Санкт-Петербургский Национальный Исследовательский

Университет ИТМО

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная №3

Вариант: 15

Выполнили:

Студентки группы P3266

Фурзикова Александра,

Савельева Елизавета

Преподаватель:

Машина Екатерина Алексеевна

г. Санкт-Петербург

2025 г.

Цель работы: найти приближенное значение определенного интеграла с требуе мой точностью различными численными методами.

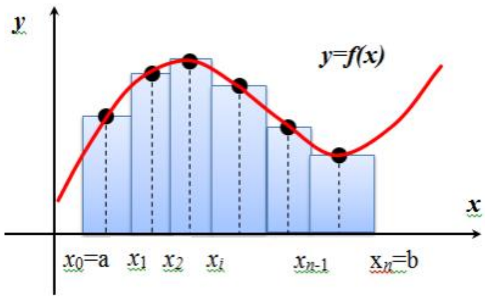
Порядок выполнения работы и рабочие формулы методов:

Метод прямоугольников. Метод средних

Для аналитически заданных функций более точным является использование значений в средних точках элементарных отрезков (полуцелых узлах):

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} h_i\, f(x_{i-1/2})$$

$$x_{i-1/2} = \frac{x_{i-1} + x_i}{2} = x_{i-1} + \frac{h_i}{2}, i = 1, 2, \dots n$$



При $h_i = h = \frac{b-a}{n} = const$:

$$\int_a^b f(x)dx = h\sum_{i=1}^{n} f(x_{i-1/2})$$

Метод трапеций

Подынтегральную функцию на каждом отрезке $[x_i; x_{i+1}]$ заменяют интерполяционным многочленом первой степени:

$$f(x) \approx \varphi_i(x) = a_i x + b$$

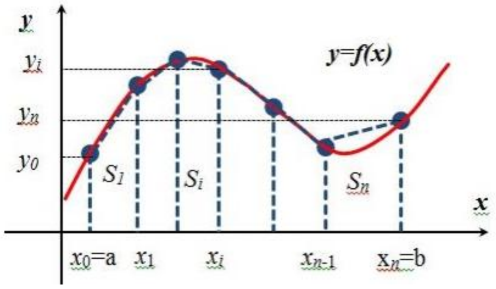
Используют линейную интерполяцию, т.е. график функции $y = f(x)$ представляется в виде ломаной, соединяющий точки (x_i, y_i) . Площадь всей фигуры (криволинейной трапеции):

$$S_{\text{общ}} = S_1 + S_2 + \dots + S_n = \frac{y_0 + y_1}{2}h_1 + \frac{y_1 + y_2}{2}h_2 + \dots + \frac{y_{n-1} + y_n}{2}h_n$$

$$y_0 = f(a), \quad y_n = f(b), \quad y_i = f(x_i), \quad h_i = x_i - x_{i-1}$$

Складывая все эти равенства, получаем формулу трапеций для численного интегрирования:

$$\int_a^b f(x)dx = \frac{1}{2}\sum_{i=1}^{n} h_i(y_{i-1} + y_i)$$



При $h_i = h = \frac{b-a}{n} = const$ формула трапеций:

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

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

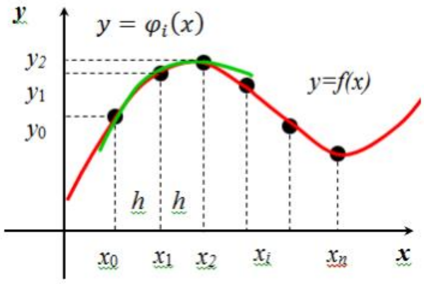
Метод Симпсона (**Симпсон Томас**(20.8.1710–14.5.1751) – английский математик)

Разобьем отрезок интегрирования [*a, b*] на четное число *n* равных частей с шагом *h.* На каждом отрезке $[x_0, x_2]$, $[x_2, x_4]$, …, $[x_{i-1}, x_{i+1}]$, … , $[x_{n-2}, x_n]$ подынтегральную функцию заменим интерполяционным многочленом второй степени:

$$f(x) \approx \varphi_i(x) = a_i x^2 + b_i x + c_i, \quad x_{i-1} \le x \le x_{i+1}$$

Коэффициенты этих квадратных трехчленов могут быть найдены из условий равенства многочлена и подынтегральной функции в узловых точках.

В качестве $\varphi_i(x)$ можно принять интерполяционный многочлен Лагранжа второй степени, проходящий через точки $(x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1})$.



