

МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт математики, механики и компьютерных наук им. И. И. Воровича
Кафедра математического моделирования

«Проект 2 курс»

Мелехов Андрей Петрович

Задание 3. Пакет numpy

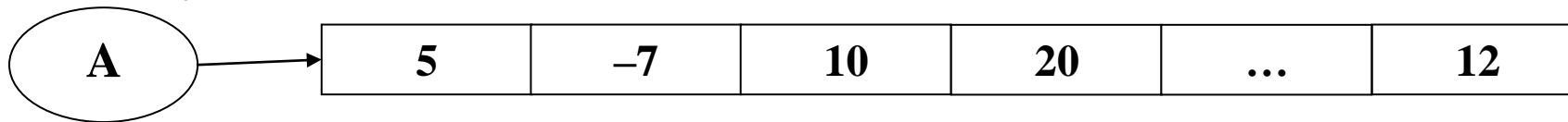
Ростов-на-Дону

2021

Библиотека Numeric Python (NumPy)

Для быстрой работы с большими данными используются массивы из пакета numpy (см. www.numpy.org).

Используются массивы (array) элементов одинакового типа:



Размер задается при создании, нельзя поменять. Можно менять значения. Можно создавать одномерные (вектора), двумерные (матрицы) и большей размерности массивы. Все элементы массива одного типа (например, все целые или вещественные). Быстрый доступ к элементам. Пакет содержит кроме самих массивов еще и методы для работы с ними. Быстрая обработка данных в массиве, особенно, если использовать методы пакетов numpy, scipy, pandas, и не использовать циклы и методы базового пакета Python.

Функции создания массивов

<https://numpy.org/doc/stable/reference/routines.array-creation.html> и <https://numpy.org/doc/stable/user/basics.creation.html#arrays-creation>

Функция	Описание
array(object, dtype=None, ...)	Создание массива из объекта object
arange([start,]stop, [step,] dtype=None)	Создание массива из диапазона
linspace(start, stop, num=50, ...)	Создание массива num чисел, равномерно расположенных на отрезке [start, stop]
zeros(shape, dtype = float, order = 'C')	Возвращает новый массив заданной формы и типа (по умолчанию 'float64'), заполненный нулями
empty(shape, ...)	новый массив (не заполнен)
ones(shape, ...)	новый массив из 1
full(shape, fill_value, dtype = None, order = 'C')	новый массив, заполненный заданным значением

empty_like, full_like,
ones_like, zeros_like

новый массив (размер и тип по образцу другого массива)

Пример. Создание массивов

```
# Подключаем пакет numpy:
import numpy as np
# Создание массива из списка:
A = np.array([1, 4, 2, 5, 3])
# Создание двумерного массива из списка:
np.array([[i]*3 for i in [2, 4, 6]])
# Результат: array([[2, 2, 2], [4, 4, 4], [6, 6, 6]])

# Создание массивов, заполненных нулями:
np.zeros(5) # числа по умолчанию вещественные
# array([ 0.,  0.,  0.,  0.,  0.])
np.zeros((5,), dtype = int)
# array([0, 0, 0, 0, 0])
```

```
np.empty( (2, 1) ) # без заполнения
# array([[ 8.46612366e-83], [-1.84402114e-26]])
np.ones( (2, 2) ) # заполненного единицами
# array([[ 1.,  1.], [ 1.,  1.]])

np.arange(0, 10, 2) # диапазон
# array([0, 2, 4, 6, 8])
np.linspace(0, 1, 5) # 5 чисел на отрезке [0, 1]
array([0., 0.25, 0.5, 0.75, 1.])
# случайные числа:
np.random.random( (2, 3) ) # вещественные
# array([[0.61646814, 0.57419779, 0.1674393 ],
#        [0.92308608, 0.69372043, 0.24111806]])
np.random.randint(0, 10, (3, 3)) # целые
# array([[9, 7, 4], [1, 7, 2], [1, 6, 6]])
```

