Additional exercesis R bioinfo programming. Master ISCIII. 2014-2015. First set.

Ramon Diaz-Uriarte[*]

2014-11-19 (Release 1.1: Rev: b2ab80f)

Contents

List of Figures

1 Recommendations

2 Purpose of the exercises

The immediate objective of the exercises is for you to use the Leukemia data and produce the figures that follow (see next). The purpose is for you to:

- Use some extremely useful types of plots for visualizing data.
- Learn to change options to represent what you want.

[*] Dept. of Biochemistry, Universidad Autónoma de Madrid, Spain, http://ligarto.org/rdiaz, rdiaz02@gmail.com

- Annotate plots appropriately.
- Use vector and, generally, data transformations as needed for your plots.
- Use the help system.

3 General hints

3.1 General

Look at the figures before doing anything. Try to understand what the figures are about, and why you might want to do a figure like this. Then, try to think how you would produce such a figure. Finally, read the hints carefully and reproduce the figure.

(No, you do NOT need to provide figure captions at the bottom.)

Oh, there is one case where we put to figures in one single page. Use `par(mfrow=something)` to be able to do that.

3.2 The data

- Read the data "leukemia.data.txt". Note that there are row names that we do not want to be part of the data itself (look at the help for "read.table", the option "row.names"). Call this "leuk.dat" These data are based on the famous leukemia data from Golub et al. I've modified a few things below: the gene names and the sex of the patients are invented.
- That will be a data frame. For most figures your life will be simpler if you convert it into a matrix, using `data.matrix`. Call this data matrix "leuk.dat.m". (Why is it a good idea to convert this into a matrix? Try doing the figures using the original data frame.)
- Read the data "leukemia.class.txt". Use scan for this (do you know why?). Since these are labels, use `what = ""` in `scan`. Convert the classes into a factor. Call this "leuk.class".
- Create a vector for the sex of the patients. The patients are Male and Female, alternating, and staring with Male. Call this factor "sex".

4 The boxplot of PTEN and scatterplots with lots of info

4.1 The boxplot

We will start with the boxplot (the top figure in Figure 1). The boxplot is a great representation of data: it shows the median, the range, the 1st and 3rd quartiles, and outliers. The boxplot here is very clear: there is a large difference in expression in PTEN.

- We want to plot the expression of gene 2124 (yes, that is row 2124 in the data).
- We want to plot it as a function of, or grouped according to, the type of patient.
- And how do you create a boxplot . . . there are actually several different ways. Let's use `boxplot`.
- How do you get the colors? It is the argument `col`. I've used "orange" and "lightblue".
- How do we get the title, where it says "a) Boxplot of PTEN by patient group."? Look at the help and check the argument `main`.

