Московский Государственный Университет
имени М. В. Ломоносова

Компьютерный практикум по учебному курсу
«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 1
Подвариант №2

«ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ НА ПРИМЕРЕ МЕТОДОВ ЗЕЙДЕЛЯ И ВЕРХНЕЙ РЕЛАКСАЦИИ»

ОТЧЕТ
о выполненном задании

студента 206 учебной группы факультета ВМК МГУ
Оганисяна Эдгара Гагиковича

Москва, 2019 г.

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 $n \times n$ с невырожденной матрицей A. Написать программу, численного решения СЛАУ (n - параметр программы), использующий численный алгоритм итерационного метода верхней релаксации:

$$(\mathrm{D} + \omega A^{(-)})\frac{x^{k+1} - x^k}{\omega} + Ax^k = f$$

где w - итерационный параметр.

Цели и задачи практической работы

1) Решить заданную СЛАУ итерационным методом верхней релаксации;
2) Разработать критерий остановки итерационного процесса, гарантирующий получение приближение решения исходной СЛАУ с заданной точностью;
3) Изучить скорость сходимости итераций к точному решению задачи; Рассмотреть результаты при различных параметрах w.
4) Правильность решения подтвердить системой тестов.

Описание Метода решения

- **Итерационный процесс**

  Рекуррентная формула для итерационного подсчета выглядит следующим образом:

  $$(*) \quad (\mathrm{D} + \omega A^{(-)})\frac{x^{k+1} - x^k}{\omega} + Ax^k = f \quad , \text{ где}$$

  D — диагональная матрица;
  $A^{(-)}$ — нижнетреугольная матрица;
  k — номер текущей итерации;
  w — итерационный параметр

  Если преобразовать векторную формулу (*) и перейти к непосредственной записи уравнений, то можно составить формулу для вычисления $k+1$ элемента:

  $$x_i^{(k+1)} = x_i^k + \frac{w}{A_{ii}}\left(f_i - \sum_{j=1}^{i-1} A_{ij} x_{ij}^{(k+1)} - \sum_{j=1}^{n} A_{ij} x_j^k\right)$$

- **Условие сходимости**

  Если A — симметрическая положительно определенная матрица, и при этом выполняется выполняется неравенство $0 < w < 2$, то метод сходится. Если начальная матрица A, не удовлетворяет заданным условиям, то мы просто умножаем ее и правую часть f на A^T. Решение системы от этого не изменится, однако матрица коэффициентов A^TA будет самосопряженной и положительно определенной.

- **Критерий остановки**

  Для оценки точности и корректности решения будем использовать вектор невязки. То есть, если мы хотим найти решение с точностью ε, то наше решение x, должно удовлетворять неравенству:

  $$|| Ax - f || < \varepsilon$$

- ## Описание программы

Основной частью программы является функция:
`int relax(double **a, double *x, double *f, int n, double eps, double w)`,
которая в качестве параметров принимает матрицу коэффициентов *a*, ветор *x* для записи ответа, столбец правых значений *f*, размерность *n* и параметры *eps* и *w*, отвечающие за точность и шаг решения. Данная функция реализует метод верхней релаксации. Остальные функции вспомогательные.

- ## 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 mul_AX(double **a, double *x, int n)` — ф-ция умножения матрицы на вектор;

`void mul_AA(double **a, double **b, int n)` — ф-ция перемножения двух матриц;

`void cpy1(double *a, double *b, int n)` — копирование значений вектора *b*;

`void cpy2(double **a, double **b, int n)` — копирование значений матрицы *b*;

`void transp(double **At, double **a, int n)` — ф-ция транспонирования матрицы *a*;

`double norm(double *x, int n)` — подсчет евклидовой нормы вектора *x*;

Тесты

Вариант 9

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

Результаты работы программы при $\varepsilon = 10^{-6}$:

```
x = { 0.000002 , -2.999998,  -5.333330,  5.999998}
```

