

3.3. Многомерные массивы

Библиотека [NumPy](#) позволяет создавать и обрабатывать двумерные массивы.

Двумерный массив – это упорядоченная совокупность однотипных данных, имеющих общее имя. Обращение к элементам двумерного массива осуществляется по двум индексам. На рисунке 25 массив с именем **a** состоит из трех строк и 4 столбцов, нумерация индексов строк и столбцов начинается с 0. Для обращения к элементу двумерного массива после имени массива указываются квадратные скобки, в которых через запятую сначала указывается индекс строки, а затем индекс столбца.

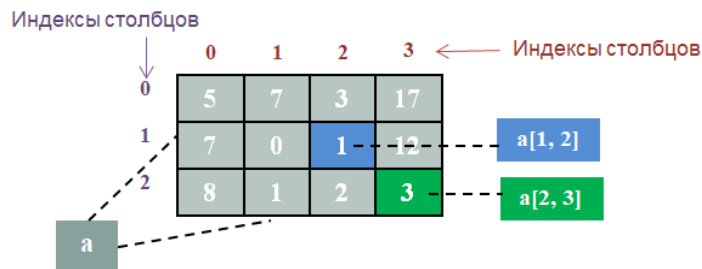


Рисунок 25. Структура двумерного массива

Двумерный массив можно использовать для описания числовых матриц.

3.3.1. Создание двумерного массива

В NumPy существует много способов создать массив. Проще всего создать массив из списка, каждый элемент которого представляет собой новый список. Для этого используется функция **array()**. На основе списка будет создан массив, количество строк которого будет равно количеству вложенных списков, количество столбцов – количеству элементов в каждом.

```
import numpy as np
a = np.array([[3, 4, -2], [-2, -1, 4], [1, 2, 1]])
```

В нашем случае **a** будет представлять собой массив, состоящий из 3 строк и 3 столбцов.

При создании двумерного массива можно переопределить тип его элементов.

```
a = np.array([[ '3', '4', '-2'], [-2, '-1', '4'], [ '1', '2', '1']],
             dtype=int)
```

Другой способ – создать массив на основе данных, введенных пользователем с клавиатуры, по следующему алгоритму:

- сначала определим пустой список, в котором будут накапливаться списки для каждой строки массива, дадим ему имя **array_list**;
- затем в цикле для каждой строки:
 - введем строку, в которой числа разделяются пробелами;
 - элементы каждой строки занесем в список с помощью функции **split()**;
 - созданный список добавляется к списку **array_list**;
- после ввода всех строк, полученный список преобразуется в двумерный массив **numpy**.

В примере вводится двумерный массив, состоящий из трех строк и трех столбцов.

```
import numpy as np
array_list = [ ]

for i in range(3):
    line = input("Введите строку чисел через пробел: ")
    array_str = line.split()
    array_list.append(array_str)

a = np.array(array_list, dtype=int)
```

3.3.2. Вывод массива

Вывод массива осуществляется с помощью оператора **print()**:

```
import numpy as np
a = np.array([[3, 4, -2], [-2, -1, 4], [1, 2, 1]])
print(a)
```

Результат:
[[3 4 -2]
 [-2 -1 4]
 [1 2 1]]

3.3.3. Операции с матрицами

Операции с матрицами реализованы в модулях [numpy](#) и [numpy.linalg](#). Последний необходимо импортировать в начале программы и для простоты обращения к его функциям дать ему имя **alg**.

```
import numpy.linalg as alg
import numpy as np
```

Сложение/вычитание матриц

Данные операции выполняются для матриц одинакового размера **A(n, m)** и **B(n, m)**. Получается матрица размера **n** на **m**, каждый элемент которой является суммой/разностью соответствующих элементов матриц **A** и **B**.

Пример:

```
a = np.array([[3, 4, -1], [-2, 1, 3]])
b = np.array([[1, 0, -5], [2, 2, -1]])

c = a + b

print("a:\n", a)
print("b:\n", b)
print("c:\n", c)
```

Пояснение. В операторе **print()** в строковых константах можно использовать специальные символы, которые используются для оформления вывода. Так **"\n"** означает, что все последующие элементы вывода будут располагаться на новой строке.

Результат:

```
a: [[ 3  4 -1] [-2  1  3]]
b: [[ 1  0 -5] [ 2  2 -1]]
c: [[ 4  4 -6] [ 0  3  2]]
```

Умножение матриц

Операция умножения матрицы **A(n, m)** на матрицу **B(m, k)** выполняется с помощью функции модуля **numpy dot()**. В результате получается матрица размером **n** на **k**. Элемент с индексами **i** и **j** матрицы является скалярным произведением **i**-й строки матрицы **A** и **j**-го столбца матрицы **B** (рисунок 24).

