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
Тарянник Антон Александрович
гр. J3112
ИСУ 467677
Мавров Артём Николаевич
гр. J3112
ИСУ 466574
Вебер Кирилл Владимирович
гр. J3112
ИСУ 465380

Отчет сдан: 14.12.2024

Санкт-Петербург
2024

Цели

Цели данной лабораторной работы заключаются в приобретении практических навыков работы с матрицами, хранимыми в разреженном виде. Изучении преимущества разреженных матриц перед плотными.

Задачи

Для того, чтобы научиться работать с разреженными матрицами были поставлены и выполнены следующие задачи:

- Реализовать хранение матриц в разреженно-строчном виде. На выходе должен быть класс с методами подсчёта следа матрицы и вывода элемента по строке и столбцу.
- Реализовать функции сложения и произведения матриц, а также функцию умножения матрицы на скаляр.
- Реализовать функцию, которая считает определитель заданной матрицы, а также определяет, существует ли обратная матрица к данной.
- Написать тестирующую программу, чтобы убедиться в правильности реализованных функций.

Ход решения

.

Реализация первой задачи

Чтобы решить поставленные задачи, был выбран язык программирования C++, так как он обладает более высокой скоростью по сравнению с Python. Для реализации хранения матрицы в разреженном виде в классе были созданы следующие поля:

```
    int n, m;
    long double scalar = 1.0;
    vector<long double> values;
    vector<int> col;
    vector<int> rowpointer;
    string path;
```

Где:

- n и m - количество строк и столбцов матрицы.
- Scalar - коэффициент, на который домножаются все элементы матрицы (по умолчанию равен единице).

- values - вектор непустых элементов матрицы, col - вектор, который хранит номер столбца каждого ненулевого элемента в values, rowpointer - вектор размерности n + 1, хранит для каждой строки количество ненулевых элементов в пред идущих строках, причём для первой строки это количество считается равным нулю, т.е. rowpointer[0] = 0

Чтобы перейти к решению первой задачи лабораторной работы необходимо дать определение следа матрицы: **следом матрицы** называется сумма элементов на главной диагонали матрицы. Теперь, чтобы посчитать след матрицы, нужно в каждой строке i найти такой элемент j в values, что col[j] = i. Для этого переберём все $i \in [1, n]$ и $j \in [rowpointer[i-1], rowpointer[i])$. Тогда values[j] соответсвует элементу в строке i - 1. Стоит заметить, что все нулевые элементы будут проигнорирваны данным алгоритмом, поэтому уже на данном этапе разреженные матрицы могут ускорить время выполнения программы и сократить используемую память. Конечная сумма также должна умножаться на скаляр - некоторый коэффициент, на который умножается матрица.

Код метода подсчёта следа матрицы:

```
long double get_trace() {
        long double trace = 0;
        for (int i = 1; i < rowpointer.size(); i++)
        {
            for(int j = rowpointer[i - 1]; j < rowpointer[i]; j++)
            {
                if(col[j] == i - 1)
                {
                    trace += values[j];
                }
            }
        }

        return trace * scalar;
    }
```

Реализация метода поиска элемента по номеру строки и столбца не менее тривиальна: по аналогии с методом подсчёта следа матрицы переберём все элементы из искомой строки и найдём искомый элемент в values.

Код метода поиска элемента:

```
long double get_element(int i, int j)
    {
        j--;
        if(col[rowpointer[i - 1]] > j || col[rowpointer[i] - 1] < j)
        {
            return 0;
        }
        for(int value_index = rowpointer[i - 1]; value_index < rowpointer[i];
        value_index++)
        {
            if(col[value_index] == j)
            {
                return values[value_index];
            }
        }
        return 0;
    }
```

