Taller I Análisis Numérico

Primer semestre 2020

Víctor Barón, Oscar Falla, Estefanía Aristizabal

I. Introducción.

En este documento se encontrará la descripción del desarrollo de los algoritmos propuestos en clase. Además, se cuenta con los documentos correspondientes todos los GitHub de los integrantes del grupo.

II. Primera sección de ejercicios.

1. Suponga que un dispositivo solo puede almacenar únicamente los cuatro primeros dígitos decimales de cada número real, y trunca los restantes (esto es redondeo inferior). Calcule el error de redondeo si se quiere almacenar el número 536.78.

Número para almacenar: x= 536.78 expresamos el numero el numero sin enteros con potencias de 10: x= 0.5367*10*^3. Después, dividimos el numero en dos partes $x = 0.536710^3$ + 0.00008*10^3 se almacena x = 0.5367 el cual es un valor aproximado.

El error de redondeo debido a la limitación

E = $0.0000810^3 = 0.810^{(3-4)}$=0.8*$10^{-1}$

En general el error de redondeo absoluto esta acotado por:

|E| < 1 ∗ 10^n −m

donde n es la cantidad de enteros del número normalizado (sin enteros) y m la cantidad de cifras que se pueden almacenar.

2. Implemente en cualquier lenguaje el siguiente algoritmo que sirve para calcular la raíz cuadrada. Aplíquelo para evaluar la raíz cuadrada de 7, analice su precisión, como podría evaluar la convergencia y validez del algoritmo.

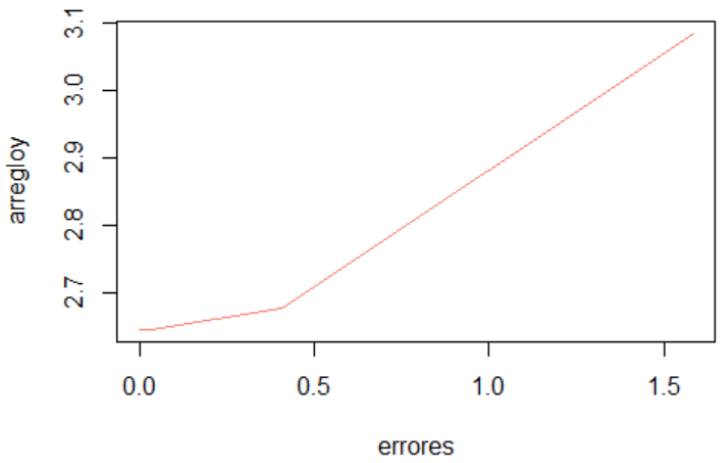
En este ejercicio se utilizan 3 parámetros esenciales: n es el número al que se le quiere calcular la raíz, un dato x y el error permitido. El primer paso es calcular es calcular y = 0.5*(x + n / x) y después se hace un ciclo hasta que el valor absoluto de la resta de **x** y **y** sea menor que el error (|x − y| < $error$) siendo la aproximación más exacta de la raíz.

Entonces siendo n = 7, x=100, tolerancia = 10^{-8} se obtienen los siguientes resultados:

```r
#n representa el numero del que quiero hallar la raiz
#Error que representa la tolerancia o el error del resultado
raizCuadrada = function(n, Error)
{
  x = 100;
  y = 0.5*( x + n / x);
  repeat
  {
    if( abs(x - y) < Error)
    {
      cat("Raíz aproximada de ",n, " es: ", y , " Con error de " ,abs(x-y))
      break;
    }
    x = y
    y = 0.5 * (x+n/x)
  }

}
raizCuadrada(7,1e-9)

#Raíz aproximada de  7  es:  2.645751  Con error de  9.009682e-12
```



3. Utilizando el teorema de Taylor hallar la aproximación de e^0.5 con cinco cifras significativas.

En este ejercicio se utilizan 2 parámetros, n = número de polinomios de Taylor y x= el exponente de la función. Se empieza inicializando los valores de las variables, poniéndolas en un ciclo mientras que i>0 actualizándose a su vez los valores de las variables. Finalmente, al tener a n =5 y x=0.5 se obtienen los siguientes resultados:

```
#n representa el número de polinomios de taylor
#x representa el exponente de la expresión
Aproximacion=function (n,x){
  suma=1
  i=n
  while(i>0){
    suma=1+x*suma/i
    cat("Polinomio: ",suma,"\n")
    i=i-1
  }
  cat(signif(suma,digits=5))
}
Aproximacion(5,0.5)
#Polinomio:  1.1
#Polinomio:  1.1375
#Polinomio:  1.189583
#Polinomio:  1.297396
#Polinomio:  1.648698
#1.6487
```

4. La velocidad de una partícula constante e igual a **4m/s**, medida con un error de **0.1 m/s** durante un tiempo recorrido de **5 seg** medido con el error de **0.1**. Determine el error absoluto y el error relativo en el valor de la distancia recorrida

Para resolver este ejercicio contábamos con cuatro parámetros la velocidad media m_v, el error de esta velocidad e_v, el tiempo medio m_t y el error de medición de este tiempo et. El primer paso es calcular el valor del error absoluto (error absoluto = m_v *m_ev +m_t *m_et) y después el valor del error relativo (error relativo = m_eV/v + m_et/t).

