Introduction to the xps Package: Function `express()`

Christian Stratowa

September, 2010

Contents

1 Introduction

This document describes how to use function `express` to combine different preprocessing methods, which are currently available in package `xps` i.e. methods for (i) background correction, (ii) probe-level normalization, and (iii) summarization. These different methods, or a subset thereof, can be computed with a single call to function `express`, or in a stepwise manner by calling `express` consecutively with the result of one call used as input for the next call. Applying `express` stepwise has certain advantages, which will be explained later. Since `express` is a wrapper function to method `xpsPreprocess` both function calls can be used.

In addition to the mentioned preprocessing steps it is sometimes advisable to add a further preprocessing step after the summarization step, i.e. (iv) probeset-level normalization. One example is the MAS5 algorithm where the expression measures are usually scaled to a predefined target intensity (**?**). This additional step is handled by function `normalize`, which is a wrapper to method `xpsNormalize`.

Using a subset of the Affymetrix Exon Array Data Set "Tissue Mixture" for the expression array "Human Genome U133 Plus 2.0", examples will be presented for all available algorithms for the different preprocessing methods. The aim of this document is to explain all parameters for all methods in detail based on the examples presented.

Note: In order to keep this document short, only example code will be presented here, the full source code is available in file "script4xpsPreprocess.R" located in directory "xps/examples".

2 Computing RMA using function `express()`

In the following subsections we will demonstrate how to compute RMA using the general function `express`. However, first it is necessary to load the `ROOT` *scheme* and `ROOT` *data* files, as explained in vignette "xps.pdf" (Overview):

```
> scheme.u133p2 <- root.scheme(file.path(scmdir, "Scheme_HGU133p2_na28.root"))
> data.u133p2   <- root.data(scheme.u133p2, file.path(datdir, "HuTissuesU133P2_cel.root"))
```

Usually, RMA is computed with a simple call to function `rma`:

```
> data.rma <- rma(data.u133p2, "MixU133P2RMA", filedir=outdir)
```

2.1 Compute RMA with a single call to express

Using a single call to function express RMA is computed as follows:

```
> expr.rma <- express(data.u133p2, "U133P2Exprs", filedir=outdir, tmpdir="", update=FALSE,
+             bgcorrect.method="rma", bgcorrect.select="none", bgcorrect.option="pmonly:epanechnikov",
+             bgcorrect.params=c(16384), normalize.method="quantile", normalize.select="pmonly",
+             normalize.option="transcript:together:none", normalize.logbase="0",
+             normalize.params=c(0.0), summarize.method="medianpolish", summarize.select="pmonly",
+             summarize.option="transcript", summarize.logbase="log2", summarize.params=c(10, 0.01, 1.
```

Valid expression levels can be obtained by calling

```
> expr  <- validExpr(expr.rma)
```

2.2 Compute RMA stepwise

Alternatively, function express can be used to compute RMA step-by-step. This can have the following advantages:

- different algorithms can be tested at each step and combined w/o the need to re-calculate the earlier step(s)
- quality control tests for each array can be done separately at each step
- stepwise computation is useful when computing RMA for 10,000 CEL-files or more

1. step: background correction for *DateTreeSet* data.u133p2:

The code for computing rma background correction is as follows:

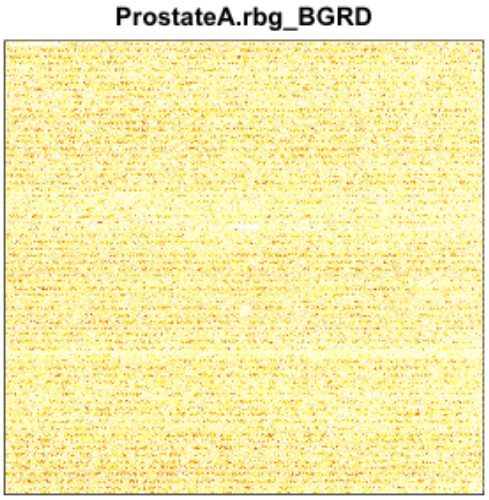
```
> bgrd.rma <- express(data.u133p2, "BgrdRMA", filedir=outdir, tmpdir="", update=FALSE,
+             bgcorrect.method="rma", bgcorrect.select="none", bgcorrect.option="pmonly:epanechnikov",
+             bgcorrect.params=c(16384))
```

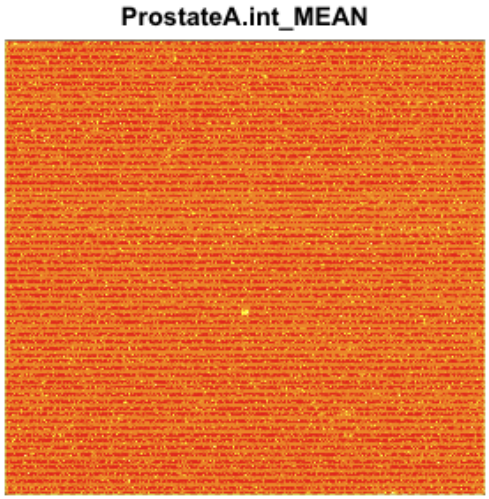
Note: For background correction is is not allowed to define a tmpdir, otherwise all trees would be stored in a temporary ROOT file and file BgrdRMA.root would be empty.

Now we can do some quality controls such as drawing images, density plots and boxplots.

First we draw the images for background values and background-corrected intensities, respectively, for e.g. array "ProstateA":

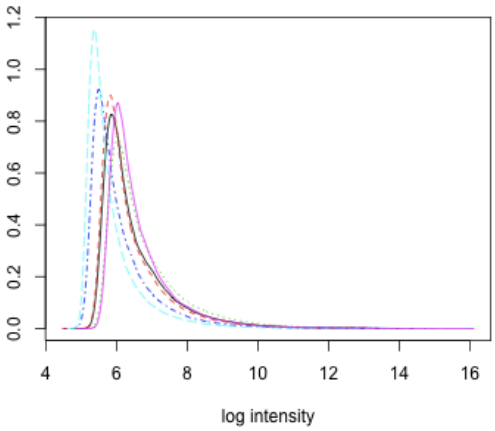
```
> bgrdnames <- colnames(validBgrd(bgrd.rma))
> datanames <- colnames(validData(bgrd.rma))
> image(bgrd.rma, bg=TRUE, transfo=log2, col=heat.colors(12), names=bgrdnames[4])
> image(bgrd.rma, transfo=log2, col=heat.colors(12), names=datanames[4])
```

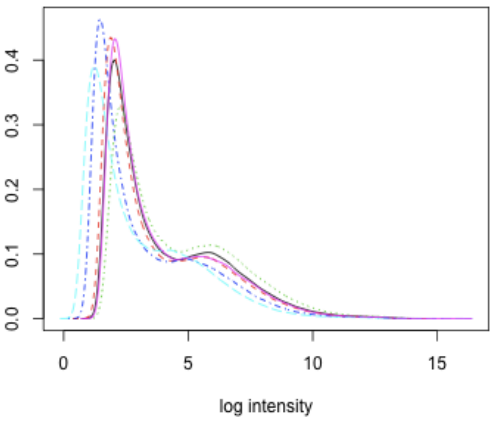




Then we draw the density plots for raw and background-corrected intensities, respectively:

```
> hist(data.u133p2, which="pm")
> hist(bgrd.rma, which="pm")
```



