Базовые методы обработки данных с использованием Python

Лекция 3. Работа с модулем Numpy в Python. Векторные и матричные вычисления.

Киреев Василий Сергеевич

к.т.н., доцент

Москва, 2024

Пакет Numpy

Python NumPy - это пакет обработки массивов общего назначения, предоставляющий средства для работы с n-мерными массивами. Он предоставляет различные вычислительные инструменты, такие как комплексные математические функции, процедуры линейной алгебры. NumPy обладает гибкостью языка Python и скоростью хорошо оптимизированного компилированного кода на языке C. Простой в использовании синтаксис делает его очень доступным и продуктивным для программистов с любым уровнем подготовки.

Пакет Numpy. Предназначение

- Арифметические операции
- Статистические операции
- Побитовые операторы
- Линейная алгебра
- Копирование и просмотр массивов
- Объединение
- Поиск, сортировка, подсчет
- Математические операции
- Автоматическое расширение размерности
- Матричные операции

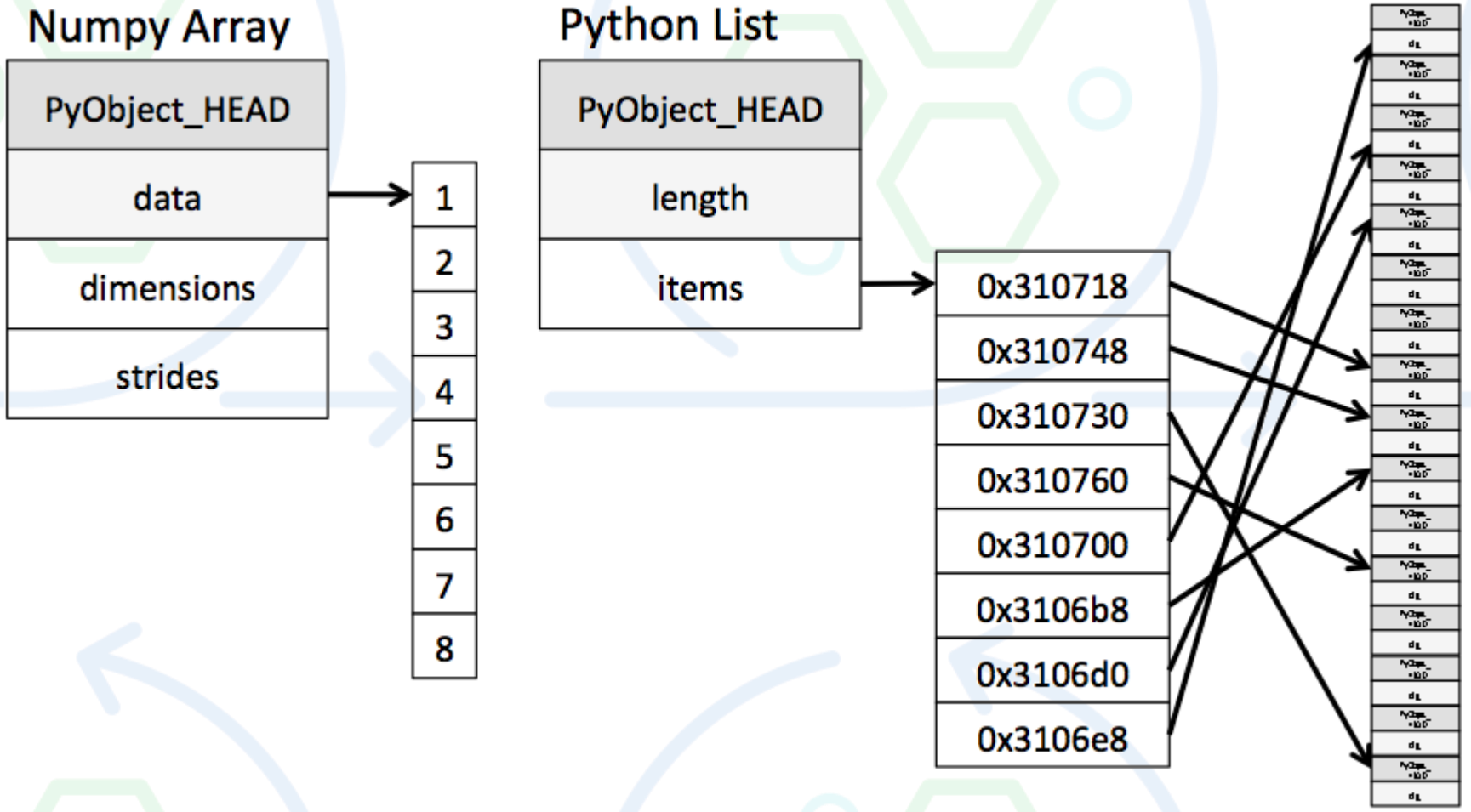
Массивы Numpy

Разница со списками заключается в том, что массивы Numpy однородны, что облегчает работу с ними. Мы можем инициализировать элементы массива различными способами, одним из которых является использование списков Python.

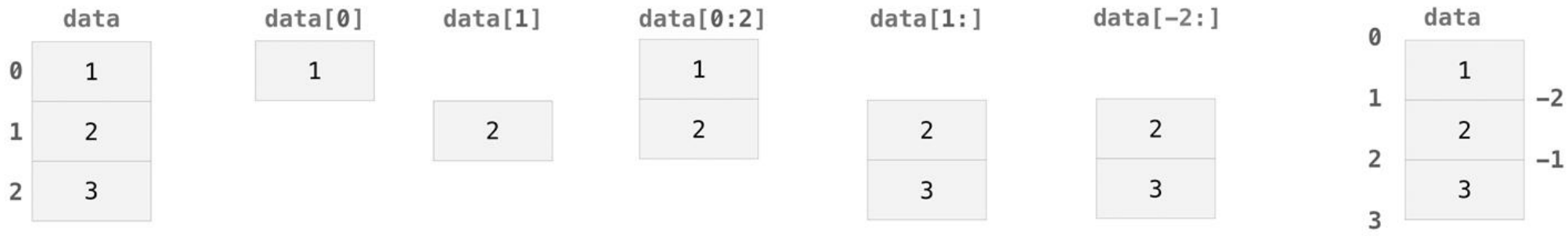
Массивы Numpy удобны тем, что обладают следующими тремя свойствами:

- Меньшая потребность в памяти
- Более быстрая обработка
- Удобство использования

Массивы Numpy. Структура



Индексы и срезы массива



Объединение массивов

```
1 a1 = np.array([[1, 1],
2                [2, 2]])
3
4 a2 = np.array([[3, 3],
5                [4, 4]])
```

```
1 np.vstack((a1, a2))
```

```
