**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

|  |
| --- |
|  |

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ЮЖНО-РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ**

**ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (НПИ) им. М.И. Платова»**

|  |
| --- |
|  |

**ФАКУЛЬТЕТ** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**НАПРАВЛЕНИЕ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ОТЧЕТ**

**по лабораторным работам**

**по дисциплине: \_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

Выполнил студент \_\_\_\_\_\_курса, группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Фамилия, имя, отчество

Принял \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Должность, звание Фамилия, имя, отчество

Работа принята «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

Новочеркасск 2024 г.

**РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ: ИТЕРАЦИОННЫЕ МЕТОДЫ**

**Цели:**

1. Реализовать метод Якоби решения системы линейных алгебраических уравнений.

2. Реализовать метод сопряженных градиентов решения системы линейных алгебраических уравнений.

**1.1 Реализация метода Якоби**

**Исходнные данные:**

Система линейных уравнений Ax=b задается матрицей и вектором

A =

16, -4, -28,

-4, 10, 1,

-28, 1, 69

b = 52, -25, -99

**Код программы:**

using System;

namespace JacobiMethod

{

class Program

{

public static double[] MultiplyMatrix(double[][] matrix, double[] vector)

{

int rows = 3; // Количество строк в матрице

int cols = 3; // Количество столбцов в матрице

double[] result = new double[rows]; // Результирующий вектор

// Умножение

for (int i = 0; i < rows; i++)

{

result[i] = 0; // Инициализация результата

for (int j = 0; j < cols; j++)

{

result[i] += matrix[i][j] \* vector[j];

}

}

return result; // Возвращаем результирующий вектор

}

static void Main(string[] args)

{

Console.WriteLine("\t{ 16, -4, -28 },");

Console.WriteLine("A = \t{ -4, 10, 1 },");

Console.WriteLine("\t{ -28, 1, 69 },\n");

Console.WriteLine("B = \t{ 52, -25, -99 },\n");

// Определение матрицы A и вектора B

double[][] A = new double[][]

{

new double[]{ 16, -4, -28 },

new double[]{ -4, 10, 1 },

new double[]{ -28, 1, 69 }

};

double[] B = { 52, -25, -99 };

// Параметры метода Якоби

double[] x = new double[B.Length]; // Начальное приближение

double tolerance = 1e-10; // Допустимая точность

int maxIterations = 100; // Максимальное число итераций

// Вызов метода Якоби

double[] result = Jacobi(A, B, x, tolerance, maxIterations);

// Вывод результата

Console.WriteLine("Решение СЛАУ методом Якоби:");

foreach (var value in result)

{

Console.WriteLine(value);

}

double[] check = MultiplyMatrix(A, result);

Console.WriteLine("Проверка решения СЛАУ:\n");

Console.Write("{");

foreach (var value in check)

{

Console.Write(value);

Console.Write("\t");

}

Console.Write("}");

Console.ReadLine();

}

static double[] Jacobi(double[][] A, double[] B, double[] x, double tolerance, int maxIterations)

{

int n = B.Length; // Количество уравнений

double[] xNew = new double[n]; // Новый вектор решений

for (int iteration = 0; iteration < maxIterations; iteration++)

{

for (int i = 0; i < n; i++)

{

double sum = 0;

for (int j = 0; j < n; j++)

{

if (j != i)

{

sum += A[i][j] \* x[j];

}

}

xNew[i] = (B[i] - sum) / A[i][i]; // Обновление значения x

}

// Проверка на сходимость

double norm = 0;

for (int i = 0; i < n; i++)

{

norm += Math.Pow(xNew[i] - x[i], 2);

}

norm = Math.Sqrt(norm);

if (norm < tolerance)

{

break; // Достигнута допустимая точность

}

Array.Copy(xNew, x, n); // Копирование нового значения в x для следующей итерации

}

return xNew; // Возвращаем вектор решений

}

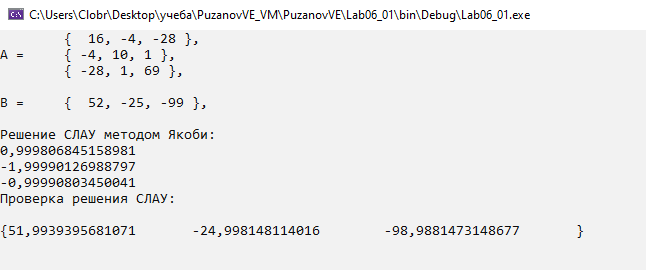
}

}

### Объяснение кода:

Метод Якоби: Реализован с помощью итерационного алгоритма, который обновляет решение на каждом шаге, пока не будет достигнута заданная точность (tol).

**Пример работы:**



**2.1 Реализация метода сопряженных градиентов**

**Исходнные данные:**

Система линейных уравнений Ax=b задается матрицей и вектором

A =

16, -4, -28,

-4, 10, 1,

-28, 1, 69

b = 52, -25, -99

**Код программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PuzanovVE.NM

{

/// <summary>

/// Метод сопряженных градиентов для решения СЛАУ.

/// </summary>

public class ConjugateGradientSolver : SlauSolver

{

/// <summary>

/// Решает СЛАУ Ax = b методом сопряженных градиентов.

