Parcial

Andres David Gomez B

28 de febrero de 2018

## 1. Matriz triangular

a)Primer punto usa como ejemplo una matriz triangular de 3x3

rm(list=ls())  
cat("recuerde que la matriz debe ser cuadrada para que el algoritmo funcione","\n")

## recuerde que la matriz debe ser cuadrada para que el algoritmo funcione

superior <- function(a) {  
y<-(dim(a))  
i<-1  
sum<-0  
j<-1  
while(i<=y[c(1)])  
{  
 j<-i  
 while(j<=y[c(1)])  
 {  
 sum<-sum+a[(i),(j)]  
 j<-j+1  
 }  
 i<-i+1  
}  
cat(sum)  
}  
a<-matrix(c(1,1,1,0,1,1,0,0,1), nrow=3, byrow=T)  
superior(a)

## 6

1. Dada la formula basta con el siguiente codigo que depende del tamaño:

rm(list=ls())  
eficiencia<- function(a) {  
 t <- proc.time() # Inicia el cronómetro  
 j<-1  
 i<-1  
 sum<-0  
 while(i<=a){  
 j<-i  
 while(j<=a)  
 {  
 sum<-sum+1  
 j<-j+1  
 }   
 i<-i+1  
 }  
 proc.time()-t  
 cat(sum,"Tiempo: ",t[3]) # Detiene el cronómetro  
}  
  
i<-2  
while(i<=25){  
   
 cat(i,": ")  
 eficiencia(i)  
 cat("\n")  
 i<-i+1  
}

## 2 : 3 Tiempo: 1.7  
## 3 : 6 Tiempo: 1.7  
## 4 : 10 Tiempo: 1.7  
## 5 : 15 Tiempo: 1.7  
## 6 : 21 Tiempo: 1.7  
## 7 : 28 Tiempo: 1.7  
## 8 : 36 Tiempo: 1.7  
## 9 : 45 Tiempo: 1.7  
## 10 : 55 Tiempo: 1.7  
## 11 : 66 Tiempo: 1.7  
## 12 : 78 Tiempo: 1.7  
## 13 : 91 Tiempo: 1.7  
## 14 : 105 Tiempo: 1.7  
## 15 : 120 Tiempo: 1.7  
## 16 : 136 Tiempo: 1.7  
## 17 : 153 Tiempo: 1.7  
## 18 : 171 Tiempo: 1.7  
## 19 : 190 Tiempo: 1.7  
## 20 : 210 Tiempo: 1.7  
## 21 : 231 Tiempo: 1.7  
## 22 : 253 Tiempo: 1.7  
## 23 : 276 Tiempo: 1.7  
## 24 : 300 Tiempo: 1.7  
## 25 : 325 Tiempo: 1.7

1. Para poder ver la demostracion de f(n)=n(n+1)/2 ver en el link:

<https://docs.google.com/document/d/1LxlYIbw-R4bNqshdZ8uqnlKm8QlkTP3Hh6rspbti6WI/edit?usp=sharing>

## 2. Numeros Aitken

Primero se evalua la sucesion para de e:

# Remueve todos los objetos creados  
rm(list=ls())  
# Halla la raiz de Fx  
Fx <- function(x) 1/factorial(x)  
convergencia <- function() {  
 suma<-0  
 i<-0  
 error<-1  
 while (error > 1.e-5) {  
 y<-suma  
 suma<-suma+Fx(i)  
 i<-i+1  
 error<-abs(y-suma)  
 cat("X=",suma,"\tE=",error,"\n")  
 }  
}  
convergencia()

## X= 1 E= 1   
## X= 2 E= 1   
## X= 2.5 E= 0.5   
## X= 2.666667 E= 0.1666667   
## X= 2.708333 E= 0.04166667   
## X= 2.716667 E= 0.008333333   
## X= 2.718056 E= 0.001388889   
## X= 2.718254 E= 0.0001984127   
## X= 2.718279 E= 2.480159e-05   
## X= 2.718282 E= 2.755732e-06

Entonces el valor de euler evaluado en uno es igual a 2.71828, para hallar el orden de divergencia observamos con el siguiente algoritmo como la constante de convergencia se se acerca cada vez mas a uno, lo que concuerda con el limite encontado.

lim ax- ∞(1/(n+1 )/(1/n)^ p lim ax->∞ (n^p)/(n+1) Si p=1 el limite converge a 1.

