Московский Государственный Университет
имени М. В. Ломоносова

Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 1
Подвариант №1

«РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ МЕТОДОМ ГАУССА И МЕТОДОМ ГАУССА С ВЫБОРОМ ГЛАВНОГО ЭЛЕМЕНТА»

ОТЧЕТ
о выполненном задании

студента 206 учебной группы факультета ВМК МГУ
Оганисяна Эдгара Гагиковича

Москва, 2019 г.

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 $n\times n$ с невырожденной матрицей A. Написать программу, решающую систему линейных алгебраических уравнений заданного размера 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 A^{-1};
4) Определить число обусловленности $M_A=\|A\|\times\|A^{-1}\|$;
5) Исследовать вопрос вычислительной устойчивости метода Гаусса
6) Правильность решения подтвердить системой тестов.

Описание метода решения

- **Алгоритм Гаусса**

Алгоритм основывается на использовании элементарных преобразований матрицы. С помощью них начальная матрица приводится к единичной матрице, и тогда решение становится очевидным — оно единственно и задается правым столбцом значений f.

Рассмотрим подробнее действие алгоритма (дана матрица размером N):

1) Строка делится на ведущий элемент (то есть на первый элемент строки); Это действие проделывается со всеми строками матрицы; Таким образом в конце получаем, что левый столбец значений единичный.

2) Теперь будем вычитать первую строку из всех последующих. Таким образом, все элементы правого столбца, кроме первого обнулятся.

3) Условно выбросив 1 строку и столбец изначальной матрицы, получим матрицу размерности N-1. Будем проделывать шаги п.1 и п.2 пока не получим верхнетреугольную матрицу.

4) п.1- п.3 — прямой проход алгоритма. Теперь выполним обратный проход, то есть занулим и верхнюю часть матрицы и приведем ее к единичной. Из каждой i строки, будет вычитаться нижняя (вида 0 0 … 1), умноженная на крайний правый элемент i строки. Соответственно весь столбец занулится. Теперь аналогично п.3 выбросим последнюю строку и столбец и получем матрицу на размерность меньше. Будем выполнить п.4 пока не получим единичную матрицу.

(! Все действия, которые проводятся с матрицей, проводятся так же и над правой частью значений f)

После получения единичной матрицы, преобразованный вектор f является решение начального матричного уравнения $Ax = f$.

- **Выбор главного элемента**

Перед выполнением п.1 алгоритма Гаусса находится в столбце i строка с наибольшим по модулю значением первого элемента. Меняем местами i и 1 строки и продолжаем выполнение алгоритма. Выбор главного элемента происходит при каждом «понижении размерности матрицы»

- **Асимптотика**

На деление строки и вычитание их друг из друга требуется в худшем случае $n^2 + n$ операций (для матрицы размерности n). Таких оперций нам необходимо выполнить n штук (с каждым шагом размерность «уменьшается» на 1). Таким образом сложность алгоритма $\sim O(N^3)$.

- **Описание программы**

Основной частью программы является функция;

`void gauss_mod(double **a, double *x, double *f, int n)`

Она принимает в качестве параметров матрицу коэффициентов *a*, векто правой части *f* и размерность матрицы *n.* В вектор *x* будет записываться ответ. Также в программе присутствует глобальная переменная *flags*, выставленные биты которой определяют, какие фунцкии необходимо выполнить (найти определитель, обратную матрицу и т.д). Сделано это для оптимизации использования ресурсов и параллельного выполнения нескольких задач, т.к за полный проход алгоритма Гаусса можно не только найти корни уравнения $Ax = f$, но и вычислить определитель, найти обратную матрицу, число обусловленности.

- **Листинг программы**

Т.к программа достаточна велика, здесь приведем пояснения ко всем функциям. Текст программы будет доступен в приложении.

`void check_print(double **a, double *x, int n)` — проверочная ф-ция, использующаяся, как для отладки, так и для вывода полученных значений

`void init_matr(FILE *f, double **a, double *x, int n, int mode)` ф-ция создания и заполнения матрицы и векторов путем считывания из файла или с помощью задания формулой (в зависимости от параметра mode).

`void gauss_mod(double **a, double *x, double *f, int n)`— основная ф-ция, которая выполняет элементарные преобразования и приводит матрицу а к единичной матрице, параллельно вычисляя все необходимые значения.

`double *mul_matr(double **a, double *x, int n)`— ф-ция умножения матрицы на вектор (используется для подсчета невязки).

