Информационные технологии

Основы программирования на Python 3

Каф. ИКТ РХТУ им. Д.И. Менделеева
Ст. преп. Васецкий А.М.

Москва, 2018

Лекция 7. Введение в библиотеки Python

- Матрицы
- Библиотеки

Библиотеки-1

1. **Библиотека численных методов NumPy**
   - ✔ NumPy.linalg (Линейная алгебра);
   - ✔ NumPy.random (Случайные числа);
   - ✔ NumPy (Раздел «Индексация» (Indexing routines));
   - ✔ NumPy (Раздел «Ввод и вывод» (Input and output);
   - ✔ NumPy (Разделы строковых и логических операций);
   - ✔ NumPy (Разделы операций с массивами);
   - ✔ NumPy (Раздел математических функций);
   - ✔ NumPy (Разделы сортировки и поиска);
2. **Библиотека алгоритмов и математических инструментов SciPy**
   - ✔ SciPy.linalg (Линейная алгебра);
   - ✔ SciPy.integrate (Интегрирование и решение обыкновенных дифференциальных уравнений);
   - ✔ SciPy.stats (Статистические функции);
   - ✔ SciPy.optimize (Оптимизация);
   - ✔ SciPy.interpolate (Интерполяция);

Библиотеки-2

3. **Blaze** – численные методы для больших данных;
4. **Pandas** – библиотека обработки данных;
5. **Fuzzywuzzy** – библиотека сравнения данных;
6. **matplotlib** – построение графиков;
7. **Seaborn** – визуализация статистических моделей;
8. **Altair** – визуализация данных;
9. **Glueviz** – визуализация данных;
10. **Pyglet** – 3D-анимация;
11. **Vpython** – 3D-графика;
12. **Plotly** – библиотека работы с графикой;
13. **Pillow** – библиотека работы с графикой;
14. **Gnuplot** – библиотека работы с графическими изображениями;
15. **PyX** – библиотека работы с графическими изображениями, PDF и Postscript;
16. **geopy** – библиотека для геолокации;
17. **Requests** – HTTP библиотека;
18. **Urllib** и **Urllib2** – работа с Интернет;
19. **BeautifulSoup** – XML и HTML библиотека;
20. **Scrapy** – библиотека для парсинга сайта;
21. **Django** – Web-фреймворк;
22. **Flask** – Web-фреймворк;

Библиотеки-3

23. **wxPython** – пользовательский интерфейс;
24. **Tkinter** – пользовательский интерфейс;
25. **pyQT** – пользовательский интерфейс;
26. **pyGtk** – пользовательский интерфейс;
27. **pywin32** – библиотека взаимодействия с Windows;
28. **Nose** – среда тестирования;
29. **Nltk** – работа со строками и пр.;
30. **ParaText** – библиотека для обработки текста;
31. **SymPy** – библиотека для символьных вычислений;
32. **ChemPy** – библиотека химических расчётов;
33. **SciKit-Learn** – инструмент для обработки изображений и имитации искусственного интеллекта;
34. **Theano** – библиотека, которая используется для разработки систем машинного обучения;
35. **PyCrypto** – криптографическая библиотека;
36. **mxODBC** – библиотека для связи с базами данных;
37. **pyGame** – библиотека для написания игровых приложений;
38. **pyQuery** – аналог библиотеки ***jquery*** для работы с XML и HTML документами;
39. Модули сериализации и десериализации данных (**pickle**, **json**, **csv**, **yaml** и др.)

Введение в NumPy

- NumPy – фундаментальный пакет для научных вычислений с Python. В нём содержатся:
  - ✔Мощный объект N-мерного массива.
  - ✔сложные функции.
  - ✔Инструментарий для интеграции с кодом на языках C/C++ и Fortran.
  - ✔Линейная алгебра, трансформации Фурье, случайные числа.
- Кроме того, данный пакет может быть использован как многомерный контейнер общих данных. Произвольные типы данных могут быть определены. Это позволяет NumPy легко и быстро интегрироваться с широким спектром баз данных.

Ошибка multiarray

В pyCharm отмечено наличие ошибки при установке ***numpy*** и некоторых других библиотек:

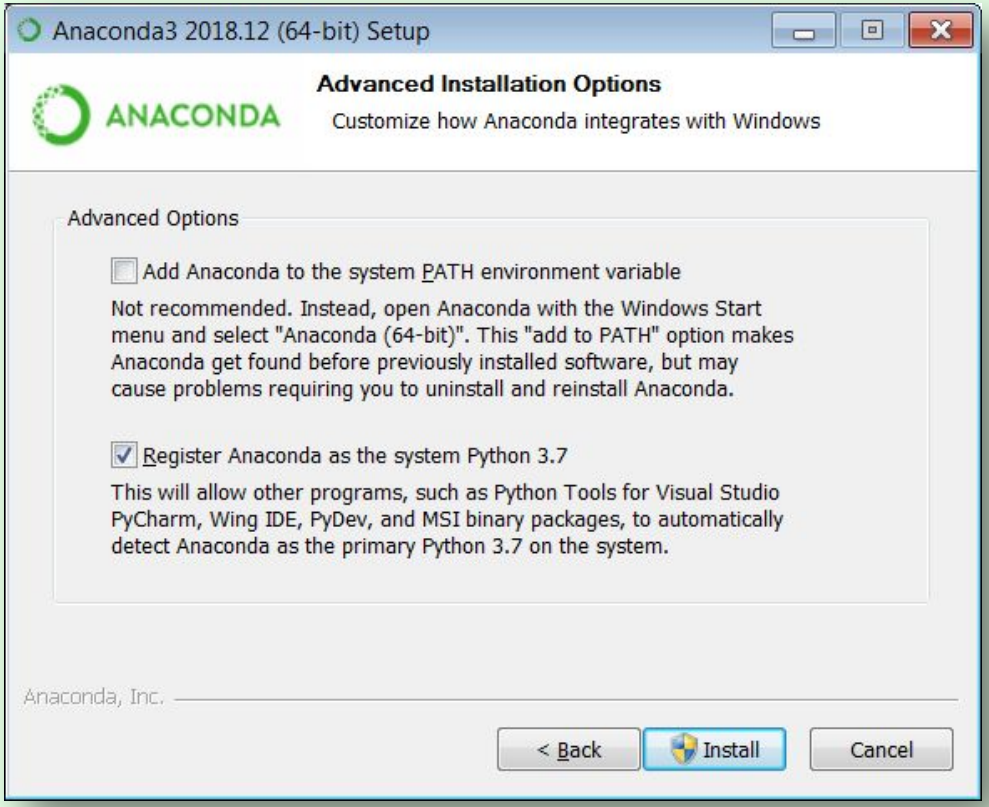
ImportError:
Importing the multiarray numpy extension module failed. Most likely you are trying to import a failed build of numpy.
If you're working with a numpy git repo, try `git clean -xdf` (removes all files not under version control). Otherwise reinstall numpy.