# Remueve todos los objetos creados  
rm(list=ls())  
# Halla la raiz de Fx  
Fx <- function(x) 1/factorial(x)  
Oconvergencia <- function() {  
 suma<-0  
 i<-0  
 Asintotico<-0  
 error<-1  
 while (error > 1.e-7) {  
 y<-suma  
 suma<-suma+Fx(i)  
 Asintotico<-abs(suma-2.71828)/(abs(y-2.71828))  
 i<-i+1  
 error<-abs(y-suma)  
 cat("X=",suma,"\tE=",error,"\n")  
 cat("Cte error asintotico",Asintotico,"\n")  
 cat("\n")  
 }  
}  
Oconvergencia()

## X= 1 E= 1   
## Cte error asintotico 0.6321203   
##   
## X= 2 E= 1   
## Cte error asintotico 0.4180227   
##   
## X= 2.5 E= 0.5   
## Cte error asintotico 0.3038926   
##   
## X= 2.666667 E= 0.1666667   
## Cte error asintotico 0.2364547   
##   
## X= 2.708333 E= 0.04166667   
## Cte error asintotico 0.1927151   
##   
## X= 2.716667 E= 0.008333333   
## Cte error asintotico 0.1621984   
##   
## X= 2.718056 E= 0.001388889   
## Cte error asintotico 0.1391185   
##   
## X= 2.718254 E= 0.0001984127   
## Cte error asintotico 0.115983   
##   
## X= 2.718279 E= 2.480159e-05   
## Cte error asintotico 0.0472561   
##   
## X= 2.718282 E= 2.755732e-06   
## Cte error asintotico 1.240143   
##   
## X= 2.718282 E= 2.755732e-07   
## Cte error asintotico 1.180636   
##   
## X= 2.718282 E= 2.505211e-08   
## Cte error asintotico 1.013909

Entonces se puede aplicar la aceleracion con aitken

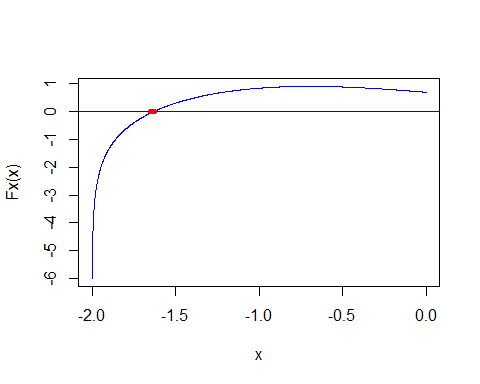
# Remueve todos los objetos creados  
  
# Remueve todos los objetos creados  
rm(list=ls())  
# Halla la raiz de Fx  
Fx <- function(x) 1/factorial(x)  
Aitken <- function() {  
 suma<-0  
 i<-0  
 res<-0  
 error<-1  
 cat("S1","\n")  
 while (i<=7) {  
 y<-suma  
 suma<-suma+Fx(i)  
 i<-i+1  
 res[c(i)]<-suma  
 error<-abs(y-suma)  
 cat("X=",suma,"\tE=",error,"\n")  
 }  
 aux<-0  
 i<-5  
 j<-0  
 y<-0  
 k<-2  
 while (i >0) {  
 cat("S",k,"\n")  
 while (j <i) {  
 j<-j+1  
 a<-res[c(j)]  
 b<-res[c(j+1)]  
 c<-res[c(j+2)]  
 aux[c(j)]<-(a-(((b-a)^2)/(c-2\*b+a)))  
 error<-abs(y-aux[c(j)])  
 y<-aux[c(j)]  
 cat("X=",aux[c(j)],"\tE=",error,"\n")  
 }  
 res<-aux  
 j<-0  
 i<-i-2  
 k<-k+1  
 }  
}  
Aitken()

