Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №1

Вариант 7

Выполнил:

Зинатулин А.В.

P32121

Преподаватель:

Наумова Н.А.

Санкт-Петербург, 2023 г.

Цель работы

• Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.
• Вывод вектора неизвестных: $x1, x2, \dots, xn$
• Вывод количества итераций, за которое было найдено решение.
• Вывод вектора погрешностей

Описание метода

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{a_{ij}}{a_{ii}} x_j^k , \qquad i = 1, 2, \dots, n$$

Достаточным условием сходимости итерационного процесса к решению системы при любом начальном векторе xi является выполнение условия преобладания диагональных элементов или доминирование диагонали

Код программы

https://github.com/uvuv-643/Computational_Mathematics

Листинг кода:

```
#include "Lab1.h"

void Lab1::runFromKeyboard() {

    Matrix<CFloat> a;
    CFloat eps;

    cerr << "> Enter matrix A below following instruction" << endl;
    cin >> a;
    size_t n = a.n;

    CVector<CFloat> b(n);
    cerr << "> Enter vector B below following instruction" << endl;
    cin >> b;
```

```
    cerr << "> Enter eps below" << endl;
    cin >> eps;

    IterMethodInformation answer = applyIterMethod(a, b, eps);
    outputResult(answer);

}

void Lab1::runFromKeyboardWithGeneration() {

    CSize n;
    cerr << "> Enter number of equations" << endl;
    cin >> n;

    Matrix<CFloat> a(n);
    a.setRandom();
    CFloat eps;
    CVector<CFloat> b(n);
    b.setRandom();

    cerr << "> Enter eps below" << endl;
    cin >> eps;

    cout << "Randomly generated matrix A: " << endl;
    cout << a << endl;
    cout << "Randomly generated matrix B: " << endl;
    cout << b << endl << endl;

    IterMethodInformation answer = applyIterMethod(a, b, eps);
    outputResult(answer);

}

void Lab1::runFromFile() {

    string file_path = std::getenv(ENV_PATH);
    if (file_path.empty()) {
        cerr << "Not found env. variable '" << ENV_PATH << "'" << endl;
        return;
    }
    ifstream fs;
    fs.open(std::getenv(ENV_PATH));
    if (fs.fail()) {
        cerr << "File not found. Make sure that it exists" << endl;
        return;
    }

    Matrix<CFloat> a;
    CFloat eps;
    fs >> a;
    size_t n = a.n;
    CVector<CFloat> b(n);
    fs >> b;
```

```
    if (!fs.eof()) {
        fs >> eps;
        IterMethodInformation answer = applyIterMethod(a, b, eps);
        outputResult(answer);
    } else {
        cerr << "Something wrong with file. Consider you are using correct
format of file" << endl;
    }
    fs.close();

}

void Lab1::outputResult(IterMethodInformation information) {
    Matrix<CFloat> initial_matrix = information.getInitialMatrix();
    if (information.getCountOfIterations() > 0) {
        CVector<CVector<CFloat>> iterations = information.getAnswers();
        CVector<CFloat> errors = information.getEps();
        cout << "Diagonal domination was reached" << endl;
        cout << "Updated matrix A: " << endl;
        cout << initial_matrix << endl;
        cout << "Number of iterations: " << information.getCountOfIterations()
<< endl;
        cout << "Iterations: " << endl;
        for (size_t k = 0; k < (size_t) information.getCountOfIterations();
k++) {
            cout << "k = " << k << " | ";
            cout << iterations[k] << " | ";
            cout << "eps = " << errors[k] << endl;
        }
    } else {
        cerr << "There is no diagonal dominance, unable to solve system" <<
endl;
        cerr << initial_matrix << endl;
    }
}

enum DiagonalDominanceStatus Lab1::checkOrApplyDiagonalDominance(Matrix<CFloat>
&a, CVector<CFloat> &b) {
    size_t n = a.n;
    set<size_t> good_indexes;
    map<size_t, size_t> maximums_by_indexes;
    for (size_t row = 0; row < n; row++) {
        maximums_by_indexes[row] = -1;
    }
    for (size_t row = 0; row < n; row++) {
        CFloat sum_in_row = 0;
        CFloat maximum_element_in_row = abs(a[row][0]);
        size_t maximum_element_in_row_index = 0;
        for (size_t col = 0; col < n; col++) {
            sum_in_row += abs(a[row][col]);
            if (abs(a[row][col]) > maximum_element_in_row) {
                maximum_element_in_row_index = col;
                maximum_element_in_row = abs(a[row][col]);
            }
```

```
        }
        maximums_by_indexes[row] = maximum_element_in_row_index;
        if (sum_in_row - maximum_element_in_row <= maximum_element_in_row) {
            good_indexes.insert(maximum_element_in_row_index);
        }
    }
    if (good_indexes.size() != n) {
        return DIAGONAL_DOMINANCE_IS_NOT_REACHABLE;
    } else {
        vector<pair<size_t, size_t>> have_to_be_swapped = {};
        for (size_t current_column = 0; current_column < n; current_column++) {
            if (current_column != maximums_by_indexes[current_column]) {
                bool found_pair_to_current_column = false;
                for (size_t found_column = current_column + 1; found_column <
n; found_column++) {
                    if (maximums_by_indexes[found_column] == current_column) {
                        have_to_be_swapped.emplace_back(current_column,
found_column);
                        maximums_by_indexes[found_column] =
maximums_by_indexes[current_column];
                        maximums_by_indexes[current_column] = current_column;
                        found_pair_to_current_column = true;
                        break;
                    }
                }
                if (!found_pair_to_current_column) {
                    return DIAGONAL_DOMINANCE_IS_NOT_REACHABLE;
                }
            }
        }
        for (pair<size_t, size_t> swapped_elements: have_to_be_swapped) {
            swap(b[swapped_elements.first], b[swapped_elements.second]);
            for (size_t col = 0; col < n; col++) {
                swap(a[swapped_elements.first][col],
a[swapped_elements.second][col]);
            }
        }
        if (have_to_be_swapped.empty()) {
            return DIAGONAL_DOMINANCE_INITIALLY_PRESENT;
        }
        return DIAGONAL_DOMINANCE_WAS_REACHED;
    }
}

IterMethodInformation &Lab1::applyIterMethod(Matrix<CFloat> &a, CVector<CFloat>
&b, CFloat eps) {
    enum DiagonalDominanceStatus dominance_status =
checkOrApplyDiagonalDominance(a, b);
    auto *answer = new IterMethodInformation(a);
    if (dominance_status >= 0) {
        size_t k = 1;
        size_t n = a.n;
        vector<vector<float>> dp(1, vector<float>(n));
        float current_eps = INITIAL_EPS;
```

```
        for (size_t i = 0; i < n; i++) {
            dp[0][i] = b[i] / a[i][i];
        }
        while (k < MAX_NUMBER_OF_ITERATIONS && current_eps >= eps) {
            vector<float> new_solution(n);
            dp.emplace_back(new_solution);
            for (size_t i = 0; i < n; i++) {
                dp[k][i] += b[i] / a[i][i];
                for (size_t j = 0; j < n; j++) {
                    if (i != j) {
                        dp[k][i] -= (a[i][j] * dp[k - 1][j] / a[i][i]);
                    }
                }
            }
            float maximum_difference = 0;
            for (size_t i = 0; i < n; i++) {
                maximum_difference = max(maximum_difference, abs(dp[k][i] -
dp[k - 1][i]));
            }
            current_eps = maximum_difference;
            k++;
        }
        for (size_t s = 0; s < dp.size(); s++) {
            float maximum_difference = 0;
            CVector<CFloat> current(n);
            for (size_t i = 0; i < n; i++) {
                current[i] = dp[s][i];
            }
            if (s != 0) {
                for (size_t i = 0; i < n; i++) {
                    maximum_difference = max(maximum_difference, abs(dp[s][i] -
dp[s - 1][i]));
                }
                answer->append(current, maximum_difference);
            } else {
                answer->append(current, INITIAL_EPS);
            }
        }
    }
    return *answer;
}
```

