Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04

Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №3

Вариант 11

Выполнил: Савельева Диана Александровна

P32082

Преподаватель: Машина Екатерина Алексеевна

Санкт-Петербург, 2023 г.

***1. Цель:*** найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

***2. Порядок выполнения работы:***

*Исходные данные:*

1. Пользователь выбирает функцию, интеграл которой требуется вычислить (3-5 функций), из тех, которые предлагает программа.

2. Пределы интегрирования задаются пользователем.

3. Точность вычисления задается пользователем.

4. Начальное значение числа разбиения интервала интегрирования: n=4.

5. Ввод исходных данных осуществляется с клавиатуры.

*Программная реализация задачи:*

1. Реализовать в программе методы по выбору пользователя:

• Метод прямоугольников (3 модификации: левые, правые, средние)

• Метод трапеций

• Метод Симпсона

2. Методы должны быть оформлены в виде отдельной(ого) функции/класса.

3. Вычисление значений функции оформить в виде отдельной(ого) функции/класса.

4. Для оценки погрешности и завершения вычислительного процесса использовать правило Рунге.

5. Предусмотреть вывод результатов: значение интеграла, число разбиения интервала интегрирования для достижения требуемой точности.

*Вычислительная реализация задачи:*

1. Вычислить интеграл, приведенный в таблице 1, точно.

2. Вычислить интеграл по формуле Ньютона – Котеса при n=5.

3. Вычислить интеграл по формулам средних прямоугольников, трапеций и Симпсона при n=10.

4. Сравнить результаты с точным значением интеграла.

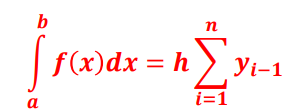
5. Определить относительную погрешность вычислений для каждого метода.

6. В отчете отразить последовательные вычисления.

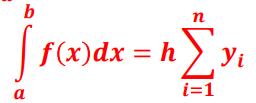
***3. Рабочие формулы методов:***

* *Метод прямоугольников (3 модификации: левых, правых, средних)*

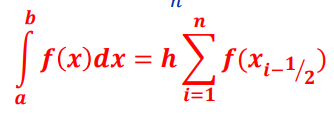
Левые:



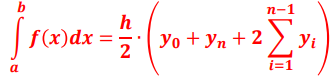
Правые:



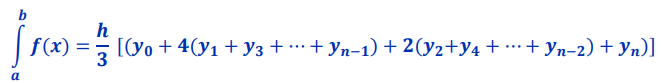
Средние:



* *Метод трапеций*

**

* *Метод Симпсона*

**

***4. Листинг программы:***

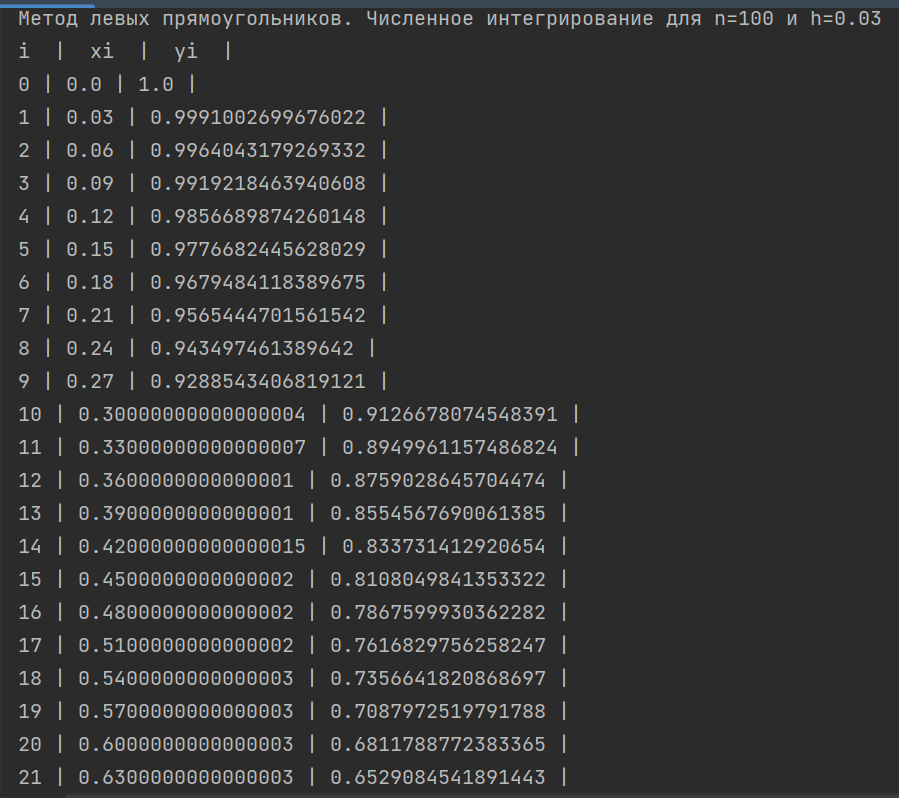
from Functions import \*  
from NumericalIntegration import \*  
  
