Reto 1

Juan Esteban Rincon
Camilo Bustos
David Gonzales
Carolina Molina

Marzo 14 2021

1 Algoritmo Brent

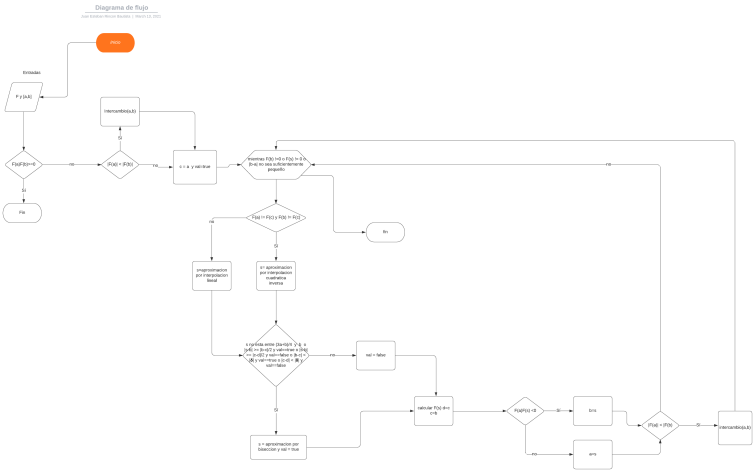
- **Problema:**Aplicar el algoritmo de Brent para encontrar las raıces del polinomio, con un error menor de 2^{-50}.

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

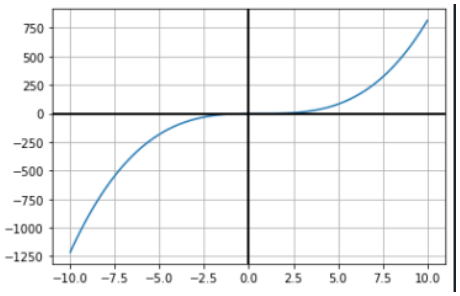
- **Solucion por pasos:**

  1. **Descripción del método**

     Brent es un algoritmo que combina algunos métodos de aproximación de raíces como el método de bisección, el método de la secante y el método de interpolación cuadrática inversa(Müller), consiste básicamente en aplicar el método de la secante o de interpolación cuadrática inversa si es posible y posteriormente hacer uso de bisección en caso de que los métodos anteriormente mencionados tuvieran dificultades para converger[4].



2. **Implementación:**
   Para la implementación del algoritmo de brent a esta función particular decidimos hacer uso del método root_scalar el cual hace parte de la librería optimize de scipy definidas dentro de python, elegido debido a que dentro de sus posibilidades tiene implementado el algoritmo de brent haciendo uso de la interpolación lineal junto con la interpolación cuadrática inversa[1], además este método permite ajustar el nivel de tolerancia y el intervalo inicial, finalmente además del valor de la aproximación este método nos retorna otras métricas como el número de iteraciones realizadas o el estado de convergencia.
   Funcion de prueba: $f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27}$



   – **Prueba 1 [-2,2] tolerancia=2^{-50}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.6666684421187378806195056313 30
Iteraciones:  58
```

– **Prueba 2 [-1,1] tolerancia=2^{-50}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.66666678545543223144420608 1505
Iteraciones:  55
```

– **Prueba 3 [0,1] tolerancia=2^{-50}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.66666867094508086566406745 9875
Iteraciones:  51
```

– **Prueba 4 [0.5,0.7] tolerancia=2^{-50}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.66666826492548925120473768 4020
Iteraciones:  43
```

– **Prueba 5 [0.5,0.7] tolerancia=2^{-60}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.66666826492548925120473768 4020
Iteraciones:  43
```

– **Prueba 6 [0.5,0.7] tolerancia=2^{-70}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.66666826492548925120473768 4020
Iteraciones:  43
```

– **Prueba 7 [0.5,0.7] tolerancia=2^{-8}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.66666826492548925120473768 4020
Iteraciones:  43
```

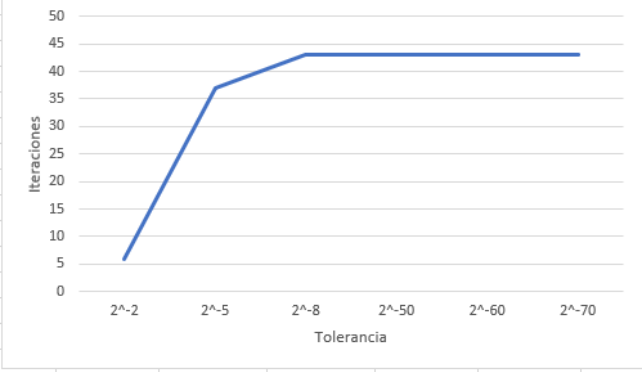
– **Prueba 8 [0.5,0.7] tolerancia=2^{-5}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.66666783573031340726089410 9641
Iteraciones:  37
```

