R Help Desk

An Introduction to Using R's Base Graphics

Marc Schwartz

Preface

As the use of R grows dramatically, an increasingly diverse base of users will begin their exploration of R's programmatic approach to graphics. Some new users will start without prior experience generating statistical graphics using coded functions (ie. they may have used GUI based "point-and-click" or "drag-and-drop" graphic processes) and/or they may be overwhelmed by the vast array (pardon the pun) of graphic and plotting functions in R. This transition can not only present a steep learning curve, but can perhaps, by itself, become a barrier to using R entirely, which would be an unfortunate outcome.

R has essentially two separate core plotting environments in the default (**base** plus 'recommended package') installation. The first is the extensive set of base graphic functions and the second is the combination of the **grid** (Murrell, 2002) and **lattice** packages (Sarkar, 2002), which together provide for extensive Trellis conditioning plots and related standardized functionality. For the purpose of this introduction, I shall focus exclusively on the former.

The key advantages of a programmatic plotting approach are much finer control over the plotting process and, importantly, reproducibility. Days, weeks or even months later, you can return to reuse your same code with the same data to achieve the same output. Ultimately, productivity is also enhanced because, once created, a single plotting function can be called quickly, generating one or an entire series of graphics in a largely automated fashion.

R has a large number of "high" and "low" level plotting functions that can be used, combined and extended for specific purposes. This extensibility enables R to meet a wide spectrum of needs, as demonstrated by the number of contributed packages on CRAN that include additional specialized plotting functionality.

The breadth of base plotting functions is usually quite satisfactory for many applications. In conjunction with R's innate ability to deal with data in vectorized structures and by using differing 'methods', one can further reduce the need for lengthy, repetitive and complex code. In many cases, entire data structures (ie. a linear model object) can be passed as a single argument to a single plotting function, creating a default plot or series of plots.

Further, where default plot settings are perhaps inappropriate for a given task, these can be adjusted to your liking and/or disabled. The base graphic can be enhanced by using various lower level plotting functions to add data points, lines, curves, shapes, titles, legends and text annotations. Formatted complex mathematical formulae (Murrell and Ihaka, 2000; Ligges, 2002) can also be included where required.

If a graphics 'device' is not explicitly opened by the user, R's high level plotting functions will open the default device (see `?Devices`) specified by `options("device")`. In an interactive session, this is typically the screen. However, one can also open an alternative device such as a bitmap (ie. PNG/JPEG) or a PostScript/PDF file for publishing and/or presentation. I will focus on using the screen here, since the particulars concerning other devices can be platform specific. Note that if you intend to create plots for output to something other than the screen, then you must explicitly open the intended device. Differences between the screen and the alternate device can be quite significant in terms of the resultant plot output. For example, you can spend a lot of time creating the screen version of a plot, only to find out it looks quite different in a PostScript file,

Various parameters of the figure and plot regions within a device can be set in advance by the use of the `par()` function before calling the initial plot function. Others can be set as named arguments to the plot functions. Options set by `par()` affect all graphics; options set in a graphics call affect only that call. (See `?par` and `?plot.default` for some additional details).

It is possible to divide the overall graphic device into a row/column grid of figures and create individual plots within each grid section (ie. a matrix of scatterplots like a `pairs()` plot) or create a graphic that contains different plot types (ie. a scatterplot with boxplots placed on the x and y axes). For more information, see `?layout`, `?split.screen` and graphic parameters 'mfcol' and 'mfrow' in `?par`.

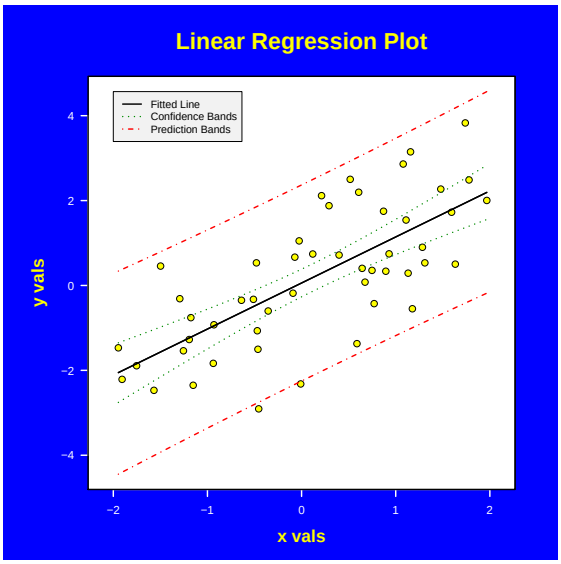
For additional details regarding graphic devices, parameters and other considerations, please review "Graphical Procedures" (Ch. 12) in "An Introduction to R" (Venables, Smith and R Core, 2003) and "Graphics" (Ch. 4) in "Modern Applied Statistics with S" (Venables and Ripley, 2002).

