CompStat2022HA1 .

Edvin Magnusson

2022-01-26

Question 1

```
## Setting the working directory.

setwd("C:/Users/edvin/OneDrive/Skrivbord/Statistiska beräkningar")

## Import CSV-file.
villorSW <- read.csv("C:/Users/edvin/OneDrive/Skrivbord/Statistiska beräkningar/villorSW.csv", sep=";")

## Question 1

# Preparing the matrices.

X<- as.matrix(villorSW[,3:6])
X<-cbind(1,X)
colnames(X)[1]<-"(intercept)"
Y<- as.matrix(villorSW[,2])


# Calculating transpose of the matrix X.

XT <- t(X)

# Calculating the matrix multiplication of XT and X

XTX_Multi <- XT%*%X

## Checking if the matrix is non-singular.

det(XTX_Multi)
```

```
## [1] 5.903508e+18
```

```
# The determinant is not zero so the matrix is non-singular.


# Calculating the inverse of the matrix multiplication XTX_Multi.

inverse_XTX <- solve(XTX_Multi)

# Calculating beta-hat.

Beta_hat <- inverse_XTX%*%XT%*%Y
Beta_hat
```

```
##                     [,1]
## (intercept) -104.3522684
## x1            19.8272312
## x2             0.1858251
## x3           377.5787082
## x4            -6.0513371
```

```
# Using the built in function lm

Linear_Model<- lm(villorSW$y~villorSW$x1+villorSW$x2+villorSW$x3+villorSW$x4)
Linear_Model
```

```
##
## Call:
## lm(formula = villorSW$y ~ villorSW$x1 + villorSW$x2 + villorSW$x3 +
##     villorSW$x4)
##
## Coefficients:
## (Intercept)  villorSW$x1  villorSW$x2  villorSW$x3  villorSW$x4
##   -104.3523      19.8272       0.1858     377.5787      -6.0513
```

```
# So the results are equivalent when estimating beta manually with matrix algebra
# and with the built in lm-function.

# Now lets Compare the computing times for the two methods of estimating beta.
# Testing to run 10000 iterations.

# Manual matrix multiplication method
start_time<- Sys.time()

Manual_Method<- c()
for (j in 1:10000){
Manual_Method[j] <- inverse_XTX%*%XT%*%Y
}

end_time <- Sys.time()

computing_time <- end_time-start_time
computing_time
```

```
## Time difference of 0.966238 secs
```

