HW2

Xiaofu Dai

February 16, 2020

Part 1

```
library(ggplot2)
library(viridis)
```

```
## Loading required package: viridisLite
```

```
palette(viridis(256))
library(future.apply)
```

```
## Warning: package 'future.apply' was built under R version 3.6.2
```

```
## Loading required package: future
```

```
## Warning: package 'future' was built under R version 3.6.2
```

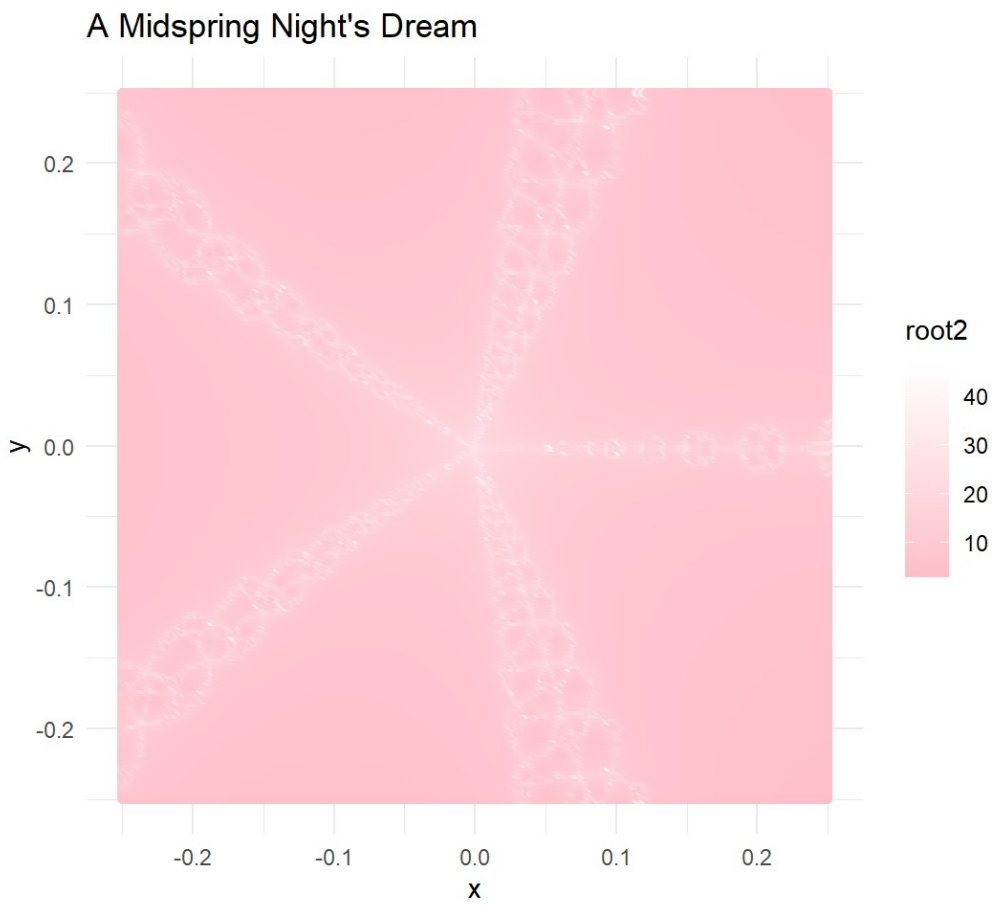
```
plan(multicore)
```

function

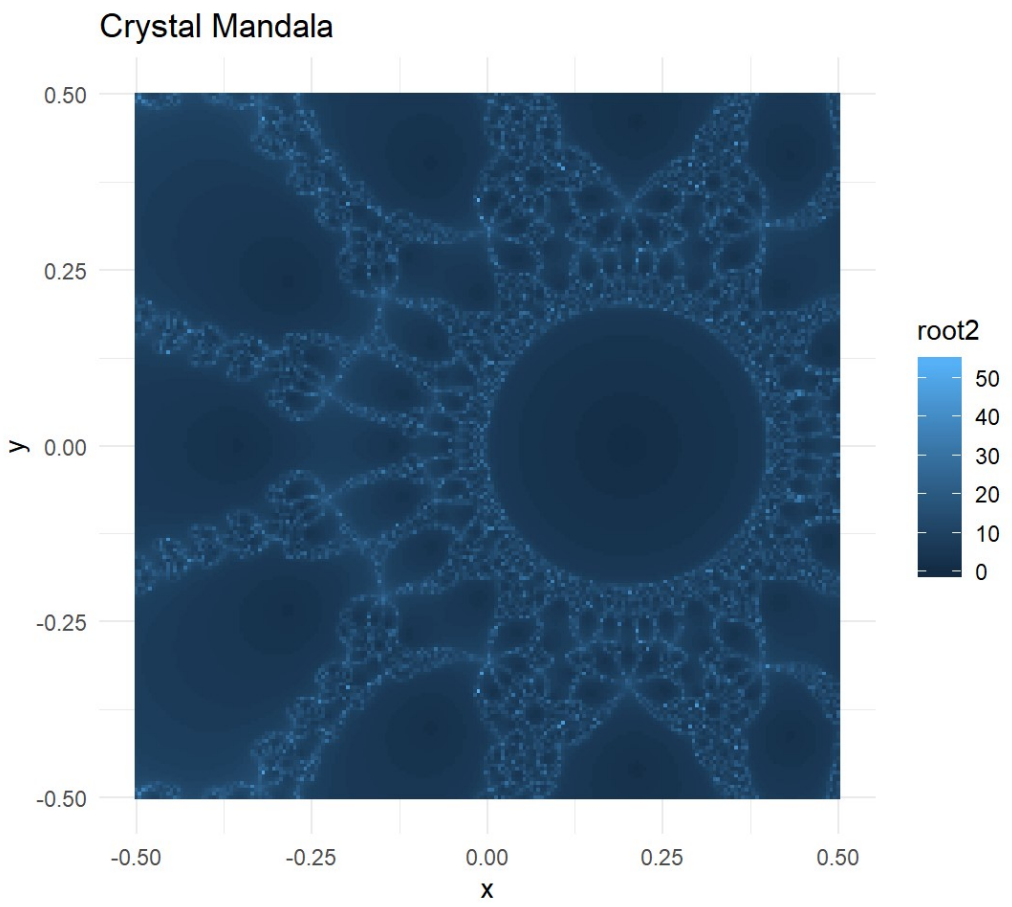
I use f(x) = x^5 - x^2 +1, f(x) = x^6 + 1/x - 5 to plot the art show plots.

```
# Functions
F1 <- function(x){
  fx = x^5 - x^2 +1
  dfx = 5*x^4-2*x
  return(c(fx, dfx))
}

F2 <- function(x){
  fx = x^6 + 1/x - 5
  dfx = 6*(x^5) - 1/(x^2)
  return(c(fx, dfx))
}

# global control variable
bRootOrIterations<-0

# Here's the function that performs Newton-Raphson
TwoDNewtonRaphson<-function(func,StartingValue,Tolerance,MaxNumberOfIterations){
  i<-0
  NewZ<- StartingValue
  Deviation=abs(func(StartingValue)[1])

  while ((i<MaxNumberOfIterations)&&(Deviation>Tolerance))
  {
    Z<-func(NewZ)
    if ((Z[1]=="NaN")||(Z[2]=="NaN")){
      cat("Function or derivative not defined error.\n"); break
    }
    NewZ <- NewZ-Z[1]/Z[2]
    NewVal <- func(NewZ)
    Deviation <- abs(NewVal[1])
    i<-i+1
  }

  # output the result
  if (Deviation>Tolerance){
    cat(paste("\nConvergence failure. Deviation:",Deviation, "after ", i,   "iteration
s"))
  }

  if (bRootOrIterations==1){