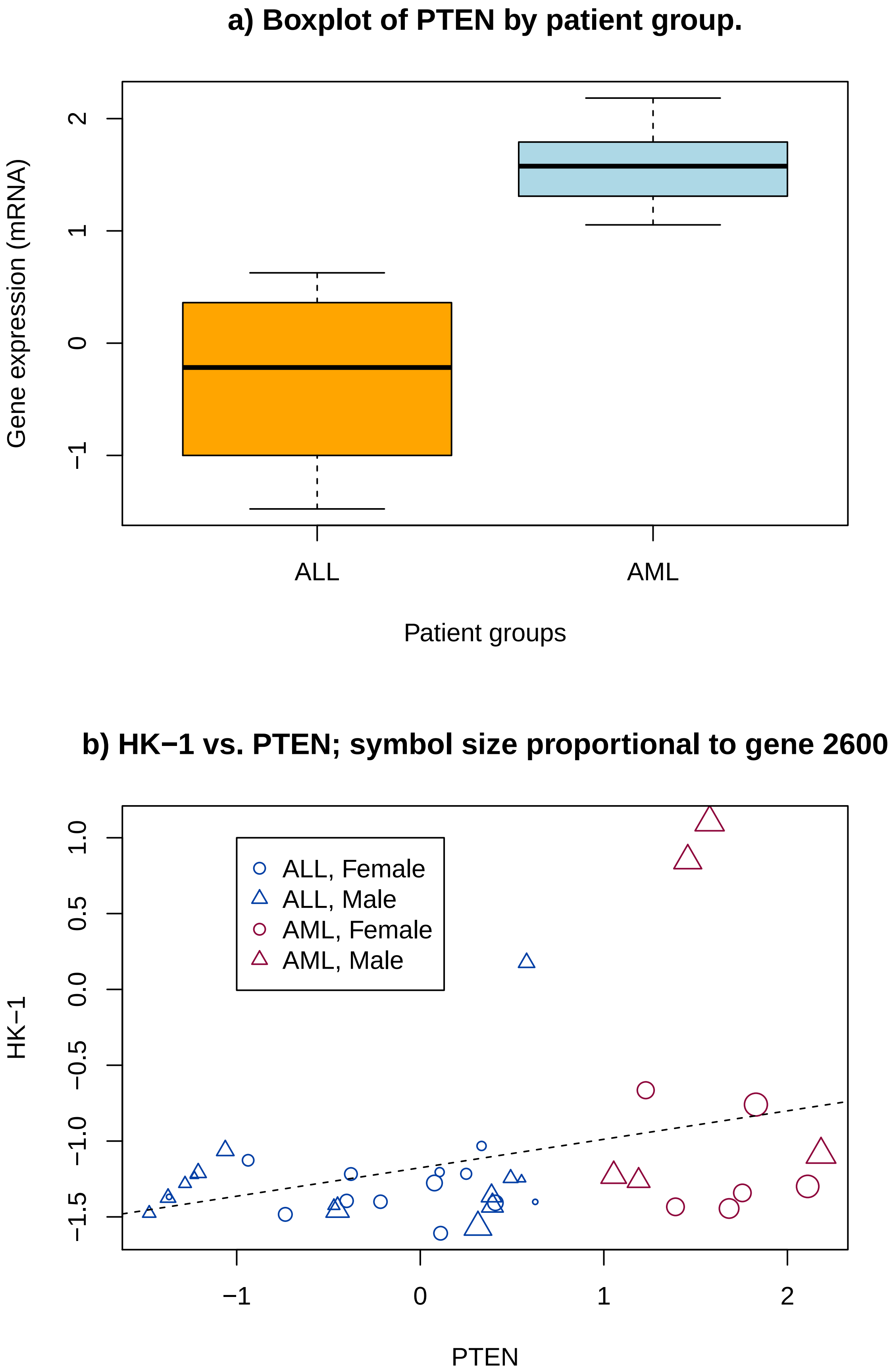

Figure 1: Boxplot of PTEN (gene 2124) and scatterplot of PTEN vs. HK-1 (gene 1). In the scatterplot, a linear fit has been added. Size of symbols is proportional to the expression of gene 2600

4.2 PTEN, HK-1, a third gene, patient status, and some sex (this is a lot of stuff!!)

Now we will create a figure that contains A LOT of information. Look at the bottom of figure 1. We are plotting the expression of HK-1 (a housekeeping gene? —the first gene in the data, so row 1 in the data) vs. the expression of PTEN, and we are showing different symbols and colors according to sex and group of patients.

There is a legend that uses different colors for type of patient and different symbols for sex. You can see that there is a large different in PTEN according to type of patient (this we knew already from the boxplot), but sex seems unrelated to gene expression of PTEN.

We have added a regression line of HK-1 on PTEN (i.e., modeling HK-1 as a function of PTEN). There is a mild trend (we will revisit this later).

And why do the symbol sizes differ? Because the size of the symbol is proportional to the expression of another gene, gene 2600 (yes, this is row 2600 in the data). Notice something: the size of the symbols seems to increase as we move to the right, and that tells us that gene 2600 is correlated with PTEN. In fact, all large symbols seem to correspond to subjects with AML and all small symbols to subjects with ALL (and that is in fact the case). And there seems to be no difference in symbol size with sex, so sex also seems unrelated to the expression of gene 2600.

Finally, we have used a different range of colors, colors which are probably much more appropriate than saturated, vibrant colors such as red and blue (if you are curious, take a look at this technical report `http://eeecon.uibk.ac.at/~zeileis/papers/Zeileis%2BHornik%2BMurrell-2009.pdf`). To do that, you will need to install a package called "colorspace" (you know how to install packages) and then load it (remember the `library` command).

A few detailed hints:

- How do we use different symbols? I used an open circle and an open triangle. There are several ways of getting these. Yes, but how do we get different symbols according to type of patient? I used this

  ```
  pch = c(21, 24)[sex]
  ```

  Please, be sure to understand what we are doing. (And remember that "sex" was a factor).

- How do we get the size of symbols to vary? The `cex` argument to plot. All we need to do is pass a vector which is proportional to the expression of gene 2600. But beware, that gene takes negative values. This is what I did (other solutions are possible); make sure you fully understand it and you understand the consequences of using, say, a 3 instead of a 2, or of not dividing by the max.

  ```
  for.cex <- leuk.dat.m[2600, ] - min(leuk.dat.m[2600, ]) + 1
  the.cex <-   2 * for.cex/max(for.cex)
  ```

  And `the.cex` will be the symbol size.

- And the colors? This is what I passed as an argument to `col`:

  ```
  col = diverge_hcl(2)[leuk.class]
  ```

  There are many other options, but this works OK. Understanding the full details of why that set of colors (i.e., why we use `diverge_hcl`) is not needed here. Of course, you do need to understand what we are doing when we use the above (and notice specially the "[leuk.class]").

- The legend ... We will use the function `legend`. Most of what you pass to it is already passed on as arguments in the col to plot (things such as `pch`, `col`). Where you place the legend is up to you (of course, be reasonable: do not place it on top of a value).

  The text for the legend (which is, confusingly, an argument called `legend`) can be created in a tedious way, being explicit about every thing, and being very, very careful you match the symbol and colors in the figure. Or it can be done using a `rep` and `paste`. Do it however you want, but this is a hint:

  ```
  lclass <- rep(levels(leuk.class), rep(2, 2))
  lsex <- rep(levels(sex), 2)
  text.legend <- paste(lclass, lsex, sep = ", ")
  ```

- Don’t forget to add the regression line.

