```
##############################################################################
##### HEK293 Exon array analysis for differential expression with APT and R ##
##############################################################################

## Background info: arrays are human exon arrays ST 1.0 v2r2
## Considering the purpose of our analysis, we only worked with core probesets.
## A large part of this code was based on the 2010 paper in Briefings in
Bioinformatics by H. Lockstone. We refer to this paper for readers interested in
a detailed tutorial on exon array data analysis using APT and R.
## For convenience, we'll refer to HEK293 as "A", 293S as "S", 293T as "T",
293SG as "RicR" (Ricin Resistant), 293SGGD as "EndoT" and 293FTM as "FlpIn".

# libraries needed in R
library(Biobase)
library(affy)
library(gplots)
library(vsn)
library(pvclust)
library(limma)
library(simpleaffy)
library(siggenes)


##############################################################################
# Background correction, normalisation and probeset summarisation (with APT) #
##############################################################################

# Using Affymetrix Power Tools (APT) under Linux
# Make sure all the CEL files and Affymetrix library files for the Human Exon
Arrays (download from the Affymetrix website) are in your working directory.
# Equally, after downloading APT for Linux, make sure the APT commands are part
of the PATH (the bin folder files need to be in your own personal bin
directory).

# Generate exon-level intensity estimates for core probesets using RMA (if want
to use extended or full, similar), and save it in a folder called OUT_cEXON
apt-probeset-summarize -a rma-sketch -p HuEx-1_0-st-v2.r2.pgf -c
HuEx-1_0-st-v2.r2.clf -s HuEx-1_0-st-v2.r2.dt1.hg18.core.ps --qc-probesets
HuEx-1_0-st-v2.r2.qcc -o OUT_cEXON *.CEL

# Generate gene-level intensities for core probesets using RMA, and save it in a
folder called OUT_cGENE
apt-probeset-summarize -a rma-sketch -p HuEx-1_0-st-v2.r2.pgf -c
HuEx-1_0-st-v2.r2.clf -m HuEx-1_0-st-v2.r2.dt1.hg18.core.mps --qc-probesets
HuEx-1_0-st-v2.r2.qcc -o OUT_cGENE *.CEL

#####################################################################
# Comparison with negative controls: antigenomic probes (with APT) #
#####################################################################

# For each probeset (so per exon), the estimated signal intensities are compared
with those from the antigenomic controls. The degree of overlap is used to
compute a p-value. If <0.05, that probeset is considered as detected. The
results are written in a text file. This is also called the DABG -detection
above background.

apt-probeset-summarize -a dabg -p HuEx-1_0-st-v2.r2.pgf -c HuEx-1_0-st-v2.r2.clf
-b HuEx-1_0-st-v2.r2.antigenomic.bgp -o ./OUT_cDABG *.cel

############################
# Some QC in R: exon level #
############################

# The "hekphenodata" text file (tab-delimited) describes which CEL files
correlate with which sample name and group.

pd <- read.table("hekphenodata.txt", header=TRUE)
qcexonc <- read.table("OUT_cEXON/rma-sketch.report.txt", sep="\t", header=T)
```

```
d.exonc <- read.table("OUT_cEXON/rma-sketch.summary.txt", sep="\t", header=T,
row.names=1)

# Set the column-names in the d.exonc object to match our sample definitions
colnames(d.exonc) <- pd$Name

# To plot the Average raw signal instensity
plot(qcexonc$pm_mean, ylim=c(0,1000), xlab="Array", ylab="Signal Intensity",
main="Average Raw Intensity Signal")

# Plot of the deviation of residuals from median from the intensities gotten
from apt
plot(qcexonc$all_probeset_mad_residual_mean, ylim=c(0,0.5), xlab="Array",
ylab="Mean absolute deviation", main="Deviation of Residuals from Median")

# Density plot
plot(density(d.exonc[,1]), main="Distribution of RMA-normalised intensities",
xlab="RMA normalised intensity")
for(i in 2:ncol(d.exonc)) {lines (density(d.exonc[,i]))}

# Hierarchical clustering
d.t <- dist(t(d.exonc))
plot(hclust(d.t), main="Hierarchical clustering")

# PCA
colors<-rainbow(nlevels(pd$Line))
pca <- princomp(d.exonc)
plot(pca$loadings, main="Principal Component Analysis", col=colors[pd$Line],
pch=19, cex=2)
