Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №3

Вариант № 1

Выполнил: студент группы P3208, Васильев Н. А.

Преподаватель: Машина Е.А.

Санкт-Петербург 2025

# Текст задания

Численное интегрирование.

Цель работы

Найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

Описание метода, расчётные формулы

Метод Ньютона-Котеса:

Метод левых прямоугольников:

Метод правых прямоугольников:

Метод средних прямоугольников:

Метод трапеций:

Метод Симпсона:

Вычислительная реализация задачи:

Вычислить интеграл:

Точное решение:

Получаем точное значение:

Вычисление по формуле Ньютона – Котеса при 𝑛 = 6:

Коэффициенты Котеса для i = 6:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

Получаем:

Вычисление по формулам средних прямоугольников, трапеций и Симпсона при 𝑛=10:

Через метод средних прямоугольников:

Через метод трапеций:

Через метод Симпсона:

**Сравним полученные результаты.**

Точное значение:

По формуле Ньютона – Котеса при 𝑛 = 6 получаем значение , что совпадает с точным значением.

По формуле средних прямоугольников при 𝑛 = 10 получаем значение , получаем разницу: .

По формуле трапеций при 𝑛 = 10 получаем значение , получаем разницу:

По формуле Симпсона при 𝑛 = 10 получаем значение , что совпадает с точным значением.

**Определим относительную погрешность вычислений для каждого метода.**

Для метода Ньютона – Котеса результат совпадает с точным значением, следовательно, погрешности нет.

Для метода средних прямоугольников погрешность равна: .

Для метода трапеций погрешность равна: .

Для метода Симпсона результат совпадает с точным значением, следовательно, погрешности нет.

Листинг программы

class Calculation:  
 def \_\_init\_\_(self, function, lower\_limit, upper\_limit, accuracy, method):  
 self.function = function  
 self.lower\_limit = lower\_limit  
 self.upper\_limit = upper\_limit  
 self.accuracy = accuracy  
 self.method = method  
  
 def left\_rectangle(self, n):  
 h = (self.upper\_limit - self.lower\_limit) / n  
 return h \* sum(self.function(self.lower\_limit + i \* h) for i in range(n))  
  
 def right\_rectangle(self, n):  
 h = (self.upper\_limit - self.lower\_limit) / n  
 return h \* sum(self.function(self.lower\_limit + (i + 1) \* h) for i in range(n))  
  
 def middle\_rectangle(self, n):  
 h = (self.upper\_limit - self.lower\_limit) / n  
 return h \* sum(self.function(self.lower\_limit + (i + 0.5) \* h) for i in range(n))  
  
 def trapezoid(self, n):  
 h = (self.upper\_limit - self.lower\_limit) / n  
 return h \* (0.5 \* self.function(self.lower\_limit) + 0.5 \* self.function(self.upper\_limit) + sum(self.function(self.lower\_limit + i \* h) for i in range(1, n)))  
  
 def simpson(self, n):  
 h = (self.upper\_limit - self.lower\_limit) / n  
 result = self.function(self.lower\_limit) + self.function(self.upper\_limit)  
  
 odd\_sum = sum(self.function(self.lower\_limit + i \* h) for i in range(1, n, 2))  
 result += 4 \* odd\_sum  
  
 even\_sum = sum(self.function(self.lower\_limit + i \* h) for i in range(2, n - 1, 2))  
 result += 2 \* even\_sum  
  
 return (h / 3) \* result  
  
 def runge(self, I1, I2, p):  
 return abs(I1 - I2) / (2\*\*p - 1)  
  
 def calculate(self):  
 n = 2  
 max\_n = 1000000  
 max\_iterations = 1000  
 I\_old = 0  
 iteration = 0  
 while iteration < max\_iterations:  
 if self.method == 1:  
 I\_new = self.left\_rectangle(n)  
 elif self.method == 2:  
 I\_new = self.right\_rectangle(n)  
 elif self.method == 3:  
 I\_new = self.middle\_rectangle(n)  
 elif self.method == 4:  
 I\_new = self.trapezoid(n)  
 elif self.method == 5:  
 I\_new = self.simpson(n)  
  
 if iteration > 0:  
 p = 1 if self.method in [1, 2] else 2 if self.method == 3 else 4  
 error = self.runge(I\_old, I\_new, p)  
 if error < self.accuracy:  
 return I\_new, n  
  
