RETO 1: ANALISIS NUMERICO

Juan Alejandro Diaz, Cristian Camilo Benitez, Andres Ricardo Porras

14 de Marzo 2021

1. Algoritmo Brent

Es un algoritmo híbrido de busqueda de raices que combina el metodo de biseccion, el metodo de la secante y la interpolacion cuadratica inversa. Tiene la confiabilidad de la biseccion, pero puede ser tanr apido como algunos de los metodos menos confiables. El algoritmo intenta ultilizar el método de la secante de convergecia rápida potencialmente o la interpolacion cuadrática inversa si es posible, pero recurre al método de bisección más robusto si es necesario, El metodo de brent se debe a Richard brent y se basa en un algoritmo anterior de Theodorus Dekker. En consecuencia, el método tambien se conoce como método de brent dekker. Se deben satisfacer dos desigualdades simultáneamente: Dada una tolerancia númerica especifica.

Algoritmo

```
input a, b, and (a pointer to) a function for f
calculate f(a)
calculate f(b)
if f(a)f(b) ≥ 0 then
    exit function because the root is not bracketed.
end if
if |f(a)| < |f(b)| then
    swap (a,b)
end if
c := a
set mflag
repeat until f(b or s) = 0 or |b − a| is small enough (convergence)
    if f(a) ≠ f(c) and f(b) ≠ f(c) then
s := af(b)f(c)/((f(a) − f(b))(f(a) − f(c))) + bf(a)f(c)/((f(b) − f(a))(f(b) − f(c))) + cf(a)f(b)/((f(c) − f(a))(f(c) − f(b)))
(inverse quadratic interpolation)
    else
        s := b − f(b)(b − a)/(f(b) − f(a))  (secant method)
    end if
    if (condition 1) s is not between (3a + b)/4 and b or
       (condition 2) (mflag is set and |s−b| ≥ |b−c|/2) or
       (condition 3) (mflag is cleared and |s−b| ≥ |c−d|/2) or
       (condition 4) (mflag is set and |b−c| < |δ|) or
       (condition 5) (mflag is cleared and |c−d| < |δ|) then
        s := (a + b)/2  (bisection method)
        set mflag
    else
        clear mflag
    end if
    calculate f(s)
    d := c  (d is assigned for the first time here; it won't be used above on the
first iteration because mflag is set)
    c := b
    if f(a)f(s) < 0 then
        b := s
    else
        a := s
    end if
    if |f(a)| < |f(b)| then
        swap (a,b)
    end if
end repeat
output b or s (return the root)
```

Figura 1: Funcionamiento general del algoritmo

- Graficado con python

  Primero se utilizo las librerias de ***matlotlib.pyplot***, esto con el fin de graficar la funcion dada en el enunciado del ejercicio.Se hizo de esta manera por que permitio realizar un acercamiento al punto donde se obvervo que estaba la raiz. Luego de hacer la grafica de esta funcion se pudo observar el interbalo en el cual esta la raiz, el cual fue enviado como paramatro a la funcion que usara del metodo de brent para calular la raiz.

