BIOS 7659 Homework 2

Tim Vigers

28 September 2020

Quality Control

a) Read the 8 CEL files

```
pd=read.AnnotatedDataFrame("./targets.txt" , header=TRUE,
                           row.names=1,as.is=TRUE)
Data=ReadAffy(filenames=pData(pd)$FileName, phenoData=pd,
              sampleNames=sampleNames(pd))
```

Practice extracting information from data

```
kable(head(exprs(Data)),digits=3)
```

High1	High2	High3	High4	Low1	Low2	Low3	Low4
197	81	106	69	174	103	100	100
13571	9974	9267	6591	7231	4831	4724	4724
248	137	100	94	194	147	137	137
13810	9895	9550	6866	7407	4886	4919	4919
123	63	60	76	77	79	71	71
173	73	61	68	116	81	103	103

```
sampleNames(Data)

## [1] "High1" "High2" "High3" "High4" "Low1"  "Low2"  "Low3"  "Low4"

head(probeNames(Data))

## [1] "1007_s_at" "1007_s_at" "1007_s_at" "1007_s_at" "1007_s_at" "1007_s_at"

kable(head(mm(Data)),digits=3)
```

	High1	High2	High3	High4	Low1	Low2	Low3	Low4

370871	338	123	115	113	131	152	145	145
564482	831	435	393	397	695	791	852	852
1050513	1190	520	345	583	685	688	1033	1033
239977	1288	691	592	564	837	846	891	891
1141565	2740	1161	1035	1477	1540	1938	2269	2269
1131946	786	376	335	355	591	631	791	791

```r
kable(head(pm(Data)),digits=3)
```

	High1	High2	High3	High4	Low1	Low2	Low3	Low4
369707	368	220	216	248	405	379	476	476
563318	1350	687	590	585	1455	1626	1765	1765
1049349	1414	674	599	586	1464	1422	1701	1701
238813	3838	1830	1472	1446	2980	3205	3701	3701
1140401	3018	1338	1165	1445	2208	2616	3151	3151
1130782	1652	783	668	700	1249	1520	1784	1784

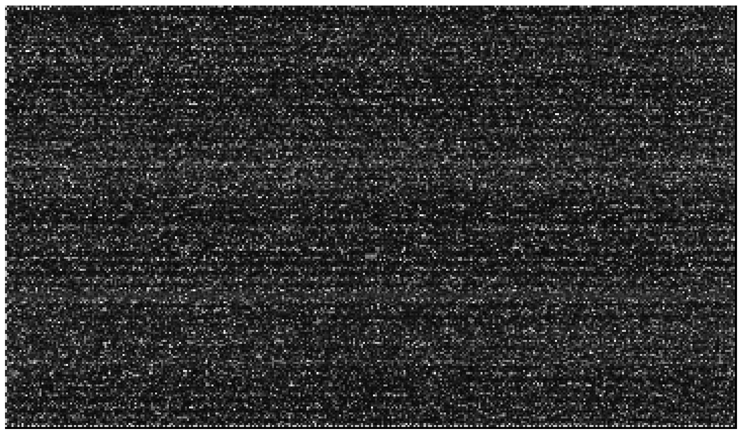
```r
kable(pData(Data),digits=3)
```

	FileName	Target
High1	High_1_HG-U133_Plus_2.CEL	High
High2	High_2_HG-U133_Plus_2.CEL	High
High3	High_3_HG-U133_Plus_2.CEL	High
High4	High_4_HG-U133_Plus_2.CEL	High
Low1	Low_1_HG-U133_Plus_2.CEL	Low
Low2	Low_2_HG-U133_Plus_2.CEL	Low
Low3	Low_3_HG-U133_Plus_2.CEL	Low
Low4	Low_4_HG-U133_Plus_2.CEL	Low

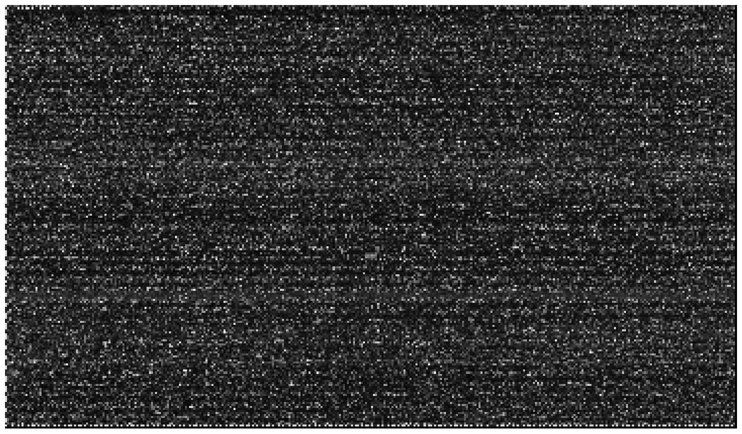
b) Plot the raw microarray images

```
image(Data)
```

