qusage Package Vignette

Christopher Bolen

1 September 2012, Revised 3 October 2015

Contents

1 Introduction

This package is an implementation the Quantitative Set Analysis for Gene Expression (qusage) method described in Yaari et al. (Nucleic Acids Res, 2013). This is a novel Gene Set Enrichment-type test, which is designed to provide a faster, more accurate, and easier to understand test for gene expression studies. The package is designed primarily for use with microarray expression data, but can be extended for use with count data as well.

qusage accounts for inter-gene correlations using a Variance Inflation Factor technique that extends the method proposed by Wu et al. (Nucleic Acids Res, 2012). In addition, rather than simply evaluating the deviation from a null hypothesis with a single number (a P value), qusage quantifies gene set activity with a complete probability density function (PDF). From this PDF, P values and confidence intervals can be easily extracted. Preserving the PDF also allows for post-hoc analysis (e.g., pair-wise comparisons of gene set activity) while maintaining statistical traceability. Finally, while qusage is compatible with individual gene statistics from existing methods (e.g., LIMMA), a Welch-based method is implemented that is shown to improve specificity.

2 Getting Started

The qusage package can be downloaded from the Bioconductor website using the following commands:

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

3 A Simple Example

The simplest way to run the qusage algorithm is by using the `qusage` function. This function is a wrapper for the three primary algorithms used in the package, `makeComparison` `aggregateGeneSet`, and `calcVIF`, each of which can be run individually if you want to have greater control over the parameters used in the algorithm. A discussion of each function can be found in section 4.

With default settings, `qusage` takes four inputs: `eset`, `labels`, `contrast`, & `geneSets`

1. `eset`

   This variable can be provided in two formats: either as an objet of class `ExpressionSet` containing log-normalized expression data (as created by the affy and lumi packages), or as matrix of log_2 gene expression measurements, with rows of genes and columns of samples. For best results, we recommend that this dataset be trimmed to remove genes with very low expression.

   For this example, we'll use part of a dataset from a study by Huang Y et al. (GEO ID: GSE30550), which looks at gene expression in 17 patients before and 77 hours after exposure to the influenza virus. This dataset has already been trimmed to remove genes with very low expression. Here is what the first few rows of our expression data look like:

```
> dim(eset)

[1] 4147   34

> eset[1:5,1:5]

       GSM757899 GSM757915 GSM757931 GSM757947 GSM757962
AKT3    4.600442  4.804845  4.454166  4.459647  4.589498
ACOT8   8.235681  7.667416  8.483951  8.471545  8.343489
GNPDA1  8.074555  8.732101  8.308964  8.499822  8.285027
CDH2    4.673311  5.088127  4.764211  4.630295  4.517296
TANK    9.039308  8.435497  8.835157  9.031125  8.951254
```

2. `labels`

   This is a character vector which contains labels describing the group structure in eset. The vector must have one entry for each column in the eset, and the labels must be legal variable names in R, meaning they can't contain spaces and must start with a letter. Often data will consist of two groups, described using "mock" and "treatment", or "healthy" and "disease". For our example data, we have 17 patients, each with a time point before and after viral exposure, which we will call "t0" and "t1". We generate the labels as follows:

```
> labels = c(rep("t0",17),rep("t1",17))
> labels

 [1] "t0" "t0" "t0" "t0" "t0" "t0" "t0" "t0" "t0" "t0" "t0" "t0" "t0"
[14] "t0" "t0" "t0" "t0" "t1" "t1" "t1" "t1" "t1" "t1" "t1" "t1" "t1"
[27] "t1" "t1" "t1" "t1" "t1" "t1" "t1" "t1"
```

3. `contrast`

   The contrast is a single character string indicating how you want to compare the groups in your data. In almost all cases, this will be of the form `"treatment-mock"`, which will test whether our gene sets are significantly different in treatment vs mock samples. We could also switch the order of the two groups, i.e. `"mock-treatment"`, which would result in negative fold changes for pathways that are higher in `treatment`. In our example, the contrast of interest is simply:

   ```
   > contrast = "t1-t0"
   ```

4. `geneSets`

   This is either a vector describing a single gene set, or a list of vectors representing a group of gene sets, such as the ones available from Broad's Molecular Signatures Database. In this first example, we will use a single list of IFN-stimulated genes, which looks like this:

   ```
   > ISG.geneSet[1:10]

    [1] "ADAR"    "ADM"     "ALDH1A1" "ANKFY1"  "APOL1"   "APOL2"
    [7] "APOL6"   "ARG2"    "ARHGEF3" "ATF3"
   ```

   Notice that both this vector and the rownames of our `eset` are using gene symbols. We could have just as easily used the original Affymetrix probe IDs or even converted everything to REFSEQ IDs. The only requirement is that the rownames of `eset` match the symbols used in `geneSets`. If they are in a different format, you will receive an error stating that all of your gene sets are of size 0.

With these four parameters, we can run `qusage` as follows:

```
> qs.results = qusage(eset, labels, contrast, ISG.geneSet)

Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.Done.
Calculating variance inflation factors...Done.
```

The `qusage` method will return a QSarray object containing statistical data on both the distributions of individual genes and on the pathway itself (for a full description of the data in the QSarray object, refer to section 8). While the results of this can be analyzed

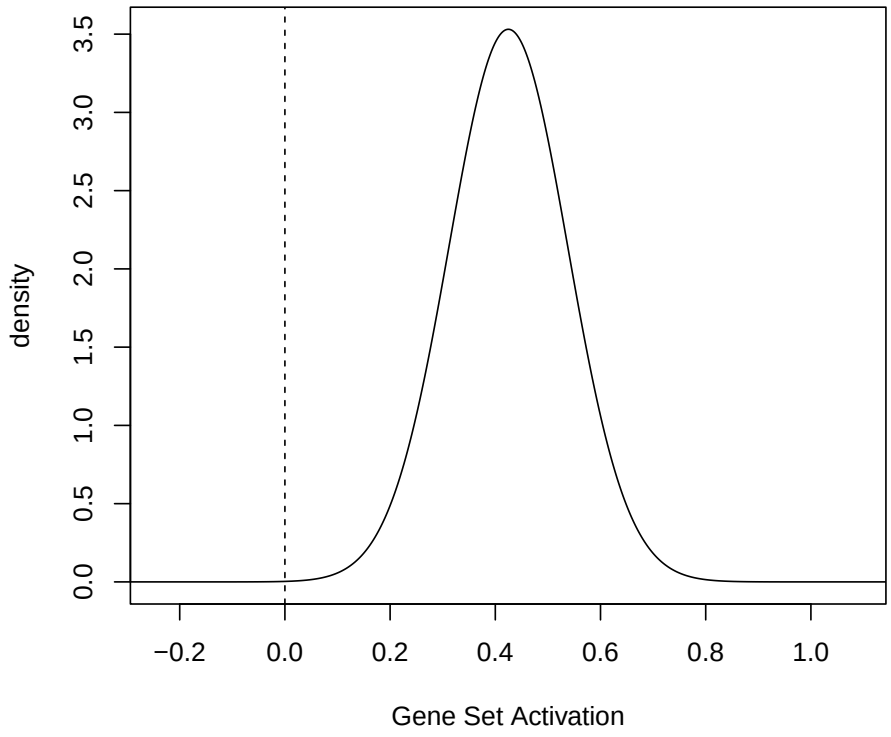
however you choose, there are a variety of functions available in the qusage package for exploring these results.

To find out if our `geneSet` is significantly enriched for this comparison, we use the `pdf.pVal` function, which calculates a p-value asking whether the convoluted distribution of our gene set is significantly enriched (i.e. that the mean fold change of the gene set is different from 0).

```
> pdf.pVal(qs.results)

[1] 0.0001818507
```

In order to view the probability density function, you can either use the `plotDensityCurves` function, or directly call `plot` on your QSarray object. In this case, `plot` will automatically call `plotDensityCurves` with default parameters. This will appear as follows:

```
> plot(qs.results, xlab="Gene Set Activation")
```



The `plotDensityCurves` function also invisibly returns the x- and y- coordinates of the PDFs being plotted in the form of a list of matrices, which can be used to generate custom plots for the PDFs. For more information on custom plots, refer to section 7).

3.1 Multiple Gene Sets

Most often, qusage is not going to be run with a single gene set. Instead, it is most useful when run on a group of gene sets as a tool for hypothesis discovery.

To demonstrate this, we will use MsigDB's KEGG database (which can be downloaded at `www.broadinstitute.org/gsea/msigdb`). We can read this file in using the `read.gmt` function. With the resulting object, we run `qusage` exactly as before:

```
>   MSIG.geneSets = read.gmt("c2.cp.kegg.v4.0.symbols.gmt")

>   summary(MSIG.geneSets[1:5])

                                            Length Class  Mode
KEGG_GLYCOLYSIS_GLUCONEOGENESIS             62     -none- character
KEGG_CITRATE_CYCLE_TCA_CYCLE                32     -none- character
KEGG_PENTOSE_PHOSPHATE_PATHWAY              27     -none- character
KEGG_PENTOSE_AND_GLUCURONATE_INTERCONVERSIONS 28     -none- character
KEGG_FRUCTOSE_AND_MANNOSE_METABOLISM        34     -none- character

>   MSIG.geneSets[2]

$KEGG_CITRATE_CYCLE_TCA_CYCLE
 [1] "LOC642502" "OGDHL"     "OGDH"      "PDHB"      "IDH3G"
 [6] "LOC283398" "IDH2"      "IDH1"      "PDHA2"     "PDHA1"
[11] "SUCLA2"    "FH"        "DLST"      "ACO2"      "SUCLG2"
[16] "ACO1"      "SUCLG1"    "CS"        "IDH3B"     "ACLY"
[21] "DLAT"      "PCK2"      "IDH3A"     "PCK1"      "SDHA"
[26] "SDHB"      "SDHC"      "SDHD"      "DLD"       "MDH2"
[31] "PC"        "MDH1"

>   qs.results.msig = qusage(eset, labels, contrast, MSIG.geneSets)

Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.....................................Done.
Calculating variance inflation factors...Done.

>   numPathways(qs.results.msig)

[1] 186
```

The `qs.results.msig` object now contains information for each of the gene sets in `MSIG.geneSets`. Once again, we can get the p-values for the comparison by using the `pdf.pVal` function. However, because we have calculated such a large number of p-values, it's generally not valid to look at them without performing some kind of multiple hypothesis testing. Below, we calculate the p-values and use R's built-in `p.adjust` method to calculate the FDR values.

