An Introduction to Some Graphics in Bioconductor

June 4, 2003

Introduction

We first need to set up the basic data regarding the genome of interest. The `chromLocation` class describes the necessary components for plotting expression data in its chromosomal location. To do the plotting we need to know how many chromosomes there are, what their lengths are and various other pieces of information. Additionally mappings from the probe identifiers to chromosomes and specific locations on chromosomes is also needed. These data are all contained in the Bioconductor meta-data packages and one of these packages can be used to create an instance of the `chromLocation` class.More details on how to do this and examples are given in the vignettes for the *geneplotter* package.

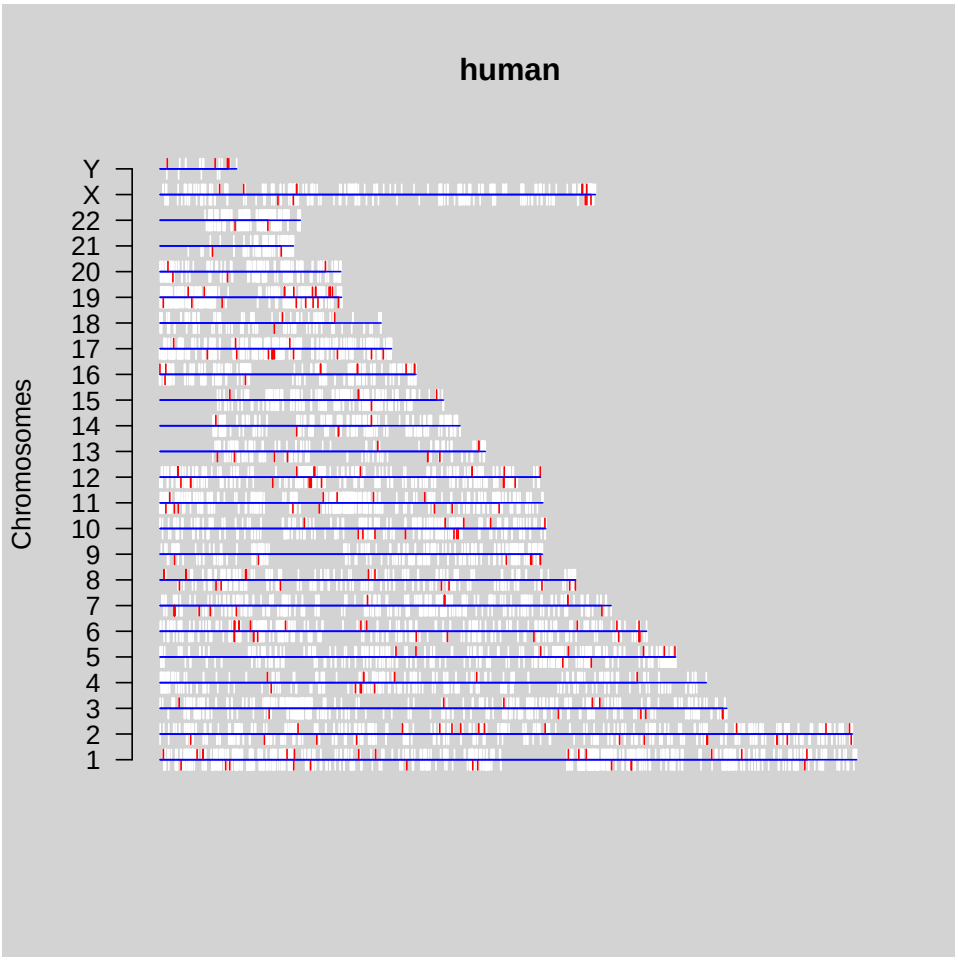
```
> library(annotate)
> library(geneplotter)
> library(hgu95av2)
> newChrom <- buildChromLocation("hgu95av2")
> newChrom

Instance of a chromLocation class with the following fields:
        Organism:  human
        Data source:  hgu95av2
        Number of chromosomes for this organism:  24
        Chromosomes of this organism and their lengths in base pairs:
                 1 : 245172244
                 2 : 242772882
                 3 : 199087371
                 4 : 191589910
                 5 : 180746422
                 6 : 170567309
                 7 : 158124502
                 8 : 145883541
                 9 : 134075627
```

```
10 : 135284517
11 : 134282883
12 : 133383967
13 : 114069855
14 : 105231991
15 : 99787871
16 : 89825221
17 : 81318073
18 : 77653635
19 : 63750318
20 : 63567468
21 : 46913208
22 : 49339342
X : 152601851
Y : 50314613
```