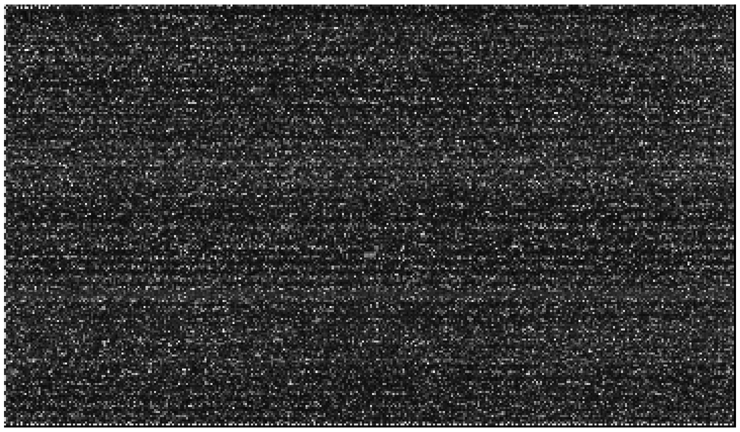
High1



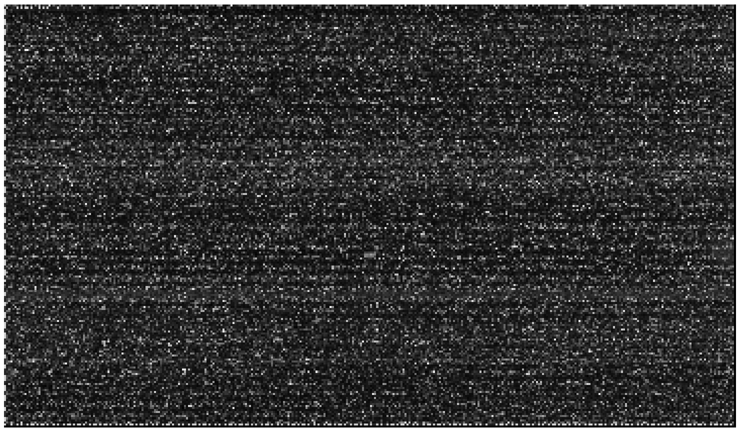
High2



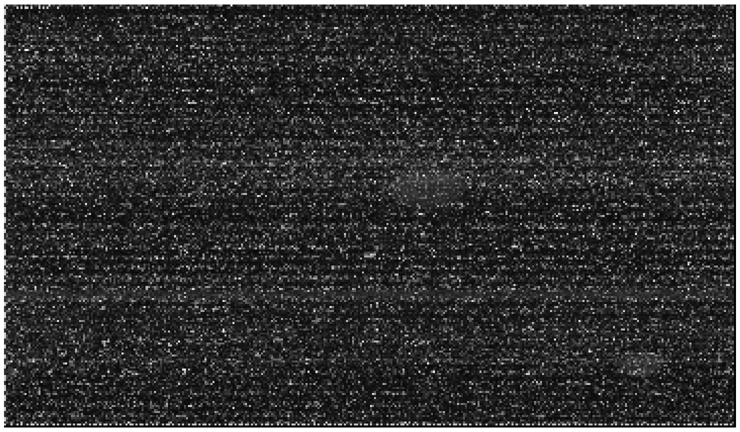
High3



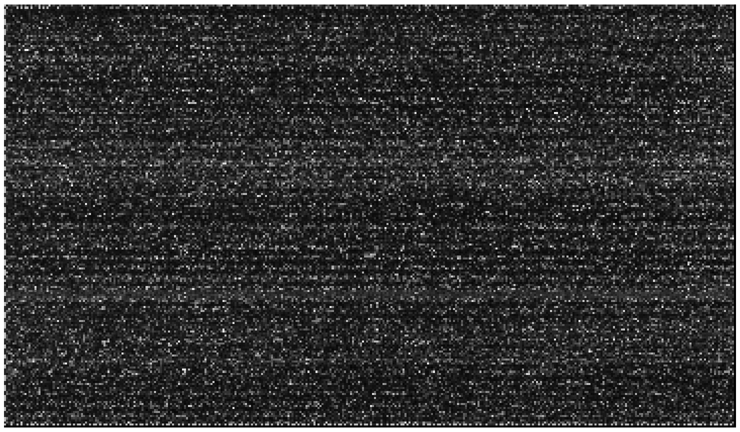
High4



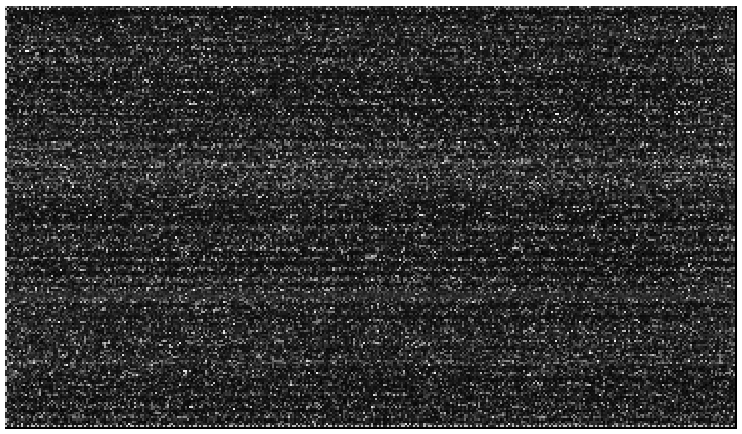
Low1



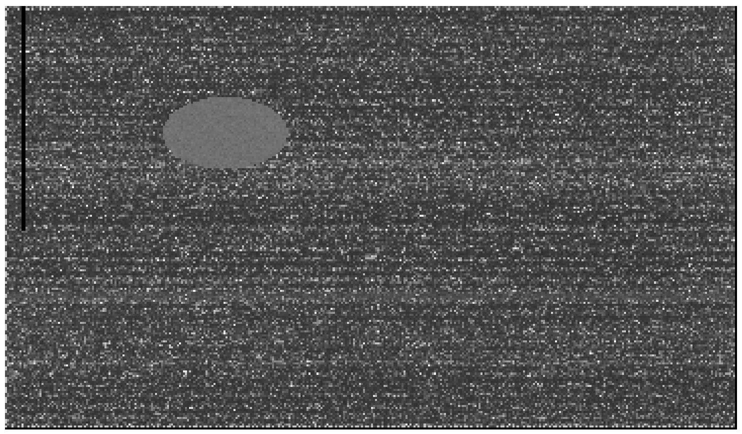
Low2



Low3



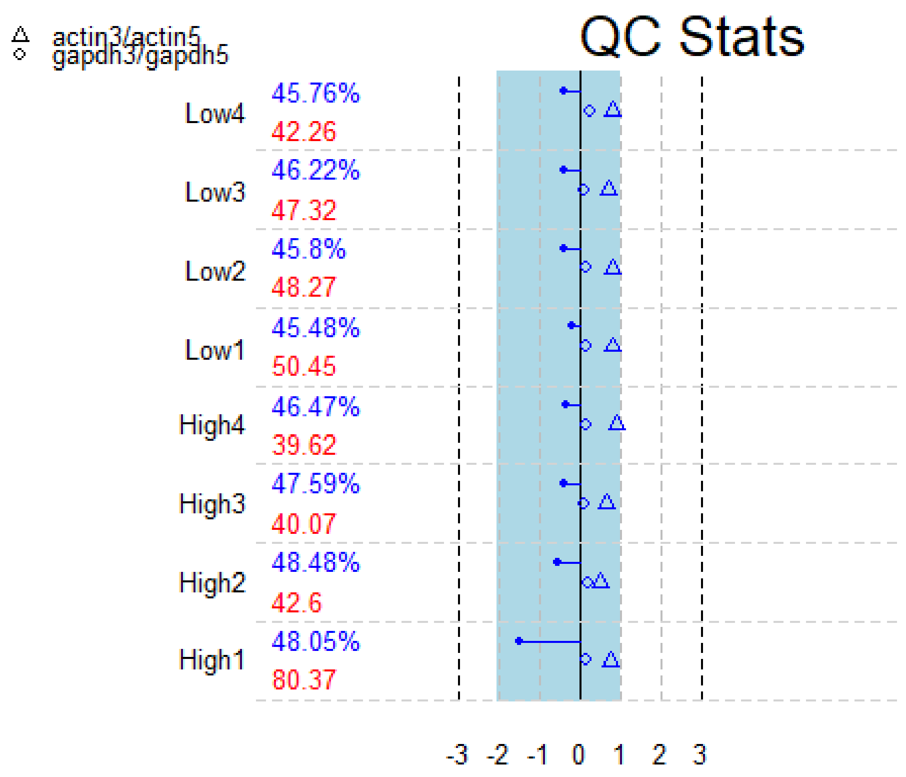
Low4



For the most part nothing in these plots stands out, except for “Low 4.” This array appears to be generally bright (washed out), with an oval patch of high fluorescence and a thin strip of probes with no fluorescence. It is not obvious exactly what happened, but it appears that the processing for this plate went wrong at some point.

c) Plot quality control metrics

```
plot.qc.stats(qc.affy(Data))
```



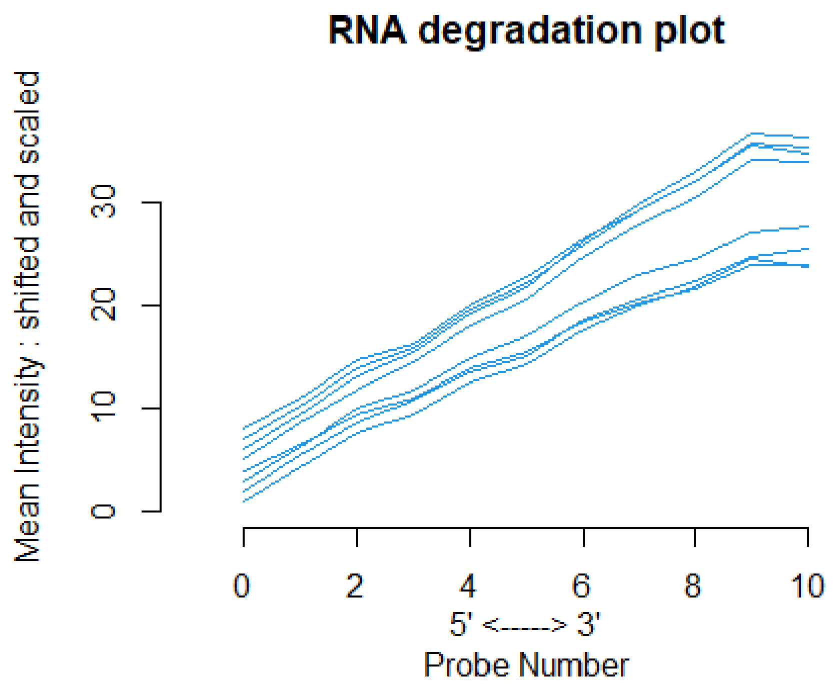
Interestingly, the quality control metrics appear to be reasonable based on this plot, even for array "Low4." The top number (a percentage) for each array is the present call, which is the percentage of genes reliably detected, and can be considered a measure of overall quality. In general these arrays appear to have similar percentages, which is a good sign. The bottom number represents average background level, and this appears to be pretty similar across arrays except for "High1" which has a much higher level than the rest. This suggests potential signal-to-noise problems with "High1."

Also, it is a little bit surprising that the background for "Low4" is among the lower end of background values, because visual inspection suggested there was a lot of noise. The package automatically flags significantly different (by default defined as more than 10% difference for present call and 20 units for background) values in red. So in general these values are acceptable aside from the background value of "High1."

