Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО

Лабораторная работа №1

по дисциплине

Линейная алгебра и анализ данных

Семестр I

Выполнили:
студенты
Смирнов Олег Николаевич
гр. J3110
ИСУ 467510
Муртазалиев Матвей Асланович
гр. J3110
ИСУ 466797

Отчет сдан:
99.99.9999

Санкт-Петербург
2024

Введение

Целью данной лабораторной работы является написание кода для выполнения различных операций с матрицами.

Задачи:

1: Реализовать хранение матриц в разреженно-строчном виде и функции для подсчёта следа матрицы, вывода элемента по индексу.

2: Реализовать функции для выполнения различных операций над матрицами: сложение и умножение матриц, умножение матрицы на число.

3: Реализовать функцию, которая считает определитель матрицы и выясняет, есть ли обратная матрица к заданной.

Ход выполнения задач

Задача 1

Реализовать хранение матриц в разреженно-строчном виде и функции для подсчёта следа матрицы, вывода элемента по индексу.

Теоретическая справка

Хранение матрицы в разреженно-строчном формате (CSR, Compressed Sparse Row) осуществляется с использованием трёх массивов:

1. `val` — ненулевые элементы матрицы,
2. `col` — номера столбцов, в которых расположены ненулевые элементы,
3. `row` — массив, определяющий границы строк, т.е. где начинается каждая строка в массиве `val`.

Определение следа матрицы

След может быть вычислен только для квадратных матриц по формуле:

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii},$$

где a_{ii} — элементы на главной диагонали, а n — размер матрицы.

Решение:

```cpp
class Matrix {
    private:
        int n, m; // Размеры матрицы
        vector<double> val; // Ненулевые элементы
        vector<int> col; // Индексы столбцов для ненулевых элементов
        vector<int> row; // Сумма ненулевых элементов по строкам и предыдущим

    public:
        // Конструктор с инициализацией размеров
        Matrix(int r, int c) : n(r), m(c) {
            row.resize(n + 2, 0); // Начальное заполнение row нулями
        }

        // Метод для добавления элемента в матрицу
        void add(int i, int j, double elem) {
            if (i <= 0  i > n  j <= 0  j > m) {
                throw out_of_range("Индекс выходит за пределы матрицы");
            }

            if (elem == 0) // Пропускаем нулевые элементы
                return;

            // Добавляем элемент в val и col
            val.push_back(elem);
            col.push_back(j);

            // Увеличиваем row[i + 1] и последующие элементы
            for (int k = i + 1; k <= n + 1; k++) {
                row[k]++;
            }
        }

        // Метод для подсчета следа матрицы
        double trace() {
            if (n != m)
                throw logic_error("Не квадратная матрица");
            double trace = 0;
            for (int i = 1; i <= n; i++) {
                trace += (*this)[i][i];
            }
            return trace;
        }
```

Задача 2

Реализовать сложение матриц, умножение матрицы на скаляр, умножение матриц.

Теоретическая справка

Сложение матриц. Две матрицы одинакового размера A и B складываются покомпонентно:

$$c_{ij} = a_{ij} + b_{ij}.$$

Умножение матрицы на скаляр. Матрица A умножается на число α следующим образом:

$$b_{ij} = \alpha \cdot a_{ij}.$$

Умножение матриц. Произведение матриц A $(n \times m)$ и B $(m \times p)$ определяется как:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}.$$

Решение:

```
// Перегрузка оператора [] для доступа к строке
RowProxy operator[](int rowIndex) const {
    if (rowIndex <= 0  rowIndex > n) {
        throw out_of_range("Индекс строки выходит за пределы матрицы");
    }
    return RowProxy(*this, rowIndex);
}

Matrix operator+(Matrix &other) const {
    if (this->n != other.n  this->m != other.m)
        throw logic_error("Разные размеры");

    Matrix res(n, m);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            res.add(i, j, (*this)[i][j] + other[i][j]);
        }
    }
    return res;
}

Matrix operator*(double scalar) const {
    Matrix res(n, m);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            res.add(i, j, (*this)[i][j] * scalar);
        }
    }
    return res;
}

Matrix operator*(Matrix &other) const {
    if (this->m != other.n)
        throw logic_error("Неподходящие размеры");

    Matrix res(this->n, other.m);
    for (int row = 1; row <= n; row++) {
        for (int col = 1; col <= other.m; col++) {
            double sum = 0;
            for (int k = 1; k <= m; k++) {
                sum += (*this)[row][k] * other[k][col];
            }
            res.add(row, col, sum);
        }
    }
    return res;
}
```

Задача 3

Реализовать функцию для подсчета определителя и проверки обратимости матрицы.