Entonces, al ingresar los siguientes valores **m_v = 4**, **m_ev=0.1**, **m_t=5**, **m_et=01** obtenemos los siguientes resultados:

```
#m_V representa la velocidad medida
#m_eV representa el error de la velocidad
#m_T representa el tiempo medido
#m_eT representa el error del tiempo

calcularErrorD = function(m_V, m_eV, m_T, m_eT)
{
  v=m_V
  t=m_T
  d=v*t
  absError = v*m_eV+t*m_eT
  relError =  m_eV/v + m_eT/t
  cat("Distancia: ",d," Error Absoluto: " ,absError, "Error Relativo: ", relError )
}

calcularErrorD(4,0.1,5,0.1)

#Distancia:  20  Error Absoluto:  0.9 Error Relativo:  0.045
```

5. Evaluar el valor de un polinomio es una tarea que involucra para la maquina realizar un número de operaciones la cual debe ser mínimas. Como se puede evaluar el siguiente polinomio con el número mínimo de multiplicaciones.

Para la solución de este problema utilizamos el Algoritmo de Horner ya que se puede calcular de manera eficiente el valor del polinomio de grado n. Un polinomio de grado n se puede evaluar en n multiplicaciones y n adiciones.

Para esto, se cuenta con 3 parámetros: el polinomio que se va a evaluar (p), el valor del exponente máximo (n) y la representación del polinomio en x0 y se realizan los siguientes pasos:

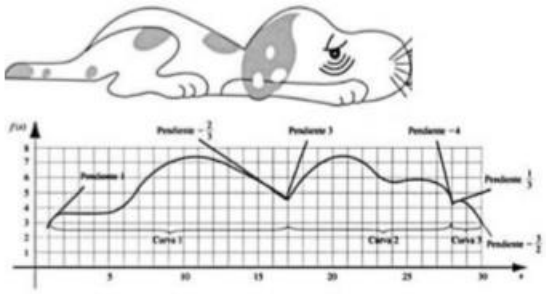
El primer paso es la inicialización de la variable resultado=funcion[1], que es el valor de x en el polinomio en el término x^1 y después, se hace un recorrido con un for iniciando en 2 hasta llegar a n, haciendo a su vez el calculo del valor final resultado= resultado*x0 + funcion[i] valor de x en el polinomio en el término x^i.

```
#funcion representa el polinomio a evaluar
#g representa el máximo exponente del polinomio
#x0 representa el x0 del polinomio
Horner = function (funcion, g, x0){
  resultado=funcion[1]
  n=0
  for(i in 2:(g+1)){
    resultado= resultado*x0 + funcion[i]
    n=n+2
  }
  cat("El resultado del polinomio es: ", resultado, " en ",n,"sumas.")
}
funcion<-c(2,0,-3,3,-4)
Horner(funcion,4,-2)
```

Entonces, es así como se le puede dar solución al polinomio planteado **P(x)=2x^4-3x^2+3x-4** con x0 = -2 y con 4 puntos máximos en donde se obtienen los siguientes resultados:

6. Reconstruir la silueta del perrito utilizando la menor cantidad de puntos para reproducir el dibujo del contorno completo del perrito sin bigotes, con la información dada:

   Coordenadas:
   y=c(3,3.7,3.9,4.5,5.7,6.69,7.12,6.7,4.45,7,6.1,5.6,5.87,5.15,4.1,4.3,4.1,3)
   x=c(1,2,5,6,7.5,8.1,10,13,17.6,20,23.5,24.5,25,26.5,27.5,28,29,30)

