

Лекция 12

Библиотека NumPy

Модуль numpy

- Библиотека функций для научных вычислений NumPy представлена модулем numpy
- Традиционная инструкция импорта:

```
import numpy as np
```

- Библиотека NumPy *не входит* в стандартную поставку Питона и должна быть установлена как отдельный пакет или модуль
- В Linux модуль numpy оформлен как пакет дистрибутива
- Для MacOS и Windows существуют готовые бинарные сборки Питона с библиотеками для научных расчетов, например Anaconda (лицензия BSD):

<https://www.anaconda.com/distribution/>

Массив `ndarray`

- Основная структура данных библиотеки `numpy` это класс `ndarray` (N-Dimensional Array)
- `ndarray` представляет собой массив произвольной размерности
- Класс реализован на языке C, что обеспечивает высокую скорость вычислений
- Все элементы массива должны быть одного типа, то есть размер всех элементов одинаков
- Данные хранятся в блоке памяти языка C содержащем `n` байт:
$$n = \text{число_элементов_в_массиве} * \text{размер_элемента_в_байтах}$$
- Массив может иметь любое количество размерностей, размерности также называют *осями*
- Оси нумеруются целыми числами начиная с нуля. При обращении к элементу массива индексы осей с меньшими номерами записываются левее

Организация многомерного массива

- Многомерный массив это абстракция - отображение элементов многомерного массива в линейную последовательность
 - для массива в стиле языка C отображение начинается с *правых* индексов, самый правый индекс изменяется быстрее всего (row-major)
 - для массива в стиле языка Fortran отображение начинается с *левых* индексов, самый левый индекс изменяется быстрее всего (column-major)
 - различия в стиле массивов проявляются при чтении данных в массив из файла или при компоновке программы с библиотеками, написанными на Фортране
- Традиционное имя *array* введено как alias для ndarray.
- В большинстве программ используется array.
- Следует различать класс array из модуля array и alias array из модуля numpy

Элементы массива

- Пример чтения элемента массива

```
e = a[ 3, 7, 2, 4 ]
#      ^  ^  ^  ^
# Номера осей: 0, 1, 2, 3
```

- При обращении к элементам массива индексы указываются в квадратных скобках через запятую
- В параметрах функций требующих указать последовательность осей, числа для осей с меньшими номерами записываются левее
- Тип элементов - целые числа, числа с плавающей запятой, комплексные числа и строки фиксированной длины. Для каждого типа чисел можно выбрать разрядность. Также элементы могут иметь логический или структурный тип.
- При передаче массива `ndarray` в функцию `print()` выводимые на экран значения *не разделяются* запятыми:

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

Атрибуты массива

- `flags` - флаги: организация массива в памяти
 - в стиле C - правый индекс изменяется первым
 - в стиле Fortran - левый индекс изменяется первым
 - непрерывный / разреженный
 - имеет собственный буфер данных
 - запись разрешена / запрещена
- `ndim` - размерность массива (количество осей, ранг массива)
- `shape` - кортеж размеров массива по осям (форма массива)
- `size` - количество элементов в массиве
- `dtype` - тип данных элемента в массива
- `itemsize` - размер элемента массива в байтах
- `nbytes` - общий размер всех элементов массива в байтах
- `flat` - итератор для последовательного доступа к элементам массива в порядке их расположения в памяти

Создание массива

- Для создания массива используется функция, совпадающая с именем его класса

```
array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)
```

- Первым и единственным обязательным параметром является объект, данные которого вносятся в массив. Объект может быть последовательностью или иметь интерфейс массива (атрибут-словарь `__array_interface__`).
- Необязательные параметры:
 - `dtype` - желаемый тип элементов массива, например `'i4'`
 - `copy` - по умолчанию объект копируется
 - `order` - размещение элементов многомерного массива:
`'C'` - C-style, `'F'` - Fortran-style, `'A'` - any-style, `'K'` - keep style
 - `subok` - разрешается субкласс массива
 - `ndmin` - минимальная размерность массива

Создание многомерного массива

- Вложенные последовательности как параметр

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

- Массив будет иметь размерность (2, 2, 3) повторяя структуру вложенности последовательностей своего аргумента. Внутренние последовательности соответствуют правым индексам.
- Доступ к элементу многомерного массива:

```
a[0, 1, 2] # => 6
```

