scone_dshayler.Rmd

Kevin Stachelek

1/24/2018

```
# scone_stats <- read.table(scone_stats_path, header = FALSE, sep = "\t",
#   col.names = paste0("V",seq_len(77)), fill = TRUE)

library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following object is masked from 'package:matrixStats':
##
##     count
## The following objects are masked from 'package:Biobase':
##
##     combine, exprs
## The following objects are masked from 'package:GenomicRanges':
##
##     intersect, setdiff, union
## The following object is masked from 'package:GenomeInfoDb':
##
##     intersect
## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union
## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union
## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union
## The following object is masked from 'package:MASS':
##
##     select
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(purrr)
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:GenomicRanges':
##
##     reduce

## The following object is masked from 'package:IRanges':
##
##     reduce
```

```
suppressMessages(library(EnsDb.Hsapiens.v86))
edb <- EnsDb.Hsapiens.v86
library(cataract)
```

```
## Loading required package: scde

## Loading required package: flexmix

## Loading required package: lattice
```

```
qc_dirs <- c("~/single_cell_pipeline/output/FACS_20170407_sunlee_H_sapiens_output/multiqc_data/", "~/si

# left out
# "multiqc_rseqc_read_distribution",

fix_qc_df <- function(qc_dir){
  qc_file_names <- c("multiqc_fastqc", "multiqc_picard_dups", "multiqc_cutadapt", "multiqc_bowtie2")
  qc_file_kw <- paste0(".*", qc_file_names, ".txt")

  qc_file_kw <- paste(qc_file_kw, collapse="|")

  qc_paths <- list.files(qc_dir, pattern = qc_file_kw, full.names = TRUE)

  qc_dfs <- lapply(qc_paths, read.table, sep = "\t", header = TRUE)
  names(qc_dfs) <- qc_file_names

  fix_mixed_ids <- function(df){

    df <- dplyr::mutate(df, Sample = as.integer(gsub("_.*$|\\..*$", "", Sample))) %>%
      arrange(Sample)
  }

  qc_dfs <- purrr::map(qc_dfs, fix_mixed_ids)

  scone_qc <- Reduce(function(...) merge(..., by='Sample', all.x=TRUE), qc_dfs) %>%
    dplyr::filter(!grepl("trimmed", Filename)) %>%
    dplyr::select_if(is.numeric) %>%
    group_by(Sample) %>%
    top_n(2, overall_alignment_rate) %>%
    summarise_all(funs(mean)) %>%
    mutate(Sample = paste0("X", Sample))

  return(scone_qc)
}

scone_qc <- lapply(qc_dirs, fix_qc_df)

add_tag <- function(df, tag){
  df <- mutate(df, Sample = paste0(tag, "_", Sample))
```

```
  return(df)
}

tags <- c("exp1", "exp2")

scone_qc <- purrr::map2(scone_qc, tags, add_tag)

scone_qc <- dplyr::bind_rows(scone_qc)

#filter out constant columns
scone_qc <- subset(scone_qc, select=-c(Sequences.flagged.as.poor.quality))
```

```
# scone_qc_path <- paste0(qc_dir, "scone_qc.txt")

scone_qc_path <- paste0("~/single_cell_pipeline/output/merged_analyses/FACS_20170407_sunlee_FACS_201710

write.csv(scone_qc, scone_qc_path)
```

Introduction

Single-cell RNA sequencing (scRNA-Seq) technologies are opening the way for transcriptome-wide profiling across diverse and complex mammalian tissues, facilitating unbiased identification of novel cell sub-populations and discovery of novel cellular function. As in other high-throughput analyses, a large fraction of the variability observed in scRNA-Seq data results from batch effects and other technical artifacts [@hicks2015]. In particular, a unique reliance on minuscule amounts of starting mRNA can lead to widespread "drop-out effects," in which expressed transcripts are missed during library preparation and sequencing. Due to the biases inherent to these assays, data normalization is an essential step prior to many downstream analyses. As we face a growing cohort of scRNA-Seq technologies, diverse biological contexts, and novel experimental designs, we cannot reasonably expect to find a one-size-fits-all solution to data normalization.

`scone` supports a rational, data-driven framework for assessing the efficacy of various normalization workflows, encouraging users to explore trade-offs inherent to their data prior to finalizing their data normalization strategy. We provide an interface for running multiple normalization workflows in parallel, and we offer tools for ranking workflows and visualizing study-specific trade-offs.

This package was originally developed to address normalization problems specific to scRNA-Seq expression data, but it should be emphasized that its use is not limited to scRNA-Seq data normalization. Analyses based on other high-dimensional data sets - including bulk RNA-Seq data sets - can utilize tools implemented in the `scone` package.

Human Neurogenesis

We will demonstrate the basic `scone` workflow by using an early scRNA-Seq data set [@pollen2014]. We focus on a set of 65 human cells sampled from four biological conditions: Cultured neural progenitor cells ("NPC") derived from pluripotent stem cells, primary cortical samples at gestation weeks 16 and 21 ("GW16" and "GW21" respectively) and late cortical samples cultured for 3 weeks ("GW21+3"). Gene-level expression data for these cells can be loaded directly from the `scRNAseq` package on Bioconductor.

```
## ----- load summarizedexperiment object -----

sunlee_all <- readRDS("~/single_cell_pipeline/output/merged_analyses/FACS_20170407_sunlee_FACS_20171031
```

```
# sunlee_1031 <- readRDS("~/single_cell_pipeline/output/FACS_20171031_sunlee_H_sapiens_output/FACS_2017

# Set assay to RSEM estimated counts
assay(sunlee_all) = assays(sunlee_all)$counts

rnaseq_metrics <- read.csv(scone_qc_path, row.names = 1)
```

The `rsem_counts` assay contains expected gene-level read counts via RSEM [@li2011] quantification of 130 single-cell libraries aligned to the hg38 RefSeq transcriptome. The data object also contains library transcriptome alignment metrics obtained from Picard and other basic tools.

```
## ----- List all QC fields -----

#join qc data and sample metadata
sample_metadata <- data.frame(colData(sunlee_all))
sample_metadata <- dplyr::inner_join(sample_metadata, scone_qc, by = c("sample_id" = "Sample")) %>%
  mutate(seq_run = ifelse(grepl("exp1", sample_id), "exp_1", "exp_2"))


# sample_metadata <- as.data.frame(sample_metadata)
rownames(sample_metadata) <- sample_metadata[,1]

# set new Coldata for summarized experiment

sunlee_all <- SingleCellExperiment(assays=list(counts=assays(sunlee_all)$counts), colData=sample_metada


# List all qc fields (accessible via colData())
metadata(sunlee_all)$which_qc <- colnames(rnaseq_metrics)
```

All cell-level metadata, such as cell origin and sequence coverage ("low" vs "high" coverage) can be accessed using `colData()`:

```
# Joint distribution of "Sequencing Number"" and "Fetal Age""
# table(colData(sunlee_all)$paired_total,
#       colData(sunlee_all)$Total.Sequences)
```

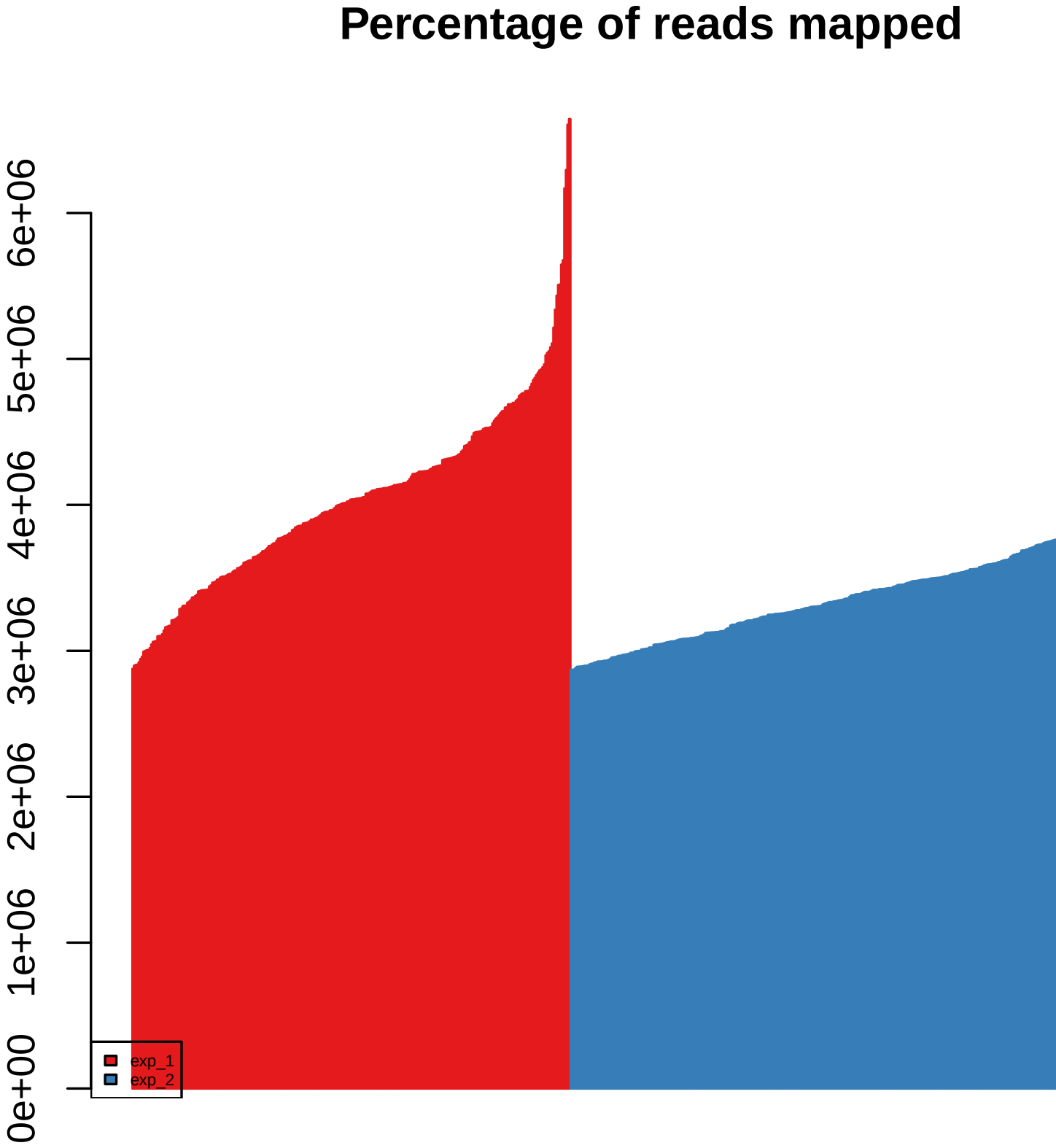
Each cell had been sequenced twice, at different levels of coverage. In this vignette we will focus on the high-coverage data. Before we get started we will do some preliminary filtering to remove the low-coverage replicates and undetected gene features:

```
is_select <- colData(sunlee_all)[colData(sunlee_all)$paired_total > quantile(colData(sunlee_all)$paired
sunlee_all = sunlee_all[,rownames(is_select)]

is_select = colData(sunlee_all)[["paired_total"]] > quantile(colData(sunlee_all)$paired_total, 0.25)
# sunlee_all = sunlee_all[,is_select]
#
# # Retain only detected transcripts
# sunlee_all = sunlee_all[which(apply(assay(sunlee_all) > 0,1,any)),]
```

