無料電子ブック

学習

numpy

Free unaffiliated eBook created from
Stack Overflow contributors.

#numpy

1: numpyをいめる

NumPy「パイ」または「」とされるは、Pythonプログラミングのであり、なと、これらのでするののなライブラリをサポートします。

バージョン

バージョン	
1.3.0	2009-03-20
1.4.0	2010-07-21
1.5.0	2010-11-18
1.6.0	2011-05-15
1.6.1	2011-07-24
1.6.2	2012-05-20
1.7.0	2013-02-12
1.7.1	2013-04-07
1.7.2	2013-12-31
1.8.0	2013-11-10
1.8.1	2014-03-26
1.8.2	2014-08-09
1.9.0	2014-09-07
1.9.1	2014-11-02
1.9.2	2015-03-01
1.10.0	2015-10-07
1.10.1	2015-10-12
1.10.2	2015-12-14
1.10.4 *	2016-01-07
1.11.0	2016529

Examples

Macでのインストール

MacでNumPyをするもなは、 pip

```
pip install numpy
```

Condaをしたインストール 。
Windows、Mac、LinuxでなConda

1. Condaをインストールします。 Condaをインストールするには、Anacondaフルパッケージ、numpyをむまたはMinicondaConda、Python、およびそれらがするパッケージのみ、のパッケージなしをインストールする2つのがあります。 AnacondaとMinicondaのがじCondaをインストールします。
2. Minicondaのコマンドにコマンドをします`conda install numpy`

Windowsへのインストール

pypi pipでされるデフォルトのパッケージインデックスによるNumpyインストールは、にWindowsコンピュータではします。 Windowsにインストールするもなは、プリコンパイルされたバイナリをすることです。

くのパッケージのプリコンパイルされたホイールの1つのソースは、 Christopher Gohkleのサイトです。 Pythonのバージョンとシステムにってバージョンをしてください。 64ビットシステムのPython 3.5の

1. `numpy-1.11.1+mkl-cp35-cp35m-win_amd64.whl`をここから`numpy-1.11.1+mkl-cp35-cp35m-win_amd64.whl`してください
2. Windowsをくcmdまたはpowershell
3. `pip install C:\path_to_download\numpy-1.11.1+mkl-cp35-cp35m-win_amd64.whl`

のパッケージでさせたくないは、ほとんどのパッケージをねるWinpythonディストリビューションをして、 するためのされたをすることができます。に、 Anaconda Pythonディストリビューションには、あらかじめくののパッケージがインストールされています。

もう1つののあるソースは、 もサポートするcondaパッケージマネージャ です 。

1. condaダウンロードしてインストールしconda 。
2. Windowsターミナルをきます。
3. コマンドをしますconda install numpy

Linuxでのインストール

NumPyは、もなLinuxディストリビューションのデフォルトのリポジトリでき、はLinuxディス

トリビューションのパッケージがインストールされるのとじでインストールできます。

いくつかのLinuxディストリビューションでは、Python 2.xとPython 3.xのNumPyパッケージがなります。 UbuntuとDebianでは、APTパッケージマネージャをってシステムレベルで`numpy`をインストールします

```
sudo apt-get install python-numpy
sudo apt-get install python3-numpy
```

のディストリビューションでは、zypperSuse、yumFedoraなどのパッケージマネージャをします。

`numpy`は、Python 2のPythonのパッケージマネージャ`pip`とPython 3の`pip3`とにインストールすることもできます

```
pip install numpy  # install numpy for Python 2
pip3 install numpy  # install numpy for Python 3
```

`pip`は、もなLinuxディストリビューションのデフォルトのリポジトリでで き、Python 2とPython 3にをしてインストールできます。

```
sudo apt-get install python-pip  # pip for Python 2
sudo apt-get install python3-pip  # pip for Python 3
```

インストール、`pip`のPython 2と`pip3` Pythonパッケージをインストールするためのピップをする Pythonの3のために。しかし、ソースからnumpyをビルドするためになくのパッケージ、コンパイラ、Fortranなどをむをインストールするがあるかもしれないことにしてください。

システムレベルで`numpy`をインストールするだけでなく、 `virtualenv`などのなPythonパッケージをして`numpy`をにインストールするのもですおそらくさらにおめします。 Ubuntuでは、 `virtualenv`はをってインストールできます

```
sudo apt-get install virtualenv
```