/// </summary>

/// <param name="A">Матрица коэффициентов.</param>

/// <param name="b">Правая часть.</param>

/// <param name="epsilon">Требуемая точность вычислений.</param>

/// <returns>Вектор неизвестных - решение СЛАУ.</returns>

public override double[] Solve(double[][] A, double[] b)

{

double epsilon = 0.0001;

int n = b.Length; // Определяем размерность системы

double[] x = new double[n]; // Начальное приближение x = 0

double[] r = new double[n]; // Вектор-невязка r = b - Ax

double[] p = new double[n]; // Вектор-градиент p = r

// Шаг 0: Вычисляем начальную невязку

r = Subtract(b, Multiply(A, x)); // r(0) = b - Ax(0)

p = (double[])r.Clone(); // p(0) = r(0)

double rsOld = Dot(r, r); // ||r(0)||^2

double rsNew;

// Итерационный процесс

for (int k = 0; k < n; k++)

{

// 3.1 Определяем величину шага

double[] Ap = Multiply(A, p); // A \* p(k)

double alpha = rsOld / Dot(p, Ap); // λ(k)

// 3.2 Вычисляем k-е приближение

for (int i = 0; i < n; i++)

{

x[i] += alpha \* p[i]; // x(k+1) = x(k) + λ(k) \* p(k)

}

// 3.3 Корректируем невязку

for (int i = 0; i < n; i++)

{

r[i] -= alpha \* Ap[i]; // r(k+1) = r(k) - λ(k) \* A \* p(k)

}

// 3.4 Проверка относительной невязки

rsNew = Dot(r, r); // ||r(k+1)||^2

if (Math.Sqrt(rsNew) <= epsilon)

{

break; // Условие выполнено, решение получено

}

// 3.5 Вычисляем коэффициент

double beta = rsNew / rsOld; // ω(k)

// 3.6 Задаем новый вектор-градиент

for (int i = 0; i < n; i++)

{

p[i] = r[i] + beta \* p[i]; // p(k+1) = r(k+1) + ω(k) \* p(k)

}

rsOld = rsNew; // Обновляем значение для следующей итерации

}

return x; // Возвращаем найденное решение

}

/// <summary>

/// Умножает матрицу на вектор.

/// </summary>

public static double[] Multiply(double[][] A, double[] x)

{

int n = A.Length; // Размерность матрицы

double[] result = new double[n]; // Результирующий вектор

for (int i = 0; i < n; i++)

{

result[i] = 0; // Инициализация элемента

for (int j = 0; j < n; j++)

{

result[i] += A[i][j] \* x[j]; // Суммирование произведений

}

}

return result; // Возвращаем результат

}

/// <summary>

/// Вычисляет скалярное произведение двух векторов.

/// </summary>

public static double Dot(double[] x, double[] y)

{

double result = 0; // Инициализация результата

for (int i = 0; i < x.Length; i++)

{

result += x[i] \* y[i]; // Суммирование произведений

}

return result; // Возвращаем скалярное произведение

}

/// <summary>

/// Вычитает один вектор из другого.

/// </summary>

public static double[] Subtract(double[] a, double[] b)

{

int n = a.Length; // Размерность векторов

double[] result = new double[n]; // Результирующий вектор

for (int i = 0; i < n; i++)

{

result[i] = a[i] - b[i]; // Вычитание соответствующих элементов

}

return result; // Возвращаем результат

}

}

internal class Program

{

public static double[] MultiplyMatrix(double[][] matrix, double[] vector)

{

int rows = 3; // Количество строк в матрице

int cols = 3; // Количество столбцов в матрице

double[] result = new double[rows]; // Результирующий вектор

// Умножение

for (int i = 0; i < rows; i++)

{

result[i] = 0; // Инициализация результата

for (int j = 0; j < cols; j++)

{

result[i] += matrix[i][j] \* vector[j];

}

}

return result; // Возвращаем результирующий вектор

}

static void Main(string[] args)

{

//14 вар

Console.WriteLine("\t{ 16, -4, -28 },");

Console.WriteLine("A = \t{ -4, 10, 1 },");

Console.WriteLine("\t{ -28, 1, 69 },\n");

Console.WriteLine("B = \t{ 52, -25, -99 },\n");

double[][] A = new double[][]

{

new double[] { 16, -4, -28 },

new double[] { -4, 10, 1 },

new double[] { -28, 1, 69 }

};

double[] B = new double[] { 52, -25, -99 };

// Создание экземпляра

ConjugateGradientSolver solver = new ConjugateGradientSolver();

// Вызов метода Solve

double[] result = solver.Solve(A, B);

// Вывод результата

Console.WriteLine("Решение СЛАУ:");

foreach (var value in result)

{

Console.WriteLine(value);

}

double[] check = MultiplyMatrix(A, result);

Console.WriteLine("Проверка решения СЛАУ:\n");

Console.Write("{");

foreach (var value in check)

{

Console.Write(value);

Console.Write("\t");

}

Console.Write("}");

Console.ReadLine();

}

}

}

**Объяснение кода:**

Метод сопряженных градиентов — это итеративный алгоритм для решения систем линейных уравнений вида *Ax*=*b*, где *A* — симметричная положительно определенная матрица.

### Основные этапы:

1. **Инициализация:** Задается начальное приближение *x*0​ и вычисляется начальный остаток *r*0​=*b*−*Ax*0​.
2. **Итерации:**
   * Вычисляется направление градиента *pk*​=*rk*​+*βk*−1​*pk*−1​, где *βk*−1​ — коэффициент сопряженности.
   * Обновляется решение: *xk*+1​=*xk*​+*αk*​*pk*​, где *αk*​ — шаг, определяемый методом наименьших квадратов.
   * Обновляется остаток: *rk*+1​=*rk*​−*αk*​*Apk*​.
3. **Завершение:** Процесс продолжается до достижения заданной точности или максимального числа итераций.

### Применение:

Метод эффективен для больших разреженных систем и часто используется в вычислительной математике и оптимизации.

**Пример работы:**