Visualizing Technical Variability and Batch Effects

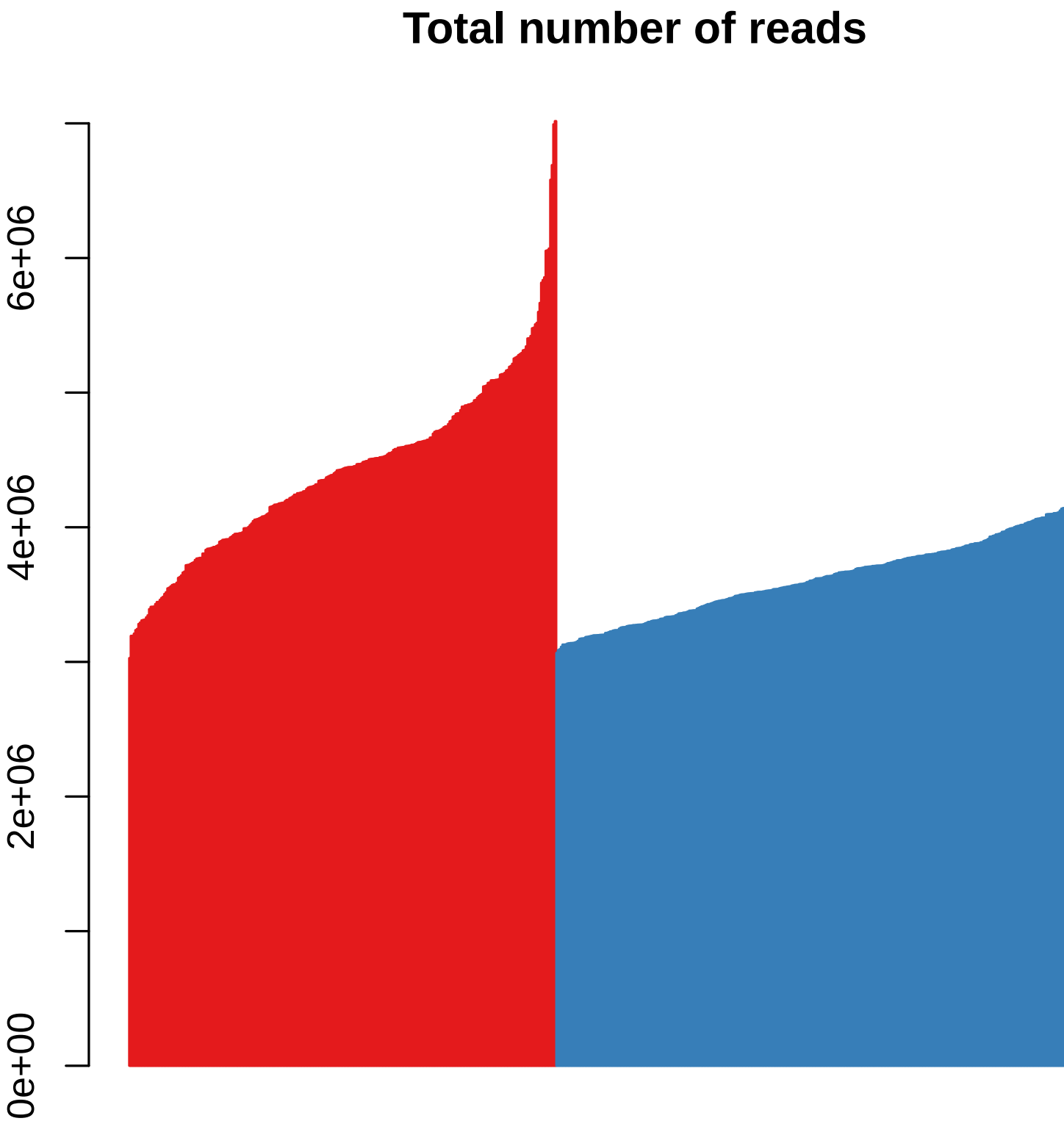
One of our alignment quality readouts is the fraction of reads aligned to the transcriptome. We can use simple bar plots to visualize how this metric relates to the biological batch.

```r
library(RColorBrewer)
# Define a color scheme
cc <- c(brewer.pal(9, "Set1"))

# One batch per Biological Condition
col_data <- colData(sunlee_all)

batch = factor(colData(sunlee_all)$seq_run)

# Alignment Quality Metrics
qc = col_data[,(names(col_data) %in% metadata(sunlee_all)$which_qc)]

# Barplot of read proportion mapping to human transcriptome
ralign = qc$paired_total
o = order(ralign)[order(batch[order(ralign)])] # Order by batch, then value

barplot(ralign[o], col=cc[batch][o],
        border=cc[batch][o], main="Percentage of reads mapped")
legend("bottomleft", legend=levels(batch), fill=cc,cex=0.4)
```



We can see modest differences between batches, and we see that there is one GW21 cell with a particularly low alignment rate relative to the rest of the GW21 batch. These types of observations can inform us of "poor-quality" libraries or batches. We may alternatively consider the number of reads for each library:

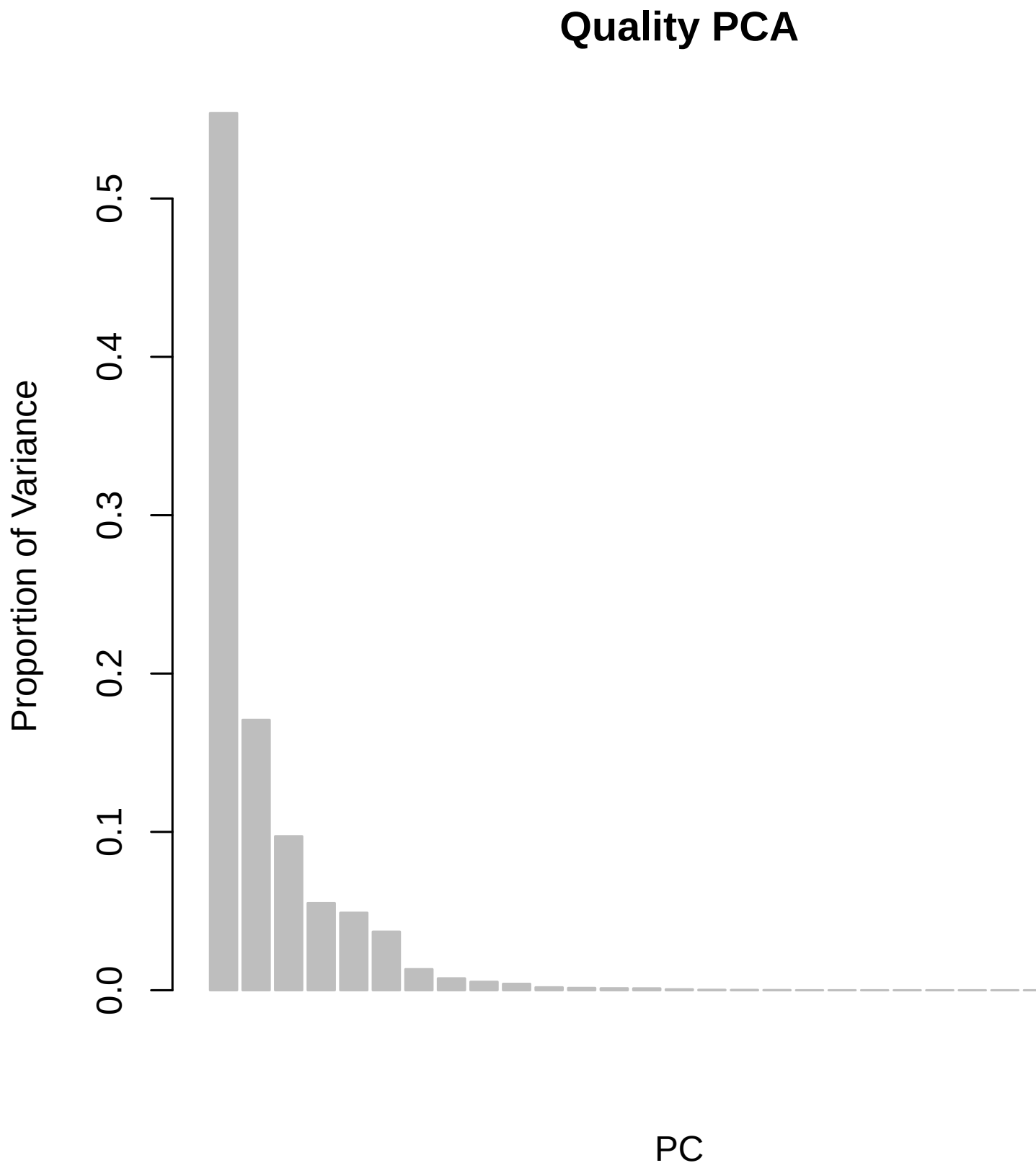
```r
# Barplot of total read number
nreads = qc$Total.Sequences
o = order(nreads)[order(batch[order(nreads)])] # Order by batch, then value

barplot(nreads[o], col=cc[batch][o],
        border=cc[batch][o], main="Total number of reads")
legend("topright", legend=levels(batch), fill=cc, cex=0.4)
```



We see that read coverage varies substantially between batches as well as within batches. These coverage differences and other technical features can induce non-intuitive biases upon expression estimates. Though some biases can be addressed with simple library-size normalization and cell-filtering, demand for greater cell numbers may require more sophisticated normalization methods in order to compare multiple batches of cells. Batch-specific biases are impossible to address directly in this study as biological origin and sample preparation are completely confounded.