## S1   
## X= 1 E= 1   
## X= 2 E= 1   
## X= 2.5 E= 0.5   
## X= 2.666667 E= 0.1666667   
## X= 2.708333 E= 0.04166667   
## X= 2.716667 E= 0.008333333   
## X= 2.718056 E= 0.001388889   
## X= 2.718254 E= 0.0001984127   
## S 2   
## X= 3 E= 3   
## X= 2.75 E= 0.25   
## X= 2.722222 E= 0.02777778   
## X= 2.71875 E= 0.003472222   
## X= 2.718333 E= 0.0004166667   
## S 3   
## X= 2.71875 E= 0.0004166667   
## X= 2.718254 E= 0.0004960317   
## X= 2.718277 E= 2.25469e-05   
## S 4   
## X= 2.718276 E= 9.802999e-07

## 3 Funcion imposible

1. Con base en la grafica se ubica un intervalo de existencia y se acomodan las cotas para no caer en el pronunciado desnivel:

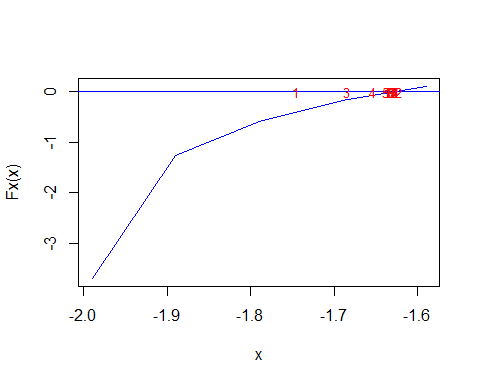
# Metodo de secante  
# Remueve todos los objetos creados  
rm(list=ls())  
Fx <- function(x) log(x+2)-sin(x)  
F1x <- function(x) (1/(x+2))-cos(x)  
# Halla la raiz de Fx  
secante <- function(x0,x1) {  
 x<-seq(-2,0,0.001)  
 plot(x,Fx(x),type="l",col="blue")  
 abline(h=0,col="blue")  
 x<-(Fx(x1)\*x0-Fx(x0)\*x1)/(Fx(x1)-Fx(x0))  
 error <-1  
 while (error > 1.e-7) {  
 x0<-x1  
 x1<-x  
 x<-(Fx(x1)\*x0-Fx(x0)\*x1)/(Fx(x1)-Fx(x0))  
 if (Fx(x) == 0) break  
 error<-abs(Fx(x)/F1x(x))  
 points(rbind(c(x,0)),pch=19,cex=0.7,col="red")  
 cat("X=",x,"\t","E=",error,"\n")  
 }  
}  
secante(-1.99,-1.5)



## X= -1.650187 E= 0.01821171   
## X= -1.628789 E= 0.002665268   
## X= -1.631368 E= 7.510581e-05   
## X= -1.631444 E= 3.003488e-07   
## X= -1.631444 E= 3.398911e-11

1. Se usa el metodo Newton, con el intervalo de existencia previo usado en el problema de la secante. Sin embargo no es posible hallar ninguna aproximacion deseada , por lo que para saber si existe una raiz se usa el metodo de biseccion por siempre garantizar la convergencia y para confirmar el valor ya hallado.

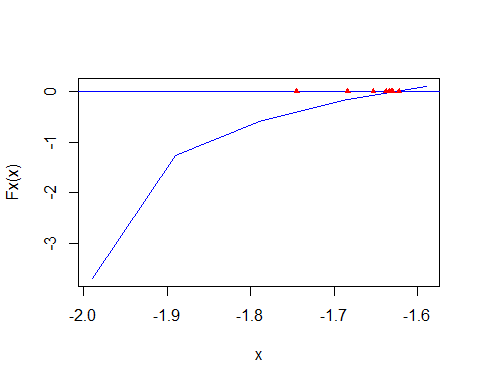
# Remueve todos los objetos creados  
rm(list=ls())  
Fx <- function(x) log(x+2)-sin(x)  
# Halla la raiz de Fx  
biseccion <- function(a,b) {  
 x<-seq(a,b,00.1)  
 plot(x,Fx(x),type="l",col="blue")  
 abline(h=0,col="blue")  
 x<-b  
 d<-(a+b)/2  
 i<-0  
 error<-abs(a-b)/2  
 while (error > 1.e-5) {  
 i<-i+1  
 if (Fx(x) == 0) break  
 if (Fx(x)\*Fx(a) < 0) b <- x else {a <- x}  
 d<-x  
 x<-(a+b)/2  
 #points(rbind(c(x,0)),pch=17,cex=0.7,col="red")  
 text(x,0,i,cex=0.8,col="red")  
 error<-abs(a-b)/2  
 cat("X=",x,"\tE=",error,"\n")  
 }  
}  
biseccion(-1.99,-1.5)



