Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
Санкт-Петербургский университет аэрокосмического приборостроения

КАФЕДРА № 2

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

доц., канд. техн. наук		С.Л. Козенко
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ № 2

Решение СЛАУ

Вариант 5

по дисциплине: ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4128	11.03.2023	В.А. Воробьёв
		подпись, дата	инициалы, фамилия

Санкт – Петербург, 2023

СОДЕРЖАНИЕ

1 Цели и постановка задачи

1.1 Цели работы

а) освоение основных методов решения систем линейных алгебраических уравнений (СЛАУ);

б) совершенствование навыков по алгоритмизации и программированию вычислительных задач.

1.2 Задание

Составить схему алгоритма и программу на языке C/C++ решения задачи по теме «Решение СЛАУ» в соответствии с индивидуальным заданием.

Решить систему линейных уравнений $AX = B$ методом Зейделя, где

$$A = \begin{pmatrix} 29 & 6 & 3 & 8 \\ 4 & 26 & 7 & 4 \\ 2 & 3 & 25 & 3 \\ 4 & 8 & 3 & 27 \end{pmatrix}, \quad B = \begin{pmatrix} 3 \\ 1 \\ 4 \\ 2 \end{pmatrix}$$

Рисунок 1 – Вариант задания

2 Описание метода решения

Итерационные методы позволяют получить значения корней системы с заданной точностью в виде предела последовательности некоторых векторов C(0), C(1), …, C(k) . Процесс получения элементов такой последовательности носит итерационный (повторяющийся) характер. Эффективность применения таких методов зависит от удачного выбора начального вектора C(0) и быстроты сходимости процесса.

Метод Зейделя относится к итерационным методам решения систем линейных уравнений, обеспечивающим хорошую сходимость итерационного процесса поиска корней системы. Пусть задана система. Прежде всего необходимо задать значения начальных (нулевых) приближений корней C1 (0), C2 (0),..., Cm(0). Если отсутствует какая–нибудь информация об этих значениях, то их можно принять равными свободным членам системы уравнений или даже принять равными нулю. Выбранные таким образом нулевые приближения корней подставим в первое уравнение системы и получим первое приближение корня C1.

$$C_1^{(1)} = \beta_1 + \alpha_{12} C_2^{(0)} + \alpha_{13} C_3^{(0)} + \dots + \alpha_{1\,m} C_m^{(0)}.$$

Используя во втором уравнении системы найденное первое приближение корня C1 и нулевые приближения остальных корней, получим первое приближение корня C2

$$C_2^{(1)} = \beta_2 + \alpha_{21} C_1^{(1)} + \alpha_{23} C_2^{(0)} + \dots + \alpha_{2\,m} C_m^{(0)}.$$

Повторяя эту процедуру последовательно для всех уравнений системы, получим в итоге первое приближение корня.

$$C_m^{(1)} = \beta_m + \alpha_{m\,1} C_1^{(1)} + \alpha_{m2} C_2^{(1)} + \dots + \alpha_{m\,\mathrm{m}-1} C_{m-1}^{(1)}.$$

Используя первые приближение корня системы, можно аналогичным образом найти вторые приближения.

$$C_1^{(2)} = \beta_1 + \alpha_{12}\, C_2^{(1)} + \alpha_{13}\, C_3^{(1)} + \ldots + \alpha_{1\,m} C_m^{(1)}$$
$$C_2^{(2)} = \beta_1 + \alpha_{21}\, C_1^{(2)} + \alpha_{23}\, C_3^{(1)} + \ldots + \alpha_{2\,m} C_m^{(1)}$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$C_m^{(2)} = \beta_m + \alpha_{m\,1}\, C_1^{(2)} + \alpha_{m\,2}\, C_2^{(2)} + \ldots + \alpha_{m\,m-1}\, C_{m-1}^{(2)}.$$

Затем, используя вторые приближения, можно вычислить третьи и т.д. Итерационный процесс решения системы линейных уравнений методом Зейделя сходится к единственному решению при любом выборе начальных приближений искомых корней, если выполняется одно из условий.

