Reto número 1

Camilo Andrés Buitrago Ladino, David Santiago Meneses Cifuentes
Paula Juliana Rojas Naranjo, Juan Carlos Suárez Motta
Pontificia Universidad Javeriana
Autor correspondiente: Camilo Andrés Buitrago Ladino; bu.camilo@javeriana.edu.co
Autor correspondiente: David Santiago Meneses Cifuentes; @davidsmenenescjaveriana.edu.co
Autor correspondiente: Juan Carlos Suarez Motta; @suarezjuan@javeriana.edu.co
Autor correspondiente: Paula Juliana Rojas Naranjo; @rojasnpaula@javeriana.edu.co

14/03/2021

Para encontrar los repositorios de GitHub respectivos:
Camilo Andrés Buitrago Ladino: https://github.com/CamiloB1999/AnalisisNumerico.
David Santiago Meneses Cifuentes: https://github.com/santiagomenesses/Analisis-numerico-David-Meneses.
Juan Carlos Suarez Motta: https://github.com/2qualcuno7/JuanCarlosSuarezMotta.
Paula Juliana Rojas Naranjo: https://github.com/ayurojasn/AnalisisNumerico-JulianaRojas.

1. Punto 1

El método de Brent es en sí un algoritmo de búsqueda de raíces, donde se especifica una combinación entre el algoritmo root bracketing, bisection y quadratic interpolation. Por esta razón se le creó un nombre asociado para recopilar los anteriores algoritmos asociados. Método Wijngaarden-Deker-Brent. Este método principalmente hace uso de un polinomio de interpolación de Lagrange de grado 2.

Este garantiza que, mediante su uso, siempre convergerá tal que los valores de su función sean computables dentro de la región donde se encuentre la raíz. Para lograr dicha afirmación, el método ajusta la x del polinomio dado tres puntos x1, x2, x3.

Para la implementación del algoritmo de Brent, se le aplica a este una función f continua creando una limitación de intervalo de a-b para tener presente que:

$$f(a) * f(b) < 0. \tag{1}$$

En el desarrollo del algoritmo, se generando distintas iteraciones hasta encontrar la raíz deseada.

En esta ocasión se realizará la comprobación del método para encontrar sus raíces, con el siguiente polinomio:

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

Se define un intervalo a y b, definido de la siguiente manera:

$$[a, b] = [0, 1] \tag{3}$$

Se define una cantidad máxima de iteraciones por desarrollar, ya que este método afirma una rápida y eficaz convergencia a 0. El número máximo de iteraciones seleccionada fue 50.

Se especifica una tolerancia exacta para la realización del método con el polinomio definido anteriormente. La tolerancia especificada fue 10e-5.

Gracias a los anteriores parámetros definidos, se crea la siguiente función del MetodoBrent en el lenguaje Python.

```
def MetodoBrent(f, x0, x1, maxIter, Tol):
```

Donde f es la función del polinomio. X0 es el número izquierdo e inferior del rango especificado, X1 es el valor derecho y máximo del mismo rango, maxIter son las cantidad de iteraciones máximas por realizar, y por último Tol hace referencia al número de tolerancia seleccionada.

Para hacer la comprobación y aproximación de la raíz del polinomio, primero realizamos la función específica del método Brent en Wolframalpha: FindRoot[$x^3-2x^2+(4x/3)-(8/27)$]*dondeseobtieneelsiguienteresultadode*

Ahora, mediante el método Brent realizado en código en Python se obtiene el siguiente resultado.

```
La raiz del polinomio x³-2x²+4x/3-8/27 es : 0.6666991372785418
El número de iteraciones alcanzadas es : 20
```

Figura 2: Solucion planteada por el equipo

Donde la raíz obtenida fue una aproximación de 0.6666991 comparando dicho resultado con el resultante de Wolframalpha, 0.6666... En la siguiente gráfica se puede observar cómo se comporta visualmente el número de Iteraciones vs Tolerancia (10e-5) desarrollando el método Brent para encontrar la raíz del polinomio