Стандартные типы данных библиотеки NumPy

<https://numpy.org/doc/stable/reference/arrays.scalars.html>

Тип данных	Описание
bool_	Булев тип (True или False), 1 байт в памяти
int_	Целочисленное значение по умолчанию (аналогичен типу long языка C; обычно int64 или int32)
intc	Идентичен типу int языка C (обычно int32 или int64)
intp	Целочисленное значение, используемое для индексов (аналогично типу ssize_t языка C; обычно int32 или int64)
int8	Байтовый тип (от -128 до 127)
int16	Целое число (от -32 768 до 32 767)
int32	Целое число (от -2 147 483 648 до 2 147 483 647)
int64	Целое число (от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807)
uint8	Беззнаковое целое число (от 0 до 255)

uint16	Беззнаковое целое число (от 0 до 65 535)
uint32	Беззнаковое целое число (от 0 до 4 294 967 295)
uint64	Беззнаковое целое число (от 0 до 18 446 744 073 709 551 615)
float_	Сокращение для названия типа float64
float16	Число с плавающей точкой с половинной точностью: 1 бит знак, 5 бит порядок, 10 бит мантисса
float32	Число с плавающей точкой с одинарной точностью: 1 бит знак, 8 бит порядок, 23 бита мантисса
float64	Число с плавающей точкой с удвоенной точностью: 1 бит знак, 11 бит порядок, 52 бита мантисса
complex_	Сокращение для названия типа complex128
complex64	Комплексное число, представленное двумя 32-битными числами
complex128	Комплексное число, представленное двумя 64-битными числами

```
# Создаем массив целых чисел:
b = np.zeros(10, dtype = 'int32')
b.dtype      #          dtype('int32')
c = np.zeros(10)
c.dtype      # По умолчанию тип float64
#           dtype('float64')
```

Атрибуты массивов библиотеки NumPy

```
# двумерный массив:
x2 = np.random.randint(10, size = (3, 4))
x2.ndim      # 2 - размерность
x2.shape     # (3, 4) - размер каждого измерения
x2.size      # 12 - количество элементов
x2.itemsize  # 4 - байт на один элемент
x2.nbytes    # 48 - байт на весь массив = 12 * 4
```


Индексация массива: срезы

Можно использовать срезы. Одномерные массивы:

```
x1 = np.arange(10)
# array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x1[0]      # 0
x1[1:5]    # array([1, 2, 3, 4])
x1[-1]     # 9
x1[::-1]   # array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Многомерные массивы:

```
x2 = np.arange(12)
x2 = x2.reshape((3, 4))
# array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]])
x2[1, 3]    # 7 индексы задаются через x2[i, j]
# В списках не так: L[i][j]
x2[:2, ::2] # Строки 0, 1 и столбцы 0, 2:
# array([[0, 2],
#        [4, 6]])
```

```
x2[:, 0]      # первый столбец массива x2
              array([0, 4, 8])
x2[0, :]      # первая строка массива x2
              array([0, 1, 2, 3])
```

Срезы массивов возвращают представления (views), а не копии (copies) данных массива. Этим срезы массивов библиотеки NumPy отличаются от срезов списков языка Python.

```
x3 = x2[:2, ::2]  # извлекаем подмассив из x2
# array([[0, 2], [4, 6]])
x3[:, :] = 55     # заменить все элементы в x3 на 55
x2              # массив x2 тоже поменялся:
# array([[55, 1, 55, 3],
#        [55, 5, 55, 7],
#        [ 8, 9, 10, 11]])
```

Срез x3 продолжает ссылаться на x2 (не создается новый массив). Представление – это работа с теми же данными, но новые индексы.

Создание копий массивов

```
# Если надо создать копию, вызываем метод copy:  
x2_copy = x2[:2, ::2].copy()
```

Изменение формы массивов

```
x = np.arange(1, 10).reshape((3, 3))  
# array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
# Преобразование в столбец:  
y = x.reshape((9, 1))  
# y = array([[1], [2], [3], [4], [5], [6], [7], [8], [9]])
```

Добавление новой размерности

```
x = np.arange(1, 10)
# array([1, 2, 3, 4, 5, 6, 7, 8, 9])
# Преобразование из строки в столбец:
# Из массива делаем матрицу размера (n, 1):
z = x[:, np.newaxis]
# и старые данные записываются в разные строки:
# z = array([[1],[2],[3],[4],[5],[6],[7],[8],[9]])
```

Это тоже представления, т.е. массивы продолжают ссылаться на одни и те же данные! Изменяя данные в `u` или `z`, меняем и `x`.

Слияние и разбиение массивов

Объединение: `concatenate((a1, a2, ...), axis=0, out=None, dtype=None)`

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
# Вдоль первой оси (axis = 0):
np.concatenate((a, b), axis = 0)
```

```
#         array([[1, 2], [3, 4], [5, 6]])
# Вдоль второй оси (axis = 1):
# b.T – транспонирование (переворот):
np.concatenate((a, b.T), axis = 1)
#         array([[1, 2, 5], [3, 4, 6]])
# Если axis = None, массивы преобраз. в одномерные:
np.concatenate((a, b), axis = None)
#         array([1, 2, 3, 4, 5, 6])

a = np.array([1, 2, 3])
b = np.array([2, 3, 4])
# np.vstack (вертикальное объединение):
np.vstack((a, b))
#         array([[1, 2, 3], [2, 3, 4]])
# np.hstack (горизонтальное объединение):
np.hstack((a, b))
```