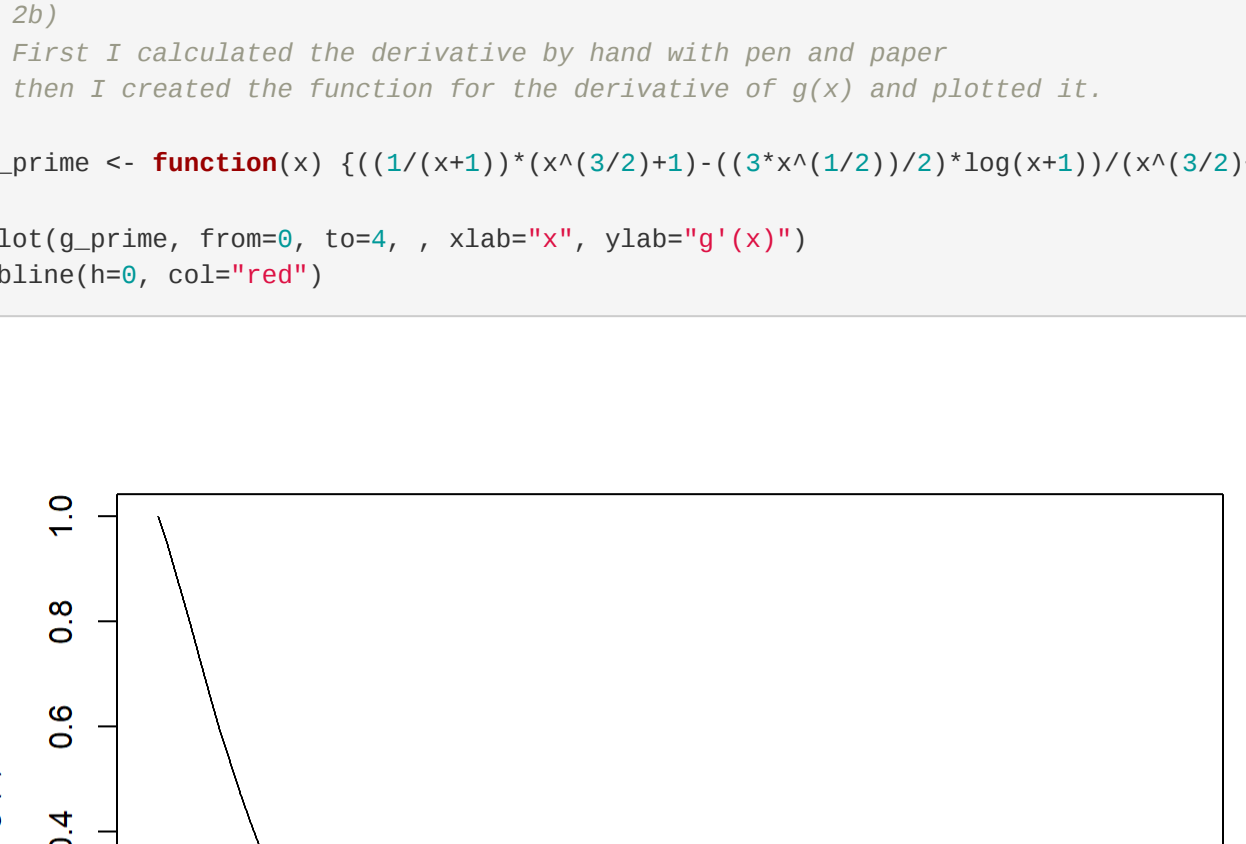
```
# built in lm-method.

start_time2 <- Sys.time()

LM_Method<-c()
for (i in 1:10000){
  LM_Method[i]<- lm(villorSW$y~villorSW$x1+villorSW$x2+villorSW$x3+villorSW$x4)
}

end_time2<- Sys.time()
