ГЛАВА 4.

**Основы NumPy: массивы и векторные вычисления**

Numerical Python, или сокращенно NumPy - краеугольный пакет для высокопроизводительных научных расчетов и анализа данных. Это фундамент, на котором возведены почти все описываемые в этой книге высокоуровневые инструменты. Вот лишь часть того, что предлагается.

* ndarray, быстрый и потребляющий мало памяти многомерный массив, предоставляющий векторные арифметические операции и возможность укладывания.
* Стандартные математические функции для выполнения быстрых операций над целыми массивами без явного выписывания циклов.
* Средства для чтения массива данных с диска и записи его на диск, а также для работы с проецируемыми на память файлами.
* Алгоритмы линейной алгебры, генерация случайных чисел и преобразование Фурье.
* Средства для интеграции с кодом, написанным на С, С++ или Fortran

Последний пункт - один из самых важных с точки зрения экосистемы. Благодаря наличию простого С API в NumPy очень легко передавать данные внешним библиотекам, написанным на языке низкого уровня, а также получать от внешних библиотек данные в виде массивов NuшPy. Эта возможность сделала Python излюбленным языком для обертывания имеющегося кода на С/С++ /Fortran с приданием ему динамического и простого в использовании интерфейса.

Хотя сам по себе пакет NumPy почти не содержит средств высокоуровневого анализа данных, понимание массивов NumPy и ориентированных на эти массивы вычислений поможет гораздо эффективнее использовать инструменты типа pandas. Если вы только начинаете изучать Python и просто хотите познакомиться с тем, как pandas позволяет работать с данными, можете лишь бегло просмотреть эту главу. Более сложные средства NumPy, в частности укладывание, рассматриваются в главе 12.

В большинстве приложений для анализа данных основной интерес представляет следующая функциональность:

* быстрые векторные операции для переформатирования и очистки данных, выборки подмножеств и фильтрации, преобразований и других видов вы­числений;
* стандартные алгоритмы работы с массивами, например: фильтрация, удаление дубликатов и теоретико-множественные операции;
* эффективная описательная статистика, агрегирование и обобщение данных;
* выравнивание данных и реляционные операции объединения и соедине­ния разнородных наборов данных;
* описание условной логики в виде выражений-массивов вместо циклов с ветвлением if-elif-else;
* групповые операции с данными (агрегирование, преобразование, применение функции). Подробнее об этом см. главу 5

И в этой главе, и далее в книге я использую стандартное принятое в NumPy соглашение - всегда включать предложение import numpy as пр. Конеч­ но, никто не мешает добавить в программу предложение from numpy import \*, чтобы не писать всюду пр., но это дурная привычка, от которой я хотел бы вас предостеречь.

NumPy ndarray: объект многомерного массива

Одна из ключевых особенностей NumPy - объект ndarray для представления N-мерного массива; это быстрый и гибкий контейнер для хранения больших наборов данных в Python. Массивы позволяют выполнять математические операции над целыми блоками данных, применяя такой же синтаксис, как для соответствующих операций над скалярами:

In [8]: data

Out [8]:

array {((0.9526, -0.246, -О.8!356],

[ 0.5639, 0.2379, 0.9104]])

In [9]: data \* 10 In [10]: data + data

Out [9]: Out [10]:

array ([[9.5256, -2.4601, -8.8565], array ([(1.9051, -0.492 , -1.7713],

[5.6385, 2.3794, 9.104]]) [1.1277, 0.4759, 1.8208]])

ndarray — это обобщенный многомерный контейнер для однородных данных, т. е. в нем могут храниться только элементы одного типа. У любого массива есть атрибут shape - кортеж, описывающий размер по каждому измерению, и атрибут dtype - объект, описывающий тип данных в массиве:

In [11]: data.shape

Out [11]: (2, 3)

In [12]: data.dtype

Out [12]: dtype('float64')

В этой главе вы познакомитесь с основами работы с массивами NumPy в объеме, достаточном для чтения книги. Для многих аналитических приложений глубокое понимание NumPy необязательно, но овладение стилем мышления и методами программирования, ориентированными на массивы, - ключевой этап на пути ста­новления эксперта по применению Python в научных приложениях.

Слова «массив», «массив NumPy» и «ndarray» в этой книге почти всегда означают одно и то же: объект ndarray.

Создание ndarray

Проще всего создать массив с помощью функции array. Она принимает любой объект, похожий на последовательность (в том числе другой массив), и порождает новый массив NнmPy, содержащий переданные данные. Например, такое преобразование можно проделать со списком:

In [13]: datal = [ 6, 7. 5, 8, 0, 1)

In [14]: arrl = np.array(datal)

In [15]: arrl

Out [15]: array ([ 6. , 7.5, 8. , 0. , 1. ])

Вложенные последовательности, например список списков одинаковой длины, можно преобразовать в многомерный массив:

In (16]: data2 = ((1, 2, 3, 4], (5, 6, 7, 8]]

In (17]: arr2 = np.array(data2)

In (17]: arr2 = np.array(data2)

Out [18]:

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

[5, 6, 7, 8]])

In [19]: arr2.ndim

Out [19]: 2

In [20]: arr2.shape

Out [2 0]: (2, 4)

Если не определено явно (подробнее об этом ниже), то функция np. array пытается самостоятельно определить подходящий тип данных для создаваемого массива. Этот тип данных хранится в специальном объекте dtype; например, в примерах выше имеем:

In [21]: arrl.dtype

Out [21]: dtype('float64')

In [22]: arr2.dtype

Out [22]: dtype('int64')

Помимо np.array, существует еще ряд функций для создания массивов. Например, zeros и ones создают массивы заданной длины, состоящие из нулей и единиц соответственно, а shape. empty создает массив, не инициализируя его элементы. Для создания многомерных массивов нужно передать кортеж, описывающий форму:

In [23]: np.zeros(10)

Out (23]: array ([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,])

In [24]: пр. zeros ((3, 6)

Out [24]:

array ([ [ 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0.]])

In [25] : пр. empty ( ( 2, 3, 2) )

Out [25):

array ( [ [ [4.94065646е-324, 4.94065646е-324],

[3.87491056е-297, 2 .46845796е-130],

[4.94065646е-324, 4.94065646е-324]],

[[1.90723115е+О83, 5. 73293533е-053],

[-2.33568637е+124, -6.70608105е-012]

[4.42786966е+160, 1.27100354е+025]]])



Предполагать, что пр. empty возвращает массив из одних нулей, небезопасно. Часто возвращается массив, содержащий неинициализированный мусор, - как в примере выше.

Функция arange - вариант встроенной в Python функции range, только возвращаемым значением является массив:

In [26}: np.arange(15)

Out [26}: array ([ О, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14})

В табл. 4.1 приведен краткий список стандартных функций создания массива. Поскольку NumPy ориентирован, прежде всего, на численные расчеты, тип данных, если он не указан явно, во многих случаях предполагается float64 (числа с плавающей точкой).

Таблица 4.1. Функции создания массива

|  |  |
| --- | --- |
| Функция | Описание |
| array | Преобразует входные данные (список, кортеж, массив или любую другую последовательность) в ndarray. Тип dtype задается явно или выводится неявно. Входные данные по умолчанию копируются |
| array | Преобразует входные данные в ndarray, но не копирует, если на вход уже подан ndarray |
| arange | Аналогична встроенной функции range, но возвращает массив, а не список |
| ones, ones like | Порождает массив, состоящий из одних единиц, с заданными атрибутами shape и dtype. Функция ones\_like принимает другой массив и порождает массив из одних единиц с такими же значениями shape и dtype |
| zeros, zeros like | Аналогичны ones и ones\_like, только порождаемый массив состоит из одних нулей |
| empty, empty \_like | Создают новые массивы, выделяя под них память, но, в отличие от ones и zeros, не инициализируют ее |
| empty, empty \_like | Создают единичную квадратную матрицу N х N (элементы на главной диагонали равны 1, все остальные - О) |

Тип данных для ndarray

Тип данных, или dtype — это специальный объект, который содержит информацию, необходимую ndarray для интерпретации содержимого блока памяти:

In [27]: arrl np.array([l, 2, 3], dtype=np.float64)

In [28]: arr2 np.array([l, 2, 3], dtype=np.int32)

In [29]: arrl.dtype In (30]: arr2.dtype

Out [29]: dtype('float64') Out [30]: dtype('int32')

Объектам dtype NumPy в значительной мере обязан своей эффективностью и гибкостью. В большинстве случаев они точно соответствуют внутреннему машинному представлению, что позволяет без труда читать и записывает двоичные потоки данных на диск, а также обмениваться данными с кодом, написанным на языке низкого уровня типа С или Fortran. Числовые dtype именуются единообразно: имя типа, например float или int, затем число, указывающее разрядность одного элемента. Стандартное значение с плавающей точкой двойной точности (хранящееся во внутреннем представлении объекта Pytlюn типа float) занимает 8 байтов или 64 бита. Поэтому соответствующий тип и NumPy называется float64. В табл. 4.2 приведен полный список поддерживаемых NuшPy типов данных.

