Homework 6

Yu-Jen Lin

b04b01036@ntu.edu.tw (mailto:b04b01036@ntu.edu.tw)

Set working directory

```
setwd("~/R")
```

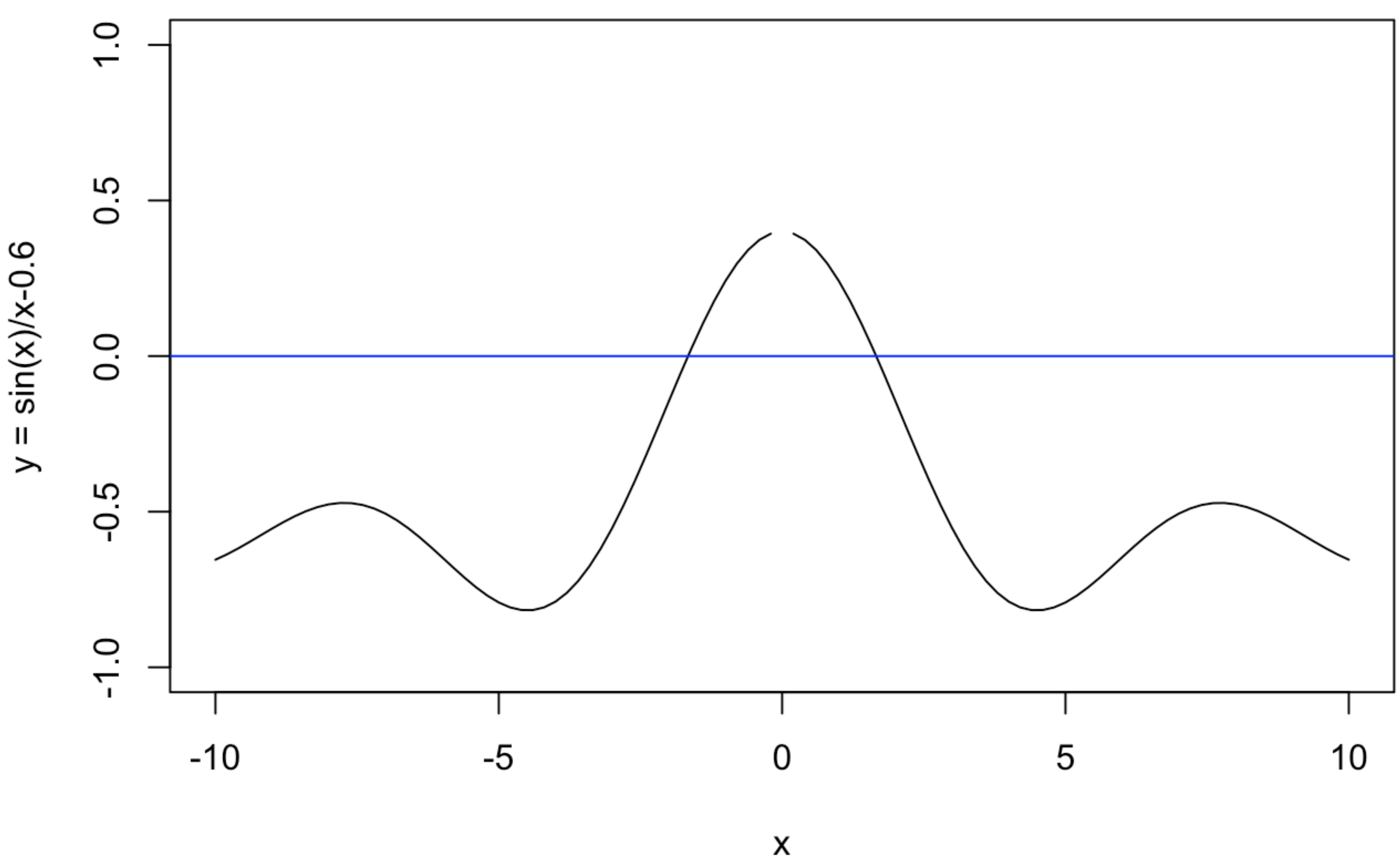
---

1. Write a function to do root-finding based on Newton-Ralphson method and solve for sin(x)/x-0.6=0. (Note: write your own codes, set tol=0.000001, try different initial values)