5 Conditioning plots

The next plots are called conditioning plots. In a sense, we repeat the theme of Figure 1 (panel b), but here we plot HK-1 vs. PTEN separately for the two sexes.

When we separate by sexes, then there does not seem to be a relationship between the expression of these genes anymore. In fact, the regression line we fitted above seems to be the result of a few points with high leverage, outliers in the HK-1 expression (compare, for example, Figures 3 and 4).

What are conditioning plots good for? For conditioning. They can be used to condition on more variables (rows and columns in the figure), in variables with more than two categories, and even in continuous variables.

The general syntax is `the_y_axis ~ the_x_axis | what_we_condition_on`

The first figure figure uses the function `coplot`, and to get the smoothed line we use this argument `panel=panel.smooth`

The second and third figure use library "lattice". You do not need to install it (it comes pre-installed) but you will need to load it. You do not call `coplot` but `xyplot`. To get the lines (and points) I have used

```
panel = function(x, y) {
    panel.xyplot(x, y)
    panel.loess(x, y)
}
```

in Figure 3 and

```
panel = function(x, y) {
    panel.xyplot(x, y)
    panel.lmline(x, y)
}
```

in Figure 4.

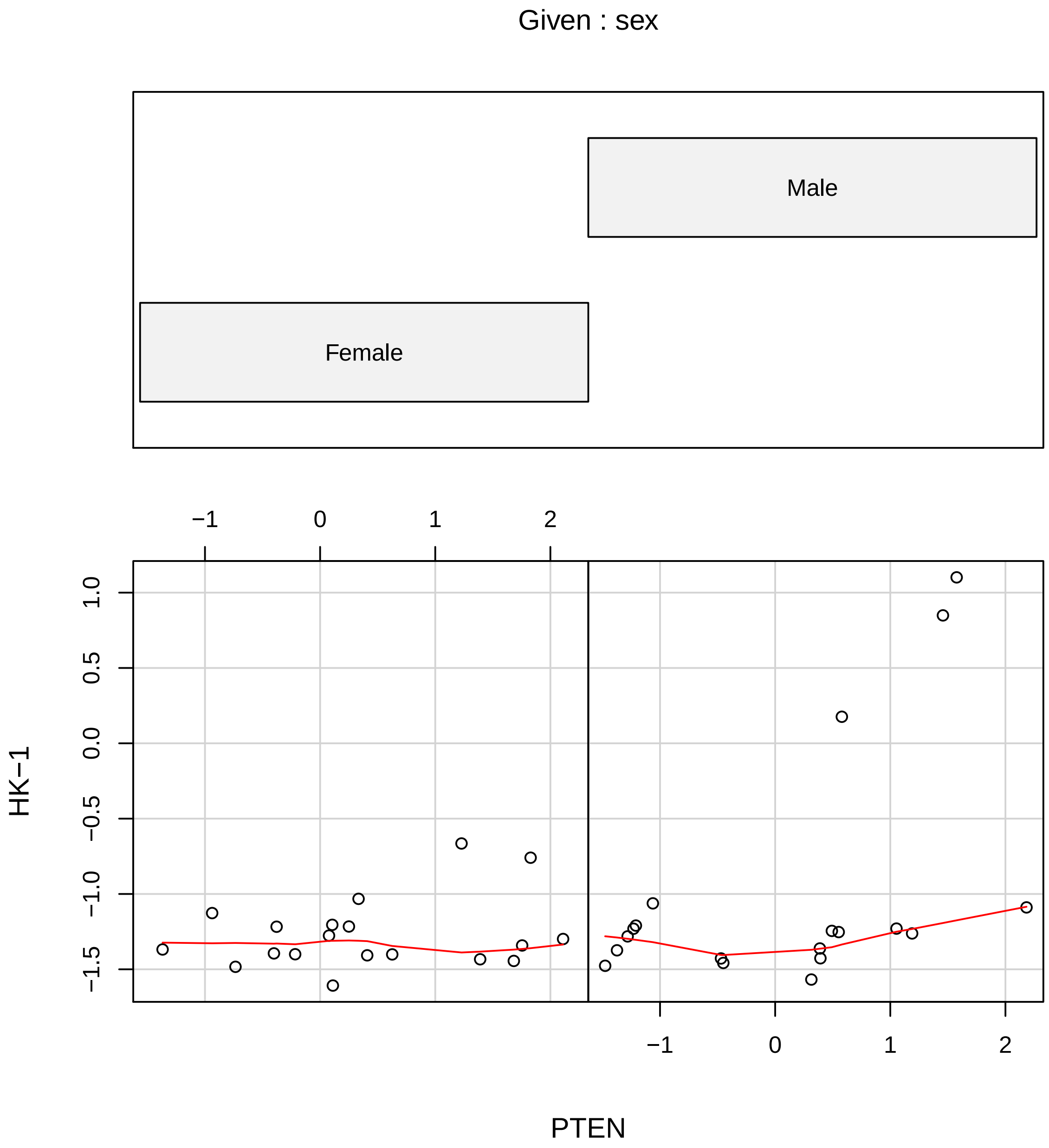

Figure 2: Conditioning plot of gene 1 vs. gene 2124 conditioned on sex with a smoothed fit.

