (why?) to distinguish biological and technical variability. Exploring the effects of such variabilities (both those recorded during the experiment and computed QC metrics) is crucial for appropriate modeling of the data.

The scater package provides a set of methods specifically for quality control of experimental and explanatory variables.

Here we will continue to work with the Blischak data that was used in the previous chapter.

```
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
umi <- readRDS("blischak/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

The `umi.qc` dataset contains filtered cells and genes. Our next step is to explore technical drivers of variability in the data to inform data normalisation before downstream analysis.

10.2 Correlations with PCs

Let's first look again at the PCA plot of the QCed dataset:

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                exprs_values = "counts")
```

PCA plot of the blischak data

scater allows one to identify principle components that correlate with experimental and QC variables of interest (it ranks principle components by R^2 from a linear model regressing PC value against the variable of interest).

Let's test whether some of the variables correlate with any of the PCs.

10.2.1 Detected genes

```
scater::plotQC(umi.qc[endog_genes, ],
               type = "find-pcs",
               variable = "total_features")
```

PC correlation with the number of detected genes

Indeed, we can see that PC1 can be completely explained by the number of the detected genes. In fact, it was also visible on the PCA plot above. This is a well-known issue in scRNA-seq and was described here.

10.2.2 Control ERCCs

```
scater::plotQC(umi.qc[endog_genes, ],
               type = "find-pcs",
               variable = "pct_counts_feature_controls_ERCC")
```

PC correlation with the percentage of reads in ERCCs

10.2.3 Control MTs

```
scater::plotQC(umi.qc[endog_genes, ],
               type = "find-pcs",
               variable = "pct_counts_feature_controls_MT")
```

PC correlation with the percentage of reads in MT genes

10.3 Explanatory variables

scater can also compute the marginal R^2 for each variable when fitting a linear model regressing expression values for each gene against just that variable, and display a density plot of the gene-wise marginal R^2 values for the variables.

```
scater::plotQC(umi.qc[endog_genes, ],
               type = "expl",
               exprs_values = "counts",
               variables = c("total_features",
                             "total_counts",
                             "batch",
                             "individual",
                             "pct_counts_feature_controls_ERCC",
                             "pct_counts_feature_controls_MT"))
```

Explanatory variables

This analysis indicates that the number of detected genes (again) and also the sequencing depth (number of counts) have substantial explanatory power for many genes, so these variables are good candidates for conditioning out in a normalisation step, or including in downstream statistical models. Expression of ERCCs also appears to be an important explanatory variable.

10.4 Other confounders

In addition to correcting for batch, there are other factors that one may want to compensate for. As with batch correction, these adjustments require extrinsic information. One popular method is scLVM which allows you to identify and subtract the effect from processes such as cell-cycle or apoptosis.

10.5 Exercise

Perform the same analysis with read counts of the Blischak data. Use `blischak/reads.rds` file to load the reads SCESet object. Once you have finished please compare your results to ours (next chapter).

Chapter 11

Confounding factors (Reads)

PCA plot of the blischak data

PC correlation with the number of detected genes

PC correlation with the percentage of reads in ERCCs

PC correlation with the percentage of reads in MT genes

Chapter 12

Normalization for library size

12.1 Introduction

In the previous chapter we identified important confounding factors and explanatory variables. scater allows one to account for these variables in subsequent statistical models or to condition them out using `normaliseExprs()`, if so desired. This can be done by providing a design matrix to `normaliseExprs()`. We are not covering this topic here, but you can try to do it yourself as an exercise.

Instead we will explore how simple size-factor normalisations correcting for library size can remove the effects of some of the confounders and explanatory variables.

12.2 Library size

Library sizes vary because of the various reasons: * scRNA-seq data is often sequenced on highly multiplexed platforms the total reads which are derived from each cell may differ substantially. * Protocols may differ in terms of their coverage of each transcript, their bias based on the average content of **A/T** nucleotides, or their ability to capture short transcripts.

Ideally, we would like to compensate for all of these differences and biases when comparing data from two different experiments.

Many methods to correct for library size have been developped for bulk RNA-seq and can be equally applied to scRNA-seq (eg. UQ, SF, CPM, RPKM, FPKM, TPM). Some quantification methods (eg. Cufflinks, RSEM) incorporated library size when determining gene expression estimates thus do not require this normalization.