array([[1, 1],
       [2, 2],
       [3, 3],
       [4, 4]])
```

```
1 np.hstack((a1, a2))
```

```
array([[1, 1, 3, 3],
       [2, 2, 4, 4]])
```

Разделение массивов

```
1 x = np.arange(0, 6).reshape(3, 2)
2 x
```

```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

```
1 np.hsplit(x, 2)
```

```
[array([[0],
        [2],
        [4]]), array([[1],
        [3],
        [5]])]
```

Операции с векторами в Numpy

Создание плоских массивов

Создание плоских массивов:

```
1 x = np.array([[1 , 2, 3], [5, 6, 7], [10, 11, 12]])
2 x.flatten()
```

```
array([ 1,  2,  3,  5,  6,  7, 10, 11, 12])
```

Векторное умножение

$$a = [a_1, a_2, \ldots, a_N]^T$$
$$b = [b_1, b_2, \ldots, b_N]^T$$

$$[a_1, a_2, \ldots, a_N] \cdot \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_N \end{bmatrix} = \sum_{i=1}^{N} a_i b_i$$

$$\begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_N \end{bmatrix} \cdot [b_1, b_2, \ldots, b_N] = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_N \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_N \\ . & . & . & . \\ a_N b_1 & a_N b_2 & \cdots & a_N b_N \end{bmatrix}$$

Норма вектора

$$a = [a_1, a_2, \ldots, a_N]^T$$
$$b = [b_1, b_2, \ldots, b_N]^T$$

$$[a_1, a_2, \ldots, a_N] \cdot \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_N \end{bmatrix} = \sum_{i=1}^{N} a_i b_i$$

$$\begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_N \end{bmatrix} \cdot [b_1, b_2, \ldots, b_N] = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_N \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_N \\ . & . & . & . \\ a_N b_1 & a_N b_2 & \cdots & a_N b_N \end{bmatrix}$$

Векторное умножение в numpy. Примеры