に、Python 2またはPython 3のvirtualenvをしてにしてから、 `pip`をして`numpy`をインストールします。

```
virtualenv venv  # create virtualenv named venv for Python 2
virtualenv venv -p python3  # create virtualenv named venv for Python 3
source venv/bin/activate  # activate virtualenv named venv
pip install numpy  # use pip for Python 2 and Python 3; do not use pip3 for Python3
```

なインポート

numpyモジュールをインポートして、そのをします。

```
import numpy as np
```

ほとんどのでは、 `np`をnumpyのとしてします。コードでは "np"は "numpy"をするものとします。

```
x = np.array([1,2,3,4])
```

Rackspaceがするなジュピターノート

Jupyterノートブックは、インタラクティブなブラウザベースのです。らはもともとのpythonをするためにされ、そのようにnumpyとによくふ。のシステムににインストールすることなく、Jupyterのノートブックでnumpyをしてみるために、Rackspaceはtmpnb.orgでのなノートブックをしています 。

これは、アップセルのあらゆるのプロプライエタリなサービスではありません。 JupyterはUC BerkeleyとCal Poly San Luis Obispoによってされたオープンソースのです。 Rackspaceはプロセスのとしてこのサービスをしています。

tmpnb.orgで`numpy`をしてみる

1. tmpnb.orgにアクセス
2. `Welcome to Python.ipynb`するか、
3. >> Python 2または
4. New >> Python 3

オンラインでnumpyをいめるをむ https://riptutorial.com/ja/numpy/topic/823/numpyをいめる

2: ndarrayをサブクラスする

- `def __array_prepare__(self, out_arr: ndarray, context: Tuple[ufunc, Tuple, int] = None) -> ndarray: # called on the way into a ufunc`
- `def __array_wrap__(self, out_arr: ndarray, context: Tuple[ufunc, Tuple, int] = None) -> ndarray: # called on the way out of a ufunc`
- `__array_priority__: int # used to determine which argument to invoke the above methods on when a ufunc is called`
- `def __array_finalize__(self, obj: ndarray): # called whenever a new instance of this class comes into existence, even if this happens by routes other than __new__`

Examples

のなプロパティをトラッキングする

```
class MySubClass(np.ndarray):
    def __new__(cls, input_array, info=None):
        obj = np.asarray(input_array).view(cls)
        obj.info = info
        return obj

    def __array_finalize__(self, obj):
        # handles MySubClass(...)
        if obj is None:
            pass

        # handles my_subclass[...] or my_subclass.view(MySubClass) or ufunc output
        elif isinstance(obj, MySubClass):
            self.info = obj.info

        # handles my_arr.view(MySubClass)
        else:
            self.info = None

    def __array_prepare__(self, out_arr, context=None):
        # called before a ufunc runs
        if context is not None:
            func, args, which_return_val = context

        return super().__array_prepare__(out_arr, context)

    def __array_wrap__(self, out_arr, context=None):
        # called after a ufunc runs
        if context is not None:
            func, args, which_return_val = context

        return super().__array_wrap__(out_arr, context)
```

ための`context`タプル、 `func`ufuncである`np.add`、 `args`ある`tuple`、び`which_return_val` ufuncのがされているりして

オンラインでndarrayをサブクラスするをむ https://riptutorial.com/ja/numpy/topic/6431/ndarrayを

サブクラスする

3: np.linalgをった

バージョン1.8では、 `np.linalg`いくつかのルーチンは、の「スタック」でることができます。つまり、のがみねられている、ルーチンはをできます。たとえば、 `A`は2つのみねられた33のとしてされます。

```
np.random.seed(123)
A = np.random.rand(2,3,3)
b = np.random.rand(2,3)
x = np.linalg.solve(A, b)

print np.dot(A[0,:,:], x[0,:])
# array([ 0.53155137,  0.53182759,  0.63440096])

print b[0,:]
# array([ 0.53155137,  0.53182759,  0.63440096])
```

の`np`ドキュメントは`a : (..., M, M) array_like`ようなパラメータをしてこれをしています`a : (..., M, M) array_like`。

Examples

np.solveでシステムをく

の3つのをえてみましょう。

```
x0 + 2 * x1 + x2 = 4
       x1 + x2 = 3
x0 +          x2 = 5
```

このを`A * x = b`としてすことができます。

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

に、 `np.linalg.solve`をって`x`をきます

```
x = np.linalg.solve(A, b)
# Out: x = array([ 1.5, -0.5,  3.5])
```

