**Московский государственный университет**

**имени М.В. Ломоносова**

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

**Компьютерный практикум по учебному курсу**

# «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

**ЗАДАНИЕ №1**

# ОТЧЕТ

**о выполненном задании**

студента 210 учебной группы факультета ВМК МГУ

Зверева Дмитрия Андреевича

**МОСКВА — 2023**

# Оглавление

# Глава 1

# Задача №1

## 1.1 Постановка задачи

Решить с помощью метода прогонки СЛАУ, получающуюся при построении кубического сплайна для функции $f(x) = x|cos(5x)|$ на отрезке $x \in [0, 2]$ по значениям в $N = 100$ узлах $x_i = \frac{2i}{N}$ (данную СЛАУ необходимо выписать).

## 1.2 Цели и задачи практической работы

1. Построить СЛАУ, получающуюся при построении кубического сплайна для функции $f(x) = x|cos(5x)|$ на отрезке $x \in [0, 2]$ по значениям в $N = 100$ узлах $x_i = 2i/N$;
2. Решить с помощью метода прогонки получившуюся СЛАУ;
3. Подтвердить правильность решения СЛАУ системой тестов;
4. Построить на одном графике исходную функцию и получившийся интерполянт.

## 1.3 Описание метода решения

### 1.3.1 Построение СЛАУ

Пусть на отрезке $x \in [0, 2]$ задана функция $f(x) = x|cos(5x)|$. Рассмотрим сетку узлов

$$a = x_0 < x_1 < ... < x_n = b$$

и обозначим через $h$ расстояние между смежными узлами. Оно равно

$$h = \frac{2}{N}$$

Задача сводится к отысканию коэффициентов полиномов третьей степени на каждом из отрезков $x \in [x_{i-1}, x_i]$. Сопоставим отрезку $x \in [x_{i-1}, x_i]$ следующий полином:

$$S_i(x) = a_i + b_i(x - x_i) + \frac{c_i}{2}(x - x_i)^2 + \frac{d_i}{6}(x - x_i^3), \quad x \in [x_{i-1}, x_i], \quad i = 1, ..., n$$

Требуя выполнения условий для кубического сплайна и выполняя несложные преобразования, получаем:

$$a_i = f(x_i) = f_i,$$

$$c_0 = c_n = 0,$$

$$c_{i-1} + 4c_i + c_{i+1} = 6\frac{f_{i-1} - 2f_i + f_{i+1}}{h^2}, \quad i = 1, ..., n,$$

$$d_i = \frac{c_i - c_{i-1}}{h}, \quad i = 1, ..., n,$$

$$b_i = \frac{1}{2}hc_i - \frac{1}{6}h^2 d_i + \frac{f_i - f_{i-1}}{h}, \quad i = 1, ..., n$$

Заметим, что полученное уравнение для коэффициента $c$ можно решить методом прогонки, представив его в виде СЛАУ вида $Ax = F$, где вектор $x$ соответствует вектору $c_i$, вектор $F$ поэлементно равен правой части уравнения, а матрица А имеет следующий вид:

$$A = \begin{bmatrix} 4 & 1 & 0 & 0 & ... & 0 \\ 1 & 4 & 1 & 0 & ... & 0 \\ 0 & 1 & 4 & 1 & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & ... & 4 \end{bmatrix}$$

При этом легко видеть, что в нашем случае матрица обладает свойством диагонального преобладания.

### 1.3.2 Решение СЛАУ методом прогонки

Метод прогонки используется для решения СЛАУ (как правило, с трёхдиагональной матрицей). Он состоит из двух этапов: прямой прогонки и обратной. На первом этапе определяются прогоночные коээфициенты, на втором – находятся неизвестные переменные.

Прямая прогонка состоит в вычислении прогоночных коэффициентов $q_i$ и $w_i$ , где $i$ – номер строки матрицы. Этот этап выполняется при $i = 1, ..., n$ строго по возрастанию значения $i$.