```
1 x = np.array([[1 , 2, 3], [5, 6, 7], [10, 11, 12]])
2 x.flatten()
```

```
array([ 1,  2,  3,  5,  6,  7, 10, 11, 12])
```

```
1 np.dot(a1, a2)  #скалярное произведение массивов
```

```
array([[ 7,  7],
       [14, 14]])
```

```
1 np.vdot(a1, a2) #векторное произведение векторов
```

```
22
```

```
1 np.linalg.norm(a1) #норма вектора
```

```
3.1622776601683795
```

Матрицы

Матрицами называются массивы элементов, представленные в виде прямоугольных таблиц, для которых определены правила математических действий. Элементами матрицы могут являться числа, алгебраические символы или математические функции.

Матричная алгебра имеет обширные применения в различных отраслях знания – в математике, физике, информатике, экономике. Например, матрицы используется для решения систем алгебраических и дифференциальных уравнений, нахождения значений физических величин в квантовой теории, шифрования сообщений в Интернете.

Матрицы. Математическая запись

$$A_{m\times n} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Матрицы Numpy. Функция zeros. Пример на Python

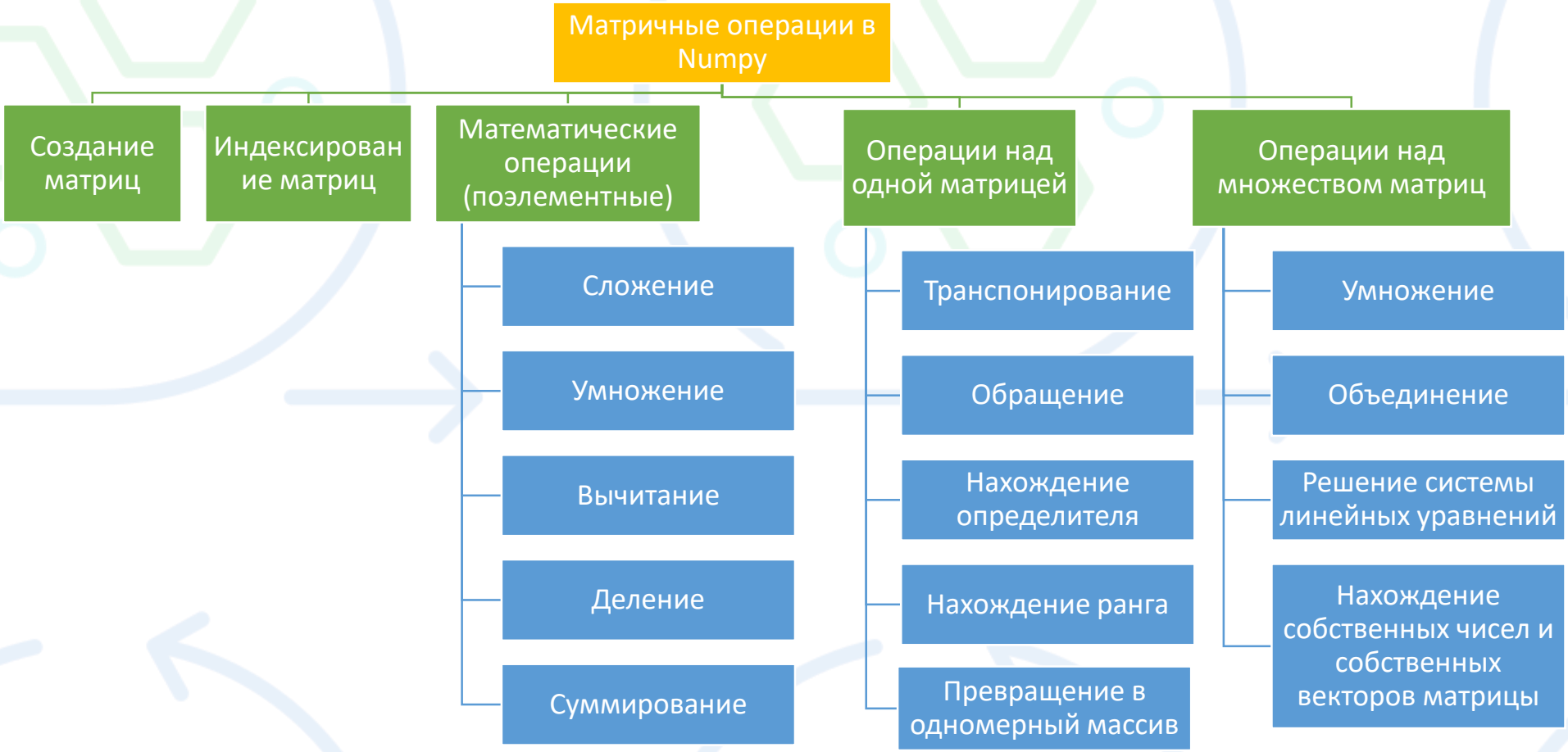
Нулевой массив или нулевая матрица создаются в Numpy с помощью функции zeros(). В системе матриц, 0-матрица обладает теми же свойствами, что и обычный нуль.

