

Introduction to the xps Package: Comparison to Affymetrix Power Tools (APT)

Christian Stratowa

June, 2009

Contents

1	Introduction	1
2	Comparison of results for the Human Genome U133 Plus 2.0 Array	2
2.1	Comparison of different implementations of RMA summarization	2
2.2	Comparison of different implementations of MAS 5.0 summarization	6
2.3	Comparison of different implementations of MAS 5.0 detection calls	7
3	Comparison of results for the Human Gene 1.0 ST Array	9
3.1	Comparison of the RMA summarization implementations	9
3.2	Comparison of the DABG call implementations	12
3.3	Comparison of results when using HuGene as exon array	13
4	Comparison of results for the Human Exon 1.0 ST Array	15
4.1	Comparison of RMA summarization at the probeset level	15
4.2	Comparison of RMA summarization at the transcript level	18
4.3	Comparison of the DABG call implementations	21
5	Comparison of Affymetrix gene expression arrays	22
5.1	Comparison of RMA summarization for common transcript probesets	22
5.2	Differentially expressed transcripts in breast vs prostate	24
6	Summary	26
A	Appendices	28
A.1	Some notes on PLIER	28

1 Introduction

The aim of this document is to compare the results obtained with package `xps` to the results obtained with Affymetrix Power Tools (APT version apt-1.10.2). For this purpose the Exon Array Data Set "Tissue Mixture" will be used, which can be downloaded from the Affymetrix web-site for expression arrays "Human Genome U133 Plus 2.0", "Human Gene 1.0 ST" and "Human Exon 1.0 ST", respectively. First, APT and `xps` will be compared for each array type separately, then the results obtained with the different arrays will be compared to each other for both APT and `xps`. To speed up analysis only two human tissues will be used, namely breast and prostate.

For analyzing expression arrays, including gene and exon arrays, APT contains the command-line program `apt-probeset-summarize`, which supports background correction (RMA, MAS5), normalization (linear scaling, quantile, sketch), and summarization (RMA, MAS5, PLIER) methods.

Note: In order to keep this document short, only example code will be presented here, the full source code is available in files "script4xps2apt.R" and "script4bestmatch.R" located in directory "xps/examples".

2 Comparison of results for the Human Genome U133 Plus 2.0 Array

2.1 Comparison of different implementations of RMA summarization

When doing RMA summarization with `apt-probeset-summarize`, sketch-quantile normalization (using only a subset of data) is applied by default in order to decrease memory consumption. Furthermore, the Affymetrix GUI application "Expression Console" can only use sketch-quantile normalization. For this reason let us first compare RMA using sketch-quantile vs quantile normalization.

The following command will compute RMA using sketch-quantile normalization:

```
apt-probeset-summarize -a rma-bg,quant-norm.sketch=-1.usepm=true.bioc=true, \
  pm-only,med-polish.expon=true -d HG-U133_Plus_2.cdf *.CEL}
```

Note that all CEL-files and the corresponding CDF-file must be located in the current directory. The output can be directly imported into R:

```
> apt.rma.sk <- read.delim("rma-bg.quant-norm.pm-only.med-polish.summary.txt",
+   row.names = 1, comment.char = "", skip = 52)
```

To compute RMA using full quantile normalization the following command is used:

```
apt-probeset-summarize -a rma-bg,quant-norm.sketch=0.usepm=true.bioc=true, \
  pm-only,med-polish.expon=true -d HG-U133_Plus_2.cdf *.CEL
```

Again, the output can be directly imported into R:

```
> apt.rma <- read.delim("rma-bg.quant-norm.pm-only.med-polish.summary.txt",
+   row.names = 1, comment.char = "", skip = 52)
```

Please note that in both cases the name of the output file is the same.

To compare the results, we will use "MvA-plots" in almost all cases. Thus we can plot:

```
> plot(log2(apt.rma[, 1] * apt.rma.sk[, 1])/2, log2(apt.rma.sk[,
+   1]/apt.rma[, 1]), main = "APT: RMA vs RMA-sketch", xlab = "A = Log2(SketchRMA*RMA)",
+   ylab = "M = Log2(SketchRMA/RMA)", log = "", ylim = c(-0.1,
+   0.1))
```

Since we use only the triplicate CEL-files for breast and prostate, respectively, column one of the data.frames `apt.rma.sk` and `apt.rma` show the normalized expression values of tissue "breast_A".

The resulting Figure 1 shows that using sketch-quantile normalization gives a slightly different result, especially at low expression values. Thus, for all following comparisons we will use the full quantile

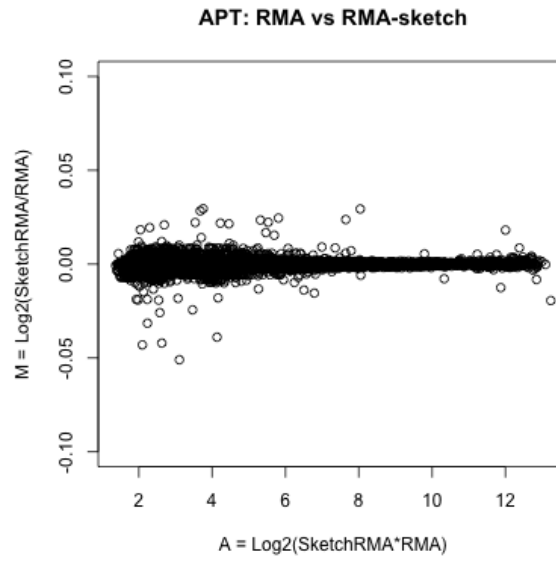


Figure 1: APT RMA using quantile vs quantile-sketch.

normalization.

Note: One advantage of package `xps` compared to APT is, that you can do RMA with full quantile normalization on computers with only 1GB RAM, even for exon arrays.

Now let us compare APT and xps. For this purpose we compute RMA as described in vignette "xps.pdf", and extract `xps.rma` from `data.rma`:

```
> data.rma <- rma(data.u133p2, "MixU133P2RMA", background = "pmonly",
+   normalize = TRUE)
> xps.rma <- validData(data.rma)
```

Now we can compare the results obtained with APT and xps for tissue "breast_A" using an "MvA-plot":

```
> plot(log2/apt.rma[, 1] * xps.rma[, 1])/2, log2(xps.rma[, 1]/apt.rma[,
+   1]), main = "RMA: XPS vs APT", xlab = "A = Log2(XPS*APT)",
+   ylab = "M = Log2(XPS/APT)", log = "", ylim = c(-0.1, 0.1))
```

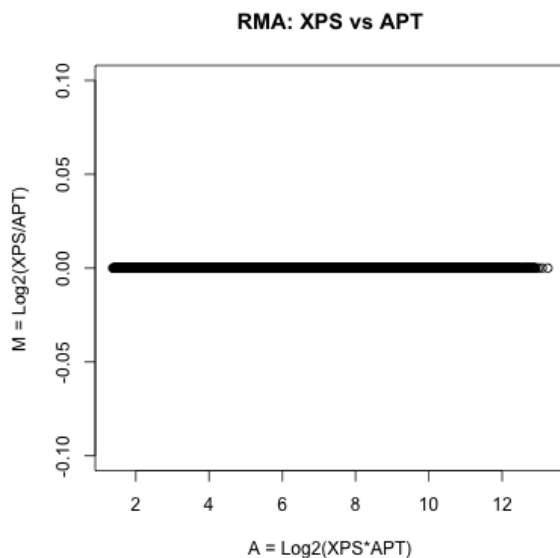


Figure 2: RMA computed with apt and xps, respectively.

As shown in Figure 2, the results obtained with APT and xps are identical.

For the sake of completeness, let us compare both results to the results obtained with package `affy`. For this purpose we compute:

```
> affy.rma <- justRMA()
> affy.rma <- 2^exprs(affy.rma)
```

and plot:

```
> tmp <- cbind(xps.rma[, 1], affy.rma[, 1], apt.rma[, 1])
> colnames(tmp) <- c("xps.rma", "affy.rma", "apt.rma")
> pairs(log2(tmp), labels = colnames(tmp))
```

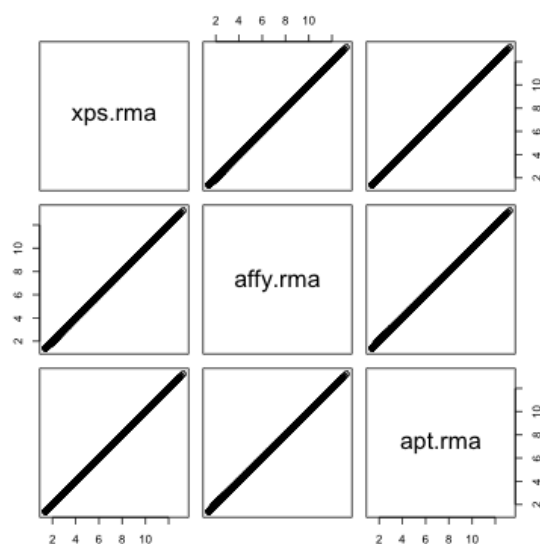


Figure 3: RMA computed with xps, affy and apt, respectively.

