MAST30025 Lab 5

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

#Part a: Calculate 95% confidence intervals for the model parameters.

y = c(8,15,16,20,25,40)  
X = matrix(c(rep(1,6),8,12,14,16,16,20),6,2)

b = solve(t(X)%\*%X, t(X)%\*%y)  
df = 6 - 2  
e = y - X%\*%b  
s = sqrt(sum(e^2)/df)

#Confidence interval for b0!

c00 = solve(t(X)%\*%X)[1,1]  
alpha = 0.05  
ta = qt(1-alpha/2, df=df)  
c(b[1]-ta\*s\*sqrt(c00),b[1]+ta\*s\*sqrt(c00))

## [1] -35.042213 3.906213

#Confidence interval for b1!

c11 = solve(t(X)%\*%X)[2,2]  
alpha = 0.05  
ta = qt(1-alpha/2, df=df)  
c(b[2]-ta\*s\*sqrt(c11),b[2]+ta\*s\*sqrt(c11))

## [1] 1.213055 3.842945

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

xst = c(1,18)  
xst%\*%b

## [,1]  
## [1,] 29.936

ta = qt(0.975, df = 6-2)

xst%\*%b - ta\*s\*sqrt(t(xst)%\*%solve(t(X)%\*%X)%\*%xst) #Lower bound

## [,1]  
## [1,] 23.0613

xst%\*%b + ta\*s\*sqrt(t(xst)%\*%solve(t(X)%\*%X)%\*%xst) #Upper bound

## [,1]  
## [1,] 36.8107

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

xst%\*%b - ta\*s\*sqrt(1+t(xst)%\*%solve(t(X)%\*%X)%\*%xst) #Lower bound

## [,1]  
## [1,] 16.10301

xst%\*%b + ta\*s\*sqrt(1+t(xst)%\*%solve(t(X)%\*%X)%\*%xst) #Upper bound

## [,1]  
## [1,] 43.76899

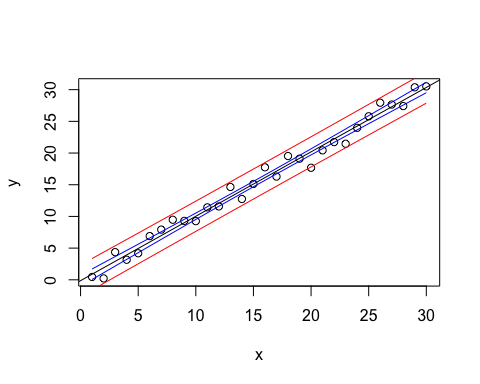
#Question 3. We can generate some data for a simple linear regression as follows:

n = 30  
x = 1:n  
y = x + rnorm(n)

MAIN TASK #Construct 95% CI’s for Ey and 95% PI’s for y, when x = 1, 2, . . . , n. Join them up and plot them on a graph of y against x.

MAIN QUESTION! #What proportion of the y’s should you expect to lie beyond the outer lines? # Tough without actual examples in the lectures

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")

 # Expect about 5% of the y’s to lie beyond the PI’s.

#Question 5: #Overfitting exercise. #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]

1. Using only the fitting data, estimate β and hence the residual sum of squares. Also calculate the residual sum of squares for the test data, that is, 􏰆100 i=51 Solution:

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

## [1] 38.16794

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

## [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,]  
Xfit

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] 1 1 0.73149807 0.09829129 0.09137221 0.29630312 0.10394677  
## [2,] 1 2 0.86049010 0.31946830 0.81196535 0.23069610 0.66582378  
## [3,] 1 3 0.34271026 0.99579370 0.60923418 0.91436058 0.77726451  
## [4,] 1 4 0.49292218 0.72628864 0.54550179 0.33242848 0.83036472  
## [5,] 1 5 0.85385328 0.68689645 0.86115736 0.55639850 0.21085632  
## [6,] 1 6 0.38725300 0.62925929 0.28915525 0.58168047 0.19001620  
## [7,] 1 7 0.72650753 0.80341767 0.05022060 0.21686789 0.35220140  
## [8,] 1 8 0.68412170 0.54319481 0.25926730 0.20481063 0.16933552  
## [9,] 1 9 0.84553296 0.86180013 0.32377074 0.97004930 0.08589194  
## [10,] 1 10 0.03280133 0.98419504 0.56787059 0.80117577 0.59034254  
## [11,] 1 11 0.72846174 0.58940472 0.92801585 0.70984189 0.99978366  
## [12,] 1 12 0.47662215 0.05576397 0.77640127 0.04232073 0.75572053  
## [13,] 1 13 0.58189198 0.52718632 0.04720967 0.03252920 0.11253652  
## [14,] 1 14 0.20705047 0.61915121 0.95085707 0.57231710 0.77132551  
## [15,] 1 15 0.02886601 0.16235734 0.94074945 0.29105410 0.57705195  
## [16,] 1 16 0.97326114 0.94045727 0.62315954 0.84213356 0.09920466  
## [17,] 1 17 0.24743524 0.09495347 0.06767010 0.48962278 0.53759064  
## [18,] 1 18 0.11967769 0.81132425 0.52966534 0.56191637 0.86358931  
## [19,] 1 19 0.68486200 0.99313126 0.11851377 0.01541537 0.18983855  
## [20,] 1 20 0.73429616 0.64432900 0.58388602 0.53254758 0.37410482  
## [21,] 1 21 0.32167106 0.25288837 0.36070055 0.96168639 0.60273621  
## [22,] 1 22 0.81046812 0.20766974 0.31959602 0.73853754 0.30637105  
## [23,] 1 23 0.38983784 0.81934261 0.17992783 0.77651419 0.74463681  
## [24,] 1 24 0.59608176 0.27172466 0.81226410 0.72826722 0.61791371  
## [25,] 1 25 0.33955052 0.83264349 0.80219439 0.31400033 0.77810131  
## [26,] 1 26 0.71805402 0.54084861 0.54363117 0.17947065 0.01379390  
## [27,] 1 27 0.94725125 0.94608273 0.31220593 0.31819082 0.01269684  
## [28,] 1 28 0.08143136 0.82190213 0.16344792 0.42555763 0.20901244  
## [29,] 1 29 0.23543762 0.96677674 0.76668807 0.62993156 0.18330910  
## [30,] 1 30 0.36254734 0.92393417 0.55944345 0.15093490 0.09902567  
## [31,] 1 31 0.32524083 0.75880468 0.26785503 0.16545047 0.43392745  
## [32,] 1 32 0.41652660 0.50516642 0.08105204 0.31968775 0.09175046  
## [33,] 1 33 0.50568759 0.88836029 0.63447741 0.29762477 0.47388819  
## [34,] 1 34 0.61733642 0.13522406 0.52594549 0.82908562 0.56725120  
## [35,] 1 35 0.58541439 0.13057995 0.96513229 0.52473321 0.30655540  
## [36,] 1 36 0.89923067 0.92284331 0.37664570 0.66009956 0.31370180  
## [37,] 1 37 0.57490272 0.74803855 0.92691961 0.54122883 0.69856812  
## [38,] 1 38 0.31326190 0.05316739 0.74598280 0.19368207 0.31123574  
## [39,] 1 39 0.48000203 0.88808590 0.65998766 0.87895913 0.51423981  
## [40,] 1 40 0.59562137 0.10972519 0.81035610 0.84763656 0.40950645  
## [41,] 1 41 0.06740248 0.59273083 0.58003224 0.65327933 0.18122948  
## [42,] 1 42 0.73487336 0.28164910 0.52076911 0.22348109 0.34221945  
## [43,] 1 43 0.64350289 0.33771894 0.95394268 0.44666081 0.81973171  
## [44,] 1 44 0.06735485 0.49814340 0.69823988 0.93655462 0.85377947  
## [45,] 1 45 0.69747219 0.59414480 0.16731636 0.35201889 0.36816810  
## [46,] 1 46 0.34926046 0.42435408 0.59506421 0.01783839 0.03282380  
## [47,] 1 47 0.31407390 0.29095801 0.74606411 0.89949275 0.53774841  
## [48,] 1 48 0.98231689 0.04817671 0.72992148 0.75291756 0.14918024  
## [49,] 1 49 0.75011625 0.30832535 0.29001394 0.24217220 0.98329912  
## [50,] 1 50 0.70485719 0.87689400 0.20661046 0.11760675 0.59382743  
## [,8] [,9] [,10] [,11] [,12]  
## [1,] 0.008232245 0.98476178 0.50557293 0.016183775 0.40125026  
## [2,] 0.986168406 0.71209087 0.46763697 0.156450682 0.44541262  
## [3,] 0.519065083 0.81067285 0.05118992 0.103208500 0.63419859  
## [4,] 0.662100740 0.27579654 0.69453119 0.465729116 0.59435284  
## [5,] 0.631618618 0.69326140 0.68755525 0.533196670 0.45037978  
## [6,] 0.734377874 0.90390184 0.21679658 0.347548860 0.25092132  
## [7,] 0.668162800 0.54402934 0.15405890 0.574689109 0.16027829  
## [8,] 0.641611923 0.32275995 0.55099391 0.173483612 0.28811773  
## [9,] 0.583565581 0.38481512 0.86226529 0.575272632 0.15352322  
## [10,] 0.740741473 0.63243041 0.99240948 0.826050059 