## X= -1.745 E= 0.245   
## X= -1.6225 E= 0.1225   
## X= -1.68375 E= 0.06125   
## X= -1.653125 E= 0.030625   
## X= -1.637813 E= 0.0153125   
## X= -1.630156 E= 0.00765625   
## X= -1.633984 E= 0.003828125   
## X= -1.63207 E= 0.001914063   
## X= -1.631113 E= 0.0009570313   
## X= -1.631592 E= 0.0004785156   
## X= -1.631353 E= 0.0002392578   
## X= -1.631472 E= 0.0001196289   
## X= -1.631412 E= 5.981445e-05   
## X= -1.631442 E= 2.990723e-05   
## X= -1.631457 E= 1.495361e-05   
## X= -1.63145 E= 7.476807e-06

Ahora antes aplicar en los algoritmos, se corrobora el resultado, modificando el codigo para evaluar si la convergencia se acerca a 0:

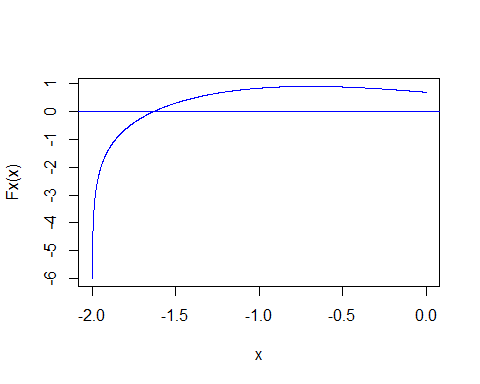
# Remueve todos los objetos creados  
rm(list=ls())  
Fx <- function(x) log(x+2)-sin(x)  
F1x <- function(x) (1/(x+2))-cos(x)  
# Halla la raiz de Fx  
biseccion <- function(a,b) {  
 x<-seq(a,b,00.1)  
 plot(x,Fx(x),type="l",col="blue")  
 abline(h=0,col="blue")  
 x<-b  
 d<-(a+b)/2  
 i<-0  
 error<-abs(a-b)/2  
 while (error > 1.e-5) {  
 i<-i+1  
 if (Fx(x) == 0) break  
 if (Fx(x)\*Fx(a) < 0) b <- x else {a <- x}  
 d<-x  
 x<-(a+b)/2  
 points(rbind(c(x,0)),pch=17,cex=0.7,col="red")  
 #text(x,0,i,cex=0.8,col="red")  
 error<-abs(a-b)/2  
 cat("X=",x,"\tE=",error,"\n")  
 cat("Evaluacion ",Fx(x),"\n")  
 cat("Evaluacion Derivada ",F1x(x),"\n","\n")  
 }  
}  
biseccion(-1.99,-1.5)