– **Prueba 9 [0.5,0.7] tolerancia=2^{-2}:**

```
prueba 1 metodo de brent con interpolacion cuadratica inversa:
raiz aproximada:  0.66999999999999892974500426 1349
Iteraciones:  6
```

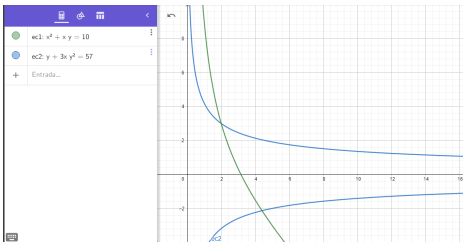
Grafica comparativa de iteraciones contra tolerancias:



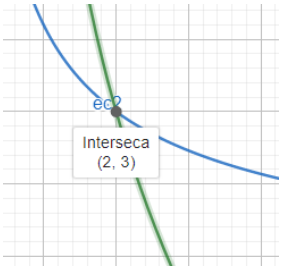
3. **Conclusion:** Son evidentes las ventajas que este algoritmo puede ofrecernos por un lado no tubo dificultades en ninguna de las pruebas para converger al valor aproximado de la raíz de esta función, como si las tubo nuestra anterior implementación del método de Müller por ejemplo, también se evidencia una mejora en su velocidad que a pesar de utilizar un número considerable de iteraciones que también eran evidentemente afectadas por otros factores como la lejanía de los puntos iniciales elegidos o la tolerancia probada, se presenta mejora a diferencia de optar por algunos de estos métodos de forma individual.

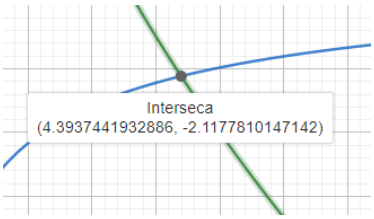
2 Intersección entre curvas

- **problema:** Aplicar la técnica de aproximación a la raíz, que desarrollo en el trabajo en grupo, para encontrar la intersección entre $x^2 + x * y = 10$ y $y + 3x * y^2 = 57$
- **Solucion por pasos:**
  1. **Descripción del método:** Para solucionar este problema aplicamos el método de Müller a las ecuaciones simplificadas resultantes al despejar y en términos de x de la primera ecuación y remplazarla en la segunda y de hacer el mismo procedimiento despejando x de la segunda ecuación en términos de y y remplazarla en la primera ecuación, finalmente al aplicar el anteriormente implementado método de Müller en las 2 ecuaciones resultantes se obtiene que sus respectivas raíces coinciden con los puntos de intersección de las 2 ecuaciones originales.
     - **Grafica sobrepuesta de las dos funciones originales:**



∗ **Puntos de intersección:**





- **1:** $x^2 + xy = 10$
  $y = \frac{10-x^2}{x}$
  $(\frac{10-x^2}{x}) + 3x(\frac{10-x^2}{x}) - 57 = 0$
  $10 - x^2 + 3(-x^2 + 10) - 57x = 0$
- **2:** $y + 3xy^2 = 57$
  $x = \frac{57-y}{3y^2}$
  $(\frac{57-y}{3y^2}) + (\frac{57-y}{3y^2})y - 10 = 0$
  $(57 - y)^2 + 3y^3(-y + 57) - 90y^4 = 0$

2. **Implementación:** Para la solución de este problema se reutilizo la implementación de Müller anteriormente desarrollada como método para aproximar las raíces de las ecuaciones resultantes y por tanto el valor de los puntos de intersección.
3. **Pruebas:**
   - **prueba 1 sobre** $10 - x^2 + 3(-x^2 + 10) - 57x = 0$ **con** $x_0 = 0$**,** $x_2 = 3$ **tolerancia** 2^{-16}

```
raiz aproximada:  2.00000000000000
6  iteraciones
```

- **prueba 2 sobre** $10 - x^2 + 3(-x^2 + 10) - 57x = 0$ **con** $x_0 = 3$**,** $x_2 = 5$ **tolerancia** 2^{-16}

```
raiz aproximada:  4.39374419328860000000000000000
8  iteraciones
```

