Introduction to R Graphics (BaRC Hot Topics)

George Bell

October 3, 2011

This document accompanies the slides from BaRC's Introduction to R Graphics and shows how to generate all figures in the slides. See the accompanying slides and data files at `http://iona.wi.mit.edu/bio/education/R2011/` or material from other Hot Topics talks at `http://iona.wi.mit.edu/bio/hot_topics/`. All of these commands should work similarly if run on Windows/Mac (using the typical R installation or RStudio) or Linux operating systems (such as tak).

1 Reading datasets used in the presentation

To get started you probably want to get R and your data in the same place. To get R's working directory (for my analysis; your directory will be different), try

```
> getwd()

[1] "/nfs/BaRC/Hot_Topics_2011-2012/R_Bioconductor/class_2/vignette"
```

To change the working directory on your computer, go to File => Change dir... menu. On tak (or you can do it this way on your computer, too), use the `setwd()` command, like `setwd("/lab/solexa_public/Graceland_Lab/Elvis")`

We can see what input data files we have in our directory. You need to download these three files from the Hot Topics page.

```
> dir(pattern=".txt")

[1] "All_vs_with_site.txt"  "color_legend.txt"      "colors.txt"
[4] "Gene_exp_KO_vs_WT.txt" "Gene_exp_with_sd.txt"
```

Let's assume that we have a tab-delimited text file of our dataset of gene expression values ("Gene_exp_with_sd.txt"), the first columns being `Gene` (with letters representing gene symbols), `WT`, and `KO`. The first row contains the column labels. The last two columns are standard deviations of the WT (`WT.sd`) and KO (`KO.sd`) groups, assuming that the gene expression values were assayed with replication. We'll use those later.

One way to read the file is with the `read.delim()` command, Note that we've included `header=T` to indicate that the first row has our column names. If we use `header=F` then the columns will just be numbered (and the data will include the first row of the file).

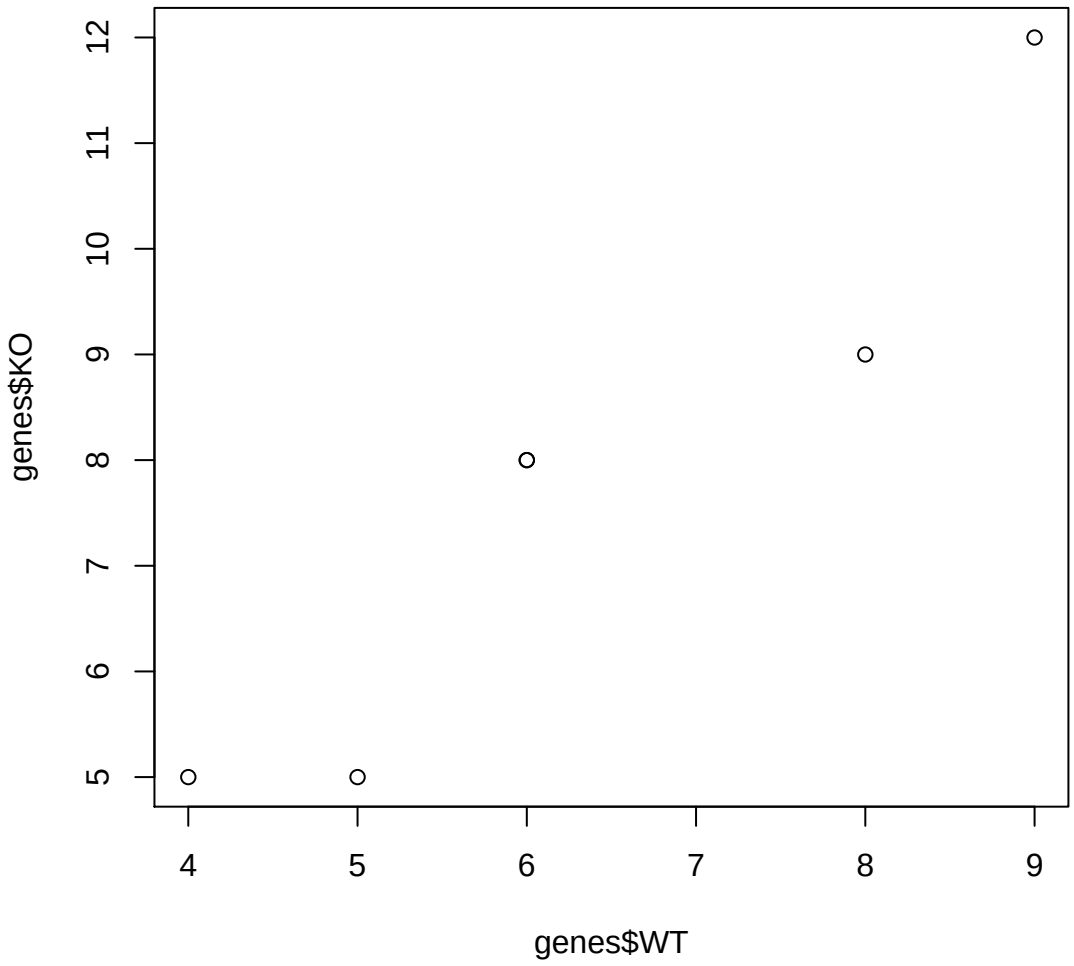
```
> genes = read.delim("Gene_exp_with_sd.txt", header=T)
> genes

  Gene WT KO WT.sd KO.sd
1    A  6  8   0.5   0.6
2    B  5  5   0.4   0.4
3    C  9 12   0.6   0.7
4    D  4  5   0.4   0.4
5    E  8  9   0.6   0.6
6    F  6  8   0.5   0.6
```

Our data looks OK, and the structure of the data is what we wanted.

2 Creating a very simple plot (slide 11)

We can start by creating the most basic scatterplot, using log2 expression values for KO (column 3) vs. WT (column 2) mice. We want to have the WT mouse shown on the x-axis. We can create the plot in at least three different ways, identifying the columns by name (two different ways) or by number. All three commands produce the same graphs with slightly different axis labels.

```
> plot(genes$WT, genes$KO)
> plot(genes[,2], genes[,3])
> plot(genes[,"WT"], genes[,"KO"])
```

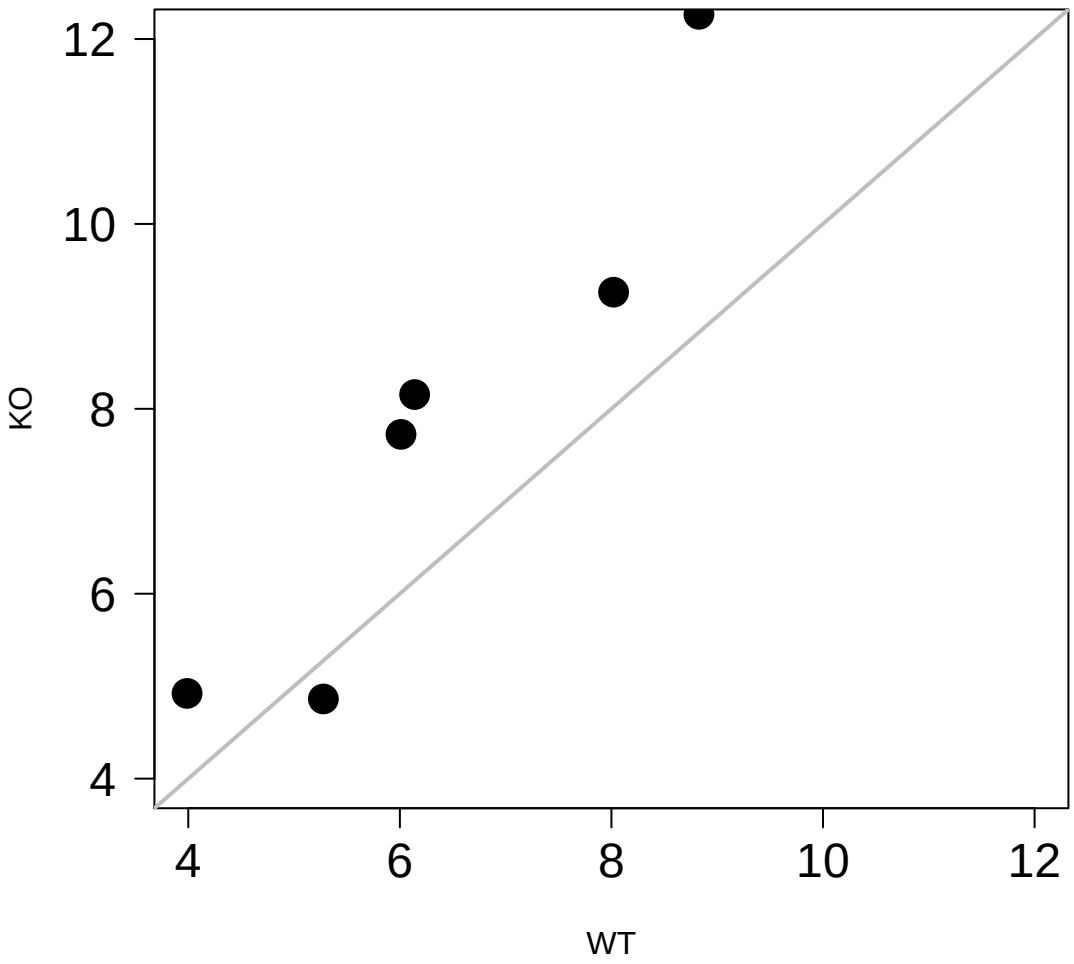


This basic figure has several problems: the graph and axes aren't clearly labeled (although the first command comes close), the graph has no title, and the points are hard to see and are not labeled. The x- and y-axes show different ranges, which may not be what we want. Also, looking back at the data, you'll notice that genes A and F have the same pair of expression values so appear as just one point. We'll cover these issues later when we discuss customizing figures.

