Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский авиационный институт
(Национальный исследовательский университет)»
Институт информационных систем и технологий
Кафедра
«Компьютерная математика»

Курсовая работа по дисциплине
«Вычислительные технологии»
на тему:
«Применение численных методов для решения различных алгебраических
уравнений и вычисления определенных интегралов»

Вариант № 4

Выполнил:
студент группы 8-Т3О-302Б-16
Дедела Артур Саулюсович

Проверил:
Михайлов И. Е.

Москва 2018

Оглавление

1. Приближенное вычисление значения функции (суммы ряда) с использованием разложения функции в ряд

Цель работы

Целью работы является получение навыков решения задач вычислительной математики с помощью ЭВМ на примере нахождения суммы ряда.

Вариант 4

$$\sum_{k=0}^{\infty}(-1)^k \frac{(2k)!}{[(2k)!]^2} x^{2k} = \sqrt{\frac{1+\sqrt{1+16x^2}}{2(1+16x^2)}}; \qquad |x| < \frac{1}{4}$$

Метод расчета

Расчет производится последовательным прибавлением каждого очередного слагаемого к сумме. В начале расчета сумма должна быть обнулена. Количество членов ряда выбирается таким образом, чтобы погрешность, вычисляемая как модуль разности правых и левых частей выражения, не превышала $\varepsilon = 10^{-15}$.

Пример 1

N	N-й член суммы	Сумма N членов ряда		
				Точность
0	1	1	ε =	0,000000000000001
1	-0.37470006000000006	0.62529993999999989		Параметр
2	0.27300026243000708	0.898300202430007	x =	0,2499
3	-0.22504507236758664	0.67325513006242033		Истинное значение
4	0.19575307634730496	0.86900820640972531	*True value* =	0.776996865078002
5	-0.17549353106047319	0.69351467534925215		
…				
33132	6.71810425885964E-15	0.776996865078008		
33133	-6.71262955019266E-15	0.776996865078001		

Из приведенных расчетов видно, что при добавлении 33134-го слагаемого достигается требуемая точность $\varepsilon = 10^{-15}$.

Пример 2

N	N-й член суммы	Сумма N членов ряда		
0	1	1		Точность
			ε =	0,000000000000001
1	-0.0054	0.9946		Параметр
2	5.67E-05	0.9946567	x =	0,03
3	-6.73596E-07	0.994656026404		Истинное значение
4	8.444007E-09	0.994656034848007	*True value* =	0.994656034740329
5	-1.0909657044E-10	0.994656034738911		
6	1.437099368796E-12	0.994656034740348		
7	-1.918764541854E-14	0.994656034740328		

Из приведенных расчетов видно, что при добавлении 8-го слагаемого достигается требуемая точность $\varepsilon = 10^{-15}$

Блок-Схемы

Блок-схема Main

Блок схема функции Calculate

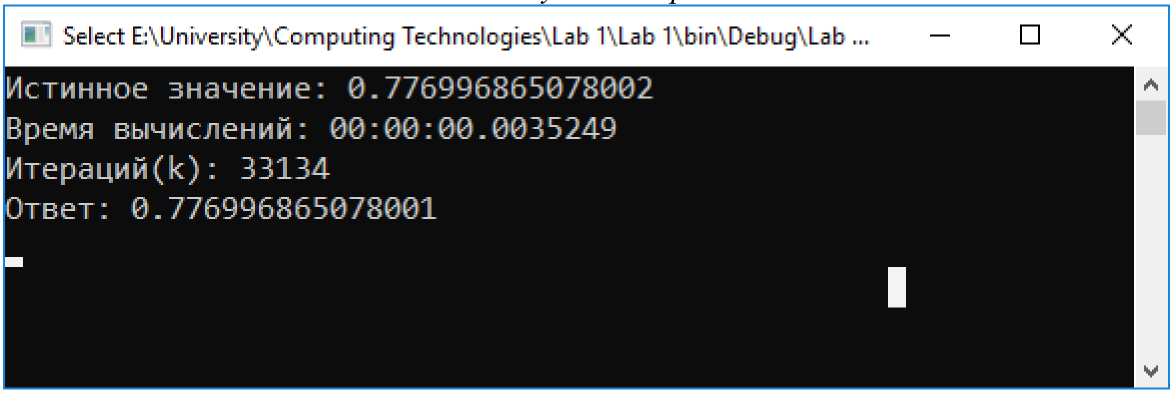
Код программы

```
using System;
using System.Diagnostics;

namespace Lab_1
{
    class Program
    {
        private static double OriginalFunc(double x)
        {
            return Math.Sqrt((1 + Math.Sqrt(1 + 16 * x * x)) / (2 * (1 + 16 * x * x)));
        }

        private static double Calculate(double x, double E = 0.1)
        {
            if (Math.Abs(x) >= 0.25) throw new ArgumentOutOfRangeException(nameof(x));

            var original = OriginalFunc(x);
            Console.WriteLine($"Истинное значение: {original}");

            Stopwatch sw = new Stopwatch();
            sw.Start();

            var sum = 0.0;
            long k = 0;
            double sumMember = 1;

            do
            {
                var sign = k % 2 == 0 ? 1 : -1;
                sum += sumMember * sign;
                ++k;

                sumMember *= (4.0 * k - 3.0) / (2.0 * k - 1.0) * (4.0 * k - 2.0) / (2.0 *
k) * (4.0 * k - 1.0) / (2.0 * k - 1.0) * (4.0 * k) / (2.0 * k) * x * x;
            } while (Math.Abs(original - sum) >= E);
            sw.Stop();

            Console.WriteLine($"Время вычислений: {sw.Elapsed.ToString()}");
            Console.WriteLine($"Итераций(k): {k}");

            return sum;
        }

        static void Main(string[] args)
        {
            var res = Calculate(0.03, 1e-15);

            Console.WriteLine($"Ответ: {res:F15}");
            Console.ReadKey();
        }
    }
}
```

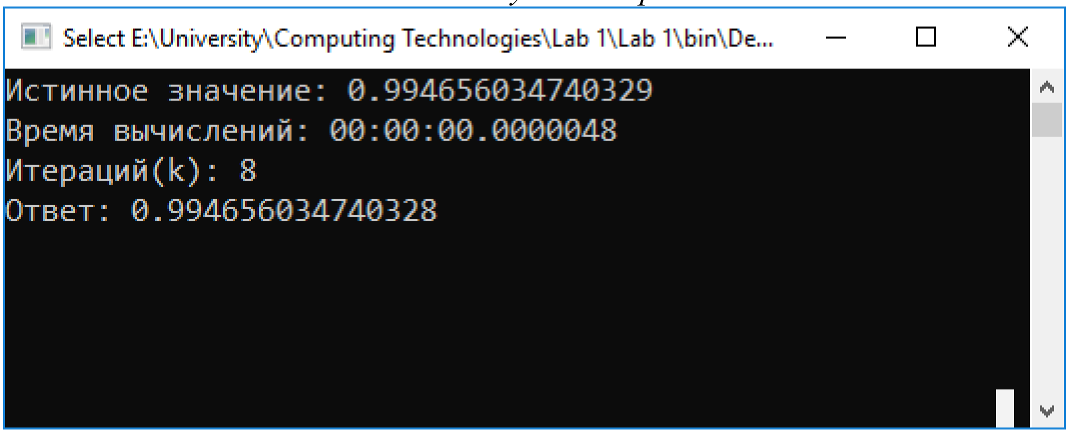
Результаты работы программы и анализ результатов

Результат работы программы полностью совпал с тестовым примером. Это говорит о том, что программа может использоваться для вычисления других рядов с заданной точностью и в заданном диапазоне значений функции.

Результат при x = 0.2499



Результат при x = 0.03



Выводы

Вычисление значения функции на компьютере невозможно никаким иным способом, кроме разложения её в ряд. Именно так считаются встроенные библиотечные функции, например *sin* или *cos*. Таким образом, написанная нами программа, позволяет создавать свои библиотечные функции любого вида.

2. Приближение вычисление корней уравнения f(x)=0 методом Ньютона и методом релаксации

Цель работы

Знакомство с возможностями приближенного вычисления корней уравнения $f(x) = 0$ при различных видах функции $f(x)$. Решение проблем отделения корней на отрезке. Подробное изучение метода Ньютона и метода релаксации. Получение навыков решения задач вычислительной математики на ЭВМ.

