1. **Постановка Задачи.**

Пусть дана система линейных алгебраических уравнений вида

*a*11*x*1 + *a*12*x*2 +  … + *a*1n*xn* = *f*1,

*a*21*x*2 + *a*22*x*2 +  … + *a*2n*xn* = *f*2,

. . . . . . . . . . . . . . . . . . . . . . . .

*an*1*x*1 + *an*2*x*2 +  … + *annxn* = *fn* .

Для всех заданий лабораторной работы:

Задать матрицу системы:

* недиагональные элементы *ai,j*, *i≠j*, выбираются из чисел 0, –1, –2, –3, *–*4 произвольным образом;
* *ai,i=*, 2≤*i*≤*n*;
* *a*11*=*+1;

матрица системы имеет диагональное преобладание, для первого уравнения преобладание строгое.

Матрица генерируется один раз, для всех заданий она одна и та же.

Задать правую часть *f* умножением матрицы *A* на вектор *x=*(*m*, *m*+1, ... , *n*+*m*–1): *f=Ax*.

Для вычислений выбрать параметры:

* *m* – номер в списке студенческой группы;
* *n* – одно из чисел в пределах от 10 до 12.

В качестве языка программирования выбрать C или C++, для вычислений использовать тип float.

Выход из итерационного процесса выполнять, если <ε, либо если *k*>*kmax*. Задать ε*=*0,0001, *kmax=*1000.

Вывести на печать полученный приближенный вектор решений и номер итерации, при которой достигнута требуемая точность. Предусмотреть сообщение о выходе из итерационного процесса из-за превышения допустимого максимального количества итераций; в этом случае вывести на печать приближенный вектор решений, полученный на итерации *kmax*.

**Задание 1.** Разработать программу численного решения СЛАУ методом Якоби:

(*i*)*=**,*

*i=* 1*,* 2, …, *n*, *k =* 0, 1, 2, …

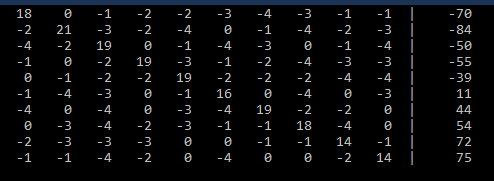
**Задание 2.** Разработать программу численного решения СЛАУ методом релаксации:

(*i*)*=* (*i*) +*,*

*i=* 1*,* 2, …, *n*, *k =* 0, 1, 2, …

Рассмотреть три случая: ω*=*0,5, ω*=*1 (это метод Зейделя), ω*=*1,5.

1. **Входные данные**



1. **Листинг программы**

Файл SoLE.h:

#pragma once

#include <vector>

#include <iostream>

class SoLE

{

public:

SoLE();

SoLE(const SoLE&);

~SoLE();

void setOmega(float val);

void solveTask1();

void solveTask2();

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

private:

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

std::vector<float> f;

std::vector<float> X;

float omega;

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 k = 2; // Номер группы

static const int k\_max = 1000;

static const float eps;

bool is\_iter\_proc\_over(std::vector<float>, std::vector<float>, int);

};

Файл SoLE.cpp:

#include "SoLE.h"

#include <iomanip>

const float SoLE::eps = 0.0001f;

SoLE::SoLE() : omega(NULL)

{

srand(time(NULL));

A.resize(n);

for (auto& v : A)

v.resize(n, 0);

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

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

if (i != j)

A[i][j] = -rand() % 5;

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

{

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

A[i][i] -= A[i][j];

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

A[i][i] -= A[i][j];

}

A[0][0] += 1;

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

X.push\_back(m + i);

f.resize(n, 0);

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

{

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

f[i] += A[i][j] \* X[j];

}

}

SoLE::SoLE(const SoLE & S)

{

A = S.A;

f = S.f;

X = S.X;

}

SoLE::~SoLE()

{

}

void SoLE::setOmega(float val)

{

omega = val;

