Reto 1

Pedro Guerrero, Carlos Erazo , María Alejandra Sandoval, Valentina Rozo

1. Evaluación de raíces de un polinomio

Ejercicios a desarrollar:Problema 1: Evaluar las raíces de un polinomio implica varios desafíos, aun para el software profesional. Como se requiere poder evaluar las posibles raíces del polinomio, es necesario dedicarle un momento a los detalles como, las derivadas evaluadas en valores x0. Problema: Sea P(x) = a0x n + a1x n1 + ... + an un polinomio entonces, implementar en R y/o Python una modificación del método de Horner que evalúe de manera eficiente f' (X0) y f"(X0) la primera y segunda derivada en cualquier punto.

Explicación del algoritmo: Inicialmente se toma un polinomio, en seguida Horner lo deriva y lo evalúa de manera eficiente, después para el cálculo de las raíces se utiliza el método de Newton-Raphson el cual las calcula utilizando la primera derivada del polinomio en cuestión, en su método utiliza Horner para derivar y calcular los valores que necesita. Luego se realiza el mismo procedimiento con Laguerre, el cual toma la primera y segunda derivada resolviéndolas mediante Horner para calcular los valores necesarios.
El método de Laguerre trabaja con la siguiente ecuación:

$$a = \frac{n}{(G \pm \sqrt{(n+1)(nH - G^2)})} \tag{1.1}$$

Donde G hace referencia a la división de la primera derivada evaluada en x y el polinomio evaluado en x, H equivale a la segunda derivada evaluada dividido el polinomio evaluado[1]. La ecuación anterior se resuelve hasta que encuentre la raíz.

El método de Newton-Raphson trabaja con la siguiente ecuación:

$$g(x_n) = x_n - \frac{p(x_n)}{p'(x_n)} \tag{1.2}$$

Esta ecuación trabaja con la primera derivada, en este caso se va actualizando la x siguiente mediante la división del polinomio evaluado entre la derivada evaluada.

El polinomio a evaluar fue el siguiente:

$$x^4 - 5x^3 - 9x^2 + 155x - 250 \tag{1.3}$$

*correspondingauthor@example.edu

1.1. **Precisión.** En este caso para asegurar una precisión se utiliza complex128() de la biblioteca Numpy de Python, esta expresión significa que utiliza dos flotantes de 64 bits. Por otro lado la razón para utlizar complex y no otro tipo de dato es porque dentro del algoritmo cabe la posibilidad de ingresar números complejos como coeficientes o se puede llegar a obtener alguna raíz compleja.

1.2. **Raíces imaginarias.** En Python para poder manipular un número complejo basta con escribir el número sustituyendo la i de imaginario con una j, lo anterior se puede entender de la siguiente manera 2 + 3j, en donde 2 equivale a la parte real y 3j a la parte imaginaria. Para el polinomio propuesto las raíces encontradas con el método de Laguerre y Newton-Raphson fueron:
1. -5
2. 2
3. 4 + 3j
4. 4 - 3j

1.3. **Pruebas.** Se probó el algoritmo con un polinomio de números complejos:

$$(1+2j)x^2 - (2+1j)x + 9 \tag{1.4}$$

Para este caso el algoritmo funcionó tomando los coeficientes de números complejos y encontrando sus rices complejas. Este se probó con una tolerancia de hasta 10**-24 con la que encontró las raíces y en términos de eficiencia, donde Laguerre es mejor que Newton-Raphson.

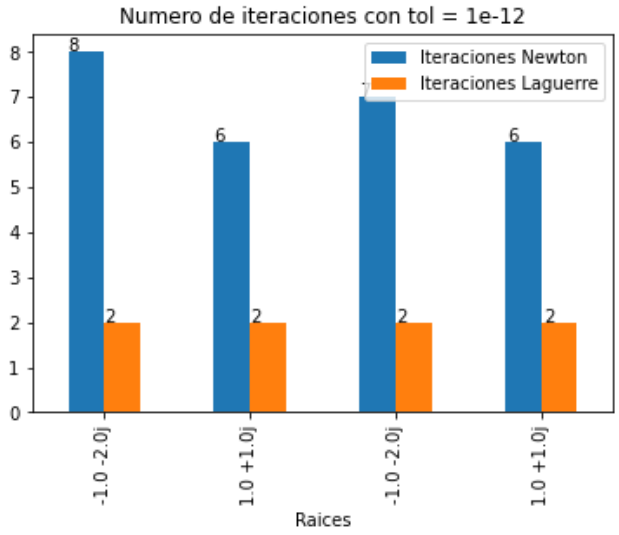

Figure 1.
Comportamiento del número de iteraciones en Laguerre y Newton-Raphson respecto a la tolerancia de 10**-12 de la función $(1+2j)x^2 - (2+1j)x + 9$