   return(NewZ)
  }else{
    return(c(i,i))
  }
}
```

```
# A function to check whether two points are close together
CloseTo<-function(x,y){
  # returns 1 if x is close to y
  if (abs(x-y)<0.1) {
    return(1)
  }else{
    return (0)
  }
}

# do plot
Root_calculator <- function(Funcn, xmin, xmax, xsteps, ymin, ymax, ysteps) {

  x <- seq(xmin, xmax, length.out = xsteps)
  y <- seq(ymin, ymax, length.out = ysteps)

  out_dat <- expand.grid(x = x, y = y)

  ThisZ <- complex(1, out_dat$x, out_dat$y)

  Root <- sapply(ThisZ,
                 FUN = TwoDNewtonRaphson,
                 func = Funcn,
                 Tolerance = 1e-1,
                 MaxNumberOfIterations = 100)

  if(bRootOrIterations == 0) {
    out_dat$color <- 104 + 5 * Root[1, ]
    out_dat$root1 <- Root[1, ]
    out_dat$root2 <- Root[2, ]
  } else {
    out_dat$color <- 5 * Root
    out_dat$root1 <- Root
  }

  return(out_dat)
}



Func1 <- Root_calculator(F1, -1/4, 1/4, 200, -1/4, 1/4, 200)
ggplot(Func1, aes(x, y,color = root2)) +
  geom_point()+
    theme_minimal() +
   scale_color_gradient(low="pink", high = "white")+
    coord_fixed() +
    labs(title = "A Midspring Night's Dream")
```




```
Func2 <- Root_calculator(F2, -1/2, 1/2, 200, -1/2, 1/2, 200)
ggplot(Func2, aes(x, y, fill = root2)) +
    geom_tile()+
    scale_color_gradient2(high="blue", mid="pink",
                    low="blue", space ="Lab" ) +
    theme_minimal() +
    coord_fixed() +
    labs(title = "Crystal Mandala")
```



