MAST30025: Linear Statistical Models

Week 5 Lab

1. Consider the dataset from the Week 4 lab. As before, do the following using both matrix calculations and R's `lm` commands.

   (a) Calculate 95% confidence intervals for the model parameters.

   Solution:

```
> y <- c(8, 15, 16, 20, 25, 40)
> X <- cbind(rep(1, 6), c(8, 12, 14, 16, 16, 20))
> b <- solve(t(X) %*% X, t(X) %*% y)
> e <- y - X %*% b
> SSRes <- sum(e^2)
> n <- length(y)
> p <- dim(X)[2]
> s <- sqrt(SSRes/(n-p))
> C <- solve(t(X)%*%X)
> b[1] + c(-1,1)*qt(0.975,df=n-p)*s*sqrt(C[1,1])

[1] -35.042213   3.906213

> b[2] + c(-1,1)*qt(0.975,df=n-p)*s*sqrt(C[2,2])

[1] 1.213055 3.842945
```

   Alternatively,

```
> income <- data.frame(income=y,education=X[,2])
> model <- lm(income ~ education, data=income)
> confint(model, level=0.95)

                 2.5 %   97.5 %
(Intercept) -35.042213 3.906213
education     1.213055 3.842945
```

   (b) Calculate a 95% confidence interval for the average income of a person who has had 18 years of formal education.

   Solution:

```
> xst <- c(1,18)
> hw <- qt(0.975, df=n-p)*s*sqrt(t(xst) %*% C %*% xst)
> t(xst) %*% b + c(-1,1)*hw

[1] 23.0613 36.8107
```

   Alternatively,

```
> person <- data.frame(education=18)
> predict(model, person, interval="confidence", level=0.95)

     fit     lwr     upr
1 29.936 23.0613 36.8107
```

   (c) Calculate a 95% prediction interval for the income of a single person who has had 18 years of formal education.

   Solution:

```
> xst <- c(1,18)
> hw <- qt(0.975, df=n-p)*s*sqrt(1+t(xst) %*% C %*% xst)
> t(xst) %*% b + c(-1,1)*hw

[1] 16.10301 43.76899
```

   Alternatively,

```
> person <- data.frame(education=18)
> predict(model, person, interval="prediction", level=0.95)

       fit      lwr      upr
1 29.936 16.10301 43.76899
```

2. For simple linear regression, $y = \beta_0 + \beta_1 x + \varepsilon$, show that a $100(1-\alpha)\%$ confidence interval for the mean response when $x = x^*$ can be written as

$$b_0 + b_1 x^* \pm t_{\alpha/2} s \sqrt{\frac{1}{n} + \frac{(x^* - \bar{x})^2}{s_{xx}}}$$

where $s_{xx} = \sum_{i=1}^n x_i^2 - n\bar{x}^2$.

Similarly, show that a $100(1-\alpha)\%$ prediction interval for a new response when $x = x^*$ can be written as

$$b_0 + b_1 x^* \pm t_{\alpha/2} s \sqrt{1 + \frac{1}{n} + \frac{(x^* - \bar{x})^2}{s_{xx}}}.$$

Solution: For simple linear regression we have

$$\begin{aligned}
X^T X &= \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \\
&= \begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix} \\
(X^T X)^{-1} &= \frac{1}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i\right)^2} \begin{bmatrix} \sum_{i=1}^n x_i^2 & -\sum_{i=1}^n x_i \\ -\sum_{i=1}^n x_i & n \end{bmatrix}.
\end{aligned}$$

Let $\mathbf{t} = [1\ x^*]^T$. Then our CI for the mean response when $x = x^*$ is

$$\mathbf{t}^T \mathbf{b} \pm t_{\alpha/2} s \sqrt{\mathbf{t}^T (X^T X)^{-1} \mathbf{t}} = b_0 + b_1 x^* \pm t_{\alpha/2} s \sqrt{\mathbf{t}^T (X^T X)^{-1} \mathbf{t}}.$$

The term in the square root is

$$\begin{aligned}
\mathbf{t}^T (X^T X)^{-1} \mathbf{t} &= \frac{1}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i\right)^2} \begin{bmatrix} 1 & x^* \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n x_i^2 & -\sum_{i=1}^n x_i \\ -\sum_{i=1}^n x_i & n \end{bmatrix} \begin{bmatrix} 1 \\ x^* \end{bmatrix} \\
&= \frac{1}{n s_{xx}} \begin{bmatrix} \sum_{i=1}^n x_i^2 - x^* \sum_{i=1}^n x_i & -\sum_{i=1}^n x_i + n x^* \end{bmatrix} \begin{bmatrix} 1 \\ x^* \end{bmatrix} \\
&= \frac{1}{n s_{xx}} \left[\sum_{i=1}^n x_i^2 - 2x^* \sum_{i=1}^n x_i + n(x^*)^2\right] \\
&= \frac{1}{n s_{xx}} \left[\sum_{i=1}^n x_i^2 - n\bar{x}^2 + n\bar{x}^2 - 2n x^* \bar{x} + n(x^*)^2\right] \\
&= \frac{1}{n s_{xx}} [s_{xx} + n(x^* - \bar{x})^2] \\
&= \frac{1}{n} + \frac{(x^* - \bar{x})^2}{s_{xx}}
\end{aligned}$$

