R tutorial, session 2

H. Seitz (IGH)
(herve.seitz@igh.cnrs.fr)

October 8, 2014

Contents

1 Plotting data

Let's define a series of values (in a vector called `x`) and a series called `y` (of the same length than `x`):

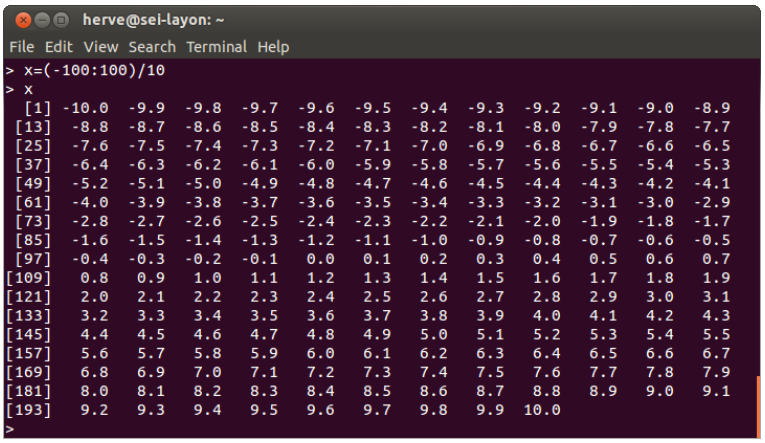

Figure 1: **Defining x values for the points to be plotted.** x is a vector or real numbers, spanning from -10 to +10, with a 0.1 increment (hence it has 201 elements).

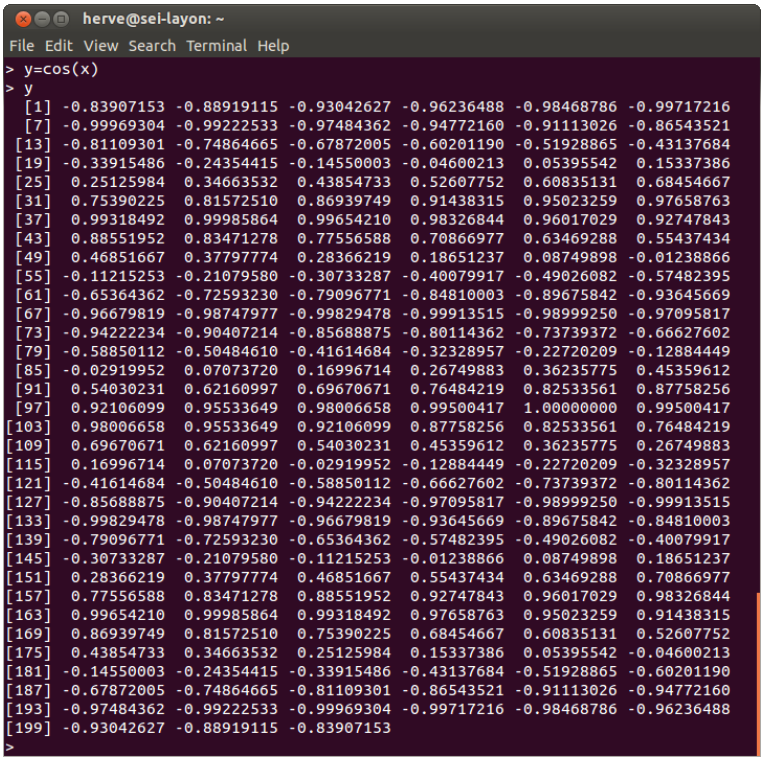

Figure 2: **Defining y values for the points to be plotted.** y is a vector or real numbers, whose values are: the cosine of the values in the x vector.

Then in order to plot these 201 points, we'll use the command `plot()`:

Figure 3: **The plot() command.**

After typing that command and pressing Enter, a new window pops up, with the graph:

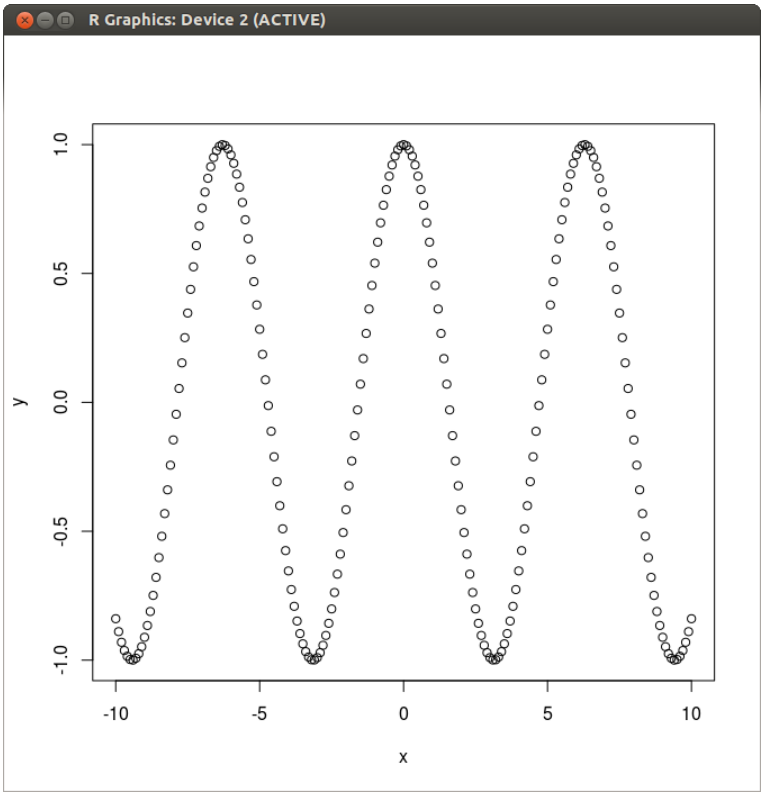

Figure 4: **A scatter plot with default settings.**

By default, each point is represented by an open circle, and the ranges of the x and y axes are set automatically to fit the extreme values in the plotted coordinates. The axes are labeled with the names of the two vectors that we plotted ("x" and "y"). Of course we could choose any name for these two variables: their names would have been displayed on the axes:

Figure 5: **Defining novel variables ("Alphonse" contains the same data than "x" and "Germaine" contains the same data than "y").**

The new graph is now annotated with the new variable names:

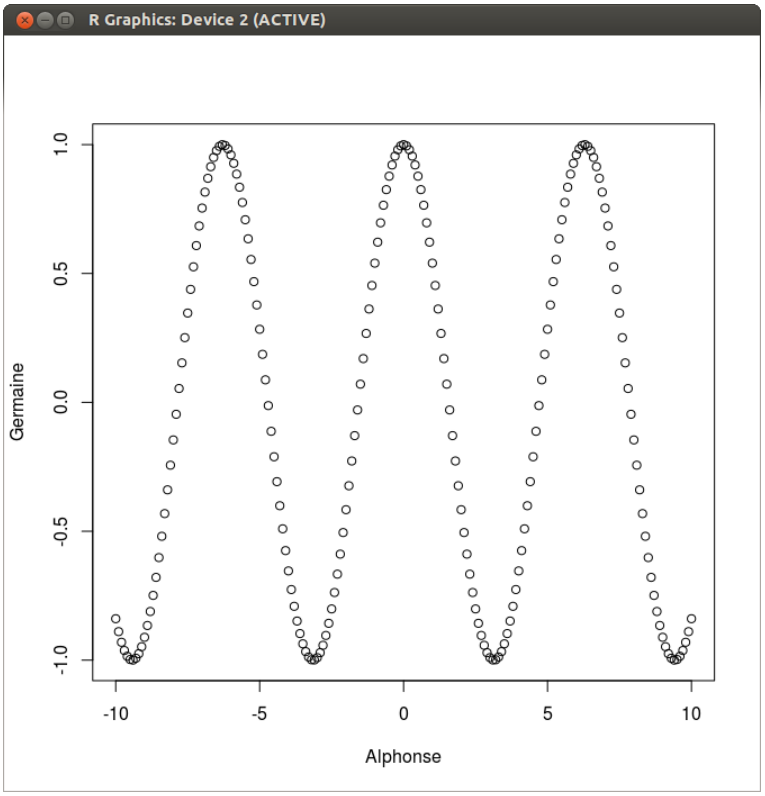

Figure 6: **Compare axis labels with those on figure 4.**

You can change many aspects of the graph, using options of the `plot()` command (type `?plot` to get a description of options; *N.B.*: as explained in that manual page, some options are more precisely described in the manual for command `par`: display it with `?par`):

