Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ»

Отчёт по лабораторной работе № 1

«Решение СЛАУ с трёхдиагональной матрицей методом прогонки»

по курсу

«Численные методы»

Студент группы ИУ9-61Б Бакланова А.Д.

Преподаватель Домрачева А.Б.

Москва, 2020 г.

Цель работы

Целью данной работы является реализация программы для решения разреженных СЛАУ методом прогонки, в которых ненулевые элементов ограниченное количество.

Задание

В ряде случаев практическая задача сводится к решению разреженных СЛАУ, в которых ненулевые элементов ограниченное количество.

Как правило, речь идет о ленточных матрицах, где ненулевые элементы находятся на главной диагонали и смежных с ней.

Ширина ленты l вычисляется по формуле (1):

$$S = 2l + 1 \tag{1}$$

Дано: $A\bar{x} = \bar{d}, A \in \mathbb{R}^{n \cdot n}, \bar{d} \in \mathbb{R}^n$

Найти: $\bar{x} \in \mathbb{R} -?$

$$\begin{pmatrix} a_1 & b_1 & 0 & \dots & 0 \\ c_1 & a_2 & b_2 & \dots & 0 \\ 0 & \dots & & \dots & 0 \\ 0 & \dots & & \dots & b_{n-1} \\ 0 & \dots & & c_{n-1} & a_n \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \dots \\ d_{n-1} \\ d_n \end{pmatrix}$$

Тогда имеем систему:

$$\begin{cases} a_1x_1 + b_1x_2 = d_1 \\ c_1x_1 + a_2x_2 + b_2x_3 = d_2 \\ \cdots \\ c_{n-1}x_{n-1} + a_nx_n = d_n \end{cases}$$

Далее выражаем x_1 через x_2 и так далее, пока не останется строчка с x_n

$$x_1 = \frac{d_1}{a_1} - \frac{b_1}{a_1} \cdot x_2$$

Далее заменяем на вид $x_1 = \alpha_1 \cdot x_2 + \beta_1$, где $\beta_1 = \frac{d_1}{a_1}$, а $\alpha_1 = -\frac{b_1}{a_1}$

И так далее индуктивно $x_i = \alpha_i \cdot x_{i+1} + \beta_i$

Итого, имеется система:

$$\begin{cases} \alpha_i = -\frac{b_i}{\alpha_{i-1} \cdot c_{i-1} + a_i} \\ \beta_i = \frac{d_i - c_{i-1} \cdot \beta_{i-1}}{\alpha_{i-1} \cdot c_{i-1} + a_i} \end{cases}, i = \overline{2,..n}, \quad \alpha_i = -\frac{b_1}{a_1}, a_1 \neq 0, \quad \beta_1 = \frac{d_1}{a_1}$$

$$x_i = \alpha_i \cdot x_{i+1} + \beta_i$$

$$i = \overline{n-1,..,1}, \quad x_n = \beta_n$$

Решение имеется только при выполнении трёх условий диагонального преобладания:

1) $|a_i| \geqslant |c_{i-1} + b_i|$

2) $\frac{|b_i|}{|a_i|} \leqslant 1$

3) $\frac{|c_{i-1}|}{|b_i|} \leqslant 1$

Реализация:

```cpp
#include <cmath>
#include <iostream>

int main(int argc, const char * argv[]) {
    int n = 0;

    std::cout << "Enter size of matrix \n";
    std::cin >> n;

    float *diag_a = new float[n];
    float *diag_b = new float[n-1];
    float *diag_c = new float[n-1];
    float *free_d = new float[n];

    std::cout << "Enter main diagonal \n";
    for (int i=0; i<n; i++) {
        std::cin >> diag_a[i];
    }

    std::cout << "Enter upper diagonal \n";
    for (int i=0; i<n-1; i++) {
        std::cin >> diag_b[i];
    }

    std::cout << "Enter lower diagonal \n";
    for (int i=0; i<n-1; i++) {
        std::cin >> diag_c[i];
    }

    std::cout << "Enter free numbers d \n";
    for (int i=0; i<n; i++) {
        std::cin >> free_d[i];
    }

    float *alpha = new float[n-1];
    float *beta = new float[n-1];
    float *x = new float[n];

    //------------------------------
    for (int i=1; i<n-1; i++) {
        if (std::abs(diag_a[i]) < std::abs(diag_c[i-1] + diag_b[i])) {
            std::cout << "Input is incorrect 1 stat ";
        }
        if (std::abs(diag_b[i])/(std::abs(diag_a[i])) > 1) {
            std::cout << "Input is incorrect 2 stat";
        }
```

```
        if (std::abs(diag_c[i-1]) / (std::abs(diag_b[i])) > 1) {
            std::cout << "Input is incorrect 3 stat";
        }
    }

    if (diag_a[0] != 0) {
        alpha[0] = - diag_b[0]/diag_a[0];
    } else {
        std::cout << "a[0] = 0 error";
        return 1;
    }
    beta[0] = free_d[0] / diag_a[0];

    for (int i=1; i<n-1; i++) {
        alpha[i] = -(diag_b[i])/(alpha[i-1]*diag_c[i-1] + diag_a[i]);
        beta[i] = (free_d[i]-diag_c[i-1]*beta[i-1])/(alpha[i-1]*diag_c[i-1] + diag_a[i]);
        //std::cout << "alpha[" << i <<"]= " << alpha[i];
        //std::cout << "beta[" << i <<"]= " << beta[i];
    }
    beta[n-1] = (free_d[n-1]-diag_c[n-2]*beta[n-2])/(alpha[n-2]*diag_c[n-2] + diag_a[n-1]);
    //std::cout << "beta[" << n-1 <<"]= " << beta[n-1];
    x[n-1] = beta[n-1];

    for (int i=n-2; i>=0; i--) {
        x[i] = alpha[i]*x[i+1] + beta[i];
    }

    for (int i=0; i<n; i++) {
        std::cout << "x[" << i << "]= " << x[i] << std::endl;
    }

    return 0;
}
```

Тестирование:

$$\begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \\ 6 \\ 5 \end{pmatrix}$$

```
Enter size of matrix
4
Enter main diagonal
4 4 4 4
Enter upper diagonal
1 1 1
Enter lower diagonal
1 1 1
Enter free numbers d
5 6 6 5
x[0]= 1
x[1]= 1
x[2]= 1
x[3]= 1
Program ended with exit code: 0
```

Чтобы вычислить вектор ошибки $\overline{e}$:

$A\overline{x^*} = \overline{d^*}$

$A(\overline{x} - \overline{x^*}) = (\overline{d} - \overline{d^*})$

$A\overline{e} = \overline{r}$, где $\overline{r}$ — вектор невязки

$A^{-1}A\overline{e} = A^{-1}\overline{r}$

$\overline{e} = A^{-1}\overline{r}$

$\overline{x} = \overline{e} + \overline{x^*}$

```
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            d[i] += matrix[i][j] * x[j];
        }
        std::cout << "d[" <<i<<"]= " << d[i] << std::endl;
    }

    for(int i=0; i<n; i++) {
        r[i] = free_d[i]-d[i];
        std::cout << "r[" <<i<<"]= " << r[i] << std::endl;
    }

    inversion(matrix, n);

    for (int i=0; i<4; i++) {
        for (int j=0; j<4; j++) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }

    for (int i=0; i<n; i++) {
        //sum = 0;
        for (int j=0; j<n; j++) {
            e[i]+= matrix[i][j] * r[j];
        }
    }

    for (int i=0; i<n; i++) {
        std::cout << "--" << std::endl;
        printf("%.15f", e[i]);
    }

    for (int i=0; i<n; i++) {
        std::cout << "----" << std::endl;

        float c =x[i]-e[i];
        printf("%.15f", c);
    }

    return 0;
}
```

```
error:
-0.000000069358137
0.000000208074397
-0.000000078027902
0.000000026009300
x[1]= 1.000000119209290
x[2]= 0.999999701976776
x[3]= 1.000000119209290
x[4]= 1.000000000000000
Program ended with exit code: 0
```

Вывод:

Программа реализована и правильно считает разреженные СЛАУ методом прогонки, однако имеет место быть математическая погрешность.