The GAPDH ratios (circles) and beta actin ratios (triangles) also appear reasonable for all arrays. In general, the triangles should be within 3-fold change and the circles within 1, and the simpleaffy package will again automatically flag values outside these ranges by coloring them red. Lastly, the horizontal line for each array represents the distance from 0-fold change to its scale factor. The ends of each line should be within the shaded blue region, so the scale factors for all of these plates are compatible.

d) Plot the mean intensity from 3' to 5' end of the target mRNA

```
plotAffyRNAdeg(AffyRNAdeg(Data))
```



Probes in a probeset are ordered relative to the 5' end. Intensities are averaged by location across all probesets, for each array. Because RNA tends to start degradation at the 5' end, one would expect lower intensities at that end. Visual inspection of this plot suggests that RNA degradation is similar across all the arrays, but it's useful to check using summaryAffyRNAdeg, which provides a slope estimate and p value to accompany the above plot:

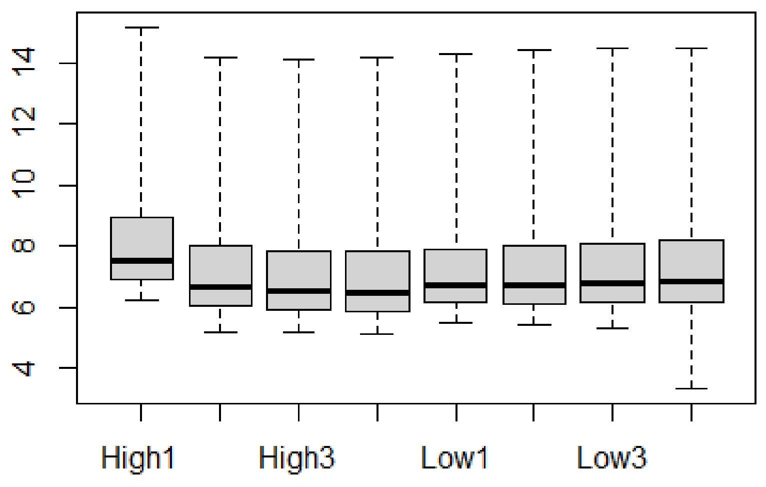
```
summaryAffyRNAdeg(AffyRNAdeg(Data))

##        High1    High2    High3    High4    Low1     Low2     Low3     Low4
## slope  2.40e+00 2.37e+00 2.53e+00 2.09e+00 3.06e+00 3.23e+00 2.97e+00 2.91e+00
## pvalue 3.32e-09 3.72e-10 3.90e-10 1.22e-09 7.00e-11 1.21e-10 3.71e-10 1.02e-10
```

This confirms that RNA degradation appears to be fairly similar across all arrays.

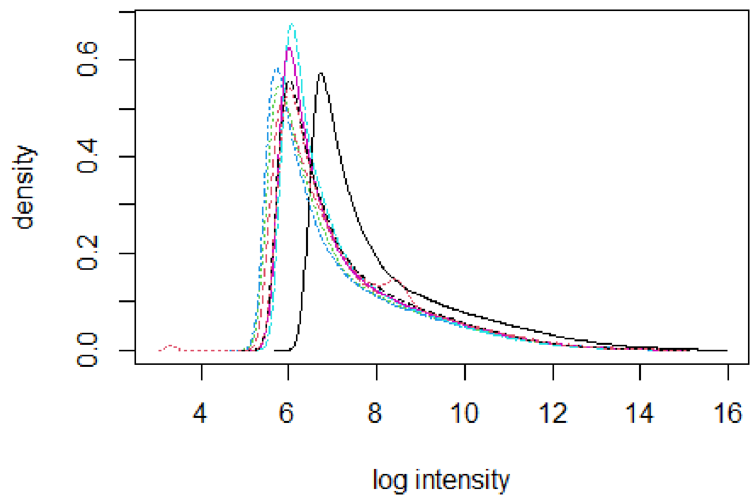
e) Examine the distribution of intensity values for the perfect-match and mismatch probes

Perfect match

```
boxplot(Data,which="pm")
```

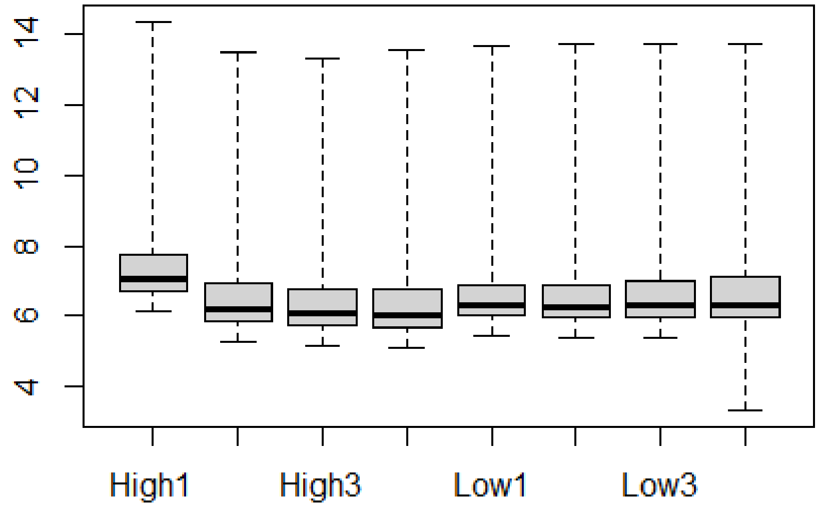


```
plotDensity.AffyBatch(Data,which="pm")
```

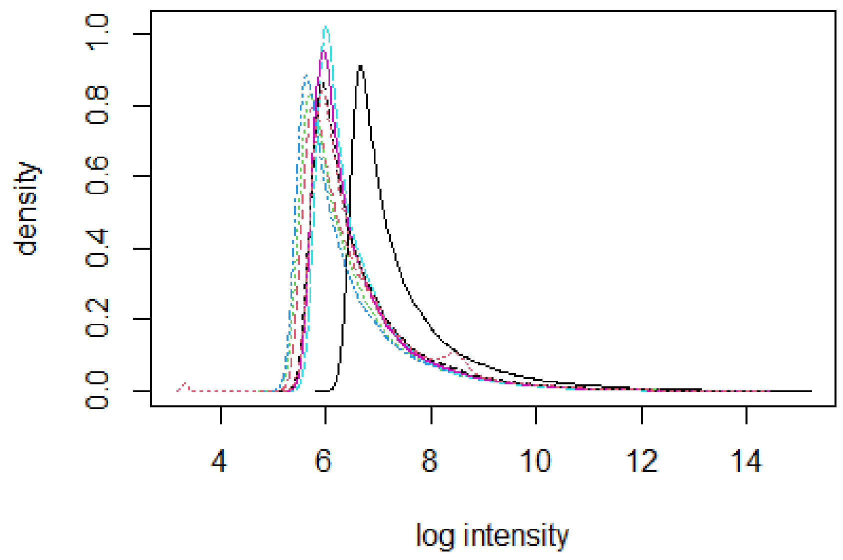


Mismatch

```
boxplot(Data,which="mm")
```



```
plotDensity.AffyBatch(Data,which="mm")
```



For both sets of probes, the intensity tends to be elevated on “High1” compared to the other arrays. The rest of the arrays appear to be in pretty close agreement based on these plots, although “Low4” has a wider range than the others.

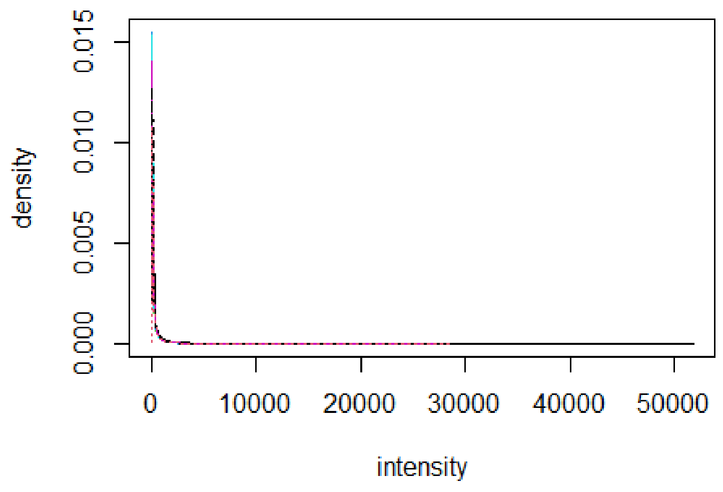
f) Recommendations

Based on the QC plots and tables, “High1” requires further investigation at the very least. Its background level of intensity seems significantly different from the other arrays, but this should be fixed during the normalization step. It might also be worth looking into array “Low4” as well, because the raw image shows clear signs of processing error despite generally passing the QC steps above. Nothing should be excluded yet, but “High1” and “Low4” are candidates for exclusion at later steps and warrant extra attention.