text(pca$loadings, as.character(pd$Name), pos=3, cex=0.8)

# Heat map
heatmap(cor(d.exonc), symm=T)

# Boxplot
boxplot(d.exonc,col=rainbow(18),names=pd$Name, main="RMA normalized",
outline=FALSE,las=3,cex.axis=0.4)

# Other dendrogram (also hierarchical clustering, but based on p-values)
corClust <- pvclust(d.exonc, nboot=5, method.dist="correlation")
plot(corClust,cex=0.4, main="p-value clustering")


# To put all of this in one pdf file

pdf("core_exon_QC.pdf")
par(mfrow=c(1,2))
plot(qcexonc$pm_mean, ylim=c(0,1000), xlab="Array", ylab="Signal Intensity",
main="Average Raw Intensity Signal")
plot(qcexonc$all_probeset_mad_residual_mean, ylim=c(0,0.5), xlab="Array",
ylab="Mean absolute deviation", main="Deviation of Residuals from Median")
plot(density(d.exonc[,1]), main="Distribution of RMA-normalised intensities",
xlab="RMA normalised intensity")
for(i in 2:ncol(d.exonc)) {lines (density(d.exonc[,i]))}
d.t <- dist(t(d.exonc))
plot(hclust(d.t), main="Hierarchical clustering")

par(mfrow=c(1,1))
boxplot(d.exonc,col=rainbow(18),names=pd$Name, main="RMA normalized",
outline=FALSE,las=3,cex.axis=0.4)
heatmap(cor(d.exonc), symm=T)
corClust <- pvclust(d.exonc, nboot=5, method.dist="correlation")
plot(corClust,cex=0.4, main="p-value clustering")
plot(pca$loadings, main="Principal Component Analysis", col=colors[pd$Line],
pch=19, cex=2)
text(pca$loadings, as.character(pd$Name), pos=3, cex=0.8)

dev.off()
```

```
##################################
# Some QC in R: transcript level #
##################################

qcgenec <- read.table("OUT_cGENE/rma-sketch.report.txt", sep="\t", header=T)
d.genec <- read.table("OUT_cGENE/rma-sketch.summary.txt", sep="\t", header=T,
row.names=1)
# Set the column-names in the d.exonc object to match our sample definitions
colnames(d.genec)
colnames(d.genec) <- pd$Name
colnames(d.genec)

# To plot the Average raw signal instensity
plot(qcgenec$pm_mean, ylim=c(0,1000), xlab="Array", ylab="Signal Intensity",
main="Average Raw Intensity Signal")

# Plot of the deviation of residuals from median from the intensities gotten
from apt
plot(qcgenec$all_probeset_mad_residual_mean, ylim=c(0,0.5), xlab="Array",
ylab="Mean absolute deviation", main="Deviation of Residuals from Median")

# Density plot
plot(density(d.genec[,1]), main="Distribution of RMA-normalised intensities",
xlab="RMA normalised intensity")
for(i in 2:ncol(d.genec)) {lines (density(d.genec[,i]))}

# Hierarchical clustering
d.t2 <- dist(t(d.genec))
plot(hclust(d.t2), main="Hierarchical clustering")

# PCA
colors<-rainbow(nlevels(pd$Line))
pca2 <- princomp(d.genec)
plot(pca$loadings, main="Principal Component Analysis", col=colors[pd$Line],
pch=19, cex=2)
text(pca$loadings, as.character(pd$Name), pos=3, cex=0.8)

# Heat map
heatmap(cor(d.genec), symm=T)

# Boxplot
boxplot(d.genec,col=rainbow(18),names=pd$Name, main="RMA normalized",
outline=FALSE,las=3,cex.axis=0.4)

# Other dendrogram
corClust2 <- pvclust(d.genec, nboot=5, method.dist="correlation")
plot(corClust,cex=0.4, main="p-value clustering")

# Now all in one pdf

pdf("core_gene_QC.pdf")

plot(qcgenec$pm_mean, ylim=c(0,1000), xlab="Array", ylab="Signal Intensity",
main="Average Raw Intensity Signal")
plot(qcgenec$all_probeset_mad_residual_mean, ylim=c(0,0.5), xlab="Array",
ylab="Mean absolute deviation", main="Deviation of Residuals from Median")
plot(density(d.genec[,1]), main="Distribution of RMA-normalised intensities",
xlab="RMA normalised intensity")
for(i in 2:ncol(d.genec)) {lines (density(d.genec[,i]))}
plot(hclust(d.t2), main="Hierarchical clustering")

boxplot(d.genec,col=rainbow(18),names=pd$Name, main="RMA normalized",
outline=FALSE,las=3,cex.axis=0.4)
heatmap(cor(d.genec), symm=T)
plot(corClust,cex=0.4, main="p-value clustering")
plot(pca$loadings, main="Principal Component Analysis", col=colors[pd$Line],
pch=19, cex=2)
```

```
text(pca$loadings, as.character(pd$Name), pos=3, cex=0.8)

dev.off()


#####################################################
# Filtering of genes undetected in every cell line  #
#####################################################

# Unlike for exon-level analysis, cross-hybridizing probesets are already
excluded in the .mps files, so there is no need to filter out those.