Не пытайтесь сразу запомнить все типы данных NumPy, особенно если вы только приступаете к изучению. Часто нужно заботиться только об общем виде обрабатываемых данных, например: числа с плавающей точкой, комплексные, целые, булевы значения, строки или общие объекты Python. Если необходим более точный контроль над тем, как данные хранятся в памяти или на диске, особенно когда речь идет о больших наборах данных, то знать о возможности такого контроля полезно.

Таблица 4.2. Типы данных NumPy

|  |  |  |
| --- | --- | --- |
| Функция | Код типа | Описание |
| int8, uint8 | i1, u1 | Знаковое и беззнаковое В-разрядное (1 байт) целое |
| int1б, uint1б | i2,u2 | Знаковое и беззнаковое 16-разрядное (2 байта) целое |
| int32, uint32 | i4,u4 | Знаковое и беззнаковое 32-разрядное (4 байта) целое |
| int64, uint64 | i8,u8 | Знаковое и беззнаковое 64-разрядное (8 байт) целое |
| float1б | f2 | С плавающей точкой половинной точности |
| float32 | f4 | Стандартный тип с плавающей точкой одинарной точности. Совместим с типом С float |
| float64 | f8 или d | Стандартный тип с плавающей точкой двойной точности. Совместим с типом С double и с типом Python float |
| float128 | f16 | С плавающей точкой расширенной точности |
| complex64,  complex128,  complex256 | с8, c16, с32 | Комплексные числа, вещественная и мнимая части которых представлены соответственно типами flоаt32, flоаt64 и float12 8 |
| bool | ? | Булев тип, способный хранить значения True и False |
| object | O | Тип объекта Python |
| string\_ | S | Тип строки фиксированной длины (1 байт на символ). Например, строка длиной 10 имеет тип S10. |
| unicode | U | Тип Unicode-cтpoки фиксированной длины (количество байтов на символ зависит от платформы). Семантика такая же, как у типа string\_ (нaпpим ep, U10). |

Можно явно преобразовать, или привести массив одного типа к другому, воспользовавшись методом astype:

In [31]: arr = np.array([l, 2, 3, 4, 5))

In [32]: arr.dtype

Out [32]: dtype('int64')

In [33]: float\_arr = arr.astype(np.float64)

In [34]: float\_arr.dtype

Out [34]: dtype('float64')

Здесь целые были приведены к типу с плавающей точкой. Если бы я попытался привести числа с плавающей точкой к целому типу, то дробная часть была бы отброшена:

In [35]: arr np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])

In [36]: arr

Out [36]: array([ 3.7, -1.2, -2.6, 0.5, 12.9, 10.1])

In [37]: arr.astype(np.int32)

Out [37]: array ([ 3, -1, -2, О, 12, 10], dtype=int32)

Если имеется массив строк, представляющих целые числа, то astype позволит преобразовать их в числовую форму:

In [38]: numeric\_strings = np.array(['l.25', '-9.6', '42'], dtype=np.string\_)

In [39]: numeric\_strings.astype(float)

Out [39]: array ([1.25, -9.6 , 42. ])

Если по какой-то причине вьшолнить приведение не удастся (например, если строку нельзя преобразовать в тип float64), то будет возбуждено исключение TypeError. Обратите внимание, что в примере выше я поленился и написал float nместо np. float64, но NumPy оказался достаточно «умным» - он умеет подменять типы Pytlюn эквивалентными dtype.

Можно также использовать атрибут dtype другого массива:

In [40]: int\_array = np.arange(10)

In [41]: calibers = np.array((.22, .270, .357, .380, .44, .50], dtype=np.float64)

In [42]: int\_array.astype(calibers.dtype)

Out[42]: array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])

На dtype можно сослаться также с помощью коротких кодов типа:

In [43]: empty\_uint32 = np.empty(8, dtype='u4')

In [44]: empty\_uint32

Out [44]:

array ([ 0,0, 65904672,0, 64856792,0, 39438163,0], dtype=uint32)

При вызове astype всегда создается новый массив (данные копируются), даже если новый dtype не отличается от старого.

Следует иметь в виду, что числа с плавающей точкой, например типа float64 или float32, предоставляют дробные величины приближенно. В сложных вычислениях могут накапливаться ошибки округления, из-за которых сравнение возможно только с точностью до определенного числа десятичных знаков.

Операции между массивами и скалярами

Массивы важны, потому что позволяют выразить операции над совокупностями данных без выписывания циклов for. Обычно это называется векторизацией. Любая арифметическая операция над массивами одинакового размера применяется к соответственным элементам:

In [45]: arr = np.array([[l., 2., 3.], [4., 5., 6.]])

In [46]: arr

Out [46]:

array ([[1.' 2.' 3.]'

[4.' 5.' 6.]])

In [47]: arr \* arr In [48]: arr – arr

Out [47]: Out [48]:

array ([ [ 1., 4., 9.], array ([ [ 0., 0., 0.],

[16., 25., 36.]]) [0., 0., 0.]])

Как легко догадаться, арифметические операции, в которых участвует скаляр, применяются к каждому элементу массива:

In [49]: 1 / arr In [50]: arr \*\* 0.5

Out [49]: Out [49]:

array ([[1., 0.5, 0.3333], array ([[1., 1.4142, 1.7321],

[0.25, 0.2,0.1667]]) [2., 2.2361,2.4495]])

Операции между массивами разного размера называются укладыванием, мы будем подробно рассматривать их в главе 12. Глубокое понимание укладывания необязательно для чтения большей части этой книги.

**Индексирование и вырезание**

Индексирование массивов NumPy - обширная тема, поскольку подмножество массива или его отдельные элементы можно выбрать различными способами. С одномерными массивами все просто; на поверхностный взгляд, они ведут себя, как списки Python:

In (51]: arr = np.arange(10)

In [52]: arr

Out [52]: array([O, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [53]: arr [5]

Out [53]: 5

In [54]: arr[5:8]

Out [54]: array([5, 6, 7])

In [55]: arr[5:8] = 12

In [56]: arr

Out [ 56] : array ( [ О, 1, 2, 3, 4, 12, 12, 12, 8, 9])

Как видите, если присвоить скалярное значение срезу, как в arr [5: 8 J = 12, то оно распространяется (или укладывается) на весь срез. Важнейшее отличие от списков состоит в том, что срез массива является представлением исходного массива. Это означает, что данные на самом деле не копируются, а любые изменения, внесенные в представление, попадают и в исходный массив.

In [57]: arr\_slice = arr[5:8]

In [57]: arr\_slice = arr[5:8]

In [59]: arr

Out [59]: array([ О, 1, 2, 3, 4, 12, 12345, 12, 8, 9])

In [60]: arr\_slice(:] = 64

In [61]: arr

Out [ 61] : array ( [ О, 1, 2, 3, 4, 64, 64, 64, 8, 9])

При первом знакомстве с NumPy это может стать неожиданностью, особенно если вы привыкли к программированию массивов в других языках, где копирование данных применяется чаще. Но NumPy проектировался для работы с большими массивами данных, поэтому при безудержном копировании данных неизбежно возникли бы проблемы с быстродействием и памятью.

Чтобы получить копию, а не представлен"1е среза массива, нужно выполнить операцию копирования явно, например:

arr [5: 8]. сору ().

Для массивов большей размерности и вариантов тоже больше. В случае двумерного массива результатом индексирования с одним индексом является не скаляр, а одномерный массив:

In [62]: arr2d = np.array([[l, 2, 3], [4, 5, 6], [7, 8, 9]])

In [63]: arr2d [2]

Out [63]: array ([7, 8, 9])

К отдельным элементам можно обращаться рекурсивно. Но это слишком громоздко, поэтому для выбора одного элемента можно указать список индексов через запятую. Таким образом, следующие две конструкции эквивалентны:

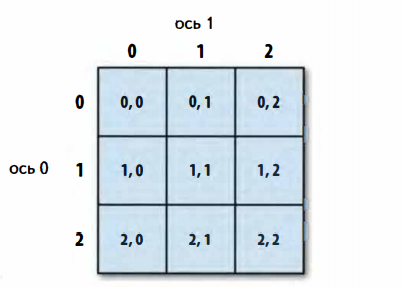
In [64]: arr2d [0][2]

Out [64]: 3

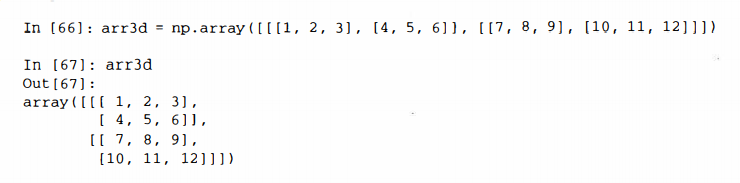
In [65]: arr2d [0, 2]

Out [65]: 3

Рисунок 4.1 иллюстрирует индексирование двумерного массива.