Листинг программы:

```python
from NumericalIntegrations.Terminal import Terminal

if __name__ == "__main__":

    terminal = Terminal()
    while True:
        terminal.work()
        print("Хотите продолжить работу с программой? y/n")
        if not input().__eq__('y'):
            break
```

```python
from Validator import Validator

5 usages
class NumericalIntegration:
    AMOUNT_OF_COLUMNS = 3

    def __init__(self, epsilon, left_border, right_border, n):
        self.__right_border = right_border
        self.__left_border = left_border
        self.__n = n
        self.__epsilon = epsilon

    10 usages
    def getEpsilon(self):
        return self.__epsilon

    10 usages
    def getN(self):
        return self.__n

    5 usages
    def doubleN(self):
        self.__n *= 2

    10 usages
    def getLeftBorder(self):
        return self.__left_border

    10 usages
    def getRightBorder(self):
        return self.__right_border
```

```python
def ruleRunge(self, I_h, I_h2, k, eps):  #Правило Рунге
    delta = abs(I_h2 - I_h) / (2 ** k - 1)
    print(
        f'Погрешность вычислений между значениями {I_h2} и {I_h} составляет
{delta} {"<" if delta < eps else ">"} {eps}.')
    if delta <= eps:
        return True
    else:
        return False

def printTableForMethods(self, iterations, method):
    if method == 'middle':
        print('i  |  x(i-1/2)  |  y(i-1/2)  |')
    else:
        print('i  |  xi  |  yi  |')
    for i in range(len(iterations)):
        for j in range(len(iterations[i])):
            print(iterations[i][j], end=' | ')
        print()
```

```python
class Terminal:

    1 usage
    def work(self):
        try:
            print('\t\t\tЛабораторная работа №3\n\t\t\tЧисленное интегрирование')
            for i in range(len(FUNCTIONS)):
                print(f"{i + 1}. {FUNCTIONS[i + 1]['FUNCTION']}")
            function_number = self.enterFunctionNumber()
            print('\t\tМетоды решения:')
            for i in range(len(METHODS)):
                print(f'{i + 1}.{METHODS[i]}')
            method_number = self.enterFunctionMethod()
            a, b = self.enterBorders()
            epsilon = self.enterEpsilon()
            n = self.enterN()

            if method_number == 1:
                right_rectangles = MethodRectangles(epsilon, a, b, n)
                right_rectangles.iterateRightRectangles(number_of_function=function_number)
            elif method_number == 2:
                left_rectangles = MethodRectangles(epsilon, a, b, n)
                left_rectangles.iterateLeftRectangles(number_of_function=function_number)
            elif method_number == 3:
                middle_rectangles = MethodRectangles(epsilon, a, b, n)
                middle_rectangles.iterateMiddleRectangles(number_of_function=function_number)
            elif method_number == 4:
                trapezoidal_method = MethodTrapezoidal(epsilon, a, b, n)
                trapezoidal_method.iterateTrapezoidalMethod(number_of_function=function_number)
            else:
                sympson_method = MethodSympson(epsilon, a, b, n)
```

```python
                sympson_method.iterateSympsonlMethod(number_of_function=function_number)
        except IncorrectValueException as e:
            print(e.message)
            return

    2 usages
    def enterFunctionNumber(self):
        try:
            print('Введите номер функции:')
            return Validator.validateFunctionNumber()
        except IncorrectValueException as e:
            print(e.message)
            return self.enterFunctionNumber()

    2 usages
    def enterFunctionMethod(self):
        try:
            print('Введите номер метода:')
            return Validator.validateFunctionMethod()
        except IncorrectValueException as e:
            print(e.message)
            return self.enterFunctionMethod()

    2 usages
    def enterN(self):
        try:
            print('Введите начальное число отрезков для разбиения:')
            n = Validator.validateN()
            return n
        except IncorrectValueException as e:
            print(e.message)
            return self.enterN()

    2 usages
    def enterBorders(self):
```

```python
        try:
            print('Введите границы интервала a и b:')
            print('a = ', end='')
            a = Validator.validateNumber(input())
            print('b = ', end='')
            b = Validator.validateNumber(input())
            Validator.validateBorders(a, b)
            return a, b
        except IncorrectValueException as e:
            print(e.message)
            return self.enterBorders()

    2 usages
    def enterEpsilon(self):
        try:
            print('Введите точность epsilon:')
            eps = Validator.validateEpsilon(input())
            return eps
        except IncorrectValueException as e:
            print(e.message)
            return self.enterEpsilon()
```

```python
x = symbols('x')

METHODS = ['Метод правых прямоугольников',
           'Метод левых прямоугольников',
           'Метод средних прямоугольников',
           'Метод трапеций',
           'Метод Симпсона']

FUNCTIONS = {
    1:
        {
            'FUNCTION': 'f(x)=sin(x)+cos(x)'
        },
    2:
        {
            'FUNCTION': 'f(x)=19*x**5-67*x**2+100*x-191'
        },
    3:
        {
            'FUNCTION': 'f(x)=5**x+5*x'
        },
    4:
        {
            'FUNCTION': 'f(x)=(cos(x))**2'
        },
    5:
        {
            'FUNCTION': 'f(x)=x**2'
        }
}


7 usages