  
K = 2  
  
  
class MethodRectangles(NumericalIntegration):  
  
 def iterateRightRectangles(self, number\_of\_function):  
 eps = self.getEpsilon()  
 iterations = 0  
 while True:  
 print(f'Номер итерации №{iterations}')  
 square\_1, 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, 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 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  
 iterations[i][2] = calculateFunction(number\_of\_function, iterations[i][1])  
 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  
  
 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, 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 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:  
 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  
  
 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, 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 middleRectangles(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 + 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

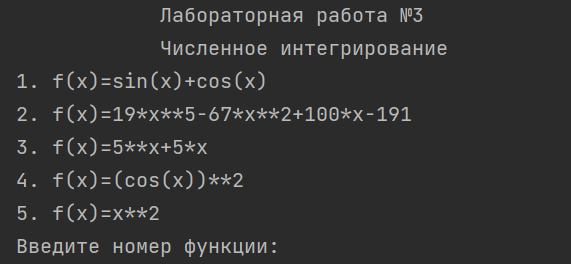
from Functions import calculateFunction, FUNCTIONS  
from NumericalIntegration import NumericalIntegration  
  
  
class MethodSympson(NumericalIntegration):  
  
 def iterateSympsonlMethod(self, number\_of\_function):  
 eps = self.getEpsilon()  
 iterations = 0  
 while True:  
 print(f'Номер итерации №{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]  
 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

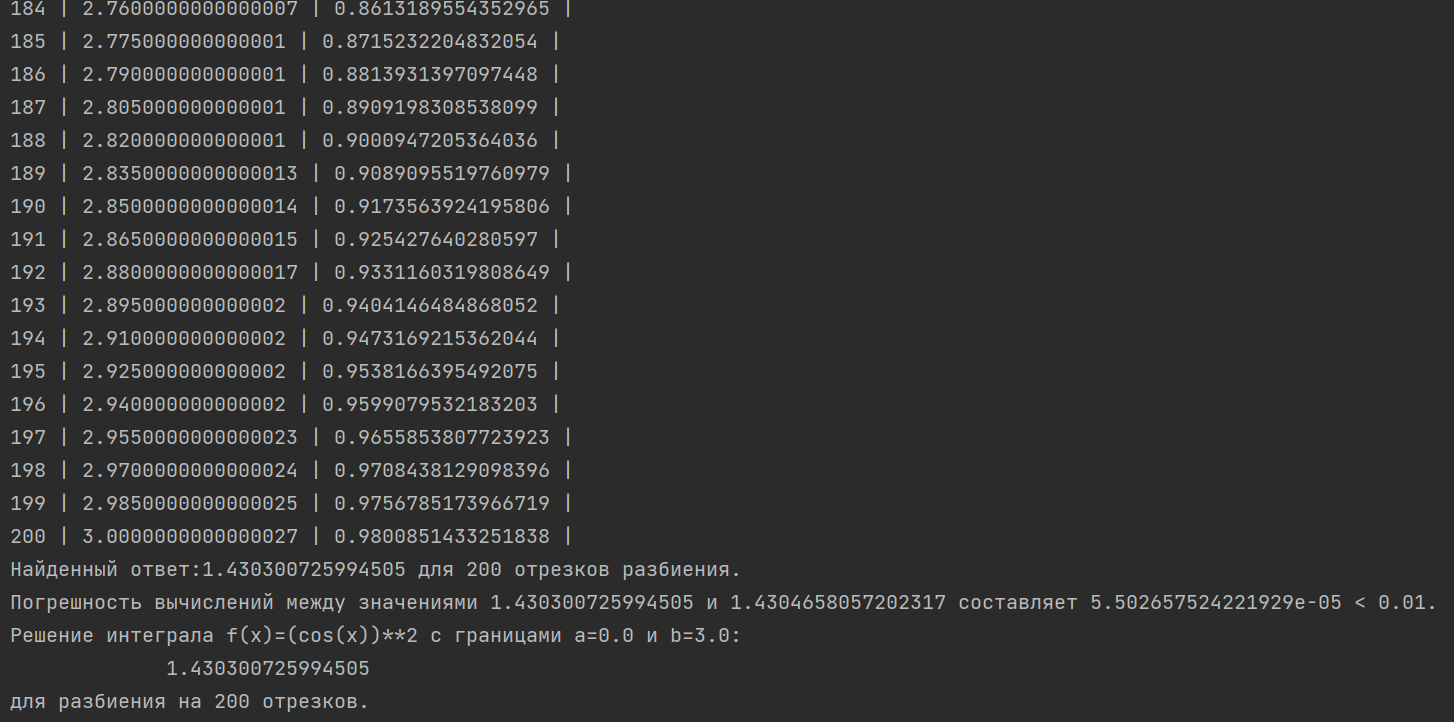
from Functions import calculateFunction, FUNCTIONS  
from NumericalIntegration import NumericalIntegration  
  
