Juan Pablo Amorocho, Diego Fernando Trujillo, Jose Alejandro López, Nicolás Pérez Fonseca

Materia: Análisis Numérico

26/09/2020

Profesor: Eddy Herrera Daza

Taller 1

Contenido

Tabla de Figuras

1.Número de Operaciones

Ejercicio 1:

a)

Utilizar el método de Horner

R/Coeficientes del polinomio de ejemplo sacado del documento: -2, 0, -3, 3, -4

$x_0 = -2$

Soltando:

Número de operaciones: 8

Numero de sumas: 0.5

Numero de multiplicaciones: 4

b)

Evaluar en $x = 1.00000000001$ con $P(x) = 1 + x + x^2 + \cdots + x^{50}$, encontrar el error y compararlo con la derivada de la expresión equivalente de $Q(x) = (x^{51} - 1)/(x - 1)$

R/El error de x sería igual a 5.1127708 con P(x) y con Q(x) sería igual a 5.1127708 dando el mismo valor para los dos siendo muy cercanos los dos tipos de funciones.

Números Binarios

a)

Encuentre los primeros 15 bits en la representación binaria de π

R/Primeros 15 bits en binario de pi: 11.10110000111…

b)

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

R/Utilizando el código implementado se tiene

- 10 = 2

-1010101 = 85

-1011.101 = 11.5

- 10111.010101... = 23.21

-111.1111... = 7.15

c)

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

R/Utilizando el código implementado se obtiene que

- 11.25= 1011.11001

-2 /3= 0.1000010

-30.6= 11110.110

- 99.9 = 1100011.1001

Representación del Punto Flotante de los Números reales

Ejercicio 1:

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

R/Se pueden utilizar distintos métodos como el de redondeo o truncamiento para así representar un valor más pequeño para la aproximación finita de bits y asi tenerlo en valores más manejable para el proceso.

Ejercicio2:

¿Cuál es la diferencia entre redondeo y recorte?

R/En el redondeo simplemente se aproxima los valores en un lugar determinado para así seguir el bit siguiente ya sea para arriba o para abajo. Mientras que en el recorte se quitan bits en una posición determinada para mejorar el proceso.

Ejercicio 3:

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

R/ Se intenta aproximar para que quede lo más parejo posible al poder ser un número tan grande para su uso por medio de método de acercamiento se tiende a utilizar una diferente precisión o forma de limitarla.

Ejercicio 4:

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

R/ Representamos el entero en binario

0,4=100110011001100…

que también podría ser visto como

1,100110011001…

luego definimos que el exponente es -1 por lo tanto la característica es 1022 que equivale a expresarlo en forma binaria como

1111111110.

La matisa corresponde a los primeros 52 dígitos después de la coma (“,”) y se utiliza el método de redondeo para aproximar lo obtenido dando

10011001…10010

por lo que el final termino así entonces fue redondeado hacia arriba.

El número 0.4 en IEEE corresponde a:

$$fl(0{,}4) = 0{,}4000000000000000222044604925031308084726333618164062 5$$

Ejercicio 5:

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

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

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

R/El error relativo corresponde a:

$$= \frac{fl(0.4) - 0{,}4}{|0.4|} = 5{,}551115 \times 10^{-17}$$

El error de la maquina seria:

$$= \frac{1}{2} * 2^{-52} = 1{,}110223 \times 10^{-16}$$

Ejercicio 6:
Verificar si el tipo de datos básico de R y Python es de precisión doble IEEE y Revisar en R y Phython el formato long.

R/Python y R utilizan la aritmética de punto flotante IEEE-754 para el mapeo de los datos se usaría doble de precisión de IEEE-754

Ejercicio 7:
Encuentre la representación en número de máquina hexadecimal del número real 9.4

R/ Se utiliza la herramienta hexmode en R para encontrar el valor hexadecimal de un número real que en este caso seria 9.4 que se expresaría como 9.4 también.

Ejercicio 8:
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 contrarrestarla.

R/Utilizando wólfram Alpha se obtiene que: $x = -\frac{282429536481}{2} - \frac{\sqrt{7976644307687250986337 3}}{2}$ y $x = \frac{6}{282429536481+\sqrt{7976644307687250986337 3}}$.