Original error was: DLL load failed: The specified module could not be found.

Устранение ошибки multiarray

Ошибка связана с переменной среды PATH.
Самый простой способ её устранить, это переустановить Anaconda и поставить флажок в первое поле.

В переменную среды PATH добавятся пути к папкам Anaconda:
*d:\ProgramData\Anaconda3;d:\ProgramData\Anaconda3\Library\mingw-w64\bin;
d:\ProgramData\Anaconda3\Library\usr\bin;
d:\ProgramData\Anaconda3\Library\bin;d:\ProgramData\Anaconda3\Scripts*



Для проверки правильности работы библиотеки наберите:
import numpy as np
print(np.__version__)

Основные разделы NumPy-1

- Создание массивов
- Операции с массивами
- Бинарные (побитовые) операции
- Строковые операции
- Функции даты и времени
- Дискретная трансформация Фурье
- Финансовые функции
- Функциональное программирование
- Процедуры индексации
- Ввод/вывод

Основные разделы NumPy-2

- Линейная алгебра
- Логические функции
- Маскированные операции с массивами
- Математические функции
- Матричные функции
- Полиномы
- Случайные числа
- Сортировка
- Статистика
- Оконные функции

Стандартные типы данных

bool_	Булев тип (True или False), хранящийся в виде 1 байта
int_	Тип целочисленного значения 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)

Стандартные типы данных-2

float_	Сокращение для названия типа float64
float 16	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

Массивы

- Библиотека NumPy обеспечивает эффективный интерфейс для хранения и работы с плотными буферами данных. Массивы в ней похожи на встроенный тип данных "список" (list) языка Python, но обеспечивают гораздо более эффективное хранение и операции с данными при росте размера массивов.
- Кроме того срезы массивов возвращают ***представления*** *(views), а не* ***копии*** *(copies)* данных массива. Этим срезы массивов NumPy отличаются от срезов списков так как в списках срезы являются копиями.
- На уровне реализации массив фактически содержит один указатель на непрерывный блок данных. Список же в языке Python содержит указатель на блок указателей, каждый из которых, в свою очередь, указывает на целый объект языка Python, например, на целое число.
- Массивам с фиксированным типом из библиотеки NumPy проигрывают спискам в гибкости, однако гораздо эффективнее хранят данные и работают с ними.

Создание массивов

- Существуют следующие основные способы создания массивов
- Преобразование из других структур Python (списки, кортежи и т.п.)
- Внутренние объекты создания массивов numpy (arange, ones, zeros, и т.п.)
- Чтение массивов с диска, как из стандартных, так и из пользовательских форматов
- Создание массивов из "сырых" байтов с использованием строк или буферов
- Использование специальных библиотечных функций (например, random)

Создание массивов

- Встроенный модуль ***array*** (доступен с версии 3.3 Python) можно использовать для создания плотных массивов данных одного типа:

```
import array
L = list(range(5))
A = array.array('i', L) # array('i', [0, 1, 2, 3, 4])
```

'i' – код типа, указывающий, что содержимое является целыми числами.

Можно напрямую задавать массивы через NumPy:

```
import numpy as np
A = np.array([1, 0, -2, 4, 3]) # [ 1  0 -2  4  3]
B = np.array([0, 1, 2, 3], dtype='float32')
# [0. 1. 2. 3.]
```

Многомерные массивы

- Вложенные списки преобразуются в многомерный массив

```
B = np.array([range(i, i + 3) for i in [0, 2, 4]])
# [[0 1 2]
# [2 3 4]
# [4 5 6]]
```

- Можно использовать одновременно списки и кортежи

```
B = np.array([[1,2.0],[0,0],(1+1j,3.)])
# [[1.+0.j 2.+0.j]
# [0.+0.j 0.+0.j]
# [1.+1.j 3.+0.j]]
```

Способы инициализации массивов

empty(shape[, dtype, order])	новый массив заданной формы и типа без инициализации записей.
empty_like(prototype[, dtype, order, subok])	новый массив той же формы и типа, что и данный массив.
eye(N[, M, k, dtype, order])	двумерный массив с единицами по диагонали и нулями в других местах.
identity(n[, dtype])	единичная матрица.
ones(shape[, dtype, order])	новый массив заданной формы и типа, заполненный единицами.
ones_like(a[, dtype, order, subok])	массив из единиц той же формы и типа, что и данный массив.
zeros(shape[, dtype, order])	новый массив заданной формы и типа, заполненный нулями.
zeros_like(a[, dtype, order, subok])	массив нулей той же формы и типа, что и данный массив.
full(shape, fill_value[, dtype, order])	новый массив заданной формы и типа, заполненный ***fill_value***.
full_like(a, fill_value[, dtype, order, subok])	полный массив с той же формой и типом, что и данный массив.

Примеры инициализации - 1

```
B = np.zeros(5, dtype=int) # [0 0 0 0 0]
B = np.ones((2, 3), dtype=float)
#[[1. 1. 1.]
# [1. 1. 1.]]
B = np.empty([3, 2], dtype=int)
# Внутри B может находиться что угодно
# Работает быстрее, чем методы с
# инициализацией
# [[-512881756       2046]
# [-512880032       2046]
# [       0          0]]
B = ([1,2,3], [4,5,6])    # B – массив-шаблон
A = np.empty_like(B)
# Внутри A может находиться что угодно
#[[2982    0    0]
# [   0    0    0]]
```

Примеры инициализации - 2

```
B = np.eye(2, dtype=int)
# Единичная целочисленная матрица
# [[1 0]
# [0 1]]
B = np.eye(2, 3, dtype=int) # Матрица 2×3
# [[1 0 0]
# [0 1 0]]
B = np.eye(3, k=1)
# Матрица с наддиагональными единичными элементами
# [[0. 1. 0.]
# [0. 0. 1.]
# [0. 0. 0.]]
```

