**NumPy** (сокращенно от *Numerical Python*)— библиотека с открытым исходным кодом для языка программирования [Python](https://ru.wikipedia.org/wiki/Python). Возможности:

* поддержка многомерных массивов (включая матрицы);
* поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

## История

В 1995 году программист Jim Hugunin написал библиотеку Numeric для языка Python. Библиотека развивалась при участии многих людей, среди которых были Jim Fulton, David Ascher, Paul DuBois и Konrad Hinsen. Библиотека доступна по сей день, считается вполне стабильной и полной, но устаревшей.

Предлагалось добавить Numeric в стандартную библиотеку языка Python, но Гвидо Ван Россум (автор Python) чётко дал понять, что код в его тогдашнем состоянии было невозможно поддерживать.

Кроме того, библиотека Numeric медленно обрабатывала большие массивы данных.

На основе библиотеки Numeric была создана библиотека NumArray. Код Numeric был полностью переписан.

Библиотека NumArray обрабатывала большие массивы данных быстрее библиотеки Numeric, но малые массивы обрабатывала медленнее.

Некоторое время использовалась и библиотека Numeric, и библиотека NumArray. Последняя версия Numeric (v24.2) была выпущена 11 ноября 2005 года. Последняя версия NumArray (v1.5.2) вышла 24 августа 2006 года[[5]](https://ru.wikipedia.org/wiki/NumPy#cite_note-5). Библиотека NumArray более не рекомендуется к использованию[[6]](https://ru.wikipedia.org/wiki/NumPy#cite_note-6).

В начале 2005 года программист Трэвис Олифант захотел объединить сообщество вокруг одного проекта и для замены библиотек Numeric и NumArray создал библиотеку NumPy. NumPy был создан на основе кода Numeric. Код Numeric был переписан так, чтобы его было легче поддерживать, и в библиотеку можно было добавить новые возможности. Возможности NumArray были добавлены в NumPy.

Изначально NumPy был частью библиотеки SciPy. Чтобы позволить другим проектам использовать библиотеку NumPy, её код был помещён в отдельный пакет.

Исходный код NumPy находится в открытом доступе. Существует большое количество документации. Имеется даже подробный «*Путеводитель по NumPy*»[[7]](https://ru.wikipedia.org/wiki/NumPy#cite_note-7).

NumPy v1.3.0 выпущен 5 апреля 2009 года и поддерживает Python v2.6[[8]](https://ru.wikipedia.org/wiki/NumPy#cite_note-8). Поддержка Python v3 была добавлена начиная с версии 1.5.0.

# SciPy

## История

В 1990-х годах Python был расширен типом массива для вычислений под названием Numeric (этот пакет в конечном итоге был заменен Трэвисом Олифантом, который написал [NumPy](https://ru.wikipedia.org/wiki/NumPy)[[4]](https://ru.wikipedia.org/wiki/SciPy#cite_note-4)[[5]](https://ru.wikipedia.org/wiki/SciPy#cite_note-5) в 2006 году, соединив Numeric и Numarray.[[6]](https://ru.wikipedia.org/wiki/SciPy#cite_note-6)

По состоянию на 2000 год число модулей росло и возрастал интерес к созданию полноценной среды для научных и технических вычислений. В 2001 году Трэвис Олифант, Эрик Джонс и Пиару Петерсон объединили написанный ими код и назвали получившийся пакет SciPy. Вскоре после этого Фернандо Перес выпустил IPython, расширенную интерактивную оболочку, широко используемую в сообществе технических вычислений, а Джон Хантер выпустил первую версию Matplotlib, библиотеки 2D-графиков для вычислений. С тех пор среда SciPy продолжала расти с появлением большего количества пакетов и инструментов для технических вычислений.[[7]](https://ru.wikipedia.org/wiki/SciPy#cite_note-7)

## Возможности

* поиск минимумов и максимумов функций;
* вычисление интегралов функций;
* поддержка специальных функций;
* обработка сигналов;
* обработка изображений;
* работа с генетическими алгоритмами;
* решение обыкновенных дифференциальных уравнений;
* и др.

Целевая аудитория — пользователи продуктов MATLAB и Scilab.

Для визуализации результатов расчётов часто применяется библиотека *Matplotlib*, являющаяся аналогом средств вывода графики MATLAB.