Y al comparar ambos resultados con precisión doble al restarlos serian: $-2{,}82429536481000 \ldots \times 10^{11}$

Por lo visto la diferencia al restarlo es grandísima por lo que pueden ser respuestas independientes.

Ejercicio 9:
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

R/Se podría utilizar algún método iterativo para simplificar el trabajo como por ejemplo em método de bisección ya que lo podríamos limitar a una tolerancia fija para que el valor de b no sea tan demandante y si no fuera útil solo se detendría para tomarlo desde otro enfoque.

2.Raíces de una Ecuación

Ejercicio 1:

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

Se implemento un algoritmo que suma la parte triangular inferior de una matriz incluyendo la diagonal. Como entrada de datos fue un arreglo con los siguientes valores:

$$n = (5{,}10{,}15{,}20{,}25{,}30{,}35{,}40{,}45{,}50)$$

El algoritmo creaba automáticamente las matrices A_n con todos sus valores en 1 y contaba cuantas sumas se realizabas en cada matriz que también sería el resultado de la suma de los valores de la triangular inferior debido a que todos los valores son igual a 1. Se obtuvo los siguientes datos:

A_n	Número de Sumas
5	15
10	55
15	120
20	210
25	325
30	465
35	630
40	820
45	1035
50	1275

Al realizar la gráfica de los datos se obtiene lo siguiente:

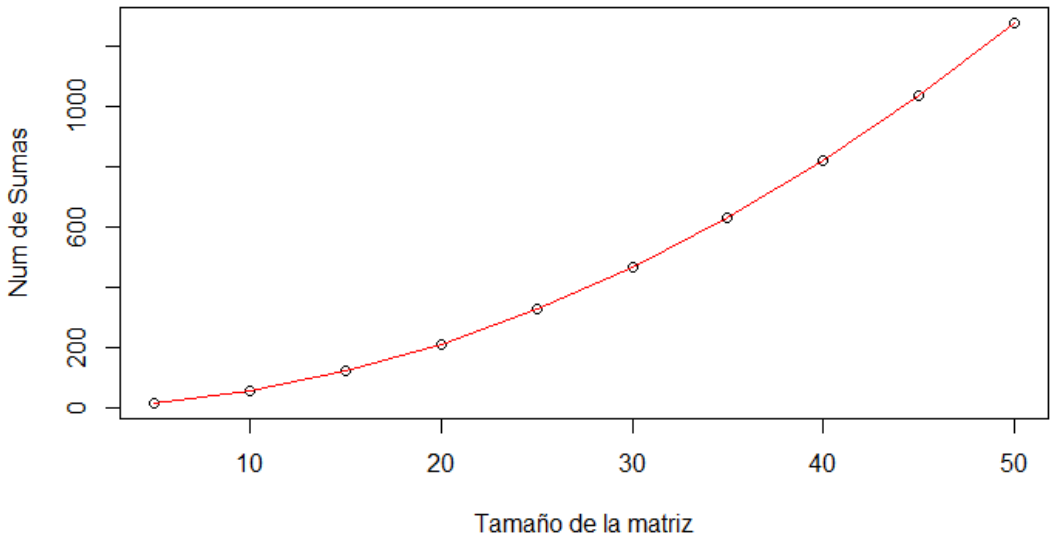

Ilustración 1 Número de Sumas vs Tamaño de la Matriz

Se puede observar que la función crece cuadráticamente, por tanto, la función f(n) tiene complejidad $O(n^2)$.

Ejercicio 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.

Se resolvió este problema de dos formas, primero se implementó un algoritmo que simplemente hiciera la suma de los valores, se hicieron pruebas con los siguientes valores:

$$n = (10,20,30,40,50,60,70,80,90,100)$$

Se obtuvieron los siguientes resultados:

n	Resultado de la suma
10	385
20	2870
30	9455
40	22140
50	42925
60	73810
70	116795
80	173880
90	247065
100	338350

Al realizar la gráfica de los resultados se obtiene lo siguiente:

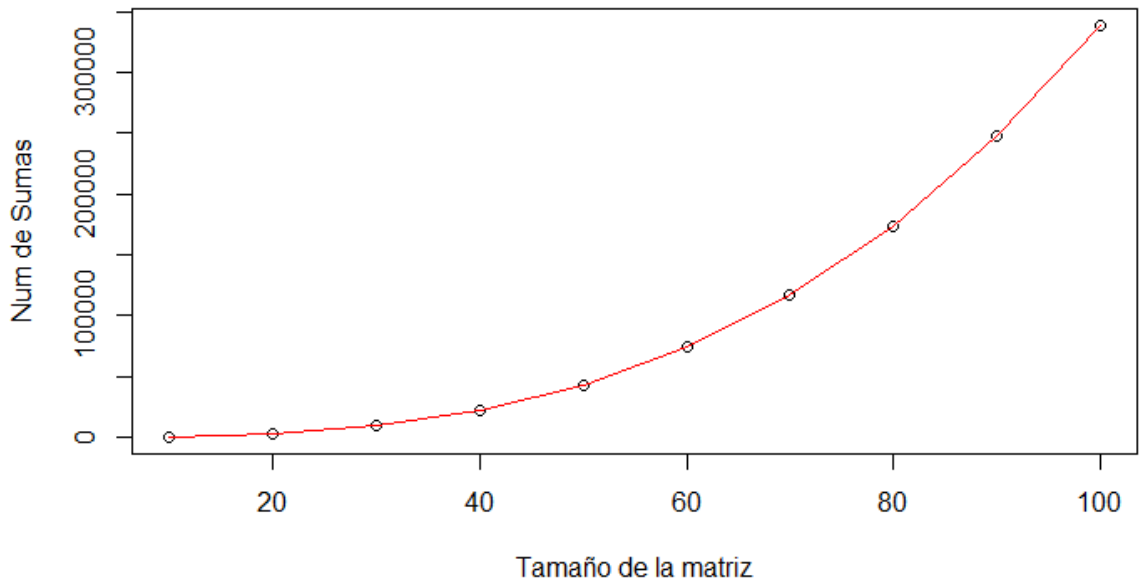

Ilustración 2 Número de Sumas vs Tamaño de la Matríz

Se puede observar que la $f(n)$ crece en función $O(n^3)$, quiere decir que, crece muy rápido cada vez que se aumenta $n+1$. En cambio, si se analiza la complejidad del algoritmo es $O(n)$ ya que siempre se van a ser $n-1$ sumas.

La segunda forma en la que se solucionamos el problema fue realizando la fórmula de la suma que es la siguiente:

$$\frac{n(n+1)(2n+1)}{6}$$

Al realizar las pruebas con los mismos valores se obtiene el mismo resultado que con la primera forma que solucionamos. Pero algo a observar es que es mejor utilizar esta segunda forma ya que la complejidad del algoritmo es $O(1)$ ya que no ha de hacer iteraciones, si no que obtiene el resultado de una vez, esto ahorra recursos para el computador y tiempo de ejecución.

Ejercicio 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 está colocado verticalmente sobre la tierra. Utilizando dos métodos de solución de ecuación no lineal, encuentre la altura máxima que alcanza el cohete.

Para encontrar el punto más alto se tiene que derivar la función y con la derivada encontrar la raíz. Se hizo la derivada por medio de la aplicación de wolframalpha dando como resultado lo siguiente:

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

Primero se trató de encontrar la raíz por medio de la implementación del método de punto fijo y Newton-Rapson pero se encontró el problema que esta función también tiene una raíz en 0, por tanto, estos dos métodos siempre convergía hacia el punto 0.

Por tanto, se utilizó la función polyroot() que está disponible en R, esta función encuentra las raíces de una función por medio del método de Jenkins-Traub(1972), al pasarle como parámetro $\boldsymbol{y'(t)}$ se obtuvo los siguientes valores:

- 0.00000000000000+0i
- 28.62220762329085-0i
- -28.62220762329085+0i

Para comprobar el valor también se utilizó la función uniroot() que también encuentra las raíces de una función, al pasarle como parámetro $\boldsymbol{y'(t)}$ se obtuvo el siguiente valor:

- 28.62220762329078

Al hacer esto ya sabemos que el valor de la raíz es aproximadamente 28.622207, ya por último implementamos el algoritmo de Brent que realizamos en el reto 1 y nos arrojó el siguiente resultado:

- 28.6222076267004 con una tolerancia de $1e^{-8}$

Por tanto, se puede concluir que el valor de la raíz es de 28.62220762 que al remplazarlo en $\boldsymbol{y(t)}$ da:

$$y \approx 878.48076$$

En conclusión, la máxima altura del cohete es de 879.48076 metros en el segundo 28.62220762.