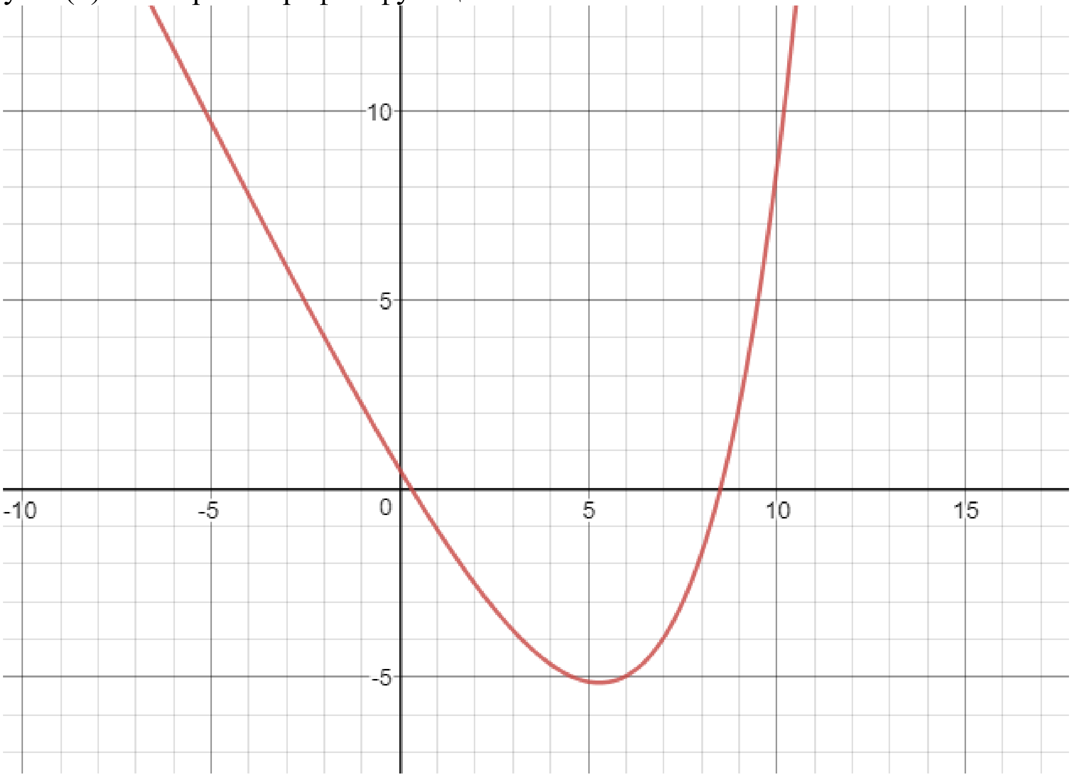
Вариант 4

$$1.4^x - 2x - 0.5 = 0$$

Метод расчета

При численном подходе задача о решении нелинейных уравнений разбивается на два этапа: локализация (отделение корней), то есть нахождение таких отрезков на оси x, в пределах которых содержится один единственный корень, и уточнение корней, т.е. вычисление приближенных значений корней с заданной точностью.

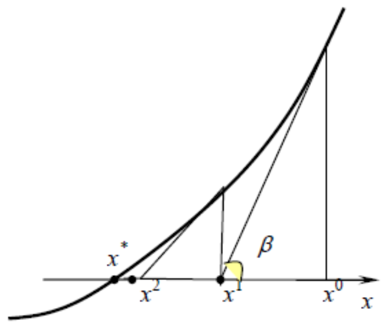
Отделение корней можно выполнить графически, если удается построить график функции y = f(x). Построим график функции: $1.4^x - 2x - 0.5 = 0$



Получаем $x_1 \approx 0.304$, $x_2 \approx 8.51$.

Метод Ньютона

Пусть для нашего уравнения известно некоторое начальное приближение x_0. В этой точке функция $f(x)$ заменяется своей касательной. Точка пересечения касательной с осью абсцисс является новым приближением. Процесс повторяется до выполнения требований к точности приближения. Для касательной, проведённой из точки:



x_0: $\frac{f(x_0)}{x_0 - x_1} = \tan\beta = f'(x_0)$.

Отсюда $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$. Повторяя процесс, получим формулу метолa Ньютона:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Из данной формулы вытекает условие применимости метода: функция f(x) должна быть дифференцируемой и f'(x) в окрестности корня не должна менять знак. Для окончания итерационного процесса может быть использовано следующее условие:

$$|x_{k+1} - x_k| < \varepsilon \ \wedge \ |f(x_{k+1})| < \varepsilon$$

Условие сходимости итерационного процесса:

$$|f(x) \cdot f''(x)| < \left(f'(x)\right)^2, x \in [a, b]$$

Если на отрезке существования корня знаки f'(x) и f''(x) не изменяются, то начальное приближение, обеспечивающее сходимость, нужно выбрать из условия:

$$f(x_0) \cdot f''(x_0) > 0, \ x_0 \in [a, b]$$

Таким образом, мы можем определить отрезки на которых находятся наши корни. Возьмем отрезок [0.1, 1], на котором находится x_1, отрезок [12, 13], содержащий x_2. При помощи метода Ньютона уточним наши корни с точностью до e = 0,00001.

Метод релаксации - частный случай метода простой итерации, он получается при $\tau(x) = \tau = \text{const}$.

$$\frac{x_{n+1} - x_n}{\tau} = f(x_n), \ n=0,1,\ldots \qquad (3)$$

Метод релаксации сходится при условии

$$-2<\tau f'(x^*)<0. \qquad (4)$$

Если в некоторой окрестности корня выполняются условия

$$f'(x)<0, \ 0<m<|f'(x)|<M, \qquad (5)$$

то метод релаксации сходится при $\tau \in (-2/M, 0)$.

Чтобы выбрать оптимальный параметр τ в методе релаксации, рассмотрим уравнение для погрешности $z_n = x_n - x^*$. Подставляя $z_n = z_n - x^*$ в (3) получим уравнение

$$\frac{z_{n+1} - z_n}{\tau} = f(x^* + z_n)$$

По теореме о среднем имеем

$$f(x^* + z_n) = f(x^*) + z_n f'(x^* + \Theta z_n) = z_n f'(x^* + \Theta z_n)$$

где $\theta \in (0,1)$. Таким образом, для погрешности метода релаксации выполняется уравнение

$$\frac{z_{n+1} - z_n}{\tau} = f'(x^* + \Theta z_n) z_n$$

Отсюда приходим к оценке

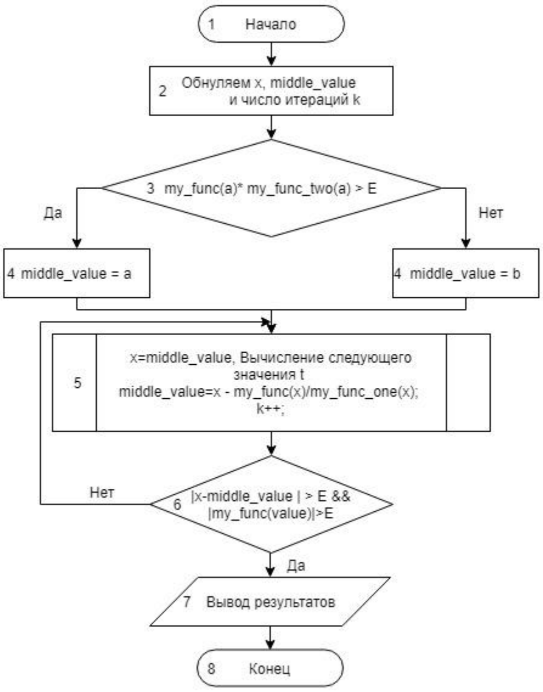
$$|z_{n+1}| \le \max\{|1+\tau M|, |1+\tau m|\}\cdot|z_n|.$$

Наилучшая оценка достигается при $|1+\tau M| = |1+\tau m|$, таким образом оптимальным значением параметра является $\tau_0 = -2/(M+m)$.

