David Alberto Sarmiento Santamaría

Ingeniería de Sistemas

Fecha: 1 de marzo del 2019

PARCIAL 1.

Solución

1.

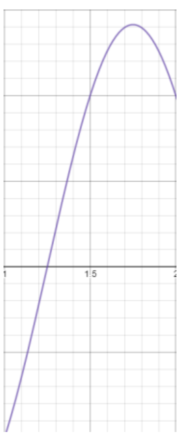
A)

B)

2.

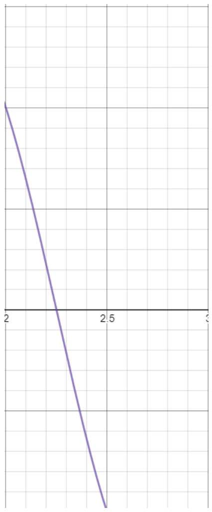
A)

Primer uso de la función: extraerRaiz(1,2, epsylon)



```
x= 2     Error: 0.001651387    iteraciones: 0
x= 2     Error: 2.39142e-11    iteraciones: 1
```

Segundo uso de la función: extraerRaiz(2,3, epsylon)




```
x= 3.03683      Error: 34.14222       iteraciones: 0
x= 1.999989     Error: 51.84234       iteraciones: 1
x= 1.999989     Error: 4.629655e-12   iteraciones: 2
```

Implementación en R

```
#Tx <- function(x)  tan(pi*x)
#Gx <- function(x)  sin(pi*x)
Fx<- function(x) tan(pi*x)-sin(pi*x)

#Se declara el error acorde a lo solicitado
epsylon <- 1.e-9

#Se declara la funcion
#Name: extraerRaiz
#Param: Punto en x, punto en x
extraerRaiz<-function(x1,x2, epsylon)
{
  error<-epsylon+1
  x0<-x1-((Fx(x1)*(x1-x2))/(Fx(x1)-Fx(x2)))

  x1<-x2
  x2<-x0
  i<-0
  while(error>epsylon)
  {
    x3<-x1-((Fx(x1)*(x1-x2))
            /
            (Fx(x1)-Fx(x2)))
    error<-abs((x3-x2)/x3)*100
    #Error arrojado a manera de porcentaje
    cat("x=",x3,"\t Error:",error,"\titeraciones:",i,"\n")
    x1<-x2
    x2<-x3
    i<-i+1
  }
}
```

B)

Algoritmo: Newton + Aitken

```
Nuevo resultado:
newtonAitken(1, 1.e-9)
R= 1     Error: 0 i: 0
```

Solución: Se prefiere usar el método de Newton +Aitken, esto debido a que nos arroja una sola iteración.

Implementación:

Función derivada: Dx = Dx <- function(x) (sec(pi*x)^2)*pi-(cos(pi*x))*pi

NewtonAitken:

```
Dx <- function(x) (sec(pi*x)^2)*pi-(cos(pi*x))*pi

newtonAitken<-function(r,epsylon){

 i<-0

 error<-1

 while(i<=3 && error>epsylon){

  if(r!=0){

   bef=r

   r<-r-((Fx(r))/Dx(r))

   error<-(abs(bef-r))/abs(bef)

   cat("R=",r,"\t Error:",error,"i:",i,"\n")

   if(i==1)

   {

    x0=r

   }

   else

   {

    if(i==2)

    {

     x1=r

    }
```

```
        else
        {
          x2=r
        }
      }
    }
    i=i+1
  }

  while(error>epsylon){
    if(r!=0){
      x3=x2-(((x2-x1)^2)/(x2-(2*x1)+x0))
      x0=x1
      x1=x2
      error<-(abs(x2-x3))/abs(x2)
      cat("R=",x3,"\t Error:",error,"i:",i,"\n")
      r<-r-((Fx(r))/Dx(r))
      x2=r
    }
    i=i+1
  }
}
```

3. Método de Jacobi

-Implementación en R

```
rm(list=ls())
library(pracma)
library(Matrix)
```

```
n = 3 #Matriz 3x3

D1<-eye(n, m = n)
D2<-ones(n, m = n)
D3<-zeros(n, m = n)

A = matrix(c(8, 9, 2,
            2, 7, 2,
            2, 8, 6), nrow=n, byrow=TRUE)

b = matrix(c( 69, 47, 68), nrow=n, byrow=TRUE)

diagonal <- function(M) {

  M[col(M)!=row(M)] <- 0

  return(M)
}

#T = -D^-1(L + U)
D = diagonal(A)
L = tril(A,k=-1,diag = FALSE)
U = triu(A,k=1,diag = FALSE)

sum = L+U
solucion = (-solve(D))

T = round((solucion)%*%(sum),3)
```

print(T)

print(round(norm(T,"F"),3))

-Tabla de resultados

```
3 x 3 Matrix of class "dgeMatrix"
       [,1]   [,2]   [,3]
[1,]  0.000 -1.125 -0.250
[2,] -0.286  0.000 -0.286
[3,] -0.333 -1.333  0.000
```