3.Convergencia de Métodos Iterativos

Ejercicio 1:

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

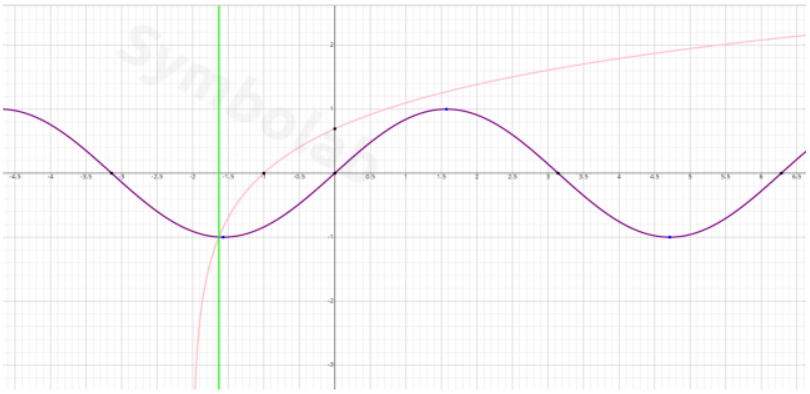

Ilustración 3Gráfia de las funciones y su intersección

a)

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

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

Aplicando la fórmula sin importar el punto de partida que tengo luego de la segunda iteración el algoritmo cae en indeterminación. Lo que sólo permite graficar un solo punto y no permite ver con claridad el comportamiento de este para determinar su complejidad mediante la notación de O grande.

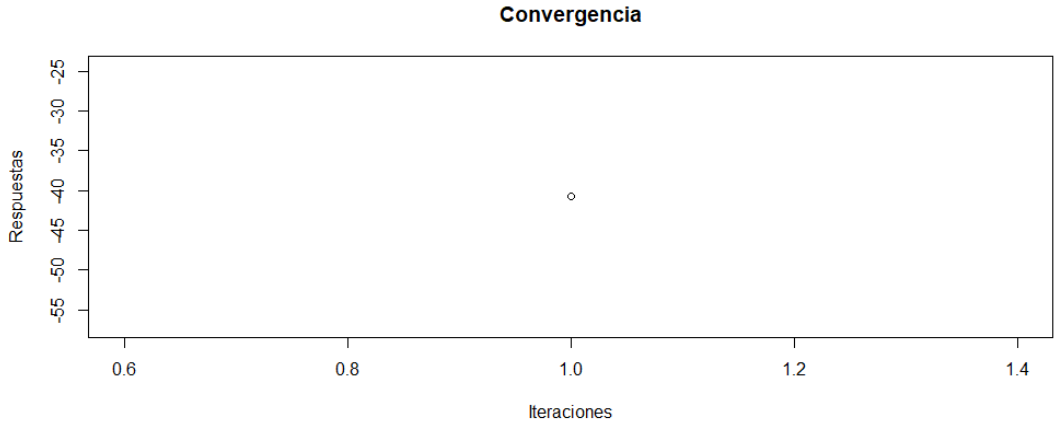

Ilustración 4 Gráfica de la convergencia del método recursivo

Se supone que la complejidad de estos algoritmos debería ser cuadrática es decir $O(n^2)$.

b)

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

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

Así sea variando el error este algoritmo iterativo, por lo general converge cuando se aproxima a su respuesta. En este caso tampoco pasa de la segunda iteración pues ambas funciones se van acercando a cero con cada interacción con la fórmula mas no se acerca a la intersección de las dos curvas. Para este caso la complejidad de este podría ser: $O(n^2)$ pero tampoco se puede estar seguro porque el comportamiento no se deja ver ya que solo permite graficar un solo punto antes de quedar indeterminada el cálculo de las imágenes en las funciones.

