iCheck: A package checking data quality of Illumina expression data

Weiliang Qiu†‡, Brandon Guo†‡, Christopher Anderson‡‡, Barbara Klanderman‡§, Vincent Carey‡¶, Benjamin Raby‡†

October 17, 2016

‡Channing Division of Network Medicine
Brigham and Women’s Hospital / Harvard Medical School
181 Longwood Avenue, Boston, MA, 02115, USA

Contents

1 Overview of iCheck 2
2 Exclude failed arrays 4
3 Check QC probes 5
4 Check squared correlations among genetic control (GC) arrays 6
5 Exclude GC arrays 8
6 Check squared correlations among replicated arrays 9
7 Obtain plot of estimated density for each array 10
8 Obtain plot of quantiles across arrays 11
9 Exclude gene probes with outlying expression levels 12
10 Obtain plot of the ratio ($p_{95}/p_{05}$) of 95-th percentile to 5-th percentile across arrays 13
11 Exclude arrays with $p_{95}/p_{05} \leq 6$ 14
12 Obtain Plot of principal components 15

*stwsq (at) channing.harvard.edu
†brandowonder (at) gmail.com
‡christopheranderson84 (a) gmail.com
§BKLANDERMAN (at) partners.org
¶stvjc (at) channing.harvard.edu
†rebar (at) channing.harvard.edu
Overview of iCheck

The iCheck package provides QC pipeline and data analysis tools for high-dimensional Illumina mRNA expression data. It provides several visualization tools to help identify gene probes with outlying expression levels, arrays with low quality, batches caused technical factors, batches caused by biological factors, and gender mis-match checking, etc.

We first generate a simulated data set to illustrate the usage of iCheck functions.

```r
> library(iCheck)
> if (!interactive()) {
+   options(rgl.useNULL = TRUE)
+ }
> # generate sample probe data
> set.seed(1234567)
> es.sim = genSimData.BayesNormal(nCpGs = 110,
+   nCases = 20, nControls = 20,
+   mu.n = -2, mu.c = 2,
+   d0 = 20, s02 = 0.64, s02.c = 1.5, testPara = "var",
+   outlierFlag = FALSE,
+   eps = 1.0e-3, applier = lapply)
> print(es.sim)
```

ExpressionSet (storageMode: lockedEnvironment)

assayData: 110 features, 40 samples
element names: exprs
protocolData: none
phenoData
  sampleNames: subj1 subj2 ... subj40 (40 total)
  varLabels: arrayID memSubj
  varMetadata: labelDescription
featureData
  featureNames: probe1 probe2 ... probe110 (110 total)
Annotation:

```r
# create replicates
> dat=exprs(es.sim)
> dat[,1]=dat[,2]
> dat[,3]=dat[,4]
> exprs(es.sim)=dat
> es.sim$arrayID=as.character(es.sim$arrayID)
> es.sim$arrayID[5:8]="Hela"
> # since simulated data set does not have 'Pass_Fail',
> # 'Tissue_Descr', 'Batch_Run_Date', 'Chip_Barcode',
> # 'Chip_Address', 'Hybridization_Name', 'Subject_ID', 'gender'
> # we generate them now to illustrate the R functions in the package
> es.sim$Hybridization_Name = paste(es.sim$arrayID, 1:ncol(es.sim), sep="_
")
> # assume the first 4 arrays are genetic control samples
> es.sim$Subject_ID = es.sim$arrayID
> es.sim$Pass_Fail = rep("pass", ncol(es.sim))
> # produce genetic control GC samples
> es.sim$Tissue_Descr = rep("CD4", ncol(es.sim))
> # assume the first 4 arrays are genetic control samples
> es.sim$Tissue_Descr[5:8]="Human Hela Cell"
> es.sim$Batch_Run_Date = 1:ncol(es.sim)
> es.sim$Chip_Barcode = 1:ncol(es.sim)
> es.sim$Chip_Address = 1:ncol(es.sim)
> es.sim$gender=rep(1, ncol(es.sim))
> set.seed(12345)
> pos=sample(x=1:ncol(es.sim), size=ceiling(ncol(es.sim)/2), replace=FALSE)
> es.sim$gender[pos]=0
> # generate sample probe data
> es.raw = es.sim[-c(1:10),]
> print(es.raw)
```

