Taller Repaso
Analisis Numerico

Por:
Nicolai Barrera
Diego Burgos

Para:
Profesora Eddy Herrera Daza

---

1. Numero de Operaciones

1.b.Implemente en R o Python para verificar los resultados del metodo de Horner

En este ejericicio implementamos el metodo de hornner probandolo asi con un polinomio de 5 terminos y una X= 5 y asi lo evaluamos buscando el resultado 781 deseado

```
Hornner <- function(x,cantidadCoe){
  total<-0
  for(i in 1:length(cantidadCoe))
  {
    total<-total * x + cantidadCoe[i] #mutiplica los coeficientes hasta que i llegue al tamaño del array
  }

  cat("el total mediante las iteraciones es:",total)

}
c<-c(1,1,1,1,1)
Hornner(5 ,c)
```

```
## el total mediante las iteraciones es: 781
```

1.c. Evaluar en x=1.0001 con $P(x)=1+x+x^{2+...+x}50$. Encuentre el error del calculo al compararlo con la expesion equivalente Q(x)=(x^51-1)/(x-1)

```
Horner <- function(x,cantidadCoe){
  totalIterativo<-0

  totalFormula <- ((x^51)-1)/(x-1) #calcula el valor mediante la formula

  for(i in 1:length(cantidadCoe))
  {
    totalIterativo<-totalIterativo * x + cantidadCoe[i] #mutiplica los coeficientes hasta que i llegue al ta
maño del array
  }

  error<-totalIterativo-totalFormula #calcula el error hallando la diferencia

  cat("el total mediante las iteraciones es:",totalIterativo," y el resultado mendiante la formula es :", to
talFormula,"\n el error que se produce en estas dos es:", error )

}
c<-c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1) #50 terminos como lo indica el problema
Horner(1.0001 ,c)
```

```
## el total mediante las iteraciones es: 51.12771  y el resultado mendiante la formula es : 51.12771
##  el error que se produce en estas dos es: 7.81597e-14
```

1.d. Encuentre los primeros 15 bits en la representacion binaria de π

```
imprimir <-function(arra2){

  contador<-length(arra2)
  while (contador!=0){
    cat(arra2[contador])
    contador=contador-1
  }
}
codificar <- function(n,arra){

  int<-trunc(n)
 float<- (n-int)

cat("el numero",n, "en binario es:")

#CODIFICA LA PARTE ENTERA
 copyN<-int
 cont<-1
 arra2<-c()
 while(copyN>0){
   if((copyN%%2)==0)
   {
     arra2[cont]<-0
   }
   if((copyN%%2)==1){
      arra2[cont]<-1


   }
   cont<-cont+1
   copyN<-(copyN%/%2)
 }
  if(length(arra2)%%4!=0){

    while (length(arra2)%%4!=0) {
      arra2[cont]<-0
    cont<-cont+1
      }
  }

  imprimir(arra2)
  cat(",")
  #CODIFICA LA PARTE FLOTANTE
 if(float!=0.0){
  contadorf<-0
  copiaf<-float
  arra3<-c()
  while(contadorf!=11){
    copiaf<- copiaf*2
    enterof<-trunc(copiaf)
    residuof<-copiaf-enterof
    arra3[contadorf]<-enterof
    copiaf<-residuof
    contadorf<-contadorf+1
  }
  imprimir(arra3)
 }
}


codificar(3.1415,arra)
```

```
## el numero 3.1415 en binario es:0011,1000010010
```

1.f. Convierta los siguientes nu´meros de base 10 a binaria: 11.25;3/2;30.6;99.9

```
codificar <- function(n,arra){

  int<-trunc(n)
 float<- (n-int)

cat("el numero",n, "en binario es:")

#CODIFICA LA PARTE ENTERA
 copyN<-int
 cont<-1
 arra2<-c()
 while(copyN>0){
   if((copyN%%2)==0)
   {
     arra2[cont]<-0
   }
   if((copyN%%2)==1){
      arra2[cont]<-1


   }
   cont<-cont+1
   copyN<-(copyN%/%2)
 }
  if(length(arra2)%%4!=0){

    while (length(arra2)%%4!=0) {
      arra2[cont]<-0
    cont<-cont+1
      }
  }

  imprimir(arra2)
  cat(",")
  #CODIFICA LA PARTE FLOTANTE
 if(float!=0.0){
  contadorf<-0
  copiaf<-float
  arra3<-c()
  while(contadorf!=11){
    copiaf<- copiaf*2
    enterof<-trunc(copiaf)
    residuof<-copiaf-enterof
    arra3[contadorf]<-enterof
    copiaf<-residuof
    contadorf<-contadorf+1
  }
  imprimir(arra3)
 }
  cat("\n")
}


codificar(11.25,arra)
```

```
## el numero 11.25 en binario es:1011,0000000001
```

```
codificar(1.5,arra)
```

```
## el numero 1.5 en binario es:0001,0000000000
```

```
codificar(30.6,arra)
```

```
## el numero 30.6 en binario es:00011110,0011001100
```

```
codificar(99.9,arra)
```

```
## el numero 99.9 en binario es:01100011,1100110011
```

1.g. ¿ Como se ajusta un numero binario infinito en un numero finito de bits?