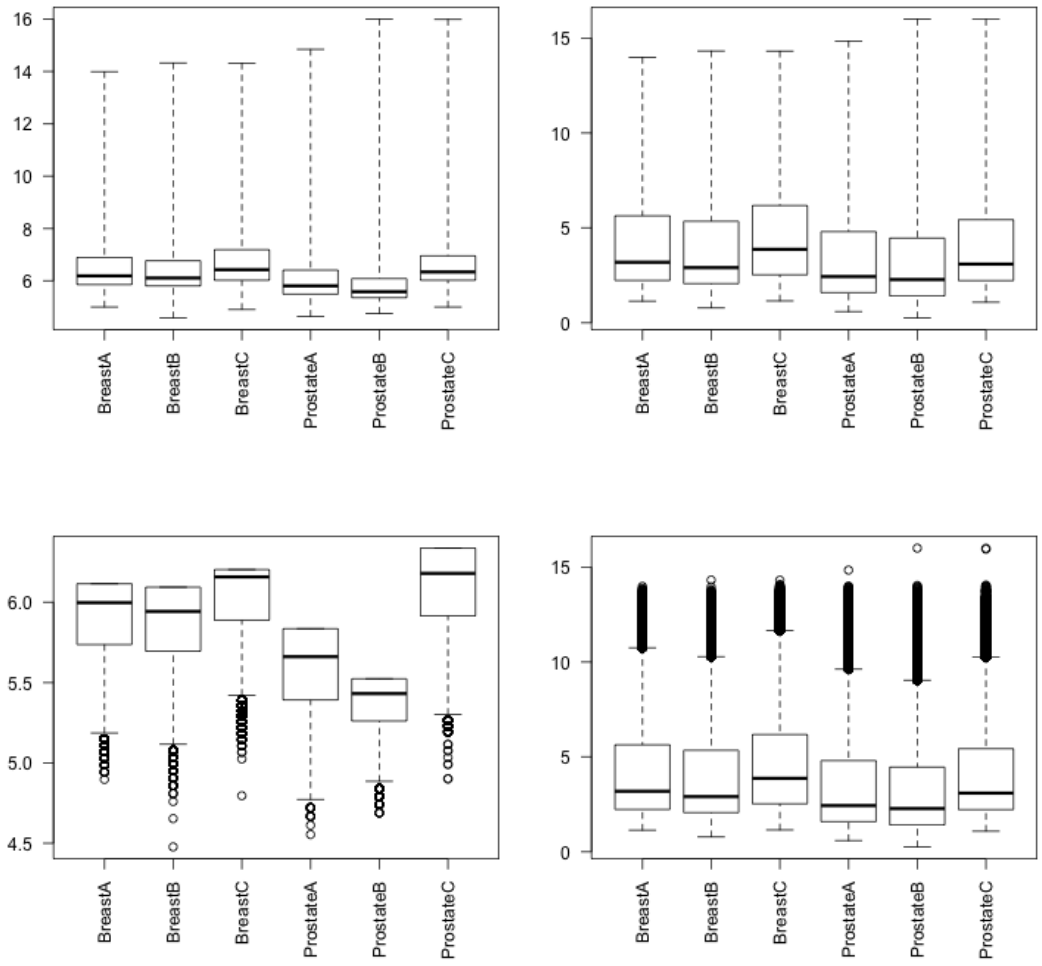


Finally we draw the boxplots for raw and background-corrected intensities, respectively:

```
> boxplot(data.u133p2, which="pm")
> boxplot(bgrd.rma, which="pm")
```

Alternatively, it is also possible to draw boxplots for background values and background-corrected intensities, respectively, as follows:

```
> bgrd <- validBgrd(bgrd.rma, which="pm")
> data <- validData(bgrd.rma, which="pm")
> boxplot(log2(bgrd), las=2)
> boxplot(log2(data), las=2)
```



2. step: quantile normalization for resulting *DateTreeSet* `bgrd.rma`:

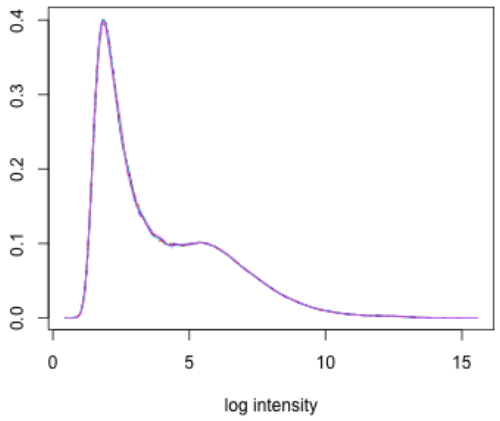
Using the background-corrected data `bgrd.rma` as input the code for computing `quantile` normalization is as follows:

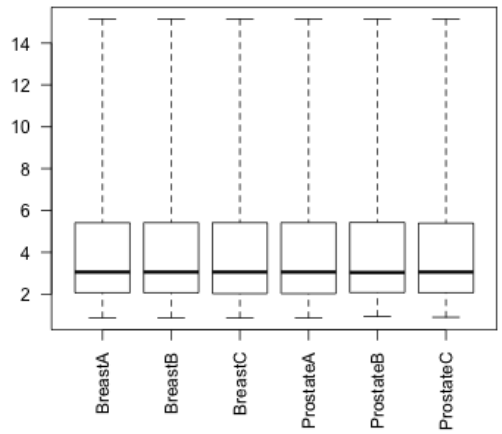
```
> norm.qu <- express(bgrd.rma, "NormQuan", filedir=outdir, tmpdir="", update=FALSE,
+             normalize.method="quantile", normalize.select="pmonly",
+             normalize.option="transcript:together:none", normalize.logbase="0",
+             normalize.params=c(0.0))
```

Note: For the normalization step is is also not allowed to define a `tmpdir`, otherwise all trees would be stored in a temporary `ROOT` file and file `NormQuan.root` would be empty.

Now we draw density plots and boxplots of the normalized intensities as quality controls:

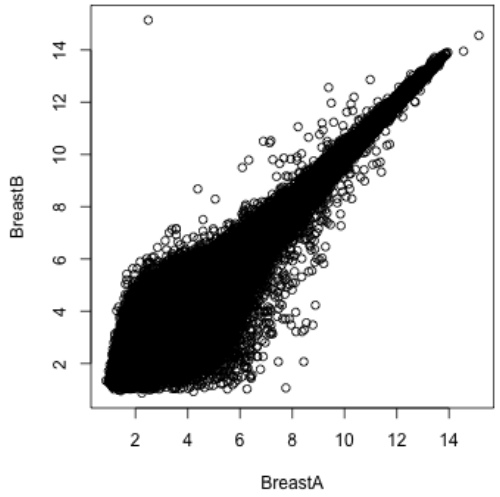
```
> hist(norm.qu, which="pm")
> boxplot(norm.qu, which="pm")
```

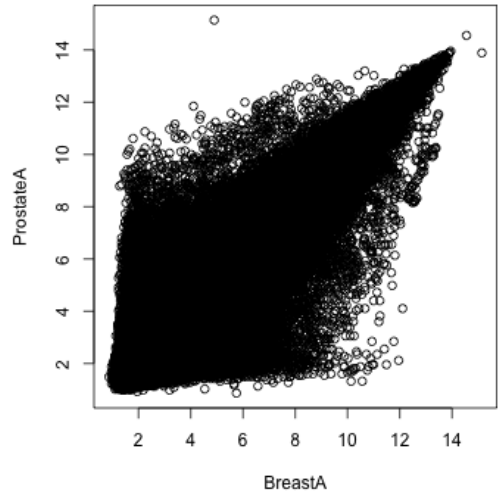




In addition we can draw scatter plots between normalized intensities at the probe-level:

```
> data <- validData(norm.qu, which="pm")
> plot(log2(data[,1]), log2(data[,2]), xlab="BreastA", ylab="BreastB")
> plot(log2(data[,1]), log2(data[,4]), xlab="BreastA", ylab="ProstateA")
```





3. step: median-polish summarization for resulting *DateTreeSet* `norm.qu`:

Finally we do `medianpolish` summarization, using the normalized data `norm.qu` as input:

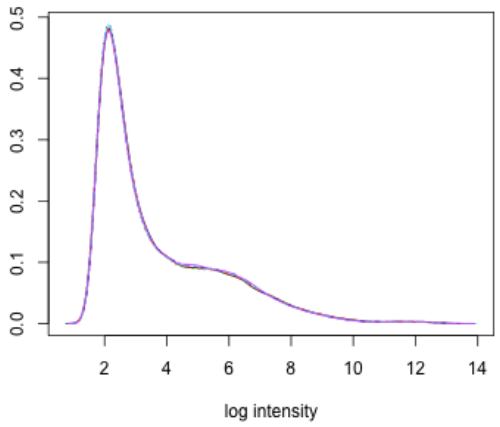
```
> expr.mp <- express(norm.qu, "ExprMedpol", filedir=outdir, tmpdir="", update=FALSE,
+             summarize.method="medianpolish", summarize.select="pmonly",
+             summarize.option="transcript", summarize.logbase="log2",
+             summarize.params=c(10, 0.01, 1.0), bufsize=32000)
```

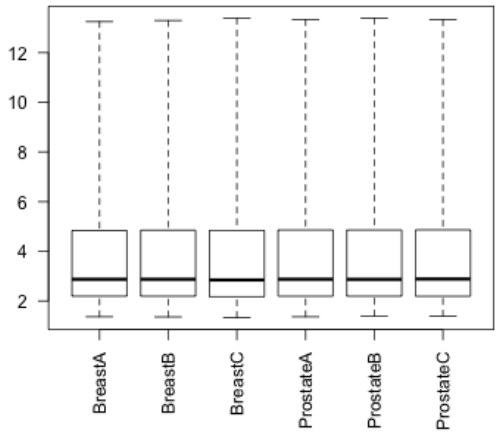
Note 1: For the summarization step is is possible to define a `tmpdir`; when handling hundreds of trees on computers with 1 GB RAM only, it is even the recommended way for multichip summarization methods such as `medianpolish`.

Note 2: When using a multichip summarization method with data consisting of many thousand trees it could also be necessary to reduce the basket size of each `ROOT` tree, e.g. to a size of `bufsize=2000`. (Internally, each `ROOT` tree consists of many tree baskets, and when reading a tree, one basket after the other is loaded into memory and not the whole tree, thus reducing memory consumption.)

