Reto1

Santiago Caroprese Hidalgo

Daniel Hernandez García

Juan Carlos Súarez jáimes

13 de Septembre del 2020

1. Evaluación de las raíces de un polinomio

El método de Newton es un algoritmo que permite calcular las raíces de funciones no lineales, cuya convergencia es cuadrática en la mayoría de los casos. En la implementación de este método, para cada iteración es necesario evaluar la función y su primera derivada en la aproximación actual de la raíz.

El método de Laguerre es otro algoritmo que permite calcular raíces, pero funciona únicamente con funciones polinómicas. La convergencia de este algoritmo es cúbica en la mayoría de los casos. En cada una de las iteraciones de este método es necesario evaluar la función y sus dos primeras derivadas.

El método de Horner es, por otro lado, un algoritmo que permite de evaluar de manera eficiente un polinomio en un punto dado. Se identificó que es posible modificar el algoritmo de Horner para que, además de evaluar el polinomio, evalúe también las derivadas de este. Esto es posible gracias a que la derivada de un polinomio es otro polinomio.

Para la implementación de esta modificación del método de Horner, se calculan simultáneamente los resultados del polinomio original, la primera derivada y la segunda derivada. Con respecto a la versión del algoritmo que solo evalúa el polinomio, para la primera derivada se tiene en cuenta que los coeficientes deben ser multiplicados por el grado de su término y que se omite la última multiplicación por x. En cambio, para la segunda derivada, los coeficientes se multiplican por el producto (grado)*(grado-1) y se omiten las dos últimas multiplicaciones por x.

Teniendo en cuenta lo anterior, esta versión modificada del método de Horner puede ser utilizada para evaluar de manera eficiente el polinomio y sus dos primeras derivadas en los métodos de Newton y Laguerre. A esta versión del método de Newton la llamaremos Newton-Horner y presenta la restricción que solo permite calcular raíces de funciones polinómicas.

1.1. Raíces con parte imaginaria

En las implementaciones de los métodos de Newton-Horner y Laguerre, para tener en cuenta las raíces con parte imaginaria, se utilizó la función as.complex() para convertir a número complejo el punto inicial recibido por los algoritmos, para que el algoritmo maneje los resultados parciales de las iteraciones como números complejos, de tal manera que no se presenten inconvenientes si en algún punto de la ejecución del algoritmo se obtiene un resultado complejo. Se identificó que para el algoritmo de Newton-Horner es necesario que el punto inicial sea también un número complejo.

En las pruebas realizadas, se encontró que, de esta manera, el algoritmo encuentra las raíces de las funciones bien sean reales o complejas. Las respuestas son impresas en notación a + bi, en donde b solo toma valores diferentes de cero cuando la raíz es compleja. Sin embargo, se encontró que trabajar con valores complejos impedía utilizar la librería RMPFR para extender la precisión, por lo que finalmente se decidió incluir dos versiones de cada función: una que permita permite obtener raíces complejas, pero con precisión doble, y otra que utiliza la librería RMPFR para extender la precisión, pero que no puede obtener raíces complejas.

1.2. Pruebas

En las pruebas realizadas, para cada caso se obtuvieron resultados con tres tolerancias diferentes: $1e-8$, $1e-16$ y $1e-32$. En primer lugar, se utilizaron los algoritmos de Newton-Horner y Laguerre para calcular las raíces del siguiente polinomio.

$$x^4 - 9x^2 - 5x^3 + 155x - 250 \tag{1}$$

Para este polinomio, el valor inicial utilizado fue 0. A continuación, se muestran los resultados obtenidos con cada algoritmo.

Tabla 1: Resultados de Newton-Horner para polinomio 1.

Tolerancia	Raíz	Número de iteraciones
1e-8	1.99999999999999999585575116327175131945	5
1e-16	1.99999999999999999999999999999999999976	6
1e-32	2.00000000000000000000000000000000000000	7

Tabla 2: Resultados de Laguerre para polinomio 1.

