Лабораторная работа 2. Основы работы с библиотекой NumPy

Цель работы: *исследовать базовые возможности библиотеки NumPy языка программирования Python.*

Ход работы

В статье рассмотрены различные способы получения элементов из массивов типа *numpy.ndarray* библиотеки *numpy*.

Если вы читаете эту статью, то, наверное, знаете, зачем нужна библиотека *numpy*. Если в “двух словах”, то *numpy* – это библиотека для языка программирования *Python*, которая предоставляет в распоряжение разработчика инструменты для эффективной работы с многомерными массивами и высокопроизводительные вычислительные алгоритмы.

В этой статье все внимание уделено работе с элементами двумерных массивов. В практике очень часто приходится извлекать из уже готовой матрицы как отдельные элементы, так и целые строки, столбцы или их комбинации, о том, как это делать мы и поговорим.

Для начала импортируем библиотеку *numpy*.

```
>>> import numpy as np
Теперь создадим матрицу, с которой будем работать.
>>> m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
>>> print(m)
[[1 2 3 4]
 [5 6 7 8]
 [9 1 5 7]]
```

Получим вот такую таблицу чисел.

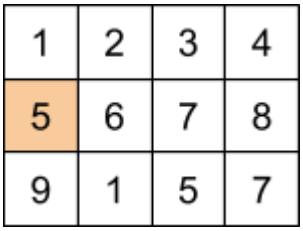
1	2	3	4
5	6	7	8
9	1	5	7

Не забывайте, что нумерация строк и столбцов в библиотеки *numpy* (и в самом *Python*) начинается с нуля.

Элемент матрицы с заданными координатами

Извлечем элемент из нашей матрицы с координатами (1, 0), 1 – это номер строки, 0 – номер столбца.

Здесь и далее, элементы, с которыми мы работаем в матрице будут окрашены в оранжевый цвет на соответствующей картинке.

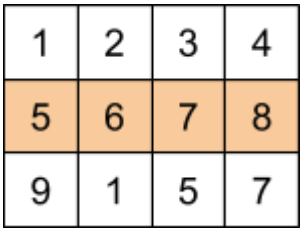



```
>>> m[1, 0]
5
```

В приведенной записи, в квадратных скобках указывается номер строки – первой цифрой и номер столбца – второй.

Строка матрицы

Получим вторую строчку матрицы.

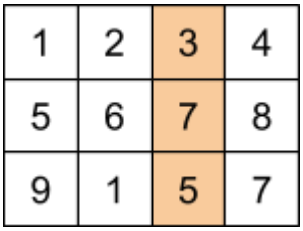



```
>>> m[1, :]
matrix([[5, 6, 7, 8]])
```

Двоеточие означает “все элементы”, в приведенном примере, первый элемент – это номер строки, второй – указание на то, что необходимо взять элементы всех столбцов матрицы.

Столбец матрицы

Извлечем третий столбец матрицы.

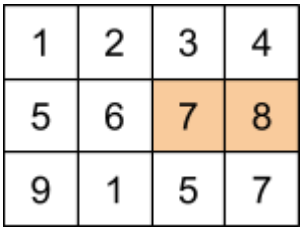



```
>>> m[:, 2]
matrix([[3],
        [7],
        [5]])
```

Здесь, также как в предыдущем примере, используется двоеточие. Первый элемент в квадратных скобках означает, что мы возьмем по элементу из каждой строки, которые находится в столбце, указанном во втором элементе.

Часть строки матрицы

Иногда возникает задача взять не все элементы строки, а только часть: рассмотрим пример, когда нам из второй строки нужно извлечь все элементы, начиная с третьего.

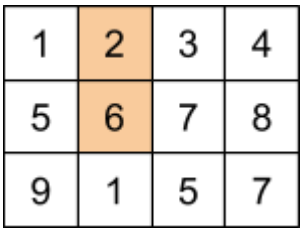



```
>>> m[1, 2:]
matrix([[7, 8]])
```

Запись **2:** означает, что начиная с третьего столбца включительно (т.к. нумерация начинается с 0, то третий элемент имеет индекс 2) взять все оставшиеся в ряду элементы .

