Курсовая работа учебного года 2023-2024 по курсу «Численные методы»

Выполнил студент группы М8О-408Б-20 Шандрюк П.Н.
Преподаватель: Пивоваров Д. Е.

Вариант курсовой работы: 1

Вариант 1

Решение систем линейных алгебраических уравнений с симметричными разреженными матрицами большой размерности. Метод сопряженных градиентов.

Метод решения

В данной работе мною было реализовано решение систем линейных алгебраических уравнений вида **Ax=b** с симметричными разреженными матрицами большой размерности методом сопряжённых градиентов. Рассмотрим сам алгоритм.

1. **Инициализация**: Задаются начальные значения, такие как матрица системы **A**, вектор правой части **b**, начальное приближение **x0** (если не задано, то используется нулевой вектор), и точность **eps** для определения критерия останова.
2. **Инициализация переменных**: Задаются переменные **x**, **r**, и **p**:
    - **x**: текущее приближение к решению.
    - **r**: вектор остатков, исходно равный **b - Ax**.
    - **p**: направление движения, исходно равное вектору остатков **r**.
3. **Итерационный процесс**: Выполняется итерационный процесс, включающий следующие шаги:
    - Вычисление **Ap**, где **A** - матрица системы, **p** - текущее направление.
    - Вычисление нормы остатков **r**.
    - Вычисление коэффициента **alpha** по формуле **alpha = norm(r)^2 / (p * Ap)**.
    - Обновление **x** по формуле **x = x + alpha * p**.
    - Обновление вектора остатков **r** по формуле **r = r - alpha * Ap**.
    - Вычисление коэффициента **beta** по формуле **beta = norm(r_new)^2 / norm(r)^2**.
    - Обновление направления **p** по формуле **p = r_new + beta * p**.

4. **Проверка условия останова**: После каждой итерации проверяется условие останова, основанное на норме вектора остатков. Если **norm(r) < eps**, алгоритм завершается.

О программе

В курсовой работе я предусмотрел два файла – один для непосредственного решения (написан на C++ - mainc.cpp), второй для генерации тестовых данных (написан на Python).

Листинг

generate_matrix.py

```
import argparse
import numpy as np
from scipy.sparse import csc_matrix, rand
from random import randint


class SparseMatrix:
    def __init__(self, rows, cols):
        self.rows = rows
        self.cols = cols
        self.elements = []

    def add_element(self, row, col, value):
        self.elements.append((row, col, value))

    def set_b(self, b_values):
        self.b = b_values


class MatrixGenerator:
    def generate(self, shape):
        matrix = rand(shape, shape, density=0.4,
random_state=randint(112, 154))
        matrix = matrix.toarray()
        for i in range(shape):
            for j in range(shape):
```

```
                matrix[j][i] = matrix[i][j]
        matrix = csc_matrix(matrix)
        return matrix


def save_matrix_to_file(matrix, b, output_file):
    with open(output_file, "w") as f:
        f.write(f"{matrix.shape[0]}\n")
        for row in matrix.toarray().round(3):
            f.write(" ".join(map(str, row)) + "\n")
        f.write(" ".join(map(str, b)) + "\n")


def main():
    parser = argparse.ArgumentParser()

    output_file = "matrix.txt"
    shape = int(input())
    if shape < 3:
        exit()

    matrix_generator = MatrixGenerator()
    matrix = matrix_generator.generate(shape)

    sparse_matrix = SparseMatrix(matrix.shape[0],
matrix.shape[1])
    for i in range(shape):
        for j in range(shape):
            sparse_matrix.add_element(i, j, matrix[i,
j])

    d = np.random.randint(5, 53, shape)
    sparse_matrix.set_b(d)

    save_matrix_to_file(matrix, d, output_file)


if __name__ == "__main__":
    main()
```

main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <chrono>

class SparseMatrix {
public:
    SparseMatrix(int rows, int cols) : rows(rows),
cols(cols) {}

    void addValue(int row, int col, double value) {
        elements.emplace_back(row, col, value);
    }

    void setB(const std::vector<double>& b_values) {
        b = b_values;
    }

    int getRows() const {
        return rows;
    }

    int getCols() const {
        return cols;
    }

    const std::vector<std::tuple<int, int, double>>&
getElements() const {
        return elements;
    }

