|  |  |  |
| --- | --- | --- |
| ЭМБЛЕМА 41 | **МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  **имени М.В.Ломоносова** | vmk |
| **Факультет вычислительной математики и кибернетики** | | |

**Компьютерный практикум по учебному курсу**

**«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»**

**ЗАДАНИЕ № 1**

**ОТЧЕТ**

**о выполненном задании**

студента 204 учебной группы факультета ВМК МГУ

Столярова Романа Константиновича

(фамилия, имя, отчество студента)

гор. Москва

2020 год

### Оглавление

Цели и задачи………………………………………………………III

Описание метода Гаусса……………………………………..V

Описание метода верхней релаксации……………..VI

Тестирование………………………………………………………VII

Исходные коды программ на языке С………………..X

Выводы……………………………………………………………….XXI

**Цели и задачи**

**Метод Гаусса**

**Цель работы**

Изучить классический метод Гаусса (а также модифицированный метод Гаусса), применяемый для решения системы линейных алгебраических уравнений.

**Постановка задачи**

Дана система уравнений *Ax=f* порядка *n×n* с невырожденной матрицей *A*. Написать программу, решающую систему линейных алгебраических уравнений заданного пользователем размера (*n* – параметр программы) методом Гаусса и методом Гаусса с выбором главного элемента.

**Цели и задачи практической работы**

1. Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента;
2. Вычислить определитель матрицы det(*A*);
3. Вычислить обратную матрицу *А*-1;
4. Определить число обусловленности *МA*=||*A*||×||*A*-1||;
5. Исследовать вопрос вычислительной устойчивости метода Гаусса (при больших значениях параметра *n*);
6. Правильность решения СЛАУ подтвердить системой тестов с помощью онлайн ресурса Wolfram Alpha.

**Метод верхней релаксации**

**Цель работы**

Изучить классические итерационные методы (Зейделя и верхней релаксации), используемые для численного решения систем линейных алгебраических уравнений; изучить скорость сходимости этих методов в зависимости от выбора итерационного параметра.

**Постановка задачи**

Дана система уравнений *Ax=f* порядка *n×n* с невырожденной матрицей *A*. Написать программу численного решения данной системы линейных алгебраических уравнений (*n* – параметр программы), использующую численный алгоритм итерационного метода Зейделя:

,

где  - соответственно диагональная и нижняя треугольные матрицы, - номер текущей итерации;

в случае использования итерационного метода верхней релаксации итерационный процесс имеет следующий вид:

,

где  - итерационный параметр (при  метод верхней релаксации переходит в метод Зейделя). Предусмотреть возможность задания элементов матрицы системы и ее правой части как во входном файле данных, так и путем задания специальных формул.

**Цели и задачи практической работы**

1. Решить заданную СЛАУ итерационным методом Зейделя (или более общим методом верхней релаксации);
2. Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заданной точностью;
3. Изучить скорость сходимости итераций к точному решению задачи (при использовании итерационного метода верхней релаксации провести эксперименты с различными значениями итерационного параметра  (в случае симметрической положительно определенной матрицы системы известно, что для сходимости итераций следует выбирать ; при  метод верхней релаксации совпадает с методом Зейделя);
4. Правильность решения СЛАУ подтвердить системой тестов с помощью онлайн ресурса Wolfram Alpha.

**Описание метода Гаусса**

Алгоритм решения СЛАУ методом Гаусса состоит из 2-х этапов – прямого и обратного хода.

В процессе прямого хода система приводится к эквивалентной путем приведения матрицы системы к верхней треугольной форме. На i-м шаге в строке выбирается первый ненулевой элемент – ведущий, который существует в силу невырожденности матрицы, после чего i-я строка делится на данный элемент. Затем из остальных строк вычитается i-я строка, умноженная на соответственно.

Сложность прямого хода делений и сложений и умножений.

