Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИТМО

Лабораторная работа No1

по дисциплине

"Линейная алгебра и анализ данных"

Семестр I

Выполнил:

студент

Семёнов Никита Викторович

торши Ромдхан

гр. **J3114**

ИСУ **467414**

ИСУ **467746**

Отчет сдан:

14.12.2024

Санкт-Петербург

2024

Цели и задачи

Цель лабораторной работы:
Изучение разреженно-строчного формата хранения матриц, реализация базовых операций над матрицами (сложение, умножение, вычисление определителя) и применение знаний линейной алгебры для работы с данными.

Задачи:
1. Реализовать класс для хранения матриц в разреженно-строчном формате с методами для подсчёта следа и получения элементов по заданным индексам.
2. Реализовать функции для выполнения базовых операций над матрицами: сложение, умножение на скаляр, умножение двух матриц.
3. Написать функцию для вычисления определителя матрицы и проверки её обратимости.
4. Протестировать разработанные функции на нескольких примерах и оформить результаты.

Ход работы

Задача 1: Разреженно-строчное хранение матрицы

Для реализации разреженно-строчного формата хранения матриц был разработан класс `SparseMatrix`. Для хранения матриц был реализован Координатный формат (COO). Разряженная матрица хранится в виде 3 списков: строк, столбцов и значений не нулевых элементов. Основные методы класса:
- `__init__`: инициализация матрицы с заданным количеством строк, столбцов и значениями.
- `trace`: вычисление следа (суммы элементов главной диагонали).
- `get_element`: получение элемента матрицы по указанным индексам.

В методе __init__ вызывается метод sparse, который переводит входящую матрицу в разряженный вид. Для этого создаётся словарь и заполняется не нулевыми элементами.

Метод trace суммирует все элементы на главной диагонали матрицы.

Метод get_element возвращает элемент с индексами i-1, j-1. -1 добав-

ляется потому что по условию нумерация должна начинаться с 1.

Также реализованы дополнительные методы

- `add_row`: для перевода обычной матрицы в разряженный вид.
- `print_sparse`: выводит разряженную матрицу
- `__str__`: выводит обычную матрицу

Реализацию этих методов можно найти на гитхабе в файле main.py https://github.com/user6778899/Linal_labs

Пример кода:

```
# Создание матрицы 3x3
mat = SparseMatrix(3, 3, [
    [1, 0, 0],
    [0, 2, 0],
    [0, 0, 3]
])

# Вывод матрицы и следа
print("Матрица:")
print_mat(mat)
print("След матрицы:", mat.trace())
```

Результат выполнения:

```
Матрица:
1 0 0
0 2 0
0 0 3
След матрицы: 6
```

Задача 2: Операции над матрицами

Для выполнения операций были реализованы следующие функции:
- `sum_matrices`: сложение двух матриц.
- `multiply_int`: умножение матрицы на скаляр.
- `multiply_matrices`: умножение двух матриц.

Функция sum_matrices делает копию первого входного аргумента, после чего проходясь по 2 разряженной матрице прибавляет к первой элементы и проверяет не равна ли сумма нулю. Если сумма равна нулю, то индекс элемента записывается в специальный zero_list и в конце эти элементы удаляются. Это сделанно для того чтобы в разряженной матрице не было нулевых элементов.

Функция multiply_int умножает все элементы разряженной матрицы на заданное число и возвращает новую матрицу.

Функция multiply_matrices создаёт 3-ий пустой экземпляр разряженной матрицы и записывает в него результат перемножения матриц.