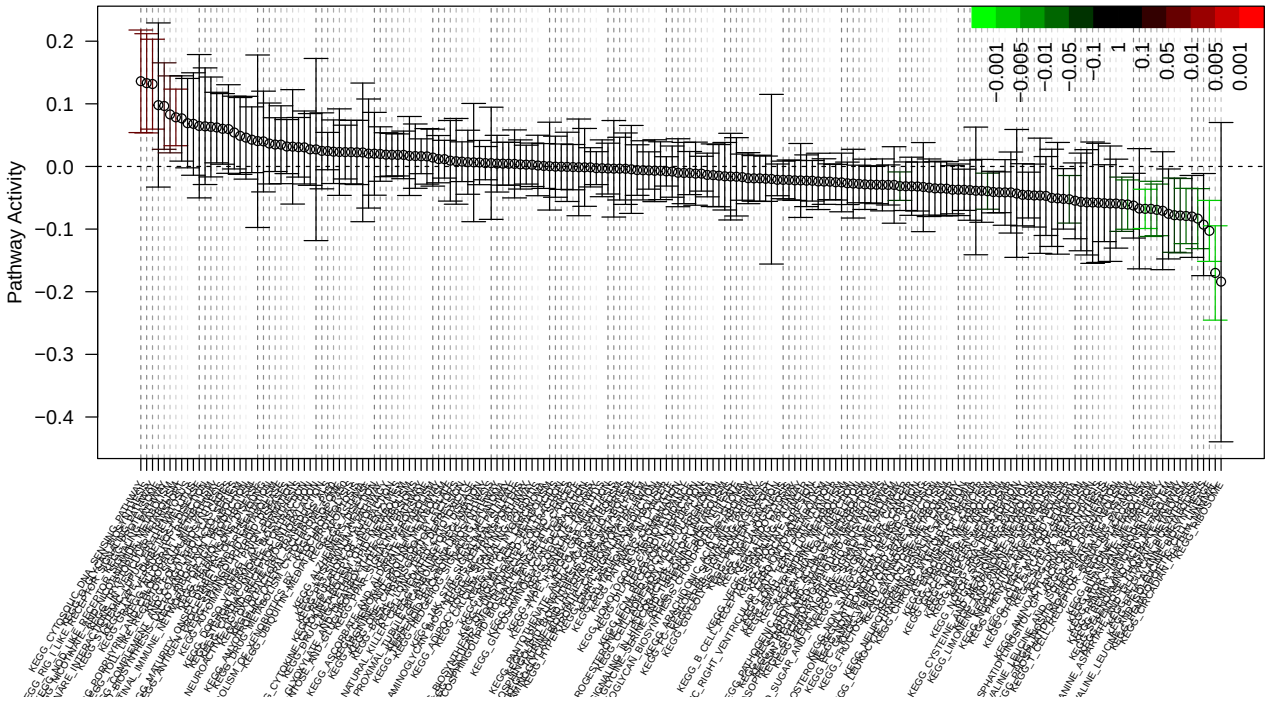
```
>   p.vals = pdf.pVal(qs.results.msig)
>   head(p.vals)
[1] 0.288368617 0.970277527 0.009600519 0.528030282 0.112549381
[6] 0.106747491
>   q.vals = p.adjust(p.vals, method="fdr")
>   head(q.vals)
[1] 0.54178346 0.98776294 0.09398403 0.75548948 0.36091243 0.36091243
```

All of this data is also presented using the `qsTable` function, which returns a table ordered according to the significance of each gene set. For our data, the results are as follows:

```
>   qsTable(qs.results.msig, number=10)
```

	pathway.name	log.fold.change	p.Value	FDR
135	KEGG_CIRCADIAN_RHYTHM_MAMMAL	-0.16994258	1.326970e-05	0.002244544
22	KEGG_HISTIDINE_METABOLISM	-0.06765522	3.205426e-05	0.002244544
127	KEGG_HEMATOPOIETIC_CELL_LINEAGE	-0.10276783	3.620233e-05	0.002244544
124	KEGG_RIG_I_LIKE_RECEPTOR_SIGNALING_PATHWAY	0.13163298	3.774956e-04	0.017553544
15	KEGG_ALANINE_ASPARTATE_AND_GLUTAMATE_METABOLISM	-0.07912495	4.845073e-04	0.018023671
26	KEGG_BETA_ALANINE_METABOLISM	-0.08318178	7.196554e-04	0.020600285
25	KEGG_TRYPTOPHAN_METABOLISM	0.07844123	7.752796e-04	0.020600285
102	KEGG_MTOR_SIGNALING_PATHWAY	-0.06927698	1.002185e-03	0.022430061
33	KEGG_O_GLYCAN_BIOSYNTHESIS	0.13299596	1.085326e-03	0.022430061
125	KEGG_CYTOSOLIC_DNA_SENSING_PATHWAY	0.13611612	1.229876e-03	0.022875703

We can also try plotting the data again using a call to `plot(qs.results.msig)`.



However, this time the qusage package creates a different kind of plot. This is because, by default, when the QSarray object contains data on more than ten pathways, `plot` will automatically use the function `plotCIs`. This function calculates the 95% confidence interval for each gene set using the `calcBayesCI` function and plots it along with the mean fold change of each gene set. This is usually a more useful view when there are a large number of gene sets, but a plot of the PDFs can still be generated using a call to `plotDensityCurves`.

3.2 Paired Samples

In some cases, samples in a gene expression experiment may be paired with each other. For example, we may have pairs of samples from the same patient both before and after treatment. Because there may be patient-specific effects in the gene expression data, we want to use this information when we calculate pathway activation. In qusage, all we have to do to indicate paired samples is supply a `pairVector` containing information on which samples should be paired together. The vector should be the same length as `eset`, and it can be either a numeric vector or a set of factors which uniquely describe each pair. A simpler way to think about the pair vector is to consider it as a vector of patient IDs for each column in eset.

Although for simplicity we have been ignoring it thus far, our gene expression data is actually a paired dataset as well. We have a set of 17 patients with two samples each (t0 and t1). For our `pairVector`, we will use the numbers 1:17 to represent our patients, as follows:

```
>   pairs = c(1:17,1:17)
>   pairs

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  1  2  3  4  5
[23]  6  7  8  9 10 11 12 13 14 15 16 17
```

Then, we can simply run the `qusage` function in a paired manner by adding the argument `pairVector=pairs` (note: we don't actually show this here, but you can see it in action in the next section).

3.3 Two-Way Comparisons

The true power of the qusage method comes when you have results from more than one run of `qusage` that you wish to compare. If a dataset contains more than two groups, it's common to compare these groups in many different ways, and the ablility to compare the results of these comparisons with each other is extremely useful. These two-way comparisons are important when examining differences in the response characteristics of various groups (e.g. to answer the question of "does group A respond to treatment in a different way than group B?").

To illustrate how to make these comparisons, we will once again be using the Influenza response data. Although in the previous examples we have grouped all patients together, the patients can be classified into two distinct groups based on their response to the virus. Of the 17 patients in the study, 9 of them had a symptomatic response, and 8 had an asymptomatic response. While we could compare these two groups of patients directly at the two time points, it might be more interesting to see if the response to virus over the time-course of the experiment is different between these two groups.

Before we can compare these groups, however, we will need to remake the `labels` in order to describe the new groups in our data. For this example, we have the response information stored in a variable called `resp`, which is just a vector with either "sx" (for symptomatic patients) or "asx" (for asymptomatic patients):

```
>   resp

 [1] sx  asx asx asx sx  sx  sx  sx  asx sx  asx sx  sx  asx sx  asx
[17] asx sx  asx asx asx sx  sx  sx  sx  asx sx  asx sx  sx  asx sx
[33] asx asx
Levels: asx sx

>   twoWay.labels = paste(resp, labels, sep=".")
>   twoWay.labels

 [1] "sx.t0"  "asx.t0" "asx.t0" "asx.t0" "sx.t0"  "sx.t0"  "sx.t0"
 [8] "sx.t0"  "asx.t0" "sx.t0"  "asx.t0" "sx.t0"  "sx.t0"  "asx.t0"
[15] "sx.t0"  "asx.t0" "asx.t0" "sx.t1"  "asx.t1" "asx.t1" "asx.t1"
[22] "sx.t1"  "sx.t1"  "sx.t1"  "sx.t1"  "asx.t1" "sx.t1"  "asx.t1"
[29] "sx.t1"  "sx.t1"  "asx.t1" "sx.t1"  "asx.t1" "asx.t1"
```

Now, we run `qusage` twice; once for each comparison that we are interested in (note: we're also including information on the sample pairs, which was described in section 3.2).

```
>   sx.results  = qusage(eset, twoWay.labels, "sx.t1-sx.t0",
+                        MSIG.geneSets, pairVector=pairs)

Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.....................................Done.
Calculating variance inflation factors...Done.

>   asx.results = qusage(eset, twoWay.labels, "asx.t1-asx.t0",
+                        MSIG.geneSets, pairVector=pairs)

Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.....................................Done.
Calculating variance inflation factors...Done.
```

We could examine the information contained in these two runs using the functions described above, but what we are most interested in is whether or not there is a significant difference between the responses of these two groups. We can directly compare the results in these two groups using the `twoCurve.pVal` function. This will return a p-value for the comparison of each gene set in the two objects.

