Affy array outlier detection via dimension reduction

A Asare, Z Gao, V Carey

May 2, 2019

Contents

1 Introduction

Clinical trials groups now routinely produce hundreds of microarrays to generate measures of clinical conditions and treatment responses at the level of mRNA abundance. Objective, quantitative measures of array quality are important to support these projects. Numerous packages in Bioconductor address quality assessment procedures. *ArrayQualityMetrics* is a particularly attractive set of tools. We provide *arrayMvout* as a module that performs parametric outlier detection after data reduction to support formal decisionmaking about array acceptability. Ultimately the measures and procedures provided by *arrayMvout* may be useful as components of other packages for quality assessment. Another closely related package is *mdqc*, which employs a variety of robustifications of Mahalanobis distance to help identify outlying arrays.

Suppose there are N affymetrix arrays to which N independent samples have been hybridized. The arrayMvout package computes Q quality measures which constitute

array-specific features. These features are then analyzed in two steps. First, principal components analysis is applied to the $N \times Q$ feature matrix. Second, parametric multivariate outlier detection with calibration for multiple testing is applied to a subset of the resulting principal components. Arrays identified as outliers by this procedure are then subject to additional inspection and/or exclusion as warranted.

In this vignette we illustrate application of the procedure for a 'negative control' (raw MAQC data) and several constructed quality defect situations.

2 Illustration with MAQC data

We have serialized sufficient information on the MAQC subset to allow a simple demonstration of a negative control set. The MAQC data should be free of outliers.

We will manually search for outliers in this data resource. We compute principal components and take the first three.

```
> library(arrayMvout)
> data(maqcQA)
> mm = ArrayOutliers(maqcQA[, 3:11], alpha=.01)
> mm

ArrayOutliers result.
The call was:
.local(data = data, alpha = alpha, alphaSeq = alphaSeq)
No outliers. First row of QC features
     avgBG      SF Present   HSAC07    GAPDH     NUSE        RLE   RLE_IQR
1 60.05505 1.17657 52.4225 1.245477 1.015217 1.057659 0.04185417 0.5516266
  RNAslope
1 3.141527
```

There are no outliers found at a false labeling rate of 0.01.

3 Illustration with arrays from a clinical trial network

Another data resource with some problematic arrays is also included.

