Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1
Задание №3, часть I по курсу «Численные методы»

Студент:	С. М. Власова
Преподаватель:	И. Э. Иванов
Группа:	М8О-306Б
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Задание №1.3

Реализовать метод простых итераций в виде программы, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.
Вариант: 12

$$\begin{cases} 14 \cdot x_1 - 4 \cdot x_2 - 2 \cdot x_3 + 3 \cdot x_4 = 38 \\ -3 \cdot x_1 + 23 \cdot x_2 - 6 \cdot x_3 - 9 \cdot x_4 = -195 \\ -7 \cdot x_1 - 8 \cdot x_2 + 21 \cdot x_3 - 5 \cdot x_4 = -27 \\ -2 \cdot x_1 - 2 \cdot x_2 + 8 \cdot x_3 + 18 \cdot x_4 = 142 \end{cases}$$

1 Описание метода решения

Метод простых итераций

Методы последовательных приближений, в которых при вычислении последующего приближения решения используются предыдущие, уже известные приближенные решения, называются *итерационными*. Их сущность состоит в том, что сначала записывается некоторая последовательность столбцов матрицы, после чего производится поочередное вычисление каждого столбца. Каждый новый столбец вычисляется на основе вычисленных предыдущих, при этом с каждым вычислением получается всё более точное приближение искомого решения. Когда достигнута необходимая *точность*, процесс вычисления прерывают и в качестве решения используют последний вычисленный столбец.
Таким образом, метод позволяет получить значения корней системы с заданной точностью в виде предела последовательности некоторых векторов (итерационный процесс). Для решения СЛАУ с разреженными матрицами предпочтительнее использовать именно итерационные методы.

Рассмотрим СЛАУ

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots + a_{1n} \cdot x_n = b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \ldots + a_{2n} \cdot x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \ldots + a_{nn} \cdot x_n = b_n \end{cases}$$

с невырожденной матрицей A ($detA \neq 0$).

Приведем СЛАУ к эквивалентному виду

$$\begin{cases} x_1 = \beta_1 + \alpha_{11} \cdot x_1 + \alpha_{12} \cdot x_2 + \ldots + \alpha_{1n} \cdot x_n \\ x_2 = \beta_2 + \alpha_{21} \cdot x_1 + \alpha_{22} \cdot x_2 + \ldots + \alpha_{2n} \cdot x_n \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n = \beta_n + \alpha_{n1} \cdot x_1 + \alpha_{n2} \cdot x_2 + \ldots + \alpha_{nn} \cdot x_n \end{cases}$$

или в векторно-матричной форме

$$x = \beta + \alpha \cdot x$$

$$x = \begin{pmatrix} x1 \\ \vdots \\ x_n \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \quad \alpha = \begin{pmatrix} \alpha_{11} & \ldots & \alpha_{1n} \\ \vdots & \ldots & \vdots \\ \alpha_{1n} & \ldots & \alpha_{nn} \end{pmatrix},$$

Такое приведение может быть выполнено различными способами. Одним из наиболее распространенных является следующий.
Разрешим исходную систему относительно неизвестных при ненулевых диагональных элементах $a_{ii} \neq 0, \quad i = \overline{1,n}$ (если какой-либо коэффициент на главной диагонали равен нулю, достаточно соответствующее уравнение поменять местами с любым другим уравнением). Получим следующие выражения для компонентов вектора β и матрицы α эквивалентной системы:

$$\beta_i = \frac{b_i}{a_{ii}}; \quad \alpha_{ij} = -\frac{a_{ij}}{a_{ii}}, \quad i,j = \overline{1,n}, \quad i \neq j;$$

$$\alpha_{ij} = 0, \quad i = j, \quad i = \overline{1,n}.$$

При таком способе приведения СЛАУ к эквивалентному виду метод простых итераций носит название **метода Якоби.**

В качестве нулевого приближения $x^{(0)}$ вектора неизвестных примем вектор правых частей $x^{(0)} = \beta$ или $(x_1^{(0)} \quad x_2^{(0)} \quad \ldots x_n^{(0)})$.
Тогда метод простых итераций имеет вид:

$$\begin{cases} x^{(0)} = \beta \\ x^{(1)} = \beta + \alpha \cdot x^{(0)} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ x^{(k)} = \beta + \alpha \cdot x^{(k-1)} \end{cases}$$

Из полученной схемы решения видно преимущество итерационных методов по сравнению, например, с методом Гаусса. В вычислительном процессе участвуют только произведения матрицы на вектор, что позволяет работать только с ненулевыми элементами матрицы, значительно упрощая процесс хранения и обработки матриц.

