

MAST30025: Linear Statistical Models

Week 4 Lab

We model an individual's income at age 30 against the number of years of formal education with a linear model. The following data is collected:

Years of formal education (x)	Income (\$k) (y)
8	8
12	15
14	16
16	20
16	25
20	40

Where possible, solve the following questions in two ways: using matrix calculations as detailed in the lectures, and using the `lm` command in R.

1. Plot the data; is a linear model appropriate?

Solution: See Question 4. From the plot, it is not unreasonable to suppose a linear relationship between income and years of education.

2. Write down the linear model in matrix form.

Solution: $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, where

$$\mathbf{y} = \begin{bmatrix} 8 \\ 15 \\ 16 \\ 20 \\ 25 \\ 40 \end{bmatrix}, X = \begin{bmatrix} 1 & 8 \\ 1 & 12 \\ 1 & 14 \\ 1 & 16 \\ 1 & 16 \\ 1 & 20 \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}, \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{bmatrix}.$$

3. Find the normal equations for this model.

Solution:

```
> y <- c(8, 15, 16, 20, 25, 40)
> X <- cbind(rep(1, 6), c(8, 12, 14, 16, 16, 20))
> t(X)%*%X
```

```
      [,1] [,2]
[1,]      6  86
[2,]     86 1316
```

```
> t(X)%*%y
```

```
      [,1]
[1,]    124
[2,]   1988
```

The normal equations are

$$\begin{bmatrix} 6 & 86 \\ 86 & 1316 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 124 \\ 1988 \end{bmatrix}.$$

4. Solve the normal equations to obtain the least squares estimates of the parameters. Add the fitted regression line to your plot (using `curve` for example).

Solution:

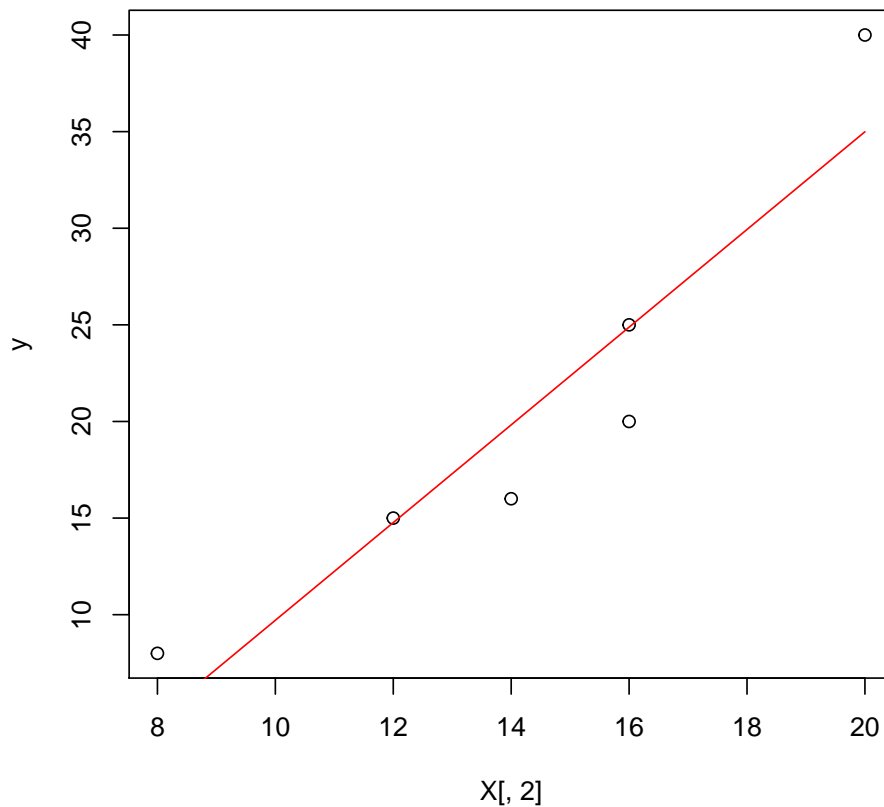
```

> n <- 6
> p <- 2
> (b <- solve(t(X) %*% X, t(X) %*% y))

      [,1]
[1,] -15.568
[2,]  2.528

> plot(X[,2], y)
> curve(b[1] + b[2]*x, add=TRUE, col="red")

```



Alternatively,

```

> income <- data.frame(income=y, education=X[,2])
> model <- lm(income ~ education, data=income)
> model$coefficients

(Intercept)  education
-15.568      2.528

```

5. This model is a simple linear regression model. Use the standard linear regression formulae,

$$b_1 = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2}, \quad b_0 = \bar{y} - b_1\bar{x},$$

to estimate the parameters again (where the bar indicates the mean). Check that you have the same answers as above.