def calculateFunction(number_of_function, x: float) -> float:
    return ne.evaluate(FUNCTIONS[number_of_function]['FUNCTION'].split('=')[1], local_dict={'x': x})
```

```python
class ImproperIntegralSolver:
    """
    Класс для вычисления несобственных интегралов 2 рода
    с использованием различных численных методов
    """

    def __init__(self, func, interval, singularity_type):
        self.func = func
        self.a, self.b = interval
        self.singularity_type = singularity_type
        self.convergent = False

    2 usages
    def _check_convergence(self):
        """Проверка сходимости интеграла"""
        try:
            if self.singularity_type == 'a':
                test = (1e-12) ** 0.5 * self.func(self.a + 1e-12)
            elif self.singularity_type == 'b':
                test = (1e-12) ** 0.5 * self.func(self.b - 1e-12)
            else:
                mid = (self.a + self.b) / 2
                left = ImproperIntegralSolver(self.func, (self.a, mid), 'b')
                right = ImproperIntegralSolver(self.func, (mid, self.b), 'a')
                return left.check_convergence() and right.check_convergence()
            self.convergent = np.isfinite(test)
            return self.convergent
        except:
            self.convergent = False
            return False

    3 usages
    def integrate(self, method='simpson', n=100000):
        """
```

```python
        Основной метод вычисления интеграла
        :param method: метод интегрирования ('left_rect', 'mid_rect', 'right_rect', 'trapezoid', 'simpson')
        :param n: количество интервалов разбиения
        """
        if not self._check_convergence():
            print("Интеграл не существует")
            return None

        try:
            if self.singularity_type == 'internal':
                return self._handle_internal_singularity(method, n)

            if method in ['left_rect', 'mid_rect', 'right_rect']:
                return self._rectangle_method(method, n)
            elif method == 'trapezoid':
                return self._trapezoid_method(n)
            elif method == 'simpson':
                return self._simpson_method(n)
            else:
                raise ValueError("Неизвестный метод интегрирования")
        except Exception as e:
            print(f"Ошибка: {str(e)}")
            return None

    3 usages
    def _generate_partition(self, n):
        """Генерация разбиения с учётом особенности"""
        if self.singularity_type == 'a':
            return np.linspace(self.a + 1e-12, self.b, n + 1)
        elif self.singularity_type == 'b':
            return np.linspace(self.a, self.b - 1e-12, n + 1)
        else:
            mid = (self.a + self.b) / 2
            return np.concatenate([
                np.linspace(self.a, mid - 1e-12, n // 2 + 1),
```

```python
                np.linspace(self.a, mid - 1e-12, n // 2 + 1),
                np.linspace(mid + 1e-12, self.b, n // 2 + 1)[1:]
            ])

    1 usage
    def _rectangle_method(self, method, n):
        """Метод прямоугольников"""
        x = self._generate_partition(n)
        h = (x[-1] - x[0]) / n

        if method == 'left_rect':
            points = x[:-1]
        elif method == 'right_rect':
            points = x[1:]
        else:  # mid_rect
            points = (x[:-1] + x[1:]) / 2

        y = self.func(points)
        return h * np.sum(y)

    1 usage
    def _trapezoid_method(self, n):
        """Метод трапеций (исправленная версия)"""
        x = self._generate_partition(n)
        y = self.func(x)
        return np.trapezoid(y, x)  # Исправлено здесь

    1 usage
    def _simpson_method(self, n):
        """Метод Симпсона"""
        if n % 2 != 0:
            n += 1  # Делаем четное количество интервалов
        x = self._generate_partition(n)
        y = self.func(x)
        h = (x[-1] - x[0]) / n
```

```python
        return h / 3 * (y[0] + 4 * np.sum(y[1:-1:2]) + 2 * np.sum(y[2:-2:2]) + y[-1])          ▲3 ▲11 ✓58 ^

    1 usage
    def _handle_internal_singularity(self, method, n):
        """Обработка внутренней особенности"""
        c = (self.a + self.b) / 2
        left = ImproperIntegralSolver(self.func, interval: (self.a, c), singularity_type: 'b')
        right = ImproperIntegralSolver(self.func, interval: (c, self.b), singularity_type: 'a')
        return left.integrate(method, n // 2) + right.integrate(method, n // 2)


# Примеры использования
if __name__ == "__main__":
    # Тестовые интегралы
    tests = [
        ("∫1/√x dx от 0 до 1", lambda x: 1 / np.sqrt(x), (0, 1), 'a'),
        ("∫1/√(1-x) dx от 0 до 1", lambda x: 1 / np.sqrt(1 - x), (0, 1), 'b'),
        ("∫1/|x-0.5|^(1/3) dx от 0 до 1", lambda x: 1 / np.abs(x - 0.5) ** (1 / 3), (0, 1), 'internal'),
        ("∫1/x dx от 0 до 1 (расходится)", lambda x: 1 / x, (0, 1), 'a')
    ]

    methods = ['left_rect', 'mid_rect', 'right_rect', 'trapezoid', 'simpson']

    for desc, func, interval, stype in tests:
        print(f"\n{desc}")
        solver = ImproperIntegralSolver(func, interval, stype)

        for method in methods:
            result = solver.integrate(method, n=100000)
            if result is not None:
                print(f"{method:10} → {result:.6f}")

        # Проверка расходимости
        if not solver._check_convergence():
            print("Результат: Интеграл не существует")
```

```python
class Validator:
    MIN_EPSILON = 0

    1 usage
    @staticmethod
    def validateEpsilon(epsilon: str):
        try:
            epsilon = float(epsilon)