As shown in Figure 3, the results are identical in all three cases. For example, the differences between xps and APT or affy are less than $4e-6$.

2.2 Comparison of different implementations of MAS 5.0 summarization

The command-line to compute MAS5 summarization using APT is:

```
apt-probeset-summarize -a mas5-bg,pm-mm,mass5-signal -d HG-U133_Plus_2.cdf *.CEL
```

The output is imported into R and must be scaled to `sc=500` since APT does not allow for signal level normalization across the chip (as mentioned in APT FAQ):

```
> apt.mas5 <- read.delim("mas5-bg.pm-mm.mas5-signal.summary.txt",
+   row.names = 1, comment.char = "", skip = 52)
> apt.mas5 <- apply(apt.mas5, 2, function(x) {
+   x * (500/mean(x, trim = 0.02))
+ })
```

Now we compute MAS5 with `xps` as described in vignette "xps.pdf":

```
> data.mas5 <- mas5(data.u133p2, "MixU133P2MAS5All", normalize = TRUE,
+   sc = 500, update = TRUE)
> xps.mas5 <- validData(data.mas5)
```

Finally, we compute MAS5 using package `affy`:

```
> affy <- ReadAffy()
> affy.mas5 <- mas5(affy, normalize = TRUE, sc = 500)
> affy.mas5 <- exprs(affy.mas5)
```

Using "MvA-plots" to compare the MAS5 expression levels for sample "Breast_A" we obtain the results shown in Figure 4.

The upper row shows how the implementation of the MAS5 algorithm in packages `xps` and `affy` compare to APT. Both packages show clear differences to APT, especially at low expression levels. However, as APT FAQ mentions, "a new implementation of the MAS5 methods was necessary to get this method to work in the APT architecture". Thus, the middle row shows how packages `xps` and `affy` compare to the MAS5 results obtained with the Affymetrix "Expression Console", which has implemented MAS5 identical to GCOS. There are still differences seen in the expression levels, with the `xps` implementation of MAS5 being slightly more close to the GCOS implementation of MAS5. Finally, the lower row shows the differences between APT and GCOS, and between `xps` and `affy`, respectively.

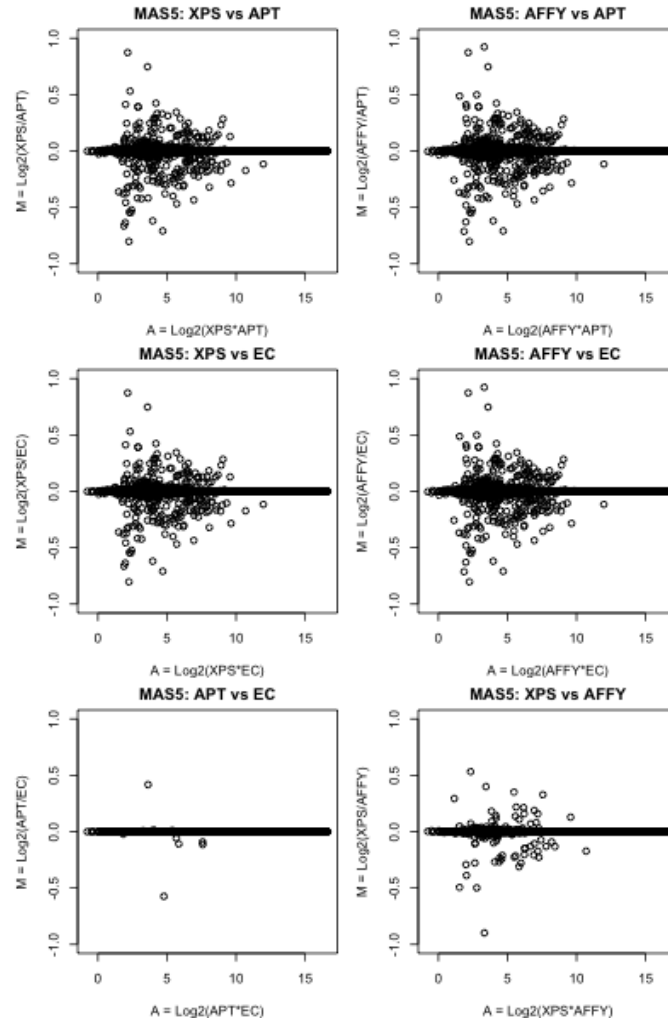


Figure 4: MAS5 computed with xps, affy, apt and EC, respectively.

2.3 Comparison of different implementations of MAS 5.0 detection calls

Now let us compute MAS5 detection calls. The command-line for APT is:

```
apt-probeset-summarize -a pm-mm,mass5-detect.calls=1.pairs=1 -d HG-U133_Plus_2.cdf -x 10 *.CEL
```

Using package xps MAS5 detection calls are computed as follows:

```
> call.mas5 <- mas5.call(data.u133p2, "MixU133P2Call")
> xps.pval <- validData(call.mas5)
```

while package affy uses the following commands:

```
> affy <- ReadAffy()
> affy.dc5 <- mas5calls(affy)
> affy.pval <- assayData(affy.dc5)[["se.exprs"]]
```

Figure 5 compares the detection p-values obtained with packages `xps` and `affy`, as well as Expression Console and APT.

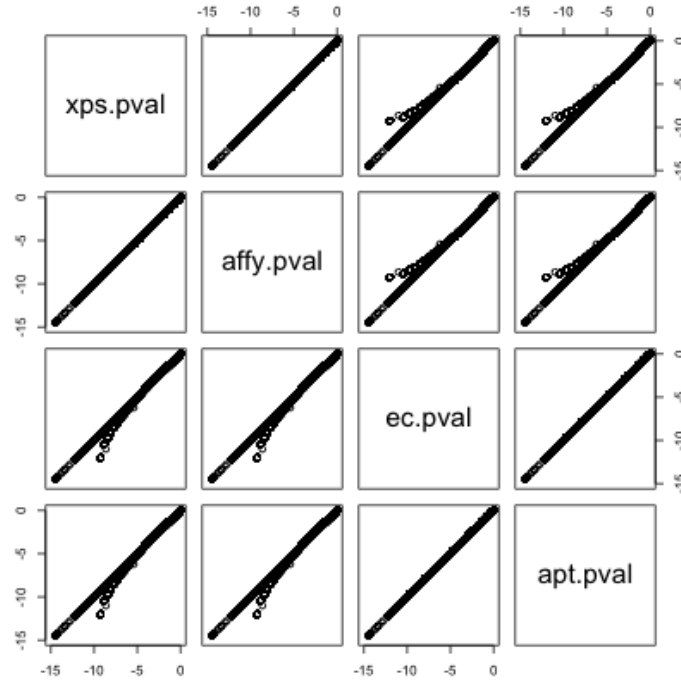


Figure 5: Comparison of p-values obtained with `xps`, `affy`, Expression Console and APT.

While Expression Console and APT result in identical p-values, both `xps` and `affy` differ from APT. However, as shown in Figure 6, the detection call p-values obtained with `xps` and `affy` are identical.

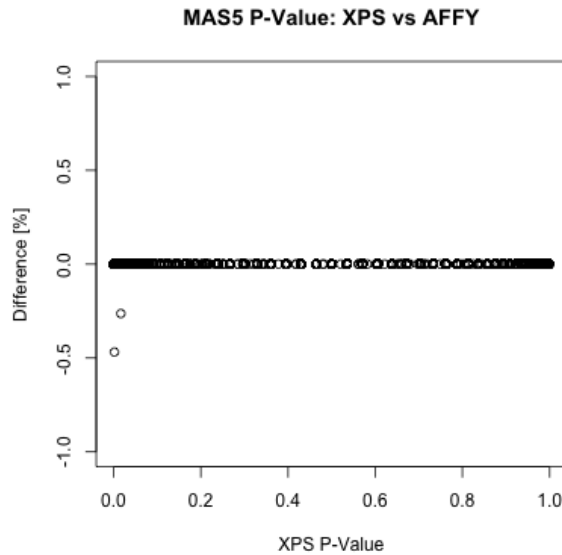


Figure 6: Difference between p-values obtained with xps and affy.

