Taller 1

John Jairo Gonzalez Martinez
Daniel Esteban Tibaquira Galindo
Karen Sofia Coral Godoy
Stiven Gonzalez Olaya

26 de Septiembre 2020

1 Numero de Operaciones

1.1 Evaluar el polinomio en $x = 1.00000000001$ con $P(x) = 1 + x + x^2 + ... + x^{50}$ y la primera derivada. Encuentre el error de cálculo al compararlo con los resultados de la expresión equivalente $Q(x) = (x^{51} - 1)/(x - 1)$

Esta solución se desarrollo por el método de Horner con un numero de multiplicaciones totales de 99.0. y un valor $P(1.00000000001) = 51.00000001275002$, y el valor de $P'(1.00000000001) = 1275.0000004165004$. Para el valor de $Q(1.00000000001) = 51.0$. Esto genera un error de $1.2750021483043383x10^{-8}$.

1.2 Encuentre los primeros 15 bits en la representación binaria de π

Con un valor de $\pi = 3.141592653589793$ se comenzo hacer la conversión a binario y dio el siguiente resultado: 11.0010010000111

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

$$1010101 = 85$$
$$1011.101 = 11.625$$
$$10111.010101... = 23.333332061767578$$
$$111.1111... = 7.9999998807907104$$

1.4 Convierta los siguientes números de base 10 a binaria: 11.25; $\frac{2}{3}$; 30.6; 99.9

$$11.25 = 1011.0100000000$$
$$\frac{2}{3} = 0.1010101010$$

$$30.6 = 11110.1001100110$$
$$99.9 = 1100011.1110011001$$

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

Según la norma IEEE-754 un número infinito (+inf o -inf) se representa de forma que los bits correspondientes al exponente mínimo se colocan todos en 1 (8 para 32 bits, 11 para 64 bits) y el bit del signo indicará el signo del infinito.

1.6 ¿Cuál es la diferencia entre redondeo y recorte?

En el proceso de redondeo se usa el valor de la siguiente para obtener un resultado más aproximado del valor. El recorte (truncamiento) corta el número de cifras decimales sin tener en cuenta las siguientes para el valor de la última cifra. Se puede decir que el redondeo es más exacto en el valor decimal final, sin embargo, un redondeo a n decimales y corte a n decimales podrán ser los mismos dependiendo de la regla usada para el redondeo.

1.7 Indique el número de punto flotante (IEEE) de precisión doble asociado a x, el cual se denota como $fl(x)$ para $x(0.4)$

Según el proceso para convertir un número decimal a su representación en binario el valor será:

$$0.4 * 2 = 0.8 \rightarrow 0$$
$$0.8 * 2 = 1.6 \rightarrow 1$$
$$0.6 * 2 = 1.2 \rightarrow 1$$
$$0.2 * 2 = 0.4 \rightarrow 0$$
$$0.8 * 2 = 1.6 \rightarrow 1$$
$$...$$
$$= 0.0110011001100110011001100110011...$$
$$= 1.1001100110011001100110011... * 2^{-2}$$

Sin embargo, el carácter de este número es periódico infinito, por lo que para poder ser representado en un computador se utiliza la norma IEEE 754 para representaciones de números en coma flotante.
Usando un tamaño de número de 64 bits, divididos en 1 bit para el signo, 11 bits para el exponente mínimo (Biased Exponent) y 52 bits para la precisión de la fracción (Fraction) se encuentra que:

$Signo = 0$
$BiasedExponent = (1021)_{10} = (01111111101)_2$
$Fraction = (1001100110011001100110011001100110011001100110011001)$

Por lo tanto, se tiene que el valor de $fl(0.4)$ es igual en el estandar IEEE 754 a:

$$(0.4)_{10} \approx (00111111110110011001100110011001...)_2$$
$$= (0.39999999999999996693309261245)_{10}$$

1.8 Error de redondeo:

En el modelo de la aritmética de computadora IEEE, el error de redondeo relativo no es más de la mitad del épsilon de máquina:

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

Teniendo en cuenta lo anterior, encuentre el error de redondeo para $x = 0.4$
Aplicando la formula para el error de conversión al estándar IEEE 754 se obtiene:

$$\frac{|0.39999999999999996693309261245-0.4|}{|0.4|} \leq \frac{2^{52}}{2}$$
$$\frac{|0.39999999999999996693309261245-0.4|}{|0.4|} \leq 2^{53}$$
$$8.32667268468875 * 10^{-17} \leq 1.110223 * 10^{-16}$$

El error de redondeo es por lo tanto $\approx 8.327 * 10^{-17}$ o $8.327 * 10^{-15}\%$

1.9 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

En R se maneja en todos los casos la doble precisión, a diferencia de Phyton que dependiendo de las variables hay doble precisión o no.

