

Análisis Numérico

Nicolás Camacho Plazas Daniela Cortes Antonio Mateo Florido Sanchez

Introducción

En el presente documento se encontrará la descripción del desarrollo de los algoritmos propuestos en clase. Además de este documento, contamos con los archivos .R (código fuente) que se encuentra en cada repositorio de GitHub de los integrantes del grupo, los cuales son: [NicolasCamachoP](#) , [dcortesantonio](#) y [mateoflorido](#).

Algoritmos para encontrar las raíces de una función:

Cuadrática

Al calcular la raíz de una función cuadrática utilizamos la función cuadrática clásica:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ (fórmula1)}$$

Pero además al racionalizar por el conjugado esta misma fórmula obtenemos:

$$x = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ (fórmula2)}$$

De esta forma si b tiene valores muy grandes la función no va tender a 0. Por tanto teniendo los valores de la fórmula como: a=3, b=9¹²), c=-3 , obtenemos los siguientes resultados:

```
## Con 8 decimales.  
## fórmula 1: -94143178827 , fórmula 2: 1.0622118e-11  
## Con 16 decimales.  
## fórmula 1: -94143178827 , fórmula 2: 1.062211848441645e-11
```

Bisección

Bisección dos parámetros

Para este algoritmo contábamos con cuatro parámetros, la función de la cual queríamos obtener su raíz, la tolerancia y un intervalo [a,b] donde la función f(x) es continua. Además de esto era necesario verificar que existiera por lo menos una raíz, esto dado que la función f(x) evaluada en los valores de a y b tuvieran signo contrario.

PASO 1: Verificar que existe una raíz en el intervalo. PASO 2: Empezar ciclo. PASO 3: Encontrar punto medio m en el intervalo. PASO 4: Si el signo de f(a) es diferente al signo f(m), entonces b = m, si no a = m. PASO 5: Si |f(m)| es igual a 0, terminar el ciclo. PASO 6: Calcular error = (b-a)/2. PASO 7: Si el error es menor al valor de la tolerancia,

terminar el ciclo. PASO 8: Si el número de iteraciones es mayor a 100, terminar el ciclo.

Dada la función $x^3 - x - 1$ con un intervalo de $[1,2]$ y una tolerancia de 10^{-8} obtenemos los siguientes resultados:

##	a	b	m	Error est.
##	1.0000000	1.5000000	1.5000000	0.2500000
##	1.2500000	1.5000000	1.2500000	0.1250000
##	1.2500000	1.3750000	1.3750000	0.0625000
##	1.3125000	1.3750000	1.3125000	0.0312500
##	1.3125000	1.3437500	1.3437500	0.0156250
##	1.3125000	1.3281250	1.3281250	0.0078125
##	1.3203125	1.3281250	1.3203125	0.0039062
##	1.3242188	1.3281250	1.3242188	0.0019531
##	1.3242188	1.3261719	1.3261719	0.0009766
##	1.3242188	1.3251953	1.3251953	0.0004883
##	1.3247070	1.3251953	1.3247070	0.0002441
##	1.3247070	1.3249512	1.3249512	0.0001221
##	1.3247070	1.3248291	1.3248291	0.0000610
##	1.3247070	1.3247681	1.3247681	0.0000305
##	1.3247070	1.3247375	1.3247375	0.0000153
##	1.3247070	1.3247223	1.3247223	0.0000076
##	1.3247147	1.3247223	1.3247147	0.0000038
##	1.3247147	1.3247185	1.3247185	0.0000019
##	1.3247166	1.3247185	1.3247166	0.0000010
##	1.3247175	1.3247185	1.3247175	0.0000005
##	1.3247175	1.3247180	1.3247180	0.0000002
##	1.3247178	1.3247180	1.3247178	0.0000001
##	1.3247179	1.3247180	1.3247179	0.0000001
##	1.3247179	1.3247180	1.3247179	0.0000000
##	1.3247179	1.3247180	1.3247180	0.0000000
##	1.3247180	1.3247180	1.3247180	0.0000000
##	1.3247180	1.3247180	1.3247180	0.0000000
##	1.3247180	1.3247180	1.3247180	0.0000000
##	1.3247180	1.3247180	1.3247180	0.0000000
##	1.3247180	1.3247180	1.3247180	0.0000000
##	Cero de la función en [1 , 2] aproximadamente es: 1.3247179556638			
##	con error <= 9.313225746154785e-10 Iteraciones: 29			
##	Predicción: 29.89735285398626			

Figura 1. Gráfica errores e iteraciones

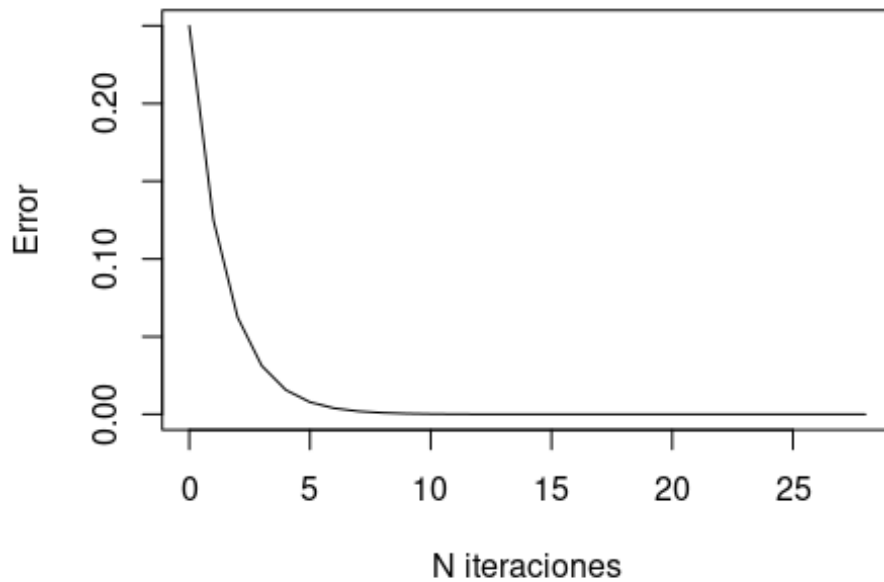
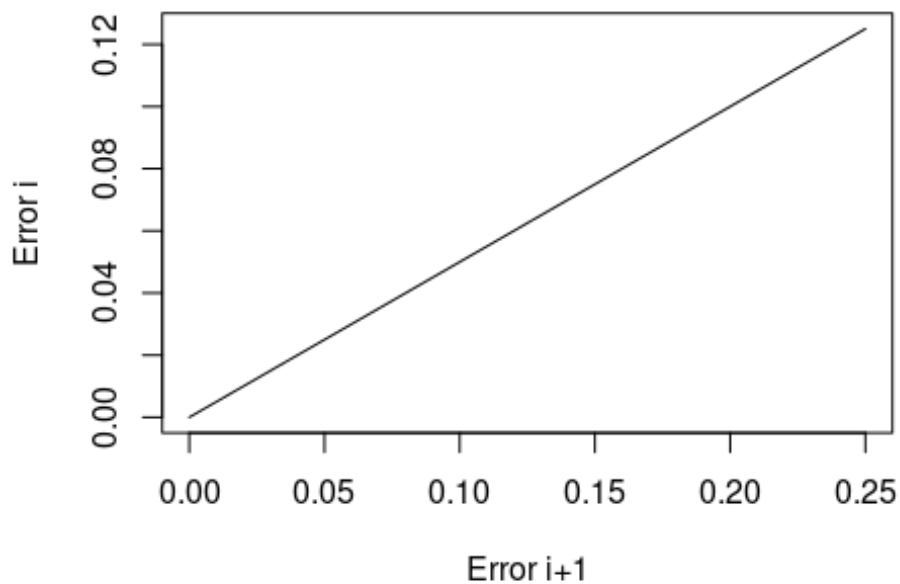