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 7 & 8 \\ 9 & 1 \\ 2 & 3 \end{bmatrix},$$
$$C = A \cdot B,$$
$$C = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 \\ 9 & 1 \\ 2 & 3 \end{bmatrix} =$$
$$= \begin{bmatrix} 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 2 & 1 \cdot 8 + 2 \cdot 1 + 3 \cdot 3 \\ 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 2 & 4 \cdot 8 + 5 \cdot 1 + 6 \cdot 3 \end{bmatrix} = \begin{bmatrix} 31 & 19 \\ 85 & 55 \end{bmatrix}$$

Рисунок 26. Умножение двух матриц

Пример:

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

c = np.dot(a, b)

print("a:\n", a)
print("b:\n", b)
print("c:\n", c)
```

Результат:

```
a: [[1 2 3] [4 5 6]]
b: [[7 8] [9 1] [2 3]]
c: [[31 19] [85 55]]
```

Операции с матрицей и числом

Умножение и другие операции с числом осуществляются поэлементно, то есть операция выполняется для каждого элемента матрицы и числа. Например, при умножении матрицы на число, каждый элемент матрицы умножается на это число.

Пример:

```
a = np.array([[1, 2, 3], [4, 5, 6]])
```

```
c = 5 * a
```

```
print("a:\n", a)
```

```
print("c:\n", c)
```

Результат:

```
a:
[[1 2 3]
 [4 5 6]]
c:
[[ 5 10 15]
 [20 25 30]]
```

Вычисление определителя

Для квадратной матрицы, у которой одинаковое число строк и столбцов, можно вычислить определитель с помощью функции модуля **numpy.linalg det()**.

Пример:

```
a = np.array([[1, 2, 3], [0, 1, -3], [2, 1, 4]])
```

```
det_a = alg.det(a)
```

```
print("a:\n", a)
```

```
print("определитель = ", det_a)
```

Результат:

```
a:
[[ 1  2  3]
 [ 0  1 -3]
 [ 2  1  4]]
определитель = -11.0
```

Обратная матрица

Для квадратной матрицы A можно вычислить обратную матрицу A^{-1} с помощью функции модуля **numpy.linalg inv()**.

Пример:

```
a = np.array([[1, 2, 3], [0, 1, -3], [2, 1, 4]])
```

```
inv_a = alg.inv(a)
```

```
print("a:\n", a)
```

```
print("inv_a:\n", inv_a)
```

Результат:

```
a:
[[ 1  2  3]
 [ 0  1 -3]
 [ 2  1  4]]
inv_a:
[[-0.63636364  0.45454545  0.81818182]
 [ 0.54545455  0.18181818 -0.27272727]
 [ 0.18181818 -0.27272727 -0.09090909]]
```

Задание для самопроверки

Дан фрагмент программы:

```
import numpy as np
import numpy.linalg as alg
```

```
a = np.array([[3, 4, -2], [-2, -1, 4]])
b = np.array([[1, -3, -2, 1], [2, 4, -2, -1], [5, -2, 0, -2]])
c = np.array([[-1, 0, 2], [2, -2, 3], [0, 5, 1]])
```

Какие из следующих операторов записаны неверно? Объясните, в чем ошибки.

- `d = np.dot(a, b)`
- `d = np.dot(c, c)`
- `det_array = alg.det(b)`
- `d = 12 - b`
- `d = a + b`
- `d = np.dot(c, b)`

3.3.4. Срезы

Для обращения к частям матрицы используются индексы и срезы. С их помощью можно выделить:

- элемент;
- строку;
- столбец;
- прямоугольный блок элементов.

Рассмотрим матрицу:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

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

В таблице 20 приведены примеры выделения различных частей матрицы **a**.

Таблица 20 – Описание срезов и индексов

Срез/индекс	Результат	Описание
<code>a[1,2]</code>	6	Элемент массива a , расположенный на пересечении строки с индексом 1 и столбца с индексом 2 (нумерация индексов начинается с 0), результат – число
<code>a[2]</code>	<code>[7 8 9]</code>	Строка с номером 2 массива a , результат – одномерный массив
<code>a[:,1]</code>	<code>[2 5 8]</code>	Столбец с номером 1 массива a , результат – одномерный массив
<code>a[:2]</code>	<code>[[1, 2, 3], [4, 5, 6]]</code>	Первые две строки матрицы a , результат – двумерный массив
<code>a[1:3]</code>	<code>[[4, 5, 6], [7, 8, 9]]</code>	Строки матрицы a с номерами 1 и 2, результат – двумерный массив
<code>a[:, :2]</code>	<code>[[1, 2], [4, 5], [7, 8]]</code>	Первые два столбца матрицы a , результат – двумерный массив
<code>a[:, 1:3]</code>	<code>[[2, 3], [5, 6], [8, 9]]</code>	Столбцы матрицы a с номерами 1 и 2, результат – двумерный массив
<code>a[0:2, 1:3]</code>	<code>[[2, 3], [5, 6]]</code>	Блок матрицы a , результат – двумерный массив

Для обработки отдельных частей двумерных массивов используются функции модуля **numpy**, приведенные в таблице 21.

Пример. Координаты точек на плоскости представлены в виде двумерного массива, каждая строка которого - координаты одной точки в декартовой системе координат. Найти «среднюю точку», координаты которой вычисляются как средние значения координат **x** и **y** всех точек. Показать точки на плоскости.

Таблица 21 – Функции модуля numpy

Функция	Описание
<code>max(a, 0)</code>	Находит максимальные элементы в каждом столбце матрицы a , результат – вектор, размерность которого равна количеству столбцов матрицы
<code>max(a, 1)</code>	Находит максимальные элементы в каждой строке матрицы a , результат – вектор, размерность которого равна количеству строк матрицы
<code>min(a, 0)</code>	Находит минимальные элементы в каждом столбце матрицы a , результат – вектор, размерность которого равна количеству столбцов матрицы
<code>min(a, 1)</code>	Находит минимальные элементы в каждой строке матрицы a , результат – вектор, размерность которого равна количеству строк матрицы
<code>mean(a, 0)</code>	Находит среднее значение каждого столбца матрицы a , результат – вектор, размерность которого равна количеству столбцов матрицы
<code>mean(a, 1)</code>	Находит среднее значение каждой строки матрицы a , результат – вектор, размерность которого равна количеству строк матрицы
<code>sum(a, 0)</code>	Находит сумму каждого столбца матрицы a , результат – вектор, размерность которого равна количеству столбцов матрицы
<code>sum(a, 1)</code>	Находит сумму каждой строки матрицы a , результат – вектор, размерность которого равна количеству строк матрицы
<code>prod(a, 0)</code>	Находит произведение каждого столбца матрицы a , результат – вектор, размерность которого равна количеству столбцов матрицы
<code>prod(a, 1)</code>	Находит произведение каждой строки матрицы a , результат – вектор, размерность которого равна количеству строк матрицы
<code>diag(a)</code>	Находит диагональные элементы квадратной матрицы, результат – вектор, размерностью равной количеству строк (или столбцов) матрицы a
<code>triu(a, i)</code>	Выделяет треугольную матрицу из матрицы a , i – целое число, которое означает «сдвиг» стороны (гипотенузы) треугольника относительно главной диагонали, результат – матрица, равная по размеру матрице a , элементы, не относящиеся к «треугольнику», заполняются нулями

Код программы, пояснения приведены в комментариях:

```
import numpy as np
import matplotlib.pyplot as plt

#задаем координаты точек
points = np.array([[1, 3], [3, -2], [2, 3], [-3, 4], [-2, -1]])

# вычисляем координаты средней точки
mean_point = np.mean(points, 0)
print("Средняя точка: ", np.round(mean_point, 2))

# выводим точки на график
plt.plot(points[:, 0], points[:, 1], "bo")
plt.plot(mean_point[0], mean_point[1], "ro")

# устанавливаем оси
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

plt.show()

Результат:
Средняя точка:  [0.2 1.4]
```

Изображение точек на плоскости, полученное в результате выполнения программы, показано на рисунке 27.

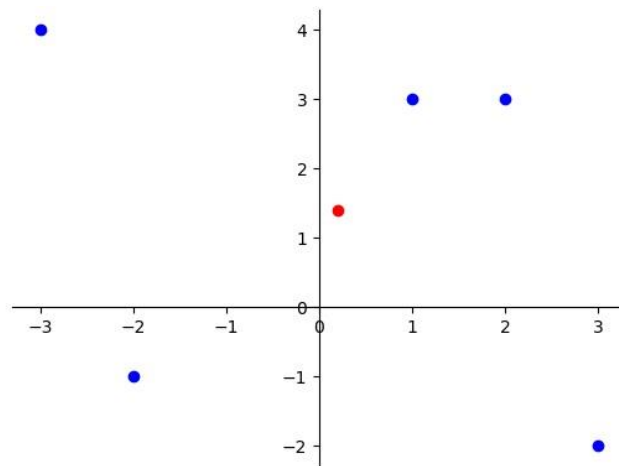


Рисунок 27. Результат выполнения программы

Задание для самопроверки

Дан фрагмент программы:

```
import numpy as np  
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

Что будет напечатано в консоли после выполнения каждого оператора:

- `print(b[:, 2])`
- `print(b[:2])`
- `print(b[2])`
- `print(b[:2, 2])`
- `print(b[:2,:2])`
- `print(np.sum(b, 1))`
- `print(np.max(b, 0))`