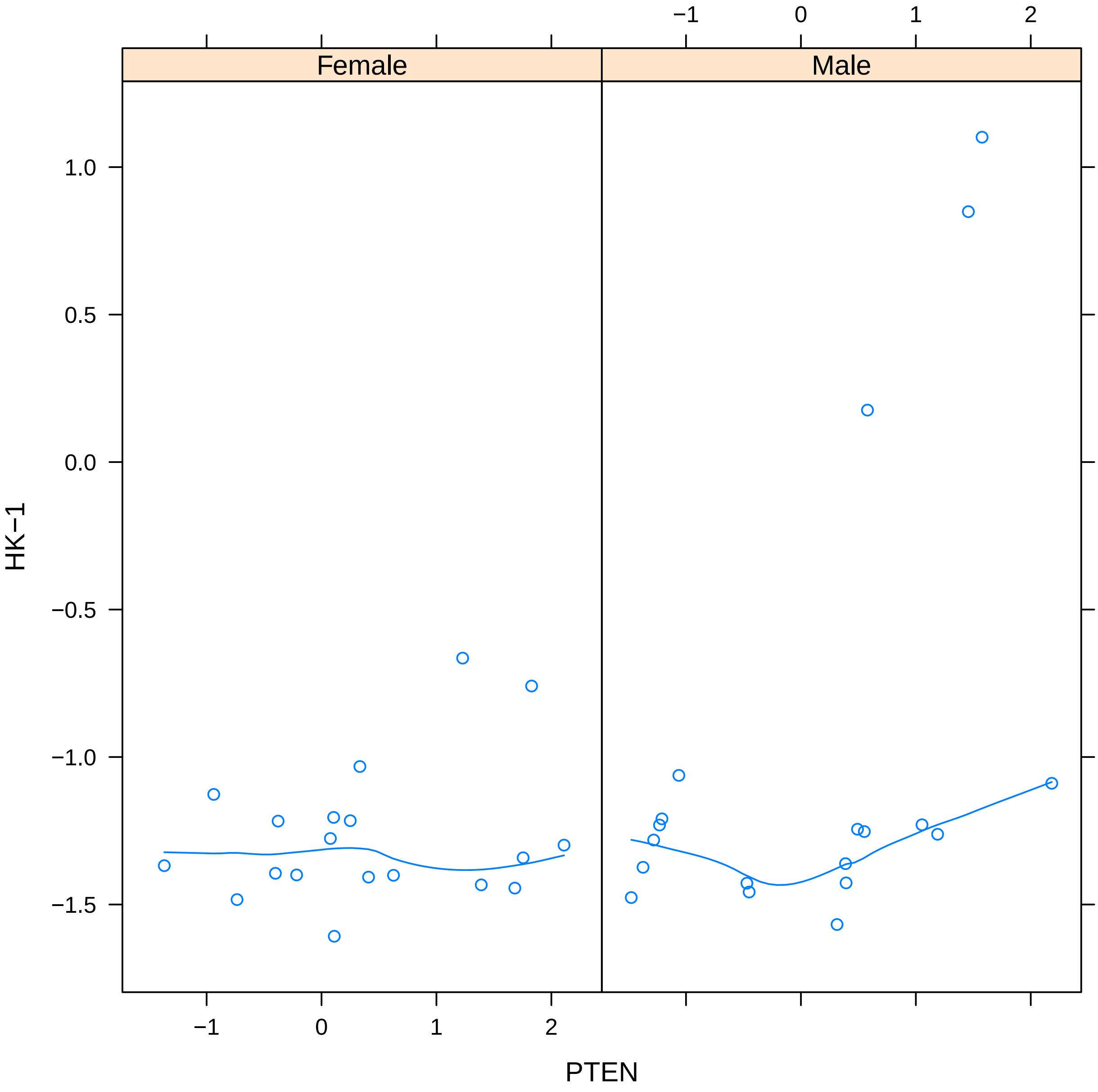

Figure 3: Conditioning plot of gene 1 vs. gene 2124 conditioned on sex with a smoothed fit, using library lattice. This is the same plot as in Figure eff04 but using lattice.

