CBCS
http://agrobiol.sggw.waw.pl/cbcs

Communications in Biometry and Crop Science
Vol. 5, No. 2, 2010, pp. 69–77

International Journal of the Faculty of Agriculture and Biology,
Warsaw University of Life Sciences, Poland

SOFTWARE TRICKS AND TIPS

dotplots.errors, a new R function to ease the pain of creating dotplots

Marcin Kozak

Department of Experimental Design and Bioinformatics, Warsaw University of Life Sciences,
Nowoursynowska 159, 02-776 Warsaw, Poland.
E-mail: nyggus@gmail.com

CITATION: Kozak, M. (2010). dotplots.errors, a new R function to ease the pain of creating dotplots. *Communications in Biometry and Crop Science* 5 (2), 69–77.

Received: 21 April 2010, Accepted: 6 July 2010, Published online: 28 July 2010

ABSTRACT

Dot plots are preferred over barplots to present estimates with error bars for grouped data, yet the latter are overwhelmingly more often used in such situations. The paper presents a code for an R function to produce a dot plot with error bars, employing the utility of the `lattice` package. Easy to use for non-advanced users of `lattice` and R, it offers much possibilities to control the appearance of the plot. Both horizontal and vertical versions of the plot can be produced.

Key Words: *barplot; dotplot; lattice; visualization.*

Cleveland's (1994) dot plots should be preferred over barplots when it comes to presenting values for grouped data (e.g., Jacoby, 2006; Sarkar 2008; Kozak, 2010), including a common situation when estimates are accompanied by error bars: they graph data by position along a common scale, which is the most efficient way of showing differences among the graphed values, at the same time being free of undesirable additional dimensions (a barplot adds one such dimension: the bars' width). Their construction enables one to present much more values than for example barplots. Unfortunately, barplots are much more often used to present such data but dotplots are seldom found in the crop science literature.

Two main reasons for this are lack of awareness of the existence of dot plots among crop science researchers and lack of easily available software to produce dot plots with error bars. This paper aims to overcome both problems, offering a simple to use and easy to control R (R Development Core Team, 2009) function to produce such plots.

In R, dot plots can be produced with various functions, two examples being `dotchart()` of the graphics package (Murrell, 2005), `dotplot()` of the lattice package (Sarkar, 2008), and

`centipede.plot()` of the `plotrix` package (Lemon, 2006). The package `ggplot2` also enables one to draw such plots (Wickham, 2009). Unfortunately, only `centipede.plot()` has a build-in possibility of easy adding error bars, and constructing `lattice` and `ggplot2` plots require some knowledge of the corresponding frameworks. Additional effort is also needed to produce appropriate labels with a "±" sign, reordering of groups, adding a reference line, etc.

The `segplot()` function of the `latticeExtra` package (Sarkar and Andrews, 2010) of R can be used to draw the dot plot with error bars. The difficulties mentioned above can be showed by the below call to the `segplot()` function, which produces a plot similar to that in Figure 1. As an example, I will use data for plant height (in cm) of 13 genotypes of *Chenopodium album* L., provided by Bhargava et al. (2008):

```
> genotype <- paste("CA", as.roman(1:13), sep = "-")
> plant.height <- c(15.38, 16.94, 17.83, 12.06, 15.41,  13.52, 14.31,
         14.69, 12.45, 13.31, 8.53, 14.89, 15.25)
> SE <- c(0.46, 2.11, 1.70, 0.91, 1.35, 0.58, 0.61, 0.70, 0.73, 0.40,
         0.51, 0.28, 0.28)
> lower <- plant.height - SE; upper <- plant.height + SE
> x <- data.frame(group = genotype, lower = lower, est = plant.height,
          upper = upper)
```

resulting in the following x data frame after binding the columns:

```
     group lower   est upper
1     CA-I 14.92 15.38 15.84
2    CA-II 14.83 16.94 19.05
3   CA-III 16.13 17.83 19.53
4    CA-IV 11.15 12.06 12.97
5     CA-V 14.06 15.41 16.76
6    CA-VI 12.94 13.52 14.10
7   CA-VII 13.70 14.31 14.92
8  CA-VIII 13.99 14.69 15.39
9    CA-IX 11.72 12.45 13.18
10    CA-X 12.91 13.31 13.71
11   CA-XI  8.02  8.53  9.04
12  CA-XII 14.61 14.89 15.17
13 CA-XIII 14.97 15.25 15.53
```

Now, let us produce the plot:

```
> library(latticeExtra)
> segplot(reorder(genotype, est) ~ lower + upper, data = x,
   draw.bands = FALSE, centers = est, segments.fun = panel.arrows,
   ends = "both", angle = 90, length = .03,
   par.settings = simpleTheme(pch = 19, col = 1),
   xlab = expression("Plant height (cm) " %+-% " SE"),
   panel = function(x, y, z, ...) {
    panel.abline(h = z, col = "grey", lty = "dashed")
    panel.abline(v = 14.20, col = "grey")
    panel.segplot(x, y, z, ...)})
```

The vertical reference line in the plot represents the overall mean (14.20, according to Bhargava et al., 2008).