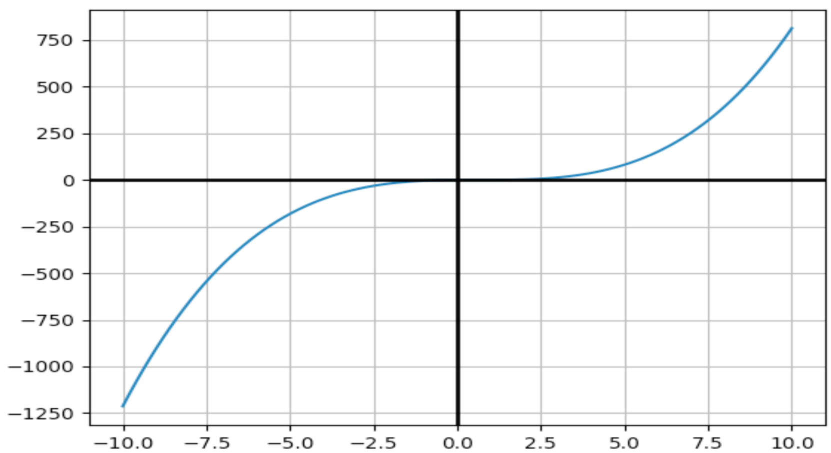


Figura 2: Funcion graficada para brent con python

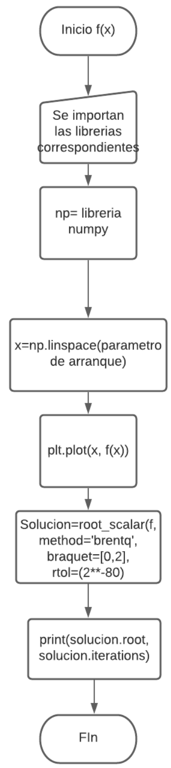


Figura 3: Diagrama de flujo con librerias

- Codigo en Python

  En el codigigo relaizado en python como ya se menciono anteriormente, lo primero que se hizo fue hacer la grafica para asi obtener el intervalo en el cual definida la raiz. Luego con las ayuda de las liberias encontradas en el punto numero tres del reto, se decidio usar de la liberia ***scipy.ptimize*** , de donde se importo la funcion ***root-scalar***, en la cual como parametros son enviados:

  1. La funicon a usar
  2. El metodo a usar (en este caso brent)
  3. El intervalo
  4. La tolerancia

  Donde la funcion aplica el metodo de brent y nos devuelve la raiz y el numero de iteraciones. A continuacion se encuntra el codigo descrito anteriormente y sumado a esto todas las pruebas realizadas.

```
import numpy as np
from scipy.optimize import root_scalar
import math
import matplotlib.pyplot as plt


def f(x):
    return ((x**3) - (2*(x**2)) + ((4*x)/3) - (8/27))


# graficar parqa recocnocer el intervalo
x = np.linspace(start=-10, stop=10, num=100)
plt.plot(x, f(x))
plt.grid()
plt.axhline(y=0, linewidth=2, c='k')
plt.axvline(x=0, linewidth=2, c='k')
#plt.show()

solucion = root_scalar(f, method='brentq', bracket=[0, 2], rtol=(2**-50))
print(f"Metodo de Brent:\n\
      - raiz= {solucion.root}\n\
      - Iteraciones ={solucion.iterations}\n")
```

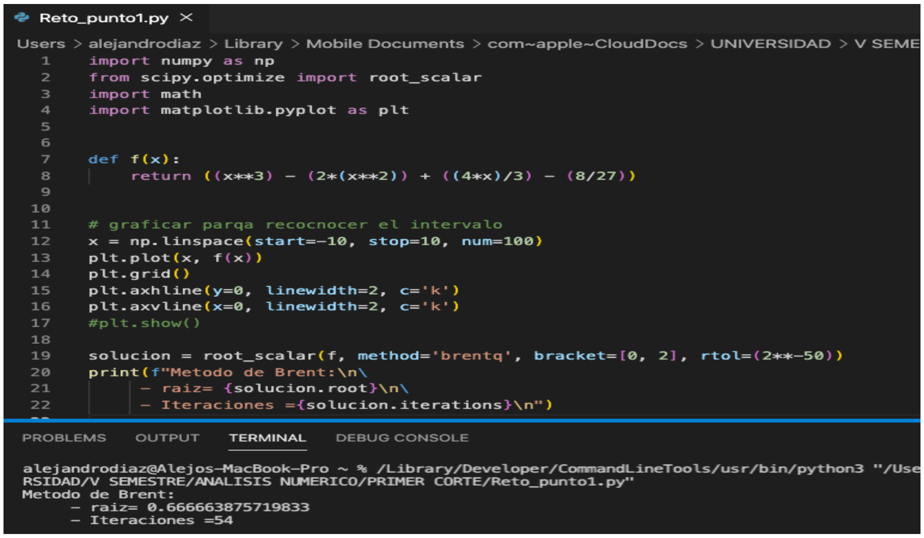
- Pruebas y comprobación

  En las pruebas se pudo obeservar en la figura 4 donde se uso la tolerancia de 2^-50 pedida en el problema, que el metodo devulve en 54 iteraciones el valor de la raiz con una buena precision.