Como se puede observar en la figura anterior el método de Laguerre converge de forma más rápida para encontrar las raíces del polinomio. A continuación se pueden observar los valores resultantes con el método de Laguerre y Newton-Raphson:

Raices de Laguerre con tolerancia 1e-12

Raiz: (-1.0229283399377174-2.259654053684345j) Iteraciones: 2
Raiz: (1.0229283399377183+1.259654053684342j) Iteraciones: 2
Raiz: (-1.0229283399377174-2.259654053684341j) Iteraciones: 2
Raiz: (1.0229283399377185+1.2596540536843435j) Iteraciones: 2

Raices de Newton con tolerancia 1e-12

Raiz: (-1.0229283399377183-2.2596540536843417j) Iteraciones: 8
Raiz: (1.022928339937718+1.2596540536843415j) Iteraciones: 6
Raiz: (-1.022928339937718-2.2596540536843417j) Iteraciones: 7
Raiz: (1.0229283399377471+1.2596540536843348j) Iteraciones: 6

1.4. **Validaciones.** Orden de convergencia

En el caso del problema a solucionar se encontró que el método de Laguerre tiene una convergencia cúbica, lo cual hace cual hace que sea más eficiente que otros algoritmos. Mientras que el método de Newton-Raphson tiene convergencia cuadrática, lo cual hace, que por más de que no sea el algoritmo más lento si sea menos eficiente que Laguerre.

1.5. **Comparaciones.** Los métodos de Laguerre-Horner y Newton-Horner se compararan con el método de bisección con una tolerancia de 10**-24.

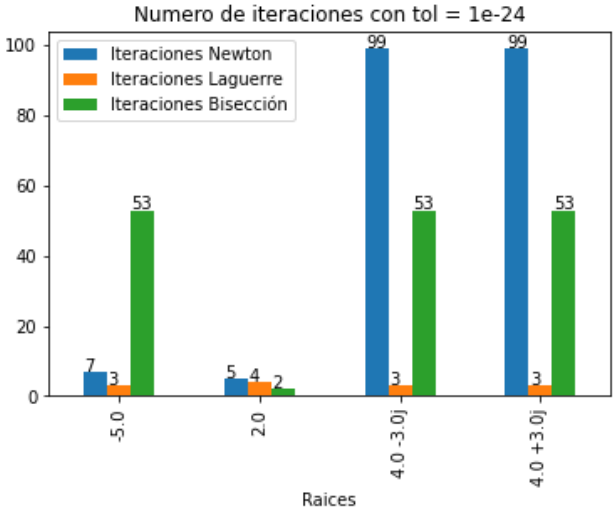

Figure 2.
Comportamiento del número de iteraciones en Laguerre, Newton-Raphson y Bisección respecto a la tolerancia de 10**-24 de la función $x^4 - 5x^3 - 9x^2 + 155x - 250$

Como se puede ver en la gráfica anterior, y comparándolo con el método de bisección se puede ver que al menos en 3 de las 4 raíces, Laguerre es más eficiente. Solo con la raíz = 2 bisección tiende a ser mejor, esto ya que hay que recordar que bisección para hacer menos iteraciones debe estar mas cerca de la raíz, y para esta raíz el rango en el que se evaluó bisección fue de 1 a 5, por lo que encontró la raíz más rápido.

1.6. **Gráficas.** Para comparar el método de Laguerre utilizando Horner, con el de Newton-Raphson implementando Horner se utilizaron tres tolerancias; 10**-8, 10**-12 y 10**24. Esto se comparó utilizando las iteraciones en cada uno de los métodos en cada una de las raíces halladas.