# To simplify the dataset for DE analysis, we'll remove genes that we consider
to be undetected in all samples. For this we'll turn to the DABG-file (see
above). Although the DABG is an exon-level feature, one can use it at the
gene-level when assuming that a gene can be considered as detected in a sample
if more than half of its exons have a dabg p-value of <0.05.
# Following this rationale, we will also consider a gene as expressed in a group
when it is detected in more than half of the samples of that group.

dabg <- read.table("./OUT_cDABG/dabg.summary.txt", sep="\t", header=T,
row.names=1)
dim(dabg)
# 1411399 18, so there are about a million probesets in total. To check what the
dabg file looks like:
head(dabg)

dabg.core <- dabg[match(row.names(d.exonc), row.names(dabg)),]
dim(dabg.core)
# 287329 18, so there are about 280 000 "core" probesets.
head(dabg.core)

annot <- read.table("HuEx-1_0-st-v2.na29.hg18.probeset.csv", sep=",", header=T)

length(intersect(row.names(dabg.core), annot[,1]))
# length command gives the same, so there are no probeset IDs are missing from
annotation file (otherwise likely to be control probes).

# Use the annotation table to add an extra column to the dabg.core set
(transcript cluster IDs).
dabg.core[,19] <- annot[match(row.names(dabg.core), annot$probeset_id), 7]
gene.ids <- unique(dabg.core[,19])
length(gene.ids) # gives 18727 - the amount of genes represented by core
probesets on the array.

# Now, define a function to calculate the proportion of probesets with p < 0.05â
this will be applied to each sample for each gene:
count.exonc.det <- function(x){length (which(x<0.05))/length(x)}

# Create an empty matrix to store results from applying the above function:

gene.detection <- matrix(nrow=length(unique(dabg.core[,19])), ncol=18)
row.names (gene.detection) <- gene.ids
colnames(gene.detection) <- colnames(d.genec)

# Mind that in this case, we use the colnames of the original d.genec (the
hyb-file names), but if you use the d.genec set after the command
`colnames(d.genec) <- pd$Name` you`ll get the `A1` etc nomenclature.

# Apply the function to each sample in turn:

for(i in 1:18)
        {
        gene.detection[,i] <- tapply(dabg.core[,i], dabg.core[,19],
count.exonc.det)
       }

# Now, another function is needed to count how many samples in each group the
gene was detected in (i.e. more than half the probesets were detected):
```

```
count.genec.det <- function(x){length (which(x>=0.5))}

aa <- cbind(gene.detection[,1], gene.detection[,7], gene.detection[,12])
genes.det.groupA <- apply(aa[,1:3], 1, count.genec.det)

bb <- cbind(gene.detection[,2], gene.detection[,8], gene.detection[,13])
genes.det.groupS <- apply(bb[,1:3], 1, count.genec.det)

cc <- cbind(gene.detection[,3], gene.detection[,9], gene.detection[,14])
genes.det.groupT <- apply(cc[,1:3], 1, count.genec.det)

dd <- cbind(gene.detection[,4], gene.detection[,15], gene.detection[,18])
genes.det.groupEndoT <- apply(dd[,1:3], 1, count.genec.det)

ee <- cbind(gene.detection[,5], gene.detection[,10], gene.detection[,16])
genes.det.groupFlpIn <- apply(ee[,1:3], 1, count.genec.det)

ff <- cbind(gene.detection[,6], gene.detection[,11], gene.detection[,17])
genes.det.groupRicR <- apply(ff[,1:3], 1, count.genec.det)

# Keep genes where counts are >= 2 (i.e. more than one-half the samples in 1
group) in at least one cell line:

keep.genes <-
which((genes.det.groupA>=2)|(genes.det.groupS>=2)|(genes.det.groupT>=2)|(genes.d
et.groupEndoT>=2)|(genes.det.groupFlpIn>=2)|(genes.det.groupRicR>=2)) # row
numbers
length(keep.genes) # gives 17027. This means that around 1700 genes are not
expressed in any of the cell lines (according to out criterium at least).