Tolerancia	Raíz	Número de iteraciones
1e-8	2.00000000000000000000000000000000000000	4
1e-16	2.00000000000000000000000000000000000000	4
1e-32	2.00000000000000000000000000000000000000	4

A continuación, se incluyen las gráficas que ilustran el comportamiento de la raíz y del error local conforme avanzan las iteraciones. Resulta evidente que,

si bien ambos algoritmos convergen en muy pocas iteraciones, el algoritmo de Laguerre presenta una convergencia más rápida.

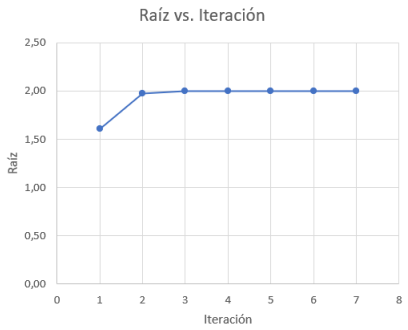

Figura 1: Raíz vs. iteración con Newton-Horner.

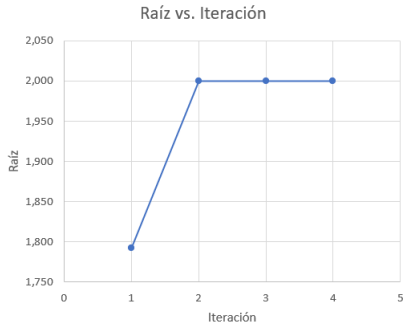

Figura 2: Raíz vs. iteración con Laguerre.

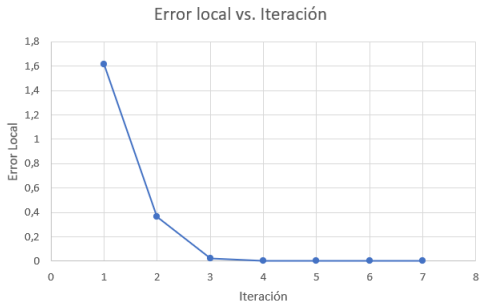

Figura 3: Error vs. iteración con Newton-Horner.

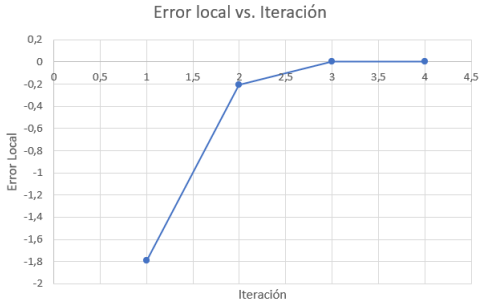

Figura 4: Error vs. iteración con Laguerre.

Adicionalmente, con el objetivo de comprobar que estos algoritmos pueden hallar también las raíces complejas de polinomios, se realizaron pruebas con el siguiente polinomio.

$$1 + 2x + 3x^2 + 4x^3 + 5x^4 \tag{2}$$

Para este polinomio, el valor inicial utilizado fue 5+1i. A continuación, se muestran los resultados obtenidos con cada algoritmo. Llama la atención que no es posible reducir el error local por debajo de 1e-32 con ninguno de los métodos.

Tabla 3: Resultados de Newton-Horner para polinomio 2.

Tolerancia	Raíz	Número de iteraciones
1e-8	-0.5378322749029899+0.358284686345127 95i	15
1e-16	-0.5378322749029900 2+0.358284686345128i	17
1e-32	-0.5378322749029899+0.358284686345127 95i	Max iteraciones

Tabla 4: Resultados de Laguerre para polinomio 2.

Tolerancia	Raíz	Número de iteraciones
1e-8	-0.5378322749029899-0.358284686345128i	6
1e-16	-0.5378322749029899-0.358284686345128i	7
1e-32	-0.5378322749029899-0.358284686345128i	Max iteraciones

1.3. Orden de convergencia

A continuación, se muestran las gráficas que ilustran la relación entre el error en la iteración i y el error de la iteración i+1. Se pude observar que en Newton-Horner esta relación se puede representar a través de una función cuadrática, por lo que se puede concluir que el orden de convergencia de este método es cuadrático. Por otro lado, se puede observar que en Laguerre esta relación se

puede representar a través de una función cúbica, por lo que se puede concluir que para este método la convergencia es cúbica.

