Первое практическое задание по курсу лекций "Численные методы линейной алгебры"

Михаил Протопопов

Ноябрь 2023

1 Постановка задания

Требуется решить систему линейных алгербраических уравнений

$$Ax = F$$

с квадратной невырожденной матрицей $A \in R^{n \times n}$. Элементы матрицы $a_{i,j}$ являются вещественными числами, расположенными на отрезке $[-1, 1]$. Матрица представлена в формате csv.

Для решения требуется использовать **метод отражений Хаусхолдера**.

Для успешного выполнения задания необходимо:

- случайным образом сгенерировать вектор-столбец решений $\bar{x}$ с равномерно распределёнными на отрезке $[-1, 1]$ компонентами $x_i, i = 1, 2, ..., n$.
- вычислить правую часть системы уравнений по формуле $\bar{F} = A\bar{x}$
- на языке программирования C++ написать программу, реализующую метод решения системы уравнений. Определить время, затраченное время на вычисление решения. Найти погрешность решения и вычислить максимум-норму погрешности.

2 Описание метода решения задачи

Пусть S и H - произволные вектор-столбцы, причём вектор H имеет единичную длину. Тогда найдётся такой вектор ω, что построенная по нему матрица отражения $U = E - 2\omega\omega^T$ переведёт вектор S в вектор, коллинеарный вектору H, то есть $US = bh$. Вектор Us - зеркальное отражение вектора S относительно гиперплоскости, ортогональной вектору ω. Для того, чтобы вектор S отразился в вектор Us, коллинеарный вектору H, вектор ω строится по правилу

$$\omega = \frac{1}{p}(s + bh) = \frac{1}{p}(s + sign(s, h)\,|s|\,h)$$

где |s| - длина вектора s

$b = sign(s, h)\,|s|$ - коэффициент пропорциональности, такой, чтобы выполнялось равенство Us=bh. Знак выбирается для того, чтобы вектор $s + bh$ не был равен нулю.

p - длина вектора $s + bh$

Будем преобразовывать расширенную матрицу системы по правилу $A^k = U^k A^{(k-1)}, k = 1, 2, ..., m-1$ с помощью умножения слева на последовательность матриц отражения $U^{(1)}, U^{(2)}, ..., U^{(m-1)}$. Для построения матрицы $U^{(1)}$ на первом шаге метода в качестве вектора S берётся первый столбец исходной расширенной матрицы $A^{(0)}$.

$$s = \begin{vmatrix} a_{1,1}^{(0)} \\ a_{2,1}^{(0)} \\ \vdots \\ a_{m,1}^{(0)} \end{vmatrix}$$

А в качестве вектора H - координатный вектор

$$h = \begin{vmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{vmatrix}$$

Так как H - координатный вектор, все координаты первого столбца расширенной матрицы, кроме координаты, соответствующей вектору H, то есть, первой, после выполнения первого шага метода будут равны нулю.

Пусть уже построена матрица $A^{(k-1)}$, у которой

$$a_{i,j}^{(k-1)} = 0, i > j, j = \overline{(1, k-1)}$$

Теперь

$$s = \begin{vmatrix} 0 \\ \vdots \\ 0 \\ a_{k,k}^{(k-1)} \\ a_{k+1,k}^{(k-1)} \\ \vdots \\ a_{m,k}^{(k-1)} \end{vmatrix} h = \begin{vmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{vmatrix}$$

где в векторе H единица стоит на k-ом месте. После выполнения k-го шага метода отражений получим матрицу $A^{(k)}$, у которой все элементы, стоящие ниже главной диагонали в первых k столбцах, будут равны нулю. Невозможность выполнения очередного шага связана только с равенством нулю вектора s, но это невозможно, если матрица A является невырожденной.

Далее значения всех неизвестных находятся аналогично обратному ходу метода Гаусса.

3 Листинг программы

В целях ускорения работы программы для хранения матриц используется std::vector<double> вместо std::vector<std::vector<d

Файл **householder.h**, содержащий заголовки всех функций, использующихся в методе отражений:

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

namespace {
    const double EPS = std::numeric_limits<double>::epsilon();
}

//res m*k A m*n B n*k
std::vector<double> MatrixMultiplication (const std::vector<double>& matrix1,
                    const std::vector<double>& matrix2, int m, int n, int k);

std::vector<double> MulMatrixByNumber(const std::vector<double>& matrix, double num);

std::vector<double> MatrixSubtraction(const std::vector<double>& matrix1,
                    const std::vector<double>& matrix2);

double VectorNorm(const std::vector<double>& vector);

double VectorNorm2(const std::vector<double>& vector);

int Sign(double a);

std::vector<double> HouseholderMethod(const std::vector<double>& matrix_, int size);
```

Функция MatrixMultiplication умножает матрицу $m \times n$ на матрицу $n \times k$

Функция MulMatrixByNumber умножает матрицу на число

Функция MatrixSubtraction вычисляет разность двух матриц

Функция VectorNorm вычисляет евклидову норму вектора

Функция VectorNorm2 вычисляет евклидову норму вектора в квадрате (данная функция написана для удобства и работает быстрее, чем использование функции VectorNorm и возведение её в квадрат)

Функция HouseholderMethod - непосредственно сам метод. Принимает расширенную матрицу размера $size \times (size + 1)$ и размер size.

Функция Sign возвращает -1, если число отрицательное, 1, если положительное и 0, если число равно нулю.

Файл **householder.cpp**, содержащий метод отражений:

```
#include "householder.h"

std::vector<double> MatrixMultiplication (const std::vector<double>& matrix1,
                     const std::vector<double>& matrix2, int m, int n, int k) {
    std::vector<double> res_matrix(m * k, 0);
    for (int i = 0; i < m; ++i) {
        for (int q = 0; q < n; ++q) {
            for (int j = 0; j < k; ++j) {
                res_matrix[i * k + j] += matrix1[i * n + q] * matrix2[q * k + j];
            }
        }
    }
    return res_matrix;
}

std::vector<double> MulMatrixByNumber(const std::vector<double>& matrix, double num) {
    int size = matrix.size();
    std::vector<double> res_matrix(size);
    for (int i = 0; i < size; ++i) {
        res_matrix[i] = matrix[i] * num;
    }
    return res_matrix;
}

std::vector<double> MatrixSubtraction(const std::vector<double>& matrix1,
                     const std::vector<double>& matrix2) {
    int size = matrix1.size();
    std::vector<double> res_matrix(size);
    for (int i = 0; i < size; ++i) {
        res_matrix[i] = matrix1[i] - matrix2[i];
    }
    return res_matrix;
}

double VectorNorm(const std::vector<double>& vector) {
    int n = vector.size();
    double res{};
    for (int i = 0; i < n; ++i) {
        res += vector[i] * vector[i];
    }
    return sqrt(res);
}

double VectorNorm2(const std::vector<double>& vector) {
    int n = vector.size();
    double res{};
    for (int i = 0; i < n; ++i) {
        res += vector[i] * vector[i];
    }
    return res;
}
```

```
int Sign(double a) {
    if (a > EPS) {
        return 1;
    } else if (a < -EPS) {
        return -1;
    }
    return 0;
}

std::vector<double> HouseholderMethod(const std::vector<double>& matrix_, int size) {
    std::vector<double> matrix(size * (size + 1));
    for (int i = 0; i < size * (size + 1); ++i) {
        matrix[i] = matrix_[i];
    }
    for (int i = 0; i < size - 1; ++i) {
        std::vector<double> vector_s(size);
        for (int j = 0; j < size + 1; ++j) {
            if (j < i) {
                vector_s[j] = 0;
            } else {
                vector_s[j] = matrix[j * (size + 1) + i];
            }
        }
        double b = Sign(matrix[i * (size + 1) + i]) * VectorNorm(vector_s);
        double p = sqrt(2 * (VectorNorm2(vector_s) + b * matrix[i * (size + 1) + i]));

        std::vector<double> vector_w(size);
        for(int j = 0; j < size; ++j) {
            if (j < i) {
                vector_w[j] = 0;
            } else if (j == i) {
                vector_w[j] = (matrix[i * (size + 1) + i] + b) / p;
            } else {
                vector_w[j] = (matrix[j * (size + 1) + i]) / p;
            }
        }
        std::vector<double> tmp_matrix = MatrixMultiplication(vector_w, matrix,
                                    1, size, size + 1);
        tmp_matrix = MatrixMultiplication(vector_w, tmp_matrix, size, 1, size + 1);
        tmp_matrix = MulMatrixByNumber(tmp_matrix, 2);
        matrix = MatrixSubtraction(matrix, tmp_matrix);
    }
    std::vector<double> res(size, 0);
    for (int i = size - 1; i >= 0; --i) {
        res[i] = matrix[i * (size + 1) + size];
        for (int j = i + 1; j < size; ++j) {
            res[i] -= res[j] * matrix[i * (size + 1) + j];
        }
        res[i] /= matrix[i * (size + 1) + i];
    }
    return res;
}
```

Файл **main.cpp** содержит в себе:

- выгрузку матрицы из файла *"SLAU_var_5.csv"*
- генерацию вектор-столбца $\bar{x}$ с равномерно распределёнными на отрезке $[-1, 1]$ компонентами $x_i, i = 1, 2, ..., n$

- вычисление правой чисти системы уравнений по формуле $\bar{F} = A\bar{x}$
- вызов метода отражений
- вычисление времени, затраченного на вычисление решения
- нахождения погрешности решения
- вычисление максимум-нормы погрешности