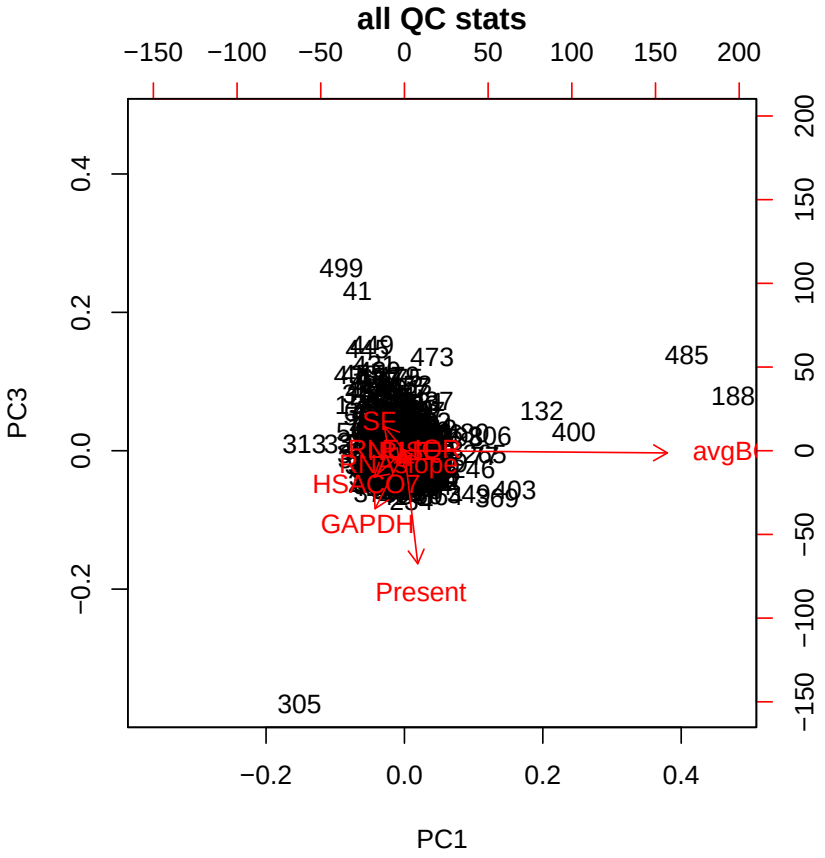
```
> data(itnQA)
> ii = ArrayOutliers(itnQA, alpha=.01)
> ii

ArrayOutliers result.
The call was:
```

```
.local(data = data, alpha = alpha, alphaSeq = alphaSeq)
There were  507  samples with  18  outliers detected.
Coordinate-wise means of inlying arrays:
       avgBG           SF      Present       HSACO7        GAPDH         NUSE
4.498102e+01 1.731430e+00 4.162066e+01 2.187432e+00 1.770430e+00 1.000867e+00
         RLE      RLE_IQR     RNAslope
6.050559e-05 2.976345e-01 3.427889e+00
Features of outlying arrays:
        avgBG         SF  Present    HSACO7      GAPDH      NUSE          RLE
403  77.07796  0.4374264 47.55007  1.433827   1.307327 0.9991413  0.005285121
16   41.00919  1.9288207 39.60311 12.647909   4.707294 1.0097374  0.021545990
449  37.26800  6.2588701 24.39140  4.131615   2.047267 1.0284928  0.127501020
189  40.61447  1.9891220 39.39095 10.360385   6.718598 1.0103497  0.036462661
445  36.28049  9.1599338 25.08642  5.930873   2.189993 1.1064651  0.036035969
473  54.36467  3.6423721 26.03567  2.497312   1.893548 1.0438432  0.108838227
235  40.36647  2.3888050 39.61957  8.852826   7.201938 1.0302444  0.051526272
323  31.73768  5.6643255 32.43347 16.760480  11.955661 1.0507761  0.052612806
268  50.32263  2.6636200 34.85871 23.990430   7.101770 1.0542473  0.092065904
274  39.08553  2.9476038 36.66575 22.884391  12.036149 1.0457600  0.087581891
132  85.82265  1.8183446 34.61363  1.432298   1.510193 1.0822086  0.009663181
499  30.14740 29.3554869 17.02423  2.629898   1.718535 1.0874808  0.132794452
41   33.71813 15.6434771 17.80887  1.073282   2.643669 1.1669600  0.321298991
400  95.45017  1.2115402 36.64563  3.508757   2.479803 1.1128690 -0.000687765
485 129.10624  1.5643922 23.87197  3.057225   1.810761 1.1098076  0.073777325
188 142.44325  1.4429218 30.23137  2.907469   2.101054 1.1475918 -0.018917415
313  25.37851 16.8069277 25.72474 56.917203  11.840316 1.1360326  0.073802881
305  38.11554 19.0772560 11.60677 67.021339 120.011952 1.1987268  0.349949413
      RLE_IQR    RNAslope
403 0.2197193  3.66807105
16  0.3299879  5.42259351
449 0.5672631  1.89557125
189 0.3305528  5.80774028
445 0.5465581  2.54468591
473 0.6448400  1.32173147
235 0.3657481  6.54325480
323 0.4407913  5.88500524
268 0.5755214  5.74973052
274 0.4367455  7.09471717
132 0.6664186  1.19952480
499 0.6250615  1.87786720
41  1.0754575  1.11521819
400 0.6759216  2.46627267
```

```
485 0.9486388 -0.04085089
188 0.8908112  1.76393357
313 0.6503561  6.13572050
305 1.3208768  6.96678147
```

