Analysis of co-knockdown data

Elin Axelsson

1 Introduction

This is the vignette to the R package *coRNAi*. *coRNAi* consists of functions for analysing combinatorial RNAi knockdown screens. The package builds on the R package *cellHTS2* and takes a `cellHTS` object as input. In the section *Creating cellHTS object for coRNAi*, we will discuss the extra annotations needed for the downstream analysis of combinatorial knockdown data and we will provide an example of a valid annotation file for *cellHTS2*. In the section *Analysis of co-RNAi screens* we will illustrate the workflow by going through the analysis of two different datasets step by step. The section *Moderated t-test* provides justifications for the usage of a moderated t-test, in the section *Choice of fitness phenotype* we will briefly describe two possible fitness measurements. In the last section *Analysis of large datasets* we will show how it is possible to speed up the analysis when working with large and interaction sparse datasets.

2 Creating cellHTS object for coRNAi

The vignette to the Rpackage *cellHTS2* illustrates how to create a `cellHTS` object and we recommend all users to follow the instructions. However, when creating your `cellHTS` object, keep in mind that co-knockdown experiments require more meta data than the straight forward single knockdown screens. This extra information should be provided in the annotation file read by the cellHTS2 function *annotate*. In addition to the mandatory columns of the annotation file: *Plate*, *Well* and *GeneID*, the following should also be provided: *ID1*, *ID2* and *Type*.

ID1 and **ID2** are the names (or identifiers) of the two RNAi supplied to the well. In many experimental setups, the two RNAis are added in slightly different ways, e.g. in sequential order. In those cases it makes sense to keep track of this by e.g. calling the first RNAi ID1 and the second ID2. The *cellHTS2df* function will use the in information in ID1 and ID2 to create a

column called *Pair* and a column called *Direction*. The Pair column does not take into account which if a RNAi treatment comes from ID1 or ID2. That is, combination x + y and combination y + x will have the same Pair ID (y x). The Direction however, will be different, 1 for treatment y+x and 2 for treatment x+y. This setup allows us to choice if we later want to consider y+x and x+y as replicates or not.

Type indicates the type of the well and should be one of the following:*double*, *comb*, *controlP1*, *controlP2*, *controlN1*, *controlN2*, *controlP1N1* or *other*.

- **double**: ID1 and ID2 are the same, that is one gene is knockdown with the double amount of RNAi.
- **comb**: ID1 and ID2 are different but both are targeting sample genes (not controls)
- **controlP1**: a positive control is knocked-down in combination with a sample knockdown.
- **controlP2**: both RNAi target a positive control
- **controlN1**: a negative control is knocked-down in combination with a sample knockdown.
- **controlN2**: both RNAi target a negative control
- **controlP1N1**: a negative control and a positive control are knocked down.
- **other**: any other type of combination, e.g. involving a intermediate control

The first lines of an example annotation file are shown below.

Plate	**Well**	**Type**	**GeneID**	**Pair**	**ID2**	**ID1**
1	A01	double	P1 P1	P1 P1	P1	P1
1	A02	comb	P1 P2	P2 P1	P2	P1
1	A03	comb	P1 P3	P3 P1	P3	P1
1	A04	comb	P1 P4	P4 P1	P4	P1
1	A05	comb	P1 P5	P5 P1	P5	P1
...	...	...	...	...	...	...

Once this upgraded annotation file has been generated, the data can be read in by *cellHTS2* as described in the *cellHTS2* vignette.

3 Analysis of co-RNAi screens

3.1 Getting started

First the package needs to be loaded into the R session:

```
> library("coRNAi")
```

The datasets we will use here are available as data objects in the *coRNAi* package and we can access them by using the *data* function. For your own saved datasets, the *load* function can be used.

```
> data(screen1_raw)
> data(screen2_raw)
> #data(num2name)
```

Once we have loaded the data into our session, we need to convert the `cellHTS` object into the data frame format that is used in the *coRNAi* package. The function *cellHTS2df* takes care of this. In addition to the `cellHTS` object we also need to provide information on which (if any) RNAi is to be considered as neutral in the experiment. In the datasets used in this vignette, dsRNA targeting Fluc was used as a negative control and therefore we set the neutral argument to Fluc. In the next step we log 2-transform the data.

```
> dfR1 = cellHTS2df(screen1_raw,neutral=c("Fluc"))
> dfR2 = cellHTS2df(screen2_raw,neutral=c("Fluc"))
> dfR1$value = log2(dfR1$value)
> dfR2$value = log2(dfR2$value)
```

If we look at the first rows of the resulting data frame we will see that we now have the two additional columns "Pair" and "Direction", generated by the *cellHTS2df* based on the ID1 and ID2 annotations.

```
> dfR2[1:2,]
```

```
    plate well controlStatus Content Combination   Type GeneID  Pair Direction
1.1     1  A01        sample  sample         1+1 double  P1 P1 P1 P1         1
1.2     1  A02        sample  sample         1+2   comb  P1 P2 P2 P1         2
    PPcontent ID2 ID1    value replicate
1.1         2  P1  P1 18.23659         1
1.2         2  P2  P1 18.21338         1
```

```
> mypars = list(cex.lab = 1,cex.main=1)
> par(mfrow=c(1,2))
> BoxPlotShorth(value~replicate,dfR1[dfR1$controlStatus=="sample",],
+               las=2,main="",xlab="plates",boxfill="lightblue",
+               outline=FALSE, ylab=expression(log[2](intensity)),pars=mypars)
> BoxPlotShorth(value~plate,dfR2[dfR2$controlStatus=="sample" &
+                                dfR2$replicate==1,], las=2,
+               main="",xlab="plates",
+               boxfill="lightblue",outline=FALSE,
+               ylab=expression(log[2](intensity)),pars=mypars)
```

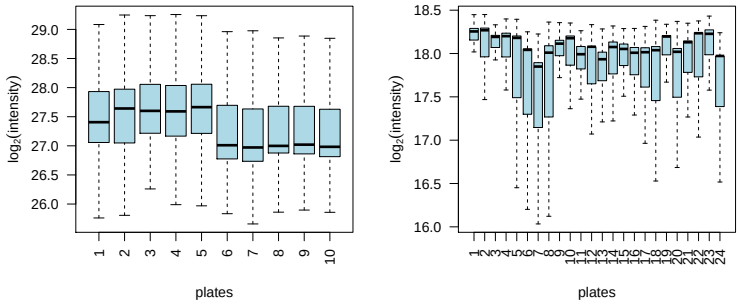

Figure 1: Boxplot for plates before normalization.