Un numero binario infinito se ajusta mediante el metodo de truncamineto de bits lo que hace es recortar a un numero “n” de bits con el cual el numero sera representado, pero pierde bastante valor y a su vez se propaga el error en los calculos proximos que se haran con el numero.

1.h. ¿Cual es la diferencia entre redondeo y recorte?

La principal diferencia entre el redondeo y el recorte es que; Si queremos aproximar un numero por el metodo de REDONDEO a n cifras para esta aproximación se tiene en cuenta la cifra n+1, en este caso si es igual o mayor a 5 se incrementa la cifra numero n. a diferencia del RECORTE que solo tiene en cuenta las n cifras que la persona quiere dejar y no importa cual sea la cifra n+1.

Esto tiene una diferencia muy grande ya que con el redondeo estamos incluyendo mas precision a nuestro numero y en el recorte podemos perder valor importante por ello debemos conocer muy bien sus diferencias y en que casos implementar cada tecnica, segun sea el caso

1.i. Idique el numero de punto flotante (IEEE) de precision doble asociado a x, el cual se denota como fl(x); para x(0.4) 4*10^1 1.k. Verificar si el tipo de datos basico de R y Python es de precision doble IEEE y Revisar en R y Phython el format long

Si el dato basico de r es de precision doble de IEEE se puede configuarar para que de la cantidad de decimales que el usuario prefiera dejear en la mantisa asi modificando a su ves el exponente.

El format long es un fromato de dato que permite almacenar de -2147483648 a 2147483647, esto le permite al usuario realizar operaciones con alta precision a la hora de no perder decimales.

```
                              2. Raices de una Ecuación
```

2.a. Implemente en R o Python un algoritmo que le permita sumar unicamente los elementos de la sub matriz triangular superior o triangular inferior, dada la matriz cuadrada An. Imprima varias pruebas, para diferentes valores de n y exprese f(n) en notacion O() con una grafica que muestre su orden de convergencia

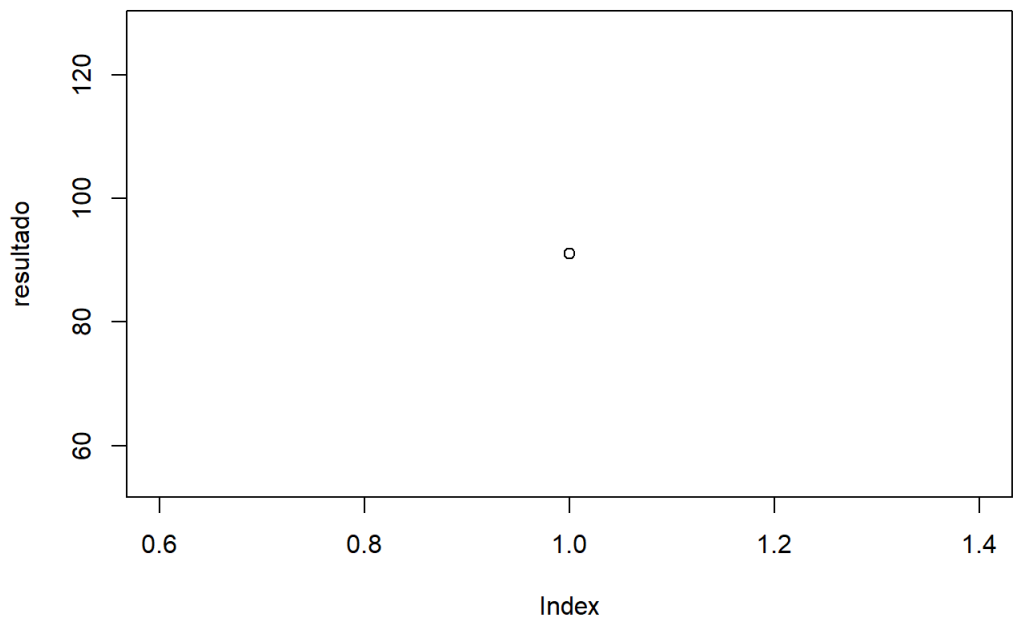
```
a1<-c(1,2,3)
a2<-c(5,6,7)
a3<-c(9,10,11)
a<-c(a1,a2,a3)

m<-matrix(a,ncol=3,byrow=F)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
```

