1. **Постановка Задачи.**

Необходимый для выполнения работы теоретический материал и формулы имеются в файле «Метод прогонки».

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

Матрицу системы задать следующим образом:

*.*

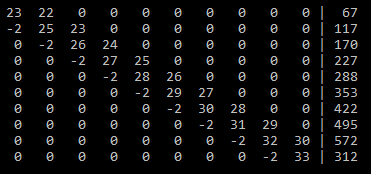
Правую часть *f* задать умножением матрицы *A* на вектор *y=*(1, 2, ... , *N*+1): *f=Ay*.

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.

1. **Входные данные**



1. **Листинг программы**

Файл SoLE.h:

#pragma once

#include <vector>

#include <iostream>

class SoLE

{

public:

SoLE();

~SoLE();

void solve();

friend std::ostream& operator<<(std::ostream&, const SoLE&);

private:

//std::vector<std::vector<float>> A;

std::vector<float> a;

std::vector<float> c;

std::vector<float> b;

std::vector<float> f;

std::vector<float> Y;

static const int m = 23; // Номер в группе

static const int n = 10; // Одно из чисел в пределах от 10 до 12

static const int N = n - 1;

static const int k = 2; // Номер группы

float error(std::vector<float>);

void print(int);

};

Файл SoLE.cpp:

#include "SoLE.h"

#include <algorithm>

#include <iomanip>

SoLE::SoLE()

{

a.resize(N, k);

c.resize(N + 1);

b.resize(N);

c[0] = m;

b[0] = -(m - 1);

for (int i = 0; i < N - 1; ++i)

{

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

b[i + 1] = -(m + i);

}

c[N] = m + k + N - 1;

Y.resize(N + 1);

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

Y[i] = i + 1;

f.resize(n);

f[0] = c[0] + 2 \* -b[0];

for (int i = 1; i < N; ++i)

{

f[i] = -a[i - 1] \* i + c[i] \* (i + 1) + -b[i] \* (i + 2);

}

f[N] = -a[N - 1] \* N + c[N] \* (N + 1);

}

SoLE::~SoLE()

{

}

void SoLE::solve()

{

std::vector<float> y(N + 1);

// Прямая прогонка

float temp;

b[0] /= c[0];

f[0] /= c[0];

for (int i = 1; i < N; ++i)

{

temp = c[i] - a[i - 1] \* b[i - 1];

b[i] /= temp;

f[i] += a[i - 1] \* f[i - 1];

f[i] /= temp;

}

f[N] += a[N - 1] \* f[N - 1];

f[N] /= c[N] - a[N - 1] \* b[N - 1];

print(2);

// Обратная прогонка

y[N] = f[N];

for (int i = N - 1; i >= 0; --i)

y[i] = b[i] \* y[i + 1] + f[i];

std::cout << std::setprecision(6) << "y\* = (";

std::for\_each(y.begin(), y.end() - 1, [](float val) { std::cout << val << ", "; });

std::cout << y[N] << ")\n"

<< "Relative error: " << std::defaultfloat << error(y) \* 100 << "%\n";

}

float SoLE::error(std::vector<float> x)

{

std::vector<float> tmp(Y);

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

tmp[i] -= x[i];

float max1 = abs(tmp[0]), max2 = abs(Y[0]);

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

{

if (abs(tmp[i]) > max1)

max1 = abs(tmp[i]);

if (abs(Y[i]) > max2)

max2 = abs(Y[i]);

}

return max1 / max2;

}

void SoLE::print(int prec)

{

std::cout << "\nTransformed matrix:\n" << std::setprecision(prec) << std::fixed;

std::cout << std::setw(prec + 3) << 1 << ' ' << std::setw(prec + 3) << b[0];

for (int i = 2; i < n; ++i)

std::cout << ' ' << std::setw(prec + 3) << 0;

std::cout << " |" << std::setw(prec + 3) << f[0] << std::endl;

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

{

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

{

std::cout << std::setw(prec + 3);

if (i == j)

std::cout << 1;

else if (i + 1 == j)

std::cout << b[i];

else

std::cout << 0;

std::cout << ' ';

}

std::cout << '|' << std::setw(prec + 3) << f[i] << std::endl;

}

for (int i = 0; i < n - 2; ++i)

std::cout << std::setw(prec + 3) << 0 << ' ';

std::cout << std::setw(prec + 3) << 0 << ' ' << std::setw(prec +3) << 1 << " |" << std::setw(prec + 3) << f[N] << std::endl;

}

std::ostream & operator<<(std::ostream & os, const SoLE & S)

{

os << std::setw(3) << S.c[0] << ' ' << std::setw(3) << -S.b[0];

for (int i = 2; i < S.n; ++i)

os << std::setw(3) << ' ' << 0;

os << " |" << std::setw(4) << S.f[0] << std::endl;

for (int i = 1; i < S.n - 1; ++i)

{

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

{

os << std::setw(3);

if (i - 1 == j)

os << -S.a[i - 1];

else if (i == j)

os << S.c[i];

else if (i + 1 == j)

os << -S.b[i];

else

os << '0';

os << ' ';

}

os << '|' << std::setw(4) << S.f[i] << std::endl;

}

for (int i = 0; i < S.n - 2; ++i)

os << std::setw(3) << 0 << ' ';

os << std::setw(3) << -S.a[S.N - 1] << ' ' << std::setw(3) << S.c[S.N] << " |" << std::setw(4) << S.f[S.N];

return os;

}

Файл lab3.cpp:

#include <iostream>

#include "SoLE.h"

#include <iomanip>

using namespace std;

int main()

{

SoLE S;

cout<< S << endl;

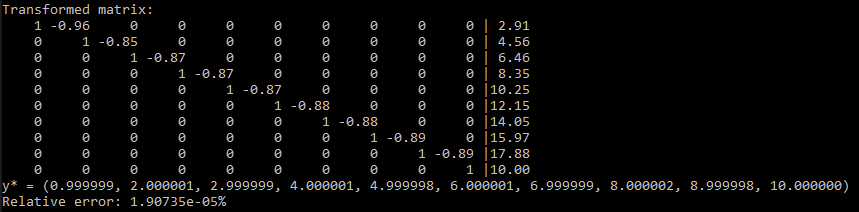
S.solve();

system("pause");

return 0;

}

1. **Выходные данные**



1. **Выводы**

Метод прогонки так же, как и метод Гаусса без выбора ведущего элемента, демонстрирует высокую точность результатов, если у исходной матрицы имеется диагональное преобладание.