В процессе обратного хода последовательно определяются все неизвестные путем решения уравнений (с одной неизвестной) и подстановки решений из решенного в следующее (получая уравнение с одной неизвестной), процесс начинается с и заканчивается на .

Сложность обратного хода: .

Общая сложность метода Гаусса: .

Однако при работе с плохо обусловленными матрицами данный «простой» алгоритм работает нестабильно из-за неточности округления в вычислениях на компьютере.

Данная проблема решается при помощи модификации этого алгоритма, которая называется метод Гаусса с выбором главного элемента. Его основная идея заключается в том, что на каждой итерации прямого хода выбирается не первый ненулевой элемент, а максимальный по модулю. За счет этого влияние ошибок округления на результат существенно снижается.

**Описание метода верхней релаксации**

Алгоритм метода верхней релаксации заключается в итерационном приближении некоторого вектора к точному решению СЛАУ по формуле:

– вектор -го приближения решения СЛАУ

– матрица, содержащая только элементы главной диагонали матрицы

– матрица, содержащая только элементы матрицы стоящие ниже главной диагонали

– итерационный параметр (при приближение соответствует методу Зейделя)

В алгоритме программной реализации используется следующая формула для пересчёта вектора приближённого решения:

Зафиксируем некое – заданную точность решения, тогда если верно: , то выполнен критерий остановки процесса.

**Дополнительные расчёты**

**Подсчет определителя**

Для вычисления определителя будем использовать модифицированный метод Гаусса с выбором главного элемента. Тогда после приведения к верхней треугольной форме , где – число перестановок столбцов в ходе выделения главного элемента.

**Подсчет обратной матрицы**

Обратную матрицу будем вычислять методом Жордана-Гаусса, согласно которому применяя к единичной матрице все действия из метода Гаусса по приведению матрицы к единичной матрице получится обратная к матрица , такая что .

**Подсчет нормы матрицы и числа обусловленности**

В качестве нормы матрицы выберем норму, подсчитываемую согласно формуле .

Число обусловленности матрицы подсчитывается согласно формуле .

**Тестирование**

Написанные алгоритмы тестировались на примерах, приложенных в *Приложении-1* (вариант 4 для метода Гаусса и вариант 3 для метода верхней релаксации) и в *Приложении-2* (вариант 2). Проверка корректности проводилась в системах Wolfram Alpha и matrixcalc.org.

**Приложение 1-1-4**

1. **СЛАУ:**

Определитель:

Обратная матрица:

Число обусловленности >> 1 – матрица является плохо обусловленной.

Решение методом Гаусса .

Решение методом Гаусса с выбором главного элемента .

Решение Wolfram Alpha .

1. **СЛАУ:**

Определитель:

Обратная матрица не существует, система имеет бесконечно много решений.

Определитель:

Обратная матрица:

Число обусловленности >> 1 – матрица является плохо обусловленной.

Решение методом Гаусса .

Решение методом Гаусса с выбором главного элемента .

Решение Wolfram Alpha .

**Приложение 1-1-3**

1. **СЛАУ:**

Определитель:

Обратная матрица:

Число обусловленности >> 1 – матрица является плохо обусловленной.

Собственные значения – матрица не является ни положительно, ни отрицательно определенной. Значит не выполнено достаточное условие применимости метода верхней релаксации.Решение методом верхней релаксации с и – не найдено. Не удалось подобрать необходимые .

Решение Wolfram Alpha .

1. **СЛАУ:**

Определитель:

Обратная матрица не существует, система не имеет решений.

1. **СЛАУ:**

Определитель:

Обратная матрица:

Число обусловленности .959

Решение методом верхней релаксации с и .

Iterations = 13

Решение методом верхней релаксации с и .

Iterations = 40

Решение методом верхней релаксации с и .5 .

Iterations = 213

Решение Wolfram Alpha .

**Приложение 2-1-2**

Элементы матрицы *A* вычисляются по формуле:

где .

