**Белорусский государственный университет**

**Факультет прикладной математики и информатики**

Лабораторная работа №1

Решение СЛАУ методом Якоби

Вариант 8

**Выполнил:**

Студент 2 курса 7 группы ПМ ФПМИ

Шевцов Евгений

Преподаватель:

Будник Анатолий Михайлович

**Минск – 2021**

**Алгоритм итерационного метода Якоби**

Решаем СЛАУ Ax=f. Для этого приведём к системе, удобной для итерации, а именно: x = Bx + b, и которой процесс нахождения приближения решения имеет вид:

x(k+1) = B x(k)+𝑏, где x(0) по условию = f, k = 0, 1, …

Решение системы сводится к нахождению матрицы B и вектора b. Метод Якоби является частным случаем метода простых итераций (МПИ), но с определённым алгоритмом нахождения матрицы B и вектора b. Использование данного метода возможно при диагональном преобладании исходной матрицы А. В противном случае метод не будет сходится.

Построим матрицу B. Для начала из матрицы А сделаем матрицу A’, которая будет являться симметрической и положительно определённой, домножением А на АТ. Так же при этом преобразовании домножается вектор неоднородных членов на АТ. В итоге получаем:

A’ = AТА, f’ = ATf

Далее поделим все строки A’ на aii, дабы получить по диагоналям 1. Матрицу B построим из формул:

A’x = f’ => при A’ = E – B: (E - B)x = f’ => x = Bx + b

От сюда:

B = E – A’ =

Вектор b находится следующим образом:

b = (, …, )T

В программе матрица B и вектор b находится покоординатно с помощью соответствующих функций, в которых реализован вышеупомянутый алгоритм.

; ; i, j = [1, n]

Условие, которое мы будем использовать для выхода из цикла итераций ||x(k+1) – x(k)|| < 10-5

**Обоснование итерационного метода Якоби**

Как говорилось ранее, метод Якоби является частным случаем МПИ => для его сходимости требуются выполнения условия сходимости для МПИ. В программе было использовано достаточное условие сходимости МПИ, а именно: хотя бы одна из норм матрицы B меньше единицы.

||B|| = (кубическая норма)

Так же, что тоже было упомянуто, что метод Якоби требует диагонального преобладания в исходной матрице. Это так же было проверено в программе с помощью функции, сравнивающую модуль диагонального элемента с суммой модулей остальных элементов в строке.

|aii| >

**Листинг**

#include <iostream>

#include <iomanip>

#include <vector>

bool diagDomination(const std::vector<std::vector<double>> M) {

for (int i = 0; i < 5; ++i) {

double sum\_i = 0;

for (int j = 0; j < 5; ++j) {

if (i != j) {

sum\_i += abs(M[i][j]);

}

}

if (abs(M[i][i]) <= sum\_i) {

return false;

}

}

return true;

}

std::vector<std::vector<double>> transpositionMatrix(std::vector<std::vector<double>> M) {

for (int i = 0; i < 4; ++i) {

for (int j = i + 1; j < 5; ++j) {

std::swap(M[i][j], M[j][i]);

}

}

return M;

}

std::vector<std::vector<double>> multMatrix(const std::vector<std::vector<double>> M1, const std::vector<std::vector<double>> M2) {

std::vector<std::vector<double>> M3 = {

{0.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 0.0, 0.0, 0.0} };

for (int i = 0; i < 5; ++i) {

for (int j = 0; j < 5; ++j) {

for (int k = 0; k < 5; ++k) {

M3[i][j] += M1[i][k] \* M2[k][j];

}

}

}

return M3;

}

std::vector<double> multVec(const std::vector<std::vector<double>> M, const std::vector<double> f) {

std::vector<double> b = { 0., 0., 0., 0., 0. };

for (int i = 0; i < 5; ++i) {

for (int j = 0; j < 5; ++j) {

b[i] += M[i][j] \* f[j];

}

}

return b;

}

std::vector<std::vector<double>> getB(const std::vector<std::vector<double>> M) {

std::vector<std::vector<double>> B = {

{0.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 0.0, 0.0, 0.0},

{0.0, 0.0, 0.0, 0.0, 0.0} };

for (int i = 0; i < 5; ++i) {

for (int j = 0; j < 5; ++j) {

if (i == j) {

continue;

}

else {

B[i][j] = -M[i][j] / M[i][i];

}

}

}

return B;

}

std::vector<double> getb(const std::vector<std::vector<double>> M, const std::vector<double> f) {

std::vector<double> result = { 0.0, 0.0, 0.0, 0.0, 0.0 };

for (int i = 0; i < 5; ++i) {

result[i] = f[i] / M[i][i];

}

return result;

}

double cubeMatNorm(const std::vector<std::vector<double>> M) {

double max = 0.;

for (int i = 0; i < 5; ++i) {

double strSum = 0.;

for (int j = 0; j < 5; ++j) {

strSum += abs(M[i][j]);

}

if (strSum > max) {

max = strSum;

}

}

return max;

}

double cubeNorm(std::vector<double> v1, std::vector<double> v2) {

double max = 0.;

for (int i = 0; i < 5; ++i) {

if (abs(v1[i] - v2[i]) > max) {

max = abs(v1[i] - v2[i]);

}

}

return max;

}

int main() {

const double EPS = 10E-5;

int count = 0;

std::vector<std::vector<double>> A =

{ {0.7941, 0.0000, -0.2067, 0.1454, 0.2423},

{-0.0485, 0.5168, 0.0000, -0.0985, 0.0323},

{0.0162, -0.1454, 0.9367, 0.0178, 0.0565},

{0.0485, 0.0000, -0.1179, 0.9367, 0.0000},

{0.0323, -0.0485, 0.2342, -0.0194, 0.6783} };

std::vector<double> f = { 1.5569, 2.0656, -2.9054, -8.0282, 3.4819 };