Plot the equation sin(x)/x-0.6=0

I first plot the equation (black) and the horizontal line that y=0 (blue). I find that there are 2 roots of this equation.

```
eq = function(x){y=sin(x)/x-0.6}
curve(eq, from=-10, to=10, xlab="x", ylab="y = sin(x)/x-0.6", ylim = c(-1, 1))
abline(h=0, col=4)
```



Calculate the derivative of a function in r

In order to calculate the derivative of a function, I install the “Deriv” package.

```
# install.packages("Deriv") # If it is the first time for the computer to use "Deriv", then we'll need to insta
ll it.
library(Deriv)
```

```
## Warning: package 'Deriv' was built under R version 3.4.3
```

```
# Use Deriv, I get the derivative of this function "eq".
# eq = function(x){y=sin(x)/x-0.6}
Deriv(eq) # Through this step, now I know that Deriv(eq) = (cos(x) - sin(x)/x)/x.
```

```
## function (x)
## c(x = (cos(x) - sin(x)/x)/x)
```

Newton-Ralphson method

x1 = x - f(x)/f'(x)

When improvement < tolerance, stop the loop.

improvement = | x1 - x | / x

tolerance = 0.000001

```
# Set the initial values
x1 = 1 # initial value
x = 10 # in order to begin the calculation

# Use for loop to repeat the process until the tol<=0.000001.
for (i in c(1:100000)) {
  if ((abs((x1-x)/x)) >= 0.000001){
    # When improvement < tolerance, stop the loop. Now improvement >= tolerance, keep repeating the loop.
    x <- x1
    f.x <- sin(x)/x-0.6
    Fun.deriv.x <- function(x){(cos(x) - sin(x)/x)/x}

    # x1 <- x - f(x)/f'(x)
    x1 <- x - f.x / Fun.deriv.x(x)

  }else{
    cat("root =",x,"\n") # print the result
    cat(i,"times of improvements","\n") # print the result
    ini.1 <- x
    runs.1 <- i
    break
  }
}
```

```
## root = 1.660035
## 5 times of improvements
```