Part 2

1.Write a program to implement the Secant method. 2.Test it, using x0=1 and x1=2 on: 1.cos(x)-x 2.log(x)-exp(-x) Compare it with the performance of Newton-Raphson on the same functions. 3. Write a function to show a plot of the iterations of the algorithm.

Compare Newton's Method with the secant method, we see Newton's method requires the evaluation of both f(x) and its derivative f'(x) at every step, while the secant method only requires the evaluation of f(x). So if it's much easier to calculate f(x) or we can't get f'(x), the secant method is a better choice.

1. Secant method

```r
# Functions
f1 = function(x){
  return(cos(x)-x)
}
f2 = function(x){
  return(log(x)-exp(-x))
}

# a sufficiently small difference between xn and xn-1
secant = function(func, x0, x1, Tolerance, MaxNumberOfIterations){
  i=0
  while (abs(x1-x0) > Tolerance & i < MaxNumberOfIterations) {
    x2 = x1 - func(x1)*(x1-x0)/(func(x1)-func(x0))
    x0 = x1
    x1 = x2
    i = i+1
  }
  return(c(i,x1))
}
```

2. Newton-Raphson

```
# Functions
f3 <- function(x){
  fx = cos(x)-x
  dfx = -sin(x)-1
  return(c(fx, dfx))
}

f4 <- function(x){
  fx = log(x)-exp(-x)
  dfx = 1/x + exp(-x)
  return(c(fx, dfx))
}

# global control variable
bRootOrIterations<-0

# Here's the function that performs Newton-Raphson
NewtonRaphson<-function(func,StartingValue,Tolerance,MaxNumberOfIterations){
  i<-0
  NewZ<- StartingValue
  Deviation=abs(func(StartingValue)[1])

  while ((i<MaxNumberOfIterations)&&(Deviation>Tolerance))
  {
    Z<-func(NewZ)
    if ((Z[1]=="NaN")||(Z[2]=="NaN")){
      cat("Function or derivative not defined error.\n"); break
    }
    NewZ <- NewZ-Z[1]/Z[2]
    NewVal <- func(NewZ)
    Deviation <- abs(NewVal[1])
    i<-i+1
  }

  # output the result
  if (Deviation>Tolerance){
    cat(paste("\nConvergence failure. Deviation:",Deviation, "after ", i,   "iteration
s"))
  }

  if (bRootOrIterations==1){
   return(NewZ)
  }else{
    return(c(i,NewZ))
  }
}
```

Compare it with the performance of Newton-Raphson on the same functions.

```
# parameters
x0 = 1
x1 =2

# for 1. cos(x)-x
# Secant Method
secant(f1, x0, x1, 1e-04, 40)
```

```
## [1] 4.0000000 0.7390851
```

```
# Newton Method
NewtonRaphson(f3, x0, 1e-04, 40)
```

```
## [1] 2.0000000 0.7391129
```

```
# for 2.log(x)-exp(-x)
# Secant Method
secant(f2, x0, x1, 1e-04, 40)
```

```
## [1] 5.0000 1.3098
```

