Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A picture containing logo

Description automatically generated

Лабораторная работа №4 по дисциплине  
«Вычислительная математика»

" Численное интегрирование"

Выполнил: Дау Конг Туан Ань

Группа: P32151

Преподаватель: Машина Е.А

г. Санкт-Петербург

2023

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

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 .

4. Сравнить результаты с точным значением интеграла.

5. Определить относительную погрешность вычислений для каждого метода.

6. В отчете отразить последовательные вычисления.

Text, letter

Description automatically generated

Text, timeline

Description automatically generated

N = 5, a = 1, b = 2 , h = 0.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 1.2 | 1.4 | 1.6 | 1.8 | 2.0 |
|  | -5 | -3.864 | -2.392 | -0.488 | 1.944 | 5 |

*3(30%)*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *1* | *1.1* | *1.2* | *1.3* | *1.4* | *1.5* | *1.6* | *1.7* | *1.8* | *1.9* | *2.0* |
|  | *-5* | *-4.468* | *-3.864* | *-3.176* | *-2.392* | *-1.5* | *-0.488* | *0.656* | *1.944* | *3.388* | *5* |

*Chart, line chart

Description automatically generated*

*a = 1, b = 2, n = 10 , h = 0.1*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *1* | *1.1* | *1.2* | *1.3* | *1.4* | *1.5* | *1.6* | *1.7* | *1.8* | *1.9* | *2.0* |
|  | *-5* | *-4.468* | *-3.864* | *-3.176* | *-2.392* | *-1.5* | *-0.488* | *0.656* | *1.944* | *3.388* | *5* |

*Error = 0.001(0.1%)*

*Timeline

Description automatically generated*

*a = 1, b = 2, n = 10 , h = 0.1*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *1* | *1.1* | *1.2* | *1.3* | *1.4* | *1.5* | *1.6* | *1.7* | *1.8* | *1.9* | *2.0* |
|  | *-5* | *-4.468* | *-3.864* | *-3.176* | *-2.392* | *-1.5* | *-0.488* | *0.656* | *1.944* | *3.388* | *5* |

*Chart, line chart

Description automatically generated*

*a = 1, b = 2, n = 10 , h = 0.1*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *1.05* | *1.15* | *1.25* | *1.35* | *1.45* | *1.55* | *1.65* | *1.75* | *1.85* | *1.95* |
|  | *-4.742* | *-4.176* | *-3.531* | *-2.797* | *-1.96* | *-1.01* | *0.067* | *1.281* | *2.646* | *4.172* |

**Программная реализация**

public class RectangleMethod\_Center implements Method {  
 private String name = "Rectangle method - Center";  
 @Override  
 public Result execute(Equation equation,  
 Double left,  
 Double right,  
 Integer n,  
 Double accuracy) {  
 Integer numOfStep = numOfStep(equation, left, right ,accuracy);  
 return new Result(  
 calculate(equation, left, right, n),  
 numOfStep,  
 error(equation, left, right, n));  
 }  
  
 @Override  
 public Double error(Equation equation, Double left, Double right, Integer n) {  
 Double new\_intergral = calculate(equation, left, right, n);  
 Double old\_intergral = calculate(equation, left, right, n \* 2);  
 return Math.*abs*((new\_intergral - old\_intergral) / 3);  
 }  
  
 @Override  
 public Integer numOfStep(Equation equation,Double left, Double right, Double Accuracy) {  
 Integer res = 1;  
 Double new\_intergral = calculate(equation, left, right, res);  
 Double old\_intergral = calculate(equation, left, right, res \* 2);  
 while(Math.*abs*(new\_intergral - old\_intergral) >= Accuracy && res <= 10000000) {  
 res \*= 2;  
 new\_intergral = calculate(equation, left, right, res);  
 old\_intergral = calculate(equation, left, right, res \* 2);  
 }  
  