Figura 2. Gráfica errores



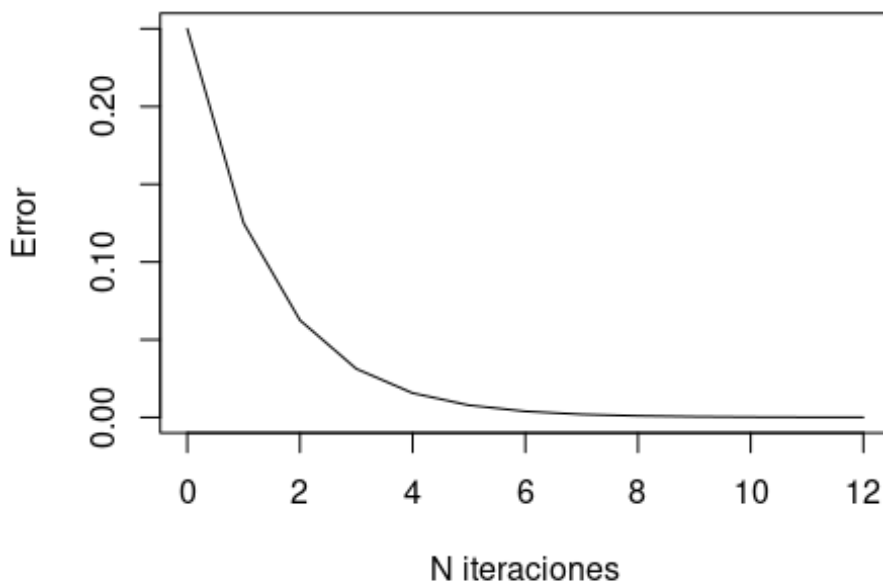
Además al graficar la cantidad de iteraciones en x y el valor de los errores en y podemos observar que a medida que aumenta la cantidad de iteraciones la curva tiende a cero sin embargo nunca toca este valor dado que el valor de tolerancia de este error es de 10^{-8} esta gráfica se encuentra en la Figura 1 , así mismo y al cambiar el

valor de la tolerancia a 10^{-3} observamos que el número de interacciones se reduce a 13 y además la curva que representa la relación entre el error y las iteraciones tiende a un valor que es mucho menos cercano a cero, dado que es una precisión mucho menor a la anterior, esta gráfica la podemos encontrar en la Figura 3.

##	a	b	m	Error est.
##	1.0000000	1.5000000	1.5000000	0.2500000
##	1.2500000	1.5000000	1.2500000	0.1250000
##	1.2500000	1.3750000	1.3750000	0.0625000
##	1.3125000	1.3750000	1.3125000	0.0312500
##	1.3125000	1.3437500	1.3437500	0.0156250
##	1.3125000	1.3281250	1.3281250	0.0078125
##	1.3203125	1.3281250	1.3203125	0.0039062
##	1.3242188	1.3281250	1.3242188	0.0019531
##	1.3242188	1.3261719	1.3261719	0.0009766
##	1.3242188	1.3251953	1.3251953	0.0004883
##	1.3247070	1.3251953	1.3247070	0.0002441
##	1.3247070	1.3249512	1.3249512	0.0001221
##	1.3247070	1.3248291	1.3248291	0.0000610

Cero de función en [1 , 2] aproximadamente es: 1.3248291015625 con error <= 6.103515625e-05 Iteraciones: 13
Predicción: 13.28771237954945

Figura 3. Gráfica errores e iteraciones



Bisección tres parámetros

Para este algoritmo contamos con los mismos parámetros de la bisección de dos parámetros, sin embargo a la hora de ejecutar el algoritmo calculamos dos pivotes

para poder dividir el intervalo en tres secciones y así verificando entre estos nuevos intervalos sus signos para comprobar si existe alguna raíz en estos intervalos, de esta forma en el mejor de los casos estamos descartando dos terceras partes del intervalo total haciendo este algoritmo más eficiente que el anterior, ya que para una misma función realiza una cantidad menor de iteraciones.

PASO 1: Verificar que existe una raíz en el intervalo. PASO 2: Empezar ciclo. PASO 3: Encontrar dos puntos medios m1 y m2 en el intervalo. PASO 4: Si el signo de f(a) es diferente a signo de f(m1), entonces b = m1. PASO 5: Si no, si el signo de f(m1) es diferente a signo de f(m2), entonces a = m1 y b = m2. PASO 6: Si no, a = m2 PASO 7: Si |f(m1)| es igual a 0 o si |f(m2)| es igual a 0, terminar el ciclo. PASO 8: Calcular error = (b-a)/2. PASO 9: Si el error es menor al valor de la tolerancia, terminar el ciclo. PASO 10: Si el número de iteraciones es mayor a 100, terminar el ciclo.

Dada la función $x^3 - x - 1$ con un intervalo de [1,2] y una tolerancia de 10^{-8} obtenemos los siguientes resultados:

##	a	b	m1	m2	Error
##	1.0000000	1.3333333	1.3333333	1.6666667	0.1666667
##	1.2222222	1.3333333	1.1111111	1.2222222	0.0555556
##	1.2962963	1.3333333	1.2592593	1.2962963	0.0185185
##	1.3209877	1.3333333	1.3086420	1.3209877	0.0061728
##	1.3209877	1.3251029	1.3251029	1.3292181	0.0020576
##	1.3237311	1.3251029	1.3223594	1.3237311	0.0006859
##	1.3246456	1.3251029	1.3241884	1.3246456	0.0002286
##	1.3246456	1.3247980	1.3247980	1.3249505	0.0000762
##	1.3246964	1.3247472	1.3246964	1.3247472	0.0000254
##	1.3247134	1.3247303	1.3247134	1.3247303	0.0000085
##	1.3247134	1.3247190	1.3247190	1.3247247	0.0000028
##	1.3247171	1.3247190	1.3247153	1.3247171	0.0000009
##	1.3247178	1.3247184	1.3247178	1.3247184	0.0000003
##	1.3247178	1.3247180	1.3247180	1.3247182	0.0000001
##	1.3247179	1.3247180	1.3247178	1.3247179	0.0000000
##	1.3247180	1.3247180	1.3247179	1.3247180	

```
0.0000000
## 1.3247180      1.3247180      1.3247180      1.3247180
0.0000000
## 1.3247180      1.3247180      1.3247180      1.3247180
0.0000000
## 1.3247180      1.3247180      1.3247180      1.3247180
0.0000000
## Cero de función en [ 1 , 2 ] aproximadamente es: 1.324717957375077
con error <= 4.301957678976009e-10 Iteraciones: 19
## Predicción bisección: 29.89735285398626
```

Figura 4. Gráfica errores e iteraciones

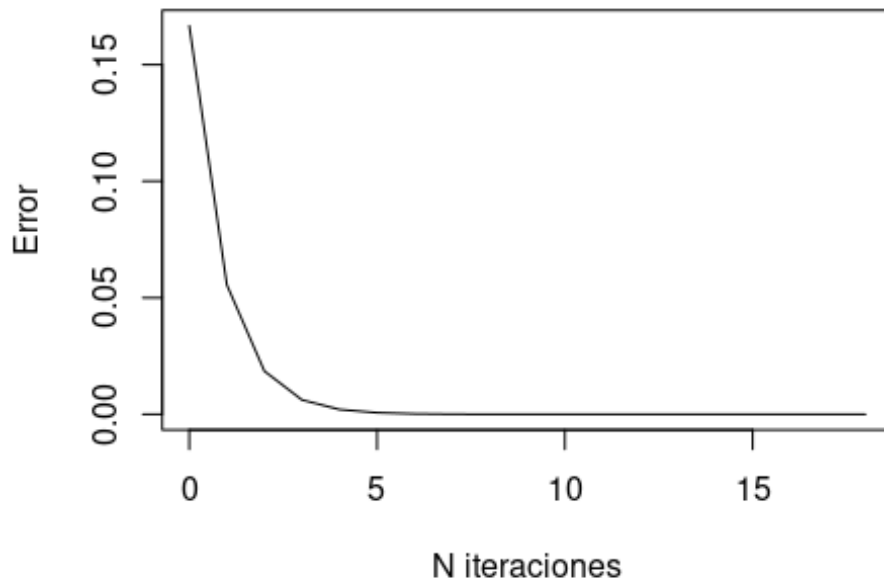
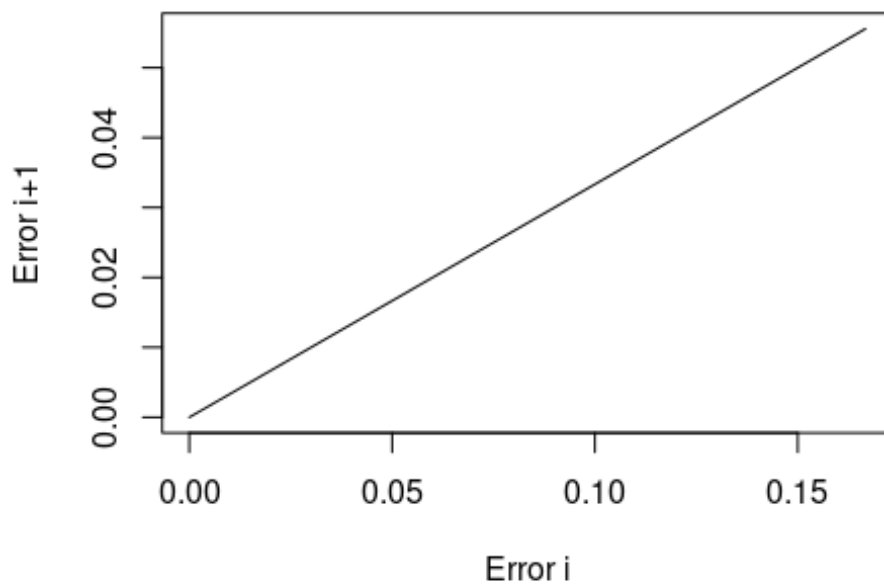


Figura 5. Gráfica errores



Newton

Para este algoritmo contábamos con cuatro parámetros, la función de la cual queríamos obtener su raíz, la tolerancia, el valor inicial de x_k es decir x_0 (el cual es un

valor cercano a la raíz) y el valor máximo de iteraciones que se podrán realizar. Ya que en para este algoritmo no contamos con un intervalo donde podamos asegurar la existencia de por lo menos una raíz, tenemos un valor máximo de iteraciones el cual nos da una condición de salida para no quedarnos en un ciclo infinito.