We will continue to work with the Blischak data that was used in the previous chapter and show how scater makes it easy to perform different types of size-factor normalizations.

```
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
umi <- readRDS("blischak/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

12.3 Normalisations

The simplest way to normalize this data is to convert it to counts per million (**CPM**) by dividing each column by its total then multiplying by 1,000,000. Note that spike-ins should be excluded from the calculation of

total expression in order to correct for total cell RNA content, therefore we will only use endogenous genes. scater performs this normalisation by default, you can control it by changing `exprs_values` parameter to `"exprs"`.

Another method is called **TMM** is the weighted trimmed mean of M-values (to the reference) proposed by Robinson and Oshlack (2010), where the weights are from the delta method on Binomial data.

Another very popular method **RLE** is the scaling factor method proposed by Anders and Huber (2010). We call it "relative log expression", as median library is calculated from the geometric mean of all columns and the median ratio of each sample to the median library is taken as the scale factor.

A similar to **RLE** the **upperquartile** is the upper-quartile normalization method of Bullard et al (2010), in which the scale factors are calculated from the 75% quantile of the counts for each library, after removing genes which are zero in all libraries. This idea is generalized here to allow scaling by any quantile of the distributions.

Let's compare all these methods.

12.3.1 Raw

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "counts")
```

PCA plot of the blischak data

12.3.2 CPM

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "cpm")
```

PCA plot of the blischak data after CPM normalisation

12.3.3 log2(CPM)

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "exprs")
```

PCA plot of the blischak data after log2(CPM) normalisation

12.3.4 TMM

```
umi.qc <-
    scater::normaliseExprs(umi.qc,
                           method = "TMM",
                           feature_set = endog_genes,
                           lib.size = rep(1, ncol(umi.qc)))
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "norm_counts")
```

PCA plot of the blischak data after TMM normalisation

12.3.5 RLE

```
umi.qc <-
    scater::normaliseExprs(umi.qc,
                           method = "RLE",
                           feature_set = endog_genes,
                           lib.size = rep(1, ncol(umi.qc)))
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "norm_counts")
```

PCA plot of the blischak data after RLE normalisation

12.3.6 Upperquantile

```
umi.qc <-
    scater::normaliseExprs(umi.qc,
                           method = "upperquartile",
                           feature_set = endog_genes,
                           p = 0.99,
                           lib.size = rep(1, ncol(umi.qc)))
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "norm_counts")
```

PCA plot of the blischak data after UQ normalisation

12.4 Interpretation

Clearly, only the CPM normalisation has reduced the effect of the number of detected genes and separated cells by individuals (though very weakly).

12.5 Other methods

Some methods combine library size and fragment/gene length normalization such as:

- **RPKM** - Reads Per Kilobase Million (for single-end sequencing)
- **FPKM** - Fragments Per Kilobase Million (same as **RPKM** but for paired-end sequencing, makes sure that paired ends mapped to the same fragment are not counted twice)
- **TPM** - Transcripts Per Kilobase Million (same as **RPKM**, but the order of normalizations is reversed - length first and sequencing depth second)

These methods are not applicable to our dataset since the end of the transcript which contains the UMI was preferentially sequenced. Furthermore in general these should only be calculated using appropriate quantification software from aligned BAM files not from read counts since often only a portion of the entire gene/transcript is sequenced, not the entire length.

However, here we show how these normalisations can be calculated using scater. First, we need to find the effective transcript length in Kilobases. However, our dataset containes only gene IDs, therefore we will be using the gene lengths instead of transcripts. scater uses the biomaRt package, which allows one to annotate genes by other attributes:

```
umi.qc <-
    scater::getBMFeatureAnnos(umi.qc,
                              filters = "ensembl_gene_id",
                              attributes = c("ensembl_gene_id",
                                             "hgnc_symbol",
                                             "chromosome_name",
                                             "start_position",
                                             "end_position"),
                              feature_symbol = "hgnc_symbol",
                              feature_id = "ensembl_gene_id",
                              biomart = "ENSEMBL_MART_ENSEMBL",
                              dataset = "hsapiens_gene_ensembl",
                              host = "www.ensembl.org")

# If you have mouse data, change the arguments based on this example:
# scater::getBMFeatureAnnos(object,
#                           filters = "ensembl_transcript_id",
#                           attributes = c("ensembl_transcript_id",
#                                          "ensembl_gene_id", "mgi_symbol",
#                                          "chromosome_name",
#                                          "transcript_biotype",
#                                          "transcript_start",
#                                          "transcript_end",
#                                          "transcript_count"),
#                           feature_symbol = "mgi_symbol",
#                           feature_id = "ensembl_gene_id",
#                           biomart = "ENSEMBL_MART_ENSEMBL",
```

```
#                             dataset = "mmusculus_gene_ensembl",
#                             host = "www.ensembl.org")
```