keep.gene.ids <- row.names (gene.detection)[keep.genes]

length(intersect(keep.gene.ids, row.names(d.genec)))
# Gives us 16283 genes. This tells us that 744 genes are not in the gene summary
output; these genes have no core probesets that map uniquely (as said before,
the mps-file already accounted for cross-hybridizing probes).

y <- match(keep.gene.ids, row.names(d.genec))

# remove the NAs
y <- y[-which(is.na(y)=="TRUE")]
d.genec.fil <- d.genec[y,]
dim(d.genec.fil) # 16283 18
write.table(d.genec.fil, "OUT_cGENE/rma-sketch.summary_core_gene_filtered.txt",
sep="\t", quote=F, row.names=T)

#########################
# Removal of noisy data #
#########################

# Despite the extensive filtering done above, there is still some noise left.
This is exemplified by the signals we get for unexpected loci (eg genes on the Y
chromosome - the HEK lines are female).
# The highest log2 intensity signal for the genes on the Y-chromosome are around
7, we'll consider only the signals 1 log2 above that noise to be true signals
(value of 8). This is also supported by the drop in signal intensity below
values of 8 in the density plot.

plot(density(d.genec.fil[,1]), main="Distribution of RMA-normalised intensities,
fil", xlab="RMA normalised intensity")
for(i in 2:ncol(d.genec.fil)) {lines (density(d.genec.fil[,i]))}

# Following that rationale, we will only keep the probesets for which the
average value is 8 or more in at least 1 group (or cell line in this case). Note
that this is a very stringent requirement.

# To present the mean signal intensity per probeset, use the function rowMeans,
after creation of group-specific data frames
```

```
groupA <- cbind(d.genec.fil[,1], d.genec.fil[,7], d.genec.fil[,12])
head(groupA)
rownames(groupA) <- rownames(d.genec.fil)
meanA <- rowMeans(groupA)
lineA <- data.frame(meanA)
head(lineA)

groupS <- cbind(d.genec.fil[,2], d.genec.fil[,8], d.genec.fil[,13])
head(groupS)
rownames(groupS) <- rownames(d.genec.fil)
meanS <- rowMeans(groupS)
lineS <- data.frame(meanS)
head(lineS)

groupT <- cbind(d.genec.fil[,3], d.genec.fil[,9], d.genec.fil[,14])
head(groupT)
rownames(groupT) <- rownames(d.genec.fil)
meanT <- rowMeans(groupT)
lineT <- data.frame(meanT)
head(lineT)

groupEndoT <- cbind(d.genec.fil[,4], d.genec.fil[,15], d.genec.fil[,18])
head(groupEndoT)
rownames(groupEndoT) <- rownames(d.genec.fil)
meanEndoT <- rowMeans(groupEndoT)
lineEndoT <- data.frame(meanEndoT)
head(lineEndoT)

groupFlpIn <- cbind(d.genec.fil[,5], d.genec.fil[,10], d.genec.fil[,16])
head(groupFlpIn)
rownames(groupFlpIn) <- rownames(d.genec.fil)
meanFlpIn <- rowMeans(groupFlpIn)
lineFlpIn <- data.frame(meanFlpIn)
head(lineFlpIn)

groupRicR <- cbind(d.genec.fil[,6], d.genec.fil[,11], d.genec.fil[,17])
head(groupRicR)
rownames(groupRicR) <- rownames(d.genec.fil)
meanRicR <- rowMeans(groupRicR)
lineRicR <- data.frame(meanRicR)
head(lineRicR)

meantable <- cbind(lineA, lineS, lineT, lineEndoT, lineFlpIn, lineRicR)
head(meantable)

# Define a new function, to count how many lines have a log2 signal intensity
value >=8. Then apply on rows.
