МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Лабораторная работа № 4
Численное интегрирование функций
Вариант №15

# ОТЧЕТ

по лабораторной работе

по дисциплине

Вычислительная математика


РУКОВОДИТЕЛЬ:

________________ Суркова А.С.

СТУДЕНТ:

________________ Сапожников В.О.

19-ИВТ-3


Работа защищена «___» _____________

С оценкой __________________________

Нижний Новгород 2021

## Содержание

# 1. Цель

Закрепление знаний и умений по численному интегрированию функций.

## 2. Постановка задачи

Вычислить интеграл по формулам центральных (средних) прямоугольников, трапеций и формуле Симпсона, при n=8 и n=20; оценить погрешность результата.

15. $\int_{0,4}^{1,2} \frac{\cos\left(x^2\right)}{x+1}dx$

## 3. Теоретические сведения

### 3.1. Метод средних (центральных) прямоугольников.

**Метод прямоугольников** — метод численного интегрирования функции одной переменной, заключающийся в замене подынтегральной функции на многочлен нулевой степени, то есть константу, на каждом элементарном отрезке. Если рассмотреть график подынтегральной функции, то метод будет заключаться в приближённом вычислении площади под графиком суммированием площадей конечного числа прямоугольников, ширина которых будет определяться расстоянием между соответствующими соседними узлами интегрирования, а высота — значением подынтегральной функции в этих узлах.

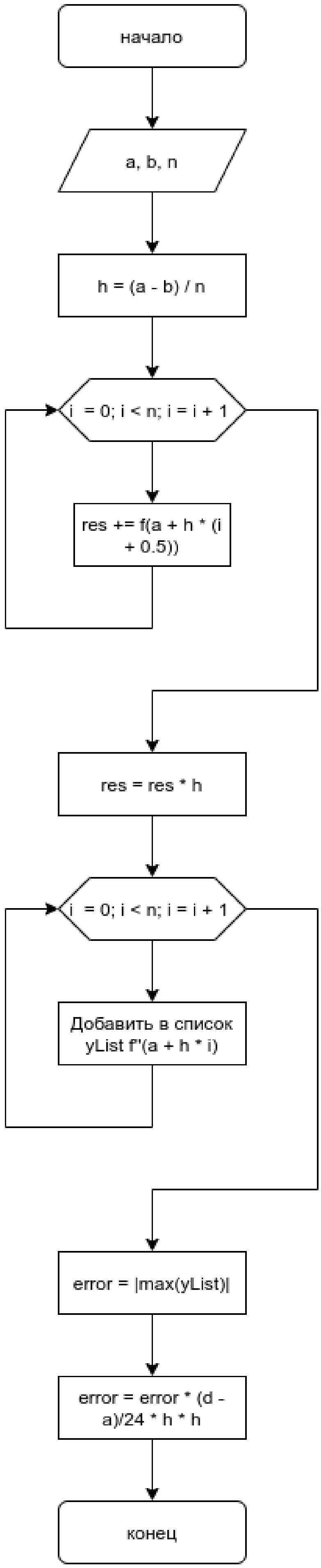
Составная квадратурная формула для метода средних (центральных) прямоугольников.

$$h=\frac{b-a}{n}$$

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

Погрешность формулы интегрирования метода средних (центральных) прямоугольников:

$$|R_n|\le\frac{(b-a)}{24}h^2 max|f''(x)|$$


начало
a, b, n
h = (a - b) / n
i  = 0; i < n; i = i + 1
res += f(a + h * (i + 0.5))
res = res * h
i  = 0; i < n; i = i + 1
Добавить в список yList f''(a + h * i)
error = |max(yList)|
error = error * (d - a)/24 * h * h
конец