`A`はとでなければなりません。すべてのはでなければなりません。 `A`は/でなければなりませんそのはゼロではありません。たとえば、 `A` 1つのがのであれば、 `linalg.solve`をびすと`linalg.solve`が`LinAlgError: Singular matrix`

```
A = np.array([[1, 2, 1],
```

```
              [2, 4, 2],   # Note that this row 2 * the first row
              [1, 0, 1]])
b = np.array([4,8,5])
```

このようなシステムは、 `np.linalg.lstsq`でできます。

np.linalg.lstsqをしてシステムにをめる

は、システムよりもられているよりもくののにするなアプローチです。

4つのをえてみましょう。

```
x0 + 2 * x1 + x2 = 4
x0 + x1 + 2 * x2 = 3
2 * x0 + x1 + x2 = 5
x0 + x1 + x2 = 4
```

これを`A * x = b`

```
A = np.array([[1, 2, 1],
              [1,1,2],
              [2,1,1],
              [1,1,1]])
b = np.array([4,3,5,4])
```

それから`np.linalg.lstsq`して`np.linalg.lstsq`

```
x, residuals, rank, s = np.linalg.lstsq(A,b)
```

`x`は、 `residuals`はであり、`A` ランク を`rank` 、 `s`は`A` をランクけ `A`。 `b`がのをつ、 `lstsq`は`b`にするシステムをします。

```
A = np.array([[1, 2, 1],
              [1,1,2],
              [2,1,1],
              [1,1,1]])
b = np.array([[4,3,5,4],[1,2,3,4]]).T # transpose to align dimensions
x, residuals, rank, s = np.linalg.lstsq(A,b)
print x # columns of x are solutions corresponding to columns of b
#[[ 2.05263158  1.63157895]
# [ 1.05263158 -0.36842105]
# [ 0.05263158  0.63157895]]
print residuals # also one for each column in b
#[ 0.84210526  5.26315789]
```

`rank`と`s`は`A`のみにし、ってとじである。

オンラインでnp.linalgをったをむ https://riptutorial.com/ja/numpy/topic/3753/np-linalgをった

4: numpy.cross

- `numpy.cross(a, b)` *a*と*b*のクロスはにおけるベクター—および*b*
- `numpy.cross(a, b, axisa=-1)` *a*と*b*のベクトルの、 *a*の stベクトルは*axisa*にってされます
- `numpy.cross(a, b, axisa=-1, axisb=-1, axisc=-1)` *a*と*b*のベクトルの、 *axisc*でされたにってされたベクトル
- `numpy.cross(a, b, axis=None)` クロスにおけるベクトルのと*B*、Bにおけるベクトル、びのにって

パラ メ—タ—

カラム	カラム
a、b	もないで`a` 、 `b`と`b`は2つの2または3のベクトルです。それらは、ベクトルのすなわち、2であってもよい。、および「B」はベクトルであり、 `a` `cross(a,b)`そのがベクトルのであるりベクトルと`a` `b`。 `b`はであり`a`はのベクトルです`cross(a,b)`は`cross(a,b)`と`b`ベクトルとのを`a`をします。 `a`と`b`は、じをしていればともになります。この、 `cross(a,b)`は`cross(a[0],b[0]), cross(a[1], b[1]), ...`します`cross(a[0],b[0]), cross(a[1], b[1]), ...`
a / b	`a`がのは、もにする、もいする、またはそれらのにされたベクトルをつことができます。 `axisa`え`cross()`ベクトルがにレイアウトされていますか。 `a`デフォルトでは、もゆっくりとするのをとります。 `axisb` `b`は`b`とじきをします。 `cross()`がになる、ベクトルはなるにできます。 `axisc`え`cross`そののベクトルをレイアウトする。デフォルトでは、`axisc`はもゆっくりするをします。
	`axisa` 、 `axisb` 、および`axisc`にじてじにするなパラメ—タ。 `axis`とのパラメ—タのいずれかがびしにする、 `axis`のはのをきします。

Examples

2つのベクトルの

Numpyは、ベクトルのクロスをするための`cross`をします。ベクトル`[1, 0, 0]`と`[0, 1, 0]` `[1, 0, 0]`は`[0, 0, 1]`です。ナンシ—はたちにこういます

```
>>> a = np.array([1, 0, 0])
>>> b = np.array([0, 1, 0])
>>> np.cross(a, b)
array([0, 0, 1])
```