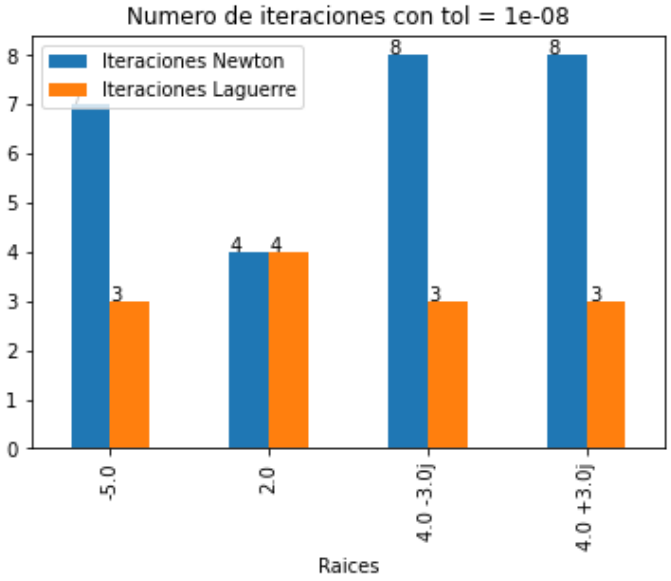

Figure 3.
Comportamiento del número de iteraciones en Laguerre y Newton-Raphson respecto a la tolerancia de 10**-8 de la función $x^4 - 5x^3 - 9x^2 + 155x - 250$

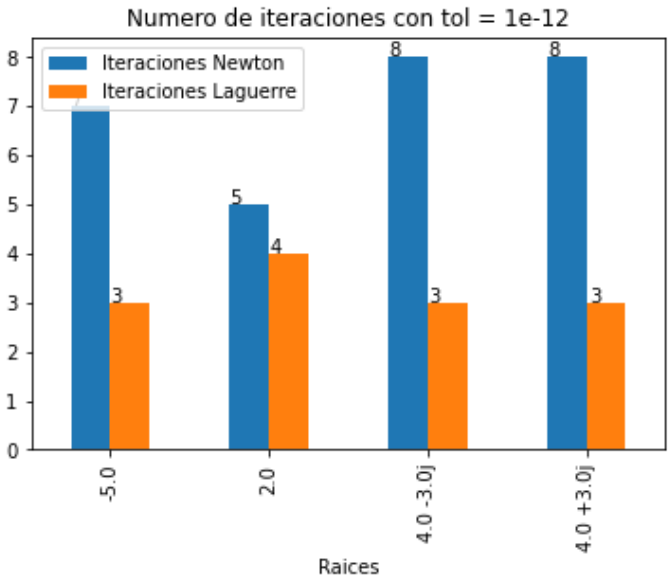

Figure 4.
Comportamiento del número de iteraciones en Laguerre y Newton-Raphson respecto a la tolerancia de 10**-12 de la función $x^4 - 5x^3 - 9x^2 + 155x - 250$

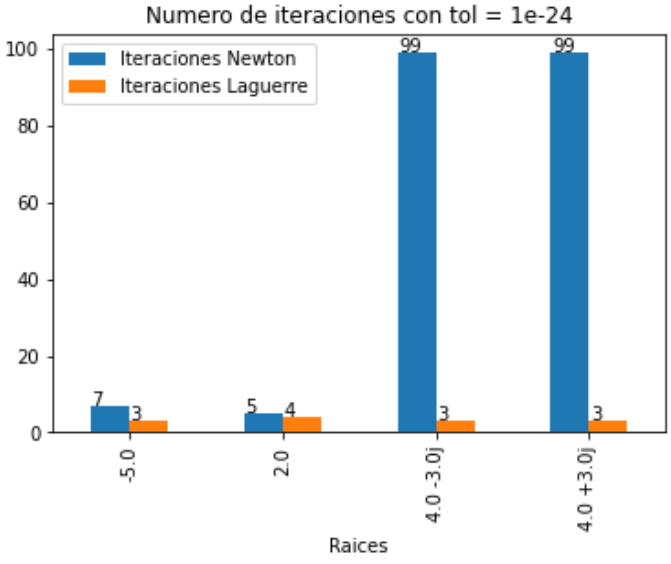

Figure 5.
Comportamiento del número de iteraciones en Laguerre y Newton-Raphson respecto a la tolerancia de 10**-24 de la función $x^4 - 5x^3 - 9x^2 + 155x - 250$

Como se puede observar en las gráficas anteriores el método de Laguerre es más eficiente al momento de comparar cada una de las raíces que el método de Newton-Raphson pese a que este último solo utiliza una de las derivadas obtenida con Horner para el cálculo de la raíz.

