**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕНЫЙ 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»

Выполнил:

Студент группы ВТ-22

Воскобойников Илья Сергеевич

Проверила: Бондаренко Т. В.

Белгород 2019

**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 диаго нального 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.

**Код программы**

#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  
  
  
  
  
//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** \*));  
 **for** (**int** i = 0; i < countEq; i++)  
 matr[i] = (**double** \*)malloc((countEq + 1) \* **sizeof**(**double**));  
 **return** matr;  
}  
  
//очищение памяти матрице  
**void** freeMemorMatr(**double** \*\*matr) {  
 **for** (**int** i = 0; i < countEq; i++)  
 free(matr[i]);  
 free(matr);  
}  
//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) {  
 **for** (**int** i = 0; i < size; i++)  
 arr2[i] = arr1[i];  
}  
  
//сложение двух массвов  
**void** additionArr(**double** \*a, **double** \*b, **double** \*res, **int** size) {  
 **for** (**int** i = 0; i < size; i++)  
 res[i] = a[i] + b[i];  
}  
  
//вычитание двух массивов  
**void** subtractionArr(**double** \*a, **double** \*b, **double** \*res, **int** size) {  
 **for** (**int** i = 0; i < size; i++)  
 res[i] = a[i] - b[i];  
}  
//копирование tab столодбца матрицы matr в res массив  
**void** copyTabMatrInArr(**double**\*\* matr, **double**\* res, **int** tab) {  
 **for** (**int** i = 0; i < countP; i++)  
 res[i] = matr[i][tab];  
}  
  
  
//проверка строки матрицы на диагональное преобладание: 1 - удволетворяет, иначе 0  
**int** proverka(**double** \*str, **int** lenStr, **int** currentElem) {  
 **double** sum = 0;  
 **for** (**int** i = 0; i < lenStr; i++)  
 **if** (i != currentElem)  
 sum += fabs(str[i]);  
 **return** (fabs(str[currentElem]) > sum) ? 1 : 0;  
}  
//приведение матрицы в требуемую форму  
**void** pravn(**double** \*\*matr) {  
 **int** flag = 1;  
 **for** (**int** i = 0; i < countEq; i++) { //ходим по строкам матрице  
 **if** (!proverka(matr[i], countEq, i)) { //если строка не удовлетворяет нашему условию  
 **for** (**int** j = i + 1; j < countEq; j++)//первая попытка  
 **if** (proverka(matr[j], countEq, i)) {  
 swapStrMatr(matr, i, j);  
 flag = 0;  
 }  
 **if** (flag) {//вторая попытка  
 **double** \*arrTmp = getMamorArr(countEq + 1);// создаём дополнительный массив, для экспериментов  
 copyArr(matr[i], arrTmp, countEq + 1);  
 **int** j = 0;//составление нового уравнение  
 **while** (!proverka(arrTmp, countEq, i) && (j < countEq)) {  
 **if** (j != i)  
 additionArr(arrTmp, matr[j], arrTmp, countEq + 1);  
 j++;  
 }  
 **if** (!proverka(arrTmp, countEq, i)) {//если вторая попытка не удалась  
 copyArr(matr[i], arrTmp, countEq + 1);  
 **int** j = 0;//составление нового уровнение  
 **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++;  
 }  
 **if** (!proverka(arrTmp, countEq, i)) {//если ничего не помогло  
 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;  
 }  
}  
  
//выражение переменных  
**void** virach\_perem(**double** \*\*matr) {  
 **double** \*tmp = getMamorArr(countEq);  
 **for** (**int** i = 0; i < countEq; i++) {//ходим по строкам  
 tmp[i] = matr[i][i]; // запомнили X  
 matr[i][i] = 0;  
 **for** (**int** j = 0; j < countEq; j++)//меняем знаки на противопложные  
 **if** (i != j)  
 matr[i][j] = -matr[i][j];  
 }  
 **for** (**int** i = 0; i < countEq; i++)  
 **for** (**int** 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) {  
 **double** sum = 0;//ищем максимальную сумму по модулю в столбцах  
 **double** MaxSum = 0;  
 **for** (**int** i = 0; i < countEq; i++) {  
 **for** (**int** 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;  
 **for** (**int** 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) {  
 **double** sum = 0;//ищем максимальную сумму по модулю в строках  
 **double** MaxSum = 0;  
 **for** (**int** i = 0; i < countEq; i++) {  
 **for** (**int** j = 0; j < countEq; j++)  
 sum += fabs(matr[i][j]);  
 **if** (MaxSum < sum) MaxSum = sum;  
 sum = 0;  
 }  
 **return** MaxSum;  
}  
  