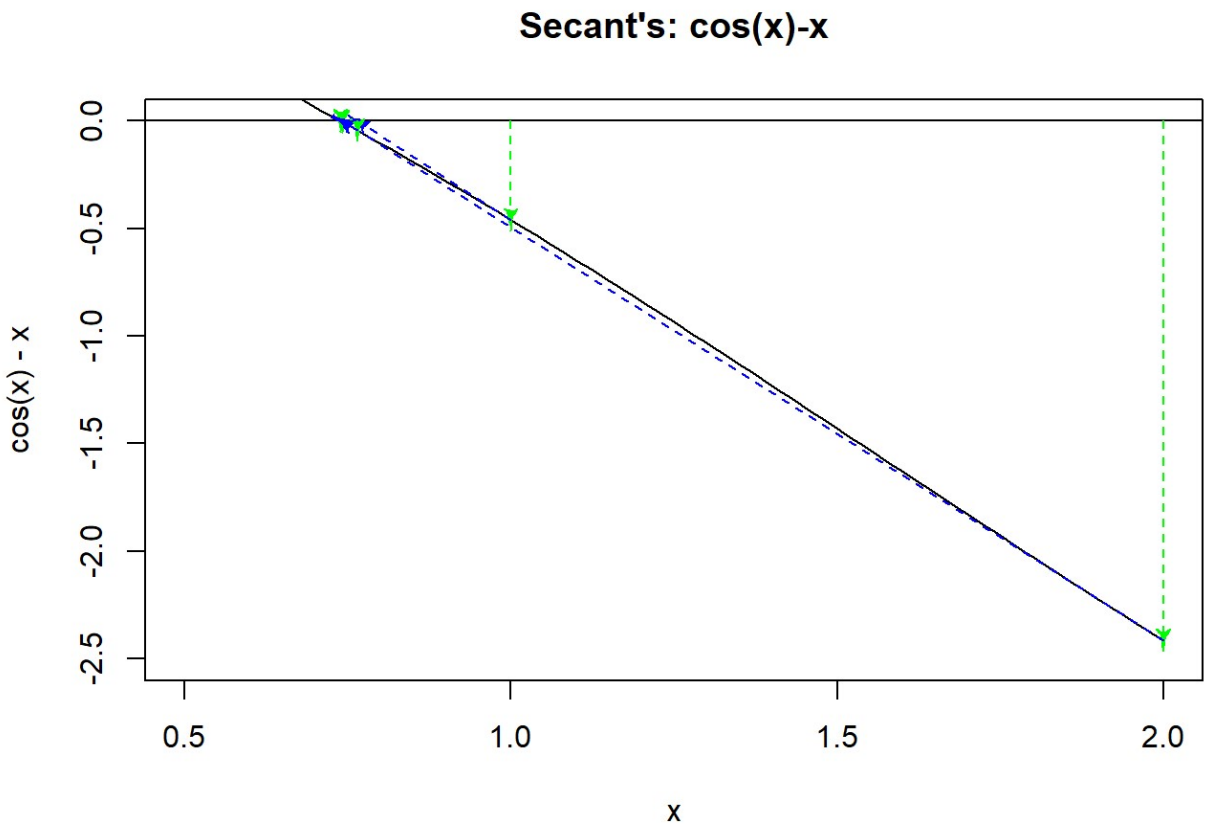
```
# Newton Method
NewtonRaphson(f4, x0, 1e-04, 40)
```

```
## [1] 3.000000 1.309799
```