`double **copy_1(double **a, double **, int n)`— создание копии матрицы коэффициентов *a.*

`double *copy_2(double *f, double *, int n)`— создание копии значений правой части *f.*

`double **create_rev(double **rev, int n)` — создание массива для вычисления обратной матрицы

`double norm(double **a, int n)`—подсчет евклидовой нормы матрицы.

`int max_replace(double **a, double *b, int n, int i, double **rev)` — ф-ция замена ведущей и *i* строки.

Тесты

Вариант 9

тест №1
$$\begin{cases} 2x_1 - 5x_2 + 3x_3 + \ \ x_4 = 5 \\ 3x_1 - 7x_2 + 3x_3 - \ \ x_4 = -1 \\ 5x_1 - 9x_2 + 6x_3 + 2x_4 = 7 \\ 4x_1 - 6x_2 + 3x_3 + \ \ x_4 = 8 \end{cases}$$

Результаты работы программы:

```
x: 0.000 -3.000 -5.333 6.000
Determinant = 18.000
Discrepancy = 0.000
Condition number: 20.498
Revers:
-1.000 -0.000 0.333 0.333
-1.000 -0.000 0.667 -0.333
-1.000 0.167 1.056 -0.944
1.000 -0.500 -0.500 0.500
```

Ответы wolframalpha.com:

x: $\{1.42109\times10^{-15}, -3., -5.33333, 6.\}$

Determinant: 18

Inverse:

Result:

$$\begin{pmatrix} -1. & -2.22045\times10^{-16} & 0.333333 & 0.333333 \\ -1. & -1.11022\times10^{-16} & 0.666667 & -0.333333 \\ -1. & 0.166667 & 1.05556 & -0.944444 \\ 1. & -0.5 & -0.5 & 0.5 \end{pmatrix}$$

тест №2
$$\begin{cases} 4x_1 + \ \ 3x_2 - \ \ 9x_3 + \ \ \ \ x_4 = 9 \\ 2x_1 + \ \ 5x_2 - \ \ 8x_3 - \ \ \ \ x_4 = 8 \\ 2x_1 + 16x_2 - 14x_3 + \ \ 2x_4 = 24 \\ 2x_1 + \ \ 3x_2 - \ \ 5x_3 - 11x_4 = 7 \end{cases}$$

Результаты работы программы:

```
x: 3.000 2.000 1.000 -0.000
Determinant = -868.000
Discrepancy = 0.000
Condition number: 30.503
Revers:
0.689 -1.364 0.253 0.233
0.118 -0.507 0.182 0.090
0.240 -0.779 0.175 0.124
0.048 -0.032 0.016 -0.081
```

Ответы wolframalpha.com:

x: $\{3, 2, 1, -8.95341\times10^{-18}\}$

Determinant: -868

Inverse:

Result:

$$\begin{pmatrix} 0.68894 & -1.36406 & 0.253456 & 0.232719 \\ 0.117512 & -0.506912 & 0.182028 & 0.0898618 \\ 0.239631 & -0.778802 & 0.175115 & 0.124424 \\ 0.0483871 & -0.0322581 & 0.016129 & -0.0806452 \end{pmatrix}$$

тест №3
$$\begin{cases} 12x_1 + 14x_2 - 15x_3 + 24x_4 = 5 \\ 16x_1 + 18x_2 - 22x_3 + 29x_4 = 8 \\ 18x_1 + 12x_2 - 21x_3 + 32x_4 = 9 \\ 10x_1 + 20x_2 - 16x_3 + 20x_4 = 4 \end{cases}$$

Результаты работы программы:

```
x: 2.222 -1.667 -0.111 0.000
Determinant = 36.000
Discrepancy = 0.000
Condition number: 87.354
Revers:
1.000 2.889 -1.667 -2.722
-2.500 -4.667 3.500 4.167
0.000 -0.444 0.333 0.111
1.000 1.000 -1.000 -1.000
```

Ответы wolframalpha.com:

x: $\{2.222, -1.667, -0.111, -3.88578\times10^{-16}\}$

Deteminant: 36

Inverse:

Result:

$$\begin{pmatrix} 1. & 2.88889 & -1.66667 & -2.72222 \\ -2.5 & -4.66667 & 3.5 & 4.16667 \\ 1.77636\times10^{-15} & -0.444444 & 0.333333 & 0.111111 \\ 1. & 1. & -1. & -1. \end{pmatrix}$$

тест №4

Элементы матрицы A вычисляются по формулам:

$$A_{ij} = \begin{cases} q_M^{(i+j)} + 0.1 \cdot (j-i), & i \neq j, \\ (q_M - 1)^{(i+j)}, & i = j, \end{cases}$$

