Presentación del Taller

Stefania García, Santiago Torres, Esteban Villa, Julian Tarazona

25 de septiembre de 2020

Introducción

El presente documento tiene como objetivo realizar una serie de ejercicios propuestos a partir del taller 1 utilizando una serie de herramientas como lo son R, Python, sus extenciones de graficación y los diversos algoritmos y métodos vistos anteriormente en el curso, todo esto con el fin de reforzar los conocimientos ya establecidos en anteriores trabajos.

1) **Numero de Operaciones** En este taller Inicialmente evaluaremos 2 polinomios en el cual en ambos encontraremos el error de calculo para así poder comparar entre ellos y concluir. Para lo anteriormente mencionado se nos propuso dos expresiones. los cuales son: polinomio en $x = 1{,}00000000001$ con $P(x) = 1 + x + x^2 + ... + x^{50}$ y compararlo con la expresión equivalente: $Q(x) = (x^{51} - 1)/(x - 1)$

```
resultado = 0;
i = 0;
x = 1.00000000001;

#polinomio P(x)

repeat{
  resultado = resultado + x^i;
  i = i + 1;
  if(i > 50)
    break;
}

#derivada del p(x)

n=51                            #Grado maximo + 1
coeficientes=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,
               1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
coeficientesDerivada=c()     #Coeficientes derivando el polinomio
resultadoD=c()
res=0
j=n-1

i1 = 1;
while (i1<n){
  coeficientesDerivada[i1] = coeficientes[i1]*(j)
  i1=i1+1
  j=j-1
}


j = 1
while(j<n){
  res = res * x + coeficientesDerivada[j]
  resultadoD[j] = res;
  j=j+1;
}

#funcion Q

Q = function(y)((y^51) - 1)/(y - 1)

rDirecto = Q(x);


#derivada de Q
dq = function(x) 50*x^49+49*x^48+48*x^47+47*x^46+46*x^45+45*x^44+44*x^43+43*x^42+42*x^41
                +41*x^40+40*x^39+39*x^38+38*x^37 +37*x^ 36 +36*x^ 35 +35*x^ 34 +34*x^33
                +33*x^ 32 +32*x^ 31 +31*x^ 30 +30*x^ 29 +29*x^ 28 +28*x^ 27 +27*x^ 26
                +26*x^ 25 +25*x^ 24 +24*x^ 23 +23*x^ 22 +22*x^ 21 +21*x^ 20 +20*x^ 19
                +19*x^ 18 +18*x^ 17 +17*x^ 16 +16*x^ 15 +15*x^ 14 +14*x^ 13 +13*x^ 12
                +12*x^ 11 +11*x^ 10+10*x^9+9*x^8+8*x^7+7*x^6+6*x^5+5*x^4+4*x^3+3*x^2+2*x
                +1

rDirectoDerivado = dq(x);

#errores
#error absoluto p(x)
errorA = abs(rDirecto - resultado);
#error absoluto p'(x)
errorDA = abs(rDirectoDerivado - res);


#error relativo p(x)
errorR=((resultado/(resultado+rDirecto))*(errorA/resultado))+(
  (rDirecto/(resultado+rDirecto))*(errorA/rDirecto));
```

```
#error relativo p'(x)
errorDR = ((res/(res+rDirectoDerivado))*(errorDA/res))+(
  (rDirecto/(res+rDirectoDerivado))*(errorDA/rDirectoDerivado));
```

Para realizar este ejercicio se realizó un programa que calcula el valor del polinomio en 1.00000000001 a través de una sumatoria, luego utilizamos el valor anteriormente mencionado en la función Q(x) y con el resultado obtenido podamos comparar el resultado obtenido de forma iterativa y de forma directa a través del calculo del error absoluto y relativo. Para el caso de la derivada se utilizó el método de Horner para determinar el valor de p'(x) en el punto dado y este resultado es comparado con el valor del punto en la derivada Q'(x), y ambos resultados son comparados a través del calculo del error absoluto y relativo.

1.1) Calculo del error: Para el calcular el error de ambas expresiones usamos un programa en R Studio para hallarlo de esta manera obteniendo los siguientes resultados:

```
 i       errorA       errorR resultado rDirecto  res errorDA   errorDR
51 1.275002e-08 2.500004e-10        51       51 1275     861 0.5725667
```

donde errorA es el error absoluto, errorR es el error relativo, resultado es el valor del polinomio en x = 1.00000000001, rDirecto es el valor de x evaluado en Q(x), errorDA es el error absoluto de las derivadas del polinomio y la función evaluada en el punto y errorDR es el error relativo de las derivadas evaluadas en el punto.

a) Encuentre los primeros 15 bits en la representación binaria de : Para este punto desarrollamos un programa en R el cual nos permitió encontrar los primeros 15 bits para .

```
cat("Los Primeros 15 digitos de pi en binario: ")
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,"]")
```

Este programa funciona esencialmente de la siguiente manera: Declaramos 2 variables a la cual una se le asigna la parte entera de pi y a la otra la parte decimal de pi a la cual ambas lo que se hace es aplicar las respectivas propiedades para hallar su equivalente en binario, es decir hallar el modulo 2 y a partir de este tomar el residuo para saber cuantas veces debemos repetir el proceso y con la decimal es una cantidad definida debido a que el ejercicio nos solicita explícitamente el hallar 15 bits y le quitamos dos bits que son el 3 y el punto quedándonos 13 valores decimal por hallar.

Obteniendo como resultado lo siguiente como primeros 15 bits de :

```
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 numeros binarios a base 10: 1010101; 1011.101; 10111.010101...; 111.1111...

El cual implementamos el siguiente código en R Studio para solucionar la conversión de los números anteriormente propuestos.

```
# Parte entera del binario convertido a base 10
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("Primer Numero: ", binario_a_entero (101010101),"\n")

cat("Segundo Numero: ",binario_a_entero(1011) + binario_a_decimal(101),"\n")

cat("Tercer Numero: ",binario_a_entero(10111) + binario_a_decimal(010101))

cat("Cuarto Numero: ",binario_a_entero(111) + binario_a_decimal(1111))
```

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

De forma sencilla este código intenta solucionar de forma fraccionada los problemas planteados es decir por un lado convertimos la parte entera y por otro lado la parte decimal para facilitar el cambio de decimal a binario lo cual al final se concatena para tener el valor completo.

c) Convierta los siguientes numeros 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")
```

Aprovechándonos de los principios del código anterior lo que hicimos para convertir es básicamente similar al pasado en este caso también dividimos en 2 funciones en la cual una se convierte la parte entera y en la otra la parte decimal con sus respectivas potencias de 2 y uniéndolos en la impresión con un "..[En] la cual obtuvimos los siguientes resultados:

```
> 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
```

Representacion del Punto Flotante de los Numeros Reales

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

Hay dos tipos de infinitos: positivos y negativos. La convención para representar estos números dice que un número se considera infinito si cada uno de los bits del exponente es 1 y los de la mantisa son 0. El signo permite distinguir entre $+\infty$ y $-\infty$.