1.7. **Procedimiento.** A continuación se enumeran los pasos principales durante el desarrollo del algoritmo:

1. El algoritmo inicia cuando llega un polinomio a evaluar, utilizando Horner se puede calcular la primera y segunda derivada en cualquier punto.

2. Luego se calculan las raíces con el método de Newton-Raphson, este método utiliza Horner para evaluar el polinomio y la primera derivada para irse aproximando a la raíz. De acá se obtiene la raíz y el número de iteraciones realizadas.

3. Se hace el mismo procedimiento con Laguerre, en este caso este método se vale de Horner para evaluar el polinomio, la primera y la segunda derivada.

1.8. **Conclusión.** Se puede concluir que el método de Laguerre implementado con Horner es más eficiente que el método de Newton implementado con Horner ya que de acuerdo a los casos presentados anteriormente presenta un menor número de iteraciones, converge más rápido.

2. Algortimo Brent

Este algoritmo de Brent, utiliza en cada punto lo más conveniente de las estrategias del de la bisección y del de la secante (o Muller). En su acepción moderna fue formulado en 1973 por Richard Peirce Brent, Australia, 1946.El método de Brent, también conocido como zeroin, ha sido el método más popular para encontrar ceros de funciones desde que se desarrollo en 1972. Este método suele converger muy rápidamente a cero; para las funciones difíciles ocasionales que se encuentran en la práctica.

Explicación del algoritmo:
El método de Brent es un algoritmo que combina el método de bisección y el método de la secante para calcular la raíz de forma eficiente[2]. El algoritmo funciona de la siguiente manera; primero verifica si ambos límites del intervalo son del mismo signo o no, de ser diferentes hace una iteración del método de la secante, de lo contrario evalúa si el próximo resultado se encuentra dentro del siguiente intervalo, de ser así aplica nuevamente le método de la secante, de lo contrario aplica bisección. Continua de esta manera hasta que

encuentra el valor de la raíz o hasta que llega al número máximo de iteraciones.

Para el calculo de los valores provisionales se sigue con la condición para secante, en caso de que una iteración y su siguiente iteración evaluados en la función sean diferentes:

$$b_k - ((b_k - b_k - 1)/(f(b_k) - f(b_k - 1))) * f(b_k) \tag{2.1}$$

Y la condición del método de bisección:

$$m = (a_k + b_k)/2 \tag{2.2}$$

Siendo $a_k y b_k contrapuntos$.[2]

2.1. **Precisión y validaciónes.** Conociendo el resultado esperado de 0.666... que es un decimal periódico, se agrega una excepción al algoritmo de Brent, buscando "completar" la posible tendencia de un numero a ser periódico, como parámetro se compara un resultado y si este en su parte decimal lleva más de tres números consecutivos iguales se busca la fracción correspondiente al decimal periódico y se evalúa en la función con el objetivo de obtener un resultado igual a 0 y terminar con el algoritmo. Para lograr esta fracción se usan las librerías "Decimal" y "fractions" garantizando la precisión que no podría lograr el algoritmo solo ejecutando las operaciones y métodos convencionales. Para casos donde esto no ocurre se garantiza la precisión usando el tipo de dato float128 de la librería "numpy".

2.2. **Raíces imaginarias.** Para tener en cuenta si el método cumple con encontrar raíces imaginarias se realizan pruebas con el polinomio 1.3 obteniendo el resultado esperado en 2 iteraciones.

2.3. **Pruebas.** Siguiendo el flujo principal del ejercicio, se evalúa el algoritmo en la siguiente ecuación:

$$x^3 - 2 * x^2 + (4/3) * x - (8/27) \tag{2.3}$$

La implementación trabajada cumple con las expectativas, mejorando la precisión al aumentar la tolerancia como se observa en la figuas 6, figura 7 y figura 8, donde se resalta el detalle que adquiere la gráfica al someterse a una mayor tolerancia. Se presentan los siguientes resultados y una comparación que se ve reforzada con la gráfica 9, los resultados con su respectiva comparación frente a otros métodos tratados:

Raíces con tolerancia 1e-8
Método de Brent: 0.668673449222, iteraciones: 27
Método de Steffensen: 0.666593695025, iteraciones: 52