Часть столбца матрицы

Аналогично предыдущему примеру, можно извлечь только часть столбца матрицы.

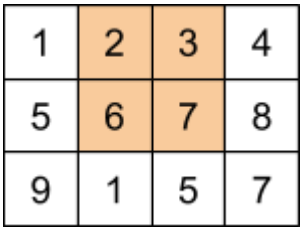



```
>>> m[0:2, 1]
matrix([[2],
        [6]])
```

В приведенной записи **0:2** означает: взять все элементы столбца начиная с индекса 0, заканчивая индексом 2, но последний элемент в результат не включать.

Непрерывная часть матрицы

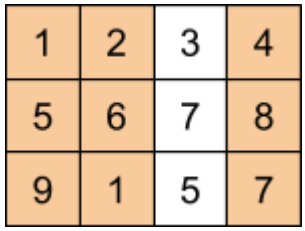
Извлечем из заданной матрицы матрицу, располагающуюся так как показано на рисунке ниже.




```
\>>> m[0:2, 1:3]
matrix([[2, 3],
        [6, 7]])
```

Произвольные столбцы / строки матрицы

Numpy позволяет извлекать произвольный набор столбцов или строк матрицы, строки (столбцы) которые нужно извлечь передаются в виде списка.




```
>>> cols = [0, 1, 3]
>>> m[:, cols]
matrix([[1, 2, 4],
        [5, 6, 8],
        [9, 1, 7]])
```

Расчет статистик по данным в массиве

Для начала создадим матрицу, которая нам понадобится в работе.

```
>>> m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
>>> print(m)
[[1 2 3 4]
 [5 6 7 8]
 [9 1 5 7]]
```

В этом случае будет создан объект типа *matrix*.

```
>>> type(m)
<class 'numpy.matrixlib.defmatrix.matrix'>
```

Matix можно превратить в *ndarray* следующим образом:

```
>>> m = np.array(m)
>>> type(m)
<class 'numpy.ndarray'>
```

В любом случае наша таблица чисел будет выглядеть следующим образом.

1	2	3	4
5	6	7	8
9	1	5	7

Размерность массива

Для определения размерности массива *Numpy* используйте атрибут *shape*.

```
>>> m.shape
(3, 4)
```

В результате мы получим кортеж из двух элементов, первый из них – это количество строк, второй – столбцов.

Вызов функции расчета статистики

Для расчета той или иной статистики, соответствующую функцию можно вызвать как метод объекта, с которым вы работаете. Для нашего массива это будет выглядеть так.

```
>>> m.max()
9
```

Тот же результат можно получить вызвав библиотечную функцию с соответствующим именем, передав ей в качестве аргумента массив.

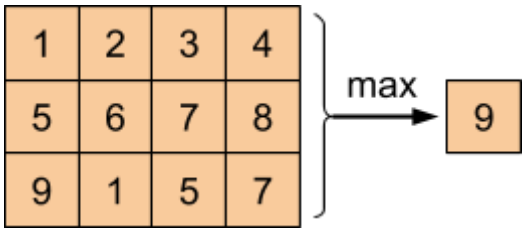
```
>>> np.max(m)
9
```

Расчет статистик по строкам или столбцам массива

Вызовем функцию вычисления статистики (максимальный элемент) без аргументов.

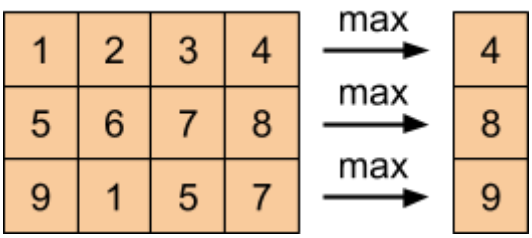
```
>>> m.max()
9
```

В этом случает будут обработаны все элементы массива.



