Отчет по практическому заданию 2
Спектр матрицы.
Метод Чебышева.

Хлопинская Арина
301 группа

7 декабря 2024 г.

1. Постановка задания:

Для заданной симметричной положительно определенной матрицы $A \in R^{n \times n}$, представленной в строковом виде в файле формата CSV, требуется выполнить следующие шаги:

1. Сгенерировать случайный вектор-столбец решений x с равномерно распределенными на отрезке $[-1, 1]$ компонентами $x_i, i = 1, 2, \ldots, n$.

2. Вычислить правую часть F системы уравнений $x + Ax = F$ (или $(E + A)x = F$).

3. Решить систему уравнений $(E+A)x = F$ прямым методом (QR-разложение с помощью поворотов Гивенса и обратная подстановка) и вычислить среднеквадратическую норму погрешности решения.

4. Используя теорему Гершгорина, оценить спектр матрицы системы уравнений $(E + A)$.

5. Написать программу на C/C++, реализующую метод Чебышева для решения системы линейных алгебраических уравнений. Решить систему $(E+A)x = F$ методом Чебышева с оптимальным набором итерационных параметров, обеспечивающих устойчивость решения к ошибкам округления.

   - Количество итераций взять равным степени двойки.
   - Подобрать наименьший показатель степени, при котором погрешность решения на последней итерации в среднеквадратической норме не превосходит погрешность прямого метода.
   - Начальное приближение итерационного метода взять равным нулю.

6. Построить график зависимости среднеквадратической нормы погрешности решения от номера итерации метода Чебышева.

7. Вычислить относительную погрешность решения, полученного методом Чебышева.

2. Первая часть задания

Случайным образом сгенерирован вектор-столбец решений x с равномерно распределенными на отрезке $[-1,1]$ компонентами $x_i, i = 1, 2, \ldots, n$.

По известному решению x вычислена правая часть F системы уравнений $x + Ax = F$.

Система уравнений решена прямым методом (QR-разложение с помощью поворотов Гивенса и обратная подстановка). Среднеквадратическая норма погрешности решения составила 5.87262×10^{-15}.

3. Оценка спектра матрицы методом Гершгорина

Для оценки спектра матрицы системы уравнений $(E + A)x = F$ воспользуемся теоремой Гершгорина. Теорема утверждает, что каждое собственное значение λ матрицы B размера $n \times n$ лежит хотя бы в одном из кругов Гершгорина на комплексной плоскости. Круг с центром в диагональном элементе b_{ii} имеет радиус $r_i = \sum_{j \neq i} |b_{ij}|$, т.е. $|\lambda - b_{ii}| \leq r_i$. В нашем случае, для матрицы $B = E + A$, центры кругов $c_i = 1 + a_{ii}$, а радиусы $r_i = \sum_{j \neq i} |a_{ij}|$. Объединение всех кругов Гершгорина даёт область, содержащую весь спектр матрицы $(E + A)$. Оценка спектра будет представлена в виде интервалов $[c_i - r_i, c_i + r_i]$ для каждого i и итогового интервала, содержащего весь спектр: $[\lambda_{min}, \lambda_{max}]$.
Следствием данной теоремы является нераевнства вида

$$\lambda_1 \leq \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$$

$$\lambda_n \geq \min_{1 \leq i \leq n} |a_{ii}| - \left(\sum_{j=1, j \neq i}^{n} |a_{ij}| \right)$$

Оценка спектра: [1.0000000000, 152.6000000000]
Таким образом, результат работы функции void gershgorin(const vector<vector<double» A):

- Maximum value: 152.600000
- Minimum value: 1.000000

4. Метод Чебышева с оптимальным набором итерационных параметров

Метод Чебышева с оптимальным набором итерационных параметров, обеспечивающих устойчивость к ошибкам округления, основан на следующей итерационной схеме:

$$B\frac{y_{k+1}-y_k}{\tau_{k+1}} + Ay_k = f, \quad k = 0, 1, \ldots, \quad y_0 \in H, \tag{1}$$

где y_0 – начальное приближение (в нашем случае нулевой вектор), $\{\tau_k\}$ – последовательность итерационных параметров, а B – произвольный невырожденный оператор (в упрощенном случае $B = E$).