ExpressionSet (storageMode: lockedEnvironment)
assayData: 100 features, 40 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: subj1 subj2 ... subj40 (40 total)
  varLabels: arrayID memSubj ... gender (10 total)
  varMetadata: labelDescription
featureData
  featureNames: probe11 probe12 ... probe110 (100 total)
  fvarLabels: probe gene chr memGenes
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'

Annotation:

> # generate QC probe data
> es.QC = es.sim[c(1:10),]
> # since simulated data set does not have 'Reporter_Group_Name'
> # we created it now to illustrate the usage of 'plotQCCurves'.
> fDat=fData(es.QC)
> fDat$Reporter_Group_Name=rep("biotin", 10)
> fDat$Reporter_Group_Name[3:4]="cy3_hyb"
> fDat$Reporter_Group_Name[5:6]="housekeeping"
> fDat$Reporter_Group_Name[7:8]="low_stringency_hyb"
> fData(es.QC)=fDat
> print(es.QC)

ExpressionSet (storageMode: lockedEnvironment)
assayData: 10 features, 40 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: subj1 subj2 ... subj40 (40 total)
  varLabels: arrayID memSubj ... gender (10 total)
  varMetadata: labelDescription
featureData
  featureNames: probe1 probe2 ... probe10 (10 total)
  fvarLabels: probe gene ... Reporter_Group_Name (5 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:

>  

2 Exclude failed arrays

The meta data variable Pass_Fail indicates if an array is technically failed. We first should exclude these arrays.

We first check the values of the variable Pass_Fail:

> print(table(es.raw$Pass_Fail, useNA="ifany"))

pass
40

If there exist failed arrays, then we exclude them:

> pos<-which(es.raw$Pass_Fail != "pass")
> if(length(pos))
+ {
+  es.raw<-es.raw[, -pos]
+  es.QC<-es.QC[, -pos]
+ }
3 Check QC probes

The function `plotQCCurves` shows plot of quantiles across arrays for each type of QC probes. We expect the trajectories of quantiles across arrays are horizontal lines.

To get a better view, the arrays will be sorted based on variables specified in the function argument `varSort`.

```r
> plotQCCurves(
+ esQC=es.QC,
+ probes = c("biotin"), #"cy3_hyb", "housekeeping"),
+ #"low_stringency_hyb"),
+ labelVariable = "subjID",
+ hybName = "Hybridization_Name",
+ reporterGroupName = "Reporter_Group_Name",
+ requireLog2 = FALSE,
+ projectName = "test",
+ plotOutPutFlag = FALSE,
+ cex = 1,
+ ylim = NULL,
+ xlab = "",
+ ylab = "log2(intensity)",
+ lwd = 3,
+ mar = c(10, 4, 4, 2) + 0.1,
+ las = 2,
+ cex.axis = 1,
+ sortFlag = TRUE,
+ varSort = c("Batch_Run_Date", "Chip_Barcode", "Chip_Address"),
+ timeFormat = c("%m/%d/%Y", NA, NA)
+ )
```

probes>>
[1] "biotin"

********** k= 1 *******
QC probe= biotin
4 Check squared correlations among genetic control (GC) arrays

Next, we draw heatmap of the squared correlations among GC arrays. We expect the squared correlations among GC arrays are high (> 0.90).

The function argument `labelVariable` indicates which meta variable will be used to label the arrays in the heatmap.

If we draw heatmap for replicated arrays, we can set the function arguments `sortFlag=TRUE`,

```r
varSort=c("Subject_ID", "Hybridization_Name", "Batch_Run_Date", "Chip_Barcode", "Chip_Address")
```

and

```r
timeFormat=c(NA, NA, "%m/%d/%Y", NA, NA)
```

so that arrays from the same subjects will be grouped together in the heatmap.