3.2 Normalisation

Screen 1 was done on one single plate but a simple boxplot (Figure 1) shows that the value distribution is different over the 10 replicates. Screen 2 includes 24 plates, again we can see that distributions varies over the plates. The function *BoxPlotShorth* plots the normal boxplots, but with horizontal bars at the midpoint of the shorth rather than at the medians.

There are many possible reasons for this variation between plates, but to allow for comparisons between individual plates one needs to normalise the plates. The function *normalizePlates* from the package *cellHTS2* takes care of the normalisation and can also be used to transform the data to the log scale. The following lines of code normalizes the screens and converts the normalized data into a data frame.

```
> dfScl1 = cellHTS2df(normalizePlates(screen1_raw,method="shorth",
+   scale="multiplicative",log=TRUE),neutral="Fluc")
```

```
> par(mfrow=c(1,2))
> BoxPlotShorth(value~replicate,dfScl1[dfScl1$controlStatus=="sample",],
+               las=2, main="",xlab="plates",boxfill="lightpink",
+               outline=FALSE,ylab=expression(log[2](intensity)))
> BoxPlotShorth(value~plate,dfScl2[dfScl2$controlStatus=="sample"&
+                                  dfScl2$replicate==1,],
+               boxfill="lightpink",las=2,main= "",
+               xlab="plates",outline=FALSE,ylab=expression(log[2](intensity)))
```

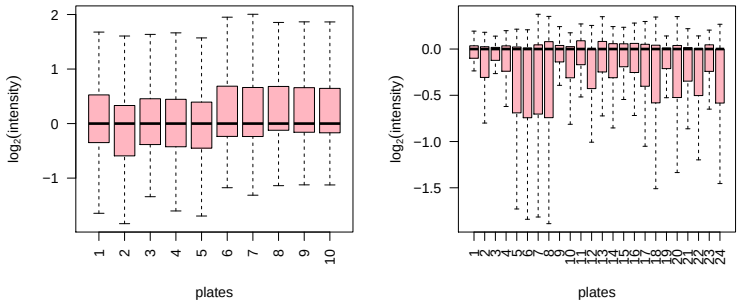

Figure 2: Boxplot for plates after normalization.

```
> dfScl2 = cellHTS2df(normalizePlates(screen2_raw,method="shorth",
+   scale="multiplicative",log=TRUE),neutral="Fluc")
```

After the normalisation the boxplots (Figure 2) look better:

3.3 Quality Assessment plots

3.3.1 Screen control plots

A screen plot is a false colour representation of all plates in an experiment. Normally, one does not wish to see any spatial trends in the screen plot, but due to the experimental setup used in the experiments analysed here, the stripped patterns across the plates are expected. Unexpected spatial trends should be removed if possible and the experimental cause should be investigated and improved upon.

Screen 1 was done with two biological replicates, each with 5 technical replicates. As can be seen in Figure 3, the colours are brighter in one of

```
> plotScreen(split(dfScl1$value,dfScl1$replicate),fill =
+             c("red","white","blue"),zrange=c(-4,4),ncol=5,
+              main="screen 1",legend.label=NULL)
```

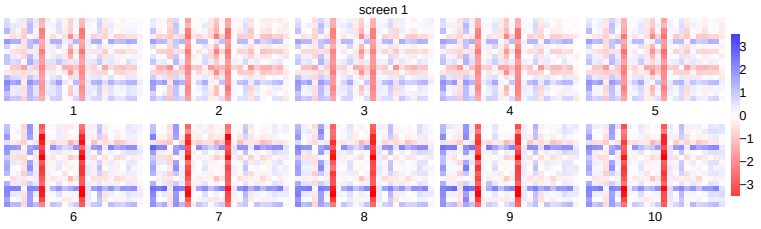

Figure 3: Screen plot for screen 1.

the two biological replicates, plates 6-10, than in the other, plates 1-5. This means that the dynamic range is larger in plates 5-10.

For screen 2 we plot each screen replicate separately, here in Figure 4 is the plot for replicate 1.

Next, plotting replicates against each other will provide important insight to how well the experiments have worked. Systematic errors as well as sporadic contaminations can be detected. The function *WitinScreenPlot* plots the two replicates from the same screen against each other.

We can see in Figure 5 that the correlations between the two within screen replicates in general are good. There are two obvious outliers in screen 2, coloured red in the plot. As the screen was done with replicates, it is possible to identify which of the values should be removed. One of the red spots represent a "P2-P1" data point. We can see that it is Direction 1 in replicate 2 that's the outlier.

```
> dfScl2[dfScl2$Pair =="P2 P1",c("Pair","Direction","replicate","value")]
```

```
      Pair Direction replicate        value
1.2  P2 P1         2         1 -0.041465144
1.25 P2 P1         1         1  0.008910473
2.2  P2 P1         2         2 -0.008517422
2.25 P2 P1         1         2 -3.329200908
3.2  P2 P1         2         3 -0.331242681
3.25 P2 P1         1         3 -0.108987607
```

```
> dfScl2$value[dfScl2$Pair =="P2 P1" & dfScl2$Direction==1 &
+              dfScl2$Replicate==2]=NA
```

```
> plotScreen(split(dfScl2$value[dfScl2$replicate==1],
+                  dfScl2$plate[dfScl2$replicate==1]),fill =
+            c("red","white","blue"),zrange=c(-9,9),
+            main="screen 2",legend.label=NULL)
```

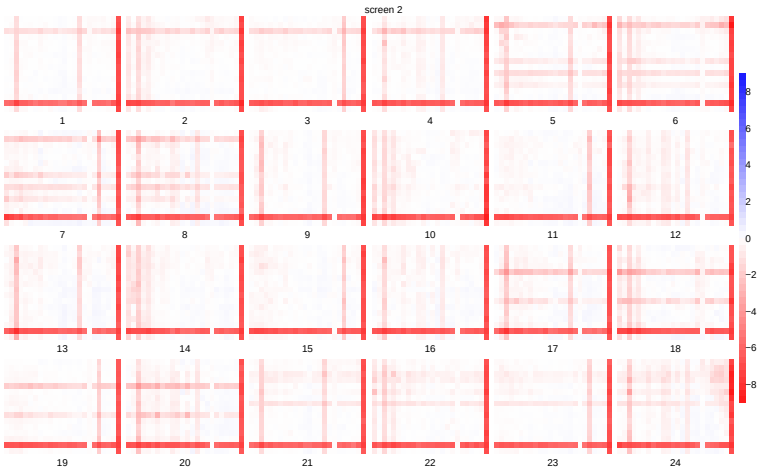

Figure 4: Screen plot for screen 2, replicate 1

For an example of how systematic errors can be detected, we have a look at the dataset `faultyscreen`. Here we have a systematic problem with column 13, caused by a faulty multichannel pipette. Figure 6 shows how the `WithinScreenPlots` look for this dataset before and after removal of the faulty values.