Предположения:

A, B, D таковы, что оператор $DB^{-1}A$ самосопряжен. Заданы γ_1 и γ_2 – постоянные энергетические эквивалентности операторов D и $DB^{-1}A$:

$$\gamma_1 D \leq DB^{-1}A \leq \gamma_2 D, \quad \gamma_1 > 0, \quad DB^{-1}A = (DB^{-1}A)^* \tag{2}$$

В простейшем случае $D = B = E$, и по условию A симметрична и действительна.

Оператор D вводится из исследования сходимости итерационной схемы. Пусть z_k - погрешность. Подставим $y_k = z_k + u$ в итерационную схему: $B\frac{z_k - z_{k-1}}{\tau_k} + Az_{k-1} = 0, k = 1, 2, \ldots$. Разрешим это уравнение относительно z_k: $z_k = (E - \tau_k B^{-1}A)z_{k-1}$ и положим $z_k = D^{-1/2}x_k$. Далее исследование сходимости в данной работе не рассматривается.

Оптимальные итерационные параметры:

$$\tau_k = \frac{\tau_0}{1 + \rho_0 \mu_k} \tag{3}$$

где

$$\begin{aligned}
\mu_k &\in \left\{-\cos\left(\frac{(2i-1)\pi}{2n}\right), i = 1, 2, \ldots, n\right\} \\
\tau_0 &= \frac{2}{\gamma_1 + \gamma_2} \\
\rho_0 &= \frac{1-\xi}{1+\xi} \\
\xi &= \frac{\gamma_1}{\gamma_2}
\end{aligned}$$

n – число итераций (степень двойки).

Устойчивость к ошибкам округления:

Если обозначить через M_n множество корней полинома Чебышева $T_n(x)$:

$$M_n = \left\{ -\cos\frac{2i-1}{2n}\pi, \quad i = 1, 2, \ldots, n \right\}, \tag{4}$$

то получим следующую формулу для итерационных параметров:

$$\tau_k = \frac{\tau_0}{1 + \rho_0 \mu_k}, \quad \mu_k \in M_n, \quad k = 1, 2, \ldots, n. \tag{5}$$

Здесь $\mu_k \in M_n$ означает, что в качестве μ_k должны выбираться последовательно все элементы множества M_n.

Исходя из множества $\theta_1 = \{1\}$, построим множество θ_{2^p} по следующему правилу. Пусть множество θ_m построено. Тогда множество θ_{2m} определим по формулам

$$\theta_{2m} = \left\{ \theta_{2m}^{2i} = 4m - \theta_m^i; \quad \theta_{2m}^{2i-1} = \theta_{2m}^i, \quad i = 1, 2, \ldots, m \right\}, \quad m = 1, 2, 4, \ldots, 2^{p-1}. \tag{6}$$

Нетрудно убедиться, что множество θ_{2^k} состоит из нечетных чисел от 1 до $2^{k+1} - 1$.

Используя построенное множество θ_{2^p}, упорядочим множество M_{2^p} следующим образом:

$$M_n = \left\{ \cos\beta_i, \quad \beta_i = \frac{\pi}{2n}\theta_n^i, \quad i = 1, 2, \ldots, n \right\}, \quad n = 2^p. \tag{7}$$

Это и есть искомое упорядочение множества M_n в случае, когда $n = 2^p$.

Оценка границ спектра:

γ_1 и γ_2 определяются из неравенства $\gamma_1(x, x) \leq (Ax, x) \leq \gamma_2(x, x)$ для произвольного вектора x. Для учёта ошибок округления можно использовать:

$$\gamma_1 = \lfloor \frac{(Ax, x)}{(x, x)} \rfloor - 10$$

$$\gamma_2 = \lceil \frac{(Ax, x)}{(x, x)} \rceil + 10$$

Подбор оптимального количества итераций:

Последним шагом осталось подобрать оптимальное количество итераций. Чтобы погрешность в среднеквадратической норме итерационного метода не превосходила погрешность прямого метода, следует взять число итерация равным 64. Это значение подобрано программно.