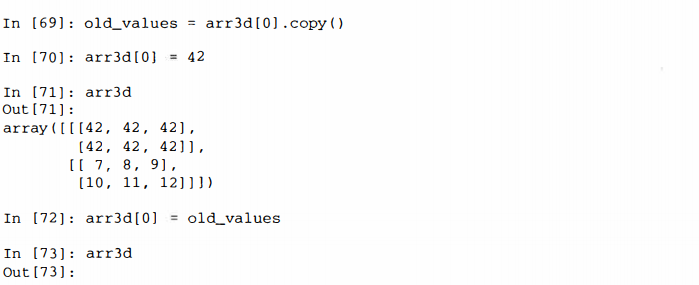
**Рис. 4.1.** Индексирование элементов в массиве NumPy

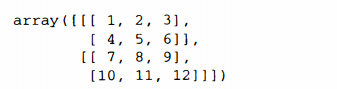


Тогда arr3d [0] - массив размерности 2 х 3:



Выражению arr3d [0] можно присвоить как скалярное значение, так и массив:





Аналогично arrЗd [1, 0] дает все значения, список индексов которых начинается с (1 , о ) , т. е. одномерный массив:



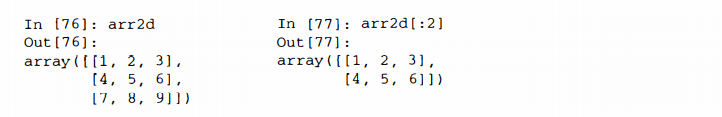
Отметим, что во всех случаях, когда выбираются участки массива, результат является представлением.

**Индексирование срезами**

Как и для одномерных объектов наподобие списков Python, для объектов ndarray можно формировать срезы:



Для объектов большей размерности вариантов больше, потому что вырезать можно по одной или нескольким осям, сочетая с выбором отдельных элементов с помощью целых индексов. Вернемся к рассмотренному выше двумерному массиву arr2d. Вырезание из него выглядит несколько иначе:



Как видите, вырезание производится вдоль оси О, первой оси. Поэтому срез содержит диапазон элементов вдоль этой оси. Можно указать несколько срезов - как несколько индексов:



При таком вырезании мы всегда получаем представления массивов с таким же числом измерений, как у исходного. Сочетая срезы и целочисленные индексы, можно получить массивы меньшей размерности:

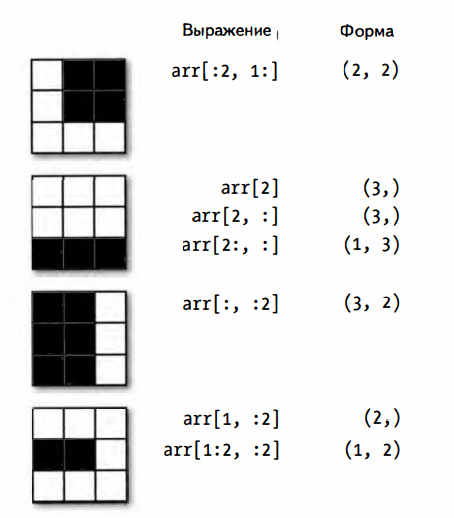


Иллюстрация приведена на рис. 4.2. Отметим, что двоеточие без указания числа означает, что нужно взять всю ось целиком, поэтому для получения осей только высших размерностей можно поступить следующим образом:



Разумеется, присваивание выражению-срезу означает присваивание всем элементам этого среза:

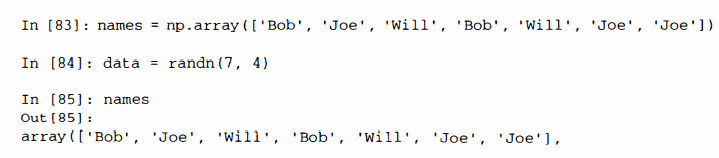
In [82]: arr2d [: 2, 1:] = 0

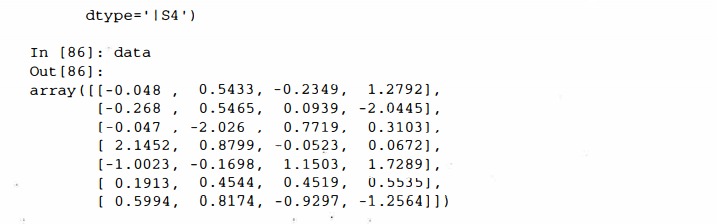


**Рис. 4.2.** Вырезание из двумерного массива

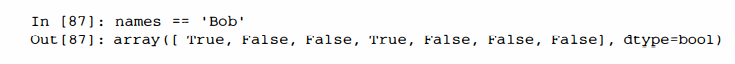
**Булево индексирование**

Пусть имеется некоторый массив с данными и массив имен, содержащий дубликаты. Я хочу воспользоваться функцией randn из модуля numpy. random, чтобы сгенерировать случайные данные с нормальным распределение

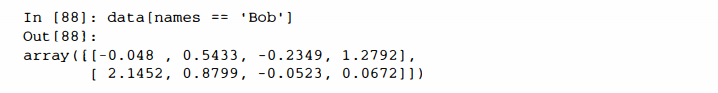




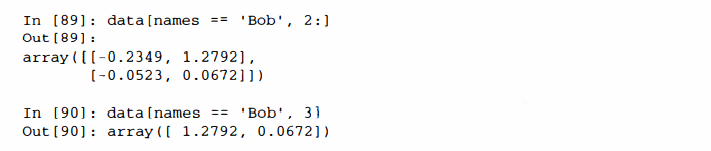
Допустим, что каждое имя соответствует строке в массиве da ta, и мы хотим выбрать все строки, которым соответствует имя “Bob”. Операции сравнения массивов (например, ==), как и арифметические, также векторизованы. Поэтому сравнение names со строкой “Bob” дает массив булевых величин:



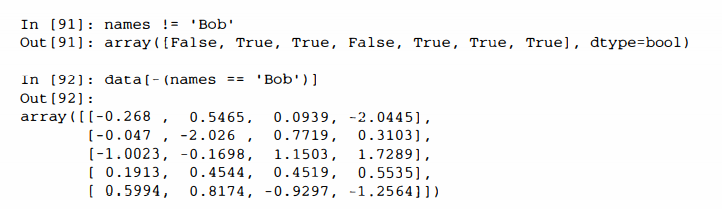
Этот булев массив можно использовать для индексирования другого массива:



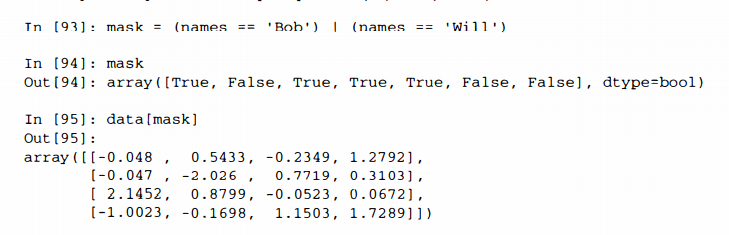
Длина булева массива должна совпадать с длиной индексируемой им оси. Можно даже сочетать булевы массивы со срезами и целыми числами (или последовательностями целых чисел, о чем речь пойдет ниже):



Чтобы выбрать все, кроме «ВоЬ», можно либо воспользоваться оператором сравнения !=, либо применить отрицание условие, обозначаемое знаком-:



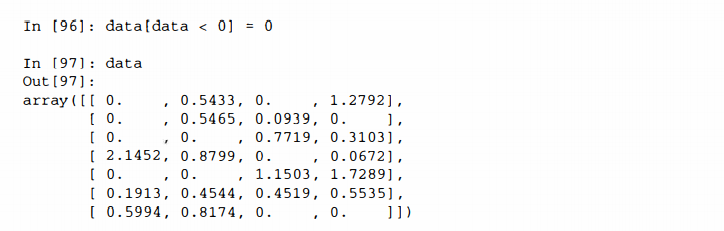
Чтобы сформировать составное булево условие, включающее два из трех имен, воспользуемся булевыми операторами & (И) и 1 (ИЛИ):



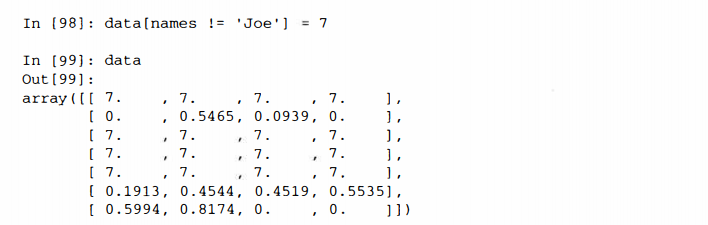
При выборке данных из массива путем булева индексирования всегда создается копия данных, даже если возвращенный массив совпадает с исходным.

Ключевые слова Python and и or с булевыми массивами не работают.