### 3.2. Метод трапеций

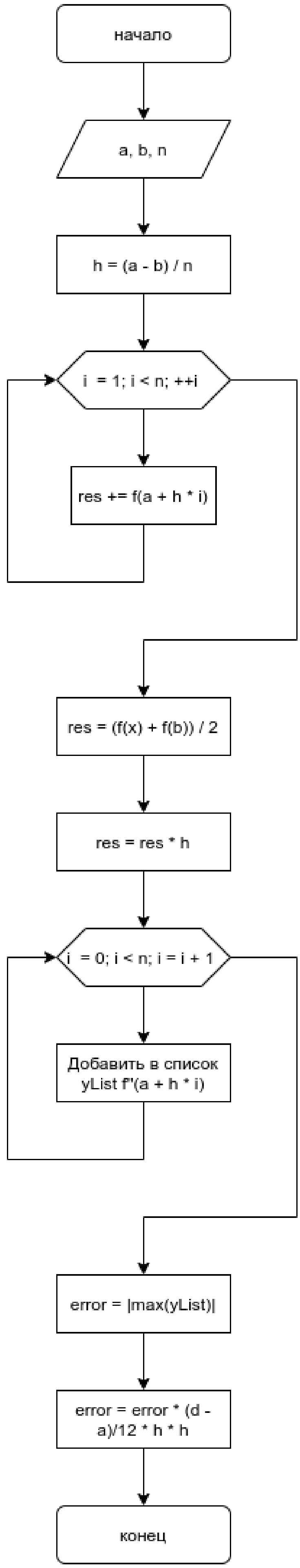
**Метод трапеций** — метод численного интегрирования функции одной переменной, заключающийся в замене на каждом элементарном отрезке подынтегральной функции на многочлен первой степени, то есть линейную функцию. Площадь под графиком функции аппроксимируется прямоугольными трапециями.

$$h=\frac{b-a}{n}$$

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

Погрешность формулы интегрирования метода средних (центральных) прямоугольников:

$$|R_n|\le\frac{(b-a)}{12}h^2 max|f''(x)|$$


начало
a, b, n
h = (a - b) / n
i  = 1; i < n; ++i
res += f(a + h * i)
res = (f(x) + f(b)) / 2
res = res * h
i  = 0; i < n; i = i + 1
Добавить в список
yList f''(a + h * i)
error = |max(yList)|
error = error * (d -
a)/12 * h * h
конец

## 3.3. Метод Симсона

**Формула Симпсона** (также **Ньютона-Симпсона**) относится к приёмам численного интегрирования. Получила название в честь британского математика Томаса Симпсона (1710—1761).

Суть метода заключается в приближении подынтегральной функции на отрезке $[a;b]$ интерполяционным многочленом второй степени $p_2(x)$, то есть приближение графика функции на отрезке параболой.

