Белорусский Государственный Университет

Факультет Прикладной Математики и Информатики

Отчет

Лабораторная работа №4

«Итерационные методы решения СЛАУ»

Профессор кафедры вычислительной математики ФПМИ

Лиходед Николай Александрович

Студент 2 группы 2 курса

Сачек Илья Валерьевич

2019 год

**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.

**Входные данные:**

Начальная матрица:

14 -1 -1 -2 -4 -1 -2 -3 -4 -4 -2 -2

-2 39 -4 -1 -3 0 -2 -4 -3 -2 -3 -4

-4 -1 36 -1 -3 -3 0 -3 -2 -3 -2 -3

0 0 -4 28 -1 -2 -3 -1 -1 0 -4 -1

-3 -2 -4 -4 44 -3 -4 -3 -2 0 -4 -4

-2 -3 -2 -3 -2 40 -2 -2 -3 -3 -3 -4

-2 0 -4 0 -3 -2 35 -4 -4 -1 -4 0

-1 -2 -2 0 -4 0 -1 26 0 -3 -1 -1

0 -2 -2 -2 -4 -2 0 -4 31 0 0 -4

-3 -2 0 -1 -1 -1 0 -4 0 28 -2 -3

-3 -2 -3 -4 0 -2 -4 -3 -2 0 35 -1

-1 -3 -3 -3 0 0 0 -2 0 0 -1 24

Вектор f:

-377 47 111 165 181 212 258 293 325 344 435 409

**Листинг программы:**

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

#include <fstream>

float R(float x) {

int k = 1000;

return round(x \* k) / (float)k;

}

float CountInfelicity(std::vector<float> x\_de\_ure, std::vector<float> x\_de\_facto) {

float sum\_x2 = 0, sum\_x = 0;

for (int i = 0; i < x\_de\_facto.size(); i++) {

sum\_x2 += x\_de\_facto[i] \* x\_de\_facto[i];

sum\_x += (x\_de\_facto[i] - x\_de\_ure[i]) \* (x\_de\_facto[i] - x\_de\_ure[i]);

}

std::ofstream fout("errors.txt");

fout << sum\_x / sum\_x2 << std::endl;

fout.close();

return sum\_x / sum\_x2;

}

void LogMatrix(std::vector<std::vector<float>> matrix) {

std::ofstream fout("logs.txt", std::ios::app);

for (const auto& vec : matrix) {

for (auto& el : vec) {

fout << el << '\t';

}

fout << std::endl;

}

fout << std::endl << std::endl;

fout.close();

}

void LogVector(std::vector<float> vector) {

std::ofstream fout("logs.txt", std::ios::app);

for (const auto& el : vector) {

fout << el << '\t';

}

fout << std::endl << std::endl;

fout.close();

}

std::vector<std::vector<float>> MultMatrix(std::vector<std::vector<float>> a, std::vector<std::vector<float>> b\_) {

std::vector<std::vector<float>> c(a.size());

for (int i = 0; i < a.size(); i++) {

c[i].resize(b\_[i].size());

for (int j = 0; j < b\_[i].size(); j++) {

c[i][j] = 0;

for (int k = 0; k < a[i].size(); k++)

c[i][j] += a[i][k] \* b\_[k][j];

}

}

return c;

}

void PrintMatrix(const std::vector<std::vector<float>>& matrix) {

for (const auto& vec : matrix) {

for (const auto& x\_ : vec) {

std::cout << std::setw(7) << std::setprecision(2) << std::left << x\_ << " ";

}

std::cout << std::endl;

}

}

void PrintVector(const std::vector<float>& vector) {

for (const auto& x\_ : vector) {

std::cout << std::setw(7) << std::setprecision(2) << std::left << x\_ << " ";

}

std::cout << std::endl;

}

class Matrix {

public:

// функция умножения матриц

std::vector<float> MultMat(std::vector<float> a, std::vector<std::vector<float>> matrix) {

std::vector<float> c(matrix.size());

for (int i = 0; i < matrix.size(); i++) {

c[i] = 0;

for (int k = 0; k < matrix.size(); k++) {

c[i] += matrix[i][k] \* a[k];

}

}

return c;

}

~Matrix() {

matrix\_.clear();

}

// создание матрицы А

Matrix(float E, int k\_max) : E\_(E), k\_max\_(k\_max) {

srand(time(NULL));

// случайное количество элементов матрицы

int n = rand() % 3 + 10;

matrix\_.resize(n);

for (auto& line : matrix\_) {

line.resize(n);

}

// заполняем матрицу А

// заполнение недиагональных элементов

int p = 0, sum\_1\_col = 0;

for (int i = 0; i < matrix\_.size(); i++) {

for (int j = 0; j < matrix\_.size(); j++) {

if (i != j) {

matrix\_[i][j] = rand() % 5 - 4;

}

}

}

// заполнение диагональных элементов

for (int i = 1; i < matrix\_.size(); i++) {

matrix\_[i][i] = 0;

for (int j = 0; j < matrix\_.size(); j++) {

if (j != i) {

matrix\_[i][i] += -matrix\_[i][j] + 1;

}

}

sum\_1\_col += matrix\_[1][i];

}

matrix\_[0][0] = sum\_1\_col + 1;

int m = 17;

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

x\_.push\_back(m + i);