            if epsilon >= Validator.MIN_EPSILON:
                return epsilon
            else:
                raise IncorrectValueException(f'Точность (эпсилон) должна быть больше или равна 0.')
        except ValueError:
            raise IncorrectValueException(f'Точность (эпсилон) - число с плавающей точкой.')

    5 usages
    @staticmethod
    def validateNumber(number: str):
        try:
            number = float(number)
            return number
        except ValueError:
            raise IncorrectValueException(f'Необходимо ввести число.')

    1 usage
    @staticmethod
    def validateFunctionNumber():
        equation_number = int(Validator.validateNumber(input()))
        if 0 < equation_number <= len(FUNCTIONS):
            return equation_number
        else:
            raise IncorrectValueException('Неверное введен номер функции. Попробуйте еще раз.')
```

```python
    @staticmethod
    def validateFunctionMethod():
        equation_method = int(Validator.validateNumber(input()))
        if 0 < equation_method <= len(METHODS):
            return equation_method
        else:
            raise IncorrectValueException('Неверное введен номер метода. Попробуйте еще раз.')

    1 usage
    @staticmethod
    def validateN():
        n = int(Validator.validateNumber(input()))
        if n > 3:
            return n
        else:
            raise IncorrectValueException('Неверное введено значение числа разбиения интервала интегрирования.

    1 usage
    @staticmethod
    def validateBorders(border_left: float, border_right: float):
        if border_left < border_right:
            return True
        else:
            raise IncorrectValueException(f'Левая граница должна быть строго меньше правой.')
```

```python
class IncorrectValueException(Exception):

    def __init__(self, message):
        self.message = message
```

```python
K = 2                                                                    ⚠19 ✓48


4 usages
class MethodRectangles(NumericalIntegration):

    1 usage
    def iterateRightRectangles(self, number_of_function):
        eps = self.getEpsilon()
        iterations = 0
        while True:
            print(f'Номер итерации №{iterations}')
            square_1, n_1 = self.rightRectangles(number_of_function=number_of_function, n=self.getN())
            self.doubleN()
            square_2, n_2 = self.rightRectangles(number_of_function=number_of_function, n=self.getN())
            if self.ruleRunge(square_1, square_2, k: 2, self.getEpsilon()):
                print(f"Решение интеграла {FUNCTIONS[number_of_function]['FUNCTION']} "
                      f"с границами a={self.getLeftBorder()} и b={self.getRightBorder()}:\n\t\t\t "
                      f"{square_2} \nдля разбиения на {n_2} отрезков.")
                break
            iterations += 1

    2 usages
    def rightRectangles(self, number_of_function, n):
        a = self.getLeftBorder()
        b = self.getRightBorder()
        h = (b - a) / n
        iterations = [[0.0 for x in range(self.AMOUNT_OF_COLUMNS)] for i in range(n + 1)]
        square = 0
        for i in range(n + 1):
            iterations[i][0] = i
            if i == 0:
                iterations[i][1] = a
            else:
                iterations[i][1] = a + h
```

```python
            iterations[i][2] = calculateFunction(number_of_function, iterations[i][1])      ⚠19 ✓48
            if i != 0:
                square += iterations[i][2]
            a = iterations[i][1]
        square *= h
        print(f'\nМетод правых прямоугольников. Численное интегрирование для n={n} и h={h}')
        self.printTableForMethods(iterations, method='right')
        print(f'Найденный ответ:{square} для {n} отрезков разбиения.')
        return square, n

    1 usage
    def iterateLeftRectangles(self, number_of_function):
        eps = self.getEpsilon()
        iterations = 0
        while True:
            print(f'Номер итерации №{iterations}')
            square_1, n_1 = self.leftRectangles(number_of_function=number_of_function, n=self.getN())
            self.doubleN()
            square_2, n_2 = self.leftRectangles(number_of_function=number_of_function, n=self.getN())
            if self.ruleRunge(square_1, square_2, k: 2, self.getEpsilon()):
                print(f"Решение интеграла {FUNCTIONS[number_of_function]['FUNCTION']} "
                      f"с границами a={self.getLeftBorder()} и b={self.getRightBorder()}:\n\t\t\t "
                      f"{square_2} \nдля разбиения на {n_2} отрезков.")
                break
            iterations += 1

    2 usages
    def leftRectangles(self, number_of_function, n):
        a = self.getLeftBorder()
        b = self.getRightBorder()
        h = (b - a) / n
        iterations = [[0.0 for x in range(self.AMOUNT_OF_COLUMNS)] for i in range(n + 1)]
        square = 0
        for i in range(1, n + 2):
            if i == 1:
```

```python
                iterations[i - 1][1] = a
            else:
                iterations[i - 1][1] = a + h
            iterations[i - 1][0] = i - 1
            iterations[i - 1][2] = calculateFunction(number_of_function, iterations[i - 1][1])
            if i != n + 1:
                square += iterations[i - 1][2]
            a = iterations[i - 1][1]
        square *= h
        print(f'\nМетод левых прямоугольников. Численное интегрирование для n={n} и h={h}')
        self.printTableForMethods(iterations, method='left')
        print(f'Найденный ответ:{square} для {n} отрезков разбиения.')