5. Листинг программы на C++

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <chrono>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <cassert>
#include <random>

using namespace std;
using namespace chrono;

// Функция для вычисления нормы вектора
double vector_norm(const vector<double>& v) {
   double max_val = 0.0;
   for (double val : v) {
       max_val = max(max_val, abs(val));
   }
   return max_val;
}


// Функция для вычисления матричной нормы (подчиненной максимум-норме)
double matrix_norm(const vector<vector<double>>& A) {
   int n = A.size();
   double max_norm = 0.0;
   for (int i = 0; i < n; ++i) {
       double row_sum = 0.0;
       for (int j = 0; j < n; ++j) {
           row_sum += abs(A[i][j]);
       }
       max_norm = max(max_norm, row_sum);
   }
   return max_norm;
}

// Функция для умножения матриц
vector<vector<double>> matrix_multiply(const vector<vector<double>>& A,
const vector<vector<double>>& B) {
   int n = A.size();
   int m = B[0].size();
   int p = B.size();
```

```
    vector<vector<double>> C(n, vector<double>(m, 0.0));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            for (int k = 0; k < p; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return C;
}


// Функция для вычисления разности матриц
vector<vector<double>> matrix_subtract(const vector<vector<double>>& A,
const vector<vector<double>>& B) {
    int n = A.size();
    vector<vector<double>> C(n, vector<double>(n, 0.0));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
    return C;
}

// Функция для транспонирования матрицы
vector<vector<double>> transpose(const vector<vector<double>>& A) {
    int n = A.size();
    int m = A[0].size();
    vector<vector<double>> At(m, vector<double>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            At[j][i] = A[i][j];
        }
    }
    return At;
}

// Функция для установки диагональных элементов R положительными
void ensure_positive_diagonal(vector<vector<double>>& Q,
vector<vector<double>>& R) {
    int n = min(Q.size(), R.size());
    for (int i = 0; i < n; ++i) {
```

```
        if (R[i][i] < 0) {
            // Инвертируем знак столбца Q
            for (size_t j = 0; j < Q.size(); ++j) {
                Q[j][i] = -Q[j][i];
            }
            // Инвертируем знак строки R
            for (size_t j = i; j < R[0].size(); ++j) {
                R[i][j] = -R[i][j];
            }
        }
    }
}

// Функция для установки небольших значений в ноль
void zero_small_values(vector<vector<double>>& R, double threshold =
1e-10) {
    int n = R.size();
    int m = R[0].size();
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if (i > j && abs(R[i][j]) < threshold)
                R[i][j] = 0.0;
}


// 1.QR-разложение методом Гивенса
pair<vector<vector<double>>, vector<vector<double>>>
qr_givens(vector<vector<double>> A) {
    int n = A.size();
    int m = A[0].size();
    assert(m <= n); // Обеспечиваем, что матрица не "широкая"

    vector<vector<double>> R = A;
    vector<vector<double>> Q(n, vector<double>(n, 0.0));

    // Инициализация Q как единичной матрицы
    for (int i = 0; i < n; ++i)
        Q[i][i] = 1.0;

    // auto start = high_resolution_clock::now();

    // Применяем повороты Гивенса для каждой поддиагональной позиции
    for (int j = 0; j < min(n, m); ++j) {
```

```
        for (int i = j + 1; i < n; ++i) {
            double a = R[j][j];
            double b = R[i][j];
            double c, s;

            if (b == 0.0) {
                c = 1.0;
                s = 0.0;
            } else {
                double r = hypot(a, b);
                c = a / r;
                s = -b / r;
            }

            // Обновляем матрицу R
            for (int k = j; k < m; ++k) {
                double temp = c * R[j][k] - s * R[i][k];
                R[i][k] = s * R[j][k] + c * R[i][k];
                R[j][k] = temp;
            }

            // Обновляем матрицу Q
            for (int k = 0; k < n; ++k) {
                double temp = c * Q[k][j] - s * Q[k][i];
                Q[k][i] = s * Q[k][j] + c * Q[k][i];
                Q[k][j] = temp;
            }
        }
    }

    ensure_positive_diagonal(Q, R);
    zero_small_values(R);

    // auto end = high_resolution_clock::now();
    // auto duration = duration_cast<milliseconds>(end - start);