    const std::vector<double>& getB() const {
        return b;
    }

private:
    int rows;
```

```
    int cols;
    std::vector<std::tuple<int, int, double>> elements;
    std::vector<double> b;
};

class Solver {
public:
    Solver(const SparseMatrix& matrix, const
std::vector<double>& b,
           const std::string& output_file, const
std::vector<double>& x0 = {},
           double eps = 1e-5)
        : output(output_file.empty() ? "res_default" :
output_file),
          matrix(matrix), b(b), eps(eps),
shape(matrix.getRows()), x0(x0), k(0) {}

    std::vector<double> solve(int max_iter = 100000) {
        std::vector<double> x_new, r_new, p_new;
        std::vector<double> x0 = this->x0;
        std::vector<double> r0 = vectorSubtract(b,
matrixMultiply(matrix, x0));
        std::vector<double> p0 = r0;

        for (int iter = 0; iter < max_iter; ++iter) {
            std::vector<double> temp =
matrixMultiply(matrix, p0);
            double norm_0 = vectorDot(r0, r0);
            double alpha_i = norm_0 / vectorDot(temp,
p0);
            x_new = vectorAdd(x0, vectorMultiply(p0,
alpha_i));
            r_new = vectorSubtract(r0,
vectorMultiply(temp, alpha_i));
            double norm_new = vectorDot(r_new, r_new);
            double beta_i = norm_new / norm_0;
            p_new = vectorAdd(r_new, vectorMultiply(p0,
beta_i));

            r0 = r_new;
```

```
            p0 = p_new;
            x0 = x_new;

            k += 1;

            if (vectorNorm(r_new) < eps) {
                break;
            }
        }
        return x0;
    }

    void solveAndPrint() {
        auto start =
std::chrono::high_resolution_clock::now();
        std::vector<double> x = solve();
        auto end =
std::chrono::high_resolution_clock::now();
        auto start2 =
std::chrono::high_resolution_clock::now();
        std::vector<double> x2 =
solveLinearSystem(matrix, b);
        auto end2 =
std::chrono::high_resolution_clock::now();

        std::cout << "Custom solution:\n";
        printVector(x);
        std::cout << "eps=" << eps << " shape=" << shape
<< " iterations=" << k
                  << " mean=" << vectorMean(x) << "
time=" << std::chrono::duration<double>(end -
start).count() << " seconds\n";

        std::cout << "NumPy solution:\n";
        printVector(x2);
        std::cout << "mean=" << vectorMean(x2) << "
time=" << std::chrono::duration<double>(end2 -
start2).count() << " seconds\n";
    }
```

```
private:
    std::string output;
    SparseMatrix matrix;
    std::vector<double> b;
    double eps;
    int shape;
    std::vector<double> x0;
    int k;

    static std::vector<double> vectorAdd(const
std::vector<double>& a, const std::vector<double>& b) {
        std::vector<double> result(a.size());
        for (size_t i = 0; i < a.size(); ++i) {
            result[i] = a[i] + b[i];
        }
        return result;
    }

    static std::vector<double> vectorSubtract(const
std::vector<double>& a, const std::vector<double>& b) {
        std::vector<double> result(a.size());
        for (size_t i = 0; i < a.size(); ++i) {
            result[i] = a[i] - b[i];
        }
        return result;
    }

    static std::vector<double> vectorMultiply(const
std::vector<double>& a, double scalar) {
        std::vector<double> result(a.size());
        for (size_t i = 0; i < a.size(); ++i) {
            result[i] = a[i] * scalar;
        }
        return result;
    }

    static double vectorDot(const std::vector<double>&
a, const std::vector<double>& b) {
        double result = 0.0;
        for (size_t i = 0; i < a.size(); ++i) {
```