Método de bisección: 0.666669867933, iteraciones: 29
Raíces con tolerancia 1e-16
Método de Brent: 0.666671439616, iteraciones: 48
Método de Steffensen: 0.666593695025, iteraciones: 52
Método de bisección: 0.666669870727, iteraciones: 32
Raíces con tolerancia 1e-32
Método de Brent: 0.666669872646, iteraciones: 74
Método de Steffensen: 0.666593695025, iteraciones: 52
Método de bisección: 0.666669870727, iteraciones: 32

2.4. **Comparaciones.** Para verificar la eficiencia del método de Brent se evalúa la misma ecuación del flujo principal (2.1) frente a otros métodos ya trabajados, los resultados se presentan en la Figura 9. El análisis de esta figura indica lo esperado frente a la definición del algoritmo de Brent, obteniendo muy buenos resultados para niveles de tolerancia bajos, garantizando un numero menor de iteraciones comparado con bisección, pero conforme la tolerancia aumenta, el numero de iteraciones aumenta considerablemente al punto de reducir su "competencia" frente a otros métodos.

2.5. **Gráficas.**

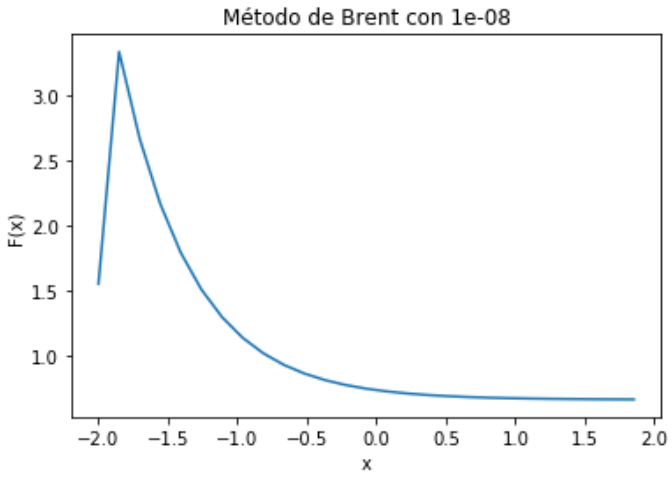

Figure 6.
Resultado con el método de Brent bajo una tolerancia de 10**-8.

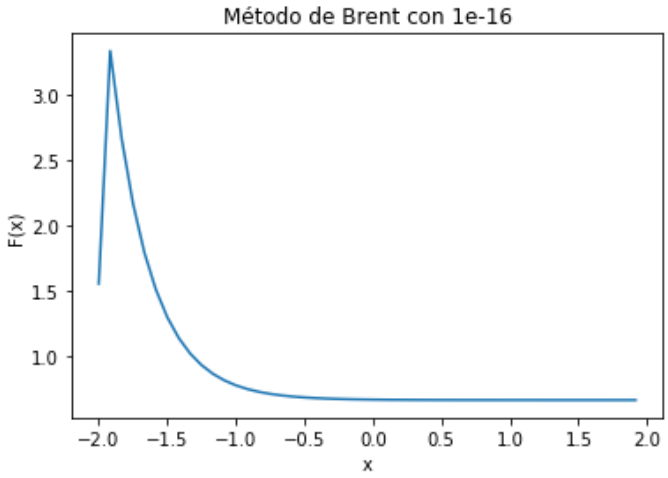

Figure 7.
Resultado con el método de Brent bajo una tolerancia de 10**-16.

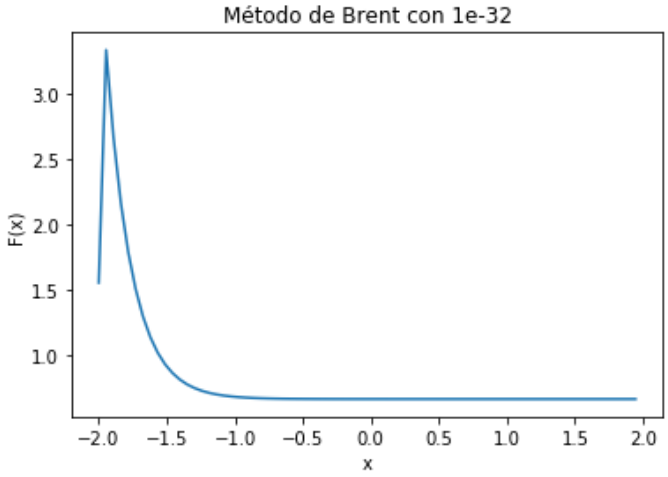

Figure 8.
Resultado con el método de Brent bajo una tolerancia de 10**-32.