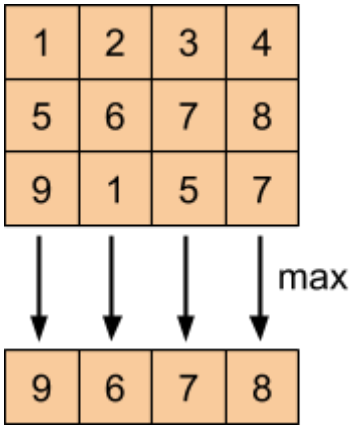
Если необходимо найти максимальный элемент в каждой строке, то для этого нужно передать в качестве аргумента параметр *axis=1*.

```
>>> m.max(axis=1)
matrix([[4],
        [8],
        [9]])
```



Для вычисления статистики по столбцам, передайте в качестве параметра аргумент *axis=0*.

```
>>> m.max(axis=0)
matrix([[9, 6, 7, 8]])
```



Функции (методы) для расчета статистик в *Numpy*

Ниже, в таблице, приведены методы объекта *ndarray* (или *matrix*), которые, как мы помним из раздела выше, могут быть также вызваны как функции библиотеки *Numpy*, для расчета статистик по данным массива.

Имя метода	Описание
argmax	Индексы элементов с максимальным значением (по осям)
argmin	Индексы элементов с минимальным значением (по осям)
max	Максимальные значения элементов (по осям)
min	Минимальные значения элементов (по осям)
mean	Средние значения элементов (по осям)
prod	Произведение всех элементов (по осям)
std	Стандартное отклонение (по осям)
sum	Сумма всех элементов (по осям)
var	Дисперсия (по осям)

Вычислим некоторые из представленных выше статистик.

```
>>> m.mean()
4.833333333333333
>>> m.mean(axis=1)
matrix([[2.5],
        [6.5],
        [5.5]])
>>> m.sum()
58
>>> m.sum(axis=0)
matrix([[15,  9, 15, 19]])
```

Использование boolean массива для доступа к ndarray

Использование *boolean* массивов для доступа к данным порой является более лучшим вариантом, чем использование численных индексов.

Для начала создадим несколько массивов, с которыми мы будем работать.

```
>>> nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> letters = np.array(['a', 'b', 'c', 'd', 'a', 'e', 'b'])
```

Как вы знаете, в *Python* есть такой тип данных – *boolean*. Переменные этого типа принимают одно из двух значений: *True* или *False*. Такие переменные можно создать самостоятельно:

```
>>> a = True
```

либо они могут являться результатом какого-то выражения:

```
>>> b = 5 > 7
>>> print(b)
False
```

Используя второй подход, можно построить на базе созданных нами в самом начале *ndarray* массивов массивы с элементами типа *boolean*.

```
>>> less_then_5 = nums < 5
>>> less_then_5
array([ True,  True,  True,  True, False, False, False, False, False,
       False])
```

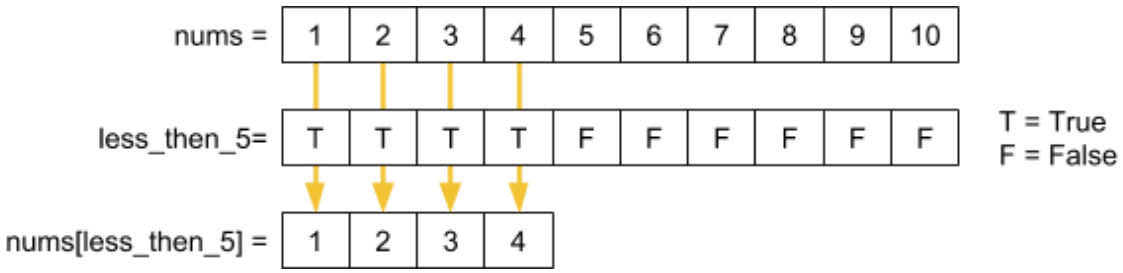
В этом примере мы создали *boolean* массив, в котором на месте элементов из *nums*, которые меньше пяти стоит *True*, в остальных случаях – *False*. Построим массив, в котором значение *True* будут иметь элементы, чей индекс совпадает с индексами, на которых стоит символ *'a'* в массиве *letters*.

```
>>> pos_a = letters == 'a'
>>> pos_a
array([ True, False, False, False,  True, False, False])
```

Самым замечательным в использовании *boolean* массивов при работе с *ndarray* является то, что их можно применять для построения выборок. Вернемся к рассмотренным выше примерам.

```
>>> less_then_5 = nums < 5
>>> less_then_5
array([ True,  True,  True,  True, False, False, False, False, False,
       False])
```