computing_time2 <- end_time2-start_time2
computing_time2
```

```
## Time difference of 9.631192 secs
```

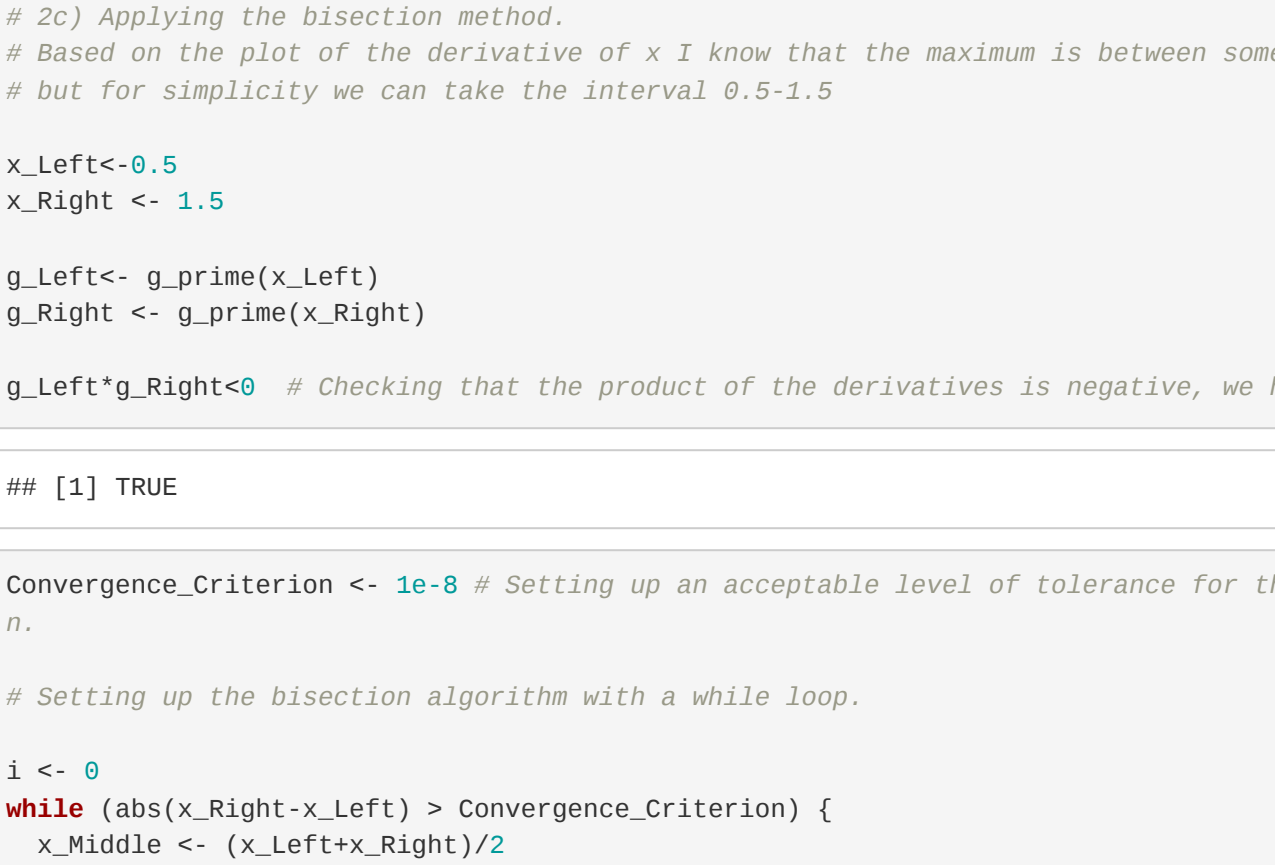
```
# We can see that the built in lm-function is much slower.
```