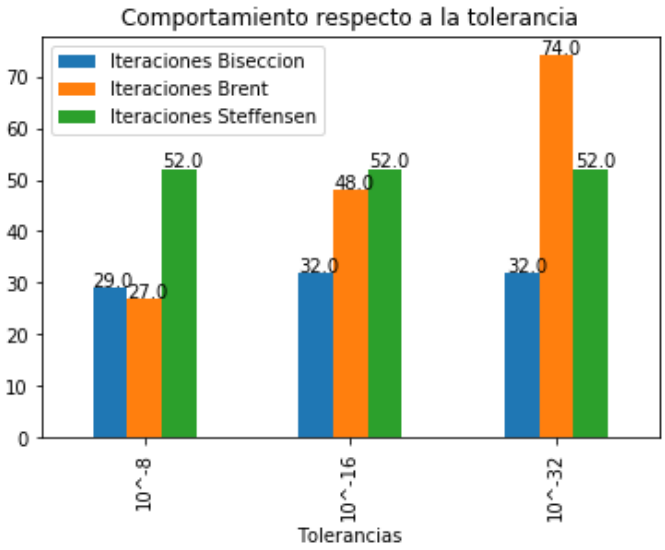

Figure 9.
Comparativa del método de Brent frente a otros algoritmos a partir de sus iteraciones.

2.6. **Procedimiento.** La implementación del algoritmo de Brent recibe como parámetros la función a evaluar, un punto inicial, un punto final, una tolerancia y un máximo de iteraciones. Después de esto se busca cumplir con una de las siguientes condiciones:
1. Verificar que la multiplicación resultante de evaluar la función en un punto a y un punto b sea menor que 0.
1.1. De ser así se buscan unos nuevos puntos a y b con la función secante.
1.2. De no ser así se comprueba si se puede aplicar nuevamente al función de secante.
1.2.1. De no ser así se aplica la función de bisección para unos puntos c y b.
2. Se repite el paso 1 sin sus caminos alternos y si se cumple nuevamente la condición, se guarda como resultado.
3. Si al evaluar la función en un punto b, el valor resultante es menor a la tolerancia establecida se termina el algoritmo de Brent.

2.7. **Conclusión.** Para esta sección se puede concluir que el método de Brent al implementar varios métodos logra resultados más rápidos, pero no a un nivel considerable como para dejar de lado métodos previos como bisección. Por otro lado, cuando la tolerancia no es alta, en casos donde se desea una mayor precisión es recomendado recurrir a otro método.

3. Óptima aproximación polinómica

La implementación de punto flotante de una función en un intervalo a menudo se reduce a una aproximación polinómica, siendo el polinomio tıpicamente proporcionado por el algoritmo Remez. Sin embargo, la evaluación de coma flotante de un polinomio de Remez a veces conduce a cancelaciones catastróficas. El algoritmo Remez es una metodología para localizar la aproximación racional minimax a una función. La cancelaciones que también hay que considerar en el caso de del método de Horner, suceden cuando algunos de los coeficientes polinómicos son de magnitud muy pequeña con respecto a otros. En este caso, es mejor forzar estos coeficientes a cero, lo que también reduce el recuento de operaciones. Esta técnica, usada clásicamente

Explicación del algoritmo y su procedimiento : El algoritmo de Remez es un algoritmo que se usa para encontrar las aproximaciones las funciones.El algoritmo funciona de la siguiente manera; primero se calculan los números de Chebysev para poder evaluarlos en la siguiente función:

$$e * *(sen(x) - cos(x^2) \tag{3.1}$$

Para poder realizar el calculo de los números de Chebysev se utilizó la siguiente fórmula:

$$X = \frac{1}{2}(a+b) + \frac{1}{2}(b-a) * cos(\frac{2i+1}{2n})\pi \tag{3.2}$$

En la anterior función a y b hacen referencia a los intervalos en donde se vana calcular los números Chebysev, i hace referencia a la iteración en la que se encuentre y n hace referencia al número máximo de números que se desean encontrar. Después de tener estos número se toman n + 2 puntos para interpolar, en nuestro caso n = 4. Los números que se generaron a partir de estos fueron los siguientes:
-1.9318516525781366e-08
-1.9318516525781366e-08
-1.4142135623730952e-08
-5.176380902050415e-09
- -5.176380902050417e-09
- -1.414213562373095e-08

