**Лабораторная работа**

**«Решение СЛАУ методом прогонки»**

**Арсений Анищенко**

**Группа 2**

**Задание**

Разработать программу численного решения методом прогонки СЛАУ вида *Ay=f* (вида (1) в файле «Метод прогонки»).

Матрицу системы задать следующим образом:

*.*

Правую часть *f* задать умножением матрицы *A* на вектор *y=*(1, 2, ... , *N*+1): *f=Ay*.

Для вычислений выбрать параметры:

* *m* – номер в списке студенческой группы;
* *N*+1 – (порядок матрицы) одно из чисел в пределах от 10 до 12;
* *k* – номер студенческой группы.

Программно реализовать (в качестве языка программирования выбрать C или C++) вычисления для рассматриваемого примера. Для вычислений использовать тип float.

В отчете должны быть представлены:

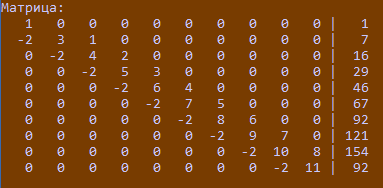
1. Преобразованная матрица после прямой прогонки

2. Вектор приближённого решения *y\**.

3. Относительная погрешность вида , где *y* – точное решение.

**Решение**

Входные данные:



Реализация программы численного решения СЛАУ методом прогонки:

#include <iostream>

#include <vector>

#include <fstream>

#include <iomanip>

#include <algorithm>

#include <math.h>

using namespace std;

vector< vector<float> > arr;

vector< float > f, y, \_y;

int m = 1, n = 9, k = 2;// m - номер в группе, k - номер группы

void do\_sweep() {

//Заполнение a, b, c

vector<float> a, b, c;

a.push\_back(0);

c.push\_back(arr[0][0]);

for (int i = 1; i < n + 1; ++i) {

a.push\_back(-arr[i][i - 1]);

b.push\_back(-arr[i - 1][i]);

c.push\_back(arr[i][i]);

}

b.push\_back(0);

//Вычисление коэффициентов alpha и beta

vector<float> alp, bet;

alp.push\_back(b[0] / c[0]);

bet.push\_back(f[0] / c[0]);

for (int i = 1; i < n; ++i) {

alp.push\_back(b[i] / (c[i] - a[i] \* alp[i - 1]));

bet.push\_back((f[i] + a[i] \* bet[i - 1]) / (c[i] - a[i] \* alp[i - 1]));

}

bet.push\_back((f[n] + a[n] \* bet[n - 1]) / (c[n] - a[n] \* alp[n - 1]));

//Вывод матрицы после прямой прогонки

cout << "\nМатрица после прямой прогонки:\n";

for (int i = 0; i < n + 1; i++)

{

for (int j = 0; j < n + 1; j++)

{

if (i == j) arr[i][j] = 1; else

if (i + 1 == j) arr[i][j] = -alp[i]; else

arr[i][j] = 0;

cout << setw(8) << fixed << setprecision(4) << arr[i][j];

}

cout << setw(2) << "|" << setw(8) << fixed << setprecision(4) << f[i] << endl;

}

//Вычисление вектора приближенного решения

\_y.push\_back(bet[n]);

for (int i = n - 1; i >= 0; --i) {

\_y.push\_back(alp[i] \* \_y[n - i - 1] + bet[i]);

}

reverse(\_y.begin(), \_y.end());

//Вывод вектора приближенного значения

cout << "\nВектор приближенного решения:\n";

for (int i = 0; i < n + 1; ++i) {

cout << setw(8) << fixed << setprecision(9) << \_y[i] << " ";

}

float q = 0, w = 0;

//Подсчет относительной погрешности

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

q += pow(\_y[i] - y[i], 2);

w += y[i] \* y[i];

}

q = sqrt(q);

w = sqrt(w);

double error = q / w;

cout << endl << "Относительная погрешность: " << fixed << setprecision(8) << error << endl;

}

int main() {

setlocale(LC\_ALL, ".1251");

for (int i = 0; i < n + 1; ++i) {

y.push\_back(i + 1);

}

for (int i = 0; i < n + 1; ++i) {

arr.push\_back({});

for (int j = 0; j < n + 1; ++j) {

arr[i].push\_back(0);

}

}

//Заполнение матрицы изначальными значениями

arr[0][0] = m;

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

arr[i + 1][i] = -k;

arr[i][i + 1] = m + i - 1;

arr[i + 1][i + 1] = m + k + i;

}

//Вычисление f

for (int i = 0; i < n + 1; ++i) {

float val = 0;

for (int j = 0; j < n + 1; ++j) {

val += arr[i][j] \* y[j];

}

f.push\_back(val);

}

cout << "Матрица: \n";

for (int i = 0; i < n + 1; ++i) {

for (int j = 0; j < n + 1; j++) {

cout << setw(4) << arr[i][j];

}

cout << setw(2) << "|" << setw(4) << f[i] << endl;

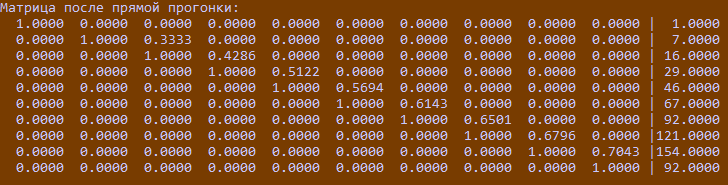
}

do\_sweep();

return 0;

}

Выполнение программы:





**Вывод**

Была реализована программа для численного решения СЛАУ методом прогонки. В приведенном выше примере относительная погрешность составила 0.00000692%, что показывает высокую точность и эффективность метода прогонки для решения поставленной задачи, то есть для решения СЛАУ, где матрица А является трёхдиагональной.