    // cout << "Время выполнения QR разложения Гивенса: " <<
duration.count() << " ms" << endl;

    return make_pair(Q, R);
}

// Функция для чтения CSV файла в матрицу
```

```
vector<vector<double>> read_csv(const string& filename) {
   vector<vector<double>> matrix;
   ifstream file(filename);
   string line, cell;

   if (!file.is_open()) {
       cerr << "Не удалось открыть файл: " << filename << endl;
       exit(EXIT_FAILURE);
   }

   while (getline(file, line)) {
       vector<double> row;
       stringstream ss(line);
       while (getline(ss, cell, ',')) {
           row.push_back(stod(cell));
       }
       matrix.push_back(row);
   }

   file.close();
   return matrix;
}


// Функция для вывода матрицы
void print_matrix(const vector<vector<double>>& A) {
   for (const auto& row : A) {
       for (const auto& val : row) {
           cout << setw(10) << val << " ";
       }
       cout << endl;
   }
}

// Функция для решения системы уравнений Rx = Q^T * b методом обратной
подстановки
vector<double> back_substitution(const vector<vector<double>>& R, const
vector<double>& b) {
   int n = R.size();
   vector<double> x(n);
   for (int i = n - 1; i >= 0; --i) {
       x[i] = b[i];
       for (int j = i + 1; j < n; ++j) {
```

```
            x[i] -= R[i][j] * x[j];
        }
        x[i] /= R[i][i];
    }
    return x;
}

bool compare_vectors(const vector<double>& v1, const vector<double>&
v2, double tol = 1e-6) {
    if (v1.size() != v2.size()) return false;
    for (size_t i = 0; i < v1.size(); ++i) {
        if (abs(v1[i] - v2[i]) > tol) return false;
    }
    return true;
}


void gershgorin(const vector<vector<double>>& A) {
    int n = A.size();
    if (n == 0 || A[0].size() != n) {
        cerr << "Ошибка: некорректный размер матрицы." << endl;
        return;
    }

    vector<double> centers(n);
    vector<double> radii(n);
    double min_eigenvalue = numeric_limits<double>::max(); //
Инициализируем максимальным значением double
    double max_eigenvalue = numeric_limits<double>::lowest(); //
Инициализируем минимальным значением double

    for (int i = 0; i < n; ++i) {
        centers[i] = A[i][i];
        radii[i] = 0.0;
        for (int j = 0; j < n; ++j) {
            if (i != j) {
                radii[i] += abs(A[i][j]);
            }
        }

        // Вывод кругов и интервалов Гершгорина
        // cout << "Круг " << i + 1 << ": |λ - " << fixed <<
setprecision(10) << centers[i] << "| ≤ " << radii[i] << endl;
```

```
       // cout << "Интервал " << i + 1 << ": [" << fixed <<
setprecision(10) << centers[i] - radii[i] << ", " << centers[i] +
radii[i] << "]" << endl;

       min_eigenvalue = min(min_eigenvalue, centers[i] - radii[i]);
       max_eigenvalue = max(max_eigenvalue, centers[i] + radii[i]);
   }

   cout << "\nОценка спектра: [" << fixed << setprecision(10) <<
min_eigenvalue << ", " << max_eigenvalue << "]" << endl;
}


// Структура для хранения границ спектра
struct Spectrum {
   double min_eigenvalue;
   double max_eigenvalue;
};

// Функция для вычисления скалярного произведения векторов
double dot_product(const vector<double>& v1, const vector<double>& v2)
{
   double result = 0.0;
   for (size_t i = 0; i < v1.size(); ++i) {
       result += v1[i] * v2[i];
   }
   return result;
}

// Функция для вычисления Евклидовой нормы вектора
double euclidean_norm(const vector<double>& v) {
   return sqrt(dot_product(v, v));
}

// Функция для умножения матрицы на вектор
vector<double> matrix_vector_multiply(const vector<vector<double>>& A,
const vector<double>& x) {
   int n = A.size();
   vector<double> result(n, 0.0);
   for (int i = 0; i < n; ++i) {
       for (int j = 0; j < n; ++j) {
           result[i] += A[i][j] * x[j];
       }
```