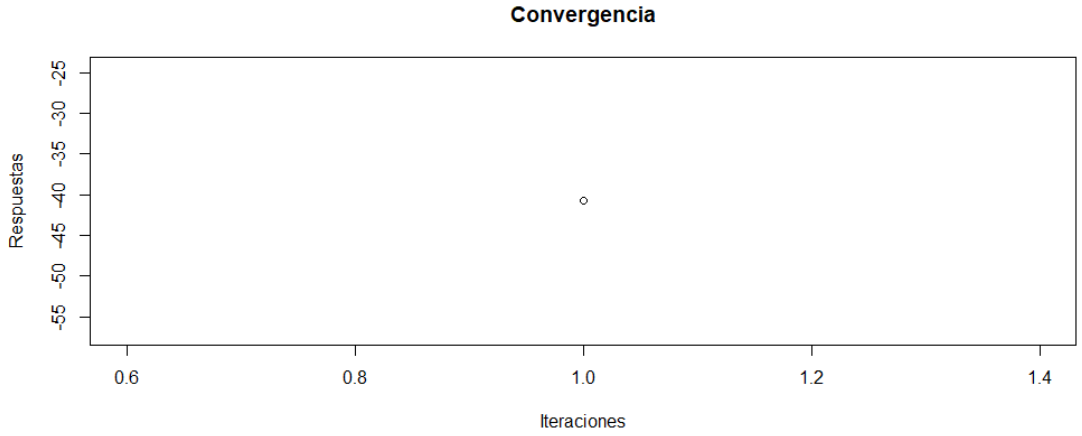

Ilustración 5 Gráfica de convergencia del método iterativo

Ejercicio 2:

Determine el valor de los coeficientes de 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 $E = 10^{-8}$.

Tenemos el siguiente sistema de ecuaciones no lineales, porque tenemos exponenciales.

$$a + (a + b)e^{a+b} = 3$$
$$a + (2a + b)e^{2a+b} = 4$$

La idea es tomar estas ecuaciones y convertirlas en un arreglo, luego hacer las derivadas parciales de las mismas primero respecto a a y luego respecto a b, en cada una de las dos, con lo que el seguido arreglo queda también con do variables, pero dentro de cada entrada tiene las 2 derivadas.

Dentro de la función se recién los dos arreglos que acabamos de hacer, la tolerancia y los puntos iniciales, que en este caso son (1,2). Luego se calcula el delta de x resolviendo las ecuaciones ya que tenemos un sistema lineal, y enseguida calculamos el error, haciendo el cociente entre la norma del delta de x, y x que es son nuestras respuestas, dicho error debe ser menor que la tolerancia para que salga del ciclo, de lo contrario, lo hará tantas veces como sea necesario.

Desafortunadamente no funcionó y no se pudieron hallar a y b. Pero acá están los vectores usados:

$$ec1 = lambda\ x1, x2: [x1 + (x1 + x2) * 2.7 ** (x1 + x2) - 3, x1 + (2 * x1 + x2) * 2.7 ** (2 * x1 + x2)]$$
$$ec2 = lambda\ x1, x2: [[2.7 ** (x1 + x2) + 2.7 ** (x1 + x2) * (x1 + x2) + 1, 2.7 * (x2 + x1) + 2.7 ** (x2 + x1) * (x2 + x1)], [2 * 2.7 ** (2 * x1 + x2) + 2 * 2.7 ** (2 * x1 + x2) * (2 * x1 + x2) + 1, 2.7 ** (x2 + 2 * x1) + 2.7 ** (x2 + 2 * x1) * (x2 + 2 * x1)]]$$

Y el comando usado para llevar a cabo el proceso es el siguiente:

$$ecuaciones(ec1, ec2, [1,2], 1e - 8)$$

Newton

Ejercicio1:

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

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

Luego, para demostrar su multiplicidad debe poder escribirse así:

$$f(x) = (x - p)^m q(x)$$

Tomamos a q(x) como;

$$q(x) = -x - 1$$

Y a el primer producto como:

$$(x - p)^m = w^a = e^x$$

El exponente que nos indica la multiplicidad del cero lo vamos a llamar a y es equivalente

$$a = \frac{x}{\ln(w)}$$

Sabiendo que la complicidad es 2 porque nos lo dicen desde el comienzo, podemos averiguar w

$$2 = \frac{x}{\ln(w)}$$
$$w = e^{\frac{x}{2}}$$

Reemplazando en la expresión general de la multiplicidad según el teorema tenemos:

$$f(x) = \left(e^{\frac{x}{2}} - 0\right)^2 - x - 1$$

Por lo tanto, como pudo ser reescrita de esta forma, se comprueba que su multiplicidad 2.

