Университет ИТМО

Факультет ПИиКТ

Вычислительная математика

Лабораторная работа №2
Вариант – метод прямоугольников

Выполнила: Наумова Надежда

Группа P3201

Преподаватель: Перл Ольга Вячеславовна

Санкт-Петербург
2020 г.

Описание использованного метода

Пусть есть функция $f(x)$, непрерывная на отрезке $[a; b]$, тогда можем вычислить значение интеграла $\int_a^b f(x)\,dx$. Воспользуемся заменой определенного интеграла интегральной суммой. Разобьем отрезок $[a; b]$ на n частей $[x_{i-1}; x_i]$ где i ∈ $[1; n]$ и выбираем точку со значением ς_i. Существует определенный тип интегральных сумм при бесконечном уменьшении длины такой части. Это выражается формулой $\lambda = \max\limits_{i = 1,2,\dots,n} (x_i - x_{i-1}) \to 0$, тогда получаем, что любая из таких интегральных сумм – приближенное значение интеграла $\int_a^b f(x)dx \approx \sum_{i=1}^n f(\varsigma_i) \cdot (x_i - x_{i-1})$

Суть метода прямоугольников заключается в том, что приближенное значение считается интегральной суммой.

В качестве точек ς_i могут выбираться левые, правые или средние точки отрезков, то получаем формулы левых, правых и средних прямоугольников.

Обозначаем $f(x_i) = y_i, f(a) = y_0, f(b) = y_n, h = \frac{b-a}{n}$

Метод левых прямоугольников:

$$\int_a^b f(x)\,dx \approx h \cdot \sum_{i=1}^{n} y_{i-1}.$$

Метод правых прямоугольников:

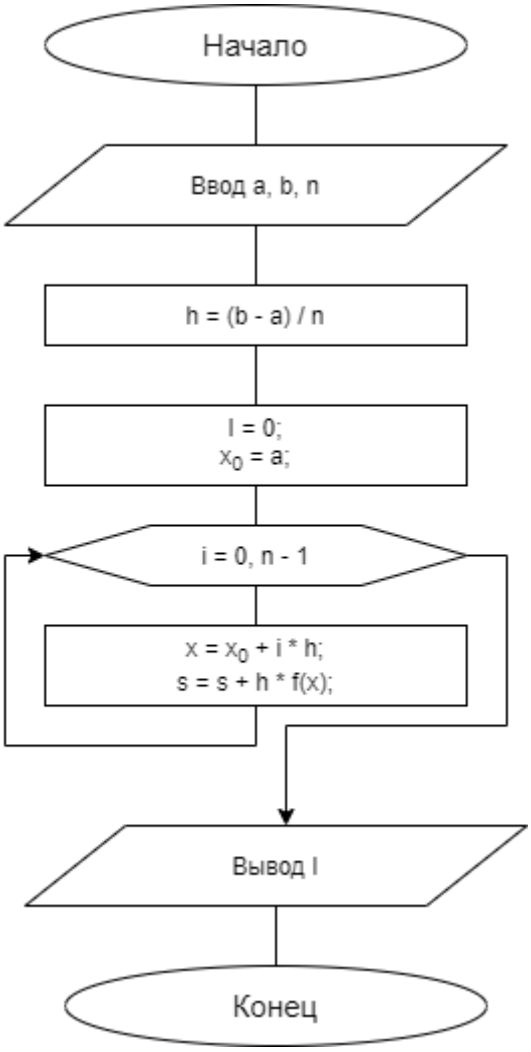
$$\int_a^b f(x)\,dx \approx h \cdot \sum_{i=1}^{n} y_i.$$

Метод средних прямоугольников:

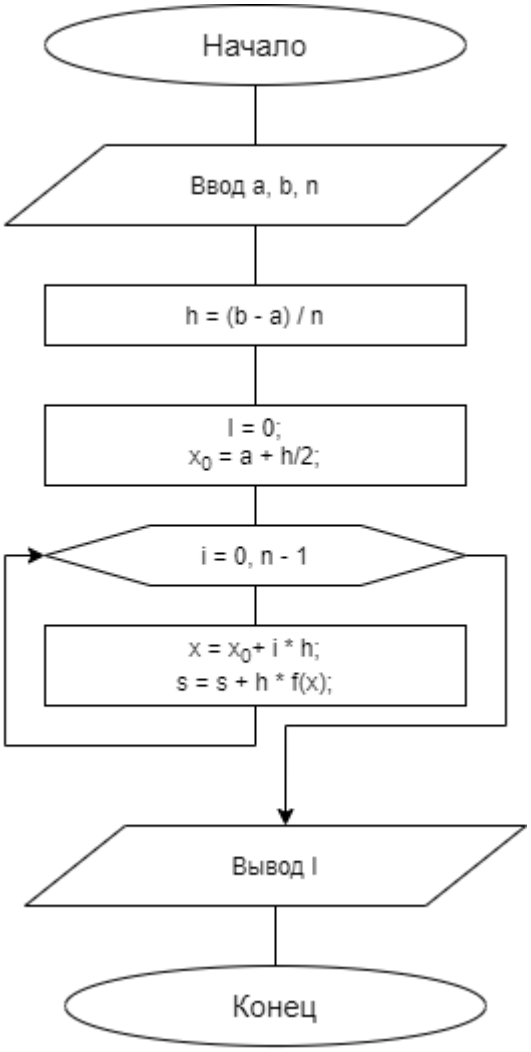
$$\int_a^b f(x)\,dx \approx h \cdot \sum_{i=1}^{n} f(x_{i-1/2}).$$

Блок-схемы

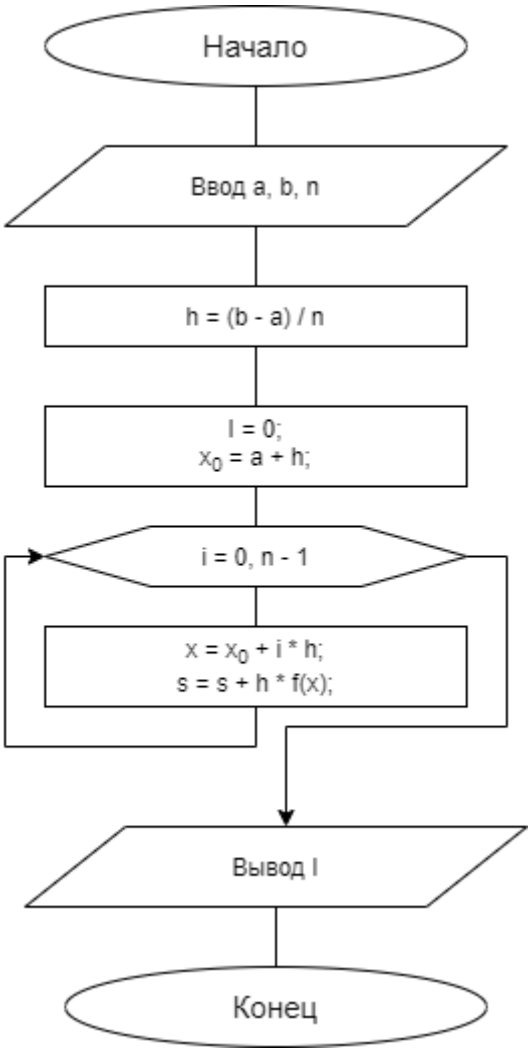
Метод левых прямоугольников:



Метод средних прямоугольников:



Метод правых прямоугольников:



Листинг численного метода

```
    //Algorithm.java
    public void calculate(Function function, double low, double high, double
userAccuracy) {
    if (userAccuracy == 0) {
        combiner.report(0, "");
        return;
    }

    if (high == low) {
        combiner.report(1, "");
        return;
    }

    for (String method : new String[] {"left", "middle", "right"}) {
        int stepCounter = 4;

        double curValue = calculateByMethod(method, function, low, high,
stepCounter);
        double prevValue;

        do {
            stepCounter <<= 1;
```

```
            if (stepCounter > 1000000000) {
                combiner.report(2, method);
                return;
            }

            prevValue = curValue;
            curValue = calculateByMethod(method, function, low, high, stepCounter);

            if (!Double.isFinite(curValue) || !Double.isFinite(prevValue)) {
                combiner.report(2, "");
                return;
            }
        } while (calculateError(prevValue, curValue, 3.0) > userAccuracy);

        combiner.reportOnSuccess(method, curValue, stepCounter,
calculateError(prevValue, curValue, 3.0));
    }
}

private double calculateByMethod(String method, Function function, double low, double
high, int stepCounter) {
    double step = calculateStep(low, high, stepCounter);
    double x;
    switch (method) {
        case "left" :
            x = low;
            return calculateIntegral(function, stepCounter, step, x);
        case "middle":
            x = low + step/2;
            return calculateIntegral(function, stepCounter, step, x);
        case "right":
            x = low + step;
            return calculateIntegral(function, stepCounter, step, x);
        default:
            return Double.NaN;
    }
}


private double calculateIntegral(Function function, int stepCounter, double step,
double x) {
    double result = 0;

    for (int i = 0; i < stepCounter; i++) {
        double fx = function.getY(x);

        if (!Double.isFinite(fx)) {
            if (i == 0) {
                fx = function.getY(x + EPSILON);
            } else if (i == stepCounter - 1) {
                fx = function.getY(x - EPSILON);
            } else {
                fx = (function.getY(x - EPSILON) + function.getY(x + EPSILON)) / 2;
            }
        }

        x += step;
        result += fx;
    }
    return result * step;
}
```

```
private double calculateError(double integralN, double integral2N, double
coefficient) {
    return (Math.abs(integral2N - integralN)) / coefficient;
}
private double calculateStep(double low, double high, double stepCounter) {
    return (high - low) * 1.0 / (stepCounter * 1.0);
}
```