Элементы вектора *f* (вектор правой части системы) задаются формулой , где i = 1...n, n = 20, m = 8.

Решение методом Гаусса

Решение методом Гаусса с выбором главного элемента

Решение методом Гаусса совпадает с корректным решением и решением методом верхней релаксации, что объясняется тем, что данная система обладает диагональным преобладанием при n = 20, m = 8.

Решение методом верхней релаксации с и

Iterations = 4

Решение методом верхней релаксации с и

Iterations = 13

Решение методом верхней релаксации с и

Iterations = 10

**Исходный код программы:**

**Метод Гаусса**

#include <stdio.h>

#include <stdlib.h>

#include <stdio.h>

#include <limits.h>

#include <string.h>

#include <unistd.h>

#include <math.h>

void

swap (double \*a, double \*b)

{

double tmp = \*a;

\*a = \*b;

\*b = tmp;

}

double

\*\*fill\_matrix(double \*\*a, int N, int M) {

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

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

if (i != j) {

a[i][j] = (i + j + 2) / (N + M);

} else {

a[i][j] = N + M \* M + (j + 1) / M + (i + 1) / N;

}

}

}

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

a[i][N] = 200 + 50 \* (i + 1);

}

return a;

}

int

main (void)

{

int N, M;

printf("mode = ");

// s - стандартный, g - с выбором главного элемента

char c;

scanf("%c", &c);

printf("mode matrix = ");

// 1 - ввод матрицы с калвиатуры, 2 - матрица заданная функционально

int mc;

scanf("%d", &mc);

double \*\*a;

if (mc == 1) {

printf("size = ");

scanf("%d", &N);

a = calloc(N, sizeof(double));

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

a[i] = calloc(N + 1, sizeof(double));

}

printf("matrix:\n");

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

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

scanf("%le", &a[i][j]);

}

}

} else if (mc == 2) {

N = 20;

M = 8;

a = calloc(N, sizeof(double));

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

a[i] = calloc(N + 1, sizeof(double));

}

a = fill\_matrix(a, N, M);

}

if (c == 'g') {

double a\_max = 0;

// поиск максимального "ведущего" элемента

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

int i = k;

a\_max = fabs(a[k][k]);

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

if(fabs(a[m][k]) > a\_max) {

i = m;

a\_max = fabs(a[m][k]);

}

}

// проверка на вырожденность матрицы

if (a\_max == 0) {

printf("No solutions!\n");

return 0;

}

// перестановка строк

if (i != k) {

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

swap(&a[i][j], &a[k][j]);

}

}

// преобразования коэфициентов (деление на ведущий)

a\_max = a[k][k];

a[k][k] = 1;

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

a[k][j] = a[k][j] / a\_max;

}

// дальнейшее вычитание строки

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

double save\_elem = a[i][k];

a[i][k] = 0;

if (save\_elem != 0) {

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

a[i][j] = a[i][j] - save\_elem \* a[k][j];

}

}

}

}

} else if (c == 's') {

double a\_base = 0;

// поиск первого ненулевого элемента

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

int i = k;

a\_base = fabs(a[k][k]);

for (int m = k + 1; a\_base == 0 && m < N; m++) {

a\_base = fabs(a[m][k]);

}

// далее все аналогично...

if (a\_base == 0) {

printf("No solutions!\n");

return 0;

}

if (i != k) {

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

swap(&a[i][j], &a[k][j]);

}

}

a\_base = a[k][k];

a[k][k] = 1;

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

a[k][j] = a[k][j] / a\_base;

}

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

double save\_elem = a[i][k];

a[i][k] = 0;

if (save\_elem != 0) {

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

a[i][j] = a[i][j] - save\_elem \* a[k][j];

}

}

}

}

}

// обратный ход

double \*x = calloc(N + 1, sizeof(double));

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

x[i] = 0;

double a\_tmp = a[i][N];

for (int j = N; j > i; j--) {

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

}

x[i] = a\_tmp;