Note that although the meta variable `Batch_Run_Date` records time, it is vector of string character in R. The function `R2PlotFunc` will automatically
convert it to time variable if we set the value of the argument \texttt{timeFormat} corresponding to the variable \texttt{Batch_Run_Date} as a time format like \texttt{"%m/%d/%Y"}. Details about the time format, please see the R function \texttt{strptime}.

The followings show example R code to draw heatmap of GC arrays.

\begin{verbatim}
> R2PlotFunc(
+    es=es.raw,
+    hybName = "Hybridization_Name",
+    arrayType = "GC",
+    GCid = c("128115", "Hela", "Brain"),
+    probs = seq(0, 1, 0.25),
+    col = gplots::greenred(75),
+    labelVariable = "subjID",
+    outFileName = "test_R2_raw.pdf",
+    title = "Raw Data R^2 Plot",
+    requireLog2 = FALSE,
+    plotOutPutFlag = FALSE,
+    las = 2,
+    keysize = 1,
+    margins = c(10, 10),
+    sortFlag = TRUE,
+    varSort=c("Batch_Run_Date", "Chip_Barcode", "Chip_Address"),
+    timeFormat=c("%m/%d/%Y", NA, NA)
+  )
\end{verbatim}

quantile of R^2
\begin{verbatim}
0%  25%  50%  75%  100%
4.807015e-06 1.015917e-03 1.675295e-03 4.155686e-03 6.798879e-03
\end{verbatim}
5 Exclude GC arrays

We next exclude GC arrays and will focus on sample arrays to check data quality.

```r
> print(table(es.raw$Tissue_Descr, useNA="ifany"))

       CD4 Human Hela Cell
 36         4

> # for different data sets, the label for GC arrays might
> # be different.
> pos.del<-which(es.raw$Tissue_Descr == "Human Hela Cell")
> cat("No. of GC arrays=", length(pos.del), "\n")

No. of GC arrays= 4

> if(length(pos.del))
+ {
+   es.raw<--es.raw[-pos.del]
+   es.QC<--es.QC[-pos.del]
```
6 Check squared correlations among replicated arrays

Check squared correlations among replicated arrays (excluding GC arrays). We expect within subject correlations will be high.

```r
R2PlotFunc(
  es=es.raw,
  arrayType = c("replicates"),
  GCid = c("128115", "Hela", "Brain"),
  probs = seq(0, 1, 0.25),
  col = gplots::greenred(75),
  labelVariable = "subjID",
  outFileFileName = "test_R2_raw.pdf",
  title = "Raw Data $R^2$ Plot",
  requireLog2 = FALSE,
  plotOutPutFlag = FALSE,
  las = 2,
  keysizie = 1,
  margins = c(10, 10),
  sortFlag = TRUE,
  varSort=c("Subject_ID", "Hybridization_Name", "Batch_Run_Date", "Chip_Barcode",
    timeFormat=c(NA, NA, "%m/%d/%Y", NA, NA))
)
```

quantile of $R^2$>>

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.007151179</td>
<td>0.007151179</td>
<td>0.007151179</td>
<td>0.751787795</td>
<td>1.000000000</td>
</tr>
</tbody>
</table>

quantile of within-replicate $R^2$>>

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

>
7 Obtain plot of estimated density for each array

We next draw plot of estimated density for each array. We expect the estimated densities of all arrays to be similar. However, for real data, some patterns of the estimated densities might appear indicating the existence of some batch effects.

Note that by default, the function argument `requireLog2 = TRUE`. Since the distributions of simulated data are from normal distribution, we don’t need to do log2 transformation here.

```r
> densityPlots(
+   es = es.raw,
+   requireLog2 = FALSE,
+   myxlab = "expression level",
+   datExtrFunc = exprs,
+   fileFlag = FALSE,
+   fileFormat = "ps",
+   fileName = "densityPlots_sim.ps")
```
8 Obtain plot of quantiles across arrays

We next draw plot of quantiles across sample arrays. We expect the trajectories of quantiles be horizontal. However, for real data, some patterns of the trajectories might appear indicating the existence of some batch effects.

