Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Петрозаводский государственный университет»  
Физико-технический институт

Сафонов Владислав Денисович

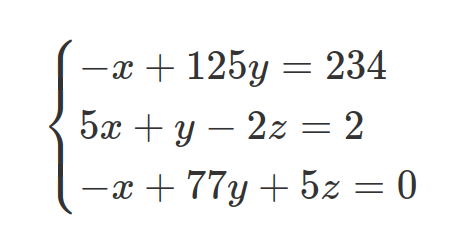
ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

«СЛАУ методом Гаусса»

отчет обучающегося 2 курса очной формы обучения  
по направлению подготовки 09.03.01 "ИВТ"

Петрозаводск 2023 г.

**Цель:** Составить программу на языке Программирования высокого уровня, решающую систему линейных алгебраических уравнений методом Гаусса. Рассчитать невязки. С помощью любой экспертной системы найти решение этой системы.



**Сведения о численном методе:**

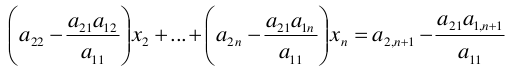
1) Прямой ход.

Идея метода состоит в последовательном исключении неизвестных из системы n линейных

уравнений. На примере первого уравнения системы (2) рассмотрим выражение для x1:



Подставим выражение для x1 во второе и все остальные уравнения системы:





Для расширенной матрицы коэффициентов это означает, что каждый элемент первой строки следует поделить на диагональный элемент, а все остальные строки преобразовать, как показано выше. Таким образом, станут равны нулю все коэффициенты первого столбца, лежащие ниже главной диагонали. Затем аналогичная процедура проводится со второй строкой матрицы и 15 нижележащими строками, при этом первая строка и первый столбец уже не изменяются. И так далее до тех пор, пока все коэффициенты, лежащие ниже главной диагонали, не будут равны нулю.

Общие формулы прямого хода:



k = 1… n, j = 1… n+1. Звездочкой отмечены элементы k-й строки с измененными значениями, которые будут подставлены в следующую формулу. Для определенности будем считать первый индекс – по строкам, второй – по столбцам.



i = k +1… n, j = 1… n+1, k фиксировано в уравнении (3). Для уменьшения количества действий

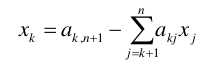
достаточно изменять значения элементов, находящихся выше главной диагонали.

2) Обратный ход.

Второй этап решения СЛАУ методом Гаусса называется обратным ходом и состоит в

последовательном определении xk, начиная с xn, так как для последнего решение фактически

получено. Общая формула:





Таким образом, вычисление корней x происходит за 2/3 n3 арифметических действий.

3) Выбор главного элемента.

Для уменьшения погрешности вычислений следует стремиться к тому, чтобы на главной диагонали матрицы стояли максимальные по модулю значения коэффициентов. Алгоритмически этого можно добиться, переставляя строки таким образом, чтобы на диагонали стоял наибольший по модулю элемент текущего столбца. Такая процедура называется выбором главного элемента и осуществляется всякий раз при переходе к новой строке в прямом цикле метода Гаусса.

4) Погрешность метода. Расчет невязок.

Точность результатов будет определяться только точностью выполнения арифметических операций при преобразовании элементов матрицы, т.е. ошибкой округления. Контроль правильности полученного решения осуществляется подстановкой полученных значений x1...xn в исходную систему уравнений и вычислением невязок, т.е. разностей между правыми и левыми частями уравнений:



Специально отметим, что подставлять найденные значения x следует в исходную (не преобразованную к верхнетреугольному виду) систему.

**Словесно-формульный алгоритм:**

1. Считать из файла количество уравнений num

2. Считать из файла матрицу (num\*num+1) уравнений equMatrix

3. Клонировать матрицу equMatrix в equMatrixCopy

4. Решаем методом Гаусса:

4.1 Прямой ход:

4.1.1 Цикл по i по столбцам от 0 до num-1:

4.1.1.1 Найти максимальный элемент в i-ом столбце и присвоить значение его строки maxRow;

4.1.1.2 Ставим данный элемент на главную диагональ (стр. i и стр. maxRow меняем местами);

4.1.1.3 Цикл по j по строкам i+1 до num:

4.1.1.3.1 Вычислим множитель для сложения двух строк: **multiplier = - matrix[j][i] / matrix[i][i]**;

4.1.1.3.2 Цикл по k по строкам от i, до num+1:

4.1.1.3.2.1 Складываем строки matrix[j][k] и matrix[i][k], умноженную на множитель **multiplier**

4.2 Обратный ход (вычисляем вектор ответов):

4.2.1 Для суммы известной части в данный момент, введем переменную knownSum (куда можно подставить)

4.2.2 Создадим вектор ответов solution

4.2.3 Введем переменную n = rows — 1 (все строки, кроме свободного члена и последнего кф)

4.2.4 Вычисляем значение x[num]:

solution[n] =matrix[n][n+1]) / matrix[n][n];  
 4.2.5 Цикл по i по строкам от n-1, до 0:

4.2.5.1 Зануляем knownSum

4.2.5.2 Цикл по j по столбцам от i+1 до n:

4.2.5.2.1 Умножаем известные корни на соотв-ие им кф. matrix[i][j] \*= solution[j]

4.2.5.2.2 Присваиваем knownSum элемент matrix[i][j]

4.2.5.3 Присваиваем вектору ответу solution[i]

значение (matrix[i][n+1] - knownSum) / matrix [i][i]

4.3. Расчет невязок:

4.3.1 Заводим переменную для левой части, вектор невязки(leftPart, error)