Ejercicio2:

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

Luego de ejecutar el algoritmo, podemos ver como la convergencia de este a medida que pasan las iteraciones se acerca a cero, pero forma solamente media parábola como se puede ver en el gráfico que tenemos a continuación.

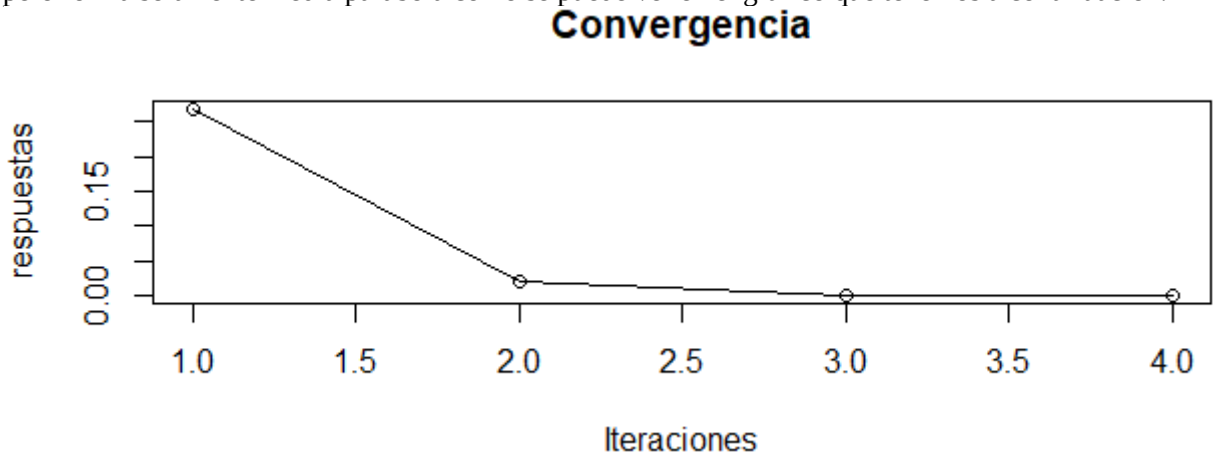

Ilustración 6 Gráfica de convergencia del método de Newton

La salida del programa muestra a continuación la siguiente salida:

```
> newton(1)
[1] -3.565619e-17
>
```

Con el error más alto permitido por R estándar, lo hace en 4 iteraciones, cada punto representa una iteración.

Ejercicio3:

¿Utilizando el método de Newton generalizado, mejora la tasa de rendimiento? explique su respuesta.
Al parecer, requiere una iteración menos en hallar la raíz de la función, más sin embargo requiere de la segunda derivada para llevarse a cabo. A continuación, la comparación lado a lado de las dos corriendo en igualdad de condiciones.

Newton	Newton Generalizado

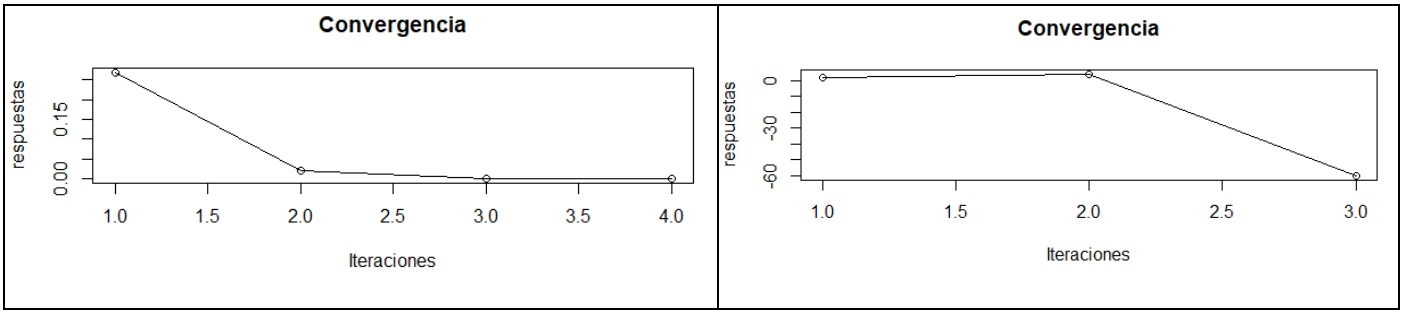

Ilustración 7 Gráfica comparativa de Newton vs Newton Generalizado

Acá se puede ver cómo le toma una iteración menos en hallar la raíz de la función.