        return square, n

    1 usage
    def iterateMiddleRectangles(self, number_of_function):
        eps = self.getEpsilon()
        iterations = 0
        while True:
            print(f'Номер итерации №{iterations}')
            square_1, n_1 = self.middleRectangles(number_of_function=number_of_function, n=self.getN())
            self.doubleN()
            square_2, n_2 = self.middleRectangles(number_of_function=number_of_function, n=self.getN())
            if self.ruleRunge(square_1, square_2, k: 2, self.getEpsilon()):
                print(f"Решение интеграла {FUNCTIONS[number_of_function]['FUNCTION']} "
                      f"с границами a={self.getLeftBorder()} и b={self.getRightBorder()}:\n\t\t\t "
                      f"{square_2} \nдля разбиения на {n_2} отрезков.")
                break
            iterations += 1

    2 usages
    def middleRectangles(self, number_of_function, n):
        a = self.getLeftBorder()
        b = self.getRightBorder()
        h = (b - a) / n
```

```python
        iterations = [[0.0 for x in range(self.AMOUNT_OF_COLUMNS)] for i in range(n + 1)]
        square = 0
        for i in range(1, n + 1):
            iterations[i][0] = i
            if i == 1:
                iterations[i][1] = (a + a + h) / 2
            else:
                iterations[i][1] = a + h
            iterations[i][2] = calculateFunction(number_of_function, iterations[i][1])
            square += iterations[i][2]
            a = iterations[i][1]
        square *= h
        print(f'\nМетод средних прямоугольников. Численное интегрирование для n={n} и h={h}')
        self.printTableForMethods(iterations, method='middle')
        print(f'Найденный ответ:{square} для {n} отрезков разбиения.')
        return square, n
```

```python
class MethodSympson(NumericalIntegration):

    def iterateSympsonlMethod(self, number_of_function):
        eps = self.getEpsilon()
        iterations = 0
        while True:
            print(f'Номер итерации №{iterations}')
            square_1, n_1 = self.sympsonMethod(number_of_function=number_of_function, n=self.getN())
            self.doubleN()
            square_2, n_2 = self.sympsonMethod(number_of_function=number_of_function, n=self.getN())
            if self.ruleRunge(square_1, square_2, 2, self.getEpsilon()):
                print(f"Решение интеграла {FUNCTIONS[number_of_function]['FUNCTION']} "
                      f"с границами a={self.getLeftBorder()} и b={self.getRightBorder()}:\n\t\t\t "
                      f"{square_2} \nдля разбиения на {n_2} отрезков.")
                break
            iterations += 1

    def sympsonMethod(self, number_of_function, n):
        a = self.getLeftBorder()
        b = self.getRightBorder()
        h = (b - a) / n
        iterations = [[0.0 for x in range(self.AMOUNT_OF_COLUMNS)] for i in range(n + 1)]
        square_even = 0
        square_odd = 0
        for i in range(n + 1):
            iterations[i][0] = i
            iterations[i][1] = a if i == 0 else a + h
            iterations[i][2] = calculateFunction(number_of_function, iterations[i][1])
            if i != 0 and i != n:
                if i % 2 == 0:
                    square_even += iterations[i][2]
                else:
                    square_odd += iterations[i][2]
```

```python
            a = iterations[i][1]
        square = h / 3 * (iterations[0][2] + iterations[-1][2] + 2 * square_even + 4 * square_odd)
        print(f'\nМетод Симпсона. Численное интегрирование для n={n} и h={h}')
        self.printTableForMethods(iterations, method='trapezoidal')
        print(f'Найденный ответ:{square} для {n} отрезков разбиения.')
        return square, n
```

```python
class MethodTrapezoidal(NumericalIntegration):

    def iterateTrapezoidalMethod(self, number_of_function):
        eps = self.getEpsilon()
        iterations = 0
        while True:
            print(f'Номер итерации №{iterations}')
            square_1, n_1 = self.trapezoidalMethod(number_of_function=number_of_function, n=self.getN())
            self.doubleN()
            square_2, n_2 = self.trapezoidalMethod(number_of_function=number_of_function, n=self.getN())
            if self.ruleRunge(square_1, square_2, 2, self.getEpsilon()):
                print(f"Решение интеграла {FUNCTIONS[number_of_function]['FUNCTION']} "
                      f"с границами a={self.getLeftBorder()} и b={self.getRightBorder()}:\n\t\t\t "
                      f"{square_2} \nдля разбиения на {n_2} отрезков.")
                break
            iterations += 1

    def trapezoidalMethod(self, number_of_function, n):
        a = self.getLeftBorder()
        b = self.getRightBorder()
        h = (b - a) / n
        iterations = [[0.0 for x in range(self.AMOUNT_OF_COLUMNS)] for i in range(n + 1)]
        square = 0
        for i in range(n + 1):
            iterations[i][0] = i
            if i == 0:
                iterations[i][1] = a
            else:
                iterations[i][1] = a + h
            iterations[i][2] = calculateFunction(number_of_function, iterations[i][1])
            if i != 0 and i != n:
                square += iterations[i][2]
```

```
        square = h * ((iterations[0][2] + iterations[-1][2]) / 2 + square)
        print(f'\nМетод трапеций. Численное интегрирование для n={n} и h={h}')
        self.printTableForMethods(iterations, method='trapezoidal')
        print(f'Найденный ответ:{square} для {n} отрезков разбиения.')
        return square, n
```