$где\, q_M = 1.001 - 2 \cdot M \cdot 10^{-3}, \quad i, j = 1, \ldots n.$

Элементы вектора f задаются формулами:

$$b_i = \left| x - \frac{n}{10} \right| \cdot i \cdot \sin(x), \quad i = 1, \ldots n$$

M = 2; N = 40

Результаты работы программы:

```
x: {-33.076253 0.013094 0.026123 0.039085 0.051980 0.064805
0.077557 0.090235 0.102837 0.115361 0.127804 0.140165 0.152441
0.164630 0.176731 0.188740 0.200656 0.212476 0.224199 0.235821
0.247341 0.258755 0.270063 0.281260 0.292346 0.303316 0.314170
0.324904 0.335515 0.346002 0.356361 0.366590 0.376687 0.386648
0.396470 0.406152 0.415690 0.425081 0.434323 0.443413}

Determinant = 2.254305
Discrepancy = 0.000000 (означает, что найденное решение верное)
Condition number: 87.138393
```

Выводы

Был изучен метод Гаусса, метод Гаусса с выбором главного элемента для решения СЛАУ и написана программа, реализующая эти методы. Результаты работы совпали с реальными решениями, полученными в WolframAlpha. Помимо нахождения решения СЛАУ, программа дополнительно вычисляет определитель матрицы, её обратную матрицу и число обусловленности, если матрица невырожденная.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

enum
{
    FILES = 0,
    FORMULES = 1,
    WP = 1,
    DET = 1 << 1,
    DISCRP = 1 << 2,
    REV = 1 << 3,
    COND = 1 << 4,
};

int flags =  COND | DET | DISCRP | REV;

// данные для примера 2
const int M = 2, N = 40; // пункт 2.2 приложение 2
const double pi = 3.1415926535;

void check_print(double **a, double *x, int n);
void init_matr(FILE *f, double **a, double *x, int n, int mode);
void gauss_mod(double **a, double *x, double *f, int n);
double *mul_matr(double **a, double *x, int n);
double **copy_1(double **a, double **,  int n);
double *copy_2(double *f, double *, int n);
double **create_rev(double **rev, int n);
double norm(double **a, int n);
int max_replace(double **a, double *b, int n, int i, double **rev);

int main(int argc, char **argv)
{
    FILE *fin = fopen(argv[1], "r");
    int n;
    fscanf(fin, "%d", &n);

    int mode;
    sscanf(argv[2], "%d", &mode);

    if (mode == 1)
        n = N;

    double **A, *x, *f;
    x = calloc(n, sizeof(double));
    f = calloc(n, sizeof(double));
    A = calloc(n, sizeof(double *));



    init_matr(fin, A, f, n, mode);
//check_print(A, f, n);
    gauss_mod(A, x, f, n) ;
//check_print(A, x, n);


    return 0;
}

void gauss_mod(double **a, double *x, double *f, int n)
{
    int i, j, k , m, replace_cnt = 0;
    double first, determinant = 1. , cond_num = 0. , **a_cpy, **rev, *f_cpy;

    if (flags & DET)
        printf("DET - ON\n");

    if (flags & WP)
        printf("WP - ON\n");
```

```
    if (flags & DISCRP) {
        printf("DISCRP - ON\n");
        a_cpy = copy_1(a_cpy, a, n);
        f_cpy = copy_2(f_cpy, f, n);
    }

    if (flags & COND) {
        printf("COND - ON\n");
        cond_num += norm(a, n);
    }

    if ((flags & REV) || (flags & COND)) {
        if (flags & REV)
            printf("REV - ON\n");
        rev = create_rev(rev, n);
    }

    printf("\n");

    for (i = 0; i < n; i++) {
        if (flags & WP) // fkag WP - it's gauss with a principal element
            replace_cnt += max_replace(a, f, n, i, rev);

        for (j = i; j < n; j++) { // делим строку на первый строки (приводим к единице первый эл)
            if (a[j][i] == 0) first = 1;
            else first = a[j][i];

            for (k = 0; k < n; k++) // делим на старший элемент
                a[j][i+k] /= first;

            if ((flags & REV) || (flags & COND))
                for (int v = 0; v < n; v++)
                    rev[j][v] /= first;

            f[j] /= first;

            if (flags & DET) // паралелллльное вычисление определителя)
                determinant *= first;
        }

        for (m = i+1; m < n; m++){ // вычитаем строки друг из друга
            if (a[m][i] == 0) continue;

            for (k = 0; k < n-i; k++)
                a[m][k+i] -= a[i][i+k];

            if ((flags & REV) || (flags & COND)) // работаетм с флагом для -1 матрицы
                for(int v = 0; v < n; v++)
                    rev[m][v] -= rev[i][v];