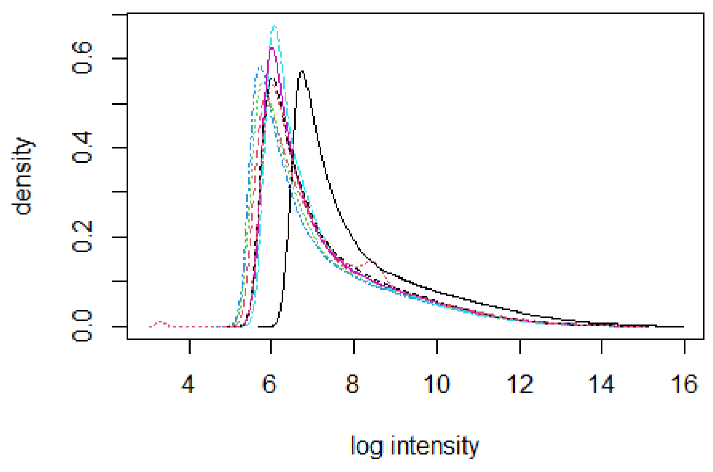
Normalization

a) Plot log-transformed and non-transformed data

```
plotDensity.AffyBatch(Data,log=F)
```



```
plotDensity.AffyBatch(Data)
```

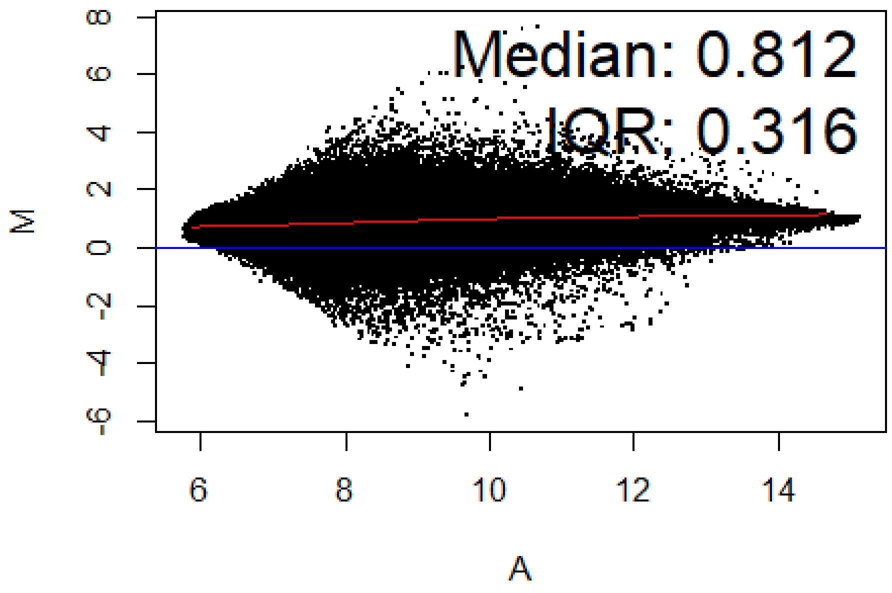


The plot with non-transformed data is almost impossible to read because the data are so skewed and the range is so large. Most intensity values are close to 0, but some are as high as 50,000. Again, one of the arrays seems a little different from the others in the log intensity plot. The distribution is approximately the same shape though, so hopefully normalization will fix this.