Some times, the quantile plots can show that some probes have some outlying expression levels. In this case, we can delete those gene probes.

Note that by default, the function argument `requireLog2 = TRUE`. Hence, we need to take log2 transformation to identify which gene probes containing outlying expression levels.

By default, we will sort the arrays by the ascending order of the median absolute deviation (MAD) to have a better view of the trajectories of quantiles.

```r
> quantilePlot(
+   dat=exprs(es.raw),
+   fileName,
+   probs = c(0, 0.05, 0.25, 0.5, 0.75, 0.95, 1),
+   plotOutPutFlag = FALSE,
+   requireLog2 = FALSE,
```
+ sortFlag = TRUE,
+ cex = 1,
+ ylim = NULL,
+ xlab = "",
+ ylab = "log2(intensity)",
+ lwd = 3,
+ main = "Trajectory plot of quantiles\n(sample arrays)",
+ mar = c(15, 4, 4, 2) + 0.1,
+ las = 2,
+ cex.axis = 1
+ )

***** Arrays were sorted by MAD (median absolute deviation)!

9 Exclude gene probes with outlying expression levels

if quantile plots show some outlying expression levels, we can use the following R code to identify the gene probes with outlying expression levels.
> # note we need to take log2 transformation
> # if requireLog2 = TRUE.
> requireLog2 = FALSE
> if(requireLog2)
> + {
> + minVec<-apply(log2(exprs(es.raw)), 1, min, na.rm=TRUE)
> + # suppose the cutoff is 0.5
> + print(sum(minVec< 0.5))
> + pos.del<-which(minVec<0.5)
> +
> + cat("Number of gene probes with outlying expression levels>>",
> + length(pos.del), ",\n")
> + if(length(pos.del))
> + {
> + es.raw<-es.raw[-pos.del,]
> + }
> + }
>

10 Obtain plot of the ratio ($p_{95}/p_{05}$) of 95-th percentile to 5-th percentile across arrays

We next draw the plot of the ratio of $p_{95}$ over $p_{05}$ across arrays, where $p_{95}$ ($p_{05}$) is the 95-th (5-th) percentile of a array. If an array with the ratio $p_{95}/p_{05}$ is less than 6, then we regard this array as a bad array and should delete it before further analysis.

Note that we should set requireLog2 = FALSE.

````
> plotSamplep95p05(
> + es=es.raw,
> + labelVariable = "memSubj",
> + requireLog2 = FALSE,
> + projectName = "test",
> + plotOutPutFlag = FALSE,
> + cex = 1,
> + ylim = NULL,
> + xlab = "",
> + ylab = "",
> + lwd = 1.5,
> + mar = c(10, 4, 4, 2) + 0.1,
> + las = 2,
> + cex.axis=1.5,
> + title = "Trajectory of p95/p05",
> + cex.legend = 1.5,
> + cex.lab = 1.5,
> + legendPosition = "topright",
> + cut1 = 10,
> + cut2 = 6,
> + sortFlag = TRUE,
```
11 Exclude arrays with $p_{95}/p_{05} \leq 6$

If there exist arrays with $p_{95}/p_{05} < 6$, we then need to exclude these arrays from further data analysis. The followings are example R code:

```r
> p95 <- quantile(exprs(es.raw), prob=0.95)
> p05 <- quantile(exprs(es.raw), prob=0.05)
> r <- p95/p05
> pos.del <- which(r<6)
> print(pos.del)

95%
  1

> if(length(pos.del))
+ {
```
12 Obtain Plot of principal components

We next draw pca plots to double check batch effects or treatment effects indicated by dendrogram.

