**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧЕРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г. Шухова»**

(БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Дисциплина: вычислительная математика

Лабораторная работа №9

Тема: «Решение систем линейных уравнений методом итераций»

Выполнила студентка группы ВТ-21

Сидорова Ангелина Сергеевна

Проверила: Бондаренко Т.В.

Белгород 2018 г.

**Цель работы:** изучить метод итераций для решения систем линейных уравнений и получить практические навыки его применения.

**Задания к работе**

1. Преобразовать систему линейных уравнений заданного варианта к виду, требуемому для метода итераций (получить матрицу В коэффициентов при неизвестных, для которой выполняется условие диагонального преобладания).

2. Вычислить нормы первую ||В||1 и бесконечную ||В||∞ матрицы В и выбрать из них меньшую. 3. Вычислить количество шагов (итераций) метода, используя априорную оценку.

4. Выбрать начальное приближение метода итераций.

5. Методом итераций получить приближенное решение системы уравнений с заданной точностью ε=0,1.

6. Подсчитать число фактически выполненных итераций.

7. Сравнить количество итераций, полученных применением априорной и апостериорной оценок. 8. Вычислить вектор невязки для полученного приближенного решения.

9. Описать в модуле функцию, выполняющую решение заданной системы линейных уравнений с заданной точностью методом итераций за конечное число шагов.

\* Дополнительное задание: Предусмотреть в программе возможность преобразования системы уравнений к требуемой форме, то есть к системе уравнений с диагональным преобладание.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdio.h"

#include "stdlib.h"

#include "locale.h"

#include "math.h"

//константы ошибок

const short NoSolution = 0; //нет решений

const short Ok = 1; //успешная работа

const short ZERO = 2; //деление на ноль

double EPS = 0.1;

short Err; //переменная ошибок

int countP; //кол-во неизвестных в системе

int countEq; //кол-во уравнений в системе

//копирование tab столодбца матрицы matr в res массив

void copyTabMatrInArr(double\*\* matr, double\* res, int tab);

//условие остановки: 1 - решение найдено с данной точностью, иначе 0

int stop(double\* tmpRes, double\* res);

//решение СЛУ методом итераций

void iterationMethod(double\*\* matr, double\* res);

//точность вычисления

double accuracyCalc(double normB);

//оценка числа итераций

int countIteration(double normB, double normC);

//проверка строки матрицы на диагональное преобладание: 1 - удволетворяет, иначе 0

int proverka(double \*str, int lenStr, int currentElem);

//бесконечная норма вектора начального приближения

double Approximation(double \*\*matr);

//вычисление 1 нормы матрицы

double normFirst(double \*\*matr);

//вычисление бесконечной нормы матрицы

double normEndless(double \*\*matr);

//сложение двух массвов

void additionArr(double \*a, double \*b, double \*res, int size);

//вычитание двух массивов

void subtractionArr(double \*a, double \*b, double \*res, int size);

//вывод матрицы

void outputMatr(double \*\*matr);

//меняем строки матрицы местами

void swapStrMatr(double \*\*matr, int i, int j);

//копирование одного массива во второй

void copyArr(double \*arr1, double \*arr2, int size);

//инициализация матрицы (данные читаются из файла)

double\*\* initMatr(void);

//выделение памяти массиву

double \*getMamorArr(int n);

//очищение памяти массива

void freeMemorArr(double \*arr);

//выделение памяти матрицы

double \*\*getMemorMatr();

//очищение памяти матрицы

void freeMemorMatr(double \*\*matr);

//приведение матрицы в требуемую форму

void pravn(double \*\*matr);

//выражение переменных

void virach\_perem(double \*\*matr);

int main() {

setlocale(0,"Rus");

printf("введите кол-во уравнений в системе и кол-во неизвестных ");

scanf("%d%d", &countP, &countEq);

//матрица коэфицентов СЛУ

double\*\* matr = initMatr();

//результат вектор

double\* res = getMamorArr(countP);

//решение СЛУ методом итераций

iterationMethod(matr, res);

if(Err == Ok) {

printf("\nрезультат:\n");

int i;

for(i = 0; i < countP; i++)

printf("%2.4lf\n", res[i]);

} else

printf("\nрешений нет \n");

freeMemorArr(res);

freeMemorMatr(matr);

return 0;