り。

クロスは、、3ベクトルにしてのみされます。しかし、Numpyののいずれかは、2つのベクトルになります。ベクトルの`c`でえられる`[c1, c2]`のは、のにゼロを`[c1, c2, 0]`そう、

```
>>> c = np.array([0, 2])
>>> np.cross(a, c)
array([0, 0, 2])
```

Numpyと`ndarray`メソッドのとしてする`dot`とはなり、 `cross`はスタンドアロンとしてのみします。

```
>>> a.cross(b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'numpy.ndarray' object has no attribute 'cross'
```

1のコールでのクロス

どちらのも、3または2のベクトルのにすることができます。

```
>>> a=np.array([[1,0,0],[0,1,0],[0,0,1]])
>>> b=np.array([1,0,0])
>>> np.cross(a,b)
array([[ 0,  0,  0],
       [ 0,  0, -1],
       [ 0,  1,  0]])
```

こののは、[np.crossa [0]、b、np.crossa [1]、b、np.crossa [2]、b]です。

`b`は3-または2-ベクトルのでもいませんが、 `a`とじでなければなりません。そうでなければ、は "" エラーでする。だから々はできる

```
>>> b=np.array([[0,0,1],[1,0,0],[0,1,0]])
>>> np.cross(a,b)
array([[ 0, -1,  0],
       [ 0,  0, -1],
       [-1,  0,  0]])
```

は`array([np.cross(a[0],b[0]), np.cross(a[1],b[1]), np.cross(a[2],b[2])])`

のクロスによるの

の2つのでは、numpyは`a[0,:]`がのベクトル、 `a[1,:]`が2、 `a[2,:]`が3のベクトルで`a[1,:]`としまし`a[0,:]`。 Numpy.crossにはオプションのaxisaがあり、ベクトルをするをすることができます。そう、

```
>>> a=np.array([[1,1,1],[0,1,0],[1,0,-1]])
>>> b=np.array([0,0,1])
>>> np.cross(a,b)
array([[ 1, -1,  0],
```

```
       [ 1,  0,  0],
       [ 0, -1,  0]])
>>> np.cross(a,b,axisa=0)
array([[ 0, -1,  0],
       [ 1, -1,  0],
       [ 0, -1,  0]])
>>> np.cross(a,b,axisa=1)
array([[ 1, -1,  0],
       [ 1,  0,  0],
       [ 0, -1,  0]])
```

`axisa=1`びのはに`(np.cross([1,1,1],b), np.cross([0,1,0],b), np.cross([1,0,-1],b))`。 デフォルトでは、 `axisa`にのもゆっくりとするをします。 `axisa=0`は`(np.cross([1,0,1],b), np.cross([1,1,0],b), np.cross([1,0,-1],b))`。

のオプションのパラメータ`axisb`も、2でも`b`にしてじをします。

パラメータaxisaとaxisbは、データをどのようにするかをnumpyにします。3のパラメータaxiscは、 `a`または`b`がである、をどのようにするかをnumpyにします。とじ`a`と`b`をして、

```
>>> np.cross(a,b,1)
array([[ 1, -1,  0],
       [ 1,  0,  0],
       [ 0, -1,  0]])
>>> np.cross(a,b,1,axisc=0)
array([[ 1,  1,  0],
       [-1,  0, -1],
       [ 0,  0,  0]])
>>> np.cross(a,b,1,axisc=1)
array([[ 1, -1,  0],
       [ 1,  0,  0],
       [ 0, -1,  0]])
```

したがって、 `axisc=1`とデフォルトの`axisc`でじがられます。 `axisc` 、ベクトルのはのもいインデックスでしています。 axiscはデフォルトでののです。 `axisc=0`は、ベクトルのをのもの`axisc=0`にします。

`axisa` 、 `axisb` 、および`axisc`をすべてじにするには、3つのパラメータをすべてするはありません。 4つのパラメータ、 `axis`をなのにすることができ、の3つのパラメータはにされます。がびしにするは、axisa、axisb、またはaxiscをオーバーライドします。

オンラインでnumpy.crossをむ https://riptutorial.com/ja/numpy/topic/6166/numpy-cross

5: numpy.dot

- numpy.dota、b、out = None

パラ メ—タ—

a	のない
b	のない
でる	のない

numpy.dot

aとbのをします。 aとbがにスカラ—またはの1である、スカラ—がされます。それのはがされます。 outがえられた、それがされます。

Examples

は、ドットで2つのなでうことができます。 1つのは、numpy.ndarrayのドットメンバ—をすることです。

```
>>> import numpy as np
>>> A = np.ones((4,4))
>>> A
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
>>> B = np.ones((4,2))
>>> B
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]])
>>> A.dot(B)
array([[ 4.,  4.],
       [ 4.,  4.],
       [ 4.,  4.],
       [ 4.,  4.]])
```