```
import numpy as np

np.zeros((3, 6))
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

Операции с матрицами в Numpy



Базовые операции с матрицами в Numpy

Функция	Описание
array()	создает матрицу
dot()	выполняет умножение матриц
transpose()	транспонирует матрицу
linalg.inv()	вычисляет обратную матрицу
linalg.det()	вычисляет определитель матрицы
flatten()	преобразует матрицу в одномерный массив

Создание матрицы в numpy. Пример на Python

В Python матрица может быть реализована в виде двумерного списка или двумерного массива. Формирование матрицы из последнего дает дополнительные функциональные возможности для выполнения различных операций над матрицей.

```
import numpy

x = numpy.array([[1, 2], [4, 5]])

print(x)
```

```
[[1 2]
 [4 5]]
```

Индексирование матриц в numpy. Пример на Python

Для получения элементов матрицы можно использовать несколько способов. Элемент на пересечении строки **i** и столбца **j** можно получить с помощью выражения array[i, j]. Из матрицы можно получать целые строки или столбцы с помощью выражений array[i, :] или array[:, j] соответственно.

```
print ("Вторая строка матрицы d:\n", x[1, :])
print ("второй столбец матрицы d:\n", x[:, 1])
```

```
Вторая строка матрицы d:
 [4 5]
Второй столбец матрицы d:
 [2 5]
```

Транспонирование матрицы

Транспонированной матрицей A^T называется матрица, полученная из исходной матрицы A заменой строк на столбцы. Формально: элементы матрицы A^T определяются как $a^T_{ij} = a_{ji}$, где a^T_{ij} — элемент матрицы A^T, стоящий на пересечении строки с номером i и столбца с номером j.

Транспонирование матрицы. Математическая запись

$$A^T = \begin{bmatrix} 1 & 3 & 2 \end{bmatrix}, \; B^T = \begin{bmatrix} 5 \\ 2 \\ 6 \end{bmatrix}, \; C^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Обращение матрицы

Обратная матрица - это просто обратная матрица, как это делается в обычной арифметике для единичного числа, которая используется для решения уравнений с целью нахождения значений неизвестных переменных. Обратной матрицей называется такая матрица, которая при умножении на исходную матрицу дает матрицу тождеств.

Обратная матрица существует только в том случае, если матрица несингулярна, т.е. ее определитель не должен быть равен 0. Используя определитель и смежность, мы можем легко найти обратную квадратную матрицу.

Обращение матрицы. Математическая запись

$$A^{-1}A = I = AA^{-1}$$

Обращение матрицы в numpy. Пример на Python

С помощью функции numpy.linalg.inv(array) можно обратить любую квадратную матрицу.

```
import numpy as np

arr = np.array([[1, 2], [5, 6]])
inverse_array = np.linalg.inv(arr)
print("Инверсный массив - это ")
print(inverse_array)
```

```
Инверсный массив - это
[[-1.5   0.5 ]
 [ 1.25 -0.25]]
```

Поэлементное сложение матриц

Обратная матрица - это просто обратная матрица, как это делается в обычной арифметике для единичного числа, которая используется для решения уравнений с целью нахождения значений неизвестных переменных. Обратной матрицей называется такая матрица, которая при умножении на исходную матрицу дает матрицу тождеств.