}

void SoLE::solveTask1()

{

std::vector<float> x\_prev, x(f);

int k = 0;

do

{

x\_prev = x;

for (float &val : x)

val = 0;

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

{

x[i] += f[i];

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

x[i] -= A[i][j] \* x\_prev[j];

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

x[i] -= A[i][j] \* x\_prev[j];

x[i] /= A[i][i];

}

++k;

} while (!is\_iter\_proc\_over(x, x\_prev, k));

if (k > k\_max)

{

std::cout << "k > k\_max\n";

x = x\_prev;

}

else

std::cout << "k = " << k << std::endl;

std::cout << "x = (";

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

std::cout << x[i] << ", ";

std::cout << x[n - 1] << ")";

}

void SoLE::solveTask2()

{

std::vector<float> x\_prev, x(f);

int k = 0;

do

{

x\_prev = x;

for (float &val : x)

val = 0;

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

{

x[i] += f[i];

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

x[i] -= A[i][j] \* x[j];

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

x[i] -= A[i][j] \* x\_prev[j];

x[i] /= A[i][i];

x[i] \*= omega;

x[i] += (1 - omega)\*x\_prev[i];

}

++k;

} while (!is\_iter\_proc\_over(x, x\_prev, k));

if (k > k\_max)

{

std::cout << "k > k\_max\n";

x = x\_prev;

}

else

std::cout << "k = " << k << std::endl;

std::cout << "x = (";

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

std::cout << x[i] << ", ";

std::cout << x[n - 1] << ")";

}

bool SoLE::is\_iter\_proc\_over(std::vector<float> x1, std::vector<float> x2, int k)

{

if (k > k\_max)

return true;

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

if (std::abs(x1[i] - x2[i]) >= eps)

return false;

return true;

}

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

{

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

{

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

{

os << std::setw(4) << S.A[i][j] << ' ';

}

os << " | " << std::setw(6) << S.f[i] << std::endl;

}

return os;

}

Файл lab4.cpp:

#include <iostream>

#include "SoLE.h"

#include <iomanip>

using namespace std;

int main()

{

SoLE S, S1(S), S2(S), S3(S);

cout << S << endl;

cout << "Task 1: \n";

S.solveTask1();

cout << "\n\n";

cout << "Task 2, omega = 0.5: \n";

S1.setOmega(0.5);

S1.solveTask2();

cout << "\n\n";

cout << "Task 2, omega = 1.0: \n";

S2.setOmega(1);

S2.solveTask2();

cout << "\n\n";

cout << "Task 2, omega = 1.5: \n";

S3.setOmega(1.5);

S3.solveTask2();

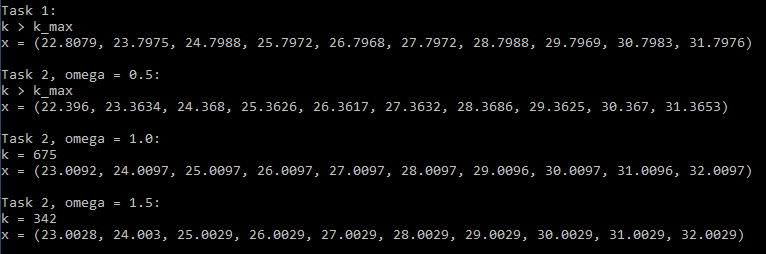
cout << "\n\n";

system("pause");

return 0;

}

1. **Выходные данные**



1. **Выводы**

Для полученных входных данный имеем:

1. Метод Якоби сходиться довольно медленно.
2. Скорость сходимости метода Релаксации существенно зависит от ω

* При ω=0,5, сходимость не достаточно быстрая
* При ω=1,0 (это метод Зейделя), скорости сходимости хватило для завершения итерационного процесса до 1000-й итерации
* При ω=1,5, наибольшая скорость сходимости среди использованный значений ω

как и ожидалось, значения ω близкие к оптимальному лежат в промежутке (1;2)