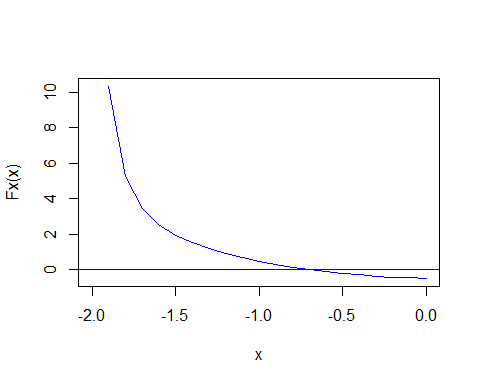
## X= -1.745 E= 0.245   
## Evaluacion -0.3816269   
## Evaluacion Derivada 4.094893   
##   
## X= -1.6225 E= 0.1225   
## Evaluacion 0.02447895   
## Evaluacion Derivada 2.700687   
##   
## X= -1.68375 E= 0.06125   
## Evaluacion -0.1575947   
## Evaluacion Derivada 3.274769   
##   
## X= -1.653125 E= 0.030625   
## Evaluacion -0.06217789   
## Evaluacion Derivada 2.965119   
##   
## X= -1.637813 E= 0.0153125   
## Evaluacion -0.01783799   
## Evaluacion Derivada 2.827967   
##   
## X= -1.630156 E= 0.00765625   
## Evaluacion 0.003564057   
## Evaluacion Derivada 2.76317   
##   
## X= -1.633984 E= 0.003828125   
## Evaluacion -0.007074956   
## Evaluacion Derivada 2.79527   
##   
## X= -1.63207 E= 0.001914063   
## Evaluacion -0.001740089   
## Evaluacion Derivada 2.779146   
##   
## X= -1.631113 E= 0.0009570313   
## Evaluacion 0.0009158066   
## Evaluacion Derivada 2.77114   
##   
## X= -1.631592 E= 0.0004785156   
## Evaluacion -0.0004111834   
## Evaluacion Derivada 2.775138   
##   
## X= -1.631353 E= 0.0002392578   
## Evaluacion 0.0002525508   
## Evaluacion Derivada 2.773138   
##   
## X= -1.631472 E= 0.0001196289   
## Evaluacion -7.925648e-05   
## Evaluacion Derivada 2.774138   
##   
## X= -1.631412 E= 5.981445e-05   
## Evaluacion 8.66621e-05   
## Evaluacion Derivada 2.773638   
##   
## X= -1.631442 E= 2.990723e-05   
## Evaluacion 3.70655e-06   
## Evaluacion Derivada 2.773888   
##   
## X= -1.631457 E= 1.495361e-05   
## Evaluacion -3.777403e-05   
## Evaluacion Derivada 2.774013   
##   
## X= -1.63145 E= 7.476807e-06   
## Evaluacion -1.703351e-05   
## Evaluacion Derivada 2.77395   
##

Dado codigo concluimos que la raiz es valida puesto que se acerca cada vez mas a 0 y a su vez que la derivada de la funcion evaluada en la misma es diferente de 0 por lo que cumple con el intervalo local de convergencia de Newton. Sin embargo al graficar las funciones: f(x)

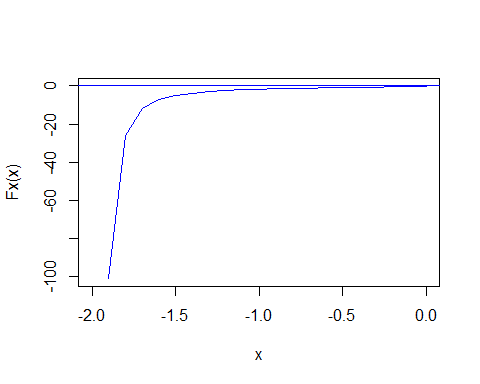
Fx <- function(x) log(x+2)-sin(x)  
x<-seq(-2,0,0.001)  
plot(x,Fx(x),type="l",col="blue")  
abline(h=0,col="blue")

 f`(x)

Fx <- function(x) (1/(x+2))-cos(x)  
x<-seq(-2,0,0.1)  
plot(x,Fx(x),type="l",col="blue")  
abline(h=0,col="blue")

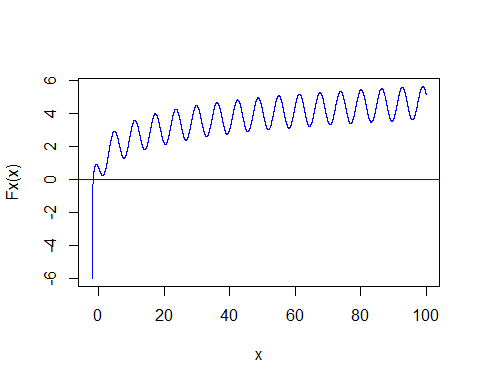
 f``(x)

Fx <- function(x) (-1/(x+2)^2)+sin(x)  
x<-seq(-2,0,0.1)  
plot(x,Fx(x),type="l",col="blue")  
abline(h=0,col="blue")

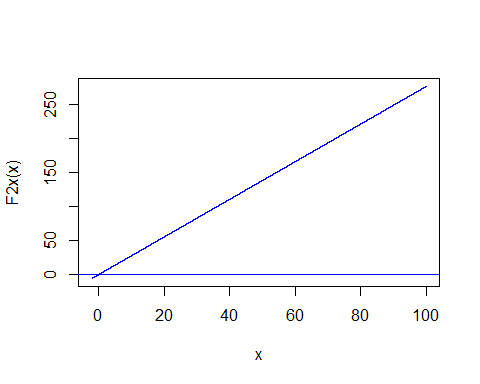
 Es posible verificar que la tercera derivada de la formula es menor a 0 en el intervalo, lo tanto viola el teorema de convergencia y no es posible garantizar que un x0 pueda ser usado para hallar la raiz mediante el metodo de newton, mucho menos el punto x0=1.