Изменение формы массива

- Метод `reshape()` изменяет форму массива:

```
b = np.array((1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)).reshape(2, 2, 3))
```

- Линейная последовательность форматируется как массив с размерностью (2, 2, 3) с помощью метода `reshape()` класса `ndarray`
- Массив `b` равен массиву `a` из предыдущего примера. Для сравнения массивов используется функция `array_equal()`:

```
if np.array_equal(a, b):  
    print('OK') # Массив b равен массиву a
```

 - операции сравнения `==`, `>`, `<` и другие, при применении к массивам выполняют поэлементное сравнение
- Метод `reshape()` не изменяет содержимое буфера данных, его результатом будет новый кортеж, записанный в атрибут `shape`

Последовательность чисел в массиве

- Функция `np.arange()` создает массив, содержащий последовательность чисел
- Функция `np.arange()` по результату работы аналогична встроенной функции `range()`, но допускает числа с плавающей запятой в качестве всех трех параметров

```
c = np.arange(1, 13).reshape(2, 2, 3) # Также равен массиву a
```

```
d = np.arange(0.0, 1.1, 0.1) # 11 элементов от 0 до 1 с шагом 0.1
```

- Как и в случае функции `range()`, максимальное значение *не включается* в состав последовательности

Функции `linspace()` и `logspace()`

- Функции `np.linspace()` и `np.logspace()` заполняют массив линейной или логарифмической последовательностью чисел
- Функция *`linspace`* возвращает последовательность чисел, разделенных равными интервалами от начального (`start`) до конечного (`stop`) значения. Последнее значение включается в последовательность если аргумент `endpoint=True`

```
linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
```

- Функция *`logspace`* создает линейную последовательность степеней и возводит значение `base` в эти степени

```
logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
```

- Пример

```
a = np.linspace(-np.pi, np.pi, 11, endpoint=True)
b = np.logspace(0, 1, 11, base=np.e, endpoint=True)
```

Однородное заполнение массива

- Пустой (неинициализированный) массив:

```
np.empty(shape, dtype=float, order='C')
```

- в "пустом" массиве в соответствии с его размерностью все элементы существуют и имеют произвольные значения

- Массив заполненный нулями:

```
np.zeros(shape, dtype=float, order='C')
```

- Массив заполненный единицами:

```
np.ones(shape, dtype=float, order='C')
```

- По умолчанию создаются массивы с элементами типа float и расположением в памяти в стиле языка C (правый индекс изменяется быстрее всех)

Заполнение случайными числами

- Случайные числа типа float в интервале [0,1)

```
np.random.rand(d0, d1, ..., dn) # d0 ... dn - размерности по осям  
np.random.random_sample(size=None) # size - число или кортеж
```

- Случайные числа типа float с нормальным распределением
По умолчанию среднее значение 0, среднеквадратическое отклонение 1.

```
np.random.randn(d0, d1, ..., dn)  
np.random.standard_normal(loc=0.0, scale=1.0, size=None)
```

- Случайные целые числа в интервале [low, high)

```
np.random.randint(low, high=None, size=None, dtype='l')
```

- Случайная последовательность байт

```
np.random.bytes(length)
```

Заполнение массива данными

- Итератор, как источник данных, указание типа данных обязательно

```
np.fromiter(iterable, dtype, count=-1)
```

- Объект с интерфейсом буфера (buffer protocol), например bytes

```
np.frombuffer(buffer, dtype=float, count=-1, offset=0)
```

```
b = b'\x24\x17\xa4\x1f\x24\x17\xa4\x1f'
```

```
a = np.frombuffer(b, dtype='int16') # => [5924 8100 5924 8100]
```

Чтение данных из файла

- Функция `np.fromfile()` создает массив и помещает в него данные, считанные из файла

```
np.fromfile(file, dtype=float, count=-1, sep='', offset=0)
```

- Если параметр `sep` не задан, файл считается бинарным. В этом случае допускается использование параметра `offset` - сколько байт пропустить, прежде чем начать считывание.
- Если параметр `sep` задан, файл считается текстовым, а `sep` это разделитель между числами
- К функции `fromfile` существует парный метод `tofile()`

```
ndarray.tofile(file, sep='', format='%s')
```