Задание значений с помощью булевых массивов работает в соответствии с ожиданиями. Чтобы заменить все отрицательные значения в массиве data нулем, нужно всего лишь написать:

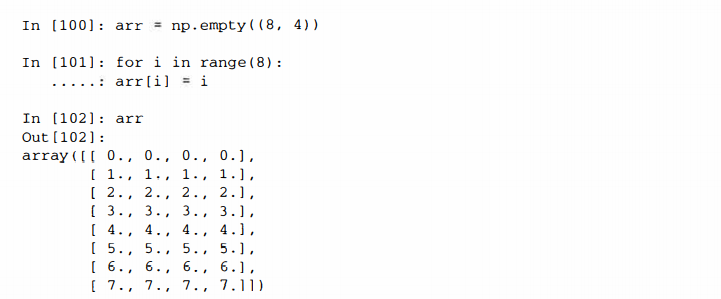


Задать целые строки или столбцы с помощью одномерного булева массива тоже просто:



**Прихотливое индексирование**

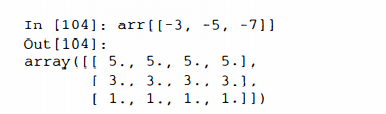
Термином прихотливое индексирование (fancy indexing) в NumPy обозначается индексирование с помощью целочисленных массивов. Допустим, имеется массив 8 х 4:



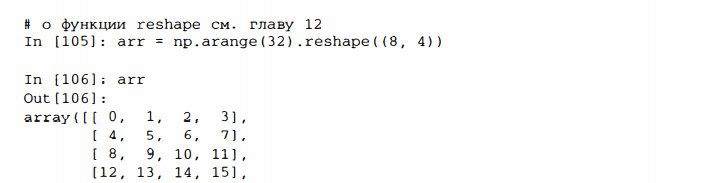
Чтобы выбрать подмножество строк в определенном порядке, можно просто передать список или массив целых чисел, описывающих желаемый порядок:

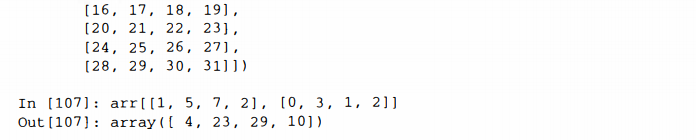


Надеюсь, что этот код делает именно то, что вы ожидаете! Если указать отрицательный индекс, то номер соответствующей строки будет отсчитываться с конца:

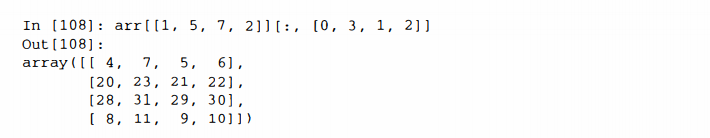


При передаче нескольких массивов индексов делается несколько иное: выбирается одномерный массив элементов, соответствующих каждому кортежу индексов:

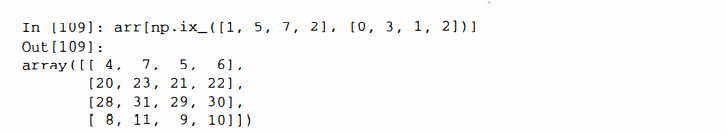




Давайте разберемся, что здесь происходит: отбираются элементы в позициях (1, 0), (5, 3), (7, 1) и (2, 2). В данном случае поведение прихотливого индексирования отличается от того, что ожидают многие пользователи (я в том числе): получить прямоугольный регион, образованный подмножеством строк и столбцов матрицы. Добиться этого можно, например, так:



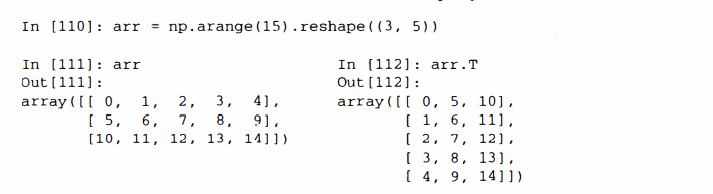
Другой способ - воспользоваться функцией np.ix\_, которая преобразует два одномерных массива целых чисел в индексатор, выбирающий квадратный регион:



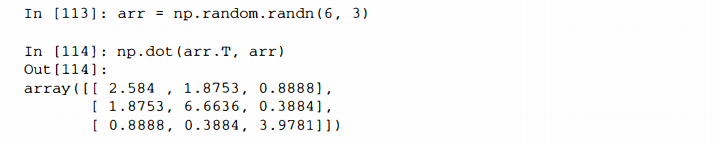
Имейте в виду, что прихотливое индексирование, в отличие от вырезания, всегда порождает новый массив, в который копируются данные.

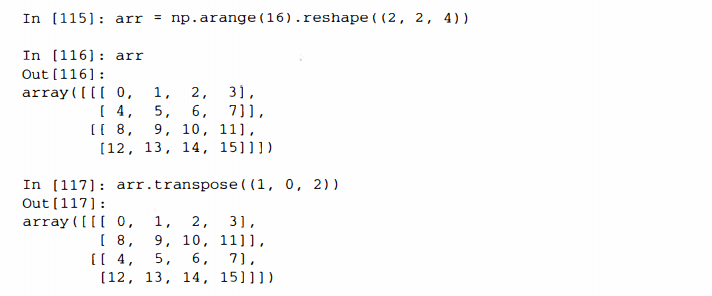
**Транспонирование массивов и перестановка осей**

Транспонирование - частный случай изменения формы, при этом также возвращается представление исходных данных без какого-либо копирования. У массивов имеется метод *transpose* и специальный атрибут T:

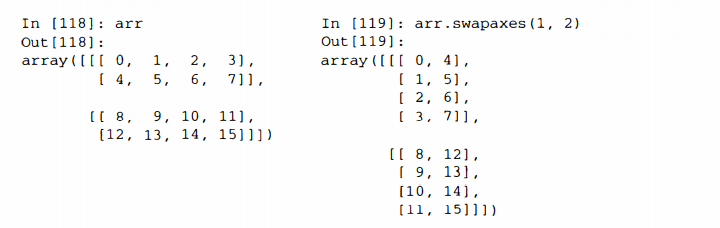


При вычислениях с матрицами эта операция применяется очень часто. Вот, например, как вычисляется матрица xTx с помощью метода np.dot:

Для массивов большей размерности метод transpose принимает кортеж номеров осей, описывающий их перестановку (чтобы ум за разум совсем заехал):



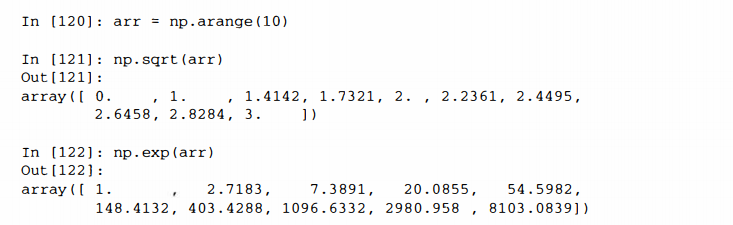
Обычное транспонирование с помощью. т - частный случай перестановки осей. У объекта ndarray имеется метод swapaxes, который принимает пару номеров осей:



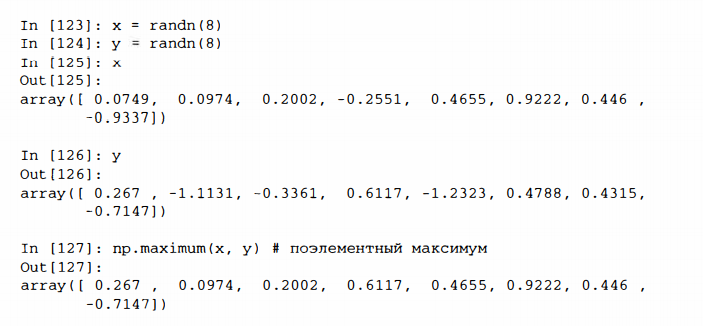
Метод swapaxes также возвращает представление без копирования данных.

**Универсальные функции: быстрые поэлементные операции над массивами**

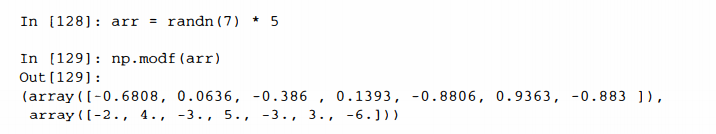
Универсальной функцией, или u-функцией называется функция, которая выполняет поэлементные операции над данными, хранящимися в объектах ndaпay. Можно считать, что это векторные обертки вокруг простых функций, которые принимают одно или несколько скалярных значений и порождают один или несколько скалярных результатов. Многие u-функции - простые поэлементные преобразования, например sqrt или ехр:



Такие u-функции называются унарными. Другие, например add или maximum, принимают 2 массива (и потому называются бинарными) и возвращают один результирующий массив:



Хотя и нечасто, но можно встретить н-функцию, возвращающую несколько массивов. Примером может служить modf, векторный вариант встроенной в Python функции divmod: она возвращает дробные и целые части хранящихся в массиве чисел с плавающей точкой:



В таблицах 4.3 и 4.4 перечислены имеющиеся и-функции.