PASO 1: Empezar ciclo. PASO 2: Calcular nuevo valor de x1.

$$x1 = x0 - \frac{f(x0)}{df(x0)}$$

PASO 3: $x0 = x1$ PASO 4: Calcular diferencia $dx = |x1 - x0|$ PASO 5: Si dx es menor o igual al valor de la tolerancia, terminar el ciclo. PASO 6: Si el número de iteraciones es mayor al valor máximo de iteraciones, terminar el ciclo.

Dada la función $e^x - \pi x$ con un valor inicial de x igual a 1.6, un máximo de iteraciones igual a 1000 y una tolerancia de 10^{-8} , obtenemos los siguientes resultados:

##	x0	x1	dx	Error est.
##	1.6405842	1.6405842	0.0405842	-0.0405842
##	1.6385338	1.6385338	0.0020504	0.0020504
##	1.6385284	1.6385284	0.0000054	0.0000054
##	1.6385284	1.6385284	0.0000000	0.0000000
##	Iteraciones= 4 X: 1.638528419970363 f(x): -8.881784197001252e-16			
	Error Estimado: 3.743240808769806e-11			

Figura 6. Gráfica errores e iteraciones

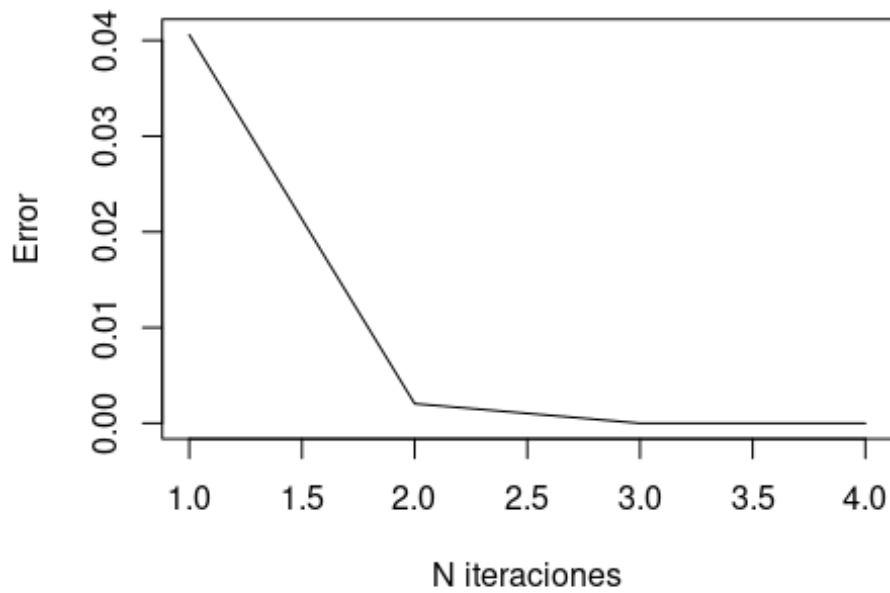
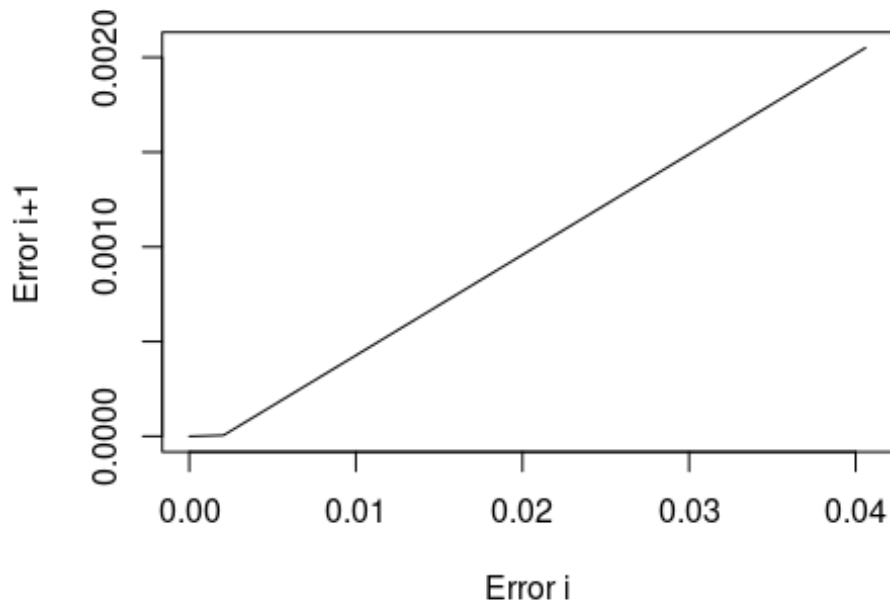


Figura 7. Gráfica errores



Esta función tiene dos raíces, aunque solo muestra la más cercana al punto inicial ingresado como parámetro, es decir que para esta misma función, con el mismo valor de tolerancia, el mismo valor de iteraciones máximas, pero esta vez el valor inicial de x_0 haciendolo igual a 0.2 nos da como resultado lo siguiente:

##	x0	x1	dx	Error est.
##	0.5088675	0.5088675	0.3088675	-0.3088675
##	0.5526725	0.5526725	0.0438050	-0.0438050
##	0.5538262	0.5538262	0.0011537	-0.0011537
##	0.5538270	0.5538270	0.0000008	-0.0000008
##	0.5538270	0.5538270	0.0000000	-0.0000000

Iteraciones= 5 X: 0.5538270366445135 f(x): 2.220446049250313e-16
Error Estimado: -4.22800679558094e-13

Figura 8. Gráfica errores e iteraciones

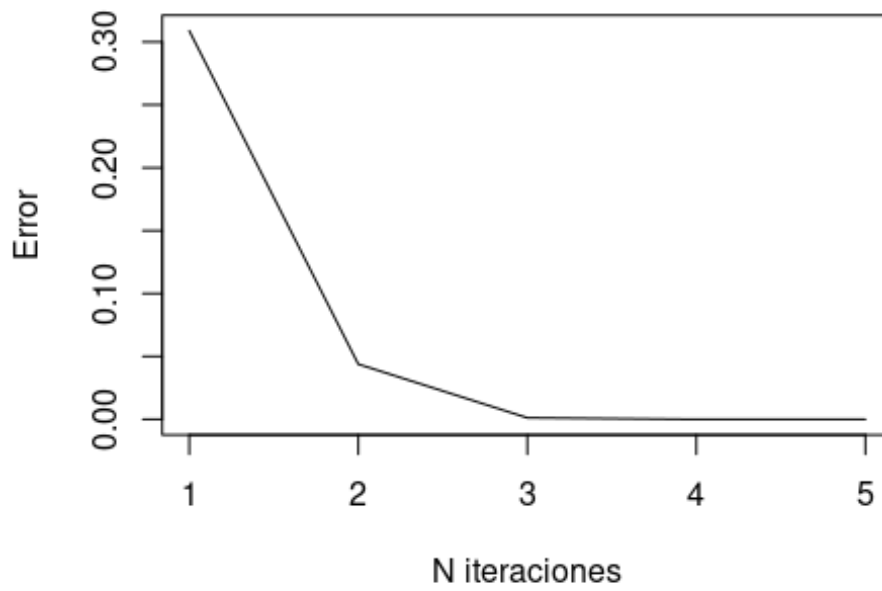
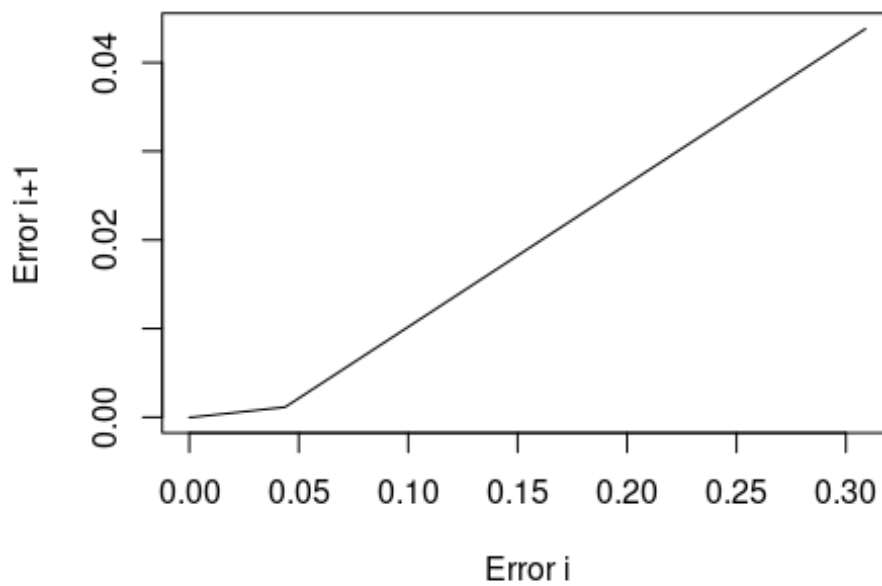


Figura 9. Gráfica errores



Hibrido (Bisección y Newton)

Como primer punto analizamos las ventajas que trae cada uno de los algoritmos y pudimos observar que el método de Newton tiene una convergencia local mientras que el algoritmo de bisección una convergencia global en el intervalo dado.