4.Convergencia acelerada

Método de Δ2 Aitken

Ejercicio 1:

1. La función $1 = \lim_{n\to\infty} \cos\frac{1}{n}$. En este caso, similar a con el seno, se encuentra fácilmente que $O(\frac{1}{n})$

2. Para comparar la diferencia entre usar el método de Aitken, se hizo una iteración de punto fijo sobre la función $x = \cos\frac{1}{n}$**.** Se compararon los primeros tres valores, y se puede ver la gran diferencia que cause esta aceleración:

```
Sin Aitken:
Resultado 1:  -0.4161468365471424
Resultado 2:  -0.7394154115062934
Resultado 3:  0.2166450495288271
Aitken:
Resultado 1 = [1] -0.4978322893242716
Resultado 2 = [1] -0.494316168298044
Resultado 3 = [1] -0.4894248389471225
Final del Programa de Comparación.
```

Ilustración 8 Resultados sin aceleración

3. Al usar el método de punto fijo con el método de Aitken (En este caso se usó el método de Steffensen, ya que da una respuesta, aunque se puede llegar a la misma conclusión usando Aitken), se puede ver que las dos funciones resultan con los mismos valores en las iteraciones.

```
Valores de la funcion f(t): t_ 1 = 1 'mpfr' number of precision  16   bits
[1] 0.922668
Valores de la funcion g(t): t_ 1 = 1 'mpfr' number of precision  16   bits
[1] 0.922668
Valores de la funcion f(t): t_ 2 = 1 'mpfr' number of precision  16   bits
[1] 1.16028
Valores de la funcion g(t): t_ 2 = 1 'mpfr' number of precision  16   bits
[1] 1.16028
Valores de la funcion f(t): t_ 3 = 1 'mpfr' number of precision  16   bits
[1] 1.22888
Valores de la funcion g(t): t_ 3 = 1 'mpfr' number of precision  16   bits
[1] 1.22888
Valores de la funcion f(t): t_ 4 = 1 'mpfr' number of precision  16   bits
[1] 1.23715
Valores de la funcion g(t): t_ 4 = 1 'mpfr' number of precision  16   bits
[1] 1.23715
Valores de la funcion f(t): t_ 5 = 1 'mpfr' number of precision  16   bits
[1] 1.2373
Valores de la funcion g(t): t_ 5 = 1 'mpfr' number of precision  16   bits
[1] 1.2373
Valores de la funcion f(t): t_ 6 = 1 'mpfr' number of precision  16   bits
[1] 1.2373
Valores de la funcion g(t): t_ 6 = 1 'mpfr' number of precision  16   bits
[1] 1.2373
Final del Programa
```

Ilustración 9 Valores de las funciones

Método de Steffensen

Se aplicó el método de Steffensen para la función $x^2 - \cos x$ y se comparó con el método de Aitken, resultando en las siguientes conclusiones:

Con Tolerancia: 10^{-8}

Aitken:

```
Resultados de la Iteración de Aitken con tolerancia 1e-8:
> iteracion(f,X0, Error1e8, 100, 0)
x_ 1 = [1] -0.5441370020386918

x_ 2 = [1] -0.5499290916610792

x_ 3 = [1] -0.5500093349592889

x_ 4 = [1] -0.5500093501293957

x_ 5 = [1] -0.5500093640911168

x_ 6 = [1] -0.5500093501018347

x_ 7 = [1] -0.5500093468492401

x* es aproximadamente [1] -0.5500093468492401
```

Ilustración 10 Resultados de Aitken con tolerancia 1e-8

Steffensen:

```
Resultados de la Iteración de Steffensen con tolerancia 1e-8:
> iteracion(f,X0, Error1e8, 100, 1)
x_ 1 = [1] -0.5441370020386918

x_ 2 = [1] -0.5499290916610787

x_ 3 = [1] -0.5500093349592922

x_ 4 = [1] -0.550009349927261

x_ 5 = [1] -0.5500093499272616

x* es aproximadamente [1] -0.5500093499272616
 con error menor que  1e-08
```