We have a simple visualization.

```
> plot(ii, choices=c(1,3))
```



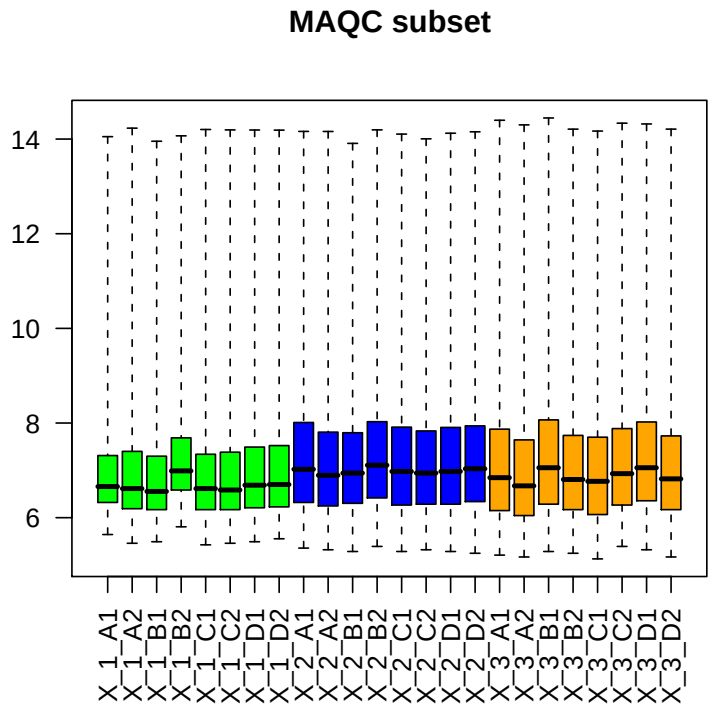
4 Manual work with the MAQC subset

The remaining text of this vignette is computed statically. The source code with eval=FALSE is given as an appendix.

We consider an AffyBatch supplied with the Bioconductor MAQCsubset package. Marginal boxplots of raw intensity data are provided in the next figure. Sample labels are decoded AFX _ [lab] _ [type] [replicate] .CEL where [lab] $\in (1, 2, 3)$, [type] denotes

mixture type (A = 100% USRNA, B = 100% Ambion brain, C = 75% USRNA, 25% brain, D = 25% USRNA, 75% brain), and replicate $\in (1, 2)$.

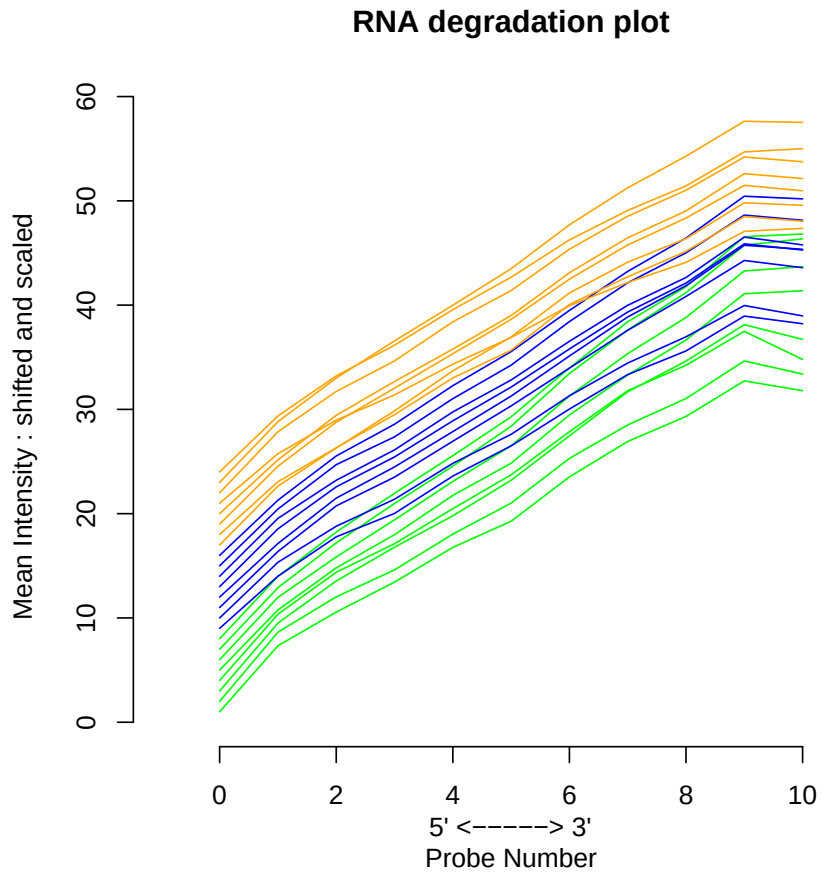
```
> library(arrayMvout)
> library(MAQCsubset)
> if (!exists("afxsub")) data(afxsub)
> sn = sampleNames(afxsub)
> if (nchar(sn)[1] > 6) {
+     sn = substr(sn, 3, 8)
+     sampleNames(afxsub) = sn
+ }

> opar = par(no.readonly = TRUE)
> par(mar = c(10, 5, 5, 5), las = 2)
> boxplot(afxsub, main = "MAQC subset", col = rep(c("green", "blue",
+     "orange"), c(8, 8, 8)))
> par(opar)
```