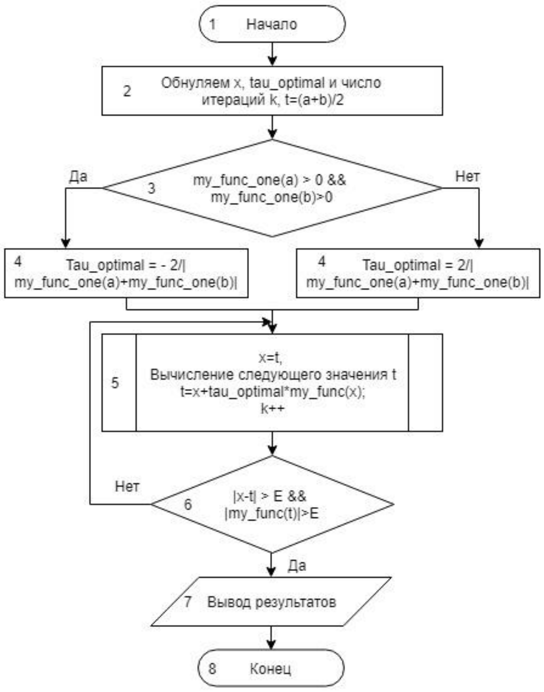
Блок-Схемы

Блок-схема Main

Блок схема функции Metod_Newton



Блок-схема функции Method_Relax



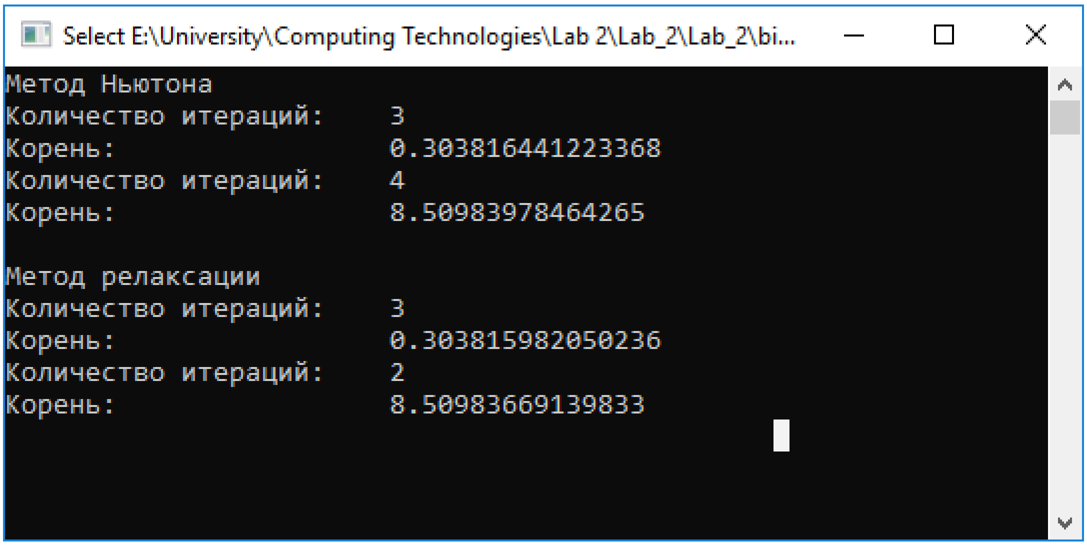
Код программы

```
using System;

namespace Lab_2
{
    class Program
    {
        static double Equation(double x)
        {
            return Math.Pow(1.4, x) - 2 * x - 0.5;
        }

        static double EquationDerivative(double x)
        {
            return Math.Pow(1.4, x) * Math.Log(1.4) - 2;
        }

        static double EquationSecondDerivative(double x)
        {
            var log = Math.Log(1.4);
            return log * log * Math.Pow(1.4, x);
        }

        static void Main()
        {
            var a1 = 0.0;
            var b1 = 1.0;

            var a2 = 8;
            var b2 = 9;

            Console.WriteLine("Метод Ньютона");
            Newton(a1, b1);
            Newton(a2, b2);
            Console.WriteLine();

            Console.WriteLine("Метод релаксации");
            Relax(a1, b1);
            Relax(a2, b2);
            Console.ReadLine();
        }

        static void Relax(double a, double b)
        {
            var E = 1e-5;
            var i = 0;
            var x = (a + b) / 2.0;
            var inf = Math.Min(EquationDerivative(a), EquationDerivative(b));
            var sup = Math.Max(EquationDerivative(a), EquationDerivative(b));
            var tau = -2.0 / (inf + sup);

            var x1 = x + Equation(x) * tau;
            do
            {
                x = x1;
                x1 = x + Equation(x) * tau;
                i++;
            } while (Math.Abs(x1 - x) > E);

            Console.WriteLine($"Количество итераций: \t{i}");
            Console.WriteLine($"Корень: \t\t{x}");
        }
```

```
        static void Newton(double a, double b)
        {
            var i = 0;
            var E = 1e-5;

            var middleValue = Equation(a) * EquationSecondDerivative(b) > 0 ? a : b;
            double x;

            do
            {
                x = middleValue;
                middleValue = x - Equation(x) / EquationDerivative(x);
                i++;
            } while (Math.Abs(x - middleValue) > E);

            Console.WriteLine($"Количество итераций: \t{i}");
            Console.WriteLine($"Корень: \t\t{x}");
        }
    }
}
```

Результаты работы программы и анализ результатов

Результат работы программы полностью совпал с тестовым примером. Это говорит о том, что программа может использоваться для вычисления других вариантов с заданной точностью и в заданном диапазоне значений функции.



Выводы

На основании полученных результатов можно сделать вывод о достоинствах и недостатках методов. Главное достоинство метода Ньютона - высокая скорость сходимости (второй порядок сходимости). Недостатки - сложность метода (необходимо вычислять производные, сильная зависимость сходимости от вида функции и выбора начального приближения). К недостаткам метода релаксации можно отнести медленную сходимость (первый порядок сходимости), а к достоинствам - простоту алгоритма.

3. Приближение вычисление корней системы нелинейных уравнений методом Ньютона

Цель работы

Освоение методов приближенного вычисления корней системы нелинейных уравнений $f(x) = 0$. Распространение навыков, полученных при выполнении лабораторной работы №2 на систему уравнений. Подробное изучение метода Ньютона. Получение навыков решения задач вычислительной математики на ЭВМ.

Вариант 4

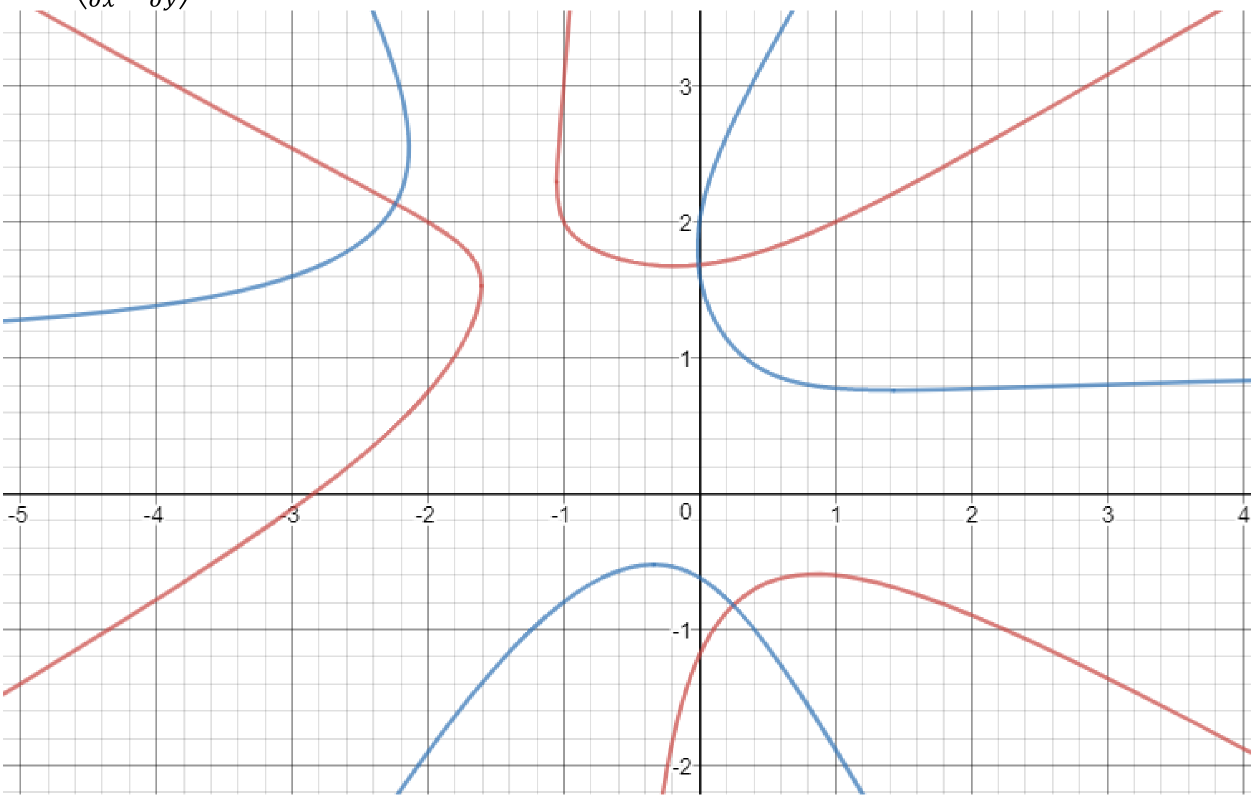
$$\begin{cases} x^3 + 2x^2 - 3xy^2 + 6xy - x - 2y^2 + y + 4 = 0 \\ -y^3 + 3y^2 + 3x^2y + 4xy - y - 3x^2 - x - 2 = 0 \end{cases}$$

Метод расчета

$f(x,y) = x^3 + 2x^2 - 3xy^2 + 6xy - x - 2y^2 + y + 4,$
$g(x,y) = -y^3 + 3y^2 + 3x^2y + 4xy - y - 3x^2 - x - 2.$
Введем переменные a, b, c, d, где
$a = f'_x(x,y) = 3x^2 + 4x - 3y^2 + 6y - 1,$
$b = f'_y(x,y) = -6xy + 6x - 4y + 1,$
$c = g'_x(x,y) = 6xy + 4y - 6x - 1,$
$d = g'_y(x,y) = -3y^2 + 6y + 3x^2 + 4y - 1.$

В данном случае производные находятся аналитически. Запишем якобиан для системы из двух уравнений:

$$A = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$



Искомые x, y находятся следующим образом: $\binom{x}{y}_{n+1} = \binom{x}{y}_n - A^{-1}(x_n, y_n)\binom{f(x_n,y_n)}{g(x_n,y_n)}$, где $A^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$.

Фактически надо вычислять:

$$x_{n+1} = x_n - \frac{d \cdot f(x_n,y_n) - b \cdot g(x_n,y_n)}{ad-bc},$$

$$y_{n+1} = y_n - \frac{a \cdot g(x_n,y_n) - c \cdot f(x_n,y_n)}{ad-bc}.$$

В качестве начального приближения стоит взять точку, которая недалеко отстоит от корня системы. Для поиска такой точки следует построить графики уравнений системы и на основе графиков выбрать нужное начальное приближение.

Теперь найдем частные производные численно. Для этого обратимся к определению частной производной. Пусть функция $z = f(x,y)$ определена в некоторой окрестности точки $D(x,y)$. Придадим переменной x приращение h, оставляя при этом значение переменной y без изменения так, чтобы точка $D(x+h,y)$ принадлежало этой окрестности, где $h = 10^{-3}$.

Тогда $\Delta_x z = f(x+h,y) - f(x,y)$ называют частным приращением функции $f(x,y)$. Аналогично $\Delta_y z = f(x,y+h) - f(x,y)$.