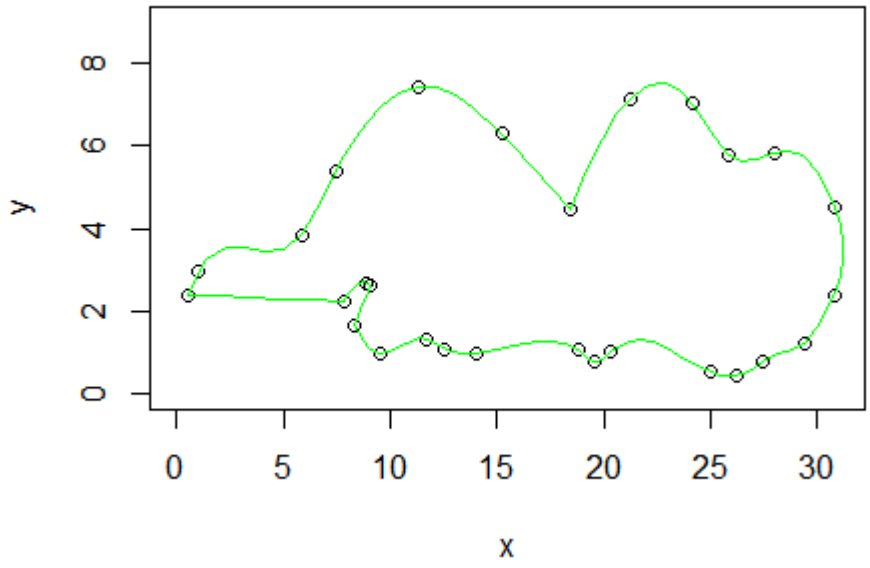


```
x=c( 00.50 , 01.01 , 05.85 , 07.46 , 11.28 , 15.20 , 18.46 , 21.25
     , 24.15 , 25.80 , 28.00, 30.80 , 30.81 , 29.40 , 27.40 , 26.21 ,
     24.97 , 20.32 , 19.54 , 18.80 , 14.04 , 12.54  , 11.68 , 09.55 ,
     08.30 , 09.10 , 08.85 , 07.80 , 00.50)
y=c( 02.40 , 02.95 , 03.86 , 05.41 , 07.45 , 06.30 , 04.49 , 07.15 ,
     07.05 , 05.80 , 05.85, 04.50 , 02.40 , 01.20 , 00.80 , 00.44 ,
     00.54 , 01.01 , 00.80 , 01.08 , 00.98 , 01.08, 01.33 , 01.00 ,
     01.64 , 02.65 , 02.70 , 02.24 , 02.40)
```

```
segx1=x[1:7]
segx2 = x[7:12]
segx3 = x[12:14]
segx4 = x[14:15]
segx5 = x[15:18]
segx6 = x[18:20]
segx7 = x[20:25]
segx8 = x[25:26]
segx9 = x[26:28]
segx10 = x[28:29]
segy1 = y[1:7]
segy2 = y[7:12]
segy3 = y[12:14]
segy4 = y[14:15]
segy5 = y[15:18]
segy6 = y[18:20]
segy7 = y[20:25]
segy8 = y[25:26]
segy9 = y[26:28]
segy10 = y[28:29]
seg = spline(segy3,segx3)
```

```
i = seg$x
seg$x = seg$y
seg$y = i
plot(x, y,sub="Interpolación Perro",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx1, segy1), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx2, segy2), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx4, segy4), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx5, segy5), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx6, segy6), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx7, segy7), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx8, segy8), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx9, segy9), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(spline(segx10, segy10), col = "green",xlim=c(0,31),ylim=c(0,9))
lines(seg, col = "green")
```

Y obtenemos la siguiente grafica:



III. Segunda sección de ejercicios.

1. Número de Operaciones

Utilice el método de inducción matemática para demostrar el resultado del método.

El método de inducción matemática se basa en dos pasos; Paso básico y paso inductivo.

El paso básico consta de probar los resultados de la hipótesis para n(grado del polinomio) = 1, este paso es relativamente para un polinomio de grado 1 nos damos cuenta que hacen falta n productos.

Para el paso inductivo es necesario asumir que el método es cierto para cualquier polinomio de grado n y a continuación se encuentra la hipótesis de inducción con el caso n+1.

En la demostración se suma la iteración n+1 y se intenta llegar a la hipótesis de inducción; si logramos comprobar este resultado queda demostrado que el número de productos para el segundo método es 2n-1.

```
Paso Básico:

        Para n = 1
        P1(Xo) = (a1 , X0) + ao

Paso Inductivo:

        Hipotesis de inducción (Suponga que vale para n)
        Pn(x) = an X^n + an-1 X^n-1 + ... + a1 X + ao
        Pn(xo); Hay 2n-1 multiplicaciones

Demostración:

        Pn+1(x) = an-1 X^n+1 + anX^n + ... +  a1 X + ao
        = x(an+1 X^n + anX^n-1 + ... + a2 X+a1) + ao

        Pn+1(Xo) = Xo (qn(Xo) + a1)
                        1
                        1
                        1
        //Hay (2n-1)^Xo -> Multiplicaciones  //

        Xo qn (Xo) + Xo.a1 + ao

        (2n-1) + 1 + 1 = 2n + 1
```

2. Implemente en R o Python para verificar los resultados del método de Horner

El método de Horner o multiplicaciones anidadas reduce el número de productos efectuados por el método tradicional a solo n productos.

Algoritmo

```
5  bn = arr[1]
6
7  for(i in 2:length(arr)){
8    bn = arr[i] + bn*xo
9  }
```

Resultado con el ejemplo P(x) = 2x^4 - 3x^2 + 3x – 4:

3. Evaluar en x = 1.0001 con P(x) = 1+x+x^2 +...+x^50. Encuentre el error de cálculo al compáralo con la expresión equivalente Q(x) = (x^51 −1) / (x−1)

```
resultadoIterativo = 0;
i = 0;
x = 1.0001;
#Inicio del Ciclo
repeat{
 resultadoIterativo=resultadoIterativo+x^i;
 i=i+1;
 if(i>50) break;
}
#function Q
q=function(b) ((b^51)-1)/(b-1)
resultadoDirecto=q(x);
#error absoluto
errorA=abs(resultadoDirecto-resultadoIterativo);
#error relativo
errorR=((resultadoIterativo/(resultadoIterativo+resultadoDirecto))(errorA/resultadoIterativo))+(
(resultadoDirecto/(resultadoIterativo+resultadoDirecto))(errorA/resultadoDirecto));
#Tabla
Datos = data.frame(i,errorA,errorR,resultadoIterativo,resultadoDirecto)
print(Datos);
```