Propiedad	Simple precisión	Doble precisión
Mayor número representable	$3.402823466\ 10^{38}$	$1.7976931348623157\ 10^{308}$
Menor número sin pérdida de precisión	$1.175494351\ 10^{-38}$	$2.2250738585072014\ 10^{-308}$
Menor númer representable	$1.401298464\ 10^{-45}$	$5\ 10^{-324}$
Bits de mantisa	23	52
Bits de exponente	8	11
Epsilon	$1.1929093\ 10^{-7}$	$2.220446049250313\ 10^{-16}$

2. ¿Cuál es la diferencia entre redondeo y recorte?

En esencia el redondeo es una técnica que nos permite a partir de unos principios aproximar los números a al siguiente 0 muchas veces con el propósito de disminuir el largo del numero sin perder mucha precisión en cambio el recorte generalmente se proporciona un n que es el largo limite que tendrá el numero y justo en esa posición el numero sera recortado sin importar proximidad.

3. ¿ Como se ajusta un numero binario infinito en un numero finito de bits?
Signo Exponente Mantisa
0 11111111 00000000000000000000000
1 11111111 00000000000000000000000

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

0011111011001100110011001100110 01101

5. Error de redondeo En el modelo de la arimetica de computadora IEEE, el error de redondeo relativo no es mas de la mitad del epsilon de maquina:

$\frac{|fl(x)-x|}{|x|} \leq \frac{1}{2}emac$

Teniendo en cuenta lo anterior encuentre el error de redondeo para x = 0.4.

Al realizar el calculo del error los resultados conseguidos fueron:

epsilon	7.45058059692383e-09
error	0.0750000476837159

Al observar los datos podemos reafirmar que el error de redondeo relativo es menor que la mitad del epsilon de la maquina.

7.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

8. Encuentre la representacion en número de maquina hexadecimal del número real 9.4

La representación hexadecimal del número 9.4 es igual a 5E

9. Encuentre las dos raıces de la ecuacion cuadratica $x^2 + 9^{12}x = 3$ Intente resolver el problema usando la arimetica de precision doble, tenga en cuenta la perdida de significancia y debe contrarestarla

Para hallar las raíces de la ecuación utilizamos la ecuación de segundo orden. Pero lo que ocurre con esta ecuación es que puede sufrir una cancelación de la raíz, haciendo que esta sea totalmente inexacta. Entonces se utilizó una variación de la ecuación de segundo orden para hallar las raíces de forma mas precisa.

$$\frac{-(b+sign(b)\sqrt{b^2-4ac})}{2a}$$

código en R

```
raicesd = function(a,b,c){
  raiz = c()
  if (b^2 - 4*a*c >= 0){
    raizd1 = (-b - sign(b)*sqrt(b^2 - 4*a*c))/(2*a)
    raizd2 = (2*c)/(-b - sign(b)*sqrt(b^2 - 4*a*c))
  }
  else{
    raizd1 = (-b - sign(b)*sqrt(as.complex(b^2 - 4*a*c)))/(2*a)
    raizd2 = (2*c)/(-b - sign(b)*sqrt(as.complex(b^2 - 4*a*c)))
  }
  raiz[1] = raizd1
  raiz[2] = raizd2

  return(raiz)

}

resultados = raicesd(1, 9^12, -3)

sprintf("%.60f",resultados[1])
sprintf("%.60f",resultados[2])
```

En este código se verifica primero si las raíces son enteras si $b^2 - 4ac \geq 0$ y si no entonces se le añade a la ecuación as.complex para ver su parte imaginaria. En cualquiera de los casos el código ejecuta la ecuación de segundo orden modificada.

Con esta ecuación se lograron los siguientes resultados.

```
[1] "-282429536481.000000000000000000000000000000000000000000000000000000000000"
> sprintf("%.60f",resultados[2])
[1] "0.000000000010622118484416449921007088996738332298264140263200"
```

2) **Raıces de una Ecuacion**: en esta sección se busca realizar unos ejercicios propuestos con el fin de evaluar el orden de convergencia de ciertas soluciones y realzar diferentes pruebas para constatar la validez de la solución.

a) 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 A_n. Imprima varias pruebas, para diferentes valores de n y exprese f(n) en notacion O() con una grafica que muestre su orden de convergencia.

```
# Sub matriz triangular

numeros = c(5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,60)

matriz_triangular <- function (numeros) {

  total <- c()

  for (i in numeros) {

    matriz_triangular = matrix (1 , ncol = i, nrow = i)
    aux = 0

    for (j in 1:i) {
      for (k in 1:j) {

        aux = aux + matriz_triangular[j,k]
      }
    }

    total = c(total , aux)
    aux = 0
  }

  return (total)
}

grafica = matriz_triangular (numeros)

plot (numeros , grafica , type = 'b',col = 1)
```

Este algoritmo funciona con un arreglo que contiene la cantidad de matrices

que se desee generar, posteriormente se utiliza un ciclo que crea una matriz triangular superior con unos abajo de la diagonal, dichas matrices se recorren y se suman los valores que contienen, estos valores después se guardan en una variable auxiliar (aux), lo que contenga esa variable auxiliar lo va ubicando en el número de filas y columnas en el que se encuentre, finalmete el total de las sumas se muestra respectivamente con el orden n adecuado de cada matriz.