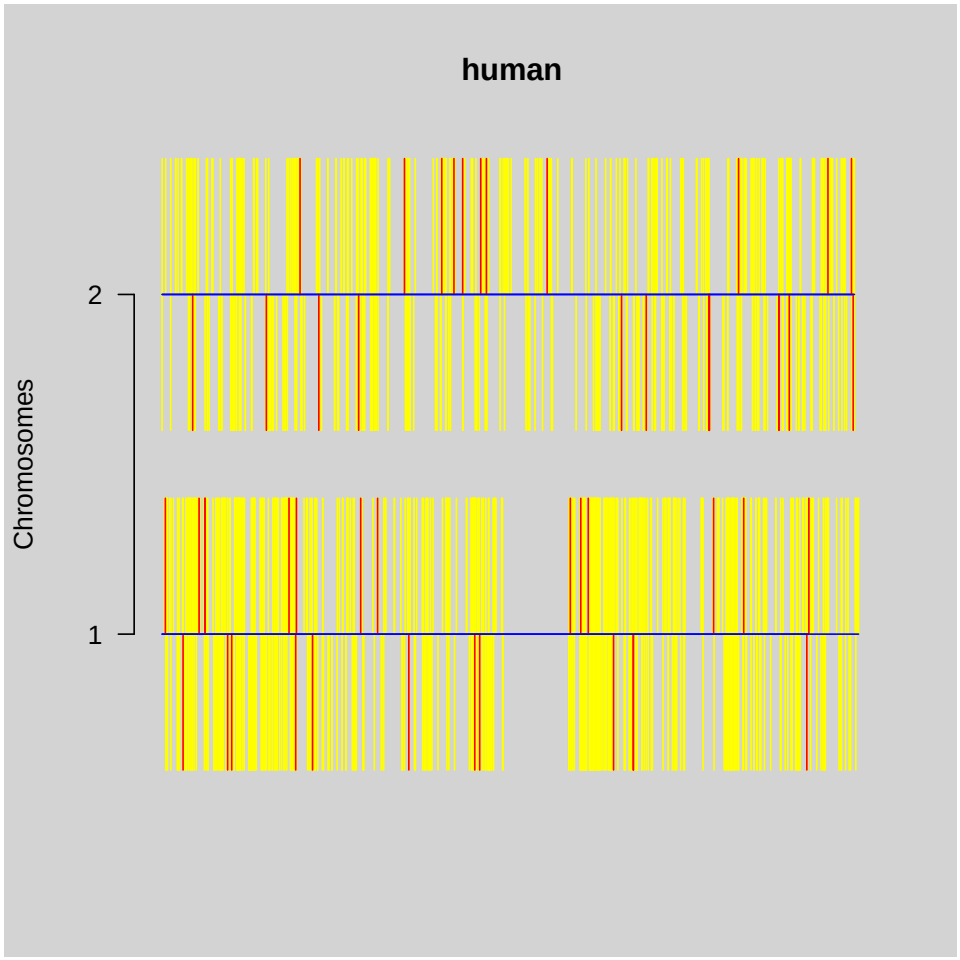
Whole Genome Plots

We have a number of different plotting features available to us. The functions `cPlot` and `cColor` provide plots where the chromosomes are laid out (3' to 5') in a linear fashion and the genes are displayed as short perpendicular lines. Lines that go up indicate genes that are located on the sense strand while lines that go down indicate genes that are located on the antisense strand. The perpendicular lines can be colored differently. In some cases you could color those genes high in one group red and those that were highly expressed in another group blue.

You can plot all chromosomes for an organism or any selected subset of the chromosomes. Chromosomes can be rendered using their relative lengths or scaled to fill the whole plot. Again, more details are available in the manual pages and the vignettes of the *geneplotter* package.