- *метод `tofile` удобен для отладки*

Функция `genfromtxt()`

- Максимум настроек при чтении данных из файла предоставляет функция `genfromtxt(fname, ...)`
- Функция имеет около двадцати параметров, которые позволяют:
 - пропускать комментарии
 - пропускать строки заголовка и / или заключительные строки
 - определять функции преобразования данных для каждого столбца индивидуально
 - устанавливать значения по умолчанию для пропущенных данных
 - считывать и сохранять заголовки столбцов
 - модифицировать текстовые строки для исключения запрещенных символов и / или идентификаторов
 - осуществлять ряд других настроек
- Все параметры кроме имени файла имеют умолчания

Функция `copy()`

- Функция `copy()` создает копию массива, буфер данных также копируется

```
np.copy(a, order='K')
```

- Необязательный параметр `order` задает организацию массива:
 - 'C' - в стиле языка C
 - 'F' - в стиле языка Fortran
 - 'A' - auto, в стиле языка Fortran если оригинальный массив это непрерывный Fortran-массив, иначе в стиле языка 'C'
 - 'K' - keep, сохраняет оригинальную организацию массива

Метод `view()`

- Метод `view()` класса `ndarray` создает другой "взгляд" на уже существующий буфер данных

```
a.view(dtype=None, type=None)
```

- Параметр `dtype` задает тип данных к которому приводится содержимое буфера
- Параметр `type` задает тип массива, здесь может быть указан класс производный от `ndarray`, например `matrix`

Индексация одномерного массива

```
ald = np.array((1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
```

```
ald[5] # => 6           # Один элемент
ald[2:5] # => [3 4 5]   # Фрагмент (slice)
ald[2:8:2] # => [3 5 7] # Фрагмент (slice) с шагом
ald[[1, 3, 9, 2]] # => [2 4 10 3] # Несколько элементов
ald[[i**2 for i in range(4)]] => [ 1  2  5 10] # Генератор списка
```

```
ald[5] = 18
ald[8:11] = (28, 29, 30)
ald[[1, 3, 2]] = 101, 103, 102
print(ald) # => [  1 101 102 103   5  18   7   8  28  29  30  12]
```

- Двоеточие в квадратных скобках создает объект типа *slice*
- Индексация реализуется методами класса ndarray
 - `__getitem__(self, i)`
 - `__setitem__(self, i, v)`
- Попытка удалить элемент вызывает ошибку

Индексация двумерного массива

```
a2d = np.array((1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)).reshape(3, 4)

print(a2d) # =>
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
print(a2d[2, 1]) # => 10
print(a2d[2]) # => [ 9 10 11 12]
print(a2d[0:2, 1:3]) # =>
# [[2 3]
#  [6 7]]
print(a2d[[0,2,1], [1,3,2]]) # => [ 2 12  7]
```

- Индексы многомерного массива указывают в квадратных скобках через запятую
- Если при индексации массива размерностью N указано K индексов, то результатом будет массив размерностью N-K. Недостающие индексы трактуются как самые правые.

Объект Ellipsis (...)

- Фрагмент (slice) в виде одиночного двоеточия (:) означает "вся последовательность"
- Пример:

```
a = np.arange(720).reshape(2, 3, 4, 2, 3, 5)
# => shape = (2, 3, 4, 2, 3, 5)
b1 = a[1,2,:,:,:,:] # => shape = (4, 2, 3, 5)
c1 = a[1,:,:,:,:,4] # => shape = (3, 4, 2, 3)
d1 = a[:,,:,:,:,2,3] # => shape = (2, 3, 4, 2)
```

- Для сокращенной записи последовательности двоеточий используется объект ellipsis:

```
b1 = a[1,2,...] # => shape = (4, 2, 3, 5)
c1 = a[1,...,4] # => shape = (3, 4, 2, 3)
d1 = a[...,2,3] # => shape = (2, 3, 4, 2)
```

```
e1 = a[...,0,...] # => IndexError: can only have a single ellipsis
```