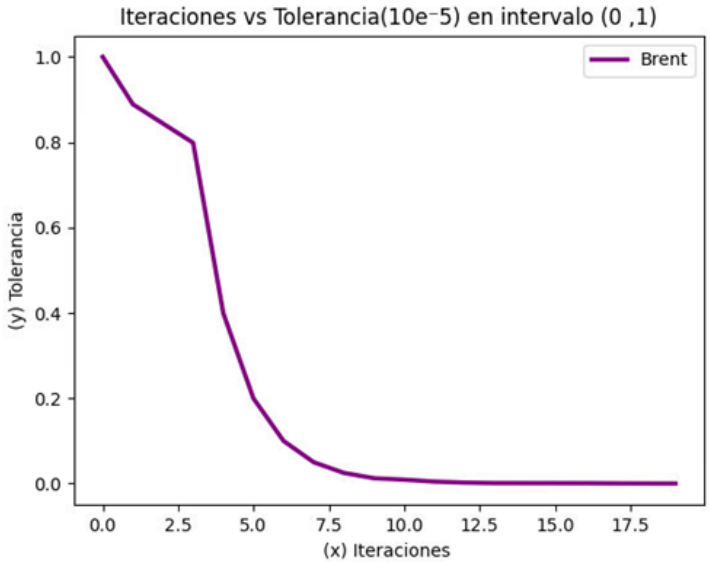

Figura 3: Iteraciones vs Tolerancia

En el siguiente diagrama de flujo se observa y se detalla el orden en el que el método de Brent es implementado.

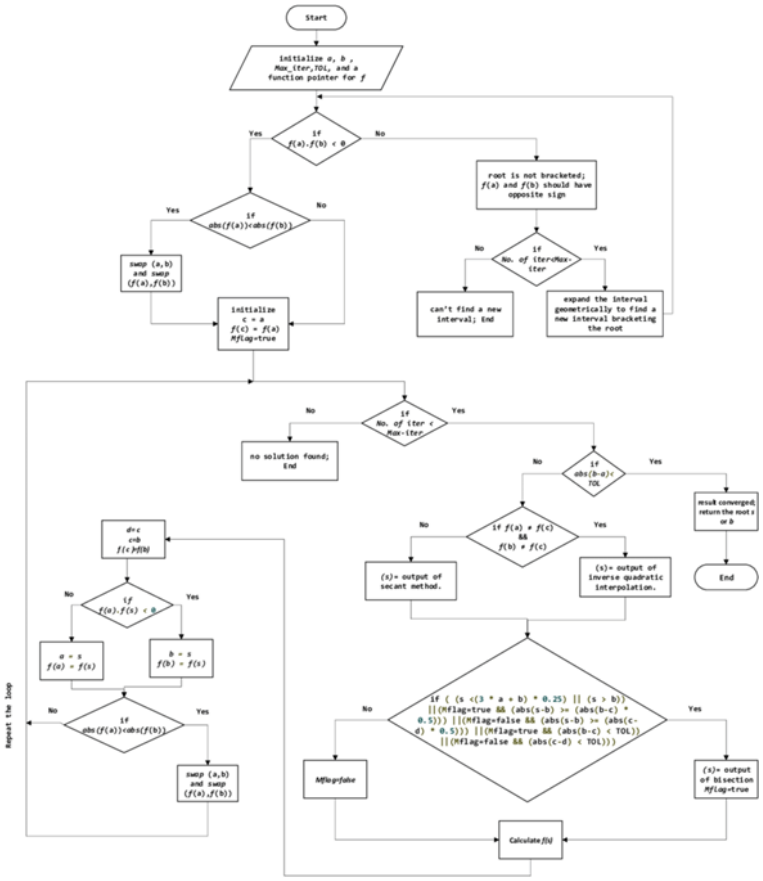

Figura 4: Diagrama de flujo del algoritmo de Brent

[3]

El diagrama se mostrará como adjunto a los archivos en caso de que sea ilegible en alguna forma.