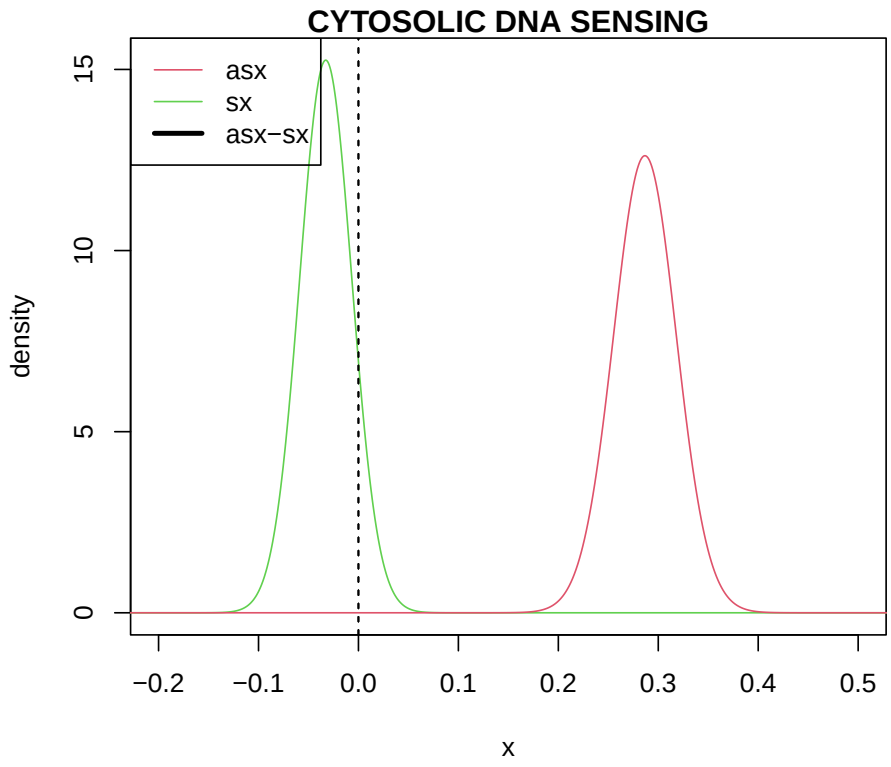
```
>   p.vals = twoCurve.pVal(sx.results,asx.results)
>   head(p.vals)

[1] 0.63040128 0.17410353 0.57256998 0.84789898 0.09554015 0.59914985
```

NOTE: In `qusage` version 2.0, we have added a streamlined version of the same methodology that also calculates the PDF of the difference between these two comparisons. In this case, a more complex comparison can be specified, indicating that the resulting PDF should be the difference between the symptomatic PDF and the asymptomatic PDF.

```
>   asx.vs.sx = qusage(eset, twoWay.labels, "(sx.t1-sx.t0) - (asx.t1-asx.t0)",
+                    MSIG.geneSets, pairVector=pairs)

Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.....................................Done.
Calculating variance inflation factors...Done.
Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.....................................Done.
Calculating variance inflation factors...Done.

>   p.vals = pdf.pVal(asx.vs.sx)
>   head(p.vals)

[1] 0.63529792 0.18139880 0.55901371 0.86541377 0.09953021 0.61440351
```

It turns out that gene set 125 – the `"CYTOSOLIC DNA SENSING"` pathway – has the most significant difference in activation. There are multiple ways we could examine this individual gene set, but perhaps the simplest way is to again use the `plotDensityCurves` function. Notice here that we are specifying which pathway to plot using the `path.index` parameter.

```
>   plotDensityCurves(asx.results,path.index=125,xlim=c(-0.2,0.5),
+                     col=3,main="CYTOSOLIC DNA SENSING")
>   plotDensityCurves(sx.results,path.index=125,col=2,add=T)
>   plotDensityCurves(asx.vs.sx, path.index=125,col=1,add=T, lwd=3)
>   legend("topleft", legend=c("asx","sx","asx-sx"),lty=1,col=c(2:3,1),
+          lwd=c(1,1,3))
```



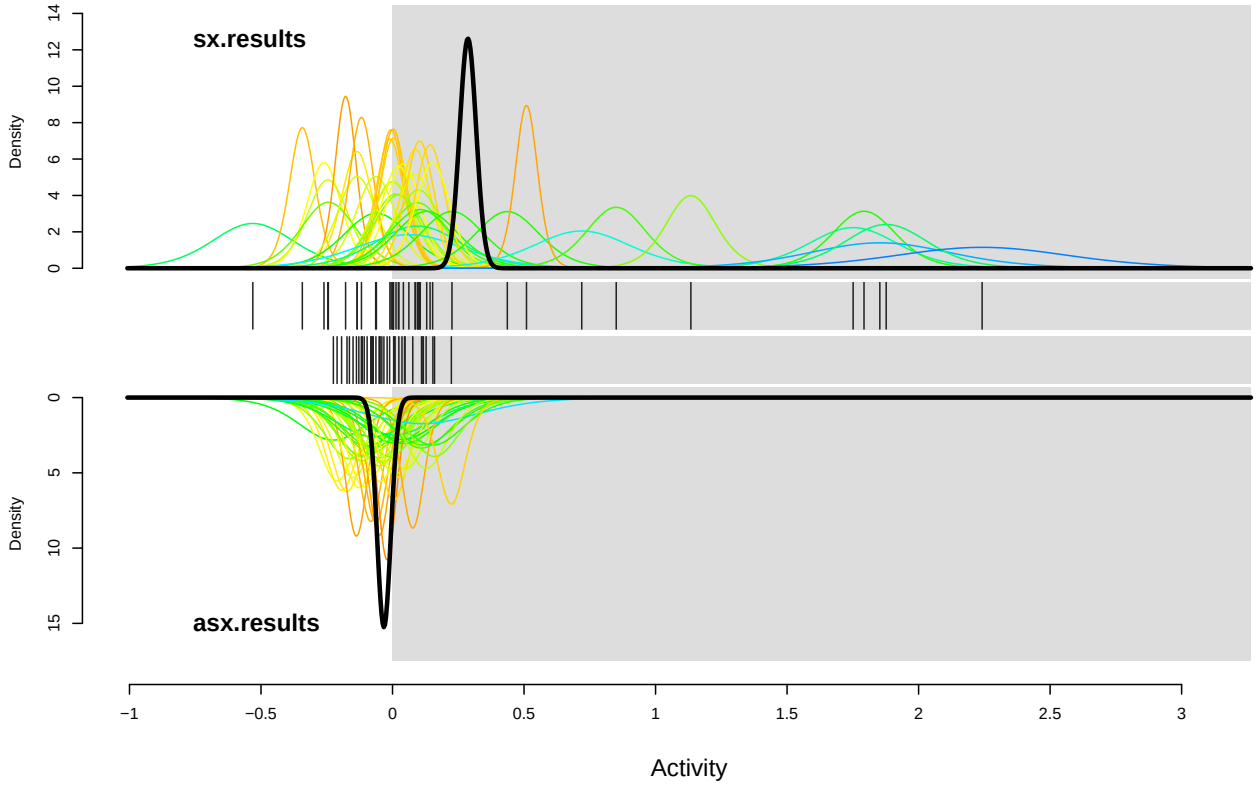
The results here are pretty interesting, actually. What we're seeing is that, in symptomatic patients, the Cytosolic DNA Sensing pathway is extremely active, while the asymptomatic patients essentially have no activity at all.

3.3.1 Individual Gene Plots

Often the next question when an interesting pathway is found is what's happening with the individual genes in that pathway. Looking at the individual genes can give some idea as to how consistent the patterns of regulation in the data are, or whether the significance levels are being driven by only one or two individual genes. Because the gene set PDF in this package is actually a convolution of the PDFs of each individual gene, we can actually look at the distributions of each gene directly to understand where our variation comes from.

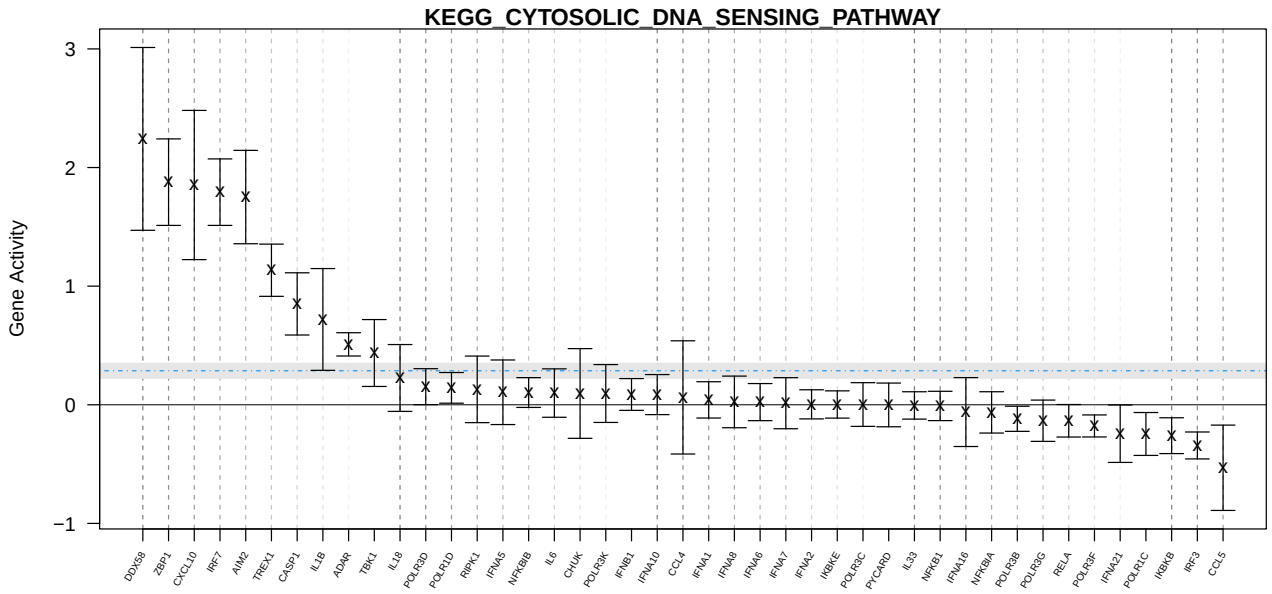
qusage provides two plotting functions which can be used to visualize this data. The first is the `plotGeneSetDistributions` function, which plots out the PDF for both the pathway and for that pathway's individual genes. The plot, which can be seen below, includes thin colored curves for each gene (color coded according to their standard deviation), as well as a thicker black curve representing the PDF of the entire dataset. Here we plot the results for both the symptomatic patients and the asymptomatic patients side-by-side.

```
> plotGeneSetDistributions(sx.results,asx.results,path.index=125)
```



Again, we can see that the symptomatic patients have much more activity than the asymptomatic patients, a pattern which seems generally consistent in many of the genes in this pathway. This plot is useful for giving a general idea of how the genes in the pathway act. However, it would also be useful to find out which genes are the most active, and since the `plotGeneSetDistributions` plot does not provide labels for the individual genes, we must use a separate function: `plotCIsGenes` to see which genes are most active. This function produces a very similar plot to the `plotCIs` function, but in this case, it will plot the means and confidence intervals for each individual gene in a pathway. To generate this plot for the CYTOSOLIC DNA SENSING pathway, we will use the following code:

```
> par(mar=c(8,4,1,2))
> plotCIsGenes(sx.results,path.index=125)
```



3.4 Running qusage with small sample sizes

Although qusage does not require as many samples as permutation-based methods, the current implementation requires at least three degrees of freedom. As the variance of a t distribution is $\frac{\nu}{\nu-2}$, this will diverge for $\nu \to 2$. Hence, in cases where the degrees of freedom falls below three, QuSAGE artificially set the degrees of freedom to be three and issues a warning message that the results should be interpreted with caution. It is imortant to keep in mind that in this case, the p-values will be inflated, and may not accurately reflect the underlying data.