Some of the genes were not annotated, therefore we filter them out:

```
umi.qc.ann <-
    umi.qc[!is.na(fData(umi.qc)$ensembl_gene_id), ]
```

Now we compute the effective gene length in Kilobases by using the `end_position` and `start_position` fields:

```
eff_length <- abs(fData(umi.qc.ann)$end_position -
                      fData(umi.qc.ann)$start_position)/1000
```

Now we are ready to perform the normalisations:

```
tpm(umi.qc.ann) <-
    calculateTPM(
        umi.qc.ann,
        eff_length
    )
fpkm(umi.qc.ann) <-
    calculateFPKM(
        umi.qc.ann,
        eff_length
    )
```

Plot the results as a PCA plot:

```
scater::plotPCA(umi.qc.ann,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "fpkm")
```

PCA plot of the blischak data after FPKM normalisation

TPM normalisation produce a zero-matrix, we are not sure why, it maybe a bug in scater.

12.6 Visualize genes

Now after the normalisation we are ready to visualise the gene expression:

12.6.1 Raw

```
scater::plotExpression(umi.qc.ann,
                       rownames(umi.qc.ann)[1:6],
                       x = "individual",
                       exprs_values = "counts",
                       colour = "batch")
```

Expression of the first 6 genes of the blischak data

12.6.2 CPM

```
scater::plotExpression(umi.qc.ann,
                       rownames(umi.qc.ann)[1:6],
                       x = "individual",
                       exprs_values = "cpm",
                       colour = "batch")
```

Expression of the first 6 genes of the blischak data after the CPM normalisation

12.6.3 log2(CPM)

```
scater::plotExpression(umi.qc.ann,
                       rownames(umi.qc.ann)[1:6],
                       x = "individual",
                       exprs_values = "exprs",
                       colour = "batch")
```

Expression of the first 6 genes of the blischak data after the log2(CPM) normalisation

12.6.4 Upperquantile

```
scater::plotExpression(umi.qc.ann,
                       rownames(umi.qc.ann)[1:6],
                       x = "individual",
                       exprs_values = "norm_counts",
                       colour = "batch")
```

Expression of the first 6 genes of the blischak data after the UQ normalisation

12.6.5 FPKM

```
scater::plotExpression(umi.qc.ann,
                       rownames(umi.qc.ann)[1:6],
                       x = "individual",
                       exprs_values = "fpkm",
                       colour = "batch")
```

Expression of the first 6 genes of the blischak data after the FPKM normalisation

12.6.6 TPM

```
scater::plotExpression(umi.qc.ann,
                       rownames(umi.qc.ann)[1:6],
                       x = "individual",
                       exprs_values = "tpm",
                       colour = "batch")
```

Expression of the first 6 genes of the blischak data after the TPM normalisation

12.7 Exercise

Perform the same analysis with read counts of the Blischak data. Use `blischak/reads.rds` file to load the reads SCESet object. Once you have finished please compare your results to ours (next chapter).

Chapter 13

Normalization for library size (Reads)

PCA plot of the blischak data

PCA plot of the blischak data after CPM normalisation

PCA plot of the blischak data after log2(CPM) normalisation

PCA plot of the blischak data after TMM normalisation

PCA plot of the blischak data after RLE normalisation

PCA plot of the blischak data after UQ normalisation

PCA plot of the blischak data after FPKM normalisation

Expression of the first 6 genes of the blischak data

Expression of the first 6 genes of the blischak data after the CPM normalisation

Expression of the first 6 genes of the blischak data after the log2(CPM) normalisation

Expression of the first 6 genes of the blischak data after the UQ normalisation

Expression of the first 6 genes of the blischak data after the FPKM normalisation

Expression of the first 6 genes of the blischak data after the TPM normalisation

Chapter 14

Remove confounders using controls

14.1 Introduction

In the previous chapter we saw that the library size normalisation is able to remove major confounders. Let us see whether further use of the controls (ERCC and MT) can provide us with more insights into the data.