Примеры инициализации - 3

```
B = np.identity(2)
# Единичная матрица типа float (по умолчанию)
# [[1. 0.]
# [0. 1.]]
B = np.full((2, 3), np.inf) # заполнение
# [[inf inf inf]
# [inf inf inf]]
A = np.full_like(B, 7) # Заполнение аналога B
# [[7. 7. 7.]
# [7. 7. 7.]]
```

Атрибуты массивов

- Основным объектом NumPy является однородный многомерный массив ***numpy.ndarray*** элементов одного типа.
- атрибуты объектов ***ndarray***:
  ['T', …, 'all', 'any', 'argmax', 'argmin', 'argpartition', 'argsort', 'astype', 'base', 'byteswap', 'choose', 'clip', 'compress', 'conj', 'conjugate', 'copy', 'ctypes', 'cumprod', 'cumsum', 'data', 'diagonal', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags', 'flat', 'flatten', 'getfield', 'imag', 'item', 'itemset', 'itemsize', 'max', 'mean', 'min', 'nbytes', 'ndim', 'newbyteorder', 'nonzero', 'partition', 'prod', 'ptp', 'put', 'ravel', 'real', 'repeat', 'reshape', 'resize', 'round', 'searchsorted', 'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze', 'std', 'strides', 'sum', 'swapaxes', 'take', 'tobytes', 'tofile', 'tolist', 'tostring', 'trace', 'transpose', 'var', 'view']

Наиболее важные атрибуты массивов

ndim	число измерений ("осей") массива.
shape	размеры массива, его форма. Кортеж натуральных чисел, показывающий длину массива по каждой оси (размерности). Для матрицы из ***n*** строк и ***m*** столбов, ***shape(n, m)***. Число элементов кортежа ***shape*** = ***ndim***.
size	количество элементов массива. Очевидно, равно произведению всех элементов атрибута shape.
dtype	описывает тип элементов массива. Типы могут быть стандартными и пользовательскими.
itemsize	размер каждого элемента массива в байтах.
data	буфер, содержащий фактические элементы массива. Обычно не используется, т.к. проще обращаться к ним с помощью индексов.

Примеры

```
A = np.array([[1,2],[3,4]])
[[1 2]
 [3 4]]
B = A.T      # Транспонирование
[[1 3]
 [2 4]]
A.nbytes     # 16 – байт
A.ndim  # 2 – размерность
A.shape# (2, 2) – форма
A.size     # 4 – количество элементов
```

Создание массивов из существующих данных

array(object[, dtype, copy, order, subok, ndmin])	Создать массив.
asarray(a[, dtype, order])	Преобразовать входные данные в массив.
asanyarray(a[, dtype, order])	Преобразовать входные данные в ***ndarray***, но пропустить подклассы ***ndarray***.
ascontiguousarray(a[, dtype])	Вернуть непрерывный массив (***ndim***> = 1) в памяти (порядок языка C).
asmatrix(data[, dtype])	Интерпретировать входные данные как матрицу.
copy(a[, order])	Вернуть копию массива данного объекта.
frombuffer(buffer[,dtype, count, offset])	Интерпретировать буфер как одномерный массив.
fromfile(file[, dtype, count, sep])	Построить массив из данных в текстовом или двоичном файле.
fromfunction(function, shape, **kwargs)	Создать массив, выполнив функцию над каждой координатой.
fromiter(iterable, dtype[, count])	Создать новый одномерный массив из итерабельного объекта.
fromstring(string[, dtype, count, sep])	Новый одномерный массив, инициализированный из текстовых данных в строке.
loadtxt(fname[, dtype, comments, delimiter, …])	Загрузить данные из текстового файла.

Примеры создания массивов

```
A = np.fromstring("1 2", dtype=int, sep=" ") # [1 2]
A = np.array([1, 2, 3])          # [1 2 3]
A = np.array([[1., 2], [0, 3]]) # [[1. 2.]
                                 # [0. 3.]]
B = np.array(A, copy=True)
print(A is B)                    # False
D = np.array(A, copy=False)
print(A is D)                    # True
D[0, 0] = 7
print(A)                         # [[7. 2.]
                                 # [0. 3.]]
print(B)                         # [[1. 2.]
                                 # [0. 3.]]
print(D)                         # [[7. 2.]
                                 # [0. 3.]]
F = np.copy(A)                   # [[7. 2.]
                                 # [0. 3.]]
```

Создание массивов с использованием функций

```
A = np.fromfunction(lambda i, j: i == j, (3, 3), dtype=int) # Массив размерности 3×3
# dtype – тип данных координат, подаваемых в функцию
# [[ True False False]
# [False  True False]
# [False False  True]]
A = np.fromfunction(lambda i, j: i*10 + j, (3, 3), dtype=int)
# [[ 0  1  2]
# [10 11 12]
# [20 21 22]]
```

Создание массивов при помощи последовательностей

arange([start,] stop[, step,][, dtype])	равномерно распределенные значения в заданном интервале.
linspace(start, stop[, num, endpoint, …])	равномерно распределенные числа внутри указанного интервала.
logspace(start, stop[, num, endpoint, base, …])	числа, равномерно распределенные в логарифмическом масштабе.
geomspace(start, stop[,	числа, равномерно распределенные в логарифмическом масштабе (геометрическая прогрессия).
num, endpoint, …])	координатные матрицы из координатных векторов.
meshgrid(*xi, **kwargs)	Экземпляр ***nd_grid***, который возвращает плотную многомерную «сетку».
mgrid	Экземпляр ***nd_grid***, который возвращает открытую многомерную «сетку».

