Multi-sample normalization techniques such as quantile normalization [1, 2] have become a standard and essential part of analysis pipelines for high-throughput data. Although it was originally developed for gene expression microarrays, it is now used across many different high-throughput applications including genotyping arrays, DNA Methylation, RNA Sequencing (RNA-Seq) and Chromatin Immunoprecipitation Sequencing (ChIP-Seq). These techniques transform the original raw data to remove unwanted technical variation. However, quantile normalization and other global normalization methods rely on assumptions about the data generation process that are not appropriate in some context. Until now, it has been left to the researcher to check for the appropriateness of these assumptions.

Quantile normalization assumes that the statistical distribution of each sample is the same. Normalization is achieved by forcing the observed distributions to be the same and the average distribution, obtained by taking the average of each quantile across samples, is used as the reference. This method has worked very well in practice but note that when the assumptions are not met, global changes in distribution, that may be of biological interest, will be wiped out and features that are not different across samples can be artificially induced. These types of assumptions are justified in many
biomedical applications, for example in gene expression studies in which only a minority of genes are expected to be differentially expressed. However, if, for example, a substantially higher percentage of genes are expected to be expressed in only one group of samples, it may not be appropriate to use global adjustment methods.

The quantro R-package can be used to test for global differences between groups of distributions which assess whether global normalization methods such as quantile normalization should be applied. Our method uses the raw unprocessed high-throughput data to test for global differences in the distributions across a set of groups. The main function `quantro()` will perform two tests:

1. An ANOVA to test if the medians of the distributions are different across groups. Differences across groups could be attributed to unwanted technical variation (such as batch effects) or real global biological variation. This is a helpful step for the user to verify if there is any technical variation unaccounted for.
2. A test for global differences between the distributions across groups which returns a test statistic called `quantroStat`. This test statistic is a ratio of two variances and is similar to the idea of ANOVA. The main idea is to compare the variability of distributions within groups relative to between groups. If the variability between groups is sufficiently larger than the variability within groups, then this suggests global adjustment methods may not be appropriate. As a default, we perform this test on the median normalized data, but the user may change this option.

## 2 Getting Started

Load the package in R

```r
library(quantro)
```

## 3 Data

### 3.1 flowSorted Data Example

To explore how to use `quantro()`, we use the `FlowSorted.DLPFC.450k` data package in Bioconductor [3]. This data set contains raw data objects of 58 Illumina 450K DNA methylation microarrays, formatted as `RGset` objects. The samples represent two different cellular populations of brain tissues on the same 29 individuals extracted using flow sorting. For more information on this data set, please see the `FlowSorted.DLPFC.450k` User’s Guide. For the purposes of this vignette, a `MethylSet` object from the `minfi` Bioconductor package [4] was created which is a subset of the rows from the original `FlowSorted.DLPFC.450k` data set. This `MethylSet` object is found in the `/data` folder and the script to create the object is found in `/inst`.

Here we will explore the distributions of these two cellular populations of brain tissue (`NeuN_pos` and `NeuN_neg`) and then test if there are global differences in the distributions across groups. First, load the `MethylSet` object (`flowSorted`) and compute the Beta values using the function `getBeta()` in the `minfi` Bioconductor package. We use an offset of 100 as this is the default used by Illumina.

```r
library(quantro)
library(minfi)
data(flowSorted)
p <- getBeta(flowSorted, offset = 100)
pd <- pData(flowSorted)
dim(p)
```
3.2 Plot distributions

quantro contains two functions to view the distributions of the samples of interest: `matdensity()` and `matboxplot()`. `matdensity()` computes the density for each sample (columns) and uses the `matplot()` function to plot all the densities. `matboxplot()` orders and colors the samples by a group level variable. These two functions use the RColorBrewer package and the brewer palettes can be changed using the arguments `brewer.n` and `brewer.name`.

The distributions of the two groups of cellular populations are shown here. The `NeuN_neg` samples are colored in green and the `NeuN_pos` are colored in red.

```r
matdensity(p, groupFactor = pd$CellType, xlab = " ", ylab = "density", main = "Beta Values", brewer.n = 8, brewer.name = "Dark2")
legend('top', c("NeuN_neg", "NeuN_pos"), col = c(1, 2), lty = 1, lwd = 3)
```
Beta Values

```
matboxplot(p, groupFactor = pd$CellType, xaxt = "n", main = "Beta Values")
```
4 Using the quantro() function

4.1 Input for quantro()

The quantro() function must have two objects as input:

- an object which is a data frame or matrix with observations (e.g. probes or genes) on the rows and samples as the columns.
- a groupFactor which represents the group level information about each sample. For example if the samples represent tumor and normal samples, provide quantro() with a factor representing which columns in the object are normal and tumor samples.