Question 2

```
# 2a)
# Start with plotting the function g(x)

g <- function(x){(log(x+1))/(x^(3/2)+1)}

plot(g, from=0, to=4, , xlab="x", ylab="g(x)")
```




```
# I would guess the maximum point is around  x=1


# 2b)
# First I calculated the derivative by hand with pen and paper
# then I created the function for the derivative of g(x) and plotted it.

g_prime <- function(x) {((1/(x+1))*(x^(3/2)+1)-((3*x^(1/2))/2)*log(x+1))/(x^(3/2)+1)^2}

plot(g_prime, from=0, to=4, , xlab="x", ylab="g'(x)")
abline(h=0, col="red")
```




```
# 2c) Applying the bisection method.
# Based on the plot of the derivative of x I know that the maximum is between somewhere around 0.9
# but for simplicity we can take the interval 0.5-1.5

x_Left<-0.5
x_Right <- 1.5

g_Left<- g_prime(x_Left)
g_Right <- g_prime(x_Right)

g_Left*g_Right<0  # Checking that the product of the derivatives is negative, we have a maximum point.
```

```
## [1] TRUE
```

```
Convergence_Criterion <- 1e-8 # Setting up an acceptable level of tolerance for the absolute convergence criterio
n.

# Setting up the bisection algorithm with a while loop.

i <- 0
while (abs(x_Right-x_Left) > Convergence_Criterion) {
  x_Middle <- (x_Left+x_Right)/2
  if (g_prime(x_Middle)*g_prime(x_Right) < 0){
    x_Left <- x_Middle
  }else{
    x_Right <- x_Middle
  }
  i <- i+1
  cat("At iteration", i, "X is equal to", x_Middle, "\n")
}
```

```
## At iteration 1 X is equal to 1
## At iteration 2 X is equal to 0.75
## At iteration 3 X is equal to 0.875
## At iteration 4 X is equal to 0.9375
## At iteration 5 X is equal to 0.96875
## At iteration 6 X is equal to 0.953125
## At iteration 7 X is equal to 0.9609375
## At iteration 8 X is equal to 0.9648438
## At iteration 9 X is equal to 0.9628906
## At iteration 10 X is equal to 0.9619141
## At iteration 11 X is equal to 0.9614258
## At iteration 12 X is equal to 0.9611816
## At iteration 13 X is equal to 0.9610596
## At iteration 14 X is equal to 0.9611206
## At iteration 15 X is equal to 0.9610901
## At iteration 16 X is equal to 0.9610748
## At iteration 17 X is equal to 0.9610672
## At iteration 18 X is equal to 0.9610634
## At iteration 19 X is equal to 0.9610615
## At iteration 20 X is equal to 0.9610605
## At iteration 21 X is equal to 0.96106
## At iteration 22 X is equal to 0.9610603
## At iteration 23 X is equal to 0.9610602
## At iteration 24 X is equal to 0.9610602
## At iteration 25 X is equal to 0.9610603
## At iteration 26 X is equal to 0.9610603
## At iteration 27 X is equal to 0.9610603
```

```
# 2d) Using the Secant Method.

Secant_method<- function(fun,x0,x1, tolerance= 1e-8){
  i<-0
  for (i in 1:100){
    x2<- x1-fun(x1)*(x1-x0)/(fun(x1)-fun(x0))
    if (abs(x2-x1)<tolerance){
      return(c(x2,i))

    }
    x0<-x1
    x1<-x2
  }
  i <- i+1


}

Secant_method(g_prime, x0=0.5,x1=1.5)
```

```
## [1]  0.9610603 11.0000000
```