Выборка по массиву логических величин

- Если в качестве индекса используется массив логических величин результатом выборки будет *одномерный массив*, содержащий элементы сопоставленные со значением True в массиве индексов
- Оба массива должны иметь одинаковую размерность
- Пример

```
a = np.array([ [1, 2, 3], [4, 5, 6] ])
b = np.array([ [False, True, True], [False, False, True] ])
print(a[b]) # => [2 3 6]
```

Выборка по условию

- В качестве индекса можно использовать выражение, результат которого может быть интерпретирован как логическая величина. Массив рассматривается как одномерный. Результат - *одномерный массив*, содержащий элементы исходного массива, удовлетворяющие условию.
- Имя переменной в выражении совпадает с именем массива

```
a = np.array((1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12))  
print(a[a % 3 == 0]) # => [ 3  6  9 12]
```

```
i = 3  
a[a % i == 0] = (30, 60, 90, 120)  
print(a) # => [ 1  2 30  4  5 60  7  8 90 10 11 120]
```

Массив, как итерируемый объект

- В контексте итерации происходит перебор по самому левому индексу, элементами будут массивы размерности N-1

```
a = np.arange(1, 13).reshape(2, 2, 3)
for e in a:
    print('Subarray:\n', e)
# => Subarray:
# => [[1 2 3]
# =>  [4 5 6]]
# => Subarray:
# => [[ 7  8  9]
# =>  [10 11 12]]
```

- Для итерации по отдельным элементам в порядке их следования по осям используется атрибут-итератор flat:

```
for e in a.flat:
    print(e, end=' ')
# => 1 2 3 4 5 6 7 8 9 10 11 12
```

Матрицы

- Матрица, `class matrix`, является субклассом класса `ndarray` и представляет собой двумерный массив
- Матрица может быть создана теми же способами, что `ndarray` или прочитана из текстовой строки, где строки матрицы разделены точкой с запятой

```
m = np.matrix('1 2 3 4; 5 6 7 8; 9 10 11 12')
```

- Атрибуты матрицы:
 - `m.T` - транспонированная матрица
 - `m.H` - сопряженная матрица
 - `m.I` - обратная матрица
 - `m.A` - `ndarray` представляющий матрицу
 - `m.A1` - одномерный `ndarray` представляющий содержимое матрицы

Операции с массивами

- Массивы поддерживают основные арифметические операции
- Операция совершается над элементами, занимающими в своих массивах идентичные позиции, то есть *поэлементно*
- Если один из массивов имеет меньшую размерность, его содержимое может быть размножено (broadcast) на недостающие измерения
- Если один из операндов является простым объектом (скаляром) операция совершается над этим объектом и каждым элементом массива индивидуально
- В результате простой арифметической операции создается новый массив
- Если операция совмещена с операцией присваивания (составное присваивание), происходит модификация уже существующего массива.
- Для классического умножения матриц служит функция `np.dot()`

Универсальные функции

- Поэлементные операции реализованы *универсальными функциями* (universal functions или ufunc), например при выполнении операции сложения вызывается функция `np.add()`
- Кроме арифметических действий универсальные функции позволяют совершать тригонометрические и логарифмические операции, извлекать корни, находить максимум и минимум, а также выполнять ряд других действий
- Универсальные функции имеют ряд именованных параметров, позволяющих указать массив для вывода результата, а также задавать тип данных элементов и организацию результирующего массива
- Универсальные функции работают значительно быстрее, чем цикл по элементам массива, в теле которого выполняется соответствующая операция

Примеры операций с массивами

```
a = np.array((1,2,3))
b = a + 10 # Создается новый массив
print(b)   # => [11 12 13]
print(a)   # => [1 2 3]
a += 10    # Модифицируется уже существующий массив
print(a)   # => [11 12 13]
```

```
a = np.array(((1,2,3), (11, 12, 13), (21, 22, 23)))
print(a)
# [[ 1  2  3]
#  [11 12 13]
#  [21 22 23]]
b = a + (2, 3, 4) # Расширение массива вдоль оси 0 (здесь ось Y)
print(b)
# [[ 3  5  7]
#  [13 15 17]
#  [23 25 27]]
```

Методы массива

- Большинство методов создают иное представление массива (view), которое ссылается на те же самые данные. Случаи, когда данные копируются отмечены особо.