3 Comparing sets of numbers (slide 13)

Of course we can graph this dataset in many different ways. For now let's customize our `plot` command and also try a scatterplot by groups and a boxplot. For the customized plot, we're going to change the types of points and add axis labels. We're also going to add some noise ("jitter") to our data so we can see all points. We assume that the noise won't change our data interpretation; otherwise, adding it would be a bad idea. We're also going to add a diagonal line, showing where WT and KO values are equal, which may help with interpretation.

```
> plot(jitter(genes[,2], 1.5), jitter(genes[,3], 1.5), pch=19, las=1, cex=2,
+   cex.axis=1.5, xlab="WT", ylab="KO", xlim=c(4,12), ylim=c(4,12))
> abline(0,1, lwd=2, col="gray")
```



If we run R on our computer, a figure will pop up and can then be saved by right-clicking on it. On tak, a figure will end up in a file called `"Rplots.pdf"`.

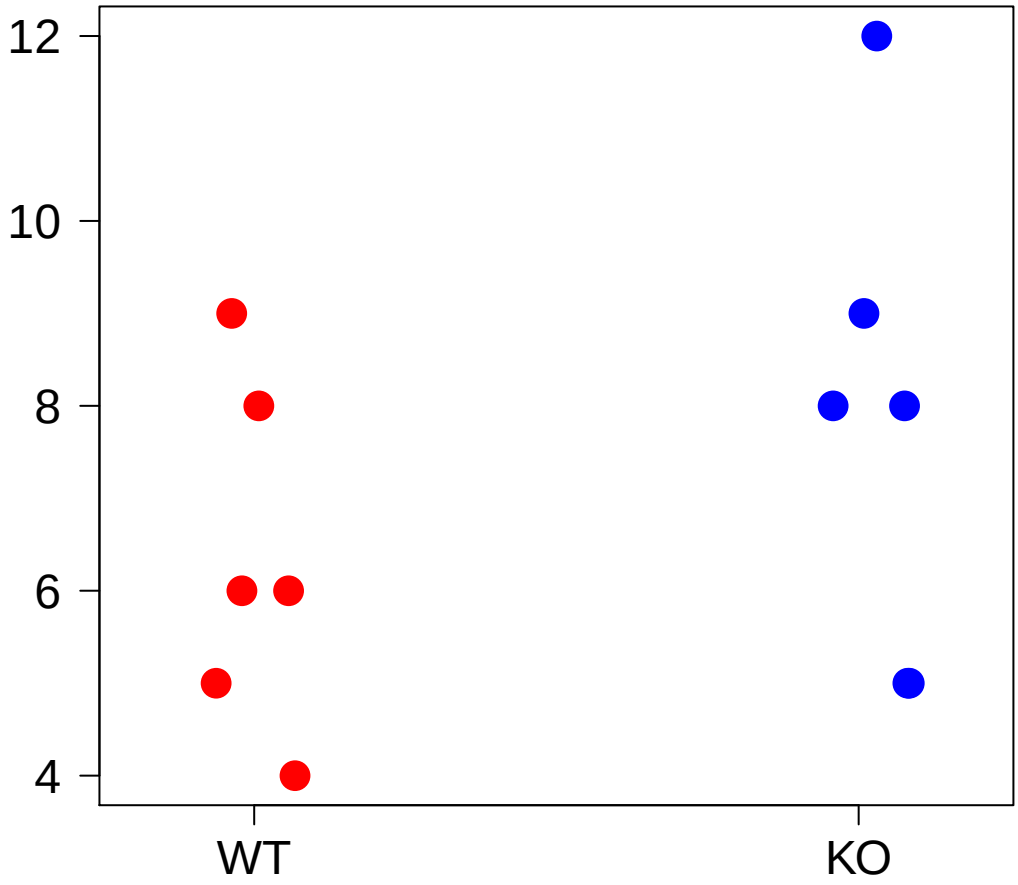
Some other details we've specified:

- `jitter(genes[,3], 1.5)` => add jitter to genes[,3] (with '1.5' indicating the amount of jitter); see `?jitter` for details
- `pch` => the shape of point (which we'll discuss later)
- `las=1` => draw axis scales horizontal; see `?par` for other choices
- `cex=2` => draw points twice the usual size
- `cex.axis=1.5` => draw axis labels 1.5x the usual size
- `xlim=c(4,12)` => create an x-axis scale from 4 to 12

Hopefully the other details are self-explanatory.

If our data wasn't paired (in other words, if we had one set of WT values and another set of KO values), we could do a grouped scatterplot using the `stripchart` command. This graph takes its group labels from column headers of `genes`.

```
> stripchart(genes[,2:3], vert=T, method="jitter", jitter=0.1,
+ col=c("red","blue"), pch=19, cex=2, cex.axis=1.5, las=1)
```

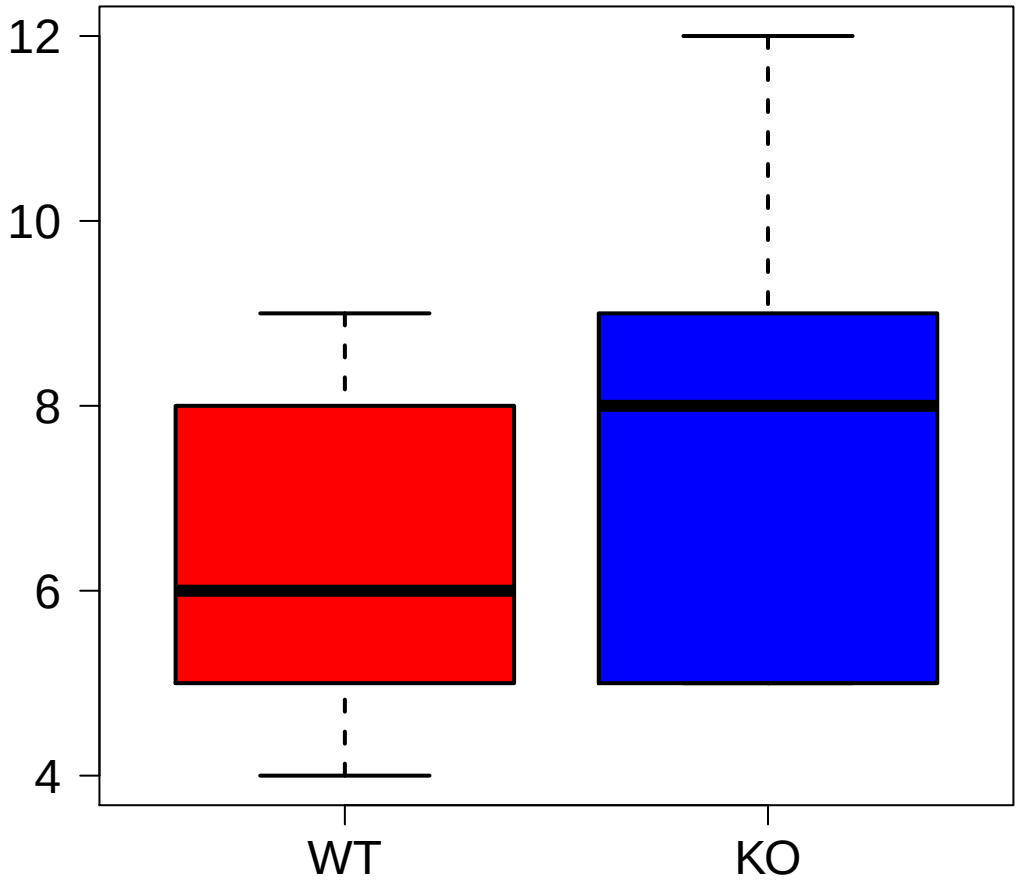


Note that here

- We've added horizontal jitter as additional details of the `stripchart` command
- `col=c("red","blue")` colors (red and blue) points by group
- `cex.axis=1.5` draws axis labels that are 1.5x as wide as usual.

We can also create a boxplot, which may be especially useful if we have too many points to show each one.

```
> boxplot(genes[,2:3], col=c("red","blue"), lwd=2, cex=1.5, cex.axis=1.5, las=1)
```



where `lwd=2` draws lines that are twice as thick as usual.

4 Drawing gene expression plots (slide 14)

We're going to use a full-size dataset here, with expression (more precisely, RNA abundance) values for all genes assayed. These values have already been log2-transformed.

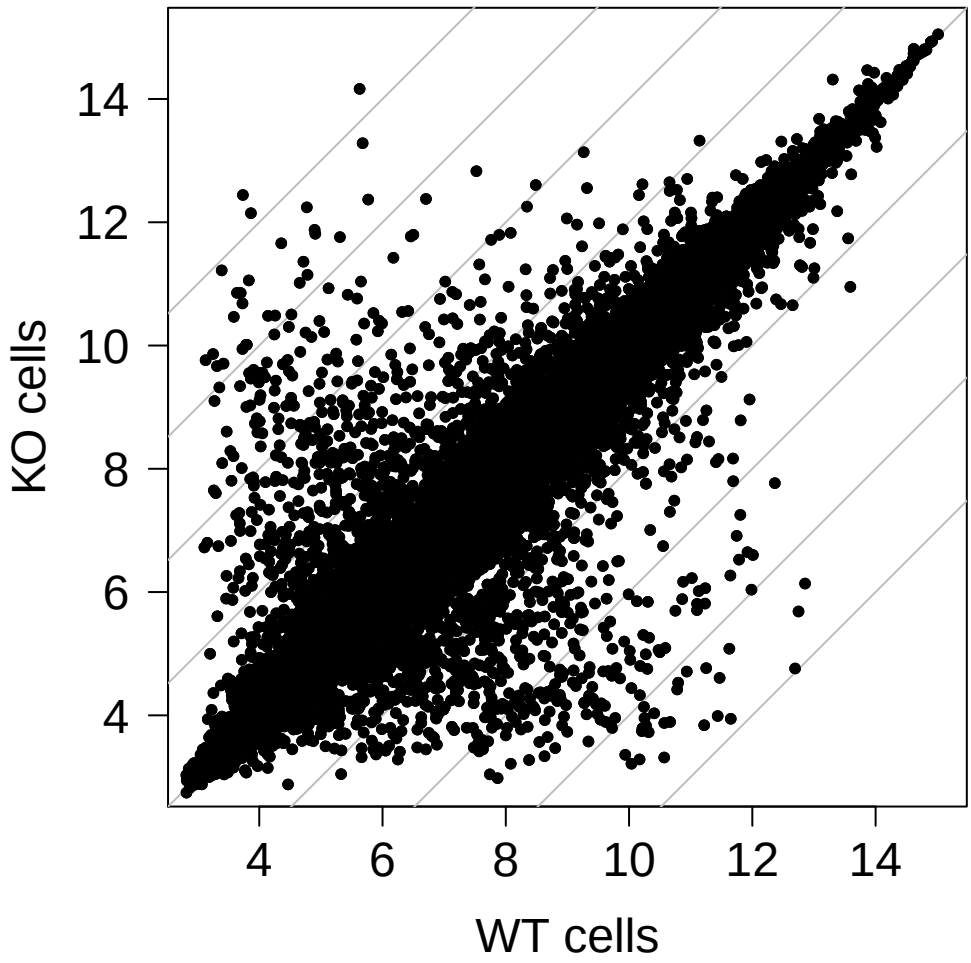
```
> genes.all = read.delim("Gene_exp_KO_vs_WT.txt", header=T, check.names=F)
> head(genes.all)
```

```
                WT cells KO cells
MUC5B::727897     4.9888   5.3842
HAPLN4::404037    5.2241   5.5785
SIGLEC16::400709  4.5920   5.0337
HES5::388585      5.0662   5.1480
AMIGO2::347902    3.7988  10.0131
FREM2::341640     8.0805   3.2143
```

The `head` command, which prints the top of data structure, shows that our data looks OK. Note that this time, the genes symbols are used as row labels and the data matrix contains just two columns, both of expression values. Also, including the option `check.names=F` keeps the spaces in our header line, rather than converting them to dots.