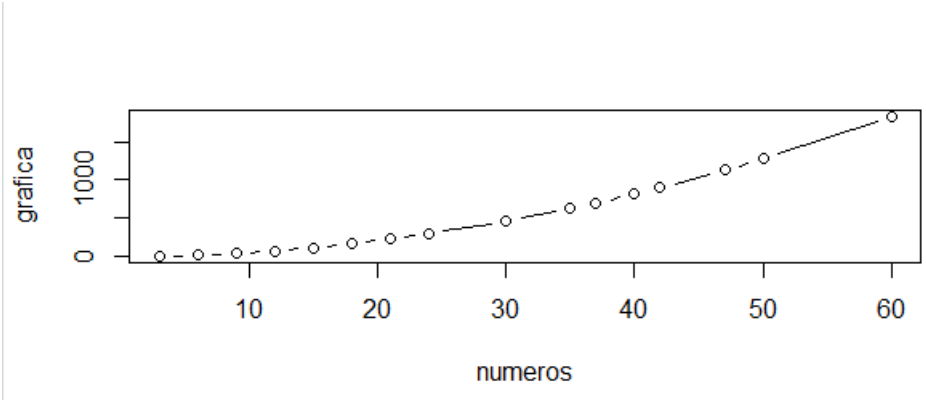
Para la primera prueba la selección de n fue:

```
numeros = c(3,6,9,12,15,18,21,24,30,35,37,40,42,47,50,60)
```

Los resultados fueron:

```
 6   21   45   78  120  171  231  300  465  630  703  820  903 1128 1275 1830
```

la gráfica de convergencia para estos n es:



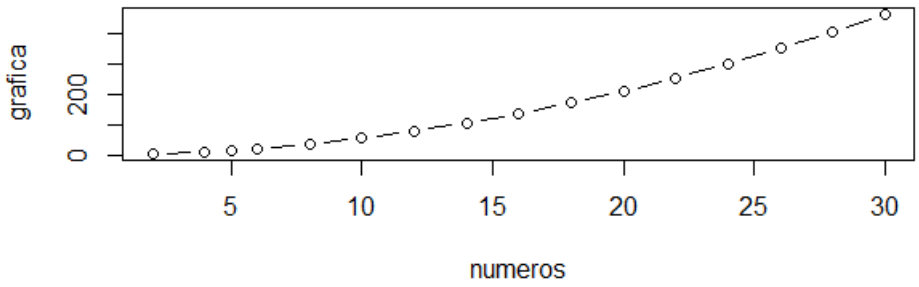
Para la segunda prueba la selección de n fue:

```
numeros = c(2,4,5,6,8,10,12,14,16,18,20,22,24,26,28,30)
```

Los resultados fueron:

```
 3  10  15  21  36  55  78 105 136 171 210 253 300 351 406 465
```

la gráfica de convergencia para estos n es:



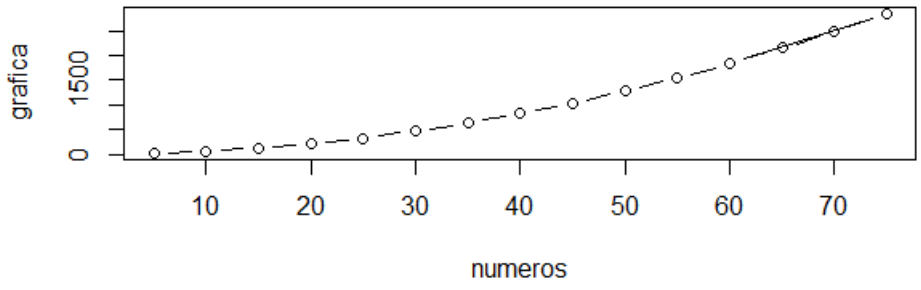
Para la tercera prueba la selección de n fue:

```
numeros = c(5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80)
```

Los resultados fueron:

```
15   55  120  210  325  465  630  820 1035 1275 1540 1830 2145 2485 2850 3240
```

la gráfica de convergencia para estos n es:



Como se muestra en las gráficas se puede evidenciar que su complejidad algorítmica es de $O(n^2)$.

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.

```r
# Sumar los n al cuadrado numeros

numero = c(seq(2,100,by=2))

suma <- function (numero) {

  total <- c()
  for (i in numero) {

    aux = 0

    for (j in 1 : i) {

      aux = aux + j^2
    }

    total = c (total, aux)
  }
  print(total)
  return (total)
}

graficar = suma (numero)
plot (numero , graficar , type = 'l', col ="red",main ="  Sumar numeros cuadrados")
```

En principio se tiene una función seq que crea una secuencia desde un valor predeterminado hasta un valor final y adicionalmente tiene un parámetro llamado Bay que se encarga de establecer el intervalo de salto en el que se desea recorrer la serie de números pedidos. Posteriormente lo que hace es tener una variable auxiliar que se encarga de guardar la suma de cada n^2 dado en la función hasta llegar al final, al igual que en el anterior punto la variable total guarda el auxiliar y muestra las sumas de cada n^2.

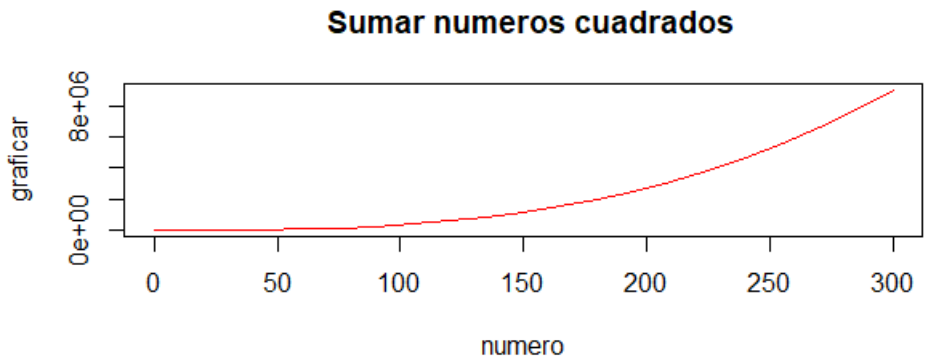
Para la primera prueba el n fue:

```r
numero = c(seq(0,300,by=15))
```

Los resultados fueron:

```
> graficar = suma (numero)
 [1]       1    1240    9455   31395   73810  143450  247065  391405  583220  829260
[11] 1136275 1511015 1960230 2490670 3109085 3822225 4636840 5559680 6597495 7757035
[21] 9045050
```

la gráfica de convergencia para estos n es:



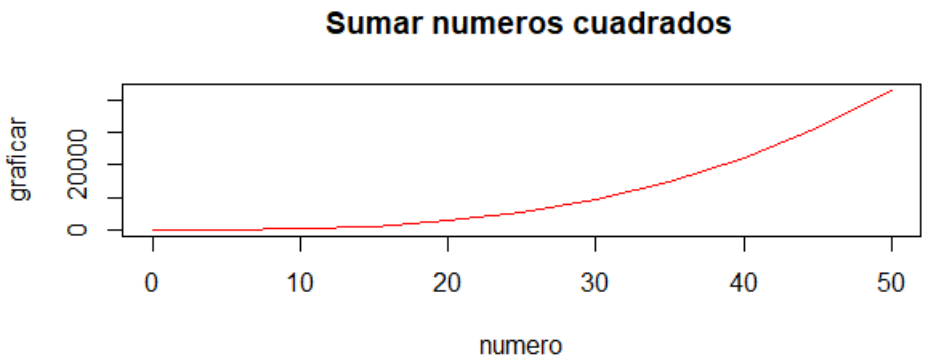
Para la segunda prueba el n fue:

```
numero = c(seq(0,50,by=5))
```