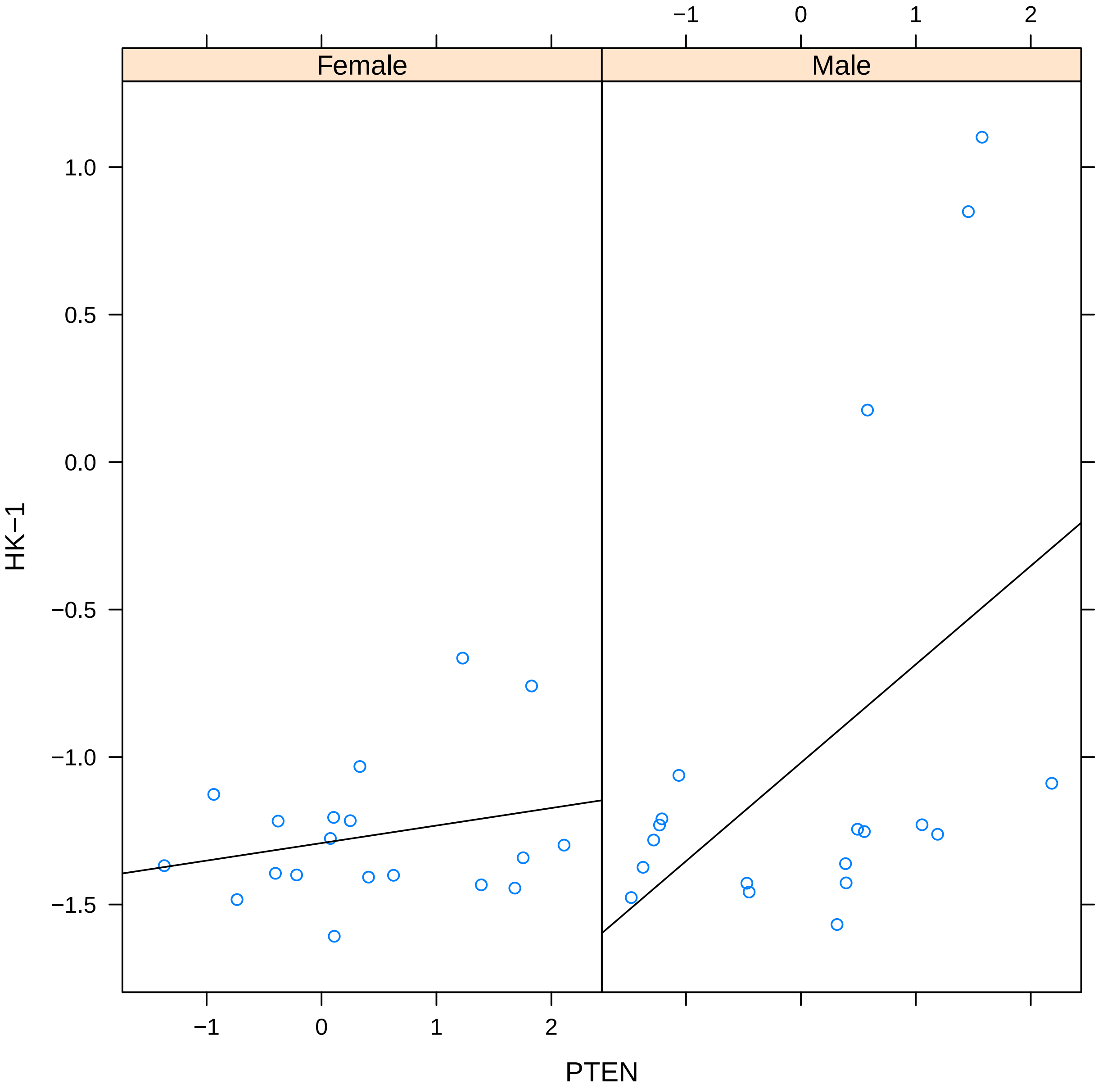

Figure 4: Conditioning plot of gene 1 vs. gene 2124 conditioned on sex with a linear fit, using library lattice. This is like Figure eff05 but with a linear (instead of a smoothed) fit.

The final figure, Figure 5, uses “ggplot2”. Maybe you can argue that this is the nicer and more elegant figure. Or maybe not.

You will need to install “ggplot2”. This is not an easy to use package, so here are three links that will allow you to solve the problem (note that two of them basically show how to translate an xyplot to a ggplot one): http://docs.ggplot2.org/0.9.3.1/translate_qplot_lattice.html, http://learnr.wordpress.com/2009/07/15/ggplot2-version-of-figures-in-lattice-multivaria
data-visualization-with-r-part-5/, http://www.cookbook-r.com/Graphs/Scatterplots_(ggplot2)/. I think the first link should be enough, but just in case.

A couple of suggestions:

- Create a data frame for the data; in other words, something like

  ```
  dgg <- data.frame(PTEN = leuk.dat.m[2124, ],
                    HK = leuk.dat.m[1, ],
                    Sex = sex)
  ```

- For “conditioning” you want what are called “facets” in ggplot. So you will end up using something like: `facet_wrap( ~ Sex)`

- Note how the plots are composed by “adding” things. So the facet code will probably be something like `ggplot(blablabla) + facet_wrap( ~ Sex)`

- I would use the plain `ggplot` function, not the supposedly easier `qplot`, but this is up to you. Note that using `ggplot` it is easy to add both types of line in the same plot. But you do not need to do that (use just the smoothed one).

- We cannot name a variable “HK-1” in the data frame, so setting the axis label for the y axis is done via `labs(y = "HK-1")`.

