Statistical computing MATH10093
Computer lab 1

Finn Lindgren

16/1/2019

Summary

In this lab session you'll gradually develop R code for estimating multiple linear models, and plotting the results. You will not hand in anything, but you should keep your code script file for later use.

1. Start `RStudio`.

2. Create a *project* for your course files:
   `File`→`New Project`→`New Directory`→`New Project`
   Next time you want to work on the course files, use `File`→`Open Project` instead. This will automatically open the script files you had open when you last worked on the course.
   NOTE: It's *highly* recommended to avoid spaces in all directory and file names! Including spaces may result in unexpected problems.

3. Go to the Learn page and open the `L01.pdf` lecture notes so you can refer to them easily during the lab. Read through the pages on formulas and functions before you continue with lab.

   (a) Create a new R script file:
   `File`→`New File`→`R Script`

   (b) Save the new R script file:
   `File`→`Save`, or
   Press `Ctrl+S`,
   and choose a descriptive filename (e.g. `lab_1_code.R`)
   During the lab, remember to save the script file regularly, to avoid losing any work.

   Some useful menu options, buttons, and keyboard shortcuts:

   - Configuring code syntax diagnostics:
     `Tools`→`Global Options`→`Code`→`Diagnostics`
     This allows you to turn on margin notes that automatically alert you to potential code problems.

- Run the current line of code (or a selected section of code) and step to the next line:
  Press the `Run` button in the top right corner of the editor window, or
  Press `Ctrl+Enter`
- Run all the code in the current script file:
  Press the `Source` button in the top right corner of the editor window, or
  Press `Ctrl+Shift+S`, or
  Press `Ctrl+Shift+Enter`
  Note: the first two options disable the normal automatic value printing; you'll only see the ouput that is explicitly fed into `print()`. The third option displays both the code and the results in the Console window.
- Optionally, follow the instructions from the lecture notes to install the `styler` package. This will enable options to reformat your script code to (usually) more readable code, by pressing the `Addins` button below the main program menu and choosing e.g. `Style active file`.

4. We will start with the simple linear model

   $$y_i = \beta_0 + z_i\beta_z + e_i,$$

   where y_i are observed values, z_i are observed values that we believe have a linear relationship with y_i, e_i are observation noise components, and β_0 and β_z are model parameters.

   First, generate synthetic data to use when developing the code for estimating models of this type. Enter the following code into your script file and run it with e.g. `Ctrl+Enter`:

   ```
   z <- rep(1:10, times = 10)
   data <- data.frame(z = z, y = 2 * z + rnorm(length(z), sd = 4))
   ```

   What does `rep()`, `length()`, and `rnorm()` do? Look at the help pages by running `?rep`, `?rnorm`, and `?length` in the interactive Console window, or searching for them in the `Help` pane.

   What values of β_0 and β_z did the synthetic data generation use?

   You can plot the data with

   ```
   plot(data$z, data$y)
   ```

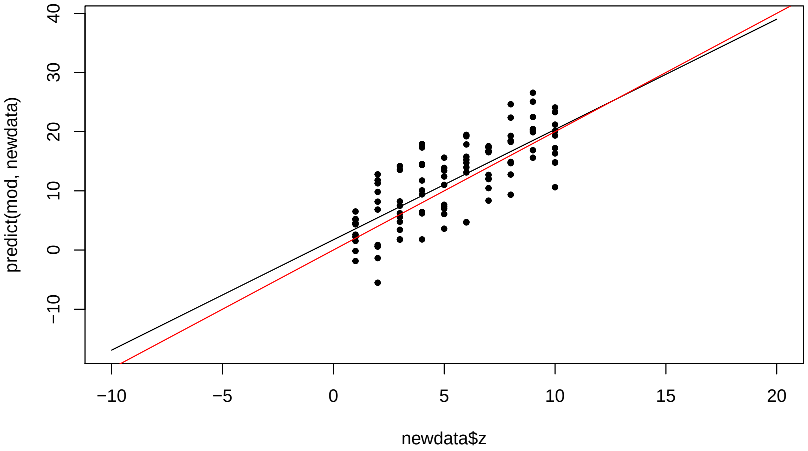
5. Use the `lm()` function to estimate the model and save the estimated model in a variable called `mod`.

6. Now we want to plot the linear predictor for the model as a function of z, between $z = -10$ and $z = 20$. First, create a new `data.frame`:

```r
newdata <- data.frame(z = -10:20)
```

Note that we don't have any corresponding y-values!

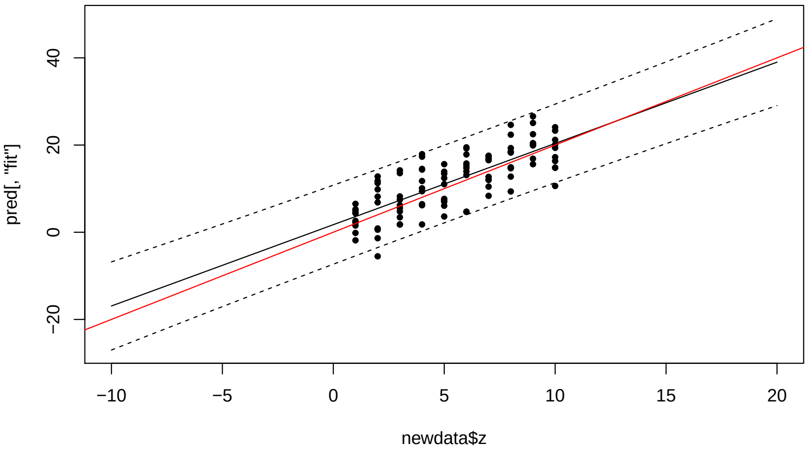
Use `predict()`, `plot()`, and `points()` to plot the linear predictor and the original data points in a single figure. Also add the true predictor with the help of `abline()` (use the `col` parameter to set a different color!). The result should look similar to this:



7. Now modify the plotting code to also show *prediction intervals*. Hint: use the `interval="prediction"` parameter for `predict()`, and note that the output is now a matrix:

```r
# Solution
pred <- predict(mod, newdata, interval = "prediction")
str(pred)
```

The `lines(..., lty=2)` function can be used add the prediction intervals to the figure, with a new line type to differentiate them from the linear predictor. You'll need to use the `ylim=range(pred)` parameter to expand the y-axis limits to contain the prediction interval curves. The result should look similar to this:



8. Let's say we want to redo the data analysis using different models (after all, we only know the true model because we generated synthetic data!). Instead of copying the plotting code for each new model, let's create a function instead! Start with the following skeleton code, and fill in the missing bits:

```
plot_predictions <- function(x, newdata, xname = "x") {
  pred <- ???
  ylim <- range(pred)
  plot(???, ylim = ylim)
  lines(???)
  lines(???)
}
```

   You should then be able to run the following code to regenerate your previous figure:

```
plot_predictions(mod, newdata, xname = "z")
points(data$z, data$y, pch=20)
abline(0, 2, col = 2)
```

9. You'll notice that the axis labels aren't very pretty. You can allow the caller of the function to control further `plot()` parameters such as `xlab` by using the `...` parameter feature:

```
plot_predictions <- function(x, newdata, xname = "x", ...) {
  ???
  plot(???, ylim = ylim, ...)
  ???
}
plot_predictions(mod, newdata, xname = "z", xlab = "z", ylab = "y")
```

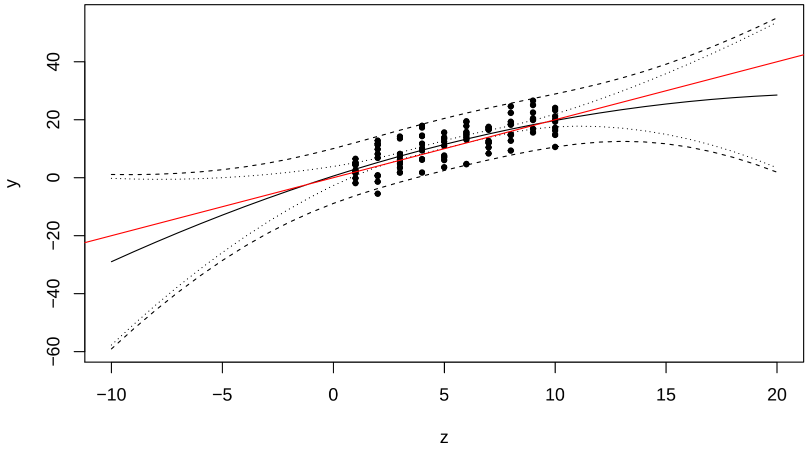
This passes any additional parameters through to the `plot()` function.

The lab solution document has a further generalised version of this plotting function, that also adds the confidence intervals for the predictor curve to the plot.

10. Use your new function to plot the predictions for the quadratic model

```
mod <- lm(y ~ 1 + z + I(z ^ 2), data)
```

The `I()` syntax means that you don't have to create a separate data variable equal to the square of the z-values; the `lm()` function will create it for you internally. The result should look similar to this:



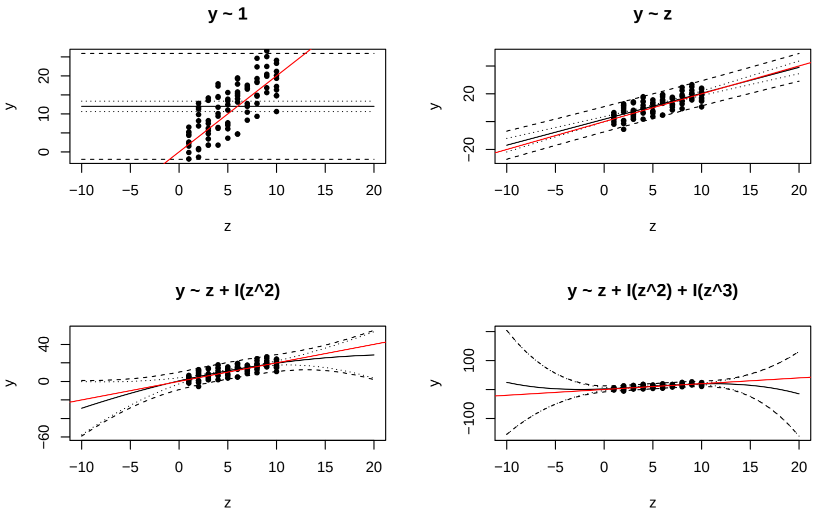
11. Finally, let's estimate and plot predictions for four polynomial models, by first creating a list of formulas:

```
formulas <- c(y ~ 1,
              y ~ z,
              y ~ z + I(z^2),
              y ~ z + I(z^2) + I(z^3))
```

Look at the lecture notes and use the `lapply()` technique to estimate all four models and store the result in a variable called `mods`.

12. You can now use something like the following code to plot all the results:

```
par(mfrow = c(2, 2))
for (k in seq_along(formulas)) {
  plot_prediction(mods[[k]], newdata, xname = "z", ylab = "y",
                  main=as.character(formulas[k]))
  points(data$z, data$y, pch=20)
  abline(0, 2, col = 2)
}
par(mfrow = c(1, 1))
```



Look at the help text for `seq_along()`!