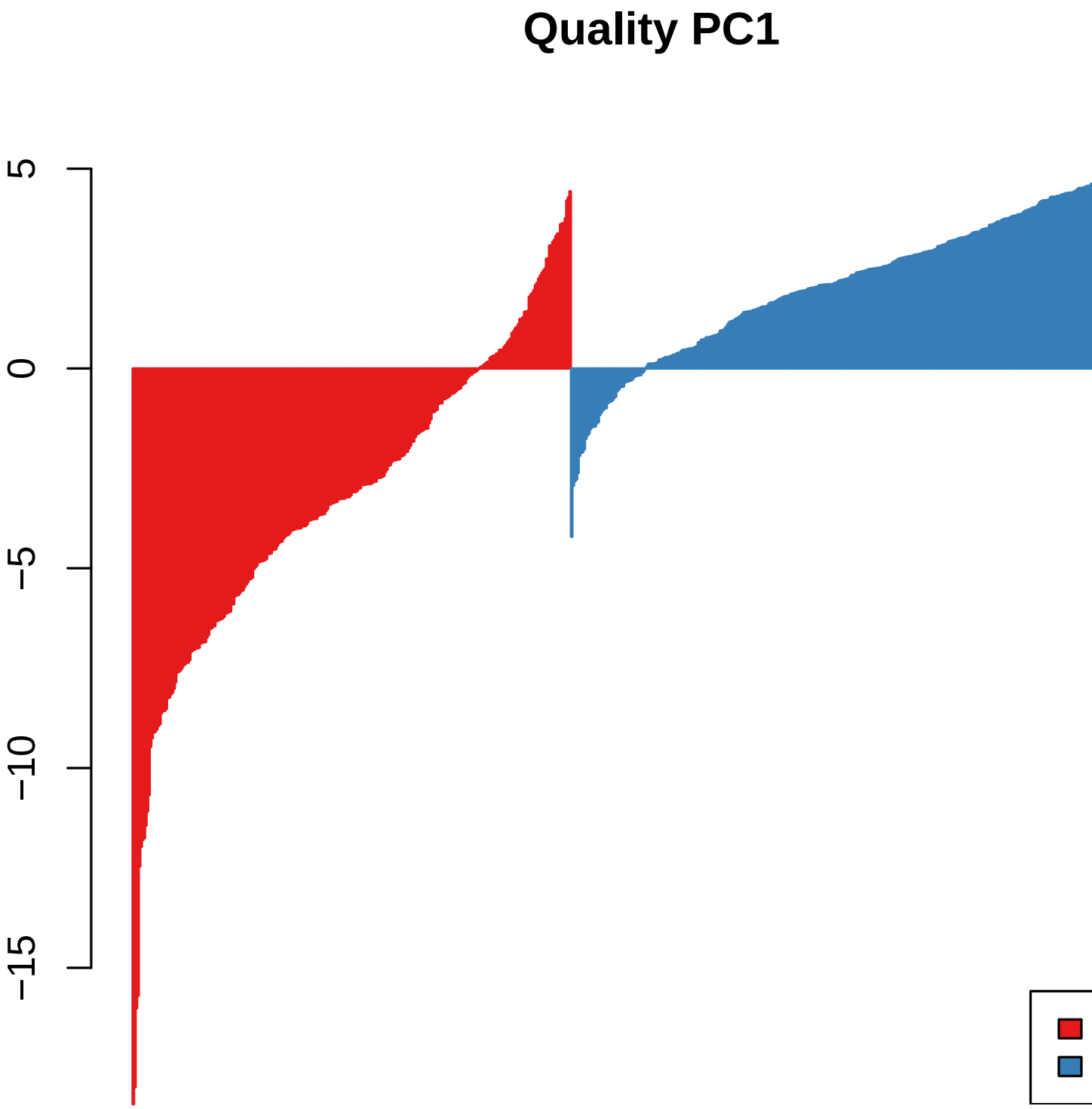
While it can be very helpful to visualize distributions of single quality metrics it should be noted that QC metrics are often correlated. In some cases it may be more useful to consider Principal Components (PCs) of the quality matrix, identifying latent factors of protocol variation:

```
## ----- PCA of QC matrix -----
qpc = prcomp(qc,center = TRUE,scale. = TRUE)
barplot((qpc$sdev^2)/sum(qpc$sdev^2), border="gray",
        xlab="PC", ylab="Proportion of Variance", main="Quality PCA")
```



Even though 19 different QC metrics have been quantified in this analysis, PCA shows us that only a small number of PCs are needed to described a majority of the QC variance (e.g. 3 to explain 76%). We will now visualize the distribution of the first PC in the context of batch:

```r
# Barplot of PC1 of the QC matrix
qc1 = as.vector(qpc$x[,1])
o = order(qc1)[order(batch[order(qc1)])]

barplot(qc1[o], col=cc[batch][o],
        border=cc[batch][o], main="Quality PC1")
legend("bottomright", legend=levels(batch),
       fill=cc, cex=0.8)
```



This first PC appears to represent both inter-batch and intra-batch sample heterogeneity, similar the the total number of reads. If this latent factor reflects variation in sample preparation, we may expect expression artifacts to trace this factor as well: in other words, we should be very skeptical of genes for which expression correlates strongly with the first PC of quality metrics. In this vignette we will show how latent factors like this can be applied to the normalization problem.

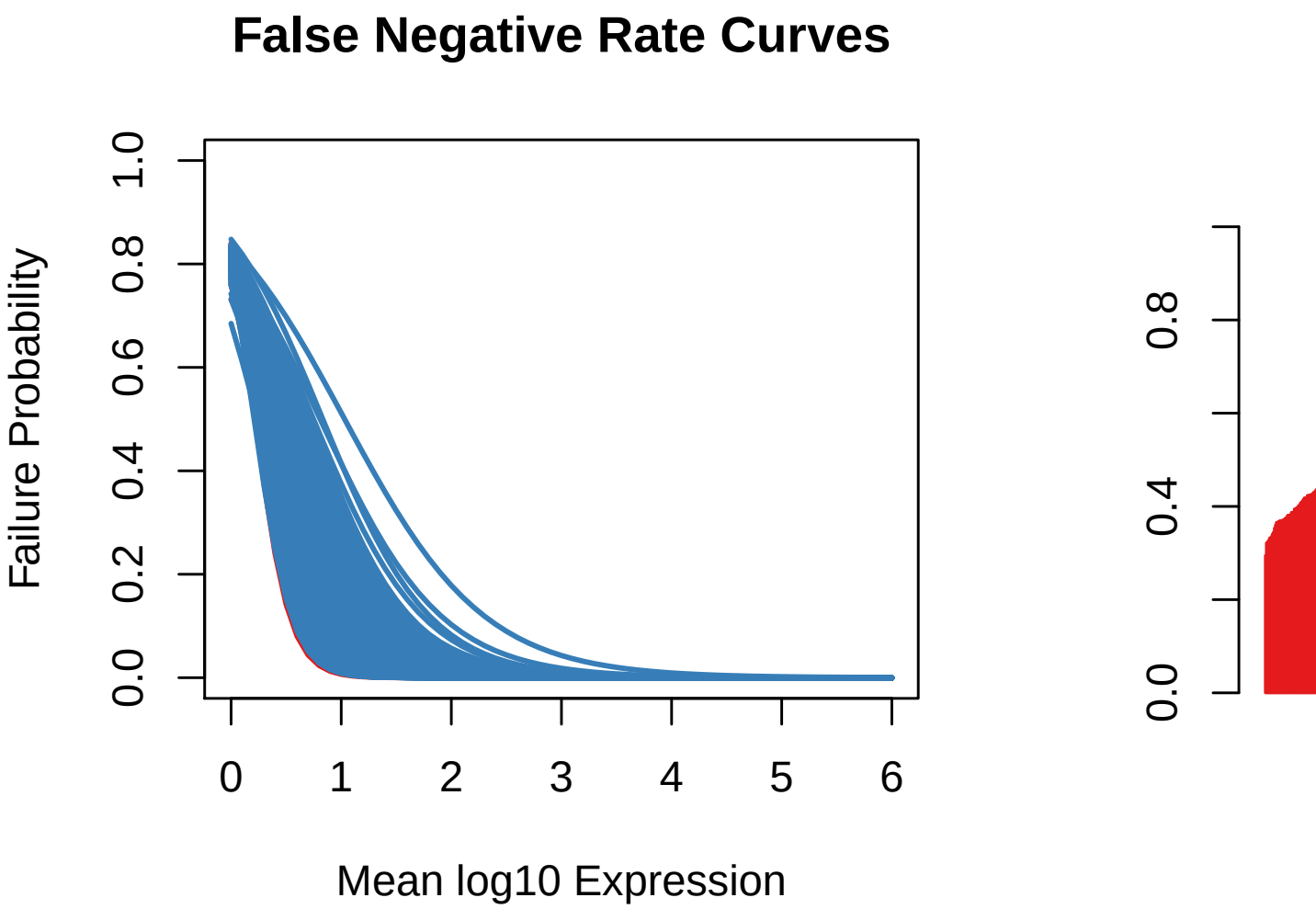
Drop-out Characteristics

Before we move on to normalization, let's briefly consider a uniquely single-cell problem: "drop-outs." One of the greatest challenges in modeling drop-out effects is modeling both i) technical drop-outs and ii) biological expression heterogeneity. One way to simplify the problem is to focus on genes for which we have strong prior belief in true expression. The `scone` package contains lists of genes that are believed to be ubiquitously and even uniformly expressed across human tissues. If we assume these genes are truly expressed in all cells, we can label all zero abundance observations as drop-out events. We model detection failures as a logistic function of mean expression, in line with the standard logistic model for drop-outs employed by the field:

```
# Extract Housekeeping Genes
data(housekeeping)

housekeeping <- cataract::lookup_transcripts(housekeeping[[1]])
hk = intersect(housekeeping,rownames(sunlee_all))
hk <- hk[rowSums(assays(sunlee_all)$counts[hk,]) > 1]

# Mean log10(x+1) expression
mu_obs = rowMeans(log10(assays(sunlee_all)$counts[hk,]+1))

# Assumed False Negatives
drop_outs = assay(sunlee_all)[hk,] == 0

# Logistic Regression Model of Failure
ref.glms = list()
for (si in 1:dim(drop_outs)[2]){
  fit = glm(cbind(drop_outs[,si],1 - drop_outs[,si]) ~ mu_obs,
            family=binomial(logit))
  ref.glms[[si]] = fit$coefficients
}
```