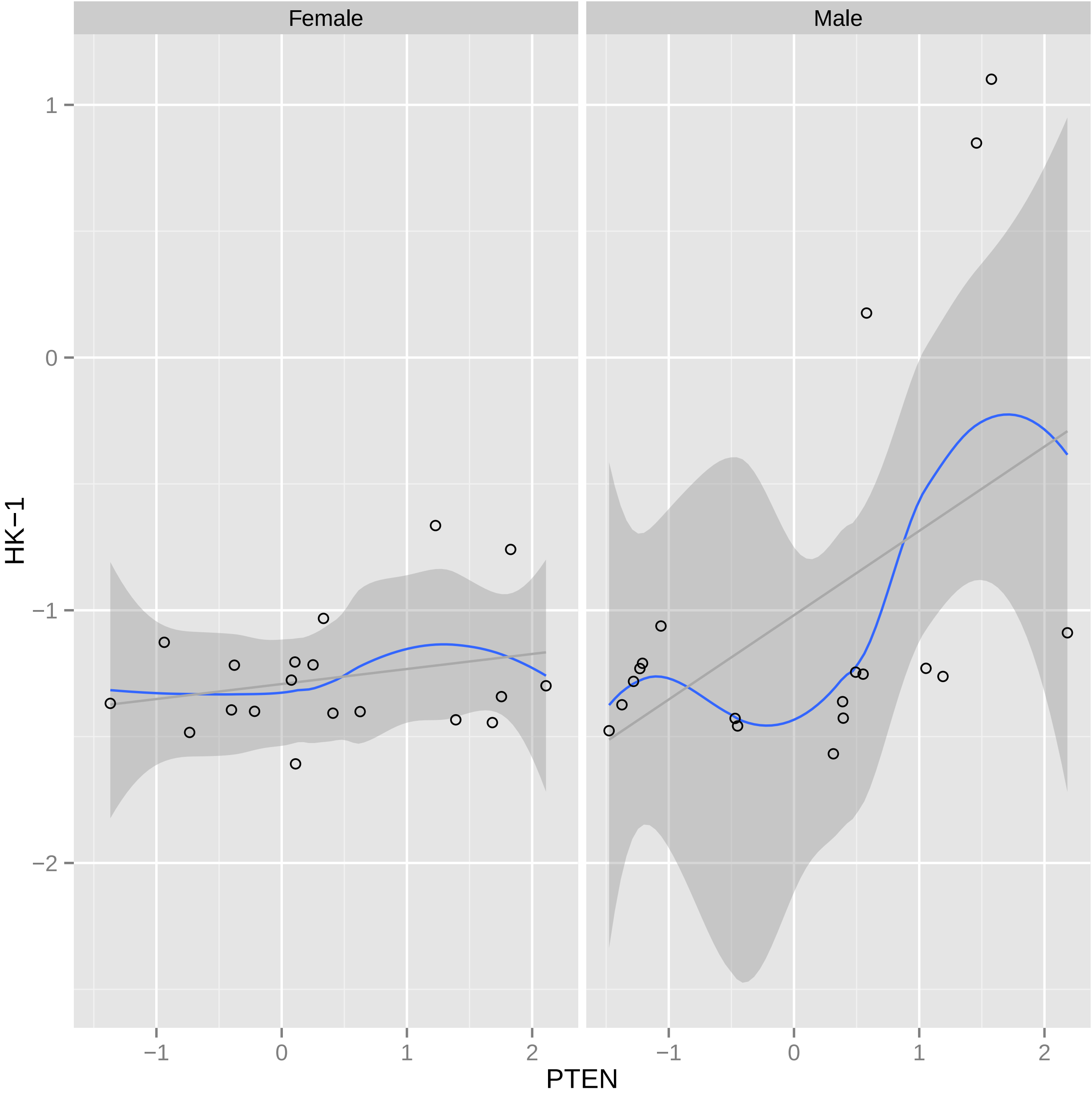

Figure 5: Yet again the conditioning plot, now using ggplot2

6 The histogram of p-values

(This exercise is really not hard, in terms of the figure itself. But maybe understanding what this is all about requires more statistics.)

- Compute the p-value from a t-test for each gene comparing the two classes of patients. Store this as "p.v.t". The code to this is in the notes.
- You can do a histogram of this immediately. Us 50 breaks. However, we want to show the thresholds for the p-values after we find the FDR (the red and blue lines).
- Oh, but notice the histogram is of density, not of frequencies (the area, summed from 0 to 1, is 1). How do you get that?
- The function `p.adjust` (with argument `method="fdr"`) will compute FDR-adjusted p-values.
- That is still not enough: that will give you the FDR-adjusted p-values, but it is easier if you . . .
- . . . sort the p-values before calling `p.adjust`. Something like this

```
sorted.p <- sort(p.v.t)
sorted.adj.p <- p.adjust(something,
                         method = something_else)
```