Seguidamente estos datos se ingresan a una matriz en donde se expresa un sistema de ecuaciones lineales con cada punto, con el fin de ser operados junto con los resultados de los valores evaluados en la función, los cuales se encuentran en un vector; la operación de estos da como resultado un vector que contiene los coeficientes del polinomio generado por la función minimax. El polinomio generado es el siguiente:

$$1.83939718e-01 + 1.83939723e-01x + 3.67879441e-01x^2 + 2.58821068e-09x^3 \tag{3.3}$$

3.1. **Precisión.** Para este algoritmo como tal se mantuvo la precisión con la que trabaja el programa, la cual es de 16 cifras significativas; no se pensó en adicionar más debido a que el resultado de los números eran demasiado similares, sus cifras eran constantes hasta la la novena cifra. Por lo cual se pudo verificar la precisión con las 7 últimas cifras.

3.2. **Pruebas.** Al probar la función se puede observar que es suave y difícil de sacar de su trayectoria de comportamiento, es decir, que sin importar el valor en el que se evalué siempre va a tender a comportarse de la misma manera. La implementación soporta una alta cantidad de números a procesar, para un caso en particular se utilizaron los siguientes datos de prueba:
Intervalo = [-2^8 , 2^8]
n = 4
Tolerancia = 2^90
número de datos = 25

Con estos datos se obtuvieron los siguientes resultados al evaluar el algoritmo de Taylor y el de Remez:
Taylor = 0,3678794...
Remez = 0,3678794...

3.3. **Comparaciones.** Para verificar la eficiencia del algoritmo de Remez se evalúa la misma función con los mismos parámetros de entrada en el algoritmo de Taylor; los resultados de esta comparación se presenta en la siguiente sección.

3.4. **Gráficas.** A continuación se presentan una series de gráficas en donde se muestran la relación que existe entre los dos algoritmos en los que se evalúan, también se muestra que tan grande es el error que se presenta para estos algoritmos.

En la Figura 10 se evidencia el comportamiento uniforme de la función $e^{\hat{}}(\text{sen}(x) - \cos(x\hat{2})$, y se puede observar que su error relativo es muy mínimo, por lo cual su comportamiento se ve uniforme.

En la Figura 11 se evidencia un comportamiento del polinomio uniforme para la función $e^{\hat{}}(\text{sen}(x) - \cos(x\hat{2})$, en donde a medida que se avanza en el intervalo se puede ver que el error absoluto también va creciendo, esto se debe a que los resultados del polinomio al presentar un mínimo error este se va propagando, generando una mala adaptación a largo plazo.

En la Figura 12 y 13 se evidencia como el comportamiento tan similar que se maneja entre Taylor y Remes, observando que la diferencia entre estos tiende a cero; por lo cual en la Figura 12 no se aprecia con exactitud la diferencia entre estas, la cual si se puede divisar con más claridad en la Figura 13.