のをう2のはnumpyライブラリです。

```
>>> np.dot(A,B)
array([[ 4.,  4.],
       [ 4.,  4.],
       [ 4.,  4.],
       [ 4.,  4.]])
```

ベクトルドット

ドットは、2つの1numpyののベクトルドットをするためにもできます。

```
>>> v = np.array([1,2])
>>> w = np.array([1,2])
>>> v.dot(w)
5
>>> np.dot(w,v)
5
>>> np.dot(v,w)
5
```

outパラメータ

numpyドットには、オプションのパラメータout = Noneがあります。このパラメータをすると、をきむをできます。このは、されたとまったくじとでなければならないか、がスローされます。

```
>>> I = np.eye(2)
>>> I
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> result = np.zeros((2,2))
>>> result
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> np.dot(I, I, out=result)
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> result
array([[ 1.,  0.],
       [ 0.,  1.]])
```

のdtypeをintにしてみましょう。

```
>>> np.dot(I, I, out=result)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: output array is not acceptable (must have the right type, nr dimensions, and be a
C-Array)
```

また、Fortranスタイルがではなくしているためのようななるなメモリーをしようとすると、エラーもします。

```
>>> result = np.zeros((2,2), order='F')
>>> np.dot(I, I, out=result)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: output array is not acceptable (must have the right type, nr dimensions, and be a
C-Array)
```

ベクトルのにする

numpy.dotは、ベクトルのリストにをけるためにできますが、ベクトルのきは、8つの2つのベクトルのリストが2つの8つのベクトルのようにえるように、でなければなりません。

```
>>> a
array([[ 1.,  2.],
       [ 3.,  1.]])
>>> b
array([[  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.],
       [  9.,  10.,  11.,  12.,  13.,  14.,  15.,  16.]])
>>> np.dot(a, b)
array([[ 19.,  22.,  25.,  28.,  31.,  34.,  37.,  40.],
       [ 12.,  16.,  20.,  24.,  28.,  32.,  36.,  40.]])
```

ベクトルのリストがをにしてされているくのそうです、はのようにドットのにれえるがあります

```
>>> b
array([[  1.,   9.],
       [  2.,  10.],
       [  3.,  11.],
       [  4.,  12.],
       [  5.,  13.],
       [  6.,  14.],
       [  7.,  15.],
       [  8.,  16.]])
>>> np.dot(a, b.T).T
array([[ 19.,  12.],
       [ 22.,  16.],
       [ 25.,  20.],
       [ 28.,  24.],
       [ 31.,  28.],
       [ 34.,  32.],
       [ 37.,  36.],
       [ 40.,  40.]])
```

ドットはにですが、エインサムをするがいもあります。とのものはのようになります

```
>>> np.einsum('...ij,...j', a, b)
```

しくなりますが、のリストにするのリストがけられます。これは、ドットをったになプロセスです。

```
>>> a
array([[[ 0,  1],
        [ 2,  3]],
       [[ 4,  5],
        [ 6,  7]],
       [[ 8,  9],
        [10, 11]],
       [[12, 13],
        [14, 15]],
       [[16, 17],
        [18, 19]],
       [[20, 21],
        [22, 23]],
       [[24, 25],
```

```
        [26, 27]],
       [[28, 29],
        [30, 31]]])
>>> np.einsum('...ij,...j', a, b)
array([[   9.,   29.],
       [  58.,   82.],
       [ 123.,  151.],
       [ 204.,  236.],
       [ 301.,  337.],
       [ 414.,  454.],
       [ 543.,  587.],
       [ 688.,  736.]])
```

numpy.dotは、リストのベクトルのをのリストのするベクトルとつけるためにできます。これは、のにったごとのおよびとすると、かなりでいです。このようなものにはもっときながですが、ほとんどされます

```
>>> np.diag(np.dot(b,b.T))
array([  82.,  104.,  130.,  160.,  194.,  232.,  274.,  320.])
```

ごとのとをしたドットプロダクト

```
>>> (b * b).sum(axis=-1)
array([  82.,  104.,  130.,  160.,  194.,  232.,  274.,  320.])
```