The first step is to obtain principal components using the function `getPCAFunc`. For large data set, this function might be very slow.

```r
> pcaObj <- getPCAFunc(es = es.raw,
+   labelVariable = "subjID",
+   requireLog2 = FALSE,
+   corFlag = FALSE
+ )
>
> pca2DPlot(pcaObj = pcaObj,
+   plot.dim = c(1,2),
+   labelVariable = "memSubj",
+   outFileName = "test_pca_raw.pdf",
+   title = "Scatter plot of pcas (memSubj)",
+   plotOutPutFlag = FALSE,
+   mar = c(5, 4, 4, 2) + 0.1,
+   lwd = 1.5,
+   equalRange = TRUE,
+   xlab = NULL,
+   ylab = NULL,
+   xlim = NULL,
+   ylim = NULL,
+   cex.legend = 1.5,
+   cex = 1.5,
+   cex.lab = 1.5,
+   cex.axis = 1.5,
+   legendPosition = "topright"
+ )
```
13 Perform background correction, data transformation and normalization

```r
> tt <- es.raw
> es.q <- lumiN(tt, method = "quantile")
```

Perform quantile normalization ...

14 Obtain Plot of principal components for pre-processed data

After pre-processing data, we do principal component analysis again. Note that we should set `requireLog2 = FALSE`.

```r
> pcaObj <- getPCAFunc(es = es.q,
+ labelVariable = "subjID",
+ requireLog2 = FALSE,
+ corFlag = FALSE
```
pca2DPlot(pcaObj=pcaObj,
    plot.dim = c(1,2),
    labelVariable = "memSubj",
    outFileName = "test_pca_raw.pdf",
    title = "Scatter plot of pcas (memSubj)\n(log2 transformed and quantile normalized)",
    plotOutPutFlag = FALSE,
    mar = c(5, 4, 4, 2) + 0.1,
    lwd = 1.5,
    equalRange = TRUE,
    xlab = NULL,
    ylab = NULL,
    xlim = NULL,
    ylim = NULL,
    cex.legend = 1.5,
    cex = 1.5,
    cex.lab = 1.5,
    cex.axis = 1.5,
    legendPosition = "topright"
)
15 Incorporate phenotype data

In addition meta data, we usually have phenotype data to describe subjects. We can now add them in.

16 Data analysis

16.1 lmFitWrapper and lmFitPaired

iCheck provide 2 limma wrapper functions \texttt{lmFitPaired} (for paired data) and \texttt{lmFitWrapper} (for unpaired data).

Note that the function argument \texttt{pos.var.interest = 1} requests the results (test statistic and p-value) for the first covariate will be print out.

If \texttt{pos.var.interest = 0}, then the results (test statistic and p-value) for the intercept will be print out.

The outcome variable must be gene probes. Can not be phenotype variables.

\begin{verbatim}
> res.limma=lmFitWrapper(
+   es=es.q,
\end{verbatim}
formula="as.factor(memSubj),
pos.var.interest = 1,
pvalAdjMethod="fdr",
alpha=0.05,
probeID.var="probe",
gene.var="gene",
chr.var="chr",
verbose=TRUE)

dim(dat)>
[1] 100 35

Running lmFit...
Running eBayes...
Preparing output...