The list `ref.glm` contains the intercept and slope of each fit. We can now visualize the fit curves and the corresponding Area Under the Curves (AUCs):

```
par(mfrow=c(1,2))

# Plot Failure Curves and Calculate AUC
plot(NULL, main = "False Negative Rate Curves",
     ylim = c(0,1),xlim = c(0,6),
     ylab = "Failure Probability", xlab = "Mean log10 Expression")
x = (0:60)/10
AUC = NULL
for(si in 1:ncol(assay(sunlee_all))){
  y = 1/(exp(-ref.glms[[si]][1] - ref.glms[[si]][2] * x) + 1)
  AUC[si] = sum(y)/10
  lines(x, 1/(exp(-ref.glms[[si]][1] - ref.glms[[si]][2] * x) + 1),
        type = 'l', lwd = 2, col = cc[batch][si])
}

# Barplot of FNR AUC
o = order(AUC)[order(batch[order(AUC)])]

barplot(AUC[o], col=cc[batch][o], border=cc[batch][o], main="FNR AUC")
legend("topright", legend=levels(batch), fill=cc, cex=0.4)
```



Model-based metrics such as these may be more interpretable with respect to upstream sample preparation, and can be very useful for assessing single-cell library quality.

The `scone` Workflow

So far we have only described potential problems with single-cell expression data. Now we will take steps to address problems with our example data set. The basic QC and normalization pipeline we will use in this vignette allows us to:

- Filter out poor libraries using the `metric_sample_filter` function.
- Run and score many different normalization workflows (different combinations of normalization modules) using the main `scone` function.
- Browse top-ranked methods and visualize trade-offs with the `biplot_color` and `sconeReport` function.

In order to run many different workflows, SCONE relies on a normalization workflow template composed of 3 modules:

1) Data imputation: replacing zero-abundance values with expected values under a prior drop-out model. As we will see below, this module may be used as a modifier for module 2, without passing imputed values forward to downstream analyses. 2) Scaling or quantile normalization: either i) normalization that scales each sample's transcriptome abundances by a single factor or ii) more complex offsets that match quantiles across samples. Examples: TMM or DESeq scaling factors, upper quartile normalization, or

full-quantile normalization.

2) Regression-based approaches for removing unwanted correlated variation from the data, including batch effects. Examples: RUVg [@risso2014] or regression on Quality Principal Components described above.

Sample Filtering with `metric_sample_filter`

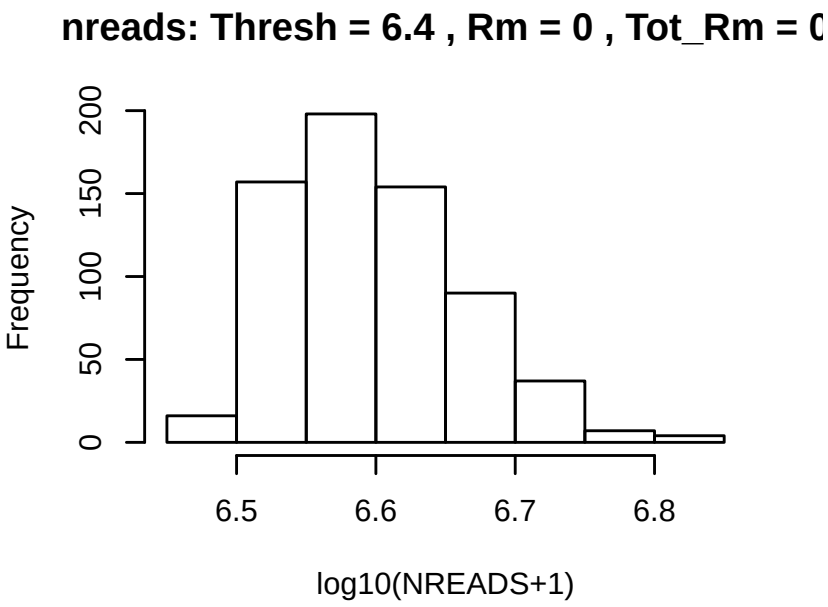
The most basic sample filtering function in `scone` is the `metric_sample_filter`. The function takes a consensus approach, retaining samples that pass multiple data-driven criteria.

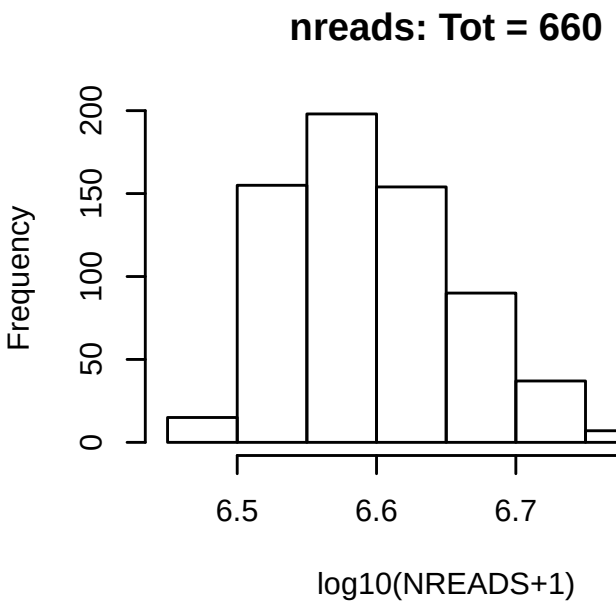
`metric_sample_filter` takes as input an expression matrix. The output depends on arguments provided, but generally consists of a list of 4 logicals designating each sample as having failed (TRUE) or passed (FALSE) threshold-based filters on 4 sample metrics:

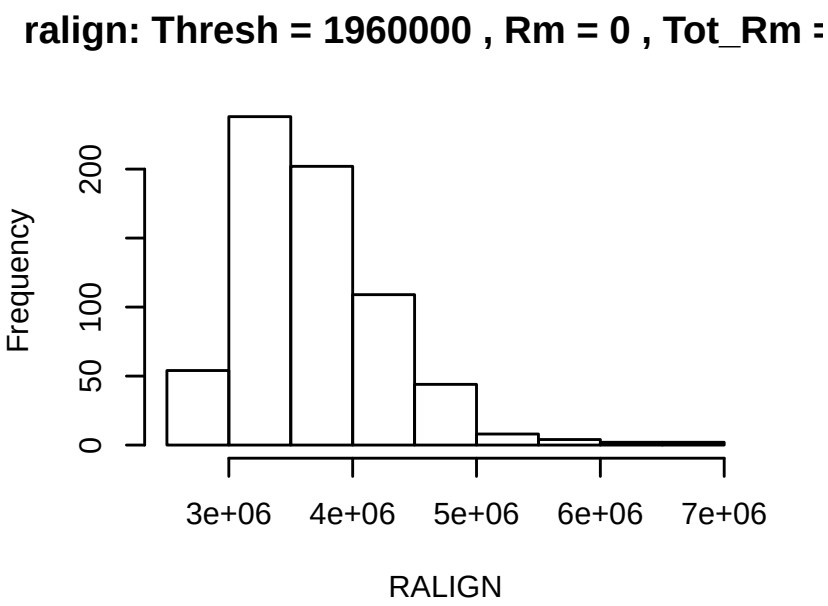
- Number of reads.
- Ratio of reads aligned to the genome. Requires the `ralign` argument.
- "Transcriptome breadth" - Defined here as the proportion of "high-quality" genes detected in the sample. Requires the `gene_filter` argument.
- FNR AUC. Requires the `pos_controls` argument.

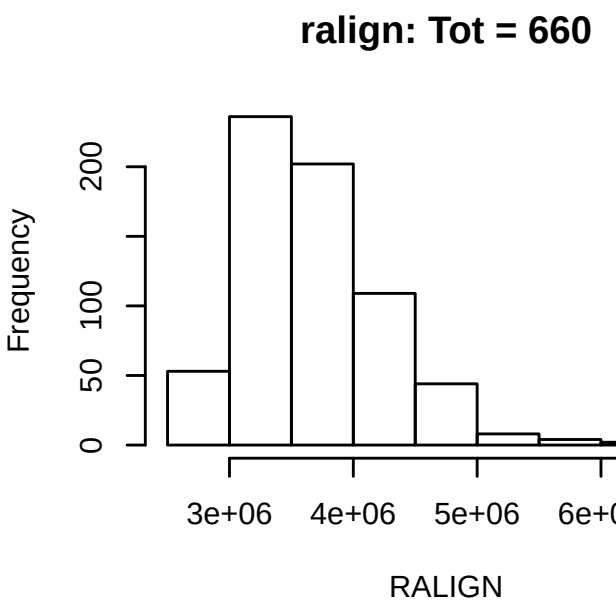
If required arguments are missing for any of the 4, the function will simply return NA instead of the corresponding logical.