Результаты выполнения программой:

```
            Лабораторная работа №3
            Численное интегрирование
1. f(x)=sin(x)+cos(x)
2. f(x)=19*x**5-67*x**2+100*x-191
3. f(x)=5**x+5*x
4. f(x)=(cos(x))**2
5. f(x)=x**2
Введите номер функции:
1
        Методы решения:
1.Метод правых прямоугольников
2.Метод левых прямоугольников
3.Метод средних прямоугольников
4.Метод трапеций
5.Метод Симпсона
Введите номер метода:
2
Введите границы интервала a и b:
a = 2
b = 3
Введите точность epsilon:
```

```
Введите точность epsilon:
0.01
Введите начальное число отрезков для разбиения:
4
Номер итерации №0

Метод левых прямоугольников. Численное интегрирование для n=4 и h=0.25
i  |  xi  |  yi  |
0 | 2.0 | 0.4931505902785393 |
1 | 2.25 | 0.14989957416518207 |
2 | 2.5 | -0.20267147144297726 |
3 | 2.75 | -0.5426413865801318 |
4 | 3.0 | -0.8488724885405782 |
Найденный ответ:-0.025565673394846916 для 4 отрезков разбиения.

Метод левых прямоугольников. Численное интегрирование для n=8 и h=0.125
i  |  xi  |  yi  |
0 | 2.0 | 0.4931505902785393 |
1 | 2.125 | 0.32405345511414696 |
2 | 2.25 | 0.14989957416518207 |
3 | 2.375 | -0.02659343950341997 |
4 | 2.5 | -0.20267147144297726 |
```

```
4 | 2.5 | -0.20267147144297726 |
5 | 2.625 | -0.3755868828558952 |
6 | 2.75 | -0.5426413865801318 |
7 | 2.875 | -0.7012281529578954 |
8 | 3.0 | -0.8488724885405782 |
Найденный ответ:-0.1102022142228064 для 8 отрезков разбиения.
Погрешность вычислений между значениями -0.1102022142228064 и -0.025565673394846916 составляет 0.028212180275986497 > 0.01.
Номер итерации №1

Метод левых прямоугольников. Численное интегрирование для n=8 и h=0.125
i  |  xi  |  yi  |
0 | 2.0 | 0.4931505902785393 |
1 | 2.125 | 0.32405345511414696 |
2 | 2.25 | 0.14989957416518207 |
3 | 2.375 | -0.02659343950341997 |
4 | 2.5 | -0.20267147144297726 |
5 | 2.625 | -0.3755868828558952 |
6 | 2.75 | -0.5426413865801318 |
7 | 2.875 | -0.7012281529578954 |
8 | 3.0 | -0.8488724885405782 |
Найденный ответ:-0.1102022142228064 для 8 отрезков разбиения.
```

```
Найденный ответ:-0.1102022142228064 для 8 отрезков разбиения.

Метод левых прямоугольников. Численное интегрирование для n=16 и h=0.0625
i  |  xi  |  yi  |
0 | 2.0 | 0.4931505902785393 |
1 | 2.0625 | 0.409401374499418 |
2 | 2.125 | 0.32405345511414696 |
3 | 2.1875 | 0.23744011392146347 |
4 | 2.25 | 0.14989957416518207 |
5 | 2.3125 | 0.06177367977966641 |
6 | 2.375 | -0.02659343950341997 |
7 | 2.4375 | -0.11485671197435021 |
8 | 2.5 | -0.20267147144297726 |
9 | 2.5625 | -0.2896948031528289 |
10 | 2.625 | -0.3755868828558952 |
11 | 2.6875 | -0.4600123038190429 |
12 | 2.75 | -0.5426413865801318 |
13 | 2.8125 | -0.6231514663392586 |