Библиотека SciPy распространяется по условиям лицензии BSD. Разработчиков финансирует фирма «[Enthought](https://ru.wikipedia.org/w/index.php?title=Enthought&action=edit&redlink=1)».

conda install numpy

или

pip install numpy

import numpy as np

**Главный объект библиотеки NumPy — массив.**

# **В чем разница между списком Python и массивом NumPy?**

NumPy предоставляет вам огромный выбор быстрых и эффективных числовых опций. Хотя список Python может содержать разные типы данных в одном списке, все элементы в массиве NumPy должны быть однородными. Математические операции, которые должны выполняться над массивами, были бы невозможны, если бы они не были однородными.

Массивы NumPy быстрее и компактнее, чем списки Python. Массив потребляет меньше памяти и намного удобнее в использовании. NumPy использует гораздо меньше памяти для хранения данных и предоставляет механизм задания типов данных, который позволяет оптимизировать код еще дальше.

# **Как создать базовый массив**

np.array()  
np.zeros()  
np.ones()  
np.empty()  
np.arange()  
np.linspace()  
dtype

Чтобы создать массив NumPy, вы можете использовать функцию **np.array ()**,

Все, что вам нужно сделать, чтобы создать простой массив, это передать ему список. При желании вы также можете указать тип данных в вашем списке

import numpy as np

a = np.array([1, 2, 3])

Вы можете визуализировать ваш массив следующим образом:



Помимо создания массива из последовательности элементов, вы можете легко создать массив, заполненный **0**:

**Входные данные:**

np.zeros(2)

**Выход:**

array([0., 0.])

Или массив заполнен **1s**:

**Входные данные:**

np.ones(2)

**Выход:**

array([1., 1.])

Или даже **опорожнить** массив! Функция *опорожнить* создает массив, исходное содержимое которого является случайным и зависит от состояния памяти.

**Входные данные:**

# Create an empty array with 2 elements  
np.empty(2)

Вы можете создать массив с диапазоном элементов:

**Входные данные:**

np.arange(4)

**Выход:**

array([0, 1, 2, 3])

И даже массив, который содержит диапазон равномерно распределенных интервалов. Для этого вам нужно будет указать **первый** количество, **прошлой** номер, а **размер шага**,

**Входные данные:**

np.arange(2,9,2)

**Выход:**

array([2, 4, 6, 8])

Вы также можете использовать**np.linspace ()**создать массив со значениями, которые расположены линейно в заданном интервале:

**Входные данные:**

np.linspace(0,10,5)

**Выход:**

array([ 0. , 2.5, 5. , 7.5, 10. ])

**Указание вашего типа данных**

Хотя тип данных по умолчанию - с плавающей запятой (float64), вы можете явно указать, какой тип данных вы хотите использовать, используя dtype.

**Входные данные:**

array = np.ones(2, dtype=int)

**Выход:**

array([1, 1])

# **Добавление, удаление и сортировка элементов**

* np.append()
* np.delete()
* np.sort()

# **Как вы знаете форму и размер массива?**

ndarray.ndim()  
ndarray.size()  
ndarray.shape()

**ndarray ndim** скажет вам количество осей или размеров массива.

**ndarray.size** скажет вам общее количество элементов массива.

**ndarray.shape** будет отображать кортеж целых чисел, которые указывают количество элементов, хранящихся вдоль каждого измерения массива. Если, например, у вас есть 2D-массив с 2 строками и 3 столбцами, форма вашего массива будет (2,3).

# **Случайные числа**

Важная часть каждой симуляции это способность генерировать случайные числа. Для этого мы используем встроенный в NumPy генератор псевдослучайных чисел в под-модуле random. Числа являются *псевдо* случайными, в том плане что, они сгенерированы детерминистически из порождающего элемента (seed number), но рассредоточены в статистическом сходстве с случайным образом. Для генерации NumPy использует особенный алгоритм который имеет название Mersenne Twister.

Задать порождающий элемент последовательности случайных чисел можно так:

>>> np.random.seed(293423)

Seed это целое число. Каждая программа которая запускается с одинаковым seed`ом будет генерировать одинаковую последовательность чисел каждый раз. Это может быть полезно для отладки, но вообще нам не нужно задавать seed, на самом деле, когда мы запускаем программу несколько раз, мы хотим получать каждый раз разную последовательность чисел. Если эта команда не будет выполнена, то NumPy автоматически выбирает случайный seed (базирующийся на времени), который является разным при каждом запуске программы.