4.2 Running quantro()

In this example, the groups we are interested in comparing are contained in the CellType column in the pd dataset. To run the quantro() function, input the data object and the object containing the phenotypic data. Here we use the flowSorted data set as an example.
The quantro user’s guide

qtest <- quantro(object = p, groupFactor = pd$CellType)

## [quantro] Average medians of the distributions are not equal across groups.
## [quantro] Calculating the quantro test statistic.
## [quantro] No permutation testing performed.
## Use B > 0 for permutation testing.

qtest

## quantro: Test for global differences in distributions
## nGroups: 2
## nTotSamples: 58
## nSamplesinGroups: 29 29
## anovaPval: 0.01206
## quantroStat: 8.80735
## quantroPvalPerm: Use B > 0 for permutation testing.

The details related to the experiment can be extracted using the summary accessor function:

summary(qtest)

## $nGroups
## [1] 2
##
## $nTotSamples
## [1] 58
##
## $nSamplesinGroups
## NeuN_neg NeuN_pos
## 29 29

To assess if the medians of the distributions different across groups, we perform an ANOVA on the medians from the samples. Those results can be found using anova:

anova(qtest)

## Analysis of Variance Table
##
## Response: objectMedians
## Df  Sum Sq Mean Sq  F value Pr(>F)
## groupFactor 1 0.006919 0.0069194 6.7327 0.01206 *
## Residuals  56 0.057553 0.0010277
## ---
## Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

The full output can be seen. The test statistic produced from quantro() testing for global differences between distributions is given by quantroStat:

quantroStat(qtest)

## [1] 8.807348
The quantro user's guide

4.3 eSets

quantro() also can accept objects that inherit eSets such as an ExpressionSet or MethylSet. The groupFactor must still be provided.

```r
is(flowSorted, "MethylSet")
```

```r
## [1] TRUE
```

```r
qtest <- quantro(flowSorted, groupFactor = pData(flowSorted)$CellType)
```

```r
## [quantro] Average medians of the distributions are not equal across groups.
## [quantro] Calculating the quantro test statistic.
## [quantro] No permutation testing performed.
## Use B > 0 for permutation testing.
```

```r
qtest
```

```r
## quantro: Test for global differences in distributions
## nGroups: 2
## nTotSamples: 58
## nSamplesinGroups: 29 29
## anovaPval: 0.01206
## quantroStat: 8.80735
## quantroPvalPerm: Use B > 0 for permutation testing.
```

4.4 Output from quantro()

Elements in the S4 object from quantro() include:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| summary          | Returns a list of three elements related to a summary of the experiment: nGroups: number of groups  
                  | nTotSamples: total number of samples  
                  | nSamplesinGroups: number of samples in each group  |
| anova            | Results from an ANOVA to test if the average medians of the distributions are different across groups  |
| MSbetween        | Mean squared error between groups  |
| MSwithin         | Mean squared error within groups  |
| quantroStat      | A test statistic which is a ratio of the mean squared error between groups of distributions (MSbetween) to the mean squared error within groups of distributions (MSwithin)  |
| quantroStatPerm  | If B is not equal to 0, then a permutation test was performed to assess the statistical significance of quantroStat. These are the test statistics resulting from the permuted samples  |
| quantroPvalPerm  | If B is not equal to 0, then this is the p-value associated with the proportion of times the test statistics resulting from the permuted samples were larger than quantroStat  |
5 Assessing the statistical significance

To assess statistical significance of the test statistic, we use permutation testing. We use the foreach package which distribute the computations across multiple cross in a single machine or across multiple machines in a cluster. The user must pick how many permutations to perform where $B$ is the number of permutations. At each permutation of the samples, a test statistic is calculated. The proportion of test statistics ($\text{quantroStatPerm}$) that are larger than the $\text{quantroStat}$ is reported as the $\text{quantroPvalPerm}$. To use the foreach package, we first register a backend, in this case a machine with 4 cores.

```r
library(doParallel)
registerDoParallel(cores=4)
qtestPerm <- quantro(p, groupFactor = pd$CellType, B = 1000)

## [quantro] Average medians of the distributions are not equal across groups.
## [quantro] Calculating the quantro test statistic.
## [quantro] Starting permutation testing.
## [quantro] Parallelizing using 4 workers/cores
## (backend: doParallelMC, version: 1.0.10).

qtestPerm

## [quantro] Test for global differences in distributions
## nGroups: 2
## nTotSamples: 58
## nSamplesinGroups: 29 29
##anovaPval: 0.01206
##quantroStat: 8.80735
##quantroPvalPerm: 0.002
```