14 | 2.875 | -0.7012281529578954 |
15 | 2.9375 | -0.7765665586456534 |
16 | 3.0 | -0.8488724885405782 |
Найденный ответ:-0.15233027434456486 для 16 отрезков разбиения.
```

```
Погрешность вычислений между значениями -0.15233027434456486 и -0.1102022142228064 составляет 0.01404268670725282 > 0.01.
Номер итерации №2

Метод левых прямоугольников. Численное интегрирование для n=16 и h=0.0625
i  |  xi  |  yi  |
0 | 2.0 | 0.4931505902785393 |
1 | 2.0625 | 0.409401374499418 |
2 | 2.125 | 0.32405345511414696 |
3 | 2.1875 | 0.23744011392146347 |
4 | 2.25 | 0.14989957416518207 |
5 | 2.3125 | 0.06177367977966641 |
6 | 2.375 | -0.02659343950341997 |
7 | 2.4375 | -0.11485671197435021 |
8 | 2.5 | -0.20267147144297726 |
9 | 2.5625 | -0.2896948031528289 |
10 | 2.625 | -0.3755868828558952 |
11 | 2.6875 | -0.4600123038190429 |
12 | 2.75 | -0.5426413865801318 |
13 | 2.8125 | -0.6231514663392586 |
14 | 2.875 | -0.7012281529578954 |
15 | 2.9375 | -0.7765665586456534 |
16 | 3.0 | -0.8488724885405782 |
```

```
Метод левых прямоугольников. Численное интегрирование для n=32 и h=0.03125
i  |  xi  |  yi  |
0 | 2.0 | 0.4931505902785393 |
1 | 2.03125 | 0.4514964216858581 |
2 | 2.0625 | 0.409401374499418 |
3 | 2.09375 | 0.3669065538184322 |
4 | 2.125 | 0.32405345511414696 |
5 | 2.15625 | 0.2808839237102183 |
6 | 2.1875 | 0.23744011392146347 |
7 | 2.21875 | 0.19376444789088776 |
8 | 2.25 | 0.14989957416518207 |
9 | 2.28125 | 0.10588832604914089 |
10 | 2.3125 | 0.06177367977966641 |
11 | 2.34375 | 0.01759871256020329 |
12 | 2.375 | -0.02659343950341997 |
13 | 2.40625 | -0.07075962352466603 |
14 | 2.4375 | -0.11485671197435021 |
15 | 2.46875 | -0.15884164479394136 |
16 | 2.5 | -0.20267147144297726 |
17 | 2.53125 | -0.24630339283953917 |
18 | 2.5625 | -0.2896948031528289 |
19 | 2.59375 | -0.3328033314070399 |
```

```
19 | 2.59375 | -0.3328033314070399 |
20 | 2.625 | -0.3755868828558952 |
21 | 2.65625 | -0.41800368008745353 |
22 | 2.6875 | -0.4600123038190429 |
23 | 2.71875 | -0.5015717333424873 |
24 | 2.75 | -0.5426413865801318 |
25 | 2.78125 | -0.5831811597125531 |
26 | 2.8125 | -0.6231514663392586 |
27 | 2.84375 | -0.6625132761341341 |
28 | 2.875 | -0.7012281529578954 |
29 | 2.90625 | -0.7392582923903266 |
30 | 2.9375 | -0.7765665586456534 |
31 | 2.96875 | -0.8131165208350088 |
32 | 3.0 | -0.8488724885405782 |
Найденный ответ:-0.1733468330895452 для 32 отрезков разбиения.
Погрешность вычислений между значениями -0.1733468330895452 и -0.15233027434456486 составляет 0.007005519581660115 < 0.01.
Решение интеграла f(x)=sin(x)+cos(x) с границами a=2.0 и b=3.0:
            -0.1733468330895452
для разбиения на 32 отрезков.
Хотите продолжить работу с программой? y/n
```