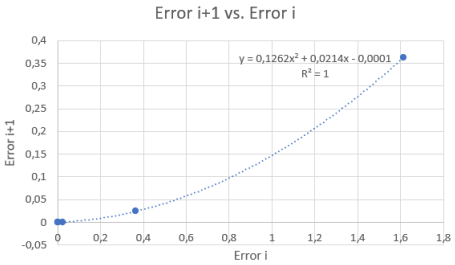

Figura 5: Error i+1 vs. Error i con Newton-Horner.

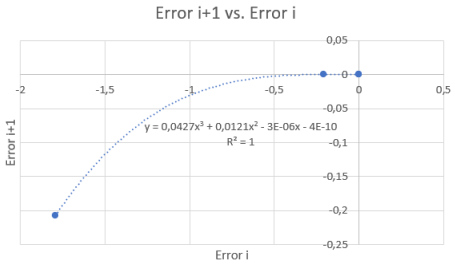

Figura 6: Error i+1 vs. Error i con Laguerre.

2. Algoritmo de Brent

El metodo de Brent, tambien conocido como metodo de van Wijngaarden-Dekker-Brent, es un algoritmo para encontrar raíces de funciones, que mezcla el método de bisección, secante y la interpolación cuadrática inversa. Este método esta garantizado a converger si se usan las condiciones iniciales. Además siempre converge linealmente [Numerical recipes].

La condición para usar este algoritmo, es definir un intervalo [a,b], en donde f(a)f(b)¡0. Además f no necesita ser continuo en [a,b] [Recipes].

El método comienza con una tolerancia tol y los puntos a y b. De ser necesario, a y b se intercambian para que se cumpla —f(b)—¡=—f(a)—. Por lo cual b sería la mejor aproximación a la raiz. Luego se inicia un tercer punto, c = a.

En cada iteración, el método garantiza a, b y c de forma que b sea diferente de c y

- f(b)f(c) ¡0, para que la solución esté entre b y c, si f es continua
- —f(b)—¡=—f(c)—, para que b sea la solución aproximada actual

- a es distinto de b y c, o a=c y es el valor anterior de b

Cada iteración es de la siguiente manera [wpi]:

1. Sí —b-c—¡= tol, entonces el método retorna b como la solución aproximada. 2. Sí no, entonces se define s como punto de prueba, y se siguen los siguientes pasos:

- Sí a=c, entonces s es determinada con la interpolación lineal (secante).
- Sí no, a, b, y c son distintos y s es determinada con la interpolación cuadrática inversa

3. Sí se cumplen ciertas condiciones, s es determinada con bisección con a y b. Las condiciones son (Verificar es un valor booleano):

- (condición 1) s no está entre (3a+b)/4 y b
- (condición 2) (Verificar es verdadero, y $|s-b| >= |b-c|/2$)
- (condición 3) (Verificar es falso, y $|s-b| >= |c-d|/2$)
- (condición 4) (Verificar es verdadero, y $|b-c| < |tol|$)
- (condición 5) (Verificar es falso, y $|c-d| < |tol|$)

4. Al final del ciclo, a,b,c y s son usadas para determinar los nuevos valores de a,b, y c.

2.1. Pruebas

Para este problema, se realizo una implementación del algoritmo de Brent, para evaluar la ecuación 1. Pero para robar su funcionamiento en otros casos, también se realizaron pruebas con la ecuación 2. Hay que mencionar que el método converge en un número fijo de iteraciones por tolerancia; Con tolerancia = $1x10^{-8}$ converge en 28 iteraciones, $1x10^{-16}$ converge en 55 iteraciones, y con $1x10^{-32}$ converge en 108 iteraciones.

