iCheck: A package checking data quality of Illumina expression data

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

October 14, 2015

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

Contents

[*]stwxq (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

1 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.

```
>   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)
  fvarLabels: probe gene chr memGenes
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

```
>   # 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$arrayID[1]=es.sim$arrayID[2]
>   es.sim$arrayID[3]=es.sim$arrayID[4]
>   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

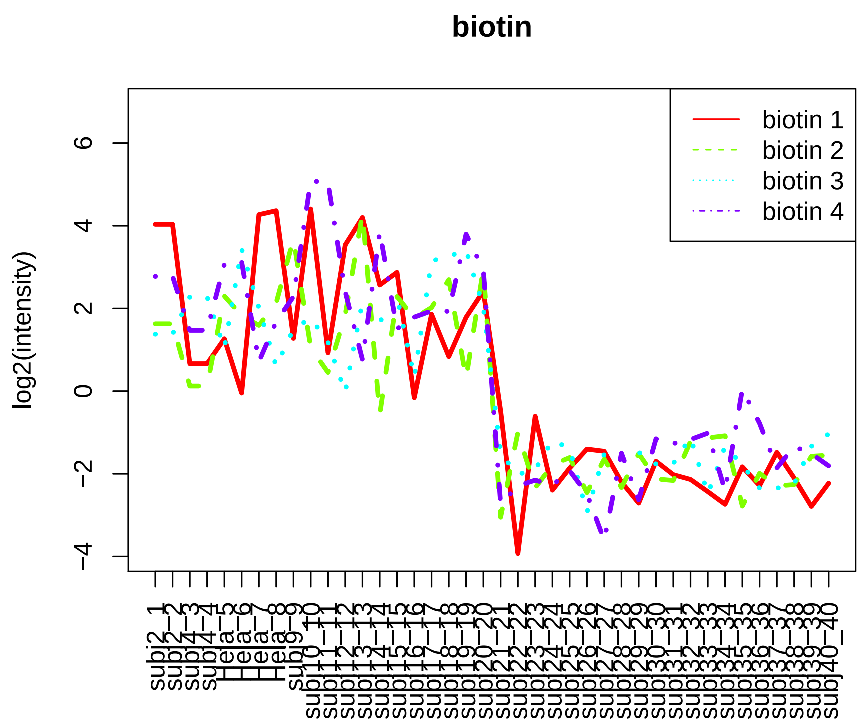
The function `plotQCCurves` shows plot of quantiles across arrays for each type of QC probes. We expect the trajectories of quantiles across arrays are horizontal lines.

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

```
>       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`,

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

and

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

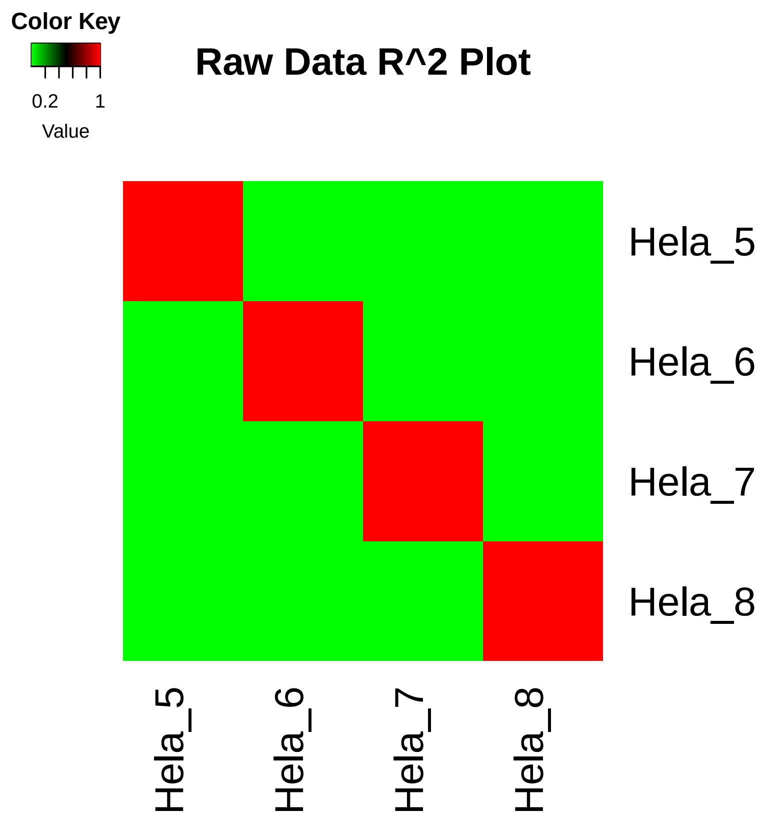
so that arrays from the same subjects will be grouped together in the heatmap.

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 `timeFormat` corresponding to the variable `Batch_Run_Date` as a time format like `"%m/%d/%Y"`. Details about the time format, please see the R function `strptime`.

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