}

//решение СЛУ методом итераций

void iterationMethod(double\*\* matr, double\* res) {

//проверка на наличие решения

if(countP > countEq) {

Err = NoSolution;

return;

}

//привидение матрицы к требуемому виду

pravn(matr);

outputMatr(matr);

printf("\n");

//выражение неизвестных

virach\_perem(matr);

outputMatr(matr);

printf("\n");

//бесконечная норма вектора начального приближения

double normC = Approximation(matr);

//вычисление 1 нормы матрицы

double normB1 = normFirst(matr);

//вычисление бесконечной нормы матрицы

double normBn = normEndless(matr);

//выбор наименьшей нормы

double normB = (normB1 < normBn) ? normB1 : normBn;

//оценка числа итераций

int countIter = countIteration(normB, normC);

//точность вычисления

EPS = accuracyCalc(normB);

printf("E = %lf\n", EPS);

// для проверки на остановку вычисления приближения

double\* tmpRes = getMamorArr(countP);

copyTabMatrInArr(matr, tmpRes, countP);

//вычисление приближения к точному решению

int q = 0;

while(q < countIter) {

q++;

int i, j;

for(i = 0; i < countP; i++) {

res[i] = 0;

for(j = 0; j < countP; j++)

res[i] += matr[i][j] \* tmpRes[j];

res[i] += matr[i][countP];

}

//условие остановки

if(stop(tmpRes, res)) {

Err = Ok;

freeMemorArr(tmpRes);

return;

} else {

copyArr(res, tmpRes, countP);

int t;

for(t = 0; t < countP; t++ )

printf("%lf ", res[t]);

printf("\n");

}

}

if(q == countIter)

Err = NoSolution;

freeMemorArr(tmpRes);

}

//копирование tab столодбца матрицы matr в res массив

void copyTabMatrInArr(double\*\* matr, double\* res, int tab) {

int i;

for(i = 0; i < countP; i++)

res[i] = matr[i][tab];

}

//условие остановки: 1 - решение найдено с данной точностью, иначе 0

int stop(double\* tmpRes, double\* res) {

double max = 0, f = 0;

int i;

for(i = 0; i < countP; i++)

if((f = fabs(res[i] - tmpRes[i])) > max)

max = f;

return (max <= EPS) ? 1 : 0;

}

//точность вычисления

double accuracyCalc(double normB) {

return (double) (1 - normB)\*EPS / normB;

}

//оценка числа итераций

int countIteration(double normB, double normC) {

return (int) (log(EPS) + log(1 - normB) - log(normC)) / log(normB);

}

//выражение переменных

void virach\_perem(double \*\*matr) {

double \*tmp = getMamorArr(countEq);

//ходим по строкам

int i, j;

for (i = 0; i < countEq; i++) {

tmp[i] = matr[i][i]; // запомнили X

matr[i][i] = 0;

//меняем знаки на противопложные

for (j = 0; j < countEq; j++) {

if (i != j)

matr[i][j] = -matr[i][j];

}

}

for (i = 0; i < countEq; i++) {

for (j = 0; j < (countEq + 1); j++) {

if (tmp[i] != 0)

matr[i][j] = matr[i][j] / tmp[i];

else {

Err = ZERO;

return;

}

}

}

}

//вычисление 1 нормы матрицы (столцы)

double normFirst(double \*\*matr) {

//ищем максимальную сумму по модулю в столбцах

double sum = 0;

double MaxSum = 0;

int i, j;

for (i = 0; i < countEq; i++) {

for (j = 0; j < countEq; j++)

sum += fabs(matr[j][i]);

if (MaxSum < sum) MaxSum = sum;

sum = 0;

}

return MaxSum;

}

//бесконечная норма вектора начального приближения

double Approximation(double \*\*matr) {

double max = 0;

int i;

for (i = 0; i < countEq; i++)

if (fabs(matr[i][countEq]) > max) {

max = fabs(matr[i][countEq]);

}

return max;

}

//вычисление бесконечной нормы матрицы (строки)

double normEndless(double \*\*matr) {

//ищем максимальную сумму по модулю в строках

double sum = 0;

double MaxSum = 0;

int i, j;

for (i = 0; i < countEq; i++) {

for (j = 0; j < countEq; j++)

sum += fabs(matr[i][j]);

if (MaxSum < sum) MaxSum = sum;

sum = 0;