En la tabla 1, se muestran los resultados de las primeras 20 iteraciones del método, para la ecuación 1

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

$$f(x) = xsin(x) - 1 \tag{4}$$

Tabla 5: Resultados brent, ecuación 1

Iteraciones	Resultados
1	1
2	0,5
3	0,75
4	0,625000000000000
5	0,687500000000000
6	0,656250000000000
7	0,671875000000000
8	0,664062500000000
9	0,667968750000000
10	0,666015625000000
11	0,666992187500000
12	0,666503906250000
13	0,666748046875000
14	0,666625976562500
15	0,666687011718750
16	0,666656494140625
17	0,666671752929687
18	0,666664123535156
19	0,666664123535156

En la tabla 1, se muestran los resultados de las primeras 20 iteraciones del método, para la ecuación x+1

Tabla 6: Resultados Brent, ecuación 2

Iteraciones	Resultados
1	1
2	0,500000000000000
3	0,500000000000000
4	0,500000000000000
5	0,500000000000000
6	0,500000000000000
7	0,515625000000000
8	0,515625000000000
9	0,515625000000000
10	0,515625000000000
11	0,514648437500000
12	0,515136718750000
13	0,514892578125000
14	0,514892578125000
15	0,514953613281250
16	0,514923095703125
17	0,514938354492187
18	0,514930725097656
19	0,514934539794921
20	0,514932632446289

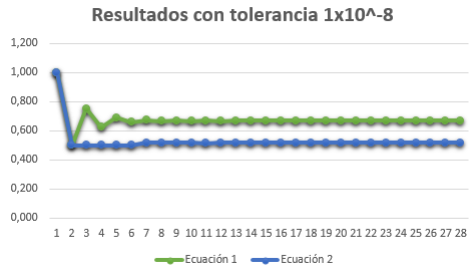

Figura 7: Resultados Brent, tolerancia = 1e-8.

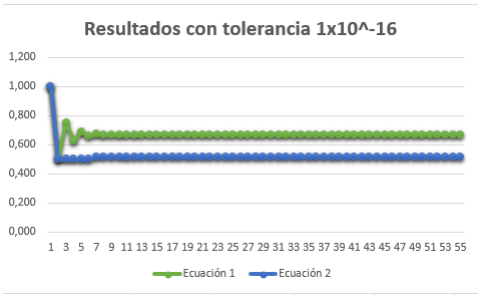

Figura 8: Resultados Brent, tolerancia = 1e-16.

2.2. Comparaciones

Figura 9: Brent y bisección, ecuación 1.

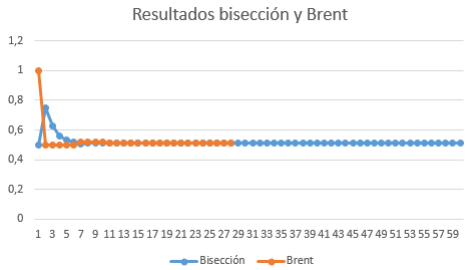

Figura 10: Brent y bisección, ecuación 2.

2.3. Orden de convergencia

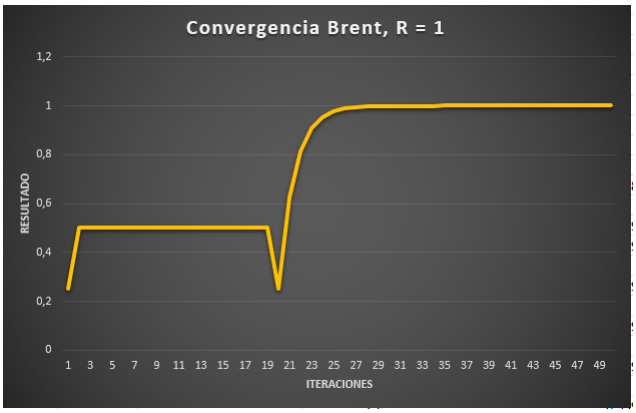

Figura 11: Convergencia Brent, ecuación 1.