El error arrojado por R al momento de evaluar X en la función P(x) comparándolo con la expresión equivalente de Q(x) fue;

```
  i        errorA       errorR resultadoIterativo resultadoDirecto
51 7.105427e-14 1.389741e-15           51.12771         51.12771
```

2. *Números Binarios*

En ambos casos como para pasar de real decimal a real binario y viceversa se dividió el número tanto en su parte entera como en su parte decimal

Se divide el número utilizando modulo (%%) de 10 del numero para poder evaluar el numero por cifras y utilizando división entera (% / %)

En el caso de transformar el numero de decimal a binario, se toma la parte entera se divide en dos hasta que el numero se vuelve cero y los residuos de las divisiones invertidas es el numero en binario

En la parte decimal se multiplica por dos y se invierte el número y se toma el del extremo se aplica este mismo proceso

Al transformar de binario a decimal depende de la posición se eleva dos a la potencia de la posición y se suman todas

En el algoritmo se hicieron funciones independientes para cada una de estas operaciones y después se sumaron para generar los resultados

a) Encuentre los primeros 15 bits en la representación binaria de π

```
#a) Encuentre los primeros 15 bits en la representaci´on binaria de π
cat("Pi en numero binario es: ")
parte_entera_pi = trunc (pi)
parte_decimal_pi = pi-3
residuo = 0
numero_residuos = 0
binario <- c()
# Parte entera pi
while (parte_entera_pi > 0){
   residuo = trunc (parte_entera_pi %% 2)
   parte_entera_pi = trunc (parte_entera_pi / 2)
   numero_residuos = numero_residuos + 1
   binario[numero_residuos] = residuo
}
 numero_binario <- c()
 j <- 0
 for(i in 1 : numero_residuos){
   numero_binario[i] = binario [numero_residuos-j]
   j = j+1
 }
```

```
  cat (numero_binario)
binario_decimal <- c()
# Parte decimal de pi
for (i in 1:13) {
  parte_decimal_pi = parte_decimal_pi * 2;
  if(parte_decimal_pi >= 1) {
    parte_decimal_pi = parte_decimal_pi - 1;
    binario_decimal[i] = 1
  }
  else {
    binario_decimal[i] = 0
  }
}
  cat ("El numero Pi en binario de 15 bits es: [" , numero_binario, binario_decimal,"]")
```

Una vez ejecutado el código anterior, el resultado arrojado para el numero PI en binario es el siguiente:

```
El numero Pi en binario de 15 bits es: [ 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 ]
```

b) Convertir los siguientes números binarios a base 10: 1010101; 1011.101; 10111.010101...; 111.1111...

```
binario_a_entero = function (parte_entera) {
  numero_potencia = 0
  binario = 0
  total = 0
  while(parte_entera > 0) {
    binario = parte_entera %% 10
    total = total + (binario*(2 ^ numero_potencia))
    numero_potencia = numero_potencia + 1
    parte_entera = parte_entera %/% 10
  }
  return (total)
}
# Parte decimal del binario convertida a base 10
binario_a_decimal <- function (parte_decimal) {
  vector_binario <- c()
  i = 1
  while(parte_decimal > 0){
    vector_binario[i] = parte_decimal %% 10
    parte_decimal = parte_decimal %/% 10
    i = i + 1
  }
```

```
 j <- 0
 numero_decimal = c()
 for(i in 1 : length(vector_binario)) {
  numero_decimal[i] = vector_binario[length(vector_binario)-j]
  j = j+1
 }
 binario = 0
 total = 0
 numero_potencia=1
 print(numero_decimal)
 for(i in 1 : length(numero_decimal)) {
  binario = numero_decimal[i]
  if(binario == 1){
  total = total + 1/(2 ^ numero_potencia)
  }
  numero_potencia = numero_potencia +1
 }
 return(total)
}
cat("Numero 1: ", binario_a_entero (101010101),"\n")
cat("Numero 2: ",binario_a_entero(1011) + binario_a_decimal(101),"\n")
cat("Numero 3:",binario_a_entero(10111) + binario_a_decimal(010101))
cat("Numero 4: ",binario_a_entero(111) + binario_a_decimal(1111))
```

Posteriormente se realizó la ejecución del código anterior en donde lo que se busca es convertir 4 números binarios decimales a enteros decimales, arrojando los siguientes resultados;

```
> cat("Numero 1: ", binario_a_entero (101010101),"\n")
Numero 1:  341
>
> cat("Numero 2: ",binario_a_entero(1011) + binario_a_decimal(101),"\n")
[1] 1 0 1
Numero 2:  11.625
>
> cat("Numero 3:",binario_a_entero(10111) + binario_a_decimal(010101))
[1] 1 0 1 0 1
Numero 3: 23.65625>
> cat("Numero 4: ",binario_a_entero(111) + binario_a_decimal(1111))
[1] 1 1 1 1
Numero 4:  7.9375
```