Частной производной функции $f(x,y)$ по переменной x в точке (x,y) называют предел

$\lim\limits_{h\to 0}\frac{\Delta_x z}{h} = \lim\limits_{h\to 0}\frac{f(x+h,y)-f(x,y)}{h}$, если он существует.

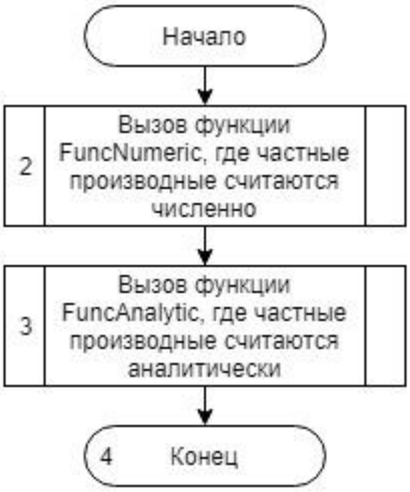
Воспользуемся данным определением и запишем переменные a, b, c, d следующим образом:

$$a = f'_x(x,y) \approx \frac{f(x_n+h,y_n)-f(x_n,y_n)}{h},\ b = f'_y(x,y) \approx \frac{f(x_n,y_n+h)-f(x_n,y_n)}{h},$$

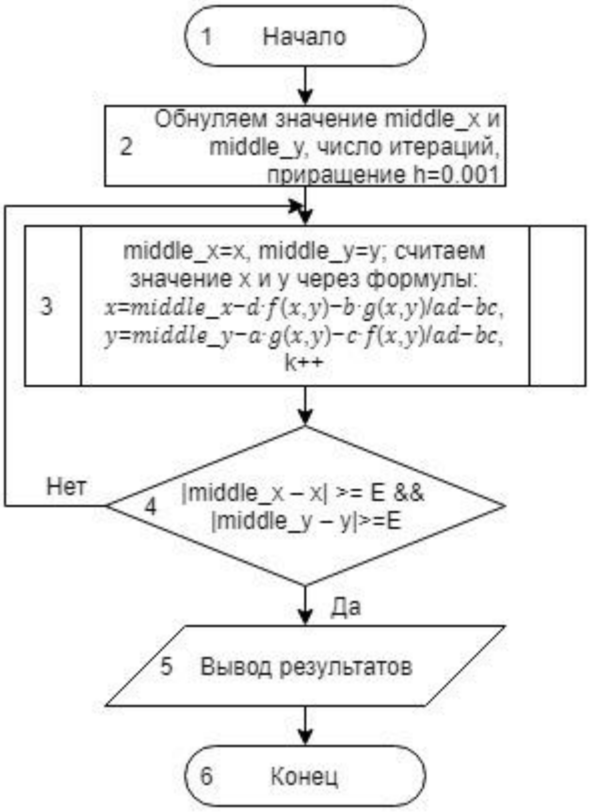
$$c = g'_x(x,y) \approx \frac{g(x_n+h,y_n)-g(x_n,y_n)}{h},\ d = g'_y(x,y) \approx \frac{g(x_n,y_n+h)-g(x_n,y_n)}{h}.$$