  
//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);  
}  
  
  
  
  
  
//условие остановки: 1 - решение найдено с данной точностью, иначе 0  
**int** stop(**double**\* tmpRes, **double**\* res) {  
 **double** max = 0, f = 0;  
 **for** (**int** i = 0; i < countP; i++)  
 **if** ((f = fabs(res[i] - tmpRes[i])) > max)  
 max = f;  
 **return** (max <= EPS) ? 1 : 0;  
}  
  
  
  
//инициализация матрицы (данные читаются из файла)  
**double**\*\* initMatr(**void**) {  
 **double** \*\*matr = getMemorMatr();  
 FILE \*f = fopen(**"w.txt"**, **"r"**);  
 **for** (**int** i = 0; i < countEq; i++)  
 **for** (**int** j = 0; j <= countEq; j++)  
 fscanf(f, **"%lf"**, &matr[i][j]);  
 fclose(f);  
 **return** matr;  
}  
  
//вывод матрицы  
**void** outputMatr(**double** \*\*matr) {  
 **for** (**int** i = 0; i < countEq; i++) {  
 **for** (**int** j = 0; j <= countEq; j++)  
 printf(**"%lf "**, matr[i][j]);  
 printf(**"\n"**);  
 }  
}  
  
//решение СЛУ методом итераций  
**void** iterationMethod(**double**\*\* matr, **double**\* res) {  
 **if** (countP > countEq) {//проверка на наличие решения  
 Err = NoSolution;  
 **return**;  
 }  
 pravn(matr);//привидение матрицы к требуемому виду  
 outputMatr(matr);  
 printf(**"\n"**);  
 virach\_perem(matr);//выражение неизвестных  
 outputMatr(matr);  
 printf(**"\n"**);  
 **double** normC = Approximation(matr);//бесконечная норма вектора начального приближения  
 **double** normB1 = normFirst(matr);//вычисление 1 нормы матрицы  
 **double** normBn = normEndless(matr);//вычисление бесконечной нормы матрицы  
 **double** normB = (normB1 < normBn) ? normB1 : normBn;//выбор наименьшей нормы  
 **int** countIter = countIteration(normB, normC);//оценка числа итераций  
 EPS = accuracyCalc(normB);//точность вычисления  
 printf(**"E = %lf\n"**, EPS);  
 **double**\* tmpRes = getMamorArr(countP);// для проверки на остановку вычисления приближения  
 copyTabMatrInArr(matr, tmpRes, countP);  
 **int** q = 0;//вычисление приближения к точному решению  
 **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];  
 }  
 **if** (stop(tmpRes, res)) {//условие остановки  
 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);  
}  
  
  
  
  
  
**int** main() {  
  
 printf(**"введите кол-во уравнений в системе и кол-во неизвестных "**);  
 scanf(**"%d%d"**, &countP, &countEq);  
 **double**\*\* matr = initMatr();//матрица коэфицентов СЛУ  
 **double**\* res = getMamorArr(countP);//результат вектор  
 iterationMethod(matr, res);//решение СЛУ методом итераций  
 **if** (Err == Ok) {  
 printf(**"\nрезультат:\n"**);  
 **for** (**int** i = 0; i < countP; i++)  
 printf(**"%2.4lf\n"**, res[i]);  
 }  
 **else** printf(**"\nрешений нет \n"**);  
 freeMemorArr(res);  
 freeMemorMatr(matr);  
 **return** 0;  
}