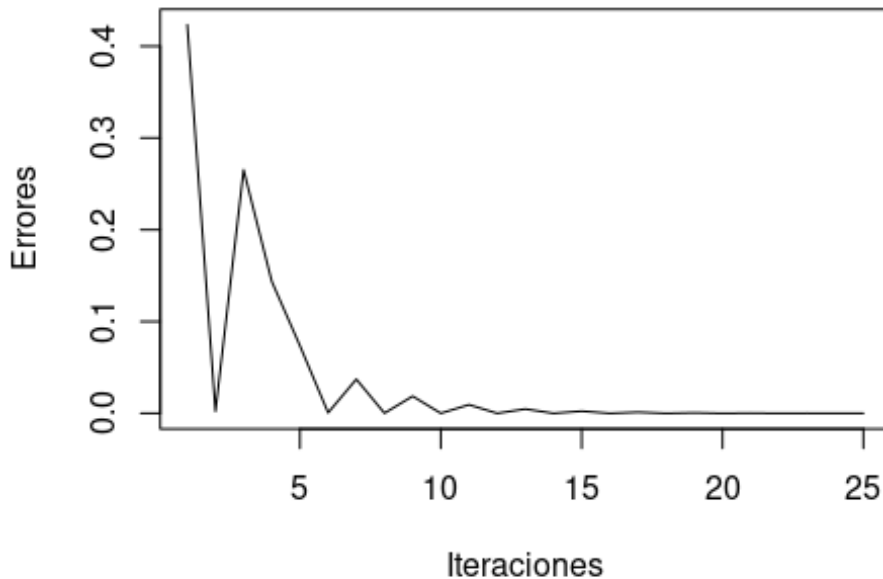
Por tanto por medio de la bisección con dos intervalos podemos ir acotando el intervalo a analizar ,además dado los signos extremos del intervalo conocer si existe o no una raíz allí y así mismo ir posicionando el x_k del método de Newton para poder aplicar este método de forma simultánea.

Dada la función $e^x - \pi x$ con un intervalo $[0,2]$, un máximo de iteraciones igual a 1000 y una tolerancia de 10^{-8} , obtenemos los siguientes resultados:

```
## xi: 0.552198029112459 Corrección -0.05219802911245899 Iteración: 1
## xi: 0.7760990145562294 Corrección 0.2738016311126862 Iteración: 2
## xi: 0.6641485218343441 Corrección 0.1198312652575797 Iteración: 3
## xi: 0.6081732754734015 Corrección 0.05638869439561096 Iteración: 4
## xi: 0.5533731278898788 Corrección 0.02681252440305147 Iteración: 5
## xi: 0.5807732016816401 Corrección 0.02742108930608239 Iteración: 6
## xi: 0.5537153234461075 Corrección 0.01335784133965191 Iteración: 7
## xi: 0.5672442625638738 Corrección 0.01353188189576753 Iteración: 8
## xi: 0.5537992149114261 Corrección 0.006680578093564592 Iteración: 9
## xi: 0.56052173873765 Corrección 0.006722878042628352 Iteración: 10
## xi: 0.5538200960728709 Corrección 0.003340380751667182 Iteración:
11
## xi: 0.5571709174052604 Corrección 0.003350865017061631 Iteración:
12
## xi: 0.5538253033913783 Corrección 0.001670203347687424 Iteración:
13
## xi: 0.5554981103983193 Corrección 0.001672812429327686 Iteración:
14
## xi: 0.5538266035698027 Corrección 0.000835103325045971 Iteración:
15
## xi: 0.554662356984061 Corrección 0.0008357540896667627 Iteración:
16
## xi: 0.5538269284058066 Corrección 0.0004175518711252905 Iteración:
17
## xi: 0.5542446426949338 Corrección 0.0004177143734038098 Iteración:
18
## xi: 0.5538270095885925 Corrección 0.0002087759617775462 Iteración:
19
## xi: 0.5540358261417632 Corrección 0.0002088165636958321 Iteración:
20
## xi: 0.5538270298810035 Corrección 0.0001043879841742865 Iteración:
21
## xi: 0.5539314280113834 Corrección 0.0001043981316950806 Iteración:
22
## xi: 0.5538270349536948 Corrección 5.219399249864719e-05 Iteración:
23
## xi: 0.5538792314825391 Corrección 5.219652900852327e-05 Iteración:
24
```

```
## xi: 0.5538270362218163 Corrección 2.609699630065741e-05 Iteración: 25
## Raíz: 0.5538270362218163 Iteraciones Necesarias: 25
```

Figura 10. Gráfica errores e iteraciones



Punto fijo

Para este algoritmo contábamos con cuatro parámetros, la función de la cual queríamos obtener su punto para el cual $f(x) = x$, la tolerancia, el valor inicial de x_k es decir x_0 y el valor máximo de iteraciones que se podrán realizar.

Cuando queremos hallar la raíz de una función por medio del método de punto fijo, requerimos volver a escribir la ecuación, entonces: $f(x) = 0$ a la forma $x = g(x)$, encontrando así un punto en la función $g(x)$ el cual $x = g(x)$ y este punto x será entonces una raíz de la función $f(x)$.

PASO 1: Empezar ciclo. PASO 2: Calcular nuevo valor de x_1 .

$$x_1 = f(x_0)$$

PASO 3: $x_0 = x_1$ PASO 4: Calcular diferencia $dx = |x_1 - x_0|$ PASO 5: Si dx es menor o igual al valor de la tolerancia, terminar el ciclo y el valor de la raíz es x_1 . PASO 6: Si el número de iteraciones es mayor al valor máximo de iteraciones, terminar el ciclo y no hubo convergencia, es decir no se encontró la raíz.

Si queremos hallar la raíz de la función $e^x - \pi i * x$, por medio de este método primero despejamos una x de la función de esta forma:

$$f(x) = e^x - pi * x$$

$$e^x = pi * x g(x) = \frac{e^x}{pi} = x$$

Así la función de la cual hallaremos el punto en el que $g(x)=x$ será esta última, además con un valor inicial de $x_0 = 1$, una tolerancia igual a 10^{-8} y un valor máximo de iteraciones igual a 100, dandonos como resultado la raíz aproximada de la función inicial $e^x - pi * x$, como lo vemos a continuación:

```
## x 1 = 0.8652559794322651
## x 2 = 0.7561814590413005
## x 3 = 0.6780403857531705
## x 4 = 0.6270747668420679
## x 5 = 0.5959162636569479
## x 6 = 0.577634697018414
## x 7 = 0.5671705715535849
## x 8 = 0.5612665715043985
## x 9 = 0.5579626165110687
## x 10 = 0.5561221751800245
## x 11 = 0.5550996062214397
## x 12 = 0.5545322687154927
## x 13 = 0.55421775098848
## x 14 = 0.5540434670902935
## x 15 = 0.5539469146490965
## x 16 = 0.5538934323041607
## x 17 = 0.55386380957671
## x 18 = 0.5538474028630411
## x 19 = 0.5538383161218281
## x 20 = 0.5538332835592404
## x 21 = 0.5538304963655912
## x 22 = 0.5538289527349002
## x 23 = 0.553828097828191
## x 24 = 0.5538276243570368
## x 25 = 0.5538273621356944
## x 26 = 0.5538272169103591
## x 27 = 0.5538271364806215
## x 28 = 0.5538270919364522
## x 29 = 0.553827067266685
## x 30 = 0.5538270536039003
## x 31 = 0.5538270460370806
## x* es aproximadamente 0.5538270460370806 con un error menor que 1e-
08 en 32 iteraciones.
```

