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:

$$\int_a^b f(x)\,dx \approx \int_a^b L_n(x)\,dx = \sum_{i=0}^{n} f(x_i) c_n^i$$

Метод левых прямоугольников:

$$\int_a^b f(x) = h\sum_{i=0}^{n-1} f(x_i)$$

Метод правых прямоугольников:

$$\int_a^b f(x) = h\sum_{i=0}^{n} f(x_i)$$

Метод средних прямоугольников:

$$\int_a^b f(x) = h\sum_{i=0}^{n} f\left(x_{i-\frac{1}{2}}\right)$$

Метод трапеций:

$$\int_a^b f(x) = h\left(\frac{y_0 - y_n}{2} + \sum_{i=0}^{n-1} y_i\right)$$

Метод Симпсона:

$$\int_a^b f(x) = \frac{h}{3}\left(y_0 + 4\sum_{i=1}^{n-1} y_{i\%2=0} + 2\sum_{i=2}^{n-2} y_{i\%2\neq 0} + y_n\right)$$

Вычислительная реализация задачи:

Вычислить интеграл: $\int_0^2 (-x^3 - x^2 - 2x + 1)\,dx$

Точное решение:

$$F(x) = -\frac{x^4}{4} - \frac{x^3}{3} - x^2 + x$$

$$-\frac{x^4}{4} - \frac{x^3}{3} - x^2 + x \Big|_0^2$$

$$-\frac{2^4}{4} - \frac{3^3}{3} - 3^2 + 2 - \left(-\frac{0^4}{4} - \frac{0^3}{3} - 0^2 + 0\right) = -\frac{26}{3} = -8,(6)$$

Получаем точное значение: $I_{\text{точн}} - 8,(6)$

Вычисление по формуле Ньютона – Котеса при $n = 6$:

$$h = \frac{b-a}{6} = \frac{2-0}{6} = \frac{1}{3}$$

$$\int_a^b f(x)\,dx \approx \int_a^b L_n(x)\,dx = \sum_{i=0}^{n} f(x_i)c_n^i = c_6^0 f(a) + c_6^1 f(a+h) + c_6^2 f(a+2h) + c_6^3 f(a+3h) + c_6^4 f(a+4h) + c_6^5 f(a+5h) + c_6^6 f(b)$$

Коэффициенты Котеса для i = 6:

$n = 0 = 6$	$n = 1 = 5$	$n = 2 = 4$	$n = 3$
$\frac{41(2-0)}{840} = \frac{41}{420}$	$\frac{216(2-0)}{840} = \frac{18}{35}$	$\frac{27(2-0)}{840} = \frac{13}{210}$	$\frac{272(2-0)}{840} = \frac{68}{105}$

Получаем:

$$\frac{41}{420}f(0) + \frac{18}{35}f\left(\frac{1}{3}\right) + \frac{13}{210}f\left(\frac{2}{3}\right) + \frac{68}{105}f(1) + \frac{13}{210}f\left(\frac{4}{3}\right) + \frac{18}{35}f\left(\frac{5}{3}\right) + \frac{41}{420}f(2) = -8,(6)$$

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

$$h = \frac{b-a}{10} = \frac{2-0}{10} = \frac{1}{5}$$

Через метод средних прямоугольников:

$$\int_a^b f(x) = h\sum_{i=0}^{n} f\left(x_{i-\frac{1}{2}}\right) = h \cdot \left(f\left(\frac{h}{2}\right) + f\left(\frac{3h}{2}\right) + f\left(\frac{5h}{2}\right) + f\left(\frac{7h}{2}\right) + f\left(\frac{9h}{2}\right) + f\left(\frac{11h}{2}\right) + f\left(\frac{13h}{2}\right) + f\left(\frac{15h}{2}\right) + f\left(\frac{17h}{2}\right) + f\left(\frac{19h}{2}\right)\right) = -8{,}64$$

Через метод трапеций:

$$\int_a^b f(x) = h\left(\frac{y_0 - y_n}{2} + \sum_{i=0}^{n-1} y_i\right) = h \cdot \left(\frac{f(0)+f(2)}{2}\sum_{i=0}^{n-1} y_i\right) = 0.2\left(\frac{f(0)+f(2)}{2} + f(0.2) + f(0.4) + f(0.6) + f(0.8) + f(1) + f(1.2) + f(1.4) + f(1.6) + f(1.8)\right) = -8{,}72$$

Через метод Симпсона:

$$\int_a^b f(x) = \frac{h}{3}\left(y_0 + 4\sum_{i=1}^{n-1} y_{i\%2=0} + 2\sum_{i=2}^{n-2} y_{i\%2\neq0} + y_n\right) = \frac{0.2}{3}\left(f(0) + 4 * \left(f(0.2) + f(0.6) + f(1) + f(1.4) + f(1.8)\right) + 2 * (f(0.4) + f(0.8) + f(1.2) + f(1.6)) + f(2)\right) = -8,(6)$$

Сравним полученные результаты.

Точное значение: $I_{\text{точн}} - 8,(6)$

По формуле Ньютона – Котеса при $n = 6$ получаем значение $I_{\text{котес}} - 8,(6)$, что совпадает с точным значением.

По формуле средних прямоугольников при $n = 10$ получаем значение $I_{\text{ср.пр.}} - 8{,}64$, получаем разницу: $|-8,(6) - (-8{,}64)| = 0{,}02(6)$.

По формуле трапеций при $n = 10$ получаем значение $I_{\text{трап}} - 8{,}72$, получаем разницу: $|-8,(6) - (-8{,}72)| = 0{,}05334$

По формуле Симпсона при $n = 10$ получаем значение $I_{\text{симпс}} - 8,(6)$, 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: $\frac{|-8,(6)-(-8{,}64)|}{|-8,(6)|} \approx 0{,}31\%$.

Для метода трапеций погрешность равна: $\frac{|-8,(6)-(-8{,}72)|}{|-8,(6)|} \approx 0{,}62\%$.

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
```

```python
        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 $2x^3-4x^2+6x-25$ 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.