3 Comparison of results for the Human Gene 1.0 ST Array

3.1 Comparison of the RMA summarization implementations

While the results for expression arrays obtained with `xps` and APT, respectively, can be directly compared, since the probesets used are identical, comparison of `xps` and APT with gene/exon arrays requires the definition of common probesets first.

Let us first compute RMA using package `xps`:

```
> data.rma.tc <- rma(data.genome, "MixHuGeneRMACore_tc", background = "antigenomic",
+   normalize = TRUE, option = "transcript", exonlevel = "core")
> xps.rma.tc <- validData(data.rma.tc)
```

Now we can create a file specifying the probesets to summarize for APT:

```
> tmp <- as.data.frame(rownames(xps.rma.tc))
> colnames(tmp) <- "probeset_id"
> write.table(tmp, "transcriptList.txt", quote = FALSE, row.names = FALSE)
```

This `transcriptList` contains now the probesets used by `xps` and can be added as option when computing RMA using APT:

```
apt-probeset-summarize -a rma-bg,quant-norm.sketch=0.usepm=true.bioc=true, \
  pm-only,med-polish.expon=true \
  -p HuGene-1_0-st-v1.r3.pgf -c HuGene-1_0-st-v1.r3.clf \
  -s transcriptList.txt *.CEL
```

Figure 7 shows the resulting "MvA-plot" for tissue "breast_A".

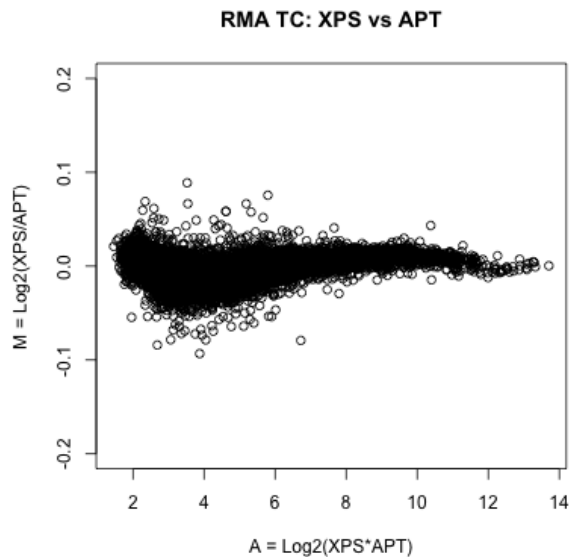


Figure 7: RMA computed with xps and apt, respectively.

As Figure 7 shows, there is clearly a difference of few percent in the expression levels obtained when using xps or APT, respectively, including a non-linearity at low expression levels.

In order to understand this difference let us first compare the "median-polish" summarization step (w/o background correction and quantile normalization).

In package xps this is done as follows:

```
> expr.mp.tc <- express(data.genome, "MixHuGeneMedPolcore_tc",
+   summarize.method = "medianpolish", summarize.select = "pmonly",
+   summarize.option = "transcript", summarize.logbase = "log2",
+   summarize.params = c(10, 0.01, 1), exonlevel = "core")
> xps.mp.tc <- validData(expr.mp.tc)
```

while APT uses the following command:

```
apt-probeset-summarize -a pm-only,med-polish.expon=true \
  -p HuGene-1_0-st-v1.r3.pgf -c HuGene-1_0-st-v1.r3.clf \
  -s transcriptList.txt *.CEL
```

The resulting Figure 8 shows that the results obtained with xps and APT are identical (the difference being less than 4e-6, see scale of y-axis).

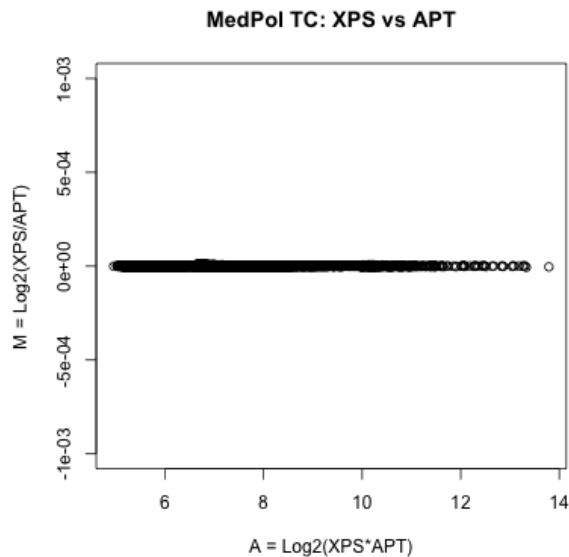


Figure 8: Median-polish only, computed with xps and apt, respectively.

Since the results of median-polish are identical for `xps` and `APT`, the differences seen in Figure 7 are due to background correction and/or quantile normalization, respectively. Package `xps` uses only those probes to compute background and quantile normalization, which are also used for summarization, as defined by parameter `exonlevel`. The reason for this is that RMA estimates the background from the PM probe intensities only, as described in (Irizarry et al., 2003). Analogously, quantile normalization (Bolstad et al., 2003) is applied only to the probes which are used for summarization.

It seems that `APT` uses (almost) all probes on the array for background correction and quantile normalization. Since package `xps` has an (unsupported) option to use (almost) all probes (which include internal control probes and background probes), there is a possibility to use all probes by setting parameter `exonlevel` individually for background correction, quantile normalization and median-polish summarization, respectively. This is done as follows:

```
> data.rma.tc.bq99 <- rma(data.genome, "HuGeneMixRMAbgqu99mp9",
+   background = "antigenomic", normalize = TRUE, exonlevel = c(992316,
+   992316, 9216))
> xps.rma.tc.bq99 <- validData(data.rma.tc.bq99)
```

Here, we have set `exonlevel="core+affx+unmapped+exon+intron+antigenomic"` for background correction and quantile normalization, respectively, and `exonlevel="core"` for median-polish summarization (see `?exonLevel`). Comparing the results of this setting to `APT`, we obtain the following "MvA-plot" for tissue "breast_A", see Figure 9.

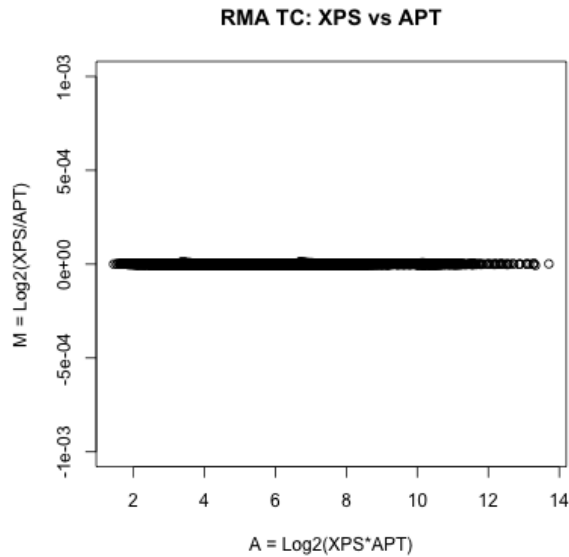


Figure 9: RMA, computed with xps and apt, respectively.

As Figure 9 shows, the expression levels obtained when using `xps` or `APT`, respectively, are now identical, indicating that `APT` does indeed use all probes on the array for background correction and quantile normalization (excluding only the probes hybridizing to the control oligo).

3.2 Comparison of the DABG call implementations

Now let us compute the detected above background (DABG) calls. The command-line for `APT` is:

```
apt-probeset-summarize -a dabg \
  -p HuGene-1_0-st-v1.r3.pgf -c HuGene-1_0-st-v1.r3.clf \
  -b HuGene-1_0-st-v1.r3.bgp -s probesetList.txt -x 8 *.CEL
```

Using package `xps` DABG calls are computed as follows:

```
> call.dabg.tc <- dabg.call(data.genome, "HuGeneMixDABGcore_tc",
+   option = "transcript", exonlevel = "core")
> xps.pval.tc <- validData(call.dabg.tc)
```

Figure 10 shows the difference in the detection p-values obtained with `xps` and `APT`, respectively. The detection p-values are identical (the minor apparent difference only due to different numbers of digits exported).