  
class MethodTrapezoidal(NumericalIntegration):  
  
 def iterateTrapezoidalMethod(self, number\_of\_function):  
 eps = self.getEpsilon()  
 iterations = 0  
 while True:  
 print(f'Номер 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]  
 a = iterations[i][1]  
 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

from Validator import Validator  
  
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  
  
 def getEpsilon(self):  
 return self.\_\_epsilon  
  
 def getN(self):  
 return self.\_\_n  
  
 def doubleN(self):  
 self.\_\_n \*= 2  
  
 def getLeftBorder(self):  
 return self.\_\_left\_border  
  
 def getRightBorder(self):  
 return self.\_\_right\_border  
  
 def ruleRunge(self, I\_h, I\_h2, k, eps):  
 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()

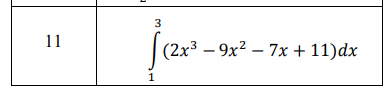
***5. Результаты выполнения программы:***







***6. Вычислительная часть:***

******

***Задание:***

*1. Вычислить интеграл, приведенный в таблице 1, точно.*

*2. Вычислить интеграл по формуле Ньютона – Котеса при n=5.*

Пусть функция f(x) = задана в шести точках a= 1, x1=1.4, x2 =1.8, x3 = 2.2, x4=2.6, b=3.

* (b-a) = 3-1 = 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *a* | *1* | *f(a)* | *-3* | *c0;5* | *0,131944444* |
| *x1* | *1,4* | *f(x1)* | *-10,952* | *c1;5* | *0,520833333* |
| *x2* | *1,8* | *f(x2)* | *-19,096* | *c2;5* | *0,347222222* |
| *x3* | *2,2* | *f(x3)* | *-26,664* | *c3;5* | *0,347222222* |
| *x4* | *2,6* | *f(x4)* | *-32,888* | *c4;5* | *0,520833333* |
| *b* | *3* | *f(b)* | *-37* | *c5;5* | *0,131944444* |

*3. Вычислить интеграл по формулам средних прямоугольников, трапеций и Симпсона при n=10.*

**Формула средних прямоугольников:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
| *xi* | *1* | *1,2* | *1,4* | *1,6* | *1,8* | *2* | *2,2* | *2,4* | *2,6* | *2,8* | *3* |
| *yi* | *-3* | *-6,904* | *-10,952* | *-15,048* | *-19,096* | *-23* | *-26,66* | *-29,99* | *-32,89* | *-35,26* | *-37* |
| *xi-1/2* |  | *1,1* | *1,3* | *1,5* | *1,7* | *1,9* | *2,1* | *2,3* | *2,5* | *2,7* | *2,9* |
| *yi-1/2* |  |  | *-4,928* | *-8,916* | *-13* | *-17,084* | *-21,072* | *-24,87* | *-28,38* | *-31,5* | *-34,14* |

* h=(b-a)/n = (3-1)/10=0,2.
* ΔIср.=I - =|-44-(-44,02) |=0,02
* Относительная погрешность: Δ = ΔIср./ I = |0,02/-44| \*100% = 0,05%

**Формула трапеций:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
| *xi* | *1* | *1,2* | *1,4* | *1,6* | *1,8* | *2* | *2,2* | *2,4* | *2,6* | *2,8* | *3* |
| *yi* | *-3* | *-6,904* | *-10,952* | *-15,048* | *-19,096* | *-23* | *-26,66* | *-29,99* | *-32,89* | *-35,26* | *-37* |

* h=(b-a)/n = (3-1)/10=0,2.
* ΔIтр.=I - =|-44-(-43,96) |=0,04
* Относительная погрешность: Δ = ΔIтр./ I = |0,04/-44| \*100% = 0,091%

**Формула Симпсона:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
| *xi* | *1* | *1,2* | *1,4* | *1,6* | *1,8* | *2* | *2,2* | *2,4* | *2,6* | *2,8* | *3* |
| *yi* | *-3* | *-6,904* | *-10,952* | *-15,048* | *-19,096* | *-23* | *-26,66* | *-29,99* | *-32,89* | *-35,26* | *-37* |

* h=(b-a)/n = (3-1)/10=0,2.
* ΔIСимпсон.=I - =|-44-(-44) |=0
* Относительная погрешность: Δ = ΔIСимпсон./ I = |0/-44| \*100% = 0%

***7. Выводы:*** я научилась находить приближенное значение определенного интеграла с требуемой точностью различными численными методами.