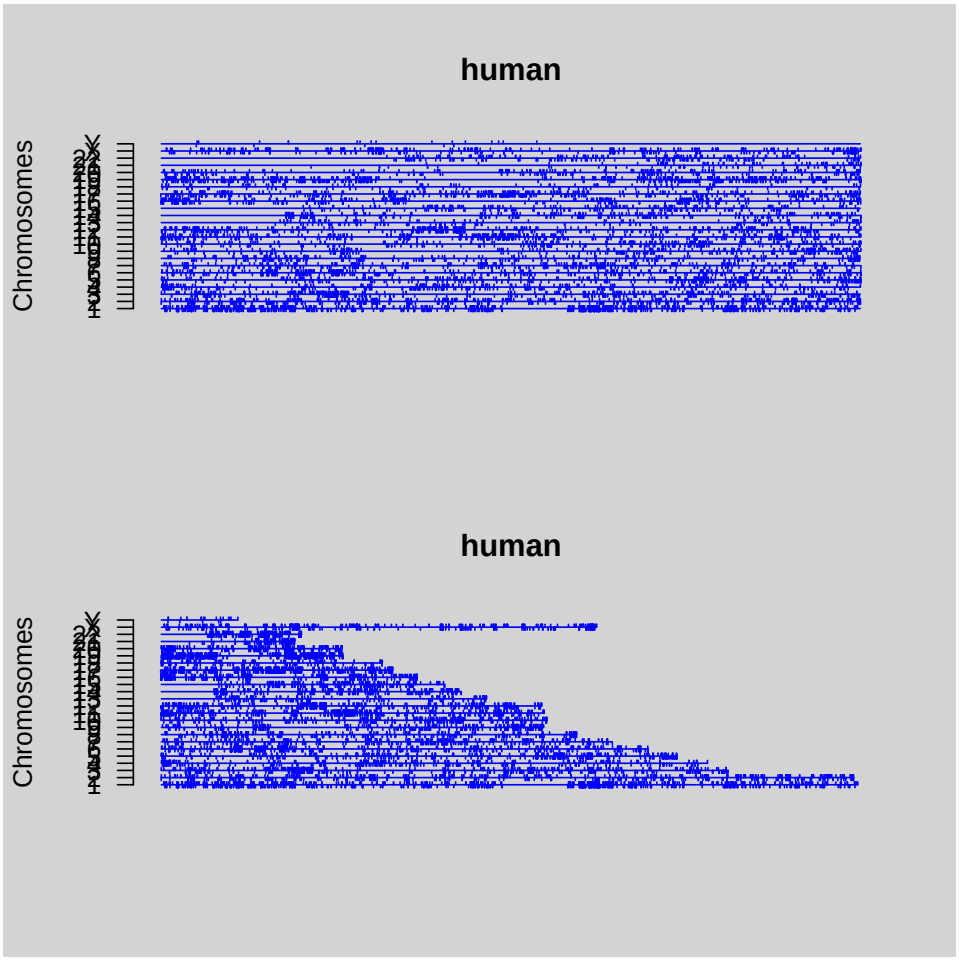
Specific genes can be plotted in color if desired. We can now add the locations of the genes contained in the test data set `eset`. Alternatively, you can restrict attention to just



a specific set of chromosomes.

In this next plot we show the differences between the two scalings that can be used for the chromosomes. These are relative and max, where all chromosomes are presented as being the same length.

```
> par(mfrow = c(2, 1))
> for (sc in c("max", "relative")) cPlot(newChrom, fg = "blue",
+     scale = sc)
```



There are many other sources of information that could be easily added to these plots. Particularly things such as indicating syntenic regions, regions of high GC content, or CpG islands could all be added to these plots.