**Таблица 4.3**. Унарные u-функции

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| abs, fabs | Вычислить абсолютное значение целых, вещественных или комплекснь1х элементов массива. Для вещественных данных f abs работает быстрее |
| sqrt | Вычислить квадратный корень из каждого элемента. Эквивалентно arr \* \* О. 5 |
| square | Вычислить квадрат каждого элемента. Эквивалентно arr \*\* 2 |
| ехр | Вычислить экспоненту каждого элемента |
| log, loglO, log2, log1p | Натуральный (по основанию е), десятичный, двоичный логарифм и функция log (1 + х) соответственно |
| sign | Вычислить знак каждого элемента: 1 (для положительных чисел), 0 (для нуля) или -1 (для отрицательных чисел) |
| ceil | Вычислить для каждого элемента наименьшее целое число, не меньшее его |
| floor | Вычислить для каждого элемента наибольшее целое число, не большее его |
| rint | Округлить элементы до ближайшего целого с сохранением dtype |
| modf | Вернуть дробные и целые части массива в виде отдельных массивов |
| isnan | Вернуть булев массив, показывающий, какие значения являются NaN (не числами) |
| isfinite, isinf | Вернуть булев массив, показывающий, какие элементы являются конечными (не inf и не NaN) или бесконечными соответственно |
| cos, cosh, sin, sinh, tan, tanh | Обычные и гиперболические тригонометрические функции |
| arccos, arccosh, arcsin, arcsinh, arctan, arctanh | Обратные тригонометрические функции |
| logical\_not | Вычислить значение истинности not х для каждого элемента. Эквивалентно -arr |

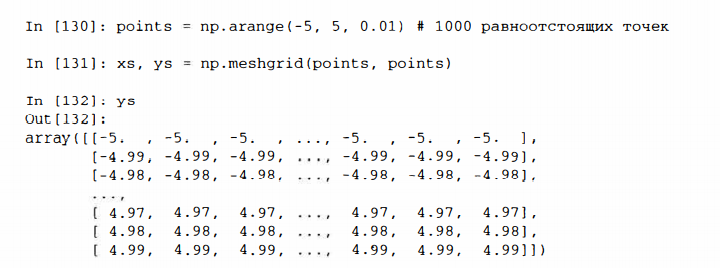
**Таблица 4.4**. Бинарные u-функции

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| add | Сложить соответственные элементы массивов |
| suЬtract | Вычесть элементы второго массива из соответственных элементов первого |
| multiply | Перемножить соответственные элементы массивов |
| divide, floor\_divide | Деление и деление с отбрасыванием остатка |
| power | Возвести элементы первого массива в степени, указанные во втором массиве |
| maximum, fmax | Поэлементный максимум. Функция fmax игнорирует значения NaN |
| maximum, fmax | Поэлементный минимум. Функция fmin игнорирует значения NaN |
| mod | Поэлементный модуль (остаток от деления) |
| copysign | Копировать знаки значений второго массива в соответственные элементы первого массива |
| greater, greater\_equal, less, less\_equal, equal, not\_equal | Поэлементное сравнение, возвращается булев массив. Эквивалентны инфиксным операторам>, > =, <,==, ! = |
| logical\_and, logical\_or, logical\_xor | Вычислить логическое значение истинности логических операций. Эквивалентны инфиксным операторам & , ! , ˆ |

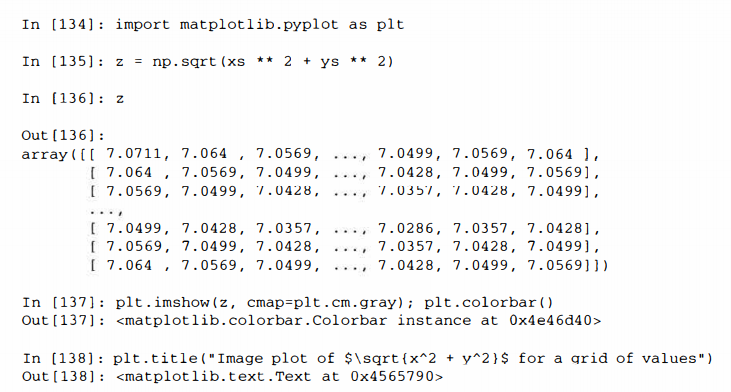
**Обработка данных с применением массивов**

С помощью массивов NumPy многие виды обработки данных можно записать очень кратко, не прибегая к циклам. Такой способ замены явных циклов выражениями-массивами обычно называется векторизацией. Вообще говоря, векторные операции с массивами выполняются на один-два (а то и больше) порядка быстрее, чем эквивалентные операции на чистом Python. Позже, в главе 12 я расскажу об укладывании, действенном методе векторизации вычислений.

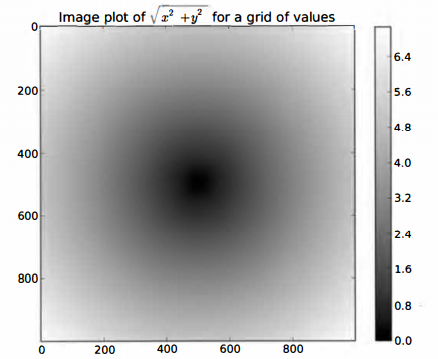
В качестве простого примера предположим, что нужно вычислить функцию sqrt (х ˆ 2 + у ˆ2) на регулярной сетке. Функция np.meshgrid принимает два одномерных массива и порождает две двумерные матрицы, соответствующие всем парам (х, у) элементов, взятых из обоих массивов:



Теперь для вычисления функции достаточно написать такое же выражение, как для двух точек:



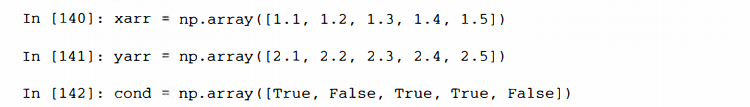
На рис. 4.3 показан результат применения функции imshow из библиотеки matplotlib для создания изображения по двумерному массиву значений функции.



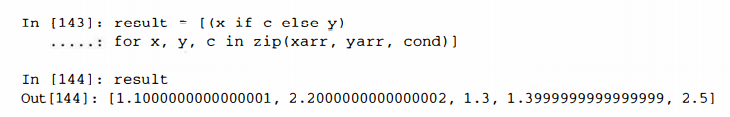
**Рис. 4.3**. График функции двух переменных на сетке

**Запись логических условий в виде операций с массивами**

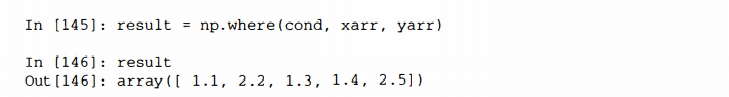
Функция numpy.where - это векторный вариант тернарного выражения х i f condition else у. Пусть есть булев массив и два массива значений:



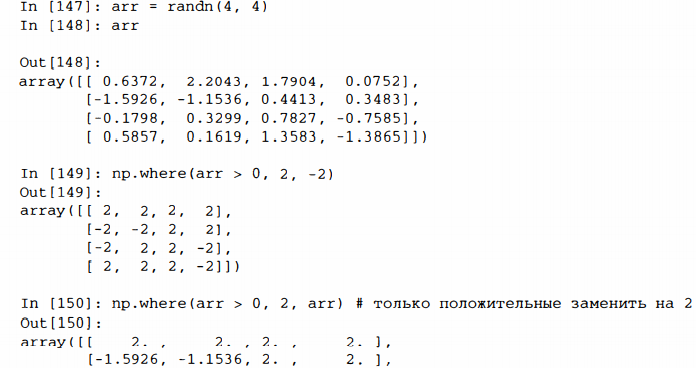
Допустим, что мы хотим брать значение из массива xarr, если соответственное значение в массиве cond равно True, а в противном случае - значение из уarr. Эту задачу решает такая операция спискового включения:



Здесь сразу несколько проблем. Во-первых, для больших массивов это будет не быстро (потому что весь код написан на чистом Python). Во-вторых, к многомерным массивам такое решение вообще неприменимо. С помощью функции np.where можно написать очень лаконичный код:

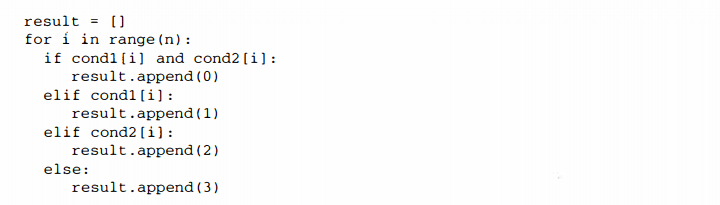


Второй и третий аргументы np.where не обязаны быть массивами - один или оба могут быть скалярами. При анализе данные where обычно применяется, чтобы создать новый массив на основе существующего. Предположим, имеется матрица со случайными данными, и мы хотим заменить все положительные значение на 2, а все отрицательные - на -2. С помощью np.where сделать это очень просто:

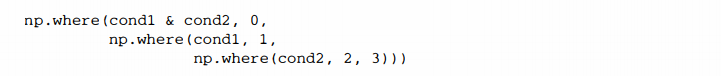




Передавать where можно не только массивы одинакового размера или скаляры. При некоторой изобретательности where позволяет выразить и более сложную логику. Пусть есть два булева массива cond1 и cond2, и мы хотим сопоставить разные значения каждой из четырех возможных комбинаций двух булевых значений:



Хотя сразу это не очевидно, показанный цикл for можно преобразовать во вложенное выражение where:

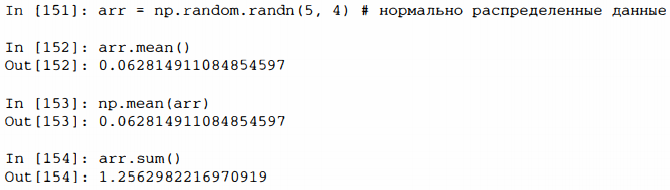


В этом конкретном примере можно также воспользоваться тем фактом, что во всех вычислениях булевы значения трактуются как О и 1, поэтому выражение можно записать и в виде такой арифметической операции (хотя выглядит она загадочно):

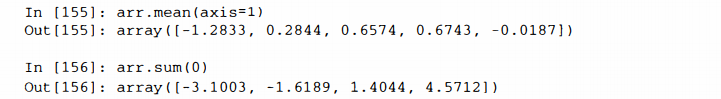


**Математические и статистические операции**

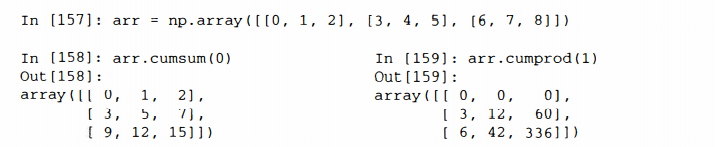
Среди методов массива есть математические функции, которые вычисляют статистики массива в целом или данных вдоль одной оси. Выполнить агрегирование (часто его называют редукцией) типа sum, mean или стандартного отклонения std можно как с помощью метода экземпляра массива, так и функции на верхнем уровне NumPy:



Функции типа mean и sum принимают необязательный аргумент axis, при наличии которого вычисляется статистика по заданной оси, и в результате порождается массив на единицу меньшей размерности:



Другие методы, например cumsum и cumpod, ничего не агрегируют, а порождают массив промежуточных результатов:



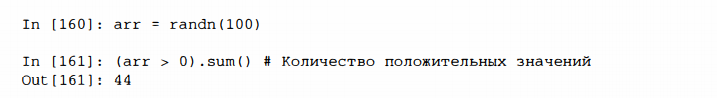
Полный список приведен в табл. 4.5. Как многие из этих методов применяются на практике, мы увидим в последующих главах.

**Таблица 4.5.** Статистические методы массива

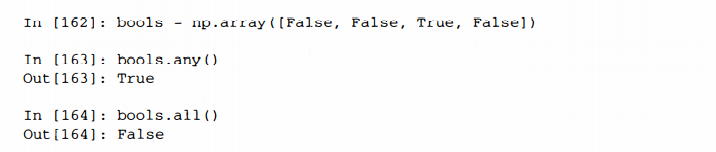
|  |  |
| --- | --- |
| **Метод** | **Описание** |
| sum | Сумма элементов всего массива или вдоль одной оси. Для массивов нулевой длины функция sum возвращает О |
| mean | Среднее арифметическое. Для массивов нулевой длины равно NaN |
| std, var | Стандартное отклонение и дисперсия, соответственно. Может быть задано число степеней свободы (по умолчанию знаменатель равен n) |
| min, max | Минимум и максимум |
| argmin, argmax | Индексы минимального и максимального элемента |
| cumsum | Нарастающая сумма с начальным значением О |
| cumprod | Нарастающее произведение с начальным значением 1 |

**Методы, булевых массивов**

В вышеупомянутых методах булевы значения приводятся к 1 (True) и О (False). Поэтому функция sum часто используется для подсчета значений True в булевом массиве:



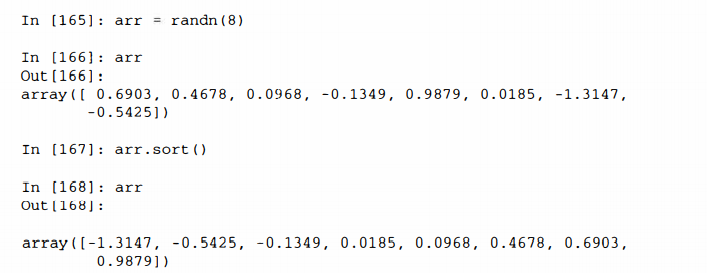
Но существуют еще два метода, any и all, особенно полезных в случае булевых массивов. Метод any проверяет, есть ли в массиве хотя бы одно значение, равное True, а all - что все значения в массиве равны True:



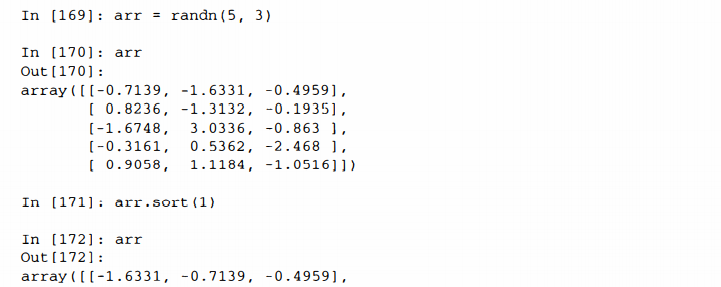
Эти методы работают и для небулевых массивов, и тогда все отличные от нуля элементы считаются равными True.

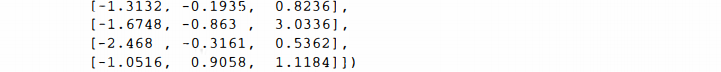
**Сортировка**

Как и встроенные в Python списки, массивы NumPy можно сортировать на месте методом sort:

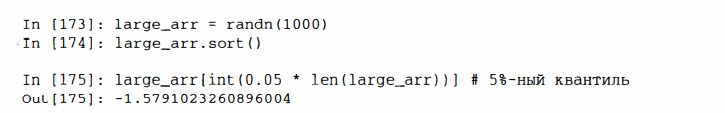


В многомерных массивах можно сортировать на месте одномерные секции вдоль любой оси, для этого нужно передать sort номер оси:





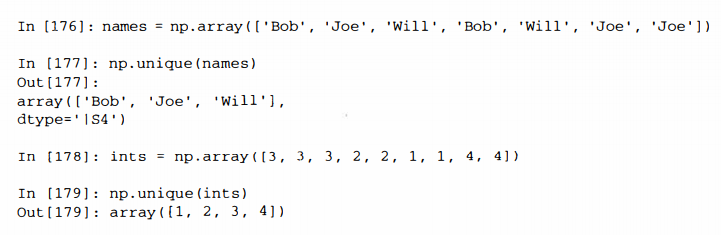
Метод верхнего уровня np.sort возвращает отсортированную копию массива, а не сортирует массив на месте. Чтобы, не мудрствуя лукаво, вычислить квантили массива, нужно отсортировать его и выбрать значение с конкретным рангом:



Дополнительные сведения о методах сортировки в NuшPy и о более сложных приемах, например косвенной сортировке, см. главу 12. В библиотеке pandas есть еще несколько операций, относящихся к сортировке (например, сортировка таблицы по одному или нескольким столбцам).

**Устранение дубликатов и другие теоретико-множественные операции**

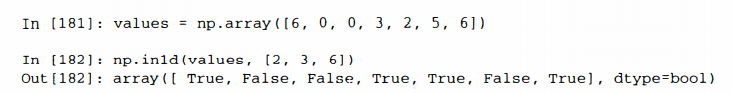
В NumPy имеется основные теоретико-множественные операции для одномерных массивов. Пожалуй, самой употребительной является nр.unique, она возвращает отсортированное множество уникальных значений в массиве:



Сравните nр.unique с альтернативой на чистом Python:



Функция np.in1d проверяет, присутствуют ли значения из одного массива в другом, и возвращает булев массив:



В табл. 4.6 перечислены все теоретико-множественные функции, имеющиеся в NumPy.

**Таблица 4.6.** Теоретико-множественные операции с массивами

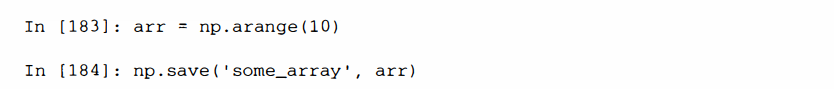
|  |  |
| --- | --- |
| **Метод** | **Описание** |
| unique(x) | Вычисляет отсортированное множество уникальных элементов |
| intersectld(x, у) | Вычисляет отсортированное множество элементов, общих для х и у |
| unionld(x, у) | Вычисляет отсортированное объединение элементов |
| inld(x, у) | Вычисляет булев массив, показывающий, какие элементы х встречаются в у |
| setdiffld(x, у) | Вычисляет разность множеств, т. е. элементы, принадлежащие х, но не принадлежащие у |
| setxorld(x, у) | Симметрическая разность множеств; элементы, принадлежащие одному массиву, но не обоим сразу |

**Файловый ввод-вывод массивов**

NumPy умеет сохранять на диске и загружать с диска данные в текстовом или двоичном формате. В последующих главах мы узнаем, как в pandas считываются в память табличные данные.

**Хранение массивов на диске в двоичном формат**

np.save и np.load - основные функции для эффективного сохранения и загрузки данных с диска. По умолчанию массивы хранятся в несжатом двоичном формате в файле с расширением. npy.