Now we can draw density plots and boxplots of the final expression values:

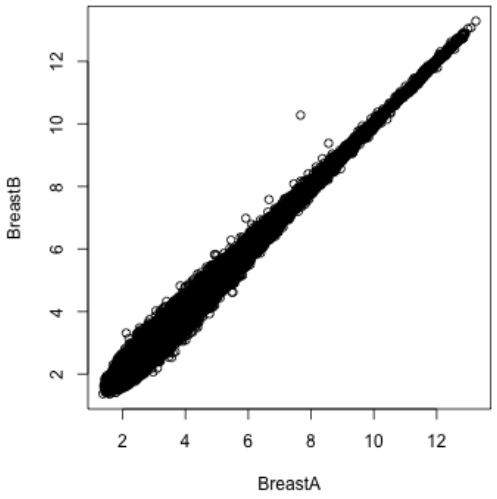
```
> hist(expr.mp)
> boxplot(expr.mp)
```

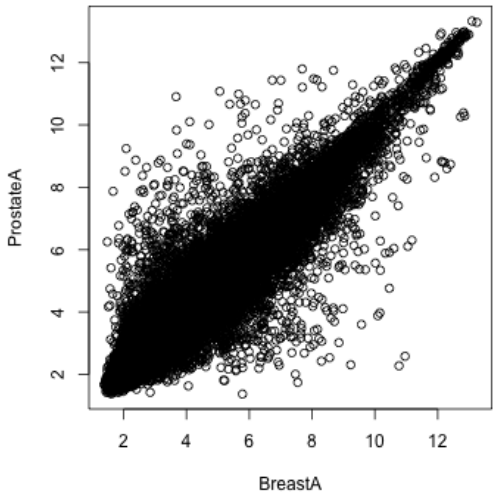




In addition we can draw scatter plots between expression values at the transcript-level:

```
> expr  <- validData(expr.mp)
> plot(log2(expr[,1]), log2(expr[,2]), xlab="BreastA", ylab="BreastB")
> plot(log2(expr[,1]), log2(expr[,4]), xlab="BreastA", ylab="ProstateA")
```





3 Background Correction using `express()`

Currently the following background correction methods are available as `bgcorrect.method`:

- `rma`: background is calculated using a global model for the distribution of probe intensities.
- `sector`: a constant background is calculated for each sector separately and subtracted from intensities.
- `weightedsector`: first, a constant background is calculated for each sector separately, then a weight is calculated and each background value multiplied with the weight to smooth the transition between sectors, and finally the smoothed background is subtracted from intensities.
- `gccontent`: background is calculated based on the mean intensity of probes with identical GC content.

Note: By default the summarization methods `farms` and `dfw` skip the background correction step.

3.1 rma

The `rma` background adjustment method is described in Irizarry (**?**). By default PM probe intensities are corrected by using a global model for the distribution of probe intensities. When using function `express()` the following `bgcorrect` parameters can be used for `bgcorrect.method="rma"`:

Settings to be used for `bgcorrect.select`:
`none`: scheme mask is not changed by selector

Settings to be used for `bgcorrect.option = "<option>:<kernel>"`:
`option`: pmonly, mmonly, both
`kernel`: epanechnikov, gaussian, rectangular, triangular, biweight, cosine, optcosine

Settings to be used for `bgcorrect.params`:
`numpoints`: size of density array, preferrable power of 2 (default: 16384)

The default settings for `rma` background correction are:

```
> bgrd.rma <- express(data.u133p2, "BgrdRMA", filedir=outdir, tmpdir="", update=FALSE,
+              bgcorrect.method="rma", bgcorrect.select="none", bgcorrect.option="pmonly:epanechnikov",
+              bgcorrect.params=c(16384))
```

3.2 sector (mas4)

The `sector` background subtraction method divides the array into a certain number of sectors, identifies the lowest 2% of probe intensities for each sector, and computes their average, whereby all probes are used (**?**). When using function `express()` the following `bgcorrect` parameters can be used for `bgcorrect.method="sector"`:

Settings to be used for `bgcorrect.select`:
`all`: all probes are selected (default)
`pmonly`: only PM probes are selected
`mmonly`: only MM probes are selected
`both`: both PM and MM probes are selected
`none`: scheme mask is not changed by selector

Settings to be used for `bgcorrect.option`:
`subtractbg`: subtract bgrd from intensity (default) - result can be negative
`correctbg`: correct bgrd with noise fraction to avoid negative results
`attenuatebg`: use generalized log-transform to avoid negative results

Settings to be used for `bgcorrect.params`:
The following parameters are necessary:
`pcntcells`: precent of cells with lowest intensities in each sector (default: 0.02)
`secrows`: number of sector rows (default: 4)
`seccols`: number of sector columns (default: 4)
`smoothiter`: number of iterations used for smoothing of sector bgrd values (default: 0)
Optional parameter when using `bgcorrect.option="correctbg"`:
`noisefrac`: fraction of global background variation (default: 0.5)
Optional parameters when using `bgcorrect.option="attenuatebg"`:
`l`: tunable parameter, 0<=l<=1 (default: 0.005)
`h`: parameter (default: -1)

Using the default settings the MAS4 `sector` background correction is obtained as follows:

```
> bgrd.mas4 <- express(data.u133p2, "BgrdMAS4", filedir=outdir, tmpdir="", update=FALSE,
+              bgcorrect.method="sector", bgcorrect.select="all", bgcorrect.option="subtractbg",
+              bgcorrect.params=c(0.02, 4, 4, 0))
```

However, using the default settings has severe disadvantages:

First, using `bgcorrect.option="subtractbg"` results in a high percentage of background-corrected probe intensities with negative values! This was the most severe problem of the MAS4 algorithm (**?**) since it resulted in negative expression levels, too.

Second, to divide an array into 16 sectors was sufficient for the initial expression arrays, e.g. HuGeneFL, but for the newer arrays, e.g. HG-U133_Plus_2, and especially the whole genome and exon arrays it would be of advantage to divide the array into more sectors, e.g. 64 sectors.

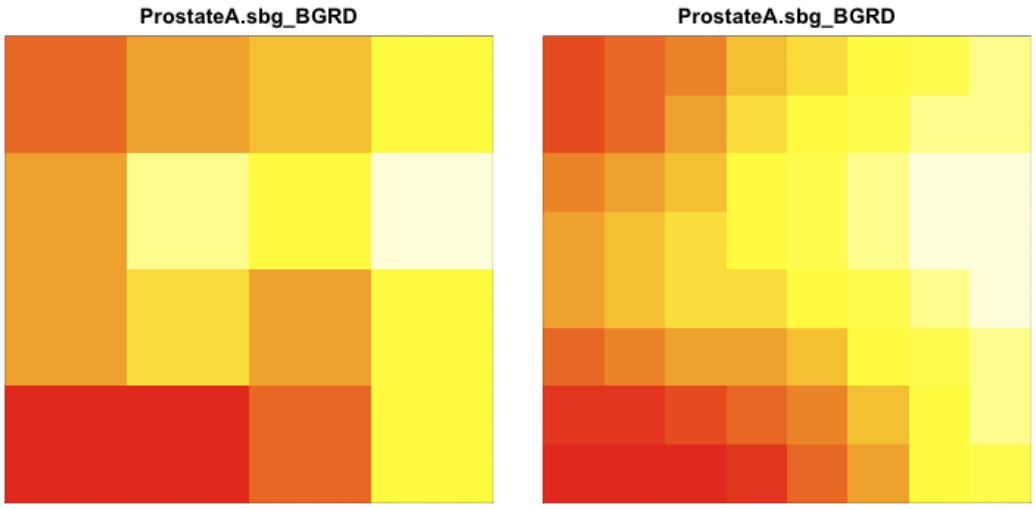
Thus, the following setup is recommended for `sector` background correction:

```
> bgrd.mas <- express(data.u133p2, "BgrdMAS8x8", filedir=outdir, tmpdir="", update=FALSE,
+             bgcorrect.method="sector", bgcorrect.select="both", bgcorrect.option="correctbg",
+             bgcorrect.params=c(0.02, 8, 8, 3, 0.5))
```

Here we divide each array into 8x8 sectors, use only PM+MM probes, and use `bgcorrect.option="correctbg"`, which is the default option for MAS5 and avoids negative probe intensities.

Now we can compare the resulting background values by drawing the corresponding images:

```
> bgrdnames <- colnames(validBgrd(bgrd.mas4))
> image(bgrd.mas4, bg=TRUE, transfo=log2, col=heat.colors(12), names=bgrdnames[4])
> bgrdnames <- colnames(validBgrd(bgrd.mas))
> image(bgrd.mas, bg=TRUE, transfo=log2, col=heat.colors(12), names=bgrdnames[4])
```



As both figures show there is a density gradient indicating that there might have been a problem with the staining/washing step of this specific array. However, the right image shows this effect more clearly, and additionally it results in more smooth transitions between adjacent sectors.