Ilustración 11 Resultados de Steffensen con tolerancia de 1e-8

Conclusiones:

Para los dos métodos, se puedo encontrar una raíz. Claramente se puede ver que, para este nivel de tolerancia, se encontraron las raíces de manera rápida. Steffensen encuentra la respuesta de una manera más rápida, ya que, por su naturaleza, cambia el orden de convergencia de la función a una cuadrada.

Con Tolerancia: 10^{-16}

Con esta tolerancia, se tuvo que usar la librería Rmpfr para incrementar la precisión.

Aitken:

```
x_ 90 = 1 'mpfr' number of precision  16   bits
[1] -0.549805

x_ 91 = 1 'mpfr' number of precision  16   bits
[1] -0.548706

x_ 92 = 1 'mpfr' number of precision  16   bits
[1] -0.550308

x_ 93 = 1 'mpfr' number of precision  16   bits
[1] -0.549332

x_ 94 = 1 'mpfr' number of precision  16   bits
[1] -0.550201

x_ 95 = 1 'mpfr' number of precision  16   bits
[1] -0.547104

x_ 96 = 1 'mpfr' number of precision  16   bits
[1] -0.549911

x_ 97 = 1 'mpfr' number of precision  16   bits
[1] -0.554047

x_ 98 = 1 'mpfr' number of precision  16   bits
[1] -0.549896

x_ 99 = 1 'mpfr' number of precision  16   bits
[1] -0.546509

x_ 100 = 1 'mpfr' number of precision  16   bits
[1] -0.549957

Se llego al numero de iteraciones máximas > cat("R
ncia 1e-16: \n")
```

Ilustración 12 Resultados de Aitken con tolerancia 1e-16

Steffensen:

```
Resultados de la Iteración de Steffensen con tolerancia 1e-16:
> iteracion(f,X0, Error1e16, 100, 1)
x_ 1 = 1 'mpfr' number of precision  16   bits
[1] -0.544128

x_ 2 = 1 'mpfr' number of precision  16   bits
[1] -0.549927

x_ 3 = 1 'mpfr' number of precision  16   bits
[1] -0.550003

x_ 4 = 1 'mpfr' number of precision  16   bits
[1] -0.550003

x* es aproximadamente 1 'mpfr' number of precision  16   bits
[1] -0.550003
 con error menor que  1e-16
```

Ilustración 13 Resultados de Steffensen con tolerancia 1e-16

Conclusiones:

Podemos ver dos resultados totalmente distintos para los dos métodos. Gracias una restricción mayor en términos de tolerancia, el método de Aitken no puede llegar a un resultado, ya que sobrepasa incluso las 100 iteraciones, y los datos por cada iteración resultan en un ciclo, nunca llegando a la raíz deseada. Steffensen, llega a la respuesta en cuatro iteraciones, significando que una mayor tolerancia no cambia en mucho el tiempo que le toma a Steffensen a

llegar a su respuesta. Esta respuesta toma menos iteraciones que la anterior gracias a la mayor precisión usada en esta comparación.

Referencias

[1] J. C. Gorostizaga, «EHU,» [En línea]. Available: http://www.ehu.eus/juancarlos.gorostizaga/mn11b/temas/horner.pdf. [Último acceso: 26 Septiembre 2020].

[2] Zator, «Zator Curso de C++,» 2016. [En línea]. Available: https://www.zator.com/Cpp/E2_2_4a.htm. [Último acceso: 26 Septiembre 2020].

[3] W. Alpha, «Wolfram Alpha,» Wolfram Alpha, 2020. [En línea]. Available: https://www.wolframalpha.com/input/?i=x%5E2%2B9%5E12+x%3D3+. [Último acceso: 26 Septiembre 2020].

[4] Desconocido, «Tarwi,» Universidad Agraria del Perú, 2008. [En línea]. Available: https://tarwi.lamolina.edu.pe/~fmendiburu/index-filer/academic/script_numerico.htm. [Último acceso: 26 Septiembre 2020].