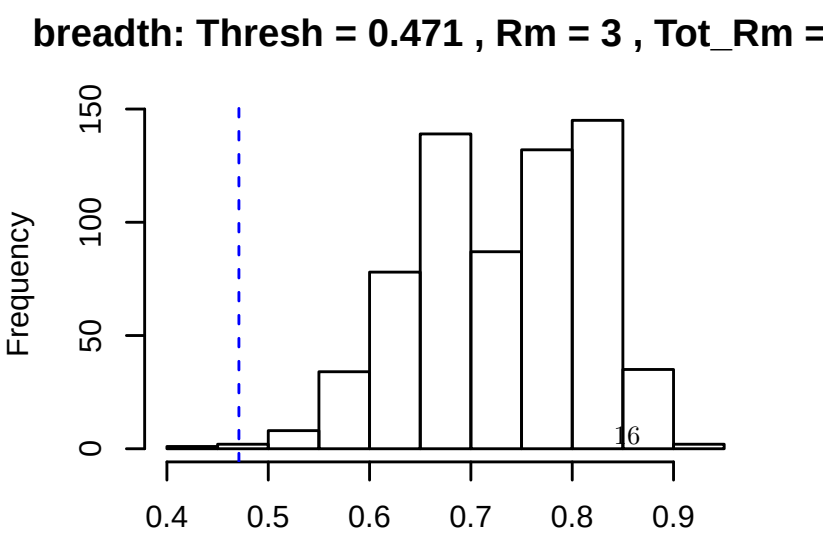
```
# Initial Gene Filtering:
# Select "common" transcripts based on proportional criteria.
num_reads = quantile(assay(sunlee_all)[assay(sunlee_all) > 0])[4]
num_cells = 0.25*ncol(sunlee_all)
is_common = rowSums(assay(sunlee_all) >= num_reads ) >= num_cells


# Metric-based Filtering
mfilt = metric_sample_filter(assay(sunlee_all),
                             nreads = colData(sunlee_all)$Total.Sequences,
                             ralign = colData(sunlee_all)$paired_total,
                             gene_filter = is_common,
                             pos_controls = rownames(sunlee_all) %in% hk,

                             zcut = 3, mixture = FALSE,
                             plot = TRUE)
```

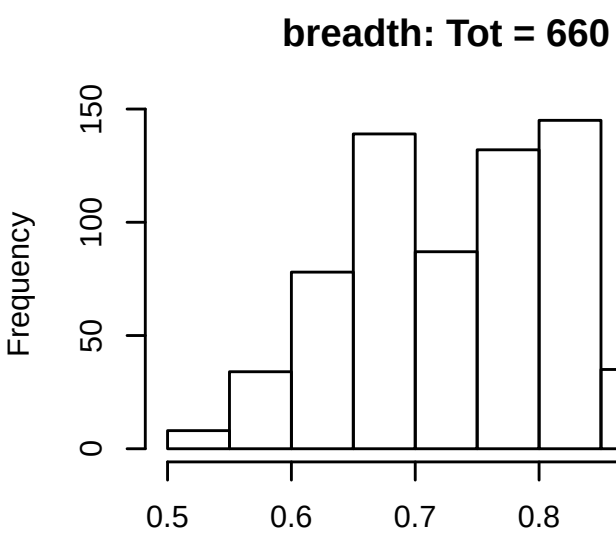

nreads: Thresh = 6.4 , Rm = 0 , Tot_Rm = 0
Frequency
0
50
100
150
200
6.5
6.6
6.7
6.8
log10(NREADS+1)


nreads: Tot = 660
Frequency
0
50
100
150
200
6.5
6.6
6.7
log10(NREADS+1)


ralign: Thresh = 1960000 , Rm = 0 , Tot_Rm =
Frequency
0
50
100
200
3e+06
4e+06
5e+06
6e+06
7e+06
RALIGN


ralign: Tot = 660
Frequency
0
50
100
200
3e+06
4e+06
5e+06
RALIGN


breadth: Thresh = 0.471 , Rm = 3 , Tot_Rm =
Frequency
0
50
100
150
0.4
0.5
0.6
0.7
0.8
0.9


breadth: Tot = 660
Frequency
0
50
100
150
0.5
0.6
0.7
0.8

```
# Simplify to a single logical
mfilt = !apply(simplify2array(mfilt[!is.na(mfilt)]),1,any)
```

In the call above, we have set the following parameters:

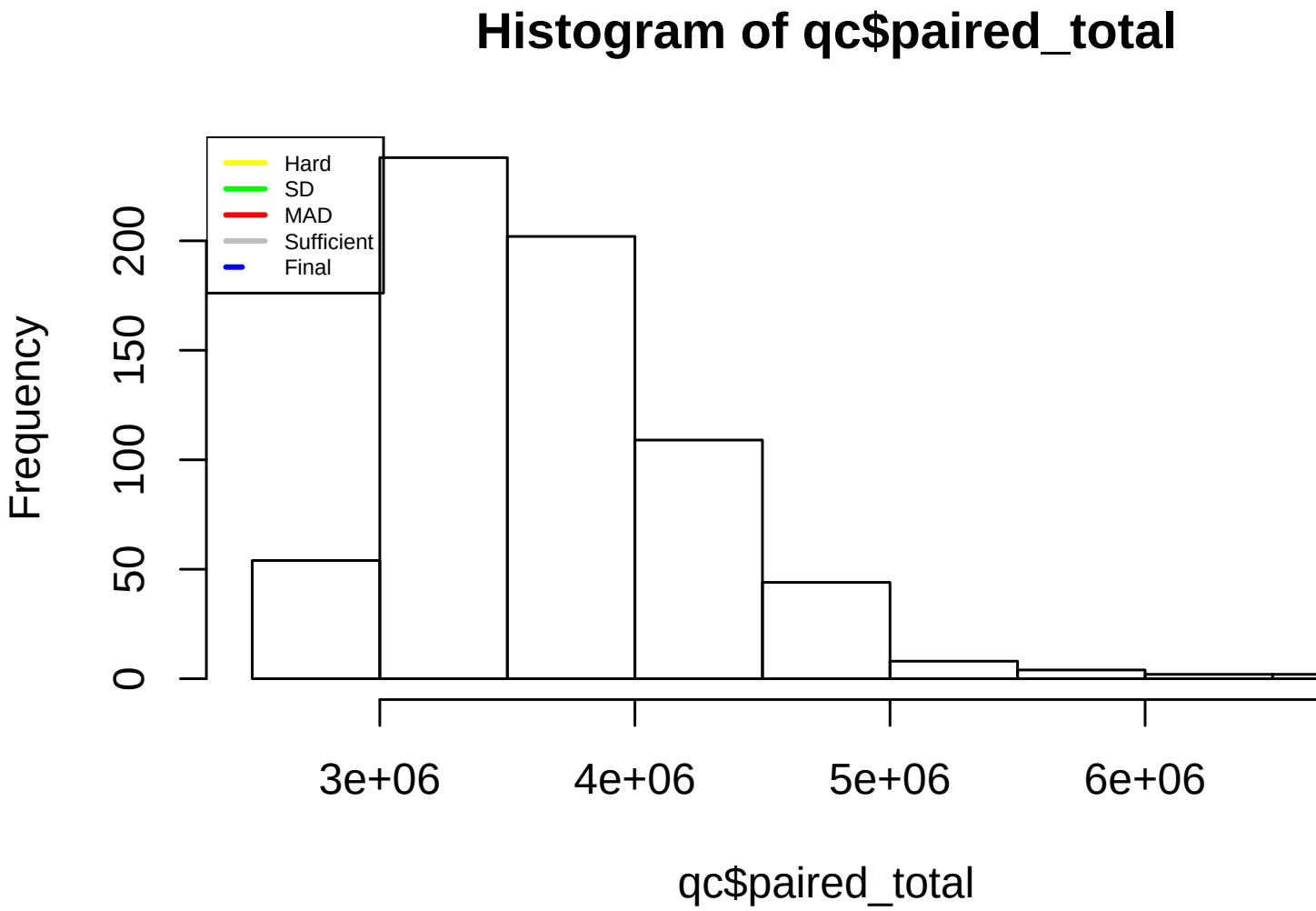
- zcut = 3. Filter leniency (see below).
- mixture = FALSE. Mixture modeling will not be used (see below).
- plot = TRUE. Plot distributions of metrics before and after filtering.

On Threshold Selection

Let's take a closer look at the computation behind selecting the ralign filter. In choosing a threshold value 67.7, `metric_sample_filter` is taking 4 values into account:

1) `hard_ralign`, the default "hard" threshold at 15 - rather forgiving... 2) 3 (`zcut`) times the standard deviation below the mean `ralign` value. 3) 3 (`zcut`) times the MAD below the median `ralign` value. 4) `suff_ralign`, the sufficient threshold set to NULL by default.

```
# hist(qc$paired_total, breaks = 0:100)
hist(qc$paired_total)
# Hard threshold
abline(v = 15, col = "yellow", lwd = 2)
# 3 (zcut) standard deviations below the mean paired_total value
abline(v = mean(qc$paired_total) - 3*sd(qc$paired_total), col = "green", lwd = 2)
# 3 (zcut) MADs below the median paired_total value
abline(v = median(qc$paired_total) - 3*mad(qc$paired_total), col = "red", lwd = 2)
# Sufficient threshold
abline(v = NULL, col = "grey", lwd = 2)

# Final threshold is the minimum of
# 1) the sufficient threshold and
# 2) the max of all others
thresh = min(NULL,
             max(c(15,mean(qc$paired_total) - 3*sd(qc$paired_total),
                   median(qc$paired_total) - 3*mad(qc$paired_total))))
abline(v = thresh, col = "blue", lwd = 2, lty = 2)

legend("topleft",legend = c("Hard","SD","MAD","Sufficient","Final"),
       lwd = 2, col = c("yellow","green","red","grey","blue"),
       lty = c(1,1,1,1,2), cex = .5)
```



We see here that the 3rd "MAD" threshold exceeds the first two thresholds ("Hard" and "SD"), and as the "Sufficient" threshold is NULL `metric_sample_filter` settles for the the third threshold. If the "Sufficient" threshold was not NULL and was exceeded by any of the other three thresholds ("Hard","SD","MAD"), `metric_sample_filter` would settle for the "Sufficient" threshold. Note also that if `mixture=TRUE` an additional criterion is considered: distributions may be fit to a two-component mixture model, and a threshold is defined with respect to the mean and standard deviation of the "best" component.

