Portfolio 1

Sam Bowyer

Reproducibility

In order that we may be confident in the conclusions of a scientific investigation, the research must be reproducible—that is, given the same raw data as the original investigation, we should be able to reach the same conclusions through the same analysis. Much literature exists on the importance and standards of reproducible research, one particularly useful resource is The Turing Way's Guide for Reproducible Research.

The Turing Way's guide mentions the differing definitions used throughout the literature for the terms 'reproducible' and 'replicable', before settling on the following:

- Reproducible: A result is reproducible when the same analysis steps performed on the same dataset consistently produces the same answer.
- Replicable: A result is replicable when the same analysis performed on different datasets produces qualitatively similar answers.

Usefully, definitions of 'robustness' (whether different analysis (e.g. with a different programming language) can produce the same results/conclusions using the same raw data) and 'generalisability' (whether different analysis on different raw data can produce "qualitatively similar or identical" results/conclusions) are also provided in the guide. Whilst these four qualities are clearly desirable for any and all research, they are unfortunately not always upheld. This is exemplified by the ongoing replication crisis in which questions are being raised about the proportion of published results which are unreproducible and/or unreplicable. This crisis is often discussed in regards to psychological and medical research (see, for instance, Ioannidis JPA (2005) Why most published research findings are false. PLoS Med 2(8): e124), however, it is applicable to all research subjects.

Some of the issues in the replication crisis hinge on analysis being performed on inadequate datasets (perhaps ones too small to obtain general results), but a large part of it is down to the actual analysis of data. Historically this could have been due to difficulties in sharing code that would work across different machines, but this is (for the most part) no longer a reasonable issue. Another common reason for unreproducible research is simply that analysis methodologies aren't adequately recorded, meaning that another researcher wouldn't even be able to recreate the analysis to check for the validity of the results. This is an issue that can easily be solved through following reproducible-research standards and would help improve the validity of an investigation's results.

It is important note that in some cases, research cannot be fully reproducible, for instance, in the case of medical data, sharing the dataset might entail privacy issues. However, the methodology of the analysis should still be clear and thorough, with the code used in the analysis still available, thus allowing others to potentially identify any bugs or issues.

Literate Programming

One way to encourage clear and readable explanations of data analysis methodology is in the use of *literate programming*, whereby executable code is presented alongside (i.e. in the same file as) text explanation of what the code is doing, thereby helping any analysis performed by the code more easily reproducible.