```
    }
    return result;
}


// Функция для генерации упорядочивания Чебышева
vector<int> generate_chebyshev_ordering(int n) {
    if (n == 1) {
        return {0};
    }

    int m = n / 2;
    vector<int> prev_ordering = generate_chebyshev_ordering(m);
    vector<int> ordering;
    for (int i = 0; i < m; ++i) {
        ordering.push_back(2 * prev_ordering[i]);
        ordering.push_back(n - 1 - prev_ordering[i]);
    }
    return ordering;
}

// Метод Чебышева для решения СЛАУ
vector<double> chebyshev_method(const vector<vector<double>>& A, const
vector<double>& f, double gamma1, double gamma2, int n_iterations,
const vector<int>& ordering) {
    int n = A.size();
    vector<double> x(n, 0.0);
    double tau0 = 2.0 / (gamma1 + gamma2);
    double rho0 = (1.0 - gamma1 / gamma2) / (1.0 + gamma1 / gamma2);

    vector<double> r = f;  // r = (E+A)x - f
    for (int k = 0; k < n_iterations; ++k) {
        int mu_index = ordering[k];
        double mu_k = -cos((2.0 * mu_index + 1.0) * M_PI / (2.0 *
n_iterations));
        double tau_k = tau0 / (1.0 + rho0 * mu_k);

        vector<double> Ax = matrix_vector_multiply(A, x);
        for (int i = 0; i < n; ++i) {
            r[i] = f[i] - x[i] - Ax[i]; // Обновляем невязку r
            x[i] += tau_k * r[i];
        }
    }
```

```
    return x;
}

int main(int argc, char* argv[]) {
    // ввод
    if (argc != 2) {
        cout << "Использование: " << argv[0] << " <имя_файла.csv>" <<
endl;
        return EXIT_FAILURE;
    }
    string filename = argv[1];
    cout << "Чтение файла: " << filename << endl;
    vector<vector<double>> A = read_csv(filename);

    int n = A.size();
    if (n == 0 || A[0].size() != n) {
        cerr << "Ошибка: некорректный размер матрицы в файле." << endl;
        return 1;
    }

    // Создаем матрицу EA = E + A
    vector<vector<double>> EA = A;
    for (int i = 0; i < n; ++i) {
        EA[i][i] += 1.0;
    }
    vector<vector<double>> E(n, vector<double>(n, 0.0));
    for (int i = 0; i < n; ++i) {
        E[i][i] = 1.0;
    }

    // Вычисляем QR-разложение для EA
    pair<vector<vector<double>>, vector<vector<double>>> qr_result =
qr_givens(EA);
    vector<vector<double>>& Q = qr_result.first;
    vector<vector<double>>& R = qr_result.second;

    // Генерация случайного вектора x и вычисление f = Ax
    n = EA.size();
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis(-1.0, 1.0);

    //x_gen
```

```cpp
vector<double> x_true(n);
for (int i = 0; i < n; ++i) {
    x_true[i] = dis(gen);
}

vector<double> f(n, 0.0);
for(int i = 0; i < n; ++i){
    for(int j = 0; j < n; ++j){
         f[i] += EA[i][j] * x_true[j];
    }
}

// Решение системы Ax = f с помощью QR-разложения
vector<double> QTf(n, 0.0);
vector<vector<double>> QT = transpose(Q);
for(int i = 0; i < n; ++i){
     for(int j = 0; j < n; ++j){
         QTf[i] += QT[i][j] * f[j];
     }
}
vector<double> x = back_substitution(R, QTf);

// Оценка точности
// r = f - Ax
vector<double> r(n, 0.0);
for(int i = 0; i < n; ++i) {
    for(int j = 0; j < n; ++j) {
         r[i] -= EA[i][j] * x[j];
    }
}
for(int i = 0; i < n; ++i) {
     r[i] += f[i];
}
double r_norm = vector_norm(r);

vector<double> delta(n);
for(int i = 0; i < n; ++i) {
    delta[i] = x[i] - x_true[i];
}
double delta_norm = euclidean_norm(delta);

const int width = 15;
```

```
    cout << left << setw(width) << "x_true:" << "x:" << endl;
    cout << string(width * 2, '-') << endl; // Разделительная линия

    size_t min_size = min(x_true.size(), x.size());