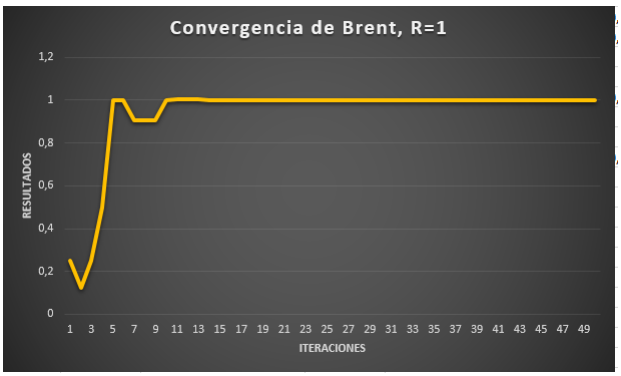

Figura 12: Convergencia Brent, ecuación 2.

2.4. Error relativo

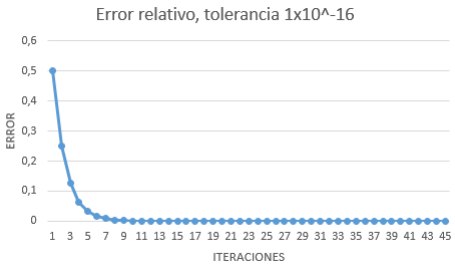

Figura 13: Error relativo Brent, ecuación 1.

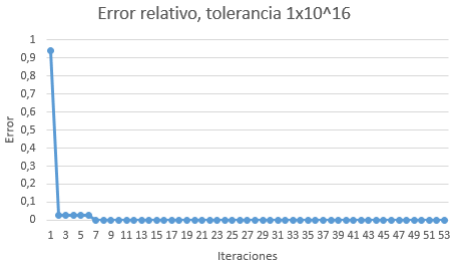

Figura 14: Error relativo Brent, ecuación 2.

2.5. Conclusiones Brent

3. Optima aproximación Polinómica

Lo primero que se realizó para poder resolver el ejercicio fue investigar acerca del método de Remez. Este método se usa como una aproximación minmax, que busca un polinomio de grado n que aproxime una función dada, en un intervalo dado, tal que el error absoluto máximo sea minimizado [D1].

Remez es un algoritmo que inicia con un conjunto arbitrario de n+2 puntos en un intervalo [a,b], una función f y el grado n del polinomio deseado. Este conjunto inicial X contiene lo valores x_i que pueden ser los puntos de control Chebyshev [D2]. Cada iteración se compone de dos pasos principales. En el primer paso se calcula los coeficientes, tal que la función de error obtenga magnitudes iguales, alternando el signo en los n+2 puntos dados [D1].

$$Pn(x_i) + (-1)^i * E^i = f(x_i) \tag{5}$$

$$c_0 + c_1 * x + c_2 * x^2 + ... + c_n * x_i^n + (-1) * E^i = f(x_i) \tag{6}$$

$$i = 0, 1, 2, ..., n + 1 \tag{7}$$

Donde E representa el máximo valor absoluto del error de aproximación. Con esto lo que se tiene es un sistemas de ecuaciones de la siguiente manera:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 & ... & x_0^n & +1 \\ 1 & x_1 & x_1^2 & x_1^3 & ... & x_1^n & -1 \\ 1 & x_2 & x_2^2 & x_2^3 & ... & x_2^n & +1 \\ ... & ... & ... & ... & ... & ... & ... \\ 1 & x_i & x_i^2 & x_i^3 & ... & x_i^n & (-1)^i \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ ... \\ c_i \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ ... \\ f(x_i) \end{pmatrix}$$

Al resolver dicho sistemas de ecuaciones, el resultado es un vector con n+2 coeficientes (c_0,c_1,c_2,...,c_i,), el cual representa el polinomio de grado n $P_n(x)$. Después de este primer paso, se computan los coeficientes de tal manera que la función de error en los n+2 valores dados sea de igual magnitud pero alternando en signo. Para esta implementación se tiene que el vector de errores M contiene la diferencia de $P_n(x)$ menos f(x), para todos los valores x_i.