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

Разбиение массивов выполняется с помощью функций: `np.split`, `np.hsplit`, `np.vsplit`.

```
a = np.array([10, 20, 30, 40, 50])  
# Разбить по 2-му элементу:  
np.split(a, [2])  
#          [array([10, 20]), array([30, 40, 50])]
```

Выполнение вычислений над массивами библиотеки

NumPy: универсальные функции

<https://numpy.org/doc/stable/reference/ufuncs.html#ufuncs>

С массивами пакета numpy циклы работают медленно. Лучше использовать векторизованные операции и функции из numpy.

Операции: +, −, *, /, //, %, **.

Операция	Функция	Описание
+	<code>np.add</code>	Сложение
−	<code>np.subtract</code>	Вычитание
−	<code>np.negative</code>	Унарная операция изменения знака
*	<code>np.multiply</code>	Умножение
/	<code>np.divide</code>	Деление
//	<code>np.floor_divide</code>	Деление с остатком
**	<code>np.power</code>	Возведение в степень
%	<code>np.mod</code>	Остаток

Примеры.

```
# Определяем два массива:
x = np.array([1, 2, 3])
y = np.array([2, 4, 8])
# Операции (поэлементно с одинаковыми массивами) :
# Сумма через операцию или функцию:
x + y          # array([ 3,  6, 11])
np.add(x, y)   # array([ 3,  6, 11])
x - y          # array([-1, -2, -5])
-x            # array([-1, -2, -3])
x * y          # array([ 2,  8, 24])
x / y          # array([0.5 , 0.5 , 0.375])
x // y         # array([0, 0, 0], dtype=int32)
x % y          # array([1, 2, 3], dtype=int32)
# Операции (поэлементно массив и число) :
x + 5          # array([6, 7, 8])
x ** 2         # array([1, 4, 9], dtype=int32)
```


Функции: absolute (или abs), exp, log, sqrt, тригонометрические (sin, cos, tan, arcsin, arcos, arctan) и др.

```
# Определяем массив:
```

```
x = np.array([1, 2, 4])
```

```
# Вычисляем функции:
```

```
np.sqrt(x)          # array([1., 1.41421356, 2.])
```

```
np.sin(x)           # array([ 0.84147098, 0.90929743,  
                             -0.7568025  ])
```

Линейная алгебра

Операции линейной алгебры – умножение и разложение матриц, вычисление определителей и другие – важная часть любой библиотеки для работы с массивами. В отличие от некоторых пакетов, например, MATLAB, в NumPy применение оператора * к двум двумерным массивам вычисляет поэлементное, а не матричное произведение. А для перемножения матриц имеется функция dot:

```

x = np.array([[1, 2], [3, 4]])
y = np.array([[5, -6], [-1, 7]])
# Умножение матриц:
z = np.dot(x, y)
# или z = x.dot(y)
# z = [[ 3  8], [11 10]]

```

$$z = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 5 & -6 \\ -1 & 7 \end{pmatrix} = \begin{pmatrix} 3 & 8 \\ 11 & 10 \end{pmatrix}.$$

В модуле `numpy.linalg` имеется стандартный набор алгоритмов, в частности, разложение матриц, нахождение обратной матрицы и вычисление определителя.

```

A = np.array([[1, 2], [3, 0]])
# Обратная матрица к A:
B = np.linalg.inv(A)
# [[ 0.  0.33333333] [ 0.5 -0.16666667]]
# Произведение матриц A * A-1 = E:
np.dot(A, B) # [[1. 0.] [0. 1.]]