- if you want to display a line joining these points (rather than open circles), use option "type" (or its abbreviation, "ty"): `plot(x,y,type='line')` or `plot(x,y,ty='l')` ("l" stands for "line");
- if you want to display both lines and open circles, set "type" to setting "b" (for: "both"): `plot(x,y,ty='b')`;
- if you want to change the color of the displayed points (or lines), set "col" (for "color") to the color you want (don't forget to put the name of the color between quotes; R has a built-in list of colors which it will recognize by their names, so "red", "blue", "green", *etc.* will be fine, but, for example, "indigo" won't work ...): `plot(x,y,ty='l',col='red')`;
- if you want to change the ranges of displayed x and y values on the graph, use the "xlim" and "ylim" options, *e.g.*, `plot(x,y,type='l',xlim=c(-20,30),ylim=c(-4,0.5))`;
- if you want to display a title at the top of the graph, use the "main" option, and if you want to change axis labels, use the "xlab" and "ylab" options: `plot(x,y,type='l',main='My plot',xlab='x values',ylab='y values')`;

2 Plotting histograms

Let's define a series of values, called `z`:

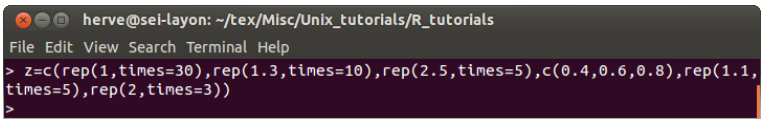

Figure 7: **Defining a data series using the rep() command.**

(`rep()` stands for "repeat": here I used it to generate large series of identical values)
Then if you type `hist(z)`, then Enter, a new graphics window will appear:

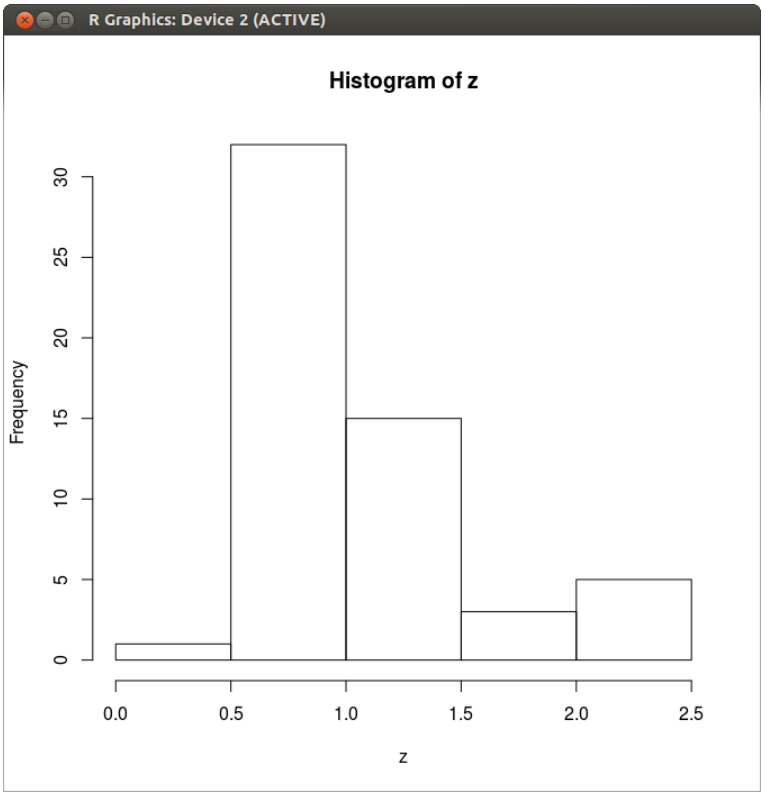

Figure 8: **A histogram, plotted with default settings.**

Just like with the `plot()` command, you can specify new axis labels, graph title, you can change the color of the histogram bars, ...:
`hist(z,xlab='My variable',ylab='Number of occurences',col='blue',main='My histogram')`

You may also want to change the width of categories to be plotted (by default, R tries to find a smart way of categorizing values; but it can be less smart than you `;-)`):
`hist(z,breaks=20)`
(this one will create 20 categories of equal width)
`hist(z,breaks=c(0,0.5,1,1.5,2,3,4))`
(this one will create breaks at the specified positions: as the numbers are not equally spaced, the categories won't have the same width)
and you can also ask R to define categories automatically, with various functions (see `?hist` for details).