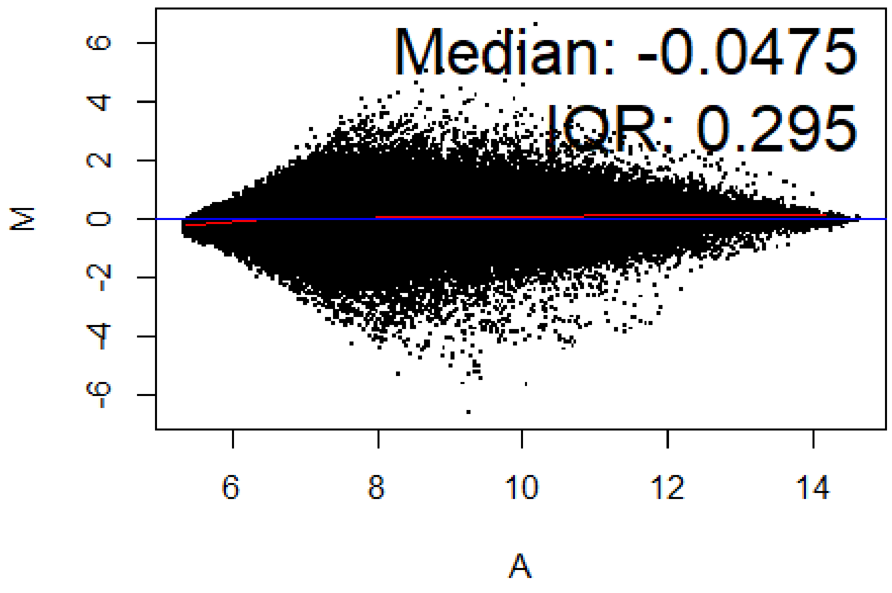
b) MA plots

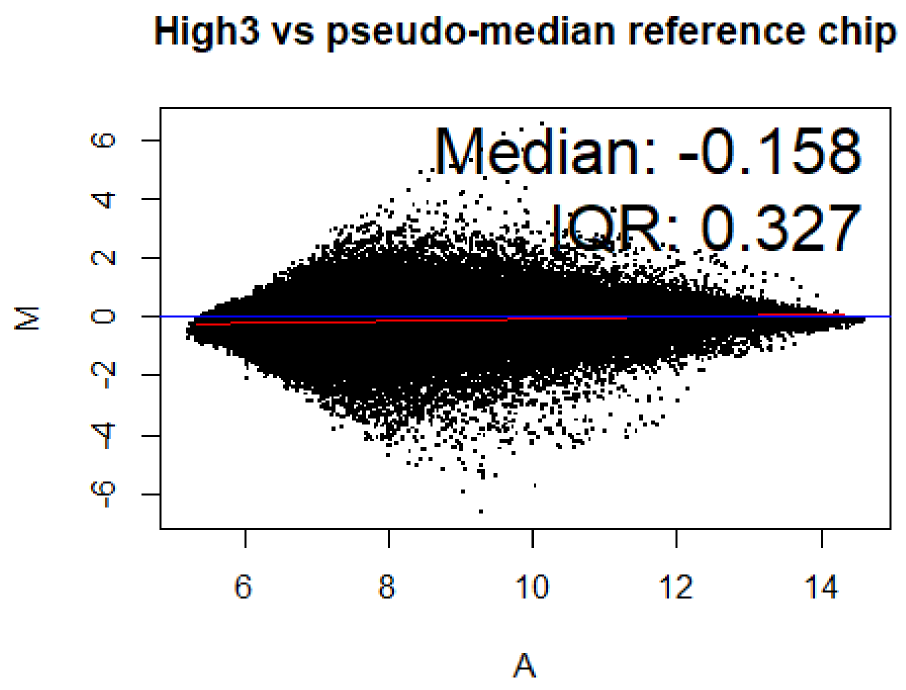
```
MAplot(Data)
```

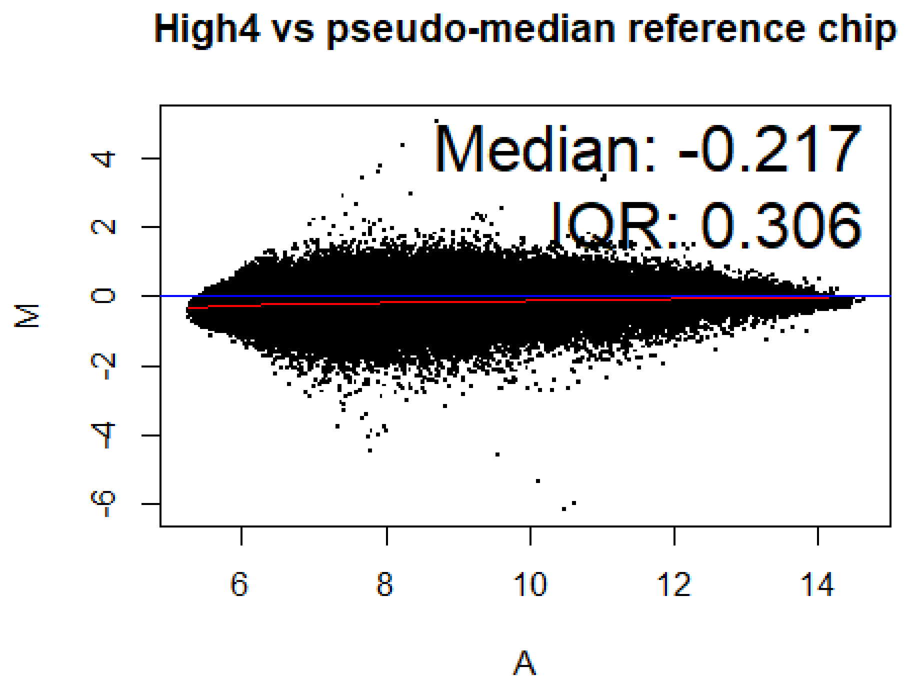
High1 vs pseudo-median reference chip



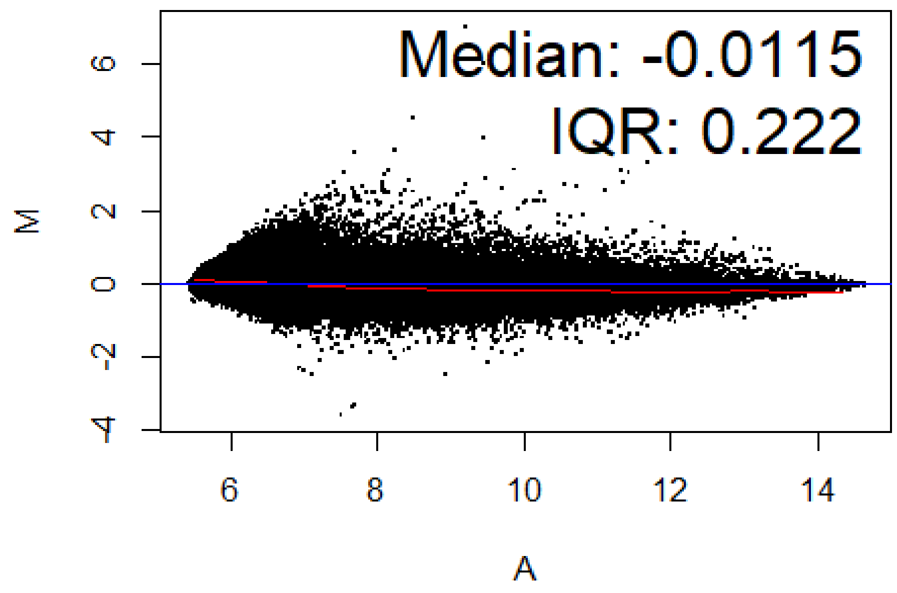
High2 vs pseudo-median reference chip



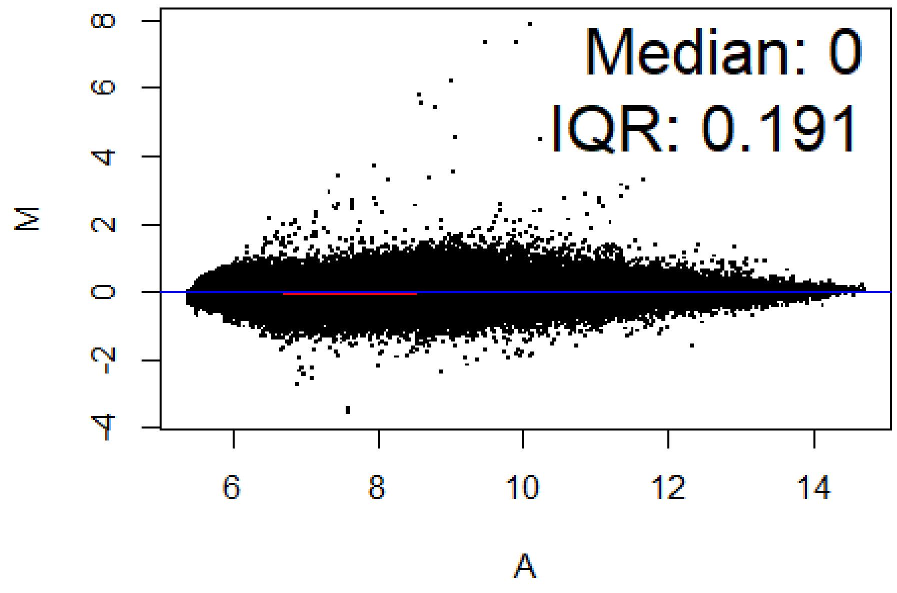

High3 vs pseudo-median reference chip
Median: -0.158
IQR: 0.327
M
A
6
4
2
0
-2
-4
-6
6
8
10
12
14


High4 vs pseudo-median reference chip
Median: -0.217
IQR: 0.306
M
A
4
2
0
-2
-4
-6
6
8
10
12
14

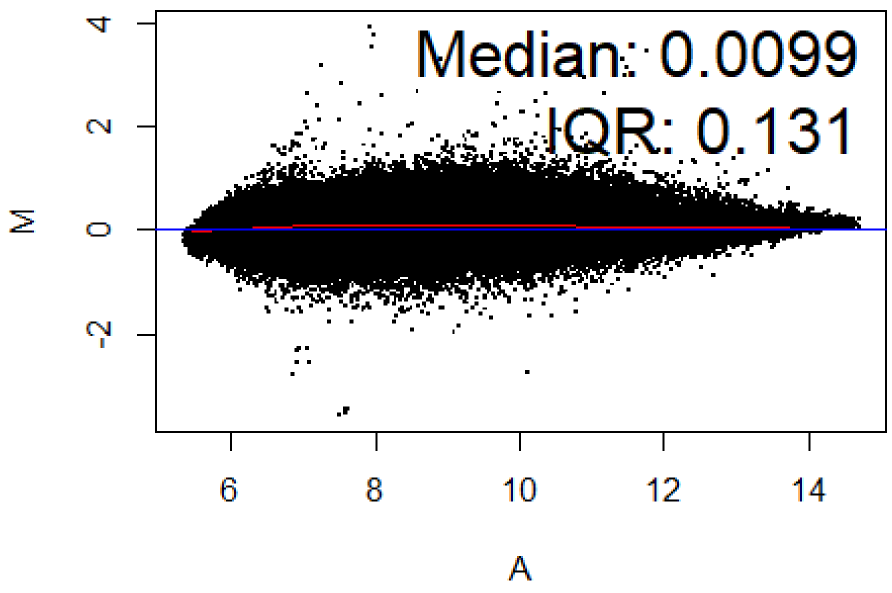
Low1 vs pseudo-median reference chip



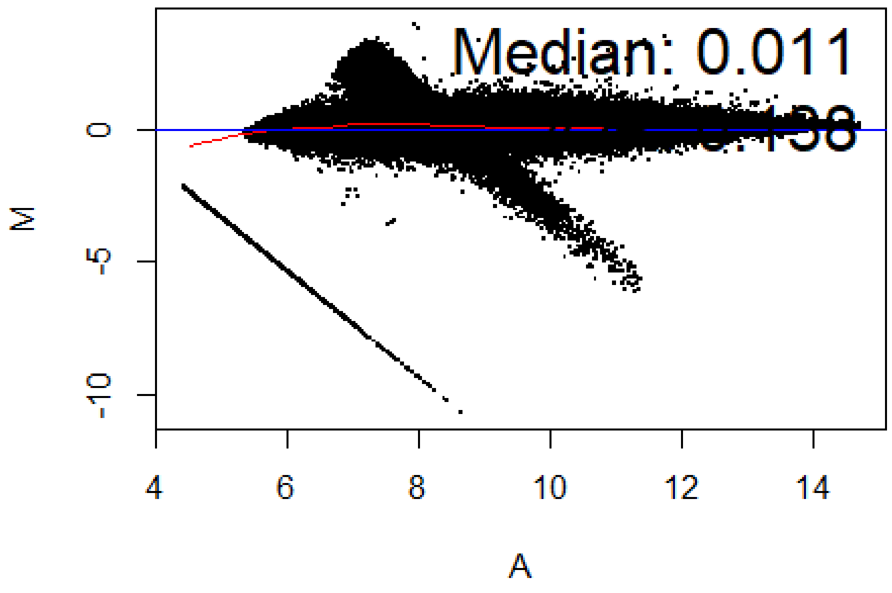
Low2 vs pseudo-median reference chip



Low3 vs pseudo-median reference chip



Low4 vs pseudo-median reference chip



The MA plots generally support the patterns observed in the QC plots, with most of the chips looking reasonably similar. However, "High1" is not clustered around the x-axis and the loess line is a little bit too high. This could be seen during QC as well, with intensity for that array being generally higher than the others, and the large background value in the plot.qc.stats() graph.