Now back to our good datasets, we will now look at the correlation between the screen replicates. The function *BetweenScreenPlot* plot all replicates of a screen against all. It also shows the spearman correlation between all pairwise comparisons. Here we will plot the between screen plot for screen 1:

The correlations between the screens are high and we do not see any outliers or systematic trends. However, an interesting observation is that replicates 1 to 5 and 6 to 10 have higher similarities to each other than to replicates in the other group. As mentioned before, replicates 1 to 5 are technical replicates of one biological replicate and replicates 6 to 10 are technical replicates of the other biological replicate. For simplicity we will refer to replicates 1-5 as batch 1 and replicates 6-10 as batch 2. The following lines add this information.

```
> par(mfrow=c(1,2))
> WithinScreenPlot(dfScl1,what="value",main="",smooth=T,pch=".")
> WithinScreenPlot(dfScl2,what="value",main="",smooth=T,pch=".")
> points(dfScl2$value[dfScl2$Pair=="P2 P1" & dfScl2$replicate==2 &
+                     dfScl2$Direction==1],dfScl2$value[dfScl2$Pair=="P2 P1" &
+                       dfScl2$replicate==2 & dfScl2$Direction==2],
+        col="red",pch=19)
> points(dfScl2$value[dfScl2$Pair=="P57 P1" & dfScl2$replicate==2 &
+                     dfScl2$Direction==1],dfScl2$value[dfScl2$Pair=="P57 P1" &
+                       dfScl2$replicate==2 & dfScl2$Direction==2],
+        col="red",pch=19)
```

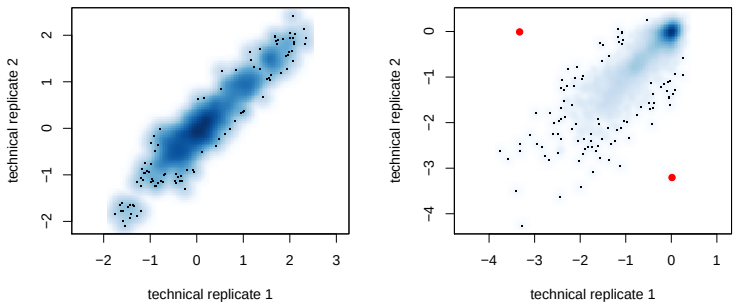

Figure 5: Within screen replication for screen 2.

```
> data(faultyscreen)
> par(mfrow=c(1,2))
> fdf = cellHTS2df(normalizePlates(faultyscreen,method="shorth",
+   scale="multiplicative",log=TRUE),neutral="Fluc")
> WithinScreenPlot(fdf[fdf$replicate==2,],main="with plate column 13"
+                  ,pch=".",smooth=FALSE)
> systerr = which(fdf$Pair%in%(unique(fdf$Pair[grep(13,fdf$well)])))
> fdf$value[systerr]=NA
> WithinScreenPlot(fdf[fdf$replicate==2,],main="without plate column 13"
+                  ,pch=".",smooth=FALSE)
```

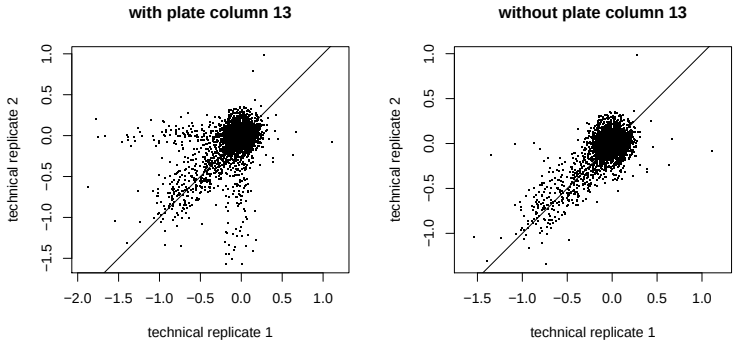

Figure 6: Within screen replicates in faulty screen

```
> BetweenScreenPlot(dfScl1,smooth=FALSE)
```

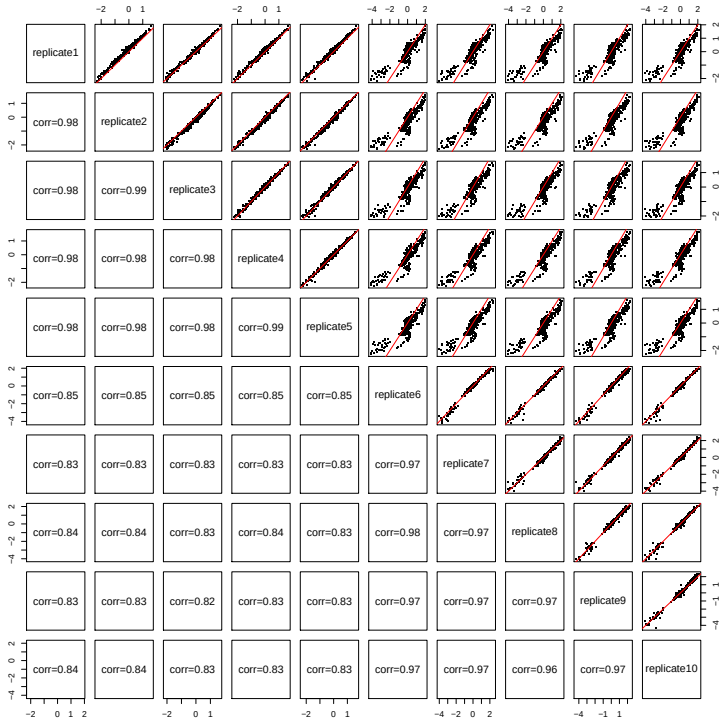

Figure 7: Between screen replicates in screen 1

```
> dfScl1$batch[dfScl1$replicate%in%1:5]=1
> dfScl1$batch[dfScl1$replicate%in%6:10]=2
```

Figure **??** suggests that we might gain power by fitting the main effects separately for each biological replicate, we will investigate this in more details in Section 4 .