count.noise <- function(x){length (which(x>=8))}
count.table <- apply(meantable[,1:6], 1, count.noise)

# The probesets for which at least one cell line has a mean value of 8 or more
noisefree <- which(count.table>=1)
head(noisefree)
class(noisefree)
noisefree.sorted <- sort(noisefree)
length(noisefree.sorted) # is now 8748, so roughly half is kept

#####################################################################
# Differential gene expression - linear model fit via limma in R #
#####################################################################

# Mind that in this case, the colnames are the hybnumbers.
# The probeset_ids also have a header.
hekeset
<-read.table("OUT_cGENE/rma-sketch.summary_core_gene_filtered.txt",header=TRUE,r
ow.names=1)
head(hekeset)
```

```
# You can opt to do the analysis on the entire filtered set (then ignore this
command) or the one excluding noisy data, the rest is the same.
hekeset <- hekeset[noisefree.sorted,]

# Set the column names to match sample definition. If you`re starting a new
session, you`ll have to redefine `pd`
pd <- read.table("hekphenodata.txt", header=TRUE)
pd
colnames(hekeset) <- pd$Name

# Make a design matrix
genotypes <- factor(pd$Line, levels=c("A","S","T","EndoT","FlpIn","RicR"))
hekdesign <- model.matrix(~0+genotypes)
colnames(hekdesign) <- c("A","S","T","EndoT","FlpIn","RicR")
hekdesign

# Make the contrast matrix: this defines what samples you want to compare
contrast.matrix <- makeContrasts(FlpIn-A, T-A, S-A, RicR-S, EndoT-RicR, levels=
hekdesign)
contrast.matrix

# Start fitting
hekfit <- lmFit(hekeset, hekdesign)
hekfit2 <- contrasts.fit(hekfit, contrast.matrix)
hekfit2 <- eBayes(hekfit2)
topTable(hekfit2, coef=1) # default BH, you can also try other ones
topTable(hekfit2, coef=1, adjust="BH")

# The decideTests command gives which genes are upregulated, and which down. lfc
allows you to determine the min. log-fold change (in absolute values!)
# The outcome of each hypothesis can be assigned to
result <- decideTests(hekfit2, p.value=0.01, lfc=1, adjust="BH")
# Below command now selects all those genes for which the value of the
decideTests command is different from (!=) 0
keep2 <- result@".Data" !=0
head(keep2)

# This `keep2` list thus contains all comparisons defined, and whether each gene
is differentially expressed, at least 2-fold, with p-value of 0.01.
# It is a table with the all comparisons as colnames, the rownames being the
probeset_id, and the elements being either TRUE (diff expr) or FALSE.
# You can check how many genes it holds (16283 rows, 5 columns)
dim(keep2)
# and how many genes are significant for each comparison
sum(keep2[,1]) # 358 [without noise reduction: 476], FlpIn vs A
sum(keep2[,2]) # 489 [without noise reduction: 681], T vs A
sum(keep2[,3]) # 542 [without noise reduction: 750], S vs A
sum(keep2[,4]) # 85 [without noise reduction: 122], RicR vs S
sum(keep2[,5]) # 10 [without noise reduction: 12], EndoT vs RicR

# keep seperate lists from each comparison
FlpInvsA <-rownames(result@".Data")[keep2[,1]]
head(FlpInvsA)
length(FlpInvsA)

TvsA <-rownames(result@".Data")[keep2[,2]]
SvsA <-rownames(result@".Data")[keep2[,3]]
RicRvsS <-rownames(result@".Data")[keep2[,4]]
EndoTvsRicR <-rownames(result@".Data")[keep2[,5]]

# Now annotate the genes as to make sense of the results.
# Mind that the probeset_ids from the gene-level analysis are not the same as
for exon-level analysis, this can be confusing.