A second problem with small sample sizes is the large range over which the gene set PDF must be sampled. Whereas a t-distribution with 25 degrees of freedom is sampled from $(-10, 10)$, a distribution with four degrees of freedom must be sampled from $(-416, 416)$. When using the default number of points (`n.points=4096`), this can result in inadequate sampling of the distribution and inaccurate estimates of pathway significance. In order to prevent this, you must increase the n.points parameter when running the aggregateGeneSet function (this is done automatically when using the `qusage` function if the degrees of freedom are below 5). However, note that the run time of aggregateGeneSet increases linearly with the number of points, so increasing this parameter may significantly slow down the algorithm.

4 The qusage Algorithm

The qusage algorithm involves two basic steps: 1) Calculating gene-level comparisons, and 2) Aggregating gene-level data for each gene set. In addition, there is one optional (but highly recommended) step, 3) determining the Variance Inflation Factor (VIF), which corrects for gene-gene correlations in the gene sets. In section 3, we discussed how to use the `qusage` algorithm to run all three steps together with the default parameters. Below, we discuss each function and their parameters in detail, and look at examples of how to run each function separately.

4.1 Calculating Gene Level Comparisons

This section describes the `makeComparison` function, which is the first step in the qusage algorithm. This function defines the t-distributions for every gene in `eset` by calculating the fold change and standard deviation between two groups of samples. The function has three required parameters: `eset`, `labels`, and `contrast`, which were described in section 3. A `pairVector` can also be provided to specify that the comparisons should be done in a paired manner (see section 3.2). The `makeComparison` function will return a QSarray object containing information on the means and standard deviations of each gene, which will be used in the subsequent functions to calculate pathway enrichment.

`var.equal`

There are two main algorithms used to calculate a t-distribution from two groups of samples, known as the "Pooled Variance" and "Welch's approximate" approach. In the first method (the pooled variance approach), the variances of the two groups are assumed to be equal, and the difference in the mean expression values is modeled as a Student's t distribution. The second method, proposed by B. L. Welch (Biometrika, 1947), is also based on a Student's t distribution, but relaxes the assumption of equal variances. Most recent gene expression studies take the first approach and assume that the variances of the two groups are equal. For example, this assumption underlies the individual gene statistics in the widely used LIMMA package in Bioconductor.

The pooled variance approach can slightly improve sensitivity (if the equal variance assumption holds), and is easily compatible with linear models and ANOVA. However, this approach can be extremely biased when the assumption of equal variances is broken, which we find is often the case in many real gene expression data sets. By default, the `makeComparison` function will assume unequal variances, but this behavior can be changed by specifying the parameter `var.equal = TRUE`. If `var.equal` is set to `TRUE`, the t-distribution is actually calculated using the linear model approach implemented by LIMMA. This may be appealing in some cases, because using LIMMA's linear model framework makes more complicated contrasts possible, while also making it possible to use LIMMA's framework for the Bayesian estimation of gene variances, as described below.

`bayesEstimation`

If `var.equal` is set to `TRUE` and LIMMA is used to calculate the t-distributions, we can use some of the additional features of LIMMA when calculating the parameters for each gene. The `bayesEstimation` parameter is used by LIMMA to optionally apply a bayesian step in the estimation of the pooled variances of each gene. This bayesian step is designed to shrink the variances of the genes on the chip towards a common value. In some cases, this may help to give slightly more stable values for the variances, especially in cases where there are very few samples. The Bayesian step also prevents any genes from having a variance of 0, which would otherwise lead to divergence of the t-distribution. For complete details of how the bayesian estimation step works, refer to the documentation of the LIMMA package.

`min.variance.factor`

If `var.equal` is set to `FALSE`, the `min.variance.factor` can be used as an alternative to the Bayesian estimation step described above. The min.variance.factor is designed to add a very small value to the variance of each gene on the chip in order to prevent any variances from being set to 0. The default, which is set to 10^-8, is small enough that it will not impact the variance of expressed genes, but will prevent the t-distribution from diverging on genes with variance 0.

4.2 Aggregating gene-level data into gene set PDFs

The second step of the `qusage` algorithm is the aggregation step, which convolutes individual gene t-distributions into a single PDF for each gene set. This step is performed by the `aggregateGeneSet` function. The function has two required parameters: a `QSarray` object such as the one returned by `makeComparison` (see section 4.1), and a list of `geneSets`. As mentioned in section 3, the geneSets parameter can either be a character vector containing a single gene set, or a list of character vectors representing multiple different gene sets.

This function also contains an option to change the number of points that the convoluted PDF is sampled over. By default the parameter `n.points` is set to 2^{12}, or 4096 points, which will give very accurate p-values in most cases. Sampling at more points will increase the accuracy of the resulting p-values, but will also linearly increase the amount of time needed to calculate the result. In many cases, as few as 1/4 this number of points can be used without seriously affecting the accuracy of the resulting p-values.

The PDF for each individual gene set is generated by using numerical convolution applied to the individual gene PDFs. Briefly, a Fast Fourier Transform (FFT) is calculated for each individual gene PDF to arrive at a k-component vector. The product of each component across all of the genes is then taken to arrive at a new k-component vector for the gene set. The real part of the resulting product is then transformed back to a PDF using a reverse FFT, and assured to be normalized and centered around zero. The mean of the combined PDF is simply the mean fold change of the individual genes.

The range for sampling is determined by the lowest degrees of freedom of the individual genes, such that at most 10^{-8} of the cumulative distribution at the tails are excluded (i.e., assumed to be 0). For example, when $\nu = 3$, the range is $(-480, 480)$, and when $\nu = 120$, the range is $(-6, 6)$.

Technically, the output of this step is the PDF of the *sum* of differences in expressions over all genes in the gene set under the assumption that the genes are independent. In order to estimate the *mean* differential expression PDF, this distribution is scaled by a factor of $1/N$, where N is the number of genes in the gene set. The resulting PDFs of the input gene sets are stored as a matrix in `path.PDF` slot of the returned QSarray object. However, the x-coordinates for these PDFs are not stored in the QSarray object, and must be calculated using the `getXcoords` function.

4.3 The Variance Inflation Factor

The expression measurements of individual genes in a set are almost always correlated. As such, it is important to take these correlations into account when testing for gene set enrichment. In qusage, we build off of a technique proposed by Wu et al. (Nucleic Acids Res, 2012), which calculates a Variance Inflation Factor (VIF) for each gene set based on the correlation of the genes in that set. This method, referred to as CAMERA, uses the linear model framework created by LIMMA to calculate gene-gene correlations, but consequently it must assume equal variance not only between all groups in the dataset, but also across each gene in the gene set. While this assumption leads to a slightly more computationally efficient VIF calculation, it is not valid for most gene sets, and its violation can greatly impact specificity. In the qusage paper (Yaari et al., NAR, 2013), we show that this assumption of equal variances leads to over-estimation of the VIF, and consequently an overall loss of power.

Although our recommendation is to use the qusage algorithm for calculating the VIF, the `calcVIF` function includes the option to use either the CAMERA method (as implemented in the `limma` package in R) or the qusage method. If the parameter `useCAMERA=TRUE` is specified when running `calcVIF`, the CAMERA method will be used when calculating the VIF for each gene set. By default, this will only be used if `var.equal` was set to `TRUE` during the original call to `makeComparison`.

`useAllData`

The `calcVIF` algorithm contains an additional parameter, `useAllData`, which specifies whether all of the groups in `eset` should be used when calculating the VIF using the qusage algorithm, or if only the two groups used in the original comparison should be used. By default (`useAllData=TRUE`), all of the data will be used to calculate the VIF. In most cases, using more data will increase the accuracy of the VIF calculation, especially in cases where individual groups are very small. However, it may not always be valid to include all of the data when calculating gene-gene correlations (e.g. if the data set contains expression values from more than one species), so it may be useful to

limit what groups are used in the VIF calculation. Note that this parameter is only used if `useCAMERA` is `FALSE`, as the CAMERA algorithm automatically uses all data in the eset provided.

5 Using qusage with Generalized Linear Mixed Models

The qusage methodology was extended by Turner et al (BMC Bioinformatics 2015) to incorporate general linear mixed models available from the nlme package. The function `qgen` is now available through the qusage package to implement the extension. The main difference lies in that the t-statistic information in qusage are based on two sample t-tests that cannot incorporate additional covariates to adjust the fold change estimates due to potential confounders that can exist within observational studies. The qgen algorithm uses the t-statistic information derived from a linear mixed model and estimates the VIFs using the raw residuals rather than group labelings as this can yield biased VIF estimates.

5.1 The qgen function

Although the general structure of the `qgen` function is similar to `qusage`, there are more inputs required in order for the user to incorporate added complexity in the statistical modeling step.

With default settings, `qgen` takes six inputs: `eset`, `design`,`fixed`, `contrast.factor`, `contrast`, & `geneSets`, however, additional inputs are often necessary to handle repeated measures data through `random` and `correlations`.

1. `eset`

   This variable is a matrix of log2 gene expression measurements, with rows of genes and columns of samples. For best results, we recommend that this dataset be trimmed to remove genes with very low expression.