            f[m] -= f[i]; // и столбец значений тоже вычитаем (для гауса)
        }
    }

    for (int g = n-1; g >= 0; g--) { // обратный ход метода гауса
        for(int h = g - 1; h >= 0; h--) {
            f[h] -= f[g]*a[h][g];

            if((flags & REV) || (flags & COND))
                for(int v = 0; v < n; v++)
                    rev[h][v] -= rev[g][v] * a[h][g];

            a[h][g] -= a[g][g] * a[h][g]; // как будто бы вычли
        }

```

```c
        x[g] = f[g];
    }


    printf("x: ");
    for(int h = 0; h < n; h++)
        printf("%lf ", x[h]);
    printf("\n");

    determinant *= pow(-1., replace_cnt); // если была перестановка строк
    if (flags & DET)
        printf("Determinant = %lf\n", determinant);

    if (flags & DISCRP) { // считаем невязку
        double *res = mul_matr(a_cpy, x, n);
        double dis = 0.;
        for (int h = 0; h < n; h++)
            dis += (f_cpy[h]-res[h]) * (f_cpy[h]-res[h]);
        dis = sqrt(dis);
        printf("Discrepancy = %lf\n", dis);
    }

    if (flags & COND) {
        cond_num += norm(rev, n);
        printf("Condition number: %lf\n", cond_num);
    }
    if (flags & REV) {
        printf("Revers:\n");
        for(int g = 0; g < n; g++){
            for(int h = 0; h< n; h++){
                printf("%lf ", rev[g][h]);
            }
            printf("\n");
        }
    }

    return ;
}

double *mul_matr(double **a, double *x, int n)
{
    double *res = calloc(n, sizeof(double));
    for (int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            res[i] += a[i][j] * x[j];
        }
    }
    return res;
}

int max_replace(double **a, double *b, int N, int i, double **rev)
{
    int k, j;
    double max = fabs(a[i][i]);
    int tmp = 0;

    for (k = i+1; k < N; k++){
        if(fabs(a[k][i]) > max){
            max = fabs(a[k][i]);
            tmp = k;
        }
    }

    if (tmp == 0) return 0;

    double t;
    for(j = 0; j < N; j++){
        t = a[i][j];
        a[i][j] = a[tmp][j];
```

```c
        a[tmp][j] = t;

        if (flags & REV) {
            t = rev[i][j];
            rev[i][j] = rev[tmp][j];
            rev[tmp][j] = t;
        }
    }

    t = b[i];
    b[i] = b[tmp];
    b[tmp] = t;

    return 1;
}

double **copy_1(double **res, double **a, int n)
{
    res = calloc(n, sizeof(double *));
    for (int h = 0; h < n; h++)
        res[h] = calloc(n, sizeof(double));

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            res[i][j] = a[i][j];

    return res;
}

double *copy_2(double *res, double *f, int n)
{
    res = calloc(n, sizeof(double));
    for (int i = 0; i < n; i++)
        res[i] = f[i];
    return res;
}

double **create_rev(double **rev, int n)
{
    rev = calloc(n, sizeof(double));
    for (int i = 0; i < n; i++)
        rev[i] = calloc(n, sizeof(double));
    for (int i =0; i < n; i++)
        rev[i][i] = 1. ;

    return rev;
}

double norm(double **a, int n)
{
    double res = 0. ;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            res += a[i][j] * a[i][j];
    res = sqrt(res);
    return res;
}


void init_matr(FILE *fin, double **A, double *f, int n, int mode)
{
    double q = 1.001 - 2 * M * 0.001;

    for (int i = 0; i < n; i++)
        A[i] = calloc(n, sizeof(double));

    switch (mode) {
        case FILES:
```

```
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    fscanf(fin, "%lf", &A[i][j]);
                }
            }


            for (int i = 0; i < n; i++)
                fscanf(fin, "%lf", &f[i]);

            break;

        case FORMULES:
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (i == j) {
                        A[i][j] = pow(q-1, (double)(i+j));
                    } else {
                        A[i][j] = pow(q, (double)(i+j)) + 0.1 * (j - i);
                    }
                }
            }

            double x = pi/2; // случайный параметр для генерация вектора f
            for (int i = 0; i < n; i++)
                f[i] = fabs(x - N/10.) * i * sin(x);

            break;

        default:
            return;
    }

    return;
}


void check_print(double **A, double *f, int n)
{
    printf("A:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%lf ", A[i][j]);
        }
        printf("\n");
    }
    printf("\nf: ");
    for (int i = 0; i < n; i++)
        printf("%lf ", f[i]);
    printf("\n");
}
```