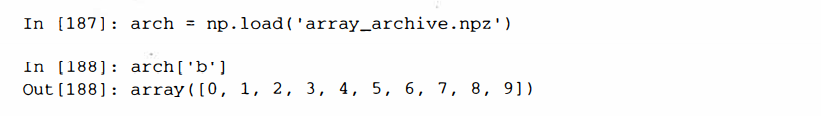
Если путь к файлу не заканчивается суффиксом. npy, то он будет добавлен. Хранящийся на диске массив можно загрузить в память функцией np. load:



Можно сохранить несколько массивов в ziр-архиве с помощью функции np. savez, которой массивы передаются в виде именованных аргументов:



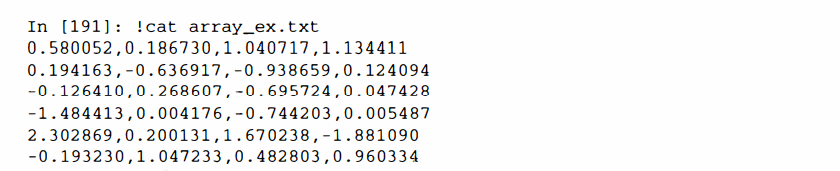
При считывании nрz-файла мы получаем похожий на словарь объект, который отложено загружает отдельные массивы:



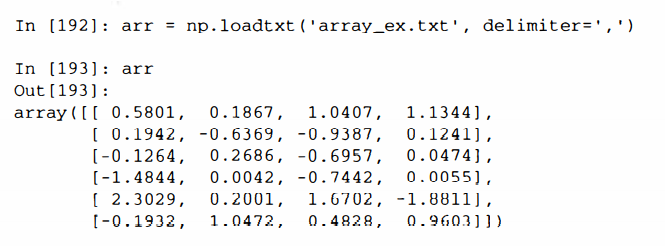
**Сохранение и загрузка текстовых файлов**

Загрузка текста из файлов - вnолне стандартная задача. Многообразие имеющихся в Python функций для чтения и записи файлов может смутить начинающего, поэтому я буду рассматривать главным образом функции read\_csv и read\_taЫe из библиотеки pandas. Иногда бывает полезно загружать данные в массивы NumPy с помощью функции np. loadtxt или более специализированной np.genfromtxt.

У этих функций много параметров, которые позволяют задавать разные разделители и функции-конвертеры для столбцов, пропускать строки и делать много других вещей. Рассмотрим простой случай загрузки файла с разделителями-запятыми (CSV):



Его можно следующим образом загрузить в двумерный массив:

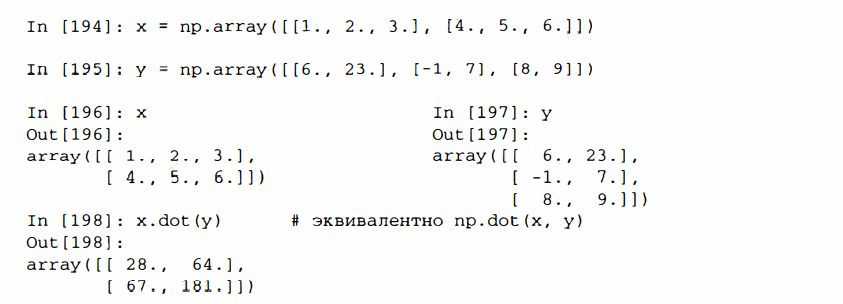


Функция np.savetxt выполняет обратную операцию: записывает массив в текстовый файл с разделителями. Функция genfromtxt аналогична loadtxt, но ориентирована на структурные массивы и обработку отсутствующих данных. Подробнее о структурных массивах см. главу 12.

Дополнительные сведения о чтении и записи файлов, особенно табличных, приведены в последующих главах, касающихся библиотеки pandas и объектов DataFrame.

**Линейная алгебра**

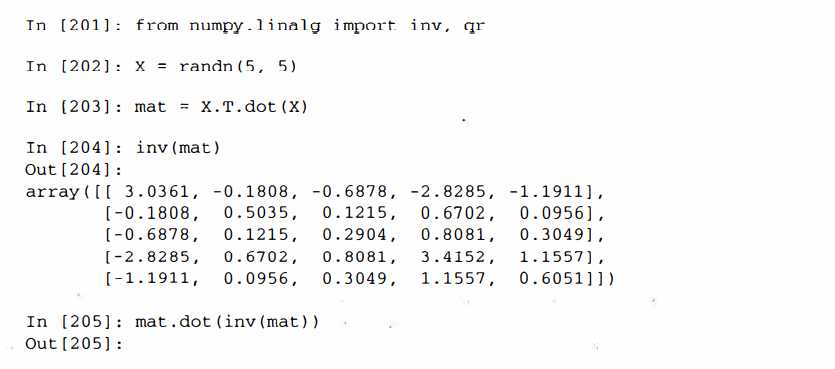
Операции линейной алгебры - умножение и разложение матриц, вычисление определителей и другие - важная часть любой библиотеки для работы с массивами. В отличие от некоторых языков, например МАТLАБ, в NumPy применение оператора \* к двум двумерным массивам вычисляет поэлементное, а не матричное произведение. А для перемножения матриц имеется функция dot - как в виде метода массива, так и в виде функции в пространстве имен numpy:

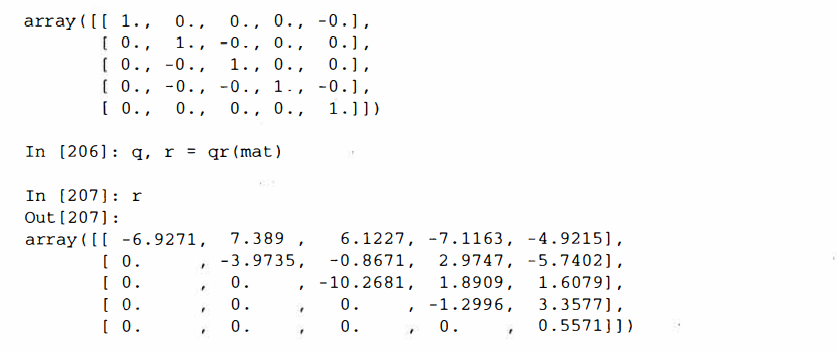


Произведение двумерного массива и одномерного массива подходящего размера дает одномерный массив:



В модуле numpy.linalg имеет стандартный набор алгоритмов, в частности, разложение матриц, нахождение обратной матрицы и вычисление определителя. Все они реализованы на базе тех же отраслевых библиотек, написанных на Fortran, которые: используются и в других языках, например МАТLАБ и R: BLAS, LAPACK и, возможно (в зависимости от сборки NumPy), библиотеки MKL, поставляемой компанией Intel:





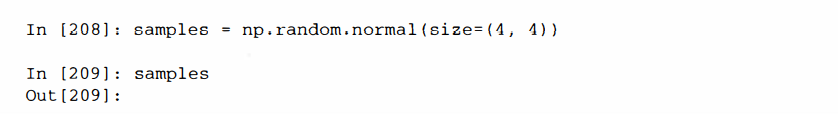
В табл. 4.7 перечислены наиболее употребительные функции линейной алгебры.

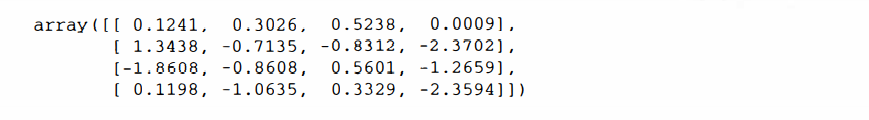
**Таблица 4.7.** Наиболее употребительные функции из модуля numpy.linalg

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| diag | Возвращает диагональные элементы квадратной матрицы в виде одномерного массива или преобразует одномерный массив в квадратную матрицу, в которой все элементы, кроме находящихся на главной диагонали, равны нулю |
| dot | Возвращает диагональные элементы квадратной матрицы в виде одномерного массива или преобразует одномерный массив в квадратную матрицу, в которой все элементы, кроме находящихся на главной диагонали, равны нулю |
| trace | Вычисляет след матрицы - сумму диагональных элементов |
| det | Вычисляет определитель матрицы |
| eig | Вычисляет собственные значения и собственные векторы квадратной матрицы |
| inv | Вычисляет обратную матрицу |
| pinv | Вычисляет псевдообратную матрицу Мура-Пенроуза для квадратной матрицы |
| qr | Вычисляет QR-разложение |
| svd | Вычисляет сингулярное разложение (SVD) |
| solve | Решает линейную систему Ах = Ь, где А - квадратная матрица |
| lstsq | Вычисляет решение уравнения у = хь по методу наименьших квадратов |

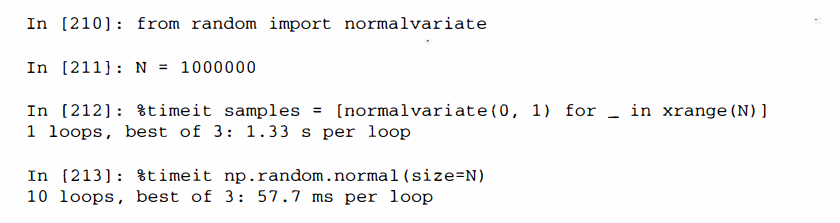
**Генерация случайных чисел**

Модуль numpy.random дополняет встроенный модуль random функциями, которые генерируют целые массивы случайных чисел с различными распределениями вероятности. Например, с помощью функции можно получить случайный массив 4 х 4 с нормальным распределением:





Встроенный в Python модуль random умеет выдавать только по одному случайному числу за одно обращение. Ниже видно, что numpy.random более чем на порядок быстрее стандартного модуля при генерации очень больших выборок:



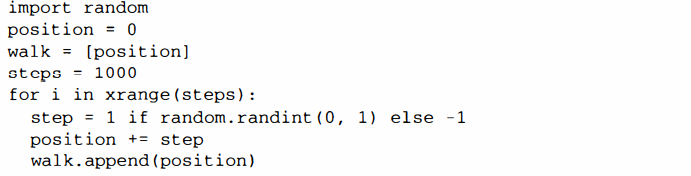
В табл. 4.8 приведен 11епол1fый перечень функций, имеющихся в модуле numpy. random. В следующем разделе я приведу несколько примеров их использования для генерации больших случайных массивов.