0.82698226  
## [11,] 0.544109944 0.78003754 0.12725362 0.586135594 0.79972600  
## [12,] 0.695229623 0.86026664 0.78795768 0.074851423 0.59629766  
## [13,] 0.023113052 0.98336436 0.49372664 0.258229256 0.80221877  
## [14,] 0.005238139 0.61820192 0.58843582 0.284971256 0.48921320  
## [15,] 0.933839983 0.70814305 0.61695427 0.600551317 0.68068636  
## [16,] 0.494936683 0.54114315 0.22127093 0.592187443 0.11359786  
## [17,] 0.638389433 0.93725473 0.99931427 0.319595568 0.10210373  
## [18,] 0.024420535 0.86476738 0.53409066 0.555119054 0.37570234  
## [19,] 0.668159914 0.95218619 0.53555609 0.116530654 0.13455375  
## [20,] 0.075707714 0.39982990 0.70893422 0.820235664 0.66350441  
## [21,] 0.778828858 0.31632509 0.71599080 0.548244868 0.02885160  
## [22,] 0.549543482 0.21860610 0.84688446 0.797096947 0.23178328  
## [23,] 0.658982719 0.35618253 0.70148822 0.150116781 0.68856513  
## [24,] 0.236532052 0.01114463 0.97851921 0.190152591 0.31014825  
## [25,] 0.024040510 0.30286919 0.32964096 0.423914856 0.24049153  
## [26,] 0.975458995 0.95705320 0.69643901 0.468485936 0.78338062  
## [27,] 0.138390428 0.65429329 0.25053938 0.165887125 0.96226325  
## [28,] 0.089551733 0.47409461 0.17884782 0.348999850 0.34779703  
## [29,] 0.856531251 0.37642951 0.98439220 0.523034851 0.40307761  
## [30,] 0.884585599 0.85437442 0.17650100 0.812696375 0.20332759  
## [31,] 0.983005512 0.24676984 0.72369469 0.017399364 0.01737532  
## [32,] 0.930172621 0.69523195 0.61360997 0.644495933 0.30868722  
## [33,] 0.306485348 0.94868635 0.35046559 0.836006432 0.27916812  
## [34,] 0.831543672 0.67071681 0.55704384 0.032331784 0.98084805  
## [35,] 0.657991263 0.24681909 0.76417078 0.644689335 0.30226272  
## [36,] 0.156407681 0.34456616 0.66980132 0.018594776 0.69233534  
## [37,] 0.979326116 0.37711110 0.95619791 0.764517837 0.69634574  
## [38,] 0.750283002 0.09399076 0.51765207 0.682753824 0.02577699  
## [39,] 0.423113247 0.26107569 0.85886090 0.969918344 0.26551748  
## [40,] 0.182134656 0.37807712 0.84130837 0.506177035 0.75943315  
## [41,] 0.914691336 0.73738503 0.78154460 0.345935923 0.57618475  
## [42,] 0.575925190 0.92303277 0.54874875 0.389551154 0.35599031  
## [43,] 0.303151664 0.50991204 0.52081535 0.543923377 0.07674989  
## [44,] 0.825672190 0.99208128 0.28260537 0.001811212 0.97541759  
## [45,] 0.236306878 0.59003003 0.13924926 0.276543437 0.65578664  
## [46,] 0.091793010 0.35053834 0.46807626 0.750121652 0.23401433  
## [47,] 0.429659042 0.40281470 0.49646693 0.662619112 0.87729389  
## [48,] 0.328053446 0.05129921 0.48257948 0.313898376 0.44920754  
## [49,] 0.343997875 0.50966297 0.97885439 0.292622774 0.62791901  
## [50,] 0.524319724 0.16609155 0.87242485 0.085384910 0.89355626

Again using only the fitting data, fit the linear model y = Xβ+ε, 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.

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

## [,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)  
SSfit

## [1] 34.27347

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

## [1] 39.07738

#With the training data, we can match some of the noise (the ε 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.

#Question 5(b) Repeat the above, but this time add x2, x3 and x4 terms:

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

Xfit <- X[1:50,]

Xtest <- X[51:100,]

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

## [,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)  
SSfit

## [1] 34.94215

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

## [1] 270310.6

#Here the fit for the test data is absolutely horrendous. The problem is that the term β2x2 + β3 x3 + β4 x4 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)

## [1] 21

#2. Suppose you needed all 2n 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 2n × n, where each row is a different binary sequence of length n. Then the following matrix contains all binary sequences of length n + 1:

#matrix(c(0,A,1,A),2,2) ?!

#Here 0 is a vector of zeros and 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.

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

#Question 3. Let A = (ai,j)n (top) i,j=1 (bottom) 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 × n matrices we have, for any i, det(A) = sigma�(j = 1 to n) (-1)^(i+j)ai,j det(A(−i,−j)).

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

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

## [1] 0.07874474