Note: To test the quality of arrays with respect to potential density gradients, I would propose to use the `sector` background with 8x8 sectors and option `"correctbg"`. In addition, these settings could also replace the default MAS5 `weightedsector` background algorithm described below.

3.3 weightedsector (mas5)

The `weightedsector` background correction method divides the array into a certain number of sectors and identifies the lowest 2% of probe intensities for each sector. However, in contrast to `sector` only PM and MM probes are used and each probe is adjusted based upon a weighted average of the backgrounds for each of the sectors. The weights are based on the distances between the location of the probe and the centriods of the different sectors (**?**). When using function `express()` the following `bgcorrect` parameters can be used for `bgcorrect.method="weightedsector"`:

Settings to be used for `bgcorrect.select`:
`all`: all probes are selected
`pmonly`: only PM probes are selected
`mmonly`: only MM probes are selected
`both`: both PM and MM probes are selected (default)
`none`: scheme mask is not changed by selector

Settings to be used for `bgcorrect.option`:
`subtractbg`: subtract bgrd from intensity - result can be negative
`correctbg`: correct bgrd with noise fraction to avoid negative results (default)
`attenuatebg`: use generalized log-transform to avoid negative results

Settings to be used for `bgcorrect.params`:
The following parameters are necessary:
`pcntcells`: precent of cells with lowest intensities in each sector (default: 0.02)
`secrows`: number of sector rows (default: 4)
`seccols`: number of sector columns (default: 4)

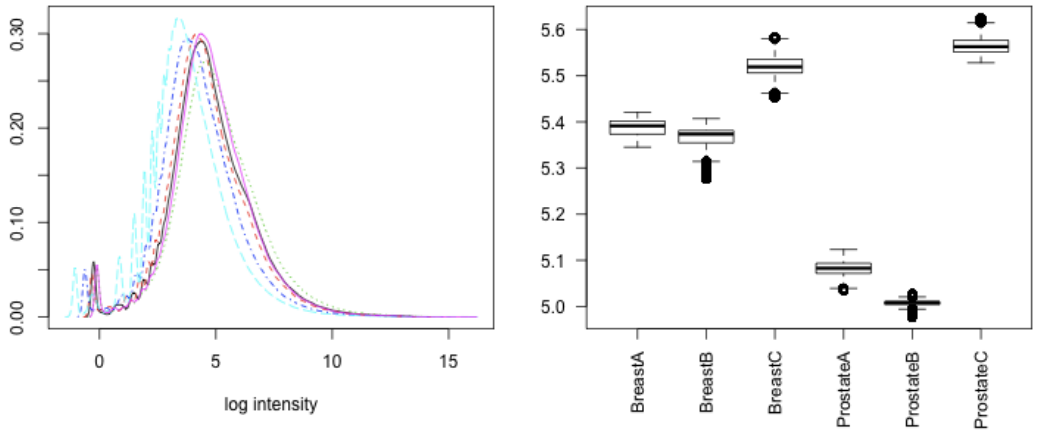
`smoothiter`: number of iterations used for smoothing of sector bgrd values (default: 0)
`smooth`: smoothing parameter added to avoid infinite weights (default: 100)
Parameter when using default `mas5` background `bgcorrect.option="correctbg"`:
`noisefrac`: fraction of global background variation (default: 0.5)
Optional parameters when using alternative background `bgcorrect.option="attenuatebg"`:
`l`: tunable parameter, 0<=l<=1 (default: 0.005)
`h`: parameter (default: -1)

Using the default settings the MAS5 `weightedsector` background correction is obtained as follows:

```
> bgrd.mas5 <- express(data.u133p2, "BgrdMAS5", filedir=outdir, tmpdir="", update=FALSE,
+              bgcorrect.method="weightedsector", bgcorrect.select="both", bgcorrect.option="correctbg
+              bgcorrect.params=c(0.02, 4, 4, 0, 100, 0.5))
```

For quality control purposes we can e.g. draw density plots of the background-corrected intensities, and boxplots of the background values:

```
> hist(bgrd.mas5, which="both")
> bgrd <- validBgrd(bgrd.mas5)
> boxplot(log2(bgrd), las=2)
```



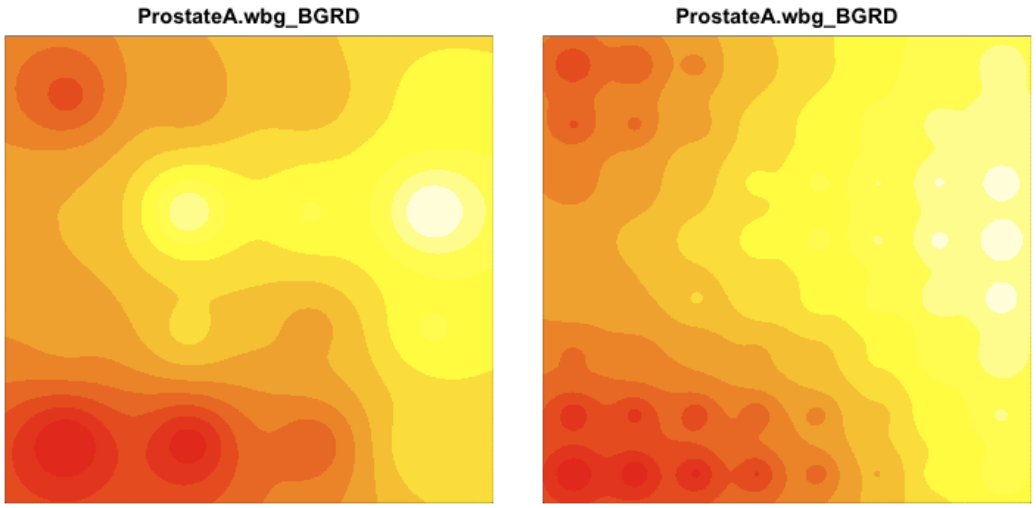
The density plot of the background-corrected intensities reveals a small peak around zero log intensity indicating that for a certain fraction of probes intensities were replaced by the noise fraction. Reducing the lowest percentage of cells used for background correction to 0.5%, i.e. `pcntcells=0.005` eliminated this peak almost completely.

As already explained for `sector` background, it is of advantage to increase the number of sectors used to 8x8 sectors:

```
> bgrd.mas <- express(data.u133p2, "BgrdMAS5_8x8", filedir=outdir, tmpdir="", update=FALSE,
+              bgcorrect.method="weightedsector", bgcorrect.select="both", bgcorrect.option="correctbg"
+              bgcorrect.params=c(0.02, 8, 8, 3, 100, 0.5))
```

As above, we can compare the resulting background values by drawing the corresponding images:

```
> bgrdnames <- colnames(validBgrd(bgrd.mas5))
> image(bgrd.mas5, bg=TRUE, transfo=log2, col=heat.colors(12), names=bgrdnames[4])
> bgrdnames <- colnames(validBgrd(bgrd.mas4))
> image(bgrd.mas, bg=TRUE, transfo=log2, col=heat.colors(12), names=bgrdnames[4])
```



Due to the weighting algorithm used, each sector is smoothed so that it is a "cone" centered in the middle of the sector. Once again, the right image shows the density gradient more clearly and results in more smooth transitions between adjacent sectors.

Note: Due to the weighting algorithm used computing the `weightedsector` background consumes large amounts of memory, which can be a severe problem when trying to use 8x8 sectors for high density arrays, especially exon arrays.