- **prueba 3 sobre** $(57-y)^2+3y^3(-y+57)-90y^4 = 0$ **con** $x_0 = 2$**,** $x_2 = 4$ **tolerancia** 2^{-16}

```
raiz aproximada:  3.00000000000000000000000000000
1  iteraciones
```

- **prueba 3 sobre** $(57 - y)^2 + 3y^3(-y + 57) - 90y^4 = 0$ **con** $x_0 = -3$**,** $x_2 = -2.5$ **tolerancia** 2^{-16}

```
raiz aproximada:  -2.11778101471418000000000000000
7  iteraciones
```

4. **Conclusion:** En pruebas se evidencia que en efecto las raíces de las dos ecuaciones encontradas coinciden con el valor de los puntos de intersección de las dos ecuaciones originales, además se confirma la efectivada que tiene en general el método de Müller ya que, dada la función y el intervalo, este logra encontrar en relativamente pocas iteraciones el valor de las raíces con un precisión decente.

3 Librerias en R y/o Python

- **Algoritmo de Brent libreria scipy:** en el ejercicio usamos la libreria de scipy que tiene una implementación del algoritmo de brent dentro de root_scalar[2].

  sol=root_scalar(f,method='brentq',bracket=[0.5,0.7],xtol=2e-60)

  Metodo que segun la libreria es un metodo un poco más seguro haciendo uso de una extrapolación cuadratica inversa hace una combinación entre corchete de raíces, la bisección de intervalo y la interpolación cuadrática inversa.

Matplotlib es una biblioteca en la cual encontramos una forma para poder implementar ayudas visuales, ya sean estéticas, animadas o inluso interactivas. Al implementar la parte de Pyplot le estamos realizando cierto cambio a la misma imágen.

Cuando implementamos np.linspace debemos poner 3 valores. El primero de ellos es el "start" en el cual debemos poner el valor inicial de la secuencia. El segundo dato que se necesita, es el "stop" que nos dice el valor final de la secuencia. Este dato se debe poner siempre y cuando el punto final no este por el dato "false". Por ultimo, tenemos el "num" en el cual se muestra el número de muestras que deseamos generar.

Al realizar "plt.plot" la gráfica se encuentra en los puntos que en esta sea mandada, ya sea en "x,y" o, en nuestro caso, "x,f(x)". Estas quedaran como líneas.

Cuando implantamos "plt.grid", ésta nos ayuda a configurar las líneas de la cuadrícula.

Con la ayuda de "plt.axhline" y "plt.axvline" sirven con el fin de agregar una línea horizontal y una vertical, respectivamente, a través de los ejes.

Por último, el "plt.show", se necesita para mostrar la gráfica final.

Root scalar es un método de Scipy de la sección de optimize, permite hallar raices según el método de solución que se ingrese por parámetro y según la tolerancia que se indique. Root scalar permite devolver las raices de la función escalar que le fue ingresada para resolver según el método de Brent con una tolerancia de 2^{-50}[2].

4 Conclusión general:

Después del desarrollo de los ejercicios se observo como se pueden aprovechar de forma combinada la ventajas de los distintos métodos, como se evidencio con el método de brent que al alternar el método según sea conveniente se logra aproximar las raíces de funciones algo problemáticas como la del primer ejercicio, la cual puede presentar problemas con la interpolación lineal o la interpolación cuadrática inversa implementadas por separado, o puede converger pero muy lentamente como seria el caso de usar únicamente bisección, en cambio con brent se aprovecha la velocidad de convergencia que pueden ofrecer las interpolaciones anteriormente mencionas y la garantía de convergencia de la bisección, Sumado a esto con el desarrollo del segundo ejercicio se evidencio como pueden aprovecharse los métodos de aproximación de raíces para fines como puede ser aproximar los puntos de intersección entre dos curvas.

5 Referencias

[1] SciPY.org, [En línea]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.brentq.html . [Último acceso: 13 03 2021].

[2] sciPY.org, [En línea]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.root-scalar.html. [Último acceso: 13 03 2021].

[3] tutorials points, [En línea]. Available: https://www.tutorialspoint.com/numpy/index.htm. [Último acceso: 13 03 2021].

[4] J. L. d. l. F. O'Connor, de Ingeniería de los Algoritmos y los Metodos Numericos, 2016, pp. 50-51.