Let's start by doing a typical x-y scatterplot and add a bunch of lines hopefully to let us better interpret expression changes between KO and WT cells. Since the lines may cover up some points, we'll draw the points again on top.

```
> par(pty="s")
> plot(genes.all, pch=20, cex.axis=1.5, cex.lab=1.5, las=1, xlim=c(3,15), ylim=c(3,15))
> for (i in seq(-8,8,2)) { abline(i,1, lwd=1, col="gray") }
> points(genes.all, pch=20)
```

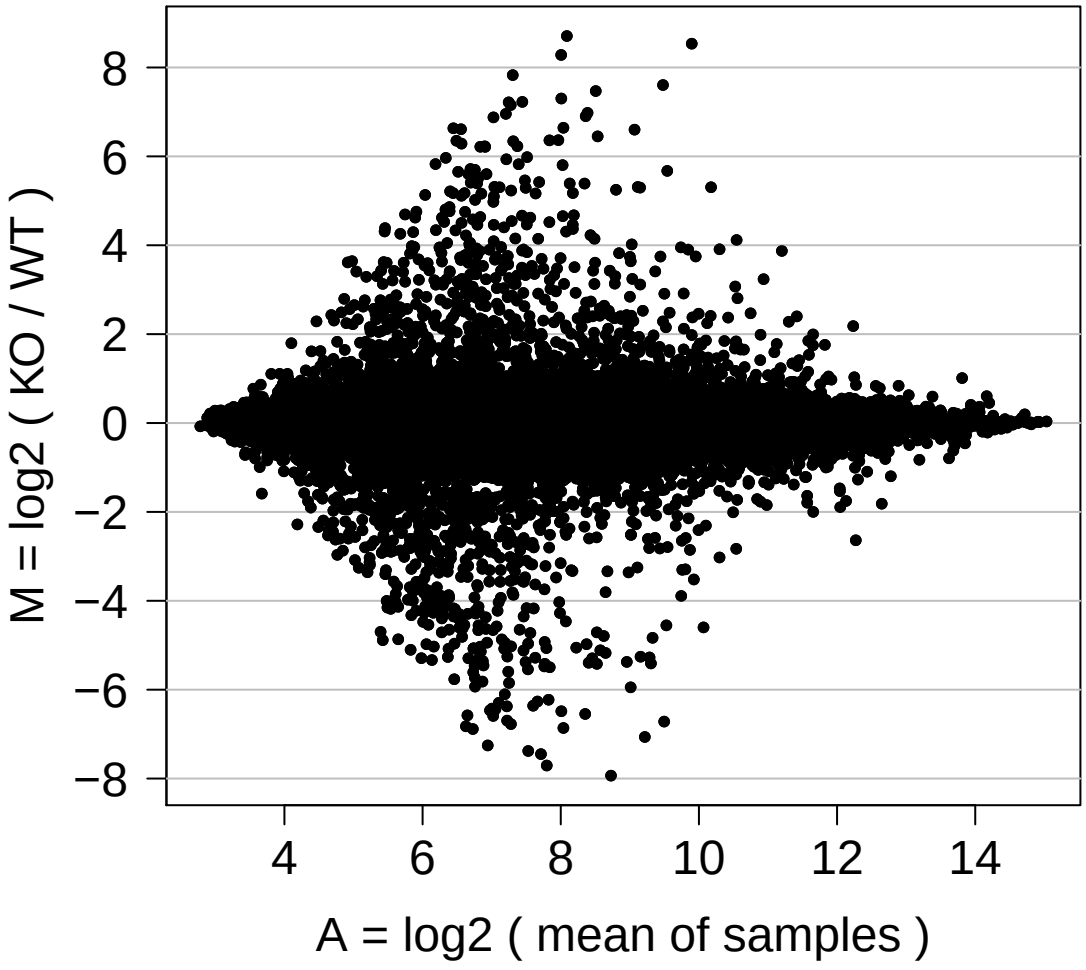


We used

- the `par(pty="s")` parameter to force the plot to be square
- the `abline(intercept, slope)` command to draw the diagonal lines (using intercepts of -8 to 8, by 2's)
- the `points(x, y)` command to redraw the points on top

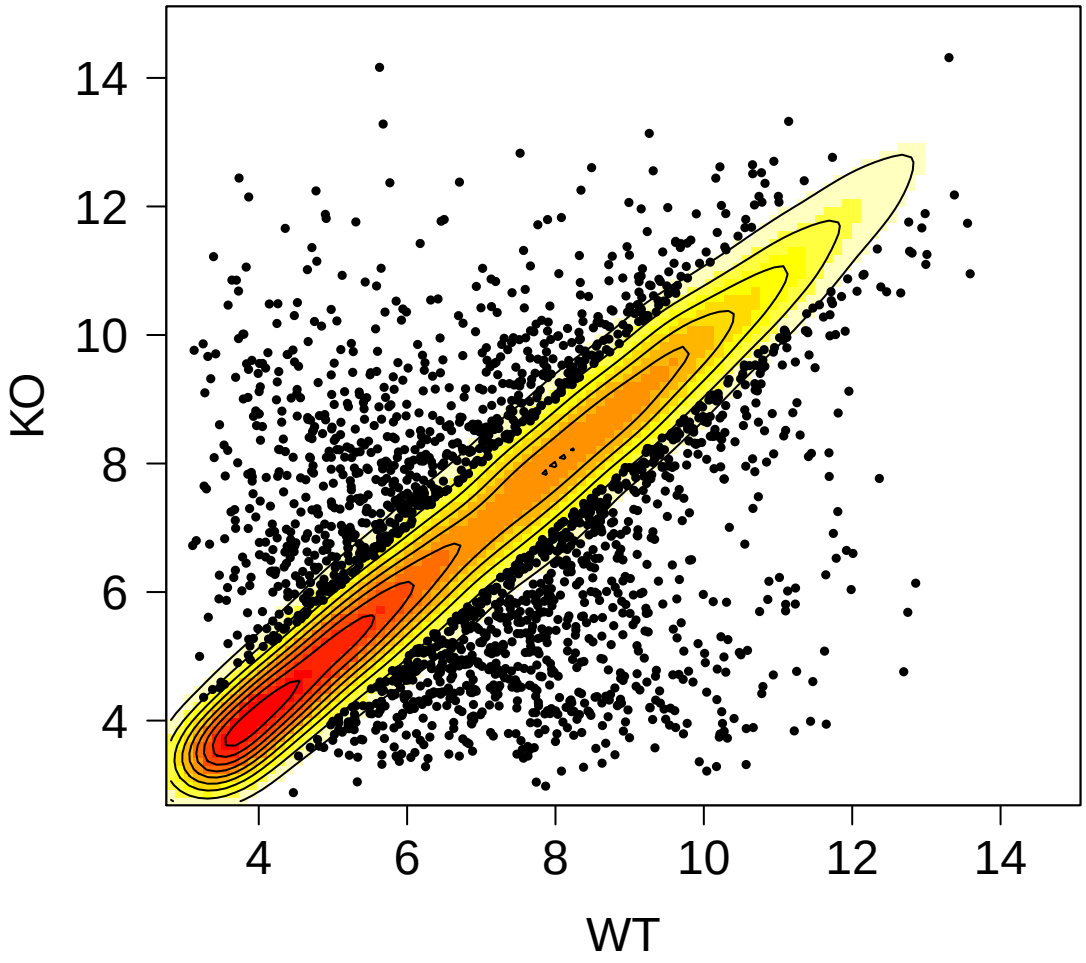
Since many people can interpret changes in horizontal space better than in diagonal space, a ratio-intensity (MA) plot may make more sense. It contains exactly the same data but the data is transformed to have the effect of rotating the figure 45 degrees. The axis labels describe our new M (ratio) and A (intensity) axes.

```
> M = genes.all[,2] - genes.all[,1]
> A = apply(genes.all, 1, mean)
> plot(A,M, pch=20, cex.axis=1.5, cex.lab=1.5, las=1, ylab="M = log2 ( KO / WT )",
+   xlab="A = log2 ( mean of samples )", yaxp=c(-10,10,10))
> abline(h=seq(-8,8,2), lwd=1, col="gray")
> points(A,M, pch=20)
```



A potential problem with any dense scatterplot is that we have no direct information about the density of overlapping points. We can solve this in several ways, including the addition of a contour plot that reflects point density. We can use a command `kde2d` to calculate density, but it's not in the basic R installation, so we have to install the "MASS" package which contains it. Once it's installed, we have to call it (to access its functions).

```
> require(MASS)
> d = kde2d(genes.all[,1], genes.all[,2], n=100)
> image(d,col=c("white",rev(heat.colors(10))),
+   cex.axis=1.5, cex.lab=1.5, las=1, xlab="WT", ylab="KO")
> contour(d,add=T, drawlabels=F)
> gt.2fold = abs(genes.all[,2]-genes.all[,1])>1
> points(genes.all[gt.2fold,], pch=19, cex=0.5)
> box()
```