hekannotfile <- "HuEx-1_0-st-v2.na29.hg18.transcript.csv"
hekanTable <- read.table(hekannotfile, sep=",", header=TRUE,
quote="\"",row.names=1)
colnames(hekanTable)
# [1] "probeset_id"           "seqname"               "strand"
```

```
 [4] "start"                 "stop"                  "total_probes"
 [7] "gene_assignment"       "mrna_assignment"       "swissprot"
[10] "unigene"               "GO_biological_process" "GO_cellular_component"
[13] "GO_molecular_function" "pathway"               "protein_domains"
[16] "category"

# We will select the genes via the lists from topTable
# the argument `resort.by` is necessary to not sort by absolute value, but also
from most negative to most positive
# coef=1 means that you only look at the first comparison (in our case, A vs S)
# number= 16283 for dataset without noise removal
a <- topTable(hekfit2, coef=1, number=8748, sort.by="logFC", resort.by="logFC",
adjust="BH")
rownames(a) <- a$ID
b <- a[FlpInvsA,]

newcolumns <- c("probeset_id","logFC", "AveExpr","t","P.Value","adj.P.Val","B" )
colnames(b) <- newcolumns

# Now merge this with the annotation table.
FlpInAtemp <- hekanTable[rownames(b),]
dim(FlpInAtemp)
FlpInvsAgenes_2fc_1p_fil_hg18 <-merge(b,FlpInAtemp, by.x="probeset_id")

# Save it as a text file.
write.table(FlpInvsAgenes_2fc_1p_fil_hg18,
"FlpInvsA_2fc_1p_cgenes_fil_hg18_nn.txt", sep="\t", quote=F, row.names=T)


# Now repeat this for the other gene comparisons

c <- topTable(hekfit2, coef=2, number=8748, sort.by="logFC", resort.by="logFC",
adjust="BH")
rownames(c) <- c$ID
d <- c[TvsA,]
dim(d)
colnames(d) <- newcolumns
TAtemp <- hekanTable[rownames(d),]
dim(TAtemp)
TvsAgenes_2fc_1p_fil_hg18 <-merge(d,TAtemp, by.x="probeset_id")
write.table(TvsAgenes_2fc_1p_fil_hg18, "TvsA_2fc_1p_cgenes_fil_hg18_nn.txt",
sep="\t", quote=F, row.names=T)

e <- topTable(hekfit2, coef=3, number=8748, sort.by="logFC", resort.by="logFC",
adjust="BH")
rownames(e) <- e$ID
f <- e[SvsA,]
dim(f)
colnames(f) <- newcolumns
SAtemp <- hekanTable[rownames(f),]
dim(SAtemp)
SvsAgenes_2fc_1p_fil_hg18 <-merge(f,SAtemp, by.x="probeset_id")
write.table(SvsAgenes_2fc_1p_fil_hg18, "SvsA_2fc_1p_cgenes_fil_hg18_nn.txt",
sep="\t", quote=F, row.names=T)

g <- topTable(hekfit2, coef=4, number=8748, sort.by="logFC", resort.by="logFC",
adjust="BH")
rownames(g) <- g$ID
h <- g[RicRvsS,]
dim(h)
colnames(h) <- newcolumns
RicRStemp <- hekanTable[rownames(h),]
dim(RicRStemp)
RicRvsSgenes_2fc_1p_fil_hg18 <-merge(h,RicRStemp, by.x="probeset_id")
write.table(RicRvsSgenes_2fc_1p_fil_hg18,
"RicRvsS_2fc_1p_cgenes_unfil_hg18_nn.txt", sep="\t", quote=F, row.names=T)

i <- topTable(hekfit2, coef=5, number=8748, sort.by="logFC", resort.by="logFC",
```

```
adjust="BH")
rownames(i) <- i$ID
j <- i[EndoTvsRicR,]
dim(j)
colnames(j) <- newcolumns
EndoTRicRtemp <- hekanTable[rownames(j),]
dim(EndoTRicRtemp)
EndoTvsRicRgenes_2fc_1p_fil_hg18 <-merge(j,EndoTRicRtemp, by.x="probeset_id")
write.table(EndoTvsRicRgenes_2fc_1p_fil_hg18,
"EndoTvsRicR_2fc_1p_cgenes_fil_hg18_nn.txt", sep="\t", quote=F, row.names=T)


# Visualise genes using volcano plots

# Saved in a pdf
pdf("Volcanoplots_filtered.pdf")
par(mfrow=c(2,2))
plot(a$logFC,-log10(a$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="FlpIn vs A gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')

plot(c$logFC,-log10(c$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="T vs A gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')

plot(e$logFC,-log10(e$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="S vs A gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')

plot(g$logFC,-log10(g$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="RicR vs S gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')

plot(i$logFC,-log10(i$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="EndoT vs RicR gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')

dev.off()