Блок-схемы

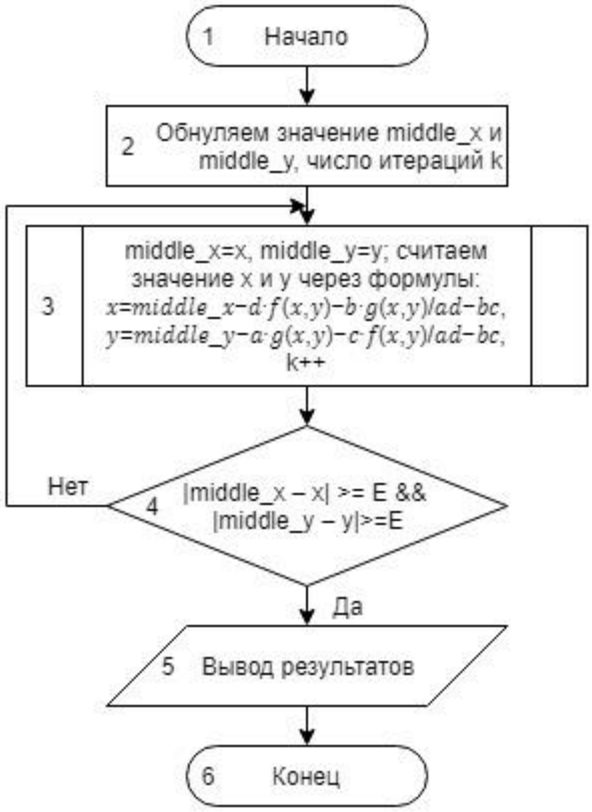
Блок-схема Main



Блок-схема функции FuncNumeric



Блок-схема функции FuncAnalytic

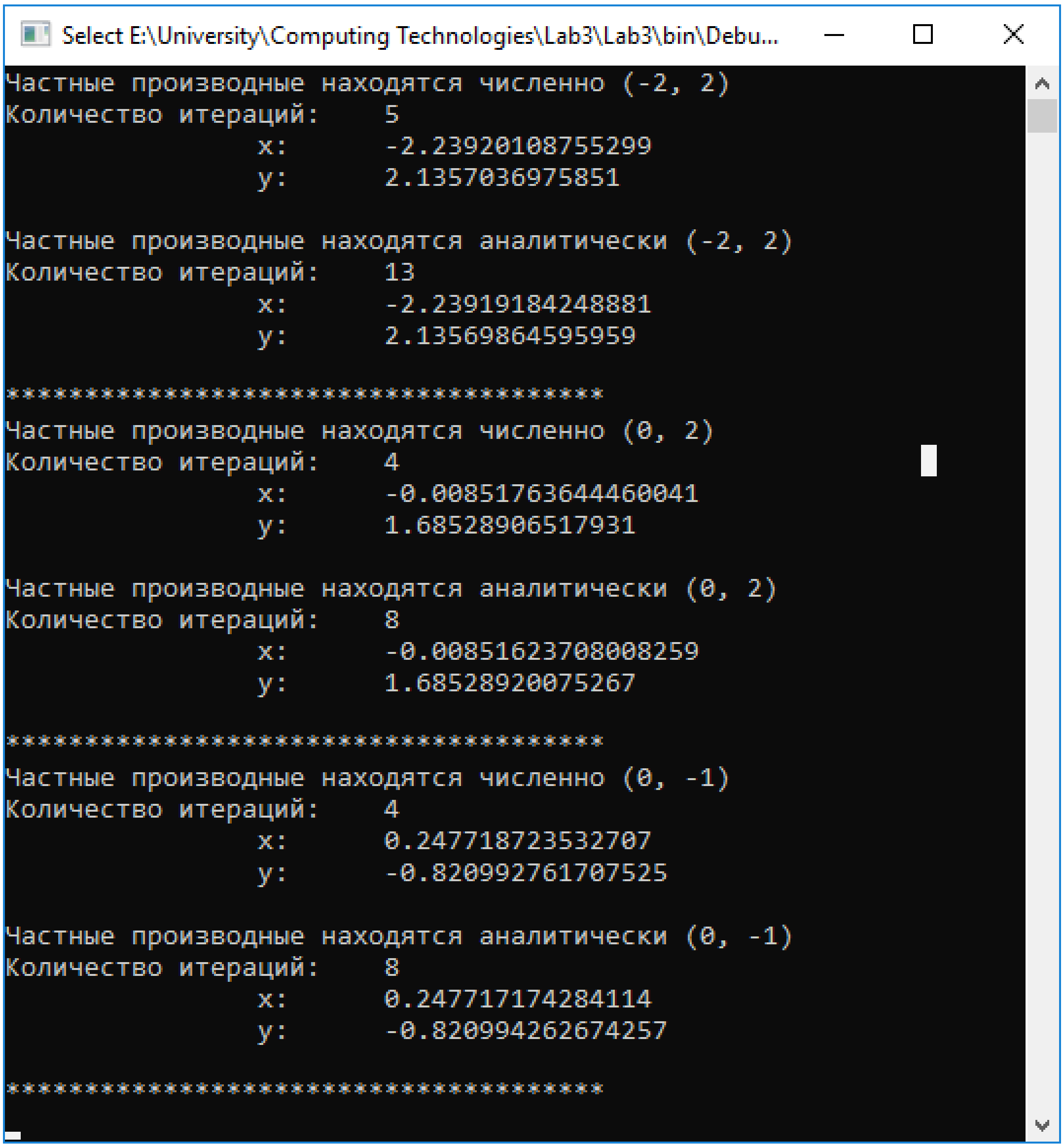


Код программы

```
using System;
using System.Runtime.InteropServices;

namespace Lab3
{
    class Program
    {
        static double f(double x, double y) => x * x * x + 2 * x * x - 3 * x * y * y + 6
* x * y - x - 2 * y * y + y + 4;
        static double g(double x, double y) => -(y * y * y) + 3 * y * y + 3 * x * x * y +
4 * x * y - y - 3 * x * x - x - 2;

        static double f_x(double x, double y) => 3 * x * x + 4 * x - 3 * y * y + 6 * y -
1;
        static double f_y(double x, double y) => -6 * x * y + 6 * x - 4 * y + 1;
        static double g_x(double x, double y) => 6 * x * y + 4 * y - 6 * x - 1;
        static double g_y(double x, double y) => -3 * y * y + 6 * y + 3 * x * x + 4 * y -
1;

        static double f_x_analytic(double x, double y, double h) => (f(x + h, y) - f(x,
y)) / h;
        static double f_y_analytic(double x, double y, double h) => (f(x, y + h) - f(x,
y)) / h;
        static double g_x_analytic(double x, double y, double h) => (g(x + h, y) - g(x,
y)) / h;
        static double g_y_analytic(double x, double y, double h) => (g(x, y + h) - g(x,
y)) / h;

        static void Analytic(double x, double y)
        {
            var k = 0;
            double xn;
            double yn;
            const double E = 0.00001;

            do
            {
                xn = x;
                yn = y;

                var delta = f_x(x, y) * g_y(x, y) - g_x(x, y) * f_y(x, y);

                x = xn - (g_y(x, y) * f(x, y) - f_y(x, y) * g(x, y)) / delta;
                y = yn - (f_x(x, y) * g(x, y) - g_x(x, y) * f(x, y)) / delta;

                k++;
            } while (Math.Abs(xn - x) >= E && Math.Abs(yn - y) >= E);

            Console.WriteLine($"Количество итераций:\t{k}\n\t\tx:\t{x}\n\t\ty:\t{y}\n");
        }

        static void Numeric(double x, double y)
        {
            var k = 0;
            double xn;
            double yn;
            var E = 0.00001;
            var h = 0.001;

            do
            {
                xn = x;
                yn = y;
```

```
                double delta = f_x_analytic(x, y, h) * g_y_analytic(x, y, h) -
g_x_analytic(x, y, h) * f_y_analytic(x, y, h);

                x = xn - (g_y_analytic(x, y, h) * f(x, y) - f_y_analytic(x, y, h) * g(x,
y)) / delta;
                y = yn - (f_x_analytic(x, y, h) * g(x, y) - g_x_analytic(x, y, h) * f(x,
y)) / delta;

                k++;
            } while (Math.Abs(xn - x) >= E && Math.Abs(yn - y) >= E);

            Console.WriteLine($"Количество итераций:\t{k}\n\t\tx:\t{x}\n\t\ty:\t{y}\n");
        }


        static void Main()
        {
            Console.WriteLine("Частные производные находятся численно (-2, 2)");
            Numeric(-2.0, 2.0);
            Console.WriteLine("Частные производные находятся аналитически (-2, 2)");
            Analytic(-2.0, 2.0);
            Console.WriteLine("**************************************");


            Console.WriteLine("Частные производные находятся численно (0, 2)");
            Numeric(0.0, 2.0);
            Console.WriteLine("Частные производные находятся аналитически (0, 2)");
            Analytic(0.0, 2.0);
            Console.WriteLine("**************************************");


            Console.WriteLine("Частные производные находятся численно (0, -1)");
            Numeric(0.0, -1.0);
            Console.WriteLine("Частные производные находятся аналитически (0, -1)");
            Analytic(0.0, -1.0);
            Console.WriteLine("**************************************");

            Console.ReadKey();
        }
    }
}
```

Результаты работы программы и анализ результатов



Результат работы программы полностью совпал с тестовым примером. Это говорит о том, что программа может использоваться для вычисления других вариантов с заданной точностью и в заданном диапазоне значений функции.

Выводы

Заданная система нелинейных уравнений имеет два корня. Приближенное решение ищем с точностью $\varepsilon = 0.00001$. В качестве начальных приближений берем точку (1 ; -1.5) и точку (3 ; 2.5). Систему нелинейных уравнений решаем методом Ньютона. Получаем решения (1.458890; -1.396767) и (3.487442 ; 2.261628). Метод Ньютона сходится за 4-6 итераций, то есть имеет высокую скорость сходимости в обоих случаях.

4. Приближенное вычисление корней системы линейных уравнений прямым (точным) и приближенными (итерационными) методами. Сравнение методов

Цель работы

Изучение точных и приближенных методов вычисления корней системы нелинейных уравнений $A\mathbf{x} = \mathbf{f}$. Получение навыков решения задач вычислительной математики на ЭВМ. Освоение умения анализировать результаты, полученные на компьютере и сравнивать методы.

Вариант 4

$$\begin{pmatrix} 1.65 & -1.76 & 0.77 \\ -1.76 & 1.04 & -2.61 \\ 0.77 & -2.61 & -3.18 \end{pmatrix} \begin{pmatrix} x1 \\ x2 \\ x3 \end{pmatrix} = \begin{pmatrix} 2.15 \\ 0.86 \\ -0.73 \end{pmatrix}$$

Метод расчета

Метод скорейшего спуска (градиентный метод) для случая системы линейных алгебраических уравнений.

В рассматриваемом ниже итерационном методе вычислительный алгоритм строится таким образом, чтобы обеспечить минимальную погрешность на шаге (максимально приблизиться к корню).

Представим систему линейных уравнений в следующем виде:

$$\begin{cases} f_1 = \sum_{j=1}^{n} a_{1j}x_j - b_1, \\ f_2 = \sum_{j=1}^{n} a_{2j}x_j - b_2, \\ \ldots \\ f_n = \sum_{j=1}^{n} a_{nj}x_j - b_n. \end{cases}$$

Запишем выражение в операторной форме: $f = A \cdot x - b$.

Здесь приняты следующие обозначения:

$$f = \begin{bmatrix} f_1 \\ f_2 \\ \ldots \\ f_n \end{bmatrix}, \quad A = [a_{ij}], \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{bmatrix}$$

В методе скорейшего спуска решение ищут в виде: $x^{p+1} = x^p - \mu_p W_p' r_p$,

Где x^p и x^{p+1} - векторы неизвестных на P и P+1 шагах итераций; вектор невязок на P-ом шаге определяется выражением $r_p = A \cdot x^p - b$, а $\mu = \frac{(r_p, WW' r_p)}{(WW' r_p, WW' r_p)}$.

В формуле используется скалярное произведение двух векторов, которое определяется следующей формулой:

$(f(x), \varphi(x)) = \sum_{i=1}^{n} f_i(x)\varphi_i(x)$; $(f(x), f(x)) = \sum_{i=1}^{n} [f_i(x)]^2$.

В формуле W_p'- транспонированная матрица Якоби, вычисленная на P-ом шаге. Матрица Якоби вектор – функции $F(X)$ определяется как

$$W = \frac{df}{dx} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}.$$

Нетрудно убедиться, что для системы матрица Якоби равна

$$W = \frac{df}{dx} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} = A.$$

Как и для метода простой итерации, достаточным условием сходимости метода градиента является преобладание диагональных элементов. В качестве нулевого приближения можно взять $x_i^0 = \frac{b_i}{a_{ii}}$.

В методе градиента итерационный процесс естественно закончить при достижении $|r_p| \leq \varepsilon$.

Итерация, k	Значения неизвестных			Невязка
	x_1	x_2	x_3	ε
100579	79.4282570	60.1600094	-29.9155948	10^{-2}
145337	79.8292311	60.4644344	-30.0673076	10^{-3}
190097	79.8693303	60.4948783	-30.0824796	10^{-4}
234855	79.8733399	60.4979224	-30.0839967	10^{-5}
279613	79.8737409	60.4982268	-30.0841484	10^{-6}
324373	79.8737810	60.4982573	-30.0841635	10^{-7}

Метод Гаусса — классический точный метод решения системы линейных алгебраических уравнений (СЛАУ). Рассмотрим систему линейных уравнений с действительными постоянными коэффициентами:

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n = y_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n = y_2 \\ \cdots \\ a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \cdots + a_{nn} \cdot x_n = y_n \end{cases}$$

или в матричной форме

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$\boldsymbol{A} \cdot \boldsymbol{X} = \boldsymbol{Y}$$

Метод Гаусса решения системы линейных уравнений включает в себя 2 стадии:

- последовательное (прямое) исключение;
- обратная подстановка.

Последовательное исключение:

Исключения Гаусса основаны на идее последовательного исключения переменных по одной до тех пор, пока не останется только одно уравнение с одной переменной в левой части. Затем это уравнение решается относительно единственной переменной. Таким образом, систему уравнений приводят к треугольной (ступенчатой) форме. Для этого среди элементов первого столбца матрицы выбирают ненулевой (а чаще максимальный) элемент и перемещают его на крайнее верхнее положение перестановкой строк. Затем нормируют все уравнения, разделив его на коэффициент $\mathbf{a_{i1}}$, где **i**– номер столбца.

$$\begin{cases} x_1 + \frac{a_{12}}{a_{11}} \cdot x_2 + \cdots + \frac{a_{1n}}{a_{11}} \cdot x_n = \frac{y_1}{a_{11}} \\ x_1 + \frac{a_{22}}{a_{21}} \cdot x_2 + \cdots + \frac{a_{2n}}{a_{21}} \cdot x_n = \frac{y_2}{a_{21}} \\ \cdots \\ x_1 + \frac{a_{n2}}{a_{n1}} \cdot x_2 + \cdots + \frac{a_{nn}}{a_{n1}} \cdot x_n = \frac{y_n}{a_{n1}} \end{cases}$$