Clearly, this function requires some knowledge of R and `lattice`. The function `dotplot.errors()` proposed in this paper is easy to use, which is agreat advantage for non-advanced users struggling to learn R syntax while satisfying their data analysis needs, and enables one to produce the same graph as with `segplot()` above with this very simple call:

```
> dotplot.errors(x, qlabel = "Plant height (cm) ", reference.line = 14.20)
```

Nevertheless, besides being simple for such regular plots, it enables one to control all the above-mentioned elements. Like for other `lattice` plots, it is relatively easy to produce a PDF, POSTSCRIPT, PNG, JPEG, BMP file with the plot, but this needs to be done outside the function.

The code for the function is presented in the Appendix. The detailed description of all the arguments of the function is presented in Table 1, where examples of use are also provided.

The plotting symbol, size and color, bars' color and font size of the label can be altered through a `simpleTheme()` function of `lattice`, which can be used to change the default appearance of these elements, e.g.:

```
> myTheme <- simpleTheme(pch = 15, cex = 2, col = "red")
          # pch = 15 indicates closed squares as plotting symbols,
            while cex = 2 means two times bigger plotting symbols
            than the default
> dotplot.errors(x, qlabel = "Plant height (cm) ",
          myTheme = myTheme, label.define = list(cex = 1.5),
          bar.color = "red")
          # reference line omitted but will be reintroduced later on
```

Note also how the font size of the x-axis label was altered with the argument `label.define`, and color of bars with the argument `bar.color`. Instead of constructing an object `myTheme`, being a `simpleTheme()`, one could simply add such `myTheme` argument to the call (as will be shown below).

Thanks to the default `reorder.groups = TRUE` and `reordering = "decreasing"`, the groups were reordered by estimate, from highest to smallest. We can reorder from smallest to highest:

```
> dotplot.errors(x, qlabel = "Plant height (cm) ", reordering = "increasing")
```

but we can also keep the original ordering of the factor `x$group` (which as default is alphabetical):

```
> dotplot.errors(x, qlabel = "Plant height (cm) ", reorder.groups = F)
```

Should one want to use some particular ordering (say, we want first the 10th row, then rows from 1 to 9, and then from 11 to 13), the `x$group` factor needs to be made an ordered factor, as here:

```
> xx <- x
> xx$group <- ordered(xx$group,
                      levels = levels(xx$group)[c(10, 1:9, 11:13)])
```

Let's change the length of endings of the error bars, add some text to the label, and once more let us use the reference line:

```
> dotplot.errors(x,
          myTheme = simpleTheme(pch = 19, cex = 1.5, col = 1),
          qlabel = "Plant height (cm)", reference.line = 14.20,
          add.text = " (over three environments)", end.length = .125)
```

Encapsulated postscript can be produced like this:

```
> myTheme <- simpleTheme(pch = 19, cex = .5, col = 1)
> myplot <- dotplot.errors(x, qlabel = "Plant height (cm) ",
          myTheme = myTheme, label.define = list(cex = .7),
          scales = list(cex = .6), aspect = 1.5, end.length = .03)
```

```
> trellis.device("postscript", file = "my_file.eps", width = 3, height = 4,
          paper = "special", horizontal = F)
   # other devices may be called by replacing the keyword with the
   # appropriate device name, always in lower case
> print(myplot)
> dev.off()
```

This will produce Figure 1. Notice in `myplot` plot the way I have passed the `scales` argument to the `stripplot()` function, decreasing the font of tick mark labels, and the `aspect` argument to set the aspect ratio of the graph (which describes the ratio of graph height and width) to 1.5.