В первой строке матрицы ($i = 1$) используются формулы:

$$y_1 = b_1, \quad q_1 = \frac{-c_1}{y_1}, \quad w_1 = \frac{d_1}{y_1}$$

Для строк $i$ от 2 до $n - 1$ используются рекуррентные формулы:

$$y_i = b_i + a_i q_{i-1}, \quad q_i = \frac{-c_i}{y_i}, \quad w_i = \frac{d_i - a_i w_{i-1}}{y_i}$$

При $i = n$ прямая прогонка завершается вычислением

$$y_n = b_n + a_n q_{n-1}, \quad w_n = \frac{d_n - a_n w_{n-1}}{y_n}$$

После этого производится обратная прогонка, в которой происходит вычисление неизвестных переменных. Этот этап обычно выполняется при $i = n, ..., 1$ строго по убыванию значения $i$.

В последней строке матрицы ($i = n$):

$$x_n = w_n$$

Для всех остальных строк при $i$ от $n - 1$ до 1 применяется формула

$$x_i = q_i x_{i+1} + w_i$$

## 1.4 Описание программы

Программа написана на языке программирования C. Программа написана единой функцией, поделенной на логические блоки:

- построение СЛАУ.
- решение СЛАУ.
- дополнительный блок - поиск оставшихся коэффициентов для построения кубического сплайна функции.

Далее приведён код программы.

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

#define N 100

int main(void) {
    /* ПОСТРОЕНИЕ СЛАУ */
    long double x[N+1];
    long double f[N+1];
    long double F[N];
    // поиск значений в точках функции
    for (int i = 0; i <= N; i++) {
        x[i] = 2 * i / (long double)N;
        f[i] = x[i] * fabs(cosl(5*x[i]));
        //printf("%Lf %Lf\n", x[i], f[i]);
    }
    // значение правой части уравнения
    for (int i = 1; i < N; i++) {
        F[i-1] = 6 * (f[i-1] - 2*f[i] + f[i+1]) * ((N/(long double)2)*(N/(long double)2));
        //printf("%Lf\n", F[i-1]);
    }


    // использован метод прогонки
    long double a[N];
    long double b[N];
    long double c[N];
    a[0] = 0;
    b[N-2] = 0;
    for (int i = 0; i < (N-2); i++) {
        a[i+1] = 1;
        c[i] = 4;
        b[i] = 1;
    }
    c[N-2] = 4;

    long double g[N];
    long double m;
    for (int i = 1; i < (N-1); i++) {
        m = a[i]/c[i-1];
        c[i] = c[i] - m*b[i-1];
        F[i] = F[i] - m*F[i-1];
    }
    g[N-2] = F[N-2]/c[N-2];
    for (int i = N-3; i >= 0; i--)
        g[i] = (F[i]-b[i]*g[i+1])/c[i];
    for (int i = 0; i < (N-1); i++)
        printf("c[%d] = %Lf\n", i+1, g[i]);