- How do we place the red and blue vertical lines? Those are the p-values such that, if we choose all genes with a p-value less than the line, we achieve the desired FDR.
- How do we find them? Since the FDR adjusted p-values are sorted, we just need to find the position where we reach the required limit. For instance, to place the blue line we need to:
  - Find what is the p-value where the FDR is just below 0.05.
  - That is the same as finding out how many sorted adjusted p-values (`sorted.adj.p`) are less than 0.05.
  - If you do the above, you will find that there are 695 p-values such that they satisfy that the FDR of that set is less than 0.05. Note: this is the position of the p-value, not the actual p-value.
  - To find the p-value, locate the p-value that corresponds to that position (the 695th ordered p-value). Call this "cut.05".
- Notice that the histogram contains a dotted horizontal line at 1. You will not get that by default. To do that, you need to use `axis`, and tell it where to put and what (see the help, and look at `at`).
- The figure will be nicer if, after you call `hist` you also call `box` (without arguments).
- I have also played with the limit of the y-axis. I set it to `c(0.3, 14)` as it looked nicer (to me).
- You create the legend with function `legend`. The text for the legend can be created as this (you will need to repeat this for 0.15):

```
paste("FDR <= 0.05. P <=", round(cut.05, 4))
```

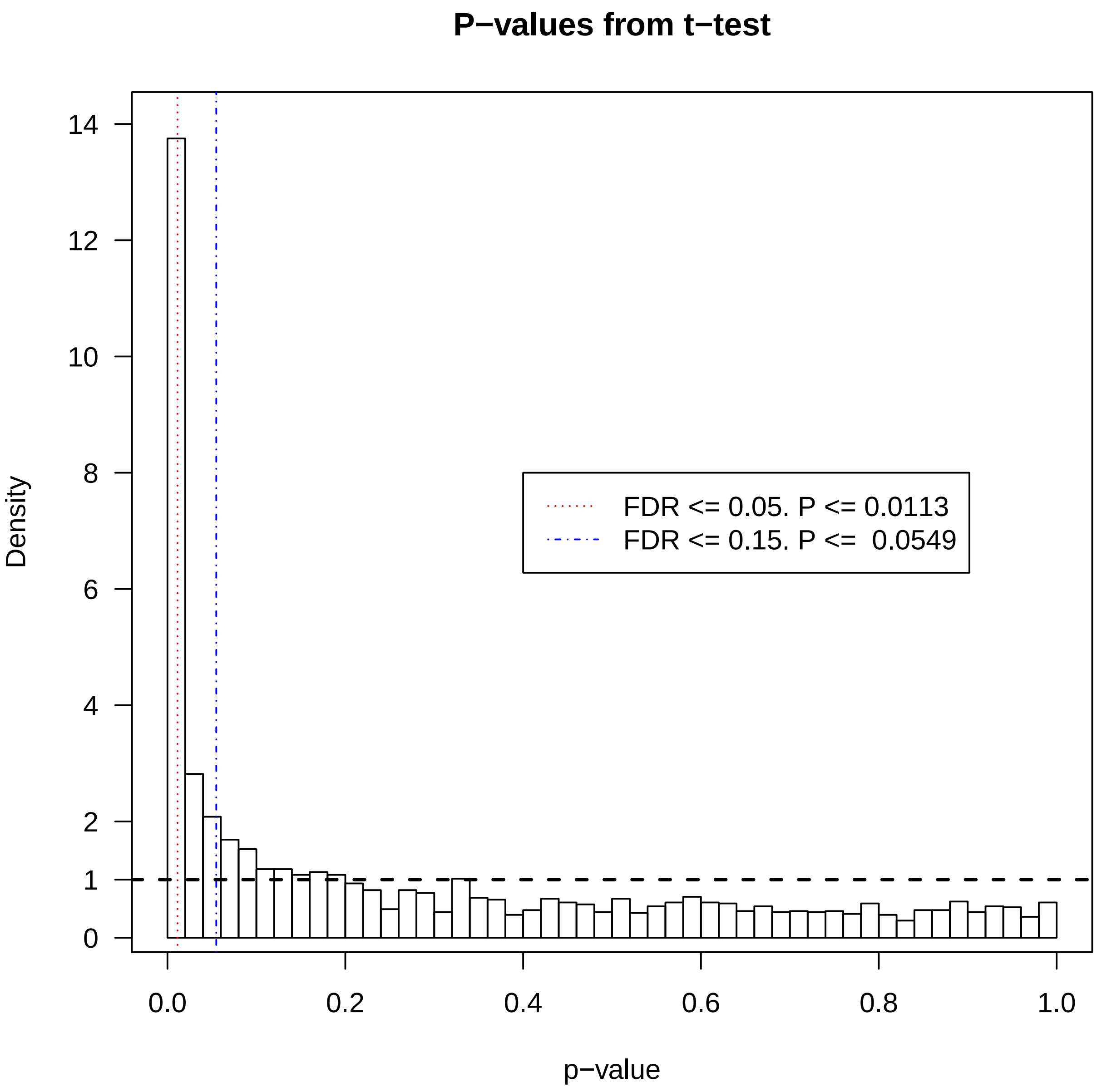

Figure 6: Histogram of p-values from comparing the ALL vs. the AML for each gene with a t-test. The cutting points for the FDR of 0.05 and 0.15 are also shown.