As `metric_sample_filter` relies on a maximum of candidate thresholds, we recommend users treat this function as a stringent sample filter.

Applying the sample filter

With the `metric_sample_filter` output in hand, it is fairly straightforward to remove the one "poor" sample from our study:

```
goodDat = sunlee_all[,mfilt]

# Final Gene Filtering: Highly expressed in at least 5 cells
num_reads = quantile(assay(sunlee_all)[assay(sunlee_all) > 0])[4]
num_cells = 5
is_quality = rowSums(assay(sunlee_all) >= num_reads ) >= num_cells
```

Running and Scoring Normalization Workflows with `scone`

Not only does `scone` normalize expression data, but it also provides a framework for evaluating the performance of normalization workflows.

Creating a SconeExperiment Object

Prior to running main `scone` function we will want to define a `SconeExperiment` object that contains the primary expression data, experimental metadata, and control gene sets.

```
# Expression Data (Required)
expr = assay(goodDat)[is_quality,]

# Biological Origin - Variation to be preserved (Optional)
bio = factor(colData(goodDat)$day)


# Processed Alignment Metrics - Variation to be removed (Optional)
col_data <- colData(sunlee_all)

# Alignment Quality Metrics
qc = col_data[,(names(col_data) %in% metadata(sunlee_all)$which_qc)]

ppq = scale(qc[,apply(qc,2,sd) > 0],center = TRUE,scale = TRUE)
ppq <- ppq[colnames(expr),]

poscon_genes <- c("ALS2", "CDK5R1", "CYFIP1",
                  "DPYSL5", "FEZ1", "FEZ2",
                  "MAPT", "MDGA1", "NRCAM",
                  "NRP1", "NRXN1", "OPHN1",
                  "OTX2", "PARD6B", "PPT1",
                  "ROBO1", "ROBO2", "RTN1",
                  "RTN4", "SEMA4F", "SIAH1",
                  "SLIT2", "SMARCA1", "THY1",
                  "TRAPPC4", "UBB", "YWHAG",
                  "YWHAH")

poscon_trx <- lookup_transcripts(poscon_genes)

# Positive Control Genes - Prior knowledge of DE (Optional)
poscon = intersect(rownames(expr),poscon_trx)

# Negative Control Genes - Uniformly expressed transcripts (Optional)
negcon = intersect(rownames(expr),hk)


# Creating a SconeExperiment Object
my_scone <- SconeExperiment(expr,
                qc=ppq, bio = bio,
                negcon_ruv = rownames(expr) %in% negcon,
                poscon = rownames(expr) %in% poscon
)
```

Defining Normalization Modules

Before we can decide which workflows (normalizations) we will want to compare, we will also need to define the types of scaling functions we will consider in the comparison of normalizations:

```
## ----- User-defined function: Dividing by number of detected genes -----

EFF_FN = function (ei)
{
  sums = colSums(ei > 0)
  eo = t(t(ei)*sums/mean(sums))
  return(eo)
}

## ----- Scaling Argument -----

scaling=list(none=identity, # Identity - do nothing

             eff = EFF_FN, # User-defined function

             sum = SUM_FN, # SCONE library wrappers...
             tmm = TMM_FN,
             uq = UQ_FN,
             fq = FQT_FN,
             deseq = DESEQ_FN)
```

If imputation is to be included in the comparison, imputation arguments must also be provided by the user:

```
# Simple FNR model estimation with SCONE::estimate_ziber
fnr_out = estimate_ziber(x = expr, bulk_model = TRUE,
                         pos_controls = rownames(expr) %in% hk,
                         maxiter = 10000)

## ----- Imputation List Argument -----
imputation=list(none=impute_null, # No imputation
                expect=impute_expectation) # Replace zeroes

## ----- Imputation Function Arguments -----
# accessible by functions in imputation list argument
impute_args = list(p_nodrop = fnr_out$p_nodrop, mu = exp(fnr_out$Alpha[1,]))

my_scone <- scone(my_scone,
                  imputation = imputation, impute_args = impute_args,
                  scaling=scaling,
                  k_qc=3, k_ruv = 3,
                  adjust_bio="no",
                  run=FALSE)
```

Note, that because the imputation step is quite slow, we do not run it here, but will run scone without imputation.

Selecting SCONE Workflows

The main `scone` method arguments allow for a lot of flexibility, but a user may choose to run very specific combinations of modules. For this purpose, `scone` can be run in `run=FALSE` mode, generating a list of

workflows to be performed and storing this list within a `SconeExperiment` object. After running this command the list can be extracted using the `get_params` method.

```
my_scone <- scone(my_scone,
                  scaling=scaling,
                  k_qc=3, k_ruv = 3,
                  adjust_bio="no",
                  run=FALSE)

head(get_params(my_scone))
```

```
##                                 imputation_method scaling_method
## none,none,no_uv,no_bio,no_batch              none           none
## none,eff,no_uv,no_bio,no_batch               none            eff
## none,sum,no_uv,no_bio,no_batch               none            sum
## none,tmm,no_uv,no_bio,no_batch               none            tmm
## none,uq,no_uv,no_bio,no_batch                none             uq
## none,fq,no_uv,no_bio,no_batch                none             fq
##                                 uv_factors adjust_biology adjust_batch
## none,none,no_uv,no_bio,no_batch      no_uv         no_bio     no_batch
## none,eff,no_uv,no_bio,no_batch       no_uv         no_bio     no_batch
## none,sum,no_uv,no_bio,no_batch       no_uv         no_bio     no_batch
## none,tmm,no_uv,no_bio,no_batch       no_uv         no_bio     no_batch
## none,uq,no_uv,no_bio,no_batch        no_uv         no_bio     no_batch
## none,fq,no_uv,no_bio,no_batch        no_uv         no_bio     no_batch
```

In the call above, we have set the following parameter arguments:

- k_ruv = 3. The maximum number of RUVg factors to consider.
- k_qc = 3. The maximum number of quality PCs (QPCs) to be included in a linear model, analogous to RUVg normalization. The qc argument must be provided.
- adjust_bio = "no." Biological origin will NOT be included in RUVg or QPC regression models. The bio argument will be provided for evaluation purposes.

These arguments translate to the following set of options:

```
apply(get_params(my_scone),2,unique)
```

```
## $imputation_method
## [1] "none"
##
## $scaling_method
## [1] "none"  "eff"   "sum"   "tmm"   "uq"    "fq"    "deseq"
##
## $uv_factors
## [1] "no_uv"   "ruv_k=1" "ruv_k=2" "ruv_k=3" "qc_k=1"  "qc_k=2"  "qc_k=3"
##
## $adjust_biology
## [1] "no_bio"
##
## $adjust_batch
## [1] "no_batch"
```

Some scaling methods, such as scaling by gene detection rate (`EFF_FN()`), will not make sense within the context of imputed data, as imputation replaces zeroes with non-zero values. We can use the `select_methods` method to produce a `SconeExperiment` object initialized to run only meaningful normalization workflows.

```
is_screened = ((get_params(my_scone)$imputation_method == "expect") &
                 (get_params(my_scone)$scaling_method %in% c("none",
                                                             "eff")))

my_scone = select_methods(my_scone,
                          rownames(get_params(my_scone))[!is_screened ])
```

Calling `scone` with `run=TRUE`

Now that we have selected our workflows, we can run `scone` in `run=TRUE` mode. As well as arguments used in `run=FALSE` mode, this mode relies on a few additional arguments. In order to understand these arguments, we must first understand the 8 metrics used to evaluate each normalization. The first 6 metrics rely on a reduction of the normalized data down to 3 dimensions via PCA (default). Each metric is taken to have a positive (higher is better) or negative (lower is better) signature.

- BIO_SIL: Preservation of Biological Difference. The average silhouette width of clusters defined by `bio`, defined with respect to a Euclidean distance metric over the first 3 expression PCs. Positive signature.
- BATCH_SIL: Removal of Batch Structure. The average silhouette width of clusters defined by `batch`, defined with respect to a Euclidean distance metric over the first 3 expression PCs. Negative signature.
- PAM_SIL: Preservation of Single-Cell Heterogeneity. The maximum average silhouette width of clusters defined by PAM clustering, defined with respect to a Euclidean distance metric over the first 3 expression PCs. Positive signature.
- EXP_QC_COR: Removal of Alignment Artifacts. R^2 measure for regression of first 3 expression PCs on first `k_qc` QPCs. Negative signature.
- EXP_UV_COR: Removal of Expression Artifacts. R^2 measure for regression of first 3 expression PCs on first 3 PCs of the negative control (specified by `eval_negcon` or `ruv_negcon` by default) sub-matrix of the original (raw) data. Negative signature.
- EXP_WV_COR: Preservation of Biological Variance. R^2 measure for regression of first 3 expression PCs on first 3 PCs of the positive control (specified by `eval_poscon`) sub-matrix of the original (raw) data. Positive signature.
- RLE_MED: Reduction of Global Differential Expression. The mean squared-median Relative Log Expression (RLE). Negative signature.
- RLE_IQR: Reduction of Global Differential Variability. The variance of the inter-quartile range (IQR) of the RLE. Negative signature.

```
BiocParallel::register(
  BiocParallel::SerialParam()
) # Register BiocParallel Serial Execution

my_scone <- scone(my_scone,
                  scaling=scaling,
                  run=TRUE,
                  eval_kclust = 2:3,stratified_pam = TRUE,
                  return_norm = "in_memory",
                  zero = "postadjust")
```

```
## Warning in .local(x, ...): none_uq returned at least one NA value. Consider removing it from the comp
##  In the meantime, failed methods have been filtered from the output.
```

In the call above, we have set the following parameter arguments:

- eval_kclust = 2:6. For PAM_SIL, range of k (# of clusters) to use when computing maximum average silhouette width of PAM clusterings.
- stratified_pam = TRUE. For PAM_SIL, apply separate PAM clusterings to each biological batch rather than across all batches. Average is weighted by batch group size.