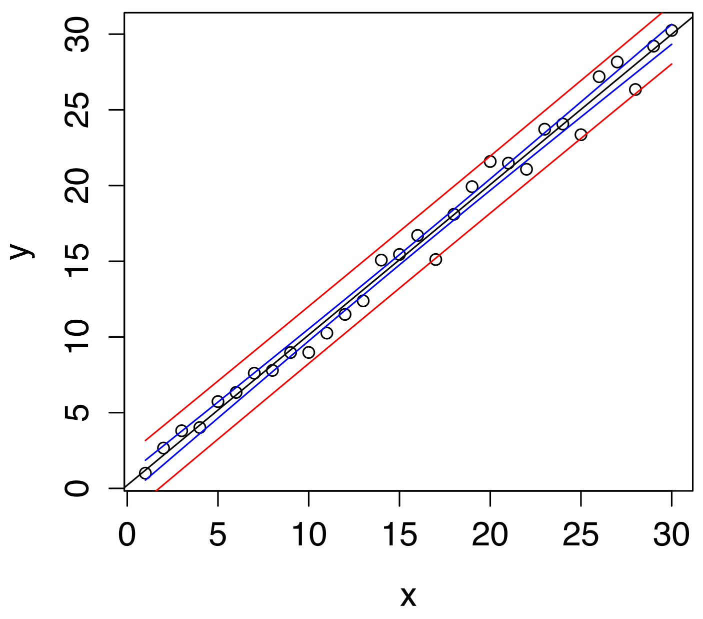
as required.

The proof for the prediction interval is similar.

3. We can generate some data for a simple linear regression as follows:

```
> n <- 30
> x <- 1:n
> y <- x + rnorm(n)
```

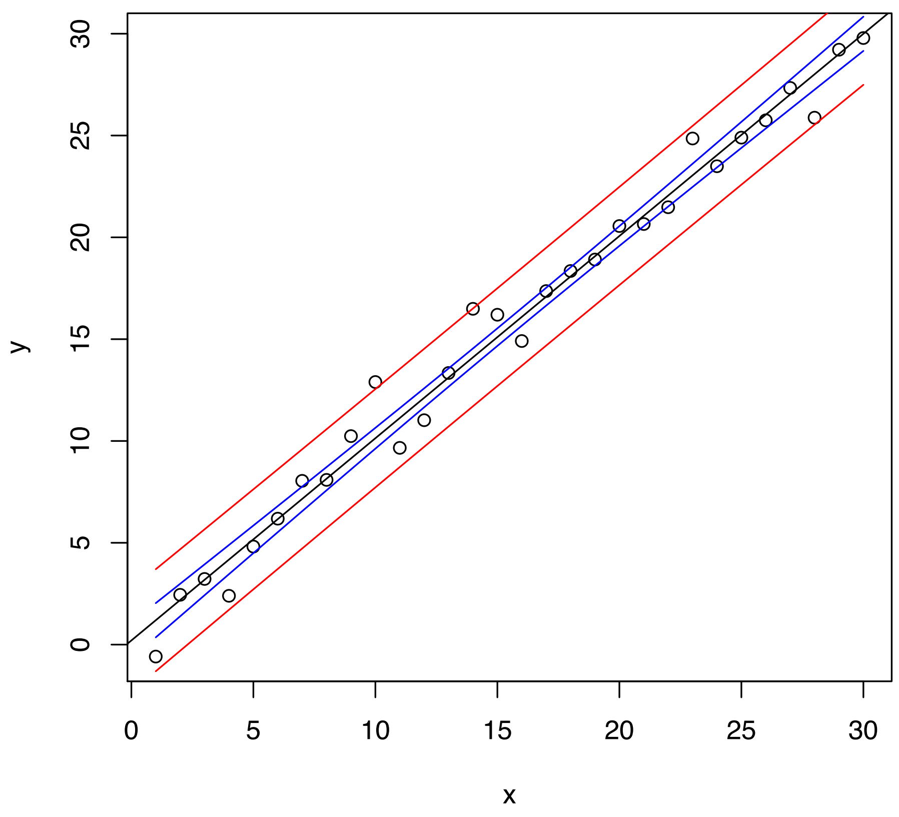
Construct 95% CI's for $\mathbb{E}[y]$ and 95% PI's for y, when $x = 1, 2, \ldots, n$. Join them up and plot them on a graph of y against x. Your plot should look something like this:



What proportion of the y's should you expect to lie beyond the outer lines?