}

printf("solution:\n");

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

printf("x[%d] = %.10lf\n", i + 1, x[i]);

}

return 0;

}

**Метод верхней релаксации**

#include <stdio.h>

#include <stdlib.h>

#include <stdio.h>

#include <limits.h>

#include <string.h>

#include <unistd.h>

#include <math.h>

double\*

next\_iteration(double \*x\_prev, double \*x\_cur, double w, double \*b, int N, double \*\*a)

{

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

double s1 = 0;

double s2 = 0;

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

s1 += a[i][j] / a[i][i] \* x\_cur[j] \* w;

}

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

s2 += a[i][j] / a[i][i] \* x\_prev[j] \* w;

}

x\_cur[i] = b[i] / a[i][i] \* w - s1 - s2 + x\_prev[i];

}

return x\_cur;

}

long long iter = 0;

int

main(void)

{

int N, M;

double w;

double eps;

printf("size = ");

scanf("%d", &N);

//N = 20;

//M = 10;

printf("eps = ");

scanf("%le", &eps);

printf("w = ");

scanf("%le", &w);

printf("matrix:\n");

double \*\*a;

a = calloc(N, sizeof(double));

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

a[i] = calloc(N + 1, sizeof(double));

}

double \*b;

b = calloc(N + 1, sizeof(double));

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

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

scanf("%le", &a[i][j]);

/\*if (i != j) {

a[i][j] = (i + j) / (N + M);

} else {

a[i][j] = N + M \* M + j / M + i / N;

}\*/

}

scanf("%le", &b[i]);

//b[i] = 200 + 50 \* i;

}

double \*x\_cur, \*x\_prev;

x\_cur = calloc(N + 1, sizeof(double));

x\_prev = calloc(N + 1, sizeof(double));

int stop = 0;

while (1) {

// x(i) = x(i+1)

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

x\_prev[i] = x\_cur[i];

}

// вызов функции следующей итерации

x\_cur = next\_iteration(x\_prev, x\_cur, w, b, N, a);

iter++;

// условие остановки

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

if (fabs(x\_prev[i] - x\_cur[i]) >= eps) {

stop = 0;

break;

} else {

stop = 1;

}

}

if (stop == 1) {

break;

}

}

printf("solution:\n");

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

printf("x[%d] = %.10lf\n", i, x\_cur[i]);

}

printf("iterations:\n");

printf("%d\n", iter);

}

**Подсчет определителя**

#include <stdio.h>

#include <stdlib.h>

#include <stdio.h>

#include <limits.h>

#include <string.h>

#include <unistd.h>

#include <math.h>

void

swap (double \*a, double \*b)

{

double tmp = \*a;

\*a = \*b;

\*b = tmp;

}

int

main (void)

{

int N;

printf("size = ");

scanf("%d", &N);

printf("matrix:\n");

double \*\*a;

a = calloc(N, sizeof(double));

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

a[i] = calloc(N, sizeof(double));

}

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

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

scanf("%le", &a[i][j]);

}

}

double det = 1;

double a\_max;

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

int i = k;

a\_max = fabs(a[k][k]);

int sign;

if (a[k][k] > 0) {

sign = 1;

} else {

sign = -1;

}

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

if(fabs(a[m][k]) > a\_max)

{

if (a[m][k] > 0) {

sign = 1;

} else {

sign = -1;

}

i = m;

a\_max = fabs(a[m][k]);

}

}

if (a\_max == 0) {

printf("det:\n");

printf("0\n");

return 0;

}

if (i != k) {

sign \*= -1;

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

{

swap(&a[i][j], &a[k][j]);

}

}

a\_max = a[k][k];

a[k][k] = 1;

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

a[k][j] = a[k][j] / a\_max;

}

det \*= a\_max;

det \*= sign;

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

double save\_elem = a[i][k];

a[i][k] = 0;

if (save\_elem != 0) {

a[i][k] = 0;