<table>
<thead>
<tr>
<th>probeIDs</th>
<th>geneSymbols</th>
<th>chr</th>
<th>stats</th>
<th>pval</th>
<th>p.adj</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>probe29</td>
<td>1</td>
<td>-3.376879</td>
<td>0.001401733</td>
<td>0.1005857</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>probe16</td>
<td>1</td>
<td>3.254099</td>
<td>0.002011714</td>
<td>0.1005857</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>probe92</td>
<td>1</td>
<td>2.688895</td>
<td>0.009634725</td>
<td>0.3211575</td>
<td>82</td>
</tr>
<tr>
<td>4</td>
<td>probe59</td>
<td>1</td>
<td>-2.210193</td>
<td>0.031558408</td>
<td>0.4968483</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>probe32</td>
<td>1</td>
<td>2.181275</td>
<td>0.033750422</td>
<td>0.4968483</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>probe17</td>
<td>1</td>
<td>-2.143605</td>
<td>0.036806038</td>
<td>0.4968483</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>probe103</td>
<td>1</td>
<td>-2.117636</td>
<td>0.039051322</td>
<td>0.4968483</td>
<td>93</td>
</tr>
<tr>
<td>8</td>
<td>probe35</td>
<td>1</td>
<td>-2.109845</td>
<td>0.039747866</td>
<td>0.4968483</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>probe54</td>
<td>1</td>
<td>1.712229</td>
<td>0.092867773</td>
<td>0.8375408</td>
<td>44</td>
</tr>
<tr>
<td>10</td>
<td>probe74</td>
<td>1</td>
<td>-1.703176</td>
<td>0.094560774</td>
<td>0.8375408</td>
<td>64</td>
</tr>
<tr>
<td>11</td>
<td>probe40</td>
<td>1</td>
<td>1.638282</td>
<td>0.107460992</td>
<td>0.8375408</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>probe61</td>
<td>1</td>
<td>-1.604613</td>
<td>0.114691882</td>
<td>0.8375408</td>
<td>51</td>
</tr>
<tr>
<td>13</td>
<td>probe12</td>
<td>1</td>
<td>1.527377</td>
<td>0.132783363</td>
<td>0.8375408</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>probe56</td>
<td>1</td>
<td>-1.521438</td>
<td>0.134263881</td>
<td>0.8375408</td>
<td>46</td>
</tr>
<tr>
<td>15</td>
<td>probe90</td>
<td>1</td>
<td>-1.445861</td>
<td>0.154271925</td>
<td>0.8375408</td>
<td>80</td>
</tr>
<tr>
<td>16</td>
<td>probe79</td>
<td>1</td>
<td>-1.406303</td>
<td>0.165637004</td>
<td>0.8375408</td>
<td>69</td>
</tr>
<tr>
<td>17</td>
<td>probe89</td>
<td>1</td>
<td>-1.367624</td>
<td>0.177366059</td>
<td>0.8375408</td>
<td>79</td>
</tr>
<tr>
<td>18</td>
<td>probe38</td>
<td>1</td>
<td>1.360555</td>
<td>0.179576911</td>
<td>0.8375408</td>
<td>28</td>
</tr>
<tr>
<td>19</td>
<td>probe96</td>
<td>1</td>
<td>-1.351534</td>
<td>0.182428796</td>
<td>0.8375408</td>
<td>86</td>
</tr>
<tr>
<td>20</td>
<td>probe23</td>
<td>1</td>
<td>1.347172</td>
<td>0.183820114</td>
<td>0.8375408</td>
<td>13</td>
</tr>
</tbody>
</table>

pvalue quantiles for intercept and covariates>>

(Intercept) as.factor(memSubj)

min 0.0002637651 0.001401733
25% 0.0356690890 0.242102862
median 0.1922335675 0.455084256
75% 0.4874162700 0.780677827
max 0.9978539142 0.998200667

formula>>

"as.factor(memSubj)"

covariate of interest is as.factor(memSubj)

Number of tests= 100
Number of arrays = 35
Number of significant tests (raw p-value < 0.05) = 8
Number of significant tests after p-value adjustments = 0

="/**************
No genes are differentially expressed!
>

16.2 glmWrapper

outcome variable can be phenotype variables. The function argument family indicates if logistic regression (family=binomial) used or general linear regression (family=gaussian) used.