Let's Get Plotting

In this limited space, it is not possible to cover all the combinations and permutations possible with R's base graphics functionality (which could be a thick book in its own right). Thus, I will put forth a finite set of practical examples that cover a modest range of base plots and enhancements. For each plot,

we will create some simple data to work with, create a basic plot using a standard function to demonstrate default behavior and then enhance the base plot with additional detail. The included graphic for each will show the final result. I recommend that you consult the R help system for each function (using `?FunctionName`) to better understand the syntax of each function call and how each argument impacts the resultant output.

Scatterplot with a regression line and confidence / prediction intervals



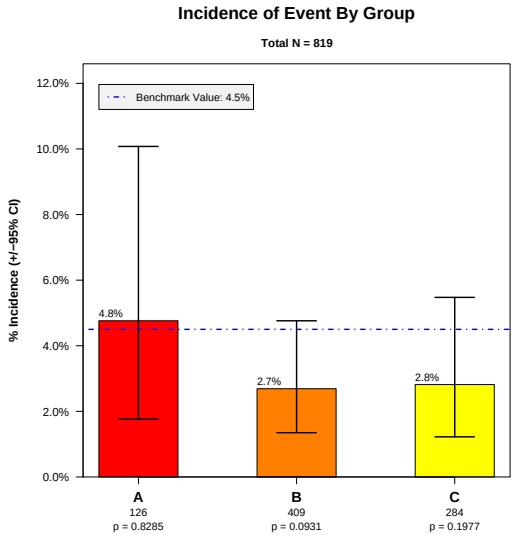
The `plot()` function is a generic graphing function that can accept of variety of data structures through specific defined 'methods'. Frequently, these arguments are numeric vectors representing the two-dimensional (x,y) coordinate pairs of points and/or lines to display. If you want to get a feel for the breadth of plotting methods available use `methods(plot)`.

In the next example we first create a series of simple plots (not shown) then create the more complex scatterplot shown above. To do this we create an x-y scatterplot using `type = "n"` so that the axis ranges are established, but nothing is plotted initially. We then add the data points, the axes, a fitted regression line, and confidence and prediction intervals for the regression model:

```
# Create our data
  set.seed(1)
  x <- runif(50, -2, 2)
  set.seed(2)
  y <- x + rnorm(50)
# Create the model object
  mod <- lm(y ~ x)
# Plot the data and add a regression line
# using default plot() behavior
  plot(x, y)
  abline(mod)
# Plot the model object, going through a
# sequence of diagnostic plots. See ?plot.lm
  plot(mod)

# Create prediction values and confidence limits
# using a new dataframe of x values, noting the
# colnames need to match your model term names.
  newData <- data.frame(x = seq(min(x), max(x),
          by = (max(x) - min(x)) / 49))
  pred.lim <- predict(mod, newdata = newData,
          interval = "prediction")
  conf.lim <- predict(mod, newdata = newData,
          interval = "confidence")
# Function to color plot region
  color.pr <- function(color = "white")
  {
    usr <- par("usr")
    if (par("xlog"))
      usr[1:2] <- 10 ^ usr[1:2]
    if (par("ylog"))
      usr[3:4] <- 10 ^ usr[3:4]
    rect(usr[1], usr[3], usr[2], usr[4],
      col = color)
  }
# Color the plot background
  par(bg = "blue")
# Define margins to enable space for labels
  par(mar = c(5, 6, 5, 3) + 0.1)
# Create the plot. Do not plot the data points
# and axes to allow us to define them our way
  plot(x, y, xlab = "x vals", ylab = "y vals",
    type = "n", col.lab = "yellow", font.lab = 2,
    cex.lab = 1.5, axes = FALSE, cex.main = 2,
    main = "Linear Regression Plot",
    col.main = "yellow", xlim = c(-2.1, 2.1),
    ylim = range(y, pred.lim, na.rm = TRUE))
# Color the plot region white
  color.pr("white")
# Plot the data points
  points(x, y, pch = 21, bg = "yellow", cex=1.25)
# Draw the fitted regression line and the
# prediction and confidence intervals
  matlines(newData$x, pred.lim, lty = c(1, 4, 4),
    lwd = 2, col = c("black", "red", "red"))
  matlines(newData$x, conf.lim, lty = c(1, 3, 3),
    lwd = 2, col = c("black", "green4", "green4"))
# Draw the X and Y axes, repectively
  axis(1, at = -2:2, col = "white",
    col.axis = "white", lwd = 2)
  axis(2, at = pretty(range(y), 3), las = 1,
    col = "white", col.axis = "white", lwd = 2)
# Draw the legend
  legend(-2, max(pred.lim, na.rm = TRUE),
    legend = c("Fitted Line", "Confidence Bands",
      "Prediction Bands"),
    lty = c(1, 3, 4), lwd = 2,
    col = c("black", "green4", "red"),
    horiz = FALSE, cex = 0.9, bg = "gray95")
# Put a box around the plot
  box(lwd = 2)
```

