

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

Тиганов Вадим Игоревич
гр. J3112
ИСУ 467701

Вагин Арсений Антонович
гр. J3112
ИСУ 465339

Отчет сдан:
XX.12.2024

Санкт-Петербург
2024

Цель лабораторной работы

Освоить основные концепции линейной алгебры и анализа данных по работе с матрицами. Научиться реализовывать и тестировать алгоритмы работы с матрицами в разреженно-строчном формате. Изучить и понять принципы работы алгоритмов, а также .tex верстания для создания отчета.

Задачи лабораторной работы

1. Реализация хранения матриц в разреженно-строчном виде.
2. Реализация операций над матрицами.
3. Вычисление определителя и проверка существования обратной матрицы.
4. Тестирование и проверка правильности работы алгоритмов.
5. Верстка отчета в формате L^AT_EX

Ход выполнения лабораторной работы

Задача 1

Задача заключалась в реализации следующих функций в классе: (был выбран ЯП Python, полный листинг кода см. в приложении А)

- Ввод матрицы заданного размера пользователем.
- Подсчет следа матрицы.
- Поиск и вывод элемента матрицы по заданным индексам.
- Тестирование работы программы и создание консольного пользовательского интерфейса.

Использованные библиотеки и инструменты языка

В ходе написания программы были использованы только стандартные средства языка. Также для удобства и лучшей читаемости кода была импортирована библиотека `typing`.

Реализация функций и основные идеи

Все функции были реализованы в классе `MatrixKeeper`

Написаны функции:

1. `inputMatrix` - ввод матрицы пользователем,
2. `trace` - поиск следа матрицы,
3. `findByIndex` - поиск элемента по введенному индексу.

Суть работы алгоритмов:

- `inputMatrix`: Приглашение пользователя ко вводу. Вначале через пробел вводятся два числа типа `int` - размер матрицы. Вторым приглашением вводится матрица по строке, элементы в строке разделяются пробелом.
- `trace`: След матрицы - сумма элементов главной диагонали этой матрицы. Циклом, оставаясь в пределах матрицы, проходимся по элементам с индексами вида `[i][i]`, считаем сумму таких элементов. Можем так делать по той причине, что матрица имеет следующую структуру в классе:


```

Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 9 5
Введите матрицу по строке, разделяя элементы строки пробелом

1 2 3 5 1
1 6 9 2 4
3 5 1 2 5
1 2 5 7 4
-10000 1 6 32 1
0.0003 1 2 6 3
4 2 8 3 7
5 4 2 3 6
5 3 1 2 7

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Найти след матрицы.
3: Найти элемент по индексу в заданной матрице.
4: Выйти из программы.

Введите номер соответствующей опции: 3
Введите номер строки: 5
Введите номер столбца: 1
Элемент матрицы [5][1] = -10000.0

```

Ручной тест поиска страшного элемента в страшной матрице 9x5.

Итак, справились с первым заданием.

Задача 2

Во второй задаче требуется реализовать три функции для операций с матрицами: (полный листинг кода см. в приложении Б)

- Сложение двух матриц.
- Умножение двух матриц.
- Умножение матрицы на скаляр.

Использованные библиотеки и инструменты языка

В ходе написания программы были использованы только стандартные средства языка. Также для удобства и лучшей читаемости кода была импортирована библиотека `typing`.

Были реализованы 3 функции:

1. `matrixAddition` - сложение двух матриц.
2. `matrixByMatrixMultiplication` - перемножение двух матриц.
3. `matrixScalarMultiplication` - умножение одной из двух матриц на заданное число.

Согласно техническому заданию, функция ввода матрицы пользователем была импортирована из файла предыдущего задания. (`inputMatrix`)

Суть работы алгоритмов:

- `matrixAddition`: Классическое сложение матрицы. Возвращаем матрицу, где каждый элемент с определенными индексами равен сумме элементов с соответствующими индексами из складываемых матриц.

$$(A + B)_{i,k} = A_{i,k} + B_{i,k}$$

- `matrixByMatrixMultiplication`: Умножение матрицы на матрицу. Возвращаем матрицу, где каждый элемент с определенными индексами равен сумме произведений элементов соответствующей строки первой матрицы и столбца второй матрицы.

$$(AB)_{i,j} = \sum_{k=1}^n A_{i,k} \cdot B_{k,j}$$

- `matrixScalarMultiplication`: Умножение матрицы на скаляр. Возвращаем матрицу, где каждый элемент равен произведению соответствующего элемента исходной матрицы и скаляра.

$$(cA)_{i,j} = c \cdot A_{i,j}$$

Тестирование программы

Написаны юниттесты для каждой функции класса с помощью стандартной библиотеки `unittest`. Также протестировано вручную.