Достаточное условие сходимости метода простых итераций:

Метод простых итераций сходится к единственному решению эквивалентной СЛАУ, а следовательно, и к решению исходной СЛАУ при любом начальном приближении $x^{(0)}$, если какая-либо норма матрицы α эквивалентной системы меньше единицы $||\alpha|| < 1$.
Если используется метод Якоби для эквивалентной СЛАУ, то достаточным условием сходимости является диагональное преобладание матрицы A, т.е.

$$|a_{ii}| > \sum_{j=1, i \neq j}^{n} |a_{ij}|, \quad \forall i$$

для каждой строки матрицы A модули элементов, стоящих на главной диагонали, больше суммы модулей недиагональных элементов.

При выполнении достаточного условия сходимости оценка погрешности решения на k-й итерации дается выражением:

$$||x^{(k)} - x^*|| \leq \varepsilon^{(k)} = \frac{||\alpha||}{1 - ||\alpha||} \cdot ||x^{(k)} - x^{(k-1)}||,$$

где x^* — точное решение СЛАУ.
Процесс итераций останавливается при выполнении условия $\varepsilon^{(k)} \leq \varepsilon$, где ε — задаваемая точность.

2 Протокол

Входные данные я храню в файле *data3*: первая строка — размерность матрицы, на следующих строках — матрица СЛАУ, коэффициенты вектора правых частей и заданная точность.
Выходные данные я записываю в файл *res3*.
Скриншот консоли:

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ g++ 1.3.cpp -o 1.3
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat data3
4
14 -4 -2 3
-3 23 -6 -9
-7 -8 21 -5
-2 -2 8 18
38 -195 -27 142
0.01
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ ./1.3 < data3 > res3
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat res3
Число итераций метода простых итераций: 9
  x_1 = -0.999996
  x_2 = -5.99997
  x_3 = -1.99999
  x_4 = 7.99996
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Точное решение СЛАУ:

$$x^* = \begin{pmatrix} -1 \\ -6 \\ -2 \\ 8 \end{pmatrix}$$

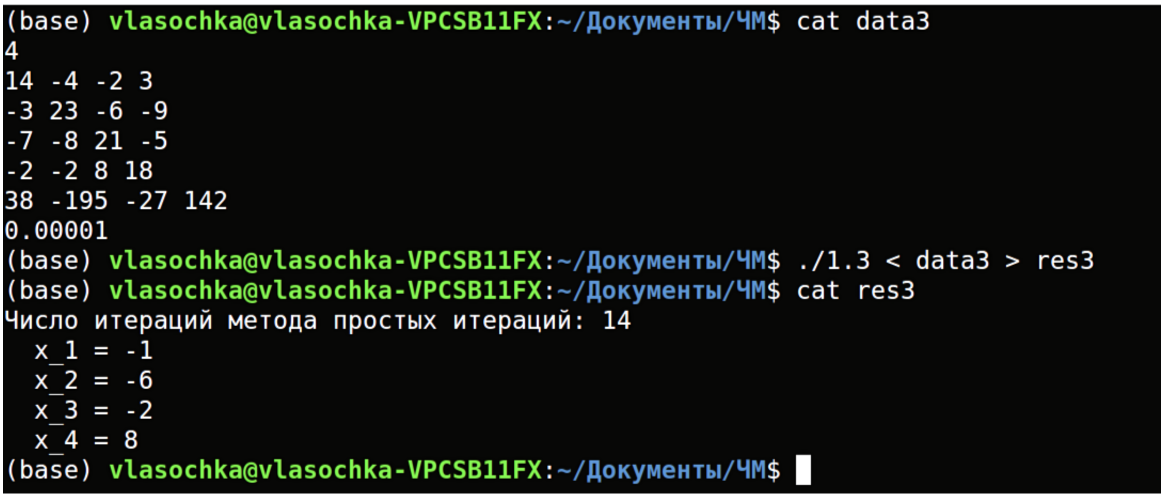
Можно видеть, что программа корректно вычислила решение системы с точностью $\varepsilon = 0.01$.

$$x = \begin{pmatrix} -0.(9) \\ -5.(9) \\ -1.(9) \\ 7.(9) \end{pmatrix}$$

Попробуем найти оценку погрешности, при которой алгоритм найдет приблизительное решение, совпадающее с точным решением. Для этого будем уменьшать оценку погрешности.