Реализация второй задачи

Для реализации функции сложения матриц был реализован следующий алгоритм: для каждой строки ненулевые элементы записываются в словарь string_values с помощью функции get_matrix_string_sum. В string_values ключом служит номер столбца элемента, а значением - значение элемента. Далее, в функции merge_string_to происходит слияние string_values в векторы new_values и new_col, а также обновляется состояние new_rowpointer. После обхода всех элементов в values матриц, которые нужно суммировать, функцией создаётся и возвращается новая матрица C, которая является результатом суммы матриц A и B. С кодом вспомогательных функций вы можете ознакомиться на GitHub

Код функции sum_of_matrix

```
Matrix sum_of_matrix(Matrix& A, Matrix& B) {
    if (A.n != B.n || A.m != B.m) {
        cout << "Mistake";
        return A;
    }
    int n = A.n;
    int m = A.m;
    vector<long double> new_values;
    vector<int> new_col;
    vector<int> new_rowpointer(n + 1, 0);
    for(int rowpointer_index = 1; rowpointer_index < n + 1; rowpointer_index++)
    {
        map<int, long double> string_values;
        get_matrix_string_sum(string_values, A, rowpointer_index);
        get_matrix_string_sum(string_values, B, rowpointer_index);
        merge_string_to_new_vectors(new_values, new_col, string_values);
        new_rowpointer[rowpointer_index] = new_rowpointer[rowpointer_index - 1]
        + string_values.size();
    }
    Matrix C = Matrix(n, m, new_values, new_col, new_rowpointer);
    return C;
}
```

Так как умножение матрицы на скаляр предполагает умножение каждого элемента матрицы на этот самый скаляр, его можно вынести как отдельное поле в классе. Это поле назовём scalar и при умножении матрицы на скаляр будем умножать не всю матрицу, а поле scalar. Тогда время такой операции сократится с O(values.size()) до 0(1), что работает гораздо быстрее. С помощью данной оптимизации функция умножения на скаляр становится тривиальной:

Функция умножения на скаляр

```
void multiply_scalar(int scalar, Matrix& A) {
    A.scalar *= scalar;
}
```

Для реализации функции умножения матриц проделаем процедуру, аналогичную сложению матриц: получим два словаря, которые содержат все ненулевые элементы матриц A и B, а затем для каждого элемента новой матрицы вычислим сумму произведений элементов вектора-строки и вектора-столбца с помощью полученных словарей.

Функции `multiply_vectors` и `multiply_matrix`:

```
Matrix multiply_matrix(Matrix& A, Matrix& B) {
    if (A.m != B.n) {
        cout << "Mistake \n";
        return A;
    }
    int new_n = A.n;
    int new_m = B.m;
    vector<long double> new_values;
    vector<int> new_col;
    vector<int> new_rowpointer(new_n + 1, 0);
    // column values of matrix B
    map<int, map<int, long double>> column_values;
    // string values of matrix A
    map<int, map<int, long double>> string_values;
    for(int i = 1; i < A.rowpointer.size(); i++)
    {
        get_matrix_string_sum(string_values[i - 1], A, i);
    }
    for(int j = 0; j < B.m; j++)
    {
        get_matrix_column_sum(column_values[j], B, j);
    }
    for(int i = 0; i < new_n; i++)
    {
        int cnt_of_not_zero_elements = 0;
        for(int j = 0; j < new_m; j++)
        {
            long double new_value = 0;
            for(pair<int, long double> string_value : string_values[i])
            {
                long double value = string_value.second;
                long double index = string_value.first;
                if(column_values[j].count(index) == 0)
                {
                    continue;
                }
                value *= column_values[j][index];
                new_value += value;
            }
            new_value = new_value * A.scalar * B.scalar;

            if(new_value != 0)
            {
                cnt_of_not_zero_elements++;
                new_values.push_back(new_value);
                new_col.push_back(j);
            }
        }
        new_rowpointer[i + 1] = new_rowpointer[i] + cnt_of_not_zero_elements;
    }
    Matrix C = Matrix(new_n, new_m, new_values, new_col, new_rowpointer);
    return C;
}
```