Barplot with confidence intervals and additional annotation



`barplot()` can draw essentially three types of plots with either vertical or horizontal bars (using the argument `horiz = TRUE / FALSE`). The first is a series of individual bars where the `height` argument (which defines the bar values) is a simple vector. The second is a series of stacked multi-segment bars where `height` is a matrix *and* `beside = FALSE`. The third is a series of grouped bars where `height` is a matrix *and* `beside = TRUE`. In the second and third cases, each column of the matrix `height` represents either the values of the bar segments in each stacked bar, or the values of the individual bars in each bar group, respectively.

`barplot()` returns either a vector or a matrix (when `beside = TRUE`) of bar midpoints that can be assigned to a variable (ie. `mp <- barplot(...)`). You can use this information to locate bar midpoints for text and/or line placement. To locate the midpoint of bar groups, use `colMeans(mp)` to enable the placement of a bar group label.

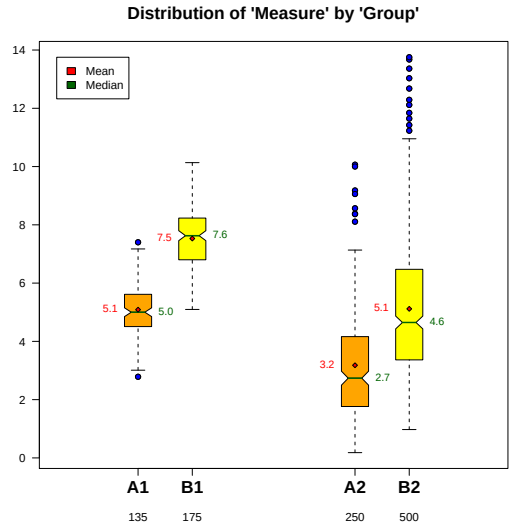
Here we will create a vertical barplot, with each of the three bars representing a proportion. We will add binomial confidence intervals and p values from `binom.test()` using a 'benchmark' value that will be plotted. We will label the y axis with percentages `(prop * 100)`, add bar values above the top of each bar and put sample sizes centered below each bar under the x axis.

```
# Create our data
  A <- data.frame(Event = c(rep("Yes", 6),
    rep("No", 120)), Group = "A")
  B <- data.frame(Event = c(rep("Yes", 11),
    rep("No", 398)), Group = "B")
  C <- data.frame(Event = c(rep("Yes", 8),
    rep("No", 276)), Group = "C")
  BarData <- rbind(A, B, C)
  attach(BarData)
# Create initial 'default' barplots
  barplot(table(Group))
  barplot(table(Group), horiz = TRUE)
  barplot(table(Event, Group))
  barplot(table(Event, Group), beside = TRUE)

# Let's get our summary data from the dataframe
  table.data <- table(Event, Group)
# Get sample sizes
  n <- as.vector(colSums(table.data))
# Get number of "Yes" events
  events <- as.vector(table.data["Yes", ])
# Proportion of "Yes" events
  prop.events <- events / n
# Group names from table dimnames
  Group.Names <- dimnames(table.data)$Group
# Define our benchmark value
  benchmark <- 0.045
# Get binomial confidence limits and p values
  stats <- mapply(binom.test, x = events, n = n,
    p = benchmark)
# ci[, 1] = lower and ci[, 2] = upper
  ci <- matrix(unlist(stats["conf.int", ]),
    ncol = 2, byrow = TRUE)
  p.val <- unlist(stats["p.value", ])