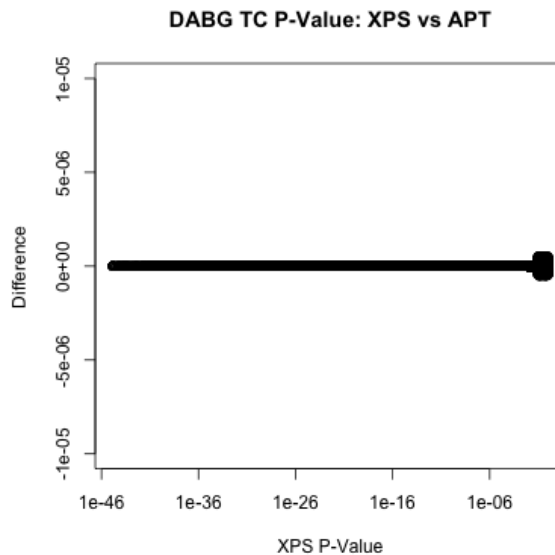


Figure 10: Comparison of p-values obtained with xps and APT.

3.3 Comparison of results when using HuGene as exon array

With release r4 of the library files Affymetrix has converted the HuGene-1_0-st-v1 array into an exon array. While the probes for "HuGene-1_0-st-v1.r3.pgf" were grouped according to the `transcript_cluster_ids`, the probes for "HuGene-1_0-st-v1.r4.pgf" are grouped according to the `probeset_ids` of the new probe-set annotation file.

Thus in order to compare xps and APT at the transcript level it is now necessary to create the transcript probesets for APT using function `metaProbesets()`:

```
> writeLines(rownames(xps.rma.tc), "core.txt")
> metaProbesets(scheme.genome, "core.txt", "coreList.mps", "core")
```

Now we need to use this 'coreList.mps' file to compute RMA and DABG, respectively, at the transcript level using APT, e.g. we compute RMA as follows:

```
apt-probeset-summarize -a rma-bg,quant-norm.sketch=0.usepm=true.bioc=true, \
  pm-only,med-polish.expon=true \
  -p HuGene-1_0-st-v1.r4.pgf -c HuGene-1_0-st-v1.r4.clf \
  -m coreList.mps *.CEL
```

The results for RMA and DABG are identical to the results obtained with release r3 of the library files shown above.

In addition to the transcript level release r4 allows now to compute both RMA and DABG on the probeset level as well, analogously to the exon arrays. For this purpose we compute first RMA at the probeset level using package xps:

```
> data.rma.ps <- rma(data.genome, "MixHuGeneRMACore_ps", background = "antigenomic",
+   normalize = TRUE, option = "probeset", exonlevel = "core")
> xps.rma.ps <- validData(data.rma.ps)
```

Then we create a file specifying the probesets to summarize for APT:

```
> tmp <- as.data.frame(rownames(xps.rma.ps))
> colnames(tmp) <- "probeset_id"
> write.table(tmp, "probesetList.txt", quote = FALSE, row.names = FALSE)
```

Finally we use this probesetList to compute RMA using APT:

```
apt-probeset-summarize -a rma-bg,quant-norm.sketch=0.usepm=true.bioc=true, \
  pm-only,med-polish.expon=true \
  -p HuGene-1_0-st-v1.r4.pgf -c HuGene-1_0-st-v1.r4.clf \
  -s probesetList.txt *.CEL
```

The resulting figure 11 shows once again a difference of few percent in the expression levels obtained when using xps or APT.

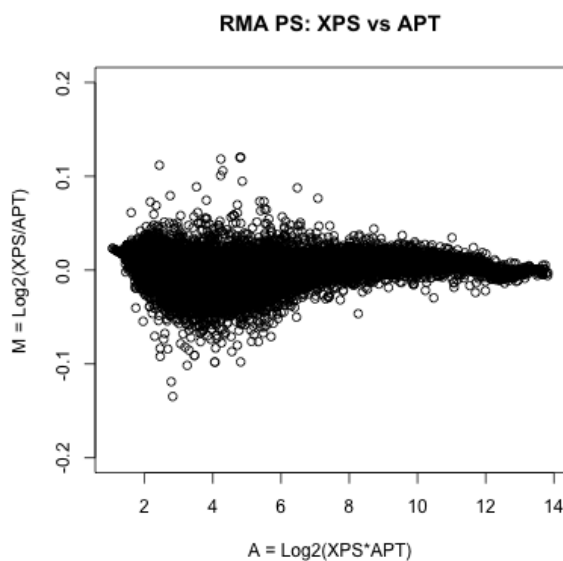


Figure 11: RMA for probesets computed with xps and apt, respectively.

However, when using the same probes for background correction and quantile normalization as APT:

```
> data.rma.ps.bq99 <- rma(data.genome, "HuGeneMixRMAbgqu99mp9_ps",
+   background = "antigenomic", normalize = TRUE, option = "probeset",
+   exonlevel = c(992316, 992316, 9216))
> xps.rma.ps.bq99 <- validData(data.rma.ps.bq99)
```

the results are once again identical to the results obtained with APT, as shown in Figure 12.

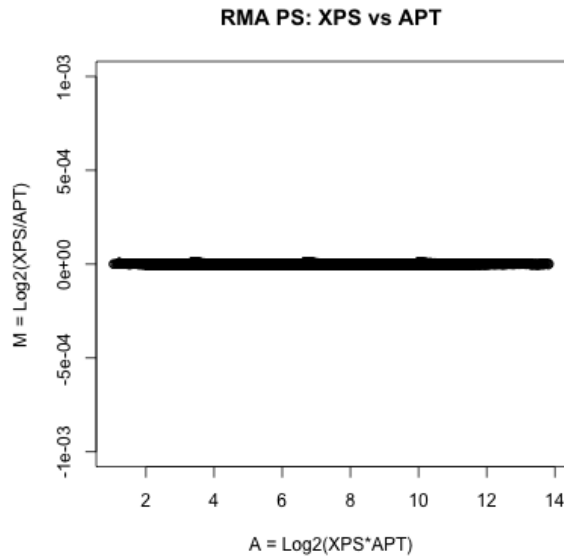


Figure 12: RMA for probesets computed with xps and apt, respectively.

4 Comparison of results for the Human Exon 1.0 ST Array

4.1 Comparison of RMA summarization at the probeset level

As already mentioned, comparison of xps and APT with exon arrays requires the definition of common probesets first.

Thus we compute first RMA at the probeset level using package xps:

```
> data.rma.ps <- rma(data.exon, "MixHuExonRMACore_ps", background = "antigenomic",
+   normalize = TRUE, option = "probeset", exonlevel = "core")
> xps.rma.ps <- validData(data.rma.ps)
```

Then we can create a file specifying the probesets to summarize for APT:

```
> tmp <- as.data.frame(rownames(xps.rma.ps))
> colnames(tmp) <- "probeset_id"
> write.table(tmp, "corePSList.txt", quote = FALSE, row.names = FALSE)
```

Finally we use this probesetList to compute RMA using APT:

```
apt-probeset-summarize -a rma-bg,quant-norm.sketch=0.usepm=true.bioc=true, \
  pm-only,med-polish.expon=true \
  -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 -s corePSList.txt *.CEL
```

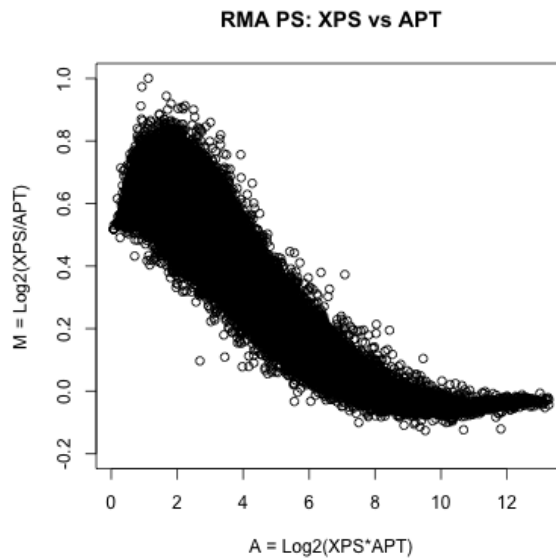


Figure 13: RMA computed with xps and apt, respectively.