 return res \* 2;  
 }  
  
 @Override  
 public Double calculate(Equation equation, Double left, Double right, Integer n) {  
 Double h = (right- left) / n;  
 Double res = 0.0;  
 for(int i = 0; i < n; ++i) {  
 res += equation.valueAt(left + i \* h + h/2) \* h;  
 }  
  
 return res;  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
}

public class RectangleMethod\_Left implements Method {  
 private String name = "Rectangle method - Left";  
 @Override  
 public Result execute(Equation equation,  
 Double left,  
 Double right,  
 Integer n,  
 Double accuracy) {  
 Integer numOfStep = numOfStep(equation, left, right ,accuracy);  
 return new Result(  
 calculate(equation, left, right, n),  
 numOfStep,  
 error(equation, left, right, n));  
 }  
  
 @Override  
 public Double error(Equation equation, Double left, Double right, Integer n) {  
 Double new\_intergral = calculate(equation, left, right, n);  
 Double old\_intergral = calculate(equation, left, right, n \* 2);  
 return Math.*abs*((new\_intergral - old\_intergral) / 3);  
 }  
  
 @Override  
 public Integer numOfStep(Equation equation,Double left, Double right, Double Accuracy) {  
 Integer res = 1;  
 Double new\_intergral = calculate(equation, left, right, res);  
 Double old\_intergral = calculate(equation, left, right, res \* 2);  
 while(Math.*abs*(new\_intergral - old\_intergral) >= Accuracy && res <= 10000000) {  
 res \*= 2;  
 new\_intergral = calculate(equation, left, right, res);  
 old\_intergral = calculate(equation, left, right, res \* 2);  
 }  
  
 return res \* 2;  
 }  
  
 @Override  
 public Double calculate(Equation equation, Double left, Double right, Integer n) {  
 Double h = (right- left) / n;  
 Double res = 0.0;  
 for(int i = 0; i < n; ++i) {  
 res += equation.valueAt(left + i \* h) \* h;  
 }  
  
 return res;  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
}

public class RectangleMethod\_Right implements Method {  
 private String name = "Rectangle method - Right";  
 @Override  
 public Result execute(Equation equation,  
 Double left,  
 Double right,  
 Integer n,  
 Double accuracy) {  
 Integer numOfStep = numOfStep(equation, left, right ,accuracy);  
 return new Result(  
 calculate(equation, left, right, n),  
 numOfStep,  
 error(equation, left, right, n));  
 }  
  
 @Override  
 public Double error(Equation equation, Double left, Double right, Integer n) {  
 Double new\_intergral = calculate(equation, left, right, n);  
 Double old\_intergral = calculate(equation, left, right, n \* 2);  
 return Math.*abs*((new\_intergral - old\_intergral) / 3);  
 }  
  
 @Override  
 public Integer numOfStep(Equation equation,Double left, Double right, Double Accuracy) {  
 Integer res = 1;  
 Double new\_intergral = calculate(equation, left, right, res);  
 Double old\_intergral = calculate(equation, left, right, res \* 2);  
 while(Math.*abs*(new\_intergral - old\_intergral) >= Accuracy && res <= 10000000) {  
 res \*= 2;  
 new\_intergral = calculate(equation, left, right, res);  
 old\_intergral = calculate(equation, left, right, res \* 2);  
 }  
  
 return res \* 2;  
 }  
  
 @Override  
 public Double calculate(Equation equation, Double left, Double right, Integer n) {  
 Double h = (right- left) / n;  
 Double res = 0.0;  
 for(int i = 1; i <= n; ++i) {  
 res += equation.valueAt(left + i \* h) \* h;  
 }  
  
 return res;  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
}

public class SimpsonMethod implements Method {  
 private String name = "Simpson Method";  
 @Override  
 public Result execute(Equation equation,  
 Double left,  
 Double right,  
 Integer n,  
 Double accuracy) {  
 Integer numOfStep = numOfStep(equation, left, right ,accuracy);  
 return new Result(  
 calculate(equation, left, right, n),  
 numOfStep,  
 error(equation, left, right, n));  
 }  
  