```
>       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)
+         )

quantile of R^2>>
          0%          25%          50%          75%         100%
4.807015e-06 1.015917e-03 1.675295e-03 4.155686e-03 6.798879e-03
```



5 Exclude GC arrays

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

```
> 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]
+   print(dims(es.raw))
+   print(dims(es.QC))
+ }

         exprs
Features   100
Samples     36
         exprs
Features    10
Samples     36
```

6 Check squared correlations among replicated arrays

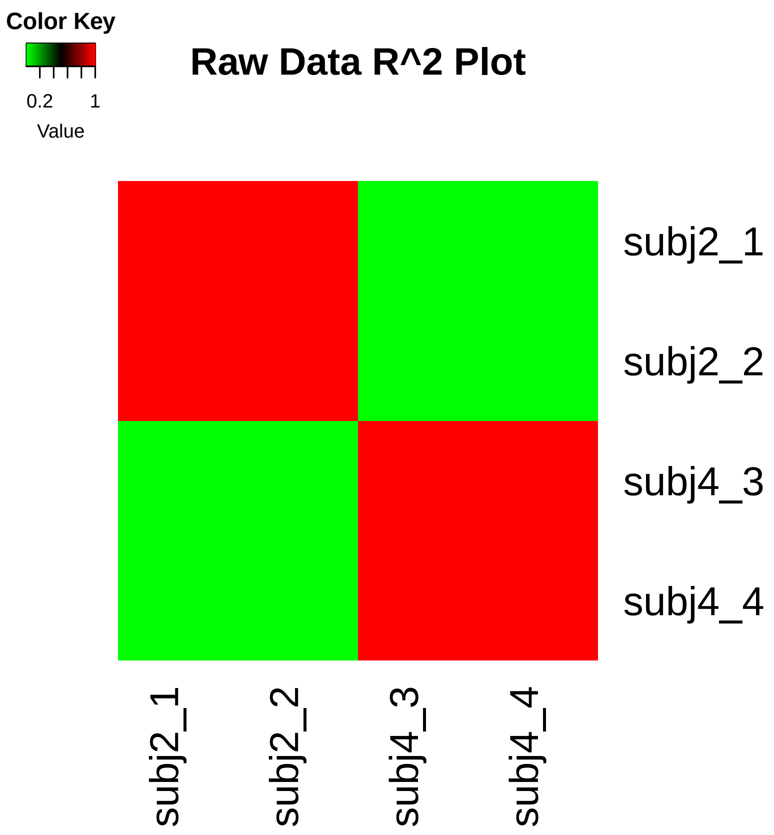
Check squared correlations among replicated arrays (excluding GC arrays). We expect within subject correlations will be high.

```
>      R2PlotFunc(
+          es=es.raw,
+          arrayType = c("replicates"),
+          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("Subject_ID", "Hybridization_Name", "Batch_Run_Date", "Chip_Barcode",
+          timeFormat=c(NA, NA, "%m/%d/%Y", NA, NA)
+        )

quantile of R^2>>
         0%         25%         50%         75%        100%
0.007151179 0.007151179 0.007151179 0.751787795 1.000000000

quantile of within-replicate R^2>>
  0%  25%  50%  75% 100%
   1    1    1    1    1