We can see that the secant method was faster and only needed 11 iterations compared to the bisection method who required 27 iterations to reach convergence for the same level of tolerance (epsilon).

Question 3

```
# 3 a)
# Setting up the matrix and then creating the function f(a).

Vector <- c(1,-1,1,-1, 1,"-a","a^2","-a^3",1,"a","a^2","a^3",1,1,1,1)
Test_Mat<- matrix(Vector,nrow = 4,ncol = 4, byrow = TRUE)
Test_Mat
```

```
##      [,1] [,2] [,3]  [,4]
## [1,] "1"  "-1" "1"   "-1"
## [2,] "1"  "-a" "a^2" "-a^3"
## [3,] "1"  "a"  "a^2" "a^3"
## [4,] "1"  "1"  "1"   "1"
```

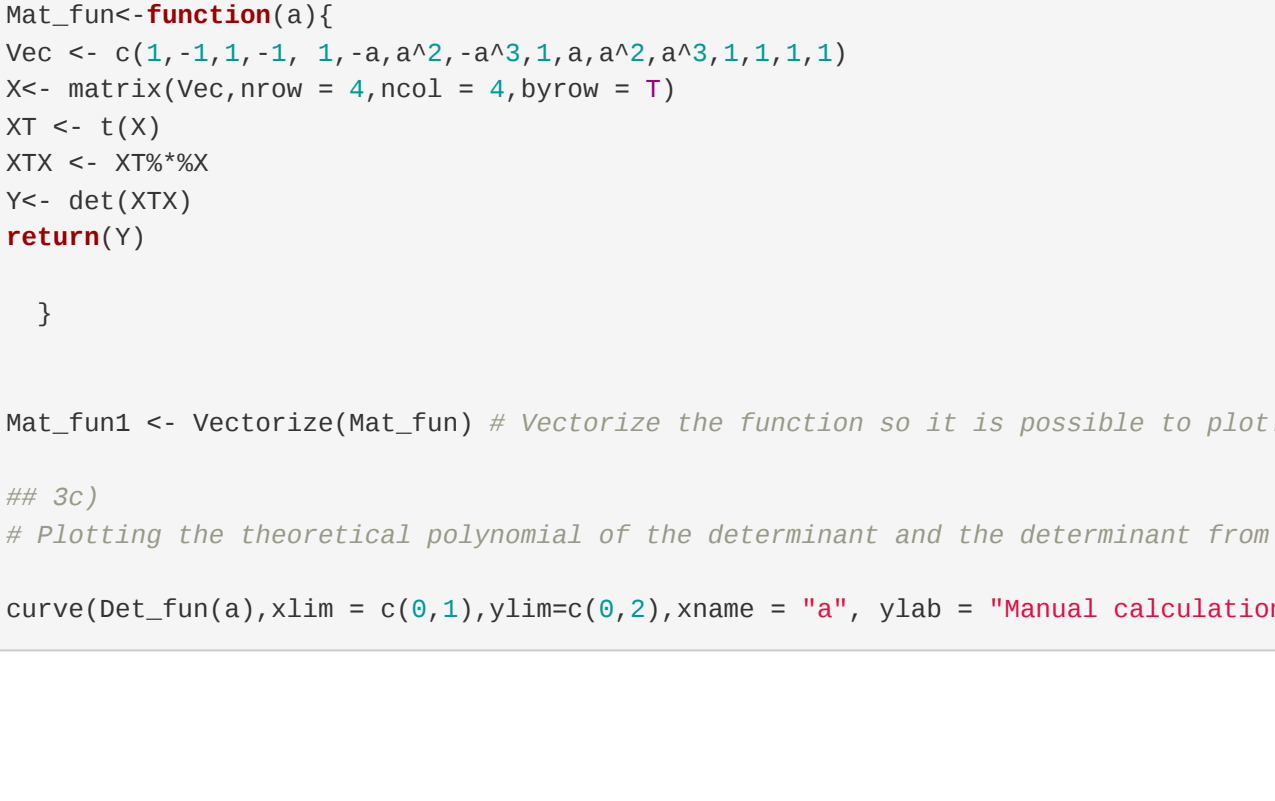
```
Test_mat_T <- t(Test_Mat) # The transpose of the matrix

# Now I have the original matrix and the transpose of the matrix.
# Now I will perform matrix multiplication of them by hand

vector2 <- c(4,0,"2+2a^2",0,0,"2+2a^2",0,"2+2a^4","2+2a^2",0,"2+2a^4",0,0 ,"2+2a^4",0,"2+2a^6")
XTX <- matrix(vector2,nrow = 4,ncol = 4)

XTX # This is the matrix obtained from multiplication, which is used to calculate the determinant.
```

```
##      [,1]     [,2]     [,3]     [,4]
## [1,] "4"      "0"      "2+2a^2" "0"
## [2,] "0"      "2+2a^2" "0"      "2+2a^4"
## [3,] "2+2a^2" "0"      "2+2a^4" "0"
## [4,] "0"      "2+2a^4" "0"      "2+2a^6"
```