$$h=\frac{b-a}{n}$$

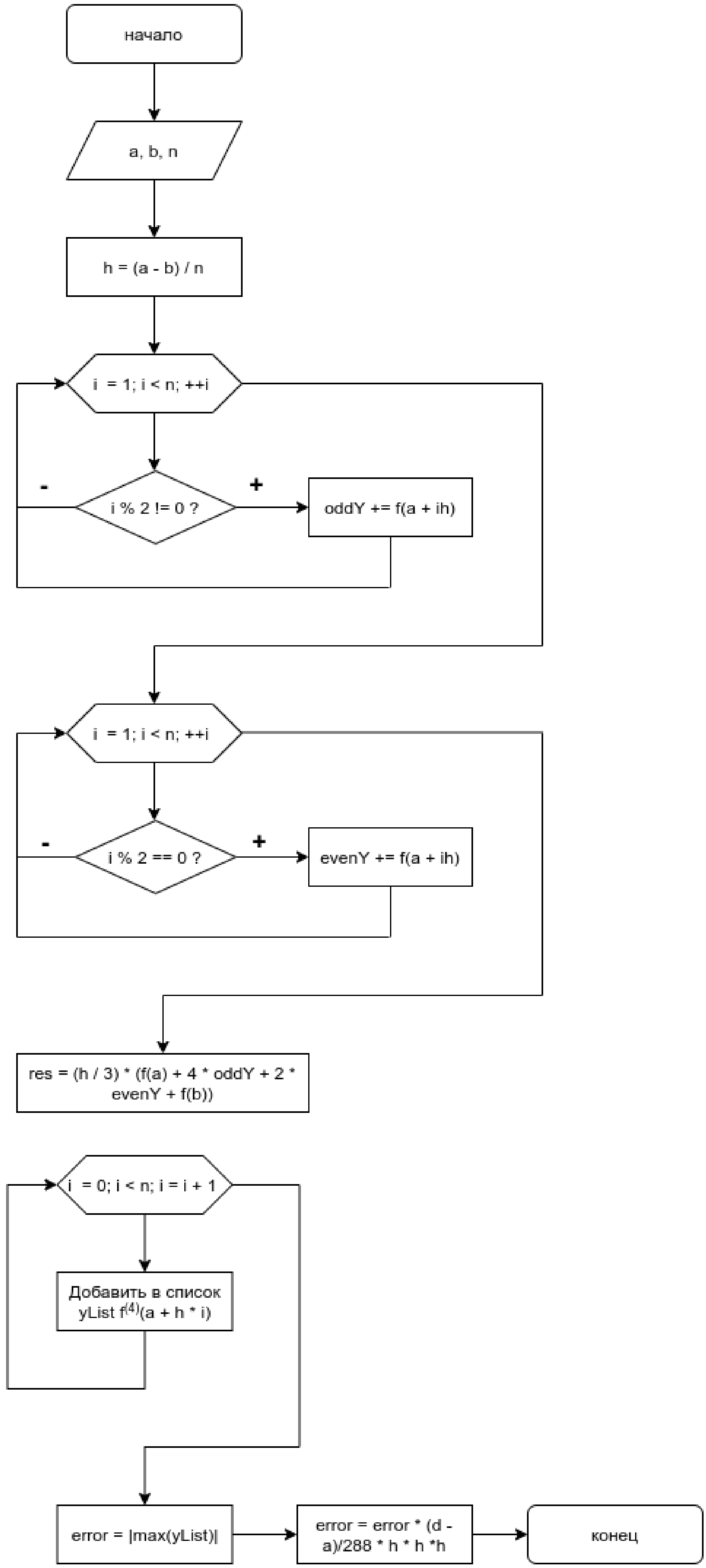
$$\int_a^b f(x)dx=\frac{h}{3}\left(y_0+4\left(y_1+y_3+\ldots+y_{n-1}\right)+2\left(y_2+y_4+\ldots+y_{n-2}\right)+y_n\right)$$

Погрешность формулы интегрирования метода Симпсона:

$$\left|R_n\right|\leq\frac{(b-a)}{2880}h^4 max\left|f^{(4)}(x)\right|$$

Погрешность формулы интегрирования метода Симпсона при невозможности ввода производной четвертого порядка:

$$\left|R_n\right|\leq\frac{(b-a)}{288}h^3 max\left|f^{(3)}(x)\right|$$


начало
a, b, n
h = (a - b) / n
i  = 1; i < n; ++i
-
i % 2 != 0 ?
+
oddY += f(a + ih)
i  = 1; i < n; ++i
-
i % 2 == 0 ?
+
evenY += f(a + ih)
res = (h / 3) * (f(a) + 4 * oddY + 2 * evenY + f(b))
i  = 0; i < n; i = i + 1
Добавить в список yList f(4)(a + h * i)
error = |max(yList)|
error = error * (d - a)/288 * h * h *h
конец

## 4. Расчетные данные

Исходная функция:

$$\int_{0,4}^{1,2} \frac{\cos\left(x^2\right)}{x+1}\,dx$$

Вторая производная:

2

Третья производная:

$$-12x^2\cos\left(x^2\right)-12x\cos\left(x^2\right)+8x^5\sin\left(x^2\right)+16x^4\sin\left(x^2\right)+8x^3\sin\left(x^2\right)+\frac{\square}{\left(x+1\right)^3}$$

$$\frac{+6\sin\left(x^2\right)+-2\sin\left(x^2\right)}{\left(x+1\right)^3}+\frac{-4x\sin\left(x^2\right)\left(x+1\right)-6\cos\left(x^2\right)}{\left(x+1\right)^4}$$

**<u>При n = 8:</u>**

| | Значение интеграла | Значение погрешности |
|---|---|---|
| Метод средних прямоугольников | 0,33853248 | 0,00005690 |
| Метод трапеций | 0,33788777 | 0,00011381 |
| Метод Симпсона | 0,33832247 | 0,00000286 |