    /* ПОИСК ОСТАЛЬНЫХ КОЭФФИЦИЕНТОВ (это уже нетрудная задача и здесь не требовалась) */
    printf("---------\n");
    for (int i = 0; i < (N-1); i++) {
        printf("a[%d] = %Lf\n", i+1, f[i+1]);
    }
    printf("---------\n");
    long double buf;
    for (int i = 1; i < N; i++) {
        if (i > 1)
            buf = ((long double)1/2)*(2/((long double)N))*g[i-1] -
            ((long double)1/6)*(2/((long double)N))*(g[i-1] - g[i-2]) +
            (f[i]-f[i-1])*((long double)N/2);
        else
            buf = ((long double)1/2)*(2/((long double)N))*g[i-1] -
            ((long double)1/6)*(2/((long double)N))*(g[i-1] - 0) +
            (f[i]-f[i-1])*((long double)N/2);
        printf("b[%d] = %Lf\n", i, buf);
    }
    printf("---------\n");
    for (int i = 0; i < (N-1); i++) {
        if (i > 0)
            printf("d[%d] = %Lf\n", i+1, (g[i]-g[i-1]) * ((long double)N/2));
        else
            printf("d[%d] = %Lf\n", i+1, g[i]*((long double)N/2));
    }
    return 0;
}
```

## 1.5 Тесты

### 1.5.1 Тест №1

Пусть $N = 5$. Получим с помощью нашей программы все необходимые коэффициенты:

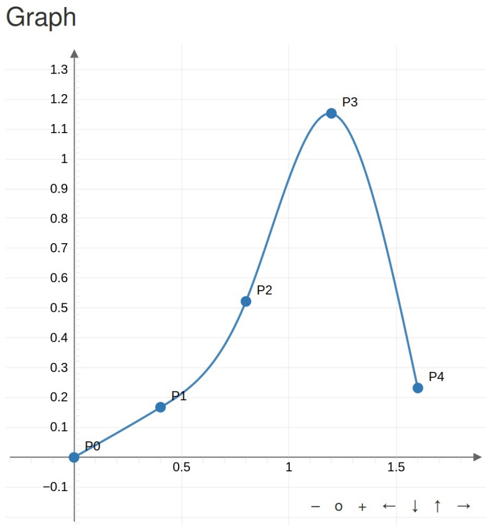
$c[1] = -0.361034$
$c[2] = 8.569042$
$c[3] = -23.683883$
$c[4] = 28.090477$
———
$a[1] = 0.166459$
$a[2] = 0.522915$
$a[3] = 1.152204$
$a[4] = 0.232800$
———
$b[1] = 0.368009$
$b[2] = 2.009610$
$b[3] = -1.013358$
$b[4] = -0.132039$
———
$d[1] = -0.902586$
$d[2] = 22.325190$
$d[3] = -80.632312$
$d[4] = 129.435900$

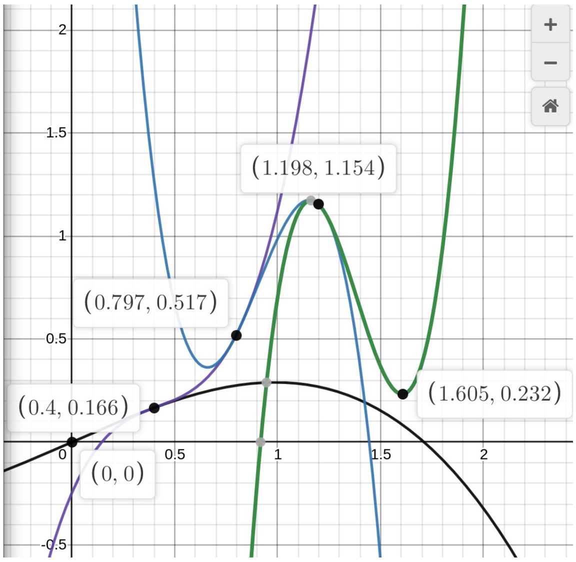
Решая описанную выше СЛАУ при помощи сервиса WolframAlpha получаем следующий результат (для вектора $x = c_i$):

$$x = \begin{bmatrix} -0.34809 & 8.55484 & -23.67129 & 28.080323 \end{bmatrix}^T$$

Данный результат (с учётом погрешности) совпадает с тем, что выдает нам программа.

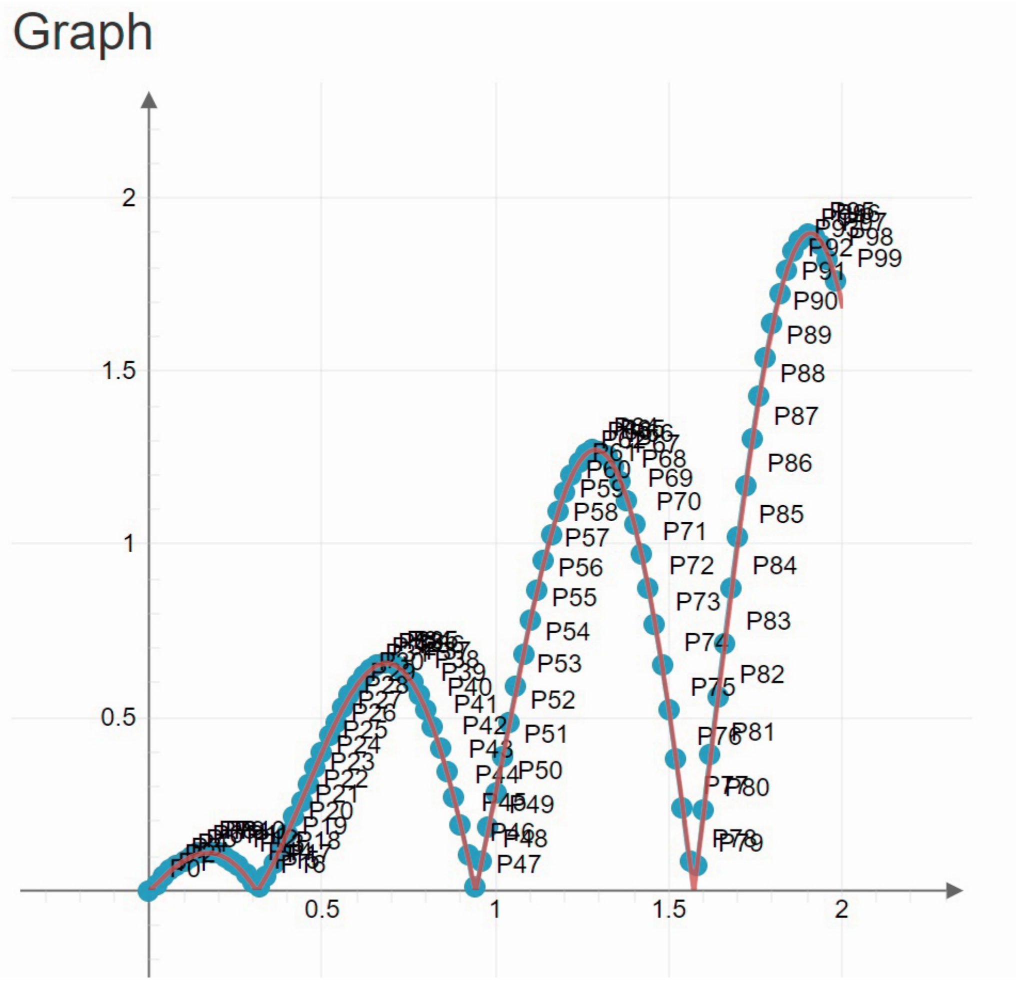