    // Вывод элементов векторов построчно
    for (size_t i = 0; i < min_size; ++i) {
        cout << left << setw(width) << x_true[i] << x[i] << endl;
    }
    // Вывод результатов
    cout << "Среднеквадратичная норма погрешности решения прямым методом
- delta: " << delta_norm << endl;
    Spectrum spectrum_estimation;
    cout << "Оценка спектра матрицы (E + A) методом Гершгорина:" <<
endl;
    gershgorin(EA);
    spectrum_estimation.min_eigenvalue = 1; // Min eigenvalue from
Gershgorin
    spectrum_estimation.max_eigenvalue = 152.6; // Max eigenvalue from
Gershgorin



     // Метод Чебышева
    int n_iterations = 2;
    vector<double> x_cheb;
    double error_chebyshev = 0.0;
    vector<int> cheb_iterations;
    vector<double> cheb_errors;
    vector<double> cheb_relative_errors;
    vector<double> x_cheb_64;

    while (true) {
        vector<int> ordering =
generate_chebyshev_ordering(n_iterations);
        x_cheb = chebyshev_method(A, f,
spectrum_estimation.min_eigenvalue, spectrum_estimation.max_eigenvalue,
n_iterations, ordering);
        // Оценка погрешности (евклидова норма)
        vector<double> diff(n);
        for(int i = 0; i < n; ++i)
            diff[i] = x_cheb[i] - x_true[i];

        error_chebyshev = euclidean_norm(diff);
```

```
        cheb_iterations.push_back(n_iterations);
        cheb_errors.push_back(error_chebyshev);

        double relative_error = error_chebyshev /
euclidean_norm(x_true);
        cheb_relative_errors.push_back(relative_error);

        if(n_iterations == 64){
            x_cheb_64 = x_cheb;
        }
        if (error_chebyshev <= delta_norm) {
            break;  // Достигнута требуемая точность
        }
        n_iterations *= 2;

    }

    cout << "Оптимальное количество итераций метода Чебышева:" << endl;
    cout << "n = " << n_iterations << endl;
    cout << "\nПогрешности метода Чебышева на каждой итерации:" << endl;
    cout << left << setw(25) << "n_iterations"
         << "Среднеквадратическая ошибка"
         << "        Относительная ошибка" << endl;
    cout << string(80, '-') << endl;
    // Установка формата вывода в экспоненциальный
    cout << scientific << setprecision(6);
    for(size_t i = 0; i < cheb_iterations.size(); ++i){
        cout << left << setw(25) << cheb_iterations[i]
             << cheb_errors[i]
             << "                        "
             << cheb_relative_errors[i] << endl;
    }
    // Сброс формата вывода на стандартный
    cout.unsetf(ios::floatfield);
    cout << fixed;

    if(!x_cheb_64.empty()){
        cout << "\nСравнение x_cheb и x_true при n = 64:" << endl;
        cout << left << setw(15) << "x_true:" << "x_cheb:" << endl;
        cout << string(30, '-') << endl;
        for(size_t i=0; i < min(x_true.size(), x_cheb_64.size()); ++i){
            cout << left << setw(15) << x_true[i] << x_cheb_64[i] <<
endl;
```

```
        }
    }
    return 0;
}
```

6. Результаты и выводы

Среднеквадратическая ошибка прямым методом: 5.91107e-15
Метод Чебышева:
Абсолютная погрешность итерационного метода: 8.911769e-016.
Относительная погрешность: 1.727871e-016.
Количество оптимальных итераций: 64.
Оценка спектра: [1.0000000000, 152.6000000000]
Maximum value: 152.600000
Minimum value: 1.000000

Выводы:

Как видно из представленных результатов, метод Чебышева с оптимальным набором итерационных параметров позволяет достичь высокой точности решения системы линейных алгебраических уравнений. В данном случае, метод Чебышева показал абсолютную погрешность меньшую, чем погрешность прямого метода QR-разложения. Для достижения точности, сравнимой с прямым методом, потребовалось 64 итерации. Это подтверждает эффективность метода Чебышева для решения СЛАУ, особенно при больших размерах матрицы, когда прямые методы могут быть вычислительно затратными.

6. Комлиляция и запуск программы