Los resultados fueron:

```
> graficar = suma (numero)
 [1]     1    55   385  1240  2870  5525  9455 14910 22140 31395 42925
```

la gráfica de convergencia para estos n es:



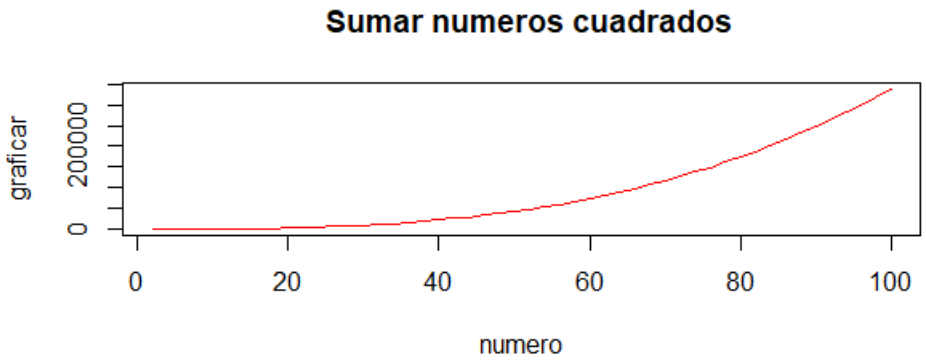
Para la tercera prueba el n fue:

```
numero = c(seq(2,100,by=2))
```

Los resultados fueron:

```
> graficar = suma (numero)
 [1]      5     30     91    204    385    650   1015   1496   2109   2870   3795
[12]   4900   6201   7714   9455  11440  13685  16206  19019  22140  25585  29370
[23]  33511  38024  42925  48230  53955  60116  66729  73810  81375  89440  98021
[34] 107134 116795 127020 137825 149226 161239 173880 187165 201110 215731 231044
[45] 247065 263810 281295 299536 318549 338350
```

la gráfica de convergencia para estos n es:



Como se muestra en las gráficas se puede evidenciar que su complejidad algorítmica es de $O(n^2)$.

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 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.

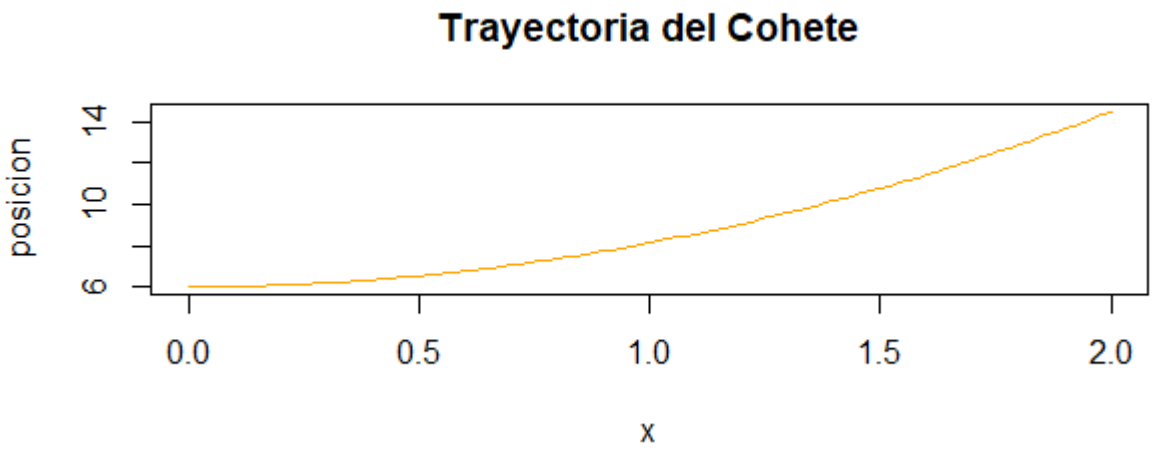
```
# Problema del Cohete
f = function()
{
  t = 0
  posicion = function(t){6+(2.13 * (t^2)) - (0.0013 * (t^4))}
  funcion = 6+(2.13 * (t^2)) - (0.0013 * (t^4))
  resultado = 6+2.13*((t+1)^2)-0.0013*((t+1)^4)
  arreglos = c(0)

  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(0,2), main = "Trayectoria del Cohete",col ="orange")

  cat("La altura mayor alcanzada es :",funcion," metros")


}
f()
```

Este algoritmo tiene una variable funcion que recibe la ecuación original de la trayectoria y luego se tiene la variable resultado que recibe la misma ecuación de la trayectoria pero buscando el valor siguiente a el t original. Después, se crea un ciclo que busca que cuando el valor que tiene la variable funcion se vuelve mayor que el valor de la variable resultado,entonces se tiene la altura máxima alcanzada

El resultado que se obtuvo en el algoritmo fue

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

La gráfica que se obtuvo para este resultado es:



3) **Convergencia de Métodos Iterativos**

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

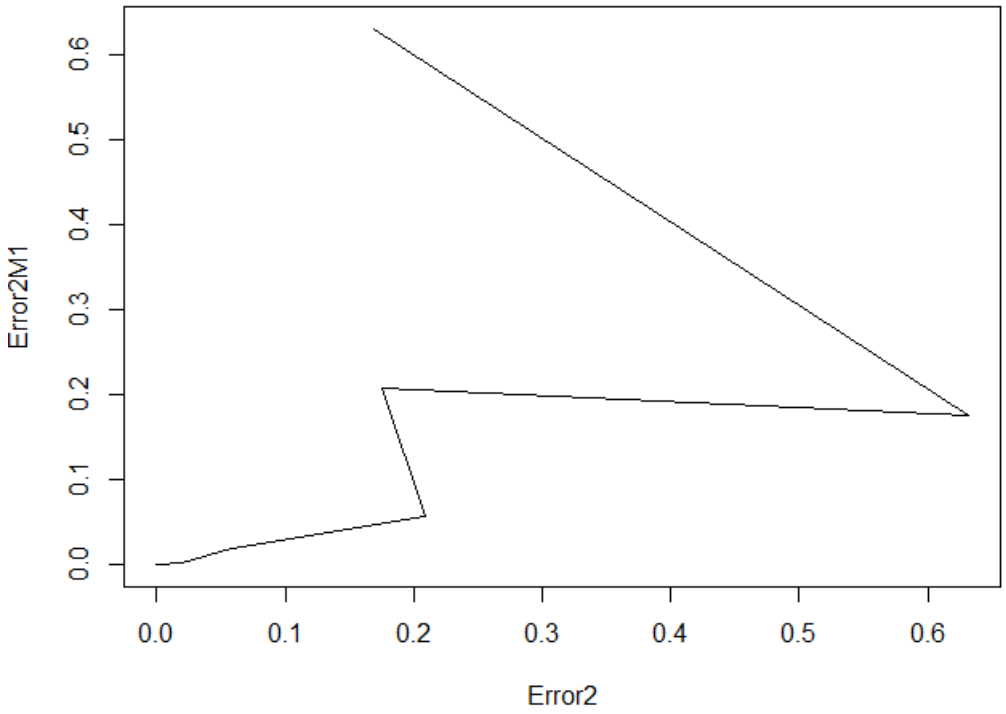
Resultado:

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

a. Utilice la siguiente formula recursiva con $E = 10^{16}$ para determinar aproximadamente el punto de interseccion:

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

Convergencia del método 1:



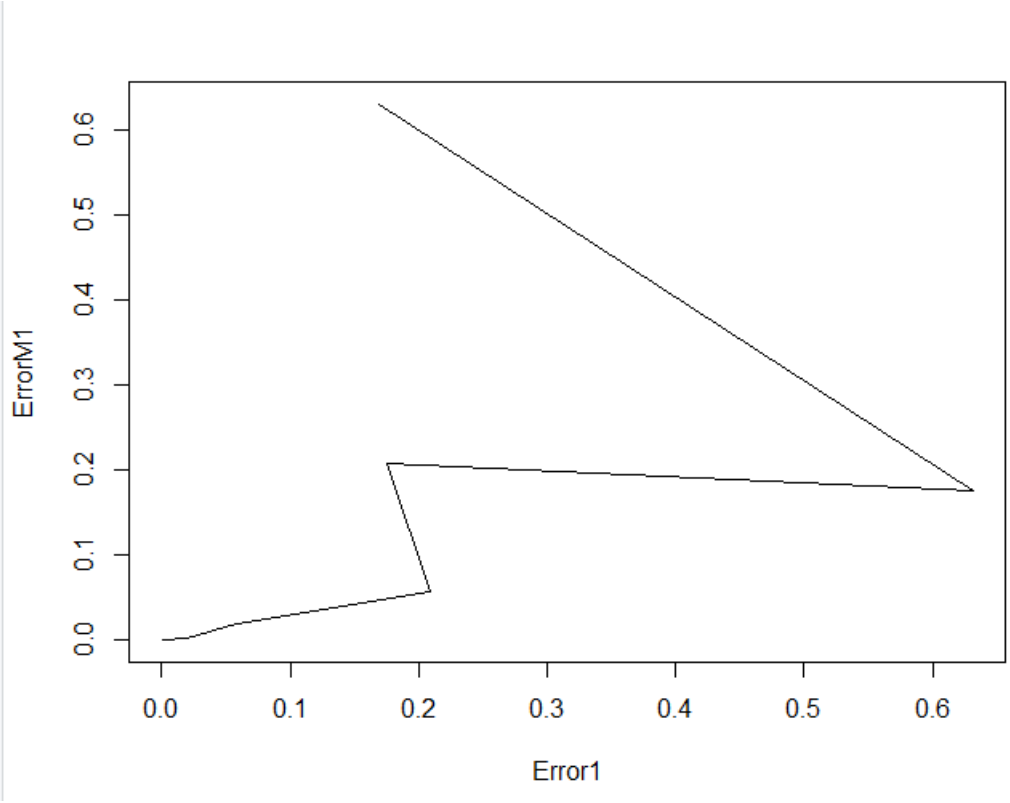
Resultados del error:

	Error2	Error2M1
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	1.194976e-07
9	1.194976e-07	8.604673e-12
10	8.604673e-12	*NA*

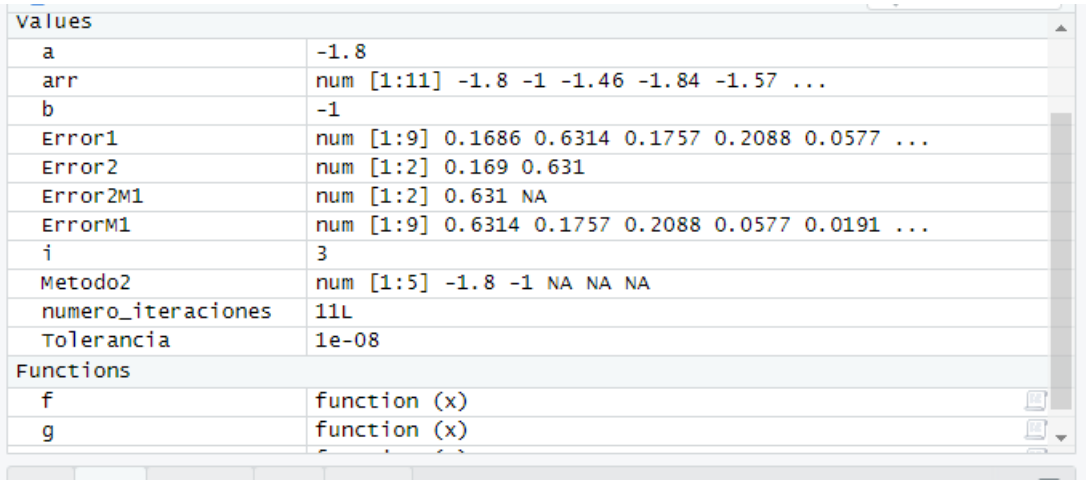
b. Aplicar el metodo iterativo siguiente con $E = 10^8$ para encontrar el punto de interseccion:

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

Resultados:



	Error2	Error2M1
1	0.1685564	0.6314436
2	0.6314436	NA



Values	
a	-1.8
arr	num [1:11] -1.8 -1 -1.46 -1.84 -1.57 ...
b	-1
Error1	num [1:9] 0.1686 0.6314 0.1757 0.2088 0.0577 ...
Error2	num [1:2] 0.169 0.631
Error2M1	num [1:2] 0.631 NA
ErrorM1	num [1:9] 0.6314 0.1757 0.2088 0.0577 0.0191 ...
i	3
Metodo2	num [1:5] -1.8 -1 NA NA NA
numero_iteraciones	11L
Tolerancia	1e-08
Functions	
f	function (x)
g	function (x)

ejercicio sea $f(x) = e^x - x - 1$ **1.** Demuestre que Tiene un cero de multiplicidad 2 en x = 0

Para que un cero tenga multiplicidad 2 el polinomio debe tener al menos orden dos para que alguno de sus ceros sea de multiplicidad dos. Por ejemplo, en $p(x) = (x-1)(x-3)^2$ el valor de x = 3 es un cero que posee una multiplicidad de 2 ya que este tiene un exponente al cuadrado. En el caso de $f(x)$ esto no es así, por lo que no es posible que la función tenga en x = 0 un cero de multiplicidad 2.