1 Single Chromosome Plotting

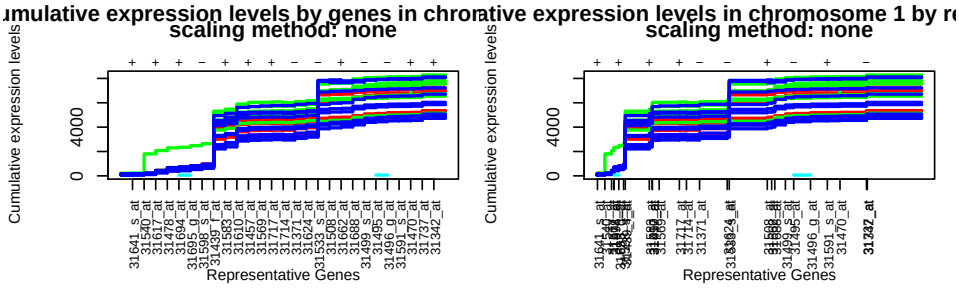
Another type of plot that is sometimes of use is provided by `alongChrom`. The purpose of this plot is to show gene expression values in certain selected regions of a chromosome. Researchers might be interested in regions with known genetic defects or suspected amplifications and deletions.

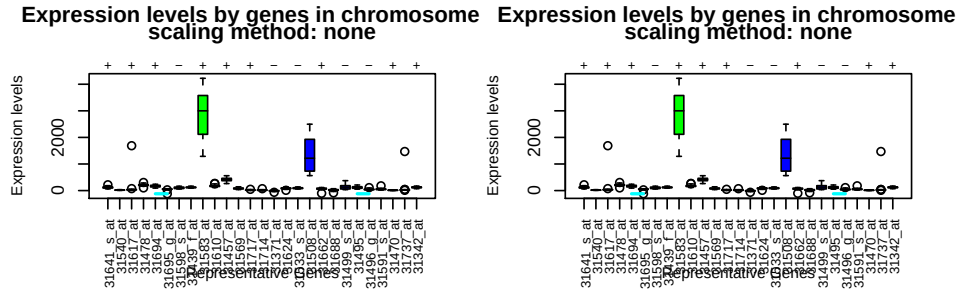
```
> cols <- c("red", "green", "blue")
> cols <- cols[eset$cov3]
> par(mfrow = c(3, 2))
> alongChrom(eset, "1", newChrom, xloc = "equispaced", plotFormat = "cumulative",
+     col = cols, lwd = 2)

<environment: 0xa98ef8c>
```

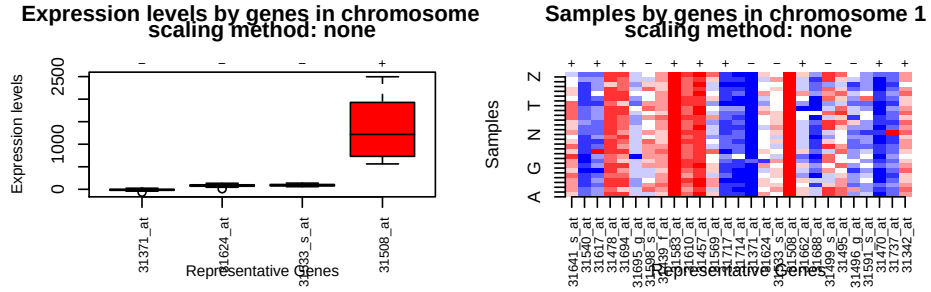
```
> alongChrom(eset, "1", newChrom, xloc = "physical", col = cols,
+     lwd = 2)
```

```
<environment: 0xa8d98b4>
```

```
> alongChrom(eset, "1", newChrom, xloc = "equispaced", plotFormat = "local",
+     col = cols, lwd = 2)
> alongChrom(eset, "1", newChrom, xloc = "equispaced", plotFormat = "local",
+     col = cols, type = "p", pch = 16)
> alongChrom(eset, "1", newChrom, xlim = c(87511280, 127717880),
+     xloc = "equispaced", plotFormat = "local", col = cols, type = "p",
+     pch = 16)
> alongChrom(eset, "1", newChrom, xloc = "equispaced", plotFormat = "image")
```







2 Heatmaps and other tools

As of release 1.7.0 of R there is a `heatmap` function available. Heatmaps are interesting data displays made popular for genomic data by **?**. The ideas are older and can be seen