Реализацию этих функций можно найти на гитхабе в файле main.py https://github.com/user6778899/Linal_labs

Пример сложения матриц:

```
# Матрица A
matrix_a = SparseMatrix(2, 2, [
    [1, 2],
    [3, 4]
])

# Матрица B
matrix_b = SparseMatrix(2, 2, [
    [5, 6],
    [7, 8]
])

# Сложение
result = sum_matrices(matrix_a, matrix_b)
print("Результат сложения:")
print_mat(result)
```

Результат выполнения:

```
Результат сложения:
6 8
10 12
```

Пример умножения матриц:

```
n1 = 4
m1 = 4
grid1 = [
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1]]
mat1 = SparseMatrix(n1, m1, grid1)
n2 = 4
m2 = 4
grid2 = [
    [0, 8, -3, 2],
    [0, 2, 0, 4],
    [1, 0, 0, 0],
    [0, 3, 0, -1]]
mat2 = SparseMatrix(n2, m2, grid2)

print('Матрица 1:')
print_grid(n1, m1, grid1)
print('разряженный вид:')
mat1.print_sparse()

print('Матрица 2:')
print_grid(n2, m2, grid2)
print('разряженный вид:')
mat2.print_sparse()

mat_sum = sum_matrices(mat1, mat2)
print('разряженный вид суммы:')
mat_sum.print_sparse()
print('обычный вид суммы:')
print_mat(mat_sum)
print('Умножение матрицы суммы на -1:')
mat_mult = multiply_int(mat_sum, -1)
print_mat(mat_mult)
print('Перемножение матриц:')
mat_mult2 = multiply_matrices(mat1, mat2)
print_mat(mat_mult2)
print('разряженный вид:')
mat_mult2.print_sparse()
```

Результат выполнения:

```
Матрица 1:
1  0  0  0
0  1  0  0
0  0  1  0
0  0  0  1
разряженный вид:
{(0, 0): 1, (1, 1): 1, (2, 2): 1, (3, 3): 1}
Матрица 2:
0  8  -3 2
0  2  0  4
1  0  0  0
0  3  0  -1
разряженный вид:
{(0, 1): 8, (0, 2): -3, (0, 3): 2, (1, 1): 2, (1, 3): 4, (2, 0): 1, (3, 1): 3, (
разряженный вид суммы:
{(0, 0): 1, (1, 1): 3, (2, 2): 1, (0, 1): 8, (0, 2): -3, (0, 3): 2, (1, 3): 4, (
обычный вид суммы:
1  8  -3 2
0  3  0  4
1  0  1  0
0  3  0  0
Умножение матрицы суммы на -1:
-1 -8 3  -2
0  -3 0  -4
-1 0  -1 0
0  -3 0  0
Перемножение матриц:
0  8  -3 2
0  2  0  4
1  0  0  0
0  3  0  -1
разряженный вид:
{(0, 1): 8, (0, 2): -3, (0, 3): 2, (1, 1): 2, (1, 3): 4, (2, 0): 1, (3, 1): 3, (
```

Как видно по результатам, нулевые элементы в разряженной матрице действительно удаляются.

Задача 3: Определитель и проверка обратимости

Для вычисления определителя реализована функция `determinant`, которая использует рекурсивное разложение по строкам. Проверка обратимости выполняется с использованием значения определителя (если он не равен нулю, то матрица обратима).

Функция determinant использует функцию minor для вычисления миноров матрицы, тем самым разбивая задачу на более маленькие. minor возвращает нужное число, если входящая матрица 2 на 2, в иных случаях снова вызывается determinant. Также предусмотрен случай для маленьких матриц 1 на 1.

Функция exists_reverse проверяет обратимость матрицы. Для этого она вычисляет определитель матрицы и возвращает 'Да' если он не равен 0, иначе 'Нет'

Реализацию этих функций можно найти на гитхабе в файле main.py https://github.com/user6778899/Linal_labs

Пример кода:

```
# Матрица 3x3
# Матрица 3x3
mat = SparseMatrix(3, 3, [
    [2, -1, 0],
    [1, 3, 2],
    [0, -2, 1]
])

# Вычисление определителя
det = determinant(mat)
print("Определитель:", det)
print("Обратима ли матрица:", "да" if det != 0 else "нет")
```

Результат выполнения:

```
Определитель: 16
Обратима ли матрица: Да
```

Выводы

В ходе выполнения лабораторной работы были достигнуты следующие результаты:
1. Изучен и реализован разреженно-строчный формат хранения матриц.
2. Выполнены основные операции над матрицами: сложение, умножение на скаляр, умножение двух матриц.
3. Реализована функция для вычисления определителя матрицы и проверки её обратимости.
4. Проведено тестирование разработанных функций, подтверждена их корректность.