Проверим теперь, какие графики получатся при построении кубического сплайна при $N = 5$. На изображении ниже слева расположен результат, полученный при помощи сервиса Online Tools, справа — результат ручного ввода функций 3-й степени с соответствующими коэффициентами на основании того, что выдает нам программа, в сервисе Desmos. Как можно видеть, если отбросить те части графиков функций, которые не входят в промежуток $x \in [x_{i-1}, x_i]$, $i = 1, ..., N$ для каждого участка соответственно, то графики будут практически полностью совпадать, что подтверждает корректность нашей программы.




## 1.6 График функции и его интерполянта

Данные графики функций построены при помощи сервисов Desmos и Online Tools. Красной линией показан график исходной функции $f(x) = x|cos(5x)|$, синей линией — график интерполянты. Как можно видеть, графики практически полностью совпадают.



# Глава 2

# Задача №2

## 2.1 Постановка задачи

Решить с помощью метода прогонки следующую СЛАУ вида $Ab = f$:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & ... & 0 & 0 & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & ... & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & 0 & 0 & ... & 6 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 & 0 & 1 \end{bmatrix}$$

где

$$A \in R^{(n+1)*(n+1)}, \quad f_l = cos(x_l) * h^4, \quad x_l = \frac{l\pi}{n}, \quad h = \frac{\pi}{n}, \quad l = 2, ..., n-2,$$
$$f_0 = f_n = 1, \quad f_1 = f_{n-1} = 0,$$
$$n = 100$$

## 2.2 Цели и задачи практической работы

1. Проверить на применимость метода прогонки данную СЛАУ;
2. Подобрать эффективный метод решения для данной задачи;
3. Подтвердить правильность решения системой тестов.

## 2.3 Описание метода решения

### 2.3.1 Приведение основной матрицы СЛАУ к симметричному диагональному виду

Из условия очевидно, что 1-й, 2-й, $(n-1)$-й и $n$-й элементы вектора $b$ равны 1. В этом случае матрицу $A$ можно свести к пятидиагональному виду:

$$A' = \begin{bmatrix} 6 & -4 & 1 & 0 & 0 & ... & 0 & 0 \\ -4 & 6 & -4 & 1 & 0 & ... & 0 & 0 \\ 1 & -4 & 6 & -4 & 1 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & 0 & ... & -4 & 6 \end{bmatrix}$$

где

$$A' \in R^{(n-1)*(n-1)},$$
$$f_i = cos(\frac{i\pi}{100}) * (\frac{\pi}{100})^4, \quad i = 4, ..., 96,$$
$$f_2 = cos(\frac{\pi}{50}) * (\frac{\pi}{100})^4 + 3,$$
$$f_{98} = cos(\frac{49\pi}{50}) * (\frac{\pi}{100})^4 + 3,$$
$$f_3 = cos(\frac{3\pi}{100}) * (\frac{\pi}{100})^4 - 1,$$
$$f_{97} = cos(\frac{97\pi}{50}) * (\frac{\pi}{100})^4 - 1$$

### 2.3.2 Метод прогонки для пятидиагональной матрицы

Как видно из полученной матрицы, для данной задачи классический метод прогонки, используемый в задаче №1 для трёхдиагональной матрицы, неприменим. Поэтому эффективным решением для данной задачи будет использование **метода прогонки для пятидиагональных матриц**.

Итак, задача сводится к нахождению решения системы $K = N - 4$ линейных уравнений с $K$ неизвестными, причем матрица системы имеет пятидиагональный вид:

$$\begin{cases} cx_0 + dx_1 + ex_2 = f_0, \\ bx_0 + cx_1 + dx_2 + ex_3 = f_1, \\ ax_0 + bx_1 + cx_2 + dx_3 + ex_4 = f_2, \\ \qquad ax_1 + bx_2 + cx_3 + dx_4 + ex_5 = f_3, \\ \qquad\qquad ... \\ \qquad ax_{K-5} + bx_{K-4} + cx_{K-3} + dx_{K-2} + ex_{K-1} = f_{K-3}, \\ \qquad\qquad ax_{K-4} + bx_{K-3} + cx_{K-2} + dx_{K-1} = f_{K-2}, \\ \qquad\qquad\qquad ax_{K-3} + bx_{K-2} + cx_{K-1} = f_{K-1} \end{cases}$$

Как и в задаче №1, метод прогонки в данном случае также состоит из двух этапов - прямой прогонки и обратной.

Прямая прогонка состоит в том, что каждое неизвестное $x_i$ выражается через $x_{i+1}$ и $x_{i+2}$ с помощью прогоночных коэффициентов $A_i$, $B_i$ и $C_i$:

$$x_i = A_i x_{i+1} + B_i x_{i+2} + C_i, \quad i = 0, ..., K-3$$

Из первого уравнения системы имеем:

$$x_0 = -\frac{d}{c}x_1 - \frac{e}{c}x_2 + \frac{f_0}{c},$$

отсюда

$$A_0 = -\frac{d}{c}, \quad B_0 = -\frac{e}{c}, \quad C_0 = \frac{f_0}{c}$$