As the resulting "MvA-plot" for tissue "breast_A", shown in Figure 13, reveals, there is quite some difference for the probeset levels computed with xps or APT, respectively. Thus let us compare the "median-polish" summarization step only, as done before for the gene array.

In package xps this is done as follows:

```
> expr.mp.ps <- express(data.exon, "HuExonMedPolcorePS", summarize.method = "medianpolish",
+   summarize.select = "pmonly", summarize.option = "probeset",
+   summarize.logbase = "log2", summarize.params = c(10, 0.01,
+     1), exonlevel = "core")
> xps.mp.ps <- validData(expr.mp.ps)
```

while APT uses the following command:

```
apt-probeset-summarize -a pm-only,med-polish.expon=true \
  -p HuEx-1_0-st-v2.r2.pgff -c HuEx-1_0-st-v2.r2.clf \
  -s corePSList.txt *.CEL
```

The resulting Figure 14 shows that the results obtained with xps and APT are identical (the difference being less than 6e-6, see scale of y-axis).

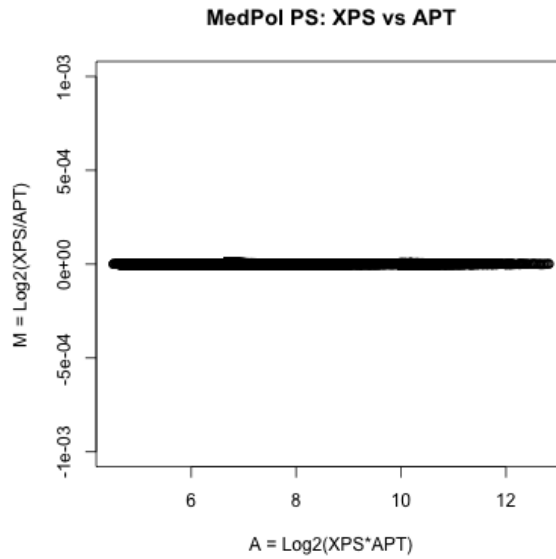


Figure 14: Median-polish only, computed with xps and apt, respectively.

As for gene arrays, the differences seen in Figure 13 are due to background correction and quantile normalization, respectively. This time we set parameter `exonlevel` individually to `exonlevel="all+genomic+antigenomic+intron+exon+unmapped"` for background correction and quantile normalization, respectively, and `exonlevel="core"` for median-polish summarization:

```
> data.rma.ps.bq103 <- rma(data.exon, "HuExonMixRMABgqu103mp9_ps",
+   background = "antigenomic", normalize = TRUE, option = "probeset",
+   exonlevel = c(1032124, 1032124, 9216))
> xps.rma.ps.bq103 <- validData(data.rma.ps.bq103)
```

Comparing the results of this setting to APT, we obtain the following "MvA-plot" for tissue "breast_A" shown in Figure 15.

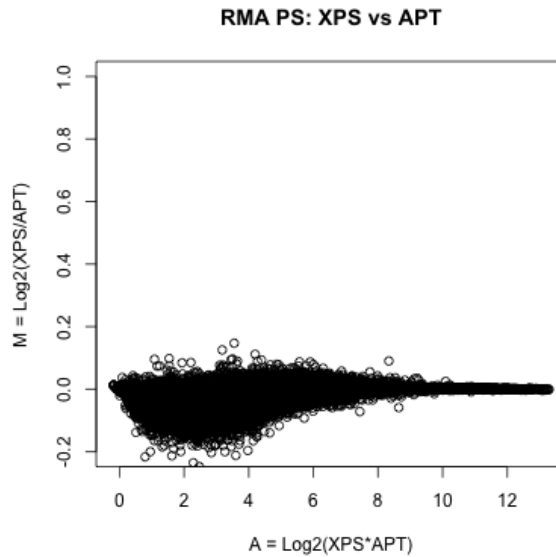


Figure 15: RMA, computed with xps and apt, respectively.

As Figure 15 shows, the differences in the expression levels obtained when using xps or APT, respectively, are dramatically reduced. At high expression levels there is no difference, while the difference at low expression levels is reduced to at most 20%. However, in contrast to the HuGene array the difference is still visible. Thus in this case it is not clear to me which probes Affymetrix is using for background correction and quantile normalization.

4.2 Comparison of RMA summarization at the transcript level

Comparing xps and APT with exon arrays at the gene level requires the definition of common probesets for each transcript, i.e. a file containing meta probeset definitions.

First we compute RMA at the transcript level using package xps:

```
> data.rma.tc <- rma(data.exon, "MixHuExonRMACore_tc", background = "antigenomic",
+   normalize = TRUE, option = "transcript", exonlevel = "core")
> xps.rma.tc <- validData(data.rma.tc)
```

Then we can create a meta probeset file using function metaProbesets():

```
> writeLines(rownames(xps.rma.tc), "core.txt")
> metaProbesets(scheme.exon, "core.txt", "coreList.mps", "core")
```

Now we use this 'coreList.mps' file to compute RMA using APT:

```
apt-probeset-summarize -a rma-bg,quant-norm.sketch=0.usepm=true.bioc=true, \
  pm-only,med-polish.exon=true \
  -p HuEx-1_0-st-v2.r2.pgff -c HuEx-1_0-st-v2.r2.clf \
  -m coreList.mps *.CEL
```

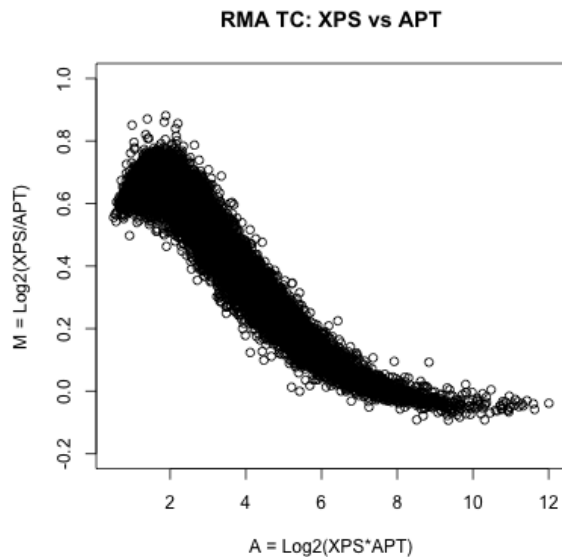


Figure 16: RMA computed with xps and apt, respectively.

As expected, the resulting "MvA-plot" for tissue "breast_A", shown in Figure 16, is similar to Figure 13. Again, we compare the "median-polish" summarization step only.

In package `xps` this is done as follows:

```
> expr.mp.tc <- express(data.exon, "HuExonMedPolcore", summarize.method = "medianpolish",
+   summarize.select = "pmonly", summarize.option = "transcript",
+   summarize.logbase = "log2", summarize.params = c(10, 0.01,
+     1), exonlevel = "core")
> xps.mp.tc <- validData(expr.mp.tc)
```

while APT uses the following command:

```
apt-probeset-summarize -a pm-only,med-polish.expon=true \
  -p HuEx-1_0-st-v2.r2.pgff -c HuEx-1_0-st-v2.r2.clf \
  -m coreList.mps *.CEL
```

As for the probesets, the resulting Figure 17 shows that the results obtained with `xps` and APT are identical (the difference being less than $6e-6$, see scale of y-axis).

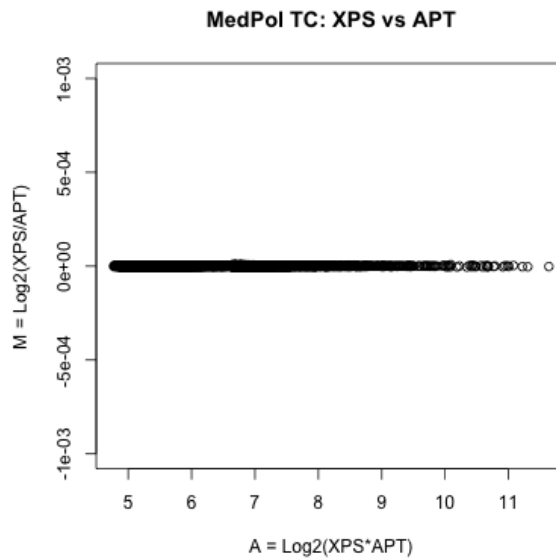


Figure 17: Median-polish only, computed with xps and apt, respectively.