```
            result += a[i] * b[i];
        }
        return result;
    }

    static double vectorNorm(const std::vector<double>&
v) {
        double result = 0.0;
        for (double val : v) {
            result += val * val;
        }
        return std::sqrt(result);
    }

    static double vectorMean(const std::vector<double>&
v) {
        double sum = 0.0;
        for (double val : v) {
            sum += val;
        }
        return sum / v.size();
    }

    static void printVector(const std::vector<double>&
v) {
        for (double val : v) {
            std::cout << val << " ";
        }
        std::cout << "\n";
    }

    static std::vector<double> solveLinearSystem(const
SparseMatrix& matrix, const std::vector<double>& b) {
        // Simple Gaussian elimination method for
solving Ax = b.
        int n = matrix.getRows();
        std::vector<std::vector<double>>
augmentedMatrix(n, std::vector<double>(n + 1));

        for (const auto& element : matrix.getElements())
```

```
{
            int row = std::get<0>(element);
            int col = std::get<1>(element);
            double value = std::get<2>(element);
            augmentedMatrix[row][col] = value;
        }

        for (int i = 0; i < n; ++i) {
            augmentedMatrix[i][n] = b[i];
        }

        for (int i = 0; i < n; ++i) {
            // Pivot for the current column
            double pivot = augmentedMatrix[i][i];

            // Make the diagonal element 1
            for (int j = i + 1; j <= n; ++j) {
                augmentedMatrix[i][j] /= pivot;
            }

            for (int k = 0; k < n; ++k) {
                if (k != i) {
                    double factor =
augmentedMatrix[k][i];
                    for (int j = i; j <= n; ++j) {
                        augmentedMatrix[k][j] -= factor
* augmentedMatrix[i][j];
                    }
                }
            }
        }

        std::vector<double> solution(n);
        for (int i = 0; i < n; ++i) {
            solution[i] = augmentedMatrix[i][n];
        }

        return solution;
    }
```

```
    static std::vector<double> matrixMultiply(const
SparseMatrix& matrix, const std::vector<double>& v) {
        std::vector<double> result(matrix.getRows(),
0.0);
        for (const auto& element : matrix.getElements())
{
            int row = std::get<0>(element);
            int col = std::get<1>(element);
            double value = std::get<2>(element);
            result[row] += value * v[col];
        }
        return result;
    }
};

SparseMatrix readMatrix(const std::string& filename) {
    std::ifstream file(filename);
    int shape;
    file >> shape;

    SparseMatrix matrix(shape, shape);

    for (int i = 0; i < shape; ++i) {
        for (int j = 0; j < shape; ++j) {
            double value;
            file >> value;
            if (value != 0.0) {
                matrix.addValue(i, j, value);
            }
        }
    }

    std::vector<double> b(shape);
    for (int i = 0; i < shape; ++i) {
        file >> b[i];
    }

    matrix.setB(b);

    return matrix;
```

```
}

int main() {
    std::string input_file = "matrix.txt";
    std::string output_file = "output_file.txt";
    double eps = 0.01;

    SparseMatrix matrix = readMatrix(input_file);
    std::vector<double> b = matrix.getB();

    Solver solver(matrix, b, output_file, {}, eps);
    solver.solveAndPrint();

    return 0;
}
```

Результаты

Входные данные (размер матрицы, матрица, вектор решений):

10
0.0 0.0 0.673 0.0 0.21 1.0 0.0 0.0 0.539 0.0
0.0 0.085 0.0 0.801 0.0 0.0 0.0 0.0 0.0 0.0
0.673 0.0 0.16 0.553 0.222 0.0 0.0 0.0 0.993 0.9
0.0 0.801 0.553 0.0 0.695 0.72 0.0 0.637 0.0 0.0
0.21 0.0 0.222 0.695 0.015 0.0 0.087 0.0 0.0 0.0
1.0 0.0 0.0 0.72 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.087 0.0 0.0 0.522 0.063 0.0
0.0 0.0 0.0 0.637 0.0 0.0 0.522 0.0 0.0 0.734
0.539 0.0 0.993 0.0 0.0 0.0 0.063 0.0 0.074 0.216
0.0 0.0 0.9 0.0 0.0 0.0 0.0 0.734 0.216 0.041
36 21 19 24 41 30 27 8 28 48

Результат работы программы:

0.4867 -139.21825 36.03465 40.9907 160.46605 -17.3655 23.0181 26.00615

-8.50406 -41.04431
 mean=8.087021967322894, time=0.002 seconds

Вывод

В процессе выполнения данного курсового проекта мною был изучен метод сопряженных градиентов для сильно разреженных матриц. Я в очередной раз попрактиковался в реализации численных методов на языке программирования C++ и получил дополнительные навыки.