4.3.2 Цикл по i по строкам от 0 до num:

4.3.2.1 Зануляем leftPart

4.3.2.2 Цикл по j по столбцам от 0 до num:

4.3.2.2.1 Прибавляем к leftPart значения matrix[i][j]

4.3.2.3 error[i] = matrix[i][num] — leftPart

5. Вывод ответов в файл (корни + невязки)

**Листинг программы: (gauss.c — файл с алгоритмом)**

#include <stdlib.h>

#include "matrix.h"

double\* gaussMethod(int rows, double \*\* matrix){

double multiplier;

for (int i = 0; i < rows-1; i++){

// Поиск максимального элемента в столбце:

int maxRow = equFindMaxRowInCol(i, rows, matrix);

// Обмен строк:

equRowsReplace(maxRow, i, matrix);

// Прямой ход:

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

multiplier = -(\*(\*(matrix+j)+i) / \*(\*(matrix+i)+i));

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

\*(\*(matrix+j)+k) += \*(\*(matrix+i)+k) \* multiplier;

}

}

}

// Обратный ход

int n = rows-1;

double knownSum;

double \*solution = (double\*)malloc(sizeof(double) \* rows);

\*(solution+n) = \*(\*(matrix+n)+n+1) / \*(\*(matrix+n)+n); // X[n] = A[n][n+1] / A[n][n]

for (int i = n-1; i >= 0; i--){

knownSum = 0;

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

\*(\*(matrix+i)+j) \*= \*(solution+j);

knownSum += \*(\*(matrix+i)+j);

}

\*(solution+i) = (\*(\*(matrix+i)+n+1) - knownSum) / \*(\*(matrix+i)+i);

}

return solution;

}

double\* solutionCheck(int rows, double \*\* source, double \* solution){

double leftPart, \*error;

error = (double\*)malloc(sizeof(double) \* rows);

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

leftPart = 0;

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

leftPart += \*(solution+j) \* \*(\*(source+i)+j);

}

\*(error+i) = \*(\*(source+i)+rows) - leftPart;

}

return error;

}

**(matrix.c — файл с функциями работы с матрицей)**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int equGetNum(FILE \*file\_ptr){

int num;

fscanf(file\_ptr, "%d ", &num);

return num;

}

double\*\* equMatrixMemoryAllocate(int num){

double \*\*tmp = (double\*\*)malloc(sizeof(double\*) \* num); for (int i = 0; i < num+1; i++){

\*(tmp+i) = (double\*) malloc(sizeof(double) \* (num+1));

}

return tmp;

}

double \*\*equMatrixInput(int num, FILE \*file\_ptr){

double \*\*equMatrix = equMatrixMemoryAllocate(num);

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

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

fscanf(file\_ptr, "%lf", \*(equMatrix+i)+j);

}

}

return equMatrix;

}

double \*\*equMatrixClone(int num, double \*\* source){

double \*\*clone = equMatrixMemoryAllocate(num);

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

for (int j = 0; j < num+1; j++){

\*(\*(clone+i)+j) = \*(\*(source+i)+j);

}

}

return clone;

}

void equAnswerOutput(int num, double \*answer\_vect, char \*answer\_type, FILE \*file\_ptr){

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

fprintf(file\_ptr, "%s%d = %lf\n", answer\_type, i+1, \*(answer\_vect+i));

}

}

void equRowsReplace(int row\_a, int row\_b, double \*\*matrix){

double \*tmp = \*(matrix+row\_a);

\*(matrix+row\_a) = \*(matrix+row\_b);

\*(matrix+row\_b) = tmp;

}

int equFindMaxRowInCol(int curr\_col, int rows, double \*\*matrix){

int max\_row = curr\_col;

for (int j = curr\_col+1; j < rows; j++) {

if (fabs(\*(\*(matrix+j)+curr\_col)) > fabs(\*(\*(matrix+max\_row)+curr\_col))) {

max\_row = j;

}

}

return max\_row;

}

**(main.c — главный файл)**

#include "matrix.h"

#include "gauss.h"

int main(int argc, char \*\*argv){

FILE \*file\_write = fopen("answers.txt", "w+");

FILE \*file\_read = fopen(argv[1],"r");

int num = equGetNum(file\_read); // Получаем количество уравнений

double \*\*equMatrix = equMatrixInput(num, file\_read); // Считываем матрицу

double \*\*equMatrixCopy = equMatrixClone(num, equMatrix); // Клонируем матрицу

double \*answers = gaussMethod(num, equMatrix); // Решаем методом Гаусса

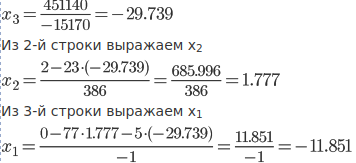
equAnswerOutput(num, answers, "X", file\_write); // Выводим корни в файл

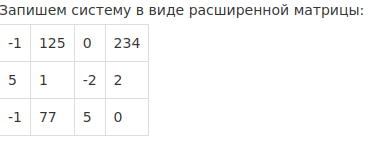
double \*error = solutionCheck(num, equMatrixCopy, answers); // Расчет невязок

equAnswerOutput(num, error, "ERROR", file\_write); // Вывод невязок в файл

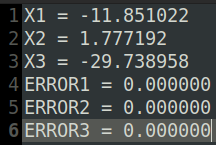
}

**Результаты расчета в экспертной системе:**

****

****

**Результаты работы программы:**



**Вывод:**

С помощью ЯП Си я написал программу, которая считает систему линейных уравнений методом Гаусса. Данные, которые получились у меня, совпадают со значениями полученными в программе Semestr.Ru.