When we set parameter `exonlevel` individually to `exonlevel="all+genomic+antigenomic+intron+exon+unmapped"` for background correction and quantile normalization, respectively, and `exonlevel="core"` for median-polish summarization:

```
> data.rma.tc.bq103 <- rma(data.exon, "HuExonMixRMAbgqu103mp9_tc",
+   background = "antigenomic", normalize = TRUE, option = "transcript",
+   exonlevel = c(1032124, 1032124, 9216))
> xps.rma.tc.bq103 <- validData(data.rma.tc.bq103)
```

and compare the results to APT, we obtain the "MvA-plot" for tissue "breast_A" shown in Figure 18.

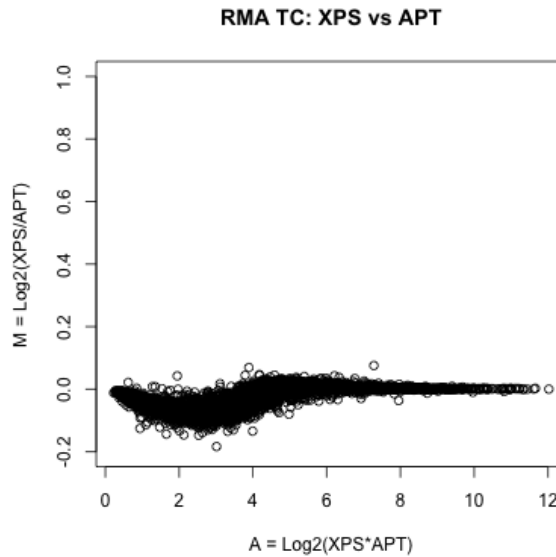


Figure 18: RMA, computed with xps and apt, respectively.

As Figure 18 shows, the differences in the expression levels obtained when using `xps` or `APT`, respectively, are also dramatically reduced. At high expression levels there is almost no difference, while the difference at low expression levels is reduced to less than 20%.

4.3 Comparison of the DABG call implementations

Now let us compare the detected above background (DABG) calls at the probeset level.

The command-line for `APT` is:

```
apt-probeset-summarize -a dabg \
  -p HuEx-1_0-st-v2.r2.pgfp -c HuEx-1_0-st-v2.r2.clf \
  -b HuEx-1_0-st-v2.r2.antigenomic.bgp -s corePSList.txt -x 12 *.CEL
```

while package `xps` computes DABG calls as follows:

```
> call.dabg.ps <- dabg.call(data.exon, "HuExonDABGcorePS", option = "probeset",
+   exonlevel = "core")
> xps.pval.ps <- validData(call.dabg.ps)
```

Figure 19 shows the difference in the detection p-values obtained with `xps` and `APT`, respectively. The detection p-values are identical (the minor apparent difference only due to different numbers of digits exported).

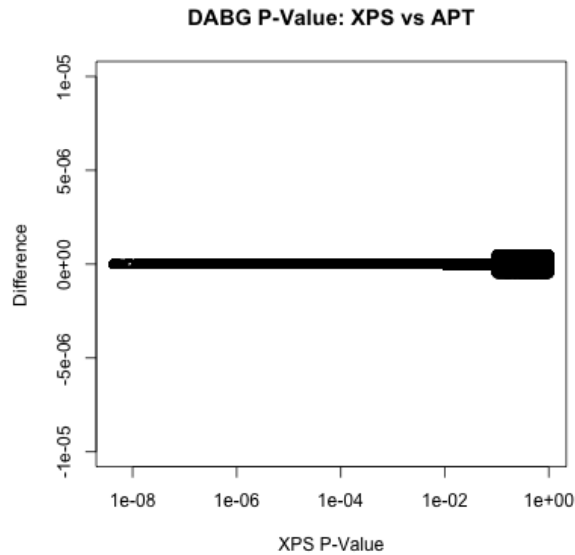


Figure 19: Comparison of p-values obtained with xps and APT.

Note: Another advantage of package `xps` compared to APT is, that you can compute DABG calls not only at the probeset level but also at the transcript level. This is simply done as follows:

```
> call.dabg.tc <- dabg.call(data.exon, "HuExonDABGcoreTC", option = "transcript",
+   exonlevel = "core")
> xps.pval.tc <- pvalData(call.dabg.tc)
```

5 Comparison of Affymetrix gene expression arrays

In this section we want to compare the gene expression levels obtained with arrays HG-U133_Plus_2, HuGene-1.0-st-v1 and HuEx-1.0-st-v2, respectively, for tissues breast and prostate. For this purpose we need to select first a subset of transcripts common to all three arrays. Then we want to find differentially expressed transcripts and compare the results for the normalized expression levels obtained with `xps` or APT, respectively.

5.1 Comparison of RMA summarization for common transcript probesets

In order to create a subset of transcripts common to all three arrays, we import as a first step the "Array Comparison Spreadsheets", which can be downloaded from the Affymetrix web-site:

```
> hx2hg <- read.delim("HuExVsHuGene_BestMatch.txt", row.names = 3)
> up2hx <- read.delim("U133PlusVsHuEx_BestMatch.txt", row.names = 3)
> up2hg <- read.delim("U133PlusVsHuGene_BestMatch.txt", row.names = 3)
```

Since we want to compare the RMA normalized expression levels `xps.urma` (HG-U133_Plus_2), `xps.grma` (HuGene-1.0-st-v1) and `xps.xrma` (HuEx-1.0-st-v2), with `xps.grma` and `xps.xrma` using only the "core" transcript probesets, we obtain the common subset `core`, containing unique exon and genome transcripts identical to U133P2 as follows:

```

> hx2hg <- uniqueframe(hx2hg)
> up2hx <- uniqueframe(up2hx)
> up2hg <- uniqueframe(up2hg)
> u2x <- intersectframes(up2hx, up2hg)
> u2g <- intersectframes(up2hg, up2hx)
> tmp <- intersectrows(u2gx, xps.grma, 1, NULL)
> core <- intersectrows(tmp, xps.xrma, 2, NULL)

```

The resulting subset `core` consists of 11033 transcripts common to all arrays.

Note: The full source code and the functions `uniqueframe`, `intersectframes` and `intersectrows` are described in file "script4bestmatch.R" located in directory "examples".

Now we obtain the following subsets of normalized expression data:

```

> xpsu <- intersectrows(xps.urma, core, NULL, NULL)
> xpsg <- intersectrows(xps.grma, core, NULL, 1, -1)
> xpsx <- intersectrows(xps.xrma, core, NULL, 2, -1)
> aptg <- intersectrows(apt.grma, core, NULL, 1, -1)
> aptx <- intersectrows(apt.xrma, core, NULL, 2, -1)

```

where `aptg` and `aptx` are the corresponding data for APT (w/o `aptu` since it is identical to `xpsu`).

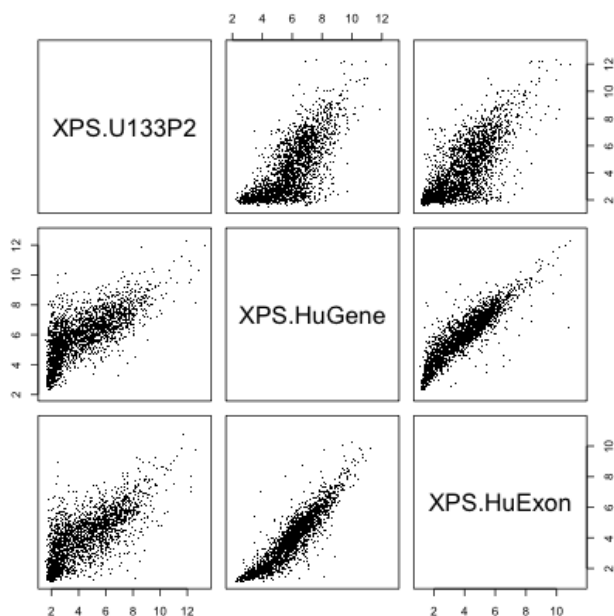


Figure 20: Scatter plot of mean expression summaries for breast.