Si se grafica en un rango considerable se obtiene:

# Metodo de Newton  
rm(list=ls())  
Fx <- function(x) log(x+2)-sin(x)  
F1x <- function(x) (1/(x+2))-cos(x)  
# Metodo de Newton  
# Halla la raiz de   
 x<-seq(-2,100,0.001)  
 plot(x,Fx(x),type="l",col="blue")  
 abline(h=0,col="blue")

 Si se compara con la derivada evaluada en la raiz previamente hallada obtenemos:

rm(list=ls())  
Fx <- function(x) log(x+2)-sin(x)  
F1x <- function(x) (1/(x+2))-cos(x)  
 a<-F1x(-1.6314)  
F2x <- function(x) a\*x  
 x<-seq(-2,100,0.001)  
 plot(x,F2x(x),type="l",col="blue")  
 abline(h=0,col="blue")

 Como se puede ver para que esta derivada corte con Fx se necesita de un punto indeterminado que no se puede hallar de forma aleatoria. Sin embargo teniendo en cuenta que ya conocemos la raiz se puede buscar los puntos de interseccion para poder aplicar el metodo de Newton. El intervalo de existencia se ubica mediante graficas pero dado que la funcion para encontrar la interseccion tambien tiene derivadas pronunciadas y el ln tiende a las indeterminaciones(graficaria pero el mismo programa no puede ubicarlo), entonces para hallar el punto mas cercano se usa una estrategia poco recomendable. Con el siguiente codigo (que estara comentado porque el programa hace que Rstudio colapse ) se busca cual es el ultimo punto que Rstudio puede procesar imprimiendolos uno a uno:

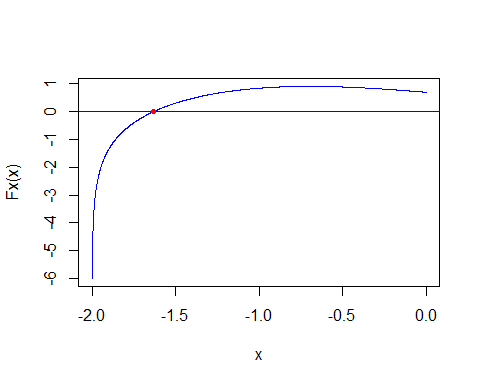
rm(list=ls()) Fx <- function(x) 2.773534\*x-log(x+2)+sin(x) x<-seq(-2.5,-1.5,0.001) i<-1 while(1) { cat(x[c(i)]) i<-i+1 }

#plot(x,Fx(x),type=“l”,col=“blue”) #abline(h=0,col=“blue”)

Se recomienda no correr el codigo. Sin embargo arroja un punto de interseccion igual a -1.501

Se procede a usar el algoritmo con el numero que sabemos esta muy cerca de la interseccion de la pendiente con la funcion:

# Metodo de Newton  
rm(list=ls())  
Fx <- function(x) log(x+2)-sin(x)  
F1x <- function(x) (1/(x+2))-cos(x)  
# Metodo de Newton  
# Halla la raiz de   
Newton <- function(x0) {  
 x<-seq(-2,0,0.001)  
 plot(x,Fx(x),type="l",col="blue")  
 abline(h=0,col="blue")  
 x<-x0-(Fx(x0)/F1x(x0))  
 error <-1  
 while (error > 1.e-5) {  
 x<-x-(Fx(x)/F1x(x))  
 if (Fx(x) == 0) break  
 error<-abs(Fx(x)/F1x(x))  
 points(rbind(c(x,0)),pch=19,cex=0.7,col="red")  
 cat("X=",x,"\t","E=",error,"\n")  
 }  
}  
Newton(-1.501)



## X= -1.63246 E= 0.001015286   
## X= -1.631445 E= 1.558379e-06

#-1.998423657