Теоретическая справка

Определитель. Определитель матрицы вычисляется рекурсивно по формуле:

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \cdot \det(\mathrm{Minor}(A, 1, j)),$$

где $\mathrm{Minor}(A, 1, j)$ — минор, полученный удалением первой строки и j-го столбца.

Обратимость. Если определитель матрицы $A \neq 0$, то существует обратная матрица.

Решение:

```
// Рекурсивная функция для вычисления определителя
double determinant() const {
    if (n != m)
        throw logic_error("Не квадратная матрица");
    return determinantRecursive(*this, n);
}

// Вспомогательная функция для рекурсивного вычисления определителя
double determinantRecursive(const Matrix &matrix, int size) const {
    if (size == 1) { //                           1x1
        return matrix[1][1];
    }
    if (size == 2) { //                           2x2
        return matrix[1][1] * matrix[2][2] - matrix[1][2] * matrix[2][1];
    }

    double det = 0;
    for (int col = 1; col <= size; col++) {
        // Создаем подматрицу (минор)
        Matrix minorMatrix(size - 1, size - 1);
        for (int i = 2; i <= size; i++) {
            for (int j = 1; j <= size; j++) {
                if (j == col) continue; // Пропускаем столбец col
                minorMatrix.add(i - 1, j < col ? j : j - 1, matrix[i][j]);
            }
        }

        // Рекурсивно вычисляем определитель минора
        det += (col % 2 == 1 ? 1 : -1) * matrix[1][col] *
           determinantRecursive(minorMatrix, size - 1);
    }

    return det;
}
```

Тесты

В некоторых тестах используется округление из-за погрешности вещественных чисел.

```
TEST(MatrixTest, Test0) {
    int n = 2, m = 2;
    vector<vector<double>> inp = {
        {1, 2},
        {3, 4}
    };
    Matrix matrix = add_matrix(n, m, inp);
    EXPECT_TRUE(5 == matrix.trace());
    EXPECT_TRUE(-2 == matrix.determinant());
}

TEST(MatrixTest, Test1) {
    int n = 4, m = 4;
    vector<vector<double>> inp = {
        {16, 2, 3, 16},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {16, 14, 15, 16}
    };
    Matrix matrix = add_matrix(n, m, inp);
    EXPECT_TRUE(49 == matrix.trace());
    EXPECT_TRUE(144 == matrix.determinant());
}

TEST(MatrixTest, Test2) {
    int n = 3, m = 4;
    vector<vector<double>> inp = {
        {16, 2, 3, 16},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };
    Matrix matrix = add_matrix(n, m, inp);
    EXPECT_ANY_THROW(matrix.trace());
    EXPECT_ANY_THROW(matrix.determinant());
}

TEST(MatrixTest, Test3) {
    int n = 2, m = 2;
    vector<vector<double>> inp = {
        {1, 2},
        {3, 4}
    };
    Matrix matrix = add_matrix(n, m, inp);
    vector<vector<double>> inp_ans = {
        {2, 4},
        {6, 8}
    };
    Matrix ans = add_matrix(n, m, inp_ans);
    EXPECT_TRUE(matrix + matrix == ans);
}

TEST(MatrixTest, Test4) {
    int n = 2, m = 2;
    vector<vector<double>> inp = {
        {1, 2},
        {3, 4}
    };
```

```
    Matrix matrix = add_matrix(n, m, inp);
    vector<vector<double>> inp_ans = {
        {3, 6},
        {9, 12}
    };
    Matrix ans = add_matrix(n, m, inp_ans);
    EXPECT_TRUE(matrix * 3 == ans);
}

TEST(MatrixTest, Test5) {
    int n = 2, m = 2;
    vector<vector<double>> inp = {
        {1, 2},
        {3, 4}
    };
    Matrix matrix = add_matrix(n, m, inp);
    vector<vector<double>> inp_ans = {
        {7, 10},
        {15, 22}
    };
    Matrix ans = add_matrix(n, m, inp_ans);
    EXPECT_EQ(matrix * matrix == ans);
}

TEST(MatrixTest, Test6) {
    int n = 2, m = 2;
    vector<vector<double>> inp = {
        {1.1, 2.12},
        {3.45, 4.21}
    };
    Matrix matrix = add_matrix(n, m, inp);
    EXPECT_NEAR(-2.683, matrix.determinant(), 1e-5);
}
```

Все тесты прошли.

Вывод

В ходе выполнения лабораторной работы были изучены и реализованы алгоритмы для работы с разреженными матрицами. Успешно решены задачи по хранению, выполнению операций и проверке свойств матриц. Результаты работы на тестовых данных подтверждают правильность реализованных функций.