A scatter plot of the averaged gene-level summaries for breast tissue is shown in Figure 20 for the data obtained with package `xps`. The corresponding figure for the data obtained with APT is almost identical. Furthermore, Figure 20 is very similar to the supplementary figure (page 10) published by Robinson & Speed in (Robinson and Speed, 2007), however, using the averaged gene-level summaries for one particular mixture, and Ensembl gene-centric probesets.

5.2 Differentially expressed transcripts in breast vs prostate

Now we consider the task of finding differentially expressed genes between the breast and prostate samples. Following Robinson & Speed (Robinson and Speed, 2007) we will use the moderated t-statistics from package *limma* to compute raw p-values, which are then adjusted by a Benjamini–Hochberg correction (the default setting).

For this purpose we create the design matrix:

```
> tissue <- c(rep("Breast", 3), rep("Prostate", 3))
> design <- model.matrix(~factor(tissue))
> colnames(design) <- c("Breast", "BreastvsProstate")
```

which we use to compute the moderated t-statistics for each platform, e.g. for data *xpsu*:

```
> tmp <- as.matrix(log2(xpsu))
> fit <- lmFit(tmp, design)
> fit <- eBayes(fit)
> xpsu.lm <- topTable(fit, coef = 2, n = length(rownames(tmp)),
+   adjust = "BH")
```

Now we can first compare the fold-change values for the different platforms.

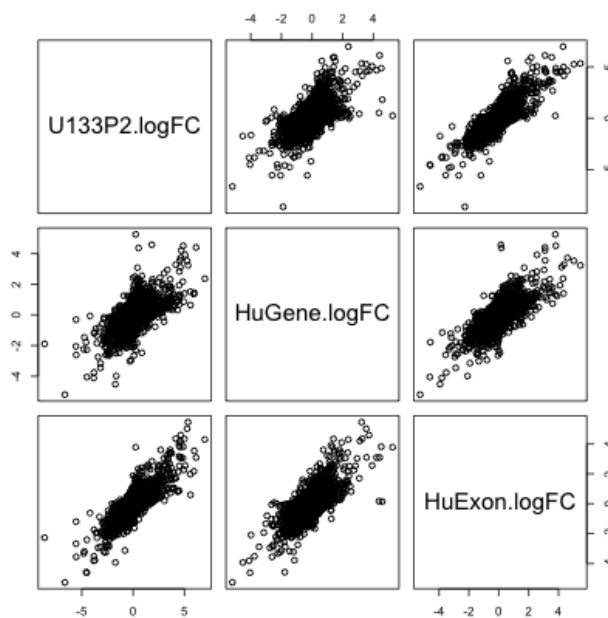


Figure 21: Scatter plot of fold-change values.

Figure 21 shows a scatter plot of the fold-change values for the data obtained with package *xps*. The corresponding figure for the data obtained with APT is (almost) identical.

Regarding the computed p-values, let us first compare the p-values for the HuGene-1_0-st-v1 data `xpsg` and `aptg`, obtained with `xps` or APT, respectively.

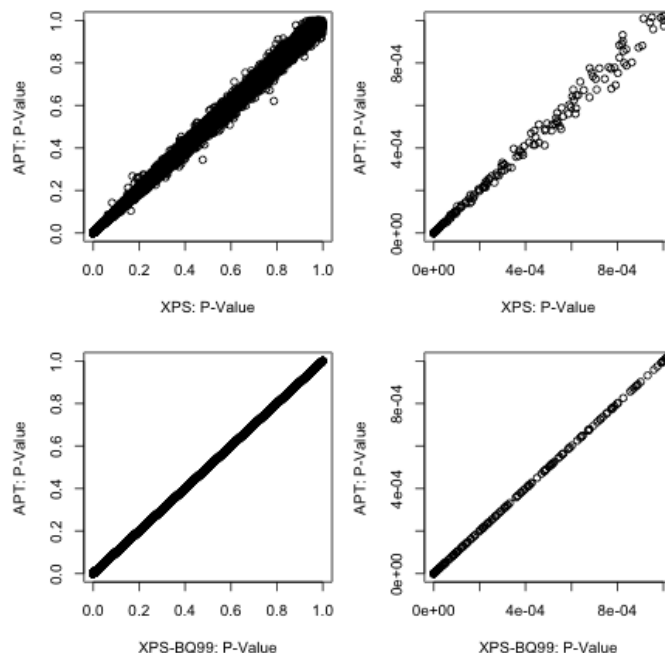


Figure 22: Scatter plot of p-values obtained with `xpsg`, `aptg`, and `bq99g`.

The upper rows of Figure 22 compare the p-values obtained with `xpsg` and `aptg`, respectively, whereby the right scatter-plot compares the lowest p-values only, i.e. the most relevant p-values. The lower rows of Figure 22 compare the p-values obtained with `bq99g` and `aptg`, respectively, where `bq99g` is identical to `aptg`, as shown in Figure 9.

Comparing the upper and lower scatter-plots reveals that the differences between `xps` and APT, seen in Figure 7, may not have a large effect when computing moderated t-statistics. The same is true when comparing the adjusted p-values. However, the differences between `xpsg` and `aptg` are only a few percent, as seen in Figure 7, thus it is not unexpected that the (adjusted) p-values are very similar.

In contrast, the differences between the expression levels obtained with array HuEx-1_0-st-v2, i.e. `xpsx` and `aptx`, respectively, are quite pronounced, as demonstrated earlier (see Figure 16). Thus it can be expected that these differences are also reflected in differences in p-values, when computing the moderated t-statistics for `xpsx` or `aptx`, respectively. As shown in Figure 23, this is indeed the case.

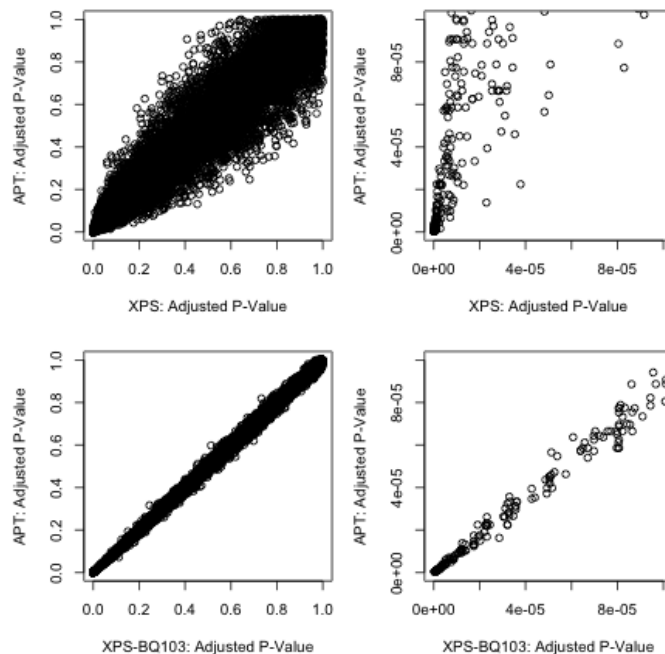


Figure 23: Scatter plot of adjusted p-values obtained with `xpsx`, `aptx`, and `bq103x`.

In this case, the upper rows of Figure 23 compare the BH-adjusted p-values obtained with `xpsx` and `aptx`, respectively, whereby the right scatter-plot compares once again the lowest p-values only, i.e. the most relevant p-values.

This time, the (adjusted) p-values computed for `xpsx` and `aptx`, respectively, show clearly are larger variance. Of special interest is the upper right part of the figure, which compares only the the most relevant p-values. It is evident, that the relevant adjusted p-values computed for `xpsx` are smaller than the corresponding p-values computed for `aptx` (as the slope of an imaginary fitted line would show). This indicates that the expression levels `xpsx` obtained with package `xps` result in a better separation between tissues breast and prostate for the most relevant genes than the expression levels `aptx` obtained with APT.

As described earlier, we have also adjusted background correction and quantile normalization to resemble the APT expression levels (see Figure 18), resulting in subset `bq103x`. Comparing the adjusted p-values obtained with these expression levels (`bq103x`), with the p-values obtained with subset `aptx` results in the scatter plot shown in the lower part of Figure 23. This part of the figure looks similar to Figure 22, and shows that the adjusted p-values for `bq103x` and `aptx` are very similar, although there is still a difference of about 20% at low expression levels (Figure 18). However, in both cases the relevant p-values are larger than the p-values obtained with subset `xpsx`. In my opinion, this is a clear indication that only those probes should be used for background correction and quantile normalization, which are also used for median-polish summarization.