This is a complex set of commands which involves these steps:

1. Use `kde2d` to calculate density of our points
2. Draw a sort of a heatmap with `image`, reflecting this density
3. Add contour lines (with `contour`) also reflecting this density
4. Identify the rows (`gt.2fold`) with genes that are at least 2-fold different between WT and KO
5. Add a point for each of those most changed genes
6. Draw a box around the final figure

5 Displaying distributions (slide 16)

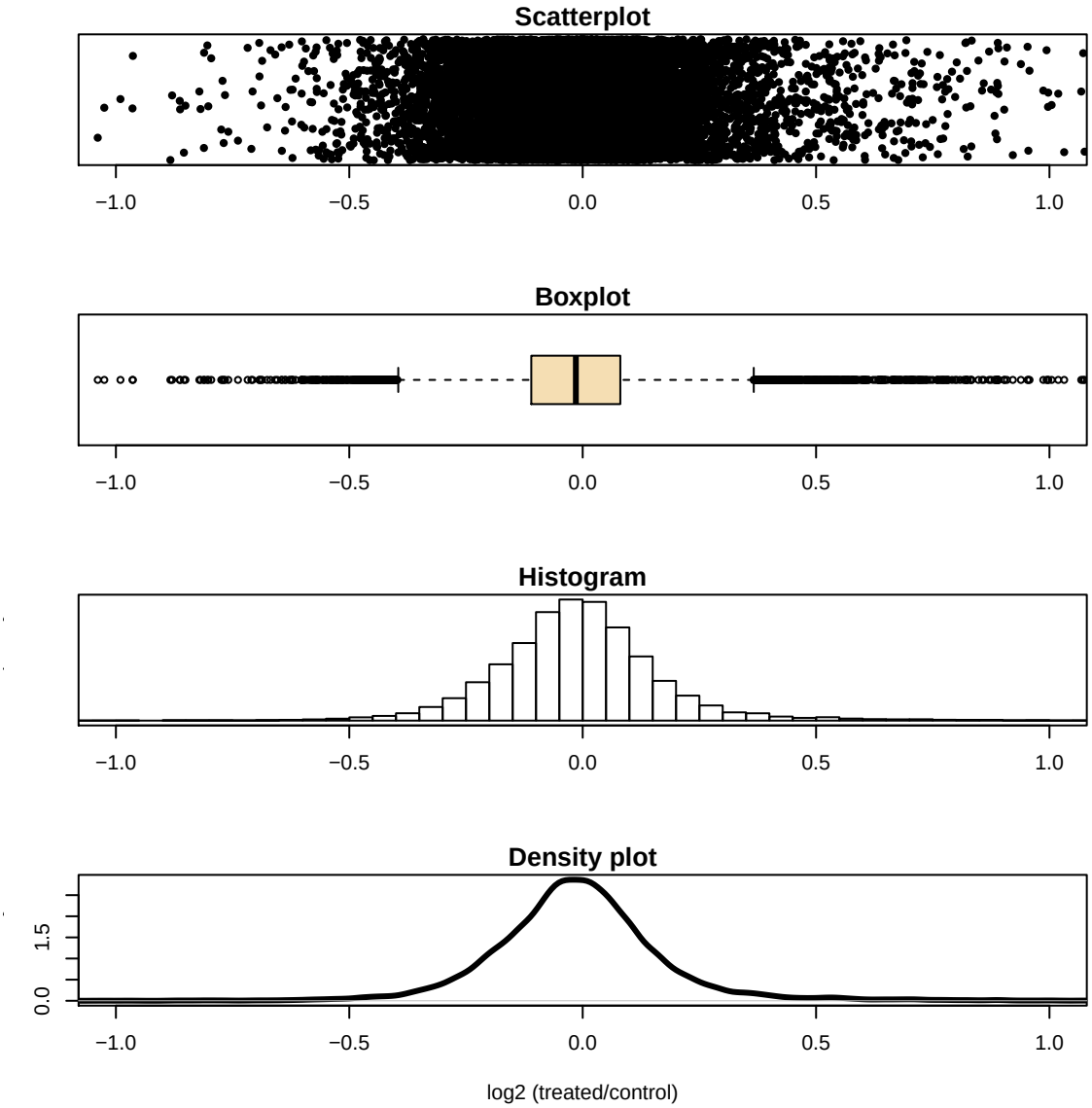
Our next (and last) dataset is another set of gene expression data but in this experiment we assayed gene expression in WT cells ("control") and cells in which an endogenous miRNA was knocked out ("treated"). The genes' log2 (treated/control) ratios are divided into two sets, those for genes that contain predicted regulatory sites for this miRNA (column 2; "yes.site") and those for all other genes (column 1; "no.site").

```
> ratios = read.delim("All_vs_with_site.txt")
> head(ratios)
```

```
  no.site yes.site
1 -0.1202   0.3020
2 -0.2930  -0.1831
3  0.0521  -0.0302
4 -0.0322   0.0997
5 -0.1748  -0.0400
6  0.0369  -0.1796
```

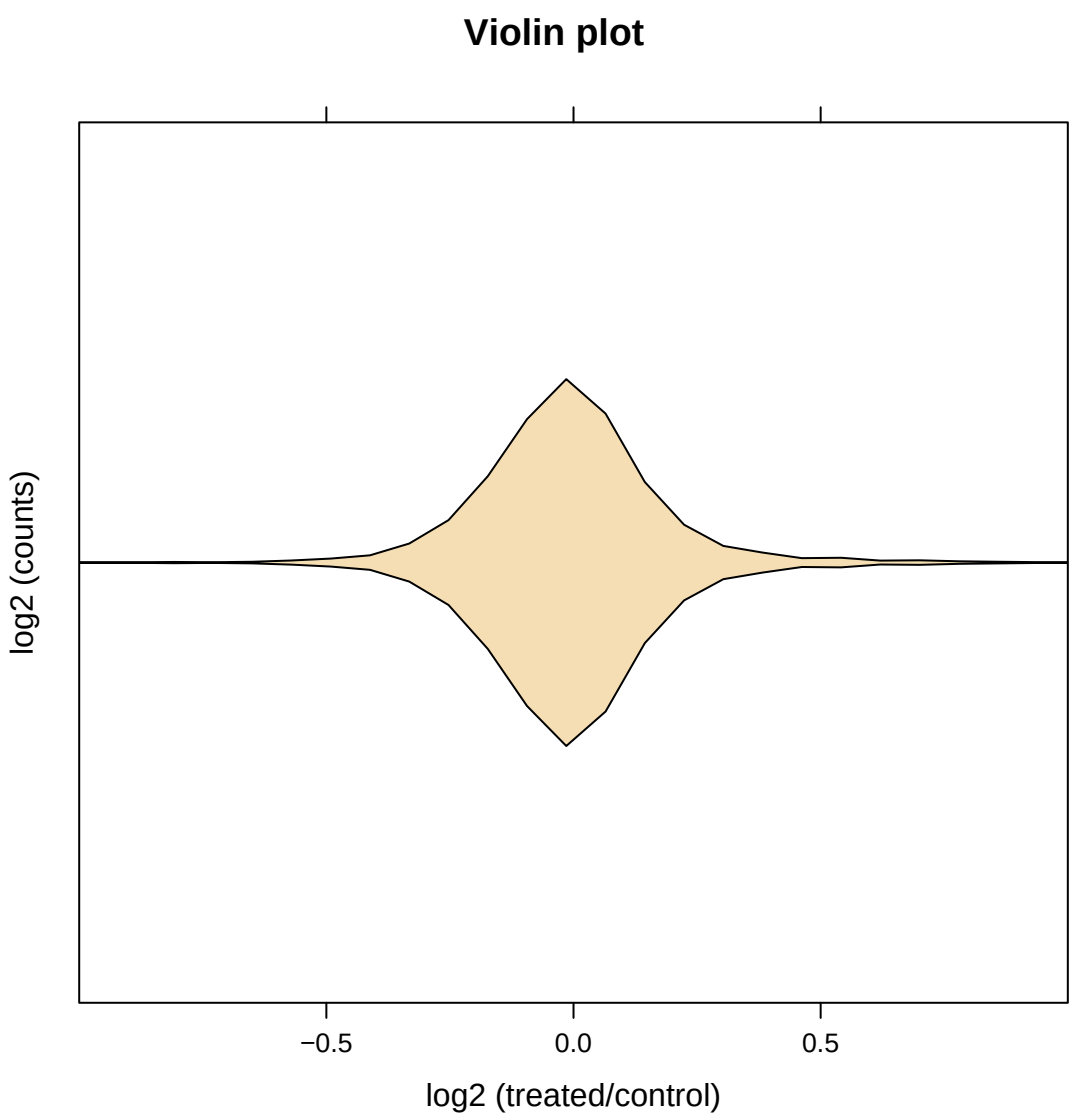
To display a distribution, we'll limit ourselves to the log2 ratios for genes without any predicted miRNA sites ("no.site"). We're going to draw four plots in the same file.

```
> par(mfrow=c(4,1), mai=c(0.6,0.4,0.2,0.2))
> plot(ratios$no.site, runif(length(ratios$no.site)), pch=20, las=1, xlab="",
+   yaxt="n", ylab="", xlim=c(-1,1), main="Scatterplot")
> boxplot(ratios$no.site, col="wheat", horizontal=T, ylim=c(-1,1), main="Boxplot")
> hist(ratios$no.site, breaks=100, xlim=c(-1,1), xlab="", yaxt="n", main="Histogram")
> box()
> plot(density(ratios$no.site), xlim=c(-1,1), lwd=3,
+   xlab="log2 (treated/control)",  main="Density plot")
```



Now let's do the violin plot, which is like a density plot and its mirror image put pack to back. Its name comes from the display of a bimodal distribution, which has a violin shape. By itself this violin plot isn't so useful, but this type of plot can be effective for comparing several distributions (while providing more detail than box plots).

```
> library(lattice)
> print (
+ bwplot(ratios$no.site, xlim=c(-1,1),
+         main="Violin plot", ylab = "log2 (counts)",
+         xlab="log2 (treated/control)",
+         panel = function(..., box.ratio)
+ {
+         panel.violin(..., col = "wheat", varwidth = T, box.ratio = 1)
+ } ) )
```