- return_norm = "in_memory". Store all normalized matrices in addition to evaluation data. Otherwise normalized data is not returned in the resulting object.
- zero = "postadjust". Restore data entries that are originally zeroes / negative after normalization to zero after the adjustment step.

The output will contain various updated elements:

```
# View Metric Scores
head(get_scores(my_scone))
```

```
##                                     BIO_SIL   PAM_SIL EXP_QC_COR
## none,fq,ruv_k=3,no_bio,no_batch    -0.1526574 0.5716390 -0.4542208
## none,fq,ruv_k=2,no_bio,no_batch    -0.1053399 0.5180952 -0.6282174
## none,deseq,ruv_k=1,no_bio,no_batch -0.1596490 0.5467250 -0.6804434
## none,eff,ruv_k=3,no_bio,no_batch   -0.1582097 0.4876071 -0.6153897
## none,eff,ruv_k=1,no_bio,no_batch   -0.1346211 0.5747233 -0.7291286
## none,tmm,ruv_k=2,no_bio,no_batch   -0.1643782 0.5154626 -0.6176912
##                                    EXP_UV_COR EXP_WV_COR RLE_MED
## none,fq,ruv_k=3,no_bio,no_batch    -0.3766072  0.3086626       0
## none,fq,ruv_k=2,no_bio,no_batch    -0.6036786  0.2816532       0
## none,deseq,ruv_k=1,no_bio,no_batch -0.7043227  0.2954373       0
## none,eff,ruv_k=3,no_bio,no_batch   -0.5046513  0.2946320       0
## none,eff,ruv_k=1,no_bio,no_batch   -0.7462429  0.2874094       0
## none,tmm,ruv_k=2,no_bio,no_batch   -0.5439366  0.2800171       0
##                                        RLE_IQR
## none,fq,ruv_k=3,no_bio,no_batch    -0.004811855
## none,fq,ruv_k=2,no_bio,no_batch    -0.004615761
## none,deseq,ruv_k=1,no_bio,no_batch -0.024591205
## none,eff,ruv_k=3,no_bio,no_batch   -0.018052434
## none,eff,ruv_k=1,no_bio,no_batch   -0.026251571
## none,tmm,ruv_k=2,no_bio,no_batch   -0.022254317
```

```
# View Mean Score Rank
head(get_score_ranks(my_scone))
```

```
##    none,fq,ruv_k=3,no_bio,no_batch    none,fq,ruv_k=2,no_bio,no_batch
##                            34.35714                            29.35714
## none,deseq,ruv_k=1,no_bio,no_batch   none,eff,ruv_k=3,no_bio,no_batch
##                            25.64286                            25.64286
##   none,eff,ruv_k=1,no_bio,no_batch   none,tmm,ruv_k=2,no_bio,no_batch
##                            25.07143                            25.07143
```

```
# Extract normalized data from top method
out_norm = get_normalized(my_scone,
                          method = rownames(get_params(my_scone))[1])

write.csv(out_norm, "~/single_cell_pipeline/output/merged_analyses/FACS_20170407_sunlee_20121031_scone.
write.csv(colData(goodDat), "~/single_cell_pipeline/output/merged_analyses/FACS_20170407_sunlee_2012103
```

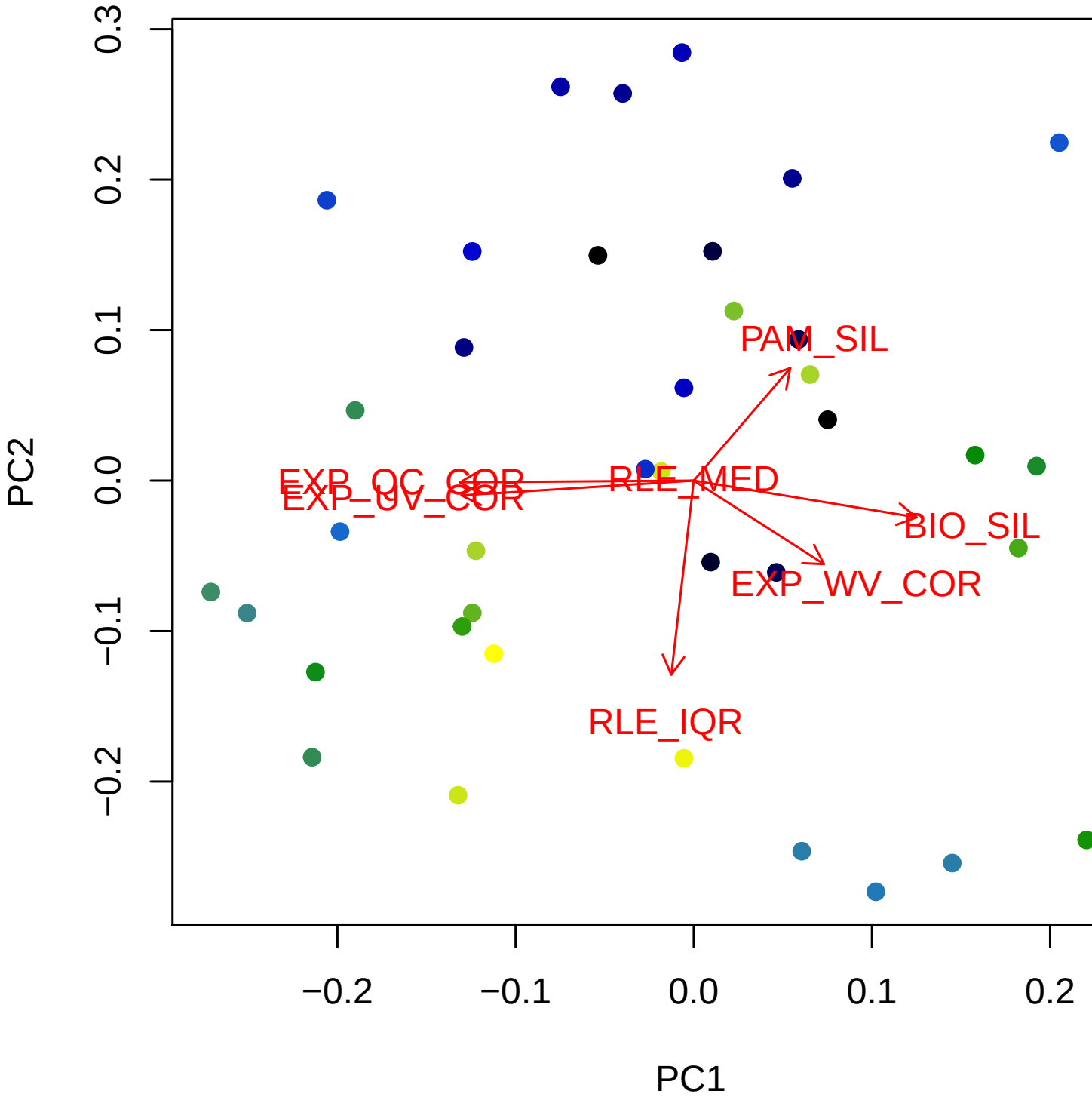
`get_scores` returns the 8 raw metrics for each normalization multiplied by their signature - or "scores." `get_score_ranks` returns the mean score rank for each normalization. Both of these are sorted in decreasing order by mean score rank. Finally `get_normalized` returns the normalized expression data for the requested method. If the normalized data isn't stored in the object it will be recomputed.

Step 3: Selecting a normalization for downstream analysis

Based on our sorting criteria, it would appear that `none,uq,ruv_k=1,no_bio,no_batch` performs well compared to other normalization workflows. A useful way to visualize this method with respect to others is the `biplot_color` function

```
pc_obj = prcomp(apply(t(get_scores(my_scone)),1,rank),
                center = TRUE,scale = FALSE)
bp_obj = biplot_color(pc_obj,y = -get_score_ranks(my_scone),expand = .6)
```

```
## Warning in arrows(0, 0, yy[, 1] * 0.8/ratio, yy[, 2] * 0.8/ratio, col =
## 2, : zero-length arrow is of indeterminate angle and so skipped
```