# Similarly, you can save the plots as image files.

png("Volcanoplot_FlpInvsA_fil.png")
plot(a$logFC,-log10(a$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="FlpIn vs A gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')
dev.off()

png("Volcanoplot_TvsA_fil.png")
plot(c$logFC,-log10(c$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="T vs A gene-level" )
```

```
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')
dev.off()

png("Volcanoplot_SvsA_fil.png")
plot(e$logFC,-log10(e$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="S vs A gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')
dev.off()

png("Volcanoplot_RicRvsS_fil.png")
plot(g$logFC,-log10(g$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="RicR vs S gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')
dev.off()

png("Volcanoplot_EndoTvsRicR_fil.png")
plot(i$logFC,-log10(i$adj.P.Val), ylab="-log10(adj P-value)", xlab="log2 Fold
Change", main="EndoT vs RicR gene-level" )
abline(h=-log10(0.01), col='blue')
abline(h=-log10(0.001), col='green')
abline(v=-log2(2), col='red')
abline(v=log2(2), col='red')
dev.off()

##############################
# Venn Diagrams of S-lineage #
##############################

# As the S-RicR-EndoT lines are from the same lineage, we expect the genes that
we detect to be differentially expressed in S vs A to be conserved as DE genes
in RicR vs A and EndoT vs A.

contrast.matrix2 <- makeContrasts(S-A, RicR-A, EndoT-A, levels= hekdesign)
contrast.matrix2

# Start fitting
hekfit.Slineage <- contrasts.fit(hekfit, contrast.matrix2)
hekfit.Slineage <- eBayes(hekfit.Slineage)
topTable(hekfit.Slineage, coef=1) # default BH, you can also try other ones

result.Slineage <- decideTests(hekfit.Slineage, p.value=0.01, lfc=1,
adjust="BH")

vennDiagram(result.Slineage[,1:3], include=c("up","down"), circle.col=2:4,
main="S lineage")
vennDiagram(result.Slineage[,1:2], include=c("up","down"), circle.col=2:4,
main="S lineage")

png("Slineage_Venn_nonoise.png")
vennDiagram(result.Slineage[,1:3], include=c("up","down"), circle.col=2:4,
main="S lineage")
dev.off()

png("SandRicR_Venn_nonoise.png")
vennDiagram(result.Slineage[,1:2], include=c("up","down"), circle.col=2:4,
main="S lineage")
dev.off()

##############################################################################
```

```
# To extract a list of all genes detected (eg for functional analysis in DAVID)#
#################################################################################

annot.expressed<-hekanTable[rownames(hekeset),]
write.table(annot.expressed, "Expressed_core_fil_genes.txt", sep="\t", quote=F,
row.names=T)

########################
# Integration with IGV #
########################

### The IGV track "GenesOnExonArray" provides a link towards the processed data,
including (per comparison between 2 lines) raw and adjusted p-value, t-statistic
and log2 fold change. For the sake of clarity and completeness, here we will use
the dataset proir to noise-removal. Make sure you redefine the hekfit2 object to
reflect the dataset you want.

a <- topTable(hekfit2, coef=1, number=16283, sort.by="logFC", resort.by="logFC",
adjust="BH")
newcolumns <- c("probeset_id","logFC", "AveExpr","t","P.Value","adj.P.Val","B" )
colnames(a) <- newcolumns
rownames(a) <- a$probeset_id
# Mind that the probeset_id here reflects the transcript_cluster_id, and is not
the same as the probeset_id of the exon-level analysis.