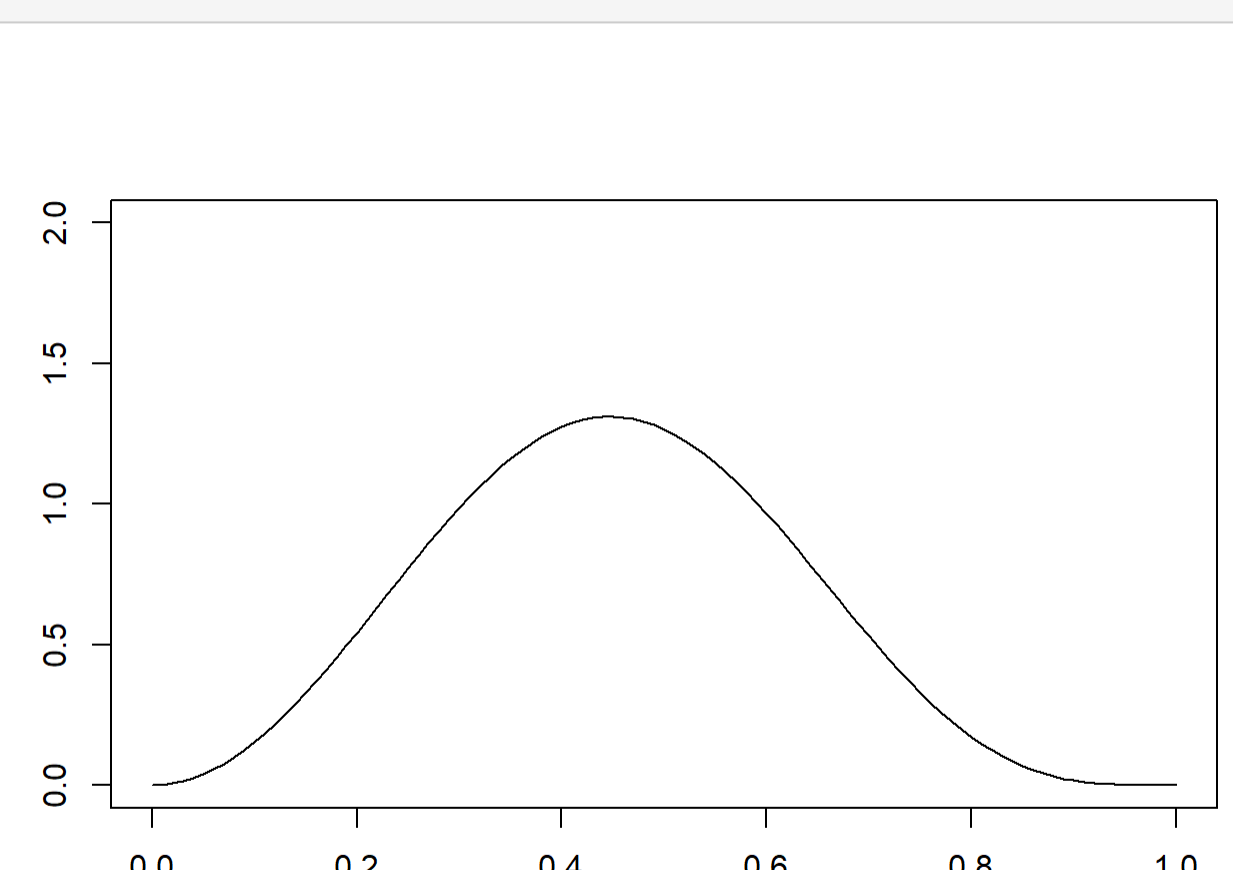
```
# Now I rearrange columns and rows in the matrix to obtain the following matrix.

vector3 <- c(4,"2+2a^2",0,0,"2+2a^2","2+2a^4",0,0,0,0,"2+2a^2","2+2a^4",0,0, "2+2a^4", "2+2a^6")
Block_Matrix <- matrix(vector3,nrow = 4, ncol = 4)
Block_Matrix
```

```
##      [,1]     [,2]     [,3]     [,4]
## [1,] "4"      "2+2a^2" "0"      "0"
## [2,] "2+2a^2" "2+2a^4" "0"      "0"
## [3,] "0"      "0"      "2+2a^2" "2+2a^4"
## [4,] "0"      "0"      "2+2a^4" "2+2a^6"
```

```
# Then the determinant of the matrix is the product of the determinants of the two blocks
# in the matrix. The first block is in the upper  left corner
# and the second is in the lower right corner of the matrix.
# This is solved by pen and paper.


# I received the following polynomial for the variable a:
# 16a^10-64a^8+96a^6-64a^4+16a^2

# 3b)
# Starts with creating a function of the polynomial of the determinant found in a)


Det_fun <- function(a){
  (16*a^10)-(64*a^8)+(96*a^6)-(64*a^4)+(16*a^2)
}



plot(Det_fun, from=0, to=1, , xlab="a", ylab="g(a)")
```




```
# Now we need to create a function of the matrix to be able to use det-function on it.


Mat_fun<-function(a){
Vec <- c(1,-1,1,-1, 1,-a,a^2,-a^3,1,a,a^2,a^3,1,1,1,1)
X<- matrix(Vec,nrow = 4,ncol = 4,byrow = T)
XT <- t(X)
XTX <- XT%*%X
Y<- det(XTX)
return(Y)

  }


Mat_fun1 <- Vectorize(Mat_fun) # Vectorize the function so it is possible to plot.

## 3c)
# Plotting the theoretical polynomial of the determinant and the determinant from the det-function.

curve(Det_fun(a),xlim = c(0,1),ylim=c(0,2),xname = "a", ylab = "Manual calculation of det(a)")
```