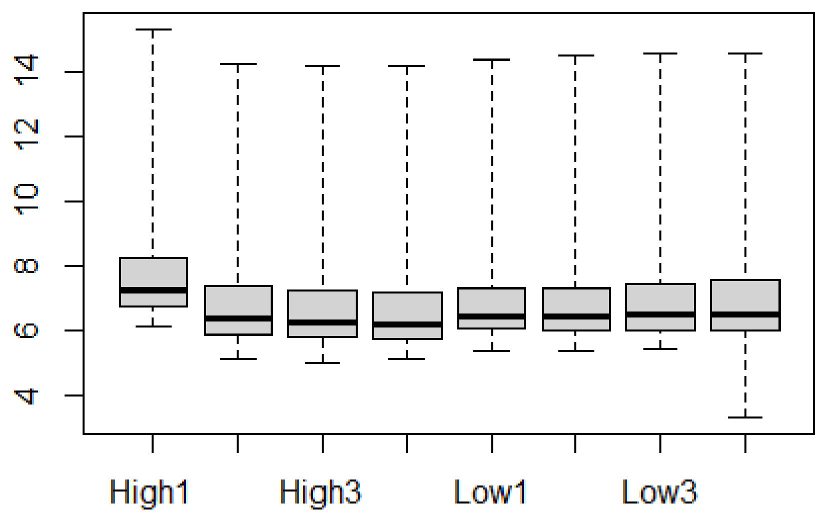
Also, the MA plots clearly show the problems with the “Low4” array. The loess line is actually reasonably close to the x-axis (except maybe in the left tail), which might explain why some of the QC intensity plots didn’t look too bad. However, there are points that form a thin straight line and another cloud, both at a pretty severe angle to the x-axis. You would have to assume that these are due to the obvious array defects that you can see in the raw image.

c) Using expresso(), try different methods

Changing the normalization method

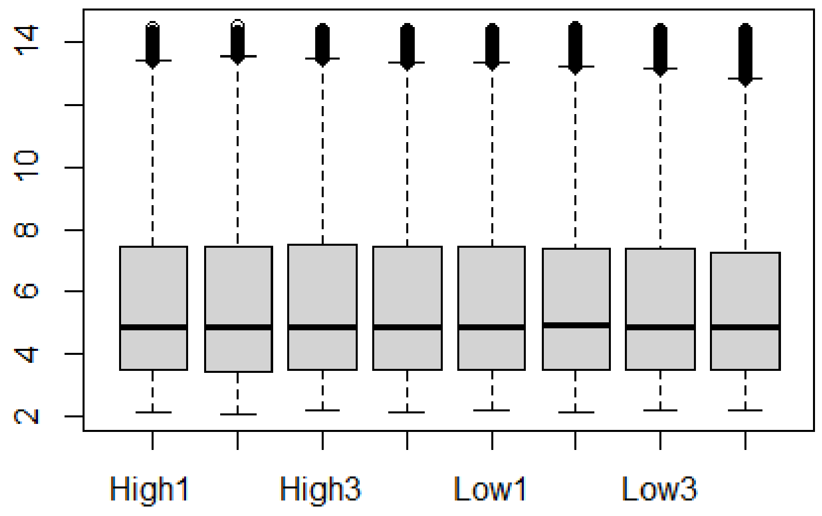
Non-normalized

```
boxplot(Data)
```



Quantiles

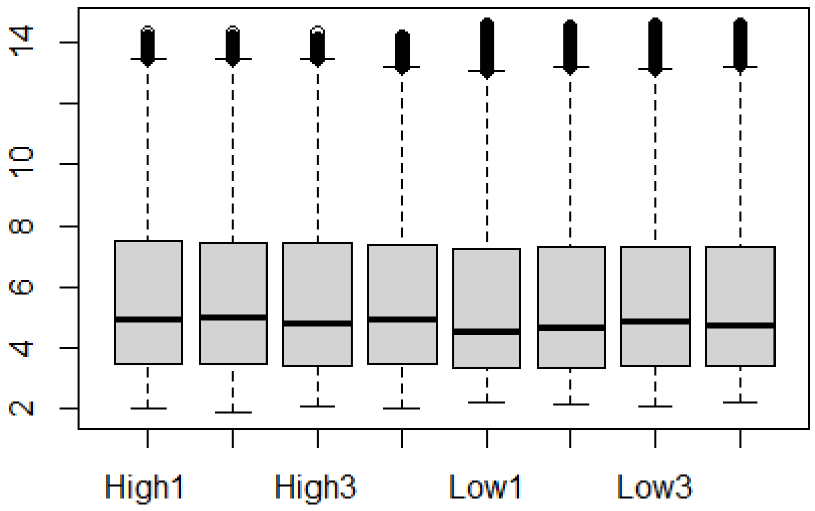
```
quantile_data <- expresso(Data,normalize.method="quantiles",
                          summary.method="avgdiff",
                          bgcorrect.method="rma",
                          pmcorrect.method="pmonly",
                          verbose=F)

## 54675 ids to be processed
## |                    |
## |####################|

boxplot(log(exprs(quantile_data),base=2))
```



Loess

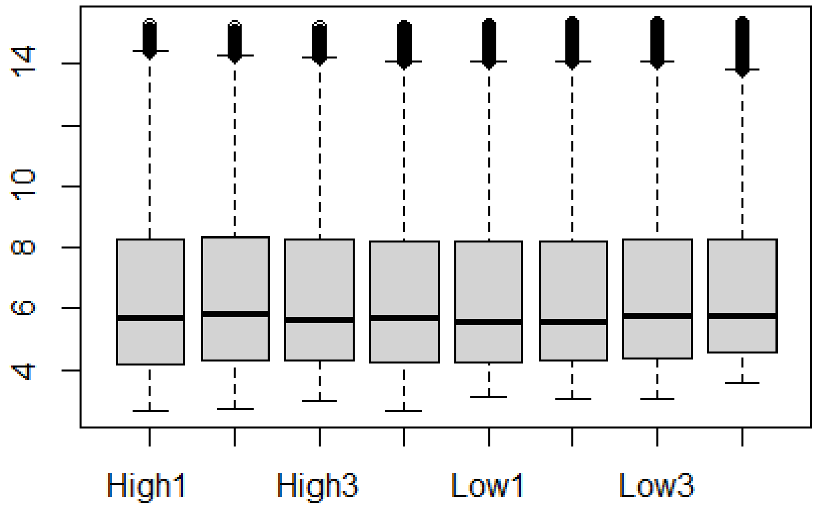
```
loess_data <- expresso(Data,normalize.method="loess",
                       summary.method="avgdiff",
                       bgcorrect.method="rma",
                       pmcorrect.method="pmonly",
                       verbose=F)

## Done with 1 vs 2 in iteration 1
## Done with 1 vs 3 in iteration 1
## Done with 1 vs 4 in iteration 1
## Done with 1 vs 5 in iteration 1
## Done with 1 vs 6 in iteration 1
## Done with 1 vs 7 in iteration 1
## Done with 1 vs 8 in iteration 1
## Done with 2 vs 3 in iteration 1
## Done with 2 vs 4 in iteration 1
## Done with 2 vs 5 in iteration 1
## Done with 2 vs 6 in iteration 1
## Done with 2 vs 7 in iteration 1
## Done with 2 vs 8 in iteration 1
## Done with 3 vs 4 in iteration 1
## Done with 3 vs 5 in iteration 1
## Done with 3 vs 6 in iteration 1
## Done with 3 vs 7 in iteration 1
## Done with 3 vs 8 in iteration 1
## Done with 4 vs 5 in iteration 1
## Done with 4 vs 6 in iteration 1
## Done with 4 vs 7 in iteration 1
## Done with 4 vs 8 in iteration 1
## Done with 5 vs 6 in iteration 1
## Done with 5 vs 7 in iteration 1
## Done with 5 vs 8 in iteration 1
## Done with 6 vs 7 in iteration 1
## Done with 6 vs 8 in iteration 1
## Done with 7 vs 8 in iteration 1
## 1 0.6082201
## 54675 ids to be processed
## |                    |
## |####################|
```

```
boxplot(log(exprs(loess_data),base=2))
```



Constant

```
constant_data <- expresso(Data,normalize.method="constant",
                          summary.method="avgdiff",
                          bgcorrect.method="rma",
                          pmcorrect.method="pmonly",
                          verbose=F)

## 54675 ids to be processed
## |                    |
## |####################|

boxplot(log(exprs(constant_data),base=2))
```



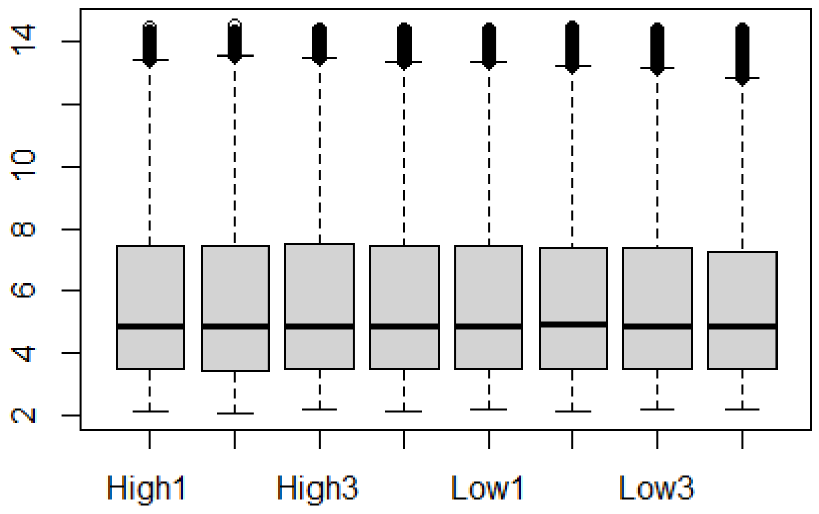
Based on visual inspection of these boxplots, no normalization method appears to perform better than the others (at least not with summary.method="avgdiff", bgcorrect.method="rma", and pmcorrect.method="pmonly"). All of the arrays look similar to each other, and with all methods "High1" and "Low4" look much improved. However, the quantile method is much faster than the others. The loess method is particularly slow and should be avoided unless the other two methods cannot be used for some reason.