`reshape(shape, order='C')`

- возвращает массив с тем же содержимым но измененной размерностью

`flatten(order='C')`

- возвращает копию содержимого в виде одномерного массива

`transpose(*axes)`

- транспозиция, порядок осей задается аргументами. При вызове без аргументов порядок осей меняется на противоположный

`swapaxes(axis1, axis2)`

- меняет местами две оси массива

`squeeze(axis=None)`

- удаляет оси массива, размер которых равен единице

Виды операций

- *Поэлементная операция* совершается над каждым элементом индивидуально, например каждый элемент может быть умножен на константу
- *Операция над массивом в целом* вычисляет функцию всех элементов массива, например сумму. В таких операциях многомерный массив трактуется как одномерный
- *Операция вдоль оси* трактует N-мерный массив элементов как (N-1)-мерный массив векторов (одномерных массивов). Операция совершается над каждым вектором индивидуально
- Аналогично осуществляется операция вдоль K осей. N-мерный массив трактуется как (N-K)-мерный массив K-мерных массивов.
- Многие функции библиотеки имеют аргумент *axis*, определяющий ось или оси, вдоль которых производится операция
- Функция также может иметь аргумент *keepdims*. Если он задан как True, размерность массива сохраняется.

Функции для работы с массивами

`np.concatenate(*arrays, axis=0)`

- соединяет несколько массивов вдоль заданной оси

`np.stack(*arrays, axis=0)`

- создает новую ось и соединяет массивы вдоль нее

`np.split(*arrays, indices_or_sections, axis=0)`

- разделяет массив на подмассивы вдоль указанной оси

- Варианты функций `stack` и `split` с префиксами `h`, `v`, и `d` подразумевают оси X (0), Y (1) и Z (2), например `vsplit()` разделяет массив вдоль оси Y

`np.resize(a, shape)`

- Функция `resize()` создает новый массив заданной размерности, заполняя его элементами оригинального массива. Лишние элементы отбрасываются, недостающие элементы получаются возвращением к началу оригинального массива.

Функции для работы с массивами

`np.append(a, values, axis)`

- добавляет элементы в массив вдоль заданной оси

`np.insert(a, ndx, values, axis)`

- вставляет элементы в массив вдоль заданной оси в позиции заданной значением индекса

`np.delete(a, ndx, axis)`

- удаляет элементы из массива на заданной оси в позиции заданной значением индекса

- Все три функции создают новый массив, как модифицированную копию оригинала

Сортировка

- Элементы массива можно отсортировать используя функцию `sort()` или метод `sort()`

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

 - функция `sort` создает новый отсортированный массив
 - метод `sort` изменяет сам обрабатываемый массив
- Аргумент `kind` позволяет выбрать алгоритм сортировки
 - 'quicksort'
 - 'heapsort'
 - 'stable' или сохраненный для совместимости 'mergesort'
- Функция `argsort(a)` возвращает массив индексов элементов массива `a`. Последовательный проход по массиву индексов позволит выбрать элементы массива `a` в отсортированном виде.
- Функция `argsort` полезна в тех случаях, когда элементы сортируемого массива имеют большой размер

Функция `unique()`

- Функция *`unique(a)`* возвращает *отсортированный массив неповторяющихся элементов* из массива *a*

```
np.unique(a, return_index=False, return_inverse=False,  
         return_counts=False, axis=None)
```

- Если задан один из необязательных параметров `return_*`, функция возвращает кортеж из нескольких элементов, дополнительными элементами которого будут:
 - `return_index` - массив индексов первых включений уникальных элементов
 - `return_inverse` - массив индексов, позволяющий восстановить оригинальный массив
 - `return_counts` - массив количества включений каждого элемента

Статистические функции

`np.amax(a, axis=None)`

`np.amin(a, axis=None)`

- возвращает максимальный (минимальный) элемент в массиве, или массив размерности N-K вдоль K осей `axis`, если они заданы

`np.ptp(a, axis=None)`

- возвращает разность между максимальным и минимальным элементами в массиве, или массив размерности N-K вдоль осей `axis`, если они заданы (peak-to-peak)

`np.median(a, axis=None)`

- возвращает медиану, то есть значение, которое не превысят половина элементов массива, или массив медиан размерности N-K вдоль K осей `axis`, если они заданы

`np.percentile(a, q, axis=None)`

- возвращает перцентиль, то есть значение, которое не превысят `q` процентов элементов массива, или массив перцентилей размерности N-K вдоль K осей `axis`, если они заданы

Среднее значение

`np.mean(a, axis=None)`

- возвращает арифметическое среднее, или массив арифметических средних размерности N-K вдоль K осей `axis`, если они заданы

`np.average(a, axis=None, weights=None)`

- возвращает взвешенное среднее, или массив взвешенных средних размерности N-K вдоль K осей `axis`, если они заданы. Параметр `weights` это последовательность весовых коэффициентов.