Above we needed to assign the plot into an object in order to print it into a file. This is a regular feature of `lattice` plots, and in fact the plots produced with the `dotplot.errors()` function are still `lattice` objects, which you can see by typing

```
> str(myplot)
```

Hence they can be further modified as regular `lattice` plots, e.g.

```
> myplot$aspect.ratio <- 1.75
> myplot$x.limits <- c(0, 20)
> print(myplot)
```

Regular `lattice` plots can be updated, and so – to some extent – can `dotplot.errors` plots:

```
> myplot2 <- update(myplot, cex = 1.25)
> print(myplot2)
```

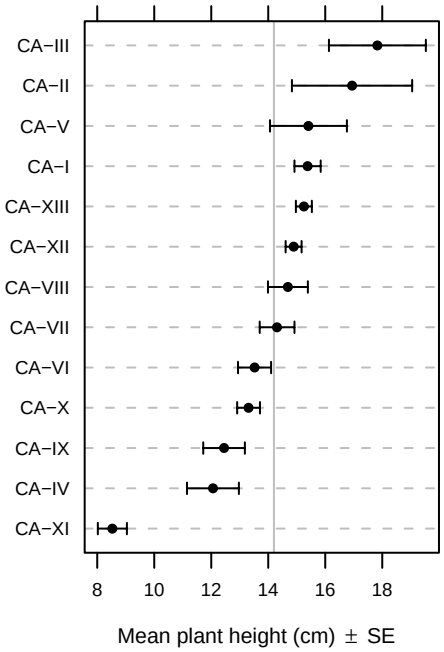

Figure 1. Plant height of 13 *Chenopodium album* L. genotypes. Error bars represent standard errors, and a reference lines the overall mean. Data source: Bhargava et al. (2008).

Sometimes a researcher can prefer a vertical version of the dot plot, although this is acceptable only when the resulting plot is equally well readable as the horizontal dot plot (Kozak, 2010). None of the above-mentioned function will easily produce such a plot, but `dotplot.errors` will, by setting the argument `horizontal` = FALSE will draw the vertical dot plot:

```
> dotplot.errors(x, qlabel = "Plant height (cm) ",
          aspect = .5, reference.line = 14.20, horizontal = F)
```

which will produce Figure 2. The aspect ratio of the plot needed to be changed to deal with the genotype names forming the tick mark labels at the horizontal axis.

Finally, function arguments in R can be abbreviated if the abbreviated name matches only one argument from the list of possible arguments to the function. So this will work:

```
> dotplot.errors(x, q = "Plant height (cm) ", ref = 14.20)
```

but this will not:

```
> dotplot.errors(x, q = "Plant height (cm) ", ref = 14.20, reo = F)
```

because it is not unequivocal because both `reordering` and `reorder.groups` arguments share the three initial letters.

This paper's web page is http://agrobiol.sggw.waw.pl/~cbcs/articles/5_2_2/, where one can find the function and some demos. In practice, downloading the function into the R console can be done by typing

```
> source("http://agrobiol.sggw.waw.pl/~cbcs/articles/5_2_2/function_www.R")
```

and the plots can now be drawn.

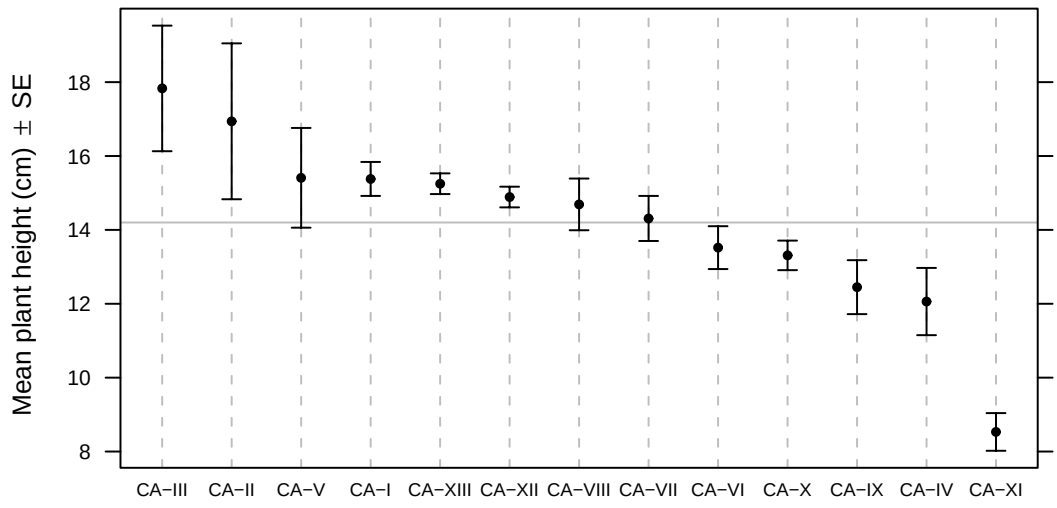

Figure 2. A vertical version of the dot plot from Figure 1. Data source: Bhargava et al. (2008).