7 The scatterplot of p-values

- You've hear of another test, called Wilcoxon. How do p-values from that test compare to those from the t-test? Let's look at them.

- Compute p-values from a Wilcoxon test. These should be done in a way similar to the t-test, but using a Wilcoxon test.

- Plot one against each other.

- How do you get that kind of "rug". Use function `rug` (look at the help to see how to place it in each side).

- Why is rug such a nice thing here? For one thing, it easily shows that the p-values of the Wilcoxon test are discretized, unlike those from the t-test.

- Now put a line that corresponds to the place where we would select genes according to an FDR of 0.15 from the p-value of a t-test.

- Oh, and how do you get these smaller dots? Use the `cex` argument to plot.

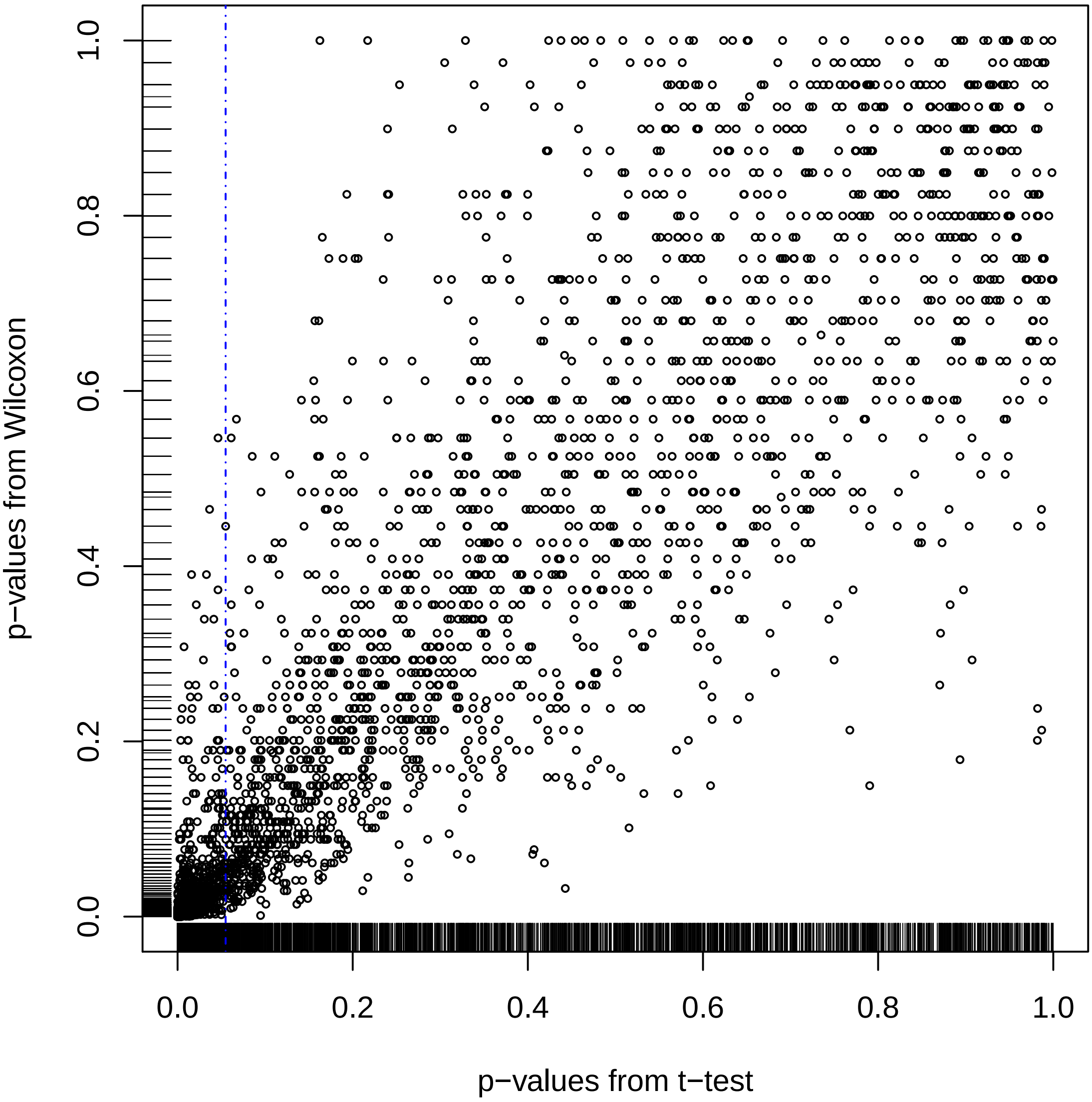

Figure 7: P-values from comparing the ALL vs. the AML with a t-test and a Wilcoxon test. In the scatterplot for the Wilcoxon vs. t-test, the blue lines shows the threshold for 0.15 FDR computed from the t-test.