Массив случайных чисел из полуинтервала [0.0, 1.0) может быть сгенерирован так:

>>> np.random.rand(5)

array([ 0.40783762, 0.7550402 , 0.00919317, 0.01713451, 0.95299583])

Функция rand может быть использована для генерации двумерных массивов, или можно использовать функцию reshape:

>>> np.random.rand(2,3)

array([[ 0.50431753, 0.48272463, 0.45811345], [ 0.18209476, 0.48631022, 0.49590404]])

>>> np.random.rand(6).reshape((2,3))

array([[ 0.72915152, 0.59423848, 0.25644881], [ 0.75965311, 0.52151819, 0.60084796]])

Для генерации единичного случайного числа на интервале [0.0, 1.0):

>>> np.random.random() 0.70110427435769551

Для генерации случайного целочисленного числа в диапазоне [min, max) используем функцию randint(min, max):

>>> np.random.randint(5, 10) 9

В каждом нашем примере, мы генерировали числа из непрерывного равномерного распределения. NumPy также включает генераторы для других распределений, таких как: Бета, биномиальное, хи-квадрат, Дирихле, экспоненциальное, Фишера, Гамма, геометрическое, Гамбала, гипергеометрическое, Лапласа, логистическое, логнормальное, логарифмическое, мультиномиальное, многомерное нормальное, отрицательное биномиальное, нецентральное хи-квадрат, нецентральное Фишера, нормальное (Гаусса), Парето, Пуассона, степенное, Рэлея, Коши, Стьюдента, треугольное, Фон-Миса, Вальда, Вейбулла и Ципфа. Рассмотрим два примера.

Для генерации из дискретного распределения Пуассона при λ = 6.0,

>>> np.random.poisson(6.0)

5

Для генерации числа из нормального распределения (Гаусса) при среднем значении μ = 1.5 и стандартной девиации σ = 4.0:

>>> np.random.normal(1.5, 4.0)

0.83636555041094318

Для получении числа из нормального распределения (μ = 0, σ = 1), без указания аргументов:

>>> np.random.normal()

0.27548716940682932

Для генерации нескольких значений используем аргумент size:

>>> np.random.normal(size=5)

array([-1.67215088, 0.65813053, -0.70150614, 0.91452499, 0.71440557])

Модуль для генерации случайных чисел также может быть использован для случайного распределения значений в списке. Это может быть полезно если мы хотим случайно распределить значения в списке:

>>> l = range(10)

>>> l [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> np.random.shuffle(l)

>>> l [4, 9, 5, 0, 2, 7, 6, 8, 1, 3]

Заметим, что функция shuffle модифицирует уже существующий массив и не возвращает новый.

# **Некоторая дополнительная информация**

NumPy включает еще много других функций о которых мы не упоминали здесь. В частности это функции для работы с дискретным преобразованием Фурье, более сложными операциями в линейной алгебре, тестированием массивов на размер / размерность / тип, разделением и соединением массивов, гистограммами, создания массивов из каких-либо данных разными путями, созданием и оперированием grid-массивов, специальными значениями (NaN, Inf), set-операции, созданием разных видов специальных матриц и вычислением специальных математических функций (Например: функции Бесселя).

# **Модули SciPy**

SciPy очень хорошо расширяет функционал NumPy. Мы не будем говорить о его деталях, но рассмотрим некоторые его возможности. Большинство функций SciPy доступны после импорта модуля:

Функция help обеспечит полезной информацией о SciPy:

>>> help(scipy)

Help on package scipy:

NAME

scipy

FILE

c:\python25\lib\site-packages\scipy\\_\_init\_\_.py

DESCRIPTION

SciPy --- A scientific computing package for Python

===================================================

Documentation is available in the docstrings and

online at http://docs.scipy.org.