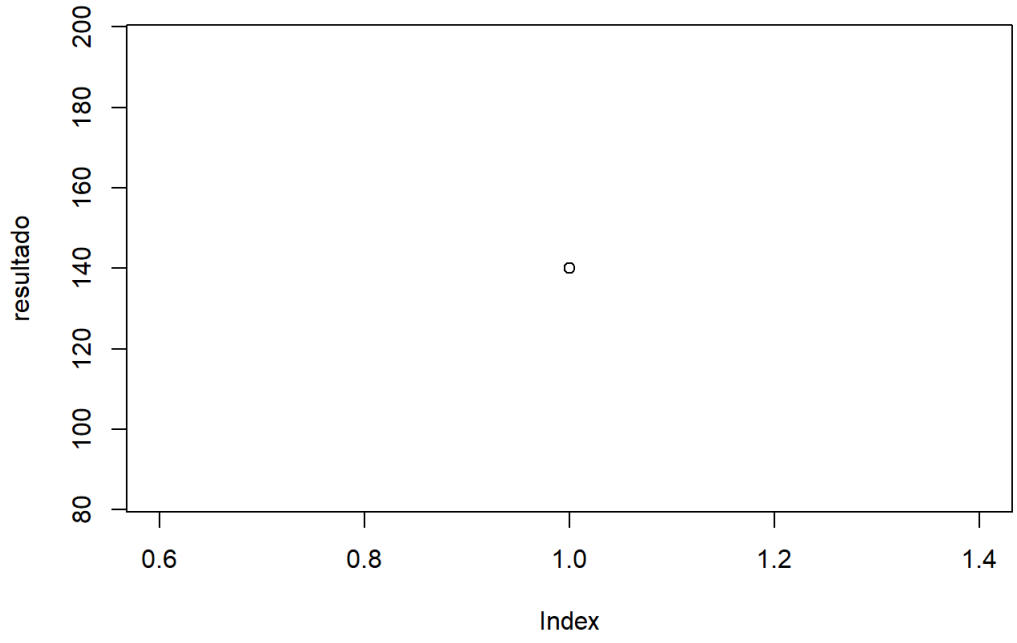
```
c1<-c(0,0,0)
c2<-c(0,0,0)
c3<-c(0,0,0)
c<-c(c1,c2,c3)


m<-matrix(a,ncol=3,byrow=F)

Q<-matrix(c,ncol=3,byrow=F)

R<-matrix(c,ncol=3,byrow=F)

qr <- function (m, Q, R)
{
  V <- m
  n <- nrow(m)
  cols <- ncol(m)
  for ( i in 1:n){
    R[i,i]<- 0
    for ( j in 1:n ) {
      R[i,i] = R[i,i] + (V[j,i])^2

    }
    R[i,i]<-sqrt(R[i,i])
    for ( j in 1:n) {
      Q[j,i]<- V[j,i]/R[i,i]
    }
    z<-i+1
    while (z<= n) {
      R[i,z]<- 0
      for(k in 1 : cols ) {
        R[i,z]<- R[i,z] + Q[k,i] %*%V[k,z]
      }
      for(k in 1:cols) {
        V[k,z]<- V[k,z] - R[i,z]%*%Q[k,i]
      }
      z <- z + 1
    }
  }
  return ( R )
}

sumatoria<- function (total)
{
  w<-total
  filas <- nrow(total)
  columnas <- ncol(total)
  resultado <-0
  for(i in 1:filas)
  {
    for ( j in 1:columnas)
    {
      resultado = w[j,i] + resultado
    }
  }
  return(resultado)
}

total<-qr(m,Q,R)

res<-sumatoria(total)
```

2.b.Implemente en R o Python un algoritmo que le permita sumar los n^2 primeros numeros naturales al cuadrado. Imprima varias pruebas, para diferentes valores de n y exprese f(n) en notacion O() con una grafica que muestre su orden de convergencia.

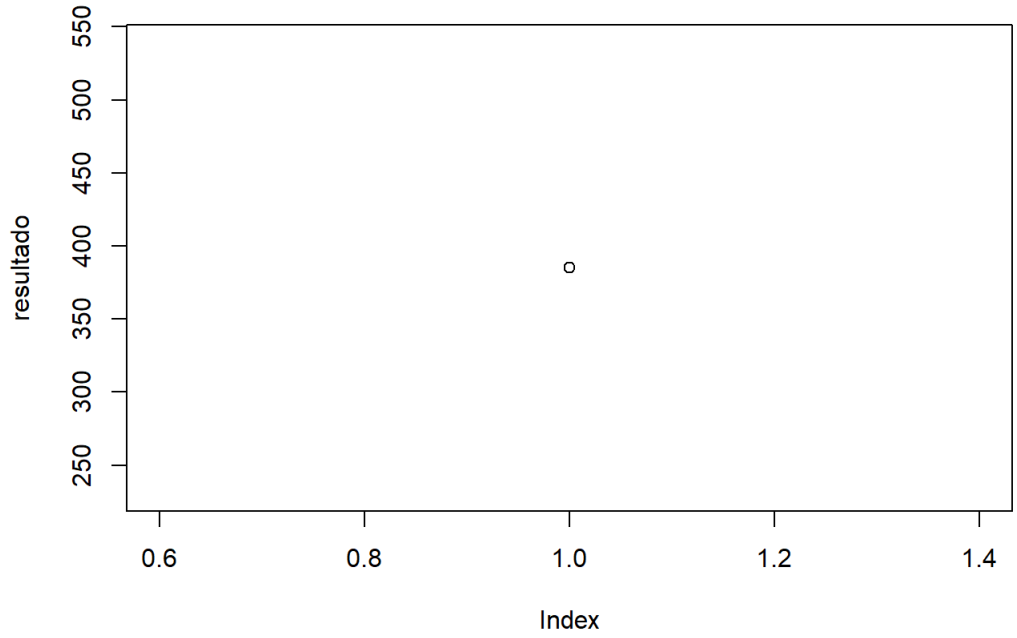
```
puntob<- function(u)
{
  resultado = (u*(u+1)*(2*(u)+1))/6
  plot(resultado)
  return (resultado)

}

v<-puntob(6)
```



