МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Второе практичское задание по курсу лекций «Численные методы линейной алгебры»

ОТЧЕТ

о выполненном задании

студента 301 учебной группы факультета ВМК МГУ

Мартьянова Артема Олеговича

гор. Москва

2023 год

Содержание

Постановка задачи

Дана система уравнений $x + Ax = F$, где $A \in \mathcal{R}^{n \times n}$ - симметричная положительно определенная матрица. Матрица предоставляется в виде файла в формате csv. Требуется написать программу на языке программирования C (или C++), реализующую метод Чебышева решения СЛАУ с оптимальным набором итерационных параметров, обеспечивающих устойчивость решения к ошибкам округления. Количество итераций взять равным степени двойки, при котором погрешность решения на последней итерации в среднеквадратической норме не превосходит погрешность прямого метода. Начальное приближение взять равным нулю. Также требуется построить график среднеквадратической нормы погрешности решения как функции номера итерации метода Чебышева и вычислить относительную погрешность решения, полученного методом Чебышева. Оценить собственные значения с помощью теоремы Гершгорина.

Описание метода решения задачи

Решать СЛАУ будем с помощью метода Чебышева с оптимальным набором итерационных параметров. Далее опишем выбор всех параметров.

Метод

В общем случае итерационный метод Чебышева можно записать так:

$$B\frac{x^{k+1} - x^k}{\tau_{k+1}} + Ax^k = b, \qquad k = 0, 1, \ldots$$

Поскольку в настоящее время неизвестно регулярных способов хорошего выбора матрицы B для произвольной A, то положим $B = I$.
Согласно теории, итерационные параметры для наибольшей скорости схождения следует выбирать следующим образом:

$$\tau_j = \frac{\tau_0}{1 - \rho_0 \mu_j}, j = 1, 2, \ldots, k,$$

где

$$\mu_j \in \mathfrak{M}_n = \left\{ cos\frac{2i-1}{2n}\pi, \quad i = 1, 2, \ldots, n \right\}, \quad \tau_0 = \frac{2}{\lambda_k + \lambda_1}, \quad \rho_0 = \frac{\lambda_k - \lambda_1}{\lambda_k + \lambda_1},$$

где λ_k и λ_1 - максимальное и минимальное собственные значения матрицы A соответственно.
К сожалению, вычисления по этим формулам при произвольном использовании итерационных параметров не является устойчивым с точки зрения машинной арифметики. Но если выбирать эти параметры в определенном порядке, требуемая устойчивость все же будет. Далее опишем нужный порядок выбора параметров. Иными словами, нам нужно построить оптимальное упорядочение множества $\mathfrak{M}_n$.
Приведем решение этой задачи в случае, когда $n = 2^p$(в соответсвии с постановкой задачи). Обозначим через θ_m множество, состоящее из m целых чисел:

$$\theta_m = \{\theta_1^{(m)}, \theta_2^{(m)}, \ldots, \theta_m^{(m)}\}.$$

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

$$\theta_{2m} = \{\theta_{2i}^{(2m)} = 4m - \theta_i^{(m)}, \theta_{2i-1}^{(2m)} = \theta_i^{(m)}, \quad i = 1, 2, \ldots, m\}, \quad m = 1, 2, \ldots, 2^{p-1}.$$

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 $\mathfrak{M}_{2^p}$ следующим образом:

$$\mathfrak{M}_n^* = \left\{cos\beta_i, \quad \beta_i = \frac{\pi}{2n}\theta_i^{(n)}, \quad i = 1, 2, \ldots, n\right\}, \quad n = 2^p.$$

Такое построение обеспечивает минимальное влияние вычислительной погрешности на сходимость чебышевского метода.

Оценка собственных значений

Собственные значения(которые нужны для нахождения τ_0 и ρ_0) оценим с помощью теоремы Гершгорина.
В общем случае теорема Гершгорина говорит о том, что все собственные значения компексной матрицы лежат внутри так называемых кругов Гершгорина. Пусть A - комплексная матрица $n \times n$ с элементами a_{ij}. Обозначим через R_i сумму модулей внедиагональных элементов i-й строки (при $i \in \{1, \ldots n\}$):

$$R_i = \sum_{j \neq i} |a_{ij}|.$$

Рассмотрим $D(a_{ii}, R_i) \subseteq \mathbb{C}$ - круг с центром в a_{ii} и радиусом R_i. Такой круг называется кругом Гершгорина. Наша матрица A по условию положительно определенная, а значит все ее собственные значения вещественны. Тогда круги Гершгорина вырождаются в отрезки на вещественной прямой. Таким образом, чтобы получить оценку для собственных значений, нам нужно найти минимальное значения среди левых границ и максимальное среди правых:

$$\lambda_{min} = \min_{i=1,n}\{a_{ii} - R_i\} \leqslant \lambda_1, \lambda_n \leqslant \lambda_{max} = \max_{i=1,n}\{a_{ii} + R_i\}$$

Описание программы

Здесь приведем описание основных функций программы(не будем заострять внимание на перегрузках операторов, они были реализованы для большего удоства и лучшей читаемости кода).