Выражая из второго уравнения системы и заменяя $x_0$ по полученной выше формуле, можем выразить прогоночные коэффициенты для $x_2$:

$$A_1 = -\frac{bB_0 + d}{c + bA_0}, \quad B_1 = -\frac{e}{c + bA_0}, \quad C_1 = \frac{f_1 - bC_0}{c + bA_0}$$

Аналогичным образом можно выразить прогоночные коэффициенты для любого номера $i$:

$$A_i = \frac{-aA_{i-2}B_{i-1} - bB_{i-1} - d}{p_i}, \quad B_i = -\frac{e}{p_i}, \quad C_i = \frac{-a(A_{i-2}C_{i-1} + C_{i-2}) - bC_{i-1} + f_i}{p_i},$$

где

$$p_i = c + a(A_{i-2}A_{i-1} + B_{i-2}) + bA_{i-1}, \quad i = 2, ..., K-3$$

Приступим к обратной прогонке, заключающейся в последовательном вычислении неизвестных $x_i$.

Используя систему уравнений и обозначая $s = aA_{K-4} + b$, после преобразований получаем:

$$A_{K-2} = aA_{K-3} + b,$$

$$B_{K-2} = aB_{K-3} + c,$$

$$C_{K-2} = f_{K-1} - aC_{K-3},$$

$$A_{K-1} = sA_{K-3} + aB_{K-4} + c,$$

$$B_{K-1} = sB_{K-3} + d,$$

$$C_{K-1} = f_{K-2} - aC_{K-4} - sC_{K-3}$$

Найдем отсюда $x_{K-2}$, $x_{K-1}$:

$$x_{K-2} = \frac{C_{K-2} - B_{K-2}x_{K-1}}{A_{K-2}}, \quad x_{K-1} = \frac{C_{K-1}A_{K-2} - A_K C_{K-2}}{B_{K-1}A_{K-2} - B_{K-2}A_{K-1}}$$

Далее, используя рекуррентную формулу для вычисления $x_i$, описанную выше, последовательно вычисляем все оставшиеся неизвестные.

## 2.4 Описание программы

Программа написана на языке программирования C. Программа написана единой функцией, поделенной на логические блоки:

- построение правой части СЛАУ.
- решение СЛАУ.

Далее приведён код программы.

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

#define _USE_MATH_DEFINES            // необходимо для получения значения числа Пи
#define N 12

int main(void) {
    int K = N-4;
    /* ПОСТРОЕНИЕ ПРАВОЙ ЧАСТИ СЛАУ */
    long double F[N-3];
    F[0] = cos(2.0*M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N) + 3;
    F[1] = cos(3.0*M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N) - 1;