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

a[i][j] = a[i][j] - save\_elem \* a[k][j];

}

}

}

}

printf("det:\n");

printf("%.10lf\n", det);

return 0;

}

**Подсчет обратный матрицы и числа обусловленности**

#include <stdio.h>

#include <stdlib.h>

#include <stdio.h>

#include <limits.h>

#include <string.h>

#include <unistd.h>

#include <math.h>

void

swap (double \*a, double \*b)

{

double tmp = \*a;

\*a = \*b;

\*b = tmp;

}

int

main (void)

{

int N;

printf("size = ");

scanf("%d", &N);

printf("matrix:\n");

double \*\*a;

double \*\*b;

double \*\*res;

a = calloc(N, sizeof(double));

b = calloc(N, sizeof(double));

res = calloc(N, sizeof(double));

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

a[i] = calloc(N, sizeof(double));

b[i] = calloc(N, sizeof(double));

res[i] = calloc(N, sizeof(double));

}

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

res[i][i] = 1;

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

scanf("%le", &a[i][j]);

b[i][j] = a[i][j];

}

}

double a\_max;

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

int i = k;

a\_max = fabs(a[k][k]);

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

if(fabs(a[m][k]) > a\_max)

{

i = m;

a\_max = fabs(a[m][k]);

}

}

if (a\_max == 0) {

return 0;

}

if (i != k) {

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

swap(&a[i][j], &a[k][j]);

}

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

swap(&res[i][j], &res[k][j]);

}

}

a\_max = a[k][k];

a[k][k] = 1;

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

a[k][j] = a[k][j] / a\_max;

}

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

res[k][j] = res[k][j] / a\_max;

}

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

double save\_elem = a[i][k];

a[i][k] = 0;

if (save\_elem != 0) {

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

a[i][j] = a[i][j] - save\_elem \* a[k][j];

}

}

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

res[i][j] = res[i][j] - save\_elem \* res[k][j];

}

}

}

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

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

double save\_elem = a[j][i] / a[i][i];

for (int k = N - 1; k >= 0; k--) {

a[j][k] = a[j][k] - save\_elem \* a[i][k];

res[j][k] = res[j][k] - save\_elem \* res[i][k];

}

}

}

printf("solution:\n");

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

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

printf("%.3lf ", res[i][j]);

}

printf("\n");

}

double norm\_1 = 0;

double cur\_1 = 0;

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

cur\_1 = 0;

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

cur\_1 += fabs(res[i][j]);

}

if (cur\_1 > norm\_1) {

norm\_1 = cur\_1;

}

}

double norm\_2 = 0;

double cur\_2 = 0;

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

cur\_2 = 0;

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

cur\_2 += fabs(b[i][j]);

}

if (cur\_2 > norm\_2) {

norm\_2 = cur\_2;

}

}

printf("condition number:\n");

printf("%.3lf\n", norm\_1 \* norm\_2);

return 0;

}

**Выводы**

1. В ходе выполнения данной работы был подробно изучен метод Гаусса решения систем неоднородных уравнений и его модификация с выбором главного элемента. В ходе тестирования было выявлено, что при большом значении числа обусловленности матрицы и при больших порядках n (размер матрицы) метод может быть неустойчивым, допускать ошибки округления при работе с числами с плавающей точкой. Однако часто этого можно избежать используя модификацию с выбором главного элемента и матрицы специального вида.
2. В ходе выполнения второй части работы был подробно изучен метод верхней релаксации и его частный случай при w = 1 – метод Зейделя.

Было выявлено, что количество итераций метода верхней релаксации напрямую зависит от правильного подбора итерационного параметра w. В ходе тестирования выяснилось, что существуют матрицы, для которых данный параметр подобрать невозможно, метод всегда будет расходиться. А для матриц, где это возможно, за счет правильного выбора итерационного параметра можно существенно ускорить алгоритм.