Затем вычитают получившуюся после перестановки первую строку из остальных строк:

$$\begin{cases} x_1 + \frac{a_{12}}{a_{11}} \cdot x_2 + \cdots + \frac{a_{1n}}{a_{11}} \cdot x_n = \frac{y_1}{a_{11}} \\ 0 + \left(\frac{a_{22}}{a_{21}} - \frac{a_{12}}{a_{11}}\right) \cdot x_2 + \cdots + \left(\frac{a_{2n}}{a_{21}} - \frac{a_{1n}}{a_{11}}\right) \cdot x_n = \left(\frac{y_2}{a_{21}} - \frac{y_1}{a_{11}}\right) \\ \cdots \\ 0 + \left(\frac{a_{n2}}{a_{n1}} - \frac{a_{12}}{a_{11}}\right) \cdot x_2 + \cdots + \left(\frac{a_{nn}}{a_{n1}} - \frac{a_{1n}}{a_{11}}\right) \cdot x_n = \left(\frac{y_n}{a_{n1}} - \frac{y_1}{a_{11}}\right) \end{cases}$$

Получают новую систему уравнений, в которой заменены соответствующие коэффициенты.

$$\begin{cases} x_1 + a'_{12} \cdot x_2 + \cdots + a'_{1n} \cdot x_n = y'_1 \\ 0 + a'_{22} \cdot x_2 + \cdots + a'_{2n} \cdot x_n = y'_2 \\ \cdots \\ 0 + a'_{n2} \cdot x_2 + \cdots + a'_{nn} \cdot x_n = y'_n \end{cases}$$

После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают указанный процесс для всех последующих уравнений пока не останется уравнение с одной неизвестной:

$$\begin{cases} x_1 + a'_{12} \cdot x_2 + a'_{13} \cdot x_3 + \cdots + a'_{1n} \cdot x_n = y'_1 \\ 0 + x_2 + a''_{23} \cdot x_3 + \cdots + a''_{2n} \cdot x_n = y''_2 \\ 0 + 0 + x_3 + \cdots + a'''_{3n} \cdot x_n = y'''_3 \\ \dots \\ 0 + 0 + 0 + \cdots + x_n = y_n^{n\prime} \end{cases}$$

Обратная подстановка

Обратная подстановка предполагает подстановку полученного на предыдущем шаге значения переменной $\mathbf{x_n}$ в предыдущие уравнения:

$$x_{n-1} = y_{n-1}^{(n-1)'} - a_{(n-1)n}^{(n-1)\prime} \cdot x_n$$

$$x_{n-2} + a_{(n-2)(n-1)}^{(n-2)\prime} \cdot x_{n-1} = y_{n-2}^{(n-2)'} - a_{(n-2)n}^{(n-2)\prime} \cdot x_n$$

$$\dots$$

$$x_2 + a''_{23} \cdot x_3 + \cdots + a''_{2(n-1)} \cdot x_{n-1} = y''_2 - a''_{2n} \cdot x_n$$

$$x_1 + a'_{12} \cdot x_2 + a'_{13} \cdot x_3 + \cdots + a'_{1(n-1)} \cdot x_{n-1} = y'_1 - a'_{1n} \cdot x_n$$

Эта процедура повторяется для всех оставшихся решений:

$$x_{n-2} = \left(y_{n-2}^{(n-2)'} - a_{(n-2)n}^{(n-2)'} \cdot x_n\right) - a_{(n-2)(n-1)}^{(n-2)\prime} \cdot x_{n-1}$$

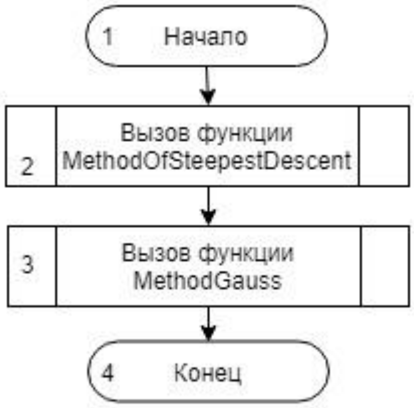
$$\dots$$

$$x_2 + a''_{23} \cdot x_3 + \cdots = (y''_2 - a''_{2n} \cdot x_n) - a''_{2(n-1)} \cdot x_{n-1}$$

$$x_1 + a'_{12} \cdot x_2 + a'_{13} \cdot x_3 + \cdots = (y'_1 - a'_{1n} \cdot x_n) - a'_{1(n-1)} \cdot x_{n-1}$$