Примеры

Enter the command >>> choose 3;
Enter the lower limit of the integration >>> -2
Enter the higher limit of the integration >>> 4
Enter the accuracy >>> 0.01
Value of the integral by the method of left rectangles is 5.982421875 count of steps: 512, error: 0.005859375
Value of the integral by the method of middle rectangles is 6.0 count of steps: 4, error: 0.0
Value of the integral by the method of right rectangles is 6.017578125 count of steps: 512, error: 0.005859375

Enter the command >>> choose 4;
Enter the lower limit of the integration >>> 0
Enter the higher limit of the integration >>> 0
Enter the accuracy >>> 0.0001
The integral is 0, the integration limits are equal.

Enter the command >>> choose 2;
Enter the lower limit of the integration >>> -3
Enter the higher limit of the integration >>> 3
Enter the accuracy >>> 0.000001
Target accuracy not achieved.

Enter the command >>> choose 1;
Enter the lower limit of the integration >>> 2
Enter the higher limit of the integration >>> 5
Enter the accuracy >>> 0.0001
Value of the integral by the method of left rectangles is 0.9165104959875809 count of steps: 2048, error: 7.327973841379325E-5
Value of the integral by the method of middle rectangles is 0.9162138618395328 count of steps: 32, error: 7.673362785790931E-5
Value of the integral by the method of right rectangles is 0.9160710428625811 count of steps: 2048, error: 7.320463658622156E-5

Вывод:

В данной лабораторной работе я реализовала алгоритм вычисления интеграла с помощью метода прямоугольников. В процессе изучения других методов удалось найти некоторые различия.

Что касается сравнений разных вариаций метода прямоугольников между собой, то метод средних прямоугольников даст большую точность, чем методы левых и правых прямоугольников для заданного *n*. Однако из-за нарушения симметрии в формулах правых и левых прямоугольников, их погрешность значительно больше, чем в методе средних прямоугольников. В то же время, объем вычислений одинаков, так что использование метода средних прямоугольников предпочтительнее.

В отличие от метода прямоугольников, в методе трапеций на каждом $[x_{i-1}; x_i]$ подынтегральную функцию заменяют интерполяционным многочленом первой степени. Интерполяция кусочно-линейная, поэтому график исходной функции представляется как ломаная, которая соединяет точки (x_i, y_i). Площадь всей фигуры состоит из площадей всех таких трапеций: сложив все равенства, получаем формулу для численного интегрирования. Погрешность метода трапеций выше, чем у метода средних прямоугольников (результат от метода средних прямоугольников будет более точным из-за способа вычисления элементарных площадей, который использует значение функции в центральной точке отрезка), но ниже, чем у методов левых и правых прямоугольников. Заметим, что метод прямоугольников в том виде, в котором он описан выше, в отличие от метода трапеций, не применим в общем случае к функциям, значения которых мы знаем в конечном числе точек, так как, например, мы не всегда можем разбить отрезок интегрирования на подотрезки, серединами которых являются точки, в которых нам известно значение функции; в методе трапеций же можно взять в качестве узлов интегрирования данные точки.

В методе Симпсона (парабол) разбиваем отрезок $[a;\ b]$ на четное число n равных частей с шагом, равным h. На каждом отрезке $[x_{i-1}; x_{i+1}]$ подынтегральную функцию заменяют интерполяционным многочленом второй степени. Тогда общая формула: $S = \frac{h}{3} \cdot (y_0 + 4(y_1 + y_3 + \ldots + y_{n-1}) + 2(y_2 + + y_4 + \ldots + y_{n-2}) + y_n)$. Точность метода Симпсона выше точности метода прямоугольников и трапеций.