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

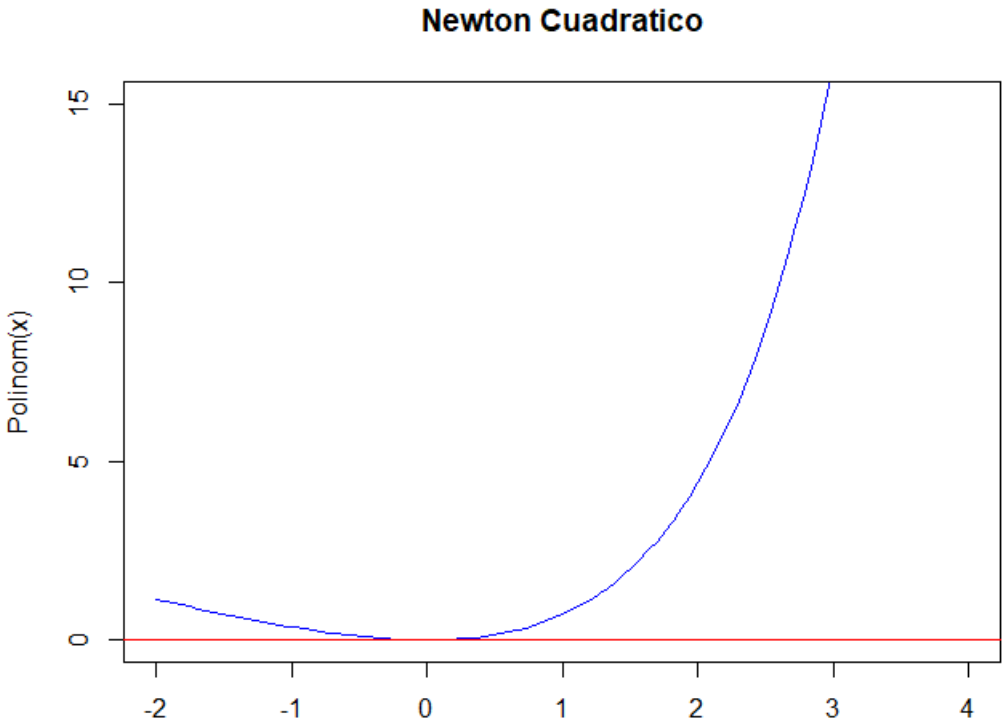
Para poder hallar esta encontramos la siguiente solución en Rstudio el cual nos permitió hallar el error y graficar el punto de convergencia cuando p sub 0 es igual a 1

```
polinomio = expression (exp(x) - x-1)
derivada = D(polinomio, "x")
#Variables
x = 0 ;
xaproximado = 0.000005
resultado = c(0)

i =1
#Inicio del while
while ( x != xaproximado) {

  x = xaproximado
  reemplazarPoli = eval(polinomio)

  remplazarDeri = eval(derivada)

  #Forma de Realizar Newton
  xaproximado = x - (reemplazarPoli/remplazarDeri)
  resultado[i] =x
  i = i +1
}
#Graficacion de la funcion

Polinom =function(x)(exp(x) - x-1)
curve(Polinom,-2,4,100, ylim = c(0,15),main = "Newton Cuadratico",col ="blue")
abline(0,0,col="red")

#Tabla

tabla = data.frame(resultado)
tabla
```

El cual esta solución nos mostró la siguiente información respecto a la convergencia de de la operación con respecto al método de newton.

	resultado
1	5.000000e-06
2	2.500006e-06
3	1.249988e-06
4	6.248829e-07
5	3.125409e-07
6	1.562420e-07
7	7.807836e-08
8	3.826420e-08
9	2.085540e-08
10	1.020854e-08

Gráfica de convergencia:



3. Utilizando el metodo de Newton generalizado, mejora la tasa de rendimiento? explique su respuesta

Dado que el orden de convergencia del método de Newton es dos y requiere una evaluación de función y una de su derivada, por lo que el método de Newton tiene una eficiencia de $2^{1/2} \approx 1{,}4142$, De manera similar el índice de eficiencia del método de Halley y el método de HouseHolder es $3^{1/3} \approx 1{,}4422$, porque ambos métodos requieren una función evaluaciones y dos evaluaciones derivadas y estos métodos logran un orden cúbico de convergencia. los El método generalizado de Newton Raphson tiene convergencia cúbica, requiere una evaluación de función y dos evaluaciones derivadas para que su índice de eficiencia sea $3^{1/3} \approx 1,4422$. Nuestro método de Newton Raphson generalizado modificado (algoritmo 3.1) desarrollado en este artículo tiene seis orden de convergencia, requiere dos evaluaciones de funciones y dos de su derivada para que el índice de eficiencia es $6^{1/4} \approx 1{,}5651$. Ahora, pasamos a calcular el índice de eficiencia de nuestro método de Newton Raphson generalizado libre

de segunda derivada (algoritmo 3.2) como sigue: El método generalizado de Newton Raphson libre de la segunda derivada necesita dos evaluaciones de la función y una de sus primeras derivadas. Entonces, el número total de evaluaciones de este método es tres. es decir Nf = 3.

Method	Number of function or derivative evaluations	Efficiency index
Newton, quadratic	2	$2^{\frac{1}{2}} \approx 1.4142$
Halley, Cubic	3	$3^{\frac{1}{3}} \approx 1.4422$
HouseHölder,Cubic	3	$3^{\frac{1}{3}} \approx 1.4422$
Generalized Newton Raphson, Cubic	3	$3^{\frac{1}{3}} \approx 1.4422$
Modified Generalized Newton Raphson's Method 6th order	4	$6^{\frac{1}{4}} \approx 1.5651$
Generalized Newton Raphson, free from second derivative 5th order	3	$5^{\frac{1}{3}} \approx 1.7100$