1.1. Comparaciones con taza de error de 10e-4

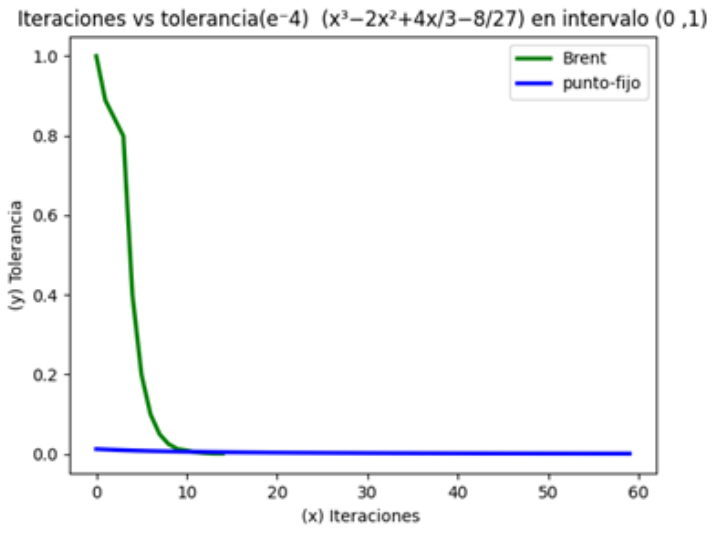

Figura 5: Iteracion,tolerancia, Punto fijo vs Brent

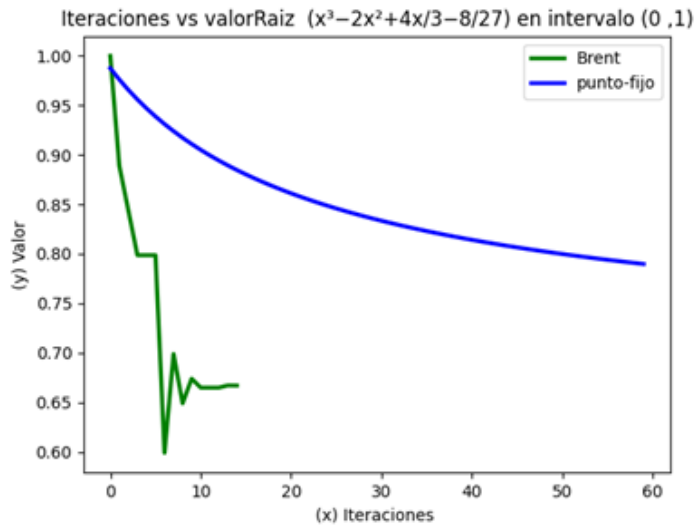

Figura 6: Iteracion,Raices. Punto fijo vs Brent

```
La raiz del polinomio x³−2x²+4x/3−8/27 es : 0.6668635629756641
El número de iteraciones alcanzadas es : 15
La raiz del polinomio x³−2x²+4x/3−8/27 por punto fijo es : 0.7897254126653006
El número de iteraciones alcanzadas por punto fijo es : 60
```