Если мы переменную *less_then_5* передадим в качестве списка индексов для *nums*, то получим массив, в котором будут содержаться элементы из *nums* с индексами равными индексам *True* позиций массива *less_then_5*, графически это будет выглядеть так.




```
>>> nums[less_then_5]
array([1, 2, 3, 4])
```

Данный подход будет работать с массивами большей размерности. Возьмем уже знакомую нам по предыдущим урокам матрицу.

```
>>> m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
>>> print(m)
[[1 2 3 4]
 [5 6 7 8]
 [9 1 5 7]]
```

Построим логическую матрицу со следующим условием: *m>=3 and m<=7*, в *Numpy* нельзя напрямую записать такое условие, поэтому воспользуемся функцией *logical_and()*, ее и многие другие полезные функции вы сможете найти на странице *Logic functions*.

```
>>> mod_m = np.logical_and(m>=3, m<=7)
>>> mod_m
matrix([[False, False,  True,  True],
        [ True,  True,  True, False],
        [False, False,  True,  True]])
>>> m[mod_m]
matrix([[3, 4, 5, 6, 7, 5, 7]])
```

В результате мы получили матрицу с одной строкой, элементами которой являются все отмеченные как *True* элементы из исходной матрицы.

Boolean выражение в *Numpy* можно использовать для индексации, не создавая предварительно *boolean* массив. Получить соответствующую выборку можно, передав в качестве индекса для объекта *ndarray,* условное выражение. Для иллюстрации данной возможности воспользуемся массивом *nums*.

```
>>> nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> nums[nums < 5]
array([1, 2, 3, 4])
```

Еще одно возможное применение *boolean* массивов в *Numpy* – это модификация данных. В этом случае будет удобен подход индексации с использованием *boolean* выражения. Присвоим всем элементам из массива *nums* меньшим 5 значение 10, сделать это можно так.

```
>>> nums[nums < 5] = 10
>>> print(nums)
[10 10 10 10  5  6  7  8  9 10]
```

Как вы можете видеть, элементы 1, 2, 3 и 4 были заменены на 10. Данных подход работает и с двумерными массивами. Покажем это на матрице *m*.

```
>>> m[m > 7] = 25
>>> print(m)
[[ 1  2  3  4]
 [ 5  6  7 25]
 [25  1  5  7]]
```

Дополнительные функции

Вектора и матрицы – это основные объекты, которыми приходится оперировать в машинном обучении. *Numpy* предоставляет довольно много удобных функций, которые строят эти объекты.

np.arange()

Функция *arange()* аналогична по своему назначению функции *range()* из стандартной библиотеки *Python*. Ее основное отличие заключается в том, что *arange()* позволяет строить вектор с указанием шага в виде десятичной дроби.

Синтаксис использования функции следующий:

arange(stop)
arange(start, stop)
arange(start, stop, step)

В первом варианте будет создан вектор из целых чисел от 0 до *stop*.

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Второй вариант позволяет задавать интервал, в этом случае вектор также будет содержать целые числа.

```
>>> np.arange(5, 12)
array([ 5,  6, 7, 8, 9, 10, 11])
```

Третий вариант позволяет определить интервал чисел и шаг, который может быть десятичным числом

```
\>>> np.arange(1, 5, 0.5)
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

np.matrix()

Matrix является удобным инструментом для задания матрицы. При этом можно использовать *Matlab* стиль, либо передать в качестве аргумента список *Python* (или массив *Numpy*).

Вариант со списком *Python*.

```
>>> a = [[1, 2], [3, 4]]
>>> np.matrix(a)
matrix([[1, 2],
        [3, 4]])
```

Вариант с массивом *Numpy*.

```
>>> b = np.array([[5, 6], [7, 8]])
>>> np.matrix(b)
matrix([[5, 6],
        [7, 8]])
```

Вариант в *Matlab* стиле.

```
>>> np.matrix('[1, 2; 3, 4]')
matrix([[1, 2],
        [3, 4]])
```

np.zeros(), np.eye()

В арсенале *Numpy* есть функции для создания специальных матриц: нулевых и единичных. Нулевой называется матрица, состоящая полностью из нулей. Для ее создания удобно использовать функцию *zeros()*, в качестве аргумента в нее передается кортеж из двух элементов, первый из них – это количество строк, второй – столбцов.