6 Summary

Comparison of packages `xps` and `affy` with APT for expression arrays shows that the different implementations of the RMA algorithm result in identical expression levels, as shown in Figure 3. (The negligible difference of less than $5e-6$ is due to the fact that package `xps` does all computations with

double-precision (64bit), but saves the results as floats (32bit)).

Interestingly, all three implementations of the MAS5 algorithm (`xps`, `affy`, `APT`) differ from the original GCOS implementation, with `xps` probably being slightly more close to GCOS than `affy` (Figure 4), but I may be biased.

The implementations of MAS5 detection calls are identical for `xps` and `affy`, but differ slightly from the `APT` implementation which is identical to the GCOS implementation (Figure 5).

For the gene and exon arrays I have only compared `xps` and `APT`, which both use the new CLF-files and PGF-files as library files, and are able to use the separate probeset groups 'core', 'extended' and/or 'full', which are defined in the Affymetrix annotation files.

Although the `xps` and `APT` implementations of the RMA algorithm give identical results for expression arrays, the results for both gene arrays and exon arrays seem to differ, as shown in Figures 7 and 16, respectively. As we have demonstrated earlier, this difference is caused by using different probes for background correction and quantile normalization, respectively, since using identical probes (HuGene) or applying only median-polish summarization gives identical results (Figures 9, 8 and 17).

While package `xps` uses by default the identical "PM" probes for all three steps of the RMA algorithm, e.g. for `exonlevel="core"` only probes belonging to the 'core' probesets defined in the Affymetrix annotation files are used, `APT` uses (almost) all probes on the respective array. Thus `xps` follows the implementation of RMA for expression arrays, while `APT` differs in the probes usage at the different steps.

In order to determine whether this difference in the probes usage may have an influence on subsequent expression analysis steps, I have applied moderated t-statistics from package `limma` to find differentially expressed genes between tissues breast and prostate. Computing raw and adjusted p-values for the gene array data results in almost identical p-values, indicating that the small difference seen in Figure 7 does not affect further processing. In contrast, computing raw and adjusted p-values for the exon array data reveals that the `xps` default usage of identical probes for RMA normalization results in lower (better) p-values in the relevant range to select genes of interest.

The implementation of the DABG call algorithm in `xps` is identical to `APT`, however for exon arrays, `xps` allows to apply DABG at both the probeset level as well as the transcript level.

In summary, package `xps` offers the following advantages compared to `APT`:

- full quantile normalization can be done on computers with 1GB RAM only.
- for RMA the same probes are used by default for background correction, quantile normalization and median-polish summarization for all three types of arrays.
- in addition, for gene and exon arrays it is possible to select probes to be used for background correction, quantile normalization and median-polish summarization, respectively, independently.
- MAS5 detection calls can be applied to all three types of arrays.
- DABG calls can be applied to all three types of arrays.
- for exon arrays DABG can be applied at the probeset level and the transcript level, respectively.

A Appendices

A.1 Some notes on PLIER

As an alternative to RMA, Affymetrix has developed a new algorithm, called PLIER: "The PLIER (Probe Logarithmic Error Intensity Estimate) method produces an improved signal by accounting for experimentally observed patterns in feature behavior and handling error at the appropriately at low and high signal values." (according to the APT manual)

Using the HG-U133_Plus_2 data set, the command-line for APT is:

```
apt-probeset-summarize -a plier-mm -d HG-U133_Plus_2.cdf *.CEL
```

Now we import the results into R for visualization:

```
> apt.plier <- read.delim("plier-mm.summary.txt", row.names = 1,  
+   comment.char = "", skip = 52)
```

There are many methods to visualize the quality of the normalization step, often based on advanced statistical methods. However, often a simple visualization, e.g. a simple scatter-plot is already very helpful. Thus let us create a scatter-plot of the normalized data for the replicated breast tissue:

```
> plot(apt.plier[, 1], apt.plier[, 2], main = "APT: PLIER", xlab = "Breast_A",  
+   ylab = "Breast_B", log = "xy")
```

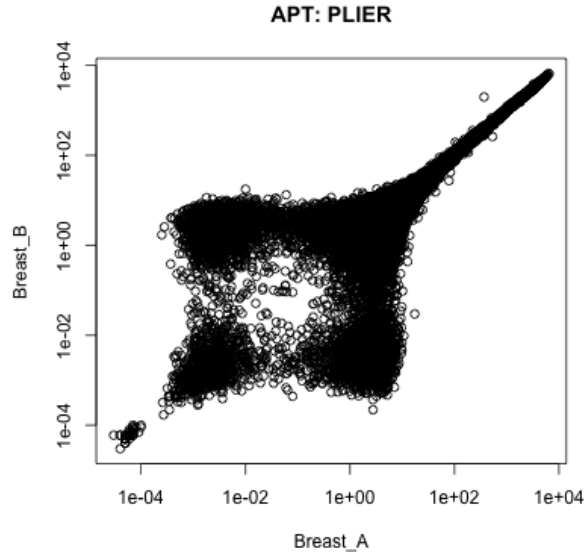


Figure 24: Scatter-plot.

For comparison purposes, we create also "MvA-plots" for "Breast_A" vs "Breast_B" using either PLIER or RMA for normalization:

```

> plot(log2(apt.plier[, 1] * apt.plier[, 2])/2, log2(apt.plier[,
+   1]/apt.plier[, 2]), main = "APT: PLIER", xlab = "A = Log2(BrA*BrB)",
+   ylab = "M = Log2(BrA/BrB)", log = "")
> plot(log2(apt.rma[, 1] * apt.rma[, 2])/2, log2(apt.rma[, 1]/apt.rma[,
+   2]), main = "APT: RMA", xlab = "A = Log2(BrA*BrB)", ylab = "M = Log2(BrA/BrB)",
+   log = "", ylim = c(-10, 10))

```

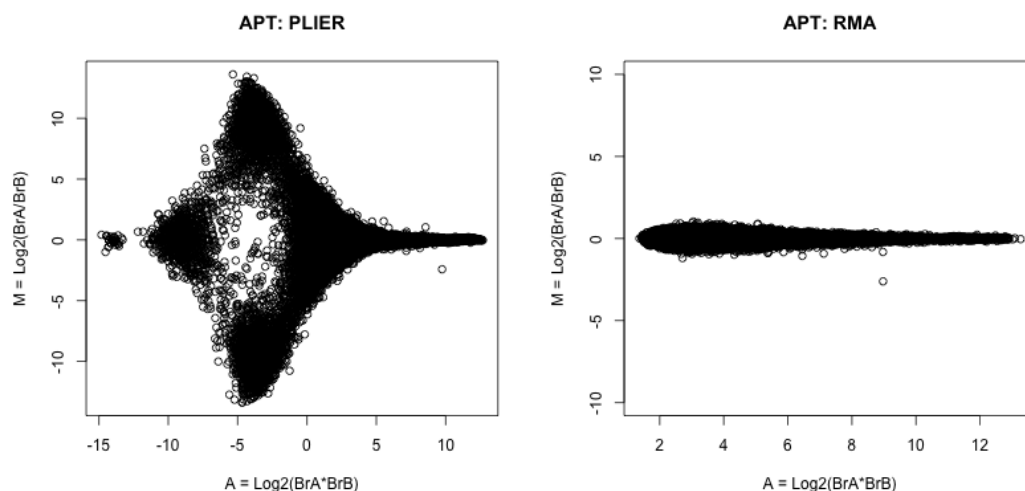


Figure 25: MvA-plot using PLIER (left) or RMA (right).

As Figure 25 shows, the differences between PLIER and RMA are dramatically.

Please note that the following is only my personal subjective opinion:

As both Figure 24 and Figure 25 reveal, PLIER produces severe artifacts, even for replicate data, which are supposed to have very similar expression values. Probably other people will not agree with me, however for this reason I never use PLIER, and this is one of the reasons why I have decided not to implement PLIER in package *xps*.

References

- B.M. Bolstad, R.A. Irizarry, M. Åstrand, and T.P. Speed. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19(2):185–193, Jan 2003.
- Rafael A. Irizarry, Bridget Hobbs, Francois Collin, Yasmin D. Beazer-Barclay, Kristen J. Antonellis, Uwe Scherf, and Terence P. Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, 2003.
- Mark Robinson and Terence Speed. A comparison of affymetrix gene expression arrays. *BMC Bioinformatics*, 8(449):1–16, Nov 2007.