Figura 7: Resultado Codigo Brent

1.2. Comparaciones con taza de error de 10e-5

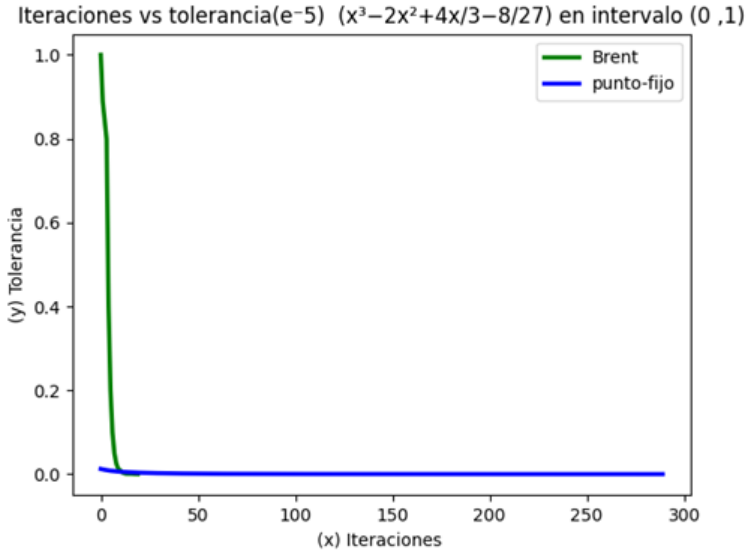

Figura 8: Iteraciones,tolerancia, Punto Fijo vs Brent

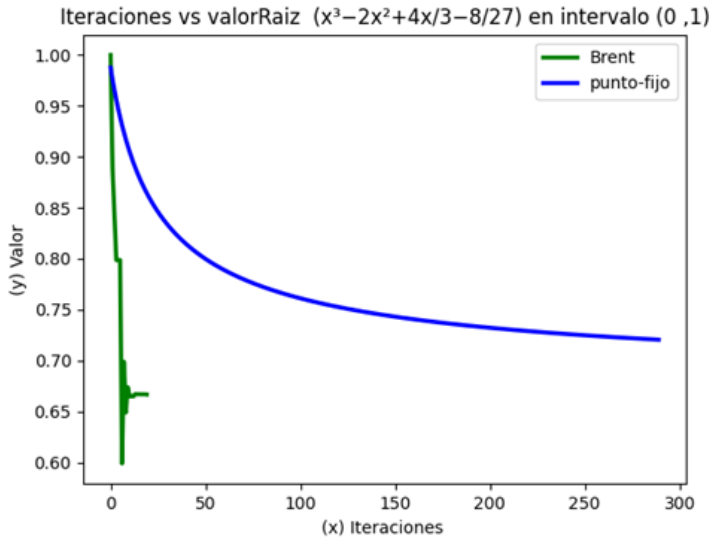

Figura 9: Iteraciones,raices, Punto Fijo vs Brent

```
La raiz del polinomio x³-2x²+4x/3-8/27 es : 0.6666991372785418
El número de iteraciones alcanzadas es : 20
La raiz del polinomio x³-2x²+4x/3-8/27 por punto fijo es : 0.7204050648452216
El número de iteraciones alcanzadas por punto fijo es : 290
```

Figura 10: Resultados Codigo Brent

1.3. Comparaciones con taza de error de 10e-6

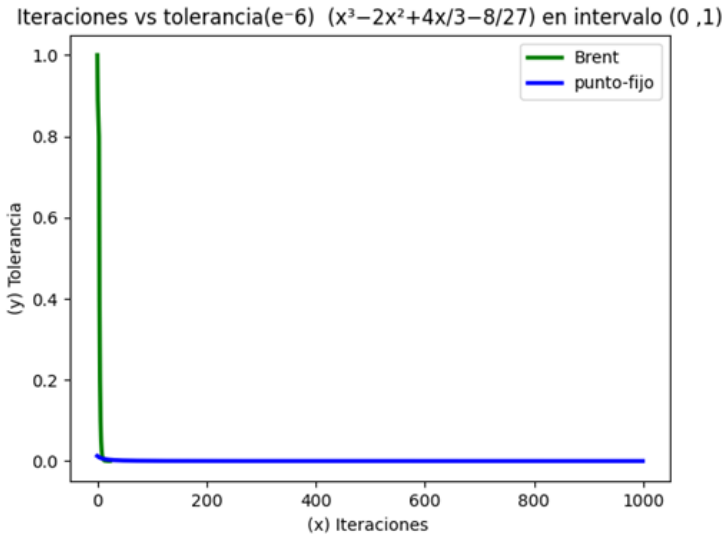

Figura 11: Iteraciones,tolerancia, Punto Fijo vs Brent

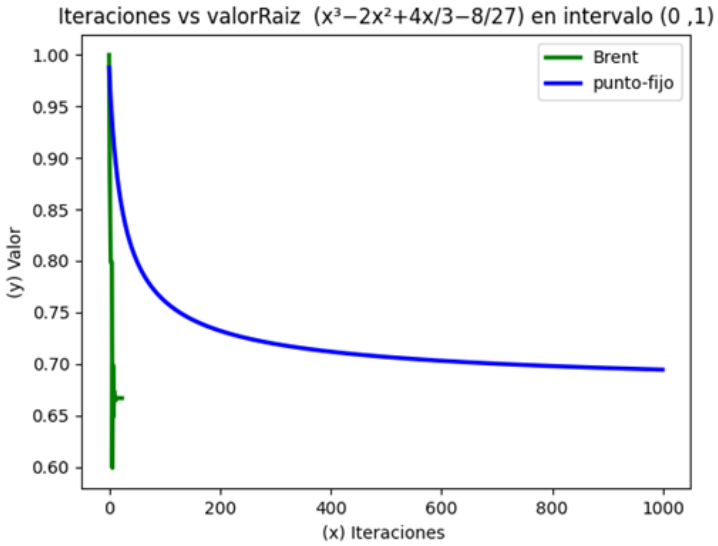

Figura 12: Iteraciones,raices, Punto Fijo vs Brent

```
La raiz del polinomio x³−2x²+4x/3−8/27 es : 0.6666636491332107
El número de iteraciones alcanzadas es : 24
La raiz del polinomio x³−2x²+4x/3−8/27 por punto fijo es : 0.6943514823707911
El número de iteraciones alcanzadas por punto fijo es : 1000
```