Several methods (eg. BASiCS, scLVM, RUV) have been developed for normalisation using ERCCs using different noise models and different fitting procedures.

We will demonstrate some of the methods starting from the simplest one proposed by Brennecke et al., which identifies genes with significant variation above technical noise (ERCCs).

```
library(scRNA.seq.funcs)
library(RUVSeq)
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
umi <- readRDS("blischak/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

14.2 Brennecke method

To use the method, we first normalize for library size then calculate the mean and the square coefficient of variation (variation divided by the squared mean expression). A curve is fit for the relationship between these two variables for the ERCC spike-in (subject to just technical variation) then a chi-square test is used to find genes significantly above the curve. This has been provided for you as the Brenneck_getVariableGenes(counts, spikes) function.

```
umi.qc <-
    scater::normaliseExprs(umi.qc,
                           method = "RLE",
                           feature_set = endog_genes,
                           lib.size = rep(1, ncol(umi.qc)))
erccs <- grep("ERCC-", rownames(assayData(umi.qc)$norm_counts))
highly.var.genes <- scRNA.seq.funcs::Brennecke_getVariableGenes(
            assayData(umi.qc)$norm_counts,
            erccs
)
```

```
## Loading required package: statmod
```

```
## Warning in scRNA.seq.funcs::Brennecke_getVariableGenes(assayData(umi.qc)
## $norm_counts, : Only 21 spike-ins to be used in fitting, may result in poor
## fit.
```

Results of using the Brennecke method on the Blischak dataset

In the figure above blue points are the ERCC spike-ins. The red curve is the fitted technical noise model and the dashed line is the 95% CI. Pink dots are the genes with significant biological variability after multiple-testing correction. Since our dataset is relatively homogeneous only 148 genes are identified as significantly variable.

14.3 Remove Unwanted Variation

Factors contributing to technical noise frequently appear as "batch effects" where cells processed on different days or by different technicians systematically vary from one another. Removing technical noise and correcting for batch effects can frequently be performed using the same tool or slight variants on it. We will be considering the Remove Unwanted Variation (RUV) method which uses singular value decomposition (similar to PCA) on subsets of the dataset which should be invariant (e.g. ERCC spike-ins). Then the method removes the identified unwanted factors.

```
assayData(umi.qc)$ruv_counts <- RUVSeq::RUVg(
    round(assayData(umi.qc)$norm_counts),
    erccs,
    k = 1)$normalizedCounts
```

14.4 Effectiveness 1

We evaluate the effectiveness of the normalization by inspecting the PCA plot where shape corresponds the technical replicate and colour corresponds to different biological samples (individuals from whom the iPSC lines where derived). Separation of biological samples and interspersed batches indicates that technical variation has been removed.

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "norm_counts")
```

PCA plot of the blischak data after RLE normalisation

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "ruv_counts")
```

PCA plot of the blischak data after RLE and RUV normalisations

14.5 Effectiveness 2

We can also examine the relative log expression (RLE) across cells to confirm technical noise has been removed from the dataset.

```
boxplot(list(scRNA.seq.funcs::calc_cell_RLE(assayData(umi.qc)$norm_counts),
             scRNA.seq.funcs::calc_cell_RLE(assayData(umi.qc)$ruv_counts)))
```

Comparison of the relative log expression of the blischak data before and after the RUV normalisation

14.6 Exercise

Perform the same analysis with read counts of the Blischak data. Use `blischak/reads.rds` file to load the reads SCESet object. Once you have finished please compare your results to ours (next chapter). Additionally, experiment with other combinations of normalizations and compare the results.

Chapter 15

Remove confounders using controls (Reads)

```
## Loading required package: statmod
```

```
## Warning in scRNA.seq.funcs::Brennecke_getVariableGenes(assayData(reads.qc)
## $norm_counts, : Only 17 spike-ins to be used in fitting, may result in poor
## fit.
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 7 x values <= 0 omitted
## from logarithmic plot
```

Results of using the Brennecke method on the Blischak dataset

PCA plot of the blischak data after RLE normalisation

PCA plot of the blischak data after RLE and RUV normalisations

Comparison of the relative log expression of the blischak data before and after the RUV normalisation

Bibliography