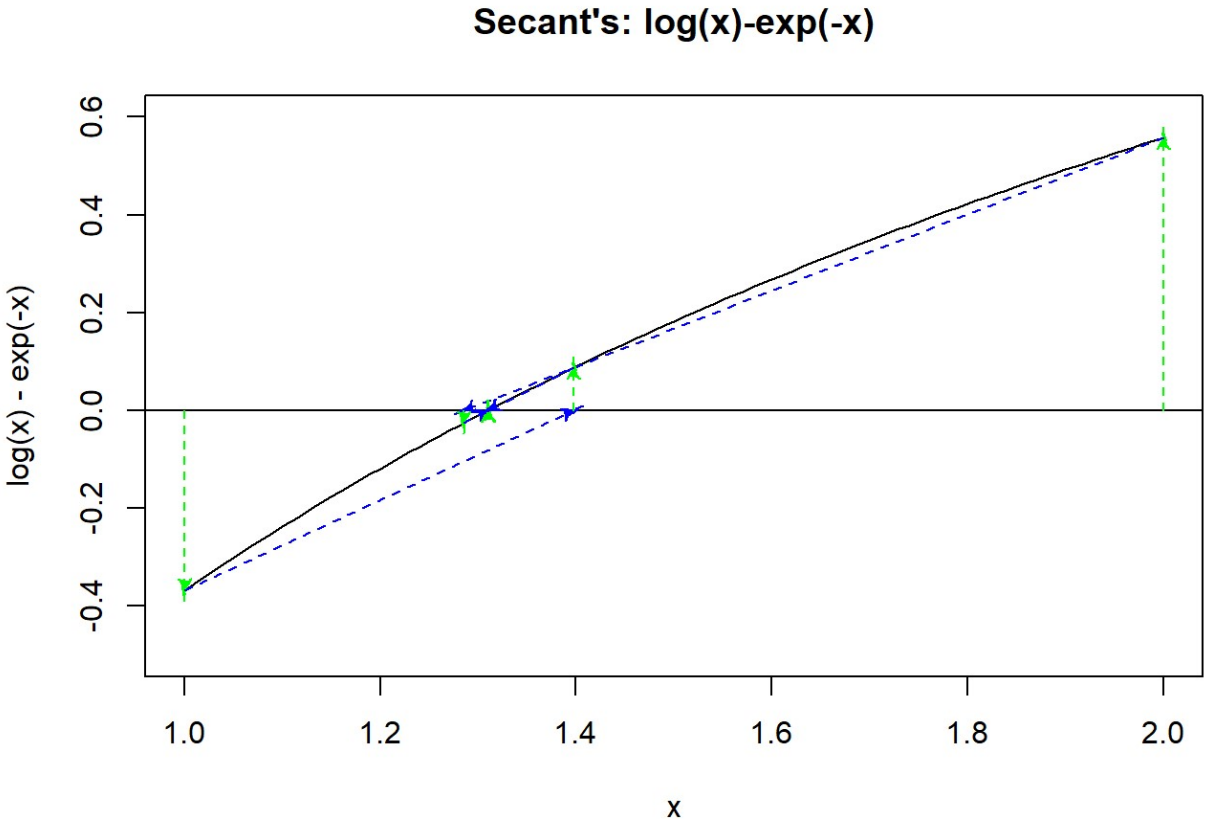
```
# Two methods got similar roots for two functions, separately.
# But the Newton's method took less interations to get the root than that of the secan
t method.
# The Newton's method took 2 and 3 interations to get the root of function 1 and funct
ion 2, respectively, while secant method took 4 and 5 interactions, respectively.

# Compare Newton's method with the secant method, we see that Newton's method converge
s faster. And as mentioned before, Newton's method requires the evaluation of both f
(x) and its derivative f'(x) at every step, while the secant method only requires the
evaluation of f(x).
```

Write a function to show a plot of the iterations of the algorithm.