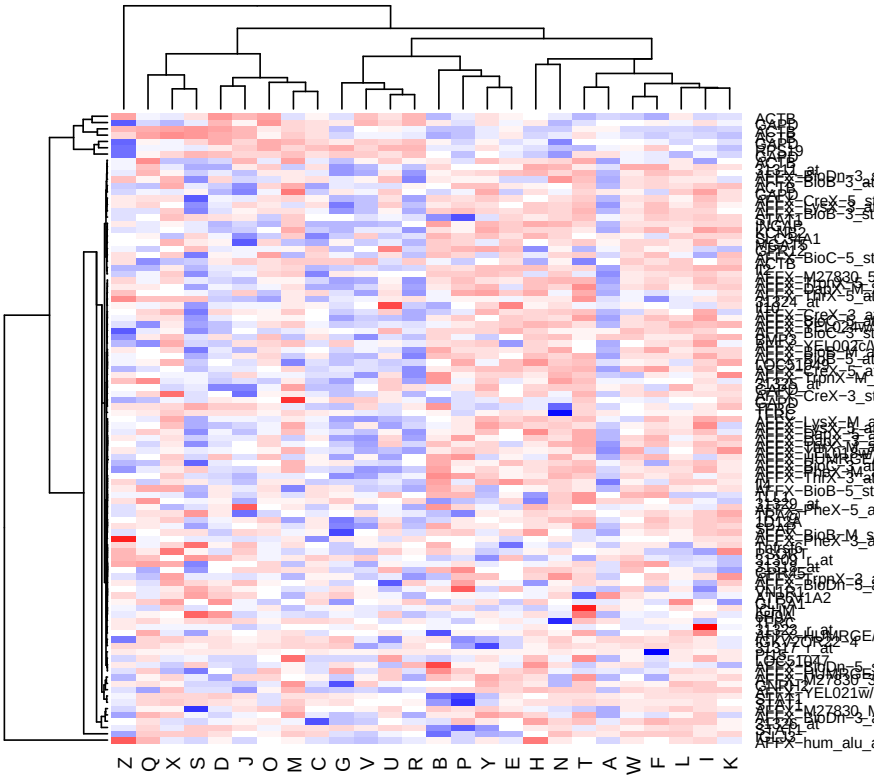
in work of Bertin (and other sources).

Basically a heatmap is a false color display where the rows and the columns have been permuted to show *interesting* patterns. In the R implementation the ordering is carried out by sorting the data using a hierarchical clustering algorithm.

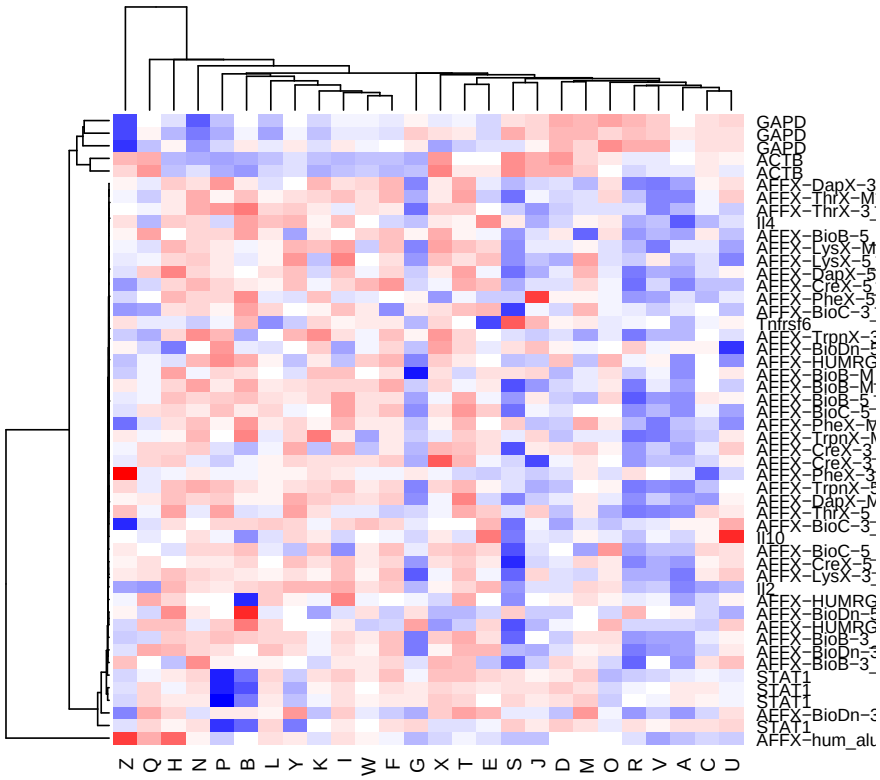
In the code segment below we extract the expression values from the test data set, `eset`, and then get meaningful names for the genes. We do this by finding the mapping between the probe identifiers that are stored with the data and the gene symbols.