c) Convierta los siguientes números de base 10 a binaria: 11.25; 2/3 ; 30.6; 99.9

```
rm(list = ls())
# Parte decimal del numero decimal:
decimal_a_binario <- function (parte_decimal) {
 bits_permitidos <- 10
 round (parte_decimal , 1)
 binario <- c()
 for (i in 1 : bits_permitidos) {
  parte_decimal = parte_decimal * 2;
  if(parte_decimal >= 1) {
   parte_decimal = parte_decimal - 1
   binario[i] = 1
```

```
    }
    else
    {
      binario[i] = 0
    }
  }
  return(binario)
}
# Parte entera del numero decimal:
entero_a_binario <- function (parte_entera) {
  residuo = 0
  numero_residuos = 0
  binario <- c()
  while (parte_entera > 0) {
    residuo = (parte_entera %% 2)
    parte_entera = trunc (parte_entera / 2)
    numero_residuos = numero_residuos + 1
    binario[numero_residuos] = residuo
  }
  numero_binario <- c()
  j <- 0
  for(i in 1 : numero_residuos){
    numero_binario[i] = binario [numero_residuos-j]
    j = j+1
  }
  return (numero_binario)
}
cat("Primero numero de base 10 a binario: ", entero_a_binario(11) , "."
,decimal_a_binario (0.25), "\n")
cat("Segundo Numero a Base 10 a binario: ", "0" , "." , decimal_a_binario (2/3), "\n")
cat("Tercer Numero a Base 10 a binario: ", entero_a_binario(30), "." ,
decimal_a_binario(0.6), "\n")
cat("Cuarto Numero a Base 10 a binario: ", entero_a_binario (99), "." ,
decimal_a_binario(0.9), "\n")
```

Los resultados de las conversiones de entero a binario son los siguientes;

```
> cat("Primero numero de base 10 a binario: ", entero_a_binario(11) , "." ,decimal_a_bi
nario (0.25), "\n")
Primero numero de base 10 a binario:  1 0 1 1 . 0 1 0 0 0 0 0 0 0 0
> cat("Segundo Numero a Base 10 a binario: ", "0" , "." , decimal_a_binario (2/3), "\n"
)
Segundo Numero a Base 10 a binario:  0 . 1 0 1 0 1 0 1 0 1 0
> cat("Tercer Numero a Base 10 a binario: ", entero_a_binario(30), "." , decimal_a_bina
rio(0.6), "\n")
Tercer Numero a Base 10 a binario:  1 1 1 1 0 . 1 0 0 1 1 0 0 1 1 0
> cat("Cuarto Numero a Base 10 a binario: ", entero_a_binario(99), "." , decimal_a_bina
rio(0.9), "\n")
Cuarto Numero a Base 10 a binario:  1 1 0 0 0 1 1 . 1 1 1 0 0 1 1 0 0 1
```

3. *Épsilon de una Maquina*

- ¿Cómo se ajusta un numero binario infinito en numero finito de bits?

  Para representar un número infinito en un numero finito de bits, se debe colocar el número 1 en todos los bits del exponente y el número 0 en todos los del significado, y dependiendo del primero bit será infinito negativo o positivo.

- ¿Cuál es la diferencia entre redondeo y recorte?

  El redondeo consiste en llevar un numero al más cercano terminado en cero, redondear es sustituir un numero por el más próximo. En cambio, el recorte sin importar que decenas, centenas, unidades de mil etc. dependiendo de cuantos dígitos, este será recortado sin ningún tipo de aproximación.

- ¿Cómo se ajusta un numero binario infinito en un numero finito de bits?

signo	Exponente	Mantisa
0	11111111	00000000000000000000000
1	11111111	00000000000000000000000

- Indique el numero de punto flotante (IEEE) de precisión doble asociado a x, el cual se denota como fl(x); para x(0.4)

  0011111011001100110011001101

- Verificar si el tipo de datos básico de R y Python es de precisión doble IEEE y Revisar en R y Phython el format long

  El tipo de dato básico de R y Python es de precisión doble

- Encuentre la representaci´on en número de maquina hexadecimal del número real 9.4

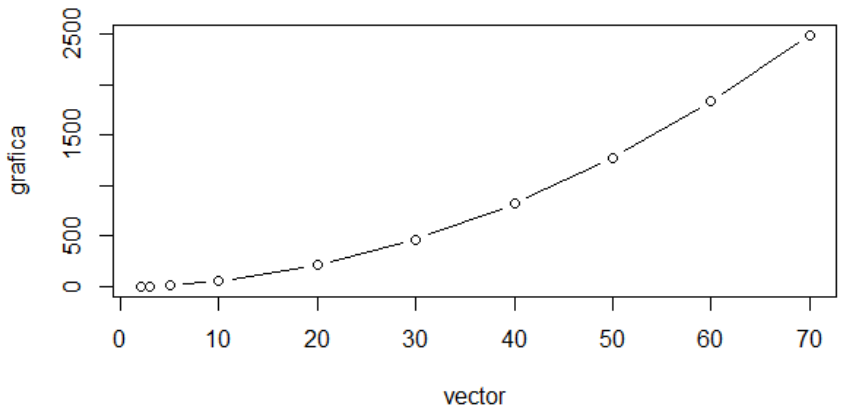
  La representación hexadecimal del número 9.4 es igual a 5E

