Creating Plots Using R's Base Graphics

One attractive feature of R is the ability to display your data, or the results of your manipulation of that data, in a variety of graphical formats. There are three main packages in R for visualizing data—base graphics, lattice, and ggplot—the first of which is included in R's base installation and is available whenever R is running; the other two are separate packages that you must install and load before they are available for use. For this document our focus is on base graphics only.

Creating a Simple Scatterplot

One of the most common, and most important, visualizations in analytical chemistry is a scatterplot in which we plot something we measure (the dependent variable) on the y-axis as a function of something we control (the independent variable) on the x-axis. A good example of this is the calibration data used to establish Beer's law

$$A_{std} = \epsilon b \times C_{std}$$

where A_{std} is the absorbance for a standard solution of the analyte whose concentration is C_{std}; thus, absorbance is the dependent variable and is plotted on the y-axis and concentration is the independent variable and is plotted on the x-axis. Here is a simple example that uses the following data from the Chem 260 lab on the bleaching of dyes

concentration	absorbance
0.20	0.192
0.40	0.364
0.60	0.556
0.80	0.713
1.00	0.909

where the concentrations are given as the fraction of the standard with the greatest concentration of dye. First, we create vectors to hold the values for absorbance and for concentration

```
abs = c(0.192, 0.364, 0.556, 0.713, 0.909)
conc = c(0.2, 0.4, 0.6, 0.8, 1.0)
```

and then we use the plot() function to create the scatterplot shown in Figure 1.

```
plot(x = conc, y = abs)
```

Note: The help file gives the function's format as `plot(x, y, ...)`. We can simplify the code we type to `plot(conc, abs)` as the R defaults to assuming that the first object is x and that the second object is y.

Adding Feature to Plots

The set of ellipses (. . .) in the `plot(x, y, ...)` function allow us to pass additional arguments to customize our plot; here are some of these optional arguments:

type = "*option*". This argument is used to specify how the points are displayed; there are a number of options, but the most useful are "p" for points (this is the default), "l" for lines without points, "b" for both points and lines that do not touch the points, "o" for points and lines that pass through the points, "h" for histogram-like vertical lines, and "s" for stair steps; use "n" if you wish to supress the points.

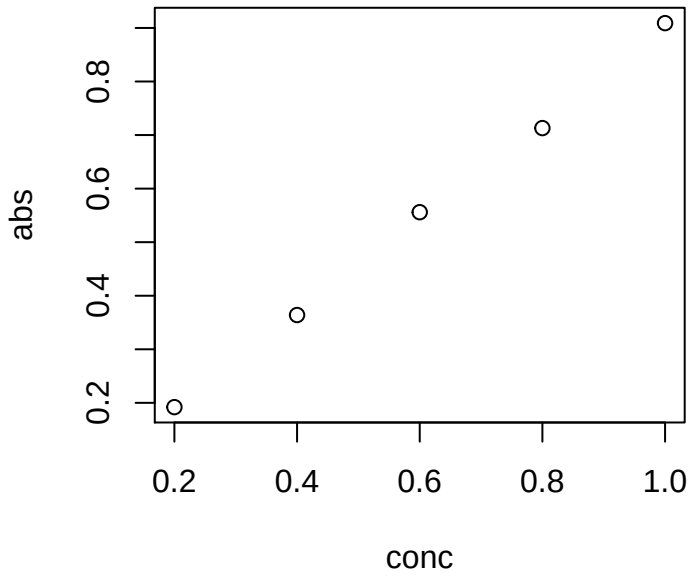

Figure 1: A simple, basic scatterplot of our calibration data.

pch = *number*. This argument is used to select the symbol used to plot the data, with the number assigned to each symbol shown in Figure 2. The default option is 1, or an open circle. Symbols 15–20 are filled using the color of the symbol's boundary, and symbols 21–25 can take a background ground that is different from the symbol's boundary. See later in this document for more details about setting colors.

lty = *number*. This argument is used to specify the type of line to draw; the options are 1 for a solid line (this is the default), 2 for a dashed line, 3 for a dotted line, 4 for a dot-dash line, 5 for a long-dash line, and 6 for a two-dash line.

lwd = *number*. This argument sets the width of the line. The default is 1 and any other entry simply scales the width relative to the default; thus lwd = 2 doubles the width and lwd = 0.5 cuts the width in half.

bty = "*option*". This argument specifies the type of box drawn around the plot; the options are "o" to draw all four sides (this is the default), "l" to draw on the left side and the bottom side only, "7" to draw on the top side and the left side only, "c" to draw all but the right side, "u" to draw all but the top side, "]" to draw all but the left side, and "n" to omit all four sides.

axes = *logical*. This argument indicates whether the axes are drawn (TRUE) or not drawn (FALSE); the default is TRUE.

xlim = c(*begin*, *end*). This argument allows you to define the limits for the x-axis, overriding the default limits set by the plot() command.

ylim = c(*begin*, *end*). This argument allows you to define the limits for the y-axis, overriding the default limits set by the plot() command.

xlab = "*text*". This argument allows you to specify the label placed along the x-axis, overriding the default label set by the plot() command.

ylab = "*text*". This argument allows you to specify the label placed along the y-axis, overriding the default label set by the plot() command.

main = "*text*". This argument allows you to specify the main title placed above the plot; overriding the default title set by the plot() command.

sub = "*text*". This argument allows you to specify the subtitle placed below the plot, overriding the default subtitle set by the plot() command.

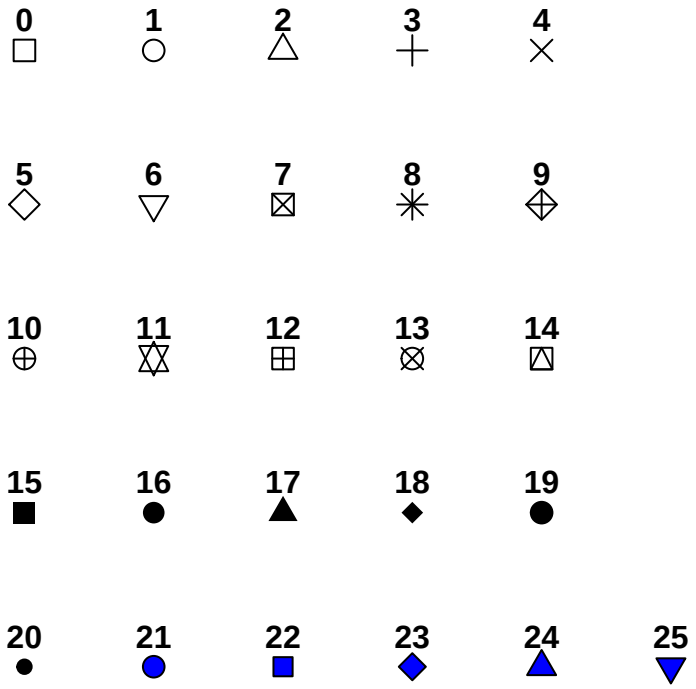

Figure 2: Plotting symbols and their associated pch values.

cex = ***number***. This argument controls the relative size of the symbols used to plot points. The default is 1 and any other entry simply scales the size relative to the default; thus cex = 2 doubles the size and cex = 0.5 cuts the size in half.

cex.axis = ***number***. This argument controls the relative size of the text used for the scale on both axes; see the entry above for **cex** for more details.

cex.lab = ***number***. This argument controls the relative size of the text used for the label on both axes; see the entry above for **cex** for more details.

cex.main = ***number***. This argument controls the relative size of the text used for the plot's main title; see the entry above for **cex** for more details.

cex.sub = ***number***. This argument controls the relative size of the text used for the plot's subtitle; see the entry above for **cex** for more details.

col = ***number*** **or "*string*"**. This argument controls the color of the symbols used to plot points. There are 657 colors available to you, for which the default is "black" or 24. You can see a list of colors (number and text string) by typing colors() in the console.

col.axis = ***number*** **or "*string*"**. This argument controls the color of the text used for the scale on both axes; see the entry above for **col** for more details.

col.lab = ***number*** **or "*string*"**. This argument controls the color of the text used for the label on both axes; see the entry above for **col** for more details.

col.main = ***number*** **or "*string*"**. This argument controls the color of the text used for the plot's main title; see the entry above for **col** for more details.

col.sub = ***number*** **or "*string*"**. This argument controls the color of the text used for the plot's subtitle; see the entry above for **col** for more details.

bg = ***number*** **or "*string*"**. This argument sets the background color for the plot symbols 21–25; see the entries above for **pch** and for **col** for more details.

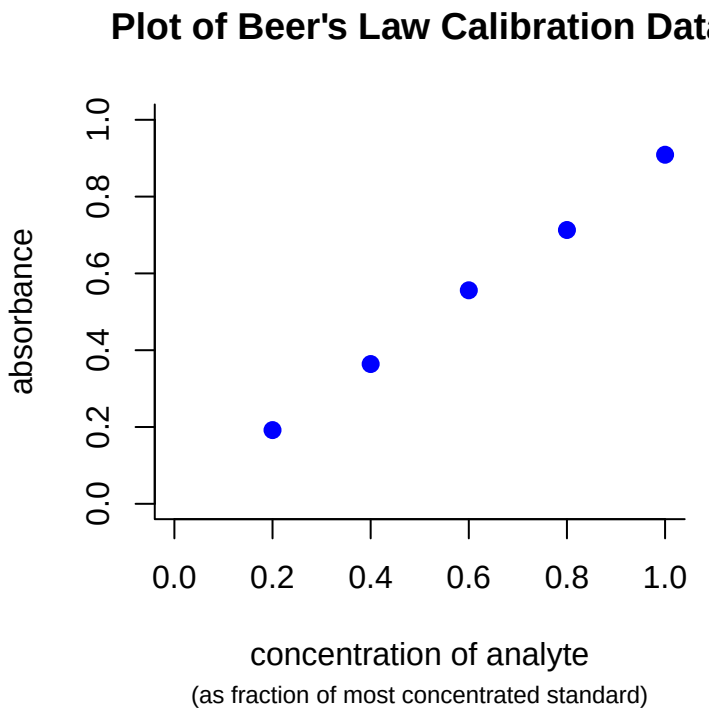

Figure 3: A much nicer plot of our calibration data.

Let's use some of these arguments to improve the plot in Figure 1 by adding some color to and adjusting the size of the symbols used to plot the data, by adding a title and some better labels for the two axes, and by adjusting the limits on both axes so that they show the lower limits for absorbance and for concentration ($A = 0$ when $C = 0$).

```
plot(conc, abs, pch = 16, col = "blue", cex = 1.25,
     xlim = c(0, 1), ylim = c(0, 1),
     xlab = "concentration of analyte",
     ylab = "absorbance", main = "Plot of Beer's Law Calibration Data",
     sub = "(as fraction of most concentrated standard)",
     cex.sub = 0.75, bty = "l")
```

Modifying an Existing Plot

We can modify an existing plot in a number of useful ways, such as adding a new set of data, adding a reference line, adding a legend, adding text, and adding a set of gridlines; here are some of the things we can do:

points(x, y, . . .). This command is identical to the plot() command, but overlays the new points on the current plot instead of first erasing that plot. Note: the points() command can not rescale the axes; thus, you must ensure that your original plot—created using the plot() command—has x-axis and y-axis limits that meet your needs.

abline(h = ***number*****, . . .)**. This command adds a horizontal line at $y =$ *number*; the line's color, type, and size are controlled using the optional arguments.

abline(v = ***number*****, . . .)**. This command adds a vertical line at $x =$ *number*; the line's color, type, and size are controlled using the optional arguments.

abline(b = ***number*****, a =** ***number*****, . . .)**. This command adds a diagonal line defined by a slope (b) and a

y-intercept (a); the line's color, type, and size are controlled using the optional arguments. As we will see later, this is a useful command for displaying the results of a linear regression.

legend(*location*, *legend*, . . .). This command adds a legend to the current plot. The location is specified in one of two ways:

- by giving the x and y coordinates for the legend's upper-left corner using x = *number* and y = *number*)
- by using location = "*keyword*" where the keyword is one of "topleft", "top", "topright", "right", "bottomright", "bottom", "bottomleft", or "left"; the optional argument include **inset = *number***, moves the legend in from the margin when using a keyword (it takes a value from 0 to 1 as a fraction of the plot's area; the default is 0)

The legend is added as a vector of character strings (one for each item in the legend), and any accompanying formatting, such as plot symbols, lines, or colors, are passed along as vectors of the same length; look carefully at the example at the end of this section to see how this command works.

text(*location*, *label*, . . .). This command adds the text given by "*label*" to the current plot. The location is specified by giving the x and y using x = *number* and y = *number*. By default, the text is centered at its location; to set the text so that it is left-justified (which is easier to work with), add the argument adj = c(0, NA).

grid(*col*, *lty*, *lwd*). This command adds a set of gridlines to the plot using the color, line type, and linewidth defined by "*col*", "*lty*", and "*lwd*", respectively.

In Figure 4 we use these commands to add a second data set, a legend, some additional text, and gridlines to the plot in Figure 3.

```
# first we have to replot Figure 3 so that we can add new feature to it
# this is not necessary if we are working in the console
plot(conc, abs, pch = 16, col = "blue", cex = 1.25,
     xlim = c(0, 1), ylim = c(0, 1),
     xlab = "concentration of analyte",
     ylab = "absorbance", main = "Plot of Beer's Law Calibration Data",
     sub = "(as fraction of most concentrated standard)",
     cex.sub = 0.75)

# create a second set of data
abs2 = abs * 0.75

# plot the second data set using points
points(conc, abs2, pch = 15, col = "red", cex = 1.25)

# add a legend
legend("topleft", inset = 0.05, legend = c("Group 1", "Group 2"),
       pch = c(19, 15), col = c("blue", "red"), bty = "n")

# add some additional text
text(x = 0.4, y = 0.1, label = "data collected spring 2015",
     adj = c(0, NA), cex = 0.75)

# add some gridlines
grid(col = "lightgray", lty = 3, lwd = 1)
```

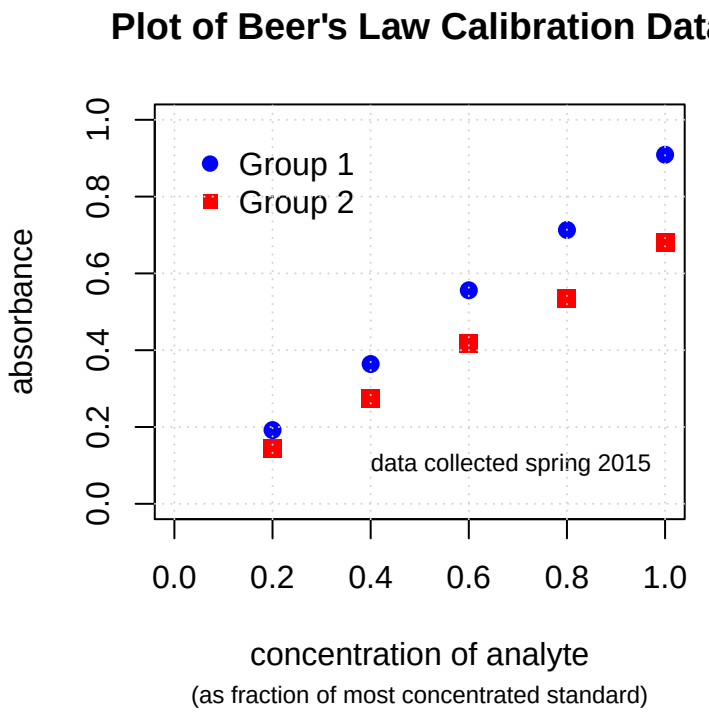

Figure 4: Our final plot showing two calibration curves.

Displaying Multiple Plots in a Single Plot

By default, each new use of the plot() command erases the previous plot before drawing the new plot. Although we can review old plots by clicking on the left-facing arrow in RStudio's **Plot Panel**, at times we need to display two or more plots at the same time. There are several ways to accomplish this, of which the simplest is to use the command `par(mfrow = c(nrows, ncols))`. The function `par()` is used to adjust R's graphical parameters. One of the possible arguments to is `mfrow()`, which creates a grid for plotting that has *nrows* rows and *ncols* columns. Each call to `plot()` places the plot in the next available space in the grid moving across the grid from left-to-right and from top-to-bottom. Once we reach the end of the grid, the next call to `plot()` erases all plots and begins again.

One problem with changing a graphical parameter is that it remains in effect until we reset it to its original value. A simple way to do this is to assign the command `par(mfrow = c(nrows, ncols))` to an object (*old.par* is a good choice)—this saves the parameter's old values in the object—and then use the command `par(old.par)` to reset the original parameters when we no longer need the changes. Figure 5 shows an example in which we plot a 2×2 grid of calibration data.

```
# create two additional data sets
abs3 = abs * 0.5
abs4 = abs * 0.25

# save the original parameters for par to old.par
old.par = par(mfrow = c(2, 2))

# create our plots
plot(conc, abs, xlim = c(0, 1), ylim = c(0, 1), main = "Group 1")
plot(conc, abs2, xlim = c(0, 1), ylim = c(0, 1), main = "Group 2")
plot(conc, abs3, xlim = c(0, 1), ylim = c(0, 1), main = "Group 3")
plot(conc, abs4, xlim = c(0, 1), ylim = c(0, 1), main = "Group 4")
```

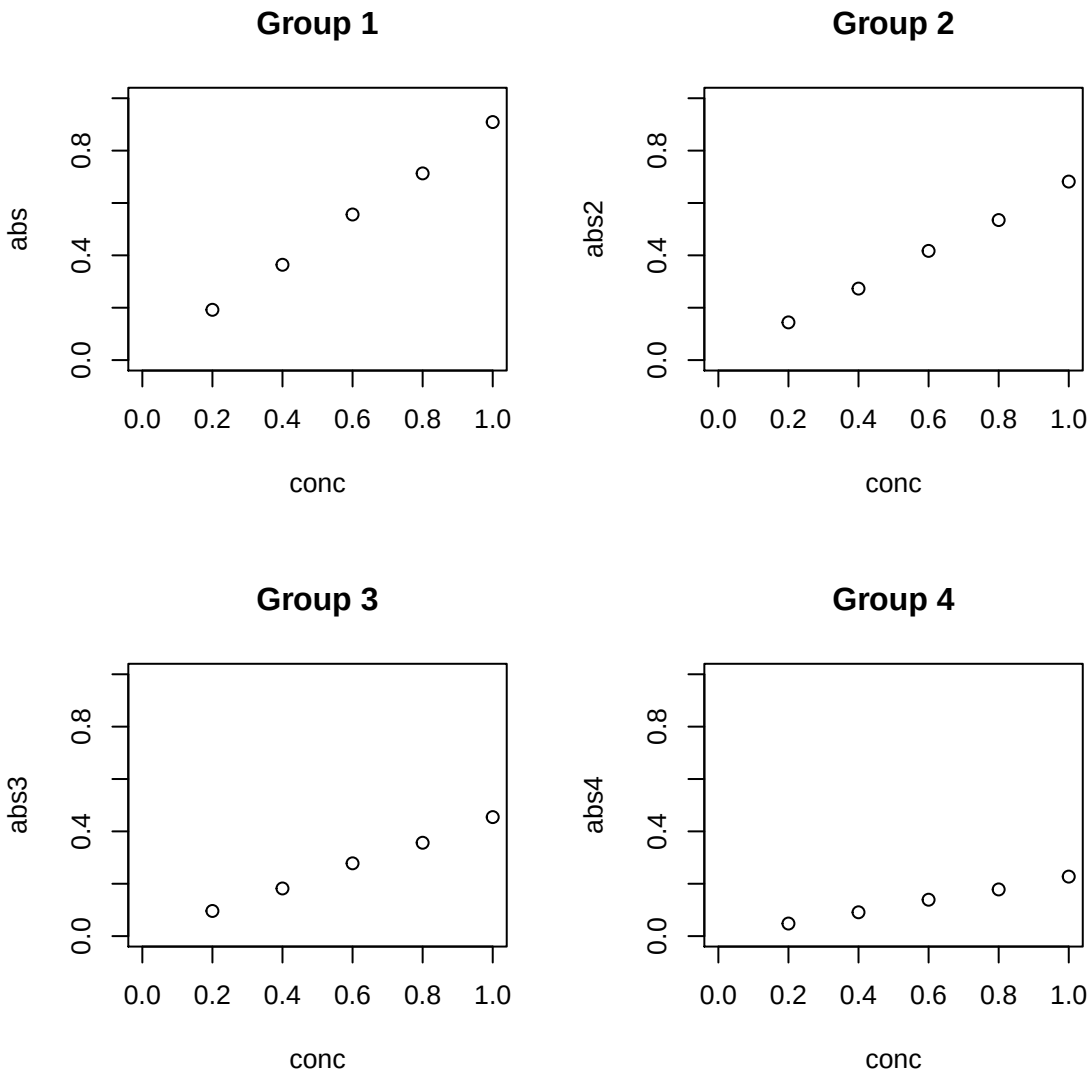

Figure 5: Using par(mfrow()) to create a grid for plots.

```
# restore the original parameters to par
par(old.par)
```

Inserting Plots Into Other Documents and Saving Plots

From RStudio's **Plots Panel** you can use the *Export* Menu to copy the panel's contents to your clipboard; after selecting "Copy to Clipboard" you have an opportunity to adjust the size of your plot to suit your needs.

To save the contents of the **Plots Panel**, choose "Save as Image..." or "Save as PDF..." from the *Export* Menu; when saving as an image you can choose from a variety of different formats.