1.10 Encuentre la representación en número de máquina hexadecimal del número real 9.4

La representación del numero 9.4 en hexadecimal es $0x9.6666666666666$

1.11 Encuentre las dos raíces de la ecuación cuadrática $x^2 + 9^{12}x = 3$ Intente resolver el problema usando la arimética de precisión doble, tenga en cuenta la pérdida de significancia y debe contrarestarla.

Para encontrar el valor de las raices se utilizo el método de Newton-Horner donde nos dio el valor de $X_1 = 1.062211848441645x10^{-11}$ y $X_2 = -282429536481.0$ con una tolerancia de 10^{-32}

1.12 Explique cómo calcular con mayor exactitud las raíces de la ecuación:

$$x^2 + bx - 10^{-12} = 0$$

Donde b **es un número mayor que 100**

2 Raíces de una Ecuación

2.1 Implemente en R o Python un algoritmo que le permita sumar únicamente los elementos de la submatriz triangular superior o triangular inferior, dada la matriz cuadrada A_n. Imprima varias pruebas,para diferentes valores de n y exprese $f(n)$ en notación $O()$ con una gráfica que muestre su orden de convergencia.

Para resolver el problema se implemento el siguiente algoritmo:

```
Data: n
Result: Retorna la suma de los elementos de la submatriz triangular
        superior o inferior, de una matriz cuadrada A_n
initialization;
mat= matrix(1, nrow=z, ncol= z)
cont= 0
numop= 0
for i en 1: z do
    for j en 1: i do
        cont += mat[i, j]
        numop += 1
    end
end
numop -= 1
return numop
```

Algorithm 1: Algoritmo sumaTriangular

Matriz *nxn*	Operaciones
2x2	2
4x4	9
8x8	35
12x12	77
16x16	135
20x20	209
24x24	299
26x26	350
27x27	377
28x28	405
32x32	527
36x36	665
40x40	819

Tabla 1: Resultados algoritmo sumaTriangular

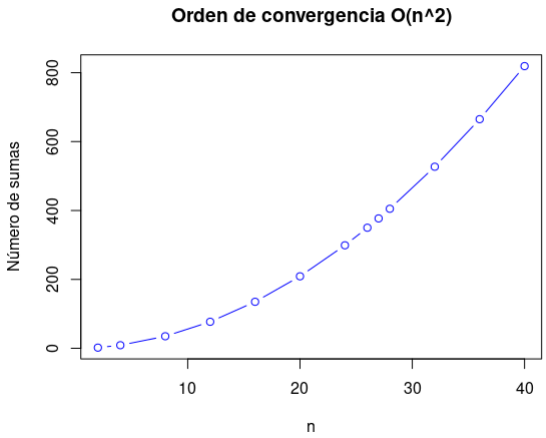

Figura 1: Grafica Orden de convergencia algoritmo sumaTriangular

Al aplicar la regresión polinomica de grado 2 a la grafica de convergencia se obtiene la ecuacion de la recta $y = \frac{1}{2}x^2 + \frac{1}{2}x - 1$ con un $R^2 = 1$ lo que indica que este algoritmo es de orden de convergencia cuadratica $O(n^2)$.

Tambien podemos aplicar la formula:

$$(\frac{n*(n+1)}{2}) - 1$$

2.2 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.

Para resolver el problema se implemento el siguiente algoritmo:

Data: z
Result: Retorna la suma de los n^2 primeros numeros naturales al cuadrado
initialization;
cont= 0
for *i en 1: z* **do**
  cont += i^2
end
return *cont*

Algorithm 2: Algoritmo suma n^2 numeros naturales

n	Suma Total
2	5
4	30
5	55
8	204
12	650
16	1496
20	2870
24	4900
28	7714
32	11440
36	16206
40	22140