# Define Y axis range to include CI's and
# space for a legend in the upper LH corner
  YMax <- max(ci[, 2]) * 1.25
# Define margins to enable space for labels
  par(mar = c(5, 6, 5, 3) + 0.1)
# Do the barplot, saving bar midpoints in MidPts
  MidPts <- barplot(prop.events, space = 1,
    axes = FALSE,axisnames = FALSE,
    ylim = c(0, YMax))
# Define formatted Y axis labels using
# axTicks() and draw the Y Axis and label
  YLabels <- paste(formatC(axTicks(2) * 100,
      format = "f", digits = 1),
    "%", sep = "")
  YAxisLab <- "% Incidence (+/-95% CI)"
  axis(2, labels = YLabels, at = axTicks(2),
    las = 1)
 mtext(YAxisLab, side = 2, adj = 0.5,
    line = 4.5, cex = 1.1, font = 2)
# Draw the X axis using Group Names at bar
# midpoints
  axis(1, labels = Group.Names, at = MidPts,
    font = 2, cex.axis = 1.25)
# Draw Sample Sizes and p Values below Group
# Names
  mtext(n, side = 1, line = 2, at = MidPts,
    cex = 0.9)
  p.val.text <- paste("p = ",
    formatC(p.val, format = "f", digits = 4),
    sep = "")
  mtext(p.val.text, side = 1, line = 3,
    at = MidPts, cex = 0.9)
# Place formatted bar values above the left edge
# of each bar so that CI lines do not go through
# numbers. Left edge = MidPts - ('width' / 2)
  bar.vals <- paste(formatC(
      prop.events * 100, format = "f", digits=1),
   "%", sep = "")
  text(MidPts - 0.5, prop.events, cex = 0.9,
    labels = bar.vals, adj = c(0, -0.5), font=1)
```

```
# Draw confidence intervals, first drawing
# vertical line segments and then upper and
# lower horizontal boundary segments
  segments(MidPts, ci[, 1], MidPts, ci[, 2],
    lty = "solid", lwd = 2)
  segments(MidPts - 0.25, ci[, 1],
    MidPts + 0.25, ci[, 1], lty = "solid", lwd=2)
  segments(MidPts - 0.25, ci[, 2],
    MidPts + 0.25, ci[, 2], lty = "solid", lwd=2)
# Plot benchmark line
  abline(h = benchmark, lty = "dotdash",
    lwd = 2, col = "blue")
# Draw legend
  legend(1, YMax * 0.95, lty = "dotdash",
    legend = "Benchmark Value: 4.5%", lwd = 2,
    col = "blue", horiz = FALSE, cex = 0.9,
    bg = "gray95")
# Draw title and sub-title
  mtext("Incidence of Event By Group", side = 3,
    line = 3, cex = 1.5, font = 2)
  mtext(paste("Total N = ", sum(n), sep = ""),
    side = 3, line = 1, cex = 1, font = 2)
# Put box around plot
  box()
  detach(BarData)
```

Paired Boxplots with outliers colored and median / mean values labeled



J.W. Tukey's Box-Whisker plots (Tukey, 1977) are a quick and easy way to visually review and compare the distributions of continuous variables. For some descriptive information on the structure and interpretation of these plots including additional references, see `?boxplot.stats`.