Ответ wolframalpha.com:

x = {1.42109×10^{-15}, -3., -5.33333, 6.}

```
w = 0.1: iterations = 69396
w = 0.2: iterations = 32912
w = 0.3: iterations = 20747
w = 0.4: iterations = 14661
w = 0.5: iterations = 11007
w = 0.6: iterations = 8568
w = 0.7: iterations = 6822
w = 0.8: iterations = 5509
w = 0.9: iterations = 4483
w = 1.0: iterations = 3657
w = 1.1: iterations = 2974
w = 1.2: iterations = 2395
w = 1.3: iterations = 1890
w = 1.4: iterations = 1431
w = 1.5: iterations = 959
w = 1.6: iterations = 792
w = 1.7: iterations = 1013
w = 1.8: iterations = 1509
w = 1.9: iterations = 2993
```

Наилучшая сходимость наблюдается при *w* = 1.6

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 при $\varepsilon = 10^{-6}$:

```
x = {3.000000,  2.000000,  1.000000,  0.000000}
```

Ответы wolframalpha.com:

x = {3, 2, 1, -8.95341×10^{-18}}

```
w = 0.1: iterations = 56466
w = 0.2: iterations = 26651
w = 0.3: iterations = 16719
w = 0.4: iterations = 11758
w = 0.5: iterations = 8786
w = 0.6: iterations = 6807
w = 0.7: iterations = 5395
w = 0.8: iterations = 4334
w = 0.9: iterations = 3501
w = 1.0: iterations = 2820
w = 1.1: iterations = 2218
w = 1.2: iterations = 1570
w = 1.3: iterations = 1550
w = 1.4: iterations = 1317
w = 1.5: iterations = 1072
w = 1.6: iterations = 831
w = 1.7: iterations = 591
w = 1.8: iterations = 316
w = 1.9: iterations = 369
```

Наилучшая сходимость наблюдается при *w* = 1.8

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 при $\varepsilon = 10^{-6}$:

```
x = {2.222225,  -1.666672,  -0.111111,  0.000001}
```

Ответы wolframalpha.com:

x = {2.222, -1.667, -0.111, -3.88578×10^{-16}}

```
w = 0.1: iterations = 8588350
w = 0.2: iterations = 4057082
w = 0.3: iterations = 2547585
w = 0.4: iterations = 1793681
w = 0.5: iterations = 1342140
w = 0.6: iterations = 1041869
w = 0.7: iterations = 828078
w = 0.8: iterations = 668303
w = 0.9: iterations = 544409
w = 1.0: iterations = 445395
w = 1.1: iterations = 364111
w = 1.2: iterations = 295598
w = 1.3: iterations = 236041
w = 1.4: iterations = 181367
w = 1.5: iterations = 104780
w = 1.6: iterations = 111994
w = 1.7: iterations = 85618
w = 1.8: iterations = 48122
w = 1.9: iterations = 61065
```

Наилучшая сходимость наблюдается при *w* = 1.8

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

Элементы вектора f задаются формулами:

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

M = 2; N = 40

Результаты работы программы при $\varepsilon = 10^{-3}$:

```
x = {-33.056844 0.011439 0.024493 0.037485 0.050413
0.063274 0.076066 0.088787 0.101433 0.114003 0.126494
0.138905 0.151231 0.163472 0.175625 0.187688 0.199659
0.211536 0.223316 0.234998 0.246580 0.258061 0.269438
0.280711 0.291877 0.302937 0.313889 0.324732 0.335465
0.346087 0.356595 0.366988 0.377257 0.387395 0.397385
0.407209 0.416842 0.426257 0.435429 0.444334}
```