  Ademas en la figura 5 donde se uso la tolerancia de 2^-15 se observa como el metodo devulve en 49 iteraciones el valor de la raiz pero en este caso con una menor precisióm.

  En la ultim prueba realizada, que se puede observar en la figura 6 donde se uso la tolerancia de 2^-51 el metodo devulve que con esta tolerancia la precision aumenta pero se queda en un numero tan pequeño que la maquuina entre en un ciclo infinito.

  Por ultimo se utilizo el software de Geobra para poder comprobar las respuestas arrojadas por el metodo de brent, donde se observa que es acertada la respuesta dada por el algoritmo de brent.



Reto_punto1.py

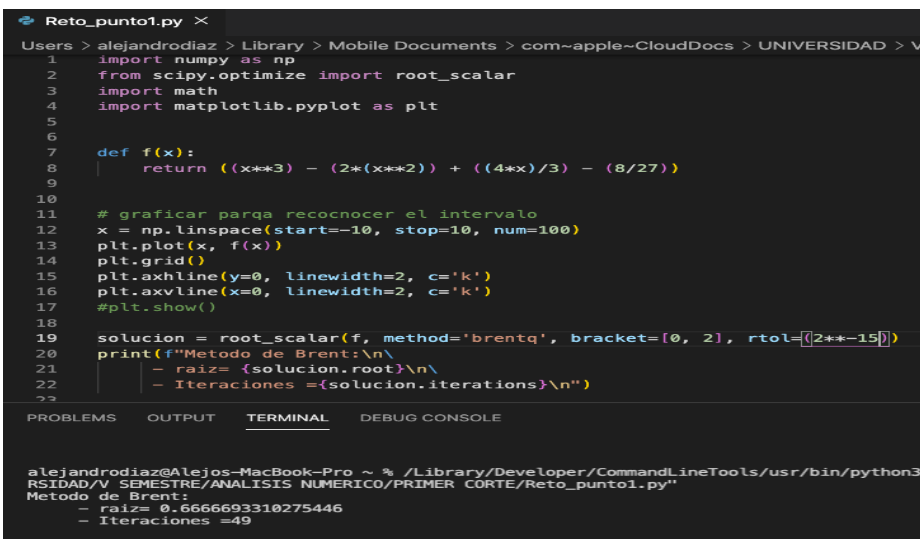
Users > alejandrodiaz > Library > Mobile Documents > com~apple~CloudDocs > UNIVERSIDAD > V SEME

```
import numpy as np
from scipy.optimize import root_scalar
import math
import matplotlib.pyplot as plt


def f(x):
    return ((x**3) - (2*(x**2)) + ((4*x)/3) - (8/27))


# graficar parqa recocnocer el intervalo
x = np.linspace(start=-10, stop=10, num=100)
plt.plot(x, f(x))
plt.grid()
plt.axhline(y=0, linewidth=2, c='k')
plt.axvline(x=0, linewidth=2, c='k')
#plt.show()

solucion = root_scalar(f, method='brentq', bracket=[0, 2], rtol=(2**-50))
print(f"Metodo de Brent:\n\
        - raiz= {solucion.root}\n\
        - Iteraciones ={solucion.iterations}\n")
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
alejandrodiaz@Alejos-MacBook-Pro ~ % /Library/Developer/CommandLineTools/usr/bin/python3 "/Use
RSIDAD/V SEMESTRE/ANALISIS NUMERICO/PRIMER CORTE/Reto_punto1.py"
Metodo de Brent:
        - raiz= 0.666663875719833
        - Iteraciones =54
```