Solution:

```
> fit <- lm(y ~ x)
> newdata <- data.frame(x=x)
> CI_u <- predict(fit,newdata,interval='confidence',level=0.95)[,3]
> CI_l <- predict(fit,newdata,interval='confidence',level=0.95)[,2]
> PI_u <- predict(fit,newdata,interval='prediction',level=0.95)[,3]
> PI_l <- predict(fit,newdata,interval='prediction',level=0.95)[,2]
> plot(x, y)
> abline(fit$coeff[1], fit$coeff[2])
> lines(x, CI_u, col="blue")
> lines(x, CI_l, col="blue")
> lines(x, PI_u, col="red")
> lines(x, PI_l, col="red")
```



We expect around 5% of the y's to lie beyond the PI's.

4. In generalised least squares we have $\mathbf{y} \sim MVN(X\boldsymbol{\beta}, V)$.

   Let R be the principal square root of V^{-1} (why does R exist?). Put $\mathbf{v} = R\mathbf{y}$, and show that $\mathbf{v} \sim MVN(Z\boldsymbol{\beta}, I)$ for some Z. From the usual least squares estimate of $\boldsymbol{\beta}$ using $\mathbf{v}$, obtain the generalised least squares estimate of $\boldsymbol{\beta}$ using $\mathbf{y}$.

   Solution: V is symmetric and positive definite (as it is a covariance matrix of full rank), thus we can write it as $V = P\Lambda P^T$ for P orthogonal and Λ diagonal with strictly positive diagonal entries. It follows that $V^{-1} = P\Lambda^{-1}P^T$ and $R = P\Lambda^{-1/2}P^T$. Observe that R is also symmetric and positive definite.

   Since $\mathbf{y} \sim MVN(X\boldsymbol{\beta}, V)$ we have $\mathbf{v} = R\mathbf{y} \sim MVN(RX\boldsymbol{\beta}, RVR^T) = MVN(RX\boldsymbol{\beta}, I)$, so $Z = RX$. Thus our LS estimate of $\boldsymbol{\beta}$ is

   $P\Lambda^{-1/2}P^T P\Lambda P^T P\Lambda^{-1/2}P^T = I$

   $$(Z^TZ)^{-1}Z^T\mathbf{v} = (X^TR^TRX)^{-1}X^TR^TR\mathbf{y} = (X^TV^{-1}X)^{-1}X^TV^{-1}\mathbf{y}$$

   as required.

5. In this exercise, we look at the dangers of overfitting. Generate some observations from a simple linear regression:

```
> set.seed(3)
> X <- cbind(rep(1,100), 1:100)
> beta <- c(0, 1)
> y <- X %*% beta + rnorm(100)
```

   Put aside some of the data for testing and some for fitting:

```
> Xfit <- X[1:50,]
> yfit <- y[1:50]
> Xtest <- X[51:100,]
> ytest <- y[51:100]
```

(a) Using only the fitting data, estimate $\boldsymbol{\beta}$ and hence the residual sum of squares. Also calculate the residual sum of squares for the test data, that is, $\sum_{i=51}^{100}(y_i - b_0 - b_1 x_i)^2$.

Solution:

```
> betafit <- solve(t(Xfit)%*%Xfit, t(Xfit)%*%yfit)
> ( SSfit <- sum((yfit - Xfit%*%betafit)^2) )

[1] 38.16794

> ( SStest <- sum((ytest - Xtest%*%betafit)^2) )

[1] 36.22107
```

Now add 10 extra predictor variables which we know have nothing to do with the response:

```
> X <- cbind(X, matrix(runif(1000), 100, 10))
> Xtest <- X[51:100,]
> Xfit <- X[1:50,]
```

Again using only the fitting data, fit the linear model $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, and show that the residual sum of squares has reduced (this has to happen). Then show that the residual sum of squares for the test data has gone up (this happens most of the time).

Explain what is going on.

Solution:

```
> ( betafit <- solve(t(Xfit)%*%Xfit, t(Xfit)%*%yfit) )

             [,1]
 [1,] -0.63651872
 [2,]  1.01113287
 [3,]  0.06578001
 [4,]  0.60104434
 [5,]  0.29716283
 [6,]  0.22470900
 [7,] -0.40165740
 [8,]  0.39186181
 [9,] -0.02118371
[10,] -0.34238715
[11,] -0.53698933
[12,]  0.12811422

> ( SSfit <- sum((yfit - Xfit%*%betafit)^2) )

[1] 34.27347

> ( SStest <- sum((ytest - Xtest%*%betafit)^2) )

[1] 39.07738
```

With the training data, we can match some of the noise (the $\boldsymbol{\varepsilon}$ term) using the new predictor variables. However, this is of no use when applied to the test data, as there is no systematic relationship between the noise and the new variables.

