CQN (Conditional Quantile Normalization)

Kasper Daniel Hansen
khansen@jhsph.edu

Zhijin Wu
zhijin_wu@brown.edu

Modified: August 8, 2012. Compiled: October 13, 2014

Introduction

This package contains the CQN (conditional quantile normalization) method for normalizing RNA-seq datasets. This method is described in [1].

```
> library(cqn)
> library(scales)
```

Data

As an example we use ten samples from Montgomery [2]. The data has been processed as described in [1]. First we have the region by sample count matrix

```
> data(montgomery.subset)
> dim(montgomery.subset)
```

```
[1] 23552    10
```

```
> montgomery.subset[1:4,1:4]
```

```
                NA06985 NA06994 NA07037 NA10847
ENSG00000000419      69      54      67      70
ENSG00000000457      53      37      27      41
ENSG00000000460      12      25      33      22
ENSG00000000938     168     270     140     103
```

```
> colnames(montgomery.subset)
```

```
 [1] "NA06985" "NA06994" "NA07037" "NA10847" "NA11920" "NA11918"
 [7] "NA11931" "NA12003" "NA12006" "NA12287"
```

Because of (disc) space issues, We have removed all genes that have zero counts in all 10 samples. Next we have the *sizeFactors* which simply tells us how deep each sample was sequenced:

```
> data(sizeFactors.subset)
> sizeFactors.subset[1:4]
```

```
NA06985 NA06994 NA07037 NA10847
3107420 2388948 3087234 2852972
```

Finally, we have a matrix containing length and GC-content for each gene.

```
> data(uCovar)
> head(uCovar)
```

```
                length gccontent
ENSG00000000419   1207 0.3976802
ENSG00000000457   2861 0.4606781
ENSG00000000460   4912 0.4338355
ENSG00000000938   3524 0.5749149
ENSG00000000971   8214 0.3613343
ENSG00000001036   2590 0.4312741
```

Note that the row ordering of the count matrix is the same as the row ordering of the matrix containing length and GC-content and that the sizeFactor vector has the same column order as the count matrix. We can formally check this

```
> stopifnot(all(rownames(montgomery.subset) == rownames(uCovar)))
> stopifnot(colnames(montgomery.subset) == names(sizeFactors.subset))
```

Normalization

The methodology is described in [1]. The main workhorse is the function `cqn` which fits the following model

$$\log_2(\text{RPM}) = s(x) + s(\log_2(\text{length}))$$

where x is some covariate, s are smooth functions (specifically natural cubic splines with 5 knots), and RPM are "reads per millions". It is also possible to just fit a model like

$$\log_2(\text{RPKM}) = s(x)$$

In this model gene length is included as a known offset. This is done by using the `cqn(lengthMethod = "fixed")`. If this is done, and `lengths` is equal to 1000, it is equivalent to not using gene length at all.

The basic call to `cqn` is relatively easy, we need the count matrix, a vector of lengths, a vector of GC content and a vector of sizeFactors. Make sure that they all have the same ordering.

```
> cqn.subset <- cqn(montgomery.subset, lengths = uCovar$length,
+                   x = uCovar$gccontent, sizeFactors = sizeFactors.subset,
+                   verbose = TRUE)
```

```
RQ fit ..........
SQN .
```

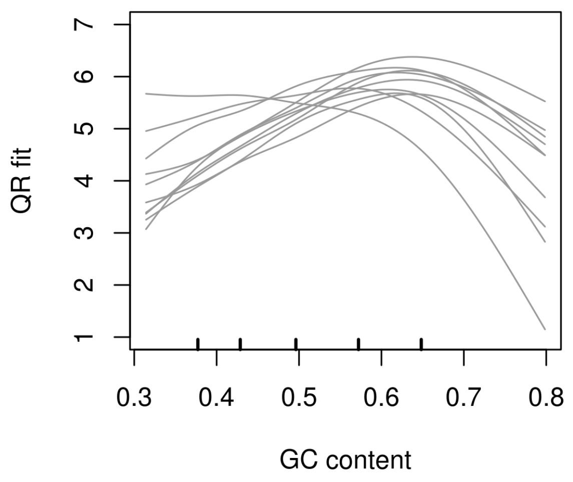
```
> cqn.subset
```

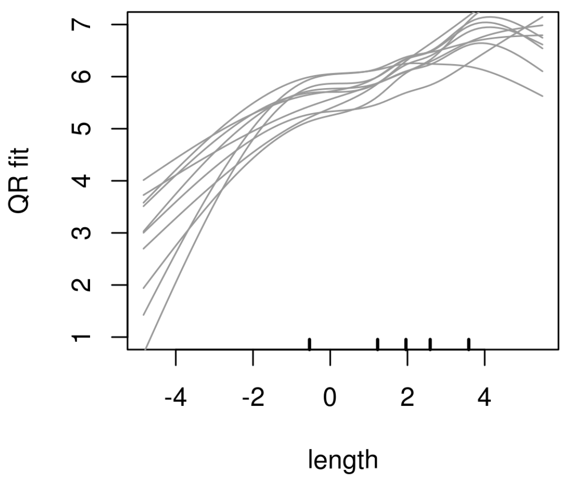
```
Call:
 cqn(counts = montgomery.subset, x = uCovar$gccontent, lengths = uCovar$length,
    sizeFactors = sizeFactors.subset, verbose = TRUE)

Object of class 'cqn' with
  23552 regions
  10 samples
fitted using smooth length
```

This normalized matrix is similar, but not equivalent, to the data examined in [1]. The main differences are (1) in [1] we normalize 60 samples together, not 10 and (2) we have removed all genes with zero counts in all 10 samples.

We can examine plots of systematic effects by using `cqnplot`. The `n` argument refers to the systematic effect, `n=1` is always the covariate specified by the `x` argument above, while `n=2` is lengths.

```
> par(mfrow=c(1,2))
> cqnplot(cqn.subset, n = 1, xlab = "GC content", lty = 1, ylim = c(1,7))
> cqnplot(cqn.subset, n = 2, xlab = "length", lty = 1, ylim = c(1,7))
```





The normalized expression values are

```
> RPKM.cqn <- cqn.subset$y + cqn.subset$offset
> RPKM.cqn[1:4,1:4]
```

```
                  NA06985  NA06994  NA07037  NA10847
ENSG00000000419 5.761813 5.568754 5.547280 5.975746
ENSG00000000457 4.436652 4.110397 3.394524 4.139537
ENSG00000000460 2.603203 3.444776 3.777542 3.068756
ENSG00000000938 5.152035 6.084140 4.698142 4.281873
```

These values are on the $\log_2$-scale.

We can do a MA plot of these fold changes, and compare it to fold changes based on standard RPKM. First we compute the standard RPKM (on a $\log_2$ scale):

```
> RPM <- sweep(log2(montgomery.subset + 1), 2, log2(sizeFactors.subset/10^6))
> RPKM.std <- sweep(RPM, 1, log2(uCovar$length / 10^3))
```

We now look at differential expression between two groups of samples. We use the same grouping as in [1], namely

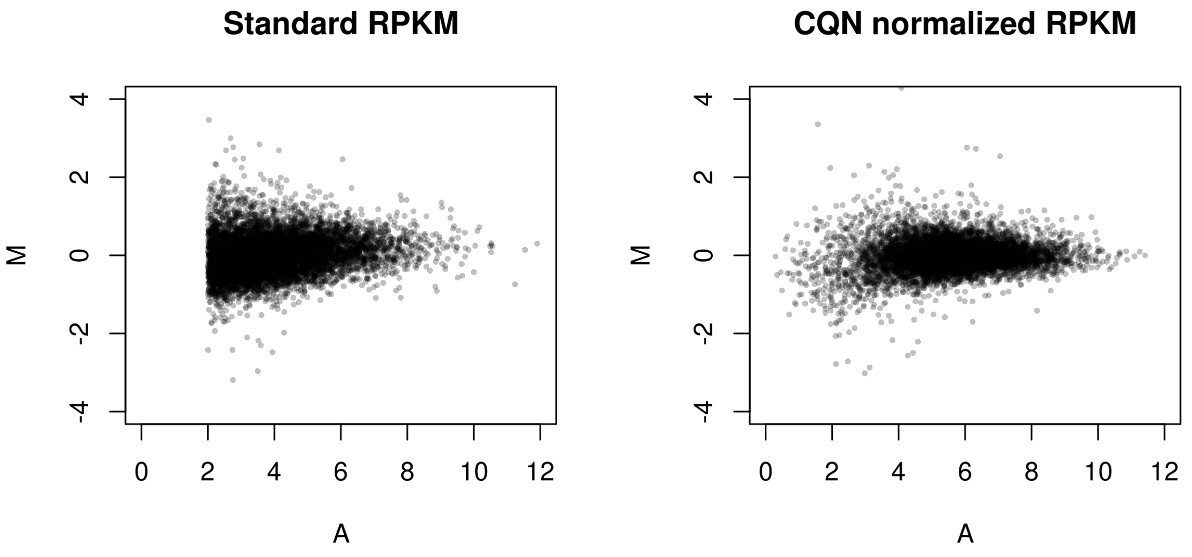
```
> grp1 <- c("NA06985", "NA06994", "NA07037", "NA10847", "NA11920")
> grp2 <- c("NA11918", "NA11931", "NA12003", "NA12006", "NA12287")
```

We now do an MA-plot, but we only choose to plot genes with average standard $\log_2$-RPKM of $\log_2(5)$ or greater, and we also form the M and A values:

```
> whGenes <- which(rowMeans(RPKM.std) >= 2 & uCovar$length >= 100)
> M.std <- rowMeans(RPKM.std[whGenes, grp1]) - rowMeans(RPKM.std[whGenes, grp2])
> A.std <- rowMeans(RPKM.std[whGenes,])
> M.cqn <- rowMeans(RPKM.cqn[whGenes, grp1]) - rowMeans(RPKM.cqn[whGenes, grp2])
> A.cqn <- rowMeans(RPKM.cqn[whGenes,])
```

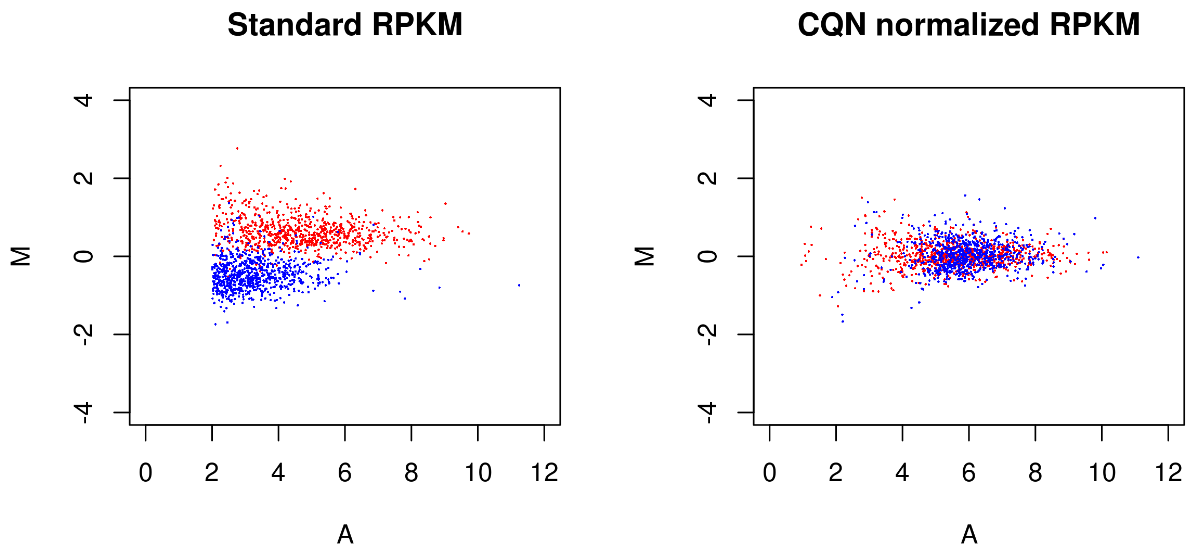
Now we do the MA plots, with alpha-blending

```
> par(mfrow = c(1,2))
> plot(A.std, M.std, cex = 0.5, pch = 16, xlab = "A", ylab = "M",
+      main = "Standard RPKM", ylim = c(-4,4), xlim = c(0,12),
+      col = alpha("black", 0.25))
> plot(A.cqn, M.cqn, cex = 0.5, pch = 16, xlab = "A", ylab = "M",
+      main = "CQN normalized RPKM", ylim = c(-4,4), xlim = c(0,12),
+      col = alpha("black", 0.25))
```



We can also color the genes according to whether they have high/low GC-content. Here one needs to be careful, because of overplotting. One solution is to leave out all genes with intermediate GC content. We define high/low GC content as the 10% most extreme genes:

```
> par(mfrow = c(1,2))
> gccontent <- uCovar$gccontent[whGenes]
> whHigh <- which(gccontent > quantile(gccontent, 0.9))
> whLow <- which(gccontent < quantile(gccontent, 0.1))
> plot(A.std[whHigh], M.std[whHigh], cex = 0.2, pch = 16, xlab = "A",
+      ylab = "M", main = "Standard RPKM",
+       ylim = c(-4,4), xlim = c(0,12), col = "red")
> points(A.std[whLow], M.std[whLow], cex = 0.2, pch = 16, col = "blue")
> plot(A.cqn[whHigh], M.cqn[whHigh], cex = 0.2, pch = 16, xlab = "A",
+       ylab = "M", main = "CQN normalized RPKM",
+       ylim = c(-4,4), xlim = c(0,12), col = "red")
> points(A.cqn[whLow], M.cqn[whLow], cex = 0.2, pch = 16, col = "blue")
```



Note that genes/regions with very small counts should not be relied upon, even if the CQN normalized fold change are big. They should be filtered out using some kind of statistical test, good packages for this are *DESeq*[3] and *edgeR*[4, 5].

Import into edgeR

First we construct a `DGEList`. In the `groups` argument we use that the first 5 samples (columns) in `montgomery.subset` is what we earlier called `grp1` and the last 5 samples (columns) are `grp2`.

```
> library(edgeR)
> d.mont <- DGEList(counts = montgomery.subset, lib.size = sizeFactors.subset,
+                         group = rep(c("grp1", "grp2"), each = 5), genes = uCovar)
```

In this object we cannot (unfortunately, yet) also store the computed offsets. Since we will use the offsets computed by *cqn*, there is no need to normalize using the normalization tools from *edgeR*, such as `calcNormFactors`. Also, as is clearly described in the *edgeR* user's guide, the `lib.size` is unnecessary, since we plan to use the offsets computed from *cqn*.

However, we need to use the component `glm.offset` which is on the natural logarithmic scale and also includes correcting for `sizeFactors`. It is possible to include the offset directly into the DGEList, by post-processing the output like

```
> ## Not run
> d.mont$offset <- cqn.subset$glm.offset
```

Using *edgeR* is well described in the user's guide, and we refer to that document for further information. The analysis presented below should be thought of as an example, and not necessarily the best analysis of this data.

The first step is estimating the dispersion parameter(s). Several methods exists, such as `estimateGLMCommonDisp` or `estimateTagwiseDisp`. We also need to setup a design matrix, which is particular simple for this two group comparison. Further information about constructing design matrices may be found in both the *edgeR* user's guide and the *limma* user's guide.

```
> design <- model.matrix(~ d.mont$sample$group)
> d.mont$offset <- cqn.subset$glm.offset
> d.mont.cqn <- estimateGLMCommonDisp(d.mont, design = design)
```

After fitting the dispersion parameter(s), we need to fit the model, and do a test for significance of the parameter of interest. With this design matrix, there are two coefficients. The first coefficient is just an intercept (overall level of expression for the gene) and it is (usually) not meaningful to test for this effect. Instead, the interesting coefficient is the second one that encodes a group difference.

```
> efit.cqn <- glmFit(d.mont.cqn, design = design)
> elrt.cqn <- glmLRT(efit.cqn, coef = 2)
> topTags(elrt.cqn, n = 2)

Coefficient:  d.mont$sample$groupgrp2 
                length gccontent     logFC   logCPM       LR
ENSG00000211642    365 0.5835616 -10.27719 6.331103 126.2188
ENSG00000211660    411 0.5888078 -10.09989 6.029329 120.5488
                      PValue          FDR
ENSG00000211642 2.753825e-29 6.485809e-25
ENSG00000211660 4.797328e-28 5.649333e-24
```