>
```



7 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.

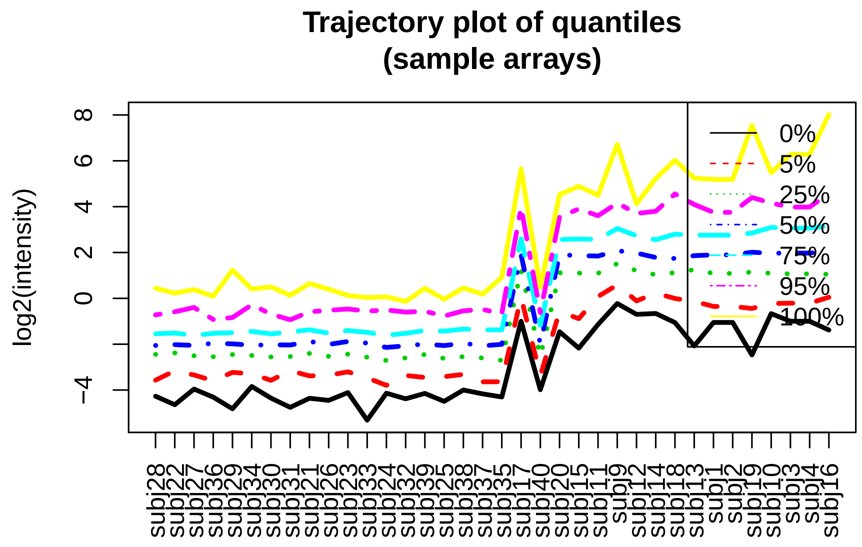
Some times, the quantile plots can show that some probes have some outlying expression levels. In this case, we can delete those gene probes.

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.

```
>       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)!
```



8 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.

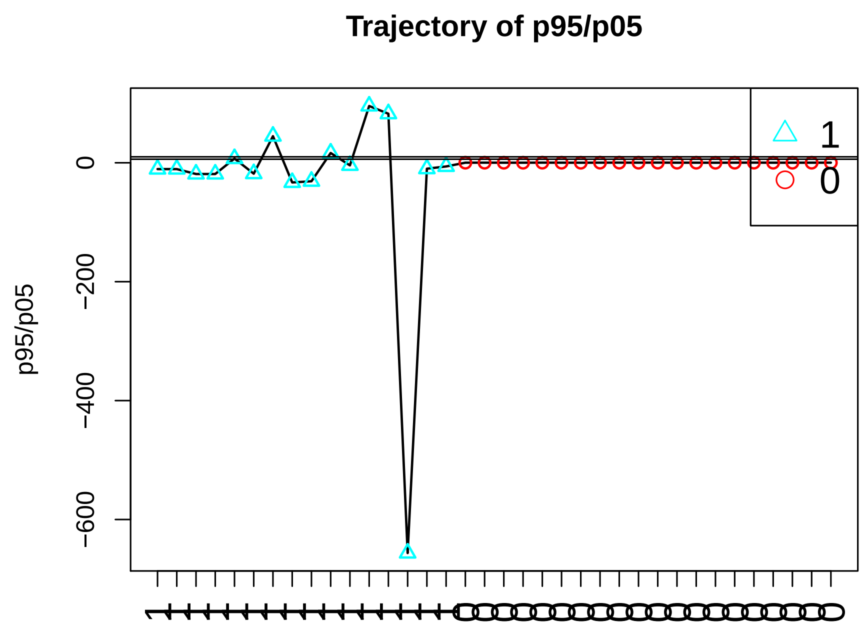
```
> # note we need to take log2 transformation
> # if requireLog2 = TRUE.
> requireLog2 = FALSE
> if(requireLog2)
+ {
+   minVec<-apply(log2(exprs(es.raw)), 1, min, na.rm=TRUE)
+   # suppose the cutoff is 0.5
+   print(sum(minVec< 0.5))
+   pos.del<-which(minVec<0.5)
+
+   cat("Number of gene probes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

9 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 p95 over p05 across arrays, where p95 (p05) is the 95-th (5-th) percentile of a array. If an array with the ratio $p95/p05$ 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,
+             varSort = c("Batch_Run_Date", "Chip_Barcode", "Chip_Address"),
+             timeFormat = c("%m/%d/%Y", NA, NA),
+             verbose = FALSE)
```



10 Exclude arrays with $p_{95}/p_{05} \leq 6$

If there exist arrays with $p95/p05 < 6$, we then need to exclude these arrays from further data analysis. The followings are example R code:

```
> 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))
+ {
```

```
+   es.raw<-es.raw[,-pos.del]
+   es.QC<-es.QC[,-pos.del]
+ }
>
```

11 Obtain Plot of principal components

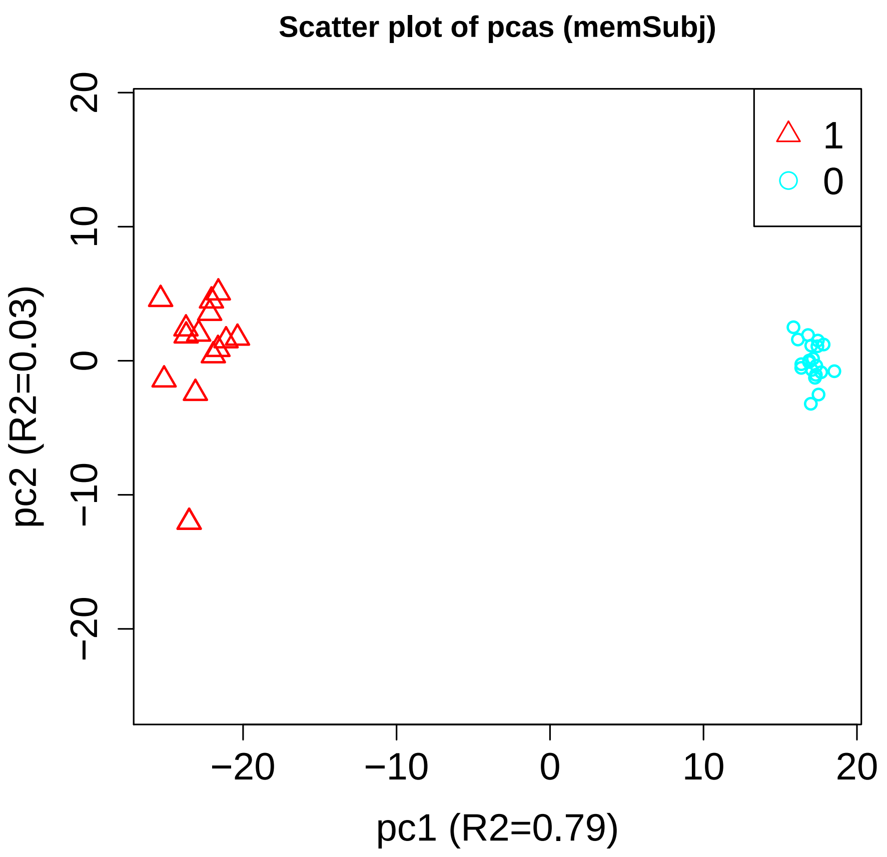
We next draw pca plots to double check batch effects or treatment effects indicated by dendrogram.

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

```
>      pcaObj<-getPCAFunc(es=es.raw,
+                 labelVariable = "subjID",
+                 requireLog2 = FALSE,
+                 corFlag = FALSE
+
+
+      )
>
```

We then plot the first 2 or 3 principal components and label the data points by meta variables of interests, such as tissue type, study center, batch id, etc..

```
>      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"
+
+               )
```



12 Perform background correction, data transformation and normalization

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

Perform quantile normalization ...
```

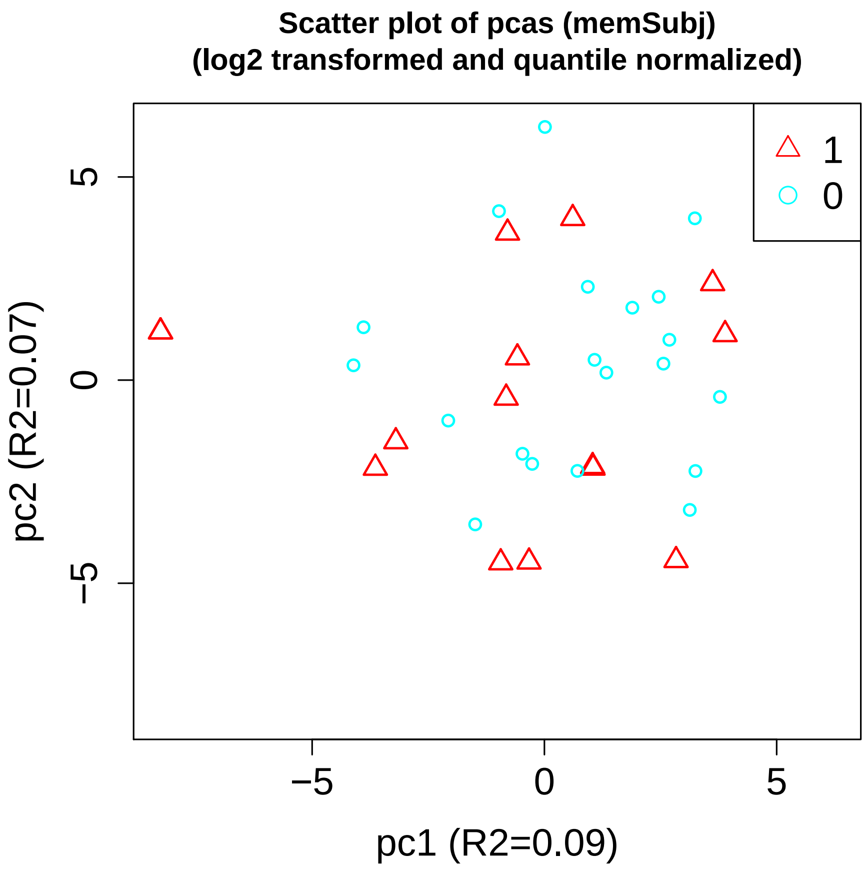
13 Obtain Plot of principal components for pre-processed data