Next we have to decide which data we want to include in the analysis. The wells with only RNAi against controls (controlN2, controlP1N1 and controlP2) do not add anything to the interaction analysis and should be excluded. In addition there might be cases when one does not want to include the so called "double" and/or "single" (controlN1) knockdowns. The function *weightDf* is used to exclude unwanted data points from the down stream analysis.

```
> dfScl1 = weightDf(dfScl1,exclude = c("controlN2", "controlP2","controlP1N1"
+                              ,"controlP1","double","controlN1"))
> dfScl2 = weightDf(dfScl2,exclude = c("controlN2", "controlP2","controlP1N1"
+                              ,"controlP1","double","controlN1"))
```

4 Effect estimation

The next step is to estimate the main effects, that is the effects caused by each RNAi separately. This is done by the function *lmmain* or by the *rlmmain* which uses a robust M estimator. As mentioned earlier, it sometimes makes sense to fit the main effects separately for each replicate. From our BetweenScreenPlot earlier we know that the different batches in screen 1 look different so we choose to fit main effects for each batch separately.

```
> lmres1 = lmmain(dfScl1,per = "batch")
```

The three replicates in screen 2 are also biological replicates and we choose to fit the main effects separately for each replicate. We can also fit the main effects separately for each Direction if we so wish.

```
> lmres2D= lmmain(dfScl2,per = c("replicate","Direction"))
> lmres2 = lmmain(dfScl2,per = c("replicate"))
```

The result is a `lm` object or, when different batches, replicates etc are fitted separately, a list with one lm object per fit. The fitted main effects are in the slot `coefficient`. We can have a look at the main effects in screen 1 in the following barplot:

```
> barplot(t(sapply(lmres1,function(x) x$coefficient)),las=2,beside=T,
+         col=c("skyblue","steelblue"))
> legend("topleft",legend=c("batch1","batch2"),fill=c("skyblue","steelblue"))
```

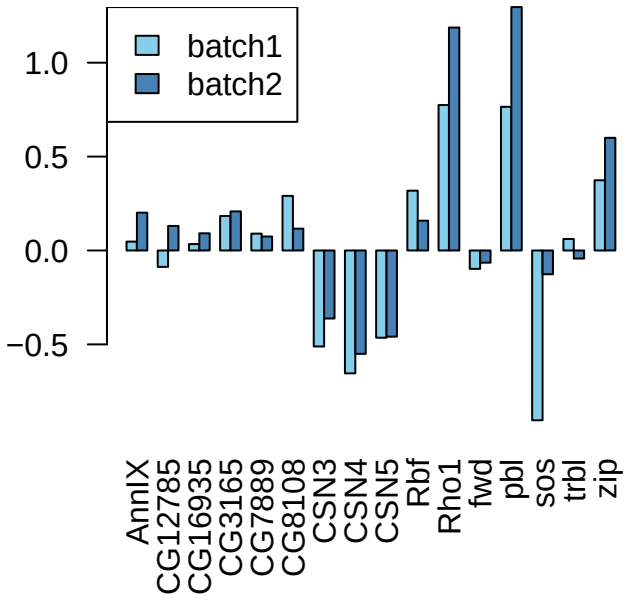

Figure 8: “Main” effects in screen 1

We now have an estimate of the non-interaction effects for each well. The residuals after fitting this model to the data are our interaction observations. To add the residuals to our data frame object we use the function *updateDf* with the `dataframe` and the result from the *lmmain* or *rlmmain* function as arguments.

```
> dfScl1 = updateDf(dfScl1,lmres1,per = "batch")
> dfScl2 = updateDf(dfScl2,lmres2,per=c("replicate"))
> head(dfScl1)
```

```
      plate well controlStatus ID1     ID2      GeneID        Pair Direction
1.1.1     1  A01        sample fwd     pbl     fwd pbl     pbl fwd         2
1.1.2     1  A02        sample fwd CG16935 fwd CG16935 fwd CG16935         1
1.1.3     1  A03        sample fwd  CG3165  fwd CG3165  fwd CG3165         1
1.1.4     1  A04        sample fwd    CSN4    fwd CSN4    fwd CSN4         1
1.1.5     1  A05        sample fwd    Rho1    fwd Rho1    fwd Rho1         1
1.1.6     1  A06        sample fwd    Fluc    fwd Fluc    fwd Fluc         1
           Type       value replicate batch weight  residuals
1.1.1      comb  0.53154683         1     1      1 -0.1355659
1.1.2      comb  0.04178004         1     1      1  0.1055307
1.1.3      comb  0.20369738         1     1      1  0.1180895
1.1.4      comb -0.39451978         1     1      1  0.3577241
1.1.5      comb  0.79709904         1     1      1  0.1205957
1.1.6 controlN1  0.31861514         1     1      0         NA
```

We can see that the data points of type 'comb' now have a residuals value, whereas the data types which had been excluded by the *weightDf* will have NA in the residuals column.

4.1 Fit diagnostic

At this point it is important to have a look at the fit diagnostics. Plotting the residuals as a function of the sum of the two main effects in each well will tell us if there is a trend in the data, e.g. that wells with high expected viability effects have larger residuals. This would indicate that the model used is inappropriate.

The red and blue lines indicate local regression estimates of local mean and standard deviation of ε_{ijk}. We don't see any large trend in the data. In is worth noticing that different knock down pairs might have different variability. For the 50 randomly chosen pairs in screen 1 the boxplot look as this:

```
> par(mfrow=c(1,2))
> MainFitPlot(lmres1,xlab=expression(hat(y)[0*k]+hat(m)[i*k]+hat(m)[j*k]),
+             ylab=expression(epsilon[i*j*k]),pch=".",main="screen 1")
> MainFitPlot(lmres2,xlab=expression(hat(y)[0*k]+hat(m)[i*k]+hat(m)[j*k]),
+             ylab=expression(epsilon[i*j*k]),pch=".",main="screen 2",sd.fit=FALSE)
```