Tabla 2: Resultados algoritmo nCuadrado

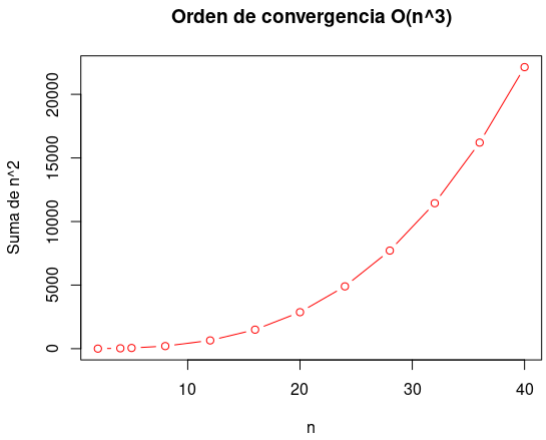

Figura 2: Grafica Orden de convergencia algoritmo nCuadrado

Al aplicar la regresión polinomica de grado 3 a la grafica de convergencia se obtiene la ecuacion de la recta $y = \frac{1}{3}x^3 + \frac{1}{2}x^2 + \frac{1}{6}x + 1e^{-9}$ con un $R^2 = 1$ lo que indica que este algoritmo es de orden de convergencia cubica $O(n^3)$.

Tambien podemos aplicar la formula:

$$(\tfrac{1}{6}n) * (n+1) * (2n+1)$$

2.3 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 métodos de solución de

ecuación no lineal, encuentre la altura máxima que alcanza el cohete. Para poder resolver este ejercicio hace falta hacer uso de la primera derivada, pues esta nos permitirá hallar el punto donde la raíz de la funcion es 0, es decir, al altura máxima a la que llega el cohete. Tal derivada la vemos representada como:

$$y'(t) = 4.26t - 0.0052t^3$$

Uno de los métodos que decidimos utilizar para hallar la respuesta fue mediante bisección que utilizando tanto $y(t)$ como $y'(t)$ nos permitió encontar que la respuesta es que la altura máxima alcanzada por el cohete corresponde a 28.622207641601562. El algoritmo fue implementado de la siguiente manera:

Data: F, x_0, x_1, e
Result: Raiz de la funcion F
while $x_1 - x_0 \geq e$ **do**
  $x_2 \leftarrow \frac{x_0+x_1}{2}$
end
if $f(x_2 = 0)$ *then* **then**
  return
end
x_2 **else**
end
$f(x_0) * f(x_2) > 0$ $x_0 = x_2$ **else**
  $x_1 = x_2$
end
return x_2

Algorithm 3: Algoritmo Bisección

Gráficamente podemos analizar la gráfica de ambas funciones para comprender cómo estas se comportan y determinar con certeza el punto más alto:

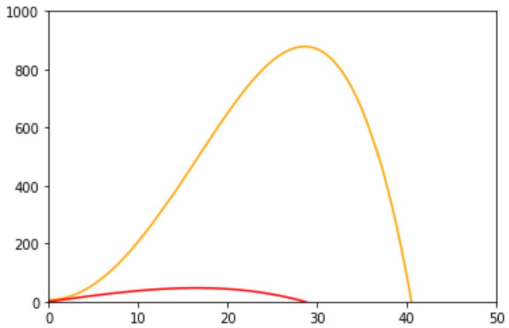

Figura 3: Grafica de la función $y(t) = 6 + 2,13t^2 - 0.0013t^4$ en rojo y $y'(t) = 4.26t - 0.0052t^3$ en amarillo

3 Convergencia de Metodos Iterativos

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

3.1.1 Utilice la siguiente formula recursiva con $E = 10^{-16}$ para determinar aproximadamente el punto de intersección:

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

```
Data: f, a, b, N, E
Result: Retorna la raíz de la función f en el intervalo a, b, teniendo en
        cuenta el número máximo de iteraciones N y E
if f(a) * f(b) ≥ 0 then
 | return None
end
aN = a
bN = b
for i en 1, N+1 do
 | mN = aN - f(aN)*(bN - aN)/(f(bN) - f(aN))
 | fmN = f(mN)
 | if |mN − a| < E then
 |  | f
 | end
 | if f(aN) * fmN < 0 then
 |  | aN = aN; bN = mN
 | end
 | if f(bN) * fmN < 0 then
 |  | aN = mN; bN = bN
 | end
 | if fmN == 0 then
 |  | return mN
 | end
 | else
 |  | return None
 | end
end
return  aN - f(aN)*(bN - aN)/(f(bN) - f(aN))
```