//Протранспонируем А, дабы потом умножить на неё же и получить симметрическую матрицу, не забывая умножить вектор f

std::vector<std::vector<double>> A\_trans = transpositionMatrix(A);

std::vector<std::vector<double>> result\_A = multMatrix(A\_trans, A);

std::vector<double> result\_f = multVec(A\_trans, f);

//Получим матрицу B

std::vector<std::vector<double>> B = getB(result\_A);

std::vector<double> b = getb(result\_A, result\_f);

//Проверим норму B для сходимости

double normB = cubeMatNorm(B);

//Присваиваем значения веткора b начальному приближению

std::vector<double> xk0 = b;

std::vector<double> xk1 = { 0, 0, 0, 0, 0 };

//Пока наша кубическая норма больше 10E-5, проводим итерационный процесс

while (cubeNorm(xk1, xk0) >= EPS) {

if (count != 0) {

xk0 = xk1;

}

count++;

xk1 = multVec(B, xk0);

for (int i = 0; i < 5; ++i) {

xk1[i] += b[i];

}

}

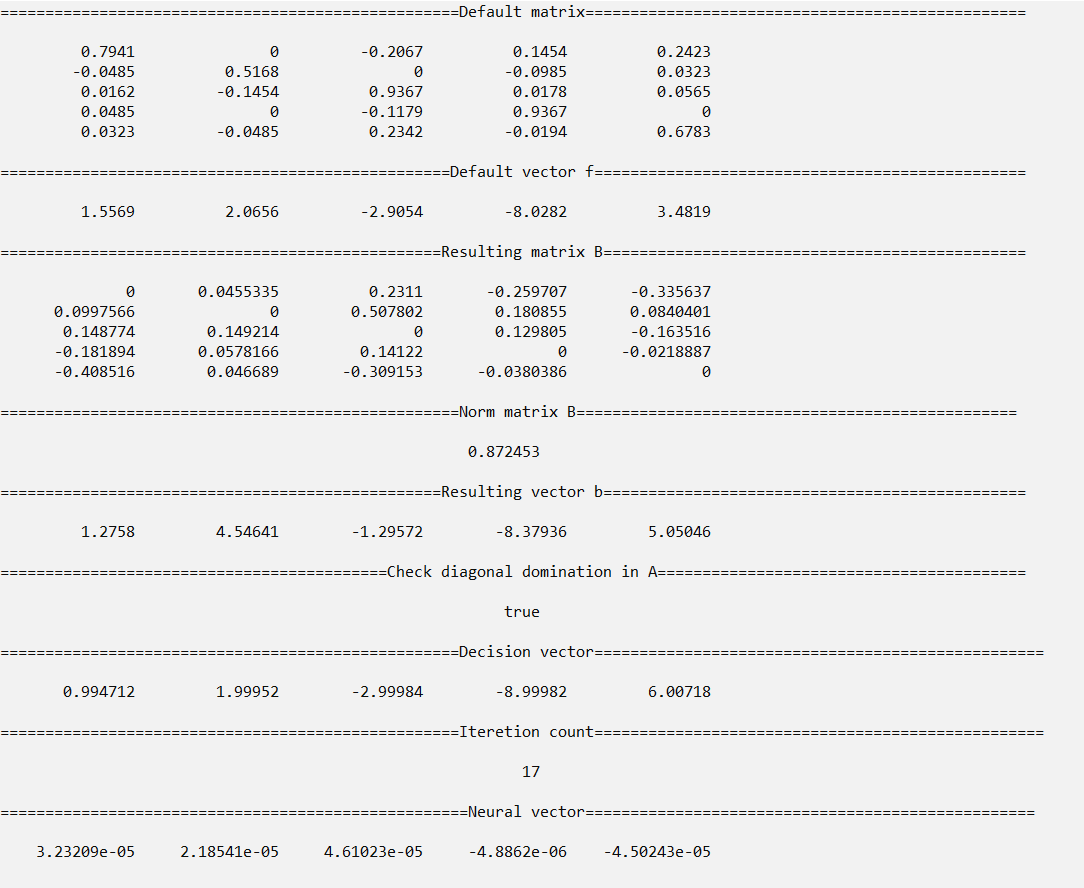
std::vector<double> neuralVector = multVec(A, xk1);

for (int i = 0; i < 5; ++i) {

neuralVector[i] -= f[i];

}

**Выходные данные**



**Вывод**

Полученный вектор решений имеет значение нормы вектора невязки равную 2.18541\*10-5 (невязка считалась у неизменённой матрицы системы А), что на 11 порядков больше, чем в методе Гаусса. Это связано с заданной изначально точностью ε = 10-5 для условия выхода из цикла итераций ||x(k+1) – x(k)|| ≤ ε.

В отличие от точных методов, где погрешность вычислений и округлений возрастает при увеличении размерности системы, у метода Якоби точность вычислений зависит только от вычислений на k+1 итерации и не зависит от предыдущих итераций, что является его преимуществом над точными методами.

Недостатком же, кроме ограничения заданной точности ε, является требования к исходной матрице системы, а так же к матрице B, которые были указаны выше (любая норма матрицы B < 1, диагональное преобладание А).

Чем меньше мы зададим точность ε, тем большее количество итераций нам потребуется для получения приближения решения с данной точностью, следовательно, меньше будет невязка решения. Для данной системы метод Якоби посчитал приближение за 17 итераций.