Figura 4: Primera prueba con Brent



Reto_punto1.py

Users > alejandrodiaz > Library > Mobile Documents > com~apple~CloudDocs > UNIVERSIDAD > V

```
import numpy as np
from scipy.optimize import root_scalar
import math
import matplotlib.pyplot as plt


def f(x):
    return ((x**3) - (2*(x**2)) + ((4*x)/3) - (8/27))


# graficar parqa recocnocer el intervalo
x = np.linspace(start=-10, stop=10, num=100)
plt.plot(x, f(x))
plt.grid()
plt.axhline(y=0, linewidth=2, c='k')
plt.axvline(x=0, linewidth=2, c='k')
#plt.show()

solucion = root_scalar(f, method='brentq', bracket=[0, 2], rtol=(2**-15))
print(f"Metodo de Brent:\n\
        - raiz= {solucion.root}\n\
        - Iteraciones ={solucion.iterations}\n")
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
alejandrodiaz@Alejos-MacBook-Pro ~ % /Library/Developer/CommandLineTools/usr/bin/python3
RSIDAD/V SEMESTRE/ANALISIS NUMERICO/PRIMER CORTE/Reto_punto1.py"
Metodo de Brent:
        - raiz= 0.6666693310275446
        - Iteraciones =49
```

Figura 5: Segunda prueba con Brent

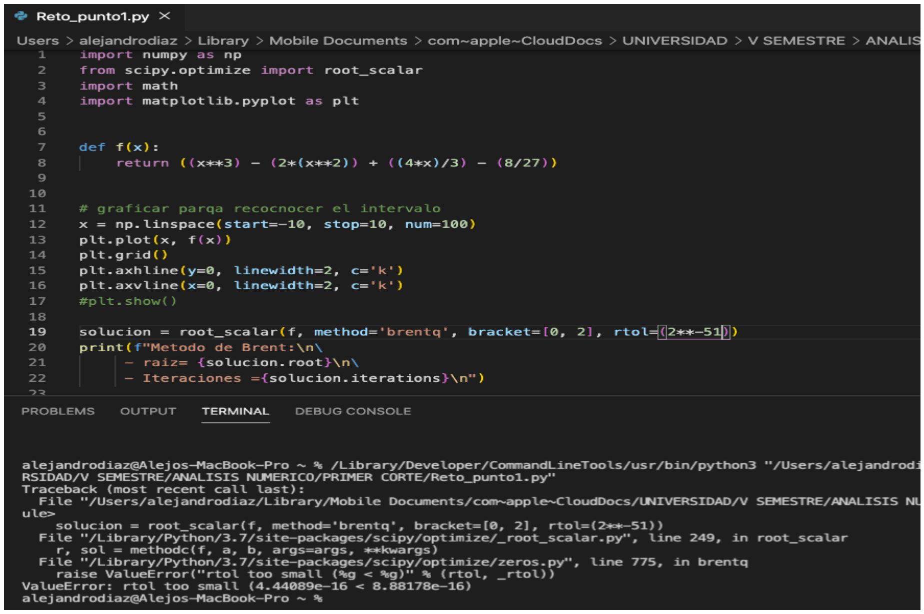


Figura 6: Tercera prueba con Brent

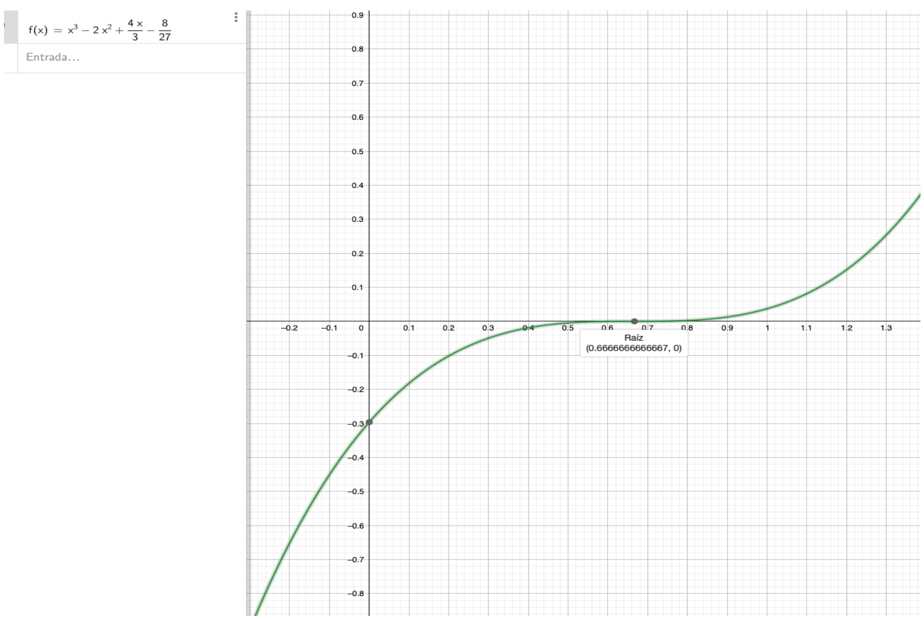


Figura 7: Raiz calcuada con geobra

2. Intersección entre curvas

El metodo por el cual vamos a llevar a cabo la interseccion entre las dos funciones es el de punto fijo. Para resumir, el metodo de punto fijo, tambien conocido como método de iteracion funcional, es el fundamento matematico para construir métodos eficientes para el cálculo de raices reales de ecuaciones no lineales. El metodo consiste en rescribir la ecuación f(x)=0 en la forma x=g(x). Esta nueva ecuacion debe ser equivalente a la ecuacion original en el sentido que debe satisfacerse con la misma raiz, es decir, la existencia de un punto fijo r de la ecuacion x=g(x) es equivalente a encontrar una raiz real r de la ecuacion f(x)=0; r=g(r) y esto de bicondicional f(r)=0.