Algorithm 4: Algoritmo Intersección 1

La intersección de las funciones corresponde al punto en el que su resultado es el mismo, es decir donde $ln(x+2) = sin(x)$, de esta forma el problema tiene cierta similitud con el problema de encontrar las raíces de una función. Al hallar la raíz de la función que cumple la restricción anterior ($f(x) = ln(x+2) - sin(x)$), encontraríamos la intersección entre ambas funciones. En

cuanto al algoritmo implementado nos permitió encontrar el punto de intersección -1.6314435969688847. Utilizando la fórmula requerida y cumpliendo las condiciones de las funciones, fue:

3.1.2 Aplicar el método iterativo siguiente con $E = 10^{-8}$ para encontrar el punto de intersección:

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

Del mismo modo que fue resulto el punto anterior, para encontrar el resultado se igualaron las funciones. El resultado obtenido fue -1.6314436002809. El algoritmo utilizado fue el siguiente:

```
Data: f, a, N, E
Result: Retorna la raíz de la función f tomando como referencia el
        punto a, y teniendo en cuenta el número máximo de
        iteraciones N y E
xn = a
dp = [0]
for n en 1, N+1 do
    fxn = f(xn)
    if |f(xn)| < E then
        return xn
    end
    xn = xn - fxn*(xn-dp[n - 1]/f(xn)-f(dp[n-1]))
    dp.append(xn)
end
return None
```

Algorithm 5: Algoritmo Intersección 2

3.2 Newton: determine el valor de los coeficientes a y b tal que $f(1) = 3$ y$f(2) = 4$ con $f(x) = a(ax + b)e^{ax+b}$. Obtenga la respuesta con $E = 10^{-8}$

3.3 Sea $f(x) = e^x - x - 1$

3.3.1 Demuestre que tiene un cero de multiplicidad 2 en $x = 0$

Utilizando el método de Newton utilizando diferentes x_0 hicimos uso del siguiente algoritmo para determinar la multiplicidad: Como se puede ver el algoritmo necesita de una función llamada polinomio, esta corresponde a aquella en la que se realiza el método de newton, se define un $x_0 = 0$ para cada nueva comprobación y se mantiene constante la tolerancia como $E = 10^-16$. Luego de la verificación fue posible hallar que en efecto la función tiene un 0 de multiplicidad 2.

Data: $y, multiplicidad$
Result: Verdadero si el resultado corresponde al mismo de la multiplicidad
p = polinomio(0)
resultado = (y - p) $^{m}ultiplicidad$
if $resultado == p$ **then**
  return *True*
end
return *False*

Algorithm 6: Algoritmo Verificación de multiplicidad

3.3.2 Utilizando el método de Newton con $p_0 = 1$ verifique que converge a cero pero no de forma cuadrática

Haciendo uso de la función abline podemos encontrar la gráfica, que corresponde a:

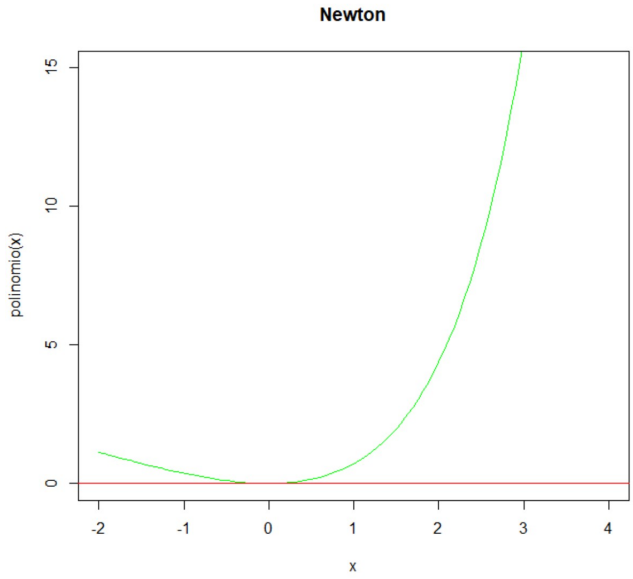

Figura 3: Gráfica Newton

Como es posible apreciar, en la línea roja esta no tiene un comportamiento cuadrático, sino que tiende a 0 mientras que la verde, visualmente tiende a presentar tal comportamiento. En cuanto a los resultados encontrar que una de las raíces fue $1.848537046e-08$ y su error se puede comprenderse a través de la gráfica:

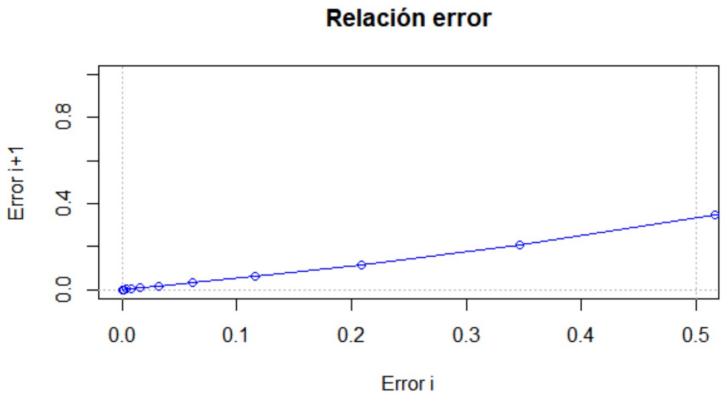

Figura 3: Gráfica Newton

3.3.3 Utilizando el método de Newton generalizado, ¿mejora la tasa de rendimiento? Explique su respuesta

En cuanto al método de Newton Generalizado es necesario definirlo, su fórmula la encontramos dada por: $x_{k+1} = x_k - \frac{f'(x_k - \sqrt{f''(x_k) - 2f(x_k)f''(x_k)})}{f''(x_k)}$. Un par de comparaciones que podemos realizar es que el número de iteraciones es menor utilizando Newton mejorado, según que:
Utilizando Newton tradicional como el que ha sido anteriormente implementado conseguimos un respuesta de que existe una raíz en 6.315492528044356530 y fue encontrada después de 53 iteraciones. Utilizando la contraparte, Newton Generalizado, el resultado conseguido fue: 8.264708341893786096 necesitando de 51 iteraciones. Ambos métodos fueron probados tomando x_0 con igual valor e igual tolerancia. En cuanto al comportamiento del algoritmo Newton Generalizado la gráfica del error resultó de la siguiente manera:

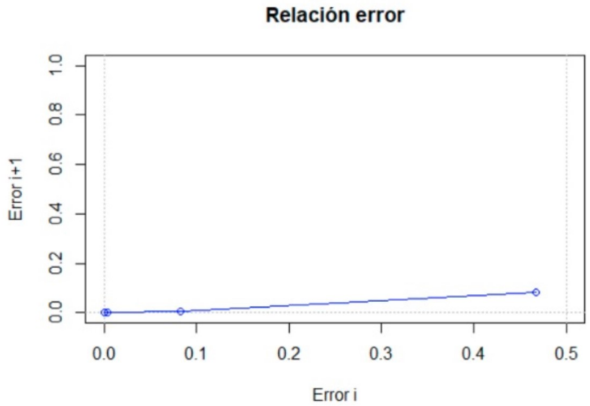

Figura 3: Gráfica Newton

4 Convergencia Acelerada

4.1 Dada la sucesión $\{x_n\}_{n=0}^{\infty}$ con $x_n = cos(1/n)$

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

Para calcular el orden de convergencia se usó la siguiente expresión

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

donde es necesario que existan un α y un λ positivos, donde α representa el orden de convergencia y λ una constante de error asintótico y donde x representa el punto de convergencia de la sucesión x. Para el caso especifico de la sucesión $\{x_n\} = cos(\frac{1}{n})$ se obtuvo un valor de convergencia de $x = 1$ luego de resolver el limite $\lim_{n\to\infty} cos(\frac{1}{n}) = 1$, así entonces, conociendo la convergencia de la sucesión es posible evaluar la expresión anteriormente señalada de forma tal que:

$$\lim_{n\to\infty} = \frac{|cos(\frac{1}{n+1}) - 1|}{|cos(\frac{1}{n}) - 1|^{1}} = 1$$

donde se evidencian los valores de $\alpha = 1$ y $\lambda = 1$. De esta forma con el valor de α es posible decir que la sucesión $\{x_n\}$ posee un orden de convergencia Lineal