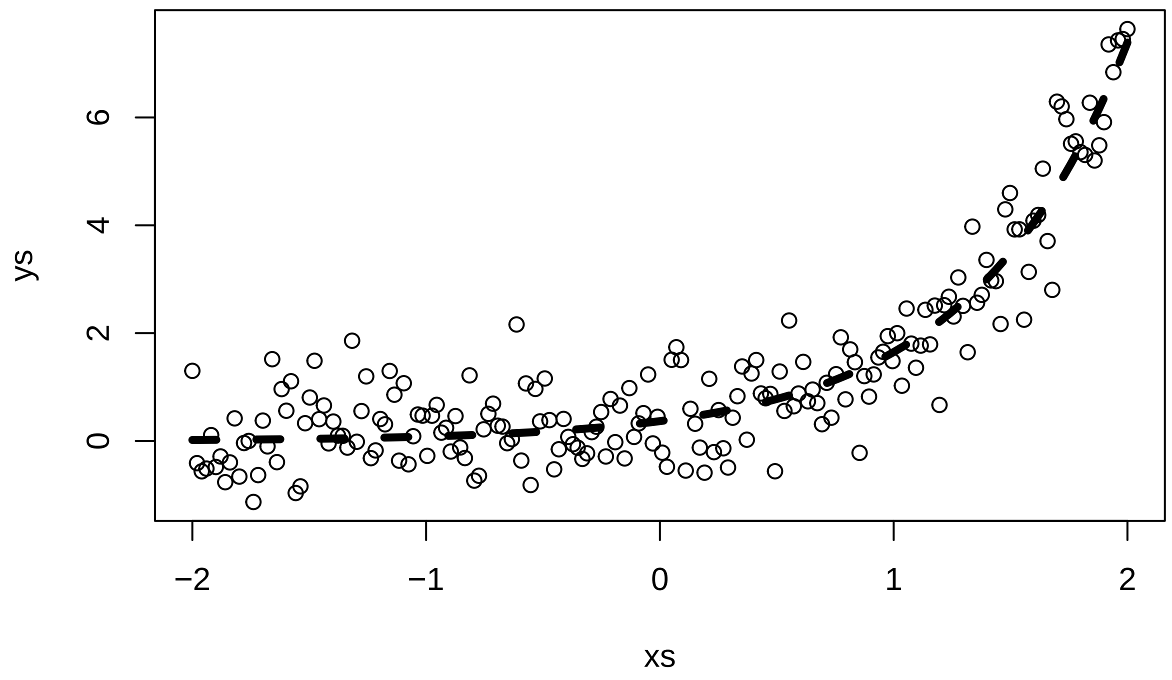
RMarkdown

RMarkdown is a common way to perform literate programming—it's what this document is written in. It allows text (plus images, links, LaTeX, etc.) to be interlaced with executable code and produces the output in various forms (RStudio by default allows .pdf, .html and .docx output).

As an example of using RMarkdown to perform reproducible analysis, we'll perform linear regression on a dataset containing entries x_i and y_i where $y_i = \exp(1.5x_i - 1) + \epsilon_i$, $\epsilon \sim \mathcal{N}(0, 0.64)$, $1 \leq i \leq 200$ with each x_i evenly spaced between -2 and 2 (a copy of this dataset can be found here).

First we can load and plot the dataset as follows (along with the data-generating function $y_i = \exp(1.5x_i - 1) + \epsilon_i$ too):

```r
data = read.csv("data.csv")
plot(data)
curve(exp(1.5*x -1), add=TRUE, lty=2, lwd=4)
```



```r
xs = data$xs
ys = data$ys
```

Defining a feature transform $\phi : \mathbb{R} \to \mathbb{R}^{b+1}$ as $\phi(x) = [1, x, x^2, ..., x^b]^T$ we can construct our design matrix

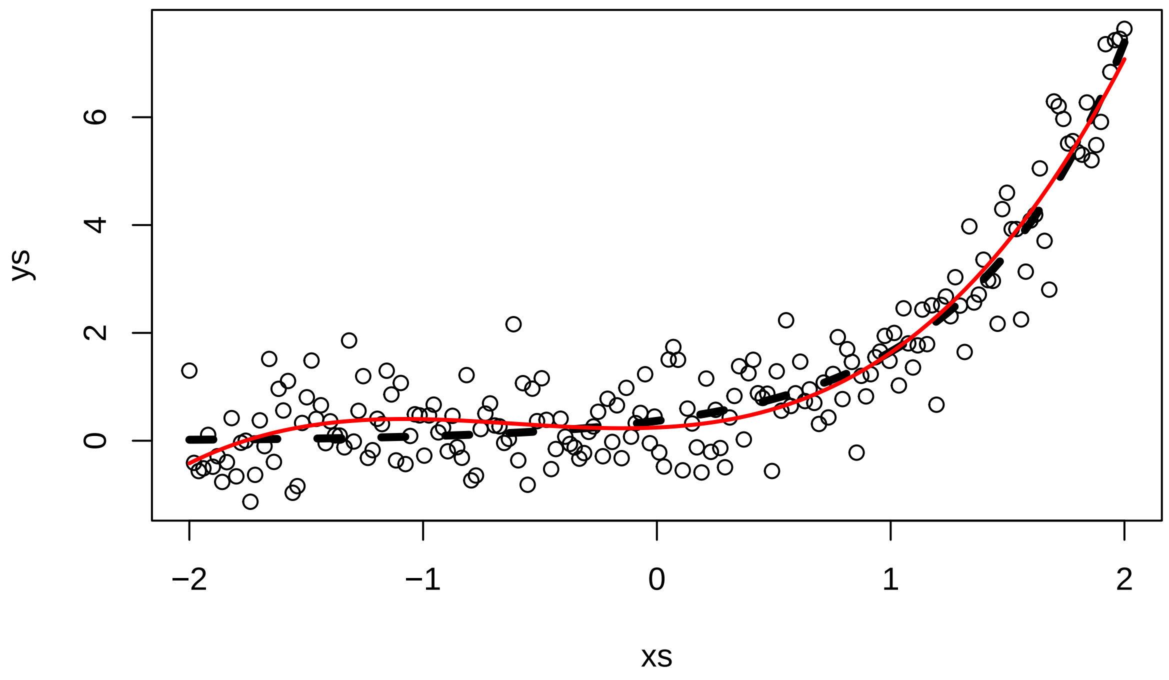
$$\phi = [\phi(x_1), \phi(x_2), ..., \phi(x_{200})],$$