4. Raíces de una Ecuación

- Implemente en R o Python un algoritmo que le permita sumar únicamente 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 notación O() con una gráfica que muestre su orden de convergencia.

```
vector = c(2,3,5,10,20,30,40,50,60,70)
mtriangular <- function (vector) {
 total <- c()
 for (i in vector) {
  mtriangular = matrix (1 , ncol = i, nrow = i)
  aux = 0
  for (j in 1:i) {
   for (k in 1:j) {
    aux = aux + mtriangular[j,k]
   }
  }
  total = c(total , aux)
  aux = 0
 }
 return (total)
}
grafica = mtriangular (vector)
plot (vector , grafica , type = 'b',col = 1)
```

La grafica obtenida de nos puede mostrar su orden de convergencia,



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

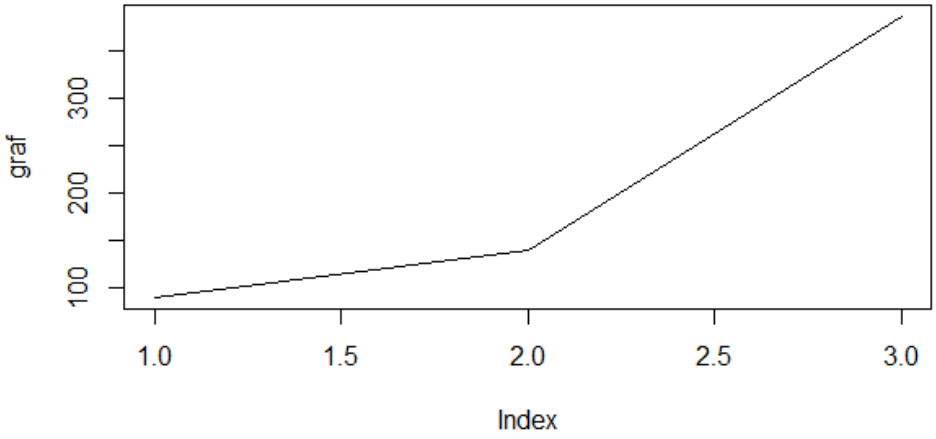
```
#PUNTO 2 - RAICES DE UNA ECUACION
puntob = function(x){
  resul = (x*(x+1)*(2*(x)+1))/6
  plot(resul)
  return (resul)
}
x = puntob(6)
y = puntob(7)
z = puntob(10)

graf=c(x,y,z)
plot(graf,type="l")
```

La tabla arrojada para la suma de los n^2 primeros números naturales cuadrado fue;

Values	
graf	num [1:3] 91 140 385
x	91
y	140
z	385

Para los diferentes valores de n, la expresión de f(n) gráficamente es la siguiente,



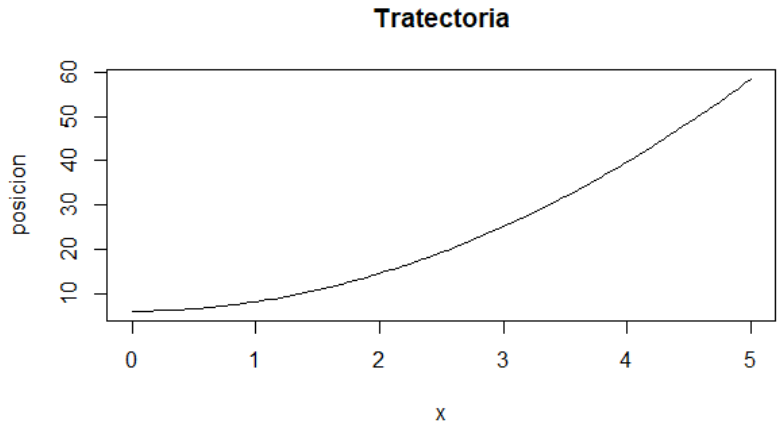
- Para describir la trayectoria de un cohete se tiene el modelo: y(t) = 6 + 2, 13t 2 − 0.0013t 4 Donde, y es la altura en [m] y t tiempo en [s]. El cohete está colocado verticalmente sobre la tierra. Utilizando dos métodos de solución de ecuación no lineal, encuentre la altura máxima que alcanza el cohete.

```
#ALGORITMO
trayecttoria <- function()
{
 t = 0
 posicion = function(t){6+(2.13 * (t^2)) - (0.0013 * (t^4))}
 funcion = 6+(2.13 * (t^2)) - (0.0013 * (t^4))
 resultado = 6+2.13*((t+1)^2)-0.0013*((t+1)^4)
 arr = c(0)
 while(funcion < resultado){
  t =t+1
  funcion = 6+2.13*(t^2)-0.0013*(t^4)
  resultado = 6+2.13*((t+1)^2)-0.0013*((t+1)^4)
 }
plot(posicion, xlim = c(0,5), main = "Tratectoria")
 cat("La altura mayor alcanzada es :",funcion)
}
trayecttoria()
```

Para la realización de este problema se procedió a tomar la función dada en el enunciado y posteriormente sumar 1 por cada t, dentro de la función. Calculando finalmente la trayectoria máxima que el cohete puede tener. El resultado fue el siguiente,

```
La altura mayor alcanzada es: 877.8647
```