Here we will generate continuous measures in four groups. We will generate default plots and then enhance the layout of the plot to visually group the data and to annotate it with key labels.

```
# Create our data
  set.seed(1)
  A1 <- data.frame(Group = "A1",
    Measure = rnorm(135, 5))
  set.seed(2)
  A2 <- data.frame(Group = "A2",
    Measure = rgamma(250, 3))
  set.seed(3)
  B1 <- data.frame(Group = "B1",
    Measure = rnorm(175, 7.5))
  set.seed(4)
  B2 <- data.frame(Group = "B2",
    Measure = rgamma(500, 5))
  BPData <- rbind(A1, A2, B1, B2)
  attach(BPData)
# Create default boxplots
  boxplot(Measure)
  boxplot(Measure, horizontal = TRUE)
  boxplot(Measure ~ Group)
# Adjust Group factor levels to put A1 / B1
# and A2 / B2 pairs together
  Group <- factor(Group,
    levels = c("A1", "B1", "A2", "B2"))
# Show default boxplot with re-grouping
  boxplot(Measure ~ Group)
# Define that boxplot midpoints to separate
# the pairs of plots
  at <- c(1.25, 1.75, 3.25, 3.75)
# Draw boxplot, returning boxplot stats in S
# which will contain summary data for each Group.
# See ?boxplot.stats
  S <- boxplot(Measure ~ Group, boxwex = 0.25,
    col = c("orange", "yellow"), notch = TRUE,
    at = at, axes = FALSE)
# Draw thicker green lines for median values
# When notch = TRUE, median width = boxwex / 2
  segments(at - 0.0625, S$stats[3, ],
           at + 0.0625, S$stats[3, ],
    lwd = 2, col = "darkgreen")
# Get Group means and plot them using a
# diamond plot symbol
  means <- by(Measure, Group, mean)
  points(at, means, pch = 23, cex = 0.75,
    bg = "red")
# Color outlier values using x,y positions from S
  points(at[S$group], S$out, pch = 21, bg="blue")
# Draw Y axis, rotating labels to horiz
  axis(2, las = 1)
# Draw X Axis Group Labels
  axis(1, at = at, labels = S$names,
    cex.axis = 1.5, font.axis = 2)
  mtext(S$n, side = 1, at = at, line = 3)
# Draw Mean values to the left edge of each
# boxplot
  text(at - 0.125, means, labels = formatC(
        means, format = "f", digits = 1),
    pos = 2, cex = 0.9, col = "red")
# Draw Median values to the right edge of
# each boxplot
  text(at + 0.125, S$stats[3, ],
    labels = formatC(S$stats[3, ], format = "f",
        digits = 1),
    pos = 4, cex = 0.9, col = "darkgreen")
# Draw a box around plot
  box()
# Add title and legend
  title("Distribution of 'Measure' by 'Group'",
```

```
  cex.main = 1.5)
legend(0.5, max(Measure),
  legend = c("Mean", "Median"),
  fill = c("red", "darkgreen"))
detach(BPData)
```

Additional Resources

For additional information on using R's plotting functionality, see: Venables, Smith and R Core (2003); Venables and Ripley (2002); Fox (2002); Dalgaard (2002). In addition, Uwe Ligges' recent R News article (Ligges, 2003) provides excellent insights into how best to utilize R's documentation and help resources.

If you are in need of expert guidance on creating analytic graphics, such as the pros and cons of using particular graphic formats and their impact on the interpretation of your data, two critically important references are "Visualizing Data" (Cleveland, 1993) and "The Elements of Graphing Data" (Cleveland, 1994).

Bibliography

Cleveland, W. S. (1993): *Visualizing Data*. Summit, NJ: Hobart Press. 6

Cleveland, W. S. (1994): *The Elements of Graphing Data*. Summit, NJ: Hobart Press, revised edition. 6

Dalgaard, P. (2002): *Introductory Statistics with R*. New York: Springer-Verlag. 6

Fox, J. (2002): *An R and S-PLUS Companion to Applied Regression*. Thousand Oaks: Sage. 6

Ligges, U. (2002): R Help Desk – Automation of Mathematical Annotation in Plots. *R News*, 2 (3), 32–34. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/. 2

Ligges, U. (2003): R Help Desk – Getting Help – R's Help Facilities and Manuals. *R News*, 3 (1), 26–28. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/. 6

Murrell, P. (2002): The grid Graphics Package. *R News*, 2 (2), 14–19. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/. 2

Murrell, P. and Ihaka, R. (2000): An Approach to Providing Mathematical Annotation in Plots. *Journal of Computational and Graphical Statistics*, 9 (3), 582–599. 2

Sarkar, D. (2002): Lattice: An Implementation of Trellis Graphics in R. *R News*, 2 (2), 19–23. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/. 2

Tukey, J. (1977): *Exploratory Data Analysis*. Reading, MA: Addison-Wesley. 5

Venables, W. N. and Ripley, B. D. (2002): *Modern Applied Statistics with S*. New York: Springer-Verlag, 4th edition. 2, 6

Venables, W. N., Smith, D. M. and the R Development Core Team (2003): *An Introduction to R*. URL http://CRAN.R-project.org/doc/manuals.html. 2, 6

Marc Schwartz
MedAnalytics, Inc., Minneapolis, Minnesota, USA
MSchwartz@MedAnalytics.com