Листинг кода теста см. в приложении Б-test.

```
✓ task2_test.py
  ✓ TestMatrixOperations
    ✓ test_matrixAddition
    ✓ test_matrixByMatrixMultiplication
    ✓ test_matrixScalarMultiplication
```

Успешное прохождение unittests

```

Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

1 2
3 4

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 2
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

6 5
7 9

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 3
Результат сложения матриц:
[7.0, 7.0]
[10.0, 13.0]

```

Ручной тест сложения двух матриц 2 на 2.

```

Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

1 2
6 7

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 5
Выберите матрицу для умножения на скаляр (1 - первая матрица, 2 - вторая матрица): 1
Введите скаляр: 2.718281828459045
Результат умножения матрицы на скаляр:
[2.718281828459045, 5.43656365691809]
[16.30969097075427, 19.027972799213316]

```

Ручной тест умножения матрицы 2 на 2 на число Эйлера с небольшим количеством знаков после запятой.

```

Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

1 6
-0.1238 52

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 2
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

0.312837 2
1 1

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 4
Результат умножения матриц:
[6.312837, 8.0]
[51.9612707794, 51.7524]

```

Ручной тест умножения двух страшных матриц 2 на 2.

Matrix Calculator

Matrix A Input

row: 2, column: 2

1	6
-0.123	52

Matrix B Input

row: 2, column: 2

.12837	2
1	1

Result

AB = $\begin{bmatrix} 6.312837 & 8 \\ 51.96127078 & 51.7524 \end{bmatrix}$

Удостоверимся в правильности полученного результата, умножив матрицы на сайте-калькуляторе. Результат правильный.

Справились со вторым заданием.

Задача 3

В третьей задаче требуется реализовать следующие функции: (полный листинг кода см. в приложении В)

- Вычисление определителя матрицы. По тз - матрицы размером до 100x100
- Ответ на вопрос: существует ли матрица, обратная данной.

Использованные библиотеки и инструменты языка

В ходе написания программы были использованы только стандартные средства языка. Также для удобства и лучшей читаемости кода была импортирована библиотека `typing`.

Были реализованы 3 функции:

1. `determinantOfMatrix` - поиск определителя матрицы.
2. `isMatrixInvertible` - функция, которая отвечает на вопрос, существует ли обратная матрица к данной.
3. `gauss` - вспомогательная функция - метод Гаусса, с помощью которого считал определитель любой квадратной матрицы.

Суть работы алгоритмов:

- `determinantOfMatrix`: оболочка функции, проверяем размер матрицы и обращаемся к функции Гаусса для поиска определителя.
- `isMatrixInvertible`: сравниваем полученный из первой функции определитель с нулем, возвращаем логическое значение "да" или "нет".
- `gauss`: самая интересная функция. Для матрицы 1x1 и 2x2 посчитано вручную.

```
1     n = len(matrix)
2     if n == 1:
3         return matrix[0][0]
4     elif n == 2:
5         return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
```

1x1: Определитель равен единственному элементу.

2x2: Определитель считаем как разность произведения элементов главной и побочной диагоналей.

Проверка на вырожденную матрицу:

```
1     for i in range(n):
2         max_row = max(range(i, n), key=lambda r: abs(matrix[r][i]))
3         if matrix[max_row][i] == 0:
4             raise ValueError("Matrix is singular, determinant is zero.")
```

Как работает:

Цикл проходит по каждому столбцу i от 0 до $n - 1$. На каждом шаге ищется ведущий элемент в текущем столбце. Для этого используется команда:

```
1     max_row = max(range(i, n), key=lambda r: abs(matrix[r][i]))
```

Здесь `range(i, n)` генерирует индексы строк от текущей строки i до последней $n-1$, `lambda r: abs(matrix[r][i])` вычисляет модуль элемента в текущем столбце для каждой строки r , а `max(..., key=...)` выбирает индекс строки `max_row`, где модуль элемента в столбце максимален.

Далее выполняется проверка:

```
1     if matrix[max_row][i] == 0:
2         raise ValueError("Matrix is singular, determinant is zero.")
```

Если максимальный элемент в столбце равен 0, то все элементы ниже i -й строки в этом столбце тоже равны 0. Это означает, что матрица вырожденная, и её определитель равен нулю, поэтому дальнейшие вычисления прекращаются.