Contents

--------

SciPy imports all the functions from the NumPy namespace, and in

addition provides:

Available subpackages

---------------------

odr --- Orthogonal Distance Regression [\*]

misc --- Various utilities that don't have

another home.sparse.linalg.eigen.arpack --- Eigenvalue solver using iterative methods. [\*]

fftpack --- Discrete Fourier Transform algorithms[\*]

io --- Data input and output [\*]

sparse.linalg.eigen.lobpcg --- Locally Optimal Block Preconditioned Conjugate Gradient Method (LOBPCG) [\*]

special --- Airy Functions [\*]

lib.blas --- Wrappers to BLAS library [\*]

sparse.linalg.eigen --- Sparse Eigenvalue Solvers [\*]

stats --- Statistical Functions [\*]

lib --- Python wrappers to external libraries [\*]

lib.lapack --- Wrappers to LAPACK library [\*]

maxentropy --- Routines for fitting maximum entropymodels [\*]

integrate --- Integration routines [\*]

ndimage --- n-dimensional image package [\*]

linalg --- Linear algebra routines [\*]

spatial --- Spatial data structures and algorithms[\*]

interpolate --- Interpolation Tools [\*]

sparse.linalg --- Sparse Linear Algebra [\*]

sparse.linalg.dsolve.umfpack --- :Interface to the UMFPACK library: [\*]

sparse.linalg.dsolve --- Linear Solvers [\*]

optimize --- Optimization Tools [\*]

cluster --- Vector Quantization / Kmeans [\*]

signal --- Signal Processing Tools [\*]

sparse --- Sparse Matrices [\*]

[\*] - using a package requires explicit import (see pkgload)

...

Заметим, что некоторым под-модулям нужен непосредственно дополнительный импорт, которые помечены звездой:

>>> **import** scipy

>>> **import** scipy.interpolate

Функции в каждом модуле хорошо задокументированы во внутренних docstring`ах и в официальной документации. Большинство из них непосредственно предоставляет функции для работы с числовыми алгоритмами и они очень просты в использовании. Таким образом, SciPy может сохранять гигантское количество времени в научных вычислениях, т.к. он обеспечивает уже написанные и тестированные функции.

Мы не будем рассматривать SciPy детально, но таблица ниже покроет некоторые его возможности:

|  |  |
| --- | --- |
| **Модуль** | **Для чего используется** |
| scipy.constants | Набор математических и физических констант |
| scipy.special | Много специальных функций для математической физики, такие как: Эйри, эллиптические, Бесселя, гамма, бета, гипергеометрические, параболического цилиндра, Матьё, шаровидной волны, Струве, Кельвина. |
| scipy.integrate | Функции для работы с численным интегрированием используя методы трапеции, Симпсона, Ромберга и другие. Также предоставляет методы для работы с полными дифференциальными уравнениями. |
| scipy.optimize | Стандартные методы поиска максимума/минимума для работы с обобщенными пользовательскими функциями. Включенные алгоритмы: Нелдера — Мида, Поулла (*Powell's*), сопряженных градиентов, Бройдена — Флетчера — Гольдфарба — Шанно, наименьших квадратов, условной оптимизации, имитации отжига, полного перебора, Брента, Ньютона, бисекции, Бройдена, Андерсона и линейного поиска. |
| scipy.linalg | Более широкий функционал для работы с линейной алгеброй чем в NumPy. Предоставляет больше возможностей для использования специальных и быстрых функций, для специфических объектов (Например: трёхдиагональная матрица). Включенные методы: поиск невырожденной матрицы, поиск определителя, решение линейных систем уравнений, расчета норм и псевдообратной матрицы, спектрального разложения, сингулярного разложения, LU-разложения, разложения Холецкого, QR-разложения, разложения Шура и много других математических операций для работы с матрицами. |
| scipy.sparse | Функции для работы с большими разреженными матрицами |
| scipy.interpolate | Методы и классы для интерполяции объектов, которые могут быть использованы для дискретных числовых данных. Линейная и сплайновая (*Прим. переводчика: математическое представление плавных кривых*) интерполяция доступна для одно- и двух-мерных наборов данных. |
| scipy.fftpack | Методы для обработки преобразований Фурье. |
| scipy.signal | Методы для обработки сигналов, например: свертка функций, корреляция, дискретное преобразование Фурье, сглаживающий B-сплайн, фильтрация, и т.д и т.п. |
| scipy.stats | Большая библиотека статистических функций и распределений для работы с наборами данных. |

Большая группа разработчиков непрерывно продолжает разрабатывать новый функционал SciPy. Хороший практический подход это: если вы думаете о реализации каких-либо числовых функций и методов в вашем коде, можете сначала заглянуть в оригинальную документацию SciPy. Есть вероятность того, что это уже кто-то реализовал и внес в SciPy.