Figura 11. Gráfica errores e iteraciones

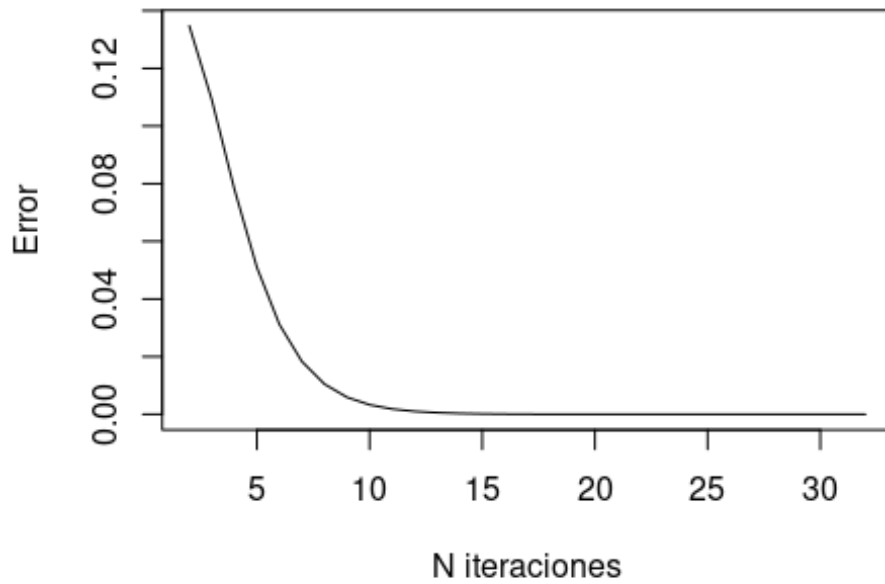
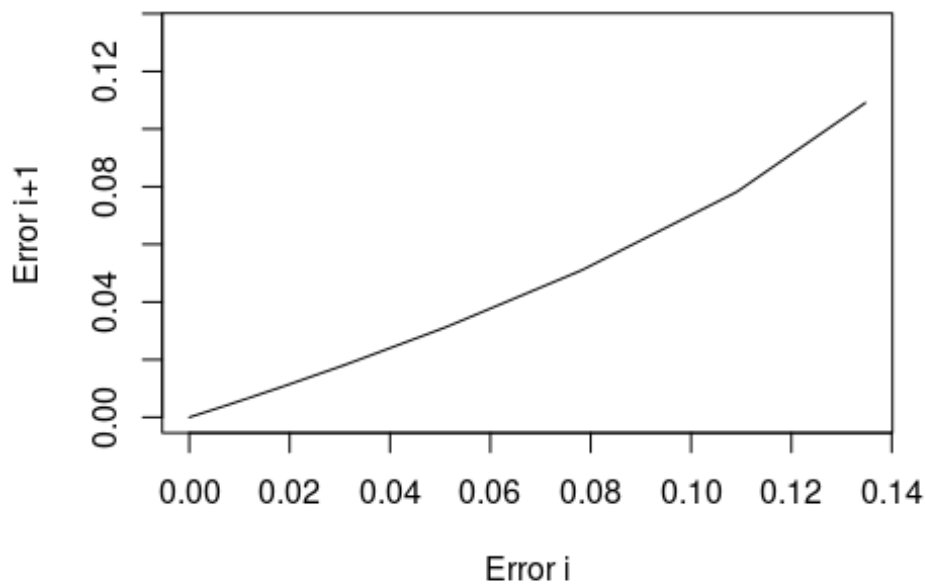


Figura 12. Gráfica errores



Steffensen

Este algoritmo tiene convergencia cuadrática, además es un híbrido entre el método de punto fijo y el método de Aitken, ya que este segundo acelera la convergencia para encontrar de una forma más rápida la raíz de una función, así mismo este método no

necesita la derivada de la función inicial lo que la hace más rápido que el método de Newton.

Para este algoritmo contabamos con cinco parámetros, la función de la cual queríamos obtener su raíz, un valor inicial x_0 , el cual es valor cercano a la raíz, la tolerancia y el valor máximo de iteraciones.

PASO 1: Empezar ciclo. PASO 2: Calcular nuevo valor de x_1 .

$$x_1 = \frac{f(x_0)^2}{f(x_0 + f(x_0)) - f(x_0)}$$

PASO 3: $x_0 = x_1$ PASO 4: Calcular diferencia error = $|x_1 - x_0|$ PASO 5: Si el error es menor o igual al valor de la tolerancia, terminar el ciclo y el valor de la raíz es x_0 .

PASO 6: Si el número de iteraciones es mayor al valor máximo de iteraciones, terminar el ciclo.

Dada la función $e^x - \pi x$ con un valor inicial de x igual a 0, un máximo de iteraciones igual a 1000 y una tolerancia de 10^{-8} , obtenemos los siguientes resultados:

## i	x_i	$f(x)$	Error est.
## 1.00000000	0.70258722	-0.18827338	0.70258722
## 2.00000000	0.55790573	-0.00570259	0.14468149
## 3.00000000	0.55383116	-0.00000578	0.00407457
## 4.00000000	0.55382704	-0.00000000	0.00000412
## 5.00000000	0.55382704	0.00000000	0.00000000

Cero de la función: 0.5538270366445135 con error <= 4.232725281383409e-12 Iteraciones: 5

Figura 13. Gráfica errores e iteraciones

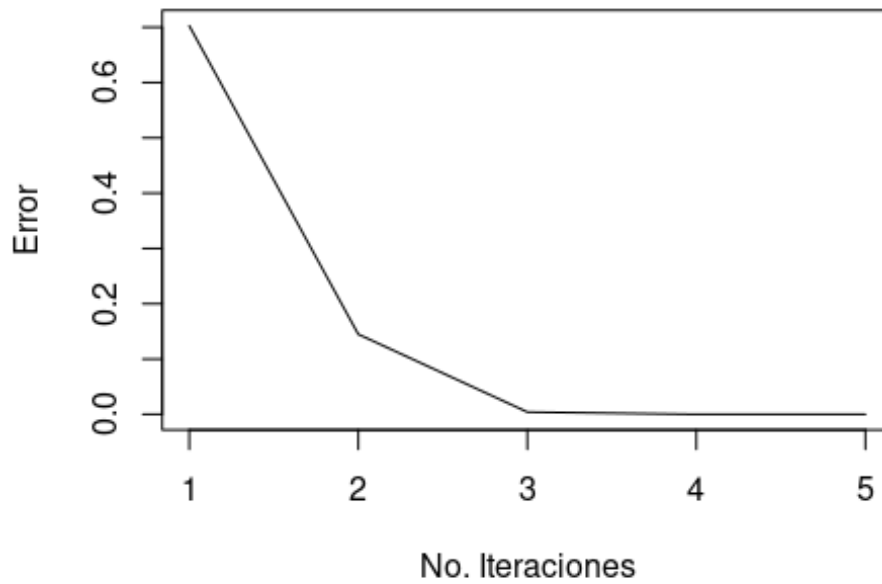
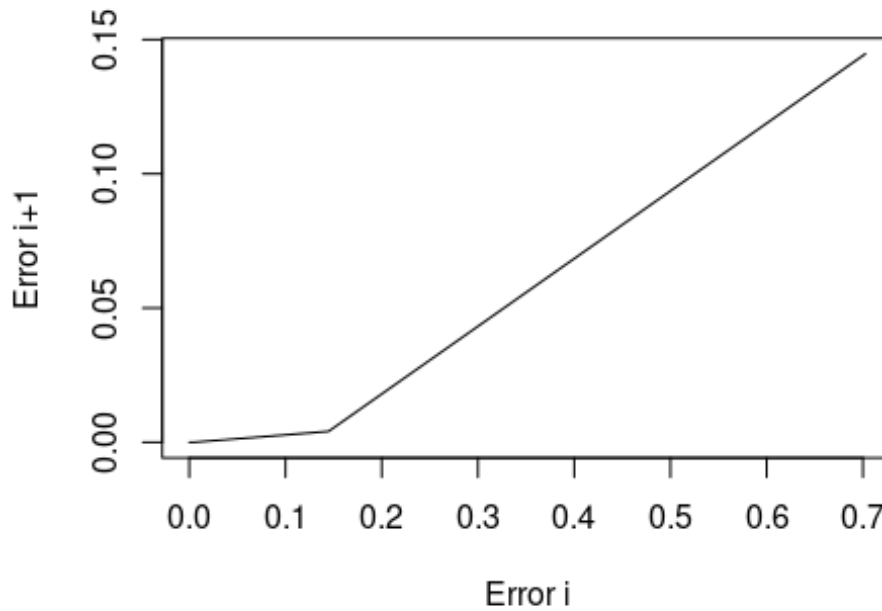


Figura 14. Gráfica errores



Secante

Este algoritmo cuenta con una convergencia lineal y a diferencia del método de Newton, por medio de la pendiente de la recta que existe entre dos puntos que son

ingresados como parámetros se intenta hallar la raíz de una función acotando la distancia entre las imágenes de estos puntos en la función a cero.