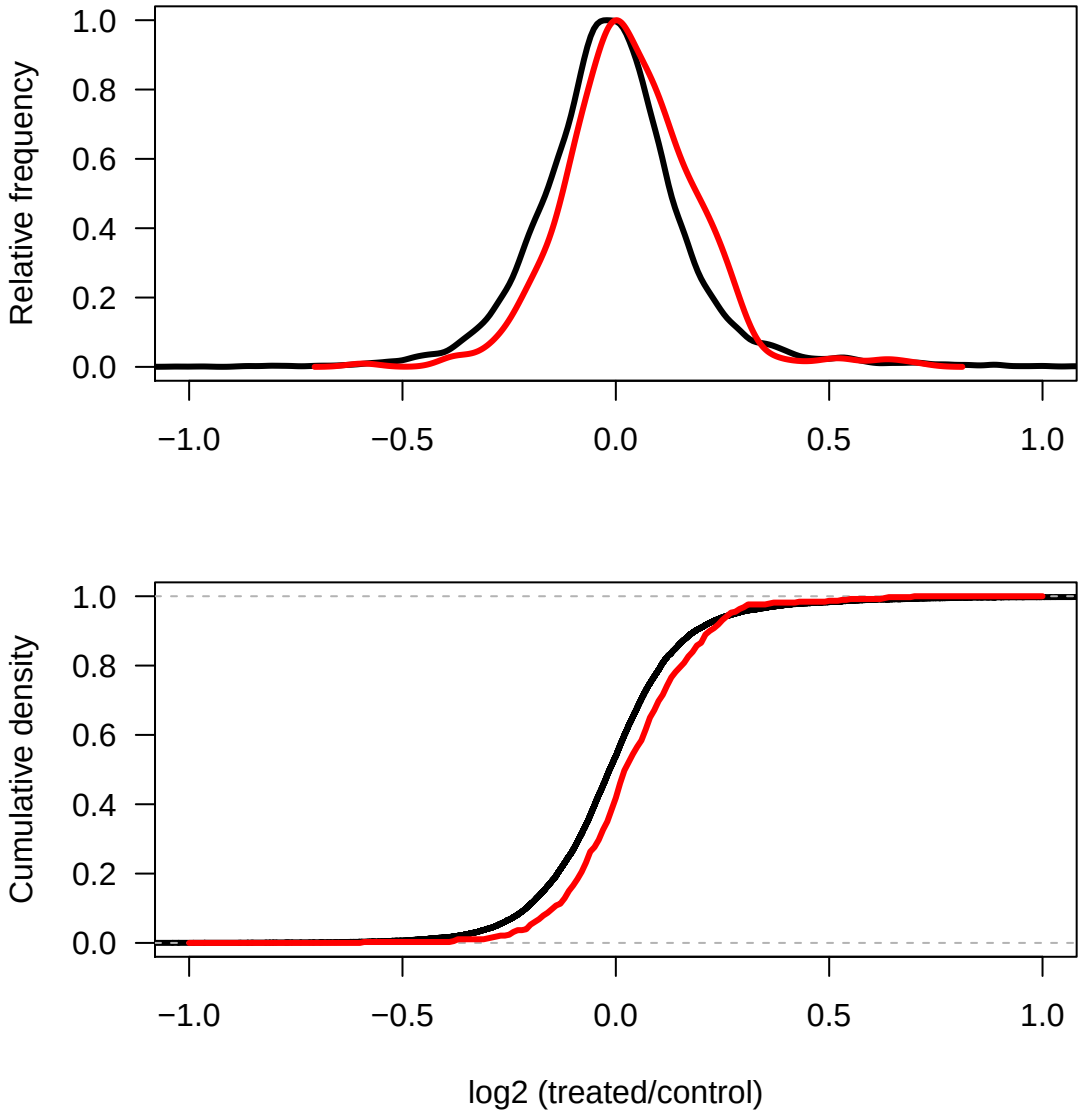
Let's get back to the complete set of expression data before and after miRNA knockout. We'd like to draw density plots comparing the log2 ratios for genes with a predicted miRNA site (in red) to all the other genes (black). Then we're going to calculate the cumulative distribution functions for these two distributions and graph them.

We'll start by calculating the density function for each list of values. The "max" values are used to scale the y-axis of each distribution to a maximum of 1 for better comparison.

```
> no.site.x = density(ratios$no.site)$x
> no.site.y = density(ratios$no.site)$y
> no.site.max = max(no.site.y)
> yes.site.x = density(ratios$yes.site, na.rm=T)$x
> yes.site.y = density(ratios$yes.site, na.rm=T)$y
> yes.site.max = max(yes.site.y)
```

Now we make two figures, one comparing densities and one comparing cumulative distributions

```
> par(mai=c(1,1,0.05,0.2), mfrow=c(2,1))
> plot(no.site.x, no.site.y/no.site.max, type="l", xlim=c(-1,1), lwd=3,
+   xlab="", ylab="Relative frequency", las=1)
> lines(yes.site.x, yes.site.y/yes.site.max, type="l", xlim=c(-1,1), lwd=3, col="red")
> plot(ecdf(ratios$no.site), xlim=c(-1,1), lwd=3, xlab="log2 (treated/control)",
+   ylab="Cumulative density", las=1, main="")
> f.ecdf = ecdf(ratios$yes.site)
> x0 = seq(-1,1, by = 0.01)
> lines( cbind(x=x0, y=f.ecdf(x0)), type="l", lwd=3, col="red")
```



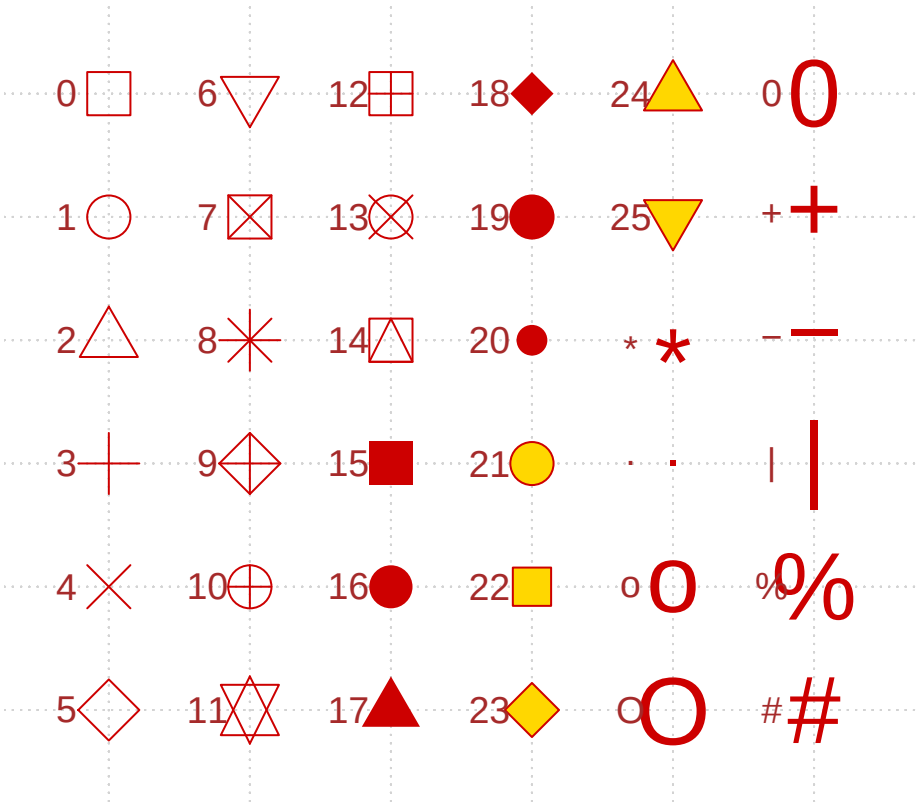
6 Drawing all point shapes (slide 19)

This requires the calling the function `pchShow` as shown on the help page for `pch`. You should have a R script (text file) containing this function called `pchShow.R` (in your current directory). Execute it by typing

```
> source("pchShow.R")
```

Nothing will happen, but you'll now have this function in memory. If you didn't have this file, you could open the `pch` help page with the command `?pch`, copy the function (starting with `pchShow <-`) and paste it in your R session. Either way, we're now ready to call the function:

```
> pchShow()
```



This is a useful reference for future graphing.

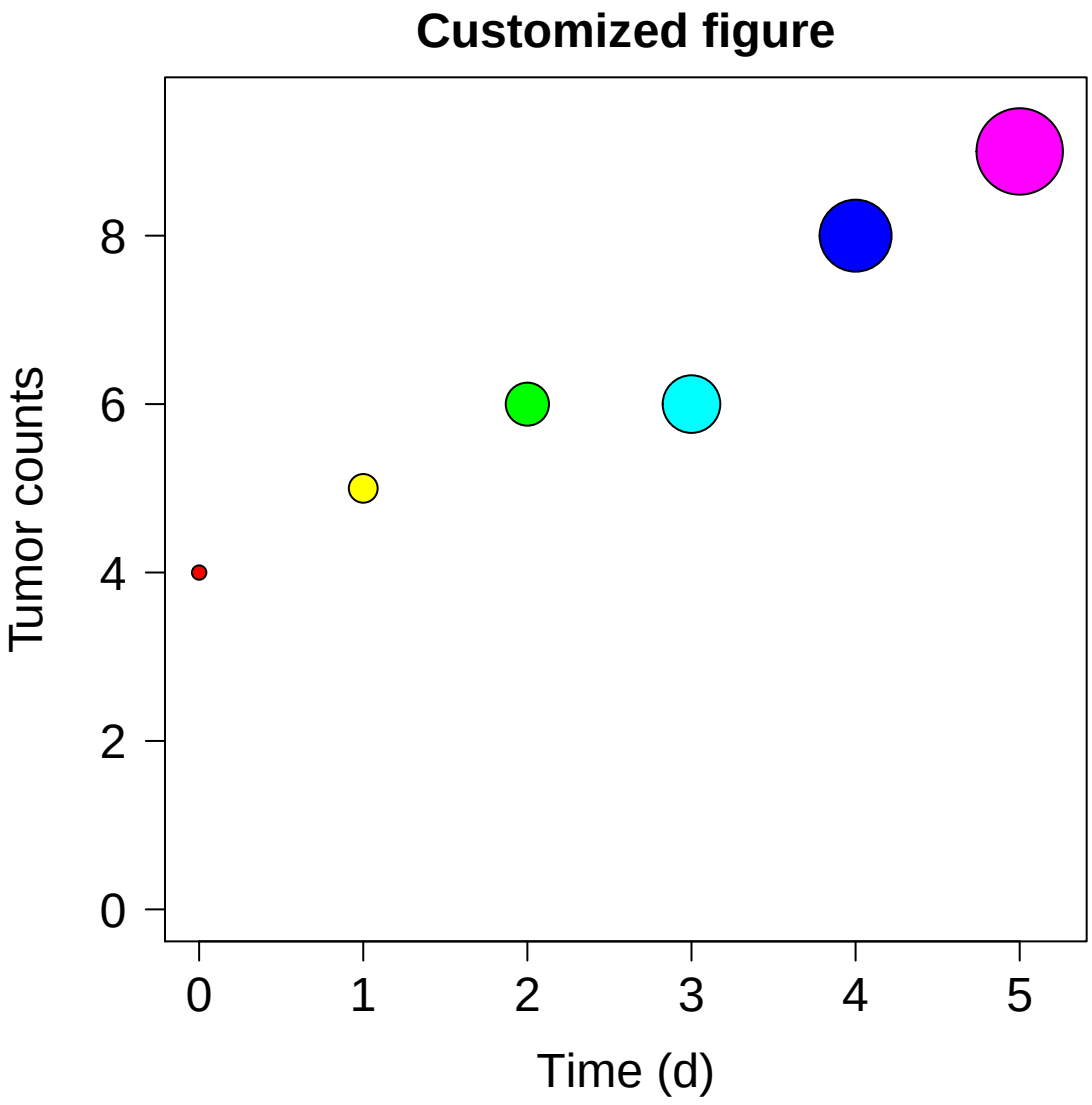
7 Customizing a plot (slide 20)