- Некоторые статистические функции имеют дополнительные необязательные параметры:
 - `out` - массив для помещения в него результата операции
 - `overwrite_input` - позволяет использовать исходный массив в качестве буферной памяти
 - `keepdims` - требует сохранять в результате размерность исходного массива

Массивы в файлах

- Для временного хранения данных объекты класса `ndarray` можно сохранять в файлах и восстанавливать из файлов

```
np.load(filename)
```

- загружает массив из бинарного файла

```
np.save(filename, a)
```

- сохраняет массив в бинарный файл

```
np.loadtxt(filename, dtype=float)
```

- загружает массив из текстового файла

```
np.savetxt(filename, a, fmt='%.18e', delimiter=' ')
```

- сохраняет массив в текстовый файл

- Функции `load/save` являются парными и используются совместно. Для загрузки данных в массив из файла с произвольным форматом служат функции `fromfile()` и `genfromtxt()`, для записи в файл *method* `tofile()`

Субмодуль char

- Субмодуль char содержит функции осуществляющие поэлементную обработку массивов, содержащих текстовые строки

`np.char.add()` - поэлементная конкатенация строк двух массивов

`np.char.multiply()` - повторение (умножение строки на число)

`np.char.center()` - центрирование текста в строке

`np.char.capitalize()` - первая буква первого слова становится прописной

`np.char.title()` - первая буква каждого слова становится прописной

`np.char.lower()` - все буквы становятся строчными

`np.char.upper()` - все буквы становятся прописными

`np.char.split()` - разделение строки на слова по пробелам или явно заданному разделителю

`np.char.splitlines()` - вариант функции `split()` делящий текст на строки

`np.char.strip()` - удаление пробелов в начале и конце строки

`np.char.join()` - соединение строк с использованием разделителя

`np.char.replace()` - замена подстроки на заданную подстроку

`np.char.decode()` - аналог `str.decode()`

`np.char.encode()` - аналог `str.encode()`

Модуль `numpy` также содержит

- Функции битовых операций
- Тригонометрические функции
- Функции округления
- Функции сортировки и поиска
- Функции для работы с комплексными числами
- Функции для работы с датой и временем
- Функции для финансовых расчетов
- Функции для работы с множествами
- Элементы функционального программирования

Субмодули модуля numpy

- Субмодуль `fft` - быстрое преобразование Фурье
- Субмодуль `linalg` - линейная алгебра
- Субмодуль `matlib` - работа с матрицами
- Субмодуль `polynomial` - работа с полиномами
- Субмодуль `random` - работа со случайными числами
- Субмодуль `testing` - инструменты тестирования программ

Библиотека SciPy

- Библиотека NumPy развивается в рамках более общего проекта SciPy
- Модуль scipy имеет расширенные версии уже имеющихся в numpy возможностей, а также предоставляет новые:
 - специальные функции
 - интегрирование
 - оптимизация
 - интерполяция
 - обработка сигналов
 - обработка изображений
 - работа с разреженными и пространственными данными

Тип данных ndarray стал фактическим стандартом реализации массива в Python-библиотеках для научных расчетов

Литература к лекции

1. <https://docs.scipy.org/doc/numpy/>
2. Jake VanderPlas. "Python Data Science Handbook".
O'Reilly Media, 2016, ISBN 978-1491912058
3. Robert Johansson, "Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib",
Apress, 2019
ISBN-13 (pbk): 978-1-4842-4245-2
ISBN-13 (electronic): 978-1-4842-4246-9
4. Jaan Kiusalaas. "Numerical methods in engineering with Python 3".
Cambridge University Press, 2013, ISBN 978-1-107-03385-6