UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
ENES JURIQUILLA
JAIMES AVILA NOÉ
LICENCIATURA EN TECNOLOGÍA

REPORTE DE PRÁCTICA: APLICACIONES DE LA MATRIZ INVERSA

Introducción.

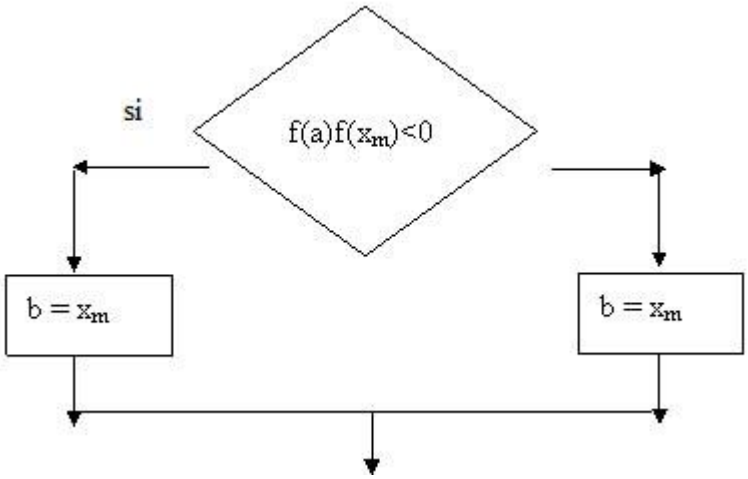
El método de la bisección consiste en obtener una mejor aproximación de la raíz a partir de un intervalo inicial (a,b) en el cual hay un cambio de signo en la función, es decir: $f(a)f(b)<0$.

Se obtiene el punto medio:

$$x_m = \frac{a+b}{2}$$

X_m es la nueva aproximación a la raíz, y se vuelve a tomar un intervalo, pero ahora mas pequeño, considerando que siga existiendo un cambio de signo en la función, es decir, el nuevo intervalo queda determinado por:

:



El método termina cuando se cumple con alguna condición de paro, generalmente es la tolerancia :

$$T=|B-A|$$

Este es un método “de encierro”, para aplicarlo se debe contar con un intervalo inicial, en donde f(a)*f(b) < 0. Este método requiere de menos pasos en un programa, sin embargo converge mas lentamente que el de Newton-Raphson.

Los pasos del método son los siguientes:

1.- Localizar un intervalo que contenga al menos una raíz.

2.- Dividir el intervalo en dos partes iguales reteniendo la mitad en donde f(x) cambia de signo, para conservar al menos una raíz.

3.- Repetir el proceso varias veces hasta cumplir con la tolerancia deseada.

En este método, podemos notar que el rango se reduce a la mitad en cada iteración, por lo mismo, hasta alcanzar la tolerancias deseada, este converge muy lentamente.

Desarrollo.

- Utilizando las librerías numpy y scypy,se implementó el método de la bisección, donde se buscan las raíces de un polinomio, mediante la búsqueda de puntos de inflexión en el rango, el cual se subdivide en dos rangos.
- Adicionalmente, se implementó el método de busqueda exhaustiva (ingenuo) y se compararon los tiempos de ejecución.

```
from sympy.abc import x, y
from sympy import *
import time
import numpy as np
import time
class NumericalMethods:
    def bisectionMethod(self, a, b, tol, func):

        if func.subs(x,a) * func.subs(x,b) < 0:
            print("Existe un cambio de signo")
        else:
            print("No existen raices reales en el intervalo")
            exit(0)
        error = 3
        c = (a + b) /2
        while(True):
            if  (b-a)<tol:
              return c
            elif float(func.subs(x,a))*float(func.subs(x,c)) > 0:
              a=c
            else:
              b=c
            c = (a + b)/2
            error=abs(b-a)
```