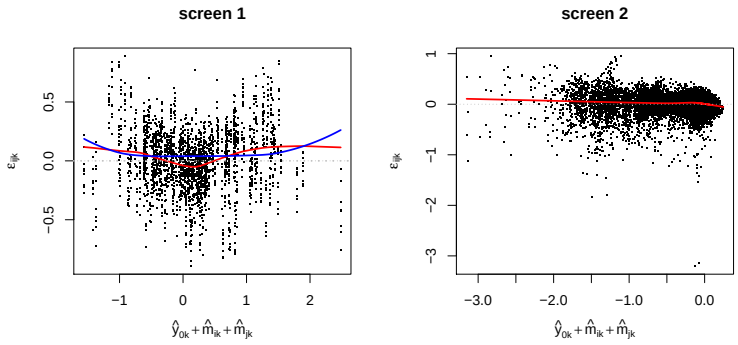

Figure 9: Fit diagnostics for screen 1 and 2.

```
> pairs=sample(unique(dfScl1$Pair[dfScl1$Type=="comb"]),50)
> sub = dfScl1[dfScl1$Pair%in%pairs,]
> boxplot(residuals~Pair,sub,las=2,axis.cex=0.5,col ="palevioletred",
+         ylab=expression(epsilon[i*j*k]),xlab="RNAi combinations",
+         names = NA,xaxt="n")
> abline(h=0,lwd=2)
```

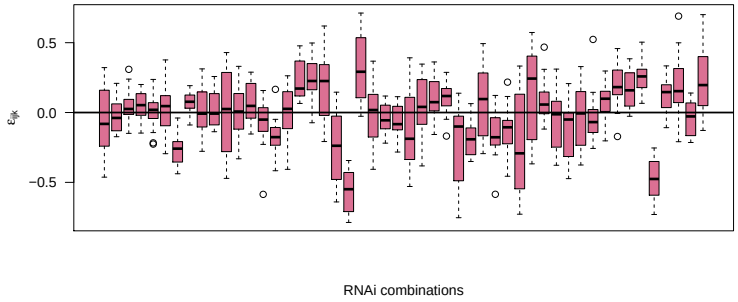

Figure 10: Boxplot of residuals from 50 interaction pairs in screen 1.

Here we can see that different pairs have different distributions. We also see that some boxes are over or under the 0-line, those are potential interactions. The next step now is to decide which of the pairs actually have a significant interaction effect.

4.2 Significance

We use the moderated t-test implemented in the *eBayes* function in the *limma* package. The *df2lmFit* function will take our data frame, format the data and then call the *lmFit* function. In the next step we can run *eBayes* on the resulting object and then summarise the results in a table using *interactiontable*.

```
> ebfit = df2lmFit(dfScl1)
> eb = eBayes(ebfit)
> tt1 =interactiontable(eb,ord.t=TRUE)
> ebfit = df2lmFit(dfScl2)
> eb = eBayes(ebfit)
> tt2 =  interactiontable(eb)
> head(tt2)
```

```
                size     AveExpr          t   P.Value adj.P.Val         B
P10 P1  -0.03268785 -0.03268785 -0.8264217 0.4330918 0.8512556 -5.944463
P11 P1  -0.03359387 -0.03359387 -0.9360597 0.3772869 0.8298942 -5.848945
P11 P10 -0.01242421 -0.01242421 -0.4585930 0.6590215 0.9200436 -6.187933
P12 P1  -0.02612789 -0.02612789 -0.8446439 0.4234339 0.8426232 -5.929263
P12 P10 -0.02318000 -0.02318000 -0.8858898 0.4021368 0.8418516 -5.893851
P12 P11 -0.03252821 -0.03252821 -1.2972469 0.2315626 0.7500673 -5.472697
```

The following command produces the p-value plot that shows the cumulative p-values from the analyses. The bends at small p-values indicates that true interactions are present in this data set.

4.3 Results

In the screen 1 experiment we know which genes are involved in cell-cycling and which genes that were randomly chosen, we can therefore provide some external information to the interaction plots.

```
> data(key)
> colkey = ifelse(key$cellCycle==0,"grey","orange")
```

```
> Pplot(tt2$P,maint = "Screen 2")
```

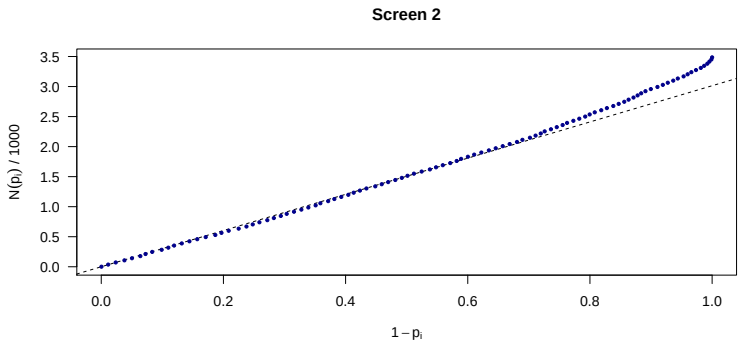

Figure 11: p-value plot for screen 2.

```
> names(colkey)=as.character(key$GeneID)
> key = key$cellCycle
> names(key) = names(colkey)
```

We now set a threshold, here we use 0.001.

```
> thrs1 = 0.001
```

Now we can look at the graph of significant interactions. As the best way to visualise the networks is to use the Graphviz software, the function *data2graph* writes a .dot file to the current directory. This .dot file can then be used to create a pdf file as shown in the code below. The graph for screen 1 is shown in Fig. 12a.

```
> tt1t = tt1
> thrs1 = 0.1
> g1= data2graph(tt1,thres=thrs1,thresBy="ord.p.adj",nodecolor=colkey,
+   sizethres=0.3, writedot=TRUE, filename = 'interactionGraph1.dot',
+   shape='ellipse',scaleFactor=10,fixedsize=FALSE,fontsize=100,
+   penwidth=20,width=2,gamma.col=0)
> system('neato -Tpdf -o interactionGraph1.pdf interactionGraph1.dot')
```

And the graph for screen 2 is shown in Fig. 13a

```
> g2 = data2graph(tt2,thres=thrs1,writedot=TRUE, scaleFactor=10,
+   filename = 'interactionGraph2.dot',fontsize=20,penwidth=10,
```

```
+   width=2,fixedsize=TRUE,nodecolor="grey",sizethres=0.3,gamma.col=0)
> system('neato -Tpdf -o interactionGraph2.pdf interactionGraph2.dot')
```