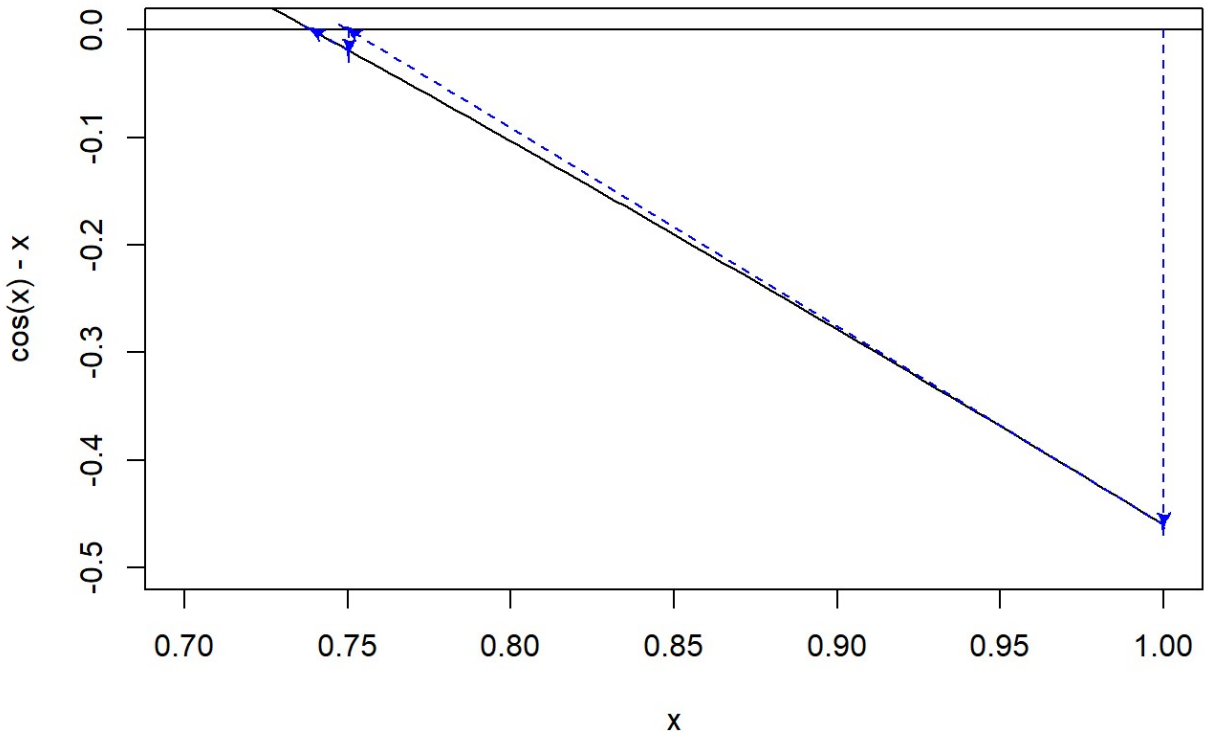
```
# For Secant Method
library(shape)
secantplot = function(func, x0, x1, Tolerance, MaxNumberOfIterations,DrawLines)
  {
  i=0
  x0 = x0
  x1 = x1

  while (abs(x1-x0) > Tolerance & i < MaxNumberOfIterations)
    {
    y0 = func(x0)
    y1 = func(x1)
    x2 = x1 - func(x1)*(x1-x0)/(func(x1)-func(x0))  # find another point on xaxis

    if (DrawLines){
            Arrows(x0,0,x0,y0,col="green",lty=2,arr.length=0.25)
      Arrows(x1,0,x1,y1,col="green",lty=2,arr.length=0.25)
            Arrows(x0,y0,x2,0,col="blue",lty=2, arr.length=0.25)
            }

    x0 = x1
    x1 = x2
    i = i+1
  }
}

# 1. cox(x)-x
curve(cos(x)-x, main="Secant's: cos(x)-x", xlim = c(0.5,2), ylim = c(-2.5,0))
abline(h = 0, v = 0)
secantplot(f1, x0, x1, 1e-04, 40,1)
```




```
# 2.log(x)-exp(-x)
curve(log(x)-exp(-x), main="Secant's: log(x)-exp(-x)", xlim = c(1,2), ylim = c(-0.5,0.
6))
abline(h = 0, v = 0)
secantplot(f2, x0, x1, 1e-04, 40, 1)
```