- `std::vector<int> theta_set_construction(int m)`

  Функция, которая строит множество θ_m, использующееся для построения оптимальной последовательности итерационных параметров.

- `std::vector<double> optim_iterative_parameters_set(int n)`

  Функция, строящая оптимально упорядоченное множество $\mathfrak{M}_n^*$, описанное выше.

- `std::vector<double> eigenvalue_estimation(const std::vector<std::vector<double>>`

  Функция, рассчитывающая оценку собственных значений с помощью теоремы Гершгорина(подробное описание выше).

- `std::vector<double> chebyshevIteration(const std::vector<std::vector<double>>& A, const std::vector<double>& F, std::vector<float> &statX, std::vector<float> &statY, int maxIterations)`

  Функция, реализующая метод Чебышева. Принимает матрицу системы, правую часть, массивы для хранения данных, требующихсяя для дальнейшего построения графика, количество итераций,

Код программы

Код основной программы для решения СЛАУ методом Чебышева на языке C++:

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

// Импорт кода из первого задания(прямого метода)
#include "../lu.cpp"

template<typename T>
void
linspace(std::vector<T> &v, float start, float stop, int amount)
{
    double step = (stop - start) / (amount - 1);
    if (v.size() < amount) v.resize(amount);
    for (int i = 0; i < v.size(); ++i) {
        v[i] = start + step * i;
    }
}

// Перегрузки операторов
template<typename T>
std::ostream&
operator<<(std::ostream &out, const std::vector<T> &v)
{
    for (auto &it : v) out << it << " ";
    return out;
}

template<typename T>
std::vector<T>
operator*(const std::vector<std::vector<T>> &m1, const std::vector<std::
    vector<T>> &m2)
{
    if (m1[0].size() != m2.size()) throw "Matrix sizes doesnt match";
    std::vector<std::vector<T>> res(m1.size());
    for (auto &it : res) it.resize(m2[0].size());
    for (int i = 0; i < m1.size(); ++i)
    {
        for (int j = 0; j < m2[0].size(); ++j)
        {
            double sum = 0.0;
            for (int k = 0; k < m1[0].size(); ++k)
            {
                sum += m1[i][k] * m2[k][j];
            }
            res[i][j] = sum;
        }
    }
    return res;
}

template<typename T>
```

```
std::vector<T>
operator*(const std::vector<std::vector<T>> &m, const std::vector<T> &x)
{
    if (m[0].size() != x.size()) throw "Matrix and vector sizes doesnt
   match";
    std::vector<T> ret(x.size());
    for (int i = 0; i < x.size(); ++i)
    {
        double sum = 0.0;
        for (int j = 0; j < x.size(); ++j)
        {
            sum += m[i][j] * x[j];
        }
        ret[i] = sum;
    }
    return ret;
}

template<typename T>
std::vector<T>
operator-(const std::vector<T> &v1, const std::vector<T> &v2)
{
    if (v1.size() != v2.size()) throw "Vectors sizes doesnt match";
    std::vector<T> ret(v1.size());
    for (int i = 0; i < v1.size(); ++i)
    {
        ret[i] = v1[i] - v2[i];
    }
    return ret;
}

template<typename T>
std::vector<T>
operator+(const std::vector<T> &v1, const std::vector<T> &v2)
{
    if (v1.size() != v2.size()) throw "Vectors sizes doesnt match";
    std::vector<T> ret(v1.size());
    for (int i = 0; i < v1.size(); ++i)
    {
        ret[i] = v1[i] + v2[i];
    }
    return ret;
}

// функция для постороения множества тетта, которое используется для ген
   ерации
// последовательности оптимальных итерационных параметров
std::vector<int>
theta_set_construction(int m)
{
    if ((m & (m - 1)) != 0) throw "Argument m must be power of 2";
    if (m == 1) return std::vector<int>{0, 1};
    m = m / 2;
    std::vector<int> smaller_set = theta_set_construction(m);
    std::vector<int> ret(m * 2 + 1);
    for (int i = 1; i <= m; ++i)
    {
```

```
        ret[2 * i] = 4 * m - smaller_set[i];
        ret[2 * i - 1] = smaller_set[i];
    }
    return ret;
}

std::vector<double>
optim_iterative_parameters_set(int n)
{
    if ((n & (n - 1)) != 0) throw "Argument n must be power of 2";
    std::vector<double> ret(n + 1);
    auto theta = theta_set_construction(n);
    for (int i = 1; i <= n; ++i)
    {
        ret[i] = cos(M_PI * theta[i] / (n * 2));
    }
    return ret;
}

double
norm2(const std::vector<double> &v1)
{
    double ans = 0.0;
    for (const auto &item : v1)
    {
        ans += item * item;
    }
    return sqrt(ans);
}