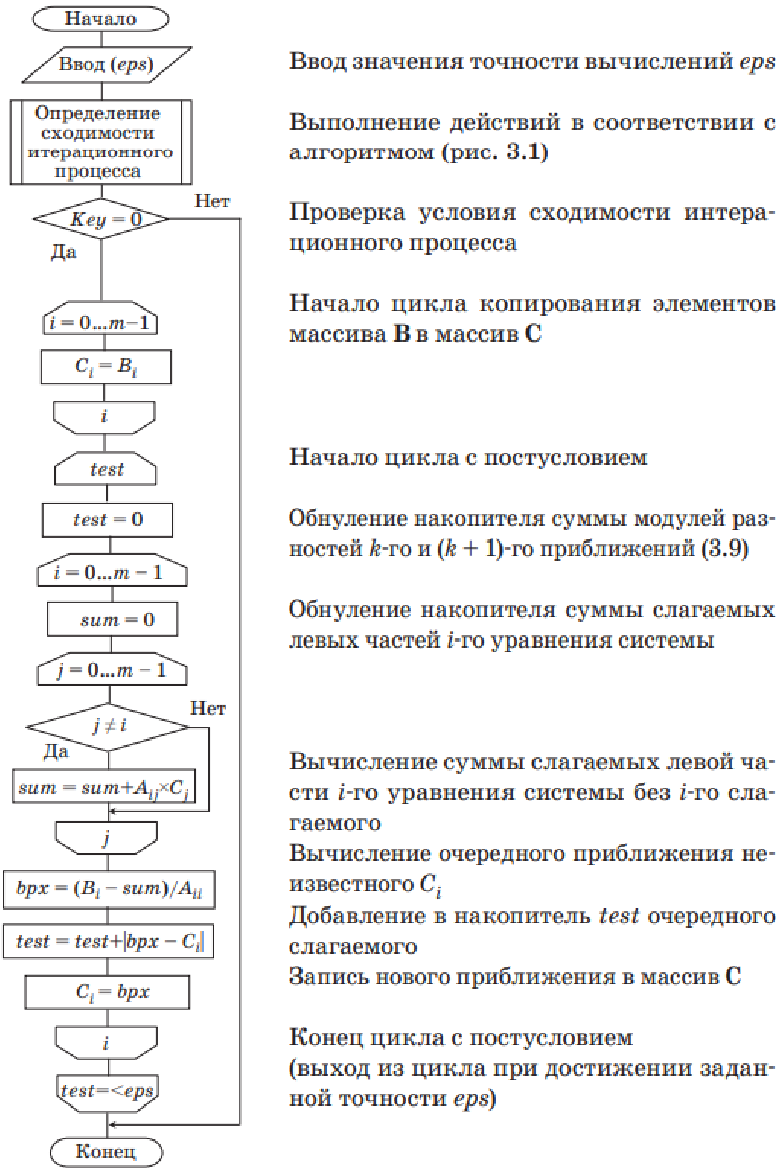


Рисунок 2 – Блок схема алгоритма

Условия сходимости выполняются, если в матрице диагональные элементы преобладают, то есть

$$|\alpha_{ii}| \geq \sum_{j \neq i}^{m} |\alpha_{ij}|. \qquad (1)$$

Для итерационного метода Зейделя достаточно, чтобы хотя бы для одного i неравенство было строгим.

3 Аналитические расчеты

Для аналитических расчетов используем online-калькулятор (URL: https://math.semestr.ru/optim/zeidel.php), и приложим его решение в виде скриншотов.

Метод Зейделя.

Метод Зейделя представляет собой модификацию метода простой итераций.

Имеем СЛАУ: A x =b (1)

Предполагая, что $a_{ii} \neq 0$ разрешим новое уравнение системы (1) относительно x_1, второе – относительно x_2,…, *n-ое* уравнение – относительно x_n. В результате получим:

$x_1=\beta_1 - \alpha_{12}x_2 - \alpha_{13}x_3 - \dots - \alpha_{1n}x_n$

$x_2=\beta_2 - \alpha_{21}x_1 - \alpha_{23}x_3 - \dots - \alpha_{2n}x_n$

$x_n=\beta_n - \alpha_{n1}x_n - \alpha_{n3}x_3 - \dots - \alpha_{nn-1}x_{n-1}$

где $\beta_i=b_i/a_{ii}$; $\alpha_{ij}=a_{ij}/a_{ii}$ при $i \neq j$; $\alpha_{ii}=0$

Известно начальное приближение: $x^0=(x^0{}_1, x^0{}_2, \dots, x^0{}_n)$.

Основная идея заключается в том, что при вычислении *(k+1)-го* приближения неизвестной x_i учитываются уже вычисленные ранее (k+1) - приближение неизвестных $x_1, x_2, \dots, x_n$.