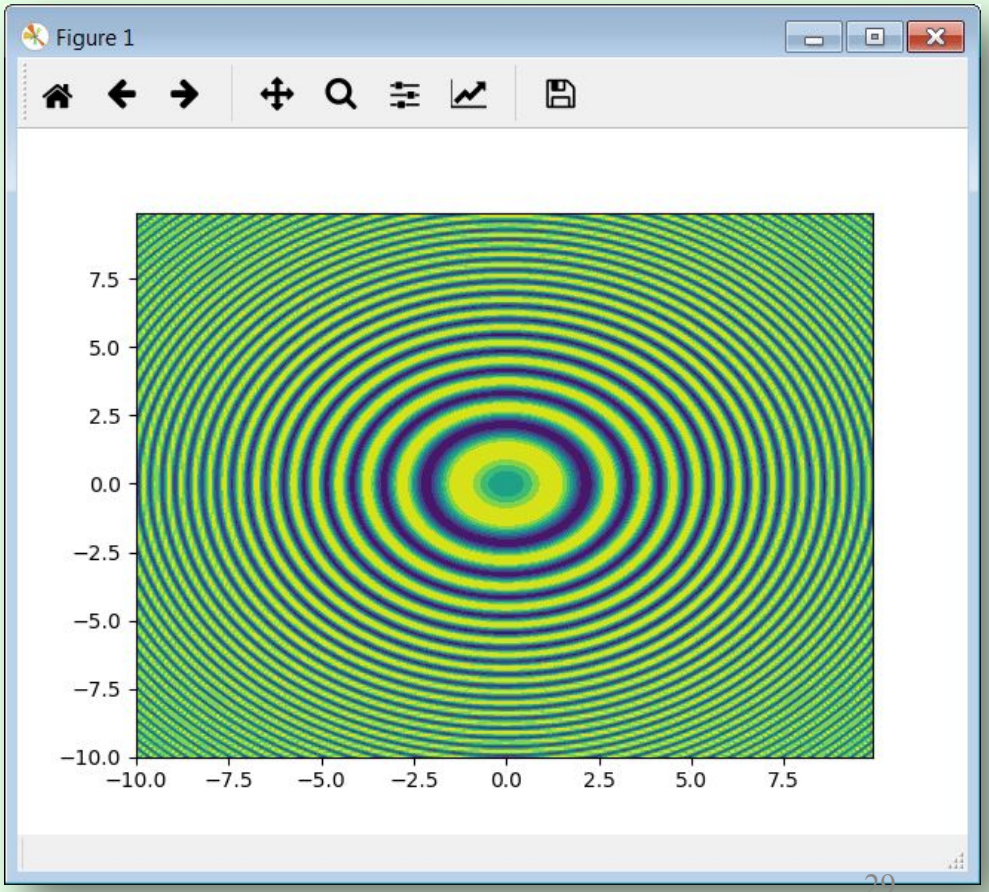
Примеры создания массивов с использованием последовательностей

```
it = (x*x for x in range(5))  # итератор
A = np.fromiter(it, float)     # [ 0.  1.  4.  9. 16.]
A = np.arange(3.)       # [0. 1. 2.]
A = np.arange(0,10,3)    # [0 3 6 9]
A = np.linspace(0.0, 4, num=5)    # [0. 1. 2. 3. 4.]
A = np.logspace(1, 5, num=5)
# array([1.e+01, 1.e+02, 1.e+03, 1.e+04, 1.e+05])
B = np.log10(A) # [1.   1.75   2.5    3.25    4.  ]
A = np.geomspace(1, 4, num=5)
# [1.         1.41421356 2.    2.82842712     4.        ]
```

meshgrid для построения графиков

```
import matplotlib.pyplot as plt
x = np.arange(-10, 10, 0.1)
y = np.arange(-10, 10, 0.1)
xx, yy = np.meshgrid(x, y, sparse=True)
z = np.sin(xx**2 + yy**2)
h = plt.contourf(x,y,z)
plt.show()
```

Подробнее см.
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html



Доступ к элементам массива

В целом доступ и индексация аналогичны спискам.

```
A = np.arange(10)  # [0 1 2 3 4 5 6 7 8 9]
A[:2]              # [0 1]
A[::-1]            # [9 8 7 6 5 4 3 2 1 0]
A = np.array([[1, 5, 3, 0],
              [7, 2, 8, 1],
                 [1, 6, 3, 7]])
B = A[:2, :3]      # две строки и три столбца
# [[1 5 3]
# [7 2 8]]
```

Не забудьте, что B, это НЕ копия!
Поэтому присваивание B[0, 0] = 9
B: [[9 5 3] отразится и на A: [[9 5 3 0]
[7 2 8]] [7 2 8 1]
[1 6 3 7]]

Базовые операции с массивами

- Математические операции над массивами выполняются поэлементно.

```
a = np.array([0, -1, -2, -3])
b = np.arange(4)
c = a + b       # array([0, 0, 0, 0])
c = a / b    # array([nan, -1., -1., -1.])
c = a + 1       # array([ 1,  0, -1, -2])
c = a < 0       # array([False,  True,  True,  True])
```

Операции с массивами-1

copyto(dst, src[, casting, where])	Копирует значения из одного массива (***src)*** в другой (***dst)***, передавая по мере необходимости
reshape(a, newshape[, order])	Придает массиву новую форму без изменения его данных.
ravel(a[, order])	Непрерывный плоский массив.
ndarray.flat	1-D итератор по массиву.
ndarray.flatten([order])	Копия массива, свернутого в одно измерение.
moveaxis(a, source, destination)	Переместить оси массива на новые позиции.
rollaxis(a, axis[, start])	Повернуть указанную ось назад, пока она не окажется в заданном положении. Удобнее использовать ***moveaxis***.
swapaxes(a, axis1, axis2)	Поменять местами две оси массива.
ndarray.T	То же, что ***self.transpose ()***, за исключением того, что ***self*** возвращается, если ***self.ndim <2***.
transpose(a[, axes])	Поменять местами размерности массива

Примеры операций с массивами

- Помимо метода ***copy*** (см. выше) можно копировать массивы при помощи метода ***copyto***

```
A = np.array([[1, 2, 3],
              [11, 12, 13]])
B = np.empty_like(A)
# Массив B той же размерности, что и A
np.copyto(B, A)          # B: [[1, 2, 3],
                                [11, 12, 13]])
np.ravel(A)          # [ 1  2  3 11 12 13]
A.reshape(-1)          # [ 1  2  3 11 12 13]
```

Фортрановский стиль хранения:

```
np.ravel(A, order='F')   # [ 1 11  2 12  3 13]
A.flat[4]                # 12
```

Примеры операций с массивами

```
A = np.arange(6).reshape((2, 3)) # [[0 1 2]
                                 # [3 4 5]]
B = np.transpose(A)              #  [[0 3]
                                 # [1 4]
                                 # [2 5]]
C = A.T                          #  [[0 3]
                                 # [1 4]
                                 # [2 5]]
A = np.ones((2, 3, 4))
A.shape          # Размерности = (2, 3, 4)
np.moveaxis(x, -1, 0).shape
# Переместить последнюю ось на 0 позицию
# (4, 2, 3)
```

Операции с массивами-2

asarray(a[, dtype, order])	Преобразовать входные данные в массив.
asmatrix(data[, dtype])	Интерпретировать ввод, как матрицу
require(a[, dtype, requirements])	Вернуть ***ndarray*** предоставленного типа, который удовлетворяет требованиям.
concatenate((a1, a2, …)[, axis, out])	Соединяет последовательность массивов вдоль существующей оси.
stack(arrays[, axis, out])	Соединяет последовательность массивов вдоль новой оси.
column_stack(tup)	Упаковывает 1-D массивов в виде столбцов в 2-D массив.
dstack(tup)	Упаковывает массивы в последовательности по глубине (вдоль третьей оси).
hstack(tup)	Упаковывает массивы в последовательности по горизонтали (по столбцам).
vstack(tup)	Упаковывает массивы в последовательности вертикально (по рядам).
block(arrays)	Собирает nd-массив из вложенных списков блоков.