La grafica para la trayectoria del cohete es la siguiente,



5. Convergencia de Métodos Iterativos

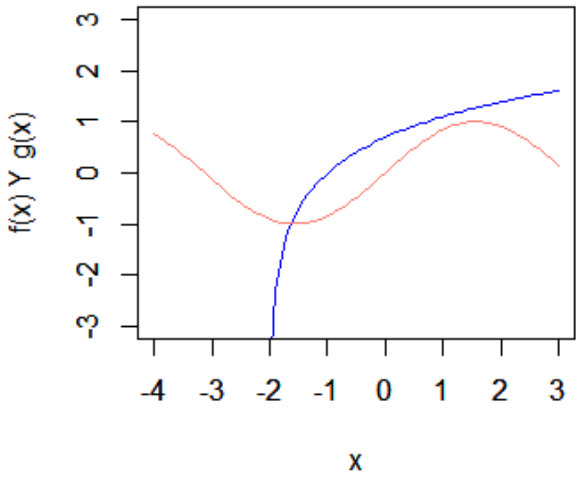
Para medir la convergencia de un método iterativo

$$lim_{n\to\infty}\frac{|x_{n+1}-x|}{|x_n-x|^{\alpha}} = lim_{n\to\infty}\frac{|E_{n+1}|}{|E_n|^{\alpha}} = \lambda$$

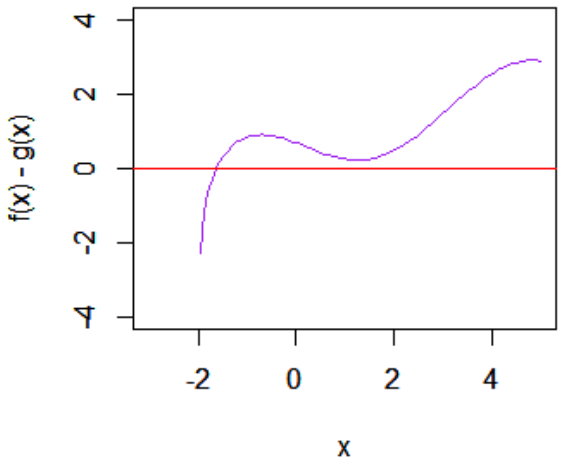
1. Sean f(x) = ln(x + 2) y g(x) = sin(x) dos funciones de valor real.

Análisis:

Tenemos dos funciones f(x) y g(x)



para encontrar el punto de intersección debemos restarlas



Para encontrar la intersección de la función basta con encontrar la raíz de la función resultante, es decir el punto de corte con el eje x

a) Utilice la siguiente formula recursiva con E = 10^−8 para el punto de intersección.:

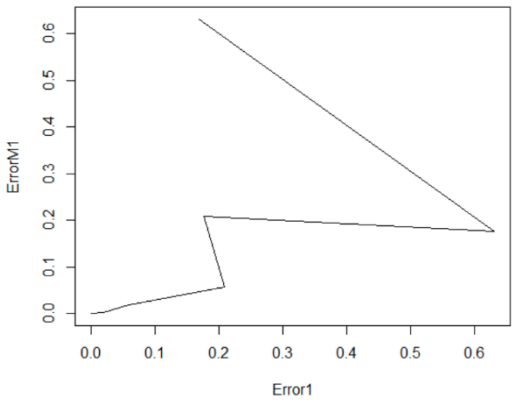
$$x_n = x_{n-1} - \frac{f(x_{n-1})(x_{n-1} - x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}$$

Resultados del método 1:

```
 [1] -1.800000 -1.000000 -1.455754 -1.840215 -1.573704 -1.612310 -1.633103 -1.631396 -1.631443
[10] -1.631444 -1.631444
```

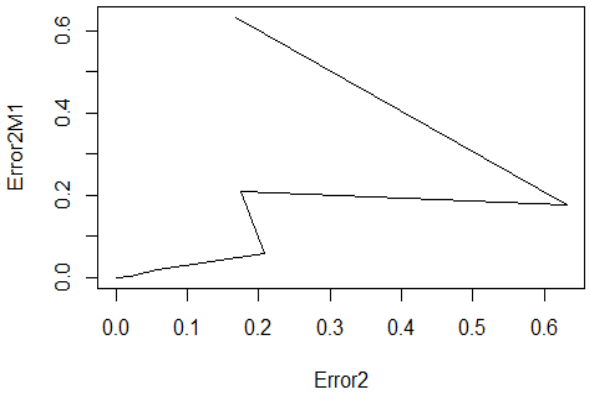
Convergencia del Metodo1:

```
        Error1      ErrorM1
1 1.685564e-01 6.314436e-01
2 6.314436e-01 1.756895e-01
3 1.756895e-01 2.087718e-01
4 2.087718e-01 5.773983e-02
5 5.773983e-02 1.913357e-02
6 1.913357e-02 1.659308e-03
7 1.659308e-03 4.778398e-05
8 4.778398e-05 1.194990e-07
9 1.194990e-07           NA
> 
```



b) Aplicar el m´etodo iterativo siguiente con E = 10^−8 para encontrar el punto de intersección:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$



Resultados del método 2:

Error2	Error2M1
1.685564e-01	6.314436e-01
6.314436e-01	1.756895e-01
1.756895e-01	2.087718e-01
2.087718e-01	5.773983e-02
5.773983e-02	1.913357e-02
1.913357e-02	1.659308e-03
1.659308e-03	4.778398e-05
4.778398e-05	1.194990e-07
1.194990e-07	NA

Conclusión: Ambos métodos aunque tienen formas diferentes convergen a la misma velocidad y se comportan de la misma manera.