$$M_i = P_n(x_i) - f(x_i) \tag{8}$$

Una vez calculado el vector de errores, se revisa que los signos sean alternantes. En seguida, se verifica la condición de parada, la cual consiste en que cada error sea menor a la tolerancia deseada.

$$M_i <= tol \tag{9}$$

Si esta condición no se satisface, se sigue con la segunda parte del algoritmo, la cual se encarga de encontrar los extremos de la nueva aproximación [D2]. Para empezar, dado que el vector de errores cambia de signo en cada posición, debe existir n+1 raíces: una raíz en cada intervalo de la manera [c_0,c_1], [c_1,c_2], ..., [c_n,c_{n+1}]. Usando algún método numérico se calculan las n+1 raíces z_0, z_1, z_2, ..., z_{n+1} [D1]. Una vez se tienen estas raíces, se divide el intervalo [a,b] de la manera [a,z_0], [z_0,z_1], ..., [z_{n+1},b]. En cada uno de estos intervalos se calcula el punto en el cual el error obtiene su máximo o mínimo valor, lo que resulta en n+2 extremos (x_0^*,x_1^*, x_2^*, ..., x_{n+1}^*). En este punto es donde se realiza la sustitución de los valores. Para ello se tienen dos opciones: cambio en varios puntos, o cambio en un solo punto[D2]. En el caso de cambio en un solo punto, se reemplaza el valor x_k por el valor z_k donde [D1]:

$$k = max|f(x_i^*) - P_n(x - i^*)| \tag{10}$$

Para el caso de cambio en varios puntos, se intercambian todos los x_i del conjunto X por los valores x_i^* [D2]. Para esta implementación se utilizó el cambio en varios puntos. Una vez realizado estos pasos se reitera hasta que la condición de parada se cumpla.

3.1. Observaciones y Resultados

Lastimosamente, la implementación del algoritmo no demuestra un comportamiento aceptable, ya que a partir de la segunda iteración el vector de los

errores M deja de presentar el comportamiento de signos alternantes. Esto hace que realizar el cálculo de las raíces resulte imposible, debido a que ya no existe el cambio de signo que garantiza la existencia de la raíz. Además, al haber tantas operaciones debido al sistema de ecuaciones, junto a la búsqueda de las raíces y las evaluaciones de f en cada iteración, es evidente que la propagación del error posee un factor alto. Al analizar las secciones del código, se especula que el error funcional se debe a el cálculo de los valores $x*$, lo que hacia que el calculo del sistema de ecuaciones fallara a parti

4. Conclusiones

En cuanto a la evaluación de las raíces de un polinomio, se puede concluir que el método de Horner modificado permite evaluar de manera eficiente las derivadas de un polinomio, por lo que resulta especialmente útil para calcular las raíces de polinomios a través de métodos como el de Newton y el de Laguerre, dado que utilizan las derivadas del polinomio. Entre estos dos métodos, se pudo observar que el método de Laguerre presenta un mayor orden de convergencia, lo que produce un mayor eficiencia. Esto se ve reflejado en un menor número de iteraciones con respecto al método de Newton.

En la teoría, Remez aparenta ser un algoritmo extremadamente útil, dado que puede simplificar expresiones matemáticas de una manera impresionante. A pesar de haber tenido problemas en la implementación

Referencias

[D1] A. Tawfik, Minimax Approximation and Remez Algorithm, 07/24/2005, https://www.math.unipd.it/ alvise/CS_2008/APPROSSIMAZIONE_2009/MFILES/Remez.pdf
[D2] John Maddock, Paul A. Bristow, Hubert Holin, Xiaogang Zhang, Bruno Lalande, Johan R; The Remez Method; 2008; https://www.boost.org/doc/libs/1_39_0/libs/math/doc/sf_and_dist/
[D3] Press, W. Numerical recipes; 2007https://e-maxx.ru/bookz/files/numerical_recipes.pdf