Para este algoritmo contábamos con cinco parámetros, la función de la cual queríamos obtener su raíz, dos valores x_0 y x_1 (un intervalo donde se encuentra la raíz de la función), la tolerancia y el valor máximo de iteraciones.

PASO 1: Calcular las imágenes en el intervalo $y_1=f(x_1)$ y $y_0 = f(x_0)$. PASO 2: Mientras que $|x_1-x_0| > \text{tol}$ y no se hayan sobrepasado el número de iteraciones continuar al paso 3. PASO 3: Calcular el valor de la pendiente de la recta que se forma entre los dos puntos x_0 y x_1 .

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

PASO 4: Actualizar los valores:

$$\begin{aligned} x_0 &= x_1 \\ x_1 &= \frac{x_1 - y_1}{m} \\ y_0 &= y_1 \\ y_1 &= f(x_1) \end{aligned}$$

PASO 5: Calcular diferencia error = $|x_1-x_0|$

Así la raíz aproximada está en x_0 y en x_1 que dependiendo del valor de la tolerancia del error tienen una menor diferencia entre ellos a medida de la cantidad de iteraciones realizadas.

Dada la función $e^x - \pi x$ con un valor inicial de x_0 igual a 0 y un valor de x_1 igual a 2, un máximo de iteraciones igual a 1000 y una tolerancia de 10^{-8} , obtenemos los siguientes resultados:

## i	x_1	x_2	Error est.
## 2.00000000	2.00000000	1.79917733	0.20082267
## 3.00000000	1.79917733	1.70026485	0.09891248
## 4.00000000	1.70026485	1.61106917	0.08919568
## 5.00000000	1.61106917	1.54803478	0.06303439
## 6.00000000	1.54803478	1.49921221	0.04882258
## 7.00000000	1.49921221	1.46271812	0.03649409
## 8.00000000	1.46271812	1.43506480	0.02765332
## 9.00000000	1.43506480	1.41422037	0.02084443
## 10.00000000	1.41422037	1.39847653	0.01574384
## 11.00000000	1.39847653	1.38659443	0.01188210
## 12.00000000	1.38659443	1.37762415	0.00897028
## 13.00000000	1.37762415	1.37085290	0.00677125
## 14.00000000	1.37085290	1.36574138	0.00511153
## 15.00000000	1.36574138	1.36188282	0.00385856
## 16.00000000	1.36188282	1.35897007	0.00291275

##	17.00000000	1.35897007	1.35677131	0.00219876
##	18.00000000	1.35677131	1.35511151	0.00165980
##	19.00000000	1.35511151	1.35385856	0.00125294
##	20.00000000	1.35385856	1.35291274	0.00094582
##	21.00000000	1.35291274	1.35219877	0.00071398
##	22.00000000	1.35219877	1.35165980	0.00053897
##	23.00000000	1.35165980	1.35125295	0.00040685
##	24.00000000	1.35125295	1.35094582	0.00030712
##	25.00000000	1.35094582	1.35071398	0.00023184
##	26.00000000	1.35071398	1.35053897	0.00017501
##	27.00000000	1.35053897	1.35040685	0.00013211
##	28.00000000	1.35040685	1.35030712	0.00009973
##	29.00000000	1.35030712	1.35023184	0.00007528
##	30.00000000	1.35023184	1.35017501	0.00005683
##	31.00000000	1.35017501	1.35013211	0.00004290
##	32.00000000	1.35013211	1.35009973	0.00003238
##	33.00000000	1.35009973	1.35007528	0.00002445
##	34.00000000	1.35007528	1.35005683	0.00001845
##	35.00000000	1.35005683	1.35004290	0.00001393
##	36.00000000	1.35004290	1.35003238	0.00001052
##	37.00000000	1.35003238	1.35002445	0.00000794
##	38.00000000	1.35002445	1.35001845	0.00000599
##	39.00000000	1.35001845	1.35001393	0.00000452
##	40.00000000	1.35001393	1.35001052	0.00000341
##	41.00000000	1.35001052	1.35000794	0.00000258
##	42.00000000	1.35000794	1.35000599	0.00000195
##	43.00000000	1.35000599	1.35000452	0.00000147
##	44.00000000	1.35000452	1.35000341	0.00000111
##	45.00000000	1.35000341	1.35000258	0.00000084
##	46.00000000	1.35000258	1.35000195	0.00000063
##	47.00000000	1.35000195	1.35000147	0.00000048
##	48.00000000	1.35000147	1.35000111	0.00000036
##	49.00000000	1.35000111	1.35000084	0.00000027
##	50.00000000	1.35000084	1.35000063	0.00000021
##	51.00000000	1.35000063	1.35000048	0.00000015
##	52.00000000	1.35000048	1.35000036	0.00000012
##	53.00000000	1.35000036	1.35000027	0.00000009
##	54.00000000	1.35000027	1.35000021	0.00000007
##	55.00000000	1.35000021	1.35000015	0.00000005
##	56.00000000	1.35000015	1.35000012	0.00000004
##	57.00000000	1.35000012	1.35000009	0.00000003
##	58.00000000	1.35000009	1.35000007	0.00000002
##	59.00000000	1.35000007	1.35000005	0.00000002
##	60.00000000	1.35000005	1.35000004	0.00000001
##	61.00000000	1.35000004	1.35000003	0.00000001

Número de iteraciones: 61 X0: 1.350000037963421 X1:
1.350000028657739 Error: 9.305682446836272e-09

Figura 15. Gráfica errores e iteraciones

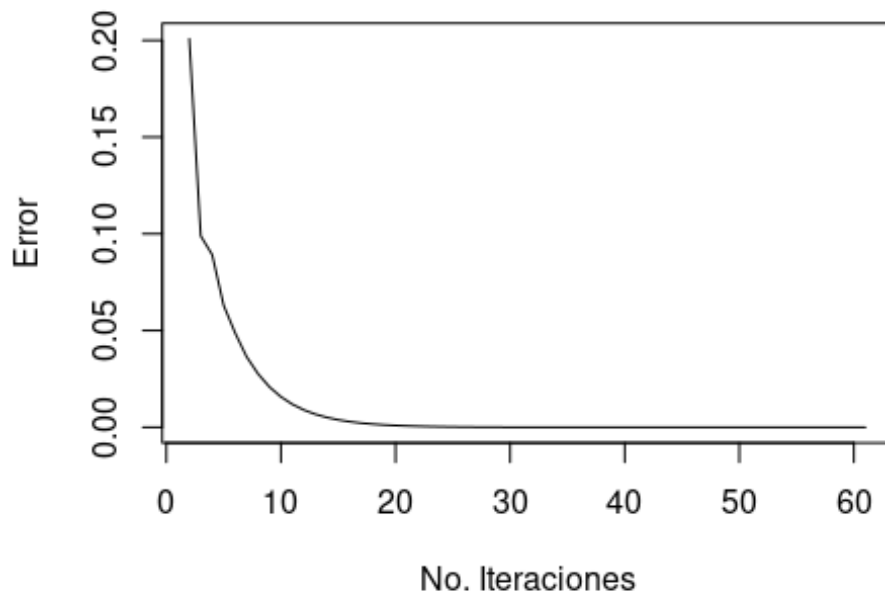
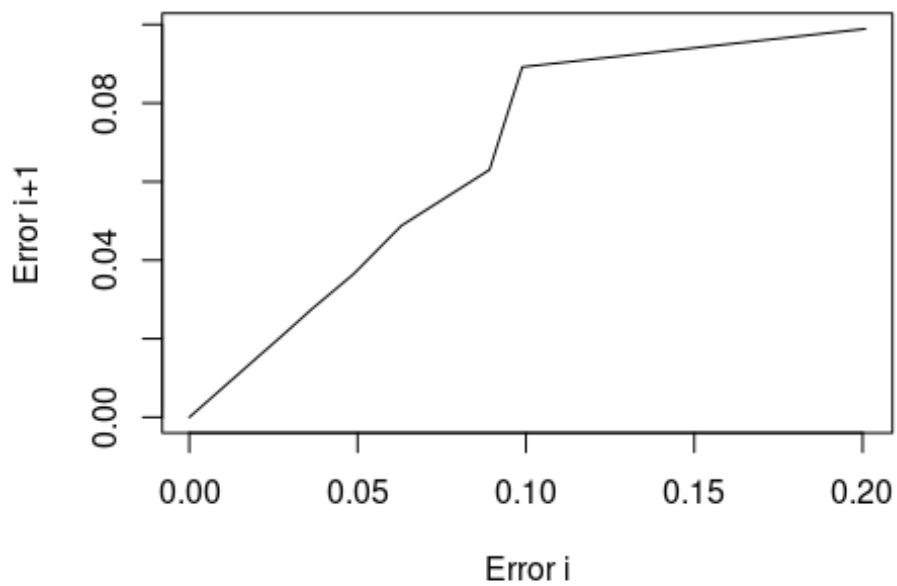


Figura 16. Gráfica errores



RegulaFalsi

Este algoritmo funciona a partir de un intervalo $[a,b]$ en donde sus imágenes tienen signos opuestos para verificar que existe una raíz en ese intervalo, así acotando este,

se encuentra la raíz de esta función a partir de calcular un punto que se encuentra en el intervalo el cual es la intersección de la recta que pasa por los puntos a y b en la función y verificando que la imagen de este punto sea cercano a cero.