}

return MaxSum;

}

//меняем строки матрицы местами

void swapStrMatr(double \*\*matr, int i, int j) {

double \*tmp = matr[i];

matr[i] = matr[j];

matr[j] = tmp;

}

//копирование arr1 массива в массив arr2

void copyArr(double \*arr1, double \*arr2, int size) {

int i;

for (i = 0; i < size; i++)

arr2[i] = arr1[i];

}

//сложение двух массвов

void additionArr(double \*a, double \*b, double \*res, int size) {

int i;

for (i = 0; i < size; i++)

res[i] = a[i] + b[i];

}

//вычитание двух массивов

void subtractionArr(double \*a, double \*b, double \*res, int size) {

int i;

for (i = 0; i < size; i++)

res[i] = a[i] - b[i];

}

//приведение матрицы в требуемую форму

void pravn(double \*\*matr) {

int flag = 1;

//ходим по строкам матрице

int i, j;

for (i = 0; i < countEq; i++) {

//если строка не удовлетворяет нашему условию

if (!proverka(matr[i], countEq, i)) {

//первая попытка

for (j = i + 1; j < countEq; j++) {

if (proverka(matr[j], countEq, i)) {

swapStrMatr(matr, i, j);

flag = 0;

}

}

//вторая попытка

if (flag) {

// создаём дополнтильеный массив, для экспериментов

double \*arrTmp = getMamorArr(countEq + 1);

copyArr(matr[i], arrTmp, countEq + 1);

//составление нового уровнение

j = 0;

while (!proverka(arrTmp, countEq, i) && (j < countEq)) {

if (j != i) {

additionArr(arrTmp, matr[j], arrTmp, countEq + 1);

}

j++;

}

//если вторая попытка не удалась

if (!proverka(arrTmp, countEq, i)) {

copyArr(matr[i], arrTmp, countEq + 1);

//составление нового уровнение

int j = 0;

while (!proverka(arrTmp, countEq, i) && (j < countEq)) {

//subtractionArr(arrTmp, matr[j++], arrTmp, countEq + 1);

if (j != i) {

subtractionArr(matr[j], arrTmp, arrTmp, countEq + 1);

}

j++;

}

//если ничего не помогло

if (!proverka(arrTmp, countEq, i)) {

printf("Не удалось привести к нормально форме\n");

return;

}

else {

copyArr(arrTmp, matr[i], countEq + 1);

}

} else {

copyArr(arrTmp, matr[i], countEq + 1);

}

}

}

flag = 1;

}

}

//инициализация матрицы (данные читаются из файла)

double\*\* initMatr(void) {

double \*\*matr = getMemorMatr();

FILE \*f = fopen("we.txt", "r");

int i, j;

for (i = 0; i < countEq; i++)

for (j = 0; j <= countEq; j++)

fscanf(f, "%lf", &matr[i][j]);

fclose(f);

return matr;

}

//вывод матрицы

void outputMatr(double \*\*matr) {

int i, j;

for (i = 0; i < countEq; i++) {

for (j = 0; j <= countEq; j++)

printf("%lf ", matr[i][j]);

printf("\n");

}

}

//проверка строки матрицы на диагональное преобладание: 1 - удволетворяет, иначе 0

int proverka(double \*str, int lenStr, int currentElem) {

double sum = 0;

int i;

for (i = 0; i < lenStr; i++)

if (i != currentElem)

sum += fabs(str[i]);

return (fabs(str[currentElem]) > sum) ? 1 : 0;

}

//выделение памяти массиву

double \*getMamorArr(int n) {

double \*arr = (double \*)malloc(n \* sizeof(double));

return arr;

}

//очищение памяти массива

void freeMemorArr(double \*arr) {

free(arr);

}

//выделение памяти матрице

double \*\*getMemorMatr() {

double\*\* matr = (double\*\*)malloc(countEq \* sizeof(double \*));

int i;

for (i = 0; i < countEq; i++)

matr[i] = (double \*)malloc((countEq + 1) \* sizeof(double));

return matr;

}

//очищение памяти матрице

void freeMemorMatr(double \*\*matr) {

int i;

for (i = 0; i < countEq; i++)

free(matr[i]);

free(matr);

}