**Лабораторная работа 5**

**«Итерационные методы решения СЛАУ»**

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.

**Входные данные**

n=12, m=13

**Листинг программы**

#include <iostream>

#include <random>

#include <cmath>

#include <iomanip>

using namespace std;

//Размерность матрицы

int n = 12;

//Номер в списке группы

int m = 13;

//Максимальное число итераций

int kmax = 1000;

//

float e = 0.0001;

//Нахождение погрешности

float Fault(float\* x, float\* b)

{

float max = 0;

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

if (fabs(x[i] - b[i]) > max)

max = fabs(x[i] - b[i]);

return max;

}

//Нахождение случайного числа

int Rand(int L, int R) {

static random\_device rd;

static mt19937 gen(rd());

uniform\_int\_distribution<> dis(L, R);

return dis(gen);

}

//Вывод матрицы

void PrintMatrix(float\*\* A)

{

cout << "A:" << endl;

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

{

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

{

cout << fixed << setprecision(3) << setw(10) << A[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

//Вывод столбца

void PrintArray(float\* x)

{

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

cout << fixed << setprecision(7) << setw(10) << x[i] << " ";

cout << endl << endl;

}

//Умножение матрицы на столбец

void Multiplication(float\*\* A, float\* x, float\* b)

{

float s;

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

{

s = 0;

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

{

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

}

b[i] = s;

}

}

float\* Jacobi(float\*\* A, float\* f)

{

float\*\* B = new float\* [n];

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

B[i] = new float[n];

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

{

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

{

if (i == j)

B[i][j] = 0;

else

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

}

}

float\* b = new float[n];

float\* ans = new float[n];

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

ans[i] = b[i] = f[i];

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

{

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

{

float s = f[i];

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

{

s += B[i][j] \* b[j];

}

ans[i] = s / A[i][i];

}

if (Fault(ans, b) < e)

{

cout << "k = " << k << endl;

delete[]b;

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

delete[]B[i];

return ans;

}

else

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

b[i] = ans[i];

}

delete[]b;

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

delete[]B[i];

cout << "k = 1000" << endl;

return ans;

}

float\* Relaxation(float\*\* A, float\* f, float w)

{

float\*\* B = new float\* [n];

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

B[i] = new float[n];

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

{

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

{

if (i == j)

B[i][j] = 0;

else

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

}

}

float\* b = new float[n];

float\* ans = new float[n];

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

ans[i] = b[i] = f[i];

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

{

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

{

float s = f[i];

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

{

s += B[i][j] \* ans[j];

}

ans[i] = ((s \* w) / A[i][i]) + (1-w) \* b[i];

}

if (Fault(ans, b) < e)

{

cout << "k = " << k << endl;

delete[]b;

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

delete[]B[i];

return ans;

}

else

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

b[i] = ans[i];

}

delete[]b;

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

delete[]B[i];

cout << "k = 1000" << endl;

return ans;

}

int main()

{

setlocale(LC\_ALL, "rus");

//Создание матрицы n\*n

float\*\* A = new float\* [n];

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

{

A[i] = new float[n];

}

//Инициализация

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

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

if (i != j)

A[i][j] = Rand(-4, 0);

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

{

A[i][i] = 0;

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

if (i != j)

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

}

++A[0][0];

PrintMatrix(A);

// Вектор x

float\* x = new float[n];

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

x[i] = m + i;

cout << "x:" << endl;

PrintArray(x);

//Вектор f

float\* f = new float[n];

Multiplication(A, x, f);

cout << "f:" << endl;

PrintArray(f);

cout << "Метод Якоби" << endl;

float\* ans = Jacobi(A, f);

PrintArray(ans);

cout << "Метод релаксации" << endl;

cout << "w = 0.5" << endl;

ans = Relaxation(A, f, 0.5);

PrintArray(ans);

cout << "w = 1" << endl;

ans = Relaxation(A, f, 1);

PrintArray(ans);

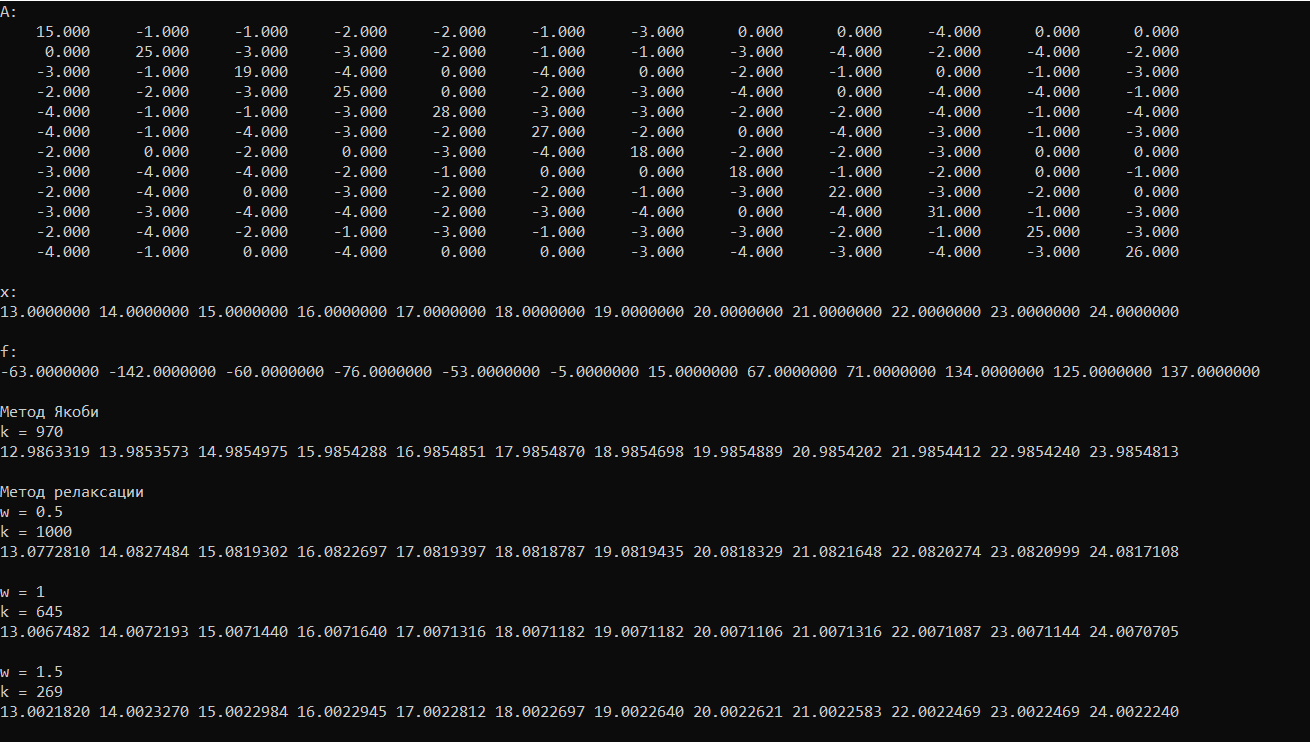
cout << "w = 1.5" << endl;

ans = Relaxation(A, f, 1.5);

PrintArray(ans);

}

**Выходные данные**



**Выводы**

Метод Якоби достаточно точен, но требует большего количества итераций, чем метод релаксации. Наименьшее число итераций было достигнуто при ω=1.5 в методе релаксации. Данные методы возможны, так как матрицы имеют строгое диагональное преобладание.