// Функция для нахождения оценки собственных значений с помощью теоремы Гершгорина
std::vector<double>
eigenvalue_estimation(const std::vector<std::vector<double>> &A)
{
    double lambdaMax = 0.0;
    double lambdaMin = 0.0;
    for (int i = 0; i < A.size(); ++i)
    {
        double sum_abs_not_diag = 0.0;
        for (int j = 0; j < A[0].size(); ++j)
        {
            if (i != j) sum_abs_not_diag += std::fabs(A[i][j]);
        }
        if (i == 0) {
            lambdaMin = sum_abs_not_diag;
        } else if (lambdaMin > sum_abs_not_diag) lambdaMin = A[i][i] - sum_abs_not_diag;
        if (A[i][i] + sum_abs_not_diag > lambdaMax) lambdaMax = A[i][i] + sum_abs_not_diag;
    }
    return std::vector<double> {lambdaMin, lambdaMax};
}

// Решение системы линейных уравнений методом Чебышева
std::vector<double> chebyshevIteration(const std::vector<std::vector<double>>& A,
```

```
                                                const std::vector<double>& F,
                                                std::vector<float> &statX,
                                                std::vector<float> &statY,
                                                int maxIterations,
                                                std::vector<double> &x_true)
{
    if (A.size() != A[0].size()) throw "Matrix should be n*n!\n";
    if ((maxIterations & (maxIterations - 1)) != 0) throw "maxIterations
    argument should be power of 2";
    statX.resize(maxIterations), statY.resize(maxIterations);
    int n = A.size();
    std::vector<double> x(n, 0.0);
    std::vector<double> xPrev(n, 0.0);

    // Оценка для собственных значений с помощью теоремы Гершгорина
    std::vector<double> estim = eigenvalue_estimation(A);
    double lambdaMin = estim[0], lambdaMax = estim[1];

    double tau0 = 2.0 / (lambdaMax + lambdaMin);
    double ro = (lambdaMax - lambdaMin) / (lambdaMax + lambdaMin);

    std::vector<double> tau_parameters = optim_iterative_parameters_set(
    maxIterations);
    for (int k = 0; k < maxIterations; ++k) {
        double tau = tau0 / (1 - tau_parameters[k + 1] * ro);
        for (int i = 0; i < n; ++i) {
            double sum = 0.0;
            for (int j = 0; j < n; ++j) {
                sum += A[i][j] * xPrev[j];
            }
            x[i] = xPrev[i] + tau * (F[i] - sum);
        }

        statX[k] = k;
        statY[k] = norm2(F - A * x);
        xPrev = x;
    }

    return x;
}

int main() {
    std::string filename = "../SLAU_var_2.csv";
    std::vector<std::vector<double>> A = read_csv(filename);
    for (int i = 0; i < A.size(); i++) ++A[i][i];
    std::vector<double> x = generate_random_vect(A.size());
    std::vector<double> F;
    try { F = A * x; }
    catch (const char* str) { std::cerr << std::string(str) << std::endl
    ; }

    // LU-разложение
    std::vector<std::vector<double>> L;
    std::vector<std::vector<double>> U;
    std::vector<std::vector<double>> tmp_A = A;
    LU_decomposition(tmp_A, L, U);

```

```
    std::vector<double> x_computed = solve_system(L, U, F);
    double direct_method_error = norm2(x_computed - x);

    // Метод Чебышева
    int pow_of_two = 0;
    std::vector<float> statX, statY;
    std::vector<double> solution(x.size(), 0);
    int maxIterations = 0;
    while (norm2(solution - x) >= direct_method_error)
    {
        ++pow_of_two;
        statX.clear(), statY.clear();
        maxIterations = pow(2, pow_of_two);
        solution = chebyshevIteration(A, F, statX, statY, maxIterations,
    x);
    }
    statX.shrink_to_fit(), statY.shrink_to_fit();

    std::cout << "Оценка спектра матрицы с помощью теоремы Гершгорина(ми
нимальное, максимальное значения): " <<
    eigenvalue_estimation(A) << std::endl;
    std::cout << "Количество итераций метода Чебышева: " <<
maxIterations << std::endl;
    std::cout << "Погрешность решения прямым методом по второй норме: "
<< direct_method_error << std::endl;
    std::cout << "Погрешность решения методом Чебышева по второй норме:
" << norm2(solution - x) << std::endl;
    std::cout << "Относительная погрешность решения методом Чебышева по
второй норме: " << norm2(solution - x) / norm2(x) << std::endl;