Try different initial values

I repeat last step by setting the different initial values = -100,-34,-7,-1,1,2,5,50 as examples to show that they all converge to the same roots.

```
######## [ REPEAT ] initial value = 2 ###################################################
# Set the initial values
x1 = 2 # initial value
x = 10 # in order to begin the calculation

# Use for loop to repeat the process until the tol<=0.000001.
for (i in c(1:100000)) {
  if ((abs((x1-x)/x)) >= 0.000001){
    # When improvement < tolerance, stop the loop. Now improvement >= tolerance, keep repeating the loop.
    x <- x1
    f.x <- sin(x)/x-0.6
    Fun.deriv.x <- function(x){(cos(x) - sin(x)/x)/x}

    # x1 <- x - f(x)/f'(x)
    x1 <- x - f.x / Fun.deriv.x(x)

  }else{
    ini.2 <- x
    runs.2 <- i
    break
  }
}
######## [ REPEAT ] initial value = 5 ###################################################
# Set the initial values
x1 = 5 # initial value
x = 10 # in order to begin the calculation

# Use for loop to repeat the process until the tol<=0.000001.
for (i in c(1:100000)) {
  if ((abs((x1-x)/x)) >= 0.000001){
    # When improvement < tolerance, stop the loop. Now improvement >= tolerance, keep repeating the loop.
    x <- x1
```

```
    f.x <- sin(x)/x-0.6
    Fun.deriv.x <- function(x){(cos(x) - sin(x)/x)/x}

    # x1 <- x - f(x)/f'(x)
    x1 <- x - f.x / Fun.deriv.x(x)

  }else{
    ini.5 <- x
    runs.5 <- i
    break
  }
}
######## [ REPEAT ] initial value = 50 ##################################################
# Set the initial values
x1 = 50 # initial value
x = 10 # in order to begin the calculation

# Use for loop to repeat the process until the tol<=0.000001.
for (i in c(1:100000)) {
  if ((abs((x1-x)/x)) >= 0.000001){
    # When improvement < tolerance, stop the loop. Now improvement >= tolerance, keep repeating the loop.
    x <- x1
    f.x <- sin(x)/x-0.6
    Fun.deriv.x <- function(x){(cos(x) - sin(x)/x)/x}

    # x1 <- x - f(x)/f'(x)
    x1 <- x - f.x / Fun.deriv.x(x)

  }else{
    ini.50 <- x
    runs.50 <- i
    break
  }
}
######## [ REPEAT ] initial value = -1 ##################################################
# Set the initial values
x1 = -1 # initial value
x = 10 # in order to begin the calculation

# Use for loop to repeat the process until the tol<=0.000001.
for (i in c(1:100000)) {
  if ((abs((x1-x)/x)) >= 0.000001){
    # When improvement < tolerance, stop the loop. Now improvement >= tolerance, keep repeating the loop.
    x <- x1
    f.x <- sin(x)/x-0.6
    Fun.deriv.x <- function(x){(cos(x) - sin(x)/x)/x}

    # x1 <- x - f(x)/f'(x)
    x1 <- x - f.x / Fun.deriv.x(x)

  }else{
    ini._1 <- x
    runs._1 <- i
    break
  }
}
######## [ REPEAT ] initial value = -7 ##################################################
# Set the initial values
x1 = -7 # initial value
x = 10 # in order to begin the calculation

# Use for loop to repeat the process until the tol<=0.000001.
for (i in c(1:100000)) {
  if ((abs((x1-x)/x)) >= 0.000001){
    # When improvement < tolerance, stop the loop. Now improvement >= tolerance, keep repeating the loop.
    x <- x1
    f.x <- sin(x)/x-0.6
    Fun.deriv.x <- function(x){(cos(x) - sin(x)/x)/x}

    # x1 <- x - f(x)/f'(x)
    x1 <- x - f.x / Fun.deriv.x(x)
```

```r
  }else{
    ini._7 <- x
    runs._7 <- i
    break
  }
}
######## [ REPEAT ] initial value = -34 ##################################################
# Set the initial values
x1 = -34 # initial value
x = 10 # in order to begin the calculation

# Use for loop to repeat the process until the tol<=0.000001.
for (i in c(1:100000)) {
  if ((abs((x1-x)/x)) >= 0.000001){
    # When improvement < tolerance, stop the loop. Now improvement >= tolerance, keep repeating the loop.
    x <- x1
    f.x <- sin(x)/x-0.6
    Fun.deriv.x <- function(x){(cos(x) - sin(x)/x)/x}

    # x1 <- x - f(x)/f'(x)
    x1 <- x - f.x / Fun.deriv.x(x)

  }else{
    ini._34 <- x
    runs._34 <- i
    break
  }
}
######## [ REPEAT ] initial value = -100 ##################################################
# Set the initial values
x1 = -100 # initial value
x = 10 # in order to begin the calculation

# Use for loop to repeat the process until the tol<=0.000001.
for (i in c(1:100000)) {
  if ((abs((x1-x)/x)) >= 0.000001){
    # When improvement < tolerance, stop the loop. Now improvement >= tolerance, keep repeating the loop.
    x <- x1
    f.x <- sin(x)/x-0.6
    Fun.deriv.x <- function(x){(cos(x) - sin(x)/x)/x}

    # x1 <- x - f(x)/f'(x)
    x1 <- x - f.x / Fun.deriv.x(x)

  }else{
    ini._100 <- x
    runs._100 <- i
    break
  }
}
```

All converge to the same roots

I conclude the results and find that different initial values = -100,-34,-7,-1,1,2,5,50 all converge to the same roots (i.e. -1.660035 & 1.660035). Moreover, they all need different amount of improvements because their starting positions.

```r
initial_values <- c(-100,-34,-7,-1,1,2,5,50)
roots <- c(ini._100, ini._34, ini._7, ini._1, ini._1, ini.2, ini.5, ini.50)
runs_of_improvements <- c(runs._100, runs._34, runs._7, runs._1, runs._1, runs.2, runs.5, runs.50)
results <- data.frame(initial_values, roots, runs_of_improvements)
results
```

```
##   initial_values     roots runs_of_improvements
## 1           -100 -1.660035                  616
## 2            -34 -1.660035                   26
## 3             -7 -1.660035                   26
## 4             -1 -1.660035                    5
## 5              1 -1.660035                    5
## 6              2  1.660035                    5
## 7              5 -1.660035                   13
## 8             50 -1.660035                   28
```

```
write.csv(results, "~/R/HW6_1.results.csv") # export the results
```

---

2. Use data from Vidal (1980) and find the Belehradek's equation for C2, C3, C4, C5 by minimizing the least square error, and set b=-2.05. Plot the data and fitted curves. (Hint: use optim in R or fminsearch in Matlab)