Пусть $\varepsilon = 0.00001$.

Скриншот консоли:



```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat data3
4
14 -4 -2 3
-3 23 -6 -9
-7 -8 21 -5
-2 -2 8 18
38 -195 -27 142
0.00001
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ ./1.3 < data3 > res3
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat res3
Число итераций метода простых итераций: 14
  x_1 = -1
  x_2 = -6
  x_3 = -2
  x_4 = 8
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Можно видеть, что найденное решение СЛАУ совпало с точным решением. Для достижения такого результата алгоритму потребовалось 14 итераций.

$$x^* = \begin{pmatrix} -1 \\ -6 \\ -2 \\ 8 \end{pmatrix} = x$$

3 Исходный код

Листинг 1: Метод простых итераций

```
#include <iostream>
#include <vector>
#include <cmath>

void Swap(std::vector<std::vector<double>>& M, int i)
{
    int n = M.size();
    int k = i + 1;
    while(M[k][i] == 0)
        k++;
    std::vector<double> N(n);
    N = M[i];
    M[i] = M[k];
    M[k] = N;
}

void PrintMatrix(std::vector<std::vector<double>>& M)
{
    for(int i = 0; i < M.size(); i++){
        for(int j = 0; j < M[0].size(); j++)
            std::cout << M[i][j] << " ";
        std::cout << std::endl;
    }
}

int IterMethod(std::vector<std::vector<double>>& A, std::vector<double>& X, double eps)
{
    int n = A.size();
    int it_count = 0;
    double norma = 0;
    double kaf = 0;
    double sum = 0;
    std::vector<double> X_1(n, 0);  //extra vector, storing the result of previous iteration

    for( int i = 0; i < n; i++)   // leading matrix to x = beta + alpha * x
    {
        if( A[i][i] == 0)
            Swap(A, i);
        for( int j = 0; j <= n; j++)
```

```
        {
            if(j == i)
                continue;
            else
            {
                if( j != n)
                {
                    A[i][j] = (-1)* A[i][j]/ A[i][i];
                    sum += fabs(A[i][j]);
                }
                else
                    A[i][j] /= A[i][i];
            }
        }

        if( sum > norma)    //norma of Alfha matrix
        {
            norma = sum;
            sum = 0;
        }
        A[i][i] = 0;
    }
    norma /= (1 - norma);    //coefficient

    for(int i = 0; i < n; i++)
        X_1[i] = A[i][n];

    while(true)
    {
        for(int i = 0; i < n; i++)
        {
            X[i] = 0;
            for(int j = 0; j < n; j++)
                X[i] += A[i][j]* X_1[j];
            X[i] += A[i][n];
            if( kaf < fabs(X[i] - X_1[i]))
            kaf = fabs(X[i] - X_1[i]);
        }
        it_count++;
        if(eps > kaf*norma)
            break;
        X_1 = X;
        kaf = 0;
    }
```

```
    return it_count;
}

int main()
{
    int n;
    double element, eps;

    std::cin >> n;

    std::vector<std::vector<double>> A(n, std::vector<double>(n + 1));
    std::vector<double> X(n, 0);

    for( int i = 0; i < n; i++)
        for( int j = 0; j < n; j++)
            std::cin >> A[i][j];
    for(int i = 0; i < n; i++ )
        std::cin >> A[i][n];

    std::cin >> eps;

    std::cout << "The number of iterations of Jacobi method: " << IterMethod(A, X, eps) <<
        std::endl;
    for(int i = 0; i < n; i++)
        std::cout << " x_" << i + 1 << " = " << X[i] << std::endl;
    return 0;
}
```

4 Выводы

Выполнив первую часть третьего задание первой лабораторной работы, я познакомилась с итерационным методом поиска решений СЛАУ — методом простых итераций. Этот метод заключается в последовательном поиске векторов-приближений, которые постепенно сходятся к точному решению СЛАУ. Критерием окончания алгоритма является достижение заданной точности. Таким образом, алгоритм является более экономичным по памяти — в вычислительном процессе участвуют только произведения матрицы на вектор, что позволяет работать только с ненулевыми элементами матрицы, значительно упрощая процесс хранения и обработки матриц.