The dendrograms on the top and side are created by the clutering function. By default this is hierarchical clustering with complete linkage.

```
> data(eset)
> Exprs <- exprs(eset)
> gN <- geneNames(eset)
> syms <- getSYMBOL(gN, "hgu95av2")
> syms <- ifelse(is.na(syms), gN, syms)
> row.names(Exprs) <- syms
> heatmap(Exprs[1:100, ], col = rev(dChip.colors(50)))
```



We can change the behavior by simply defining a function that will do single linkage clustering. Notice the usual oddities of single-linkage clustering. The plot suggests that there is a group of about 5 genes that are quite close to each other (at the top) and one off by itself at the bottom. Additionally, one of the samples seems rather far from the others. Single linkage clustering is a good mechanism for finding outliers – and hence is probably



a plot worth examining.

As for the other R functions, there are a host of options that you can set and control. The clustering mechanism, the distance metric used. You can supply your own row or column dendrograms and directly manipulate the plots.

Choosing a good set of colors is an important part of the construction of any visualization tool. We recommend that you look at the work of Cynthia Brewer (`www.colorbrewer.org`) as a source of reasonable palettes. Her work is also available in the package *RColorBrewer* available from CRAN.

Visualizing Distances

For microarray data and many other types of experimental data you will want to apply different machine learning algorithms during the analysis. All such algorithms (either

clustering or classification) depend on some notion of distance between the objects being clustered or classified. There is **no** *a priori* best distance. You will need to carefully consider the metric that is best for the data and the classifier being used.

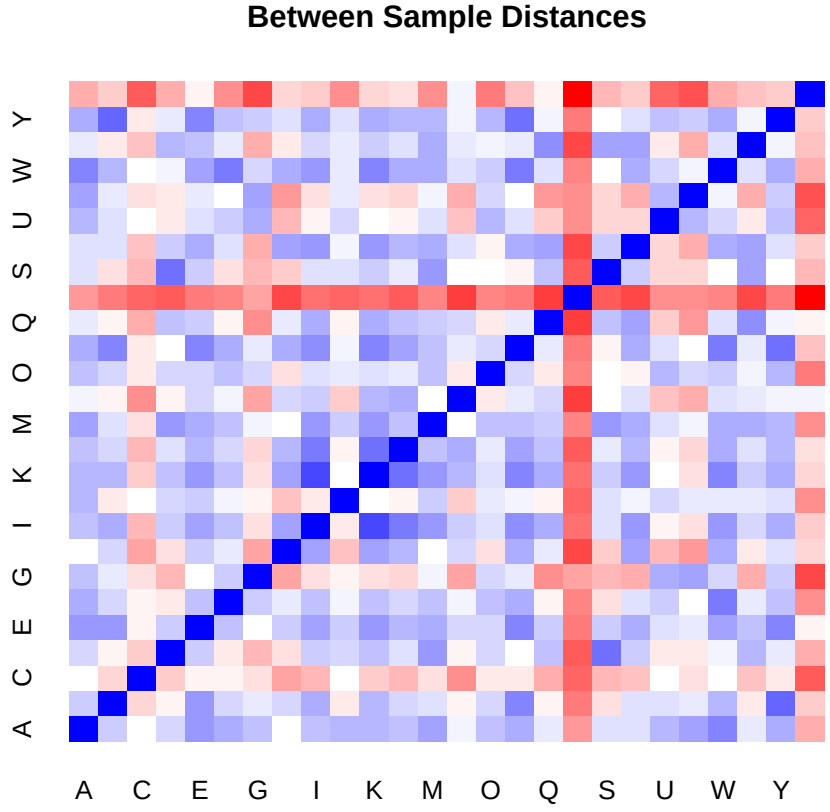
The resulting pairwise distances can be hard to comprehend and use. In this section we consider different methods of visualizing distances. We consider three different techniques that have proven useful.