As we have seen before, almost any basic graphics command can be customized by added additional details. Let's go back to the original small set of expression values (`genes`) and create two new sets of numbers by sorting (which, we should note, has no scientific rationale; we're just playing with numbers). We'll also create a set of pretend time points the same size as these datasets, starting with 0 and incrementing by 1.

```
> y1 = sort(genes$WT)
> y2 = sort(genes$KO)
> x = seq(0, length(y1)-1)
> x

[1] 0 1 2 3 4 5

> y1

[1] 4 5 6 6 8 9

> y2

[1]  5  5  8  8  9 12
```

Now let's again start with the most basic plot of points (`plot(x, y1, type="p")`) and then customize it.

```
> par(mai=c(1,1,0.5,0.2))
> plot(x, y1, type="p", pch=21, col="black", bg=rainbow(6), cex=x+1, main="Customized figure",
+   xlab="Time (d)", ylab="Tumor counts", las=1,
+   cex.axis=1.5, cex.lab=1.5, cex.main=1.5, xlim=c(0,5.2), ylim=c(0,9.5))
```

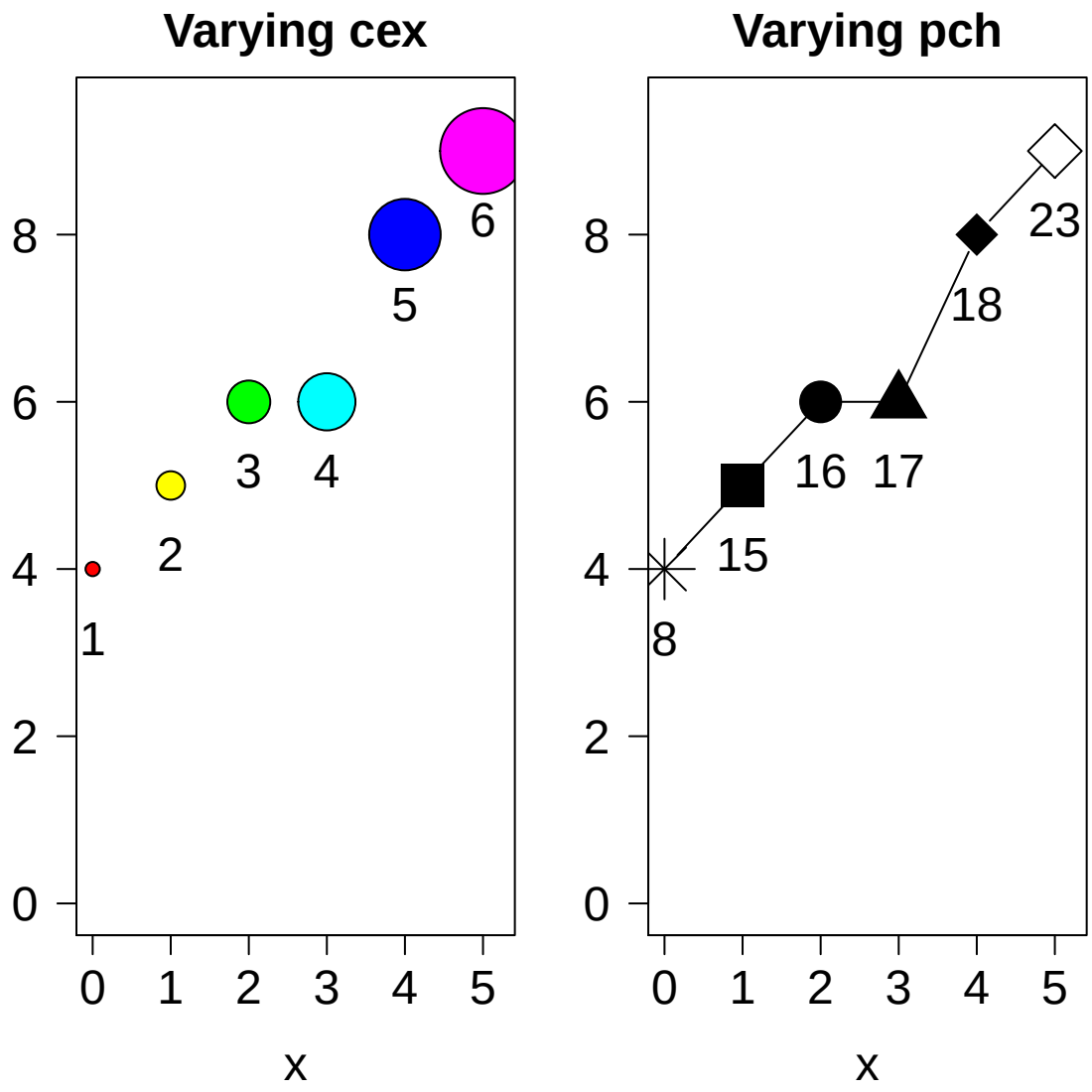


The type of point we chose (`pch=21`) includes a (perimeter) color (`col`) and fill color (`bg`), the latter of which we've created by selecting several colors from the rainbow (`rainbow(6)`). We can also vary the point size (`cex`) by defining it as a set of numbers rather than just one number.

8 Combining plots on a page (slide 21)

Let's draw two plots side by side, each with text added as point labels. On the left we vary `cex` and on the right we vary `pch`

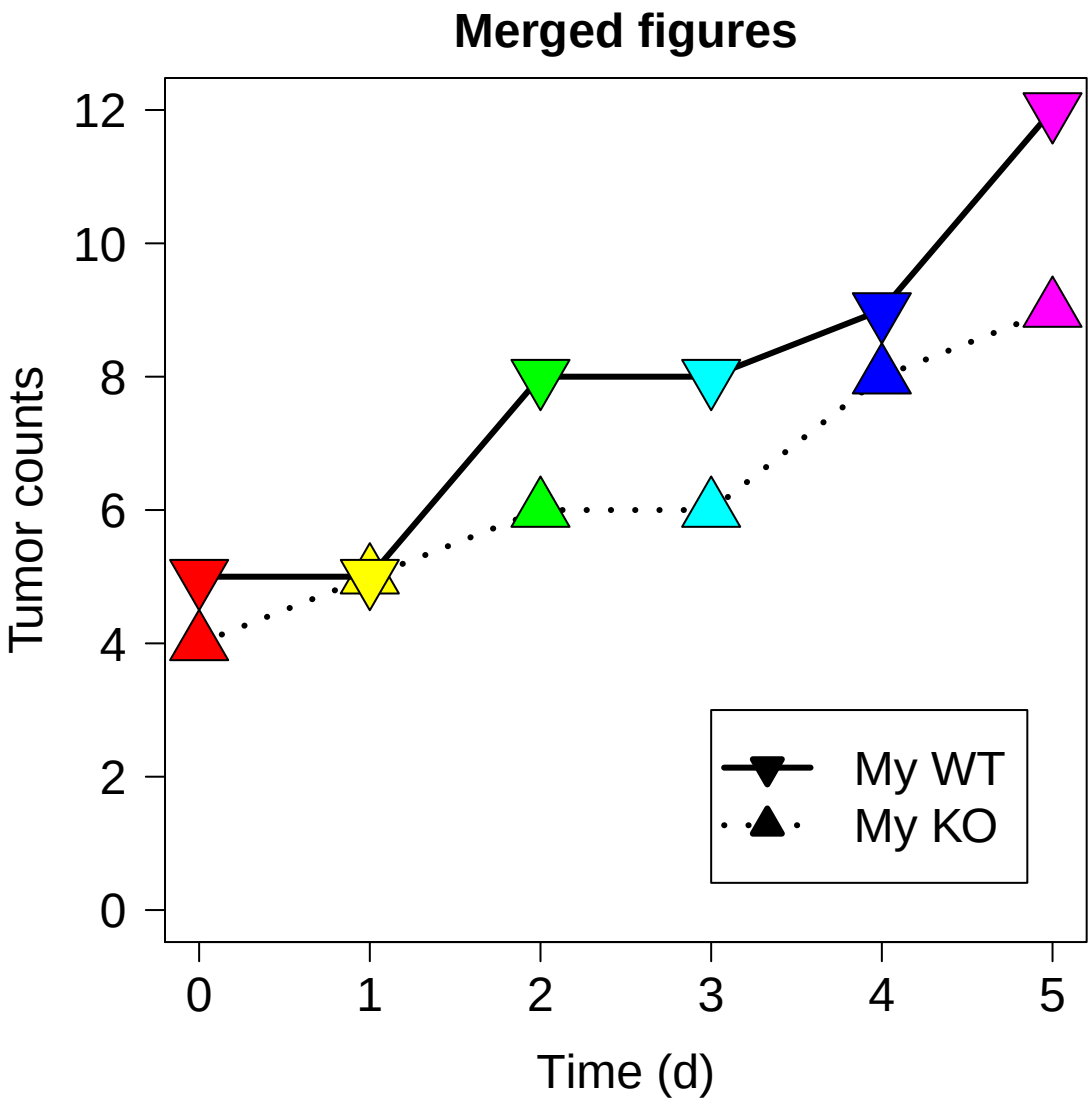
```
> par(mfrow = c(1,2), mai=c(1,0.5,0.5,0.2))
> plot(x, y1, type="p", pch=21, col="black", bg=rainbow(6), cex=x+1, las=1,
+   cex.axis=1.5, cex.lab=1.5, cex.main=1.5, xlim=c(0,5.2), ylim=c(0,9.5),
+   main="Varying cex")
> text(x, y1, x+1, pos=1, offset=1.5, cex=1.5)
> plot(x, y1, type="b", pch=c(8,15:18,23), col="black", cex=3, las=1,
+   cex.axis=1.5, cex.lab=1.5, cex.main=1.5,
+   xlim=c(0,5.2), ylim=c(0,9.5), main="Varying pch")
> text(x, y1, c(8,15:18,23), pos=1, offset=1.5, cex=1.5)
```