```

Некоторые функции из модуля `numpy.linalg`
numpy.org/doc/stable/reference/routines.linalg.html?highlight=linalg#module-numpy.linalg

Функция	Описание
trace	Вычисляет след матрицы – сумму диагональных элементов
det	Вычисляет определитель матрицы
eig	Вычисляет собственные значения и собственные векторы квадратной матрицы
eigvals	Вычисляет собственные значения квадратной матрицы
inv	Вычисляет обратную матрицу
norm	Вычисляет норму матрицы или вектора
qr	Вычисляет QR-разложение
svd	Вычисляет сингулярное разложение (SVD)
solve	Решает линейную систему $Ax = b$, где A – квадратная матрица
lstsq	Вычисляет решение уравнения $y = Xb$ по методу наименьших квадратов

Сводные показатели (reduce, accumulate)

Операция `reduce` применяет операцию к элементам массива до тех пор, пока не останется только один результат:

```
x = np.array([1, 2, 4])  
np.add.reduce(x)      # 7   = 1 + 2 + 4
```

Функция `accumulate` сохраняет все промежуточные результаты:

```
np.multiply.accumulate(x)  
#          array([1, 2, 8], dtype = int32)
```

Внешнее произведение двух векторов

```
x = np.array([1, 2, 4])  
y = np.array([0, 3, 7])  
# Произведение outer (каждый с каждым):  
np.multiply.outer(x, y)  
array([[ 0,  3,  7],  
       [ 0,  6, 14],  
       [ 0, 12, 28]])
```

```
# Сумма внешняя outhter (каждый с каждым) :  
np.add.outer(x, y)  
array([[ 1,  4,  8],  
       [ 2,  5,  9],  
       [ 4,  7, 11]])
```

Агрегирование: минимум, максимум и др.

<https://numpy.org/doc/stable/reference/routines.statistics.html#averages-and-variances>

Функция	NaN-безопасная версия	Описание
np.sum	np.nansum	Вычисляет сумму элементов
np.prod	np.nanprod	Вычисляет произведение элементов
np.mean	np.nanmean	Вычисляет среднее значение элементов
np.std	np.nanstd	Вычисляет стандартное отклонение
np.var	np.nanvar	Вычисляет дисперсию
np.min	np.nanmin	Вычисляет минимальное значение
np.max	np.nanmax	Вычисляет максимальное значение
np.argmin	np.nanargmin	Индекс минимального значения
np.argmax	np.nanargmax	Индекс максимального значения
np.median	np.nanmedian	Вычисляет медиану элементов
np.percentile	np.nanpercentile	Вычисляет квантили элементов
np.any		Существуют ли значения true
np.all		Все ли элементы имеют значение true

Причем можно вычислять для всего многомерного массива и по отдельным осям.

```
a = np.array([[1, 2], [3, 4]])
np.min(a)                # 1 - во всем массиве
np.min(a, axis = 0)      # array([1, 2])
np.min(a, axis = 1)      # array([1, 3])

np.mean(a)               # 2.5
np.std(a)                # 1.118033988749895
np.var(a)                # 1.25

b = np.array([[1, 2], [3, np.nan]])
np.nanmin(b)             # 1.0
```

Транслирование

Транслирование представляет собой набор правил по применению бинарных универсальных функций (сложение, вычитание, умножение и т. д.) к массивам различного размера.

```
a = np.array([0, 1, 2])
b = np.array([5, 5, 5])
# массивы одинакового размера – поэлементно:
a + b # [5 6 7]
# массивы и скаляр – к каждому элементу:
a + 5 # [5 6 7]
# массивы разной размерности:
M = np.ones((3, 3))
# [[1. 1. 1.] [1. 1. 1.] [1. 1. 1.]]
print(a + M) # (1, 3) и (3, 3) → (3, 3):
# [[1. 2. 3.] [1. 2. 3.] [1. 2. 3.]]
# массивы разной размерности:
```



```
c = np.array([0, 10, 20]).reshape(3,1)
# c = [[ 0] [10] [20]] # вектор-столбец
print(a + c) # (1, 3) и (3, 1) → (3, 3):
# [[ 0  1  2]
#    [10 11 12]
#    [20 21 22]]
```

Сравнения, маски и булева логика

<https://numpy.org/doc/stable/reference/routines.logic.html>

Оператор	Эквивалентная универсальная функция
<code>==</code>	<code>np.equal</code>
<code>!=</code>	<code>np.not_equal</code>
<code><</code>	<code>np.less</code>
<code><=</code>	<code>np.less_equal</code>
<code>></code>	<code>np.greater</code>
<code>>=</code>	<code>np.greater_equal</code>

Пример.

```
np.random.seed(7)
x = np.random.randint(-100, 101, 5)
# x = [ 75  96 -75 -33  51]
x > 0 # [ True  True False False  True]
x < 0 # [False False  True  True False]
abs(x) == 75 # [ True False  True False False]
abs(x) != 75 # [False  True False  True  True]
```

Работа с булевыми массивами

Созданные булевы массивы можно использовать.