write.table(a, "FlpInvsA_IGV_linkdata.txt", sep="\t", quote=F, row.names=T)

c <- topTable(hekfit2, coef=2, number=16283, sort.by="logFC", resort.by="logFC",
adjust="BH")
colnames(c) <- newcolumns
rownames(c) <- c$probeset_id
write.table(c, "TvsA_IGV_linkdata.txt", sep="\t", quote=F, row.names=T)

e <- topTable(hekfit2, coef=3, number=16283, sort.by="logFC", resort.by="logFC",
adjust="BH")
colnames(e) <- newcolumns
rownames(e) <- e$probeset_id
write.table(e, "SvsA_IGV_linkdata.txt", sep="\t", quote=F, row.names=T)

g <- topTable(hekfit2, coef=4, number=16283, sort.by="logFC", resort.by="logFC",
adjust="BH")
colnames(g) <- newcolumns
rownames(g) <- g$probeset_id
write.table(g, "RicRvsS_IGV_linkdata.txt", sep="\t", quote=F, row.names=T)

i <- topTable(hekfit2, coef=5, number=16283, sort.by="logFC", resort.by="logFC",
adjust="BH")
colnames(i) <- newcolumns
rownames(i) <- i$probeset_id
write.table(i, "EndoTvsRicR_IGV_linkdata.txt", sep="\t", quote=F, row.names=T)

### The IGV track referring to the differentially expressed genes (pairwise
comparisons) allows visualisation of every gene that has been detected as
significant (p<0.01) in the comparison of interest, starting from the filtered
and noise-removed dataset. Additionally, information about the associated
fold-change is included as a function of the height of the bar. For this, we
need to extract all the data for p<0.01, for any log2 fold change value. Make
sure you redefine the hekfit2 object to reflect the dataset you want (in our
case, noise-removed)

result.igv <- decideTests(hekfit2, p.value=0.01, adjust="BH")
keep.igv <- result.igv@".Data" !=0
head(keep.igv)

sum(keep.igv[,1]) # 2304 [2834 without noise-removal], FlpIn vs A
sum(keep.igv[,2]) # 2512 [3117 without noise-removal], T vs A
sum(keep.igv[,3]) # 2785 [3523 without noise-removal], S vs A
sum(keep.igv[,4]) # 721 [839 without noise-removal], RicR vs S
sum(keep.igv[,5]) # 46 [44 without noise-removal], EndoT vs RicR
```

```
FlpInvsA_igv <-rownames(result.igv@".Data")[keep.igv[,1]]
length(FlpInvsA_igv)
TvsA_igv <-rownames(result.igv@".Data")[keep.igv[,2]]
SvsA_igv <-rownames(result.igv@".Data")[keep.igv[,3]]
RicRvsS_igv <-rownames(result.igv@".Data")[keep.igv[,4]]
EndoTvsRicR_igv <-rownames(result.igv@".Data")[keep.igv[,5]]

subanTable <- subset(hekanTable, select=probeset_id:stop)
dim(subanTable)
colnames(subanTable)

FA1p_table <- a[FlpInvsA_igv,]
head(FA1p_table)
dim(FA1p_table)
FA_subanTable <- subanTable[FlpInvsA_igv,]
FA1p_table <- merge(FA1p_table, FA_subanTable, by.x="probeset_id")
head(FA1p_table)
dim(FA1p_table)
write.table(FA1p_table, "FlpInvsA_IGV_DEtrack_nn.txt", sep="\t", quote=F,
row.names=T)

TA1p_table <- c[TvsA_igv,]
SA1p_table <- e[SvsA_igv,]
RS1p_table <- g[RicRvsS_igv,]
ER1p_table <- i[EndoTvsRicR_igv,]

TA_subanTable <- subanTable[TvsA_igv,]
SA_subanTable <- subanTable[SvsA_igv,]
RS_subanTable <- subanTable[RicRvsS_igv,]
ER_subanTable <- subanTable[EndoTvsRicR_igv,]

TA1p_table <- merge(TA1p_table, TA_subanTable, by.x="probeset_id")
SA1p_table <- merge(SA1p_table, SA_subanTable, by.x="probeset_id")
RS1p_table <- merge(RS1p_table, RS_subanTable, by.x="probeset_id")
ER1p_table <- merge(ER1p_table, ER_subanTable, by.x="probeset_id")

write.table(TA1p_table, "TvsA_IGV_DEtrack_nn.txt", sep="\t", quote=F,
row.names=T)
write.table(SA1p_table, "SvsA_IGV_DEtrack_nn.txt", sep="\t", quote=F,
row.names=T)
write.table(RS1p_table, "RicRvsS_IGV_DEtrack_nn.txt", sep="\t", quote=F,
row.names=T)
write.table(ER1p_table, "EndoTvsRicR_IGV_DEtrack_nn.txt", sep="\t", quote=F,
row.names=T)
```