   For this example, we'll use the entire dataset from the study by Huang Y et al. (GEO ID: GSE30550), which looks at gene expression in 17 patients across a large number of time points in hours before and after exposure to the influenza virus. In addition to time points, each patient's condition is also defined as either "asymptomatic" (asx) or "symptomatic" (sx) to the exposure.

```
> dim(eset.full)

[1] 4147  252

> eset.full[1:5,1:5]

       GSM757899 GSM757900 GSM757901 GSM757902 GSM757903
AKT3    4.600442  4.436855  4.338744  4.494653  4.332082
ACOT8   8.235681  8.272591  8.242254  8.518036  7.970552
GNPDA1  8.074555  8.165120  8.364384  8.420914  8.405667
```

```
CDH2    4.673311  4.835745  4.625260  4.687651  5.486241
TANK    9.039308  8.843692  9.186436  9.328579  8.508230
```

2. `design`

   Rather than using a `labels` vector describing the group structure in eset, a dataframe must be provided that contains the information needed for the linear mixed model. This can contain the fixed effect factors such as groups, continuous explanatory variables, as well as donor specific identifiers and time points necessary to model repeated measure through random effects or residual side covariance structures. It is strongly recommended that an addition column is included corresponding to the sample names of eset. With this information, qgen checks for any discrepancies that exist between eset and design. Any samples that do not exist in both files are removed. If such an identifier is not specified using the `design.sampleid` option, qgen assumes that the order in the design file corresponds to the order of the samples given in `eset`. Here are the first few rows of the design file for FluEset:

   ```
   > head(flu.meta)

      SampleID Subject Hours Hours.Numeric Condition Gender Age
   2 GSM757899  flu001     0             0        sx Female  29
   3 GSM757900  flu001     5             5        sx Female  29
   4 GSM757901  flu001    12            12        sx Female  29
   5 GSM757902  flu001    21            21        sx Female  29
   6 GSM757903  flu001    29            29        sx Female  29
   7 GSM757904  flu001    36            36        sx Female  29
   ```

3. `fixed`

   Fixed is a one sided formula specifying the fixed effects the user wishes to include in the linear model. Typical linear models are of the two sided formula `y X1+X2+X3`, but since the response variable is defined by the expression values of eset, only the right hand side is needed and take the general form, `X1+X2+X3`.

   For this example the primary interest is determining if there are changes over time with respect to the baseline values both for the asymptomatic and symptomatic groups. Additional sample information such as age and gender could also be included to account for any additional source of variability or if they were confounded with primary variables of interest. We include fixed effects for subject's condition, time, and their interaction as well as including both age and gender.

   ```
   > fixedeffects <- ~Hours+Condition+Condition*Hours+Age+Gender
   ```

4. `contrast.factor`

   This one-sided fromula refers to one of the fixed effects components in the fixed input that tells qgen what contrast the user is interested in making a comparison. So if the user wishes to compare symptomatic versus asymptomatic regardless of time, then the user would simply provide ∼Hours. If the user would like to compare symptomatic versus asymptomatic at hour 77 then the user would provide the interaction component ∼Hours*Condition. For our example we wish to compare changes at Hour 77 versus Hour 0 among the symptomatic group only, this corresponds to:

   ```
   > con.fac <- ~Condition*Hours
   ```

5. `contrast`

   This is a character string that serves the same purpose in the original `qusage` function. It specifies which levels of contrast.factor the user wishes to compare. This is usually of the form "TrtA - TrtB". Since our comparison involves the combination of two factor levels, the levels just need to be concatenated in the corresponding order. So to compare Hour 77 vs Hour 0 among the symptomatic group, the corresponding contrast can be written as:

   ```
   > contrast = "sx77 - sx0"
   ```

   The one-sided formula in contrast.factor, and the corresponding statement in contrast are fed directly to the `makeContrasts` of the limma package to obtain the appropriate sequence of 0's and 1's for the contrast. One caveat to the naming convention of contrast is that the names in contrast must be valid R names. So if one were to specify the contrast.factor= Hours*Condition and contrast= "45Symp - 0Symp", an error will return as `makeContrasts` only allows factor level names to be valid R names and cannot start with a numeric value.

6. `geneSets`

   This is exactly the same as for the original qusage function, either a vector describing a single gene set, or a list of vectors representing a group of gene sets. For this example, we will continue to use the IFN-stimulated genes which looks like this:

   ```
   > ISG.geneSet[1:10]

    [1] "ADAR"    "ADM"     "ALDH1A1" "ANKFY1"  "APOL1"   "APOL2"
    [7] "APOL6"   "ARG2"    "ARHGEF3" "ATF3"
   ```

Notice that both this vector and the rownames of our `eset` are using gene symbols. We could have just as easily used the original Affymetrix probe IDs or even converted everything to REFSEQ IDs. The only requirement is that the rownames of `eset` match the symbols used in `geneSets`. If they are in a different format, you will receive an error stating that all of your gene sets are of size 0.

With these 6 parameters specified, qgen will run the analysis using ordinary least squares regression (the default unless additional options are specified).

5.2 Qgen with random effects

The real power of this approach is running models that can handle the repeated measures nature of the study design. This can be done by specifying either the `random` parameter, the `correlation` parameter, or a combination of both. Both of these options are directly fed to the linear model functions provided in the nlme package. See `lme` and `gls` functions for additional details.

In order to generate a model with random effects, an optional formula must be supplied to `random`. For a simple repeated measures study design such as this, the form is usually 1|g where g specifies the donor or subject ids. To run the qgen analysis using a random effect, the following command can be called:

```
> qusage.result.random<-qgen(eset.full,flu.meta,
+                            fixed=fixedeffects,
+                            contrast.factor=con.fac,
+                            contrast=contrast,
+                            geneSets=ISG.geneSet,
+                            random=~1|Subject,
+                            design.sampleid="SampleID")

All samples present in design
Samples in Design and Expression Match
Running row by row modeling.
Percent Complete:10%...20%...30%...40%...50%...60%...70%...80%...90%...100%...
Aggregating gene data for gene sets.Done.
Calculating VIF's on residual matrix.
Q-Gen analysis complete.
```

The results of this analysis can then be examined using the same functions described in section 3.

```
> qsTable(qusage.result.random)

  pathway.name log.fold.change p.Value FDR
1     geneSets       0.7989711       0   0
```

5.3 Qgen with correlations

An optional nlme corStruct object to specify within group correlation structure through the residual covariance matrix and is passed directly to the lme function within the nlme package. There are many modeling options for the user to pick from if this is desired, we strongly urge reading the documentation within the nlme package on CorStruct objects. As an example suppose we assume that the correlations that exist over the repeated measure time points die off over time exponentially. The corresponding model to incorporate this strategy would yield

```
> corStruct = corExp(value=3,~Hours.Numeric|Subject)
> corStruct
> qusage.result.corstruct<-qgen(eset.full,flu.meta,
+                               fixed=fixedeffects,
+                               contrast.factor=con.fac,
+                               contrast=contrast,
+                               geneSets=ISG.geneSet,
+                               correlation=corStruct,
+                               design.sampleid="SampleID")
```

The results of both the correlation analysis and the random effects analysis can be compared directly, revealing that in both cases, there seems to be an approximately ($2^{0.8} = 1.75$)-fold increase in the expression of ISGs at day 77 of infection. However, in this case, taking random effects into account leads to a more significant p-value, resulting from lower variance in the estimate.

```
>   plotDensityCurves(qusage.result.random, main="ISG_geneSet",
+                     xlim=c(0,1.2), xlab="Day 77 - Day 0", col="red")
>   plotDensityCurves(qusage.result.corstruct, col="blue",add=T)
```

5.3.1 Additional qgen details

The computational speed of qgen depends on a couple of factors, in particular, the complexity of the linear model, the total number of genes within the gene sets, the total number of genesets, and the total number of samples. To save computational time, qgen only performs the linear model on probes that are directly included within the gene sets as genes not included will not be used in the analysis. For the previous random effect model example, if one would to run qgen with 260 gene sets comprising of 12,000 genes, the algorithm from start to finish will take approximately 9 minutes.

The qgen function provides a series of checks and provides various warnings and errors if any of the above mentioned options are not syntactically correct or if other obvious discrepancies exists. It must be noted, that in some situations, some probes

may have little to no variation due to processing or low intensity thresholds. In these cases, the linear mixed model may not converge to the maximum likelihood estimates. Currently, probes that do not convergence are removed from the analysis and a warning is presented. qgen returns a QSarray object, therefore, any additional functions that apply to the original qusage result object such as plot and qsTable can be applied to results obtained from qgen.

The VIF estimation techniques are implemented by the method provided by Turner et al. (BMC Bioinformatics 2015). In certain situations when the number of random effect replicates are low, an alternative VIF estimation is preferred to maintain type-I error estimation. qgen checks for these situations and automatically conducts the correct VIF estimation when it can. However, sometimes the technique will not be available due to the study design or the model is more complex than the scenarios provided by the authors. In cases such as this, a warning is produced letting the user know that the VIF estimate will use the original conditional residual approach and the type-I error rate may not be conserved. We suggest a simulation study could provide insight on whether or not there is a concern. Additional approaches similar to that of Turner et al. (BMC Bioinformatics 2015) could be incorporated to handle these situations in future updates.

6 Meta-Analysis with qusage

QuSAGE meta-analysis enables gene set analysis from multiple studies to be combined into a single estimation of gene set activity. This is done by using numerical convolution to combine the probability density functions (PDFs) of each data set into a single PDF.

To illustrate how QuSAGE meta-analysis works, we analyzed gene expression data from young individuals in two Influenza vaccination response data sets (GSE59635 and GSE59654). The goal of the analysis was to detect gene sets associated with successful vaccination responses. For this analysis, we will use the blood transcription modules (BTMs) described in Li et al. (Nat Immunol., 2014).