Реализация показала высокую эффективность при работе с разреженными матрицами, а также позволила лучше понять принципы линейной алгебры и их программное применение.

Разреженно-строчный вид матрицы очень эффективен если в матрице много нулевых элементов. Хранение такой матрицы в разреженно-строчном виде занимает намного меньше места, а действия с матрицей выполняются намного быстрее, потому что не нужно проходиться по нулевым элементам.

Дополнительные тесты можно найти в файле tests.py, на гитхабе по ссылке:
https://github.com/user6778899/Linal_labs
Там же лежит весь код, в файле main.py.

Весь код (файла main.py)

```
class SparseMatrix:
    def __init__(self, n, m, grid=[]):
        self.n = n
        self.m = m
        self.rows = []
        self.cols = []
        self.values = []
        if grid != []:
```

```
            for i in range(n):
                self.add_row(grid[i], i)

    def add_row(self, row, row_index):
        for col_index, value in enumerate(row):
            if value != 0:
                self.rows.append(row_index)
                self.cols.append(col_index)
                self.values.append(value)

    def trace(self):
        trace_sum = 0
        for r, c, v in zip(self.rows, self.cols, self.values):
            if r == c:
                trace_sum += v
        return trace_sum

    def get_element(self, row_index, col_index, o=0):
        if o != 0:
            row_index, col_index = row_index - 1, col_index - 1
        for i in range(len(self.rows)):
            if self.rows[i] == row_index and self.cols[i] == col_index:
                return self.values[i]
        return 0

    def print_sparse(self): # выводит разряженную матрицу
        print(self.rows)
        print(self.cols)
        print(self.values)


    def __str__(self):
        full_matrix = [[0] * self.m for _ in range(self.n)]
        for i, j, value in zip(self.rows, self.cols, self.values):
            full_matrix[i][j] = value
        return '\n'.join(' '.join(map(str, row)) for row in full_matrix)


def print_grid(n, m, grid, k=3): # Выводит обычный вид матрицы
    for i in range(n):
        for j in range(m):
```

```
            tab = ' '*(k-len(str(grid[i][j])))
            print(grid[i][j], end=tab)
        print()


def sum_matrices(matrix1, matrix2):
    if matrix1.n != matrix2.n or matrix1.m != matrix2.m:
        print('Матрицы разного размера, их нельзя сложить!')
        return

    result = SparseMatrix(matrix1.n, matrix1.m)

    for r, c, v in zip(matrix1.rows, matrix1.cols, matrix1.values):
        result.rows.append(r)
        result.cols.append(c)
        result.values.append(v)

    for r, c, v in zip(matrix2.rows, matrix2.cols, matrix2.values):
        found = False
        for i in range(len(result.rows)):
            if result.rows[i] == r and result.cols[i] == c:
                result.values[i] += v
                found = True
                break
        if not found:
            result.rows.append(r)
            result.cols.append(c)
            result.values.append(v)

    return result


def multiply_int(matrix, scalar):
    result = SparseMatrix(matrix.n, matrix.m)
    for r, c, v in zip(matrix.rows, matrix.cols, matrix.values):
        result.rows.append(r)
        result.cols.append(c)
        result.values.append(v * scalar)
    return result
```

```
def multiply_matrices(matrix1, matrix2):
    if matrix1.m != matrix2.n:
        print('Матрицы разного размера, их нельзя перемножить!')
        return

    result = SparseMatrix(matrix1.n, matrix2.m)

    for i in range(matrix1.n):
        for j in range(matrix2.m):
            sum_value = 0
            for r1, c1, v1 in zip(matrix1.rows, matrix1.cols, matrix1.values):
                if r1 == i:
                    for r2, c2, v2 in zip(matrix2.rows, matrix2.cols, matrix2.va
                        if c1 == r2 and c2 == j:
                            sum_value += v1 * v2
            if sum_value != 0:
                result.rows.append(i)
                result.cols.append(j)
                result.values.append(sum_value)

    return result



def get_minor(matrix, row, col):
    """
    Получение минора для разряженной матрицы (матрицы без строки `row` и столбца
    """
    minor = SparseMatrix(matrix.n - 1, matrix.m - 1)
    for i in range(matrix.n):
        if i == row:
            continue
        for j in range(matrix.m):
            if j == col:
                continue
            value = matrix.get_element(i, j)
            if value != 0:
                new_row = i if i < row else i - 1
                new_col = j if j < col else j - 1
                minor.add_row([value if x == new_col else 0 for x in range(minor
    return minor
```

```
def determinant(matrix):
    """
    Вычисление детерминанта для разряженной матрицы рекурсивно.
    """
    if matrix.n != matrix.m:
        raise ValueError("Матрица должна быть квадратной для вычисления детермин

    n = matrix.n
    if n == 1:
        return matrix.get_element(0, 0)
    if n == 2:
        a = matrix.get_element(0, 0)
        b = matrix.get_element(0, 1)
        c = matrix.get_element(1, 0)
        d = matrix.get_element(1, 1)
        return a * d - b * c

    det = 0
    for col in range(n):
        minor = get_minor(matrix, 0, col)
        det += ((-1) ** col) * matrix.get_element(0, col) * determinant(minor)

    return det


def exists_reverse(mat): # Существует ли обратная
    if determinant(mat) == 0:
        return 'Нет'
    else:
        return 'Да'


if __name__ == "__main__":
    print('|--------------------|')
    print('Задание 1')
    n, m = map(int, input("Введите размер матрицы N и M через пробел: ").split()
    matrix = SparseMatrix(n, m)

    print("Введите матрицу (через пробелы на каждой строке):")
```

```
for i in range(n):
    row = list(map(float, input().split()))
    matrix.add_row(row, i)

trace = matrix.trace()
print(f"След матрицы: {trace}")

i, j = map(int, input("Введите индексы для получения элемента матрицы (через
element = matrix.get_element(i, j, 1)
print(f"Элемент на позиции ({i}, {j}): {element}")


print('|--------------------|')
print('Задание 2')
n1, m1 = map(int, input("Введите размер первой матрицы N1 и M1 через пробел:
matrix1 = SparseMatrix(n1, m1)

print("Введите первую матрицу (через пробелы на каждой строке):")
for i in range(n1):
    row = list(map(float, input().split()))
    matrix1.add_row(row, i)

n2, m2 = map(int, input("Введите размер второй матрицы N2 и M2 через пробел:
matrix2 = SparseMatrix(n2, m2)

print("Введите вторую матрицу (через пробелы на каждой строке):")
for i in range(n2):
    row = list(map(float, input().split()))
    matrix2.add_row(row, i)

print("Сложение матриц:")
sum_result = sum_matrices(matrix1, matrix2)
if sum_result:
    print(sum_result)

scalar = float(input("Введите скаляр для умножения: "))
print(f"Умножение первой матрицы на скаляр {scalar}:")
scalar_result = multiply_int(matrix1, scalar)
print(scalar_result)

print("Умножение матриц:")
```

```
multiplication_result = multiply_matrices(matrix1, matrix2)
if multiplication_result:
    print(multiplication_result)


print('|--------------------|')
print('Задание 3')
n = int(input("Введите размер матрицы (N): "))
matrix = SparseMatrix(n, n)

print("Введите матрицу:")
for i in range(n):
    row = list(map(float, input().split()))
    matrix.add_row(row, i)

print(matrix)

det = determinant(matrix)
print("Определитель:", det)
print(exists_reverse(matrix))
```