```
>>> np.zeros((3, 4))
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

Функция *eye()* создает единичную матрицу – квадратную матрицу, у которой элементы главной диагонали равны единицы, все остальные – нулю.

```
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

np.ravel()

Функция *np.ravel()* используется для того, чтобы преобразовать матрицу в одномерный вектор.

Создадим двумерную матрицу размера *3x3*.

```
>>> A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> A
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Применим функцию *ravel()* к этой матрице.

```
>>> np.ravel(A)
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

У *ravel()* есть параметр *order*, который отвечает за порядок построения одномерного массива, по умолчанию он равен *'C'*, что означает – массив будет собираться из строк исходной матрицы.

```
>>> np.ravel(A, order='C')
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Если указать *order='F'*, то в качестве элементов для сборки будут выступать столбцы матрицы.

```
>>> np.ravel(A, order='F')
array([1, 4, 7, 2, 5, 8, 3, 6, 9])
```

np.where()

Данная функция возвращает один из двух заданных элементов в зависимости от условия. Ее можно использовать для обработки численных данных.

```
>>> a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.where(a % 2 == 0, a * 10, a / 10)
array([ 0. ,  0.1, 20. , 0.3, 40. ,  0.5, 60. , 0.7, 80. , 0.9])
```

В задачах машинного обучения эта функция хорошо подходит для реализации обработки данных с помощью пороговой функции.

```
>>> a = np.random.rand(10)
>>> a
array([0.99379074, 0.98387541, 0.2043767 , 0.11935986, 0.01063287,
       0.11146634, 0.50504848, 0.96046102, 0.3645473 , 0.6843563 ])
>>> np.where(a > 0.5, True, False)
array([ True,  True, False, False, False, False,  True, True, False,
        True])
>>> np.where(a > 0.5, 1, -1)
array([ 1,  1, -1, -1, -1, -1,  1, 1, -1, 1])
```

np.meshgrid()

Функция *meshgrid()* позволят получить матрицу координат из координатных векторов. Если, например, у нас есть два одномерных вектора координат, то передав их в качестве аргументов в *meshgrid()* мы получим две матрицы, в которой элементы будут составлять пары, заполняя все пространство, определяемое этими векторами. Проще посмотреть это на примере.

Создадим два вектора

```
>>> x = np.linspace(0, 1, 5)
>>> x
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
>>> y = np.linspace(0, 2, 5)
>>> y
array([0. , 0.5, 1. , 1.5, 2. ])
```

Построим матрицу координат с помощью *meshgrid*.

```
>>> xg, yg = np.meshgrid(x, y)
>>> xg
array([[0.  , 0.25, 0.5 , 0.75, 1.  ],
       [0. , 0.25, 0.5 , 0.75, 1.  ],
       [0. , 0.25, 0.5 , 0.75, 1.  ],
       [0. , 0.25, 0.5 , 0.75, 1.  ],
       [0. , 0.25, 0.5 , 0.75, 1.  ]])
>>> yg
array([[0. , 0. , 0. , 0. , 0. ],
       [0.5, 0.5, 0.5, 0.5, 0.5],
       [1. , 1. , 1. , 1. , 1. ],
       [1.5, 1.5, 1.5, 1.5, 1.5],
       [2. , 2. , 2. , 2. , 2. ]])
```

Посмотрите внимательно на матрицы *xg* и *yg*. Каждому элементу *xg[i,j]* соответствует свой элемент *yg[i,j]*. Можно визуализировать эти данные.