After pre-processing data, we do principal component analysis again.

Note that we should set `requireLog2 = FALSE`.

```
>       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 n
+                 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"
+                 )
>
```



14 Incorporate phenotype data

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

15 Data analysis

15.1 lmFitWrapper and lmFitPaired

`iCheck` provide 2 limma wrapper functions `lmFitPaired` (for paired data) and `lmFitWrapper` (for unpaired data).

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

If `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.

```
> res.limma=lmFitWrapper(
+   es=es.q,
+   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...
   probeIDs geneSymbols chr     stats        pval     p.adj pos
1   probe29      gene29   1 -3.376879 0.001401733 0.1005857  19
2   probe16      gene16   1  3.254099 0.002011714 0.1005857   6
3   probe92      gene92   1  2.688895 0.009634725 0.3211575  82
4   probe59      gene59   1 -2.210193 0.031558408 0.4968483  49
5   probe32      gene32   1  2.181275 0.033750422 0.4968483  22
6   probe17      gene17   1 -2.143605 0.036806038 0.4968483   7
7  probe103     gene103   1 -2.117636 0.039051322 0.4968483  93
8   probe35      gene35   1 -2.109845 0.039747866 0.4968483  25
9   probe54      gene54   1  1.712229 0.092867773 0.8375408  44
10  probe74      gene74   1 -1.703176 0.094560774 0.8375408  64
11  probe40      gene40   1  1.638282 0.107456092 0.8375408  30
12  probe61      gene61   1 -1.604613 0.114691882 0.8375408  51
13  probe12      gene12   1  1.527377 0.132783363 0.8375408   2
14  probe56      gene56   1 -1.521438 0.134263881 0.8375408  46
15  probe90      gene90   1 -1.445861 0.154271925 0.8375408  80
16  probe79      gene79   1 -1.406303 0.165637004 0.8375408  69
17  probe89      gene89   1 -1.367624 0.177366059 0.8375408  79
18  probe38      gene38   1  1.360555 0.179576911 0.8375408  28
19  probe96      gene96   1 -1.351534 0.182428796 0.8375408  86
20  probe23      gene23   1  1.347172 0.183820114 0.8375408  13

pvalue quantiles for intercept and covariates>>
        (Intercept) as.factor(memSubj)1
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!

>
```

15.2 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 -2.173225 -0.6718451 0.037042472 0.6917895  25
6  probe103     gene103   1 -2.121164 -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 0.064169227 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 0.111777893 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 0.132607854 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
```

```
20  probe44      gene44   1  1.304127  0.3796869 0.201213900 0.8072836  34

pvalue quantiles for intercept and covariates>>
       pval.(Intercept) pval.as.factor(memSubj)1
min        0.0006238292             0.002293003
25%        0.0245832240             0.237742030
median     0.1952130722             0.444338213
75%        0.5048805444             0.784209263
max        0.9980082033             0.998167350

formula>>
xi ~ 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 )= 6
Number of significant tests after p-value adjustments= 0


**********************************************
No genes are differentially expressed!

>
```

15.3 lkhrWrapper

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

```
> res.lkh=lkhrWrapper(
+   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>>>
   probeIDs geneSymbols chr    Chisq Df        pval     p.adj pos
77  probe87      gene87   1 9.078788  1 0.002585913 0.2585913  77
95  probe105     gene105   1 5.535945  1 0.018629704 0.4503399  95
```