Changing the summary method

The following plots are generated using the quantile normalization method, pmcorrect.method="pmonly", and bgcorrect.method="rma" with different options for summary.method.

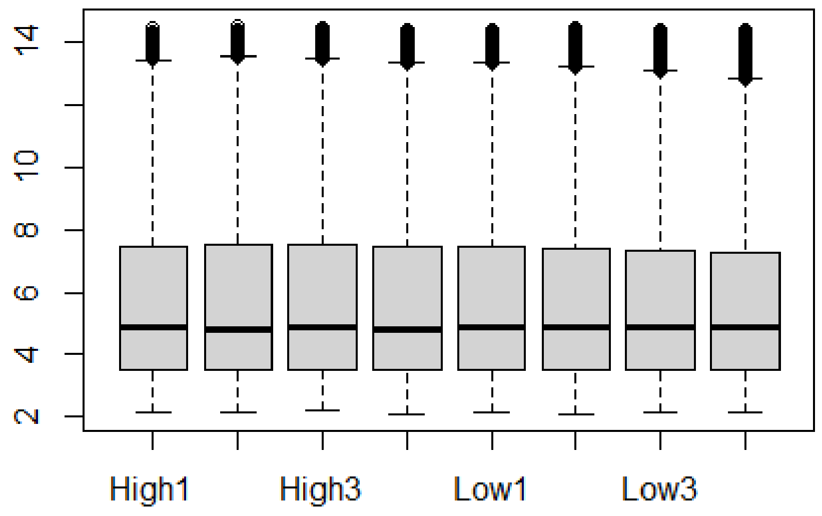
AvgDiff

```
boxplot(log(exprs(quantile_data),base=2))
```



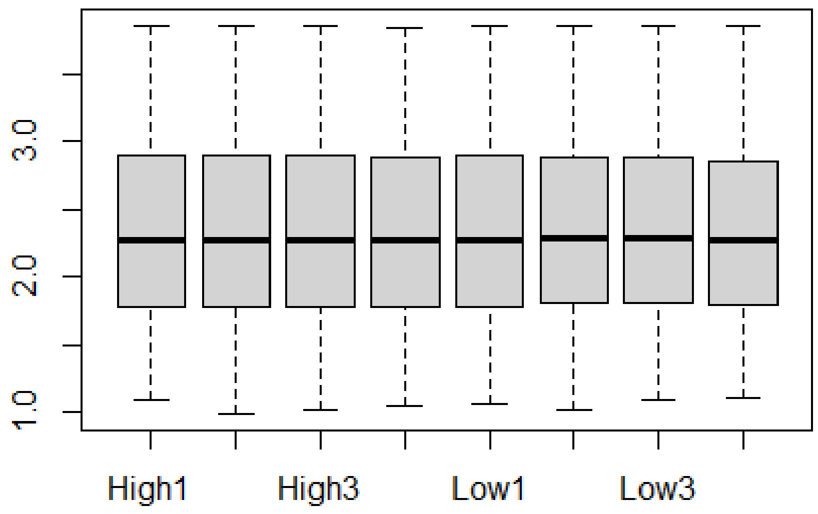
MAS

```
mas_data <- expresso(Data,summary.method="mas",
                     normalize.method="quantiles",
                     bgcorrect.method="rma",
                     pmcorrect.method="pmonly",
                     verbose=F)

## 54675 ids to be processed
## |                    |
## |####################|

boxplot(log(exprs(mas_data),base=2))
```



Median polish

```
median_data <- expresso(Data,summary.method="medianpolish",
                        normalize.method="quantiles",
                        bgcorrect.method="rma",
                        pmcorrect.method="pmonly",
                        verbose=F)

## 54675 ids to be processed
## |                    |
## |####################|

boxplot(log(exprs(median_data),base=2))
```



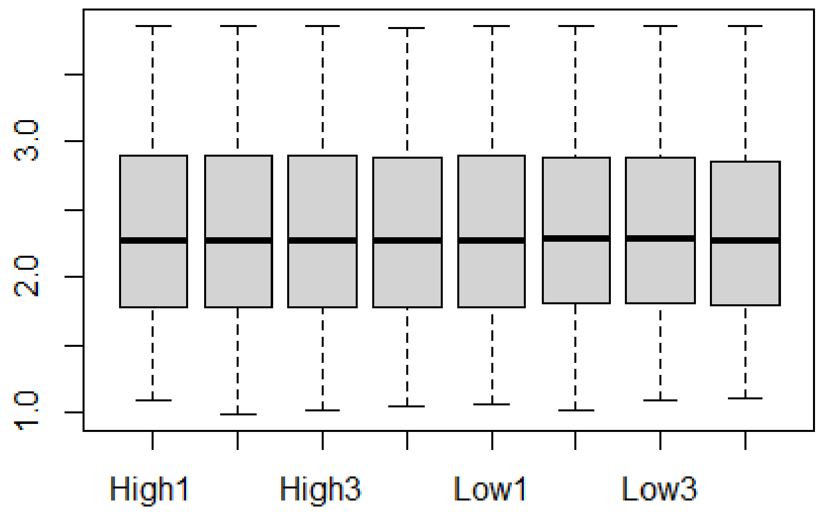
The MAS and AvgDiff boxplots look almost identical, but the median polish method does appear to be different from the other two. The median polish method produces expression values that are more normally distributed than those produced by the other methods. There were not noticeable differences in speed between the methods, so median polish is likely the best of these options (at least with the normalization, background, and PM correction methods selected).

Changing the PM correction method

Using the quantile normalization method, summary.method="medianpolish", and bgcorrect.method="rma".

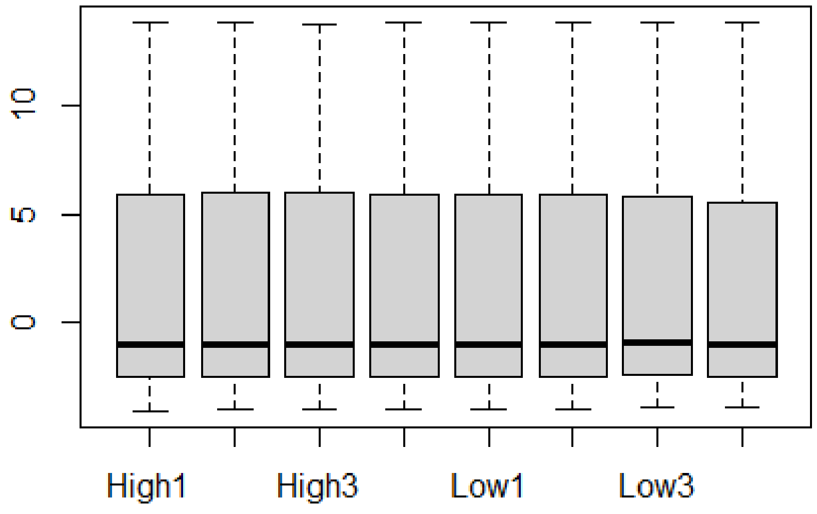
PM Only

```
pmonly_data <- expresso(Data,pmcorrect.method="pmonly",
                  summary.method="medianpolish",
                  normalize.method="quantiles",
                  bgcorrect.method="rma",
                  verbose=F)

## 54675 ids to be processed
## |                    |
## |####################|

boxplot(log(exprs(pmonly_data),base=2))
```