9 Merging plots on the same figure (slide 22)

Here we want to graph two datasets in the same space. We start with what we're calling "My KO" (upward triangles), first drawing a dotted line `lty=3` and then adding points. Then we add "My WT" (downward triangles), first with a line, solid (`lty=1`) this time, and then with points. Finally we add the legend, where we have to specify what points, lines, and/or colors go with each group.

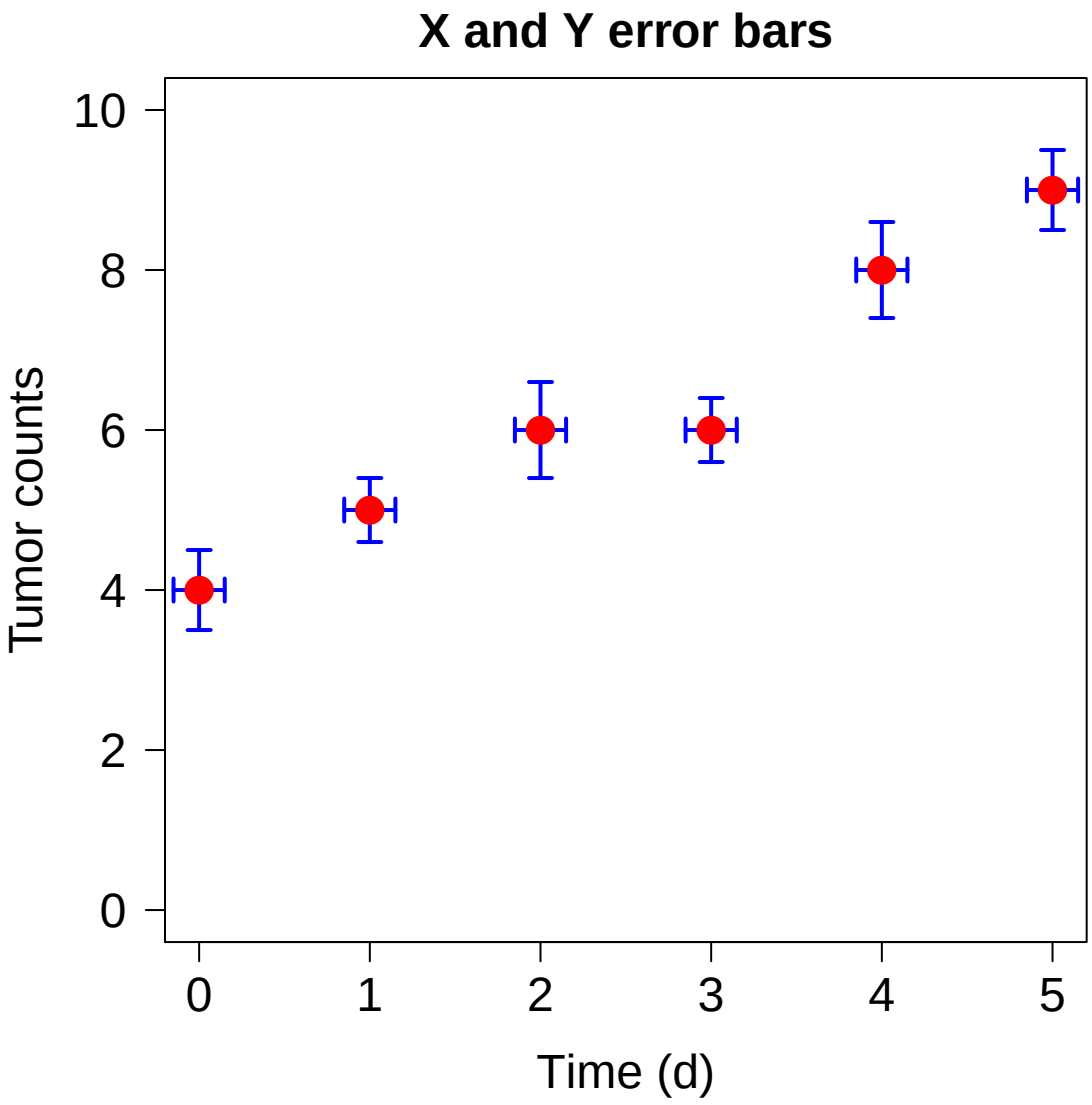
```
> par(mai=c(1,1,0.5,0.2))
> plot(x, y1, type="l", lty=3, lwd=3, cex=3, ylim=c(0, max(c(y1,y2))),
+   xlab="Time (d)", ylab="Tumor counts", las=1, cex.axis=1.5,
+   cex.lab=1.5, main="Merged figures", cex.main=1.5)
> points(x, y1, pch=24, bg=rainbow(6), cex=3)
> lines(x, y2, pch=25, col="black", lty=1, lwd=3)
> points(x, y2, pch=25, bg=rainbow(6), cex=3)
> legend(3,3, c("My WT","My KO"), pch=c(25,24),
+   pt.bg="black", cex=1.5, lty=c(1,3), lwd=3)
```



10 Using error bars (slide 24)

The package plotrix can draw nice error bars. The length of each bar is set with the uiw (upper) and liw (lower) length parameters. By default the error bars are vertical but they can also be horizontal (`err="x"`). We get the former pair from genes$WT.sd and arbitrarily set the latter to 0.15.

```
> library(plotrix)
> par(mai=c(1,1,0.5,0.2))
> plotCI(x, y1, uiw=genes$WT.sd, liw=genes$WT.sd, lwd=2, col="red", scol="blue", pch=16, cex=2,
+   xlab="Time (d)", ylab="Tumor counts", las=1, cex.axis=1.5, cex.lab=1.5,
+   main="X and Y error bars", cex.main=1.5, ylim=c(0,10))
> plotCI(x, y1, uiw=0.15, liw=0.15, lwd=2, col="red", scol="blue", pch=16, cex=2, err="x", add=T)
```



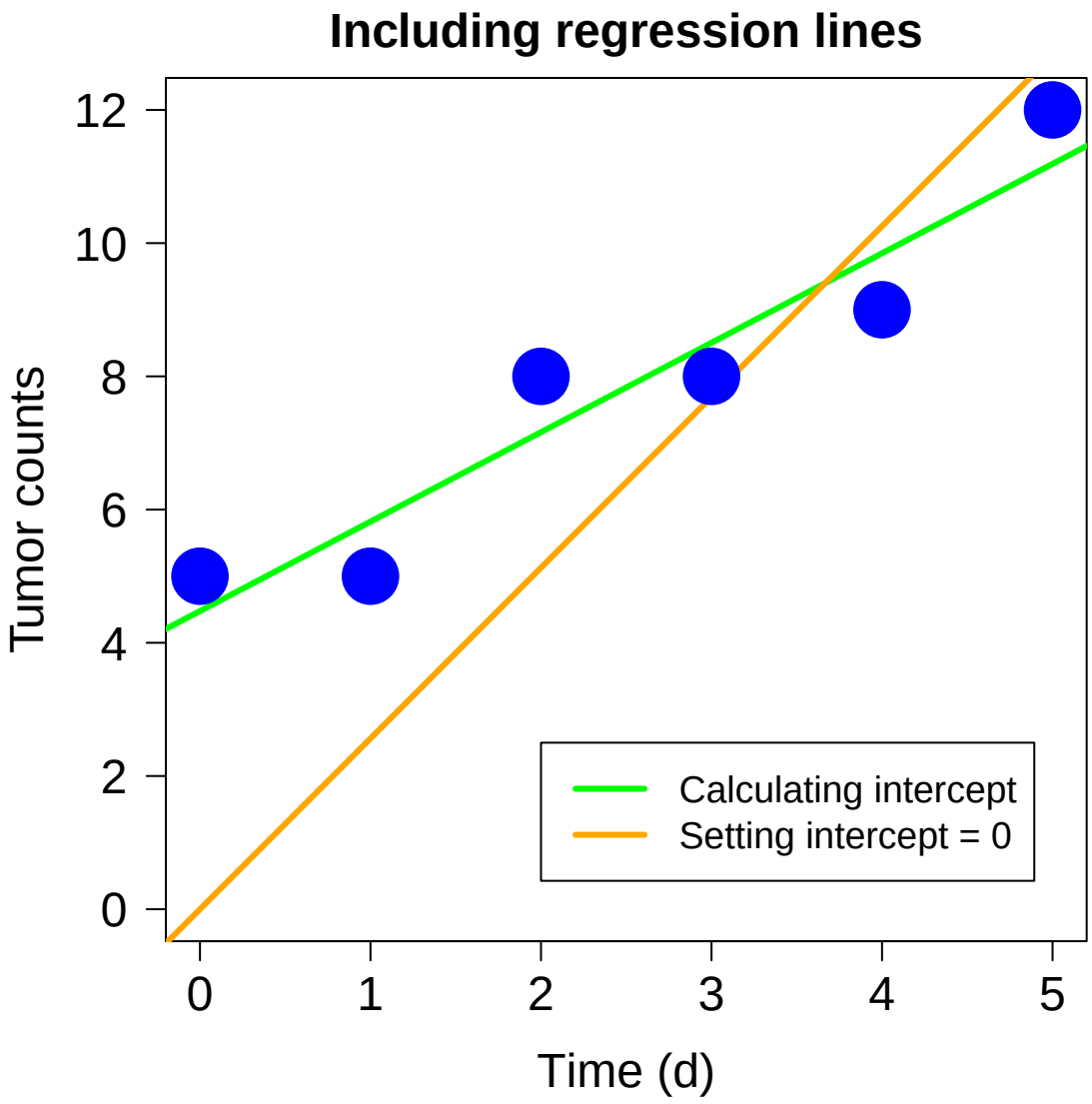
11 Drawing a regression line (slide 25)

We need to first do regression analysis. We can simply choose the best line for the points (first equation) or, if our experiment warrants it, do the same while requiring the line to go through 0,0 (second equation).

```
> my.lm = lm(y2 ~ x)
> my.lm.0 = lm(y2 ~ x + 0)
```

Then we can show the results as lines on a plot of points. We'll again redo the points (so they're all on top) and add a legend.

