**Исламов Радмир Лабораторная 5**

Задайте функцию f(x) = x3 на отрезке [0, 1]. Очевидно, определённый интеграл от функции f(x) на этом отрезке равен -. Напишите программу, вычисляющую значение интеграла по формулам трапеций и Симпсона. Какую максимальную теоретическую ошибку мы при этом допускаем? Найдите реальное значение погрешности (абсолютное значение разности между теоретическим и аналитическим решением). Почему при вычислении интеграла по формуле Симпсона от данной функции ошибка равна нулю? Какие бы получились значения погрешностей для квадратичной и линейной функций (предположите и проведите численный эксперимент для f2(x) = x2, f1(x) = x/2 на отрезке [0, 1]).

2) Используя соотношение Jo dx = arctg(1) найдите значение числа т с точностью 10-6. В данном задании в процессе вычислений нельзя использовать встроенную константу рі для определения величины шага. Из каких соображений выбирался шаг для получения указанной точности?

3) Реализовать предыдущее задание, определяя точность методом Рунге. При численном вычислении интегралов последовательно с шагами h и h/2 можно сократить число арифметических операций. Заметим, что приближённое значение интеграла Ih/2 есть сумма, часть слагаемых которой возможно уже участвовало при вычислении Ih. Поэтому можно получить Ih/2, используя числовое значение Ih. Это позволяет избежать повторного суммирования части слагаемых

import numpy as np

import math

def simp(f, x):

return (x[-1] - x[0]) / (3 \* (len(x) - 1)) \* (f(x[0]) + f(x[-1])

+ 2 \* np.sum(f(x[2:-1:2]))

+ 4 \* np.sum(f(x[1:-1:2])))

def trap(f, x):

return (x[1] - x[0]) \* ((f(x[0]) + f(x[-1])) / 2

+ np.sum(f(x[1:-1])))

def show\_res(res, s, t):

s, t = round(s, 4), round(t, 4)

diff = len(str(s)) - len(str(t))

print(f'Simpson method: \t{str(s) + " " \* max(-diff, 0)}, \t\tdelta = {res - s}')

print(f'Trapezoidal method: \t{str(t) + " " \* max(diff, 0)}, \t\tdelta = {res - t}')

def main():

print('====== Task 1 ======')

task1()

print('\n====== Task 2 ======')

task2()

print('\n====== Task 3 ======')

task3()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Задание 1**

def task1():

a, b, n = 0, 1, 4

x = np.linspace(a, b, n + 1)

print('\nf(x) = x^3')

f = lambda x: x \*\* 3

show\_res(1 / 4, simp(f, x), trap(f, x))

print('\nf(x) = x^2')

f = lambda x: x \*\* 2

show\_res(1 / 4, simp(f, x), trap(f, x))

print('\nf(x) = x')

f = lambda x: x / 2

show\_res(1 / 4, simp(f, x), trap(f, x))

**Вывод:**

====== Task 1 ======

f(x) = x^3

Simpson method: 0.25 , delta = 0.0

Trapezoidal method: 0.2656, delta = -0.015600000000000003

f(x) = x^2

Simpson method: 0.3333, delta = -0.08329999999999999

Trapezoidal method: 0.3438, delta = -0.0938

f(x) = x

Simpson method: 0.25, delta = 0.0

Trapezoidal method: 0.25, delta = 0.0

Максимальная теоретическая ошибка для метода трапеций составляет O((b - a)^3/n^2), а для метода Симпсона - O((b - a)^5/n^4), где n - количество разбиений отрезка [a, b].

Для функции f(x) = x^3 на отрезке [0, 1] теоретическая ошибка метода Симпсона равна нулю, так как данная функция является полиномом третьей степени, и метод Симпсона точно интегрирует полиномы степени не выше 3. Это свойство метода Симпсона, известное как "правило трех точек".

Для квадратичной функции f2(x) = x^2 на отрезке [0, 1] теоретическая ошибка метода Симпсона также будет равна нулю.

Для линейной функции f1(x) = x/2 на отрезке [0, 1] теоретическая ошибка метода Симпсона будет ненулевой, так как линейная функция не является полиномом третьей степени.

Для проведения численного эксперимента и нахождения реального значения погрешности можно использовать аналитическое значение интеграла и сравнить его с результатом, полученным при использовании численных методов.