    F[N-6] = cos((N-3)*M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N) - 1;
    F[N-5] = cos((N-2)*M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N) + 3;
    for (int i = 2; i <= (N-7); i++) {
        F[i] = cosl(i*M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N)*(M_PI/N);
    }

    /* РЕШЕНИЕ ИЗМЕНЕННОЙ СЛАУ */
    // использован метод прогонки для пятидиагональной матрицы
    long double a, b, c, d, e;
    a = 1;
    b = -4;
    c = 6;
    d = -4;
    e = 1;
    // прямой ход
    long double A[N];
    long double B[N];
    long double C[N];
    A[1] = -d/c;
    B[1] = -e/c;
    C[1] = F[0]/c;
    A[2] = -(b*B[1] + d)/(c + b*A[1]);
    B[2] = -e/(c + b*A[1]);
    C[2] = (F[1] - b*C[1])/(c + b*A[1]);
    long double p[N];
    for (int i = 3; i <= (K-2); i++) {
        p[i] = c + a*(A[i-2]*A[i-1] + B[i-2]) + b*A[i-1];
        A[i] = (-a*A[i-2]*B[i-1] - b*B[i-1] - d)/p[i];
        B[i] = -e/p[i];
        C[i] = (-a*(A[i-2]*C[i-1] + C[i-2]) - b*C[i-1] + F[i-1])/p[i];
    }
    // обратный ход
    A[K-1] = a*A[K-2] + b;
    B[K-1] = a*B[K-2] + c;
    C[K-1] = F[K-1] - a*C[K-2];
    long double s;
    s = a*A[K-3] + b;
    A[K] = s*A[K-2] + a*B[K-3] + c;
    B[K] = s*B[K-2] + d;
    C[K] = F[K-2] - a*C[K-3] - s*C[K-2];
    long double x[K];
    x[K-1] = (C[K]*A[K-1] - A[K]*C[K-1])/(B[K]*A[K-1] - B[K-1]*A[K]);
    x[K-2] = (C[K-1] - B[K-1]*x[K-1])/A[K-1];
    for (int i = (K-3); i >= 0; i--) {                    // вычисление оставшихся переменных
        x[i] = A[i+1]*x[i+1] + B[i+1]*x[i+2] + C[i+1];
    }
    // вывод ответа на экран (заметим, что первые 2 и последние 2 элемента не зависят от числа переменных)
    printf("x[0] = 1.000000\nx[1] = 1.000000\n");
    for (int i = 0; i < K; i++) {
        printf("x[%d] = %Lf\n", i+2, x[i]);
    }
    printf("x[%d] = 1.000000\nx[%d] = 1.000000\n", N-2, N-1);
    return 0;
}
```