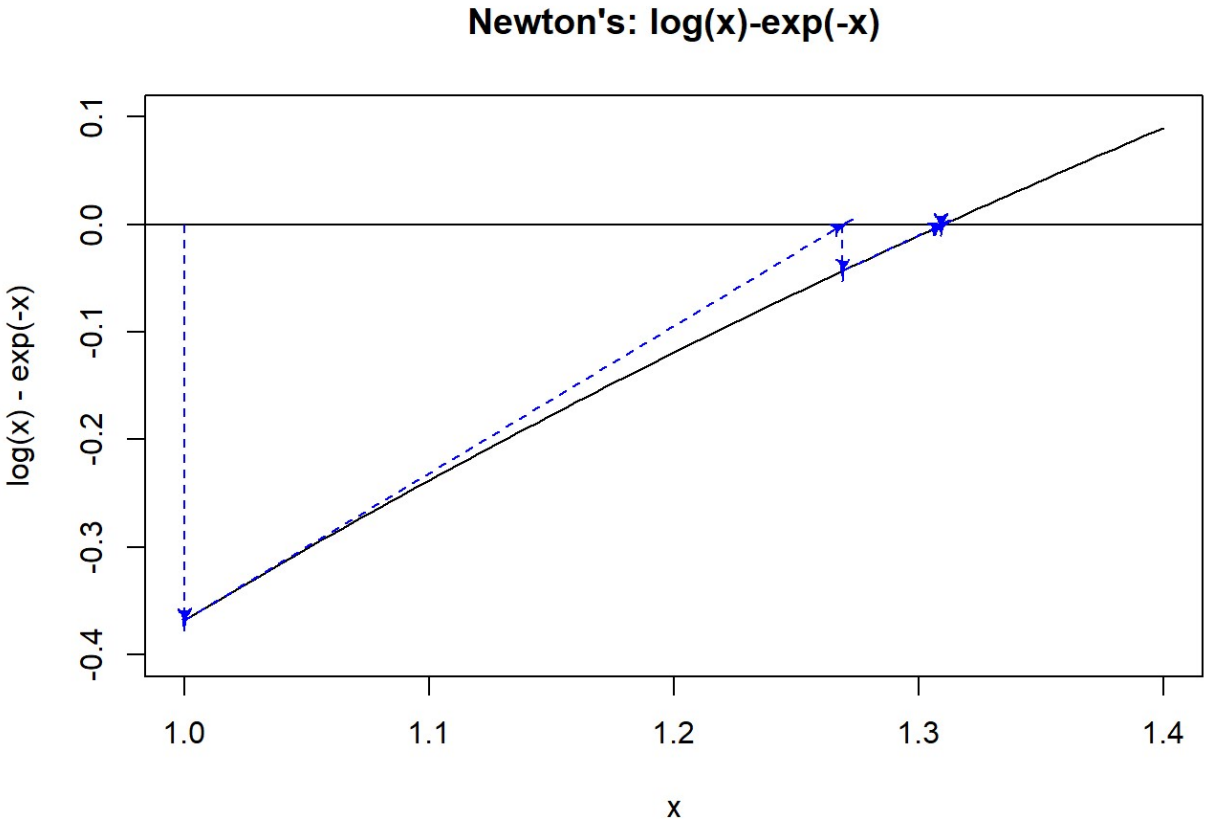

Secant's: log(x)-exp(-x)
log(x) - exp(-x)
0.6
0.4
0.2
0.0
-0.2
-0.4
1.0
1.2
1.4
1.6
1.8
2.0
x

```
# For Newton Method
FixedPointFinder<-function(func,StartingValue,Tolerance,MaxNumberOfIterations,DrawLine
s){

    i<-0
    Xprime<-StartingValue
    Dev = 1000

    while ((i<MaxNumberOfIterations)&&(Dev>Tolerance))
    {

        X<-Xprime     # x axis
        Z<-func(X)    # f(x) and f'(x)
        Y<-Z[1]       # y
        k<-Z[2]       # f'(x)
        if ((Z[1]=="NaN")||(Z[2]=="NaN")){
      cat("Function or derivative not defined error.\n"); break
        }

        X2 = X-Y/k

        if (DrawLines){
            Arrows(X,0,X,Y,col="blue",lty=2,arr.length=0.25)
            Arrows(X,Y,X2,0,col="blue",lty=2,arr.length=0.25)
            }

        Xprime = X2
        NewVal <- func(X2)
      Dev <- abs(NewVal[1])
        i<-i+1
    }
    # output the result
    if (Dev<Tolerance){
        cat(paste("\nFound the fixed point: ",X, "after ", i, "iterations"))
        }else{
        cat(paste("\nConvergence failure. Deviation: ",Dev, "after ", i,    "iteration
s"))}
}

# 1. cox(x)-x
curve(cos(x)-x, main="Newton's: cos(x)-x", ylim=c(-0.5, 0), xlim = c(0.7,1))
abline(h = 0, v = 0)
FixedPointFinder(f3, 1, 1e-04, 40,1)
```




```
##
## Found the fixed point:  0.750363867840244 after  2 iterations
```

```
# 2.log(x)-exp(-x)
curve(log(x)-exp(-x), main="Newton's: log(x)-exp(-x)", ylim=c(-0.4,0.1), xlim = c(1,1.
4))
abline(h = 0, v = 0)
FixedPointFinder(f4, 1, 1e-04, 40,1)
```




```
##
## Found the fixed point:  1.30910840327402 after  3 iterations
```