Table 1. Description of the `dotplot.errors` function.

`dotplot.errors`

Description

The function produces a dot plot with error bars, using the lattice framework. Much control over the appearance is possible through the function arguments and "...", which passes such additional arguments to the stripplot() function. The plots are lattice objects, so they can be modified as regular lattice plots. Both horizontal and vertical dot plots can be drawn.

Usage

```
dotplot.errors <- function(x, myTheme = simpleTheme(pch = 19, col = 1),
      qlabel = "Estimate ", add.text.to.qlabel = "", type.bar = "SE",
      conf.level = .95, end.length = .05, reorder.groups = TRUE,
      reordering = "decreasing", label.define = list(), reference.line = NULL,
      bar.color = 1, horizontal = TRUE, ...)
```

Arguments

x	a data frame with columns group, lower, est, upper
myTheme	a theme to control the dotplot settings, appropriate for lattice; type ?simpleTheme for help about what and how can be altered
qlabel	a character string giving the information to be placed before the ± sign in the label for the quantitative variable; end it with a blank space; the default "Estimate " has little meaning, so it is desirable to change it
add.text.to.label	a character string that can be added at the end of the label (default to nothing); start it with a blank space
type.bar	a character string informing about what the error bars represent—this will be included in the label; can be "SE" (the default), "CI" and any other; for CI, the confidence level is also added to the label
conf.level	confidence level to be used in the label if type.bar = "CI"
end.length	length of the segment at the end of error bars (in inches)
reorder.groups	logical: should the groups be reordered by estimate before plotting? Default to TRUE
reordering	used if reorder.groups = TRUE; default to "decreasing", which means that at the top (left for a vertical dot plot) the groups with the highest estimates will be put; the reversed ordering can be obtained by setting reordering = "increasing"
label.define	optional list of parameters used to affect the axis label; for example, font size can be decreased by setting label.define = list(cex = 0.75); default to nothing (an empty list)
reference.line	an optional numeric value producing a grey reference line at the coordinate of reference.line (can be an overall mean, for example); default to NULL, in which case no reference line is produced; in case a vector of numeric values is provided, several reference lines can be drawn
bar.color	color of error bars; default to black
horizontal	logical: should the horizontal dotplot be drawn (default)? FALSE will draw a vertical plot
...	other (optional) arguments passed to the stripplot() function; in particular, the aspect argument can be used to control the aspect ratio of the graph (e.g., aspect = 1.5), the scales argument (provided as a list, e.g. scales = list(y = list(cex = .8), x = list(cex = .6))) to control the font size for axis labels, and to add a label for a qualitative axis (e.g., ylab = "Genotype")

Table 1 continued on next page

Table 1 continued

Value

The plot is produced, and returned as a lattice object if assigned.

Examples

```
source("http://agrobiol.sggw.waw.pl/~cbcs/articles/5_2_2/function_www.R")

# Example from Bhargava et al. (2008):
genotype <- paste("CA", as.roman(1:13), sep = "-")
SE <- c(0.46, 2.11, 1.70, 0.91, 1.35, 0.58, 0.61, 0.70, 0.73, 0.40, 0.51, 0.28,
0.28)
plant.height <- c(15.38, 16.94, 17.83, 12.06, 15.41,  13.52, 14.31, 14.69, 12.45,
13.31, 8.53, 14.89, 15.25)
lower <- plant.height - SE; upper <- plant.height + SE
x <- data.frame(group = genotype, lower = lower, est = plant.height, upper =
upper)

dotplot.errors(x, qlabel = "Plant height (cm) ")

dotplot.errors(x, myTheme = simpleTheme(pch = 19, cex = 1.5, col = 1),
       q = "Plant height (cm) ", add.text = " (over three environments)",
       end.length = .125)

# The plots are lattice objects:

myplot <- dotplot.errors(x, qlabel = "Plant height (cm) ",
       myTheme = simpleTheme(pch = 19, cex = .75, col = 1),
       label.define = list(cex = .7), scales = list(cex = .6), aspect = 1.5,
       end.length = .03)
print(myplot)
str(myplot)

# A vertical version of the dot plot:
dotplot.errors (x, qlabel = "Plant height (cm) ",
       reference.line = 14.20, aspect = .5, horizontal = F)

# Artificial ordering plus two reference lines:
xx <- x
xx$group <- ordered(xx$group, levels = levels(xx$group)[c(10, 1:9, 11:13)])
dotplot.errors (xx, q = "Plant height (cm) ", reorder.g = F,
       ref = c(12, 14.20), aspect = .5, horiz = F)


# More examples can be found in
http://agrobiol.sggw.waw.pl/~cbcs/articles/5_2_2/demo_www.R
```