Para este algoritmo contabamos con cinco parámetros, la función de la cual queríamos obtener su raíz, dos valores x_0 y x_1 (un intervalo donde se encuentra la raíz de la función), la tolerancia y el valor máximo de iteraciones.

PASO 1: Calcular las imágenes en el intervalo $y_1=f(x_1)$ y $y_0 = f(x_0)$. PASO 2: Mientras que no se hayan sobrepasado el número de iteraciones continuar al paso 3. PASO 3: Calcular el valor del punto medio

$$x_2 = \frac{x_0 * y_1 - x_1 * y_0}{y_1 - y_0}$$

PASO 4: Si $|f(x_2)| \leq \text{tol}$, terminar el ciclo y la raíz de la función es x_2 . PASO 5: Si el signo de $f(x_2)$ es diferente al signo de $f(x_0)$ entonces:

$$\begin{aligned} x_1 &= x_2 \\ y_1 &= f(x_2) \end{aligned}$$

PASO 6: Si no, entonces:

$$\begin{aligned} x_0 &= x_2 \\ y_0 &= f(x_2) \end{aligned}$$

PASO 7: Calcular diferencia error = $|x_1 - x_0|$

Dada la función $e^x - \pi x$ con un valor inicial de x_0 igual a 0 y un valor de x_1 igual a 1, un máximo de iteraciones igual a 1000 y una tolerancia de 10^{-8} , obtenemos los siguientes resultados:

## i	x_i	$f(x)$	Error est.
## 1.00000000	0.70258722	-0.18827338	1.00000000
## 2.00000000	0.59126733	-0.05124498	0.70258722
## 3.00000000	0.56244485	-0.01201474	0.59126733
## 4.00000000	0.55576745	-0.00271659	0.56244485
## 5.00000000	0.55426175	-0.00060917	0.55576745
## 6.00000000	0.55392432	-0.00013635	0.55426175
## 7.00000000	0.55384880	-0.00003051	0.55392432
## 8.00000000	0.55383191	-0.00000682	0.55384880
## 9.00000000	0.55382813	-0.00000153	0.55383191
## 10.00000000	0.55382728	-0.00000034	0.55382813
## 11.00000000	0.55382709	-0.00000008	0.55382728
## 12.00000000	0.55382705	-0.00000002	0.55382709

Cero de la función: 0.5538270393720905 con un error <= 9.465210504266963e-09 Iteraciones: 13

Figura 17. Gráfica errores e iteraciones

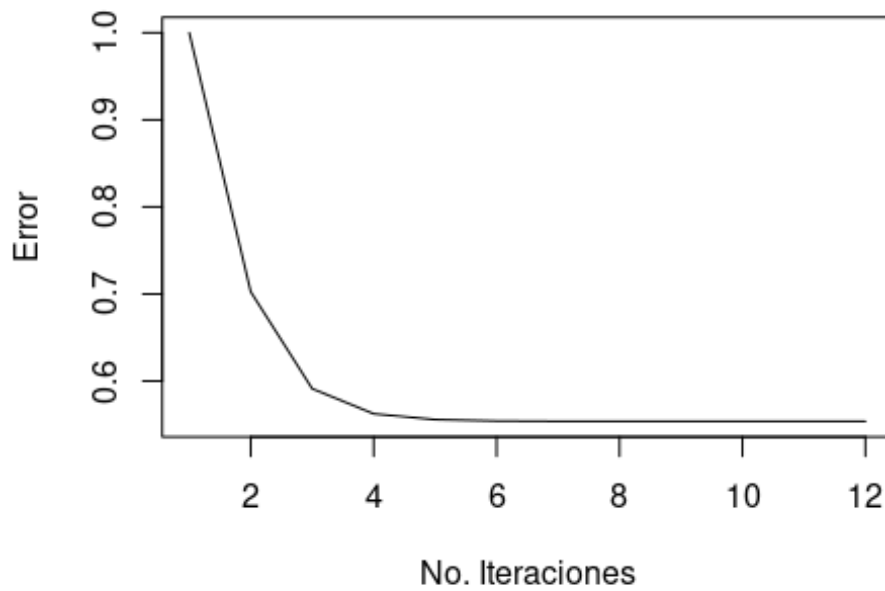
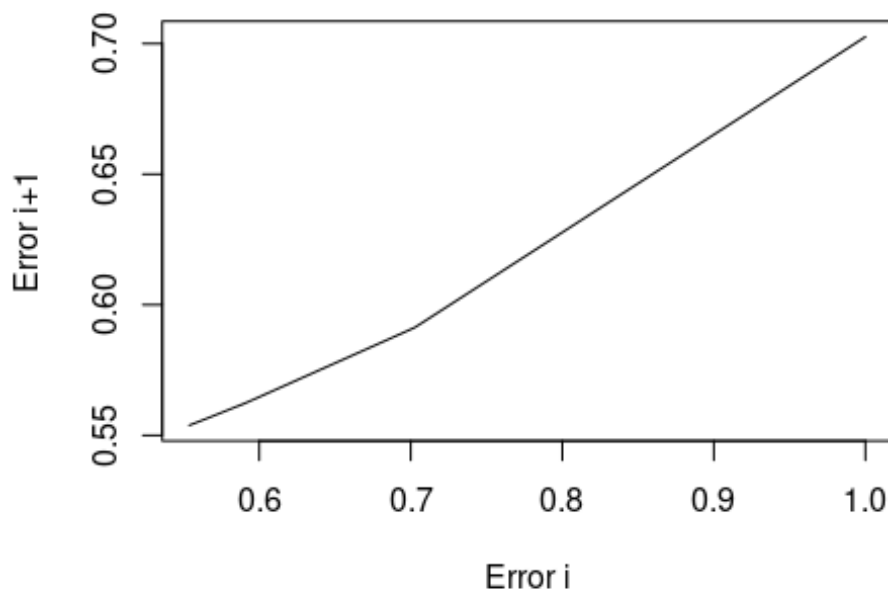


Figura 18. Gráfica errores



Illinois

Este algoritmo funciona a partir de un intervalo $[a,b]$ en donde se encuentra la raíz, calculando un punto medio en el intervalo y verificando los signos de las imágenes en a , b y en el caso que tenga un mismo signo este punto medio será la raíz de la función.

Para este algoritmo contabamos con cinco parámetros, la función de la cual queríamos obtener su raíz, dos valores x_0 y x_1 (un intervalo donde se encuentra la raíz de la función), un valor de tolerancia y un valor máximo de iteraciones.

PASO 1: Calcular las imágenes en el intervalo $y_1=f(x_1)$ y $y_0 = f(x_0)$. PASO 2: Mientras que no se hayan sobrepasado el número de iteraciones continuar al paso 3. PASO 3: Calcular el valor del punto medio

$$x_2 = \frac{x_1 * y_0 - x_0 * y_1}{y_0 - y_1}$$

PASO 4: Si $|f(x_1-x_0)| \leq \text{tol} * |x_1+x_0|$, terminar el ciclo. PASO 5: Si $f(x_2) * y_1 > 0$ entonces:

$$x_1 = x_2$$

$$y_1 = f(x_2)$$

$$y_0 *= 0.5$$

PASO 6: Si no, si $y_0 * f(x_2) > 0$ entonces:

$$x_1 = x_2$$

$$y_0 = y_2$$

$$y_1 *= 0.5$$

PASO 7: Si no, terminar el ciclo y el valor de la raíz de la función está en x_2 . PASO 8: Calcular diferencia error = $|x_1-x_0|$

Dada la función $e^x - \pi x$ con un valor inicial de x_0 igual a 0 y un valor de x_1 igual a 1, un máximo de iteraciones igual a 1000 y una tolerancia de 10^{-8} , obtenemos los siguientes resultados:

## i	x_i	$f(x)$	Error est.
## 1.00000000	0.70258722	-0.18827338	1.00000000
## 2.00000000	0.59126733	-0.05124498	0.70258722
## 3.00000000	0.53630178	0.02483068	0.59126733
## 4.00000000	0.55424223	-0.00058183	0.05496555
## 5.00000000	0.55383148	-0.00000623	0.01794046
## 6.00000000	0.55382269	0.00000609	0.01752970
## 7.00000000	0.55382704	-0.00000000	0.00000879
## 8.00000000	0.55382704	0.00000000	0.00000435
## Cero de funcion:	0.5538270366445137	con error	$\leq 4.34623589407046e-06$
Iteraciones:	8		