4.1 QA diagnostics

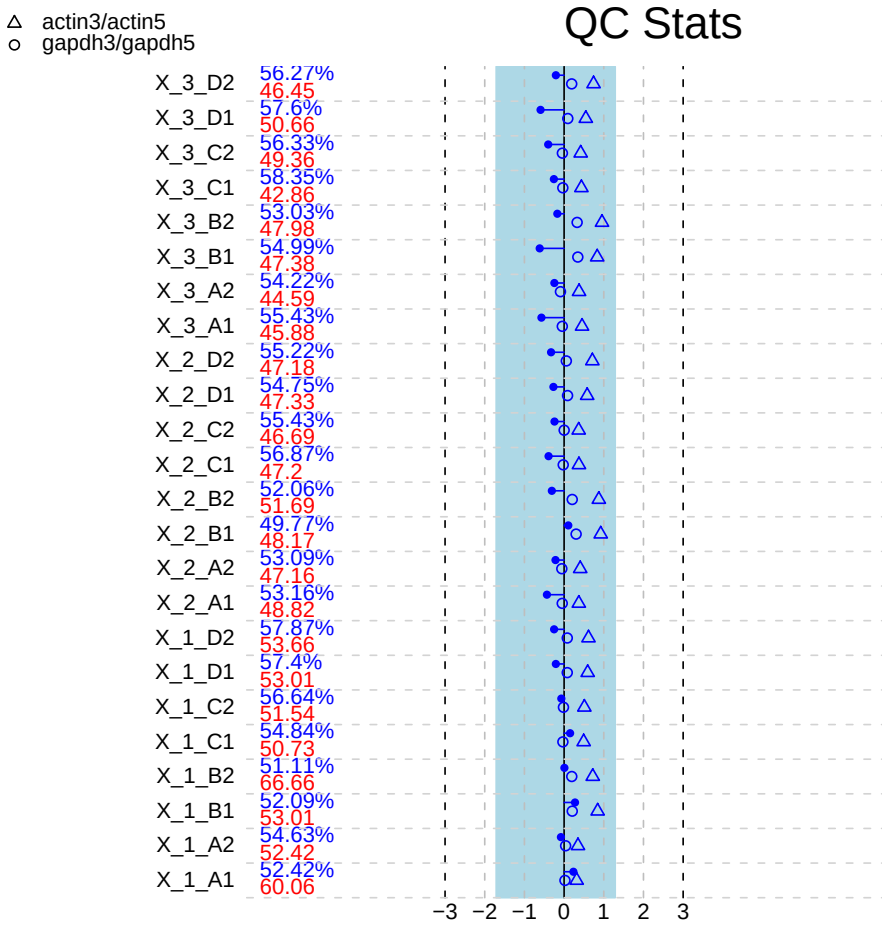
Of interest are measures of RNA degradation:

```
> library(arrayMvout)
> data(afxsubDEG)
> plotAffyRNAdeg(afxsubDEG, col = rep(c("green", "blue", "orange"),
+     c(8, 8, 8)))
```



and the general 'simpleaffy' QC display:

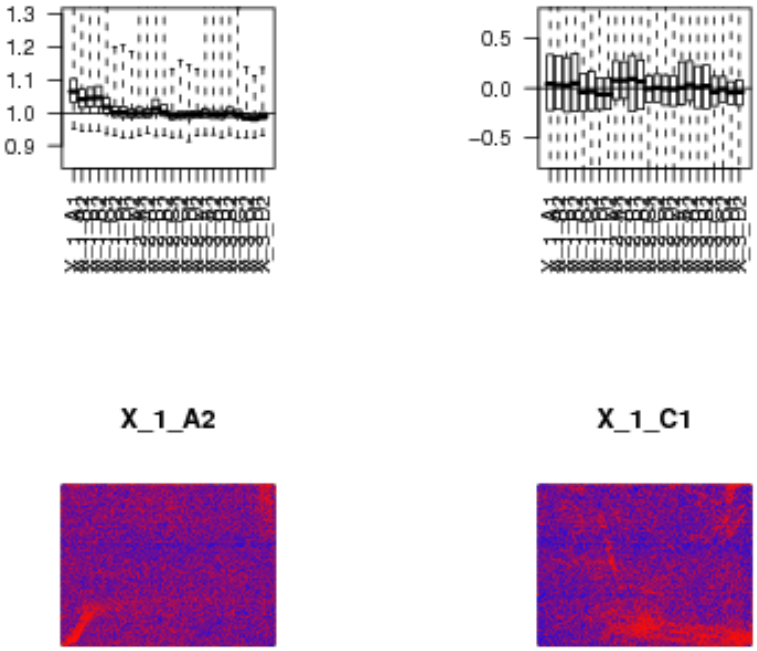
```
> data(afxsubQC)
> plot(afxsubQC)
```



The affyPLM package fits probe-level robust regressions to obtain probe-set summaries.

```
> library(affyPLM)
> splm = fitPLM(afxsub)

> png(file = "doim.png")
> par(mar = c(7, 5, 5, 5), mfrow = c(2, 2), las = 2)
> NUSE(splm, ylim = c(0.85, 1.3))
> RLE(splm)
> image(splm, which = 2, type = "sign.resid")
> image(splm, which = 5, type = "sign.resid")
```

In the following graphic, we have the NUSE distributions (upper left), the RLE distributions (upper right), and second and fifth chips in signed residuals displays.



These chips seem to have adequate quality, although there is some indication that the first four are a bit different with respect to variability.

4.2 Outlier detection using diagnostics

Let's apply the diagnostic-dimension reduction-multivariate outlier procedure `ArrayOutliers`.

```
> AO = ArrayOutliers(afxsub, alpha = 0.05, qcOut = afxsubQC, plmOut = splm,
+     degOut = afxsubDEG)
> nrow(AO[["outl"]])
```

```
[1] 0
```

We see that there are no outliers declared. This seems a reasonable result for arrays that were hybridized in the context of a QC protocol. Let us apply the mdqc procedure. As input this takes any matrix of quality indicators. The third component of our ArrayOutliers result provides these as computed using simpleaffy qc(), affy AffyRNAdeg, and affyPLM NUSE and RLE. The QC measures for the first two chips are:

```
> AO[[3]][1:2, ]

          avgBG        SF  Present   HSACO7    GAPDH     NUSE        RLE
X_1_A1 60.05505 1.1765695 52.42250 1.245477 1.065898 1.064069 0.04163564
X_1_A2 52.42248 0.9459093 54.63009 1.273977 1.094208 1.040718 0.03253112
         RLE_IQR RNAslope
X_1_A1 0.5626532 3.141527
X_1_A2 0.5459847 3.210157
```

We now use the mdqc package with MVE robust covariance estimation.

```
> library(mdqc)
> mdq = mdqc(AO[[3]], robust = "MVE")
> mdq

Method used: nogroups         Number of groups: 1
Robust estimator: MVEMDs exceeding the square root of the  90 % percentile of the Chi-S
[1]  6 16 17 18 21 24
MDs exceeding the square root of the  95 % percentile of the Chi-Square distribution
[1]  6 16 17 18 21 24
MDs exceeding the square root of the  99 % percentile of the Chi-Square distribution
[1]  6 16 17 18 21 24
```