4.1.2 Compare los primeros términos con la sucesión $\{A_n\}_{n=0}^{\infty}$

Luego de aplicar el método Aitken para la sucesión $\{x_n\} = cos(\frac{1}{n})$ y comparar dichos resultados con los obtenidos al evaluar de forma tradicional la sucesión $\{x_n\}$ se obtuvo lo siguiente

n	$\{A_n\}_{n=0}^{\infty}$	$\{x_n\}$
1	0.9617751	0.5403023
2	0.9821294	0.8775826
3	0.9897855	0.9449569
4	0.9934156	0.9689124
5	0.9954099	0.9800666
6	0.99662	0.9861432
7	0.9974083	0.9898133
8	0.9979502	0.9921977
9	0.9983384	0.9938335
10	0.9986261	0.9950042

Tabla 2: Comparación de Aitken con la sucesión

Estos datos se observan de forma mas clara en la siguiente gráfica

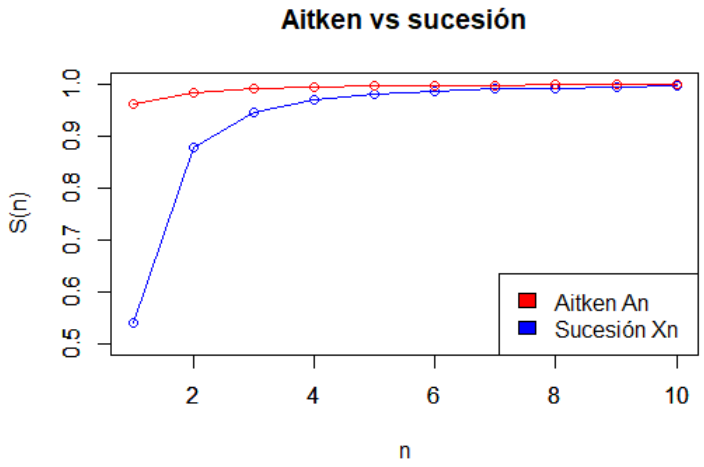

Figura 4: Aitken vs. Sucesión

donde se evidencia que el método de Aitken ofrece un mejor resultado que la forma tradicional de evaluar sucesiones al momento de llegar al punto de convergencia de la sucesión.

4.1.3 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 donde las coordenadas coinciden y muestre gráficamente la solución

Para determinar el punto de intersección entre $f(t)$ y $g(t)$ se realizó la siguiente operación

$$3sin^3(t) - 1 = 4sin(t)cos(t)$$

luego de esto se llegó a la siguiente expresión

$$3sin^3(t) - 1 - 4sin(t)cos(t) = 0$$

Obteniendo la función

$$f(t) = 3sin^3(t) - 1 - 4sin(t)cos(t)$$

de forma tal que ahora es posible aplicar alguno de los métodos para hallar raíces vistos anteriormente. Para este caso se utilizó el método de Newton-Raphson para hallar la raíz de la función anteriormente señalada. El resultado arrojado por este método es el siguiente

$$t = 1.18649502158199$$

Lo que significa que las funciones $f(t)$ y $g(t)$ se intersectan en dicho punto como se evidencia en la siguiente gráfica

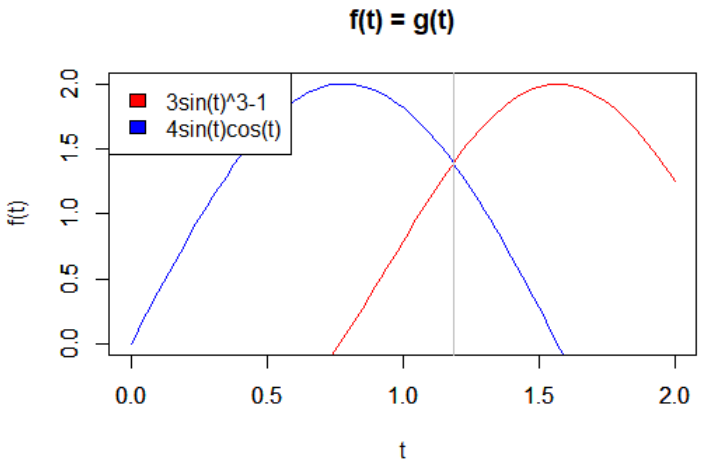

Figura 5: f(t) = g(t)

Por ultimo cabe resaltar el comportamiento del error para este procedimiento a medida de cada iteración. Para este caso se trabajó con una tolerancia de 10^{-16} y los resultados del error se demuestran en la siguiente gráfica