## 2.5 Тесты

### 2.5.1 Тест №1

Пусть $N = 12$. Заметим, что первые 2 и последние 2 элемента вычислять при каждом тесте не имеет смысла, т.к. их значения всегда равны 1 независимо от количества переменных. Результат программы приведен ниже.

$x[0] = 1.000000$
$x[1] = 1.000000$
$x[2] = 1.015400$
$x[3] = 1.034077$
$x[4] = 1.047980$
$x[5] = 1.052376$
$x[6] = 1.046602$
$x[7] = 1.033316$
$x[8] = 1.017525$
$x[9] = 1.005453$
$x[10] = 1.000000$
$x[11] = 1.000000$

Решая описанную выше СЛАУ при помощи WolframAlpha, получаем следующий результат:
$x[0] = 1.000000$
$x[1] = 1.000000$
$x[2] = 1.015400$
$x[3] = 1.034079$
$x[4] = 1.047982$
$x[5] = 1.052378$
$x[6] = 1.046604$
$x[7] = 1.033317$
$x[8] = 1.017526$
$x[9] = 1.005453$
$x[10] = 1.000000$
$x[11] = 1.000000$

Как можно видеть, результаты с учетом погрешности практически совпадают.

### 2.5.2 Тест №2

Решим исходную задачу при $N = 100$. Результат программы приведен ниже.

```
x[0] = 1.000000
x[1] = 1.000000
x[2] = 1.000232        x[3] = 1.000671        x[4] = 1.001295        x[5] = 1.002082
x[6] = 1.003011        x[7] = 1.004061        x[8] = 1.005213        x[9] = 1.006449
x[10] = 1.007751       x[11] = 1.009103       x[12] = 1.010489       x[13] = 1.011895
x[14] = 1.013306       x[15] = 1.014711       x[16] = 1.016096       x[17] = 1.017450
x[18] = 1.018763       x[19] = 1.020027       x[20] = 1.021231       x[21] = 1.022368
x[22] = 1.023431       x[23] = 1.024414       x[24] = 1.025311       x[25] = 1.026117
x[26] = 1.026828       x[27] = 1.027441       x[28] = 1.027954       x[29] = 1.028363
x[30] = 1.028668       x[31] = 1.028868       x[32] = 1.028963       x[33] = 1.028953
x[34] = 1.028840       x[35] = 1.028624       x[36] = 1.028308       x[37] = 1.027894
x[38] = 1.027386       x[39] = 1.026786       x[40] = 1.026099       x[41] = 1.025328
x[42] = 1.024479       x[43] = 1.023555       x[44] = 1.022563       x[45] = 1.021508
x[46] = 1.020396       x[47] = 1.019231       x[48] = 1.018022       x[49] = 1.016773
x[50] = 1.015491       x[51] = 1.014182       x[52] = 1.012854       x[53] = 1.011513
x[54] = 1.010165       x[55] = 1.008817       x[56] = 1.007476       x[57] = 1.006148
x[58] = 1.004839       x[59] = 1.003557       x[60] = 1.002306       x[61] = 1.001093
x[62] = 0.999923       x[63] = 0.998802       x[64] = 0.997735       x[65] = 0.996728
x[66] = 0.995784       x[67] = 0.994907       x[68] = 0.994102       x[69] = 0.993373
x[70] = 0.992721       x[71] = 0.992149       x[72] = 0.991660       x[73] = 0.991255
x[74] = 0.990935       x[75] = 0.990701       x[76] = 0.990552       x[77] = 0.990487
x[78] = 0.990505       x[79] = 0.990605       x[80] = 0.990783       x[81] = 0.991036
x[82] = 0.991360       x[83] = 0.991750       x[84] = 0.992200       x[85] = 0.992705
x[86] = 0.993258       x[87] = 0.993849       x[88] = 0.994471       x[89] = 0.995114
x[90] = 0.995768       x[91] = 0.996422       x[92] = 0.997063       x[93] = 0.997678
x[94] = 0.998254       x[95] = 0.998777       x[96] = 0.999229       x[97] = 0.999596
x[98] = 0.999859
x[99] = 1.000000
x[100] = 1.000000
```

## 2.6 Где на практике может возникнуть данная СЛАУ?

Подобные матрицы часто встречаются при численном решении краевых задач для дифференциальных уравнений 4-го порядка, при моделировании некоторых инженерных задач.

# Глава 3

# Сравнение решений

## 3.1 Сравнение решений задач на основе количества операций

### 3.1.1 Задача №1

Число арифметических операций для нахождения коэффициентов равно $6(N-2)+2+5$, для определения неизвестных — $2(N-1)$. Итак, всего арифметических операций в методе прогонки для задачи №1 — примерно $8N$.

### 3.1.2 Задача №2

Нетрудно подсчитать число действий умножения и деления, необходимых для решения данной СЛАУ при помощи метода прогонки (для пятидиагональной матрицы). Итак, мы получим, что выполняется примерно $14n$ операций.

### 3.1.3 Сравнение количества операций в задаче №1 и задаче №2

Учитывая вышесказанное, очевидно, 1-я задача будет выполняться быстрее 2-й ($8n < 14n$).

## 3.2 Сравнение решений задач на основе нормы невязки

При вычислении вектора невязки (с помощью формулы $r = Ax-f$, где $A$ — исходная матрица системы, $x$ — вектор неизвестных, $f$ — вектор, соответствующий правой части СЛАУ) для обеих задач был получен вектор, элементы которого были крайне близки к нулю. Подсчитывая его норму (как корня суммы квадратов элементов этого вектора), было получено значение, близкое к нулю, что говорит о высокой точности полученных решений.

## 3.3 Вывод

При выполнении данной практической работы был внимательно изучен метод прогонки. Данный метод является экономичным и требует для своей реализации число операций, пропорциональное $n$. Для сравнения можно отметить, что метод Гаусса требует примерно $\frac{2}{3}(n)^3$ операций, что существенно увеличивает затраты по времени при большой размерности матрицы. Если матрица системы позволяет использовать метод прогонки, то следует использовать именно его.