Итерационная схема имеет вид:

$x^{k+1}{}_1=\beta_1 - \sum\alpha_{1j}x^k{}_j$

$x^{k+1}{}_2=\beta_2 - \alpha_{21}x^{k+1}{}_1 - \sum\alpha_{2j}x^k{}_j$

$x^{k+1}{}_i=\beta_i - \sum\alpha_{ij}x^{k+1}{}_1 - \sum\alpha_{2j}x^k{}_j$

Прежде чем применять метод, необходимо переставить строки исходной системы таким образом, чтобы на диагонали стояли наибольшие по модулю коэффициенты матрицы.

29	6	3	8
4	26	7	4
2	3	25	3
4	8	3	27

Приведем к виду:

$x_1=0.1034 - (0.21x_2+0.1x_3+0.28x_4)$

$x_2=0.0385 - (0.15x_1+0.27x_3+0.15x_4)$

$x_3=0.16 - (0.08x_1+0.12x_2+0.12x_4)$

$x_4=0.0741 - (0.15x_1+0.3x_2+0.11x_3)$

Рисунок 3 – Решение с помощью online-калькулятора

Покажем вычисления на примере нескольких итераций.

N=1

x_1=0.1034 - 0*0.2069 - 0*0.1034 - 0*0.2759=0.1034

x_2=0.0385 - 0.1034*0.1538 - 0*0.2692 - 0*0.1538=0.0225

x_3=0.16 - 0.1034*0.08 - 0.0225*0.12 - 0*0.12=0.149

x_4=0.0741 - 0.1034*0.1481 - 0.0225*0.2963 - 0.149*0.1111=0.0355

N=2

x_1=0.1034 - 0.0225*0.2069 - 0.149*0.1034 - 0.0355*0.2759=0.0736

x_2=0.0385 - 0.0736*0.1538 - 0.149*0.2692 - 0.0355*0.1538=-0.0184

x_3=0.16 - 0.0736*0.08 - (-0.0184)*0.12 - 0.0355*0.12=0.1521

x_4=0.0741 - 0.0736*0.1481 - (-0.0184)*0.2963 - 0.1521*0.1111=0.0517

N=3

x_1=0.1034 - (-0.0184)*0.2069 - 0.1521*0.1034 - 0.0517*0.2759=0.0773

x_2=0.0385 - 0.0773*0.1538 - 0.1521*0.2692 - 0.0517*0.1538=-0.0223

x_3=0.16 - 0.0773*0.08 - (-0.0223)*0.12 - 0.0517*0.12=0.1503

x_4=0.0741 - 0.0773*0.1481 - (-0.0223)*0.2963 - 0.1503*0.1111=0.0525

Остальные расчеты сведем в таблицу.

N	x_1	x_2	x_3	x_4	e_1	e_2	e_3	e_4
0	0	0	0	0				
1	0.103	0.0225	0.149	0.0355	0.103	0.0225	0.149	0.0355
2	0.0736	-0.0184	0.152	0.0517	-0.0299	-0.00411	0.00305	0.0162
3	0.0773	-0.0223	0.15	0.0525	0.00369	0.00388	-0.00178	0.000802
4	0.078	-0.0221	0.15	0.0524	0.000766	-0.000237	-0.000186	-0.000163

Для оценки погрешности вычисляем коэффициент α:

$$max[\sum|\alpha_{ij}|] = 0.2069 + 0.1034 + 0.2759 = 0.5862 < 1$$

max[|x^3,x^4|] = $\rho(x^3, x^4)$ = |0.05238 - 0.05254| = 0.000766

Вычисляем погрешность:

$$\rho(x,x^4) \leq \frac{\alpha}{1-\alpha}\rho(x^3,x^4) = \frac{0.5862}{1-0.5862}0.000766 \leq 0.00109$$

Рисунок 4 – Решение с помощью online-калькулятора

Подставим полученные значения, для проверки тождественности:

$$0.078 * 29 - 0.0221 * 6 + 0.15 * 3 + \ 0.0524 * 8 \ \approx 3$$

$$0.078 * 4 - 0.0221 * 26 + 0.15 * 7 + \ 0.0524 * 4 \ \approx 1$$

$$0.078 * 2 - 0.0221 * 3 + 0.15 * 25 + \ 0.0524 * 3 \approx 4$$

$$0.078 * 4 - 0.0221 * 8 + 0.15 * 3 + \ 0.0524 * 27 \ \approx 2$$

По выше полученным значениям, мы можем утверждать, что представленное решение является верным.