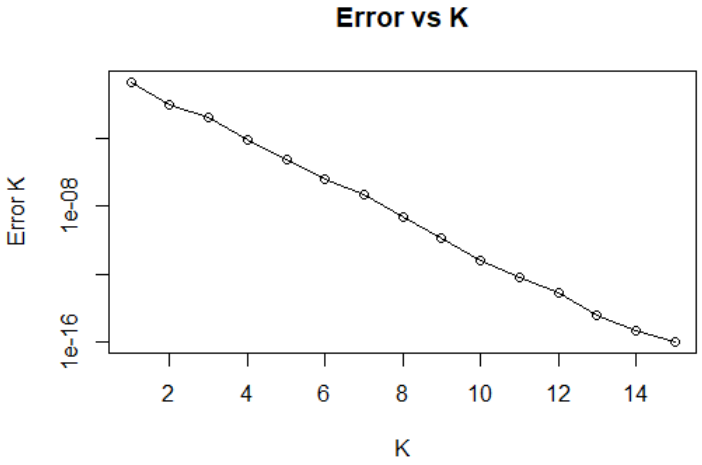

Figura 6: Comportamiento del error a lo largo de las iteraciones

4.2 Utilice el algoritmo de Steffensen para resolver $x^2 - cosx$ y compararlo con el metodo de Aitken con Tolerancia de 10^{-8}, 10^{-16}, realice una gráfica que muestre la comparación entre los métodos

Al aplicar el método de Steffensen a la funcion $f(x) = x^2 - cosx$ se obtuvieron los siguientes resultados:

Tolerancia	raíz	# Iteraciones
10^{-8}	0.8241323	4
10^{-16}	0.8241323	5

Basados en la cantidad de iteraciones se puede decir que el método de Steffensen es bastante bueno, pues llega al resultado sin necesidad de muchas iteraciones a diferencia de otros métodos. El comportamiento del error a medida de las iteraciones se evidencia con mayor claridad en las siguientes dos gráficas:

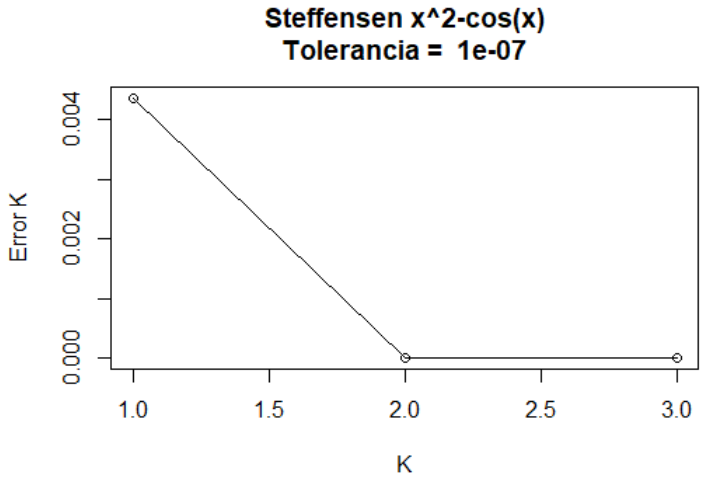

Figura 7: Error en Steffensen con tolerancia de 10^{-8}

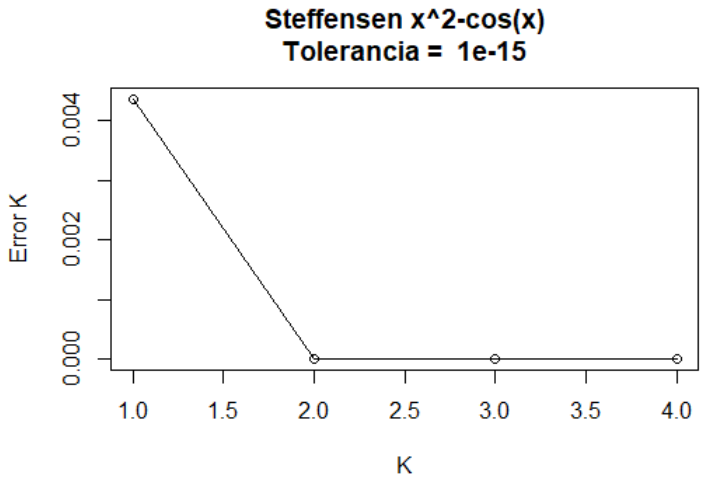

Figura 8: Error en Steffensen con tolerancia de 10^{-16}