Read VidalTvsDuration.txt into R

```
VidalTvsDuration <- read.table(file="~/R/VidalTvsDuration.txt", header=TRUE)
```

```
## Warning in read.table(file = "~/R/VidalTvsDuration.txt", header
## = TRUE): incomplete final line found by readTableHeader on '~/R/
## VidalTvsDuration.txt'
```

Belehradek s equation for zooplankton development time

D = a*(T-α)^b

```r
# D = development time of a stage
# a = parameter (unknown) = par[1]
# T = temperature
# α = parameter (unknown) = par[2]
# b = -2.05 (we set this parameter)

temp <- VidalTvsDuration$tempearture
D.C2 <- VidalTvsDuration$C2
D.C3 <- VidalTvsDuration$C3
D.C4 <- VidalTvsDuration$C4
D.C5 <- VidalTvsDuration$C5

######## C2 ###################################################
data.C2 <- data.frame(D.C2, temp)

# SSR
FUN.SSR.C2 <- function(data, par){
   with(data.C2, sum (( D.C2 - par[1]*(temp-par[2])^(-2.05) ) ^ 2 ))  # sum of (yi - yi.hat)^2
}

# Minimizing the least square error
optimization.C2 <- optim(par=c(5,5), fn=FUN.SSR.C2, data=data.C2)

# Fitted curves equation
eq.C2 = function(x){y=optimization.C2$par[1]*(x-optimization.C2$par[2])^(-2.05)}

######## C3 ###################################################
data.C3 <- data.frame(D.C3, temp)

# SSR
FUN.SSR.C3 <- function(data, par){
   with(data.C3, sum (( D.C3 - par[1]*(temp-par[2])^(-2.05) ) ^ 2 ))  # sum of (yi - yi.hat)^2
}

# Minimizing the least square error
optimization.C3 <- optim(par=c(5,5), fn=FUN.SSR.C3, data=data.C3)

# Fitted curves equation
eq.C3 = function(x){y=optimization.C3$par[1]*(x-optimization.C3$par[2])^(-2.05)}

######## C4 ###################################################
data.C4 <- data.frame(D.C4, temp)

# SSR
FUN.SSR.C4 <- function(data, par){
   with(data.C4, sum (( D.C4 - par[1]*(temp-par[2])^(-2.05) ) ^ 2 ))  # sum of (yi - yi.hat)^2
}

# Minimizing the least square error
optimization.C4 <- optim(par=c(5,5), fn=FUN.SSR.C4, data=data.C4)

# Fitted curves equation
eq.C4 = function(x){y=optimization.C4$par[1]*(x-optimization.C4$par[2])^(-2.05)}

######## C5 ###################################################
data.C5 <- data.frame(D.C5, temp)

# SSR
FUN.SSR.C5 <- function(data, par){
   with(data.C5, sum (( D.C5 - par[1]*(temp-par[2])^(-2.05) ) ^ 2 ))  # sum of (yi - yi.hat)^2
}

# Minimizing the least square error
optimization.C5 <- optim(par=c(5,5), fn=FUN.SSR.C5, data=data.C5)

# Fitted curves equation
eq.C5 = function(x){y=optimization.C5$par[1]*(x-optimization.C5$par[2])^(-2.05)}
```

Plot the data and fitted curves

```r
# Plot the data
plot(
  x = rep(temp,4),
  y = c(D.C2, D.C3, D.C4, D.C5),
  xlab = "tempearture",
  ylab = "development time",
  xlim = c(5,20),
  ylim = c(0,30)
  )
legend("topright",c("C5","C4","C3","C2"), col=c("blue","green","orange","red"), lty=1)
# Fitted curves
par(new = TRUE)
curve( eq.C2, xlab = "", ylab = "", xlim = c(5,20), ylim = c(0,30),  col = "red")
par(new = TRUE)
curve( eq.C3, xlab = "", ylab = "", xlim = c(5,20), ylim = c(0,30),  col = "orange")
par(new = TRUE)
curve( eq.C4, xlab = "", ylab = "", xlim = c(5,20), ylim = c(0,30),  col = "green")
par(new = TRUE)
curve( eq.C5, xlab = "", ylab = "", xlim = c(5,20), ylim = c(0,30),  col = "blue")
```