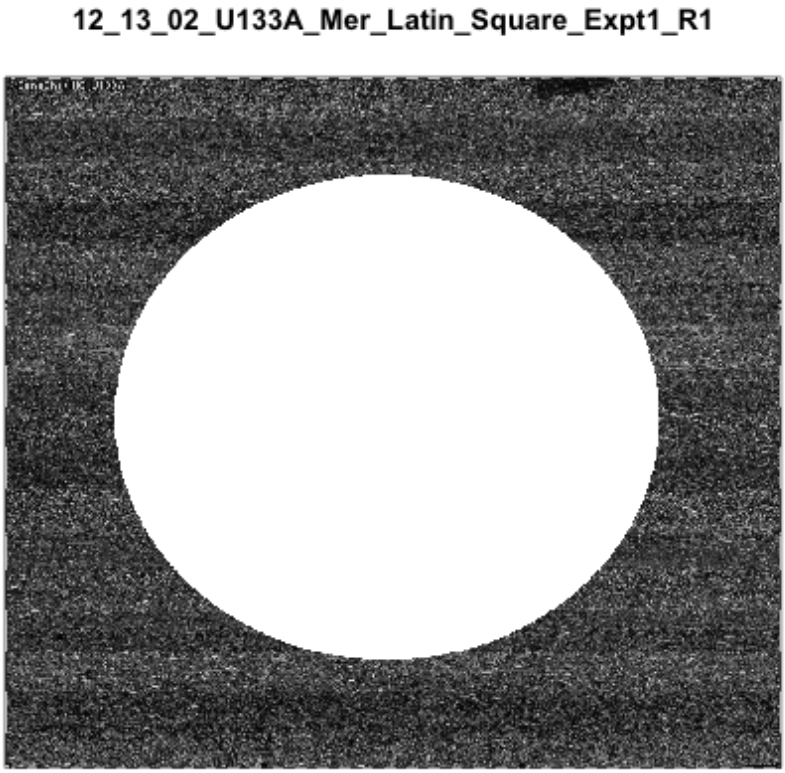
We see that a number of the arrays are determined to be outlying by this procedure according to several thresholds.

5 Intensity contamination in the spikein data

We begin with a simple demonstration of a contamination procedure that simulates severe blobby interference with hybridization.

The code below is unevaluated to speed execution. Set eval=TRUE on all chunks to see the actual process.

```
> require(mvoutData)
> data(s12c)

> image(s12c[, 1])
```



For this AffyBatch instance, we have contaminated the first two arrays in this way. We now apply the ArrayOutliers procedure:

```
> aos12c = ArrayOutliers(s12c, alpha = 0.05)

> aos12c[[1]]

                                       samp      avgBG       SF  Present
1 12_13_02_U133A_Mer_Latin_Square_Expt1_R1 7205.48413 5.494978 19.05381
2 12_13_02_U133A_Mer_Latin_Square_Expt2_R1 7205.12544 5.801398 17.62332
8 12_13_02_U133A_Mer_Latin_Square_Expt8_R1   31.23121 1.181946 45.47982
    HSAC07     GAPDH     NUSE          RLE    RLE_IQR    RNAslope
1 9.488671 8.6364494 1.578471 -0.514913919 1.33150311 -0.05195296
2 8.517500 9.7475845 1.579164 -0.514617696 1.37213796 -0.07638096
8 1.042255 0.8965249 1.000006  0.001987620 0.09164597  1.53050152
```

We find three arrays declared to be outlying. At the different candidate significance levels we have:

```
> aos12c[[4]]
[[1]]
[[1]]$inds
[1] 1 2

[[1]]$vals
           PC1          PC2          PC3
[1,] -6.345625  0.08184738  0.05419247
[2,] -6.485447 -0.07281758 -0.05421514

[[1]]$k
[1] 5

[[1]]$alpha
[1] 0.01


[[2]]
[[2]]$inds
[1] 8 1 2

[[2]]$vals
           PC1          PC2           PC3
[1,]  1.104062 -0.18796407 -0.008319271
[2,] -6.345625  0.08184738  0.054192469
[3,] -6.485447 -0.07281758 -0.054215145

[[2]]$k
[1] 5

[[2]]$alpha
[1] 0.05


[[3]]
[[3]]$inds
[1] 8 1 2

[[3]]$vals
           PC1          PC2           PC3
[1,]  1.104062 -0.18796407 -0.008319271
[2,] -6.345625  0.08184738  0.054192469
```

```
[3,] -6.485447 -0.07281758 -0.054215145

[[3]]$k
[1] 5

[[3]]$alpha
[1] 0.1
```

So at the 0.01 level we have identified only the contaminated arrays.