Пример.

```
# количество ненулевых элементов (= True) :
np.count_nonzero(x > 0)      # 3
# Сумма элементов (True = 1, False = 0) :
np.sum(x > 0)                 # 3
# Есть ли элементы > 0 :
np.any(x > 0)                 # True
# Все ли элементы > 0 :
np.all(x > 0)                 # False
```

Можно объединять несколько условий с помощью битовых и булевых операций.

Оператор	Эквивалентная универсальная функция
Битовые	
&	np.bitwise_and
	np.bitwise_or

\wedge	<code>np.bitwise_xor</code>
\sim	<code>np.bitwise_not</code>
Логические	
	<code>logical_and, logical_or, logical_not, logical_xor</code>

Для логических значений (True, False) они работают одинаково.
Пример.

```
np.random.seed(7)
x = np.random.randint(-100, 101, 5)
# x = [ 75  96 -75 -33  51]
y = np.random.randint(-100, 101, 5)
# [ 3 -8 85 42 -77]
# Все 3 оператора вернут одинаковый результат:
np.logical_and(x > 0, y > 0)
np.bitwise_and(x > 0, y > 0)
(x > 0) & (y > 0)
# array([ True, False, False, False, False])
```

Также булевы массивы можно использовать как индексы:

```
np.random.seed(0)
x = np.random.randint(-10, 11, size = (3, 3))
# x = [[ 2  5 -10]
#       [-7 -7 -3]
#       [-1  9  8]]
x > 0
# [[ True True  False]
#   [False False False]
#   [False True  True]]
# Выбрать только > 0 числа из массива x:
x[x > 0]
# Возвращается одномерный массив:
# [2 5 9 8]
```

Операции `and`, `or`, `not` не работают с массивами `numpy`. Для чисел битовые (bitwise) и логические (logical) операции отличаются

Функция where, выбор по условию

<https://numpy.org/doc/stable/reference/routines.sort.html#searching>

`where(condition[, x, y])` – возвращает элемент массива `x` или `y` в зависимости от условия.

```
a = np.arange(10) - 5
# a = [-5 -4 -3 -2 -1 0 1 2 3 4]
# В зависимости от условия выбираем -a или a:
b = np.where(a < 0, -a, a)
# b = [5 4 3 2 1 0 1 2 3 4]
```

`nonzero(a)` – возвращает индексы ненулевых элементов.

```
x = np.array([[3, 0, 0], [0, 4, 0], [5, 6, 0]])
np.nonzero(x)
# (array([0, 1, 2, 2], dtype = int64),
#   array([0, 1, 0, 1], dtype = int64))
```

«Прихотливая» индексация (fancy indexing)

Похожа на уже рассмотренную простую индексацию, но вместо скалярных значений передаются массивы индексов.

```
np.random.seed(0)
x = np.random.randint(0, 50, size = 10)
# x = [44 47 0 3 3 39 9 19 21 36]
ind = [1, 5, 3]
x[ind]
# [47 39 3]
```

Форма результата отражает форму массивов индексов, а не форму индексируемого массива:

```
ind = np.array([[1, 5], [3, 9]])
x[ind]
# [[47 39] [ 3 36]]
```

Работает и в случае многомерных массивов.

```
X = np.arange(12).reshape((3, 4))
# X = [[ 0  1  2  3]
        [ 4  5  6  7]
        [ 8  9 10 11]]
# Определяем индексы:
row = np.array([0, 1, 2])
col = np.array([2, 1, 3])
X[row, col]      # Выбирает X[0, 2] X[1, 1] X[2, 3]:
#   [ 2  5 11]
```

Можно смешивать разные индексы между собой.

Сортировка массивов

<https://numpy.org/doc/stable/reference/routines.sort.html>

В пакете есть свои более быстрые методы сортировки массивов.