In data set GSE59635, there are 8 young subjects (3 low-responders and 5 high-responders to the vaccine). Each subject has day 0 (pre-vaccination) and day 7 (post-vaccination) blood samples.

```
> head(fluVaccine$phenoData[["GSE59635"]])
```

```
  subjectId responder bloodDrawDay
1 SUB113000       low           d0
2 SUB112995      high           d0
3 SUB113008      high           d0
4 SUB113014      high           d0
5 SUB113009       low           d0
6 SUB113005      high           d0
```

In data set GSE59654, there are 11 young subjects (7 low-responders and 4 high-responders to the vaccine). Similarly, each subject has day 0 (pre-vaccination) and day 7 (post-vaccination) blood samples.

```
> head(fluVaccine$phenoData[["GSE59654"]])
```

```
  subjectId responder bloodDrawDay
1 SUB120445       low           d0
2 SUB120420       low           d0
3 SUB120472       low           d0
4 SUB120450       low           d0
5 SUB120452       low           d0
6 SUB120459      high           d0
```

Using this data, QuSAGE was first used to carry out a two-way comparison (see section 3.3 for details) between low-responders and high-responders on each data set independently. We will do this by looping over each dataset with an `lapply` statement. Note that for this analysis, we're increasing the number of points `(n.points)` above the default due to low numbers of samples:

```
> qs.results.list = lapply(c("GSE59635", "GSE59654"), function(n){
+   eset = fluVaccine$esets[[n]]
+   phenoData = fluVaccine$phenoData[[n]]
+   labels = paste(phenoData$responder, phenoData$bloodDrawDay,
+                  sep = ".")
+   ##run QuSAGE
+   qusage(eset, labels, "(high.d7-high.d0)-(low.d7-low.d0)",
+             BTM.geneSets, pairVector=phenoData$subjectId, n.points=2^14)
+ })

Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.....................................................
Calculating variance inflation factors...Done.
Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.
Low sample size detected. n.points should be increased for better accuracy.....
Calculating variance inflation factors...Done.
Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.
Low sample size detected. n.points should be increased for better accuracy.....
Calculating variance inflation factors...Done.
Calculating gene-by-gene comparisons...Done.
Aggregating gene data for gene sets.....................................................
Calculating variance inflation factors...Done.
```

Next, QuSAGE meta-analysis was applied on the two resulting QSarray objects (one from each study). Each distrisbution is first weighted by the number of samples in each dataset, and then the two distributions are combined using numerical convolution.

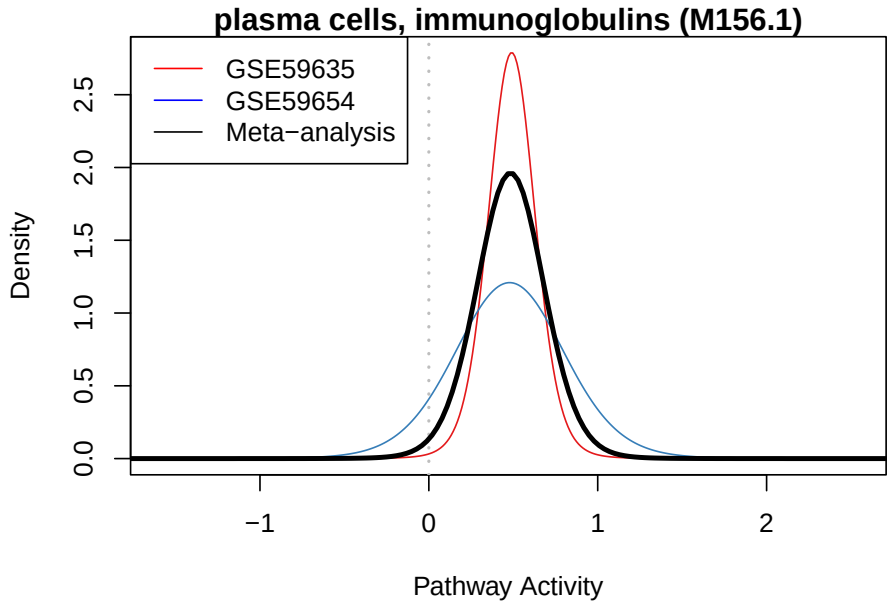
```
> combined = combinePDFs(qs.results.list)
```

P-values for each module can then be calculated the same way as for any other `QSarray` object.

```
> head(pdf.pVal(combined))

[1] 0.3349195 0.6062477 0.9832672 0.5280885 0.8995834 0.8614029
```

It turns out that "plasma cells, immunoglobulins (M156.1)" was one of top ranked gene modules from this QuSAGE meta-analysis. This module is significantly more up-regulated (day 7 vs. day 0) in high-responders compared to low-responders to flu vaccination. To plot this result, we can use the `plotCombinedPDF` function, as shown below.

```
>   plotCombinedPDF(combined, path.index=235)
>   legend("topleft", legend=c("GSE59635", "GSE59654", "Meta-analysis"),
+          lty=1, col=c("red", "blue","black"))
>
```



7 Plotting Functions

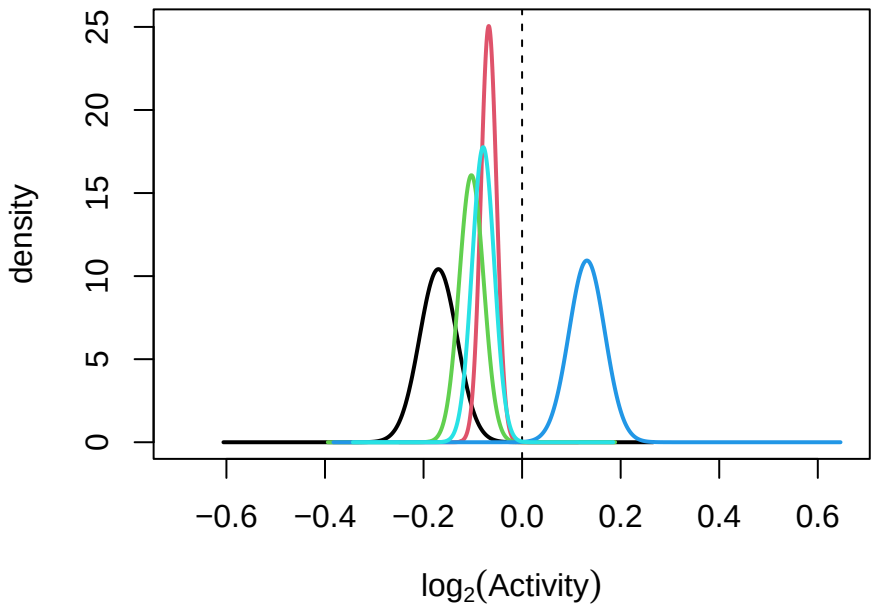
The `qusage` package contains a number of functions designed to easily and intuitively visualize the results stored in a QSarray object. Many of these functions were demonstrated in section 3, but in the following sections we will highlight the variety of options available when plotting the diverse data generated when running the qusage algorithm.

7.1 plotDensityCurves

The `plotDensityCurves` function is the simplest way to plot the data in a `QSarray` object. By default, this function generates a very simple plot containing probability density curves for each of the gene sets included in the qusage run. The density (y-axis) data for each PDF is stored in the `path.pdf` slot of the `QSarray` object, and the x-coordinates for the PDF are calculated on the fly using the `getXcoords` function (see below for details on this method).

We've seen PDF plots before in section 3, but a slightly more customized version of the plot with multiple PDFs can be seen below:

```
> ##select the best 5 gene sets
> p.vals = pdf.pVal(qs.results.msig)
> topSets = order(p.vals)[1:5]
> ##plot
> plotDensityCurves(qs.results.msig, path.index=topSets, col=1:5,
+                   lwd=2, xlab=expression(log[2](Activity)))
```



Because this function is simply an extension of the basic `plot` function, these plots can be heavily customized. This customization can be accomplished both through the basic plotting parameters, such as `xlab, ylab, main` and the parameters passed on to `par`, and via some parameters which are specific to `plotDensityCurves`. The `add` parameter can also be specified to overlay a second plot over an existing one.

The function-specific parameters are listed below.

- `path.index` This parameter controls which PDFs are plotted and the order that they are plotted in. It can be specified as either a numeric vector or a vector of names matching the gene sets.

- `zeroLine` By default, a vertical dashed line is added to the plot at 0 in order to give a visual indication of how significant the gene set is. In order to suppress the addition of this line, the parameter `zeroLine=FALSE` can be specified.

- `addVIF` Much like the other functions in qusage, this function gives you the option to specify whether the VIF should be used in calculating the PDF. By default, if the VIF has been calculated, it will always be included, but this behavior can be suppressed by explicity specifying `addVIF=FALSE`. While this will improve how significant your PDFs appear, it is strongly reccommended that the VIF is included whenever possible, as it is the only way to account for gene-gene correlations in a gene set.

The `plotDensityCurves` function is a useful way to visualize the PDF of a gene set, but in some cases it may be necessary to access the x and y coordinates of these PDFs individually. The information required to create these plots is all stored in the QSarray object, but some processing is necessary to get coordinates in the correct format for plotting. Although the y-coordinates can generally be used as-is when taken from the `path.pdf` matrix, the x-coordinates are not stored in the qusage object. Instead, the `getXcoords` function calculates x-coordinates for each point n using the following formula:

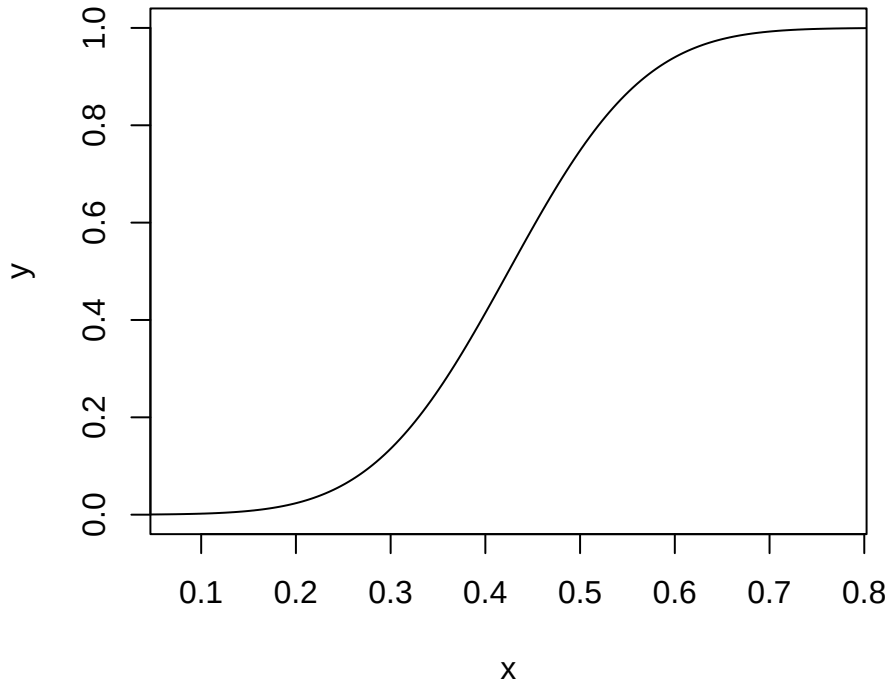
$$x_n = (-1 + \frac{2(n-1)}{N_{pts}-1}) \times r \times \sqrt{VIF} + \hat{\mu}_{path}$$

where N_{pts} is the total number of points, r is the range used for the numerical convolution (see section 4.2), VIF is the Variance Inflation Factor, and $\hat{\mu}_{path}$ is the mean fold change of the pathway.