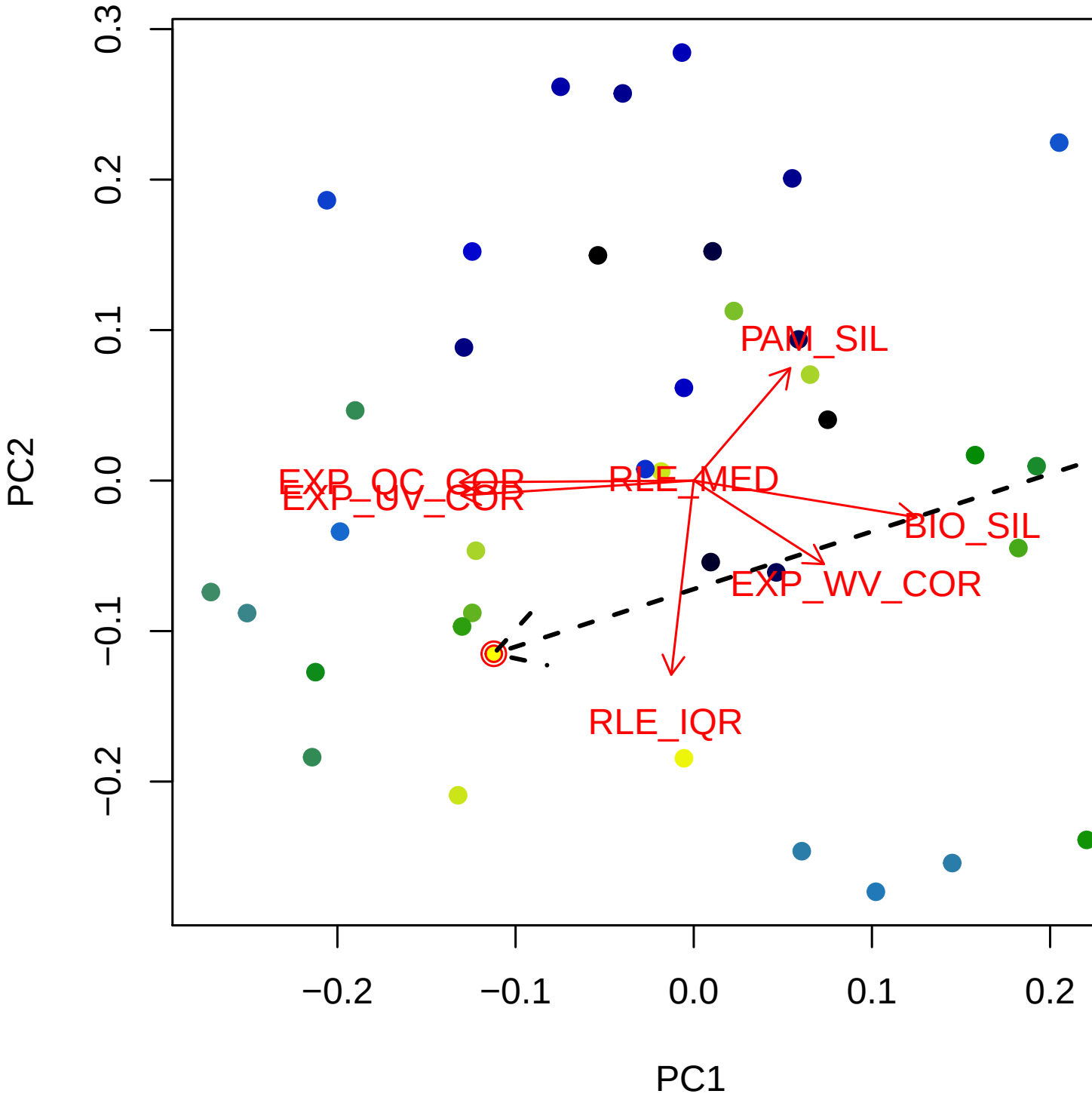


We have colored each point above according the corresponding method's mean score rank (yellow vs blue ~ good vs bad), and we can see that workflows span a continuum of metric performance. Most importantly - and perhaps to no surprise - there is evidence of strong trade-offs between i) Preserving clustering and wanted variation and ii) removing unwanted variation. At roughly 90 degrees to this axis is a direction in which distributional properties of relative log-expression (RLE_MED and RLE_IQR) improve. Let's visualize the top-performing method and it's relation to un-normalized data ("no-op" normalization):

```
bp_obj = biplot_color(pc_obj,y = -get_score_ranks(my_scone),expand = .6)
```

```
## Warning in arrows(0, 0, yy[, 1] * 0.8/ratio, yy[, 2] * 0.8/ratio, col =
## 2, : zero-length arrow is of indeterminate angle and so skipped
```

```
points(t(bp_obj[1,]), pch = 1, col = "red", cex = 1)
points(t(bp_obj[1,]), pch = 1, col = "red", cex = 1.5)

points(t(bp_obj[rownames(bp_obj) == "none,none,no_uv,no_bio,no_batch",]),
       pch = 1, col = "blue", cex = 1)
points(t(bp_obj[rownames(bp_obj) == "none,none,no_uv,no_bio,no_batch",]),
       pch = 1, col = "blue", cex = 1.5)

arrows(bp_obj[rownames(bp_obj) == "none,none,no_uv,no_bio,no_batch",][1],
       bp_obj[rownames(bp_obj) == "none,none,no_uv,no_bio,no_batch",][2],
       bp_obj[1,][1],
       bp_obj[1,][2],
       lty = 2, lwd = 2)
```



The arrow we've added to the plot traces a line from the "no-op" normalization to the top-ranked normalization in SCONE. We see that SCONE has selected a method in-between the two extremes, reducing the signal of unwanted variation while preserving biological signal.

Finally, another useful function for browsing results is `sconeReport`. This function launches a shiny app for evaluating performance of specific normalization workflows.

```
library(shiny)
library(visNetwork)
```

```
library(plotly)
# Methods to consider
scone_methods = c(rownames(get_params(my_scone))[1:12],
                  "none,none,no_uv,no_bio,no_batch")

# Shiny app
sconeReport(my_scone,methods = scone_methods,
            qc = ppq,
            bio = bio,
            negcon = negcon, poscon = poscon)
```

Session Info

```
sessionInfo()
```

```
## R version 3.4.3 (2017-11-30)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.3 LTS
##
## Matrix products: default
## BLAS: /usr/lib/libblas/libblas.so.3.6.0
## LAPACK: /usr/lib/lapack/liblapack.so.3.6.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] SingleCellExperiment_1.0.0 bindrcpp_0.2
##  [3] cataract_0.2.0             scde_2.6.0
##  [5] flexmix_2.3-13             lattice_0.20-35
##  [7] EnsDb.Hsapiens.v86_2.99.0  ensembldb_2.2.2
##  [9] AnnotationFilter_1.2.0     GenomicFeatures_1.30.3
## [11] AnnotationDbi_1.40.0       purrr_0.2.4
## [13] dplyr_0.7.4.9000           RColorBrewer_1.1-2
## [15] scone_1.2.0                SummarizedExperiment_1.8.1
## [17] DelayedArray_0.4.1         matrixStats_0.53.1
## [19] Biobase_2.38.0             GenomicRanges_1.30.3
## [21] GenomeInfoDb_1.14.0        IRanges_2.12.0
## [23] S4Vectors_0.16.0           BiocGenerics_0.24.0
## [25] MASS_7.3-49
##
## loaded via a namespace (and not attached):
```

```
##   [1] backports_1.1.2              AnnotationHub_2.10.1
##   [3] aroma.light_3.8.0            lazyeval_0.2.1
##   [5] extRemes_2.0-8               splines_3.4.3
##   [7] BiocParallel_1.12.0          digest_0.6.15
##   [9] BiocInstaller_1.28.0         htmltools_0.3.6
##  [11] gdata_2.17.0                 magrittr_1.5
##  [13] RMTstat_0.3                  memoise_1.1.0
##  [15] cluster_2.0.6                mixtools_1.1.0
##  [17] limma_3.34.9                 Biostrings_2.46.0
##  [19] annotate_1.56.1              bayesm_3.1-0.1
##  [21] R.utils_2.6.0                rARPACK_0.11-0
##  [23] prettyunits_1.0.2            blob_1.1.0
##  [25] RcppArmadillo_0.8.400.0.0    RCurl_1.95-4.10
##  [27] hexbin_1.27.2                lme4_1.1-13
##  [29] genefilter_1.60.0            bindr_0.1
##  [31] brew_1.0-6                   survival_2.41-3
##  [33] glue_1.2.0                   zlibbioc_1.24.0
##  [35] XVector_0.18.0               compositions_1.40-1
##  [37] MatrixModels_0.4-1           car_2.1-4
##  [39] kernlab_0.9-25               Rook_1.1-1
##  [41] prabclus_2.2-6               DEoptimR_1.0-8
##  [43] SparseM_1.77                 DESeq_1.30.0
##  [45] mvtnorm_1.0-6                DBI_0.7
##  [47] edgeR_3.20.9                 Rcpp_0.12.15
##  [49] xtable_1.8-2                 progress_1.1.2
##  [51] bit_1.1-12                   mclust_5.2.3
##  [53] httr_1.3.1                   gplots_3.0.1
##  [55] fpc_2.1-10                   modeltools_0.2-21
##  [57] distillery_1.0-4             pkgconfig_2.0.1
##  [59] XML_3.98-1.10                R.methodsS3_1.7.1
##  [61] nnet_7.3-12                  locfit_1.5-9.1
##  [63] tidyselect_0.2.4             rlang_0.2.0
##  [65] tools_3.4.3                  RSQLite_2.0
##  [67] evaluate_0.10.1              stringr_1.3.0
##  [69] yaml_2.1.17                  knitr_1.20
##  [71] bit64_0.9-7                  robustbase_0.92-7
##  [73] caTools_1.17.1               EDASeq_2.12.0
##  [75] nlme_3.1-131.1               mime_0.5
##  [77] quantreg_5.35                R.oo_1.21.0
##  [79] biomaRt_2.34.2               pbkrtest_0.4-7
##  [81] compiler_3.4.3               curl_3.1
##  [83] interactiveDisplayBase_1.16.0 tibble_1.4.2
##  [85] geneplotter_1.56.0           stringi_1.1.6
##  [87] RSpectra_0.12-0              trimcluster_0.1-2
##  [89] ProtGenerics_1.10.0          Matrix_1.2-12
##  [91] nloptr_1.0.4                 tensorA_0.36
##  [93] pillar_1.2.1                 bitops_1.0-6
##  [95] httpuv_1.3.6.1               rtracklayer_1.38.3
##  [97] pcaMethods_1.70.0            R6_2.2.2
##  [99] latticeExtra_0.6-28          hwriter_1.3.2
## [101] RMySQL_0.10.14               ShortRead_1.36.1
## [103] KernSmooth_2.23-15           Lmoments_1.2-3
## [105] boot_1.3-20                  energy_1.7-2
## [107] gtools_3.5.0                 assertthat_0.2.0
```

```
## [109] rhdf5_2.22.0                 rprojroot_1.3-2
## [111] rjson_0.2.15                 RUVSeq_1.12.0
## [113] GenomicAlignments_1.14.1     Rsamtools_1.30.0
## [115] GenomeInfoDbData_1.0.0       diptest_0.75-7
## [117] mgcv_1.8-23                  grid_3.4.3
## [119] minqa_1.2.4                  class_7.3-14
## [121] rmarkdown_1.9                segmented_0.5-1.4
## [123] Cairo_1.5-9                  shiny_1.0.5
```