El procedimiento empieza con una estimacion o conjetura inicial x, que es mejorada por iteracion hasta alcanzar la convergencia. Para que converja, la derivada (dg/dx) debe ser menor que 1 en magnitud (por lo menos para los valores de x que se encuentran durante las iteraciones). La convergencia será establecida madiante el requisito de que el cambio en x de una iteracion a la siguiente no sea mayor en magnitud que alguna pequeña cantidad e.

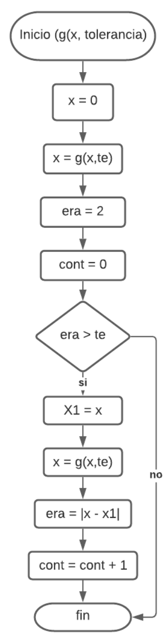


Figura 8: Diagrama de flujo metodo de punto fijo

El algoritmo que habitual mente se utilizaba era para calcular la reiz de una funcion determinando la cantidad de iteraciones y especificando una tolerancia y un error, en este caso, se el reto establecia dos funciones la cuales se generaba una interseccion entre estas dos. Para evaluar las expresiones, se tuvo que determinar el valor de y de la primera y luego remplazar estos valores en la segunda.

La libreria scipy nos permitia declarar el metodo de punto fijo para determinar la raiz de la expresion ya descomprimida, como se muestra en el algoritmo a continuacion:

- Codigo en Python Punto fijo con librerias

```
from scipy import optimize
import numpy as np
from array import array
import matplotlib.pyplot as plt


def f(x):
    return ((3*(x**4))-(61*(x**2))-(57*x)+310)

x = np.linspace(start=-10, stop=10, num=100)
plt.plot(x, f(x))
plt.grid()
plt.axhline(y=0, linewidth=2, c='k')
plt.axvline(x=0, linewidth=2, c='k')
plt.show()

respuesta = optimize.fixed_point(f, x0=[2,4.5], xtol=2**-16)
print(respuesta)
```

  Ahora en el algoritmo a continuacion, realizamos la implementacion del metodo pero mediante un algortimo contruido por nosotros mismos.