 I\_old = I\_new  
 n \*= 2  
 iteration += 1  
  
 if n > max\_n:  
 return None  
  
 return I\_old, n

class ImproperIntegralCalculator:  
 def \_\_init\_\_(self, function, lower\_limit, upper\_limit):  
 self.function = function  
 self.lower\_limit = lower\_limit  
 self.upper\_limit = upper\_limit  
 self.check\_points = 10000  
 self.eps = 1e-6  
  
 def get\_breakpoints(self):  
 breakpoints = []  
  
 try:  
 self.function(self.lower\_limit)  
 except (ZeroDivisionError, ValueError, Exception):  
 breakpoints.append(self.lower\_limit)  
  
 try:  
 self.function(self.upper\_limit)  
 except (ZeroDivisionError, ValueError, Exception):  
 breakpoints.append(self.upper\_limit)  
  
 step = (self.upper\_limit - self.lower\_limit) / self.check\_points  
 for i in range(self.check\_points):  
 x = self.lower\_limit + i \* step  
 try:  
 self.function(self.lower\_limit + i \* step)  
 except (ZeroDivisionError, ValueError, Exception):  
 breakpoints.append(x)  
  
 return list(set(breakpoints))  
  
 def try\_to\_evaluate(self, x):  
 try:  
 return self.function(x)  
 except (ZeroDivisionError, ValueError, Exception):  
 return None  
  
 def get\_coverage(self):  
 coverage = True  
 for point in self.get\_breakpoints():  
 a = self.try\_to\_evaluate(point - self.eps)  
 b = self.try\_to\_evaluate(point + self.eps)  
 if a is not None and b is not None and abs(a - b) > self.eps or a == b and a is not None:  
 coverage = False  
 return coverage  
  
 def find\_limits(self):  
 limits = []  
 breakpoints = self.get\_breakpoints()  
 if len(breakpoints) == 1:  
 if breakpoints[0] == self.lower\_limit:  
 limits.append(self.lower\_limit + self.eps)  
 limits.append(self.upper\_limit)  
 elif breakpoints[0] == self.upper\_limit:  
 limits.append(self.lower\_limit)  
 limits.append(self.upper\_limit - self.eps)  
 return limits  
  
 if not (self.try\_to\_evaluate(self.lower\_limit) is None or self.try\_to\_evaluate(breakpoints[0] + self.eps) is None):  
 limits.append(self.lower\_limit)  
 limits.append(breakpoints[0] + self.eps)  
 return limits  
  
 if not (self.try\_to\_evaluate(self.upper\_limit) is None or self.try\_to\_evaluate(breakpoints[0] - self.eps) is None):  
 limits.append(self.upper\_limit)  
 limits.append(breakpoints[0] - self.eps)  
 return limits  
  
 for point in range(len(breakpoints) - 1):  
 cur = breakpoints[point]  
 next = breakpoints[point + 1]  
  
 if not (self.try\_to\_evaluate(cur + self.eps) is None or self.try\_to\_evaluate(next - self.eps) is None):  
 limits.append(cur + self.eps)  
 limits.append(next - self.eps)  
 return limits  
  
 if not breakpoints or self.lower\_limit - self.eps in breakpoints or self.upper\_limit + self.eps in breakpoints:  
 limits.append(self.lower\_limit)  
 limits.append(self.upper\_limit)  
  
 return limits

Примеры и результаты работы программы

**Пример 1:** Интеграл 2x³-4x²+6x-25 на интервале [0; 3] методом трапеций с точностью 0,01:

Результат интегрирования: -43.47802734375

Количество разбиений: 32

Точное значение: -43,5

**Пример 2:** Интеграл sin(x) на интервале [0; 2] методом левых прямоугольников с точностью 0,01:

Результат интегрирования: 1.4090141387016843

Количество разбиений: 128

Точное значение: 1,41

**Пример 3:** Интеграл e^(-x) на интервале [-3; 0] методом правых прямоугольников с точностью 0,01:

Результат интегрирования: 19.078548444008316

Количество разбиений: 4096

Точное значение: 19,08

**Пример 4:** Интеграл 1/x на интервале [-1; 0] методом центральных прямоугольников с точностью 0,01:

Результат интегрирования: Функция имеет разрывы

Интеграл не существует

Точное значение: Расходящийся

**Пример 5:** Интеграл 1/(√x) на интервале [0; 1] методом Симпсона с точностью 0,01:

Функция имеет разрывы

Интеграл сходится

Результат интегрирования: 2.061621013539304

Количество разбиений: 4096

Точное значение: 2

Вывод

В ходе выполнения лабораторной работы были изучены численные методы интегрирования с использованием Python. В результате работы были рассмотрены различные численные методы вычисления определенных интегралов: метод прямоугольников (левых, правых, средних), метод трапеций, метод Ньютона-Котеса и метод Симпсона.

Была реализована программа, позволяющая выбрать одну из предложенных функций, задать пределы интегрирования, точность и начальное значение числа разбиения интервала интегрирования.

В ходе вычислительной реализации задачи были рассчитаны интегралы различными методами и проведено сравнение результатов с точными значениями интегралов.