We apply mdqc in the same manner.

```
> mdqc(aos12c[[3]], robust = "MVE")

Method used: nogroups          Number of groups: 1
Robust estimator: MVEMDs exceeding the square root of the  90 % percentile of the Chi-S
[1] 2
MDs exceeding the square root of the  95 % percentile of the Chi-Square distribution
[1] 2
MDs exceeding the square root of the  99 % percentile of the Chi-Square distribution
[1] 2
```

We see that only one of the contaminated arrays is identified by this procedure. This may be an instance of masking.

6 Appendix: Sources and text for statically computed sections with eval set to false

Manual work with the MAQC subset

All the code follwing has had evaluation turned off because execution times are slow.

We consider an AffyBatch supplied with the Bioconductor MAQCsubset package. Marginal boxplots of raw intensity data are provided in the next figure. Sample labels are decoded AFX _ [lab] _ [type] [replicate] .CEL where [lab] $\in (1, 2, 3)$, [type] denotes mixture type (A = 100% USRNA, B = 100% Ambion brain, C = 75% USRNA, 25% brain, D = 25% USRNA, 75% brain), and replicate $\in (1, 2)$.

```
> library(arrayMvout)
> library(MAQCsubset)
> if (!exists("afxsub")) data(afxsub)
> sn = sampleNames(afxsub)
> if (nchar(sn)[1] > 6) {
+   sn = substr(sn, 3, 8)
```

```
+   sampleNames(afxsub) = sn
+ }

> opar = par(no.readonly=TRUE)
> par(mar=c(10,5,5,5), las=2)
> boxplot(afxsub, main="MAQC subset",
+   col=rep(c("green", "blue", "orange"), c(8,8,8)))
> par(opar)
```

QA diagnostics

Of interest are measures of RNA degradation:

```
> #afxsubDEG = AffyRNAdeg(afxsub)
> #save(afxsubDEG, file="afxsubDEG.rda")
> library(arrayMvout)
> data(afxsubDEG)
> plotAffyRNAdeg(afxsubDEG,
+   col=rep(c("green", "blue", "orange"),c(8,8,8)))
```

and the general 'simpleaffy' QC display:

```
> #afxsubQC = qc(afxsub)
> #save(afxsubQC, file="afxsubQC.rda")
> data(afxsubQC)
> plot(afxsubQC)
```

The affyPLM package fits probe-level robust regressions to obtain probe-set summaries.

```
> library(affyPLM)
> #if (file.exists("splm.rda")) load("splm.rda")
> #if (!exists("splm")) splm = fitPLM(afxsub)
> splm = fitPLM(afxsub)
> #save(splm, file="splm.rda")

> png(file="doim.png")
> par(mar=c(7,5,5,5),mfrow=c(2,2),las=2)
> NUSE(splm, ylim=c(.85,1.3))
> RLE(splm)
> image(splm, which=2, type="sign.resid")
> image(splm, which=5, type="sign.resid")
```

These chips seem to have adequate quality, although there is some indication that the first four are a bit different with respect to variability.

Outlier detection using diagnostics

Let's apply the diagnostic-dimension reduction-multivariate outlier procedure `ArrayOutliers`.

```
> AO = ArrayOutliers(afxsub, alpha=0.05, qcOut=afxsubQC,
+   plmOut=splm, degOut=afxsubDEG)
> nrow(AO[["outl"]])
```

We see that there are no outliers declared. This seems a reasonable result for arrays that were hybridized in the context of a QC protocol. Let us apply the mdqc procedure. As input this takes any matrix of quality indicators. The third component of our ArrayOutliers result provides these as computed using simpleaffy qc(), affy AffyRNAdeg, and affyPLM NUSE and RLE. The QC measures for the first two chips are:

```
> AO[[3]][1:2, ]
```

We now use the mdqc package with MVE robust covariance estimation.

```
> library(mdqc)
> mdq = mdqc( AO[[3]], robust="MVE" )
> mdq
```

We see that a number of the arrays are determined to be outlying by this procedure according to several thresholds.

Intensity contamination in the spikein data

We begin with a simple demonstration of a contamination procedure that simulates severe blobby interference with hybridization.

The code below is unevaluated to speed execution. Set eval=TRUE on all chunks to see the actual process.

```
> require(mvoutData)
> data(s12c)

> image(s12c[,1])
```

For this AffyBatch instance, we have contaminated the first two arrays in this way. We now apply the ArrayOutliers procedure:

```
> aos12c = ArrayOutliers(s12c, alpha=0.05)

> aos12c[[1]]
```

We find three arrays declared to be outlying. At the different candidate significance levels we have:

```
> aos12c[[4]]
```

So at the 0.01 level we have identified only the contaminated arrays.

