Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

**­­­­­Лабораторная работа №3**

**по дисциплине «Вычислительная математика»**

**«Численное интегрирование»**

Вариант №3

Группа: P3212

Выполнил: Балин А. А.

Проверила: Наумова Н. А.

# Цель работы

Найти приближённое значение определённого интеграла с требуемой точностью различными численными методами.

# Вычислительная реализация

## Решение интеграла аналитически

## Решение интеграла по формуле Ньютона-Котеса при n=6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  | 0 |  |  |  |  |  |  |
|  | 3 |  |  | 2 |  |  | -7 |
|  |  |  |  |  |  |  |  |

Относительная погрешность:

## Решение интеграла методом средних прямоугольников

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 | 1.1 | 1.3 | 1.5 | 1.7 | 1.9 |
|  | 3.089 | 3.183 | 3.125 | 2.867 | 2.361 | 1.559 | 0.413 | -1.125 | -3.103 | -5.569 |

## Решение интеграла методом трапеций

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 0 | 0.2 | 0.4 | 0.6 | 0.8 | 1 | 1.2 | 1.4 | 1.6 | 1.8 | 2 |
|  | 3,000 | 3,152 | 3,176 | 3,024 | 2,648 | 2,000 | 1,032 | -0,304 | -2,056 | -4,272 | -7,000 |

## Решение интеграла методом Симпсона

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 0,0 | 0,2 | 0,4 | 0,6 | 0,8 | 1,0 | 1,2 | 1,4 | 1,6 | 1,8 | 2,0 |
|  | 3,000 | 3,152 | 3,176 | 3,024 | 2,648 | 2,000 | 1,032 | -0,304 | -2,056 | -4,272 | -7,000 |

## Вывод по вычислительной части

В случае с моим вариантом интеграла, наиболее точный и равный действительному значению интеграла результат дали формулы Ньютона-Котеса и Симпсона, другие методы, однако, так же показали хороший результат.

# Программная часть

## Реализация численных методов интегрирования

from numpy import sin,pi, sqrt, exp, cos, nan,divide

from lab3.Answer import Answer

**class** Integrals:

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

    f1 = **lambda** x: (x-1)\*\*2

    f2 = **lambda** x: sin(pi\*x)/2

    f3 = **lambda** x: (sqrt(x))/(exp(x)-1)

    f4 = **lambda** x: cos(pi\*x)-exp(sin(pi\*x))+1

    f5 = **lambda** x: divide(divide(1,(x-1)),(x-2))

    functions = [f1, f2, f3, f4, f5]

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**def** check\_converging(self,risks):

        miss = 0

        for i in range(len(risks)-1):

            if abs(risks[i])<abs(risks[i+1]):

                miss+=1

            else:

                miss=0

            if miss>3:

                raise ValueError("The integral is not converging")

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**def** integrating(self,mode,depth=0):

        previous\_int = 10\*\*10

        darbu\_sums = 0

        current\_partions = self.partions

        risks = []

        while abs(previous\_int-darbu\_sums) > self.eps/2 or current\_partions==self.partions\*2:

            previous\_int = darbu\_sums

            darbu\_sums = 0

            h = (self.b - self.a)/current\_partions

            for i in range(current\_partions):

                 darbu\_sums += mode(h,i)

            darbu\_sums \*= h

            current\_partions \*= 2

*#checking converging*

            if(current\_partions>self.partions\*2):

                risks.append(darbu\_sums-previous\_int)

            if len(risks)%10 == 0 and len(risks)!=0:

                self.check\_converging(risks)

                risks.clear()

            if current\_partions>2\*\*20:

                darbu\_sums = nan

                break

        if (str(darbu\_sums) == str(nan)) and depth==0:

            self.partions \*= 2

            return self.integrating(mode,1)

        elif (str(darbu\_sums) == str(nan)) and depth==1:

            self.a = self.a+self.eps\*\*2

            return self.integrating(mode,2)

        elif (str(darbu\_sums) == str(nan)) and depth==2:

            self.b = self.b-self.eps\*\*2

            return self.integrating(mode,3)

        elif (str(darbu\_sums) == str(nan)) and depth==3 and current\_partions<=2\*\*20 or "inf" in str(darbu\_sums) or "inf" in str(previous\_int):

            raise ValueError("The integral is not converging or the function is not defined in the given interval")

        elif current\_partions>2\*\*20 and depth==3:

            raise ValueError("Computation time exceeded, try to use another method or decrease the interval length")

        else:

            return Answer(darbu\_sums,current\_partions//2)

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**def** rectangle(self,mode):

        return self.integrating(**lambda** h,step: self.current\_function(self.a + h\*mode(step)))

**def** rectangle\_rights(self):

        return self.rectangle(**lambda** step: step+1)

**def** rectangle\_lefts(self):

        return self.rectangle(**lambda** step: step)

**def** rectangle\_middles(self):

        return self.rectangle(**lambda** step: step+0.5)

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**def** trapezoid(self):

        return self.integrating(**lambda** h,step: (self.current\_function(self.a + h\*step)+self.current\_function(self.a + h\*(step+1)))/2)

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**def** simpson(self):

        return self.integrating(**lambda** h,step: (self.current\_function(self.a + h\*step) +

                                             4\*self.current\_function(self.a + h\*(step+0.5)) +

                                             self.current\_function(self.a + h\*(step+1))) / 6)

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

    methods = [rectangle\_lefts, rectangle\_rights, rectangle\_middles, trapezoid, simpson]

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**def** \_\_init\_\_(self, a, b, equathion, method, eps):

        self.a = a

        self.b = b

        self.current\_function = self.functions[equathion-1]

        self.eps = eps

        self.partions = 4 *#const starting number of partions*

        self.method = self.methods[method-1]

**def** solve(self):

        return self.method(self)

Изображение выглядит как снимок экрана, диаграмма, черно-белый, дизайн

Автоматически созданное описание

Рисунок 1. Диаграмма

## Пример работы программы

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2. Пример работы программы.

## Репозиторий с исходниками

https://github.com/ta4ilka69/docs\_for\_labs/tree/main/Вычмат

# Вывод

В ходе реализации данной лабораторной работы я ознакомился с численными методами решения определённых интегралов.