Вычисление заданного интеграла:

Вычислительная реализация

Вариант 15

$$\int_1^2 (5x^3 - 2x^2 + 3x - 15)\,dx$$

① Точное вычисление интеграла

$$F(x) = \frac{5}{4}x^4 - \frac{2}{3}x^3 + \frac{3}{2}x^2 - 15x + C$$

$$F(2) = \frac{5}{4}\cdot 16 - \frac{2}{3}\cdot 8 + \frac{3}{2}\cdot 4 - 15\cdot 2 = 20 - \frac{16}{3} + 6 - 30 = -\frac{28}{3}$$

$$F(1) = \frac{5}{4} - \frac{2}{3} + \frac{3}{2} - 15 = -\frac{155}{12}$$

$$\int_1^2 (5x^3 - 2x^2 + 3x - 15)\,dx = F(2) - F(1) = -\frac{28}{3} + \frac{155}{12} = \frac{43}{12} \approx 3{,}5833$$

② По формуле Ньютона – Котеса

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

$$C_6^0 = C_6^6 = \frac{41}{840} \cdot \frac{1}{\cancel{1}} = \frac{41}{840}$$

$$C_6^1 = C_6^5 = \frac{216}{840}$$

$$C_6^2 = C_6^4 = \frac{27}{840}$$

$$C_6^3 = \frac{272}{840}$$

$f(1) = -9$

$f\left(\frac{7}{6}\right) \approx -\cancel{8{,}186}\ 6{,}282$

$f\left(\frac{8}{6}\right) = -2{,}703$

$f\left(\frac{9}{6}\right) = 1{,}875$

$f\left(\frac{10}{6}\right) = 7{,}592$

$f\left(\frac{11}{6}\right) = 14{,}588$

$f(2) = 23$

$$\int_1^2 (5x^3 - 2x^2 + 3x - 15)\,dx = \sum_{i=0}^{6} f(x_i)C_6^i = \frac{1}{840}\left(41f_0 + 216f_1 + 27f_2 + 272f_3 + \right.$$

$$\left. + 27f_4 + 216f_5 + 41f_6\right) = \frac{1}{840}\left(41\cdot(-9) + 216\cdot(-6{,}282) + 27\cdot(-2{,}703) + 272\cdot(1{,}875) \right.$$

$$\left. + 27\cdot 7{,}592 + 216\cdot 14{,}588 + 41\cdot 23\right) \approx 3{,}5833$$

③ 1) По формуле средних прямоугольников

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

$$\int_1^2 f(x)\,dx = h\sum_i f(x_{i-\frac{1}{2}}), \quad x_{i-\frac{1}{2}} = x_{i-1} + \frac{h_i}{2}$$

$f_0 = f(1{,}05) = -8{,}267$

$f_1 = f(1{,}15) = -6{,}591$

$f_2 = f(1{,}25) = -4{,}609$

$f_3 = f(1{,}35) = -2{,}293$

$f_4 = f(1{,}45) = 0{,}388$

$f_5 = f(1{,}55) = 3{,}464$

$f_6 = f(1{,}65) = 6{,}966$

$f_7 = f(1{,}75) = 10{,}922$

$f_8 = f(1{,}85) = 15{,}363$

$f_9 = f(1{,}85) = 20{,}319$

~~$f_{10} = f(2)$~~

$$h\sum_{i=0}^{10} f(x_{i-\frac{1}{2}}) = \frac{1}{10}(-6,591 - 4,609 - 2,223 + 0,388 + 3,464 + 6,966 +$$

$$+10,922 + 15,363 + 20,319) \approx 3,5832$$

2) Метод трапеций

$h = 0,1$

$$I = h\left(\frac{f(1)+f(2)}{2} + \sum_{i=1}^{9} f(x_i)\right)$$

i	0	1	2	3	4	5	6	7	8	9
x_i	1	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
y_i	−9	−7,465	−5,64	−3,495	−1	1,875	5,16	8,885	13,08	17,775

$$I = 0,1\left(\frac{-9+23}{2} + (-7,465 - 5,64 - 3,495 - 1 + 1,875 + 5,16 + 8,885 + 13,08 + 17,775)\right)$$

$$= 3,5833$$

3) Метод Симпсона

$h = 0,1$

$$I = \frac{h}{3}\left[f(x_0) + 4\sum_{неч} f(x_i) + 2\sum_{чет} f(x_i) + f(x_n)\right]$$

$$\sum_{неч} f(x_i) = -7,465 - 3,495 + 1,875 + 8,885 + 17,775 = 17,575$$

$$\sum_{чет} f(x_i) = -5,64 - 1 + 5,16 + 13,08 = 11,6$$

Подставим

$$I = \frac{0,1}{3}\left[-9 + 4 \cdot 17,575 + 2 \cdot 11,6 - 23\right] = 3,5833$$

Все использованные методы численного интегрирования (Ньютона – Котеса, средних прямоугольников, трапеций, Симпсона) при n=6n=6 и n=10n=10 дали точный результат для данного интеграла. Это связано с тем, что подынтегральная функция является полиномом третьей степени, а методы Симпсона и Ньютона – Котеса точны для полиномов до третьей степени включительно. Методы прямоугольников и трапеций также показали высокую точность при достаточно мелком разбиении.

Вывод: В ходе лабораторной работы были освоены основные методы численного интегрирования и отработаны навыки их программной реализации. Полученные результаты подтвердили эффективность численных подходов для вычисления определённых интегралов, особенно когда аналитическое решение затруднено. Работа продемонстрировала важность выбора подходящего метода и контроля точности вычислений при решении практических задач.