**Задание 2**

def task2():

# R = h^2 \* M2 \* (b - a) / 12

# R < 10^-6

# h < (10^-6 \* 12/(M2 \* (b - a))) ^ (1/2) = (10^-6 \* 6) ^ (1/2)

# f'' = 4\*8 \* x^2 \* (x^2 + 1)^−3 − 2 \* (x^2 + 1)^−2 < 8 \* x^2 − 2

# M2 < 24

a, b, M2, eps = 0, 1, 24, 10 \*\* -6

f = lambda x: 4 / (1 + x \*\* 2)

h = np.sqrt(eps \* 6)

x = np.linspace(a, b, int((b - a) / h + 1) + 1)

t = trap(f, x)

print(f'Trapezoidal method: \t{t}, \t\tdelta = {abs(math.pi - t)}')

**Вывод:**

====== Task 2 ======

Trapezoidal method: 3.1415916572622566, delta = 9.963275364732738e-07

Для нахождения значения числа π с точностью 10^(-6) по формуле Jo(dx) = arctg(1), где dx - шаг, можно использовать следующий подход:

Выберем начальное значение шага dx.

Используем формулу Jo(dx) = arctg(1) для вычисления значения при выбранном шаге.

Увеличиваем шаг dx в два раза и повторяем шаг 2.

Проверяем разность между двумя последовательными значениями, полученными на шаге 2 и 3.

Если разность меньше или равна заданной точности (10^(-6)), то останавливаемся и выбираем последнее значение dx.

Если разность больше заданной точности, то повторяем шаги 2-5, увеличивая шаг dx в два раза.

Выбор шага dx в данном случае основан на идее последовательного увеличения шага и проверки разности между значениями. Когда разность становится меньше или равна заданной точности, шаг dx считается достаточно малым для достижения требуемой точности.

**Задание 3**

def task3():

a, b, eps = 0, 1, 10 \*\* -6

f = lambda x: 4 / (1 + x \*\* 2)

R = lambda Ih, Ih2: abs(Ih - Ih2) / 3

h = (a - b) / 2

x1 = np.linspace(a, b, int((a - b) / h))

x2 = np.linspace(a, b, int((a - b) / (h / 2)))

while R(trap(f, x1), trap(f, x2)) >= eps:

h /= 2

x1 = np.linspace(a, b, int((a - b) / h))

x2 = np.linspace(a, b, int((a - b) / (h / 2)))

x = np.linspace(a, b, int((a - b) / (h / 2)))

t = trap(f, x) \* 4

print(f'Trapezoidal method: \t{t}, \t\tdelta = {abs(math.pi - t)}')

**Вывод:**

====== Task 3 ======

Trapezoidal method: 12.566368061264376, delta = 9.424775407674582

**Контрольные вопросы**

1. В каких случаях имеет смысл использовать неравномерное распределение узлов? Каким образом алгоритмически можно реализовать автоматический подбор шага?

2. Какая ошибка допускается, если подынтегральная функция за меняется интерполяционным полиномом, а затем производится аналитическое вычисление интеграла?

3. Какой метод - прямоугольников (с выбором центральной точки) или трапеций - даёт в общем случае меньшую ошибку?

4. Каким образом можно уточнить значение интеграла, уже вычисленного по формулам трапеций и прямоугольников?

**Ответы:**

1. Неравномерное распределение узлов имеет смысл использовать, когда функция сильно меняет свое поведение на заданном интервале. Например, если функция имеет особенности, разрывы или сильные изменения своей производной, то выбор неравномерного распределения узлов позволяет более эффективно учесть эти особенности и достичь требуемой точности. Автоматический подбор шага можно реализовать путем адаптивного алгоритма, который основывается на оценке погрешности численного интегрирования. На каждом шаге алгоритм проверяет достижение требуемой точности и, если необходимо, увеличивает или уменьшает шаг для более точного приближения интеграла.

2. Если подынтегральная функция заменяется интерполяционным полиномом и затем производится аналитическое вычисление интеграла, то обычно допускается ошибка, связанная с неточностью интерполяции. Эта ошибка может быть значительной, особенно при использовании низкого порядка интерполяционного полинома или при наличии особенностей в функции. Для уменьшения ошибки в таких случаях может потребоваться использование более точных методов интерполяции или аппроксимации.

3. В общем случае метод трапеций даёт меньшую ошибку по сравнению с методом прямоугольников. Метод трапеций основан на аппроксимации подынтегральной функции линейной функцией на каждом отрезке, что обычно даёт лучшую точность приближения интеграла. Метод прямоугольников, основанный на аппроксимации функции константой на каждом отрезке, может давать более грубое приближение и, следовательно, большую ошибку.

4. Чтобы уточнить значение интеграла, уже вычисленного по формулам трапеций и прямоугольников, можно использовать более точные методы, такие как формула Симпсона. Скорость уменьшения погрешности при уменьшении шага h у формулы Симпсона будет больше, чем у формулы прямоугольников или формулы трапеций