3.4 gccontent

The `gccontent` background attenuation method identifies MM probes and PM probes with the same GC content and uses the mean intensity of the MM probes to "attenuate" the corresponding PM intensities. It was introduced by Affymetrix for the exon arrays containing special "genomic" and "antigenomic" probes (**?**), but can also be used for expression arrays. When using function `express()` the following `bgcorrect` parameters can be used for `bgcorrect.method="gccontent"`:

Settings to be used for `bgcorrect.select`:

`all`:	all probes are selected
`pmonly`:	only PM probes are selected
`mmonly`:	only MM probes are selected
`both`:	both PM and MM probes are selected
`none`:	scheme mask is not changed by selector

Settings to be used for `bgcorrect.option`:

`subtractbg`:	subtract bgrd from intensity - result can be negative
`correctbg`:	correct bgrd with noise fraction to avoid negative results
`attenuatebg`:	use generalized log-transform to avoid negative results (default)

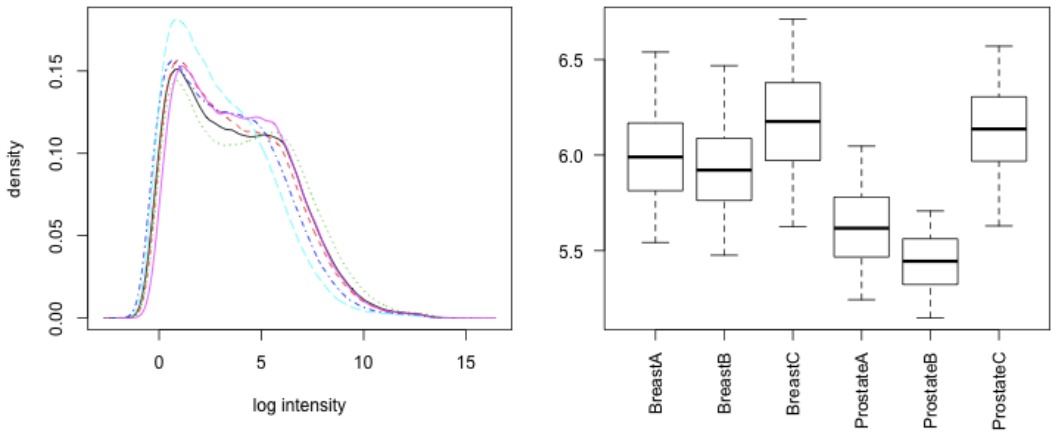
Settings to be used for `bgcorrect.params`:
The following parameters are necessary:
`trim`: trim value for trimmed mean (default: 0.5)
Parameters when using default background `bgcorrect.option="attenuatebg"`:
`l`: tunable parameter, 0<=l<=1 (default: 0.005)
`h`: parameter (default: -1)
Optional parameter when using background `bgcorrect.option="correctbg"`:
`noisefrac`: fraction of global background variation (default: 0.5)

The `gccontent` background correction is computed as follows:

```
> bgrd.gc <- express(data.u133p2, "BgrdGC", filedir=outdir, tmpdir="", update=FALSE,
+             bgcorrect.method="gccontent", bgcorrect.select="none", bgcorrect.option="attenuatebg",
+             bgcorrect.params=c(0.4, 0.005, -1.0))
```

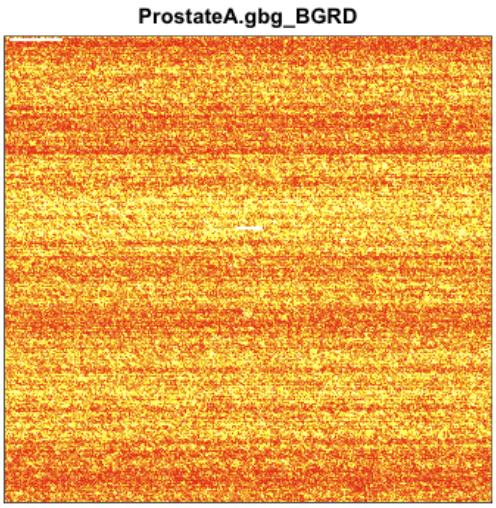
For quality control purposes we can draw density plots of the background-corrected intensities, and boxplots of the background values:

```
> hist(bgrd.gc, which="pm")
> bgrd <- validBgrd(bgrd.gc, which="pm")
> boxplot(log2(bgrd), las=2)
```



In addition we can draw the image for the background values:

```
> bgrdnames <- colnames(validBgrd(bgrd.gc))
> image(bgrd.gc, bg=TRUE, transfo=log2, col=heat.colors(12), names=bgrdnames[4])
```



4 Probe-level Normalization using `express()`

Currently the following probe-level normalization methods are available as `normalize.method`:

- `quantile`: normalization is done by replacing sorted intensities with the respective column means.
- `mean`: all arrays are scaled so that they have the same trimmed mean value.
- `median`: all arrays are scaled so that they have the same median value.
- `lowess`: each array is normalized to a reference array using locally-weighted polynomial regression.
- `supsmu`: each array is normalized to a reference array by applying Friedmann's super smoother.

Note 1: The following examples will be based on `sector` background `bgrd.mas` although we could use any of the above background algorithms, or even skip the background correction by using the raw data `data.u133p2`.

Note 2: By default the summarization methods `avgdiff` (MAS4) and `tukeybiweight` (MAS5) skip the probe-level normalization step, i.e. use `data.u133p2`.

4.1 quantile (rma)

The quantile normalization method was introduced by Bolstad (**?**) with the goal to give each array the same empirical distribution. To allow quantile normalization of many thousand arrays on computers with 1-2 GB RAM only, package `xps` uses `ROOT` trees for quantile normalization: Intensities for each array are sorted and stored in `ROOT` trees together with their ranks. After replacing the sorted tree entries with the mean entries of all trees, the trees are sorted according to their original ranks. When using function `express()` the following `normalize` parameters can be used for `normalize.method="quantile"`:

Settings to be used for `normalize.select`:

`pmonly`:	only PM probes are selected
`mmonly`:	only MM probes are selected
`both`:	both PM and MM probes are selected
`none`:	scheme mask is not changed by selector

Settings to be used for `normalize.option = "<option>:<dataopt>:<bgrdopt>"`:
`option`: use unit tree for "transcript", "exon", "probeset"
`dataopt`: applied to PM and MM "together" or "separate"
`bgrdopt`: "none" - no background subtraction (already done)

Settings to be used for `normalize.params`:
`trim`: trim value for trimmed mean (default: 0.0)
`delta`: use robust ties (0.4 for package affy >=1.16.0) or not (default: 1.0)

The `quantile` probe-level normalization is computed as follows:

```
> norm.qu <- express(bgrd.mas, "NormQuan", filedir=outdir, update=FALSE,
+            normalize.method="quantile", normalize.select="all",
+            normalize.option="transcript:together:none",
+            normalize.logbase="0", normalize.params=c(0.0, 1.0))
```

As for the initial example of step-wise RMA computation (see chapter 2.2 above), density plots, boxplots and scatter plots can be drawn as quality controls.

4.2 mean/median (constant)

All arrays are scaled so that they have the same mean/median value. This value is determined either by the mean/median value of a reference tree or by a given target intensity. When using function `express()` the following `normalize` parameters can be used for `normalize.method="mean"` or for `normalize.method="median"`, respectively:

Settings to be used for `normalize.select`:
`all`: all probes are selected
`pmonly`: only PM probes are selected
`mmonly`: only MM probes are selected
`both`: both PM and MM probes are selected

Settings to be used for `normalize.option = "<option>:<dataopt>"`:
`option`: use unit tree for "transcript", "exon", "probeset"
`dataopt`: applied to "all" selected probes

Settings to be used for `normalize.logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

Settings to be used for `normalize.params`:
`trim`: trim value for trimmed mean (only for method "mean")
`targetinten`: if > 0 then scale mean/median to value of target intensity

The trimmed `mean` probe-level normalization is computed as follows (by default, a mean reference tree is computed first):

```
> norm.mn <- express(bgrd.mas, "NormMean", filedir=outdir, tmpdir="", update=FALSE,
+            normalize.method="mean", normalize.select="both", normalize.option="transcript:all",
+            normalize.logbase="0", normalize.params=c(0.0, -1))
```

Analogously, when determining a certain target intensity (e.g. 500) as reference, the `median` probe-level normalization is computed as follows:

```
> norm.md <- express(bgrd.mas, "NormMedian", filedir=outdir, tmpdir="", update=FALSE,
+            normalize.method="median", normalize.select="pmonly", normalize.option="transcript:all",
+            normalize.logbase="log2", normalize.params=c(500), reference.index=1)
```

Here the reference index is set to `reference.index=1` to prevent computing a mean reference tree which will not be used since the intensities are normalized to target intensity 500 and not to a reference tree.