which allows us to calculate the (unregularized) least-squares weights $w_{LS} = (\phi^T\phi)^{-1}\phi^T\mathbf{y}$ that minimizes the error $\sum_{i=1}^{200}(y_i - w^T\phi(x_i))^2$ (where $\mathbf{y} = [y_1, y_2, ..., y_{200}]^T$).

```r
# First construct the design matrix with a b=3 dimensional feature transform
phi <- function(x, b){
  phiX = matrix(NA, nrow=length(x), ncol=b+1)
  for(i in 1:length(x)){
    row = rep(x[i], b+1)
    for (j in 0:b){
      row[j+1] = row[j+1]**j
    }
    phiX[i, ] = row
  }
  return(phiX)
}

# Then calculate the least-squares weights w
findWeights <- function(phiX, ys){
  w <- solve(t(phiX) %*% phiX) %*% t(phiX) %*% ys
}

b=3
w = findWeights(phi(xs, b), ys)
```

Now we can plot this against the data (and the true data-generating function) as the red line below:

```r
# First let us define the predicted function as a function in R
predict <- function(x){
  return(phi(x, b) %*% w)
}

plot(data)
curve(exp(1.5*x -1), add=TRUE, lty=2, lwd=4)
curve(predict, add=TRUE, col="red", lwd=2)
```



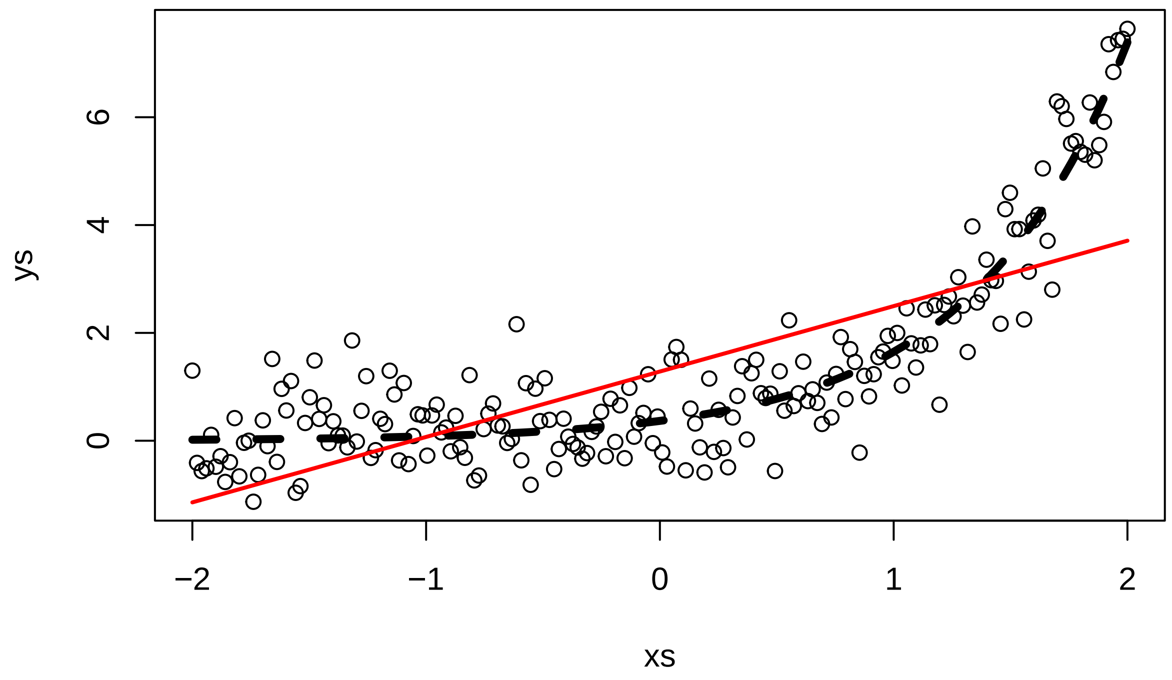
Wrapping this up in a single function, we can easily do this for different values of b.

```
leastSquares <- function(xs, ys, b){
  w = findWeights(phi(xs, b), ys)
  predict <- function(x){
    return(phi(x, b) %*% w)
  }
  print(w)

  plot(data)
  curve(exp(1.5*x -1), add=TRUE, lty=2, lwd=4)
  curve(predict, add=TRUE, col="red", lwd=2)
}
```