Обратная матрица существует только в том случае, если матрица несингулярна, т.е. ее определитель не должен быть равен 0. Используя определитель и смежность, мы можем легко найти обратную квадратную матрицу.

Поэлементное сложение матриц. Математическая запись

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & & \vdots \\ a_{p1} + b_{p1} & a_{p2} + b_{p2} & \cdots & a_{pn} + b_{pn} \end{bmatrix}$$

Умножение матриц. Математическая запись

$$[\mathbf{AB}]_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \cdots + a_{i,n}b_{n,j} = \sum_{r=1}^{n} a_{i,r}b_{r,j},$$

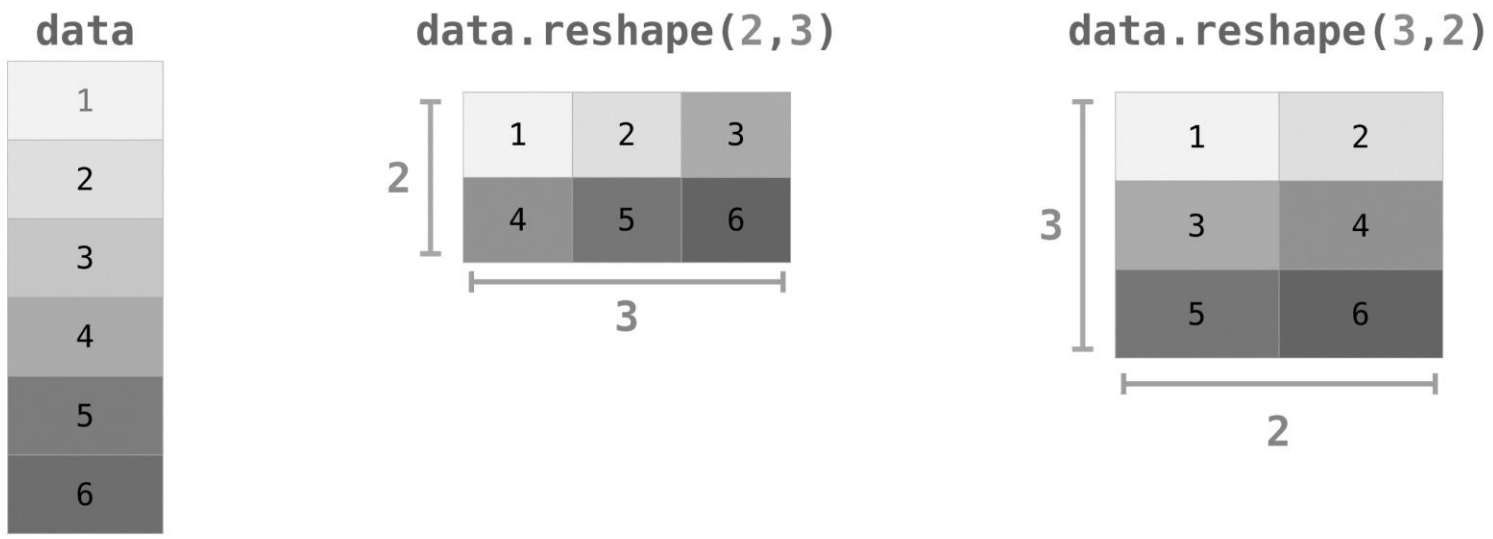
$$\begin{bmatrix} \underline{2} & \underline{3} & \underline{4} \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & \underline{1000} \\ 1 & \underline{100} \\ 0 & \underline{10} \end{bmatrix} = \begin{bmatrix} 3 & \underline{2340} \\ 0 & 1000 \end{bmatrix}$$

Автоматическое расширение размерности

Бродкастинг (broadcasting) – автоматическое расширение размерности (ndim) и размеров (shape) массивов, при совершении операций (сложение, умножение и подобные) над массивами с разными размерами или размерностями, при условии, что они совместимы с правилами бродкастинга.