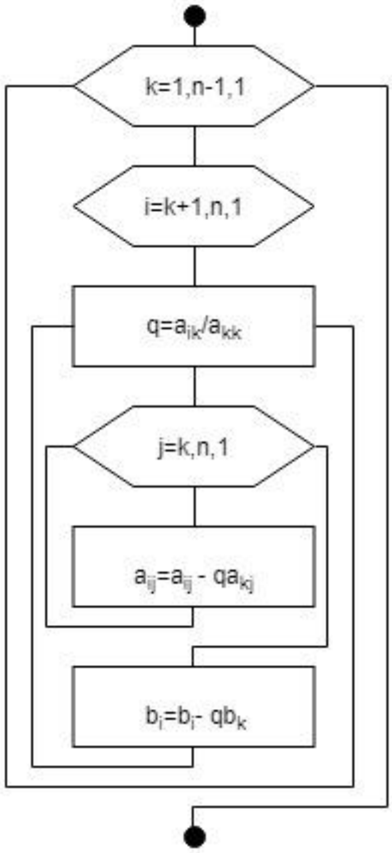
Блок-схемы

Блок-схема Main

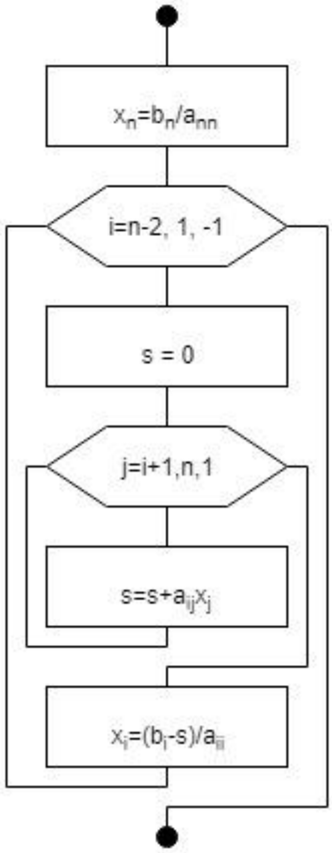


Блок-схема `GaussMethod`

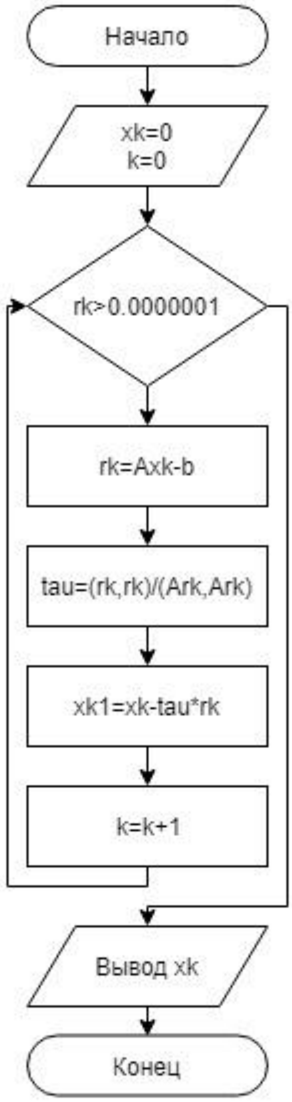
Алгоритм прямого хода



Алгоритм обратного хода



Блок-схема `SpeedyDescentMethod`



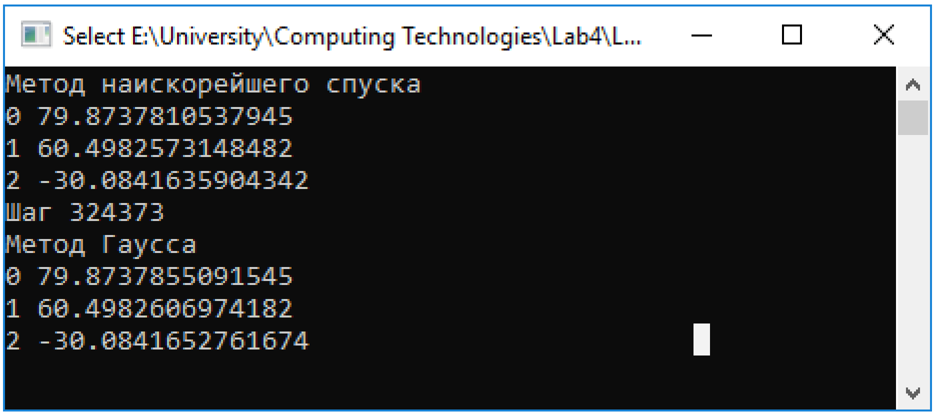
Код программы

```
using System;

namespace Lab4
{
    class Program
    {
        const int size = 3;
        static readonly double[,] A = {
            {1.65, -1.76, 0.77},
            {-1.76, 1.04, -2.61},
            {0.77, -2.61, -3.18}
        };

        static readonly double[] B = {2.15, 0.86, -0.73};

        private static void MethodGauss()
        {
            double[] Q = new double[size];
            double[] X = new double[size];

            for (int k = 0; k < size - 1; k++)
            {
                for (int i = 0; i < size - 1; i++)
                    Q[i] = A[i + 1, k] / A[k, k];

                for (int i = 0, j = k; j < size - 1;)
                {
                    A[j + 1, i] = A[j + 1, i] - Q[j] * A[k, i];

                    if (i == size - 1)
                    {
                        B[j + 1] = B[j + 1] - Q[j] * B[k];
                        j++;
                        i = 0;
                    }
                    else i++;
                }
            }

            X[size - 1] = B[size - 1] / A[size - 1, size - 1];

            double s;
            for (int i = size - 2; ; i--)
            {
                s = 0;

                for (int j = i + 1; j < size; j++)
                    s += A[i, j] * X[j];

                X[i] = (B[i] - s) / A[i, i];

                if (i == 0) break;
            }

            Console.WriteLine("Метод Гаусса");

            for (int i = 0; i < size; i++)
                Console.WriteLine($"{i} {X[i] }");
        }

        private static void MethodOfSteepestDescent(double E)
        {
```

```
            double[] Xp = new double[size], A_Xp = new double[size], rp = new
double[size], A_Transposed_A_r = new double[size], Mu = new double[size], Transposed_A_r
= new double[size];
            double[,] Transposed_A = new double[size, size];
            double[] Xp_next = new double[size];
            double[,] A_ = new double[size, size];
            double mu_, for_mu1, for_mu2;

            int[] t = { 0, 0, 0 };
            int it = 0;

            for (int i = 0; i < size; i++)
            {
                Xp[i] = B[i] / A[i, i];
            }

            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    Transposed_A[i, j] = A[j, i];
                }
            }

            for (it = 0; ; it++)
            {
                for (int i = 0; i < size; i++)
                {
                    A_Xp[i] = 0;

                    for (int j = 0; j < size; j++)
                    {
                        A_Xp[i] += A[i, j] * Xp[j];
                    }

                    rp[i] = A_Xp[i] - B[i];
                }

                for (int i = 0; i < size; i++)
                {
                    for (int j = 0; j < size; j++)
                    {
                        A_[i, j] = 0;

                        for (int k = 0; k < size; k++)
                        {
                            A_[i, j] += A[i, k] * Transposed_A[k, j];
                        }
                    }
                }

                for (int i = 0; i < size; i++)
                {
                    A_Transposed_A_r[i] = 0;

                    for (int j = 0; j < size; j++)
                    {
                        A_Transposed_A_r[i] += A_[i, j] * rp[j];
                    }
                }

                for_mu1 = for_mu2 = 0;

                for (int j = 0; j < size; j++)
                {
```

```
                for_mu1 += rp[j] * A_Transposed_A_r[j];
                for_mu2 += A_Transposed_A_r[j] * A_Transposed_A_r[j];
            }

            mu_ = for_mu1 / for_mu2;

            for (int i = 0; i < size; i++)
            {
                Mu[i] = mu_;
            }

            for (int i = 0; i < size; i++)
            {
                Transposed_A_r[i] = 0;

                for (int j = 0; j < size; j++)
                    Transposed_A_r[i] += Transposed_A[i, j] * rp[j];
            }

            for (int i = 0; i < size; i++)
            {
                Xp_next[i] = Xp[i];
                Xp[i] = Xp[i] - Mu[i] * Transposed_A_r[i];
            }

            int q = 0;



            for (int i = 0; i < size; i++)
            {
                if (Math.Abs(rp[i]) < E && (Xp_next[i] - Xp[i]) < E) t[i] = 1;
                q += t[i];
            }

            if (q == size) break;
        }

        Console.WriteLine("Метод наискорейшего спуска");

        for (int i = 0; i < size; i++)
            Console.WriteLine($"{i} {Xp[i] }");

        Console.WriteLine($"Кол-во итераций: {it}");
    }

    static void Main(string[] args)
    {
        MethodOfSteepestDescent(Math.Pow(10, -7));

        MethodGauss();

        Console.ReadKey();
    }
  }
}
```

Результаты работы программы и анализ результатов



Результат работы программы полностью совпал с тестовым примером. Это говорит о том, что программа может использоваться для вычисления других вариантов с заданной точностью и в заданном диапазоне значений функции.

Выводы

Заданная система линейных уравнений имеет 1 корень. Решение ищем с точностью $\varepsilon = 0.0000001$. Систему линейных уравнений решаем методом Гаусса и методом скорейшего спуска. В точных методах решение находится за конечное число действий, но из-за погрешности округления и их накопления прямые методы можно назвать точными, только отвлекаясь от погрешностей округления, а итерационные методы позволяют получить решение с любой заданной точностью. Прямые методы приводят к необходимости затраты большого количества времени при решении системы из-за кубической зависимости числа арифметических операций от размера матрицы, а итерационные методы экономичны, так как время решения, пропорционально квадрату размера матрицы. Поэтому точные методы неэффективны при решении матриц большой размерности в отличие от итерационных методов.

5. Приближенное вычисление определенных интегралов методом Симпсона и методом трапеций. Сравнение методов

Цель работы

Освоение приближенных методов вычисления определенных интегралов детерминированными методами. Получение навыков решения задач вычислительной математики на ЭВМ. Освоение умения анализировать результаты, полученные на компьютере и сравнивать методы.

Вариант 4

$$\int_0^{\pi} \frac{\mathbf{x} * \mathbf{sin}(\mathbf{x})}{\mathbf{1} + \boldsymbol{cos^2 x}} \mathbf{dx}$$

Метод расчета

Формула трапеций

Если на частичном отрезке подынтегральную функцию заменить полиномом Лагранжа первой степени, то есть $f(x) \approx L_{1,i}(x) = \frac{1}{h}[(x - x_{i-1})f(x_i) - (x - x_i)f(x_{i-1})]$, (1) тогда искомый интеграл запишется следующим образом:

$$\int_{x_{i-1}}^{x_i} f(x)dx \approx \frac{1}{h}[f(x_i)\int_{x_{i-1}}^{x_i}(x - x_{i-1})dx - \\ - f(x_{i-1})\int_{x_{i-1}}^{x_i}(x - x_i)dx] = \ldots = \frac{f(x_{i-1}) + f(x_i)}{2}h. \qquad (2)$$

После подстановки выражения (2) в $J = \int_a^b f(x)dx = \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} f(x)dx$, составная формула трапеций примет вид

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n}\frac{f(x_i) + f(x_{i-1})}{2}h = \\ = h[\frac{1}{2}(f_0 + f_n) + f_1 + \ldots f_{n-1}]. \qquad (3)$$

В данном методе элементарная криволинейная трапеция заменяется трапецией (кривая f(x) заменяется хордой CD).

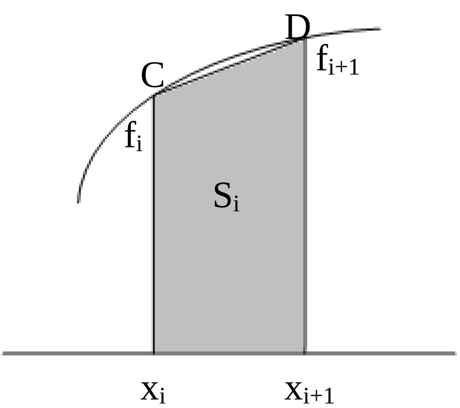


Рис. 7. Оценка элементарной площади S_i трапецией.

Из рисунка видно, что $S_i \approx \frac{f_i + f_{i+1}}{2} \cdot h.$

Погрешность формулы (3) определяется выражением: $|\Psi| \le \frac{h^2(b-a)}{12}$ (4) ,

где $M_2 = \max\limits_{x \in [a,b]} |f''(x)|.$

Таким образом, погрешность метода трапеций $\Psi \sim O(h^2)$.

Формула Симпсона

В этом методе предлагается подынтегральную функцию на частичном отрезке аппроксимировать параболой, проходящей через точки (x_j, $f(x_j)$), где $j = i$-1; i-0.5; i, то есть подынтегральную функцию аппроксимируем интерполяционным многочленом Лагранжа второй степени:

$$f(x) \approx L_{2,i}(x) = \frac{2}{h^2}[(x - x_{i-1/2})(x - x_i) f(x_{i-1}) - \\ - 2(x - x_{i-1})(x - x_i) f(x_{i-1/2}) + (x - x_{i-1})(x - x_{i-1/2}) f(x_i); \quad (5) \\ x \in [x_{i-1}, x_i].$$

Проведя интегрирование, получим:

$$\int\limits_{x_{i-1}}^{x_i} f(x)dx \approx \int\limits_{x_{i-1}}^{x_i} L_{2,i}(x)dx = \frac{h}{6}(f_{i-1} + 4 f_{i-1/2} + f_i), \quad (6)$$

$$h = x_i - x_{i-1}.$$

Это и есть формула Симпсона или формула парабол. На отрезке [a, b] формула Симпсона примет вид

$$\int\limits_a^b f(x)dx \approx \frac{h}{6}[f_0 + f_n + 2(f_1 + f_2 + \ldots + f_{n-1}) + \\ + 4(f_{1/2} + f_{3/2} + f_{5/2} + \ldots + f_{n-1/2})]. \quad (7)$$

Избавимся в выражении (7) от дробных индексов, переобозначив переменные:

$$x_i = a + 0.5h \cdot i; \qquad f_i = f(x_i); \\ i = 0,1,2,\ldots 2n; \qquad h \cdot n = b - a. \quad (8)$$

Тогда формула Симпсона примет вид

$$\int\limits_a^b f(x)dx \approx \frac{b-a}{6n}[f_0 + f_{2n} + 2(f_2 + f_4 + \ldots + f_{2n-2}) + \\ + 4(f_1 + f_3 + f_5 + \ldots + f_{2n-1})]. \quad (9)$$

Погрешность формулы (9) оценивается следующим выражением:

$$|\psi| \le \frac{h^4(b-a)}{2880} M_4, \quad (10)$$

где h·n = b - a, $M_4 = \sup\limits_{x \in [a,b]} \left| f^{IV}(x) \right|$. Таким образом, погрешность формулы Симпсона пропорциональна $O(h^4)$.