Примеры операций с массивами

```
A = [1, 2]
B = np.asarray(A)          # [1 2]
A is B                     # False
a = np.array([1, 2])
np.asarray(a) is a         # True
x = np.array([[1, 2], [3, 4]])
m = np.asmatrix(x)
x[0, 1] = 8                # [[1 8]
                           # [3 4]]
```

Примеры слияния массивов

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[10, 11]])
c = np.concatenate((a, b), axis=0)    # [[ 1  2]
                                      # [ 3  4]
                                      # [10 11]]
d = np.concatenate((a, b.T), axis=1)    # [[ 1  2 10]
                                        # [ 3  4 11]]
e = np.concatenate((a, b), axis=None)
# [ 1  2  3  4 10 11]
```

Слияние, разделение, повторение

split(ary, indices_or_sections[, axis])	Разбить массив на несколько подмассивов.
array_split(ary, indices_or_sections[, axis])	Разбить массив на несколько подмассивов.
dsplit(ary, indices_or_sections)	Разбить массив на несколько подмассивов вдоль 3-й оси (глубина).
hsplit(ary, indices_or_sections)	Разбить массив на несколько вложенных массивов по горизонтали (по столбцам).
vsplit(ary, indices_or_sections)	Разбить массив на несколько подмассивов по вертикали (по строкам).
tile(A, reps)	Создать массив, повторяя ***A*** заданное ***reps*** количество раз,.
repeat(a, repeats[, axis])	Повторить элементы массива.

Примеры разделения массивов

x = np.arange(12.0) # [0. 1. 2. … 7. 8. 9. 10. 11.]
np.split(x, 3) # [array([0., 1., 2., 3.]), array([4., 5., 6., 7.]), array([8., 9., 10., 11.])]

- Если индекс секции представляет собой 1-мерный массив целых чисел, то его элементы показывают точки разбиения. Например, [2, 3] для axis=0 дадут разбиение:
- ary[:2]
- ary[2:3]
- ary[3:]

np.split(x, [2, 3])
 # [array([0., 1.]),
array([2.]),
array([3., 4., 5., 6., 7., 8., 9., 10., 11.])]

Размножение элементов массивов

```
a = np.array([0, 1, 2])
np.tile(a, 3)                # [0 1 2 0 1 2 0 1 2]
b = np.array([[1, 2], [3, 4]])
c= np.tile(b, 2)             # [[1 2 1 2]
                             # [3 4 3 4]]
a= np.repeat(2, 3)           # [2 2 2]
c = np.repeat(b, 3, axis=1)  # [[1 1 1 2 2 2]
                             # [3 3 3 4 4 4]]
c = np.repeat(b, 3, axis=0)  # [[1 2]
                             # [1 2]
                             # [1 2]
                             # [3 4]
                             # [3 4]
                             # [3 4]]
```

Вставка и удаление элементов; переразмеривание массивов

delete(arr, obj[, axis])	Новый массив с удаленными вдоль оси подмассивами.
insert(arr, obj, values[, axis])	Вставляет значения вдоль заданной оси перед указанными индексами.
append(arr, values[, axis])	Добавляет значения в конец массива.
resize(a, new_shape)	Возвращает новый массив с заданной формой.
trim_zeros(filt[, trim])	Обрезает начальные и/или конечные нули из одномерного массива или последовательности.
unique(ar[, return_index, return_inverse, …])	Находит уникальные элементы массива.
flip(m[, axis])	Обращает порядок элементов в массиве вдоль заданной оси.
fliplr(m)	Переворачивает массив влево / вправо.
flipud(m)	Переворачивает массив в направлении вверх / вниз.
reshape(a, newshape[, order])	Придает массиву новую форму без изменения его данных.
roll(a, shift[, axis])	Прокручивает элементы массива вдоль заданной оси.
rot90(m[, k, axes])	Поворачивает массив на 90 градусов в плоскости, указанной осями.

Примеры вставки и удаления-1

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
                # array([[ 1,  2,  3,  4],
                #        [ 5,  6,  7,  8],
                #        [ 9, 10, 11, 12]])
b = np.delete(a, 1, 0)  # array([[ 1,  2,  3,  4],
                #        [ 9, 10, 11, 12]])
a = np.array([[1,2], [3,4], [5,6]]) # array([[1, 2],
                        # [3, 4],
                            # [5, 6]])
b = np.insert(a, 1, 0) # array([1, 0, 2, 3, 4, 5, 6])
c = np.insert(a, 1, 7, axis=1)   # array([[1, 7, 2],
                            # [3, 7, 4],
                            #[5, 7, 6]])
```

Примеры вставки и удаления-2

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

Добавляем строку (ось 0)

```
e = np.append(a, [[7, 8]], axis=0) # array([[1, 2],
                                   # [3, 4],
                                   # [5, 6],
                                   # [7, 8]])
```

также см. метод ***concatenate***

Примеры преобразований массивов

```
a=np.array([[0, 1], [10, 11]])
c = np.resize(a, (2, 4))     # array([[ 0,  1, 10, 11],
                             #        [ 0,  1, 10, 11]])
a =np.arange(6).reshape((3, 2))  # array([[0, 1],
                             #        [2, 3],
                             #        [4, 5]])
```

- Отражение по строке (ось 0)

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

- Отражение по столбцу (ось 1)

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

Примеры прокрутки массивов

```
a = np.arange(10) #array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
b = np.roll(a, 2) #array([8, 9, 0, 1, 2, 3, 4, 5, 6, 7])
b = np.reshape(a, (2, 5)) # array([[0, 1, 2, 3, 4],
                          #        [5, 6, 7, 8, 9]])
c = np.roll(b, 1)         # array([[9, 0, 1, 2, 3],
                          #        [4, 5, 6, 7, 8]])
```