1) Функция возвращает отсортированную копию массива:

`numpy.sort(a, axis = -1, kind = None, order = None)`

2) Метод массива сортирует массив на месте:

`ndarray.sort(axis = -1, kind = None, order = None)`

Параметры:

`axis` – выбор оси (по умолчанию последняя);

`kind` – метод (`'quicksort'`, `'mergesort'`, `'heapsort'`, `'stable'`);

`order` – можно по нескольким полям.

```
x = np.array([2, 1, 4, 3, 5])
y = np.sort(x)      # создаем новый массив
# y = [1 2 3 4 5]
# или
x.sort()            # сортируем x
# x = [1 2 3 4 5]
```

Пример. По разным осям

```
X = np.random.randint(0, 10, (3, 3))  
# X = [[5 0 3]  
       [3 7 9]  
       [3 5 2]]  
print(np.sort(X))  
# или print(np.sort(X, axis = 1))  
#      [[0 3 5]  
       [3 7 9]  
       [2 3 5]]  
print(np.sort(X, axis = 0))  
#      [[3 0 2]  
       [3 5 3]  
       [5 7 9]]  
# Сортирует все элементы в списке:  
print(np.sort(X, axis = None))  
#      [0 2 3 3 3 5 5 7 9]
```

Сохранение массива в файл

<https://numpy.org/doc/stable/reference/routines.io.html>

Сохранить в файл массив можно с помощью метода save. Создает файл со специальной структурой, в которой хранится размерность массива, тип и сами данные.

```
import numpy as np
arr = np.random.randint(-5, 6, size=(2, 4))
# Сохраняем в файл:
np.save('some_array.npy', arr)
```

Загрузить из файла такой массив обратно в программу можно с помощью метода load:

```
# Загружаем из файла и создаем массив a:
a = np.load('some_array.npy')
print(a)
#      [[-3 -2  2  2]
#       [ 4 -4  1  3]]
```

Литература

- 1. Плас Дж. Вандер Python для сложных задач: наука о данных и машинное обучение. Серия «Бестселлеры O'Reilly». СПб.: Питер, 2018. 576 с.**
- 2. Уэс Маккинли Python и анализ данных. М.: ДМК Пресс, 2015. 482 с.**

II. Задания

Создание одномерных массивов и двумерных массивов

1. Создайте 2 вектора с помощью функции `array`.
 2. Создайте 2 вектора с помощью функции `arrange`.
 3. Создайте 2 вектора с помощью функции `linspace`.
 4. Создайте 2 вектора с помощью функции `random`.
 5. Создайте 2 вектора с помощью функции `randint`.
-

Атрибуты массивов, операции с векторами

6. Выдайте на экран информацию о созданном выше массиве: тип (`dtype`), размерность (`ndim`, `shape`), количество элементов (`size`), `itemsize`, `nbytes`.
 7. С помощью срезов выделите несколько элементов в массиве и поменяйте их значения.
 8. Найти результаты арифметических операций для 2 массивов (или числа и массива). Используйте операции или функции `numpy` (т.е., например, “+” или “`np.add`”). Вычислите еще любые две функции от массивов.
-

Расстояние, норма и скалярное произведение

9. Вычислите расстояние ρ_1 между двумя векторами.
 10. Вычислите расстояние ρ_2 между двумя векторами.
 11. Вычислите расстояние ρ_∞ между двумя векторами.
 12. Вычислите норму $\|a\|_1$ вектора.
 13. Вычислите норму $\|a\|_2$ вектора.
 14. Вычислите норму $\|a\|_\infty$ вектора.
 15. Вычислите скалярное произведение двух векторов.
-

Определение матрицы и операции с матрицами

16. Создайте 2 матрицы одинакового размера. Найти результаты арифметических операций для 2 матриц. Кроме поэлементного умножения “*”, вычислите еще произведение матриц, определяемое в алгебре, с помощью функции “np.dot”.
17. Найдите определитель квадратной матрицы.
18. Найдите собственные значения квадратной матрицы.

19. Найдите собственные значения и собственные вектора квадратной матрицы.
 20. Найдите обратную матрицу к квадратной матрице.
 21. Найдите норму квадратной матрицы.
-

Агрегирование. Определите произвольный массив. Найдите для него следующие величины (можно по одной из осей или для всего массива):

22. Сумму, среднее и минимальное значения.
 23. Произведение, стандартное отклонение и индекс минимального значения.
 24. Среднее значение, дисперсию и медиану.
-

Сравнения, маски и булева логика

Определите произвольный массив.

25. Найдите количество элементов из промежутка $[-5, 5]$.
26. Определите, есть ли среди его элементов числа больше 100.
27. Определите, все ли они положительные.
28. Выбрать из массива все элементы из промежутка $[0, 10]$.
29. Заменить в массиве все отрицательные элементы на 0.