> res.glm=glmWrapper(
+ es=es.q,
+ formula = xi~as.factor(memSubj),
+ pos.var.interest = 1,
+ family=gaussian,
+ logit=FALSE,
+ pvalAdjMethod="fdr",
+ alpha = 0.05,
+ probeId.var = "probe",
+ gene.var = "gene",
+ chr.var = "chr",
+ applier=lapply,
+ verbose=TRUE
+ )

probeIDs geneSymbols chr stats coef pval p.adj pos
1 probe16 gene16 1 3.305320 1.0593407 0.002293003 0.1763387 6
2 probe29 gene29 1 -3.142649 -1.3263843 0.003526774 0.1763387 19
3 probe92 gene92 1 2.508446 1.0489929 0.017219384 0.5739795 82
4 probe59 gene59 1 -2.238212 -0.7231024 0.032072229 0.6917895 49
5 probe35 gene35 1 -2.121164 -0.7059218 0.041507370 0.6917895 93
6 probe103 gene103 1 -2.11164 -0.7059218 0.041507370 0.6917895 93
7 probe17 gene17 1 -2.029732 -0.8040525 0.050510402 0.7215772 7
8 probe32 gene32 1 1.915186 1.0631079 0.064192277 0.8021153 22
9 probe74 gene74 1 -1.786253 -0.5282879 0.083249917 0.8072836 64
10 probe61 gene61 1 -1.725751 -0.4815208 0.093746106 0.8072836 51
11 probe54 gene54 1 1.633937 0.6301915 0.11777893 0.8072836 44
12 probe40 gene40 1 1.609611 0.5661679 0.117008899 0.8072836 30
13 probe90 gene90 1 -1.605115 -0.4182303 0.117997257 0.8072836 80
14 probe38 gene38 1 1.542007 0.3851935 0.13267854 0.8072836 28
15 probe56 gene56 1 -1.464652 -0.5487800 0.152478453 0.8072836 46
16 probe99 gene99 1 1.405098 0.3951053 0.169336626 0.8072836 89
17 probe108 gene108 1 -1.380558 -0.3907145 0.176695014 0.8072836 98
18 probe79 gene79 1 -1.339224 -0.5201586 0.189649429 0.8072836 69
19 probe12 gene12 1 1.333659 0.7656150 0.191448222 0.8072836 2
pvalue quantiles for intercept and covariates

<table>
<thead>
<tr>
<th></th>
<th>pval.(Intercept)</th>
<th>pval.as.factor(memSubj)</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>0.0006238292</td>
<td>0.002293003</td>
</tr>
<tr>
<td>25%</td>
<td>0.0245832240</td>
<td>0.237742030</td>
</tr>
<tr>
<td>median</td>
<td>0.1952130722</td>
<td>0.444338213</td>
</tr>
<tr>
<td>75%</td>
<td>0.5048805444</td>
<td>0.784209263</td>
</tr>
<tr>
<td>max</td>
<td>0.9980082033</td>
<td>0.998167350</td>
</tr>
</tbody>
</table>

covariate of interest is as.factor(memSubj)

Number of tests = 100
Number of arrays = 35
Number of significant tests (raw p-value < 0.05) = 6
Number of significant tests after p-value adjustments = 0

No genes are differentially expressed!

16.3 lkhWrapper

Likelihood ratio test wrapper. Compare 2 glm models. One is reduced model. The other is full model.

```
> res.lkh=lkhWrapper(
  + es=es.q,
  + formulaReduced = xi~as.factor(memSubj),
  + formulaFull = xi~as.factor(memSubj)+gender,
  + family=gaussian,
  + logit=FALSE,
  + pvalAdjMethod="fdr",
  + alpha = 0.05,
  + probeID.var = "probe",
  + gene.var = "gene",
  + chr.var = "chr",
  + applier=lapply,
  + verbose=TRUE
  + )

Top 20 tests:

<table>
<thead>
<tr>
<th>probeIDs</th>
<th>geneSymbols</th>
<th>chr</th>
<th>Chisq</th>
<th>Df</th>
<th>pval</th>
<th>p.adj</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
<td>probe87</td>
<td>1</td>
<td>9.078788</td>
<td>1</td>
<td>0.002585913</td>
<td>0.2585913</td>
<td>77</td>
</tr>
<tr>
<td>95</td>
<td>probe105</td>
<td>1</td>
<td>5.535945</td>
<td>1</td>
<td>0.018629704</td>
<td>0.4503399</td>
<td>95</td>
</tr>
</tbody>
</table>
Once we get analysis results, we need to check if the results are reasonable or not (e.g., were results affected by outliers?).

If the phenotype variable of interest is a binary type variable, then we can draw parallel boxplots of expression level versus the phenotype for each of top results. iCheck provides function boxPlots to do such a task.

> boxPlots(
  + resFrame = res.limma$frame,
  + es = es.sim,
  + col.resFrame = c("probeIDs", "stats", "pval", "p.adj"),
  + var.pheno = "memSubj",

17 Result Visualization

Number of tests >>> 100

Number of arrays >>> 35

Number of tests with pvalue < 0.05 >>> 9

Number of tests with FDR adjusted pvalue < 0.05 >>> 0

>
If the phenotype variable of interest is a continuous type variable, then we can draw scatter plot of expression level versus the phenotype for each of top results. iCheck provides function `scatterPlots` to do such a task.