6 Visualizing the statistical significance from permutation tests

If permutation testing was used (i.e. specifying $B > 0$), then there is a second function in the package called `quantroPlot()` which will plot the test statistics of the permuted samples. The plot is a histogram of the null test statistics $\text{quantroStatPerm}$ from `quantro()` and the red line is the observed test statistic $\text{quantroStat}$ from `quantro()`.

```r
quantroPlot(qtestPerm)
```
The quantro user's guide

quantroStat = 8.807, quantroPvalPerm = 0.002

Additional options in the quantroPlot() function include:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xLab</td>
<td>the x-axis label</td>
</tr>
<tr>
<td>yLab</td>
<td>the y-axis label</td>
</tr>
<tr>
<td>mainLab</td>
<td>title of the histogram</td>
</tr>
<tr>
<td>binWidth</td>
<td>change the binwidth</td>
</tr>
</tbody>
</table>

7 SessionInfo

sessionInfo()

## R version 3.3.1 (2016-06-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.1 LTS
##
## locale:
## [1] LC_CTYPE=en_US.UTF-8    LC_NUMERIC=C    LC_TIME=en_US.UTF-8
## [4] LC_COLLATE=C            LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=en_US.UTF-8    LC_NAME=C       LC_ADDRESS=C
## [10] LC_TELEPHONE=C        LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
## attached base packages:
## [1] stats4 parallel stats    graphics grDevices utils    datasets methods
## [9] base
##
## other attached packages:
## [1] doParallel_1.0.10        minfi_1.20.0       bumphunter_1.14.0
## [4] locfit_1.5-9.1            iterators_1.0.8    foreach_1.4.3
## [7] Biostrings_2.42.0        XVector_0.14.0     SummarizedExperiment_1.4.0
## [10] GenomicRanges_1.26.0     GenomeInfoDb_1.10.0 IRanges_2.8.0
## [13] S4Vectors_0.12.0         Biobase_2.34.0     BiocGenerics_0.20.0
## [16] quantro_1.8.0
##
## loaded via a namespace (and not attached):
## [1] mclust_5.2                 base64_2.0        Rcpp_0.12.7
## [4] lattice_0.20-34           Rsamtools_1.26.0   digest_0.6.10
## [7] chron_2.3-47               R6_2.2.0          plyr_1.8.4
## [10] RSQLite_1.0.0              evaluate_0.10    highr_0.6
## [13] httr_1.2.1                 ggplot2_2.1.0     zlibbioc_1.20.0
## [16] GenomicFeatures_1.26.0    data.table_1.9.6  annotate_1.52.0
## [19] Matrix_1.2-7.1             preprocessCore_1.36.0 labeling_0.3
## [22] splines_3.3.1              BiocParallel_1.8.0 stringr_1.1.0
## [25] RCurl_1.95-4.8             biomaRt_2.30.0    munsell_0.4.3
## [28] compiler_3.3.1             rtracklayer_1.34.0 multtest_2.30.0
## [31] pkgmaker_0.22             openssl_0.9.4     GEOquery_2.40.0
## [34] quadprog_1.5-5             codetools_0.2-15  matrixStats_0.51.0
## [37] XML_3.98-1.4               reshape_0.8.5     GenomicAlignments_1.10.0
## [40] MASS_7.3-45                bitops_1.0-6      grid_3.3.1
## [43] nlme_3.1-128               xtable_1.8-2      gtable_0.2.0
## [46] registry_0.3              DBI_0.5-1         magrittr_1.5
## [49] formatR_1.4               scales_0.4.0      stringi_1.1.2
## [52] genefilter_1.56.0          doRNG_1.6         limma_3.30.0
## [55] normix_1.2-2               BiocStyle_2.2.0   RColorBrewer_1.1-2
## [58] siggenes_1.48.0            tools_3.3.1       illuminaio_0.16.0
## [61] rngtools_1.2.4             survival_2.39-5  AnnotationDbi_1.36.0
## [64] colorspace_1.2-7           beanplot_1.2      knitr_1.14

### References