We can also plot the correlation network by first calculating the correlations between the interaction profiles. The function *tt2matrix* transform the data in the interactiontable to a matrix. *cortestmatrices* calculates the correlation coefficients and the corresponding p-values.

```
> mat = tt2matrix(tt1,what="size")
> thrs=0.9
> cormat = cortestmatrices(mat,method="spearman")
> cormat[[2]][abs(cormat[[1]])<0.8]=1
> c1 = data2graph(list("size"=cormat[[1]],"pvalues"=cormat[[2]]),
+   thres=thrs,writedot=TRUE, scaleFactor=2, filename = 'correlationGraph1.dot',
+   shape='ellipse',fixedsize=FALSE,nodecolor=colkey,fontsize=30,penwidth=5,
+   gamma.col=0)
> system('neato -Tpdf -o correlationGraph1.pdf correlationGraph1.dot')
```

The correlation graph for screen 1 is shown in Fig. 12b

```
> mat = tt2matrix(tt2,what="size")
> cormat = cortestmatrices(mat,method="spearman")
> c2 = data2graph(list("size"=cormat[[1]],"pvalues"=cormat[[2]]),thres=
+   thrs1,writedot=TRUE, scaleFactor=20, filename = 'correlationGraph2.dot',
+   width=2,fontsize=50,penwidth=4,fixedsize=TRUE,nodecolor="grey")
> system('neato -Tpdf -o correlationGraph2.pdf correlationGraph2.dot')
```

The correlation graph for screen 2 is shown in Fig. 13b

Sometimes a levelplot will be more informative, like here for screen 1. It is also possible to cluster all dsRNA based on their interaction profiles.

5 Choice of Scale

In viability screens, the cells are ideally in exponential growth. The number of cells, N is then a function of the time t described by Eq. 1.

$$N(t) = N_0 e^{kt} \tag{1}$$

where N_0 is the number of cells at the beginning of the experiment and k is the growth rate. The readout measurements from the screens are proportional to the number of cells alive in each well. However, one should be

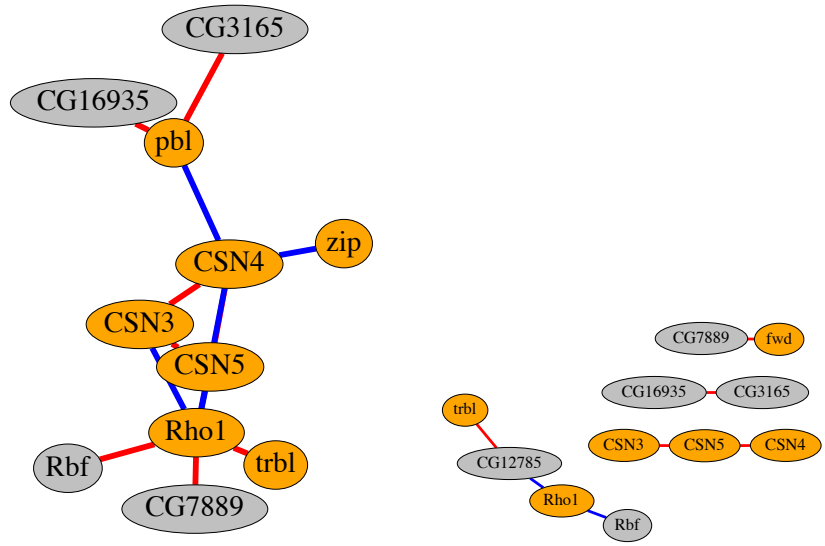

Figure 12: Interaction graph (left panel) and correlation graph (right panel) for screen 1.

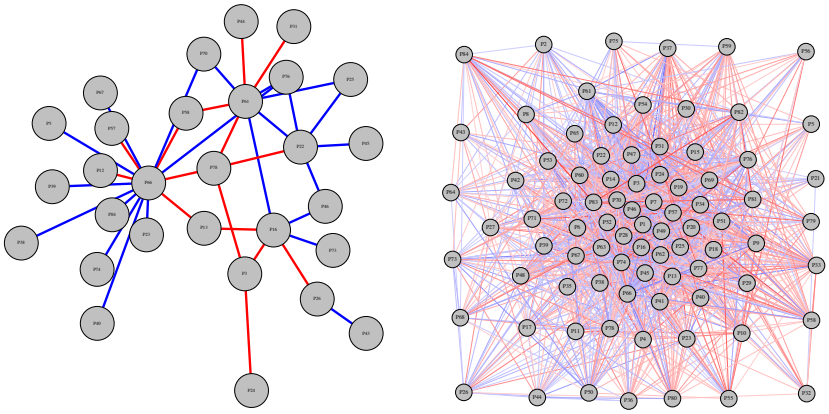

Figure 13: Interaction graph (left panel) and correlation graph (right panel) for screen2.

```
> InteractLevelPlot(tt1,key = key,thresh=thrs1,by="P.Value",zerolimit=0.1,
+                   colorRampPalette(c("blue", "white", "red")))
```

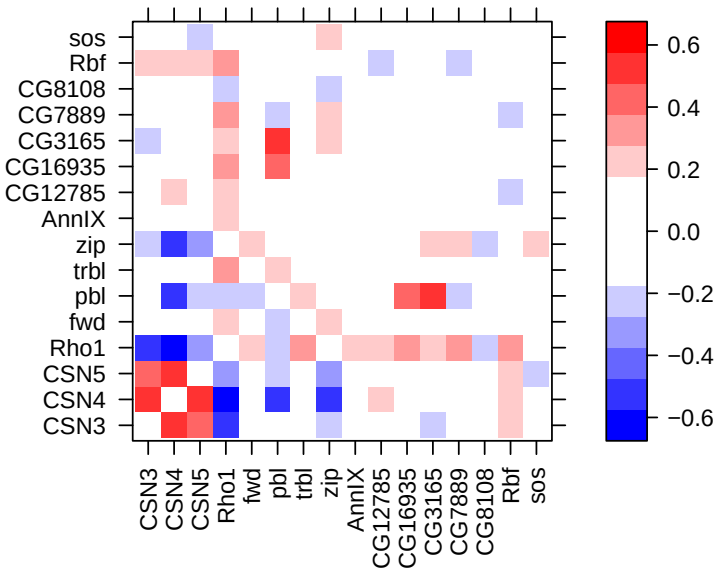

Figure 14: Levelplot for resulting interactions in screen 1.

```
> PlotHeatmap(tt2,dendrogram="none",margins = c(2,2),
+              colorRampPalette(c("blue", "white","red")),
+              lmat=rbind( c(0, 3), c(2,1), c(0,4) ),
+              lhei=c(0.1, 4, 0.1 ) ,lwid=c(0.1,2))
```