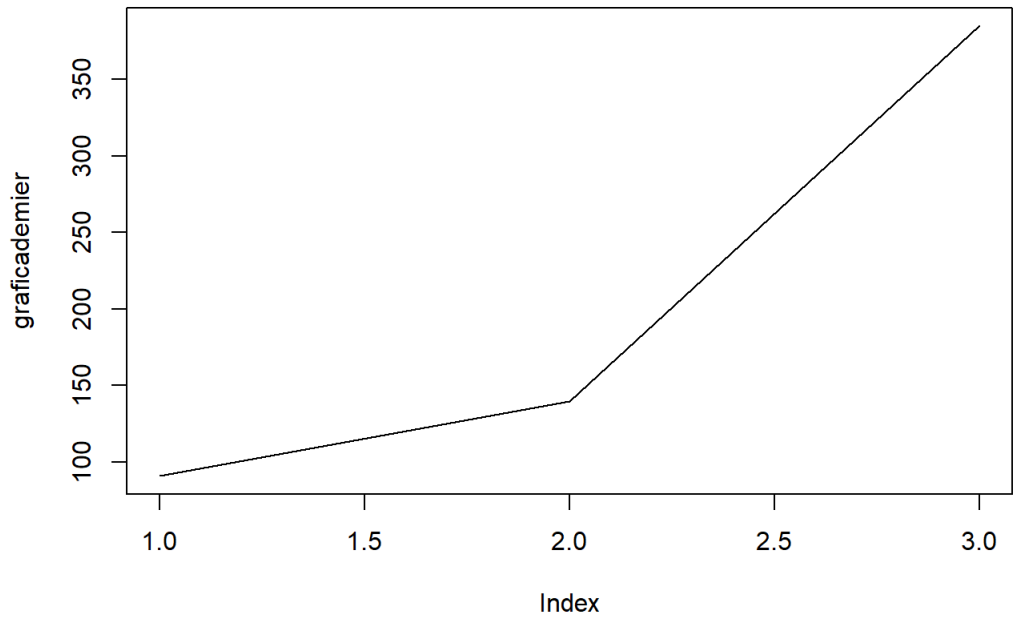

```
x<-puntob(7)
```




```
y<-puntob(10)
```




```
graficademier=c(v,x,y)
plot(graficademier,type="l")
```



2.c.Para describir la trayectoria de un cohete se tiene el modelo: $y(t)=6+2.13t^{2-0.0013t}4$ Donde, y es la altura en [m] y t tiempo en [s]. El cohete esta colocado verticalmente sobre la tierra. Utilizando dos metodos de solucion de ecuacion no lineal, encuentre la altura maxima que alcanza el cohete

```
library(boot)

expresion1=expression(6+(2.13*(t^2))-(0.0013*(t^4)))

a<-D(expresion1,"t")
a
```

```
## 2.13 * (2 * t) - 0.0013 * (4 * t^3)
```

```r
#hallamos el valor del tiempo en la altura maxima con la derivada despejando
t<- function (a)
{
  Resultado<- sqrt(4.26/0.0052)
  return (Resultado)
}

z<-t(a)
z
```

```
## [1] 28.62221
```

```r
p<- function(x) {

  return (6+(2.13*(x^2))-(0.0013*(x^4)))

}
#Utilizando el resultado de la derivada anterior
p(z)
```

```
## [1] 878.4808
```