**<u>При n = 20:</u>**

| | Значение интеграла | Значение погрешности |
|---|---|---|
| Метод средних прямоугольников | 0,33835153 | 0,00000910 |
| Метод трапеций | 0,33824871 | 0,00001821 |
| Метод Симпсона | 0,33831738 | 0,00000002 |

# 5. Листинг разработанной программы

**Main.java**

```
import function.TrigonometricFunction;
import equation_solution_strategy.*;


import java.util.Scanner;


/**
 * Класс, содержащий точку входа в программу - метод main.
 * Язык: java
 *
 * Реализация четвертой лабораторной работы по дисциплине:
 *     Вычислительная математика
 *
 * Текст задания:
 *  Вычислить интеграл по формулам центральных (средних) прямоугольников,
 *  трапеций и формуле Симпсона, при n=8 и n=20; оценить погрешность результата.
 *
 * @release: 01.04.21
 * @last_update: 01.04.21
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 */
public class Main
{
    //Константы для хранения последовательностей для
    //изменения цвета текста в консоли
    public static final String RESET = "\u001B[0m";
    public static final String RED = "\u001B[31m";
    public static final String PURPLE = "\u001B[35m";


    /**
     * Точка входа в программу
     * */
    public static void main(String[] args)
    {
        System.out.println("\t\t\t\tЛабораторная работа №4 <<" + PURPLE +
"Численное +
                + интегрирование функций" + RESET + ">>");



        //Создаем объект функцию
        TrigonometricFunction function = new TrigonometricFunction();
```

```
        //Открываем поток ввода
        Scanner scanner = new Scanner(System.in);


        System.out.println("Программа для вычисления интеграла функций методами
средний " +
                "прямоугольников, трапеций и методом Симпсона.");
        System.out.println("\tФункция: cos(x^2)/(x+1)");
        System.out.println();


        //Ввод нижней и верхней границы интегрирования
        System.out.print("Введите нижнюю  границу интегрирования: ");
        double a = scanner.nextDouble();


        System.out.print("Введите верхнюю границу интегрирования: ");
        double b = scanner.nextDouble();


        //Ввод кол-во итераций
        System.out.print("Введите кол-во итервалов разбиения (n): ");
        double numberOfIterations = scanner.nextDouble();
        System.out.println();


        //Создание ссылки на объект, реализующий интерфейс
        //SolutionStrategy
        SolutionStrategy strategy = null;


        //Переменная для хранения результата ввода
        String ch;


        //Сброс потока ввода
        ch = scanner.nextLine();


        //Выбор стратегии решения
        while (!ch.equals("q"))
        {
            System.out.println("Выберите метод для решения уранвения:");
            System.out.println("\t1. Метод средних прямоугольников");
            System.out.println("\t2. Метод трапеций");
            System.out.println("\t3. Метод Симпсона");
```

```
            System.out.println();
            System.out.println("\t#. Изменить кол-во итервалов разбиения (n): ");
            System.out.println();
            System.out.println("\tВведите q для выхода");
            System.out.print("Ввод: ");
            ch = scanner.nextLine();
            System.out.println();


            switch (ch)
            {
                case ("1") -> strategy = new MediumRectangleMethod();
                case ("2") -> strategy = new TrapeziumMethod();
                case ("3") -> strategy = new SimpsonMethod();
                case ("#") -> {


                                    System.out.print("Введите кол-во интервалов
разбиения
                                                                            (n):
");
                                    numberOfIterations = scanner.nextInt();
                                    scanner.nextLine();
                                    continue;
                               }
                case ("q") -> {
                                    System.out.println(RED + "Завершение работы..."
+
                                                     RESET);
                                    System.exit(0);
                               }
                default    -> System.out.println(RED + "Неверный ввод!" + RESET);
            }


            //Получаем значение интеграла
            assert strategy != null;
            double res = strategy.getSolution(function, a, b, numberOfIterations);


            //Получаем значение погрешности
            double error = strategy.getError(function, a, b, numberOfIterations);


            System.out.print("Значение интеграла:   ");
            System.out.printf("%.8f" + "\n", res);
            System.out.print("Значение погрешности: " + RED);
            System.out.printf("%.8f" + RESET + "\n\n", error);
```

```
        }
    }
}
```