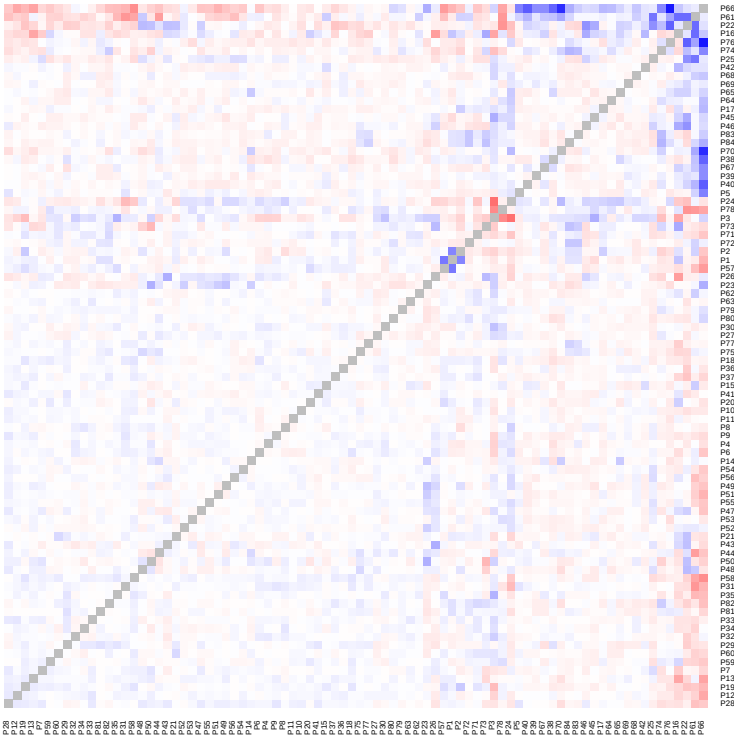

Figure 15: Heatmap representation of the resulting interactions in screen 2.

aware that suboptimal experimental conditions can cause saturation effects and/or unknown background intensities.

In the analysis above the so called *relative population size*, N_{treat}/N_{wt}, measurement was used. That means the ratio between number of cells in each treatment and the number of cells in the wild type. As the experimental conditions, as well as N_0 and t of the were constant over all treatments this measurement is appropriate fitness measurement. Another fitness measurement that has been used in literature is the *relative growth rate*, k_{treat}/k_{wt}. To use the relative growth rate measurement instead of the relative population size one needs to transform the data using Eq. 2.

$$y = \ln \frac{y}{N_0} \tag{2}$$

In the datasets used here, the choice of scale makes very little difference. In fact, for screen 1, the results are practical identical regardless of which scale used. In screen 2 a few data points are affected by the scale choice as can be seen in the following plots.

In our experiments $N_0 = 15.000$, so to get the data on the growth rate scale we do:

```
> N0=15000
> sc2 = screen2_raw
> Data(sc2) = log(Data(sc2)/N0)
```

Using the data in `sc2` we can now re-run our analysis as described above.

```
> ksdf2 = cellHTS2df(normalizePlates(sc2,method="shorth",
+   scale="multiplicative",log=TRUE),neutral="Fluc")
> ksdf2 = weightDf(ksdf2,exclude = c("controlN2", "controlP2",
+                          "controlP1N1","controlP1","double"))
> lmres2 = lmmain(ksdf2,per = c("replicate","Direction"))
> ksdf2 = updateDf(ksdf2,lmres2,per = c("replicate","Direction"))
```

A Q-Q plot of the residuals will show us if the residuals are normally distributed.

There are a few outliers but in general the residuals are normally distributed in both scales. It is not possible to tell which scale is more appropriate, but for the absolute majority of the data points it does not make a difference.

```
> par(mfrow=c(1,2))
> qqnorm(ksdf2$residuals,main="log(log(N/N0))",pch=".")
> qqnorm(dfScl2$residuals,main="log(N)",pch=".")
```

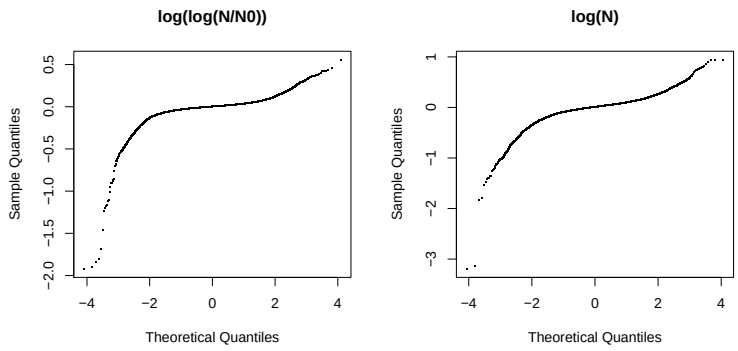

Figure 16: Q-Q plot of residuals using two different scales.

6 Usage of moderated t-test

In cases when there are few replicates, the normal t-test will occasionally underestimate the standard variation within replicates. A moderated t-test will adjust for this. The t-values from the moderated t-test will follow a t-distribution but with a higher number of degrees of freedom which will make the p-values more significant. To show how the moderated t-test compares to the normal t-test in experiments with few replicates, we produce pseudo ROC curves.

We used the data from screen 1. We applied the ranking methods to the data from a single plate, hosting two technical replicate measurements per gene pair. We applied a set of thresholds, with decreasing stringency, to the three ranking methods and obtained the corresponding hit list. We then, for each hit list, computed the true positive rate (TPR) as the ratio between the number of true (as defined by the reference) hits found by the ranking method and the total number of true hits, and the false positive rate (FPR) as the ratio between the number of false (as defined by the reference) hits and the total number of true non-hits. This resulted in an ROC curve per plate, shown in panel (a). The curves indicate clear benefits from using the moderated t or the average effect over the ordinary t. In this particular data set, the variance between replicates from the same plate was constant or close to constant across the different interaction pairs. Hence, the variance

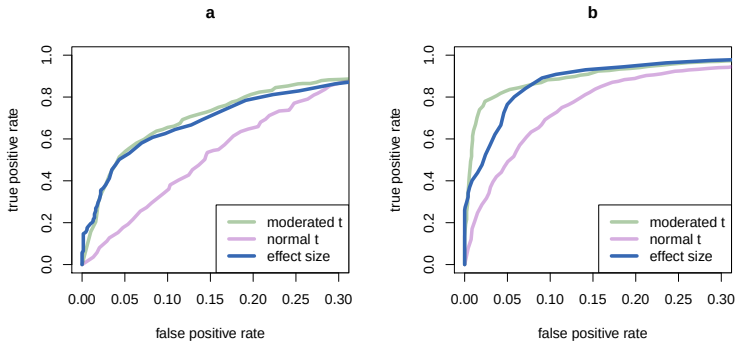

Figure 17: ROC curves for two technical replicates (a) and two biological replicates with two technical replicates each (b).

estimates used in the moderated t test were the same or almost the same for all gene pairs making the resulting moderated t-statistic proportional to the numerator of the t-statistic (the average effect $\hat{w}_{ij}$). Therefore no significant difference in performance could be seen between the two methods in this setting. Nevertheless, we prefer the moderated t approach, as it is more robust in cases where variations are high. This is illustrated by the second set of ROC curves in panel (b), where instead of using only technical replicates, we used two biological replicates hosting two technical replicates each.