---

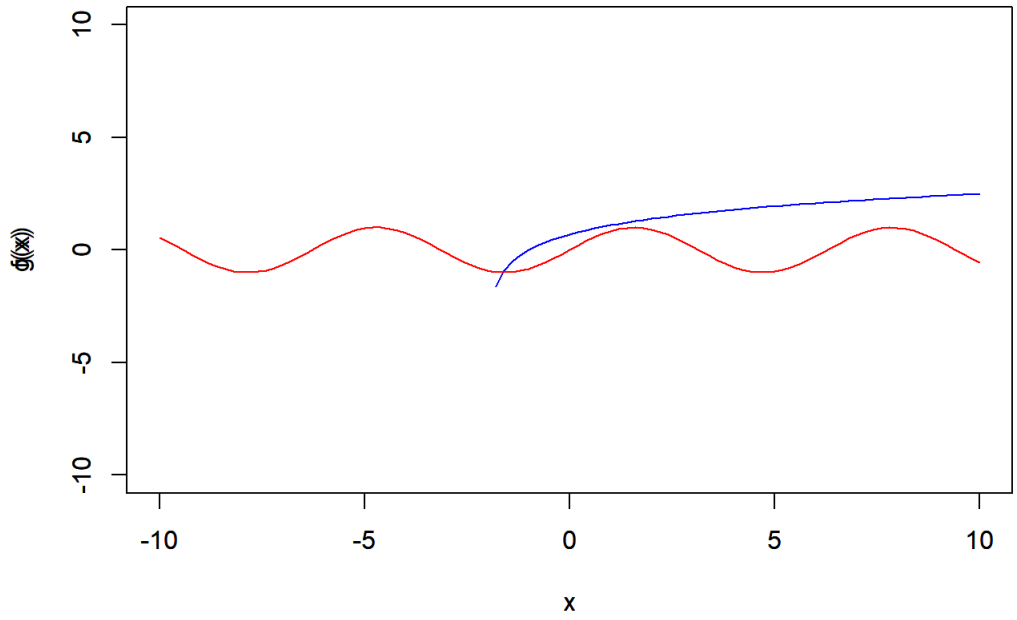
3. Convergencia de Metodos Iterativos

Parte A

```r
logaritmo<- function(x){
  log (x+2)
}
sen<- function(x)
{
  sin(x)
}
tol = 0.00000001

func<- function(x)
{

  log(x+2)-sin(x)
}
metodo = uniroot(func, c(-1.8,-1), tol = 1e-9)
plot(logaritmo, xlim = c(-10,10), ylim=c(-10,10), ylab = "f(x)", col = "blue")
```

```
## Warning in log(x + 2): Se han producido NaNs
```

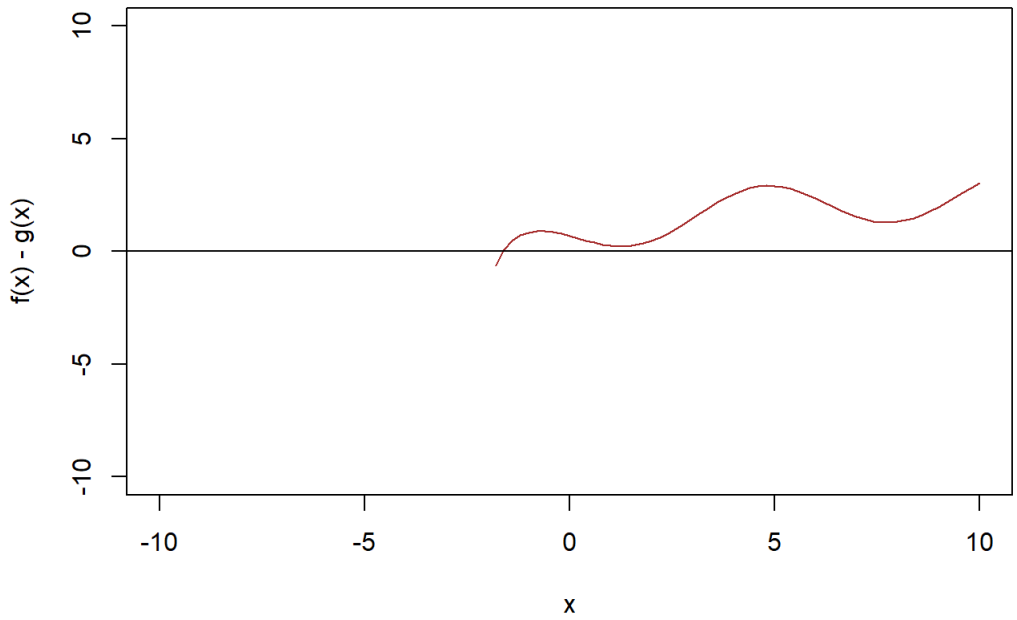
```r
par(new = TRUE)
plot(sen, xlim = c(-10,10), ylim=c(-10,10), ylab = "g(x)", col = "red")
```