    // Сохраним данные в csv файлы для отрисовки графика в Python
    std::ofstream fileX("statX.csv");
    for (const auto &value : statX)
    {
        fileX << value << ",";
    }
    std::ofstream fileY("statY.csv");
    for (const auto &value : statY)
    {
        fileY << value << ",";
    }

    return 0;
}
```

Листинг 1: second-task.cpp

Код для отрисовки графика зависимости среднеквадратической нормы погрешности от номера итерации метода Чебышева

```
import matplotlib.pyplot as plt
import numpy as np

path_to_fileX = "/home/ubuntu/code/uni/5sem/chmy/second_task/statX.csv"
path_to_fileY = "/home/ubuntu/code/uni/5sem/chmy/second_task/statY.csv"

dataX = np.genfromtxt(path_to_fileX, delimiter=",", dtype=np.float32)
dataY = np.genfromtxt(path_to_fileY, delimiter=",", dtype=np.float32)

assert dataX.shape[0] == dataY.shape[0], "Incorrect array sizes"
not_nan_mask = ~np.isnan(dataY)
dataY = dataY[not_nan_mask]
dataX = dataX[not_nan_mask]
plt.figure(figsize=(11, 7))
plt.grid()
plt.plot(dataX, dataY)
plt.title("График второй нормы как функции номера итерации")
plt.xlabel("Номер итерации")
plt.ylabel("Вторая норма погрешности")
plt.savefig("graph.png")
plt.show()
```

Листинг 2: graph.py

Полученные результаты

Результаты работы программы с матрицей из первого практического задания:

- Оценка спектра матрицы с помощью теоремы Гершгорина(минимальное, максимальное значения): 1 153.4
- Количество итераций метода Чебышева: 256
- Погрешность решения прямым методом по второй норме: 2.5601e-15
- Погрешность решения методом Чебышева по второй норме: 1.73998e-15
- Относительная погрешность решения методом Чебышева по второй норме: 3.02366e-16

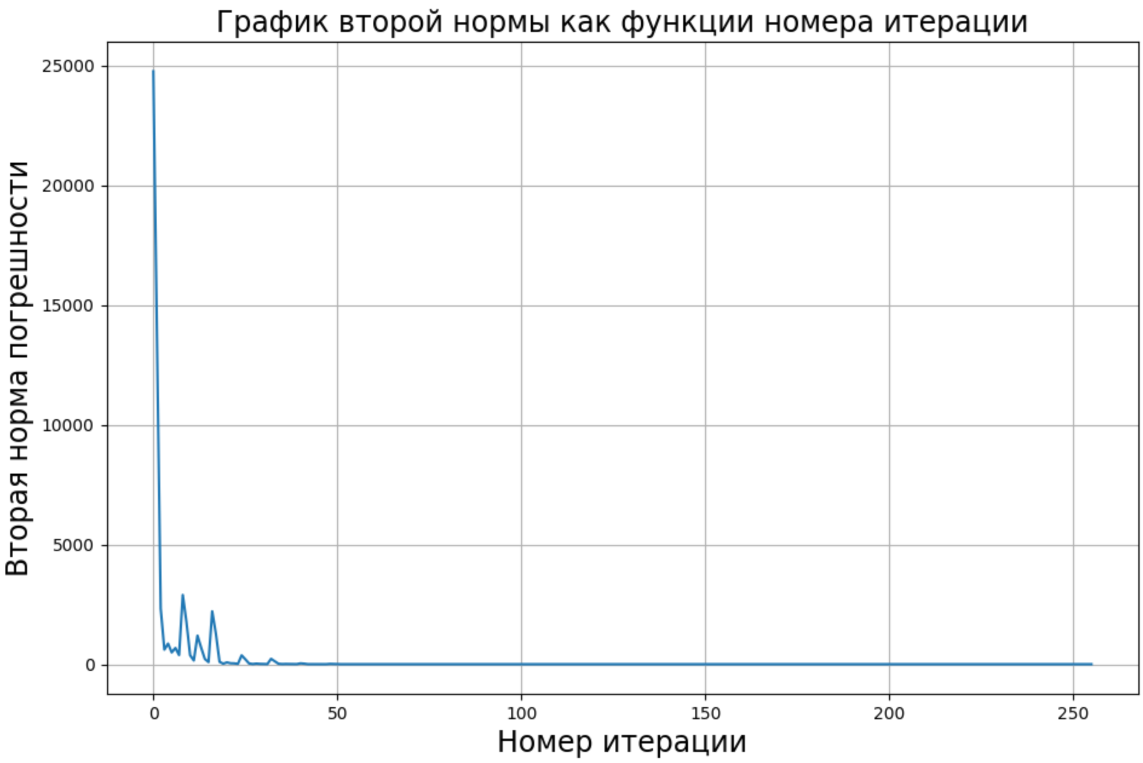

Рис. 1: График зависимости второй нормы погрешности от номера итерации