MAS

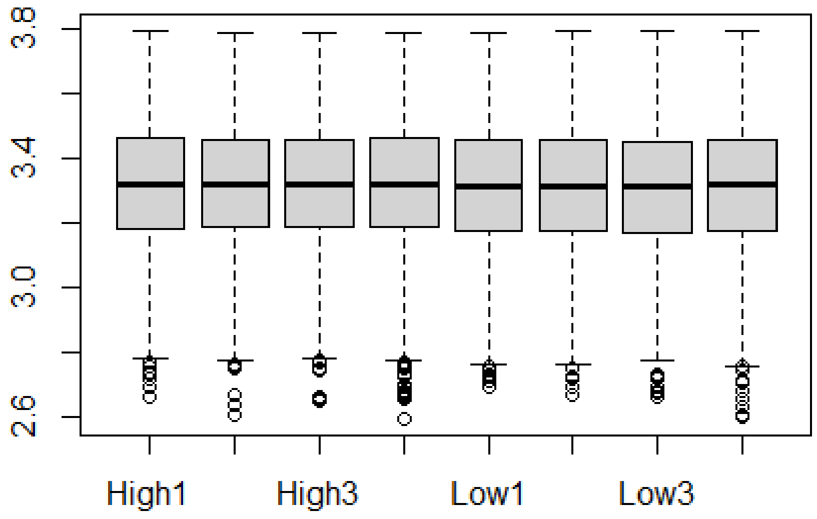
```
mas_data_2 <- expresso(Data,pmcorrect.method="mas",
                  summary.method="medianpolish",
                  normalize.method="quantiles",
                  bgcorrect.method="rma",
                  verbose=F)

## 54675 ids to be processed
## |                    |
## |##############

## Warning in medpolish(log2(x), trace.iter = FALSE, ...): medpolish() did not
## converge in 10 iterations

## #######|

boxplot(exprs(mas_data_2))
```



The plot above is on the natural scale because negative values cause problems for the log transformation. Also, it's concerning that the median polish function did not converge.

Subtract mismatch

```
subtractmm_data <- expresso(Data,pmcorrect.method="subtractmm",
                            summary.method="medianpolish",
                            normalize.method="quantiles",
                            bgcorrect.method="rma",
                            verbose=F)

## 54675 ids to be processed
## |                    |
## |####################|

boxplot(log(exprs(subtractmm_data),base=2))
```



Unfortunately, neither the MAS or subtract MM approaches seem to work well, at least not with the median polish summary method. The subtract MM plot looks reasonable, but it produces a lot of concerning “NaN” warnings that I suppressed in the output above. However, the PM only plot with median polish as the summary method looks excellent.

d) Get present and absent calls

```
# Presence/absence detection
calls <- mas5calls(Data)

## Getting probe level data...
## Computing p-values
## Making P/M/A Calls

# Get p values
calls_sig <- as.data.frame(assayData(calls)$exprs)
# Subset those with at least one significant p value in each group
filtered <- calls_sig %>%
  filter(("P"==High1|"P"==High2|"P"==High3|"P"==High4)&
         ("P"==Low1|"P"==Low2|"P"==Low3|"P"==Low4))
```

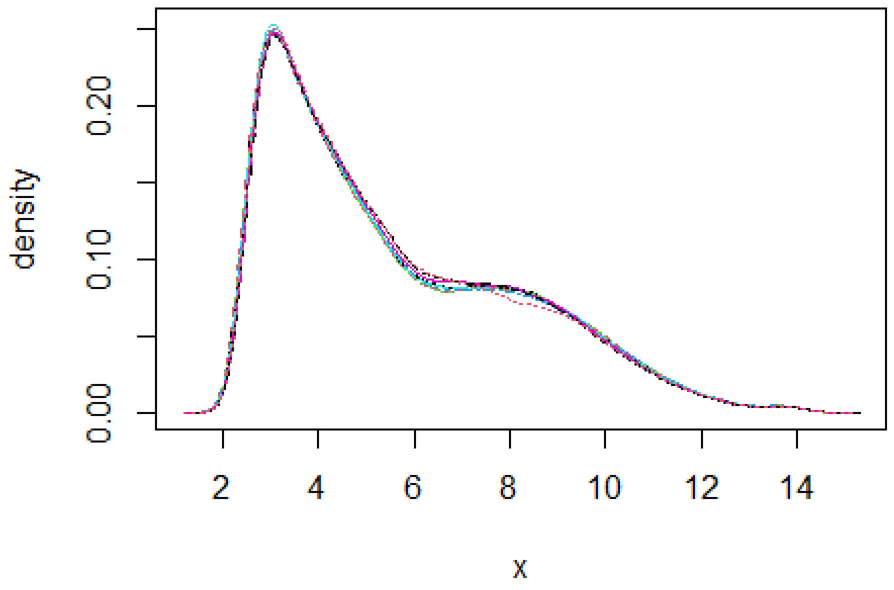
After filtering there are 25726 probesets left out of 54675 (47.1%).

e) Problematic chips

Although most of the boxplots seem reasonable for “Low4,” the MA plot in comparison to the pseudo-median chip suggests that it should be excluded. The bright oval and dark strip evident in the raw image are clearly represented in this plot, and this chip will likely be a detriment during normalization. Because the quantile method uses information from all the chips, it is important to exclude obvious problematic arrays before the normalization step.

However, the potential problems with “High1” appear to have been fixed by the normalization procedures:

```
plotDensity(exprs(pmonly_data))
```



So the most appropriate approach seems to be to exclude “Low4” but keep “High1” in the dataset for analysis.

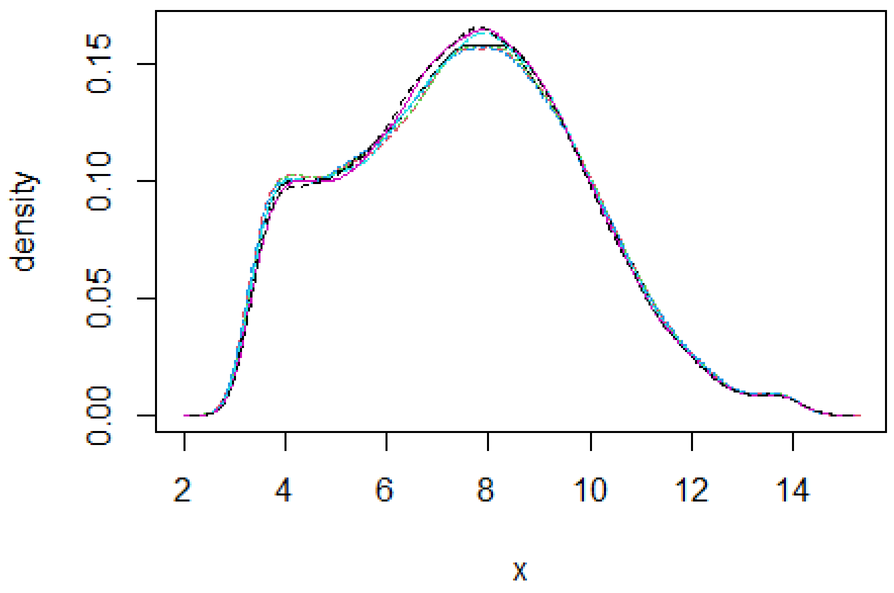
Re-normalization

Import data without “Low4”:

```
pd=read.AnnotatedDataFrame("./targets_low4_excluded.txt" ,
                    header=TRUE,row.names=1,as.is=TRUE)
Data=ReadAffy(filenames=pData(pd)$FileName, phenoData=pd,
          sampleNames=sampleNames(pd))
```

Normalize using the subset from the filtering step d) above, and generate new plots:

```
normalized <- rma(Data,subset=rownames(filtered))

## Background correcting
## Normalizing
## Calculating Expression

plotDensity(exprs(normalized))
```



Save files of the intensities and calls locally:

```
# Save normalized data
save(normalized,file="./normalized.R")
# Get calls and save
calls <- mas5calls(Data,ids=rownames(filtered))

## Getting probe level data...
## Computing p-values
## Making P/M/A Calls

save(calls,file="./calls.R")
```

Show the top lines:

```
kable(head(exprs(normalized)),digits=3)
```

	High1	High2	High3	High4	Low1	Low2	Low3
1007_s_at	9.999	9.833	9.906	9.673	11.425	11.252	11.350
1053_at	9.289	9.302	9.265	9.763	9.145	9.144	9.211
117_at	6.347	6.333	6.210	5.562	5.653	5.729	5.579
121_at	7.204	7.049	7.071	7.214	7.193	7.166	7.406
1294_at	5.188	5.154	5.336	5.128	5.217	5.304	5.303
1316_at	4.091	4.076	4.121	4.091	4.039	4.196	4.164

```
kable(head(assayData(calls)$exprs),digits=3)
```

	High1	High2	High3	High4	Low1	Low2	Low3
1007_s_at	P	P	P	P	P	P	P
1053_at	P	P	P	P	P	P	P
117_at	P	P	P	P	P	P	P
121_at	P	P	P	P	P	P	P
1294_at	P	A	P	A	M	P	P
1316_at	P	P	P	A	A	P	M