Приведение к нижнетреугольному виду и вычисление определителя:

Для приведения матрицы к нижнетреугольному виду используется следующий алгоритм. Если строка с максимальным элементом в текущем столбце `max_row` не совпадает с текущей строкой `i`, строки меняются местами:

```
1     if max_row != i:
2         matrix[i], matrix[max_row] = matrix[max_row], matrix[i]
3         swap_count += 1
```

После этого элементы ниже ведущего в текущем столбце обнуляются. Для каждой строки `j`, начиная с `i + 1`, вычисляется коэффициент `factor`, с помощью которого корректируются элементы строки:

```
1     for j in range(i + 1, n):
2         factor = matrix[j][i] / matrix[i][i]
3         for k in range(i, n):
4             matrix[j][k] -= factor * matrix[i][k]
```

После завершения приведения к нижнетреугольному виду вычисляется определитель матрицы. Определитель является произведением всех диагональных элементов приведённой матрицы:

```
1     determinant = 1.0
2     for i in range(n):
3         determinant *= matrix[i][i]
```

Учитывается знак определителя, зависящий от числа перестановок строк:

```
1     return (-1) ** swap_count * determinant
```

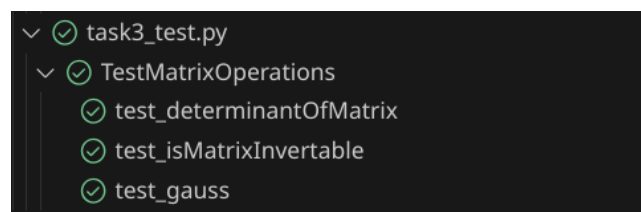
Здесь каждая перестановка строк меняет знак определителя, что учитывается выражением `(-1) ** swap_count`.

Таким образом реализован метод Гаусса для поиска определителя любой квадратной матрицы.

Тестирование программы

Написаны юниттесты для каждой функции класса с помощью стандартной библиотеки `unittest`. Также протестировано вручную.

Листинг кода теста см. в приложении B-test.



Успешное прохождение unittests

```

Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

1 2
3 4

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Вычислить определитель матрицы.
3: Проверить, существует ли обратная матрица.
4: Выйти из программы.

Введите номер соответствующей опции: 2
Определитель матрицы: -2.0

```

Вручную проверям подсчет определителя матрицы 2x2. Ответ верный.

```

Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

0 0
0 0

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Вычислить определитель матрицы.
3: Проверить, существует ли обратная матрица.
4: Выйти из программы.

Введите номер соответствующей опции: 2
Определитель матрицы: 0.0

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Вычислить определитель матрицы.
3: Проверить, существует ли обратная матрица.
4: Выйти из программы.

Введите номер соответствующей опции: 3
Матрица необратима.

```

Вручную проверяем подсчет определителя и обратимости нулевой матрицы. Ответ верный.

```

1 2 3
5 6 8

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Вычислить определитель матрицы.
3: Проверить, существует ли обратная матрица.
4: Выйти из программы.

Введите номер соответствующей опции: 2
Определитель можно вычислить только для квадратной матрицы.

```

Конечно же, вручную пытаемся обработать не квадратную матрицу.

Итак, все задания сделаны, тесты пройдены. Пора перейти к выводам.

Вывод

В ходе выполнения данной работы были реализованы и протестированы различные функции для работы с матрицами. Это включало ввод матрицы, вычисление её следа, поиск элементов по индексам, а также операции сложения, умножения матриц и умножения матрицы на скаляр. Кроме того, были реализованы функции для вычисления определителя матрицы и проверки её обратимости.

Работа над этими задачами позволила нам углубить понимание основ линейной алгебры и её применения в программировании. Особенно интересным оказался метод Гаусса для вычисления определителя матрицы, который демонстрирует применение алгоритмов линейной алгебры в решении сложных задач.

Процесс тестирования, как автоматического, так и ручного, подтвердил корректность реализованных алгоритмов и их способность работать с матрицами различных размеров. Это важно для обеспечения надежности и точности вычислений, что является важным аспектом в машинном обучении и анализе данных.

В целом, выполнение данной работы не только укрепило наши навыки программирования и работы с матрицами, но и заставило задуматься, как полученные знания помогут в будущем. Линейная алгебра является основой для многих алгоритмов машинного обучения, и понимание её принципов и методов является ключом для успешного применения этих алгоритмов на практике.

В будущем мы планируем продолжить изучение машинного обучения, включая более сложные алгоритмы и методы. Уверены, что полученные знания и навыки станут основой для дальнейшего развития в этой области.

Приложения

Список использованной литературы