Реализация третьей задачи

Определитель матрицы можно вычислить двумя известными способами: с помощью метода алгебраических дополнений и метода Гаусса. В данной работе будет приведено решение методом Гаусса, так как метод алгебраических дополнений работает за O(n!), в то время, как метод Гаусса за $O(n^3)$. Кратко опишем метод Гаусса: Для начала находится наибольший элемент строки для снижения погрешности. Далее при необходимости строки меняются местами и вычитаются (при вычитании можно умножать строку на скаляр). Алгоритм повторяется до тех пор пока матрица не придёт к треугольному виду, после чего вычисляется определитель как произведение элементов на главной диагонали. При этом, матрица называется невырожденной, если её определитель не равен нулю. У невырожденных матриц есть важное свойство: у них всегда есть обратная матрица, поэтому зная определитель, можно определить, существует обратная матрица к данной, или нет.

Код функции подсчёта определителя

:

```
pair<long double, string> get_determinant(Matrix matrix) {
    int n = matrix.n;
    vector<long double> values = matrix.values;
    vector<int> col = matrix.col;
    vector<int> rowpointer = matrix.rowpointer;

    vector<int> order(n + 1);
    vector<long double> nw_values;
    vector<int> nw_col;
    vector<int> nw_rowpointer;
    long double det = 1.0;
    long double factor;
    int pivot_row;
    long double max_value;

    for (int i = 0; i < n + 1; i++) {
        order[i] = i;
    }

    for (int i = 0; i < n; i++) {
        max_value = 0.0;

        for (int j = i; j < n; j++) {
            for (int k = rowpointer[order[j]]; k < rowpointer[order[j] + 1]; k++) {
                if (col[k] == i) {
                    if (abs(values[k]) > abs(max_value)) {
                        max_value = values[k];
                        pivot_row = j;
                    }

                    break;
                }
            }
        }

        if (max_value == 0.0) {
            return make_pair(0.0, "No");
        }

        det *= max_value;

        if (pivot_row != i) {
            det *= -1;
            swap(order[pivot_row], order[i]);
        }

        for (int j = i + 1; j < n; j++) {
            if (col[rowpointer[order[j]]] == i) {
                factor = values[rowpointer[order[j]]] / max_value;

                values[rowpointer[order[j]]] = 0.0;
                for (int k = rowpointer[order[j]] + 1, l =
                rowpointer[order[i]] + 1; k < rowpointer[order[j] + 1]; k++) {
                    if (col[k] == col[l]) {
                        values[k] -= values[l] * factor;
                        l++;
                    }
                }
            }
        }

        nw_rowpointer = rowpointer;
```

```
        for (int j = 0; j < n; j++) {
            for (int k = rowpointer[j]; k < rowpointer[j + 1]; k++) {
                if (abs(values[k]) > 1e-5) {
                    nw_values.push_back(values[k]);
                    nw_col.push_back(col[k]);
                }else {
                    for (int p = j + 1; p < n + 1; p++) {
                        nw_rowpointer[p] -= 1;
                    }
                }
            }
        }
        values = move(nw_values);
        col = move(nw_col);
        rowpointer = move(nw_rowpointer);
    }

    return make_pair(det, "Yes");
}
```

Тестирующая программа

Для тестирования выполненных заданий была разработана тестирующая программа. В ней присутствует класс который содержит аналогичные функции, только на плотных матрицах. В силу простоты этих функций и проверки их вручную, с ними можно сверяться. Далее генерируется случайные тесты на основе вводимых через консоль параметров. Также тесты учитывают заданную погрешность в силу не идеальности типа данных long double.

Вывод

В ходе данной лабораторной работы был подробно рассмотрен формат хранения матриц в разреженном виде. Также были изучены принципы работы функций и методов, необходимых для работы с разреженными матрицами. Данные знания и навыки помогут в будущем в предметной деятельности, в том числе при использовании библиотеки NumPy для работы с матрицами и тензорами.

Приложения

Ссылка на GitHub с лабораторной работой:
https://github.com/artemmavrov/lab1_linal