**function/Function.java**

```
package function;


import java.util.List;


/**
 * Интерфейс, реализующий основные методы функций любого вида.
 *
 * Общий вид уравнения:
 *      a*x^n + b*x^(n-1) + c*x^(n-1) + ... + d*x^0 = 0
 *
 * Содержит 4 метода необходимых для данной лабораторной работы:
 *  - получение значения функции в точке
 *  - получение значения второй  производной в точке
 *  - получение значения третьей производной в точке
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see TrigonometricFunction
 * */
public interface Function
{
    /**
     * Метод для получения значения функции в заданной точке.
     *
     * @param x - точка, в которой необходимо получить значение функции.
     * @return значение функции в данной точке
     * */
    double getValueAtX(double x);


    /**
     * Метод для получения значения второй производной в при заданном x.
     *
     * @param x - точка, в которой необходимо получить значение второй
производной.
     * @return значение второй производной в данной точке
     * */
    double getSecDerivativeAtX(double x);


    /**
```

```
     * Метод для получения значения третьей производной при заданном x.
     *
     * @param x - точка, в которой необходимо получить значение третьей
производной.
     * @return значение третьей производной в данной точке
     * */
    double getThirdDerivativeAtX(double x);




}
```

**function/TrigonometricFunction.java**

```
package
function;


/**
 * Класс, реализующий необходимые методы для функции Варианта №15
 *        cos(x^2)/(x+1)
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see Function
 * */
public class TrigonometricFunction implements Function
{
    /**
     * Метод для получения значения функции в заданной точке.
     *
     * @param x - значение X, в котором необходимо получить
     *            значение функции.
     *
     * @return - значение функции при заданном X
     * */
    @Override
    public double getValueAtX(double x)
    {
        return Math.cos(x*x) / (x + 1);
    }


    /**
     * Метод для получения значения второй производной в заданной точке.
     * Функция второй производной в "читаемом" виде представлена в отчете.
     *
     * @param x - значение X, в котором необходимо получить
```

```
     *            значение функции.
     *
     * @return - значение второй производной при заданном X
     * */
    @Override
    public double getSecDerivativeAtX(double x)
    {
        return (2.0 / (x + 1)) * ((-2.0 * x*x * Math.cos(x*x)) + (2 * x *
((Math.sin(x*x))/(x + 1))) - (Math.sin(x*x)) + ((Math.cos(x*x))/((x +1)*(x + 1))));
    }


    /**
     * Метод для получения значения третьей производной в заданной точке.
     * Функция третьей производной в "читаемом" виде представлена в отчете.
     *
     * @param x - значение X, в котором необходимо получить
     *            значение функции.
     *
     * @return - значение второй производной при заданном X
     * */
    @Override
    public double getThirdDerivativeAtX(double x)
    {
        return 2.0 * (-3.0 * Math.cos(x*x) / Math.pow((1.0 + x), 3.0) + 2 * x *(-
3.0 *
        Math.cos(x*x) + 2 * x*x * Math.sin(x*x)) + 3.0 * (2.0 * x*x * Math.cos(x*x)
+
        + Math.sin(x*x)) / (1.0 + x) - 6.0 * x * Math.sin(x*x) / (1.0 + x)*(1.0 +
x)) /
        / (1.0 + x);


    }


    /**
     * Конуструктор без параметров
     * */
    public TrigonometricFunction()
    {
    }
}
```

## solution_strategy/SolutionStrategy.java

```
package equation_solution_strategy;


import function.Function;


/**
 * Общий интерфейс всех методов вычисления интеграла.
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see MediumRectangleMethod
 * @see TrapeziumMethod
 * @see SimpsonMethod
 * */
public interface SolutionStrategy
{
    /**
     * Метод для вычисления значения интеграла.
     *
     * @param function           - функция, для которой необходимо вычислить
интеграл
     * @param a                  - нижний  предел интегрирования
     * @param b                  - верхний предел интегрирования
     * @param numberOfIterations - кол-во итераций
     * @return значение инетграла для данной функции.
     * */
    double getSolution(Function function, double a, double b, double
numberOfIterations);


    /**
     * Метод для вычисления значения погрешности.
     *
     * @param function           - функция, для которой необходимо вычислить
интеграл
     * @param a                  - нижний  предел интегрирования
     * @param b                  - верхний предел интегрирования
     * @param numberOfIterations - кол-во итераций
     * @return значение инетграла для данной функции.
     * */
    double getError(Function function, double a, double b, double
numberOfIterations);
}
```