- Строки:

```
c = np.roll(b, 1, axis=0)  # array([[5, 6, 7, 8, 9],
                           #        [0, 1, 2, 3, 4]])
```

- Столбцы:

```
c = np.roll(b, 2, axis=1)  # array([[3, 4, 0, 1, 2],
                           #        [8, 9, 5, 6, 7]])
```

Примеры нахождения уникальных элементов

- Уникальные элементы

```
a = np.array([[0, 1], [1, 7]])
b = np.unique(a)        # array([0, 1, 7])
```

- Уникальные строки

```
a = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])
b = np.unique(a, axis=0)     # array([[1, 0, 0],
                  #              [2, 3, 4]])
```

- Уникальные столбцы

```
a = np.array([    [1, 1, 0],
                  [1, 1, 0],
                  [2, 2, 4]])
b = np.unique(a, axis=1)     # array([[0, 1],
                                    [0, 1],
                                    [4, 2]])
```

Операции ввода и вывода

load(file[, mmap_mode, allow_pickle, …])	Загружает массивы или сериализованные объекты из ***.npy***, ***.npz*** или сериализованных файлов.
save(file, arr[, allow_pickle, fix_imports])	Сохраняет массив в двоичный файл в формате NumPy ***.npy***.
savez(file, *args, **kwds)	Сохраняет несколько массивов в один файл в несжатом формате ***.npz.***
savez_compressed(file, *args, **kwds)	Сохраняет несколько массивов в один файл в сжатом формате ***.npz.***

Документацию по форматам файлов см. numpy.lib.format

Текстовые и двоичные файлы

fromfile(file[, dtype, count, sep])	Построить массив из данных в текстовом или двоичном файле.
ndarray.tofile(fid[, sep, format])	Записать массив в файл как текстовый или двоичный (по умолчанию).
loadtxt(fname[, dtype, comments, delimiter, …])	Загрузить данные из текстового файла.
savetxt(fname, X[, fmt, delimiter, newline, …])	Сохранить массив в текстовый файл.
genfromtxt(fname[, dtype, comments, …])	Загрузка данных из текстового файла с пропущенными значениями, обработанными как указано.
fromregex(file, regexp, dtype[, encoding])	Создать массив из текстового файла, используя синтаксический анализ регулярного выражения.
fromstring(string[, dtype, count, sep])	Новый одномерный массив, инициализированный из текстовых данных в строке.
ndarray.tofile(fid[, sep, format])	Записать массив в файл как текстовый или двоичный (по умолчанию).
ndarray.tolist()	Загрузить данные из текстового файла.

Линейная алгебра (numpy.linalg)

dot(a, b[, out])	Скалярное произведение двух массивов.
linalg.multi_dot(arrays)	Вычислить скалярное произведение двух или более массивов за один вызов функции, автоматически выбирая самый быстрый порядок оценки.
vdot(a, b)	Скалярное произведение двух векторов.
inner(a, b)	Внутреннее (скалярное) произведение двух массивов.
outer(a, b[, out])	Внешнее (скалярное) произведение двух векторов.
matmul(x1, x2, /[, out, casting, order, …])	Матричное произведение двух массивов.
tensordot(a, b[, axes])	Произведение тензорной точки вдоль указанных осей для массивов> = 1-D.
einsum(subscripts,*operands[, out, dtype, …])	Оценка соглашения суммирования Эйнштейна на операндах.
einsum_path(subscripts, *operands[, optimize])	Оценка порядка сокращения самой низкой стоимости для выражения einsum, рассматривая создание промежуточных массивов.
linalg.matrix_power(a, n)	Увеличение квадратной матрицы до (целочисленной) степени n.
kron(a, b)	Произведение Кронекера из двух массивов.

Линейная алгебра-2

linalg.cholesky(a)	Разложение Холецкого.
linalg.qr(a[, mode])	Вычислить QR-разложение матрицы.
linalg.svd(a[, full_matrices, compute_uv])	SVD-Разложение
linalg.eig(a)	Вычислить собственные значения и правые собственные векторы квадратного массива.
linalg.eigh(a[, UPLO])	Собственные значения и собственные векторы комплексного эрмитова (сопряженно-симметричного) или вещественной симметричной матрицы.
linalg.eigvals(a)	Вычислить собственные значения общей матрицы.
linalg.eigvalsh(a[, UPLO])	Вычислить собственные значения комплексной эрмитовой или вещественной симметричной матрицы.

Линейная алгебра-3

linalg.norm(x[, ord, axis, keepdims])	Матрица или векторная норма.
linalg.cond(x[, p])	Число обусловленности матрицы.
linalg.det(a)	Определитель массива.
linalg.matrix_rank(M[, tol, hermitian])	Возвращает матричный ранг массива, используя SVD-метод
linalg.slogdet(a)	Вычислить знак и (натуральный) логарифм определителя массива.
trace(a[, offset, axis1, axis2, dtype, out])	Сумма по диагоналей массива.
linalg.solve(a, b)	Решение линейного матричного уравнения или системы линейных скалярных уравнений.
linalg.tensorsolve(a, b[, axes])	Решение тензорного уравнения a x = b для x.
linalg.lstsq(a, b[, rcond])	Возвращает решение наименьших квадратов в линейное матричное уравнение.
linalg.inv(a)	Вычисляет (мультипликативную) обратную матрицу.
linalg.pinv(a[, rcond])	Вычисляет псевдообратную матрицу (Мура-Пенроуза).
linalg.tensorinv(a[, ind])	Вычисляет «инверсию» N-мерного массива.

Математические функции-1

sin(x, /[, out, where, casting, order, …])	Тригонометрический синус, поэлементно.
cos(x, /[, out, where, casting, order, …])	Косинус поэлементно.
tan(x, /[, out, where, casting, order, …])	Тангенс поэлементно.
arcsin(x, /[, out, where, casting, order, …])	Обратный синус, поэлементно.
arccos(x, /[, out, where, casting, order, …])	Обратный косинус, поэлементный.
arctan(x, /[, out, where, casting, order, …])	Тригонометрическая обратный тангенс, поэлементный.
hypot(x1, x2, /[, out, where, casting, …])	Возвращает гипотенузу по катетам прямоугольного треугольника
arctan2(x1, x2, /[, out, where, casting, …])	Поэлементный арктангенс x1/x2 выбирая правильный квадрант.
degrees(x, /[, out, where, casting, order, …])	Перевести углы в радианы в градусы.
radians(x, /[, out, where, casting, order, …])	Преобразование углов из градусов в радианы.
unwrap(p[, discont, axis])	Развёртка путём изменения дельты между значениями до 2*Pi дополнения.
deg2rad(x, /[, out, where, casting, order, …])	Преобразование углов из градусов в радианы.
rad2deg(x, /[, out, where, casting, order, …])	Перевести углы в радианы в градусы.