3 Overlapping graphics

So far, every time we have been plotting a new graph, it erased the previous one. It is possible to overlap several graphs, using the `par(new=TRUE)` command:

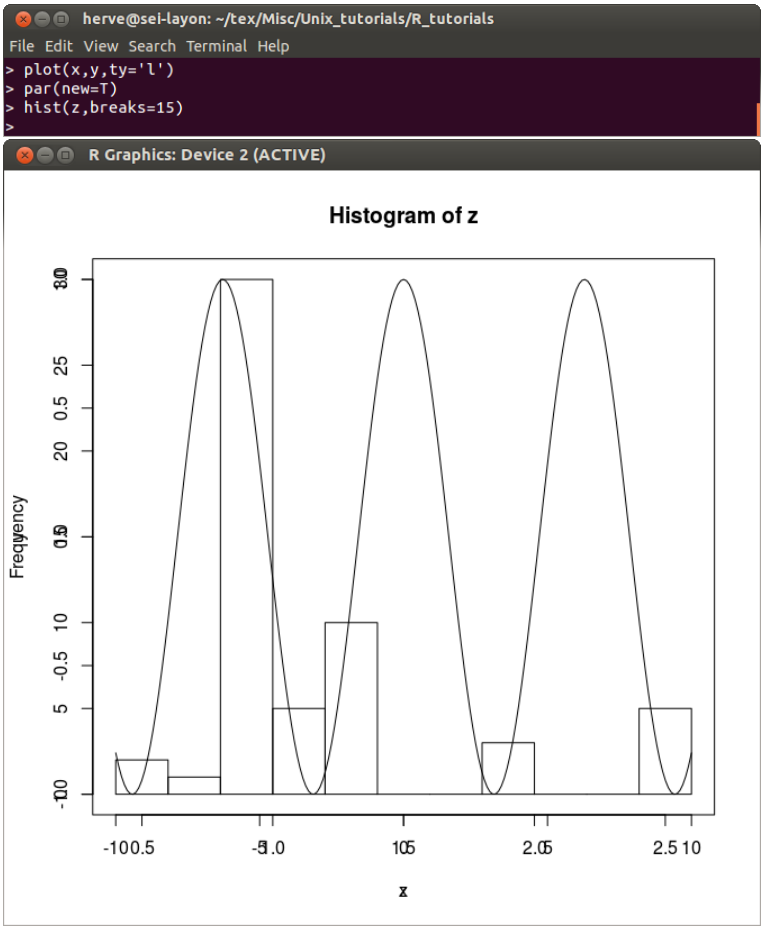

Figure 9: **Superimposing graphs.** OK, that's not the prettiest graph ever, but at least it's a good start ...

The outcome of that command may be disappointing at first: graphs are superimposed indeed, but the axes look ugly (they were superimposed too, and as they don't span the same range, ticks and tick labels look messy), axis labels and graph titles got superimposed too (well, here there was no graph title for the first graph, so you only see the histogram title), and each graph spans its own x and y ranges, so the two graphs don't have the same origin: their relative location is meaningless.

We can fix all these problems by setting new axis labels and axis ranges:

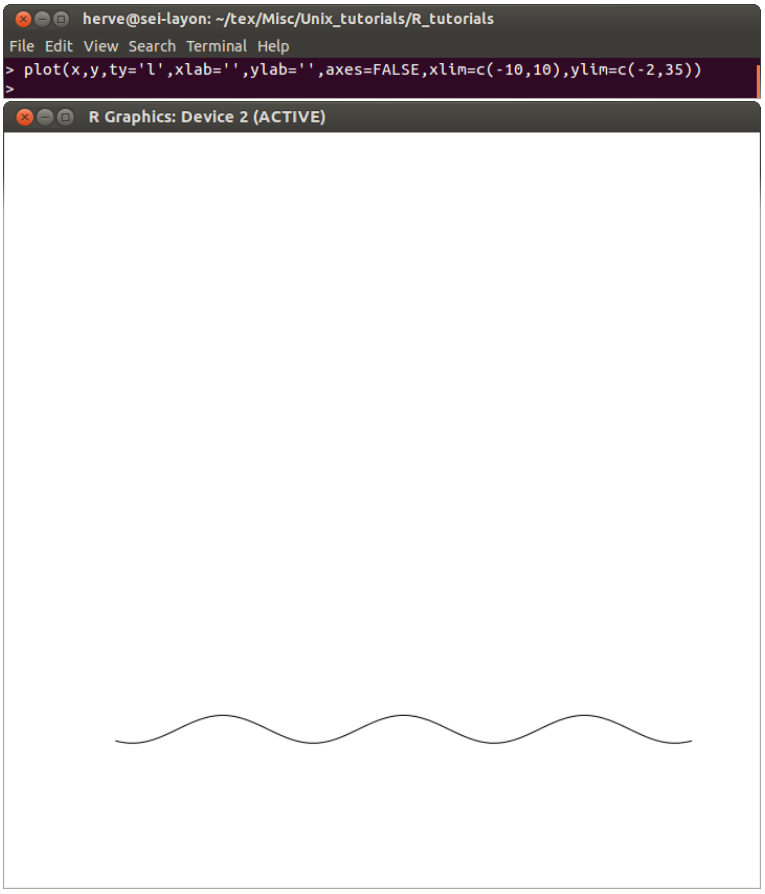

Figure 10: **Plotting a graph without axes not axis labels.**

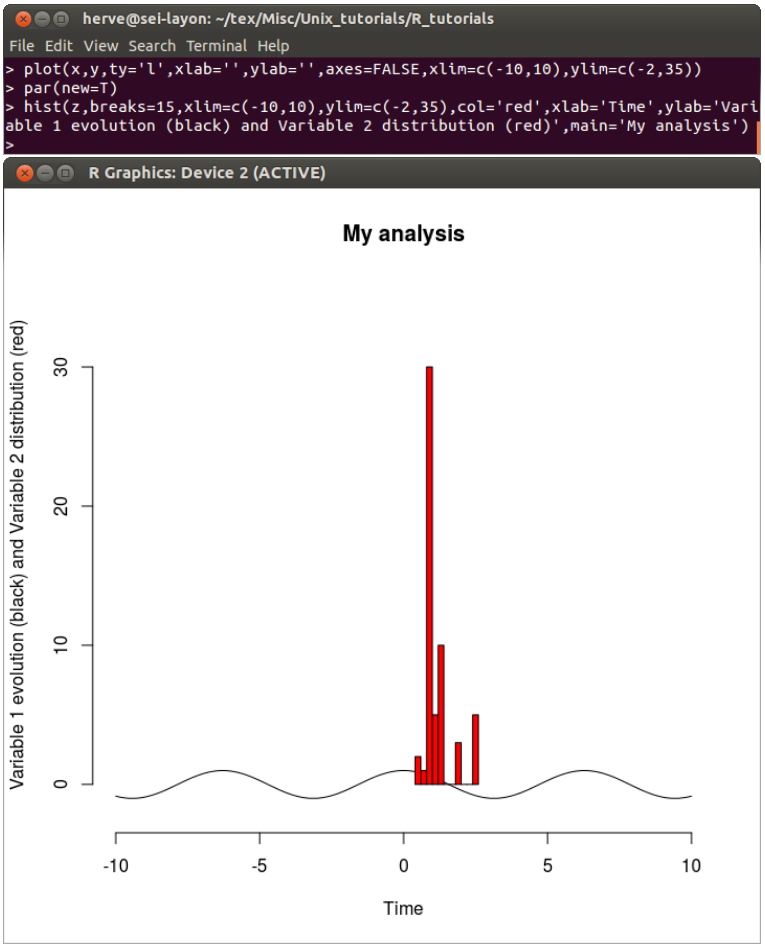

Figure 11: **Superimposing graphs in a controlled manner.**

4 Exporting graphics

If you close an R session, every pop-up window will close too: all the graphs you plotted will vanish. You may want to save them as an external file (for example, if you want to include them in a manuscript or a slideshow). You can do this using the `pdf()`, `png()`, `postscript()` or other commands (see `?png` for example).

Let's define a new variable, named `gaussian`, which will follow a normal distribution centered on 0, with a standard deviation of 1:

```
gaussian=rnorm(500,0,1)
```

(that command will generate a vector of 500 elements, following a normal distribution of mean=0 and standard deviation=1).