Solution:

```
> (b1 <- (mean(X[,2]*y) - mean(X[,2])*mean(y))/(mean(X[,2]^2) - mean(X[,2])^2))

[1] 2.528

> (b0 <- mean(y) - b1*mean(X[,2]))

[1] -15.568
```

6. Calculate the sample variance s^2 .

Solution:

```
> e <- y - X %*% b
> SSRes <- sum(e^2)
> (s2 <- SSRes/(n-p))
```

```
[1] 18.692
```

Alternatively,

```
> deviance(model)/model$df.residual

[1] 18.692
```

7. Estimate the average income of a person who has had 18 years of formal education.

Solution:

```
> xst <- c(1, 18)
> t(xst) %*% b
```

```
      [,1]
[1,] 29.936
```

Alternatively,

```
> person <- data.frame(education=18)
> predict(model, person)
```

```
      1
29.936
```

8. Calculate the standardised residuals, leverage, and Cook's distance for the first observation. You may need the R functions `rstandard`, `influence`, and `cooks.distance`.

Check your numbers against the diagnostic plots produced by R.

Solution:

```
> H <- X %*% solve(t(X) %*% X) %*% t(X)
> # standardised residual
> (z1 <- e[1]/sqrt(s2*(1-H[1,1])))
```

```
[1] 1.303668
```

```
> # leverage
> H[1,1]
```

```
[1] 0.648
```

```
> # Cook's distance
> 1/p * z1^2 * H[1,1]/(1-H[1,1])
```

```
[1] 1.564359
```

Alternatively,

```
> rstandard(model)[1]
```

```
1
1.303668
```

```
> influence(model)$hat[1]
```

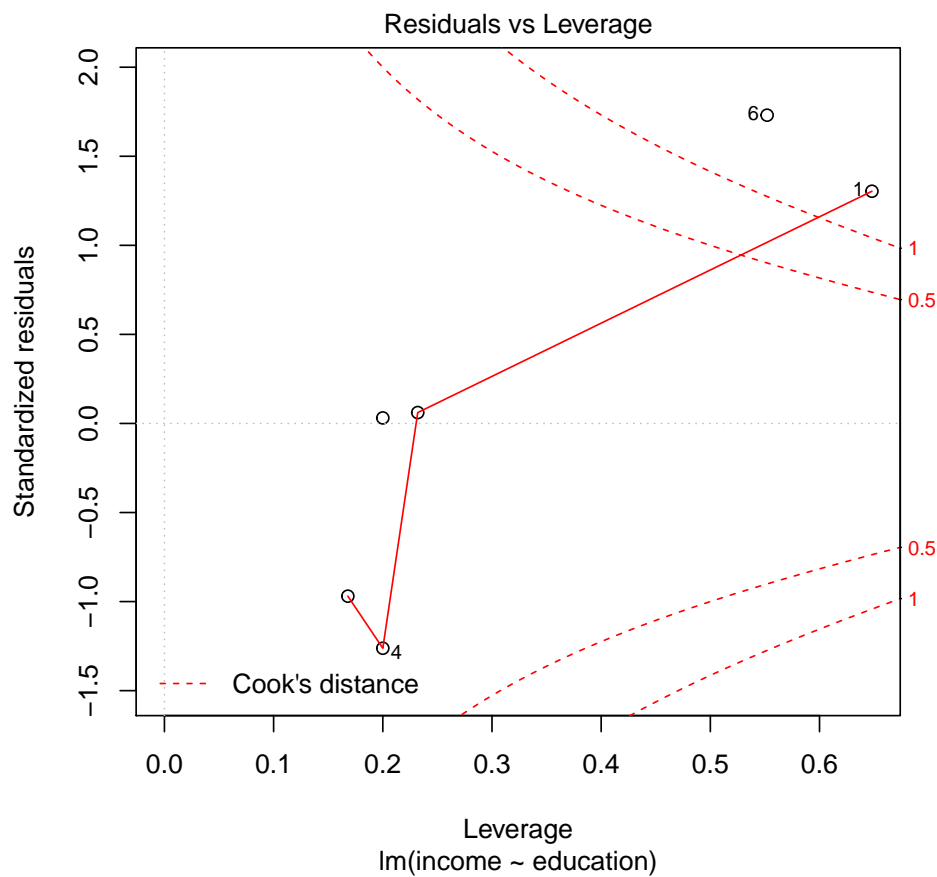
```
1
0.648
```

```
> cooks.distance(model)[1]
```

```
1
1.564359
```

Here is the relevant diagnostic plot:

```
> plot(model, which=5)
```



9. We know that the least squares estimator \mathbf{b} is an unbiased estimator for β . Show that $\mathbf{t}^T \mathbf{b}$ is an unbiased estimator for $\mathbf{t}^T \beta$, where \mathbf{t} is a vector of constants.

Solution: We know $E[\mathbf{b}] = \beta$. Therefore $E[\mathbf{t}^T \mathbf{b}] = \mathbf{t}^T E[\mathbf{b}] = \mathbf{t}^T \beta$.

R exercises

Read Sections 5.1–5.3 of spuRs, then attempt the exercises below.