Математические функции-2

sinh(x, /[, out, where, casting, order, …])	Гиперболический синус, поэлементно.
cosh(x, /[, out, where, casting, order, …])	Гиперболический косинус, поэлементно.
tanh(x, /[, out, where, casting, order, …])	Вычислить гиперболический тангенс поэлементно.
arcsinh(x, /[, out, where, casting, order, …])	Обратный гиперболический синус поэлементно.
arccosh(x, /[, out, where, casting, order, …])	Обратный гиперболический косинус, поэлементно.
arctanh(x, /[, out, where, casting, order, …])	Обратный гиперболический тангенс поэлементно.
around(a[, decimals, out])	Равномерно округлить до указанного числа десятичных знаков.
round_(a[, decimals, out])	Округлить массив до указанного числа десятичных знаков.
rint(x, /[, out, where, casting, order, …])	Округлить элементы массива до ближайшего целого числа.
fix(x[, out])	Округлить до ближайшего целого числа до нуля.
floor(x, /[, out, where, casting, order, …])	Возвращает нижний уровень ввода поэлементно.
ceil(x, /[, out, where, casting, order, …])	Возвращает верхний уровень ввода поэлементно.
trunc(x, /[, out, where, casting, order, …])	Возвращаем усеченное значение ввода поэлементно.

Математические функции-3

exp(x, /[, out, where, casting, order, …])	Экспонента всех элементов входного массива.
expm1(x, /[, out, where, casting, order, …])	exp(x)–1 для всех элементов в массиве.
exp2(x, /[, out, where, casting, order, …])	2**p для всех p во входном массиве.
log(x, /[, out, where, casting, order, …])	ln(x)
log10(x, /[, out, where, casting, order, …])	lg(x)
log2(x, /[, out, where, casting, order, …])	$\log_2(x)$
log1p(x, /[, out, where, casting, order, …])	ln(1 + x)
logaddexp(x1, x2, /[, out, where, casting, …])	log(exp(x1) + exp(x2))
logaddexp2(x1, x2, /[, out, where, casting, …])	log2(2**x1 + 2**x2)

Остальные математические функции
https://docs.scipy.org/doc/numpy/reference/routines.math.html

Библиотека SciPy

https://docs.scipy.org/doc/scipy/reference/

- Специальные функции (scipy.special)
  - ✔Функции Бесселя
- Интегрирование (scipy.integrate)
- Оптимизация (scipy.optimize)
- Интерполяция (scipy.interpolate)
- Преобразования Фурье (scipy.fftpack)
- Обработка сигналов (scipy.signal)
- Линейная алгебра (scipy.linalg)
- Статистика (scipy.stats)
- Многомерная обработка изображений (scipy.ndimage)
- Файловый ввод-вывод (scipy.io)

Интегрирование-1

quad(func, a, b[, args, full_output, …])	Вычисляет определенный интеграл.
dblquad(func, a, b, gfun, hfun[, args, …])	Вычисляет двойной интеграл.
tplquad(func, a, b, gfun, hfun, qfun, rfun)	Вычисляет тройной (определенный) интеграл.
nquad(func, ranges[, args, opts, full_output])	Интегрирование по нескольким переменным.
fixed_quad(func, a, b[, args, n])	Вычисляет определенный интеграл, используя гауссову квадратуру определённого порядка.
quadrature(func, a, b[, args, tol, rtol, …])	Вычисляет определенный интеграл, используя гауссовскую квадратуру с фиксированным допуском.
romberg(function, a, b[, args, tol, rtol, …])	Интегрирование методом Ромберга вызываемой функции или метода.
quad_explain([output])	Вывести дополнительную информацию о параметрах ***integrate.quad ()*** и возвращаемых значениях.
newton_cotes(rn[, equal])	Возвращаемые веса и коэффициент ошибок для интегрировании квадратурами Ньютона-Котеса.
IntegrationWarning	Предупреждение о проблемах при интегрировании.

Интегрирование-2

trapz(y[, x, dx, axis])	Интегрирование вдоль заданной оси, используя составное трапециевидное правило.
cumtrapz(y[, x, dx, axis, initial])	Кумулятивное интегрирование y (x), используя составное правило трапеции.
simps(y[, x, dx, axis, even])	Интегрирование y (x), используя выборки вдоль заданной оси и составное правило Симпсона.
romb(y[, dx, axis, show])	Интегрирование Ромберга с использованием примеров функции.

Решение ОДУ

solve_ivp(fun, t_span, y0[, method, t_eval, …])	Решает систему ОДУ.
RK23(fun, t0, y0, t_bound[, max_step, rtol, …])	Явный метод Рунге-Кутты порядка 3 (2).
RK45(fun, t0, y0, t_bound[, max_step, rtol, …])	Явный метод Рунге-Кутты порядка 5 (4).
Radau(fun, t0, y0, t_bound[, max_step, …])	Неявный метод Рунге-Кутты семейства Радау IIA 5-го порядка.
BDF(fun, t0, y0, t_bound[, max_step, rtol, …])	Неявный метод, основанный на формулах обратного дифференцирования.
LSODA(fun, t0, y0, t_bound[, first_step, …])	Метод Адамса / BDF с автоматическим определением жесткости и переключением.
ODESolver(fun, t0, y0, t_bound, vectorized)	Базовый класс для ОДУ решателей.
DenseOutput(t_old, t)	Базовый класс для локального интерполяции на шаге, сделанный решателем ОДУ.
ODESolution(ts, interpolants)	Непрерывное решение ОДУ.