**solution_strategy/MediumRectangelMethod.java**

```
package equation_solution_strategy;


import function.Function;


import java.util.Collections;
import java.util.LinkedList;
import java.util.List;


/**
 * Класс, реализующий вычисление интеграла методом
 * средних прямоугольников.
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class MediumRectangleMethod implements SolutionStrategy
{
    /**
     * Метод для вычисления значения интеграла
     * методом средних прямоугольников.
     *
     * @param function           - функция, для которой необходимо вычислить
интеграл
     * @param a                  - нижний  предел интегрирования
     * @param b                  - верхний предел интегрирования
     * @param numberOfIterations - кол-во итераций
     * @return значение инетграла для данной функции.
     * */
    @Override
    public double getSolution(Function function, double a, double b, double
numberOfIterations)
    {
        //Вычисляем величену шага h
        double h = (b - a)/numberOfIterations;


        //пеменная для хранения значения интеграла
        double res = 0.0;


        //Вычисляем сумму значений функции в промежутках
        for (int i = 0; i < numberOfIterations; i++)
        {
```

```
            //К результату прибавляем значение f(Xi-1/2)
            //Начальная точка a, последующие шаги вычисляются как: h * (i + 0.5)
            res += function.getValueAtX(a + h * (i + 0.5));
        }


        //Поскольку шаг равномерный, то просто домножаем результат на h
        res *= h;


        return res;
    }


    /**
     * Метод для вычисления значения погрешности при вычислении
     * интеграла методом средних прямгоугольников.
     *
     * @param function           - функция, для которой необходимо вычислить
интеграл
     * @param a                  - нижний  предел интегрирования
     * @param b                  - верхний предел интегрирования
     * @param numberOfIterations - кол-во итераций
     * @return значение инетграла для данной функции.
     * */
    @Override
    public double getError(Function function, double a, double b,
                                     double numberOfIterations)
    {
        List<Double> yList = new LinkedList<>();


        //Вычисляем величену шага h
        double h = (b - a)/numberOfIterations;


        //Получаем список значений второй производной в промежутках
        for (int i = 0; i < numberOfIterations; i++)
        {
            //Получаем значения второй производной функциийи в точках от a до b с
шагом h
            yList.add(function.getSecDerivativeAtX(a + h * (i)));
        }


        //Находим максимальное значение второй производной на промежутке от a до b
        double error = Math.abs(Collections.max(yList));
```

```
        //Вычисляем погрешность: Rn = (b-a)/24 * h^2 * max(f''(x))
        error *= ((b - a) / 24.0) * h * h;


        return error;
    }


    /**
     * Конструктор без параметров.
     * */
    public MediumRectangleMethod()
    {
    }
}
```