Although the x- and y- coordinates can be extracted by hand from a qusage object, by far the easiest way to access this information is with the `plotDensityCurves` function itself. As mentioned in section 3, the `plotDensityCurves` function silently returns the x- and y-coordinates of each PDF being plotted. This data is returned as a list of data frames, with each individual data frame containing the x- and y-coordinates for the PDF of a single gene set.

An example of how to use this data for a custom plot is shown below. In this case, we will use the results from running qusage on the ISG set, stored in the `qs.results` object, to generate a Cumulative Distribution Function, or CDF.

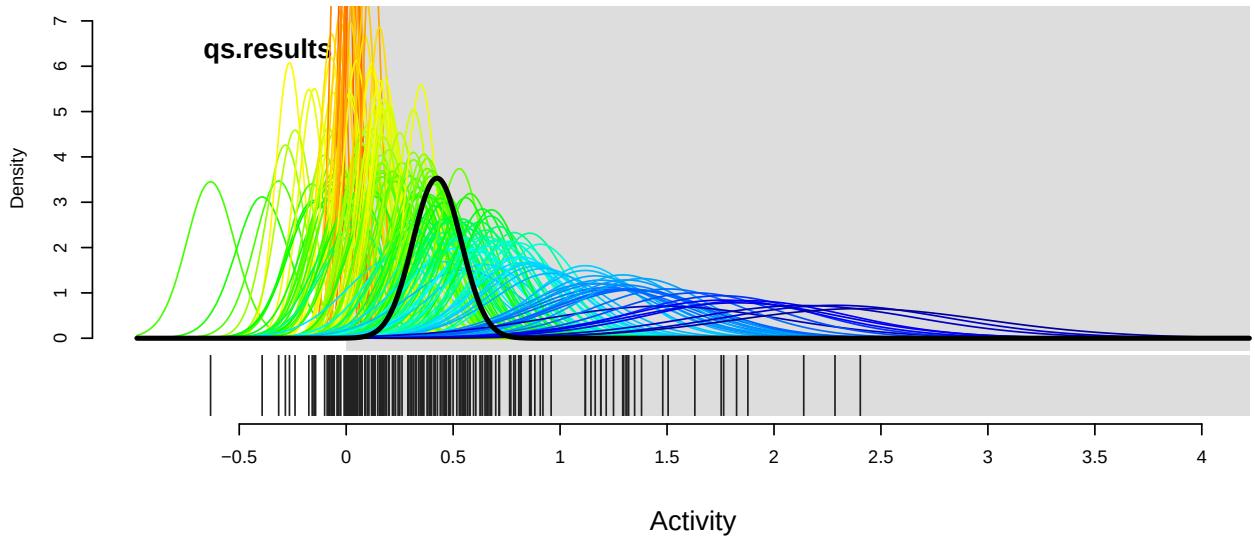
```
> ##get coordinates
> coord.list = plotDensityCurves(qs.results, plot=FALSE)
> coords = coord.list[[1]]
> x=coords$x
> ##Calculate the CDF as the integral of the PDF at each point.
> y = cumsum(coords$y)/sum(coords$y)
> plot(x,y, type="l", xlim=calcBayesCI(qs.results,low=0.001)[,1])
```



7.2 plotGeneSetDistributions

The `plotGeneSetDistributions` function is a method designed to visualize the PDFs of each individual gene in a gene set. As seen in section 3.3.1, the resulting plot is a useful and intuitive way to view how individual genes contribute to the overall PDF of the gene set. To generate a default plot, `plotGeneSetDistributions` can be called as shown below:

```
> plotGeneSetDistributions(qs.results)
```



The default gene set distribution plot is composed of two parts: a plot of the gene and gene set PDFs, and a barcode plot representing the mean of the individual genes. Both portions of the plot can be customized using the parameters in `plotGeneSetDistributions` and with the generic parameters that can be passed on to `plot`.

PDFs

The PDF plot is, in essence, an extension of the `plotDensityCurves` function discussed in section 7.1. The plot includes the average PDF for the entire gene set, which is calculated in the same way as discussed in section 7.1, as well as PDFs for each individual gene in the pathway. Because the mean and SD of each gene was calculated using a t-test, the individual gene PDFs are generated as t-distributions with parameters mean, standard deviation, and degrees of freedom defined by the values stored in `QSarray`.

`plotGeneSetDistributions` contains a number of parameters which are useful for customizing the PDF plots. These parameters are listed below:

- `colorScheme` This parameter is used to establish a color scheme for the individual gene PDFs. The default color scheme, sdHeat, will automatically color-code the gene PDFs by their standard deviations, with hotter colors being used for smaller standard deviations. In contrast, the "rainbow" color scheme will use a repeating series of colors, so that individual curves can be easily distinguished from their neighbors. Although these two options are the only automatically generated options, the `colorScheme` parameter can also accept a custom color scheme, which can be provided as a vector of colors. For a single gene set distribution plot, this can be useful for highlighting the PDFs of specific genes of interest. However, it is important to note that the order that the colors are used in is not the same as the order of genes in the original gene set. All gene sets are reordered when they are stored in the `QSarray$pathways` slot, and the vector provided to `colorScheme` will be used in this order.

- `alpha` This parameter controls the alpha channel of the individual gene PDFs. It is only used if the "sdHeat" or "rainbow" colorSchemes are used. Making the PDFs slightly transparent can be especially useful when there are a large number of genes, and the individual pdfs have a significant amount of overlap.

- `normalizePeaks` This parameter is used to control the heights of the individual gene PDFs. By default, the PDF heights will be scaled to correctly represent the density of the PDF, but in some cases, keeping the PDFs at a fixed height may make the picture easier to understand. If `normalizePeaks` is set to `TRUE`, the height of the individual gene PDFs will be scaled to be exactly the same.

- `lwds` This parameter controls the line widths for the PDFs. The first parameter controls the individual gene PDFs, and the second controls the gene set PDFs.

Barcode Plot

The barcode plot is a visual way of marking where the peaks of an individual gene PDF (i.e. the mean fold change of the genes) are located. By default, the barcode will be added as a series of vertical black lines directly below the PDF plot. The appearance of this plot can be controlled by three different parameters:

- `addBarcod`

  This boolean parameter controls the plotting of the barcode. If `addBarcode` is set to `FALSE`, the barcode will not be added to the plot.

- `barcode.col`

  This parameter controls the color of the vertical bars. This parameter can either be a single color or a vector with individual colors for each gene in the gene set. As with `colorScheme`, it is important to note that the order of the genes in the gene

set is not the same as the input parameters. All gene sets are reordered when they are stored in the `QSarray$pathways` slot, and the vector provided to `colorScheme` will be used in this order.

- `barcode.hei`

  This parameter controls the height of the barcode plot. This can be specified as a single numeric value representing the height of the bar as a fraction of the height of the main plotting region.

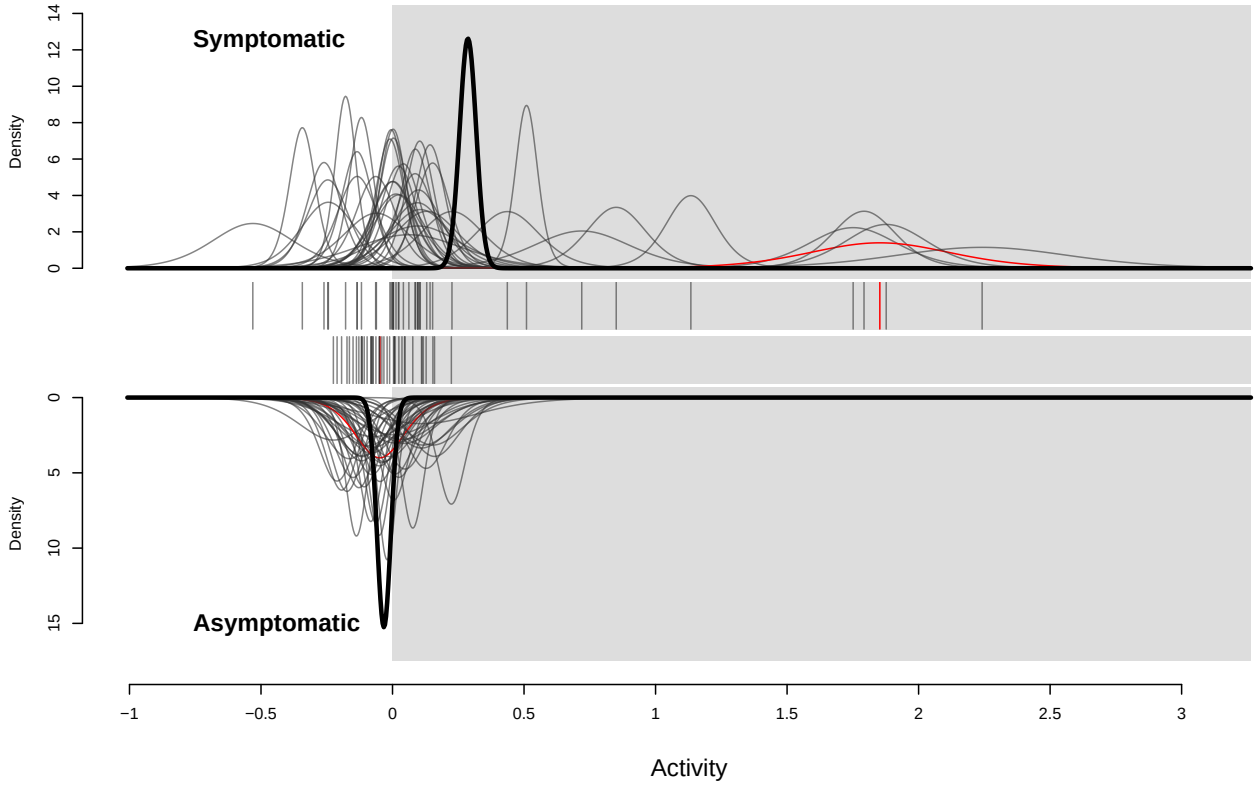
7.2.1 Comparing two gene sets

In cases where you want to compare two gene set distribution plots, the `plotGeneSetDistributions` function can automatically plot the results from a second gene set directly below the first. This can be done in two ways. First, to compare two different gene sets, a vector of length 2 can be provided to `path.index`, representing the two gene sets to be compared from `QSarray1`. Second, to compare two different runs of qusage, a second QSarray object can be provided via the `QSarray2` parameter. In this case, if `path.index` is of length 1, the single gene set specified will be pulled from both QSarray objects, making it easy to compare the activity of the gene set between runs. If both QSarray objects are provided and `path.index` is of length 2, then the first gene set will be pulled from QSarray1, and the second from QSarray2.