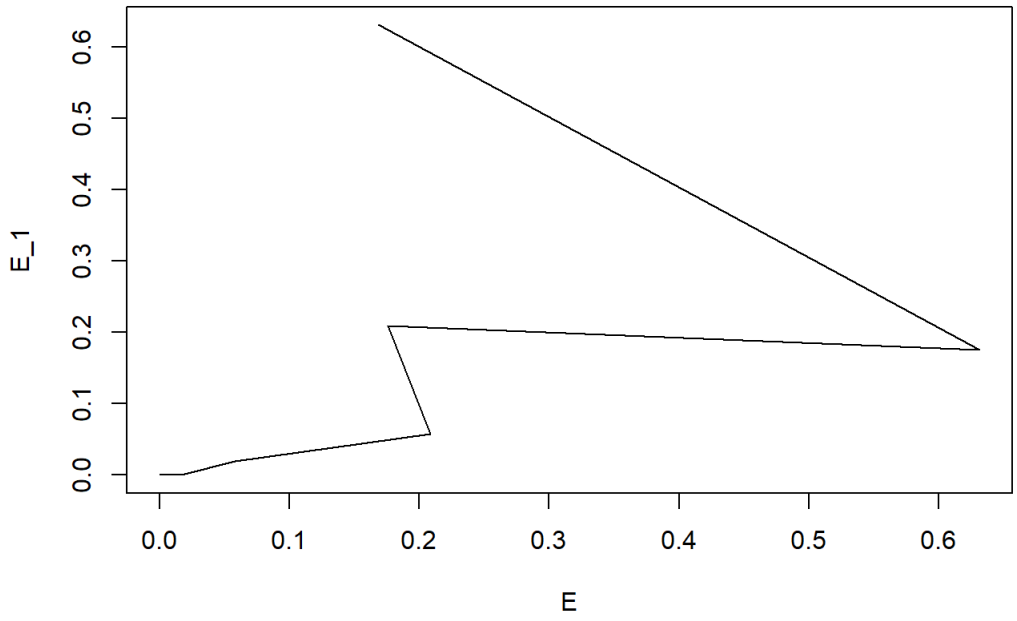

```
plot(func, xlim = c(-10,10), ylim=c(-10,10), ylab = "f(x) - g(x)", col = "brown")
```

```
## Warning in log(x + 2): Se han producido NaNs
```

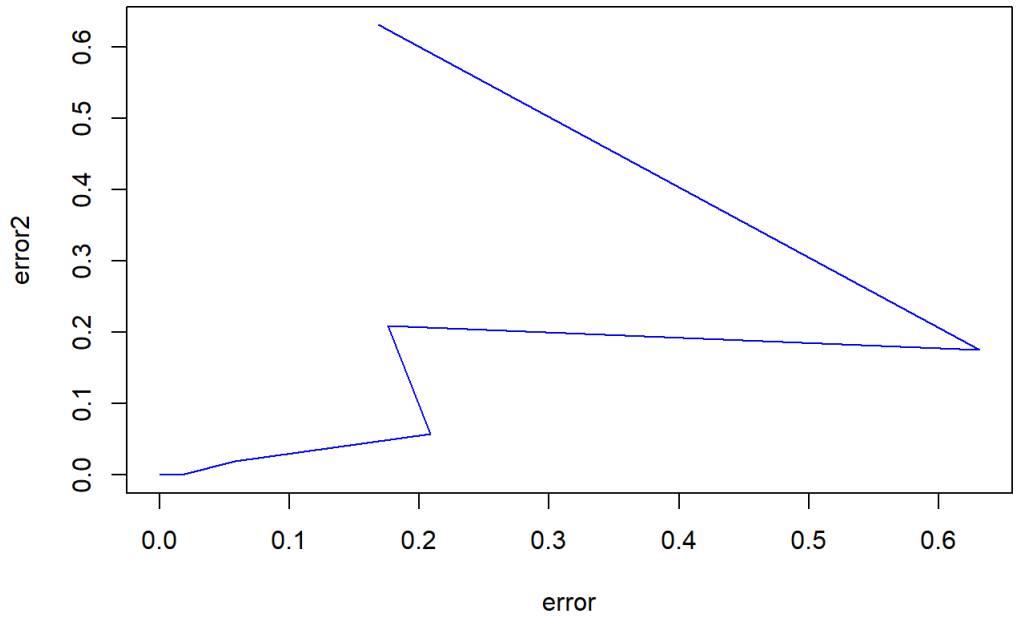
```
par(new = TRUE)
abline(0,0, col = "black")
```



```
a = -1.8
b = -1
arreglo = c(0)
arreglo[1] = a
arreglo[2] = b
i=3
E = c()
E_1 = c()
while(abs(metodo$root-arreglo[i-2]) > tol){
  E[i-2]=abs(metodo$root-arreglo[i-2])
  arreglo[i]= arreglo[i-1]-((func(arreglo[i-1])*(arreglo[i-1]-arreglo[i-2]))/(func(arreglo[i-1])-func(arreglo[i-2])))
  i = i+1
}

i=1
while(i<length(E)+1){
  E_1[i]=E[i+1]
  i = i+1
}
plot(E,E_1, type="l")
```




```
numero_iteraciones =length(arreglo)
Met = c(0)
error = c(0)
error2 = c(0)
Met[1] = a
Met[2] = b
i=3
while(abs(metodo$root-Met[i-2]) > tol){
  error[i-2]=abs(metodo$root-arreglo[i-2])
  Met[i]= Met[i-1]-((func(Met[i-1]))*(Met[i-1]-Met[i-2])/(func(Met[i-1])-func(Met[i-2])))
  i = i+1
}
i=1
while(i<length(error)+1){
  error2[i]=error[i+1]
  i = i+1
}
plot(error,error2, type ="l",col ="blue")
```