4 Схема алгоритма решения задачи

Схему алгоритма опишем при помощи блок-схем, опустив специфику ввода-вывода данных.

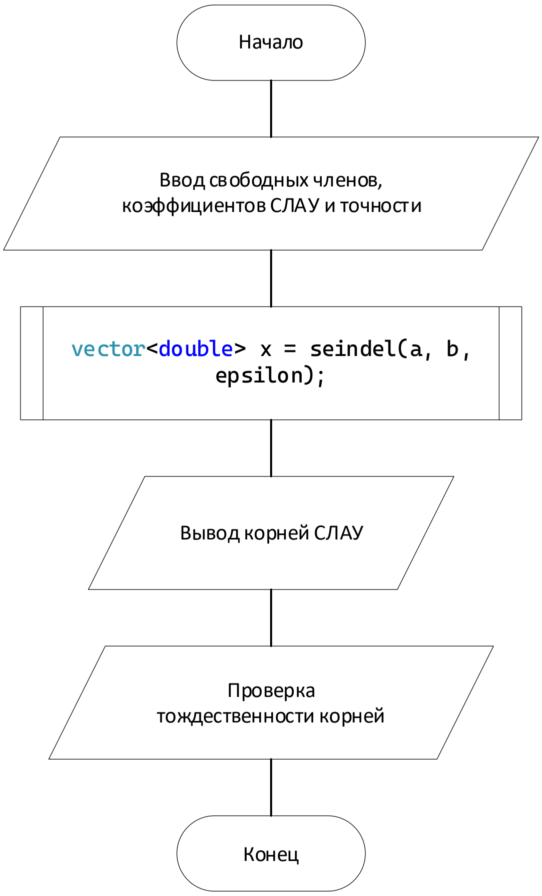


Рисунок 5 – Код функции main

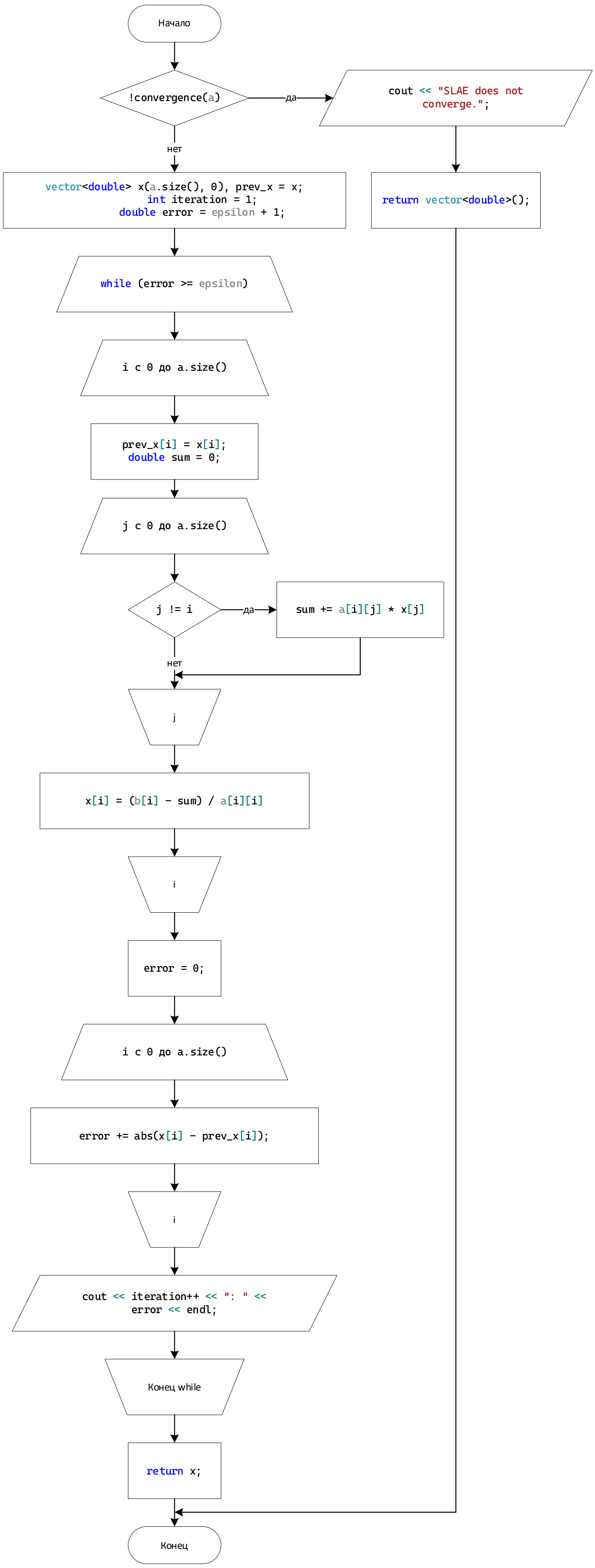


Рисунок 6 – Код функции seindel

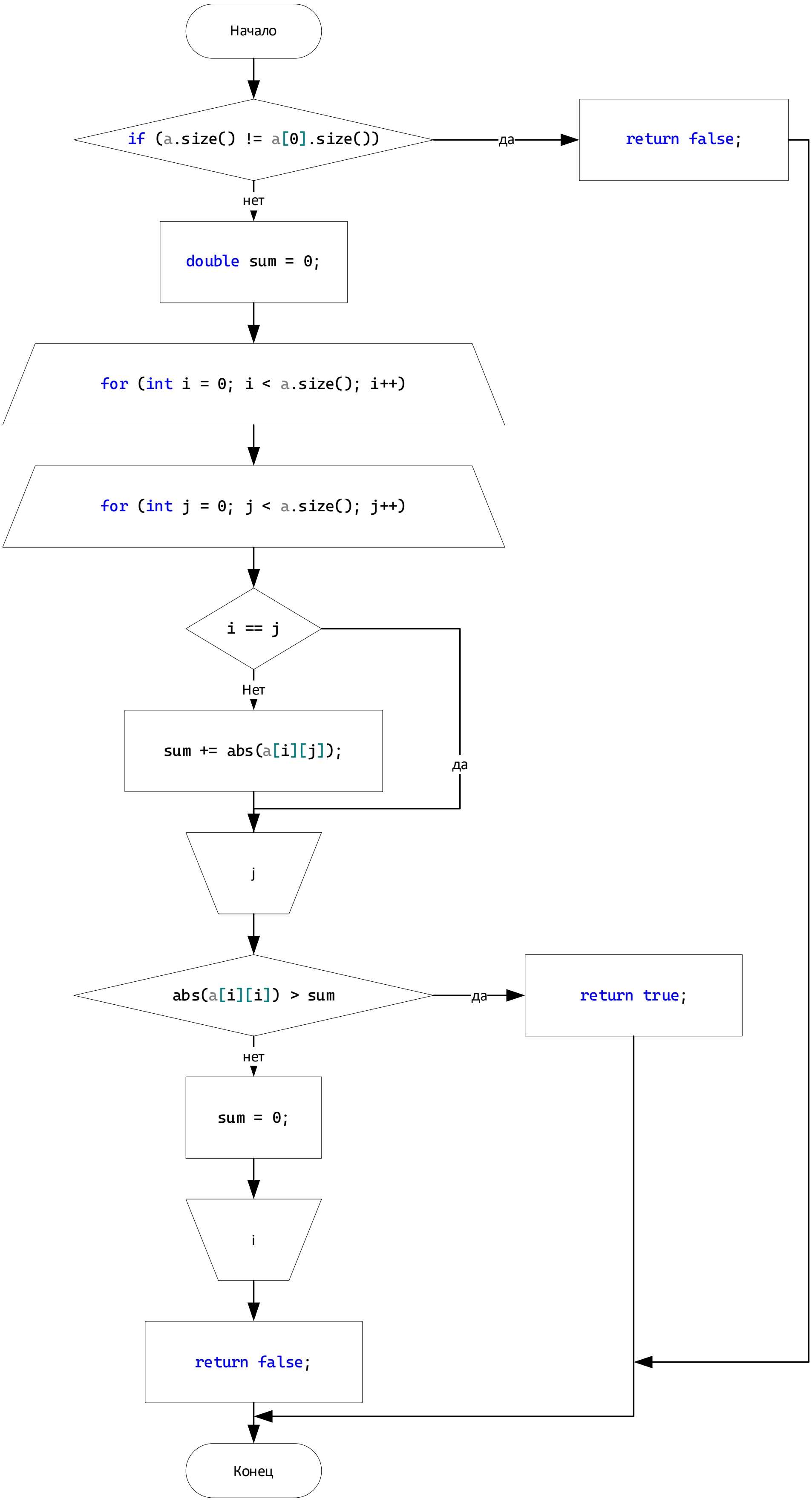