 @Override  
 public Double error(Equation equation, Double left, Double right, Integer n) {  
 Double new\_intergral = calculate(equation, left, right, n);  
 Double old\_intergral = calculate(equation, left, right, n \* 2);  
 return Math.*abs*((new\_intergral - old\_intergral) / 3);  
 }  
  
 @Override  
 public Integer numOfStep(Equation equation,Double left, Double right, Double Accuracy) {  
 Integer res = 1;  
 Double new\_intergral = calculate(equation, left, right, res);  
 Double old\_intergral = calculate(equation, left, right, res \* 2);  
 while(Math.*abs*(new\_intergral - old\_intergral) >= Accuracy && res <= 10000000) {  
 res \*= 2;  
 new\_intergral = calculate(equation, left, right, res);  
 old\_intergral = calculate(equation, left, right, res \* 2);  
 }  
  
 return res \* 2;  
 }  
  
 @Override  
 public Double calculate(Equation equation, Double left, Double right, Integer n) {  
 Double h = (right- left) / n;  
 Double res = (equation.valueAt(left) + equation.valueAt(right)) / 3;  
 for(int i = 1; i < n; i+= 2) {  
 res += 4\*equation.valueAt(left + i \* h) \* h / 3;  
 }  
 for(int i = 2; i < n; i+= 2) {  
 res += 2\*equation.valueAt(left + i \* h) \* h / 3;  
 }  
 return res;  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
}

public class TrapezoidMethod implements Method {  
 private String name = "Trapezoid Method";  
 @Override  
 public Result execute(Equation equation,  
 Double left,  
 Double right,  
 Integer n,  
 Double accuracy) {  
 Integer numOfStep = numOfStep(equation, left, right ,accuracy);  
 return new Result(  
 calculate(equation, left, right, n),  
 numOfStep,  
 error(equation, left, right, n));  
 }  
  
 @Override  
 public Double error(Equation equation, Double left, Double right, Integer n) {  
 Double new\_intergral = calculate(equation, left, right, n);  
 Double old\_intergral = calculate(equation, left, right, n \* 2);  
 return Math.*abs*((new\_intergral - old\_intergral) / 7);  
 }  
 @Override  
 public Integer numOfStep(Equation equation,Double left, Double right, Double Accuracy) {  
 Integer res = 1;  
 Double new\_intergral = calculate(equation, left, right, res);  
 Double old\_intergral = calculate(equation, left, right, res \* 2);  
 while(Math.*abs*(new\_intergral - old\_intergral) >= Accuracy && res <= 10000000) {  
 res \*= 2;  
 new\_intergral = calculate(equation, left, right, res);  
 old\_intergral = calculate(equation, left, right, res \* 2);  
 }  
  
 return res \* 2;  
 }  
  
 @Override  
 public Double calculate(Equation equation, Double left, Double right, Integer n) {  
 Double h = (right- left) / n;  
 Double res = (equation.valueAt(left) + equation.valueAt(right)) / 2;  
 for(int i = 1; i < n; ++i) {  
 res += equation.valueAt(left + i \* h) \* h;  
 }  
  
 return res;  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
}

***result***

0 : 2X^3 - 3X^2 + 5X - 9

Please choose one equation:

0

List of Method:

0 : Rectangle method - Center

1 : Rectangle method - Left

2 : Rectangle method - Right

3 : Simpson Method

4 : Trapezoid Method

Please choose one method:

2

Please type left border

2

Please type right border:

3

Please type n:

10000

Please type accuracy:

0.01

Result{Result = 17.00140001999994, Accuracy = 2048, Error = 2.3333833331543966E-4}

**Вывод**

В ходе данной лабораторной работы были изучены несколько методов для численного интегрирования. Все методы просты в программной реализации и быстро вычисляют интегралы с хорошей точностью.