**Таблица 4.8.** Наиболее употребительные функции из модуля numpy. Random

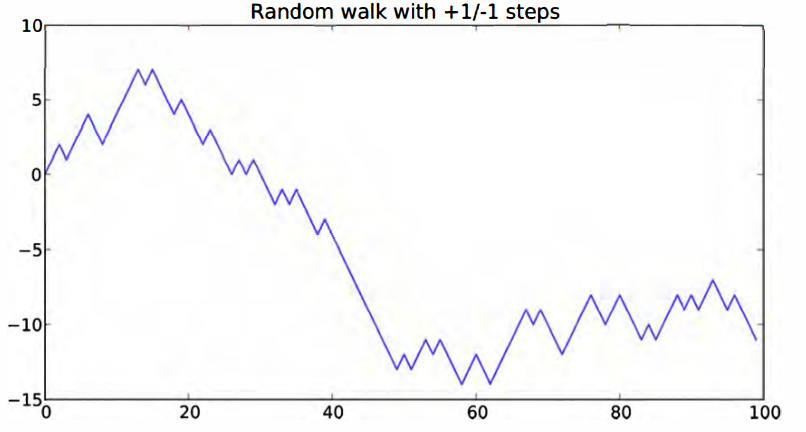
|  |  |
| --- | --- |
| **Функция** | **Описание** |
| seed | Задает начальное значение генератора случайных чисел |
| permutation | Возвращает случайную перестановку последовательности или диапазона |
| shufflе | Случайным образом переставляет последовательность на месте |
| rand | Случайная выборка с равномерным распределением |
| randint | Случайная выборка с равномерным распределением |
| randn | Случайная выборка с нормальным распределением со средним О и стандартным отклонением 1 (интерфейс похож на MATLAB) |
| Ьinomial | Случайная выборка с биномиальным распределением |
| normal | Случайная выборка с нормальным (гауссовым) распределением |
| beta | Случайная выборка с бета-распределением |
| chisquare | Случайная выборка с распределением хи-квадрат |
| gamma | Случайная выборка с гамма-распределением |
| uni form | Случайная выборка с равномерным распределением на полуинтервале [О, 1) |

**Пример: случайное блуждание**

Проиллюстрируем операции с массивами на примере случайного блуждания. Сначала рассмотрим случайное блуждание с начальной точкой О и шагами 1 и -1, выбираемыми с одинаковой вероятностью. Вот реализация одного случайного блуждания с 1 ООО шагами на чистом Python с помощью встроенного модуля random:

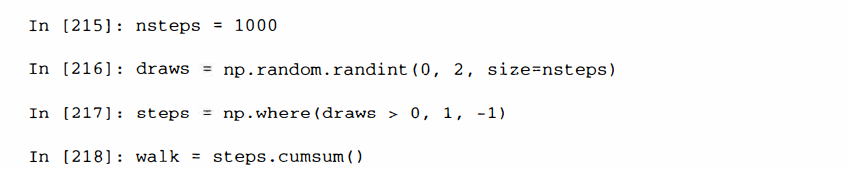


На рис. 4.4 показаны первые 100 значений такого случайного блуждания.

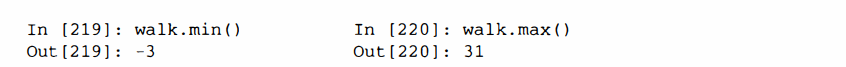


**Рис. 4.4.** Простое случайное блуждание

Наверное, вы обратили внимание, что walk — это просто нарастающая сумма случайных шагов, которую можно вычислить как выражение-массив. Поэтому я воспользуюсь модулем np.random, чтобы за один присест подбросить 1000 монет с исходами 1 и -1 и вычислить нарастающую сумму:

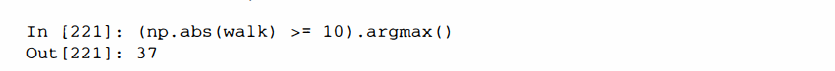


Теперь можно приступить к вычислению статистики, например минимального и максимального значения на траектории блуждания:



Более сложная статистика -. момент первого пересечения — это шаг, на котором траектория случайного блуждания впервые достигает заданного значения.

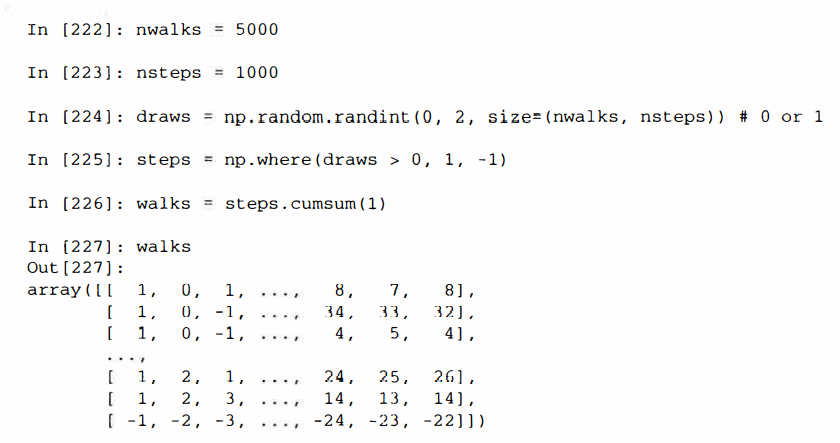
В данном случае мы хотим знать, сколько времени потребуется на то, чтобы удалиться от начала (нуля) на десять единиц в любом направлении. Выражение np.abs (walk) >= 10 дает булев массив, показывающий, в какие моменты блуждание достигало или превышало 10, однако нас интересует индекс первого значения 10 или -10. Его можно вычислить с помощью функции argmax, которая возвращает индекс первого максимального значения в булевом массиве (тrue максимальное значение):



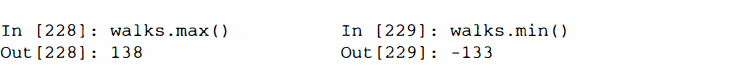
Отметим, что использование здесь argmax не всегда эффективно, потому что она всегда просматривает весь массив. В данном частном случае мы знаем, что первое же встретившееся значение True является максимальным.

**Моделирование сразу нескольких случайных блужданий**

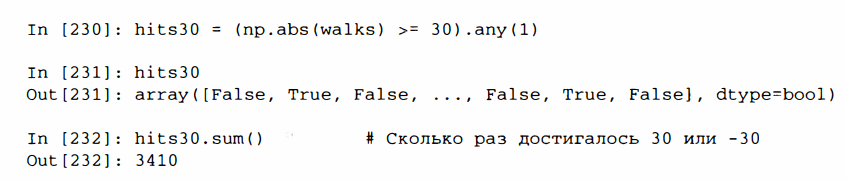
Если бы нам требовалось смоделировать много случайных блужданий, скажем 5000, то это можно было бы сделать путем совсем небольшой модификации приведенного выше кода. Если функциям из модуля numpy. random передать 2-кортеж, то они сгенерируют двумерный массив случайных чисел, и мы сможем вычислить нарастающие суммы по строкам, т. е. все 5000 случайных блужданий за одну операцию:



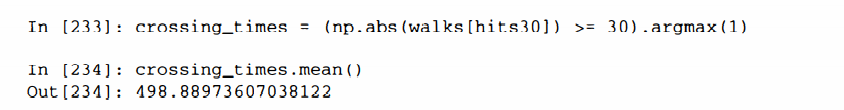
Теперь мы можем вычислить максимум и минимум по всем блужданиям:



Вычислим для этих блужданий минимальный момент первого пересечения с уровнем 30 или -30. Это не так просто, потому что не в каждом блуждании уровень 30 достигается. Проверить, так ли это, можно с помощью метода any:



Имея этот булев массив, мы можем выбрать те строки walks, в которых достигается уровень 30 (по абсолютной величине), и вызвать argmax вдоль оси 1 для получения моментов пересечения:



Поэкспериментируйте с другими распределениями шагов, не ограничиваясь подбрасыванием правильной монеты. Всего-то и нужно, что взять другую функцию генерации случайных чисел, например, normal для генерации шагов с нормальным распределением с заданными средним и стандартным отклонением:

