A pipeline of simulation for RNA-seq counted data

Xiaobei Zhou and Mark D. Robinson

March 4, 2014

1 A simple edgeR-robust pipeline

This section describes briefly how to run an ***edgeR-robust*** analysis. Users should follow the "glm edgeR" analysis analagously to "1.4 Quick start" in the edgeR Users Guide [1], except replacing the dispersion estimation as below:

```
> library(edgeR)  ##Version>=3.5.22
> group <- c(1, 1, 1, 2, 2, 2)
> design <- model.matrix(~group, data = d$samples)  ## Define the design matrix
> y <- matrix(rnbinom(100 * 6, mu = 10, size = 1/0.1),
+     ncol = 6)
> d <- DGEList(counts = y, group = group)
> d <- estimateGLMRobustDisp(d, design)
> # further commands glmFit(), glmLRT() and so on
```

Note that once the software recognizes the presence of observation weights (from `estimateGLMRobustDisp`), downstream steps, such as regression parameter estimation (`glmFit`) and statistical testing (`glmLRT`), will be also be incorporated the weights.

2 Simulation framework and evaluation

The Section describes the simulation framework and in particular, how to: i) re-run various simulations from the manuscript; ii) add new methods (or variations of existing methods) through the creation of simple wrapper functions.

2.1 General set up

All the necessary functions for the simulation and metrics for evaluating performance across different DE methods can be sourced from:

```
> ## source simulation functions
> ## source("http://130.60.190.4/robinson_lab/edgeR_robust/robust_simulation.R")
> ## check information of all the available DE
> ## methods/packages
> checkMethods()
```

[1] http://www.bioconductor.org/packages/release/bioc/vignettes/edgeR/inst/doc/edgeRUsersGuide.pdf

```
             pkg           (current)version (recommended)v source
baySeq       "baySeq"      NA               "1.16.0"       "bioc2.13"
DESeq2       "DESeq2"      "1.2.8"          "1.2.10"       "bioc2.13"
DESeq2_rmNA  "DESeq2"      "1.2.8"          "1.2.10"       "bioc2.13"
DESeq_cr     "DESeq"       "1.14.0"         "1.14.0"       "bioc2.13"
DESeq_glm    "DESeq"       "1.14.0"         "1.14.0"       "bioc2.13"
DESeq_pool   "DESeq"       "1.14.0"         "1.14.0"       "bioc2.13"
EBSeq        "EBSeq"       NA               "1.2.0"        "bioc2.13"
edgeR        "edgeR"       "3.5.15"         "3.5.15"       "bioc2.14"
edgeR_rans   "edgeR"       "3.5.15"         "3.5.15"       "bioc2.14"
edgeR_rdev   "edgeR"       "3.5.15"         "3.5.15"       "bioc2.14"
edgeR_robust "edgeR"       "3.5.15"         "3.5.15"       "bioc2.14"
limma_voom   "limma"       "3.18.3"         "3.18.13"      "bioc2.13"
samr_SAMseq  "samr"        "2.0"            "2.0"          "R-base"
ShrinkBayes  "ShrinkBayes" NA               "2.6"          "R-base"
```

```
> ## select some methods to be evaluated
> selected.method <- c("edgeR", "edgeR_robust", "limma_voom",
+     "DESeq2")
```

There are more than 10 DE methods built-in to the simulation framework that can be selected. To check the information of DE methods (whether a related package is corrected installed) run function `checkMethods`. In fact, each DE method in the simulation framework is implemented as a wrapper function. For instance, classical ***edgeR*** is implemented as `edgeR.pfun`

```
> edgeR.pfun
```

```
function(counts, group, design = NULL, mc.cores = 4, prior.df=10)
{
    ## edgeR standard pipeline ##
        library(edgeR)
        d <- DGEList(counts = counts, group = group )
        d <- calcNormFactors(d)
        d <- estimateGLMCommonDisp(d, design = design)
        d <- estimateGLMTrendedDisp(d,design=design)
        d <- estimateGLMTagwiseDisp(d, design = design, prior.df = prior.df)
        f <- glmFit(d, design = design)
        lr <- glmLRT(f, coef=2)
        pval = lr$table$PValue
        padj = p.adjust(pval, "BH")
        cbind(pval = pval, padj = padj)
}
```

Here, you can see the inputs (count table, grouping variable, design matrix, other settings) and the outputs (matrix with same number of rows as the count table and a column each for P-values and adjusted P-values). Notably, this provides a simple interface to define your own DE method, as follows:

```
> your.method <- function(counts, group, design = NULL) {
+     pval <- function(counts, group, design) ## your function to calculate pvalue
+     padj <- function(pval) ## your function to get p-adjust value finally
+     ## return a matrix containing pval and padj
+     cbind(pval = pval, padj = padj)
+ }
```

Our simulation generates "true" NB model parameters, μ and ϕ, using the joint distribution of estimates, $\hat{\mu}$ and $\hat{\phi}$, from real data, such as from published count tables at ***ReCount*** [2]: Pickrell [3], Cheung et al.[1, 4]. Here we show an example with data (count table) from the HapMap project (denoted as Pickrell [3]):

```
> library(tweeDEseqCountData)
> data(pickrell)
> pickrell <- as.matrix(exprs(pickrell.eset))
```

Similarly, you can define your own dataset to seed the simulation. Your dataset should be a `matrix` (count table):

```
> your.data <- matrix(...)
```

2.2 The NB simulation

To generate a simulation that aims to accurately reflect the reality of RNA sequencing data across a variety of reasonable test conditions, we created several options:

1. `nTags`: total number of features
2. `group`: factor containing the experimental conditions
3. `pDiff`: proportion of DE features
4. `foldDiff`: relative expression level of truly DE features
5. `pUp`: proportion of features with an increase in expression
6. `dataset`: dataset to take model parameters from
7. `pOutlier`: proportion of outliers to introduce
8. `outlierMech`: outlier generation mechanism to use ("single" (S) , "random" (R) and "model" (M). Please see the detail into the NAR robust manuscript.)
9. `drop.extreme.dispersion`: filtered out the count of real dataset with extreme high estimated dispersion (drop.extreme.dispersion=0.1)

Let us build a simulation containing a two group comparison (5 replicates of control and 5 replicates of treat)

```
> nSamp <- 10
> grp <- as.factor(rep(0:1, each = nSamp/2))
> data <- NBsim(foldDiff = 3, dataset = pickrell, nTags = 1000,
+     group = grp, verbose = TRUE, add.outlier = TRUE,
+     outlierMech = "S", pOutlier = 0.1, drop.extreme.dispersion = 0.1)
```

This simulation contains following features:
Its NB model parameters μ and ϕ are sampled from the jointed distribution of estimates from Pickrell data [3] (`dataset`=pickrell); the top 10 percent of the extreme dispersion values are filtered out (`drop.extreme.dispersion`= 0.1, default setting) and 1000 features are generated (`nTags` = 1000) , forming a two group comparison (5 control Versus 5 treat, `group` = `grp`); 10% of them are defined as DE genes (`pDiff` = 0.1), symmetrically (`pUp=.5`), with fold difference 3 (`foldDiff`= 3); 10% outliers (`pOutlier` = 0.3) are introduced by "S" method (`outlierMech`= "S"). In our simulation original simulated counts and the counts with outliers introduced are separately recorded:

```
> data$counts  ## original simulated counts
> data$S$countAddOut  ##  the counts with outliers
```

Outliers are randomly distributed amongst all genes. For convenience, we define several subsituations when an outlier is introduced:

1. `DEupOutlier`: a list of indices that represents the situation where the outlier is added to the higher expressed group
2. `DEdownOutlier`: a list of indices that represents those features where the outlier was added to the lower expressed condition
3. `DEnoOutlier`: a list of indices that represents DE features with no introduced outlier
4. `DEbothOutlier`: a list of indices that represents DE features when the outliers are added to both higher expressed group and lower groups ("S" method is not possible for this case.)
5. `NonDEOutlier`: a list of indices thatrepresents non DE features with added outliers
6. `NonDEnoOutlier`: a list of indices that represents non DE features without outliers

All the necessary information of outliers are saved according to the selected `outlierMech`, as follows:

```
> names(data$S)
```

```
 [1] "countAddOut"       "outlierMech"       "pOutlier"
 [4] "mask_outlier"      "indDEupOutlier"    "indDEdownOutlier"
 [7] "indDEbothOutlier"  "indDEnoOutlier"    "indNonDEOutlier"
[10] "indNonDEnoOutlier" "LambdaAddOut"      "DispersionAddOut"
```

Note that the simulation is stored as an `DGEList` object, with extra elements of the list added.

2.3 Running various DE methods on simulated datasets

First, the set of selected methods can be run, in a multicore fashion, using:

```
> ## get P-values for DE methods
> pvals_b <- pval(data, method = selected.method, count.type = "counts",
+     mc.cores = 4)
> pvals_s <- pval(data, method = selected.method, count.type = "S",
+     mc.cores = 4)
```

Original simulated counts (`count.type="counts"`) and the counts with outlier added by "S" method (`count.type="S"`) are evaluated separately. To speed up your evaluation, parallel computation is recommended (`mc.cores`).

2.4 Comparison metrics

We provide several standard metrics and plots: false discovery (FD) plots, receiver operating characteristic (ROC) curves, partial ROC curves and power curves.
The standard ROC curves for original simulated counts and the counts with outlier are separately plotted as:

```
> par(mfrow = c(1, 2))
> roPlot(pvals_b, plot.max.fpr = 1, threshold = NULL)
> roPlot(pvals_s, plot.max.fpr = 1, threshold = NULL)
```

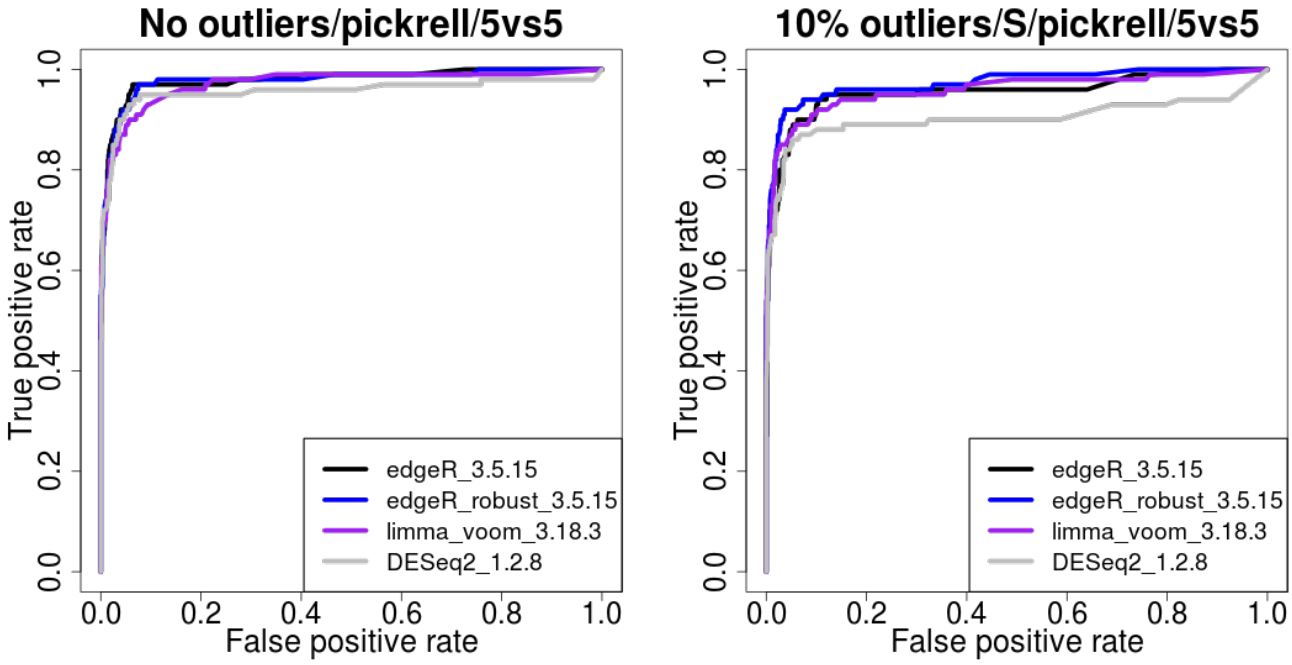

Figure 1: ROC curves

Offen, we are interesting in plotting a particial ROC curve (since true positive rates at high false positive rates are not so interesting), this can be done by setting (`plot.max.fpr`) argument. We

also provide another ineresting feature: an "X" point on the ROC curve that corresponds the methods's FDR (`threshold`= 0.05).

```
> par(mfrow = c(1, 2))
> roPlot(pvals_b, threshold = 0.05, plot.max.fpr = 0.4,
+     cex = 1)
> roPlot(pvals_s, threshold = 0.05, plot.max.fpr = 0.4,
+     cex = 1)
```

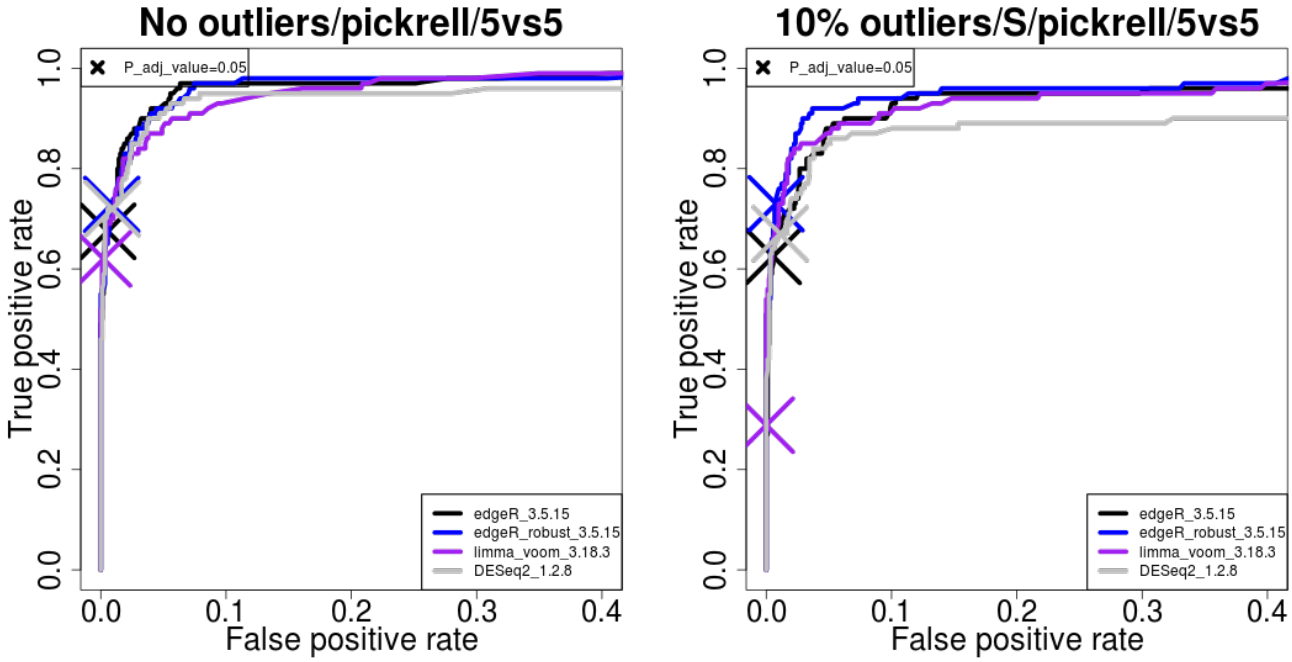

Figure 2: pROC curves

The FD plots are as follows:

```
> par(mfrow = c(1, 2))
> fdPlot(pvals_b, xlim = c(20, 100), cex = 1)
> fdPlot(pvals_s, xlim = c(20, 100), cex = 1)
```

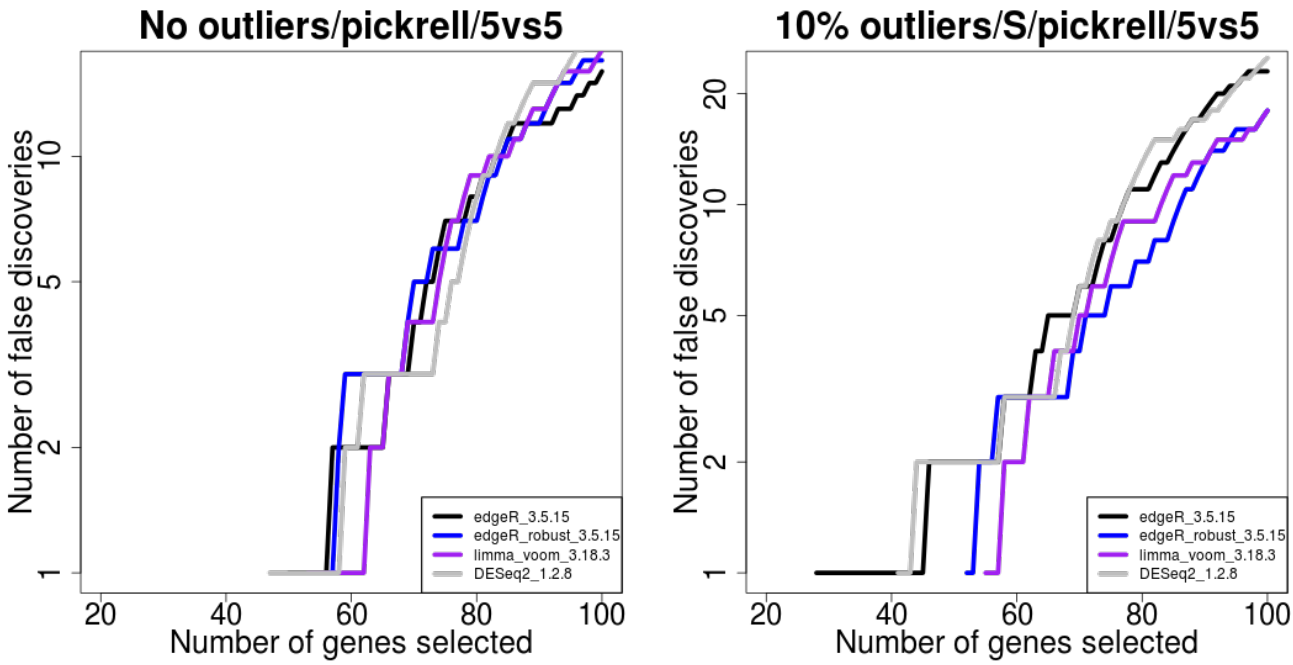

Figure 3: False discovery plots

We provide many options for plotting power at a given FDR cutoff. You can focus on the general performance of DE methods:

```
> par(mfrow = c(1, 2))
> getPower(pvals_b, plot = TRUE)
> getPower(pvals_s, plot = TRUE)
```

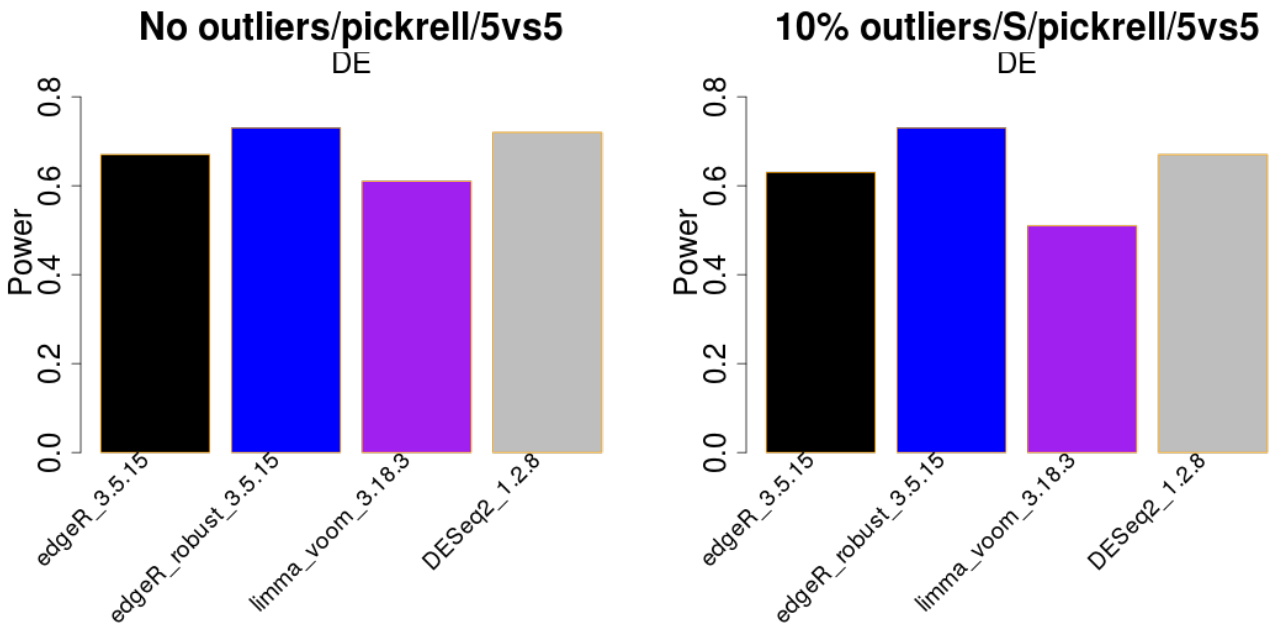

Figure 4: Power curves for general performance

One can subdivide power according to expression strength (average-log-CPM) groups (`byAveLogCPM`, `cutCPM`)

```
> par(mfrow = c(1, 2))
> getPower(pvals_b, plot = TRUE, byAveLogCPM = TRUE,
+     cutCPM = 3)
> getPower(pvals_s, plot = TRUE, byAveLogCPM = TRUE,
+     cutCPM = 3)
```

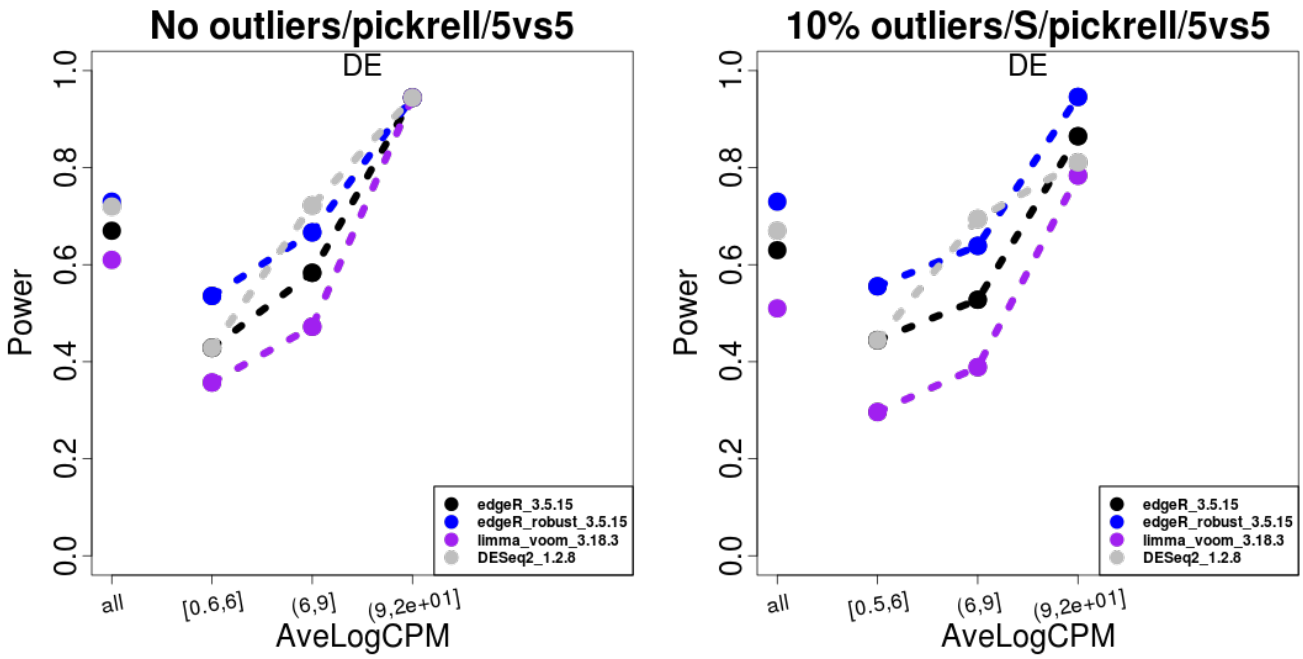

Figure 5: Power curves by average-log-CPM

Power curves can be further split by where outliers are relative to differential expression, using:

```
> par(mfrow = c(1, 3))
> getPower(pvals_s, plot = TRUE, index = "indDEupOutlier",
+     byAveLogCPM = TRUE, cutCPM = 3)
> getPower(pvals_s, plot = TRUE, index = "indDEdownOutlier",
+     byAveLogCPM = TRUE, cutCPM = 3)
> getPower(pvals_s, plot = TRUE, index = "indDEnoOutlier",
+     byAveLogCPM = TRUE, cutCPM = 3)
```

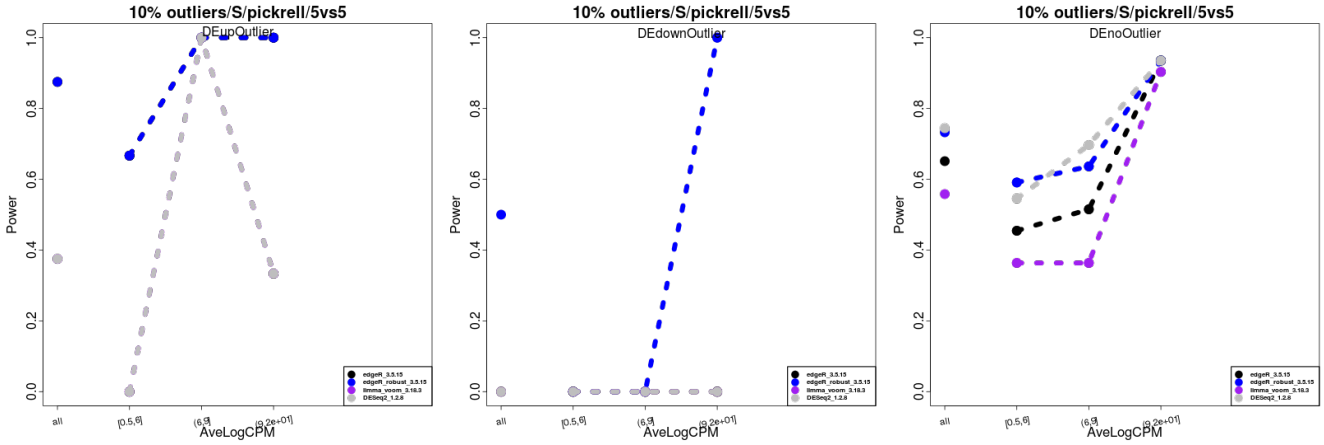

Figure 6: Power curves for different sub-situations

2.5 Timing for *edgeR* and *edgeR-robust*

Note that also the running times of each DE method are stored by default in the `time` element when function `pval` is called. **Note** that the times are comparable time but not absolute time, since function `pval` is parallelly implemented. To make sure each job (core) is forked for each DE method, the default setting of `mc.preschedule` in function `pval` has been chose as FALSE.

```
> pvals_b$time

$edgeR
   user  system elapsed
  1.992   0.092   2.082

$edgeR_robust
   user  system elapsed
  9.592   0.080   9.674

$limma_voom
   user  system elapsed
  0.512   0.024   0.539

$DESeq2
   user  system elapsed
  13.68    0.16   13.84
```

References

[1] Vivian G Cheung, Renuka R Nayak, Isabel Xiaorong Wang, Susannah Elwyn, Sarah M Cousins, Michael Morley, and Richard S Spielman. Polymorphic Cis- and Trans-Regulation of Human Gene Expression. *PLoS Biology*, 8(9):14, 2010.

[2] Alyssa C Frazee, Ben Langmead, and Jeffrey T Leek. ReCount: A multi-experiment resource of analysis-ready RNA-seq gene count datasets. *BMC Bioinformatics*, 12(1):449, 2011.

[3] Stephen B Montgomery, Micha Sammeth, Maria Gutierrez-Arcelus, Radoslaw P Lach, Catherine Ingle, James Nisbett, Roderic Guigo, and Emmanouil T Dermitzakis. Transcriptome genetics using second generation sequencing in a Caucasian population. *Nature*, 464(7289):773–777, 2010.

[4] Daniela Witten, Robert Tibshirani, Sam Guoping Gu, Andrew Fire, and Weng-Onn Lui. Ultra-high throughput sequencing-based small RNA discovery and discrete statistical biomarker analysis in a collection of cervical tumours and matched controls. *BMC Biology*, 8(1):58, 2010.