2. Utilizando el método de Newton con p0 = 1 verifique que converge a cero pero no de forma cuadrática.

   El código de este ejercicio lo podrá encontrar en el GitHub de cada uno de los integrantes del presente trabajo, en la carpeta de AnalisisNumerico/Talleres/Taller_1_Complementarios.

   Al momento de realizar la verificación de si la gráfica converge mediante el método de *Newton,* procedemos a hacer un *abline* "que describe una recta de color en la gráfica inferior" en donde se puede verificar que dicha gráfica tiene tendencia ().

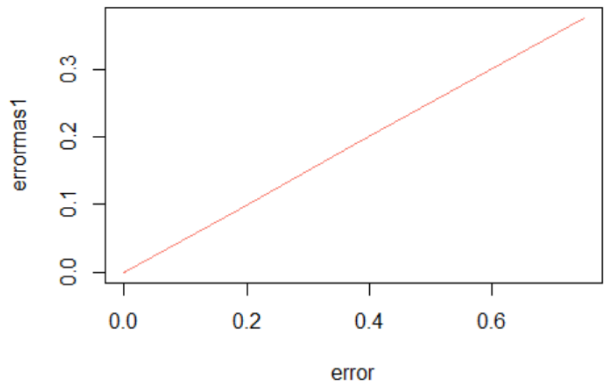
6. Convergencia Acelerada

Existen básicamente dos metodologías para acelerar la convergencia. Método de Δ2 Aitken. Técnica que se usa para acelerar la convergencia de cualquier sucesión que converja linealmente, independientemente de su origen.

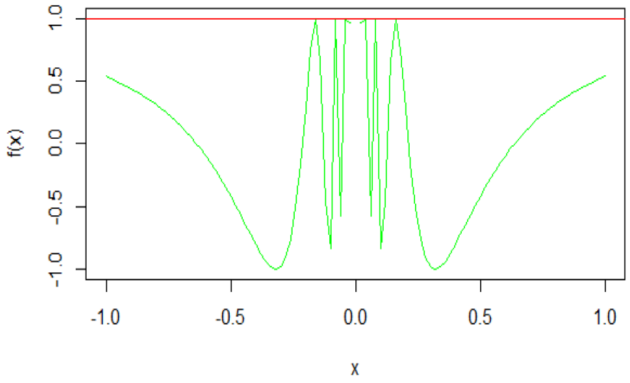
1. Verifique el tipo de convergencia en x = 1 independiente del origen

Para la convergencia acelerada utilizamos el método delta cuadrado de Aiken a el método de bisección un método con convergencia lineal

Convergencia método de bisección(lineal):



Xn = Cos(1/x)



Resultados:

14	9.155273e-05	4.577637e-05	0.6365906	0.6366211
15	4.577637e-05	2.288818e-05	0.6366364	0.6366211
16	2.288818e-05	1.144409e-05	0.6366135	0.6366211
17	1.144409e-05	5.722046e-06	0.6366249	0.6366211
18	5.722046e-06	2.861023e-06	0.6366192	0.6366211
19	2.861023e-06	1.430511e-06	0.6366220	0.6366192
20	1.430511e-06	7.152557e-07	0.6366206	0.6366192
21	7.152557e-07	3.576279e-07	0.6366199	0.6366197
22	3.576279e-07	1.788139e-07	0.6366195	0.6366199
23	1.788139e-07	8.940697e-08	0.6366197	0.6366198
24	8.940697e-08	4.470348e-08	0.6366198	0.6366198
25	4.470348e-08	2.235174e-08	0.6366198	0.6366198
26	2.235174e-08	1.117587e-08	0.6366198	0.6366198
27	1.117587e-08	5.587935e-09	0.6366198	*NA*
28	5.587935e-09	*NA*	0.6366198	*NA*

Según las tablas podemos notar que el método delta^2 Aiken si acelera la convergencia, más sin embargo solo disminuye la cantidad de iteraciones en 2 lo cual no es muy conveniente aplicarlo en este caso porque dos no es significativo teniendo el número de iteraciones.

Metodo de Steffersen

Aplicando el método alfa al cuadrado de Aitken a una sucesión que converge linealmente obtenida de la iteración de punto fijo, podemos acelerar la convergencia a cuadrática. Este procedimiento es conocido como el método de Steffersen y difiere un poco de aplicar el de Aitken directamente a una sucesión de iteración de punto fijo que sea linealmente convergente.

Utilice el algoritmo de Steffersen para resolver x 2 − cosx y compararlo con el metodo de Aitken

El código que que resuelve *x2-cosx* para el algoritmo de Steffersen comprado con el método de Aitken puede encontrar en el GitHub de cada uno de los integrantes en la carpeta de Talleres/Taller_1_Complementarios.