APPENDIX: THE R CODE FOR THE FUNCTION

```
dotplot.errors <- function(x, myTheme = simpleTheme(pch = 19, col = 1),
    qlabel = "Estimate", add.text.to.qlabel = "", type.bar = "SE",
    conf.level = .95, end.length = .05, reorder.groups = TRUE,
    reordering = "decreasing", label.define = list(), reference.line = NULL,
    bar.color = 1, horizontal = TRUE, ...)
    { require(lattice)
      o <- c(which(colnames(x) == "group"), which(colnames(x) == "lower"),
             which(colnames(x) == "est"), which(colnames(x) == "upper"))
      if (length(o) != 4) stop("Error: Incorrect data frame")
      x <- x[, o]
      x$group <- factor(x$group)
      if (horizontal == T) hor <- 1 else hor <- -1
      if (reordering == "decreasing")
             FUN.to.reorder <- function(x) mean(x) * hor
       else FUN.to.reorder <- function(x) -mean(x) * hor
      if (reorder.groups) x$group <-
             with(x, reorder(group, est, FUN = FUN.to.reorder))
      if (type.bar == "CI") xlab <- substitute(expression(lab %+-% CL),
          list(lab = qlabel, CL = paste(" ", as.character(conf.level * 100),
                    "% CI", add.text.to.qlabel, sep = ""))) else
      if (type.bar == "SE")
          xlab <- substitute(expression(lab  %+-% " SE" ~ ~ AT),
          list(lab = qlabel, AT = add.text.to.qlabel)) else
      xlab <- substitute(expression(lab %+-% Type.bar ~ ~ AT),
             list(lab = qlabel, Type.bar = type.bar, AT = add.text.to.qlabel))

      if (horizontal == T)
       p <- stripplot(group ~ est, data = x,
             lower = x$lower, upper = x$upper,
             xlim = range(x[,2:4])+c(-.04,.04)*diff(range(x[,2:4])),
             xlab = c(list(xlab), label.define),
             par.settings = myTheme,
             panel = function(x, y, lower, upper, ..., subscripts) {
             if (is.null(reference.line) == F)
                   panel.abline(v = reference.line, col = "grey")
             panel.abline(h = y, col = "grey", lty = "dashed")
             panel.arrows(x0 = lower[subscripts], y0 = y,
                   x1 = upper[subscripts], y1 = y, angle = 90, code = 3,
                   length = end.length, col = bar.color)
             panel.stripplot(x, y, ...) }, ...) else
      p <- stripplot(est ~ group, data = x, lower = x$lower, upper = x$upper,
             ylim = range(x[,2:4])+c(-.04,.04)*diff(range(x[,2:4])),
             ylab = c(list(xlab), label.define),
             par.settings = myTheme,
             panel = function(x, y, lower, upper, ..., subscripts) {
             if (is.null(reference.line) == F)
                   panel.abline(h = reference.line, col = "grey")
             panel.abline(v = x, col = "grey", lty = "dashed")
             panel.arrows(y0 = lower[subscripts], x0 = x,
                   y1 = upper[subscripts], x1 = x, angle = 90, code = 3,
                   length = end.length, col = bar.color)
             panel.stripplot(x, y, ...)       }, ...)

      print(p)
      invisible(p)
    }
```

REFERENCES

Bhargava, A., Shukla, S., Ohri, D. (2008). Genotype × environment interaction studies in *Chenopodium album* L.: an underutilized crop with promising potential. *Communications in Biometry and Crop Science* 3 (1), 3–15.

Cleveland, W.S. (1994). *The elements of graphing data*. 2nd ed. Hobart Press, Summit, New Jersey, USA.

Jacoby, W.G. (2006). The dot plot: A graphical display for labeled quantitative values. *The Political Methodologist* 14(1), 6–14.

Kozak, M. (2010). Basic principles of graphing data. *Scientia Agricola* 67, 483–494.

Lemon, J. (2006). Plotrix: a package in the red light district of R. *R-News*, 6(4), 8–12.

Murrell, P. (2005). *R Graphics*. Chapman & Hall/CRC Press.

R Development Core Team (2009). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Sarkar, D. (2008). *Lattice Multivariate Data Visualization with R*. Springer.

Wickham, H. (2009). *ggplot2: elegant graphics for data analysis*. Springer New York.

Sarkar, D., Andrews, F. (2010). *latticeExtra: Extra Graphical Utilities Based on Lattice. R package version 0.6-9*. http://CRAN.R-project.org/package=latticeExtra.