`topTags` shows (per default) the "top 10" genes. In this case, since we have biological replicates and just a random group structure, we would expect no differentially expression genes. Instead we get

```
> summary(decideTestsDGE(elrt.cqn))
```

```
   [,1]
-1   146
0  22970
1    436
```

significantly differentially expressed at an FDR (false discovery rate) of 5%. We may contrast this with the result of not using *cqn*:

```
> d.mont.std <- estimateGLMCommonDisp(d.mont, design = design)
> efit.std <- glmFit(d.mont.std, design = design)
> elrt.std <- glmLRT(efit.std, coef = 2)
> summary(decideTestsDGE(elrt.std))
```

```
   [,1]
-1   146
0  22970
1    436
```

In this evaluation, it is not clear that using CQN is better.

What is arguably as important is that we achieve a much better estimation of the fold change using *cqn*.

Question and Answers

Can I run cqn() on only 1 sample?

CQN is meant to normalize several samples together. It is not clear that it makes sense at all to use this normalization technique on a single sample. But it is possible.

Can I use this for small RNA-seq (microRNAs)?

We do not have personal experience with using CQN to normalize small RNA sequencing data. However, we believe it might be beneficial. As always, it is *highly* recommended to evaluate whether it is necessary and beneficial.

One special aspect of small RNAs is that they all have very similar length. Fitting a model with a smooth effect of gene length might very well lead to mathematical instability (you get an error). This can be avoided by using the argument `lengthMethod = "fixed"` which just divides the gene counts by the gene length instead of using a smooth function. Additionally, it may be coupled with setting `lengths = 1` which completely removes gene length from the model.

Could it be true that genes with higher GC content are higher expressed?

It has been suggested that genes that are either extremely high or extremely low expressed are under some form of selection leading to "extreme" GC content. What CQN does, is making the effect of GC content comparable across samples, and we show in [1] that this leads to improved inference. It also flattens the effect of GC content on gene expression, but we believe this is better than having the effect of GC content depend on the sample.

Does cqn remove batch effects?

No, unless a batch effect only (or mainly) affects your measurements through GC content. We believe that the sample-specific effect of GC content on gene expression is a kind of batch effect, but is unlikely to be the only one. CQN does normalize your RNA-seq data in the same way that say quantile normalization normalizes microarray data, but such normalization does not remove batch effects.

I don't understand the difference between offset and glm.offset?

This comes from a historical error. In our paper, we use the quantity

```
> cqn$y + cqn$offset
```

as the CQN-corrected estimated expression measures. However, the offset quantity is on the wrong scale for inclusion into a GLM-type model (like edgeR or DEseq2). For this purpose, use glm.offset. We have kept the original naming in order to achieve backwards compatibility.

SessionInfo

- R version 3.1.1 Patched (2014-09-25 r66681), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, splines, stats, utils
- Other packages: SparseM 1.05, cqn 1.12.0, edgeR 3.8.0, limma 3.22.0, mclust 4.4, nor1mix 1.2-0, preprocessCore 1.28.0, quantreg 5.05, scales 0.2.4
- Loaded via a namespace (and not attached): Rcpp 0.11.3, colorspace 1.2-4, munsell 0.4.2, plyr 1.8.1, tools 3.1.1

References

[1] KD Hansen, RA Irizarry, and Z Wu. Removing technical variability in RNA-seq data using conditional quantile normalization. *Biostatistics* 2012, **13**(2), 204–216. DOI: 10.1093/biostatistics/kxr054.

[2] SB Montgomery *et al.* Transcriptome genetics using second generation sequencing in a Caucasian population. *Nature* 2010, **464**, 773–777. DOI: . 10.1038/nature08903

[3] S Anders and W Huber. Differential expression analysis for sequence count data. *Genome Biology* 2010, **11**(10), R106. DOI: 10.1186/gb-2010-11-10-r106.

[4] MD Robinson, DJ McCarthy, GK Smyth. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics* 2010, **26**(1), 139–140. DOI: 10.1093/bioinformatics/btp616.

[5] DJ McCarthy, Y Chen, GK Smyth. Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 2012, **40**, 4288- 4297. DOI: 10.1093/nar/gks042.