Figura 13: Resultados Codigo Brent

1.4. Comparaciones con taza de error de 10e-7

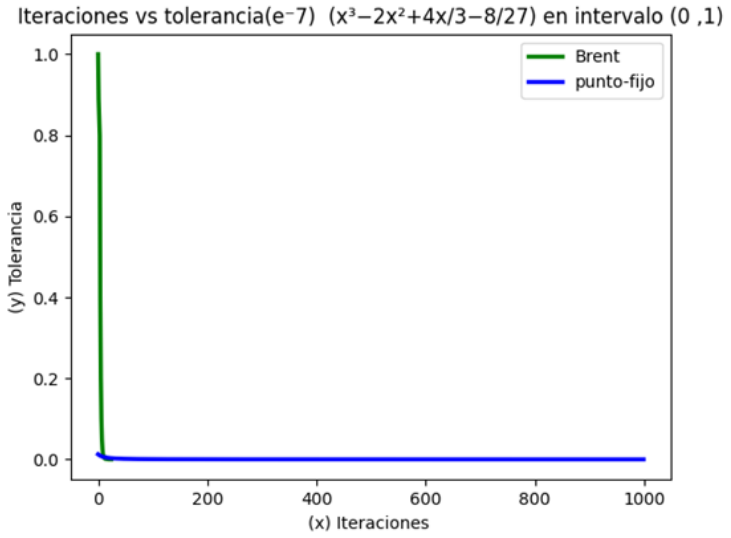

Figura 14: Iteraciones,tolerancia, Punto Fijo vs Brent

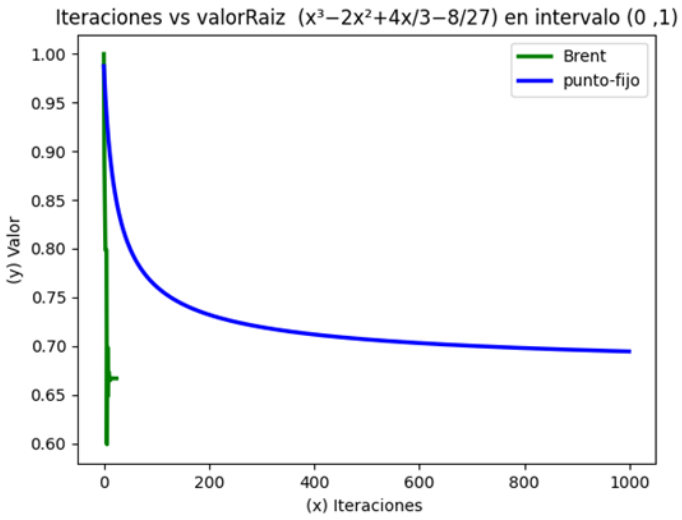

Figura 15: Iteraciones,raices, Punto Fijo vs Brent

```
La raiz del polinomio x³-2x²+4x/3-8/27 es : 0.6666636491332107
El número de iteraciones alcanzadas es : 25
La raiz del polinomio x³-2x²+4x/3-8/27 por punto fijo es : 0.6943514823707911
El número de iteraciones alcanzadas por punto fijo es : 1000
```

Figura 16: Resultados Codigo Brent

1.5. Observaciones de los resultados obtenidos

Como se puede observar en las gráficas de iteraciones vs tolerancias, el algoritmo de punto fijo tiene un comportamiento constante en cuanto a la variación de la tolerancia a lo largo del proceso de ejecución, comparándolo con el algoritmo de Brent. Este último tiene una variación más definida y la misma siempre va decayendo hasta llegar a la iteración final, donde se entrega el resultado aproximado.

A pesar del comportamiento constante del algoritmo de punto fijo, al mismo le toma el triple de iteraciones que el algoritmo de Brent para generar el resultado final y aún así no genera el valor más cercano al valor real, pues es el valor obtenido en el algoritmo de Brent el que se acerca más a dicho valor.

Ahora, en cuanto a las gráficas de iteraciones vs valor, se puede observar que el comportamiento del valor calculado por iteración es muy errático en el algoritmo de Bert, teniendo valores altos y bajos hasta que al final alcanza un equilibrio final. Por su parte el algoritmo de punto fijo tiene un comportamiento mejor definido y el mismo va decrementando de forma continua hasta llegar finalmente al resultado.