Пример работы программы

```
Randomly generated matrix A:
5
-36846.223      -8.2699738      -56.208164      35.772942       86.938576
3.8832743       -93.085579      5.9400387       -49230.184      -86.631554
37.354542       86.087296       2692.8777       30.783792       40.238117
52.43961        -45253.547      -34.353153      51.282097       -26.932266
96.510056       50.671165       -85.462822      76.941429       -6358.8594

Randomly generated matrix B:
```

```
     -73.692444          -8.2699738          -56.208164           35.772942
86.938576

Diagonal domination was reached
Updated matrix A:
5
-36846.223      -8.2699738      -56.208164      35.772942       86.938576
52.43961        -45253.547      -34.353153      51.282097       -26.932266
37.354542       86.087296       2692.8777       30.783792       40.238117
3.8832743       -93.085579      5.9400387       -49230.184      -86.631554
96.510056       50.671165       -85.462822      76.941429       -6358.8594

Number of iterations: 3
Iterations:
k = 0 |     0.0019999999   -0.00079050026     -0.020872898    0.00016798584
-0.013672039   | eps = 1000
k = 1 |     0.0019999223   -0.00076401036     -0.020672999    0.00019117883
-0.013365421   | eps = 0.0003066184
k = 2 |     0.0020003573   -0.00076431839      -0.02067869     0.0001906133
-0.013367616   | eps = 5.6903809e-06
```

Вывод

В ходе выполнения лабораторной работы я научился использовать итерационные методы для решения СЛАУ, повторил основы использования прямых методов решения СЛАУ, изучил условия существования решений СЛАУ, условие сходимости итерационных процессов