Замечание. Следует отметить, что в формуле Симпсона отрезок интегрирования обязательно разбивается на четное число интервалов.

Выбор наименьшего числа разбиений для обеспечения максимальной точности ε

Нам необходимо вычислить интеграл с заданной точностью:

$$|J - I_m| < \varepsilon,$$

где J – истинное значение интеграла, I_m – интеграл, вычисленный с помощью формулы Симпсона или формулы трапеций.

Представим наш интеграл в виде:

$$I_m = J + \alpha h^r,$$

где $h = \frac{b-a}{m}$, m – число разбиений, и где для формулы трапеций

$$\alpha = \frac{M_2(b-a)}{12}; r = 2,$$

а для формулы Симпсона

$$\alpha = \frac{M_4(b-a)}{2880}; r = 4.$$

Увеличив число разбиений в 2 раза, получим:

$$I_{2m} = J + \alpha \left(\frac{h}{2}\right)^r.$$

Вычтем из I_m, I_{2m} , получим:

$$I_m - I_{2m} = \alpha \left(\frac{h}{2}\right)^r (2^r - 1).$$

Так как

$$|I_{2m} - J| < \varepsilon \leftrightarrow \left|\alpha \left(\frac{h}{2}\right)^r\right| < \varepsilon,$$

$$\left|\frac{I_m - I_{2m}}{2^r - 1}\right| < \varepsilon$$

$$|I_m - I_{2m}| < \varepsilon |2^r - 1|$$

$$|I_m - I_{2m}| < \varepsilon.$$

Таким образом, задаем, например,

$m = 2$, считаем I_m;

$2m = 4$, считаем I_{2m}.

Тогда, если неравенство $|I_m - I_{2m}| < \varepsilon$ выполняется, то будем считать, что $2m$ - линейное число разбиений. Если это неравенство не выполняется, берем

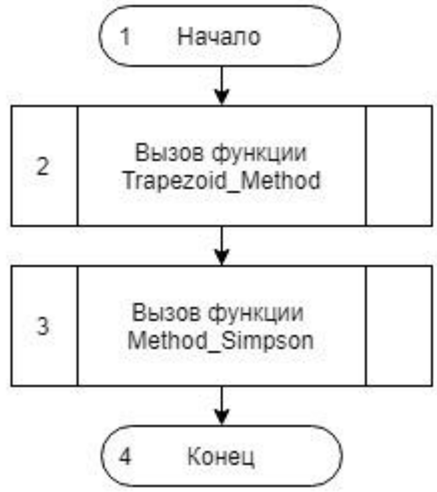
$m = 2m = 4$, считаем I_m;

$2m = 8$, считаем I_{2m}.

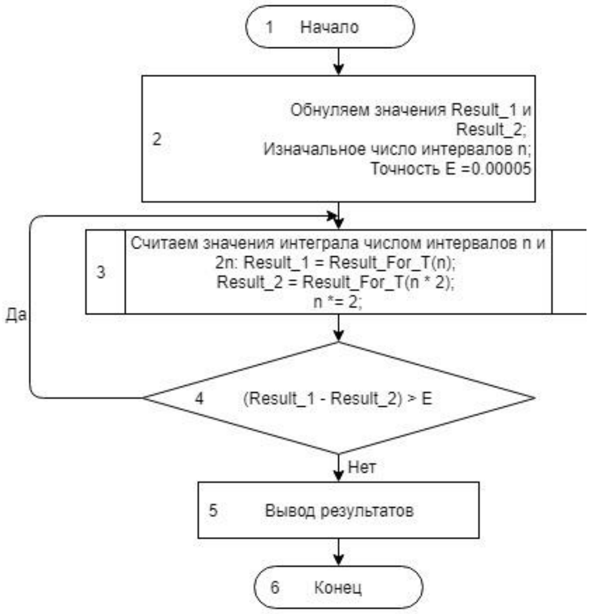
Так продолжаем, пока не обеспечится заданная точность ε.

Блок-схемы

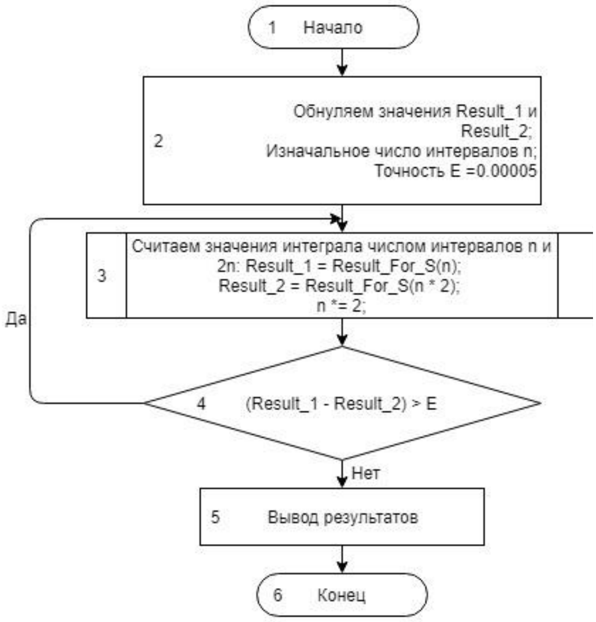
Блок-схема Main



Блок-схема функции Trapezoid_Method



Блок-схема функции Method_Simpson

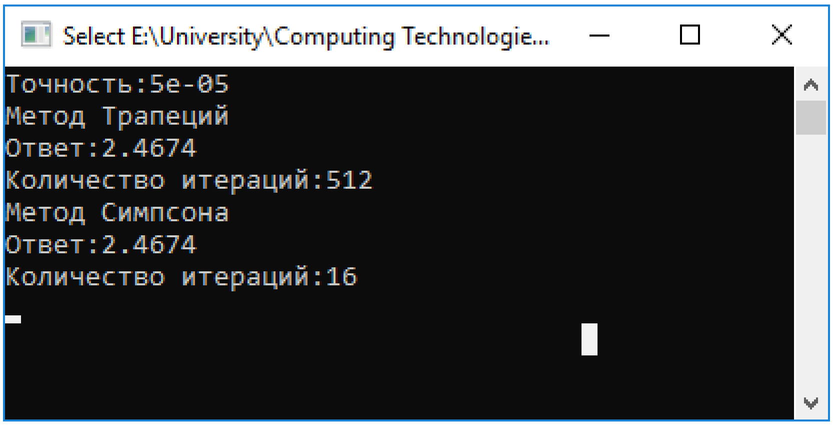


Код программы

```
using System;

namespace Lab5
{
    class Program
    {
        private static double a = 0.0;
        private static double b = Math.PI;

        static double f(double x)
        {
            var result = x.Equals(0.0) ? 0.0 : x * Math.Sin(x) / (1 +
Math.Pow(Math.Cos(x), 2));
            return result;
        }

        static void Trapezoid_Method(int n)
        {
            double Result_1, Result_2;

            double E = 0.00005;

            do
            {
                Result_1 = Result_For_T(n);
                Result_2 = Result_For_T(n * 2);

                n *= 2;

            } while (Math.Abs(Result_1 - Result_2) > E);

            Console.WriteLine(Result_1);
        }

        static double Result_For_T(int n)
        {
            double sum = 0;

            for (var i = 0; i <= n; i++)
            {
                var koef = (i == 0 || i == n) ? 1 : 2;
                sum += f(i * ((b - a) / n)) * koef;
            }

            return ((b - a) / (2.0 * n)) * sum;
        }

        static void Method_Simpson(int n)
        {
            double Result_1, Result_2;

            double E = 0.00005;

            do
            {
                Result_1 = Result_For_S(n);
                Result_2 = Result_For_S(n * 2);

                n *= 2;

            } while (Math.Abs(Result_1 - Result_2) > E);
```

```
            Console.WriteLine(Result_1);
        }

        static double Result_For_S(int n)
        {
            double sum = 0;

            for (var i = 0; i <= n; i++)
            {
                var koef = (i == 0 || i == n) ? 1 : (i % 2 == 1) ? 4 : 2;
                sum += f(i * ((b - a) / n)) * koef;
            }

            return ((b - a) / (3.0 * n)) * sum;
        }

        static void Main(string[] args)
        {
            var n = 2;

            Console.WriteLine("Метод трапеций");
            Trapezoid_Method(n);

            Console.WriteLine("\nМетод Симпсона");
            Method_Simpson(n);

            Console.ReadLine();
        }
    }
}
```

Результаты работы программы и анализ результатов



Результат работы программы полностью совпал с тестовым примером. Это говорит о том, что программа может использоваться для вычисления других вариантов с заданной точностью и в заданном диапазоне значений функции.

Выводы

Приближенное решение ищем с точностью $\varepsilon = 0.00005$ методом трапеции и методом Симпсона. Точность метода Симпсона (парабол) выше точности метода трапеций для заданного n (формула Симпсона имеет 4-ый порядок точности, а формула трапеции имеет 2-ой порядок точности), так что его использование предпочтительнее.

Список литературы

1. **Косарев В.И.** 12 лекций по вычислительной математике (вводный курс). М.: Физматкнига. 2013. 240 с.
2. **Эндрю Троелсен** Язык программирования C# 5.0 и платформа .NET 4.5. М.: ООО “И. Д. Вильямс”. 2013. 1312 с.