- Codigo en python de punto fijo Nuestro

```
import numpy as np
import math


def puntofijo(gx, a, tolera, iteramax=100):
    i = 1
    b = gx(a)
    tramo = abs(b-a)
    while(tramo >= tolera and i <= iteramax):
        a = b
        b = gx(a)
        tramo = abs(b-a)
        i = i + 1
        print("iteracion:" , i)
```

```
        print(" raiz:" , b)

    respuesta = b



# INGRESO

#def gx(x): return np.exp(-x)
def gx(x):return ((3*(x**4))-(61*(x**2))-(57*x)+310)

a = 2      # intervalo
b = 4,5
tolera = 2**-16

# PROCEDIMIENTO
puntofijo(gx, a, tolera)
```

Especificamos estas implementaciones con el fin de demostrar cual pseudocodigo es ma efciciente y ver cual nos arroja resultados mas precisos. A continuacion, en la frigura 7 y 8 se mostrara la ejecucion de las dos implemntacion mencionadas anteriormente:

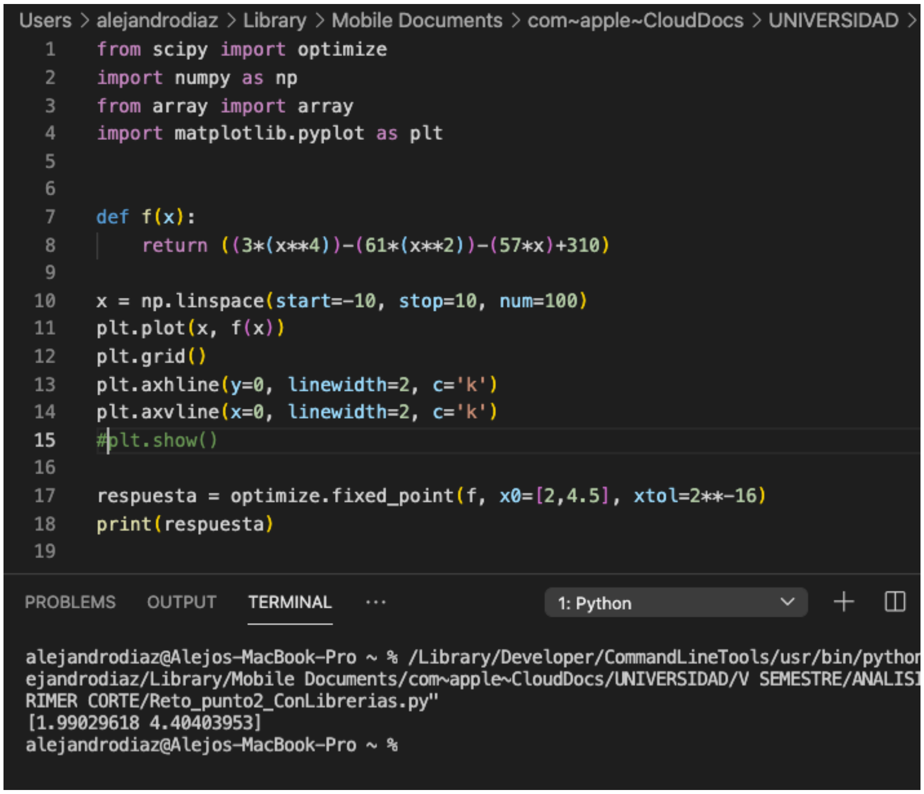


Figura 9: Prueba uno punto fijo

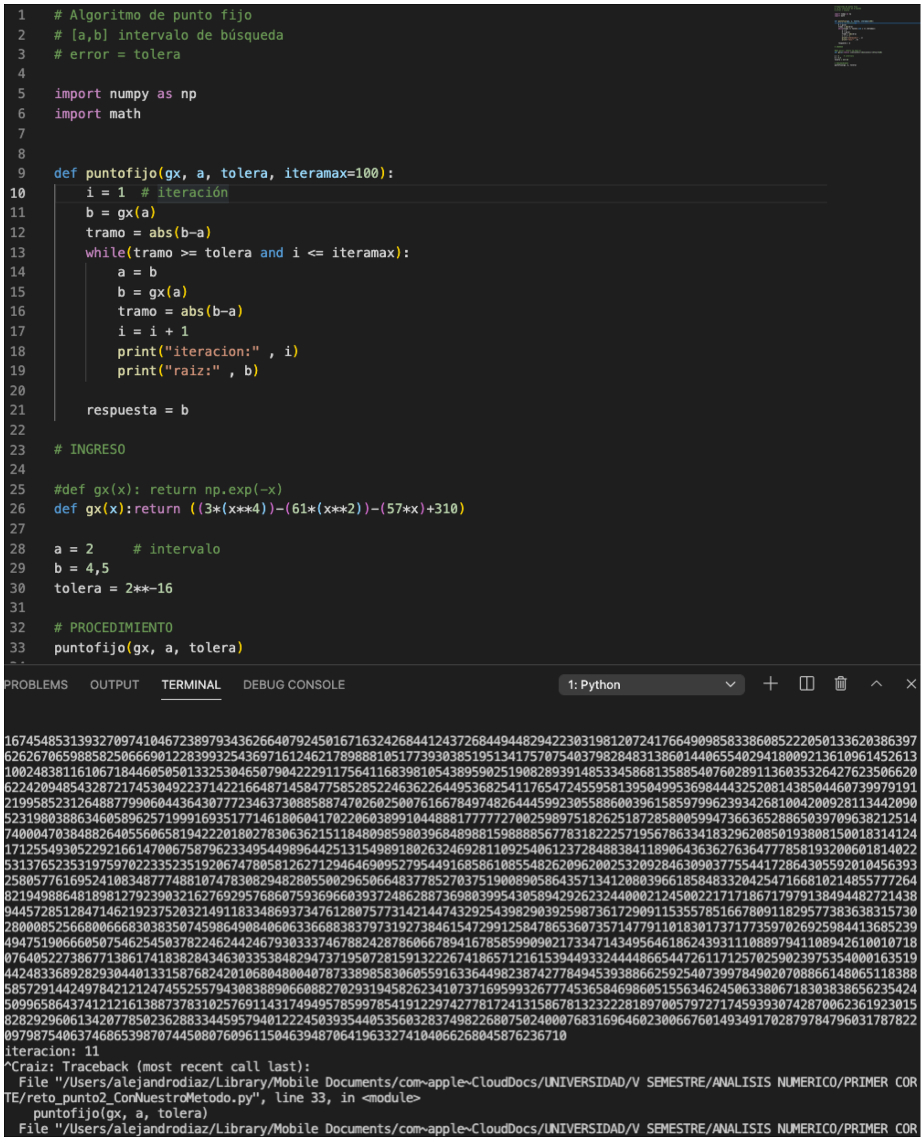


Figura 10: Prueba uno punto fijo

Evaluando los resultados en entre las dos implementacion, obtuvimos resultados completamente diferentes entre las dos; En la prueba de punto fijo aplicado con nuestro algoritmo, las iteraciones no eran precisas, eran erroneas. En la implementacion al metodo con la libreria scipy, arroja un resultados sin determinar la cantidad de iteraciones efectuadas, pero las raices son las correctas. Más adelante, en la parte de conclusiones, determinaremos el porque sucedio esto.