Con todo esto se puede concluir que, si bien el algoritmo de punto fijo tiene un comportamiento más estable y definido, el mismo requiere de un valor extremadamente alto de iteraciones en comparación al algoritmo de Bert. Este alto número de iteraciones hace que el algoritmo de punto fijo quede en desventaja, puesto que con funciones que sean relativamente más complejas este algoritmo puede tardar mucho tiempo en ejecutarse y no entregará el valor aproximado más cercano al competir con el algoritmo de Bert.

2. Punto 2

2.1. Introduccion

La solución al problema de la intersección de curvas se puede encontrar reduciendo el problema a determinar una aproximación adecuada de los ceros de una ecuación, Para aproximar esto se puede usar el método de Newton para aproximar las raíces de su diferencia.

Si dos curvas h(x) y g(x) entonces encontrar el punto de intersección para este se basa en igualar las dos curvas, que sería lo mismo que igualar la diferencia de las curvas a 0. Así creando una nueva función f(x) que es donde se van a obtener las raíces aproximadas de la fórmula que serían los puntos de intersección.

Para hallar las intersecciones entre las curvas del presente punto a través de un método de cálculo de raíces, se realizó el siguiente planteamiento matemático:

$$f(x) = x^2 + xy - 10 = 0 \tag{4}$$

$$g(x) = y + 3xy^2 - 57 = 0 \tag{5}$$

$$f(x) - g(x) = 0 \tag{6}$$

Como se puede apreciar, al igualar la resta de las ecuaciones a 0 se llega justamente a la solución del sistema de ecuaciones, lo que reduce el problema a hallar raíces de una ecuación resultante, que se define a continuación:

$$h(x) = (-1 + (1 + 684x))/6x - (10 - x^2)/x \tag{7}$$

Ya habiendo definido la ecuación resultante, se escoge el método de Newton para poder hallar de h(x)

2.2. Condiciones del algoritmo

.

Como es bien sabido, el algoritmo de Newton requiere que la función cuyas raíces han de ser halladas debe ser diferenciable en el intervalo en el que se encuentre la raíz esperada [2], algo que se verificó, llegando a:

$$h'(x) = -((1 + 684x) - 1)/(-6x^2) + (10 + x^2)/x^2 + 57/(x(1 + 684x)) + 2 \tag{8}$$

2.3. Diagrama de flujo y explicaciión

El siguiente diagrama de flujo describe el método de Newton, según lo implementado

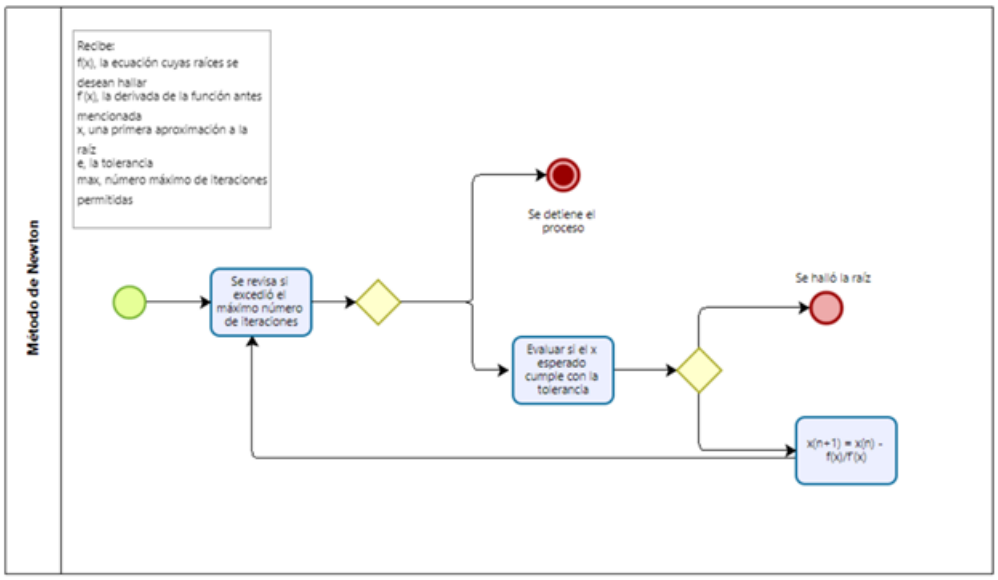

Figura 17: Diagrama de flujo del algoritmo para hallar la inserseccion de curvas

El método de Newton, también conocido como el método de Newton-Rapphson, parte de una ecuación diferenciable en el intervalo en el que se encuentra la raíz, su derivada, una tolerancia dada arbitrariamente y un número máximo de iteraciones, siendo este último imprescindible dado que el método no asegura la convergencia (ANALP). En cada iteración, que se repiten hasta que se alcance la tolerancia deseada, el método calcula de forma directa una nueva aproximación a la raíz, a partir de la ecuación:

$$x2 = x1 - f(x1)/f(x2) \tag{9}$$

[1]

Esto permite que el método converja de forma mucho más rápida que otros métodos, como el de bisección, con el cual se compara más adelante en este texto.