We apply mdqc in the same manner.

```
> mdqc(aos12c[[3]], robust="MVE")
```

We see that only one of the contaminated arrays is identified by this procedure. This may be an instance of masking.

```
> sessionInfo()
```

```
R version 3.6.0 (2019-04-26)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.2 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
[1] arrayMvout_1.42.0   lumi_2.36.0         affy_1.62.0
[4] Biobase_2.44.0      BiocGenerics_0.30.0 parody_1.42.0

loaded via a namespace (and not attached):
  [1] nlme_3.1-139               bitops_1.0-6
  [3] matrixStats_0.54.0         bit64_0.9-7
  [5] RColorBrewer_1.1-2         progress_1.2.0
  [7] httr_1.4.0                 GenomeInfoDb_1.20.0
```

```
 [9] affyContam_1.42.0          gcrma_2.56.0
[11] doRNG_1.7.1                nor1mix_1.2-3
[13] R6_2.4.0                   affyio_1.54.0
[15] KernSmooth_2.23-15         HDF5Array_1.12.0
[17] mgcv_1.8-28                colorspace_1.4-1
[19] DBI_1.0.0                  methylumi_2.30.0
[21] withr_2.1.2                tidyselect_0.2.5
[23] prettyunits_1.0.2          base64_2.0
[25] bit_1.1-14                 compiler_3.6.0
[27] preprocessCore_1.46.0      xml2_1.2.0
[29] DelayedArray_0.10.0        pkgmaker_0.27
[31] rtracklayer_1.44.0         readr_1.3.1
[33] genefilter_1.66.0          quadprog_1.5-6
[35] askpass_1.1                mdqc_1.46.0
[37] stringr_1.4.0              digest_0.6.18
[39] Rsamtools_2.0.0            illuminaio_0.26.0
[41] siggenes_1.58.0            GEOquery_2.52.0
[43] XVector_0.24.0             pkgconfig_2.0.2
[45] scrime_1.3.5               bibtex_0.4.2
[47] limma_3.40.0               rlang_0.3.4
[49] RSQLite_2.1.1              DelayedMatrixStats_1.6.0
[51] mclust_5.4.3               BiocParallel_1.18.0
[53] dplyr_0.8.0.1              RCurl_1.95-4.12
[55] magrittr_1.5               GenomeInfoDbData_1.2.1
[57] Matrix_1.2-17              Rcpp_1.0.1
[59] S4Vectors_0.22.0           Rhdf5lib_1.6.0
[61] stringi_1.4.3              nleqslv_3.3.2
[63] MASS_7.3-51.4              SummarizedExperiment_1.14.0
[65] zlibbioc_1.30.0            rhdf5_2.28.0
[67] plyr_1.8.4                 bumphunter_1.26.0
[69] grid_3.6.0                 minfi_1.30.0
[71] blob_1.1.1                 crayon_1.3.4
[73] lattice_0.20-38            Biostrings_2.52.0
[75] splines_3.6.0              multtest_2.40.0
[77] GenomicFeatures_1.36.0     annotate_1.62.0
[79] hms_0.4.2                  locfit_1.5-9.1
[81] pillar_1.3.1               beanplot_1.2
[83] GenomicRanges_1.36.0       rngtools_1.3.1.1
[85] codetools_0.2-16           biomaRt_2.40.0
[87] stats4_3.6.0               glue_1.3.1
[89] XML_3.98-1.19              data.table_1.12.2
[91] BiocManager_1.30.4         simpleaffy_2.60.0
```

```
 [93] foreach_1.4.4               tidyr_0.8.3
 [95] purrr_0.3.2                 openssl_1.3
 [97] reshape_0.8.8               assertthat_0.2.1
 [99] xtable_1.8-4                survival_2.44-1.1
[101] tibble_2.1.1                iterators_1.0.10
[103] GenomicAlignments_1.20.0    AnnotationDbi_1.46.0
[105] registry_0.5-1              memoise_1.1.0
[107] IRanges_2.18.0              cluster_2.0.9
```