For instance, $b = 1$:

```
leastSquares(xs, ys, 1)
```

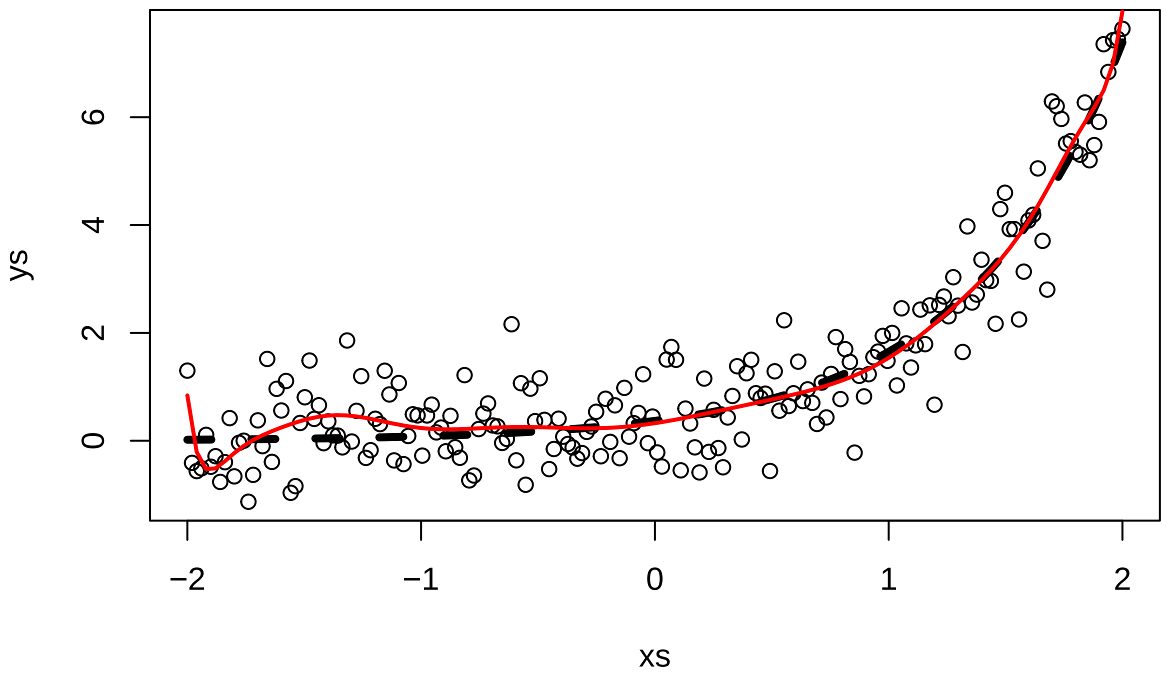
```
##          [,1]
## [1,] 1.283570
## [2,] 1.213225
```



Or $b = 16$:

```
leastSquares(xs, ys, 16)
```

```
##                [,1]
##  [1,]  0.325711960
##  [2,]  0.664087268
##  [3,]  0.965473755
##  [4,] -1.190678723
##  [5,] -0.922600432
##  [6,]  3.041744873
##  [7,] -0.619292103
##  [8,] -3.177863749
##  [9,]  2.844427145
## [10,]  1.770039854
## [11,] -2.555824871
## [12,] -0.535585412
## [13,]  1.033621674
## [14,]  0.084642614
## [15,] -0.199788461
## [16,] -0.005541524
## [17,]  0.015002766
```



As a final note, it is important to mention that literate programming is not always ideal—often low-level code libraries are better served by producing detailed documentation, such as through comments and using roxygen. However, for high-level data analysis, literate programming helps us to communicate the ideas behind our methodology in a way that aids reproducibility.