エインサムをすることは、

```
>>> np.einsum('...j,...j', b, b)
array([  82.,  104.,  130.,  160.,  194.,  232.,  274.,  320.])
```

オンラインでnumpy.dotをむ https://riptutorial.com/ja/numpy/topic/3198/numpy-dot

6: numpyのファイルIO

Examples

バイナリファイルをしたnumpyのとロード

```
x = np.random.random([100,100])
x.tofile('/path/to/dir/saved_binary.npy')
y = fromfile('/path/to/dir/saved_binary.npy')
z = y.reshape(100,100)
all(x==z)
# Output:
#     True
```

したのテキストファイルからデータをみむ

`np.loadtxt`は、CSVのファイルをみるためにできます。

```
# File:
#    # Col_1 Col_2
#    1, 1
#    2, 4
#    3, 9
np.loadtxt('/path/to/dir/csvlike.txt', delimiter=',', comments='#')
# Output:
# array([[ 1.,  1.],
#        [ 2.,  4.],
#        [ 3.,  9.]])
```

`np.fromregex`をってじファイルをむことができます

```
np.fromregex('/path/to/dir/csvlike.txt', r'(\d+),\s(\d+)', np.int64)
# Output:
# array([[1, 1],
#        [2, 4],
#        [3, 9]])
```

データをCSVのASCIIファイルとしてする

アナログ`np.loadtxt`、 `np.savetxt`をしてデータをASCIIファイルにできます

```
import numpy as np
x = np.random.random([100,100])
np.savetxt("filename.txt", x)
```

フォーマットをするには

```
np.savetxt("filename.txt", x, delimiter=", " ,
```

```
    newline="\n", comments="$ ", fmt="%1.2f",
    header="commented example text")
```

```
$ commented example text
0.30, 0.61, 0.34, 0.13, 0.52, 0.62, 0.35, 0.87, 0.48, [...]
```

CSVファイルをむ

な3つのなマニュアルページの

> `fromfile` - のデータをつバイナリデータをみみ、にされたテキストファイルをするにな。このをすると、tofileメソッドをしてきまれたデータをみることができます。
>
> `genfromtxt` - テキストファイルからデータをロードします。はどおりにされます。のskip_headerのはりでされ、コメントにくはされます。
>
> `loadtxt` - テキストファイルからデータをみみます。テキストファイルのは、じのをつがあります。

`genfromtxt`のためのラッパーである`loadtxt`。 `genfromtxt`は、ファイルをするためのくのパラメータをっているため、もです。

したの、したデータまたは

ファイル`myfile.csv`にのを`myfile.csv`します。

```
#descriptive text line to skip
1.0, 2, 3
4, 5.5, 6

import numpy as np
np.genfromtxt('path/to/myfile.csv',delimiter=',',skiprows=1)
```

をえる

```
array([[ 1. ,  2. ,  3. ],
       [ 4. ,  5.5,  6. ]])
```

したの、データ

```
1   2.0000  buckle_my_shoe
3   4.0000  margery_door

import numpy as np
np.genfromtxt('filename', dtype= None)


array([(1, 2.0, 'buckle_my_shoe'), (3, 4.0, 'margery_door')],
dtype=[('f0', '<i4'), ('f1', '<f8'), ('f2', '|S14')])
```

`dtype=None`すると、がわれます。

の

ファイル1 2 3 4 5 6 7 8 9 10 11 22 13 14 15 16 17 18 19 20 21 22 23 24

に

```
result=np.fromfile(path_to_file,dtype=float,sep="\t",count=-1)
```

オンラインでnumpyのファイルIOをむ https://riptutorial.com/ja/numpy/topic/4973/numpyのファイルio

7: データのフィルタリング

Examples

ブールによるデータのフィルタリング

numpyの`where`にえられたが1つだけの、それはtrue `numpy.nonzero`とじとされる `condition` のインデックスをします。これは、えられたをたすのインデックスをするためにできます。

```
import numpy as np

a = np.arange(20).reshape(2,10)
# a = array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
#            [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]])

# Generate boolean array indicating which values in a are both greater than 7 and less than 13
condition = np.bitwise_and(a>7, a<13)
# condition = array([[False, False, False, False, False, False, False, False,  True, True],
#                    [True,  True,  True, False, False, False, False, False, False, False]],
dtype=bool)

# Get the indices of a where the condition is True
ind = np.where(condition)
# ind = (array([0, 0, 1, 1, 1]), array([8, 9, 0, 1, 2]))

keep = a[ind]
# keep = [ 8  9 10 11 12]
```