(b) Repeat the above, but this time add x^2, x^3 and x^4 terms:

```
> X <- cbind(X[, 1:2], (1:100)^2, (1:100)^3, (1:100)^4)
```

Solution:

```
> Xfit <- X[1:50,]
> Xtest <- X[51:100,]
> ( betafit <- solve(t(Xfit)%*%Xfit, t(Xfit)%*%yfit) )
```

```
             [,1]
[1,] -5.605561e-01
[2,]  1.124641e+00
[3,] -1.252249e-02
[4,]  4.589517e-04
[5,] -5.217886e-06
```

```
> ( SSfit <- sum((yfit - Xfit%*%betafit)^2) )
```

```
[1] 34.94215
```

```
> ( SStest <- sum((ytest - Xtest%*%betafit)^2) )
```

```
[1] 270310.6
```

Here the fit for the test data is absolutely horrendous. The problem is that the term $\beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4$ is small for the training data, where it is fitting the noise term ε, but for the test data this term becomes very large, resulting in a very poor fit. So overfitting is generally a bad thing, but overfitting with polynomials can be a very bad thing, in particular when you try and apply the fit beyond the range of the fitting data.

Programming questions

1. What will be the output of the following code? Try to answer this without typing it up.

```
fb <- function(n) {
    if (n == 1 || n == 2) {
        return(1)
    } else {
        return(fb(n - 1) + fb(n - 2))
    }
}
fb(8)
```

Solution: `fb(n)` returns the n-th Fibonacci number, so `fb(8)` returns 21.

2. Suppose you needed all 2^n binary sequences of length n. One way of generating them is with nested for loops. For example, the following code generates a matrix `binseq`, where each row is a different binary sequence of length three.

```
> binseq <- matrix(nrow = 8, ncol = 3)
> r <- 0 # current row of binseq
> for (i in 0:1) {
+   for (j in 0:1) {
+     for (k in 0:1) {
+       r <- r + 1
+       binseq[r,] <- c(i, j, k)
+     }
+   }
+ }
> binseq
```

```
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    1
[3,]    0    1    0
[4,]    0    1    1
[5,]    1    0    0
[6,]    1    0    1
[7,]    1    1    0
[8,]    1    1    1
```

Clearly this approach will get a little tedious for large n. An alternative is to use recursion. Suppose that A is a matrix of size $2^n \times n$, where each row is a different binary sequence of length n. Then the following matrix contains all binary sequences of length $n+1$:

$$\left(\begin{array}{c|c} \mathbf{0} & A \\ \hline \mathbf{1} & A \end{array}\right).$$

Here $\mathbf{0}$ is a vector of zeros and $\mathbf{1}$ is a vector of ones.

Use this idea to write a recursive function `binseq`, which takes as input an integer n and returns a matrix containing all binary sequences of length n, as rows of the matrix. You should find the functions `cbind` and `rbind` particularly useful.

Solution:

```
> binseq <- function(n) {
+   # all binary sequences of length n, where n is a +ve integer
+   if (n == 1) {
+     A <- matrix(0:1, nrow=2, ncol=1)
+     return(A)
+   } else {
+     A <- binseq(n-1)
+     B <- rbind(cbind(0, A), cbind(1, A))
+     return(B)
+   }
+ }
> binseq(3)
```

```
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    1
[3,]    0    1    0
[4,]    0    1    1
[5,]    1    0    0
[6,]    1    0    1
[7,]    1    1    0
[8,]    1    1    1
```

3. Let $A = (a_{i,j})_{i,j=1}^n$ be a square matrix, and denote by $A_{(-i,-j)}$ the matrix with row i and column j removed. If A is a 1×1 matrix then $det(A)$, the determinant of A, is just $a_{1,1}$. For $n \times n$ matrices we have, for any i,

$$det(A) = \sum_{j=1}^{n} (-1)^{i+j}\, a_{i,j}\, det(A_{(-i,-j)}).$$

Use this to write a *recursive* function to calculate $det(A)$.

Solution:

```
> my_det <- function(X) {
+   if (length(X) == 1) {
+     return(X)
+   } else {
+     S <- 0
+     for (i in 1:dim(X)[1]) {
+       S <- S + X[1, i]*(-1)^(1+i)*my_det(X[-1, -i])
+     }
+     return(S)
+   }
+ }
> X <- matrix(runif(25), nrow=5, ncol=5)
> my_det(X)
```

```
[1] 0.07874474

> det(X) # R's built in version

[1] 0.07874474
```