Código del método de la bisección.

```python
class Ing_Search:
  def evaluar(self,f,lx):
    ly=[]
    for i in (lx):
        ly.append(f.subs(x,i))
    return ly

  def nearest(self,f,lx):
    evalua=self.evaluar(f,lx)
    sol=float(solve(f,x)[0])
    mascercano=min(evalua, key=lambda x:abs(x-sol))
    return mascercano
```

Código método ingenuo

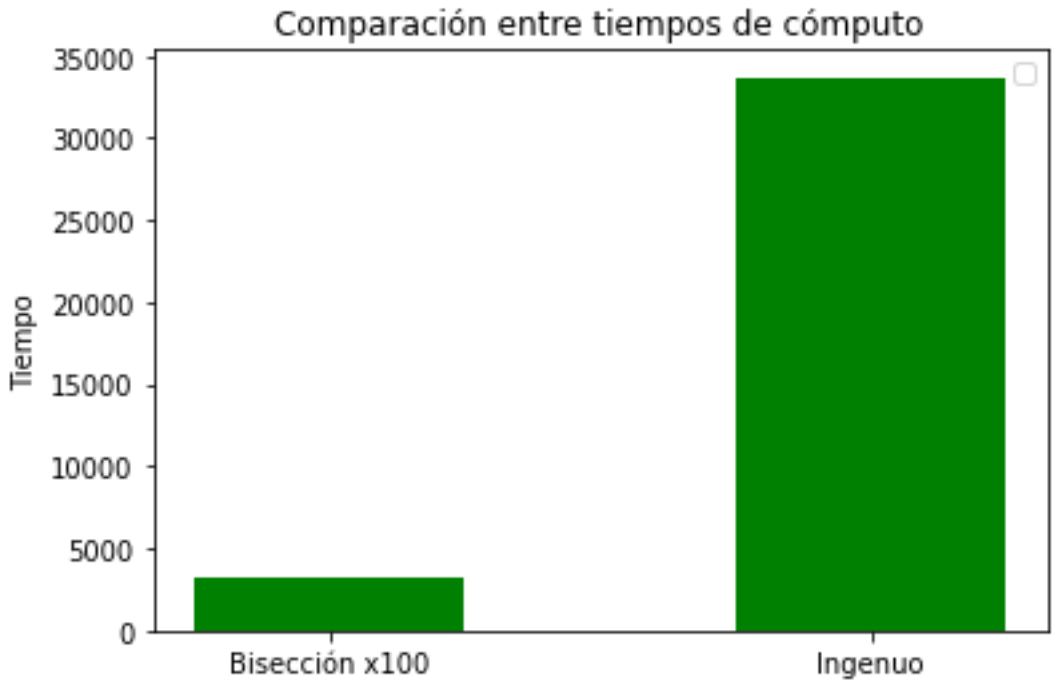
Resultados.

```
3*x**3 + 8
Existe un cambio de signo
Tiempo:  24.02186393737793 ms
Valor aproximado método de la bisección:  -1.3867229223251343
Valor real:  -1.3867225487012693
```

Método de la bisección

```
Tiempo:  33261.794328689575 ms
Valor real:  -1.3867225487012693
Valor aproximado búsqueda ingenua:  -1.33640799903287
```

Método ingenuo (Se aprecia como el tiempo es considerablemente mayor)



Podemos notar que a pesar de que el tiempo del método de la bisección se multiplicó por 100, sigue siendo mucho menor.

Conclusiones.

- Podemos concluir que dentro de los métodos numéricos, hay métodos que pueden reducir enormemente el tiempo de cómputo, por lo cual hay que ser inteligentes al momento de programar este tipo de algoritmos, ya que entre ellos hay significativas diferencias en cuanto a optimización se refiere, pero a pesar de que el método de la bisección es mas exacto, adolece de problemas como el caso de que el intervalo debe estar contemplado, ya que de lo contrario, podría no dar resultados, como en el caso de una parábola, que según el rango elegido puede o no cambiar de signo.

Referencias

http://test.cua.uam.mx/MN/Methods/Raices/Biseccion/Biseccion.php

https://www.uv.es/~diaz/mn/node18.html