Programa: cerosDeFunciones

por: Ricardo Santana

## A. Introducción

El código proporcionado es una implementación de una clase llamada "zero" en Python. Esta clase tiene varios métodos que implementan diferentes algoritmos numéricos para encontrar aproximaciones a cero de una función.

Los distintos métodos de la clase fueron copiados del libro **Métodos Numéricos con Python**, *Ovalle D., Bernal M., Posada J.*, Editorial Politecnico Grancolombiano, (2021)

## B. ceroDeFunciones

### Entrada:

- f: una función.
- fprima: la derivada de la función f.
- itera: el número máximo de iteraciones permitidas.
- tol: la tolerancia para determinar la convergencia.
- aprox_inic_1: la primera aproximación inicial.
- aprox_inic_2: la segunda aproximación inicial (opcional, con un valor predeterminado de 1).

### Salida:

- p: cero aproximado de la función

## C. Programa

### Método privado __mostrar_proceso

El método privado __mostrar_proceso se utiliza para mostrar el proceso de iteración en los algoritmos. Imprime el número de iteración y el valor actual de la aproximación.

### Método biseccion

El método biseccion implementa el algoritmo de bisección para encontrar una aproximación a cero de la función. Toma como argumentos el intervalo inicial [a, b]. El algoritmo realiza iteraciones

hasta que se cumpla una condición de convergencia. Devuelve la aproximación encontrada o None si se agotan las iteraciones.

### Método newton

El método newton implementa el algoritmo de Newton para encontrar una aproximación a cero de la función. Toma como argumento una aproximación inicial p0. El algoritmo realiza iteraciones hasta que se cumpla una condición de convergencia. Devuelve la aproximación encontrada o None si se agotan las iteraciones.

### Método secante

El método secante implementa el algoritmo de la secante para encontrar una aproximación a cero de la función. Toma como argumentos dos aproximaciones iniciales p0 y p1. El algoritmo realiza iteraciones hasta que se cumpla una condición de convergencia. Devuelve la aproximación encontrada o None si se agotan las iteraciones.

### Método puntofijo

El método puntofijo implementa el algoritmo del punto fijo para encontrar una aproximación a un punto fijo de la función. Toma como argumentos una función g_x, la derivada de la función g_xprima, y un intervalo [a, b] para verificar la convergencia. El algoritmo realiza iteraciones hasta que se cumpla una condición de convergencia. Devuelve la aproximación encontrada o None si se agotan las iteraciones.

## D. Código

```
import numpy as np
import math as mt
from funciones import *

class zero():
    def __init__(self, f, fprima, itera, tol, aprox_inic_1, aprox_inic_2 = 1):
        self.f = f
        self.fprima = fprima
        self.itera = itera
        self.tol = tol
        self.p0 = aprox_inic_1
        self.p00 = aprox_inic_2 #predeterminada
        self.proceso = True
        self.p = 0

    def __mostrar_proceso(self, i, p):
        if self.proceso==True:
            print("i = {0:<2}, p = {1:.12f}".format(i, p))
```

```
def biseccion(self, a, b):
    """
    Implementación método de bisección
    Entradas:
    f -- función
    a -- inicio intervalo
    b -- fin intervalo
    tol -- tolerancia
    itera -- número máximo de iteraciones
    Salida:
    p aproximación a cero de f
    None en caso de iteraciones agotadas
    """
    i = 1
    while i <= self.itera:
        p = a + (b - a)/2
        self.__mostrar_proceso(i, p)
        if abs(self.f(p)) <= 1e-15 or (b - a)/2 < self.tol:
            self.p = p
            return p
        i += 1
        if self.f(a)*self.f(p) > 0:
            a = p
        else:
            b = p
    print("Iteraciones agotadas: Error!")
    self.p = p
    return None

def newton(self):
    """
    Implementación método de Newton
    Entradas:
    f -- función
    fprima -- derivada función f
    p0 -- aproximación inicial
    tol -- tolerancia
    itera -- número máximo de iteraciones
    Salida:
    p aproximación a cero de f
    None en caso de iteraciones agotadas
    """
    i = 1
    p0 = self.p0 #copia
    while i <= self.itera:
        p = p0 - self.f(p0)/self.fprima(p0)
        self.__mostrar_proceso(i, p)
        if abs(p - self.p0) < self.tol:
            self.p = p
            return p
        p0 = p
```

```
            i += 1
        print("Iteraciones agotadas: Error!")
        self.p = p
        return None

    def secante(self):
        """
        Implementación método de la secante
        Entradas:
        f -- función
        p0 -- aproximación inicial 01
        p1 -- aproximación inicial 02
        tol -- tolerancia
        itera -- número máximo de iteraciones
        Salida:
        p aproximación a cero de f
        None en caso de iteraciones agotadas
        """
        if self.p00 <= self.p0:
            return f'las aproximaciones iniciales {self.p0} y {self.p00} no son adecuadas

        i = 2
        p0 = self.p0 #copia
        p1 = self.p00 #copia
        while i <= self.itera:
            p = p1 - (self.f(p1)*(p1 - p0))/(self.f(p1) - self.f(p0))
            self.__mostrar_proceso(i, p)
            if abs(p - p1) < self.tol:
                self.p = p
                return p
            p0 = p1
            p1 = p
            i += 1
        print("Iteraciones agotadas")
        self.p = p
        return None

    def puntofijo(self, g_x, g_xprima, a, b):
        """Implementación método de punto fijo
        Entradas:
        f -- función
        p0 -- aproximación inicial
        tol -- tolerancia
        n -- número máximo de iteraciones
        Salida:
        p aproximación a punto fijo de f
        None en caso de iteraciones agotadas"""
        #se verifica que la aproximacion inicial esté dentro del intervalo
        if self.p0 < a or self.p0 >b:
            return 'aproximación inicial fuera de intervalo'
```

```
#se verifica la convergencia
x = a
paso = abs(b-a)/self.itera
while x<=b:
    if abs(g_xprima(x))>1:
        return 'la funcion punto fijo escogida no converge'
    x+=paso

i = 1
p0 = self.p0
while i <= self.itera:
    p = g_x(p0)
    self.__mostrar_proceso(i, p)
    if abs(p - p0) < self.tol:
        self.p = p
        return p
    p0 = p
    i += 1
```