---

**solution_strategy/TrapeziumMethod.java**

```
package equation_solution_strategy;


import function.Function;


import java.util.Collections;
import java.util.LinkedList;
import java.util.List;


/**
 * Класс, реализующий вычисление интеграла методом
 * трапеций.
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class TrapeziumMethod implements SolutionStrategy
{
    /**
     * Метод для вычисления значения интеграла
     * методом трапеций.
     *
     * @param function           - функция, для которой необходимо вычислить
интеграл
     * @param a                  - нижний  предел интегрирования
     * @param b                  - верхний предел интегрирования
     * @param numberOfIterations - кол-во итераций
```

```
     * @return значение инетграла для данной функции.
     * */
    @Override
    public double getSolution(Function function, double a, double b,
                                       double numberOfIterations)
    {
        //Вычисляем величену шага h
        double h = (b - a)/numberOfIterations;


        //пеменная для хранения значения интеграла
        double res = 0.0;


        //Выполняем заданное кол-во итераций
        for (int i = 1; i < numberOfIterations; ++i)
        {
            //К результату прибавляем значение f(Xi)
            //Начальная точка a, последующие шаги вычисляются как: h * i
            res += function.getValueAtX(a + h * i);
        }


        //Добавляем к результату (y0 + yn) / 2;
        res += (function.getValueAtX(a) + function.getValueAtX(b)) / 2.0;


        //Поскольку шаг равномерный, то домножаем результат на h
        res *= h;


        return res;
    }


    /**
     * Метод для вычисления значения погрешности при вычислении
     * интеграла методом трапеций.
     *
     * @param function           - функция, для которой необходимо вычислить
интеграл
     * @param a                  - нижний  предел интегрирования
     * @param b                  - верхний предел интегрирования
     * @param numberOfIterations - кол-во итераций
     * @return значение инетграла для данной функции.
     * */
    @Override
    public double getError(Function function, double a, double b,
```

```
                                      double numberOfIterations)
    {
        List<Double> yList = new LinkedList<>();


        //Вычисляем величену шага h
        double h = (b - a)/numberOfIterations;


        //Получаем список значений третьей производной в промежутках
        for (int i = 0; i < numberOfIterations; i++)
        {
            //Получаем значения второй производной функциий в точках от a до b с
шагом h
            yList.add(function.getSecDerivativeAtX(a + h * (i)));
        }


        //Находим максимальное значение второй производной на промежутке от a до
b
        double error = Math.abs(Collections.max(yList));


        //Вычисляем погрешность: Rn = (b-a)/24 * h^2 * max(f''(x))
        error *= ((b - a) / 12.0) * h * h;


        return error;
    }


    /**
     * Конструктор без параметров.
     * */
    public TrapeziumMethod()
    {
    }
}
```

**solution_strategy/SimpsonMethod.java**

```
package equation_solution_strategy;


import function.Function;


import java.util.Collections;
```

```
import java.util.LinkedList;
import java.util.List;


/**
* Класс реализующий вычисление интеграла методом
* трапеций.
*
* @author Vladislav Sapozhnikov 19-IVT-3
* @see SolutionStrategy
* */
public class SimpsonMethod implements SolutionStrategy
{
    /**
     * Метод для вычисления значения интеграла
     * методом Симпсона.
     *
     * @param function           - функция, для которой необходимо вычислить
интеграл
     * @param a                  - нижний  предел интегрирования
     * @param b                  - верхний предел интегрирования
     * @param numberOfIterations - кол-во итераций
     * @return значение инетграла для данной функции.
     * */
    @Override
    public double getSolution(Function function, double a, double b,
                                        double numberOfIterations)
    {
        //Вычисляем величену шага h
        double h = (b - a)/numberOfIterations;


        //Переменная для хранения суммы нечетных y
        double evenY = 0.0;


        //Переменная для хранения суммы четных y
        double oddY  = 0.0;


        //Вычисление суммы y1 + y3 + ... + yn-1
        for (int i = 1; i < numberOfIterations; i++)
        {
            if (i % 2 != 0) oddY  += function.getValueAtX(a + i*h);
        }


        //Вычисление суммы y2 + y4 + ... + yn-2
```

```
        for (int i = 2; i < (numberOfIterations - 1); i++)
        {
            if (i % 2 == 0) evenY += function.getValueAtX(a + i*h);
        }


        //Возвращение результата
        // h/3(y0 + 4*(y1 + y3 + ... + yn-1) + 2*(y2 + y4 + ... + yn-2) - yn)
        return (h / 3.0) * (function.getValueAtX(a) + 4.0 * oddY + 2.0 * evenY +
function.getValueAtX(b));
    }


    /**
     * Метод для вычисления значения погрешности при вычислении
     * интеграла методом Симсона.
     *
     * @param function           - функция, для которой необходимо вычислить
интеграл
     * @param a                  - нижний  предел интегрирования
     * @param b                  - верхний предел интегрирования
     * @param numberOfIterations - кол-во итераций
     * @return значение инетграла для данной функции.
     * */
    @Override
    public double getError(Function function, double a, double b,
                                     double numberOfIterations)
    {
        List<Double> yList = new LinkedList<>();


        //Вычисляем величену шага h
        double h = (b - a)/numberOfIterations;


        //Вычисляем значения второй производной в промежутках
        for (int i = 0; i < numberOfIterations; i++)
        {
            //Получаем значения второй производной функциий в точках от a до b с
шагом h
            yList.add(function.getThirdDerivativeAtX(a + h * (i)));
        }


        //Находим максимальное значение второй производной на промежутке от a до b
        double error = Math.abs(Collections.max(yList));
```

```
        //Вычисляем погрешность: Rn = (b-a)/24 * h^2 * max(f''(x))
        error *= ((b - a) / 288.0) * h * h * h;


        return error;
    }


    /**
     * Конструктор без параметров.
     * */
    public SimpsonMethod()
    {
    }
}
```

