Somewhat more advanced plotting in base R: Plotting confidence bands with transparency (and other tricks)

Ian Dworkin

Here is just some examples of making nicer and more useful plots all in base R. Most of this can be easily accomplished in `ggplot2` (often pretty easily). However, I often find there are times when it is easier to make the exact modifications using base plots.

Simulate some data and fit a linear model (simple linear regression)

```
x <- 1:50
y <- rnorm(n = length(x), mean = (2 + (0.8*x)), sd= 4)

model_1 <- lm(y ~ x)
```

Now we generate a sequence of "x" values for which to estimate y values (both fitted and CI's). This can be a sequence spanning the range of the data or, the actual data sorted from lowest to highest (see later example).

```
new <- data.frame(x = seq(min(x), max(x), 1))
```

Now we predict the new data using the estimates from `model_1`. using `interval = confidence` in `pred()` will give fitted values +95%CI on estimates. These are different from the prediction intervals (which can also be called).

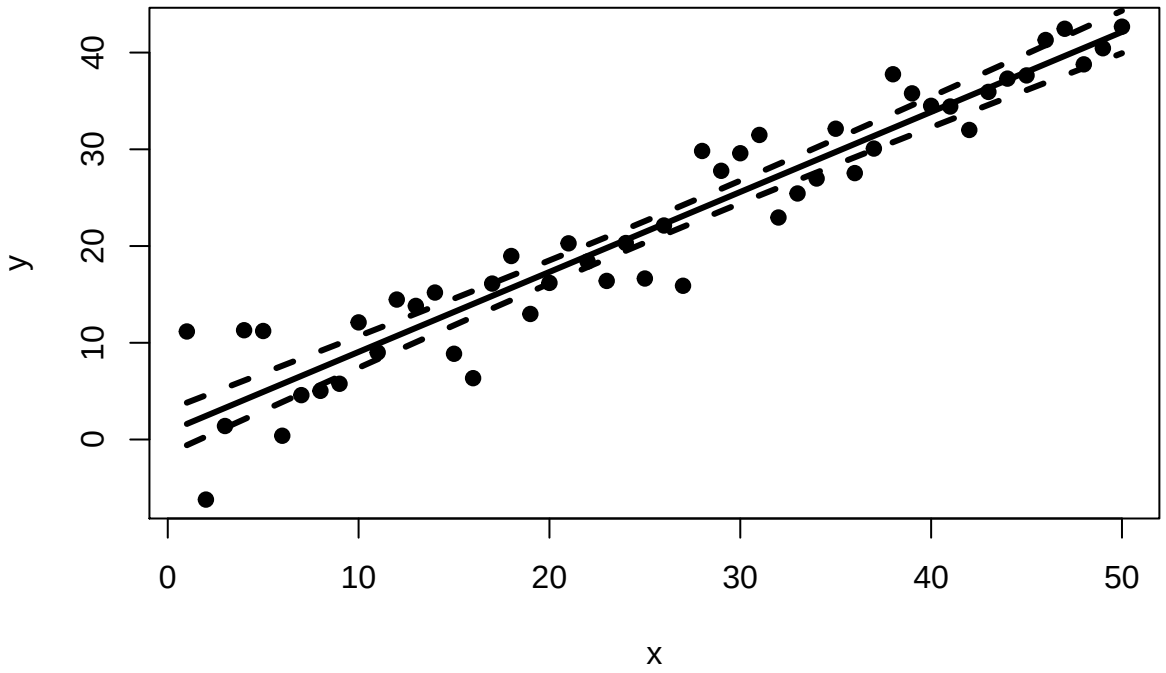
```
pred <- predict(model_1, newdata = new, interval = "confidence")
```

Now we can plot!

We will start by plotting the confidence bands around the estimated best fit line with (graphically) simple confidence bands with additional lines representing the upper and lower 95% CIs on the predicted values.