We can also generate a list of x and y coordinates for the curve showing the theoretical density of probability of the normal distribution, using the `dnorm()` command:

```
x=(-400:400)/100
y=dnorm(x,0,1)
```

If we plot the histogram of its distribution, superimposed with its theoretical density of probability:

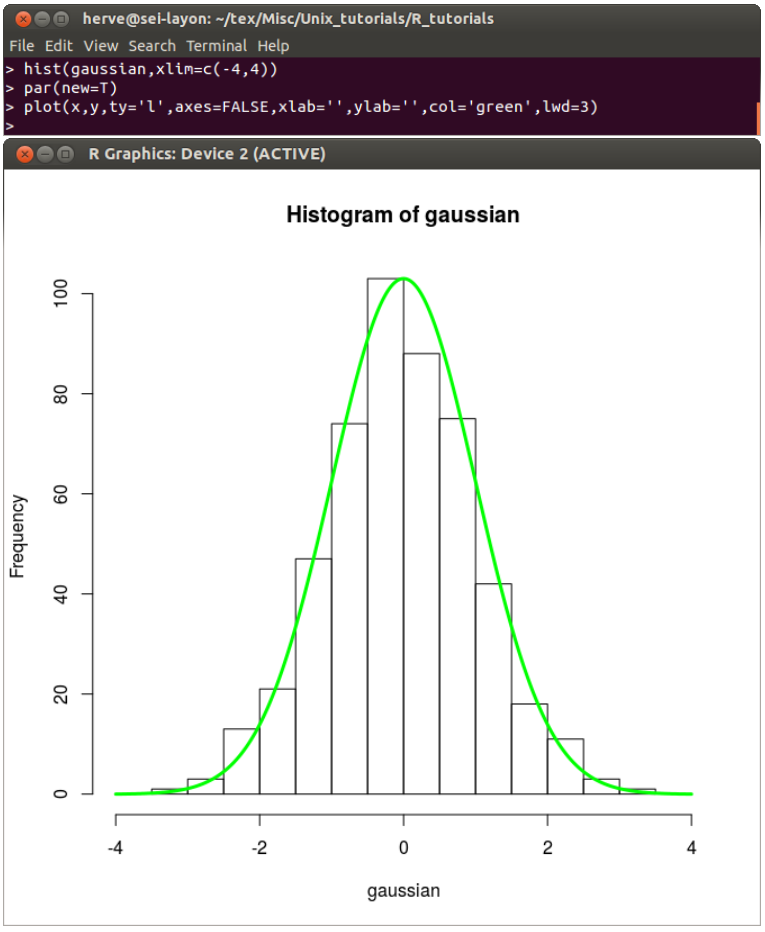

Figure 12: **Our plot, in a pop-up window.**

If we want to export it as a PNG file, we have to type the following command before plotting

the graph:
```
png('Essai.png',width=500,height=500)
```
(*N.B.*: the `width` and `height` options set the dimensions of the output file)
then the plotting commands:
```
hist(gaussian,xlim=c(-4,4))
par(new=T)
plot(x,y,ty='l',axes=FALSE,xlab='',ylab='',col='green',lwd=3)
```

then, to close the PNG file:
```
dev.off()
```
A new file, named 'Essai.png', has appeared in the current directory: it is a bitmap file containing our graph. In general, it is a good idea to export graphs under a vectorial format, rather than a bitmap (vectorial graphs usually take less disk space, they are better resolved, and they can be easily modified by an image processing program): you may export the graphs in the PDF or PS formats, using the `pdf()` and `postscript()` commands:
```
pdf('Essai.pdf',width=5,height=5)
hist(gaussian,xlim=c(-4,4))
par(new=T)
plot(x,y,ty='l',axes=FALSE,xlab='',ylab='',col='green',lwd=3)
dev.off()
postscript('Essai.ps',paper='special',width=5,height=5,horizontal=FALSE)
hist(gaussian,xlim=c(-4,4))
par(new=T)
plot(x,y,ty='l',axes=FALSE,xlab='',ylab='',col='green',lwd=3)
dev.off()
```
(*N.B.*: with the `postscript()` command, you have to specifiy `paper='special'` in addition to the dimensions of the exported file, otherwise it will be formatted under the "A4" format by default; and you have to explicitly type `horizontal=FALSE` if you don't want the graph to be rotated).