```
> # regard memSubj as continuous for illustration purpose
> scatterPlots(
+   resFrame = res.limma$frame,
+   es = es.sim,
+   col.resFrame = c("probeIDs", "stats", "pval", "p.adj"),
+   var.pheno = "memSubj",
+   var.probe = "probe",
```
18 Session Info

Finally, we need to print out the session info so that later we can know which versions the packages are from.

> toLatex(sessionInfo())

- R version 3.3.1 (2016-06-21), x86_64-pc-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,
Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils

Other packages: Biobase 2.34.0, BiocGenerics 0.20.0, gplots 3.0.1, iCheck 1.4.0, lumi 2.26.0

Loaded via a namespace (and not attached): AnnotationDbi 1.36.0, BioCInstaller 1.24.0, BiocParallel 1.8.0, Biostrings 2.42.0, DBI 0.5-1, GEOquery 2.40.0, GeneSelectMMD 2.18.0, GenomeInfoDb 1.10.0, GenomicAlignments 1.10.0, GenomicFeatures 2.16.0, GenomicRanges 1.26.0, MASS 7.3-45, Matrix 1.2-7.1, R6 2.2.0, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.0.0, Rsamtools 1.26.0, S4Vectors 0.12.0, SummarizedExperiment 1.4.0, XML 3.98-1.4, XVector 0.14.0, affy 1.52.0, affyio 1.44.0, annotate 1.52.0, base64 2.0, beanplot 1.2, biomaRt 2.30.0, bitops 1.0-6, bumphunter 1.14.0, caTools 1.17.1, chron 2.3-47, codetools 0.2-15, colorspace 1.2-7, data.table 1.9.6, digest 0.6.10, doRNG 1.6, foreach 1.4.3, gdata 2.17.0, genefilter 1.56.0, grid 3.3.1, gtools 3.5.0, htmltools 0.3.5, htmlwidgets 0.7, httr 1.2.1, illuminaio 0.16.0, iterators 1.0.8, jsonlite 1.1, knitr 1.14, lattice 0.20-34, limma 3.30.0, locfit 1.5-9.1, magrittr 1.5, matrixStats 0.51.0, mclust 5.2, methylumi 2.20.0, mgcv 1.8-15, mime 0.5, miniBi 1.20.0, multtest 2.30.0, nleqslv 3.0.3, nlme 3.1-128, nor1mix 1.2-2, openssl 0.9.4, pkgmaker 0.22, pply 1.8.4, preprocessCore 1.36.0, quadprog 1.5-5, randomForest 4.6-12, registry 0.3, reshape 0.8.5, rgl 0.96.0, rngtools 1.2.4, rtracklayer 1.34.0, scatterplot3d 0.3-37, shiny 0.14.1, siggenes 1.48.0, splines 3.3.1, stats 3.3.1, stringi 1.1.2, stringr 1.1.0, survival 2.39-5, tools 3.3.1, xtable 1.8-2, zlibbioc 1.20.0, zoo 1.7-13