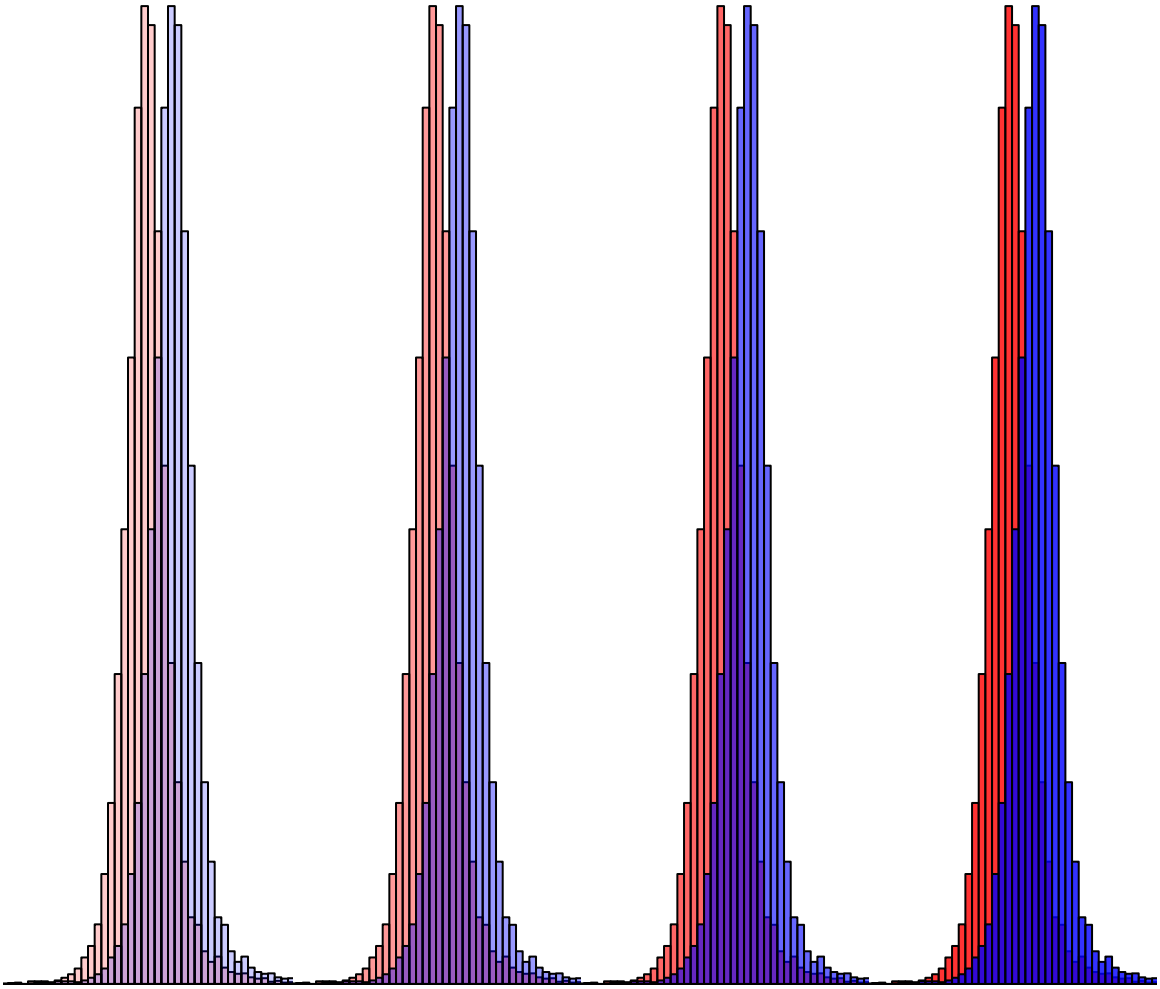
```
> par(mai=c(1,1,0.5,0.2))
> plot(x, y2, xlab="Time (d)", ylab="Tumor counts", las=1, cex.axis=1.5, cex.lab=1.5,
+   main="Including regression lines", cex.main=1.5, ylim=c(0,12))
> abline(my.lm, col="green", lwd=3)
> abline(my.lm.0, col="orange", lwd=3)
> points(x, y2, col="blue", pch=16, cex=4)
> legend(2,2.5, c("Calculating intercept","Setting intercept = 0"),
+   col=c("green","orange"), inset=0.01, lty=1, cex=1.2, lwd=3)
```



12 Transparent colors (slide 26)

Transparency can be indicated by two positions at the end of a color, expressed as RGB values. We start with the set of log2 ratios from the miRNA knockout and create a second distribution by shifting this one.

```
> par(mfrow=c(1,4), mai=c(0,0,0.5,0))
> hist(ratios$no.site, breaks=100, xlim=c(-1,1), col="#FF000033", axes=F, main="Transparency = 33")
> hist(ratios$no.site+0.2, breaks=100, add=T, col="#0000FF33")
> hist(ratios$no.site, breaks=100, xlim=c(-1,1), col="#FF000066", axes=F, main="Transparency = 66")
> hist(ratios$no.site+0.2, breaks=100, add=T, col="#0000FF66")
> hist(ratios$no.site, breaks=100, xlim=c(-1,1), col="#FF000099", axes=F, main="Transparency = 99")
> hist(ratios$no.site+0.2, breaks=100, add=T, col="#0000FF99")
> hist(ratios$no.site, breaks=100, xlim=c(-1,1), col="#FF0000CC", axes=F, main="Transparency = CC")
> hist(ratios$no.site+0.2, breaks=100, add=T, col="#0000FFCC")
```



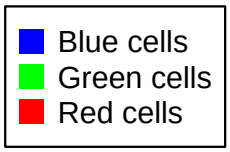
13 Colored bars (slide 27)

Copy the R script \\BaRC_Public\BaRC_code\R\draw_colored_bar_arrays.R
and the sample data from \\BaRC_Public\BaRC_code\R\draw_colored_bar_arrays_sample_data\ to your directory. The R code can be run on tak from the command line (not within R) with a command like

```
R --vanilla < draw_colored_bar_arrays.R colors.txt color_legend.txt color_bars
```

to produce output files like color_bars.pdf:

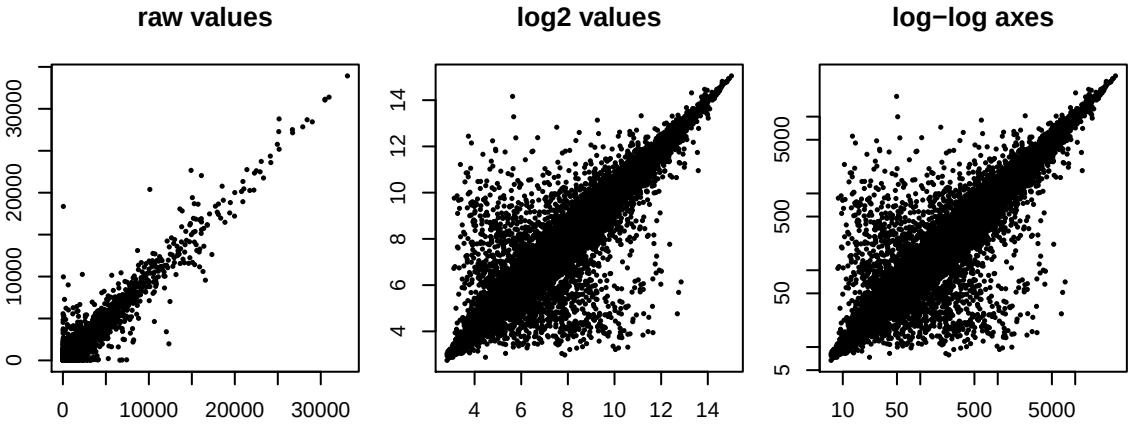
and color_bars_legend.pdf:



14 Log transformations (slide 28)

Let's start with the big file of log2 expression values. We can "un-log" the values and then plot them as is or using a log-log plot.

```
> genes.log2 = read.delim("Gene_exp_KO_vs_WT.txt", check.names=F)
> genes = 2**genes.log2
> par(mfrow=c(1,3), mai=c(0.5,0.3,0.5,0.1), pty="s")
> plot(genes, pch=20, main="raw values", xlab="", cex=0.5)
> plot(genes.log2, pch=20, main="log2 values", xlab="", ylab="", cex=0.5)
> plot(genes, pch=20, log="xy", main="log-log axes", xlab="", ylab="", cex=0.5)
```



In the interests of totally reproducible research, we'll end by printing our R environment.

```
> sessionInfo()

R version 2.12.1 (2010-12-16)
Platform: x86_64-pc-linux-gnu (64-bit)

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

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

other attached packages:
[1] plotrix_2.9-5  lattice_0.18-8 MASS_7.3-7

loaded via a namespace (and not attached):
[1] grid_2.12.1
```

This is just a start! There's lots more.... To exit R, type `q()`