2.4. Niveles de Significancia

Dada las condiciones del enunciado, se emplea, inicialmente, una significancia de $2^{-}16$.

A través de la librería Decimal de Python. Para pruebas posteriores se utilizan tolerancias más exigentes.

2.5. Comparacion

Para poder observa si el algoritmo efectivamente lograba llegar a las raíces buscadas, se utilizaron las raíces de Wolfram Alpha como marco de referencia. Al ingresar las ecuaciones originales, f(x) y g(x), se obtuvo:

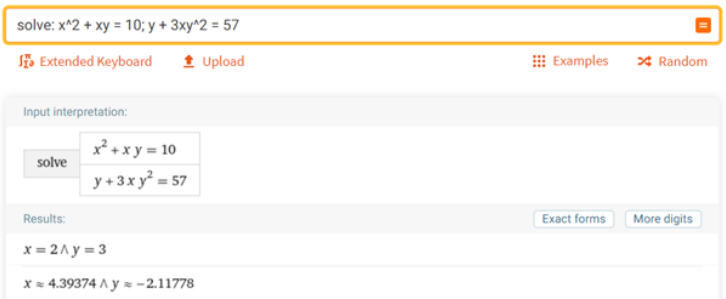

Figura 18: Solución en Wolfram Alhpa

Al ejecutar el código en Python se obtuvo el siguiente mensaje en consola:

```
Se emplearon 7 iteraciones para hallar el resultado
Las curvas se intersectan en el punto 2,3
```

Figura 19: Solución con el codigo desarrollado por el equipo

Como se puede observar, el método logró encontrar con éxito una de las raíces, es decir punto de intersección, y por ende el diseño e implementación fue efectivo. Más adelante en el texto se abordarán los resultados de la ejecución con diferentes aproximaciones iniciales y tolerancias. Sin embargo, se observó cómo no se pudo hallar la segunda raíz, identificando una de las limitaciones del método escogido, al este no asegurar convergencia para todos los casos. Las aproximaciones a la raíz para el ejemplo anterior se muestran en el siguiente gráfico:

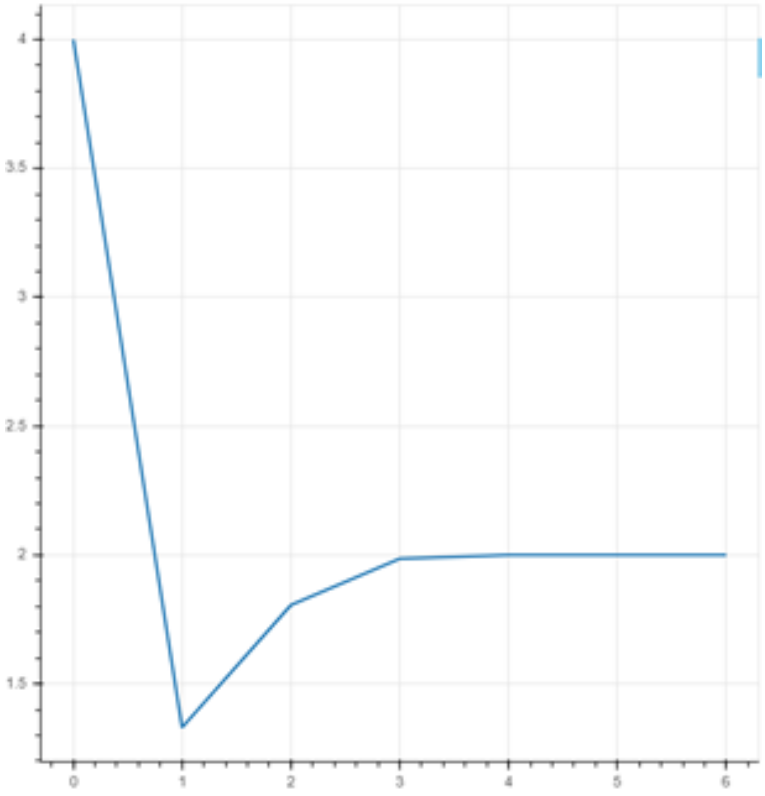

Figura 20: Aproximaciones a la raiz

2.6. Eficiencia del Algoritmo y pruebas con diferentes tolerancias y aproximaciones iniciales

Como se puede observar de la última imagen del punto anterior, el algoritmo logró converger al punto de intersección en solo 7 iteraciones cuando la aproximación inicial fue de 4, muy inferior a las 16 iteraciones mínimas requeridas para converger a través del método de bisección, que se calculó a partir de la siguiente fórmula:

$$log(3 - 21 - log12^{(} - 16))/log2 \tag{10}$$

[4]

Para valores un poco más distantes, como fue la aproximación inicial de 8, se requirió de 11 iteraciones, lo que demuestra que en la eficacia del algoritmo es altamente dependiente de la habilidad del usuario para poder llegar a resolver el sistema. Los resultados de dichas pruebas se muestran en la siguiente tabla:

Cuadro 1: Resultados de iteraciones

Aproximación inicial	Iteraciones
1	6
2	0
3	5
4	7
5	8
6	8
7	9
8	11
9	11

Ahora bien, frente a la utilización de diferentes tolerancias, no se obtuvo un cambio en el número de iteraciones requeridas, pero si se observó que el computador, con procesador Intel I7 de octava generación tomó más tiempo en converger, probablemente a raíz de tener que realizar operaciones de punto flotante más complejas.

2.7. Conclusiones

A partir de lo observado en este punto, se puede concluir que el método de Newton es eficaz para poder calcular los puntos de intersección de dos curvas, siempre y cuando estas logren cumplir con las condiciones de aplicación del método que fueron mencionadas anteriormente. A pesar de que esto es cierto, el método no resultó sencillo de implementar y garantiza poder encontrar todas las raíces deseadas.

3. Punto 3

La librería de scipy nos ayudó a modelar el algoritmo de Brent por medio de la función brentq de scipy. La función Brentq de python utiliza la función de brent para hallar un cero en la función f en el signo de un intervalo cambiante[a,b]. Generalmente esta es considerada la mejor manera de encontrar raíces aquí. Es una versión segura del metodo de secante que usa extrapolación cuadrática. Este método involucra root bracketing, interval bisection, and inverse quadratic interpolation".

```
Wolfram Alpha: 2/3 = 0.666666666667
Solución con Brentq: 0.6666632250069964
```

La librería sympy de python da lugar a muchas funciones que ayudaron a la realización de este punto. Las funcion fue utilizada para evaluar el caso de la solución de la intersección. Para ser más específicos se utilizó Sympy solve que es un solucionador de ecuaciones algebráicas. Se observa que con Wolfram Alpha hay un nivel de significancia mucho menor comparado El solver es capaz de resolver ecuaciones polinomiales, trascendentales, combinaciones de las anteriores, sistemas de ecuaciones lineales y polinomiales, Sistemas que contienen expresiones relacionales. La solución principal del problema se encuentra a continuación

```
Wolfram Alpha: interseccion 1: x = 2, y = 3
         interseccion 2: x aprox 4.39374 , y aprox -2.11778
Solución con Solve: [{x: 2, y: 3}......]
```

Por cuestion de orden en el documento anexamos un bloc de notas con el resultado completo. Se observa que con Wolfram Alpha e incluso con nuestro codigo hay una menor cantidad de soluciones disponibles comparado con Solve de Sympy, esto determina que solve utiliza mayor cantidad de puntos y además muestra estos con una complejidad mayor incluso tratandolos como imaginarios.

Referencias

[1] Gilbert Strang Edwin "Jed" Herman. *Newton's Method*. urlhttps://math.libretexts.org/Bookshelves/Calculus/Book 2021.

[2] Luis Rodriguez Ojeda. *Analisis numerico basico con python*. 2014.

[3] Unkknown. *Brent´s Diagram*. urlhttps://www.researchgate.net/figure/Flowchart-of-Brents-algorithm-used-for-efficient-extraction-of-solar-cells-and-285$_fig2_3$6769008. 2021.

[4] Unkknown. *Minimum number of iteration in Bisection method*. urlhttps://math.stackexchange.com/questions/1057 number-of-iteration-in-bisection-method. 2017.