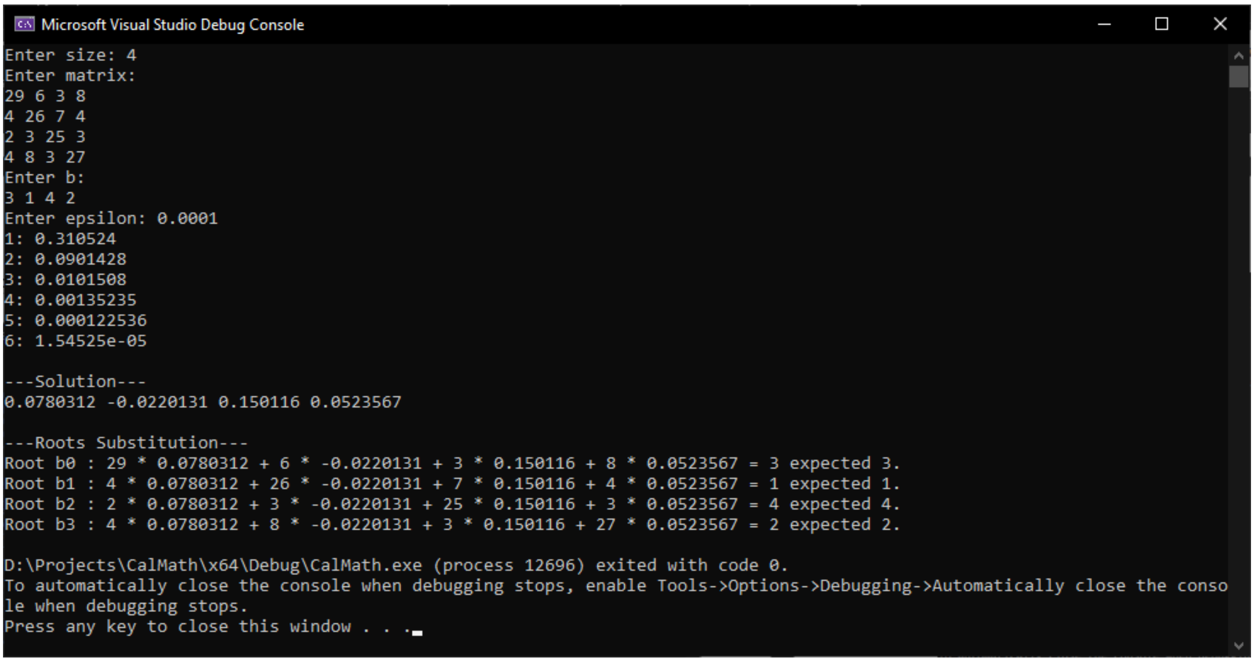
Рисунок 7 – Код функции convergence

5 Текст программы на C++

Исходный код можно посмотреть в Приложении А. Весь алгоритм решения СЛАУ методом Зейделя сосредоточен в функции seindel, написанной на основе алгоритма, изображенного на Рисунке 2. Код проверки СЛАУ на сходимость сосредоточен в функции convergence, используя формулу (1). Остальной программный код контролирует ввод-вывод данных.

Также код доступен на GitHub(URL: https://github.com/vladcto/SUAI_homework/blob/c069dfefbb7be0fd774821ded30921f97ade3277/4_semester/ComputationalMathematics/%D0%BB%D1%802/solution.cpp)

