МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа №1

«Метод Гаусса с выбором главного элемента по матрице»

Задание № СЛАУ-04

Выполнил: Снежко Лев Владимирович,

студент 3 курса, 3 группы

Дисциплина: «Численные методы»

Преподаватель: Будник А.М.

Минск, 2024

**Постановка задачи**

Найти решение системы линейных алгебраических уравнений Ax = b с основной матрицей A системы вида:

и столбца свободных членов b вида:

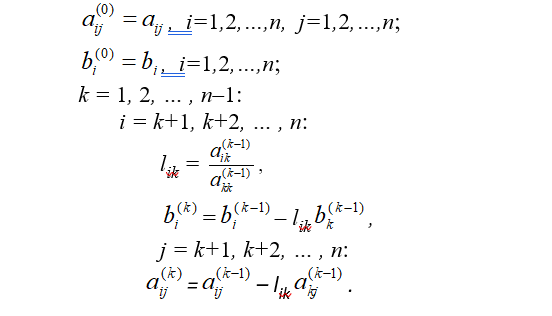
методом Гаусса с выбором главного элемента по матрице на каждом шаге алгоритма.

Используя алгоритм Гаусса, вычислить обратную матрицу A-1 , а также вычислить определитель матрицы А, число обусловленности матрицы А и невязки R1 = Ax - b, R2 = A-1A - E.

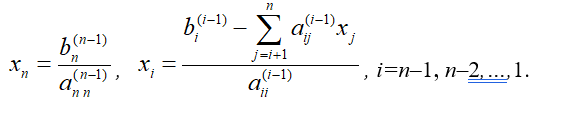
**Алгоритм решения**

В реализации метода Гаусса используем формулы

- в процессе прямого хода:



- обратного хода:

-

Для реализации поиска главного элемента используем алгоритм линейного поиска максимального элемента в матрице (точнее, её миноре). Алгоритм описан в методе find\_main\_item (см. листинг)

При поиске главного элемента может произойти перенумерация переменных. Для того чтобы это не повлияло на результат, все перенумерованные столбцы записываются в стек и затем с помощью этого стека проводятся обратные преобразования над столбцом Х. Реализация описана в методе restore\_x (см. листинг)

Для вычисления А-1 просто N раз применим метод Гаусса к матрице А и столбцам единичной матрицы.

**Листинг**

#include <iostream>

#include <vector>

#include <fstream>

#include <stack>

using namespace std;

const int N = 5;

vector<vector<double>> a;

vector<double> b;

stack<pair<int, int>> columns; //храним пары перенумерованных столбцов в порядке их перенумерации

double det\_a;

int row\_column\_change\_count = 0;

vector<vector<double>> multiply\_matrix(vector<vector<double>> a1, vector<vector<double>> a2);

void swap\_rows(vector<vector<double>>& a, vector<double>& b, int row1, int row2);

void swap\_columns(vector<vector<double>>& a, int column1, int column2);

void find\_main\_item(vector<vector<double>>& a, vector<double>& b, int k);

void restore\_x(vector<double>& x);

vector<double> calculate(vector<vector<double>> a, vector<double> b);

void input(vector<vector<double>>& a, vector<double>& b);

vector<double> calculate\_error(vector<vector<double>> a, vector<double> x, vector<double> b);

void output(vector<double> error);

void output\_matrix(vector<vector<double>> out);

double calculate\_norm(vector<vector<double>> a);

int main()

{

a = vector<vector<double>>(N, vector<double>(N));

b = vector<double>(N);

input(a, b);

vector<double> x = calculate(a, b);

vector<double> error = calculate\_error(a, x, b);

output(error);

//calculate A^(-1)

input(a, b);

vector<vector<double>> E = vector<vector<double>>(N, vector<double>(N, 0));

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

E[i][i] = 1;

}

vector<vector<double>> r\_A = vector<vector<double>>(N, vector<double>(N, 0));

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

r\_A[i] = calculate(a, E[i]);

}

//transpon A^(-1)

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

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

swap(r\_A[i][j], r\_A[j][i]);

}

}

vector<vector<double>> R = vector<vector<double>>(N, vector<double>(N, 0));

vector<vector<double>> mult = multiply\_matrix(a, r\_A);

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

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

R[i][j] = E[i][j] - mult[i][j];

}

}

output\_matrix(R);

double condition\_number = calculate\_norm(a) \* calculate\_norm(r\_A);

ofstream fout("con\_num.txt");

fout << condition\_number;

fout.close();

return 0;

}

vector<double> calculate(vector<vector<double>> a, vector<double> b) {

//прямой ход

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

find\_main\_item(a, b, k);

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

for (int j = k + 1; j < N; j++) {

a[i][j] -= a[i][k] \* a[k][j] / a[k][k];