- Plotting the pairwise distance matrix using image or the *ellipse* package.
- Using multidimensional scaling to reduce the dimensionality.
- Using the `heatmap` function on the pairwise distances.

Again, we will carry use the example data in `eset` to demonstrate some of these functionalities.

First we examine using the `image` function and the *ellipse* package.

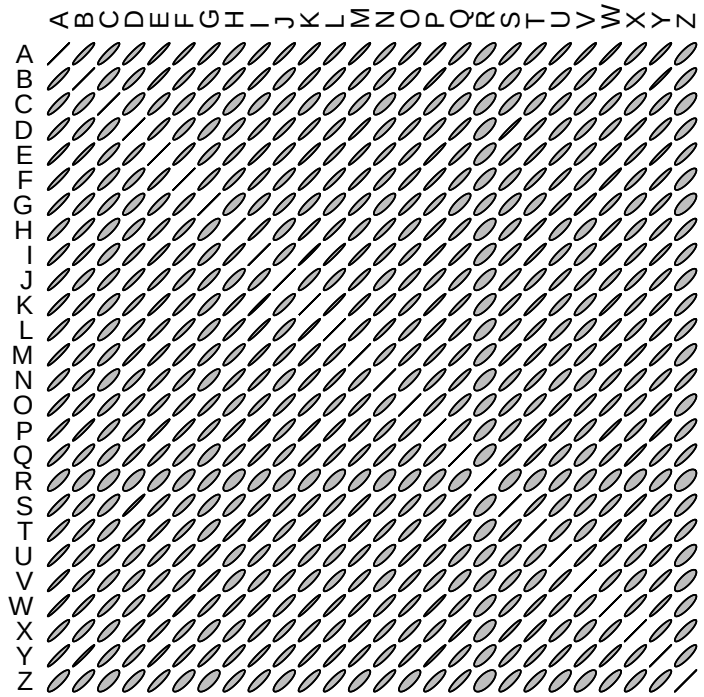
```
> d1 <- dist(t(scale(Exprs)))
> dN <- dimnames(Exprs)[[2]]
> nS <- length(dN)
> d1M <- as.matrix(d1)
> dimnames(d1M) <- list(dN, dN)
> par(mfrow = c(1, 1))
> image(1:nS, 1:nS, d1M, col = dChip.colors(50), axes = FALSE,
+     xlab = "", ylab = "", main = "Between Sample Distances")
> axis(1, at = (1:nS), label = dN, tick = FALSE)
> axis(2, at = (1:nS), label = dN, tick = FALSE)
```



We have been fairly fancy in setting up this plot, but that is one of the advantages of using R as our engine for these analyses. There are many more options that you could explore. Notice that array `R` seems to be quite different from the other arrays. The colors are encoded so that red means a large distance so we see that array `R` is a far from the others.

We can also see this using the ellipse package.