```
$ g++ -o ex_2 ex_2.cpp -std=c++11
$ ./ex_2 SLAU_var_9.csv
```

```
arina@asus-arina:~/Desktop/chm/TASK2$ g++ -o ex_2 ex_2.cpp -std=c++11
arina@asus-arina:~/Desktop/chm/TASK2$ ./ex_2 SLAU_var_9.csv
Чтение файла: SLAU_var_9.csv
x_true:        x:
------------------------------
0.131353       0.131353
0.136354       0.136354
0.968496       0.968496
0.374686       0.374686
0.505579       0.505579
-0.969189      -0.969189
-0.0154223     -0.0154223
0.121349       0.121349
-0.0312298     -0.0312298
0.200727       0.200727
0.303847       0.303847
-0.27925       -0.27925
0.470137       0.470137
-0.525843      -0.525843
-0.867013      -0.867013
-0.593637      -0.593637
-0.0448257     -0.0448257
-0.194037      -0.194037
0.836622       0.836622
-0.00850222    -0.00850222
0.0890481      0.0890481
0.0177182      0.0177182
0.0701545      0.0701545
0.306928       0.306928
-0.392741      -0.392741
-0.208178      -0.208178
-0.411515      -0.411515
-0.44358       -0.44358
0.0330061      0.0330061
-0.0298294     -0.0298294
0.107467       0.107467
-0.0403396     -0.0403396
0.407284       0.407284
0.859975       0.859975
-0.122248      -0.122248
0.523362       0.523362
0.833419       0.833419
0.295562       0.295562
0.636438       0.636438
-0.208067      -0.208067
-0.515214      -0.515214
0.144951       0.144951
0.0465909      0.0465909
-0.123655      -0.123655
-0.4243        -0.4243
0.466698       0.466698
0.29179        0.29179
0.399717       0.399717
0.725334       0.725334
0.677459       0.677459
0.463108       0.463108
```

```
0.46199         0.46199
0.582382        0.582382
-0.654473       -0.654473
-0.277808       -0.277808
-0.98233        -0.98233
0.956535        0.956535
0.36762         0.36762
0.299576        0.299576
-0.302303       -0.302303
0.742263        0.742263
-0.422276       -0.422276
-0.980084       -0.980084
-0.487368       -0.487368
-0.299191       -0.299191
-0.740465       -0.740465
0.304815        0.304815
0.784378        0.784378
0.519333        0.519333
-0.0266554      -0.0266554
-0.990114       -0.990114
0.225204        0.225204
-0.10389        -0.10389
-0.170675       -0.170675
0.852069        0.852069
0.0249262       0.0249262
-0.911667       -0.911667
-0.420089       -0.420089
0.0602295       0.0602295
0.370294        0.370294
0.0447713       0.0447713
0.865973        0.865973
-0.509497       -0.509497
-0.94952        -0.94952
-0.0918742      -0.0918742
-0.851835       -0.851835
0.375159        0.375159
0.383319        0.383319
0.283574        0.283574
-0.0789344      -0.0789344
0.812283        0.812283
0.183438        0.183438
0.37813         0.37813
-0.864424       -0.864424
-0.163438       -0.163438
-0.0281677      -0.0281677
-0.821675       -0.821675
-0.743584       -0.743584
0.166593        0.166593
-0.91375        -0.91375
Среднеквадратичная норма погрешности решения прямым методом - delta: 5.91107e-15
Оценка спектра матрицы (E + A) методом Гершгорина:

Оценка спектра: [1.0000000000, 152.6000000000]
Оптимальное количество итераций метода Чебышева:
n = 64
```

Погрешности метода Чебышева на каждой итерации:

n_iterations	Среднеквадратическая ошибка	Относительная ошибка
2	4.640239e+00	8.996793e-01
4	3.524642e+00	6.833802e-01
8	3.756926e-01	7.284169e-02
16	2.203442e-03	4.272173e-04
32	7.944463e-08	1.540323e-08
64	8.911769e-16	1.727871e-16