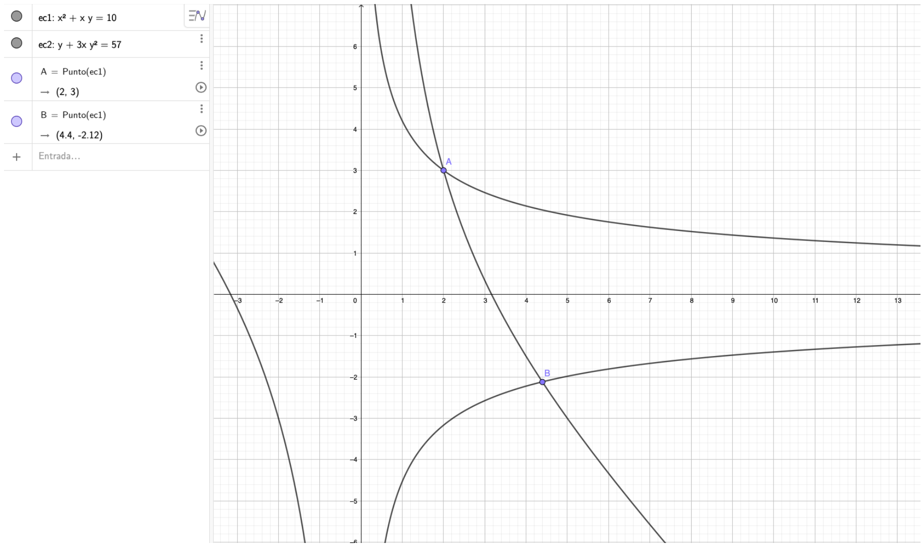


Figura 11: Intersección entre las curvas

3. Librerias en Python

Se realizo una busqueda de la documentacion de las librerias numpy y scipy para la resolucion de los puntos anteriores. Para cada libreria se encontraron diferentes funciones con su documentacion para el manejo de parametros que fueron de gran ayuda. A continuacion encontrara las funciones con sus diferentes parametros y la explicacion de su funcionamiento usadas en cada libreria en los puntos anteriores.

- Numpy: Es una biblioteca del lenguaje de programacion python que da soporte para crear matrices y vectores grandes multidimensionales

  Desde numpy se utilizaron para mejorar temas como la precision, poder manejar numeros mas pequeños y hacer graficas para determinar intervalos de las funciones

- Scipy: Es una biblioteca libre y de código abierto para Python. Se compone de herramientas y algoritmos matemáticos

  - Root-scalar: En la cual se nos da el prototipo de la funcion para jugar con los parametros que se pueden introducir como lo son la funcion a usar, la tolerancia, los intervalos, las iteraciones y tambien usar funicones como secante, newton , brent, etc.

    scip.optimize.rootscalar(f, args=(), method=None, bracket=None, fprime=None, fprime2=None, x0=None, x1=None, xtol=None, rtol=None, maxiter=None, options=None)

- Fixed-Point: Para esta funicon utilzada de optimize nos permite aplicar el metodo de punto fijo para hallar la raiz en un intervalo de una funion dada con una tolerancia especifica. Ademas de eso permite acelerar el metodo con aitken mandando por comandos que method aplicado se "del2"

  scipyoptimize.fixedpoint(func, x0, args=(), xtol=1e-08, maxiter=500, method='del2')

4. Conclusiones

- Las librerias que nos ofrecen algunos leguajes de programacion nos permite llegar a una solucion de una manera mas eficiente; En los metodos iterativos de brent y de punto fijo, las librerias numpy y scipy nos permitio integrar las funciones directamente para llegar a la solucion y sus raices junto a la cantidad de iteraciones realizadas.

- Para nuestra solucion de interseccion entre curvas aplicado al metodo de punto fijo, utilizando el codigo desarrollado por nosotros; si logra determinar un numero de iteraciones pero la expresion al ser tan grande y compleja como lectura pasada como parametro dentros de las especificaciones del algoritmo, se queda iterando y no proporciona una solucion precisa. Teniendo en cuenta lo anterior, tuvimos que aplicar el metodo utilizando librerias, mandando parametros como la tolerancia, los intervalos y la funcion, se nos proporciona una solucion precisa ya que determina las raices de las funciones para la interseccion.

5. Referencias

Online . Disponible: `docs.scipy.org/doc/scipy/reference/optimize.rootscalarbrentq`

Online . Disponible: `docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fixedpoint`

Online . Disponible: `es.qaz.wiki/wiki/Brent`