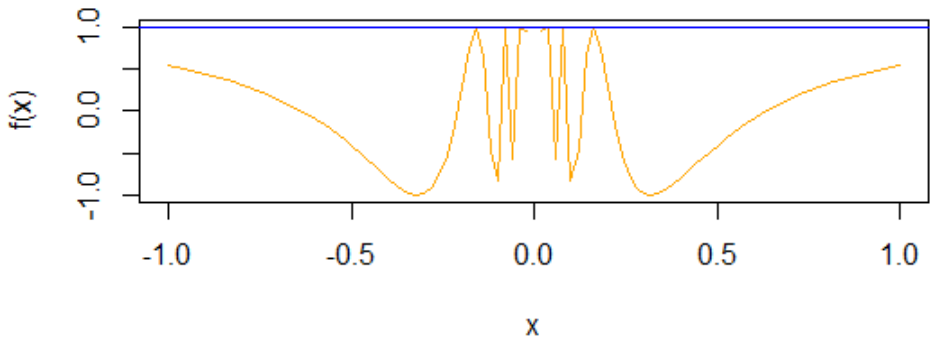
Además, en la sección anterior, hemos demostrado que el orden de convergencia de la escala generalizada de Newton Raphson método libre de la segunda derivada es cinco. es decir r = 5. Así, el índice de eficiencia del método generalizado de Newton Raphson libre de la segunda derivada es:

4) **Convergencia Acelerada**: En esta sección se busca usar el método de Aitken para resolver un ejercicio por medio de una sucesión

- Ejercicio: Dada la sucesión $[x_n]_{n=0}^{\infty}$ con $x_n = cos(\frac{1}{n})$

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

Para realizar la convergencia se tomó el algoritmo de bisección que es un método de convergencia lineal, para esto se usó un abline para saber si se intercepta con la función coseno por lo que se grafica en una escala de -1 a 1, y para la convergencia acelerada se usa el método delta al cuadrado de Aitken. Para hallar el valor de la primera aproximación a la raíz, mediante la fórmula del punto medio que toma el valor de a lo suma con el valor de b y ese resultado lo divide en 2, así con el método de bisección.



2.Compare los primeros términos con la sucesión $Ax_n][x_n]_{n=0}^{\infty}$

Para comparar los primeros términos se tiene la tabla de resultados que da el método de Aitken, para esto se usan los errores, resultados y el valor que da el algoritmo en dicho método. Los valores obtenidos en el resultado son los obtenidos por el método de punto fijo en el algoritmo de bisección, se comparan esos resultados para observar cuál numéricamente converge más rápido. El error es valor absoluto obtenido por los resultados y el error mas uno toma el resultado del error en la posición 2 hasta que halle todos los errores al momento en que encuentre la aproximación de la raíz más cercana.

	error	errormas1	resultados	aiken
39	2.728484e-12	1.364242e-12	0.6366198	0.6366198
40	1.364242e-12	6.821210e-13	0.6366198	0.6366198
41	6.821210e-13	3.410605e-13	0.6366198	0.6366198
42	3.410605e-13	1.705303e-13	0.6366198	0.6366198
43	1.705303e-13	8.526513e-14	0.6366198	0.6366198
44	8.526513e-14	4.263256e-14	0.6366198	0.6366198
45	4.263256e-14	2.131628e-14	0.6366198	0.6366198
46	2.131628e-14	1.065814e-14	0.6366198	0.6366198
47	1.065814e-14	5.329071e-15	0.6366198	0.6366198
48	5.329071e-15	2.664535e-15	0.6366198	0.6366198
49	2.664535e-15	1.332268e-15	0.6366198	0.6366198
50	1.332268e-15	6.661338e-16	0.6366198	*NA*
51	6.661338e-16	*NA*	0.6366198	*NA*

Como se observa en la tabla se encuentran los valores mencionados, pero podemos notar qué el método de Aitken converge más rápido que el método de bisección a pesar de que no tienen gran diferencia en iteraciones.

3. Sean $f(t) = 3sin^3t–1$ y $g(t) = 4sin(t)cos(t)$ para $t \geq 0$ las ecuaciones paramétricas que describe el movimiento en una partícula. Utilice un método de solución numérico con error de 10^-16 para determinar como las coordenadas coinciden y muestre gráficamente la solución

```
> vectorg
 [1]  2.375448  7.049300  8.658656 13.332449 14.941690 19.615684 21.225004 25.898855 27.508191
[10] 32.182065 33.791375 38.465249 40.074564 44.748427 46.357763 51.031623 52.640940 57.314812
[19] 58.924243 63.598023 65.207333 69.881183 71.490504 76.164541 77.773730 82.447521 84.056886
[28] 88.730738 90.340069 95.013897 96.623251
> ###############
> vectorh
 [1]  1.570795  3.141595  4.712382  6.283191  7.854035  9.424778 10.995449 12.566364 14.137174
[10] 15.707961 17.278760 18.849556 20.420349 21.991154 23.561939 25.132745 26.703539 28.274333
[19] 29.845132 31.415919 32.986730 34.557680 36.128316 37.699076 39.269903 40.840711 42.411498
[28] 43.982299 45.553093 47.123888 48.694690 50.265476 51.836284 53.407100 54.977871 56.548840
[37] 58.119457 59.690268 61.261037 62.831853 64.402645 65.973442 67.544248 69.115034 70.685838
[46] 72.256631 73.827427 75.398226 76.969013 78.539823 80.110711 81.681552 83.252134 84.822995
[55] 86.393805 87.964592 89.535392 91.106187 92.676981 94.247784 95.818570 97.389377 98.960177
> vectorsolu
[1] 0
```

Al probar e implementar los códigos con rangos de números determinados, se guardan las raíces en dos vectores y se compara el contenido de ambos con un ciclo para llenar un arreglo nuevo y si ambos tienen la misma raíz se mostrará en el vectorsolu que hay semejanzas, y como notamos en la imagen no hay raíces, por lo que el arreglo queda vacío.

Método de Steffensen: El método de Steffensen tiene una convergencia cuadrática. Aplica el método de Aitken a una sucesión que converge linealmente obtenida de la iteración de punto fijo y así es como se acelera la convergencia.

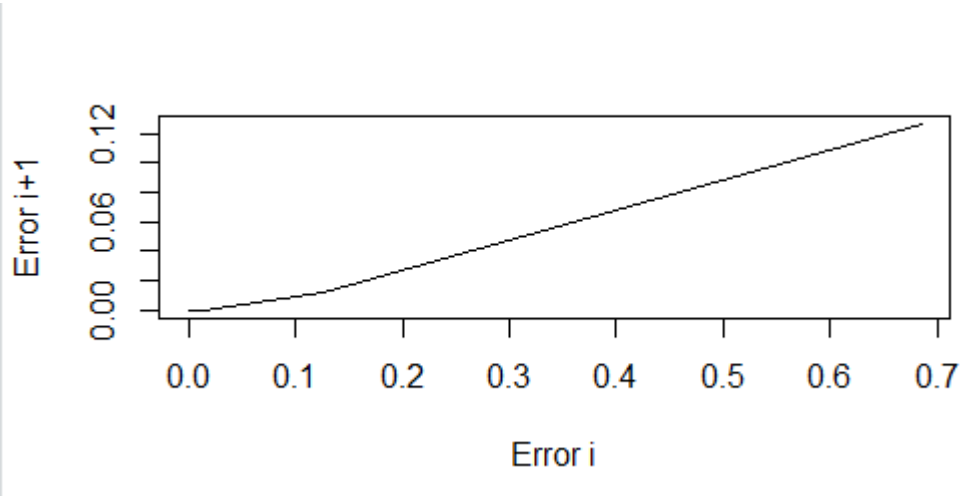
Ejercicio: Utilice el algoritmo de Steffensen para resolver $x^2 - cos(x)$ y compararlo con el método de Aitken con Tolerancia de $10^{-8}, 10^{-16}$, realice una gráfica que muestre la comparación entre los métodos.