```
#include "householder.h"
#include <string>
#include <fstream>
#include <iterator>
#include <regex>
#include <sstream>
#include <ctime>

const std::regex comma(",");

std::vector<double> ReadDataFromCSV(std::string fname) {
    std::ifstream ifs(fname.c_str());
    std::vector<double> matrix;
    std::string line;
    while (ifs.good()) {
        std::getline(ifs, line);
        std::replace(line.begin(), line.end(), ',', ' ');
        std::stringstream ss(line);
        std::string current;
        while (ss >> current) {
            try {
                const double d = std::stod(current);
                matrix.push_back(d);
            } catch (const std::exception& e) {
                std::cerr << "Error" << std::endl;
            }
        }
    }
    ifs.close();
    return matrix;
}

std::vector<double> GenerateRandomDistributedVector(int size) {
    std::vector<double> x(size);
    for (int i = 0; i < size; ++i) {
        double random_num = (double)rand() / RAND_MAX; // generate random num from [0, 1]
        double scaled_num = random_num * 2 - 1; // scale this num to [-1, 1]
        x[i] = scaled_num;
    }
    return x;
}

std::vector<double> MatrixToExtendedMatrix(const std::vector<double>& matrix,
                     const std::vector<double>& vector, int size) {
    std::vector<double> ExtendedMatrix(size * (size + 1));
    for (int i = 0; i < size; ++i) {
        for(int j = 0; j < size + 1; ++j) {
            if (j == size) {
                ExtendedMatrix[i * (size + 1) + j] = vector[i];
            } else {
```

```
                ExtendedMatrix[i * (size + 1) + j] = matrix[i * size + j];
            }
        }
    }
    return ExtendedMatrix;
}

std::vector<double> MulMatrixOnVector(const std::vector<double>& matrix,
                    const std::vector<double>& vector, int size) {
    std::vector<double> res(size, 0);
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            res[i] += matrix[i * size + j] * vector[j];
        }
    }
    return res;
}

std::vector<double> FindMismatch(const std::vector<double>& vector1,
                    const std::vector<double>& vector2) {
    int size = vector1.size();
    std::vector<double> mismatch(size);
    for (int i = 0; i < size; ++i) {
        mismatch[i] = sqrt((vector1[i] - vector2[i]) * (vector1[i] - vector2[i]));
    }
    return mismatch;
}

double FindMaxNorm(const std::vector<double>& vector) {
    double res = abs(vector[0]);
    for (int i = 0; i < vector.size(); ++i) {
        if (abs(vector[i]) > res) {
            res = abs(vector[i]);
        }
    }
    return res;
}

int main()
{
    std::srand(std::time(nullptr));
    const std::string fname = "../SLAU_var_5.csv";
    std::vector<double> matrix{ReadDataFromCSV(fname)};

    int size = sqrt(matrix.size());

    std::vector<double> x{GenerateRandomDistributedVector(size)};

    std::vector<double> vector_f = MulMatrixOnVector(matrix, x, size);

    std::vector<double> extended_matrix{MatrixToExtendedMatrix(matrix, vector_f, size)};

    clock_t start = clock();
    std::vector<double> res = HouseholderMethod(extended_matrix, size);
    clock_t end = clock();
    double seconds = (double)(end - start) / CLOCKS_PER_SEC;

    std::vector<double> mismatch{FindMismatch(res, x)};
```

```
    double MaxNorm = FindMaxNorm(mismatch);

    std::cout << "TIME - " << seconds << "s" <<std::endl;
    std::cout << "Max-Norm - " << MaxNorm << std::endl;
}
```

4 Полученные результаты

Время работы программы - 0.065807s

Максимум-норма погрешности - 2.88658e-15

Таким образом, можно сделать вывод, что программа работает достаточно быстро на матрице порядка 100. Также можно сказать о том, что алгоритм достаточно точный.