The ROC curves in Figure 17 are for normal t-statistic, moderated t-statistic and effect size. With two technical replicates (a) and two biological replicates with two technical replicates each (b). The moderated t-statistic and effect size outperform the ordinary t-statistic in both scenarios, on biological replicates the moderated t-statistics performs better than the effect size.

7 Speeding up analysis for large datasets

The combinatorial RNAi knockdown technique is suitable for large scale experiment. It is possible to test many genes against each other or even to test a gene set against the whole genome. In those cases, it is a reasonable assumption that most of the data will not be affected by either main effects nor interaction effects. This assumption allows for a shortcut in estimating the main effects. As the functions *lmmain* and *rlmmain* tend to be rather

```
> lmm = lmmain(dfScl2)
> mm = estmodel(dfScl2,estimate="median")
> par(mfrow=c(1,2))
> plot(mm$coefficient,lmm$coefficient,pch=".",ylab="OLS estimates",
+      xlab="median estimates",main="main effects")
> abline(0,1,col="red")
> plot(mm$residuals,lmm$residuals,pch=".",ylab="OLS estimates",
+      xlab="median estimates",main="residuals")
> abline(0,1,col="red")
```

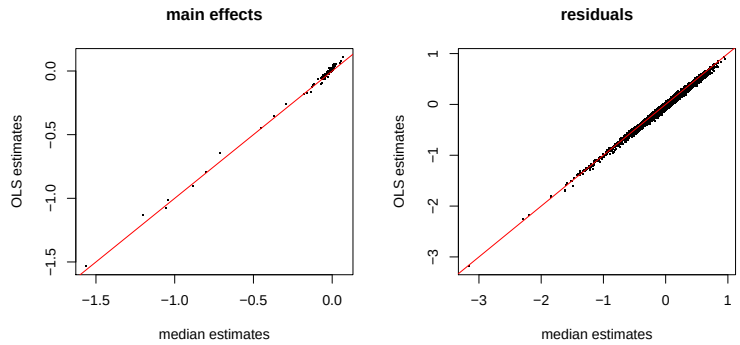

Figure 18: Effect fitted by *lmmain* and *estmodel*.

slow on large datasets this can become convenient.

The *estmodel* estimates the main effects by a location estimate. That means that the estimate for treatment j will be the median/shorth/mean of all (included by the function *weightDf*) wells where treatment j was added.

In screen 2 the underlying assumption is valid and fitting the effects with the *estmodel* generates similar results as the *lmmain*.

Consequently the fit diagnostic plots are also similar

8 Session information

```
R version 3.2.2 (2015-08-14)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```

```
> par(mfrow=c(1,2))
> MainFitPlot(mm,main="fitted by estmodel",pch=".")
> MainFitPlot(lmm,main = "fitted by lmmain",pch=".")
```

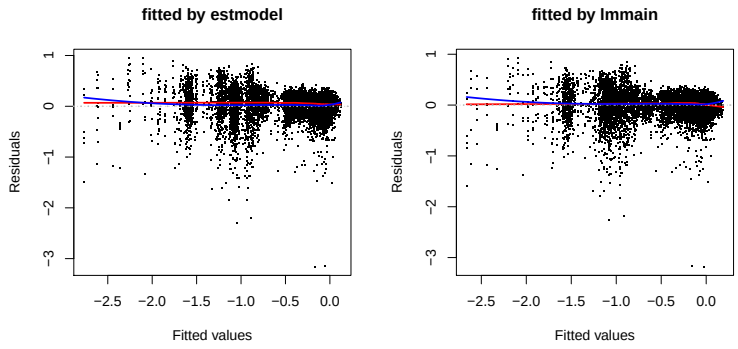

Figure 19: Fit diagnostic for *lmmain* and *estmodel*.

```
attached base packages:
[1] grid      parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] coRNAi_1.20.0       limma_3.26.0        cellHTS2_2.34.0
 [4] locfit_1.5-9.1      hwriter_1.3.2       vsn_3.38.0
 [7] splots_1.36.0       genefilter_1.52.0   Biobase_2.30.0
[10] BiocGenerics_0.16.0 RColorBrewer_1.1-2

loaded via a namespace (and not attached):
 [1] gtools_3.5.0          reshape2_1.4.1        splines_3.2.2
 [4] lattice_0.20-33       pcaPP_1.9-60          colorspace_1.2-6
 [7] stats4_3.2.2          Category_2.36.0       survival_2.38-3
[10] XML_3.98-1.3          RBGL_1.46.0           DBI_0.3.1
[13] affy_1.48.0           affyio_1.40.0         plyr_1.8.3
[16] robustbase_0.92-5     stringr_1.0.0         zlibbioc_1.16.0
[19] munsell_0.4.2         gtable_0.1.2          caTools_1.17.1
[22] mvtnorm_1.0-3         IRanges_2.4.0         BiocInstaller_1.20.0
[25] AnnotationDbi_1.32.0  preprocessCore_1.32.0 DEoptimR_1.0-3
[28] GSEABase_1.32.0       proto_0.3-10          Rcpp_0.12.1
```

```
[31] KernSmooth_2.23-15    xtable_1.7-4          scales_0.3.0
[34] gdata_2.17.0          S4Vectors_0.8.0       graph_1.48.0
[37] annotate_1.48.0       gplots_2.17.0         ggplot2_1.0.1
[40] digest_0.6.8          stringi_0.5-5         bitops_1.0-6
[43] tools_3.2.2           magrittr_1.5          RSQLite_1.0.0
[46] cluster_2.0.3         rrcov_1.3-8           MASS_7.3-44
[49] Matrix_1.2-2          prada_1.46.0
```