3.d. Utilizando el metodo de Newton con p0 = 1 verifique que converge a cero pero no de forma cuadratica

Parte B

```
metodo<- function(x)
{
  #Utilizamos el valor del numero de euler
  a<-exp(1)
  resultadoA <- (a^x)-x-1
  return(resultadoA)
}

comprobacion<-function(y,multiplicidad)
{
  #Hallamos la raiz del la ecuacion
  p<-metodo(0)

  resultado = (y-p)^multiplicidad


  if(resultado != p)
  {
    cat("para F(",y,")","decimos que es un cero de multiplicidad ", multiplicidad)
  }
  else
  {

    cat("para F(",y,")","decimos que no es un cero de multiplicidad", multiplicidad)
  }

}
metodo(0)
```

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

```
comprobacion(0,2)
```

```
## para F( 0 ) decimos que no es un cero de multiplicidad 2
```

```
#B

#Metodo de newton


newton1 = function(f, fp, x0, tol, maxiter){
  k = 0
  # Imprimir estado
  cat("--------------------------------------------------------------------------------\n")
  cat(formatC( c("x_k"," f(x_k)","Error est."), width = -20, format = "f", flag = " "), "\n")
  cat("--------------------------------------------------------------------------------\n")
  repeat{
    correccion = f(x0)/fp(x0)
    x1 = x0 - correccion
    dx = abs(x1-x0)
    # Imprimir iteraciones
    cat(formatC( c(x1 ,f(x1), dx), digits=15, width = -15, format = "f", flag = " "), "\n")
    x0 = x1
    k = k+1
    # until
    if(dx <= tol || k > maxiter ) break;
  }
  cat("--------------------------------------------------------------------------------\n")
  if(k > maxiter){
    cat("Se alcanzÃ³ el mÃ¡ximo nÃºmero de iteraciones.\n")
    cat("k = ", k, "Estado: x = ", x1, "Error estimado <= ", correccion)
  } else {
    cat("k = ", k, " x = ", x1, " f(x) = ", f(x1), " Error estimado <= ", correccion) }
}
#Prueba
x0=1;  tol=1.e-5 ; N=100
v_a = 0 ; v_b = 3
f=function(x) exp(x)-(pi*x)
fp = function(x) exp(x)-pi
newton1(f,fp, x0, tol, N)
```

```
## --------------------------------------------------------------------------------
## x_k                  f(x_k)              Error est.
## --------------------------------------------------------------------------------
##  0.000000000000000  1.000000000000000  1.000000000000000
##  0.466942206924260  0.128167007561179  0.466942206924260
##  0.549818624686229  0.005632524944027  0.082876417761969
##  0.553817140317387  0.000013871704625  0.003998515631158
##  0.553827036583731  0.000000000085199  0.000009896266344
## --------------------------------------------------------------------------------
## k =  5  x =  0.553827  f(x) =  8.519896e-11  Error estimado <=  -9.896266e-06
```