## 6. Результаты работы программы

```
                Лабораторная работа №4 <<Численное интегрирование функций>>
Программа для вычисления интеграла функций методами средний прямоугольников, трапеций и методом Симпсона.
    Функция: cos(x^2)/(x+1)

Введите нижнию  границу интегрирования: 0,4
Введите верхнюю границу интегрирования: 1,2
Введите кол-во итервалов разбиения (n): 8

Выберите метод для решения уравнения:
    1. Метод средних прямоугольников
    2. Метод трапеций
    3. Метод Симпсона

    #. Изменить кол-во итервалов разбиения (n):

    Введите q для выхода
Ввод: 1

Значение интеграла:   0,33853248
Значение погрешности: 0,00005690

Выберите метод для решения уравнения:
    1. Метод средних прямоугольников
    2. Метод трапеций
    3. Метод Симпсона

    #. Изменить кол-во итервалов разбиения (n):

    Введите q для выхода
Ввод: 2

Значение интеграла:   0,33788777
Значение погрешности: 0,00011381
```

```
Выберите метод для решения уравнения:
    1. Метод средних прямоугольников
    2. Метод трапеций
    3. Метод Симпсона

    #. Изменить кол-во итервалов разбиения (n):

    Введите q для выхода
Ввод: 3

Значение интеграла:   0,33832247
Значение погрешности: 0,00000286

Выберите метод для решения уравнения:
    1. Метод средних прямоугольников
    2. Метод трапеций
    3. Метод Симпсона

    #. Изменить кол-во итервалов разбиения (n):

    Введите q для выхода
Ввод: #

Введите кол-во итервалов разбиения (n): 20
Выберите метод для решения уравнения:
    1. Метод средних прямоугольников
    2. Метод трапеций
    3. Метод Симпсона

    #. Изменить кол-во итервалов разбиения (n):

    Введите q для выхода
Ввод: 1
```

```
Ввод: 1

Значение интеграла:   0,33835153
Значение погрешности: 0,00000910

Выберите метод для решения уравнения:
—— 1. Метод средних прямоугольников
—— 2. Метод трапеций
—— 3. Метод Симпсона

—— #. Изменить кол-во итервалов разбиения (n):

—— Введите q для выхода
Ввод: 2

Значение интеграла:   0,33824871
Значение погрешности: 0,00001821

Выберите метод для решения уравнения:
—— 1. Метод средних прямоугольников
—— 2. Метод трапеций
—— 3. Метод Симпсона

—— #. Изменить кол-во итервалов разбиения (n):

—— Введите q для выхода
Ввод: 3

Значение интеграла:   0,33831738
Значение погрешности: 0,00000002


Выберите метод для решения уравнения:
—— 1. Метод средних прямоугольников
—— 2. Метод трапеций
—— 3. Метод Симпсона

—— #. Изменить кол-во итервалов разбиения (n):

—— Введите q для выхода
Ввод: q

Завершение работы...

Process finished with exit code 0
```

## 7. Вывод

В ходе данной работы были закреплены знания и умения по вычислению интеграла при помощи методов средних прямоугольников, трапеция и методу Симпсона.

Самым точным является метод Симпсона, а самым не точным метод трапеций. При увеличении кол-во промежутков разбиения n точность результатов возрастает.