Programa: intg

por: Ovalle D., Bernal M., Posada J.

modificado por: Santana R.

A. Introducción

El código proporcionado es una implementación de métodos numéricos para calcular aproximaciones del área bajo una curva. El código define una clase llamada "integrar" que tiene varios métodos para calcular estas aproximaciones.

Los métodos principales de trapecio y simpson fueron extraidos de la siguiente bibliografía:

Arévalo Ovalle, D., Bernal Yermanos, M. A., & Posada Restrepo, J. A. (2021), **Métodos numéricos con Python**, Bogotá: Editorial Politécnico Grancolombiano.

B. intg

Entrada:

- f: Una función que se va a integrar.
- a: El límite inferior del intervalo de integración.
- b: El límite superior del intervalo de integración.
- n: (opcional, valor predeterminado: 100): El número de pasos o subdivisiones del intervalo de integración.

Salida:

- abc: La aproximación del área bajo la curva obtenida utilizando la regla del trapecio o la regla de Simpson, dependiendo del método utilizado.

Métodos de la clase integrar:

- trapecio(): Implementa la regla del trapecio para aproximar el área bajo la curva.
- simpson(): Implementa la regla de Simpson para aproximar el área bajo la curva.
- err_trapecio(f_sp): Calcula el error de aproximación utilizando la regla del trapecio.
- err_simpson(f_sp): Calcula el error de aproximación utilizando la regla de Simpson.

C. Programa

- Método trapecio()

- Método trapecio():
    1. Calcula el tamaño del intervalo h utilizando la fórmula (b - a) / n, donde a y b son los límites del intervalo y n es el número de pasos.
    2. Inicializa una variable acum en 0 para acumular la suma de los valores de la función evaluados en los puntos intermedios del intervalo.
    3. Utiliza un bucle for para iterar desde 1 hasta n - 1. En cada iteración, calcula el valor de x utilizando la fórmula a + h * j, donde j es el número de iteración.
    4. Si j es par, suma 2 * f(x) a acum, de lo contrario, suma 4 * f(x) a acum.
    5. Calcula la aproximación del área bajo la curva utilizando la fórmula (h / 2) * (f(a) + acum + f(b)).
    6. Retorna el valor de la aproximación del área bajo la curva.
- Método simpson():
    1. Calcula el tamaño del intervalo h utilizando la fórmula (b - a) / n, donde a y b son los límites del intervalo y n es el número de pasos.
    2. Inicializa las variables oddsum y evensum en 0 para acumular la suma de los valores de la función evaluados en los puntos intermedios del intervalo.
    3. Utiliza un bucle for para iterar desde 1 hasta n - 1.
    4. En cada iteración, calcula el valor de x utilizando la fórmula a + h * j, donde j es el número de iteración.
    5. Si j es par, suma 2 * f(x) a evensum, de lo contrario, suma 4 * f(x) a oddsum.
    6. Calcula la aproximación del área bajo la curva utilizando la fórmula (h / 3) * (f(a) + evensum + oddsum + f(b)).
    7. Retorna el valor de la aproximación del área bajo la curva.
- Método err_trapecio(f_sp):
    1. Calcula el tamaño del intervalo h utilizando la fórmula (b - a) / n, donde a y b son los límites del intervalo y n es el número de pasos.
    2. Calcula el error de aproximación utilizando la fórmula (h**3 / 12) * sp.diff(f_sp, x, 2).subs(x, self.xi), donde f_sp es la función simbólica y self.xi es un valor aleatorio generado dentro del intervalo.
    3. Retorna el valor del error de aproximación.
- Método err_simpson(f_sp):
    1. Calcula el tamaño del intervalo h utilizando la fórmula (b - a) / n, donde a y b son los límites del intervalo y n es el número de pasos.
    2. Calcula el error de aproximación utilizando la fórmula (h**5 / 90) * sp.diff(f_sp, x, 4).subs(x, self.xi), donde f_sp es la función simbólica y self.xi es un valor aleatorio generado dentro del intervalo.
    3. Retorna el valor del error de aproximación.

D. Código

```
import math as mt
import random as rd
import sympy as sp
from sympy.abc import x,y,z

class integrar():

    def __init__(self, f, a, b, n = 100):
        self.f = f
        self.a = a
        self.b = b
        self.n = n
        self.h = (self.b-self.a)/self.n
        self.xi = rd.uniform(self.a, self.b)

    def trapecio(self):
        """
        Implementación regla del trapecio
        Entradas:
        f -- función
        a -- inicio intervalo
        b -- fin intervalo
        n -- número de pasos
        Salida:
        abc -- aproximación área bajo la curva
        """
        h = (self.b-self.a)/self.n
        acum = 0
        for j in range(1, self.n):
            acum += 2*self.f(self.a + h*j)
            abc = (h/2)*(self.f(self.a) + acum + self.f(self.b))
        return abc

    def simpson(self):
        """
        Implementación regla de Simpson
        Entradas:
        f -- función
        a -- inicio intervalo
        b -- fin intervalo
        n -- número de pasos (par)
        Salida:
        abc -- aproximación área bajo la curva
        """
        h = (self.b-self.a)/self.n
```

```
    oddsum = 0
    evensum = 0
    for j in range(1, self.n):
        x = self.a + h*j
        if j % 2 == 0:
            evensum += 2*self.f(x)
        else:
            oddsum += 4*self.f(x)
        abc = (h/3)*(self.f(self.a) + evensum + oddsum + self.f(self.b)
    return abc

def err_trapecio(self, f_sp):
    h = (self.b-self.a)/self.n
    return (h**3 / 12)*sp.diff(f_sp, x, 2).subs(x, self.xi)

def err_simpson(self, f_sp):
    h = (self.b-self.a)/self.n
```