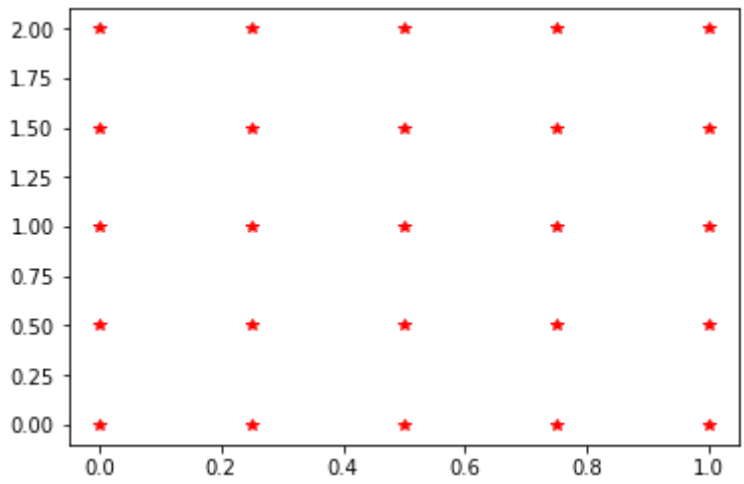
Для начала импортируем *matplotlib* (он должен быть установлен).

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Последняя строка нужна, если вы работаете в *Jupyter Notebook*, чтобы графики рисовались “по месту”.

Теперь построим график

```
plt.plot(xg, yg, color="r", marker="*",  linestyle="none")
```



np.random.permutation()

Функция *permutation()* либо генерирует список заданной длины из натуральных чисел от нуля до указанного числа, либо перемешивает переданный ей в качестве аргумента массив.

```
>>> np.random.permutation(7)
array([6, 2, 5, 1, 0, 4, 3])
>>> a = ['a', 'b', 'c', 'd', 'e']
>>> np.random.permutation(a)
array(['c', 'd', 'a', 'e', 'b'], dtype='<U1')
```

Основное практическое применение эта функция находит в задачах машинного обучения, где довольно часто требуется перемешать выборку данных перед тем, как передавать ее в алгоритм.

Например у нас есть вектор с данными

```
>>> arr = np.linspace(0, 10, 5)
>>> arr
array([ 0. ,  2.5, 5. , 7.5, 10. ])
```

Перемешаем эту выборку

```
>>> arr_mix = np.random.permutation(arr)
>>> arr_mix
array([ 2.5,  0. , 5. , 10. ,  7.5])
```

Построим массив индексов для вектора *arr*, в котором позиции находятся в случайном порядке

```
>>> index_mix = np.random.permutation(len(arr_mix))
>>> index_mix
array([2, 4, 3, 1, 0])
>>> arr[index_mix]
array([ 5. , 10. ,  7.5, 2.5, 0. ])
```

Аппаратура и материалы

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.
2. Операционная система Windows 10.
3. Система контроля версий Git.
4. Браузер для доступа к web-сервису GitHub, рекомендован к использованию Google Chrome.

Указания по технике безопасности

При работе на ЭВМ без разрешения руководителя занятия запрещается:

- подавать (снимать) напряжение на ПЭВМ и электрические розетки с распределительного щита;
- включать и выключать блоки питания ПЭВМ и мониторы;
- извлекать ПЭВМ из защитного кожуха;
- устранять неисправности, возникшие в ходе выполнения лабораторной работы.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный Вами язык программирования (выбор языка программирования будет доступен после установки флажка **Add .gitignore**).
3. Выполните клонирование созданного репозитория на рабочий компьютер.
4. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
5. Дополните файл `.gitignore` необходимыми правилами для выбранного языка программирования, интерактивной оболочки Jupyter notebook и интегрированной среды разработки.

6. Проработать примеры лабораторной работы.
7. Решить задания в ноутбуках, выданных преподавателем.
8. Создать ноутбук, в котором выполнить решение индивидуального задания. Ноутбук должен содержать условие индивидуального задания. При решении индивидуального задания **не должны быть использованы условный оператор *if*, а также операторы циклов *while* и *for*, а только средства библиотеки *NumPy***. Привести в ноутбуке обоснование принятых решений. Номер варианта индивидуального задания необходимо уточнить у преподавателя.
9. Зафиксируйте сделанные изменения в репозитории.
10. Выполните слияние ветки для разработки с веткой *main* (*master*).
11. Отправьте сделанные изменения на сервер GitHub.

Индивидуальное задание

1. Дана целочисленная прямоугольная матрица. Определить:

- количество строк, не содержащих ни одного нулевого элемента;
- максимальное из чисел, встречающихся в заданной матрице более одного раза.