4.3 lowess

Each array is normalized to a reference array using locally-weighted polynomial regression. The reference array can be one of the arrays or the mean/median of all arrays. When using function `express()` the following `normalize` parameters can be used for `normalize.method="lowess"`:

Settings to be used for `normalize.select`:
`all`: all probes are selected
`pmonly`: only PM probes are selected
`mmonly`: only MM probes are selected
`both`: both PM and MM probes are selected

Settings to be used for `normalize.option = "<option>:<dataopt>"`:
`option`: use unit tree for "transcript", "exon", "probeset"
`dataopt`: applied to "all" selected probes

Settings to be used for `normalize.logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

Settings to be used for `normalize.params`:
The following parameters are necessary:
`span`: smoother span (default: 2/3)
`iter`: number of robustifying iterations (default: 3)
`rule`: 0, 1 or 2, determines how approx interpolation should be done at ends
`f`: for approx method "constant" a value between 0 and 1 (not used)

The `lowess` probe-level normalization is computed as follows:

```
> norm.low <- express(bgrd.mas, "NormLowess", filedir=outdir, tmpdir="", update=FALSE,
+             normalize.method="lowess", normalize.select="pmonly", normalize.option="transcript:all",
+             normalize.logbase="log2", normalize.params=c(0.67, 3.0, 0.0, 0.0))
```

4.4 supsmu

Each array is normalized to a reference array by applying Friedmann's super smoother. The reference array can be one of the arrays or the mean/median of all arrays. When using function `express()` the following `normalize` parameters can be used for `normalize.method="supsmu"`:

Settings to be used for `normalize.select`:
`all`: all probes are selected
`pmonly`: only PM probes are selected
`mmonly`: only MM probes are selected
`both`: both PM and MM probes are selected

Settings to be used for `normalize.option = "<option>:<dataopt>"`:
`option`: use unit tree for "transcript", "exon", "probeset"
`dataopt`: applied to "all" selected probes

Settings to be used for `normalize.logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

Settings to be used for `normalize.params`:
The following parameters are necessary:

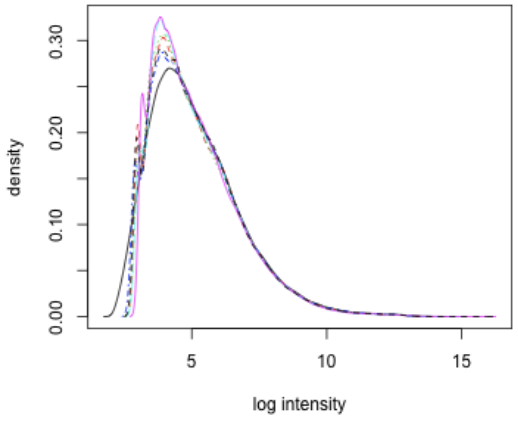
`bass`:	controls smoothness of fitted curve (default: 0.0)
`span`:	fraction of observations in span (default: 0.0)
`rule`:	0, 1 or 2, determines how approx interpolation should be done at ends
`f`:	for approx method "constant" a value between 0 and 1 (not used)

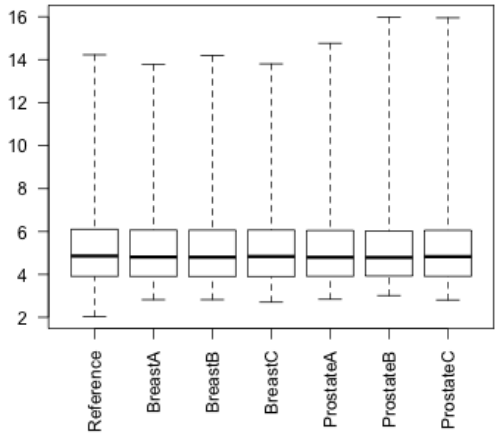
The `supsmu` probe-level normalization is computed as follows:

```
> norm.sup <- express(bgrd.mas, "NormSupsmuR2", filedir=outdir, tmpdir="", update=FALSE,
+              normalize.method="supsmu", normalize.select="pmonly", normalize.option="transcript:all",
+              normalize.logbase="log2", normalize.params=c(0.0, 0.0, 2.0, 0.0))
```

As quality control we can draw density plots and boxplots of the normalized probe intensities:

```
> hist(norm.sup, which="pm")
> boxplot(norm.sup, which="pm")
```





5 Summarization using express()

Summarization converts the (background-corrected and/or normalized) probe intensities to the corresponding expression levels for probe sets defined with summarize.option:

transcript:	summarization units are transcripts (default)
exon:	summarization units are exons (exon arrays only)
probeset:	summarization units are probesets (exon arrays only)

Currently the following summarization methods are available as summarize.method:

- medianpolish: fits an additive model using Tukey's median polish procedure.
- avgdiff: computes the average difference between PM and MM.
- tukeybiweight: a Tukey biweight estimator is used to compute a robust mean of (adjusted) intensities.
- dfw: a distribution-free weight is applied to each probe before summarization.
- farms: a factor analysis model is used for noise correction of measurement values before summarization.

5.1 medianpolish (rma)

This multichip summarization method fits a robust linear model using Tukey's median polish procedure. When using function express() the following summarize parameters can be used for summarize.method="medianpolish":

Settings to be used for summarize.select:

pmonly:	only PM probes are selected (default)
mmonly:	only MM probes are selected

Settings to be used for summarize.logbase:

logbase:	"0" - linear, "log", "log2", "log10" - log with base e, 2, 10

Settings to be used for summarize.params:

maxiter:	maximal number of iterations (default: 10)
eps:	epsilon of test for convergence (default: 0.01)
neglog:	substitution for logarithm of negative values (default: 1.0)

The default RMA medianpolish summarization is computed as follows:

```
> expr.mp <- express(norm.qu, "ExprMedpol", filedir=outdir, update=FALSE,
+            summarize.method="medianpolish", summarize.select="pmonly", summarize.option="transcript"
+            summarize.logbase="log2", summarize.params=c(10, 0.01, 1.0))
```

As for the initial example of step-wise RMA computation (see chapter 2.2 above), density plots, boxplots and scatter plots can be drawn as quality controls.

5.2 avgdiff (mas4)

This summarization method takes the difference PM-MM of every probe pair and averages the differences over the entire probe set. When using function `express()` the following `summarize` parameters can be used for `summarize.method="avgdiff"`:

Settings to be used for `summarize.select`:
`none`: scheme mask is not changed by selector (default)

Settings to be used for `summarize.logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

Settings to be used for `summarize.params`:
`STP`: number of standard deviations (default: 3)

The MAS4 `avgdiff` summarization is computed as follows (here we use the background-corrected intensities `bgrd.mas` w/o normalization):

```
> expr.adf <- express(bgrd.mas, "ExprAvgDif", filedir=outdir, update=FALSE,
+             summarize.method="avgdiff", summarize.select="none", summarize.option="transcript",
+             summarize.logbase="0", summarize.params=c(3.0))
```

5.3 tukeybiweight (mas5)

After subtracting an ideal mismatch value from PM, a 1-step Tukey biweight estimator is used to compute a robust mean of the resulting values. When using function `express()` the following `summarize` parameters can be used for `summarize.method="tukeybiweight"`:

Settings to be used for `summarize.select`:
`none`: scheme mask is not changed by selector (default)

Settings to be used for `summarize.logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

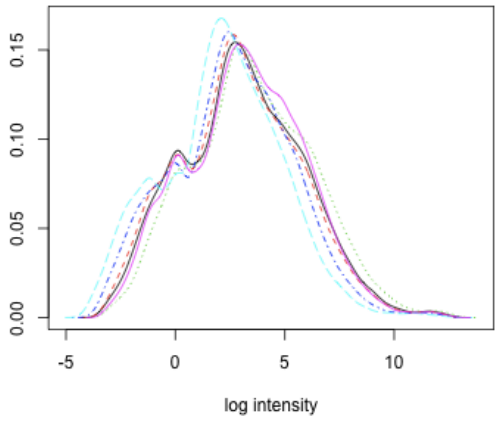
Settings to be used for `summarize.params`:
`tau`: contrast tau (default: 0.03)
`scaletau`: scale tau (default: 10)
`delta`: delta for probe value calculation (default: 2.0e-20)
`c`: a tuning constant (default: 5.0)
`eps`: small value to avoid zeros in division (default: 0.0001)
`neglog`: substitution for logarithm of negative values (default: 1.0)
`noisefrac`: fraction of global background variation (default: 0.5)

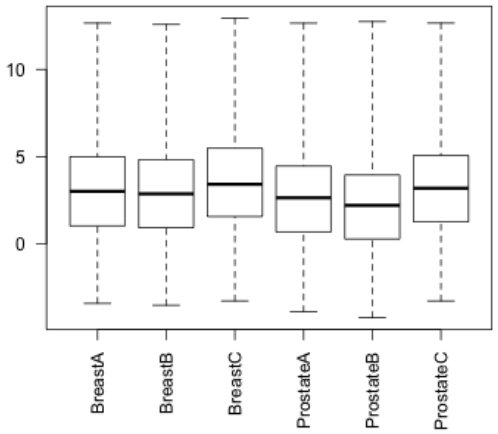
The MAS5 `tukeybiweight` summarization is computed as follows (here we use the background-corrected intensities `bgrd.mas` w/o normalization):

```
> expr.tbw <- express(bgrd.mas, "ExprTukey", filedir=outdir, update=FALSE,
+             summarize.method="tukeybiweight", summarize.select="none", summarize.option="transcript"
+             summarize.logbase="log2", summarize.params=c(0.03, 10.0, 2.0e-20, 5.0, 0.0001, 1.0, 0.5)
```

Now we can draw density plots and boxplots of the final expression values:

```
> hist(expr.tbw)
> boxplot(expr.tbw)
```