6 Результаты программных расчетов



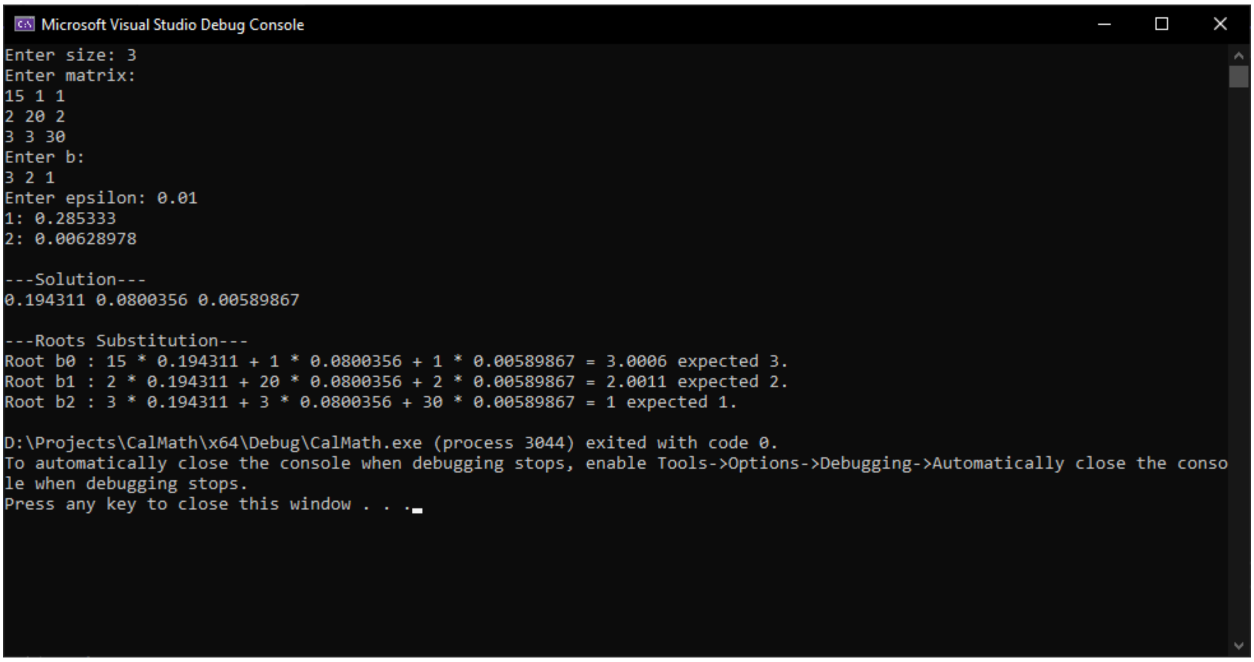
```
Microsoft Visual Studio Debug Console
Enter size: 4
Enter matrix:
29 6 3 8
4 26 7 4
2 3 25 3
4 8 3 27
Enter b:
3 1 4 2
Enter epsilon: 0.0001
1: 0.310524
2: 0.0901428
3: 0.0101508
4: 0.00135235
5: 0.000122536
6: 1.54525e-05

---Solution---
0.0780312 -0.0220131 0.150116 0.0523567

---Roots Substitution---
Root b0 : 29 * 0.0780312 + 6 * -0.0220131 + 3 * 0.150116 + 8 * 0.0523567 = 3 expected 3.
Root b1 : 4 * 0.0780312 + 26 * -0.0220131 + 7 * 0.150116 + 4 * 0.0523567 = 1 expected 1.
Root b2 : 2 * 0.0780312 + 3 * -0.0220131 + 25 * 0.150116 + 3 * 0.0523567 = 4 expected 4.
Root b3 : 4 * 0.0780312 + 8 * -0.0220131 + 3 * 0.150116 + 27 * 0.0523567 = 2 expected 2.

D:\Projects\CalMath\x64\Debug\CalMath.exe (process 12696) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Рисунок 8 – Результат работы программы с предоставленными исходными данными



```
Microsoft Visual Studio Debug Console
Enter size: 3
Enter matrix:
15 1 1
2 20 2
3 3 30
Enter b:
3 2 1
Enter epsilon: 0.01
1: 0.285333
2: 0.00628978

---Solution---
0.194311 0.0800356 0.00589867

---Roots Substitution---
Root b0 : 15 * 0.194311 + 1 * 0.0800356 + 1 * 0.00589867 = 3.0006 expected 3.
Root b1 : 2 * 0.194311 + 20 * 0.0800356 + 2 * 0.00589867 = 2.0011 expected 2.
Root b2 : 3 * 0.194311 + 3 * 0.0800356 + 30 * 0.00589867 = 1 expected 1.

D:\Projects\CalMath\x64\Debug\CalMath.exe (process 3044) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Рисунок 9 – Результат работы программы с произвольными исходными данными

7 Сравнение программных и аналитических расчётов

Результаты аналитических и программных расчётов представлены на картинках 3-4 и 8-9 соответственно.