Преобразование формы матриц

Преобразование формы матриц:



Линейная алгебра в numpy

Функция	Описание
linalg.cholesky()	Разложение Холецкого
linalg.qr()	QR-разложение матрицы
linalg.svd()	Сингулярное (SVD) разложение матрицы
linalg.norm()	Норма матрицы или вектора
linalg.cond()	Число обусловленности матрицы
linalg.det()	Определитель (детерминант) матрицы
linalg.matrix_rank()	Вычисление ранга матрицы по алгоритму SVD
np.trace()	Сумма диагональных элементов массива
linalg.eig()	Вычисление собственных значений и правых собственных векторов
linalg.eigvals()	Вычисление собственных значений матрицы

Линейная алгебра в numpy

Функция	Описание
linalg.solve()	Решение линейного матричного уравнения
linalg.tensorsolve()	Решение линейного тензорного уравнения
linalg.lstsq()	Решает задачу поиска наименьших квадратов для линейного матричного уравнения
linalg.inv()	Вычисление обратной матрицы
linalg.pinv()	Вычисление псевдообратной (Мура-Пенроуза) матрицы
linalg.tensorinv()	Вычисление обратного тензора (N-мерного массива)

Собственные значения и собственные векторы

Собственные значения и собственные векторы - это скалярные и векторные величины, связанные с матрицей, используемой для линейных преобразований. Вектор, который не изменяется даже после применения преобразований, называется собственным вектором, а скалярная величина, связанная с собственными векторами, - собственными значениями.

Собственные векторы - это векторы, которые связаны с набором линейных уравнений. Для матрицы собственные векторы также называются характеристическими векторами, и мы можем найти собственный вектор только квадратной матрицы. Собственные векторы очень полезны при решении различных задач о матрицах и дифференциальных уравнениях.

Собственные значения и собственные векторы в numpy. Пример на Python

В NumPy мы можем вычислить собственные значения и правые собственные векторы заданного квадратного массива с помощью функции numpy.linalg.eig(). Она принимает в качестве параметра квадратный массив и возвращает два значения: первое - собственные значения массива, второе - правые собственные векторы заданного квадратного массива.

```
import numpy as np
mat = np.mat("1 2;1 3")
print(mat)
print("")
evalue, evect = np.linalg.eig(mat)
print(evalue)
print("")
print(evect)
```

```
[[1 2]
 [1 3]]

[0.26794919 3.73205081]

[[-0.9390708  -0.59069049]
 [ 0.34372377 -0.80689822]]
```

Определитель матриц в numpy. Пример на Python

Numpy предоставляет нам возможность вычислить определитель квадратной матрицы с помощью функции numpy.linalg.det()

```
import numpy as np

n_array = np.array([[50, 29], [30, 44]])
print('Матрица Numpy:')
print(n_array)
det = np.linalg.det(n_array)
print('Детерминант данной матрицы 2X2')
print(int(det))
```

```
Матрица Numpy:
[[50 29]
 [30 44]]
Детерминант данной матрицы 2X2
1330
```

Поиск значений в массиве numpy. Пример на Python

Numpy предоставляет различные методы поиска различных видов числовых значений. Метод numpy.where() возвращает индексы элементов входного массива, для которых выполняется заданное условие.

```
import numpy as np

arr = np.array([10, 32, 30, 50, 20, 82, 91, 45])
print('arr = {}'.format(arr))
i = np.where(arr == 30)
print('i = {}'.format(i))
```

```
arr = [10 32 30 50 20 82 91 45]
i = (array([2]),)
```

Многомерная сортировка значений в массиве numpy. Пример на Python

С помощью метода Numpy matrix.argmax() мы можем найти сортировку элементов в заданной матрице, имеющей одно или несколько измерений, и он вернет значение индекса отсортированных элементов.

```
import numpy as np

gfg = np.matrix('[1, -2, 3, -4]')
gks = gfg.argsort()
print(gks)
```

```
[[3 1 0 2]]
```

Спасибо за внимание!