インデックスをとしないは、 `extract`をして1つのステップでできます。ここでは、 `condition`をのとしてしますが、2のとしてがtrueのをすよう`array`にえます。

```
# np.extract(condition, array)
keep = np.extract(condition, a)
# keep = [ 8  9 10 11 12]
```

2つの`x`と`y`を`where` `y`ことができます。この、には`x`のが`True`で、 `y`のは`False`ます。

```
# Set elements of a which are NOT greater than 7 and less than 13 to zero, np.where(condition,
x, y)
a = np.where(condition, a, a*0)
print(a)
# Out: array([[ 0,  0,  0,  0,  0,  0,  0,  0,  8,  9],
#             [10, 11, 12,  0,  0,  0,  0,  0,  0,  0]])
```

インデックスをフィルタリングする

なケースでは、データをフィルタリングできます。

```
a = np.random.normal(size=10)
print(a)
```

```
#[-1.19423121  1.10481873  0.26332982 -0.53300387 -0.04809928  1.77107775
# 1.16741359  0.17699948 -0.06342169 -1.74213078]
b = a[a>0]
print(b)
#[ 1.10481873  0.26332982  1.77107775  1.16741359  0.17699948]
```

オンラインでデータのフィルタリングをむ https://riptutorial.com/ja/numpy/topic/6187/データのフィルタリング

8: ブーールインデックス

Examples

ブーールの

ブーールは、をするときに`dtype=bool`をしてでできます。 `0` 、 `None` 、 `False`またはののはTrueとなされます。

```
import numpy as np

bool_arr = np.array([1, 0.5, 0, None, 'a', '', True, False], dtype=bool)
print(bool_arr)
# output: [ True  True False False  True False  True False]
```

あるいは、numpyは、とスカラーのので、またはじのでがわれるときにブーールのをにします。

```
arr_1 = np.random.randn(3, 3)
arr_2 = np.random.randn(3, 3)

bool_arr = arr_1 < 0.5
print(bool_arr.dtype)
# output: bool

bool_arr = arr_1 < arr_2
print(bool_arr.dtype)
# output: bool
```

オンラインでブーールインデックスをむ https://riptutorial.com/ja/numpy/topic/6072/ブーールインデックス

9: ランダムデータの

き

NumPyの`random`モジュールは、のおよびをするランダムデータをするためのなをする。

ここにのがあります 。

Examples

なランダムの

```
# Generates 5 random numbers from a uniform distribution [0, 1)
np.random.rand(5)
# Out: array([ 0.4071833 ,  0.069167  ,  0.69742877,  0.45354268,  0.7220556 ])
```

をする

`random.seed`

```
np.random.seed(0)
np.random.rand(5)
# Out: array([ 0.5488135 ,  0.71518937,  0.60276338,  0.54488318,  0.4236548 ])
```

ジェネレータオブジェクトをすることによって

```
prng = np.random.RandomState(0)
prng.rand(5)
# Out: array([ 0.5488135 ,  0.71518937,  0.60276338,  0.54488318,  0.4236548 ])
```

ランダムなの

```
# Creates a 5x5 random integer array ranging from 10 (inclusive) to 20 (inclusive)
np.random.randint(10, 20, (5, 5))

'''
Out: array([[12, 14, 17, 16, 18],
            [18, 11, 16, 17, 17],
            [18, 11, 15, 19, 18],
            [19, 14, 13, 10, 13],
            [15, 10, 12, 13, 18]])
'''
```

からランダムサンプルをする

```
letters = list('abcde')
```

3をランダムにします きえ - じをできます。

```
np.random.choice(letters, 3)
'''
Out: array(['e', 'e', 'd'],
      dtype='<U1')
'''
```

なしのサンプリング

```
np.random.choice(letters, 3, replace=False)
'''
Out: array(['a', 'c', 'd'],
      dtype='<U1')
'''
```

にをりてる

```
# Choses 'a' with 40% chance, 'b' with 30% and the remaining ones with 10% each
np.random.choice(letters, size=10, p=[0.4, 0.3, 0.1, 0.1, 0.1])

'''
Out: array(['a', 'b', 'e', 'b', 'a', 'b', 'b', 'c', 'a', 'b'],
  dtype='<U1')
'''
```