```
70   probe80      gene80   1 5.413218  1 0.019984838 0.4503399  70
41   probe51      gene51   1 5.142924  1 0.023341297 0.4503399  41
92  probe102     gene102   1 4.867929  1 0.027360310 0.4503399  92
15   probe25      gene25   1 4.707136  1 0.030037649 0.4503399  15
22   probe32      gene32   1 4.624209  1 0.031523793 0.4503399  22
72   probe82      gene82   1 4.118205  1 0.042424066 0.4794337  72
54   probe64      gene64   1 4.089554  1 0.043149037 0.4794337  54
90  probe100     gene100   1 3.809597  1 0.050959724 0.5095972  90
100 probe110     gene110   1 3.553968  1 0.059403229 0.5400294 100
6    probe16      gene16   1 2.740925  1 0.097808137 0.7784161   6
17   probe27      gene27   1 2.553449  1 0.110053739 0.7784161  17
48   probe58      gene58   1 2.503260  1 0.113610876 0.7784161  48
69   probe79      gene79   1 2.460220  1 0.116762420 0.7784161  69
38   probe48      gene48   1 2.176369  1 0.140144757 0.8152063  38
34   probe44      gene44   1 2.035661  1 0.153647333 0.8152063  34
58   probe68      gene68   1 2.018252  1 0.155417926 0.8152063  58
68   probe78      gene78   1 1.968644  1 0.160591920 0.8152063  68
81   probe91      gene91   1 1.924669  1 0.165342887 0.8152063  81


formulaReduced>>
xi ~ as.factor(memSubj)

formulaFull>>
xi ~ as.factor(memSubj) + gender


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

>
```

16 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 3.2.2 (2015-08-14), `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`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`

- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: Biobase 2.30.0, BiocGenerics 0.16.0, gplots 2.17.0, iCheck 1.0.0, lumi 2.22.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.32.0, BiocInstaller 1.20.0, BiocParallel 1.4.0, Biostrings 2.38.0, DBI 0.3.1, GEOquery 2.36.0, GeneSelectMMD 2.14.0, GenomeInfoDb 1.6.0, GenomicAlignments 1.6.0, GenomicFeatures 1.22.0, GenomicRanges 1.22.0, IRanges 2.4.0, KernSmooth 2.23-15, MASS 7.3-44, Matrix 1.2-2, RColorBrewer 1.1-2, RCurl 1.95-4.7, RSQLite 1.0.0, Rcpp 0.12.1, Rsamtools 1.22.0, S4Vectors 0.8.0, SummarizedExperiment 1.0.0, XML 3.98-1.3, XVector 0.10.0, affy 1.48.0, affyio 1.40.0, annotate 1.48.0, base64 1.1, beanplot 1.2, biomaRt 2.26.0, bitops 1.0-6, bumphunter 1.10.0, caTools 1.17.1, codetools 0.2-14, colorspace 1.2-6, digest 0.6.8, doRNG 1.6, ellipse 0.3-8, foreach 1.4.3, futile.logger 1.4.1, futile.options 1.0.0, gdata 2.17.0, genefilter 1.52.0, ggplot2 1.0.1, grid 3.2.2, gtable 0.1.2, gtools 3.5.0, igraph 1.0.1, illuminaio 0.12.0, iterators 1.0.8, lambda.r 1.1.7, lattice 0.20-33, limma 3.26.0, lmtest 0.9-34, locfit 1.5-9.1, magrittr 1.5, matrixStats 0.14.2, mclust 5.0.2, methylumi 2.16.0, mgcv 1.8-7, minfi 1.16.0, mixOmics 5.1.2, multtest 2.26.0, munsell 0.4.2, nleqslv 2.9, nlme 3.1-122, nor1mix 1.2-1, pheatmap 1.0.7, pkgmaker 0.22, plyr 1.8.3, preprocessCore 1.32.0, proto 0.3-10, quadprog 1.5-5, randomForest 4.6-12, registry 0.3, reshape 0.8.5, reshape2 1.4.1, rgl 0.95.1367, rngtools 1.2.4, rtracklayer 1.30.0, scales 0.3.0, scatterplot3d 0.3-36, siggenes 1.44.0, splines 3.2.2, stats4 3.2.2, stringi 0.5-5, stringr 1.0.0, survival 2.38-3, tools 3.2.2, vsn 3.38.0, xtable 1.7-4, zlibbioc 1.16.0, zoo 1.7-12