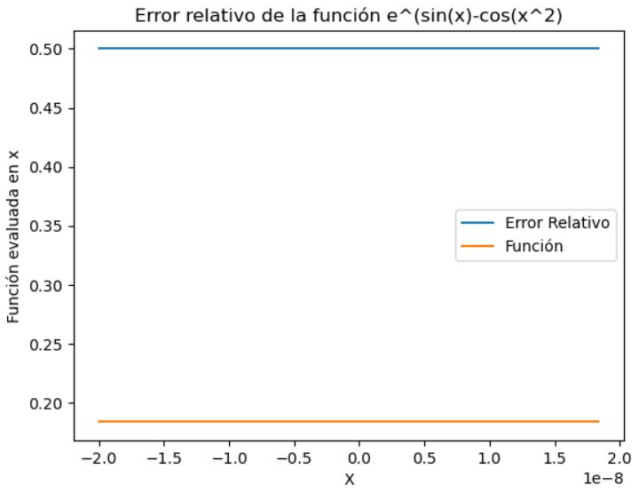

Figure 10.
Error relativo del método de Remez

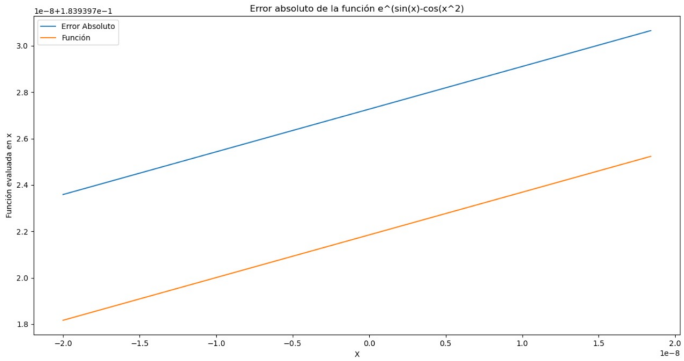

Figure 11.
Error absoluto del método de Remez

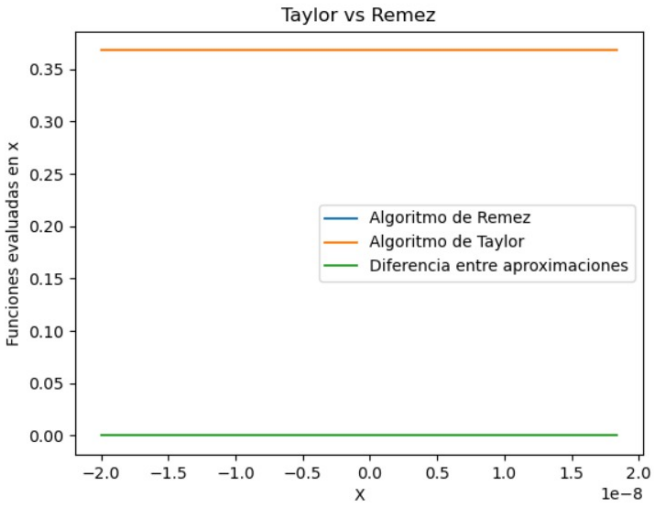

Figure 12.
Comparativa del método de Taylor frente a Remez.

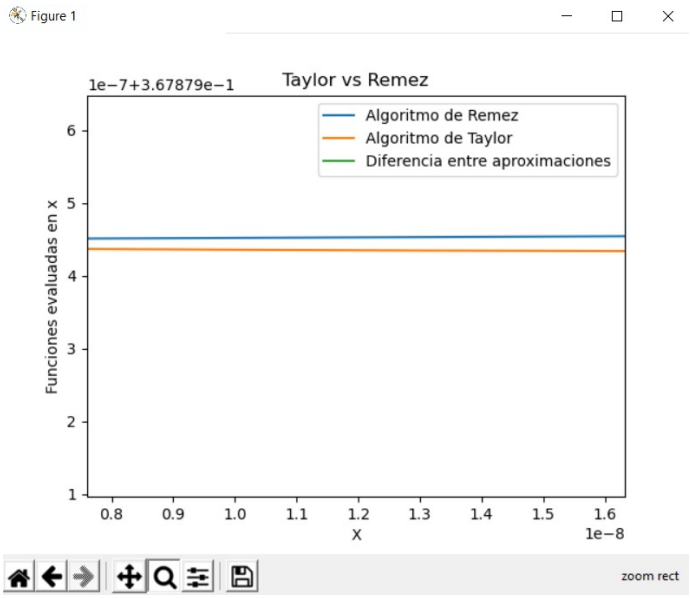

Figure 13.

3.5. **Conclusión.** Al ejecutar este algoritmo podemos concluir que el comportamiento de Taylor y Remez son muy similares, ya que e la Figura 12 se evidencia que la diferencia entre sus resultados es casi cero. También podemos concluir que la función en el intervalo que

se pide evaluar es continua y no requiere de alguna adaptación o excepción en particular para que esta funcione. En la Figura 11 se puede ver que la adaptación del polinomio con respecto a la función original empeora a medida que se mueve en el intervalo. Por último podemos concluir que la función presenta un comportamiento inelástico, esto se debe a que en el intervalo en el que se evalúa las soluciones tienden a ser muy similar.

4. Referencias

[1]W. Mekwi, "Iterative Methods for Roots of Polynomials", Core.ac.uk, 2001. [En línea]. Disponible en: https://core.ac.uk/download/pdf/96491.pdf. [Accessed: 13- Sep- 2020].

[2]W. Mora Flores, Introducción a los métodos numéricos. Cartago: Instituto Tecnológico de Costa Rica, 2010. [En línea]. Disponible en: https://tecdigital.tec.ac.cr/revistamatematica/Libros/WMora_Metodos Numericos/WMora-ITCR-MetodosNumericos.pdf