Оптимизация

show_options([solver, method, disp])	Показать документацию для дополнительных опций решателей оптимизации.
OptimizeResult	Представляет результат оптимизации.
minimize_scalar(fun[, bracket, bounds, …]) ***·minimize_scalar(method='brent')*** ***·minimize_scalar(method='bounded')*** ***·minimize_scalar(method='golden')***	Минимизация скалярной функции одной переменной. с
minimize(fun, x0[, args, method, jac, hess, …]) ***·minimize(method='Nelder-Mead')*** ***·minimize(method='Powell')*** ***·minimize(method='CG')*** ***·minimize(method='BFGS')*** ***·minimize(method='Newton-CG')*** ***·minimize(method='L-BFGS-B')*** ***·minimize(method='TNC')*** ***·minimize(method='COBYLA')*** ***·minimize(method='SLSQP')*** ***·minimize(method='trust-constr')*** ***·minimize(method='dogleg')*** ***·minimize(method='trust-ncg')*** ***·minimize(method='trust-krylov')*** ***·minimize(method='trust-exact')***	Минимизация скалярной функции одной или нескольких переменных. https://docs.scipy.org/doc/scipy/reference/optimize.html#module-scipy.optimize

Ограничения

- Ограничения передаются, чтобы минимизировать функцию как отдельный объект или как список объектов из следующих классов:
  NonlinearConstraint(fun, lb, ub [, jac,…]) – Нелинейное ограничение на переменные.
  LinearConstraint (A, lb, ub [, keep_feasible]) – Линейное ограничение на переменные.
- Простые связанные ограничения обрабатываются отдельно, и для них есть специальный класс:
  Bounds(lb, ub [, keep_feasible]) – Ограничение границ для переменных.

Случайный поиск, МНК и пр.

basinhopping(func, x0[, niter, T, stepsize, …])	Находит глобальный минимум функции, используя алгоритм случайного спуска по локальным минимумам (*basin-hopping algorithm*)
brute(func, ranges[, args, Ns, full_output, …])	Минимизирование функции в заданном диапазоне перебором (*brute force*).
differential_evolution(func, bounds[, args, …])	Находит глобальный минимум многомерной функции.
shgo(func, bounds[, args, constraints, n, …])	Находит глобальный минимум функции, используя оптимизацию SHG.
dual_annealing(func, bounds[, args, …])	Найти глобальный минимум функции с помощью алгоритма имитации отжига (*Dual Annealing*)
least_squares(fun, x0[, jac, bounds, …])	Решает нелинейную задачу наименьших квадратов с оценками переменных.
nnls(A, b[, maxiter])	Решает ***argmin_x || Ax - b || _2*** для ***x> = 0***.
lsq_linear(A, b[, bounds, method, tol, …])	Решает линейную задачу наименьших квадратов с оценками переменных.
curve_fit(f, xdata, ydata[, p0, sigma, …])	Использует нелинейные наименьшие квадраты, чтобы подогнать функцию ***f*** к данным.

Нахождение корней нелинейных уравнений

root_scalar(f[, args, method, bracket, …]) ***root_scalar(method='brentq')*** ***root_scalar(method='brenth')*** ***root_scalar(method='bisect')*** ***root_scalar(method='ridder')*** ***root_scalar(method='newton')*** ***root_scalar(method='toms748')*** ***root_scalar(method='secant')*** ***root_scalar(method='halley')***	Находит корень скалярной функции.
brentq(f, a, b[, args, xtol, rtol, maxiter, …])	Находит корень функции на интервале, используя метод Брента.
brenth(f, a, b[, args, xtol, rtol, maxiter, …])	Находит корень функции на интервале, используя метод Брента с гиперболической экстраполяцией.
ridder(f, a, b[, args, xtol, rtol, maxiter, …])	Находит корень функции в интервале, используя метод Риддера.
bisect(f, a, b[, args, xtol, rtol, maxiter, …])	Находит корень функции в пределах интервала, используя деление пополам.
newton(func, x0[, fprime, args, tol, …])	Находит ноль вещественной или сложной функции, используя метод Ньютона-Рафсона (или секущегося, или Галлея).
toms748(f, a, b[, args, k, xtol, rtol, …])	Найти ноль, используя метод TOMS Algorithm 748.
RootResults(root, iterations, …)	Представляет результат поиска корня.

Нахождение корней-2

fixed_point(func, x0[, args, xtol, maxiter, …])	Находит фиксированную точку функции. Т.е. где func(x0) == x0
root(fun, x0[, args, method, jac, tol, …]) ***root(method='hybr')*** ***root(method='lm')*** ***root(method='broyden1')*** ***root(method='broyden2')*** ***root(method='anderson')*** ***root(method='linearmixing')*** ***root(method='diagbroyden')*** ***root(method='excitingmixing')*** ***root(method='krylov')*** ***root(method='df-sane')***	Находит корень вектор-функции.

См. остальные функции раздела
https://docs.scipy.org/doc/scipy/reference/optimize.html#module-scipy.optimize

Модуль pickle

- Модуль **pickle** реализует мощный алгоритм сериализации и десериализации
- Pickling – процесс преобразования объекта Python в поток байтов (сериализация), а ***unpickling*** – обратная операция (десериализация), в результате которой поток байтов преобразуется обратно в Python-объект. Поток байтов легко можно записать в файл и модуль ***pickle*** широко применяется для сохранения и загрузки сложных объектов в Python.

С чем работает модуль pickle

Какие типы данных Pickle умеет запаковывать?

- ***None, True, False***
- Строки (обычные или Unicode)
- Стандартные числовые типы данных
- Словари, списки, кортежи
- Функции
- Классы

Функции модуля pickle

pickle.dump(obj, file, protocol=None, *, fix_imports=True)	Записывает сериализованный объект в файл. аргумент *protocol* указывает используемый протокол. По умолчанию =3 и он рекомендован для использования в Python 3 (несмотря на то, что в Python 3.4 добавили протокол версии 4 с некоторыми оптимизациями). Записывать и загружать надо с одним и тем же протоколом.
pickle.dumps(obj, protocol=None, *, fix_imports=True)	Возвращает сериализованный объект.
pickle.load(file, *, fix_imports=True, encoding="ASCII", errors="strict")	Загружает объект из файла.
pickle.loads(bytes_object, *, fix_imports=True, encoding="ASCII", errors="strict")	Загружает объект из потока байтов.

Пример записи/считывания

```
d = {"a": [1, 2.0, 3, 4+6j],
     "b": ("строка", b"byte string"),
     "c": {None, True, False}
     }
fil = "dt.pickle"
with open(fil, "wb") as f:
    pickle.dump(d, f)
with open(fil, "rb") as f:
    d_new = pickle.load(f)
print(d_new)
# {'a': [1, 2.0, 3, (4+6j)], 'b': ('строка', b'byte string'), 'c':
{False, True, None}}
```

Внимание! Не загружайте pickle-файлы из неавторизованных источников!

Документация https://docs.python.org/3/library/pickle.html

Спасибо за внимание