La función de Steffensen tiene como parámetros la función a evaluar, un x_0 y el valor de la tolerancia; además de esto dicha función calcula el error del i y el $i + 1$.

Evaluando el $x_0 = 0$ y una primera tolerancia de 10^{-8} el resultado fue:

```
> Steffensen(f, 0, 1e-8)
i                x_i              f(x)             Error est.
 1.00000000     -0.68507336     -0.30504713      0.68507336
 2.00000000     -0.81136839     -0.03018801      0.12629504
 3.00000000     -0.82400763     -0.00029700      0.01263924
 4.00000000     -0.82413230     -0.00000003      0.00012467
 5.00000000     -0.82413231     -0.00000000      0.00000001
 6.00000000     -0.82413231      0.00000000      0.00000000
Cero de funcion:  -0.8241323  con error <= 2.220446e-16 Iteraciones:  6
```

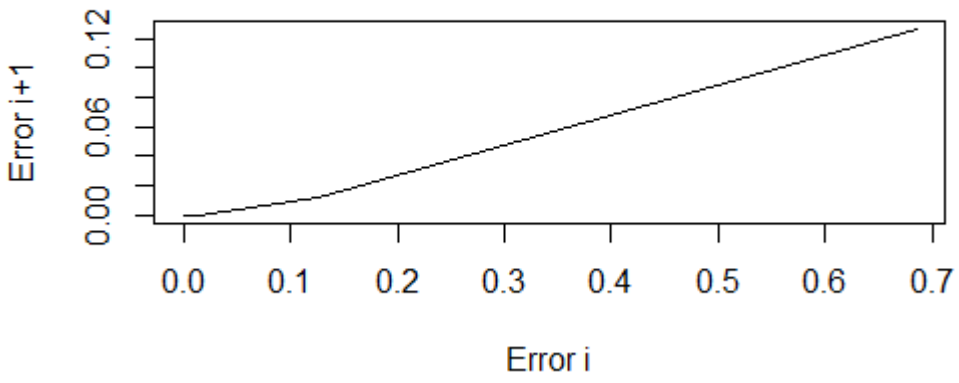
La grafica del error de i e $i + 1$ es:



Evaluando el $x_0 = 0$ y una primera tolerancia de 10^{-16} el resultado fue:

```
> Steffensen(f, 0, 1e-16)
i              x_i             f(x)            Error est.
 1.00000000    -0.68507336     -0.30504713      0.68507336
 2.00000000    -0.81136839     -0.03018801      0.12629504
 3.00000000    -0.82400763     -0.00029700      0.01263924
 4.00000000    -0.82413230     -0.00000003      0.00012467
 5.00000000    -0.82413231     -0.00000000      0.00000001
 6.00000000    -0.82413231      0.00000000      0.00000000
 7.00000000    -0.82413231     -0.00000000      0.00000000
 8.00000000    -0.82413231     -0.00000000      0.00000000
Cero de funcion:  -0.8241323  con error <= 0 Iteraciones:  8
```

La grafica del error de i e $i + 1$ es:



La función Aitken pide como parámetros la función a evaluar, el valor inicial, el numero máximo de iteraciones y por ultimo la tolerancia, comparando con el método de Steffensen se introdujo el mismo valor inicial $x_0 = 0$, el mismo número de iteraciones que arrojó el método de Steffensen al implementarlo, y las tolerancias pedidas.

Se utilizo esta función de la librería Pracma debido a que solamente se necesitaba comparar el resultado que daba el método de Aitken con el método de Steffensen, a pesar de que se tenia otro metodo de .Aitken.este otro método implementaba Bisección y Aitken y no se considero competente para la comparación, por lo que optando por la función de la librería Pracma y utilizándola de manera correcta no se llego al resultado adecuado que si arrojaba el método Steffensen y que se verifico externamente en Geogebra.

Con base en lo anterior el resultado para el método de Aitken con una tolerancia de 10^{-8} fue:

```
> aitken(f,0, 6, 1e-8)
[1] -0.5500094
```

El resultado para el método de Aitken con una tolerancia de 10^{-16} fue:

```
> aitken(f,0, 8, 1e-16)
[1] -0.5500094
```

Referenciass acelerar l
- (2020). punto fijo, podemoRetrieved 24 September 2020, from $https://www.emis.de/journals/T$
$-2831_Generalized_Newton_Raphsons_method.pdf$ Multiplicidad de ceros de polinomios (video) — Khan Academy. (2020).

- Retrieved 25 September 2020, from https://es.khanacademy.org/math/algebra2/x2ec2f6f830c9fb8
graphs/x2ec2f6f830c9fb89:poly-intervals/v/polynomial-zero-multiplicity: :text=El

-$Pérdidadeimportancia-Lossofsignificance-qwe.wiki.$(2020).$Retrieved25September$2020, $from$
$//es.qwe.wiki/wiki/Loss_of_significance$ $RPubs-Ecuacióncuadrática$: $fórmulageneral.$(2020).R
$//rpubs.com/edisonlmg/formula_general$

-(2020). Retrieved 25 September 2020, from https://matematica.laguia2000.com/general/metodo-de-biseccion

- R Language - seq () — r Tutorial. (2020). Retrieved 25 September 2020, from https://riptutorial.com/es/r/example/8499/seq—: :text=R

- Double precision (64-bit) representation of numeric value in R (sign, s., Stubner, R., CEVHER, E. (2020). Double precision (64-bit) representation of numeric value in R (sign, exponent, significand). Retrieved 25 September 2020, from https://stackoverflow.com/questions/50217954/double-precision-64-bit-representation-of-numeric-value-in-r-sign-exponent

- aitken function — R Documentation. (2020). Retrieved 25 September 2020, from https://www.rdocumentation.org/packages/pracma/versions/1.9.9/topics/aitken