Решение совпадает с ответом, полученным методом Гаусса.

```
w = 0.1: iterations = 616861
w = 0.2: iterations = 295427
w = 0.3: iterations = 189590
w = 0.4: iterations = 136157
w = 0.5: iterations = 103050
w = 0.6: iterations = 79774
w = 0.7: iterations = 61677
w = 0.8: iterations = 45477
w = 0.9: iterations = 25194
w = 1.0: iterations = 33391
w = 1.1: iterations = 29510
w = 1.2: iterations = 22205
w = 1.3: iterations = 25164
w = 1.4: iterations = 31893
w = 1.5: iterations = 44528
w = 1.6: iterations = 58119
w = 1.7: iterations = 84737
w = 1.8: iterations = 131967
w = 1.9: iterations = 279888
```

Наилучшая сходимость наблюдается при *w* = 1.2

Выводы

Был изучен метод верхней релаксации для решения СЛАУ и написана программа, реализующая этот итерационный метод. Как видно из исследования, скорость сходимости метода для каждой системы уравнений сильно зависит от выбора итерационного параметра, поэтому важной задачей является подбор наилучшего. Если решения не существует, то метод релаксации работает некорректно. То есть, перед его использованием следует исследовать систему на совместность.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include <sys/wait.h>

enum
{
    FILES = 0,
    FORMULES = 1,
};


// 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 mul_AX(double **a, double *x, int n);
void mul_AA(double **a, double **b, int n);
void cpy1(double *a, double *b, int n);
void cpy2(double **a, double **b, int n);
void transp(double **At, double **a, int n);
double norm(double *x, int n);
int relax(double **a, double *x, double *f, int n, double eps, double w);



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);

    double eps, w;
    sscanf(argv[3], "%lf", &eps);
    sscanf(argv[4], "%lf", &w);

    int steps = relax(A, x, f, n, eps, w);
    for(int i = 0; i < n; i++)
        printf("%.6lf ", x[i]);
    printf("  w = %.1lf: iterations = %d\n", w, steps);

    return 0;
}

int relax(double **a, double *x, double *f, int n, double eps, double w)
{
    int cnt = 0;
    double **At = calloc(n, sizeof(double *));
    for (int i = 0; i < n; i++)
        At[i] = calloc(n, sizeof(double));

    double *x_prev = calloc(n, sizeof(double));
    double *ft = calloc(n, sizeof(double));
```

```

    transp(At, a, n);
    mul_AA(At, a, n); // A = At * A
    mul_AX(At, f, n); // b = At * b

    do {
        cnt++;
        for(int i = 0; i < n; i++) {
            x_prev[i] = x[i];
        }
        for(int i = 0; i < n; i++) {
            double sum = 0;
            for (int j = 0; j < i; j++) {
               sum += a[i][j] * x[j];
            }

            for (int j = i; j < n; j++) {
               sum += a[i][j] * x_prev[j];
            }

            x[i] = x_prev[i] + w * (f[i] - sum) / a[i][i];
        }

        for(int i = 0; i < n; i++) {
            x_prev[i] = x[i];
        }
        mul_AX(a, x_prev, n); // x_prev = A * x
        for(int i = 0; i < n; i++) {
            x_prev[i] -= f[i];
        }
    } while (norm(x_prev, n) > eps);
    free(At);
    free(ft);
    free(x_prev);
    return cnt;
}

void transp(double **At, double **a, int n)
{
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            At[i][j] = a[j][i];

}

void cpy1(double *a, double *b, int n)
{
    for(int i = 0; i < n; i++)
        a[i] = b[i];
}

void cpy2(double **a, double **b, int n)
{
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            a[i][j] = b[i][j];
}


void mul_AX(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];
        }
    }
    cpy1(x, res, n);
    free(res);
```

```
}

void mul_AA(double **a, double **b, int n)
{
    double **res = calloc(n, sizeof(double *));
    for (int i = 0; i < n; i++)
        res[i] = calloc(n, sizeof(double));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                res[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    cpy2(b, res, n);
    free(res);
}

double norm(double *x, int n)
{
    double res = 0. ;
    for (int i = 0; i < n; i++)
        res += x[i] * x[i];
    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;
```

```c

        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");
}

```