---

4. Convergencia Acelerada

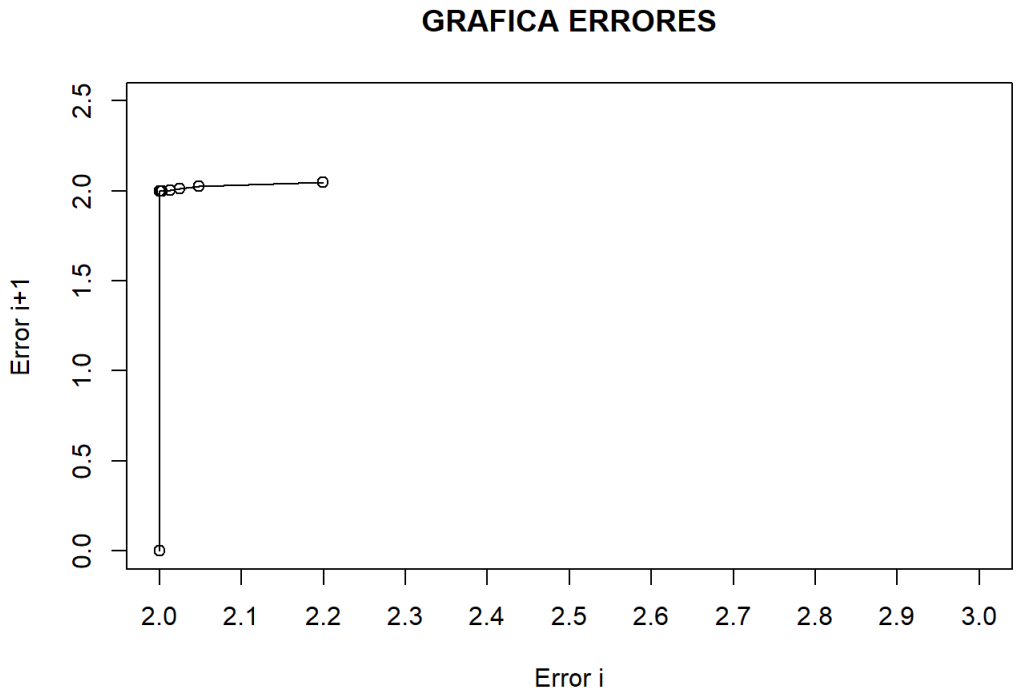
Dada la sucesión con Xn=cos(1/n) 4.a. Verifique el tipo de convergencia en x = 1 independiente del origen

```
funcion<-function(a,b,tol){
  error=0
  xa=a
  xb=b
  xmedio=(xa+xb)/2
  xanterior=0
  it=0
  bool=FALSE
  Errores<-(0)
  indice=0
  itMax=0
  v1=log(abs((xa-xb)/tol))
  v2=(1/log(2))
  ErroresSignificativo<-c(0)
  j=0
```

```
  itMax=v1*v2

  if(sign( cos(1/xa)  ) ==  sign( cos(1/xmedio) ) ){

    bool=TRUE
  }

  if(bool==TRUE){

    cat("Son iguales los signos\n")
  }
  if(bool==FALSE){

    if(cos(1/xa)   *  cos(1/xmedio)  > 0){

      xanterior=xmedio
      xa=xmedio
      xmedio=(xmedio+xb)/2


    }else if(cos(1/xa)   *  cos(1/xmedio)  < 0){

      xanterior=xmedio
      xb=xmedio
      xmedio=(xa+xmedio)/2

    }

    error=abs((xmedio+xanterior)/xmedio)

    Errores[indice]=error


    while( error > tol && it < itMax){

      it=it+1
      indice=indice+1

      if( cos(1/xa)  *  cos(1/xmedio)  > 0){

        xa=xmedio
        xanterior=xmedio
        xmedio=(xmedio+xb)/2
      }
      if( cos(1/xa)  *  cos(1/xmedio)  < 0){

        xb=xmedio
        xanterior=xmedio
        xmedio=(xa+xmedio)/2

      }


      error=abs((xmedio+xanterior)/xmedio)
      Errores[indice]=error
      ErroresSignificativo[j]=error
      j=j+1


    }

    ErroresSignificativo[j]=0

    itrs=seq(1,it)
    tabla=data.frame(itrs,Errores)
    print(tabla)

    plot(x = Errores, y = ErroresSignificativo,type = 'o',main = "GRAFICA ERRORES",ylab = "Error i+1",xlab =
"Error i", xlim=c(2,3),ylim=c(0,2.5),xaxp = c(2,3,10))

    cat("\n")
    cat("\n")
    cat("\n")
```

```
    cat("\n")
    cat("\n")
  }
  valoresAitken<-c(0)
  j=1
  indice=1
  longitud=length(Errores)-3
  it=1
  valor = Errores[indice+2] - ((Errores[indice+2] - Errores[indice+1]) * (Errores[indice+2] - Errores[indic
e+1])/ (Errores[indice+2] - (2*Errores[indice+1]) + Errores[+1] ))
  valoresAitken[j]=valor
  j=j+1
  indice=indice+1
  while(longitud > 0 ){
    it=it+1
    valor = Errores[indice+2] - ((Errores[indice+2] - Errores[indice+1]) * (Errores[indice+2] - Errores[ind
ice+1])/ (Errores[indice+2] - (2*Errores[indice+1]) + Errores[+1] ))
    valoresAitken[j]=valor
    j=j+1
    indice=indice+1
    longitud=longitud-1
  }

  itrs=seq(1,it)
  tabla=data.frame(itrs,valoresAitken)
  print(tabla)

}

funcion(0.002,2,10e-7)
```

```
##    itrs  Errores
## 1     1 2.199361
## 2     2 2.047474
## 3     3 2.024314
## 4     4 2.012307
## 5     5 2.003067
## 6     6 2.000766
## 7     7 2.000192
## 8     8 2.000096
## 9     9 2.000048
## 10   10 2.000024
## 11   11 2.000012
## 12   12 2.000006
## 13   13 2.000003
## 14   14 2.000001
## 15   15 2.000000
## 16   16 2.000000
## 17   17 2.000000
## 18   18 2.000000
## 19   19 2.000000
## 20   20 2.000000
## 21   21 2.000000
```




```
##
##
##
##
##    itrs valoresAitken
## 1     1      2.020147
## 2     2      2.011422
## 3     3      2.002587
## 4     4      2.000739
## 5     5      2.000190
## 6     6      2.000096
## 7     7      2.000048
## 8     8      2.000024
## 9     9      2.000012
## 10   10      2.000006
## 11   11      2.000003
## 12   12      2.000001
## 13   13      2.000000
## 14   14      2.000000
## 15   15      2.000000
## 16   16      2.000000
## 17   17      2.000000
## 18   18      2.000000
## 19   19      2.000000
```