Figura 19. Gráfica errores e iteraciones

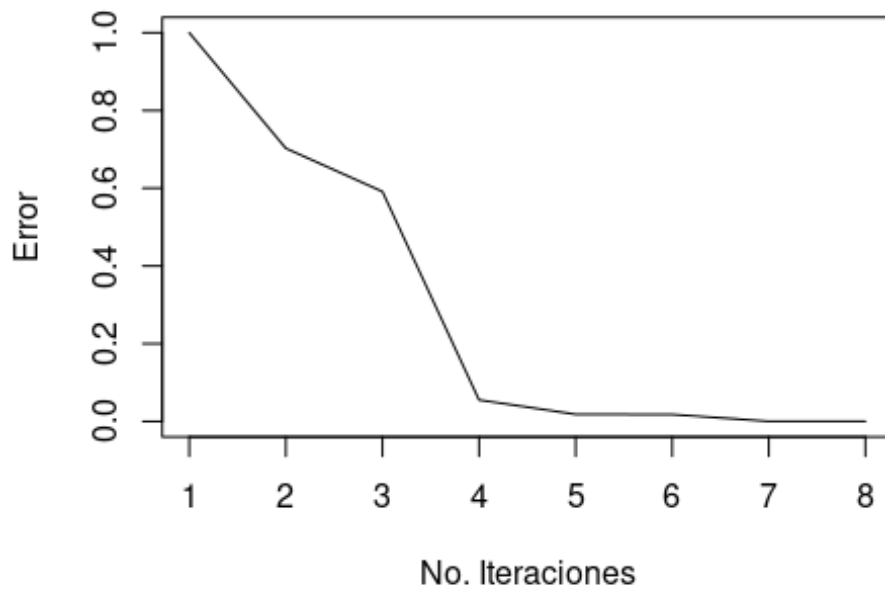
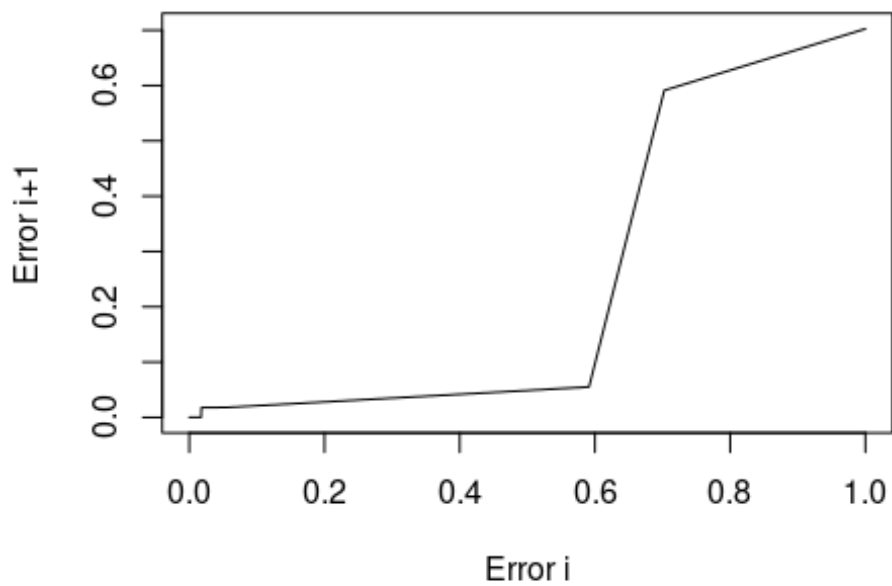


Figura 20. Gráfica errores



Solución taller:

Primer punto:

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.

Para resolver este ejercicio contabamos con dos parámetros el número que se va a almacenar n y el número de decimales que se pueden almacenar nd .

PASO 1: Mientras que el número n sea mayor a 1 entonces contamos en i la cantidad de iteraciones y actualizamos el valor de n :

$$n = n/10$$

PASO 2: Calcular el valor del error r :

$$r = n - \frac{\text{truncamiento}(n * 10^{nd})}{10^{nd}}$$

PASO 3: Además calculamos el valor por el cual está acotado el error por el lado izquierdo y derecho :

$$\text{izquierdo} = -1 * 10^{(i - nd)}$$

$$\text{derecho} = 1 * 10^{(i - nd)}$$

Así con un número n igual a 536.78 y un número de decimales nd igual a 4, obtenemos el siguiente resultado:

```
## Error de redondeo 8.0000000000008001e-05 está acotado por: -0.1 < 8.0000000000008001e-05 < 0.1
```

Segundo punto:

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.

Para resolver este ejercicio contabamos con tres parámetros el número que se va a calcular su raíz cuadrada n , un dato x y el valor del error permitido e .

PASO 1: Calcular el valor de y , la cual es la variable que guardará la respuesta :

$$y = 0.5 * (x + \frac{n}{x})$$

PASO 2: Empezar el ciclo : PASO 3: Si $|x-y| < e$, terminar el ciclo y el valor de la raíz del número n se encuentra en y . PASO 4: Si no, actualizar los valores

$$x = y$$

$$0.5 * (x + \frac{n}{x})$$

y volvemos al PASO 2.

Así con un número n igual a 7, x igual a 100 y un error igual a 10^{-9} , obtenemos los siguientes resultados:

```
## Raíz aproximada de 7 es: 2.645751311064591 Con un error de
9.00968188943807e-12
```

Tercer punto:

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

Para resolver este ejercicio contabamos con dos parámetros el número de polinomios de Taylor n y x que representa el exponente de la expresión.

PASO 1: Inicializar las variables suma e i :

$$suma = 1, i = n$$

PASO 2: Mientras que $i > 0$. PASO 3: Actualizar los valores de las variables:

$$suma = 1 + \frac{x * suma}{i}$$
$$i = i - 1$$

Así en la variable suma se encuentra la aproximación del polinomio.

Así con un número n igual a 5 y x igual a 0.5, obtenemos los siguientes resultados:

```
## Polinomio: 1.1
## Polinomio: 1.1375
## Polinomio: 1.1895833333333333
## Polinomio: 1.2973958333333333
## Polinomio: 1.6486979166666667
## 1.6487
```

Cuarto punto:

Calcule el tamaño del error dado por las operaciones aritméticas, para la solución del siguiente problema: La velocidad de una partícula es constante e igual a 4 m/s con un error = 0.1 m/s durante un tiempo = 5 s con un error = 0.1 s. Determine el error absoluto y relativo.

Para resolver este ejercicio contabamos con cuatro parámetros la velocidad media mv , el error de esta velocidad ev , el tiempo medio mt y el error de medición de este tiempo et .

PASO 1: Calcular el valor del error absoluto :

$$abs = mv * ev + mt * et$$

PASO 2: Calcular el valor del error relativo :

$$relativo = \frac{ev}{mv} + \frac{et}{mt}$$

Así con una velocidad media mv igual a 4, el error de esta velocidad ev igual a 0.1, el tiempo medio mt igual a 5 y el error de medición de este tiempo et. igual a 0.1, obtenemos los siguientes resultados:

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

Quinto punto:

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

Para resolver este ejercicio utilizamos el algoritmo de Horner ya que con un “polinomio de grado-n requiere al menos n sumas y multiplicaciones, si se calculan mediante la repetición de multiplicaciones. El algoritmo de Horner sólo requiere n sumas y multiplicaciones”[3]. Por tanto por medio de este algoritmo podemos minimizar el número de multiplicaciones.

Para resolver este ejercicio contabamos con tres parámetros el polinomio a evaluar p, el valor del exponente máximo n y x0 que representa el x0 del polinomio.

PASO 1: Inicializar en valor de las variable:

$$resultado = p[1]$$

, valor de x en el polinomio en el término x^1 PASO 2: for con i empezando en 2 hasta llegar a n. PASO 3: Calcular el valor del resultado :

$$resultado = resultado * x0 + p[i]$$

, valor de x en el polinomio en el término x^i

Así con un polinomio a evaluar $p = 2 + -3^2 + 3^3 - 4^4$, el valor del exponente máximo n igual a 4 y un x0 igual a -2, obtenemos los siguientes resultados:

```
## El resultado del polinomio es: 10 en 8 sumas.
```

Referencias

[1]Kim, T. Noh, W. Oh and S. Park, Applied Mathematical Sciences, Vol. 11, 11th ed. Korea, 2017, pp. 2790-2796. [2]L. Rodríguez Ojeda, Análisis Numérico Básico. 2014, pp. 47-57. [3]“Algoritmo de Horner”, Es.wikipedia.org, 2019. [Online]. Available: https://es.wikipedia.org/wiki/Algoritmo_de_Horner. [Accessed: 05- Aug- 2019].