2. Дана целочисленная прямоугольная матрица. Определить количество столбцов, не содержащих ни одного нулевого элемента. Характеристикой строки целочисленной матрицы назовем сумму ее положительных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.
3. Дана целочисленная прямоугольная матрица. Определить:

- количество столбцов, содержащих хотя бы один нулевой элемент;
- номер строки, в которой находится самая длинная серия одинаковых элементов.

4. Дана целочисленная квадратная матрица. Определить:

- произведение элементов в тех строках, которые не содержат отрицательных элементов;
- максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

5. Дана целочисленная квадратная матрица. Определить:

- сумму элементов в тех столбцах, которые не содержат отрицательных элементов;
- минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы.

6. Для заданной матрицы размером 8 на 8 найти такие k, что k-я строка матрицы совпадает с k-м столбцом. Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.
7. Характеристикой столбца целочисленной матрицы назовем сумму модулей его отрицательных нечетных элементов. Переставляя столбцы заданной матрицы, расположить их в соответствий с ростом характеристик. Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.
8. Соседями элемента A_j в матрице назовем элементы A_k с $i-1<k<i+1$, $j-1<1<j+1$, $(k,1)/(i,j)$. Операция сглаживания матрицы дает новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Построить результат сглаживания заданной вещественной матрицы размером 7 на 7. В сглаженной матрице найти сумму модулей элементов, расположенных ниже главной диагонали.
9. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером 10 на 10. Найти сумму модулей элементов, расположенных выше главной диагонали.

10. Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду. Найти количество строк, среднее арифметическое элементов которых меньше заданной величины.
11. Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями. Найти номер первой из строк, содержащих хотя бы один положительный элемент.
12. Осуществить циклический сдвиг элементов прямоугольной матрицы на n элементов вправо или вниз (в зависимости от введенного режима), n может быть больше количества элементов в строке или столбце.
13. Осуществить циклический сдвиг элементов квадратной матрицы размерности $\mathrm{M} \times N$ вправо на к элементов таким образом: элементы 1-й строки сдвигаются в последний столбец сверху вниз, из него - в последнюю строку справа налево, из нее - в первый столбец снизу вверх, из него - в первую строку; для остальных элементов - аналогично.
14. Дана целочисленная прямоугольная матрица. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент. Характеристикой строки целочисленной матрицы назовем сумму ее отрицательных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием характеристик.
15. Упорядочить строки целочисленной прямоугольной матрицы по возрастанию количества одинаковых элементов в каждой строке. Найти номер первого из столбцов, не содержащих ни одного отрица-тельного элемента.
16. Путем перестановки элементов квадратной вещественной матрицы добиться того, чтобы ее максимальный элемент находился в левом верхнем углу, следующий по величине - в позиции (2, 2), следующий по величине - в позиции (3, 3) и т. д., заполнив таким образом всю главную диагональ. Найти номер первой из строк, не содержащих ни одного положительного элемента.
17. Дана целочисленная прямоугольная матрица. Определить:

- количество строк, содержащих хотя бы один нулевой элемент;
- номер столбца, в котором находится самая длинная серия одина-ковых элементов.

18. Дана целочисленная квадратная матрица. Определить:

- сумму элементов в тех строках, которые не содержат отрицательных элементов;
- минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

19. Дана целочисленная прямоугольная матрица. Определить:

- количество положительных элементов в тех строках, которые не содержат нулевых элементов;
- номера строк и столбцов всех седловых точек матрицы.
  Примечание. Матрица A имеет седловую точку A_{ij} если A_{ij} является минимальным элементом в i-й строке и максимальным в j-м столбце.

Содержание отчета и его форма

Отчет по лабораторной работе оформляется письменно в рабочей тетради, должен содержать ответы на контрольные вопросы, ссылку на репозиторий с которым выполнялась работа, скриншоты Jupyter notebook.

Вопросы для защиты работы

1. Каково назначение библиотеки NumPy?
2. Что такое массивы ndarray?
3. Как осуществляется доступ к частям многомерного массива?
4. Как осуществляется расчет статистик по данным?

5. Как выполняется выборка данных из массивов ndarray?