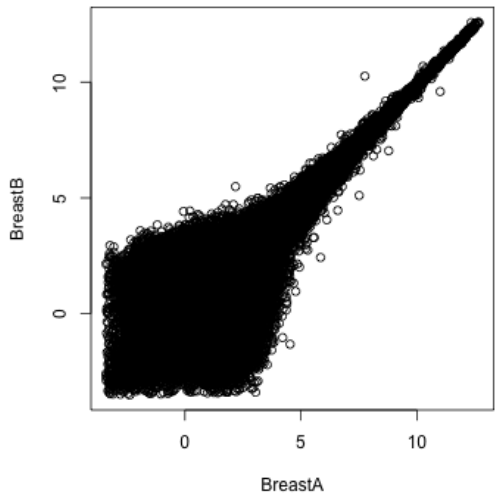


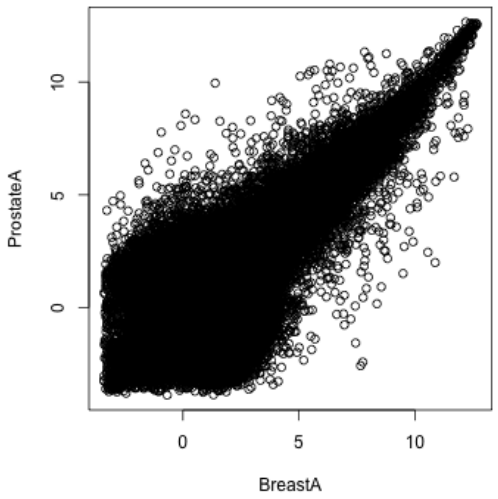


As the boxplot reveals no probe-level normalization was done since the MAS5 algorithm usually scales the expression levels to a certain target intensity (see function `normalize`).

In addition we can draw scatter plots between expression values at the transcript-level:

```
> expr  <- validData(expr.tbw)
> plot(log2(expr[,1]), log2(expr[,2]), xlab="BreastA", ylab="BreastB")
> plot(log2(expr[,1]), log2(expr[,4]), xlab="BreastA", ylab="ProstateA")
```





5.4 dfw

This multichip summarization method uses the Tukey weight function to give probes with unusually high or low variability across arrays small weights. These weighted probe intensities are used to obtain the summarized expression values (**?**). When using function `express()` the following `summarize` parameters can be used for `summarize.method="dfw"`:

Settings to be used for `summarize.select`:
`pmonly`: only PM probes are selected (default)
`mmonly`: only MM probes are selected
`both`: both PM and MM probes are selected

Settings to be used for `summarize.logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

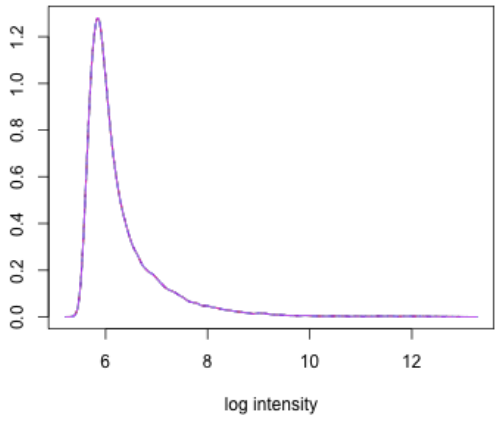
Settings to be used for `summarize.params`:
`m`: exponent for range WR (default: 3)
`n`: exponent for stdev WSD (default: 1)
`c`: scale parameter (default: 0.01)

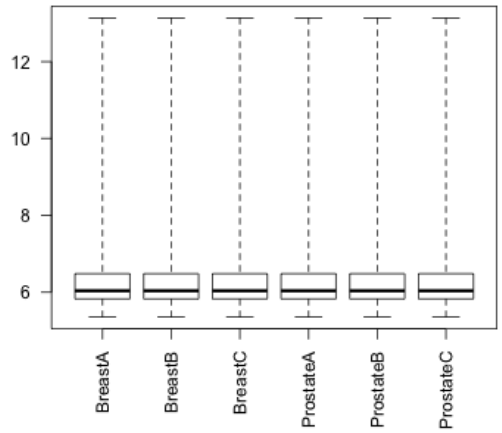
The `dfw` summarization is computed as follows (w/o background correction):

```
> expr.dfw <- express(norm.qu, "ExprDFW", filedir=outdir, update=FALSE,
+              summarize.method="dfw", summarize.select="pmonly", summarize.option="transcript",
+              summarize.logbase="log2", summarize.params=c(3.0, 1.0, 0.01))
```

Now we can draw density plots and boxplots of the final expression values:

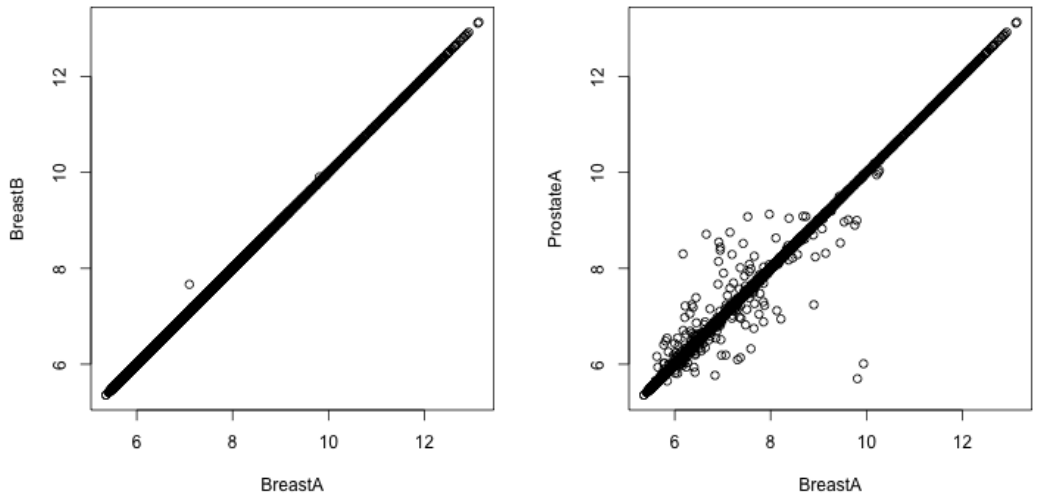
```
> hist(expr.dfw)
> boxplot(expr.dfw)
```





In addition we can draw scatter plots between expression values at the transcript-level:

```
> expr  <- validData(expr.dfw)
> plot(log2(expr[,1]), log2(expr[,2]), xlab="BreastA", ylab="BreastB")
> plot(log2(expr[,1]), log2(expr[,4]), xlab="BreastA", ylab="ProstateA")
```



5.5 farms

This multichip summarization method is based on a factor analysis model for which a Bayesian maximum a posteriori method optimizes the model parameters under the assumption of Gaussian measurement noise. Afterwards, the expression levels are estimated from the model (**?**). When using function `express()` the following `summarize` parameters can be used for `summarize.method="farms"`:

Settings to be used for `summarize.select`:
`pmonly`: only PM probes are selected (default)
`mmonly`: only MM probes are selected
`both`: both PM and MM probes are selected

Settings to be used for `summarize.logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

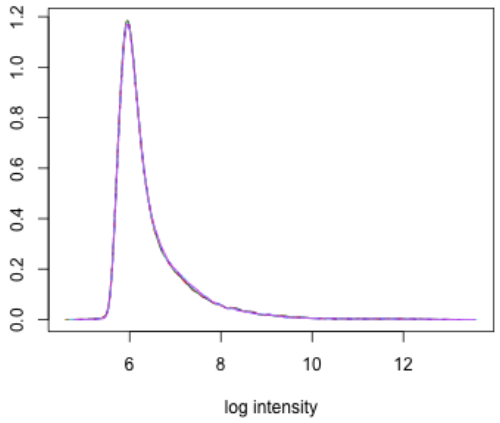
Settings to be used for `summarize.params`:
`version`: version of farms package, 131 (farms_1.3.1); 130 (farms_1.3) (default: 131)
`weight`: hyperparameter (default: 0.5)
`mu`: hyperparameter (default: 0.0)
`scale`: scaling parameter (default: 1.0)
`tol`: termination tolerance (default: 0.00001)
`cyc`: maximum number of cycles (default: 100)
`weighted`: weighted mean (for 131 only) (default: 1.0)

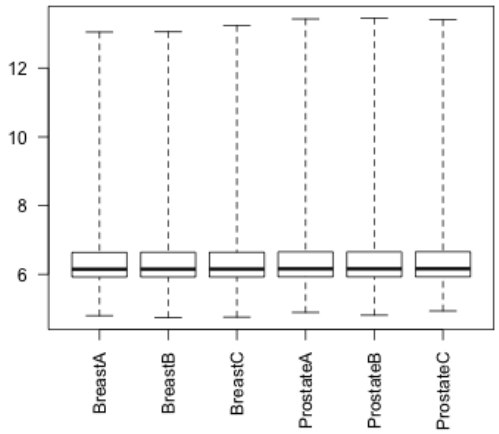
The default `farms` summarization is computed as follows (w/o background correction):

```
> expr.frm <- express(norm.qu, "ExprFARMS", filedir=outdir, update=FALSE,
+             summarize.method="farms", summarize.select="pmonly", summarize.option="transcript",
+             summarize.logbase="log2", summarize.params=c(131, 0.5, 0.0, 1.0, 0.00001, 100, 1))
```