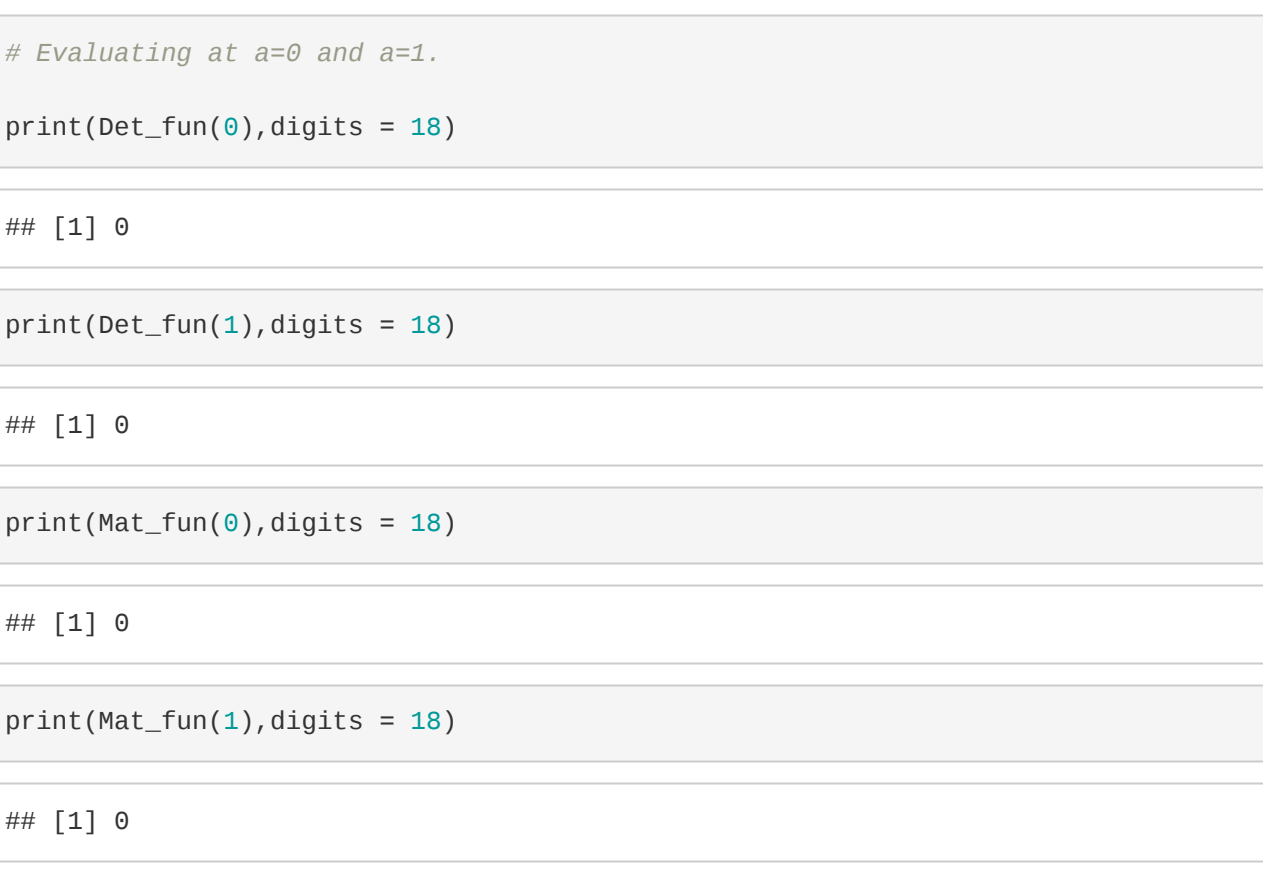



```
curve(Mat_fun1(a),xlim = c(0,1),ylim=c(0,2),xname = "a", ylab ="Built in function det(a)")
```


```
# To plot their difference we create a function for their difference

Difference <- function(a){
 Manual<- (16*a^10)-(64*a^8)+(96*a^6)-(64*a^4)+(16*a^2)
 Vec <- c(1,-1,1,-1, 1,-a,a^2,-a^3,1,a,a^2,a^3,1,1,1,1)
 X<- matrix(Vec,nrow = 4,ncol = 4,byrow = T)
 XT <- t(X)
 XTX <- XT%*%X
 Builtin_fun<- det(XTX)
  Diff <- Manual-Builtin_fun
  return(Diff)
}

Difference(0.8) # There is a small difference between the two methods
```

```
## [1] 2.831069e-15
```

```
Difference1 <- Vectorize(Difference) # Vectorize the function so it is possible to plot.

curve(Difference1(a),xlim = c(0,1),ylim=c(0,0.00000000000001),xname = "a", ylab ="Difference") # Plotting the dif
ference
```




```
# Evaluating at a=0 and a=1.

print(Det_fun(0),digits = 18)
```

```
## [1] 0
```

```
print(Det_fun(1),digits = 18)
```

```
## [1] 0
```

```
print(Mat_fun(0),digits = 18)
```

```
## [1] 0
```

```
print(Mat_fun(1),digits = 18)
```

```
## [1] 0
```

```
# The determinant is zero at a=0 and at a=1

# 3d)
# Optimizes the polynomial of the determinant

Local_max<- optimize(Det_fun,c(0,1),maximum = TRUE)
Local_max
```

```
## $maximum
## [1] 0.4472098
##
## $objective
## [1] 1.31072
```

```
# The local maximum is obtained at a= 0,4472098 and attains the maximum value 1,31072
```

Question 4

For question 4 I followed the steps of the Nelder-Mead algorithm. Which are:

1. Sort
2. Refelct
3. Extend
4. Contract
5. Shrink
6. Check convergence