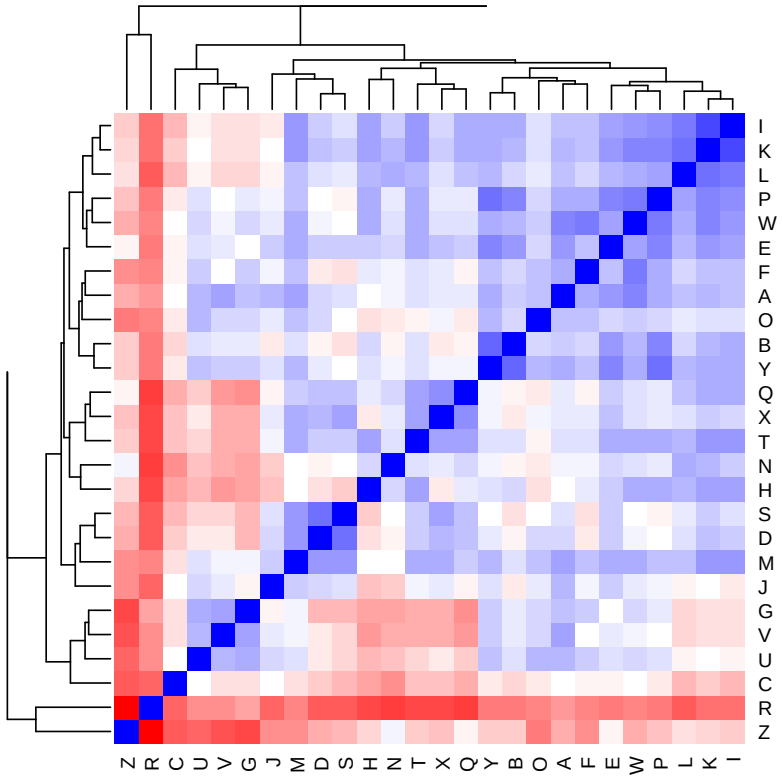
```
> library(ellipse)
> r <- cor(Exprs)
> d2M <- 1 - r
> plotcorr(r, main = "Correlation Matrix for eset data")
```



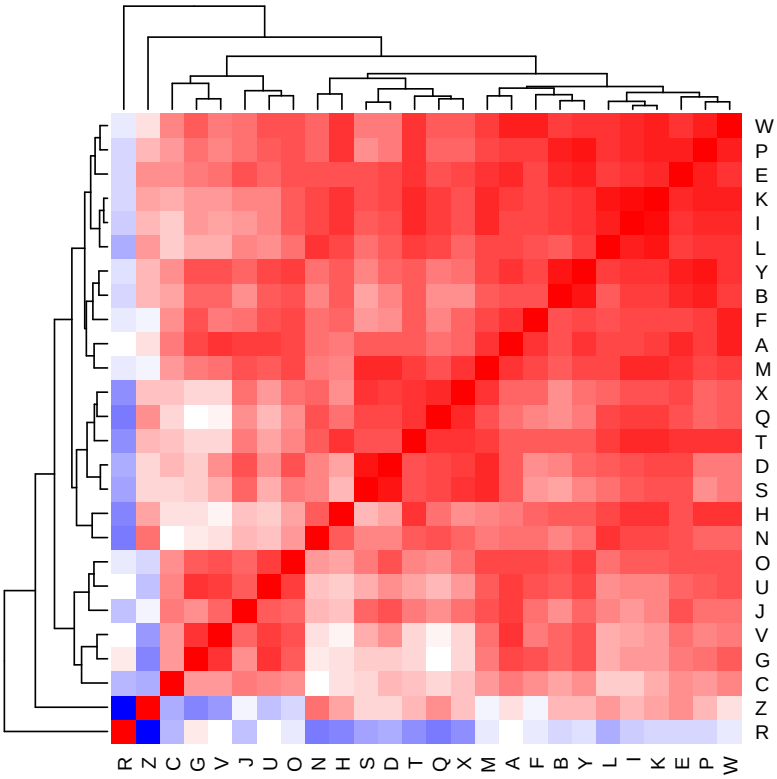
Here we see that the correlation between all samples (for these genes) is positive. The tightness of the plotted ellipse indicates the strength of the correlation. Now we see that array `R` has a lower correlation with other arrays (and hence a larger distance as we saw in the previous plot).

Finally, it is worth drawing a heatmap using the distance matrix. Here we need to make sure that the rows (and or columns) are not scaled. Scaling within rows is the default behavior for heatmap, but for distance matrices it is not appropriate.

```
> heatmap(d1M, col = dChip.colors(50), scale = "none")
```




```
> heatmap(r, col = dChip.colors(50), scale = "none")
```


W
P
E
K
I
L
Y
B
F
A
M
X
Q
T
D
S
H
N
O
U
J
V
G
C
Z
R
R
Z
C
G
V
J
U
O
N
H
S
D
T
Q
X
M
A
F
B
Y
L
I
K
E
P
W