Now we can draw density plots and boxplots of the final expression values:

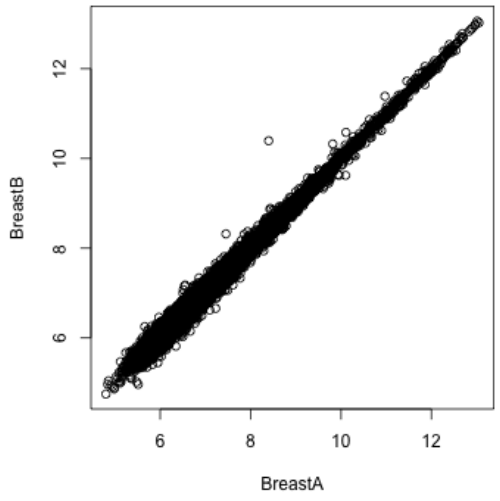
```
> hist(expr.frm)
> boxplot(expr.frm)
```

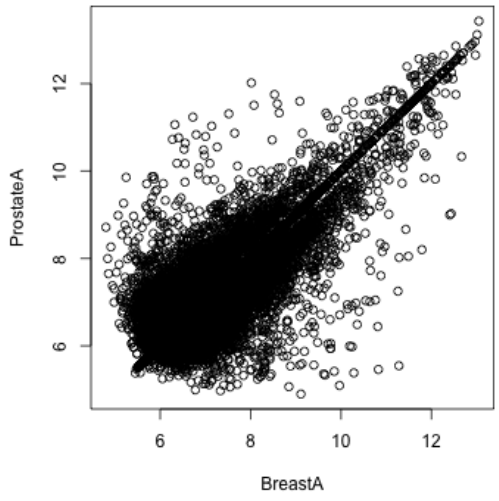




In addition we can draw scatter plots between expression values at the transcript-level:

```
> expr  <- validData(expr.frm)
> plot(log2(expr[,1]), log2(expr[,2]), xlab="BreastA", ylab="BreastB")
> plot(log2(expr[,1]), log2(expr[,4]), xlab="BreastA", ylab="ProstateA")
```





Note: Although package farms is available under the GNU GPL License, the authors state on their web site that: "This package (i.e. farms_1.x) is only free for non-commercial users. Non-academic users must have a valid license." Since I do not know if this statement applies for my C++ implementation, too, it is recommended that respective users contact the authors of the original package.

6 Probeset-level Normalization using `normalize()`

Currently the following probeset-level normalization methods are available as `method`:

- `mean`: all arrays are scaled so that they have the same trimmed mean value.
- `median`: all arrays are scaled so that they have the same median value.
- `lowess`: each array is normalized to a reference array using locally-weighted polynomial regression.
- `supsmu`: each array is normalized to a reference array by applying Friedmann's super smoother.

6.1 mean/median (mas)

All arrays are scaled so that they have the same mean/median value. This value is determined either by the mean/median value of a reference tree or by a given target intensity. When using function `normalize()` the following parameters can be used for `method="mean"` or for `method="median"`, respectively:

Settings to be used for `select`:
`separate`: use separate mask for each normalization (default)
`together`: use combined masks for normalization

Settings to be used for `normalize.option = "<option>:<dataopt>"`:
`option`: use unit tree for "transcript", "exon", "probeset"
`dataopt`: applied to "all" selected probes

Settings to be used for `normalize.logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

Settings to be used for `normalize.params`:
`trim`: trim value for trimmed mean (only for method "mean")
`targetinten`: if > 0 then scale mean/median to value of target intensity

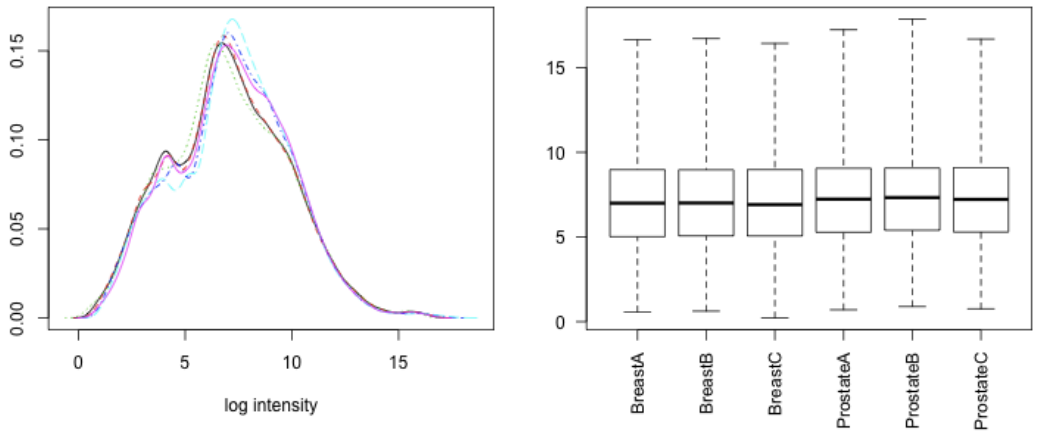
The trimmed `mean` probeset-level normalization is computed as follows:

```
> nrxp.mn <- normalize(expr.tbw,  "NrmExpMean", filedir=outdir, update=FALSE,
+            select="separate", method="mean", option="transcript:all", logbase="0",
+            refindex=1, refmethod="mean", params=c(0.02, 500))
```

Here the reference index is set to `refindex=1` to prevent computing a mean reference tree which will not be used since the intensities are normalized to target intensity 500 and not to a reference tree.

Now we can draw density plots and boxplots of the normalized expression values:

```
> hist(nrxp.mn)
> boxplot(nrxp.mn)
```



Note: Probeset-level normalization is usually necessary for the MAS4 and MAS5 algorithms, since no probe-level normalization is done and the average expression level (after removing the 2% highest and lowest expression values) (**?**) is usually scaled to a "Target Intensity" such as `targetinten=500` for expression arrays or `targetinten=100` for exon arrays.

6.2 lowess

Each array is normalized to a reference array using locally-weighted polynomial regression. The reference array can be one of the arrays or the mean/median of all arrays. When using function `normalize()` the following parameters can be used for `method="lowess"`:

Settings to be used for `select`:
`separate`: use separate mask for each normalization (default)
`together`: use combined masks for normalization

Settings to be used for `option = "<option>:<dataopt>"`:
`option`: use unit tree for "transcript", "exon", "probeset"
`dataopt`: applied to "all" probes, selected "sel" probes only, or percentage probes (range (0,1])

Settings to be used for `logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

Settings to be used for `params`:
The following parameters are necessary:
`span`: smoother span (default: 2/3)
`iter`: number of robustifying iterations (default: 3)
`rule`: 0, 1 or 2, determines how approx interpolation should be done at ends
`f`: for approx method "constant" a value between 0 and 1 (not used)

The `lowess` probeset-level normalization is computed as follows:

```
> nrxp.low <- normalize(expr.tbw,  "tmp_NrmExpLow", filedir=outdir, update=FALSE,
+             select="separate", method="lowess", option="transcript:all", logbase="log2",
+             refindex=0, refmethod="mean", params=c(0.67, 3, 0.0, 0.0))
```

6.3 supsmu

Each array is normalized to a reference array by applying Friedmann's super smoother. The reference array can be one of the arrays or the mean/median of all arrays. When using function `normalize()` the following parameters can be used for `method="supsmu"`:

Settings to be used for `select`:
`separate`: use separate mask for each normalization (default)
`together`: use combined masks for normalization

Settings to be used for `option = "<option>:<dataopt>"`:
`option`: use unit tree for "transcript", "exon", "probeset"
`dataopt`: applied to "all" probes, selected "sel" probes only, or percentage probes (range (0,1])

Settings to be used for `logbase`:
`logbase`: "0" - linear, "log", "log2", "log10" - log with base e, 2, 10

Settings to be used for `params`:
The following parameters are necessary:
`bass`: controls smoothness of fitted curve (default: 0.0)
`span`: fraction of observations in span (default: 0.0)
`rule`: 0, 1 or 2, determines how approx interpolation should be done at ends
`f`: for approx method "constant" a value between 0 and 1 (not used)

The `supsmu` probeset-level normalization is computed as follows:

```
> nrxp.sup <- normalize(expr.tbw,  "tmp_NrmExpSupR2", filedir=outdir, update=FALSE,
+              select="separate", method="supsmu", option="transcript:all", logbase="log2",
+              refindex=0, refmethod="mean", params=c(0.0, 0.0, 2, 0.0))
```