7.2.2 Custom Example: Highlighting genes of interest

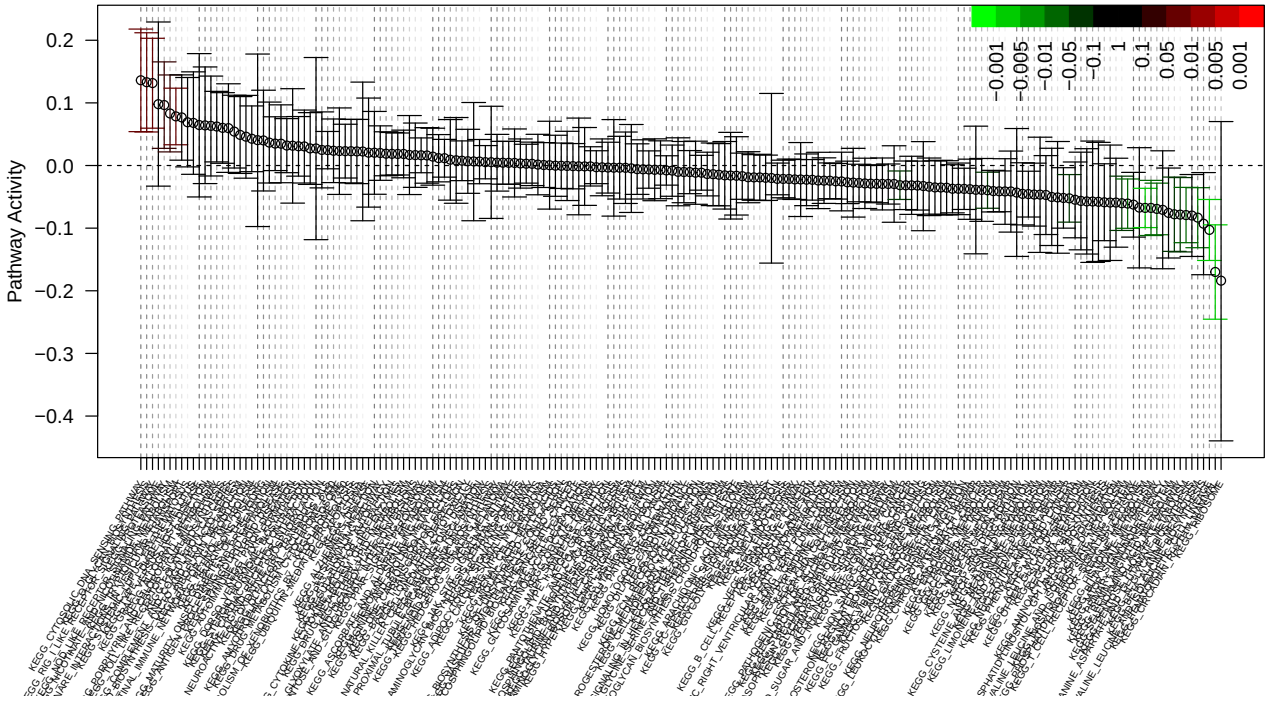
As an example of customizing the density curves plot, we will once again look at the activity of the `"CYTOSOLIC DNA SENSING"` pathway in our dataset. Perhaps the gene we are most interested in is CXCL10, and we would like to see how this specific gene compares to the others in the gene set. In order to do this, we will highlight the PDF for CXCL10 in red, and plot the rest of the gene PDFs in grey.

```
> path = 125 ##the CYTOSOLIC_DNA_SENSING pathway
> path.gene.i = sx.results$pathways[[path]] ##the genes are stored as indexes
> path.genes = rownames(eset)[path.gene.i]
> cols = rep("#33333399",length(path.genes))
> ##make the IFNA genes red, and the IFNB genes blue.
> cols[path.genes=="CXCL10"] = "#FF0000"
> plotGeneSetDistributions(sx.results,asx.results,path.index=125,
+                          groupLabel=c("Symptomatic","Asymptomatic"),
+                          colorScheme=cols, barcode.col=cols)
```


Symptomatic
Density
0
2
4
6
8
10
12
14
Asymptomatic
Density
0
5
10
15
−1
−0.5
0
0.5
1
1.5
2
2.5
3
Activity

7.3 plotCIs

As an alternative to plotting the entire PDF of a pathway, the qusage package also provides a function, `plotCIs`, which is designed to plot just the mean and 95% confidence interval of a pathway. An example of this function was seen in section 3.1, and the resulting plot is re-created below.



The default CI plot displays the mean fold change and 95% confidence interval of all the pathways contained in a `QSarray` object. The confidence intervals for each pathway are calculated using the `calcBayesCI` function, and the amount of overlap with the zero line can be used to estimate the significance of the gene set. In addition, each point is color coded by significance using the p-values calculated with `pdf.pVal`, and both the p-values and the CIs are corrected for multiple hypothesis testing using the methods defined in `p.adjust`. This plot can also be heavily customized using any of the additional parameters listed below.

- `path.index` This parameter controls which PDFs are plotted and the order that they are plotted in. It can be specified as either a numeric vector or a vector of names matching the gene sets. By default, the order of these pathways does not matter, but if `sort.by` is set to `"none"`, the pathways will be plotted in the order provided.

- `sort.by` This parameter accepts a string (either `"mean"`, `"p"`, or `"none"`) indicating how the pathways should be ordered. If `sort.by="mean"` or `"p"`, the pathways will be ordered by their means and p-values, respectively. Else, if `sort.by="none"`, the pathways will not be rearranged at all, and the order provided in path.index will be retained. The last option is most useful when plotting the results of two qusage runs in one plot, when it is necessary to ensure both plots have the same order.

- `lowerBound, upperBound` These parameters are passed on to `calcBayesCI` for use in calculating the confidence interval. The default, `lowerBound=0.025, upperBound=0.975`, represents a two-sided 95% confidence interval.

- `use.p.colors` This parameter controls the coloring of the error bars for the plot. By default (`use.p.colors=TRUE`), the bars will automatically be colored different shades of red and green based on the significance of their respective p-values (e.g. a p-value of 0.05 would result in a dark red bar, and a p-value of 0.0001 would result in a bright red bar).

- `col` This is a vector of colors which can be provided to define the color of the points. If `use.p.colors=FALSE` is specified, these colors will also be used for the error bars.

- `p.breaks` This is a numeric vector indicating where the breaks in the p-value color scheme should be placed. By default, breaks will be at 0.001, 0.005, 0.01, 0.05, & 0.1, meaning that a p-value between 0.05 and 0.1 will be a darker color than a p-value between 0.01 and 0.05. If you just wanted all significant p-values to be colored the same, you would simply set `p.breaks=0.05`

- `p.adjust.method` The method to use to adjust the p-values. Must be one of the methods in `p.adjust.methods`.

- `addLegend` This is a logical parameter specifying if a legend for the p-value color scheme be plotted. If true, the values from p.breaks will be plotted alongside their respective colors in the top right hand corner.

- `lowerColorBar` Options for plotting a color bar below each point. Automatically generated color bars have not yet been implemented, but custom bars can be created using the "lowerColorBar.cols" parameter.

- `lowerColorBar.cols` This is a vector of colors to plot as a bar along the bottom of the plot. This can be useful if there's other information you wish to include in the plot, such as a p-value for a second comparison, or a visual representation of another test statistic from a different program.

- `addGrid` This boolean parameter controls the addition of dashed lines behind the plot which help to follow a point down to its x-axis label.

- `x.labels` This is a character vector of the same length as `path.index` giving the names of the pathways. By default, `plotCIs` will use the names stored in `QSarray`.
- `cex.xaxis` This parameter controls the character expansion of the x-axis separate from the overall `cex`. The value supplied will be multiplied by the current plot's `cex` to determine the size of the x-axis text
- `shift` This is a number between 0 and 1 decribing the amount to shift points with respects to the guiding lines and axis labels. This is primarily used when `add=TRUE` and you do not want the two plots to directly overlap.
- `add` This boolean parameter is used to overlay a second plot over one that was previously generated. If `add=TRUE`, the bar plots will be added on to the existing plotting region.

8 The QSarray object

The QSarray object is designed to hold the results of the various functions and algorithms that make up the qusage package. Exactly what information is contained in a QSarray object is dependent upon which functions have been used to create the object. A new QSarray object can also be created by calling the `QSarray()` function. A list of all possible fields and their descriptions are included below.

- `mean, SD, dof` - Vectors containing the means, standard deviations, and degrees of freedom for each gene as output by `makeComparison` (see section 4.1). Together they describe the t-distribution of each gene in the dataset.
- `n.samples` - The number of samples used in the comparison. If the QSarray object is the result of a call to `combinePDF`, this will be a numeric vector detailing the number of samples in each input QSarray.
- `path.PDF` - This is a matrix which contains the density values for the PDF of each gene set provided to `aggregateGeneSet`. The number of rows of `path.PDF` is determined by the `n.points` parameter, which by default is 4096.
- `path.mean` - A numeric vector containing the mean fold change of each gene set provided to `aggregateGeneSet`.
- `ranges` - The (uncorrected) range that each PDF was calculated over. If the VIF is not used to correct the range, the x-coordinates of the PDF are the sequence of n.points from `path.mean-ranges` to `path.mean+ranges`
- `path.size` - A vector containing the number of features in each pathway that mapped to the input data.
- `vif` - A vector containing the Variance Inflation Factor for each pathway, as calculated by calcVIF.
- `pathways` - A version of the input `geneSets`, represented as a list of numeric vectors which map to the rows of `eset` which contain features in that gene set.
- `labels, contrast, pairVector, var.method, n.points` - Copies of the input parameters which are used in future calculations.
- `design` - A design matrix for the linear model fitting in LIMMA. Only created if limma is used in `makeComparison` (see section 4.1)
- `QSlist` - If the QSarray is the result of a call to `combinePDF` (see chapter 6), the QSlist will contain the list of QSarray objects used during the meta-analysis.