}

b[i] -= b[k] \* a[i][k] / a[k][k];

}

}

//calculate determinant A

det\_a = row\_column\_change\_count % 2 == 0 ? 1 : -1;

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

det\_a \*= a[i][i];

}

vector<double> x = vector<double>(N);

x[N - 1] = b[N - 1] / a[N - 1][N - 1];

for (int i = N - 2; i >= 0; i--) {

x[i] = b[i];

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

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

}

x[i] /= a[i][i];

}

restore\_x(x);

return x;

}

void find\_main\_item(vector<vector<double>>& a, vector<double>& b, int k) {

/\*

нужно найти максимальный элемент правого нижнего минора [n-k][n-k] матрицы A

и путем обмена местами строк и столбцов переставить его на позицию A[k][k]

при обмене местами двух столбцов следует сохранять номера столбцов в стек columns

при обмене местами двух строк следует менять местами соответствующие строки и в столбце b

\*/

double max = a[k][k];

int max\_column = k, max\_row = k;

//find max in minor

for (int row = k; row < N; row++) {

for (int column = k; column < N; column++) {

if (a[row][column] > max) {

max = a[row][column];

max\_column = column;

max\_row = row;

}

}

}

/\*

если максимум минора находится в k-ой строке, то не нужно менять местами строки

если максимум минора находится в k-м столбце, то не нужно менять местами столбцы

иначе следует менять местами и строки, и столбцы

\*/

if (max\_column != k) {

swap\_columns(a, k, max\_column);

}

if (max\_row != k) {

swap\_rows(a, b, k, max\_row);

}

}

void restore\_x(vector<double>& x) {

while (!columns.empty()) {

int source = columns.top().first;

int target = columns.top().second;

columns.pop();

swap(x[source], x[target]);

}

}

void swap\_rows(vector<vector<double>>& a, vector<double>& b, int row1, int row2) {

swap(a[row1], a[row2]);

swap(b[row1], b[row2]);

row\_column\_change\_count++;

}

void swap\_columns(vector<vector<double>>& a, int column1, int column2) {

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

swap(a[i][column1], a[i][column2]);

}

columns.push(pair<int, int>(column1, column2));

row\_column\_change\_count++;

}

vector<double> calculate\_error(vector<vector<double>> a, vector<double> x, vector<double> b) {

vector<double> pred\_result = vector<double>(N, 0);

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

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

pred\_result[i] += a[i][j] \* x[j];

}

}

vector<double> error = vector<double>(N);

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

error[i] = b[i] - pred\_result[i];

}

return error;

}

vector<vector<double>> multiply\_matrix(

vector<vector<double>> a1,

vector<vector<double>> a2)

{

vector<vector<double>> result = vector<vector<double>>(a1.size(), vector<double>(a2[0].size()));

for (int row1 = 0; row1 < a1.size(); row1++) {

for (int column2 = 0; column2 < a2[0].size(); column2++) {

for (int item = 0; item < a2[0].size(); item++) {

result[row1][column2] += a1[row1][item] \* a2[item][column2];

}

}

}

return result;

}

void input(vector<vector<double>>& a, vector<double>& b) {

ifstream fin\_a ("A.txt");

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

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

fin\_a >> a[i][j];

}

}

ifstream fin\_b("B.txt");

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

fin\_b >> b[i];

}

}

void output(vector<double> error) {

ofstream fout("out.txt");

for (double d : error) {

fout << d << "\n";

}

fout.close();

}

void output\_matrix(vector<vector<double>> out) {

ofstream fout("out\_R.txt");

for (vector<double> v : out) {

for (double d : v) {

fout << d << " ";

}

fout << "\n";

}

fout.close();

}

double calculate\_norm(vector<vector<double>> a) {

double result = 0;

double sum = 0;

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

sum = 0;

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

sum += abs(a[i][j]);

}

result = max(result, sum);

}

return result;

}

**Результаты и их анализ**

На описанных данных (см. раздел «Постановка задачи») алгоритм вернул следующее значение столбца x

X =

В результате работы описанного алгоритма был получен вектор невязки:

R1 = A\*x - b,

R1 =

||R1|| = 4.44089e-16

И матрица невязки R2.

R2 = E – A-1A

R2 =

||R2|| = 2.35055e-16

Число обусловленности матрицы А

ν(А) = ||A|| \* ||A-1|| = 3.4833

Нетрудно видеть, что ||R2|| < ||R1||. Это происходит потому, что при вычислении R2 в качестве столбца b поочерёдно выступают столбцы единичной матрицы, состоящие, по большей части, из нулей. Эти нули не позволяют расти погрешности вычислений при делении этих элементов на ненулевые элементы матрицы А.