1. The (Euclidean) length of a vector $v = (a_0, \dots, a_k)$ is the square root of the sum of squares of its coordinates, that is $\sqrt{a_0^2 + \dots + a_k^2}$. Write a function that returns the length of a vector.

Solution:

```
> euclid <- function(x) sqrt(sum(x^2))
```

2. Last week you wrote a program to calculate $h(x, n)$, the sum of a finite geometric series. Turn this program into a *function* that takes two arguments, x and n , and returns $h(x, n)$.

Make sure you deal with the case $x = 1$.

Solution:

```
> arithmetic_sum <- function(x, n) {  
+   # sum of x^k for k = 0, ..., n  
+   if (x == 1) {  
+     return(n + 1)  
+   } else {  
+     return((x^(n+1) - 1)/(x - 1))  
+   }  
+ }
```

3. In this question we simulate the rolling of a die. To do this we use the function `runif(1)`, which returns a ‘random’ number in the range (0,1). To get a random integer in the range $\{1, 2, 3, 4, 5, 6\}$, we use `ceiling(6*runif(1))`, or if you prefer, `sample(1:6,size=1)` will do the same job.

- (a) Suppose that you are playing the gambling game of the Chevalier de Méré. That is, you are betting that you get at least one six in four throws of a die. Write a program that simulates one round of this game and prints out whether you win or lose.

Check that your program can produce a different result each time you run it.

Solution:

```
> win <- FALSE  
> for (i in 1:4) {  
+   if (sample(1:6, size = 1) == 6) {  
+     win <- TRUE  
+   }  
+ }  
> if (win) {  
+   print("win")  
+ } else {  
+   print("lose")  
+ }  
[1] "win"
```

- (b) Turn the program that you wrote in part (a) into a function `sixes`, which returns `TRUE` if you obtain at least one six in n rolls of a fair die, and returns `FALSE` otherwise. That is, the argument is the number of rolls n , and the value returned is `TRUE` if you get at least one six and `FALSE` otherwise.

How would you give n the default value of 4?

Solution:

```
> sixes <- function(n = 4) {  
+   # plays the game of the Chevalier de Mere  
+   # returns TRUE if at least one six in n rolls  
+   # returns FALSE otherwise  
+   win <- FALSE
```

```

+   for (i in 1:n) {
+     if (sample(1:6, size = 1) == 6) {
+       return(TRUE)
+     }
+   }
+   return(FALSE)
+ }

```

Here is a vectorised version.

```

> sixes <- function(n = 4) {
+   sum(sample(1:6, size = n, replace = TRUE) == 6) > 0
+ }

```

- (c) Now write a program that uses your function `sixes` from part (b), to simulate N plays of the game (each time you bet that you get at least one six in n rolls of a fair die). Your program should then determine the proportion of times you win the bet. This proportion is an estimate of the *probability* of getting at least one six in n rolls of a fair die.

Run the program for $n = 4$ and $N = 100, 1000$, and 10000 , conducting several runs for each N value. How does the *variability* of your results depend on N ?

The probability of getting no 6's in n rolls of a fair die is $(5/6)^n$, so the probability of getting at least one is $1 - (5/6)^n$. Modify your program so that it calculates the theoretical probability as well as the simulation estimate and prints the difference between them. How does the *accuracy* of your results depend on N ?

Solution:

```

> p_estimate <- function(N, n = 4) {
+   # proportion of wins in N runs of sixes(n)
+   total_wins <- 0
+   for (i in 1:N) {
+     if (sixes(n)) total_wins <- total_wins + 1
+   }
+   return(total_wins/N)
+ }

> p_accuracy <- function(N, n = 4) {
+   # accuracy of p_estimate
+   total_wins <- 0
+   for (i in 1:N) {
+     if (sixes(n)) total_wins <- total_wins + 1
+   }
+   return(total_wins/N - 1 + (5/6)^n)
+ }

```

- (d) In part (c), instead of processing the simulated runs as we go, suppose we first store the results of every game in a file, then later postprocess the results. You should read `spuRs` Chapter 4 to see how to read and write text files.

Write a program to write the result of all N runs to a textfile `sixes_sim.txt`, with the result of each run on a separate line. For example, the first few lines of the textfile could look like

```

TRUE
FALSE
FALSE
TRUE
FALSE
.
.

```

Now write another program to read the textfile `sixes_sim.txt` and again determine the proportion of bets won.

This method of saving simulation results to a file is particularly important when each simulation takes a very long time (hours or days), in which case it is good to have a record of your results in case of a system crash.

Solution:

```
> sixes_sim <- function(N, n = 4) {  
+   # runs sixes(n) N times and saves the results in "sixes_sim.txt"  
+   cat(file="sixes_sim.txt") # deletes contents of file  
+   for (i in 1:N) {  
+     cat(file = "sixes_sim.txt", sixes(n), "\n", append = TRUE)  
+   }  
+ }  
> sixes_sim(100)  
> results <- scan("sixes_sim.txt", what = TRUE)  
> mean(results)  
[1] 0.43
```