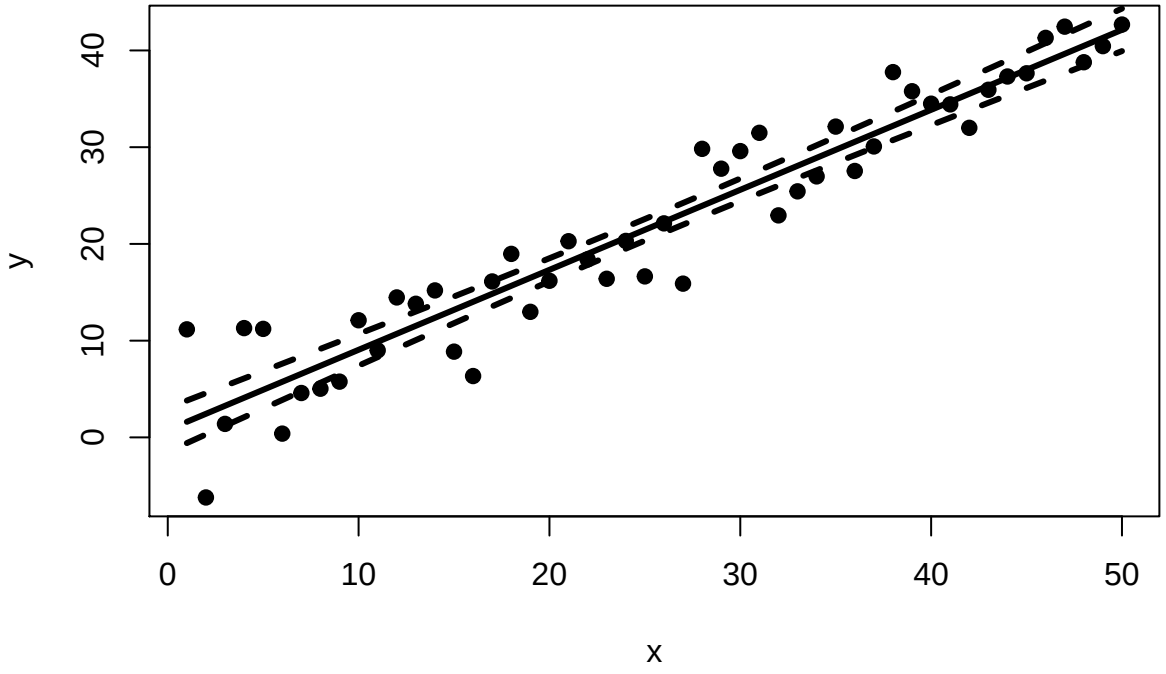
We could do this line by line:

```
plot(y ~ x, pch = 20, cex = 1.5)
lines(x = new[,1], y = pred[,1], lwd = 3) # this would be the same as abline(model.1)
lines(x = new[,1], y = pred[,3], lwd = 3, lty = 2)
lines(x = new[,1], y = pred[,2], lwd = 3, lty = 2)
```



However I prefer using the `matlines()` to do this.

```
plot(y ~ x, pch = 20, cex = 1.5)
matlines(x = new[,1], y = pred[,1:3], lwd=c(3,3,3), lty=c(1,2,2), col = 1)
```

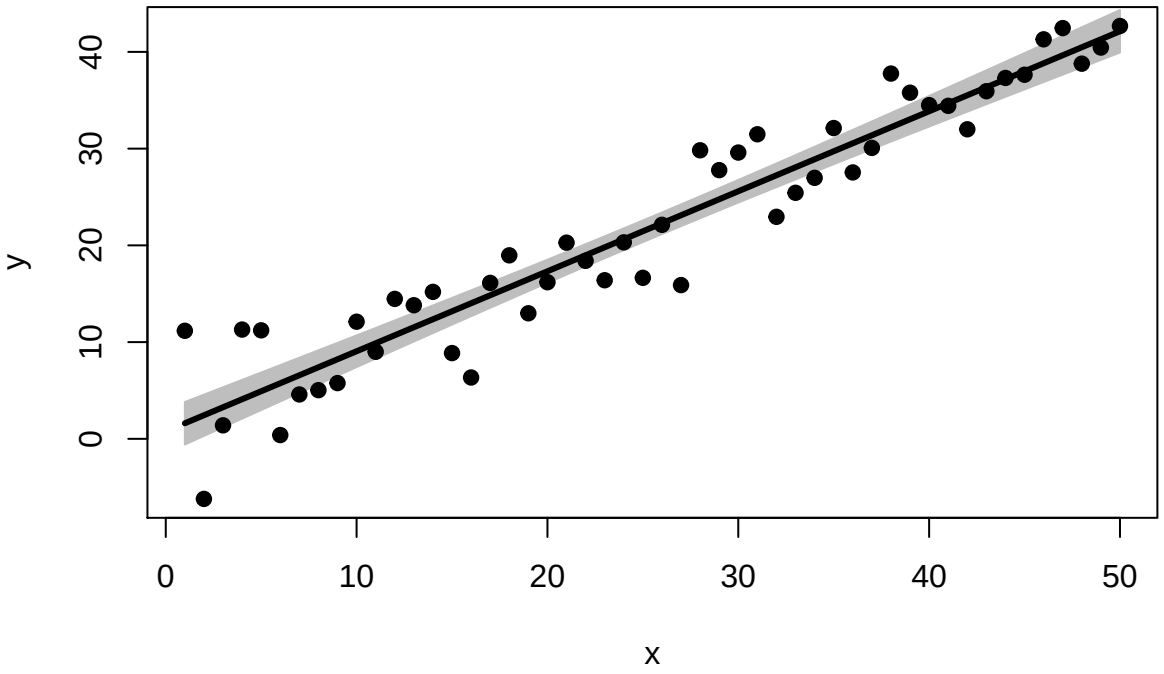


Plotting the confidence band as a shaded region.

Some prefer plotting this as shaded region. There are a few steps. First plotting an empty plot window (using `type = 'n'` in `plot()`). Then we will draw a polygon to define the shaded region, finally, we will add the observed data points back onto the plot.

We are going to make an arbitrary polygon where we stitch together the x values going forward and reverse, as well as lower CI and upper CI (I make upper go in reverse, but it is an arbitrary choice)

```
plot(y ~ x, pch= 20, type = "n")

polygon(x = c(1:nrow(new), nrow(new):1),
    y = c(pred[,2], rev(pred[,3])),
    col="grey", border = "grey")

lines(x = new[,1], y = pred[,1], lwd = 3)
# And the data
points(y ~ x, pch = 20, cex = 1.5)
```



Adding transparency

To understand how to add transparency to colours in R it is important to understand how the colours are encoded in RGB (Red Green and Blue "channels") (each 8 bit, allowing 256 distinct values per channel). We also need to know how to convert these to hexmode. In hexmode each channel is represented by a two letter code so colours can be encoded with 6 letters. First two for "red" channel, second two for "blue", etc. Finally the degree of transparency is added to the end of the colour value from hexmode.

So if we wanted to find the value for pure red with 50% opacity:

```
col2rgb("red")
```

```
##       [,1]
## red    255
## green    0
## blue     0
```

```
as.hexmode(255) # We will use fe for each channel in rgb
```

```
## [1] "ff"
```

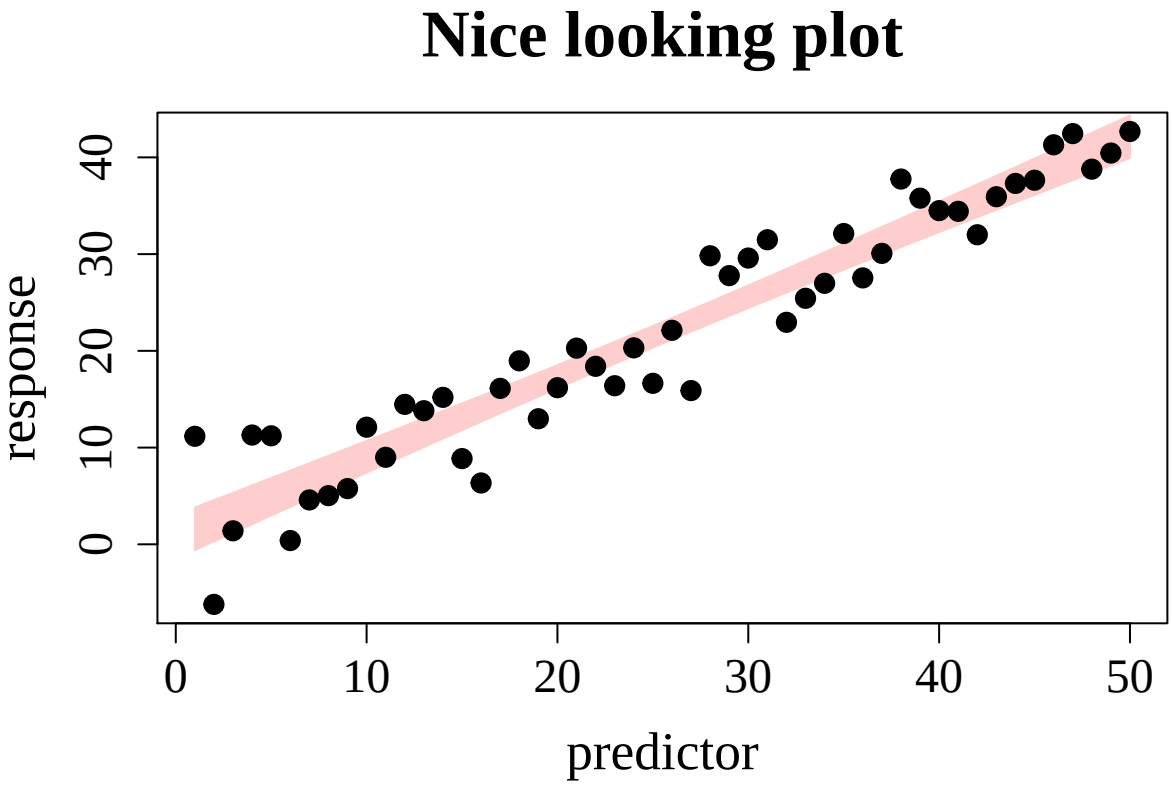
```r
as.hexmode(50) # For 50% opacity
```

```
## [1] "32"
```

So the final colour we want is `col = "#fe000032"`. The `#` lets it know its hexmode (I think!).

Anyways, we can now put this all together to make a nicer looking plot:

```r
plot(y ~ x, type = "n",
    xlab = "predictor", ylab = "response", main = "Nice looking plot",
    cex.lab = 1.7, cex.axis = 1.5, cex.main = 2.1,
    family = "serif", pch = 20)

polygon(x = c(1:nrow(new), nrow(new):1),
    y = c(pred[,2], rev(pred[,3])),
    col= "#fe000032", border = "#fe000032")

points(y ~ x, pch = 20, cex = 2)
```



You may notice I made some additional changes to improve the figure. using the various `cex.` arguments in the `plot()` allowed me to increase the size of labels, axes, etc. I also changed the font used for the text to *serif* which I thought looked nice here.

More complex example with some real data.

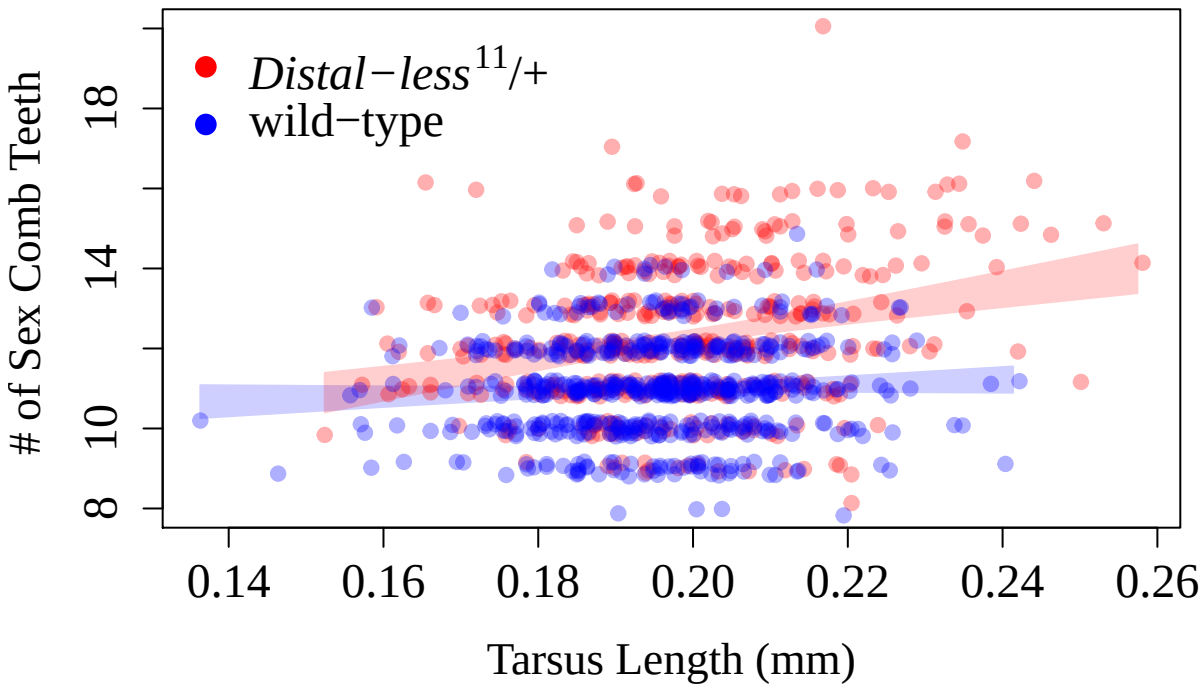
```r
dll.data = read.csv("http://datadryad.org/bitstream/handle/10255/dryad.8377/dll.csv", header=TRUE)

dll.data <- na.omit(dll.data)
```

We will fit linear models on two subsets of the data (a seperate regression for each), and generate estimated values (and their CIs) for each. There are other ways to just fit the model with the appropriate interaction term and using predict on these, but this is a demonstration for plotting purposes only.

```
model.dll.25 <- lm(SCT ~ tarsus, data=dll.data, subset=genotype=="Dll" & temp==25)

model.wt.25 <- lm(SCT ~ tarsus, data=dll.data, subset=genotype=="wt" & temp==25)

new.Dll <- with(dll.data[dll.data$genotype =="Dll" & dll.data$temp==25,],
    data.frame(tarsus = seq(range(tarsus)[1], range(tarsus)[2], by=0.005) ))

pred.Dll <-predict(model.dll.25, new.Dll, interval="confidence")

new.wt <- with(dll.data[dll.data$genotype =="wt" & dll.data$temp==25,],
    data.frame(tarsus = seq(range(tarsus)[1], range(tarsus)[2], by=0.005) ))
pred.wt <-predict(model.wt.25, new.wt, interval="confidence")
```

Now we can go ahead and make the plot. For a variety of reasons (that may be OS specific) I changed some overall graphical parameters (`par`) to improve the figure. Just so it is easy to return the settings in this session of `R` I am keeping the old par settings handy as well. In any case, I specifically expanding the left side of the palette, and making sure that the serif font is used for all text. You will also note the use of the `expression()` function so that I can use italics and superscripts in the text of the figure.

```
old_par <- par()

par(mar = c(5,5,4,1), family = "serif")

with(dll.data[dll.data$temp==25,],
    plot(SCT ~ tarsus, col=c("red", "blue")[dll.data$genotype], type = "n",
    ylab = " # of Sex Comb Teeth", xlab = "Tarsus Length (mm)",
    main = expression(paste(" SCT VS. tarsus length for ", italic(Drosophila), " genotypes" )),
    cex.lab = 1.4, cex.axis = 1.5, cex.main = 1.5,
    family = "serif", pch = 20))

# Confidence band for the wild-type flies
polygon(x = c(new.wt[,1], rev(new.wt[,1])), y = c(pred.wt[,2], rev(pred.wt[,3])),
    col="#0000ff32", border = "#0000ff32")

# lines(x = new.wt[,1], y = pred.wt[,1], lwd = 2, lty = 2) # add back if you want line for slope

# confidence band for the mutant flies
polygon(x = c(new.Dll[,1], rev(new.Dll[,1])), y = c(pred.Dll[,2], rev(pred.Dll[,3])),
    col="#ff000032", border = "#ff000032")
# lines(x = new.Dll[,1], y = pred.Dll[,1], lwd=2, lty = 2)

with(dll.data[dll.data$temp==25,],
    points(jitter(SCT) ~ tarsus, pch = 20, cex = 1.5,
    col=c("#ff000050", "#0000ff50")[dll.data$genotype] ))

legend(x = "topleft",
  legend = c(expression(paste(italic("Distal-less")^11, "/+")), "wild-type"),
  col= c("red", "blue"), pch = 16, cex = 1.5, bty = "n")
```



There are a few other arguments that may be useful t know. `bty = "n"` removes the border in the call to legend. Using the `^` in the `expression()` call is for a superscript for the allele number (11).

I also added some jitter to the y axis as number of sex comb teeth is discrete and it is a lot of data. This combined with the transparency (80% opacity) makes it easier to see when there are lots of data points.

If you want you can now return the par settings:

```
par <- old_par
```