Аналитические/программные результаты:

- x1: 0.078 / 0.078
- x2: -0.0221 / -0.0220
- x3: 0.15 / 0.15
- x4: 0.0524 / 0.0524

В программных расчетах по сравнению с аналитическими отличается лишь один корень. Это не является ошибкой, так как эта погрешность связана со спецификой округления чисел. Исходя из этого, мы может утверждать, что написанная нами программа корректна и выполняет поставленную перед ней задачу.

8 Выводы

В ходе выполнения лабораторной работы был освоен метод Зейделя для решения СЛАУ.

Метод Зейделя — это итеративный метод решения СЛАУ, основанный на последовательном изменении приближений для решения на каждом шаге. Он использует предыдущее приближение для вычисления следующего и так далее, пока не будет достигнуто достаточное совпадение в рамках точности.

По требуемому варианту была написана программа вычисления корней СЛАУ на языке программирования C++. Результат выполнения программы был сравнен с аналитическими расчетами, из чего мы пришли к выводу, что программа написана корректно и выполняет поставленные задачи.

Приложение А

Листинг программы

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

using namespace std;

bool convergence(vector<vector<double>> a) {
	if (a.size() != a[0].size()) return false;
	double sum = 0;
	for (int i = 0; i < a.size(); i++) {
		for (int j = 0; j < a.size(); j++) {
			if (i == j) continue;
			sum += abs(a[i][j]);
		}
		if (abs(a[i][i]) > sum) return true;
		sum = 0;
	}
	return false;
}

vector<double> seindel(vector<vector<double>> a,
	vector<double> b,
	double epsilon) {
	if (!convergence(a)) {
		cout << "SLAE does not converge.";
		return vector<double>();
	}

	vector<double> x(a.size(), 0), prev_x = x;
	int iteration = 1;
	double error = epsilon + 1;

	while (error >= epsilon)
	{
		// Seindel method.
		for (int i = 0; i < a.size(); i++)
		{
			prev_x[i] = x[i];
			double sum = 0;
			for (int j = 0; j < a.size(); j++)
			{
				if (j != i)
				{
					sum += a[i][j] * x[j];
				}
			}
			x[i] = (b[i] - sum) / a[i][i];
		}

		// Calculate error.
		error = 0;
		for (int i = 0; i < a.size(); i++)
		{
			error += abs(x[i] - prev_x[i]);
		}

		cout << iteration++ << ": " << error << endl;
	}

	return x;
```

```
}

void printSubstitutionRoots(vector<vector<double>> a, vector<double> x,
vector<double> b) {
	const double round_sign = 10000;
	cout << "\n---Roots Substitution---\n";
	for (int i = 0; i < x.size(); i++) {
		cout << "Root b" << i << " : ";
		double b_res = 0;
		for (int j = 0; j < a.size(); j++) {
			cout << a[i][j] << " * " << x[j] << (j != a.size() - 1 ? " + " :
"");

			b_res += a[i][j] * x[j];
		}
		cout << " = "
			<< round(b_res * round_sign) / round_sign
			<< " expected " << round(b[i] * round_sign) / round_sign
			<< ".\n";
	}
}

vector<double> split_str(string inp) {
	vector<double> res;
	string token = "";
	inp += " ";
	for (int i = 0; i < inp.size(); i++) {
		if (inp[i] == ' ') {
			res.push_back(stod(token));
			token.clear();
		}
		else {
			token += inp[i];
		}
	}
	return res;
}

int main()
{
	vector<vector<double>> a;
	vector<double> b;
	double epsilon;

	//User data input.
	{
		double size;
		string inp;
		cout << "Enter size: ";
		cin >> size;

		cout << "Enter matrix: " << endl;
		getline(cin, inp);
		for (int i = 0; i < size; i++) {
			getline(cin, inp);
			a.push_back(split_str(inp));
		}

		cout << "Enter b:" << endl;
		getline(cin, inp);
		b = split_str(inp);

		cout << "Enter epsilon: ";
		cin >> epsilon;
	}
```

```
    vector<double> x = seindel(a, b, epsilon);
    cout << "\n---Solution---\n";
    if (x.empty()) {
        cout << "SLAE solutions is empty.";
        return 0;
    };
    for (int i = 0; i < x.size(); i++)
    {
        cout << x[i] << " ";
    }
    cout << endl;
    printSubstitutionRoots(a, x, b);
    return 0;
}
```