}

// с помощью умножения матриц находим вектора f

f\_ = MultMat(x\_, matrix\_);

// пишем исходные матрицы в файл

LogMatrix(matrix\_);

}

bool EndAlgorithm(std::vector<float> x\_new) {

float max = 0.0f;

for (int i = 0; i < x\_new.size(); i++) {

if (fabs(x\_curr\_[i] - x\_new[i]) > max) {

max = fabs(x\_curr\_[i] - x\_new[i]);

}

}

return max < E\_ || k\_ > k\_max\_;

}

std::vector<float> SolveJacobi() {

// вывод на экран и в файл

LogMatrix(matrix\_); LogVector(f\_);

std::cout << "Vector x we need to get: " << std::endl;

PrintVector(x\_);

std::cout << "Input matrix: " << std::endl;

PrintMatrix(matrix\_);

std::cout << "Vector f: " << std::endl;

PrintVector(f\_);

// метод Якоби

x\_curr\_ = f\_;

std::vector<float> x\_new(matrix\_.size());

for (k\_ = 0; k\_ < k\_max\_; k\_++) {

for (int i = 0; i < matrix\_.size(); i++) {

float sum = 0.0f; // считаем суммы для вычисления следующего x

for (int j = 0; j < matrix\_.size(); j++) {

if (j != i) {

sum += matrix\_[i][j] \* x\_curr\_[j];

}

}

x\_new[i] = 1 / matrix\_[i][i] \* (f\_[i] - sum);

}

// если норма разности векторов меньше эпсилон - выходим из цикла

if (EndAlgorithm(x\_new)) {

break;

}

x\_curr\_ = x\_new;

}

// выводим сообщение если метод вышел по ограничению количества итераций

if (k\_ == k\_max\_) {

std::cout << "Exceeding the maximum number of iterations!" << std::endl;

}

else {

std::cout << "Number of iterations needed to success: " << k\_ << std::endl;

}

return x\_new;

}

std::vector<float> SolveGaussSeidel(float omega) {

// вывод на экран и в файл

LogMatrix(matrix\_); LogVector(f\_);

std::cout << "Vector x we need to get: " << std::endl;

PrintVector(x\_);

std::cout << "Input matrix: " << std::endl;

PrintMatrix(matrix\_);

std::cout << "Vector f: " << std::endl;

PrintVector(f\_);

// метод Гаусса-Зейделя

x\_curr\_ = f\_;

std::vector<float> x\_new = x\_curr\_;

for (k\_ = 0; k\_ < k\_max\_; k\_++) {

for (int i = 0; i < matrix\_.size(); i++) {

float sum = 0.0f; // считаем суммы для вычисления следующего x

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

sum += matrix\_[i][j] \* x\_new[j];

}

for (int j = i + 1; j < matrix\_.size(); j++) {

sum += matrix\_[i][j] \* x\_curr\_[j];

}

x\_new[i] = (1 - omega) \* x\_curr\_[i] + omega / matrix\_[i][i] \* (f\_[i] - sum);

}

// если норма разности векторов меньше эпсилон - выходим из цикла

if (EndAlgorithm(x\_new)) {

break;

}

x\_curr\_ = x\_new;

}

// выводим сообщение если метод вышел по ограничению количества итераций

if (k\_ == k\_max\_) {

std::cout << "Exceeding the maximum number of iterations!" << std::endl;

} else {

std::cout << "Number of iterations needed to success: " << k\_ << std::endl;

}

return x\_new;

}

private:

std::vector<std::vector<float>> matrix\_;

std::vector<float> f\_, x\_, x\_curr\_;

int k\_max\_, k\_;

float E\_;

};

int main() {

remove("logs.txt");

Matrix m(0.0001, 1000);

std::cout << std::endl;

std::vector<float> answers = m.SolveJacobi();

std::cout << "Vector of answers:" << std::endl;

PrintVector(answers);

std::cout << std::endl << std::endl;

float omega = 0.5f;

answers = m.SolveGaussSeidel(omega);

std::cout << std::endl << "Vector of answers:" << std::endl;

PrintVector(answers);

std::cout << std::endl << std::endl;

omega = 1.0f;

answers = m.SolveGaussSeidel(omega);

std::cout << "Vector of answers:" << std::endl;

PrintVector(answers);

std::cout << std::endl << std::endl;

omega = 1.5f;

answers = m.SolveGaussSeidel(omega);

std::cout << "Vector of answers:" << std::endl;

PrintVector(answers);

return 0;

}

**Выходные данные:**

Ответ метода Якоби после 37 итерации:

17 18 19 20 21 22 23 24 25 26 27 28

Ответ метода релакксации с после 64 итерации:

17 18 19 20 21 22 23 24 25 26 27 28

Ответ метода релакксации с после 22 итерации:

17 18 19 20 21 22 23 24 25 26 27 28

Ответ метода релакксации с после 42 итерации:

17 18 19 20 21 22 23 24 25 26 27 28

**Вывод:**

Данные алгоритмы достаточно точно решают СЛАУ. Кроме того, они позволяют выполнить это с достаточно малой погрешностью. Метод Якоби сошелся за 37 итерацию, метод релаксации при . сошелся за 64 итераций, при за 22 итераций, при за 42 итерацию. Таким образом заключаем, что метод релаксации при сходится лучше всех остальных методов.