のからしたをする

ガウスのサンプルをする

```
# Generate 5 random numbers from a standard normal distribution
# (mean = 0, standard deviation = 1)
np.random.randn(5)
# Out: array([-0.84423086,  0.70564081, -0.39878617, -0.82719653, -0.4157447 ])

# This result can also be achieved with the more general np.random.normal
np.random.normal(0, 1, 5)
# Out: array([-0.84423086,  0.70564081, -0.39878617, -0.82719653, -0.4157447 ])

# Specify the distribution's parameters
# Generate 5 random numbers drawn from a normal distribution with mean=70, std=10
np.random.normal(70, 10, 5)
# Out: array([ 72.06498837,  65.43118674,  59.40024236,  76.14957316,  84.29660766])
```

`numpy.random`には、 `poisson` 、 `binomial` 、 `logistic`などのいくつかのがあります

```
np.random.poisson(2.5, 5)  # 5 numbers, lambda=5
# Out: array([0, 2, 4, 3, 5])

np.random.binomial(4, 0.3, 5)  # 5 numbers, n=4, p=0.3
# Out: array([1, 0, 2, 1, 0])

np.random.logistic(2.3, 1.2, 5)  # 5 numbers, location=2.3, scale=1.2
# Out: array([ 1.23471936,  2.28598718, -0.81045893,  2.2474899 ,  4.15836878])
```

オンラインでランダムデータのをむ https://riptutorial.com/ja/numpy/topic/2060/ランダムデータの

10: な

き

ラインまたはのをデ―タポイントのセットにてはめる。

Examples

np.polyfitの

に、$ fx= mx + c $のでするデ―タセットをします。

```
npoints = 20
slope = 2
offset = 3
x = np.arange(npoints)
y = slope * x + offset + np.random.normal(size=npoints)
p = np.polyfit(x,y,1)           # Last argument is degree of polynomial
```

たちがったことをるには

```
import matplotlib.pyplot as plt
f = np.poly1d(p)                # So we can call f(x)
fig = plt.figure()
ax  = fig.add_subplot(111)
plt.plot(x, y, 'bo', label="Data")
plt.plot(x,f(x), 'b-',label="Polyfit")
plt.show()
```

このは、 https: //docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.htmlのnumpyのドキュメントににによくています。

np.linalg.lstsqの

polyfitとじデ―タセットをします。

```
npoints = 20
slope = 2
offset = 3
x = np.arange(npoints)
y = slope * x + offset + np.random.normal(size=npoints)
```

、| cA b | ** 2をすることによって、A b = cのシステムをすることによってをつけることをみる

```
import matplotlib.pyplot as plt # So we can plot the resulting fit
A = np.vstack([x,np.ones(npoints)]).T
m, c = np.linalg.lstsq(A, y)[0] # Don't care about residuals right now
```

```
fig = plt.figure()
ax  = fig.add_subplot(111)
plt.plot(x, y, 'bo', label="Data")
plt.plot(x, m*x+c, 'r--',label="Least Squares")
plt.show()
```

このは、 https: //docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.lstsq.htmlにあるnumpyのマニュアルにしています 。

オンラインでなをむ https://riptutorial.com/ja/numpy/topic/8808/な

11:

き

Nのまたは`ndarrays`は、numpyのコアオブジェクトで、じデータのをするためにされます。これらは、のPythonのよりれたなデータをします。

なり、やベクトルのからデータにするをする。ベクトルは、のループよりもはるかににされます

Examples

の

の

```
np.empty((2,3))
```

この、こののはされていないことにしてください。したがって、をするこのは、がでコードにめまれるにのみです。

リストから

```
np.array([0,1,2,3])
# Out: array([0, 1, 2, 3])
```

をする

```
np.arange(4)
# Out: array([0, 1, 2, 3])
```

ゼロをする

```
np.zeros((3,2))
# Out:
# array([[ 0.,  0.],
#        [ 0.,  0.],
#        [ 0.,  0.]])
```

する

```
np.ones((3,2))
# Out:
# array([[ 1.,  1.],
#        [ 1.,  1.],
#        [ 1.,  1.]])
```

のアイテムをする

```
np.linspace(0,1,21)
# Out:
# array([ 0.  ,  0.05,  0.1 ,  0.15,  0.2 ,  0.25,  0.3 ,  0.35,  0.4 ,
#         0.45,  0.5 ,  0.55,  0.6 ,  0.65,  0.7 ,  0.75,  0.8 ,  0.85,
#         0.9 ,  0.95,  1.  ])
```

ログのあるアイテムをする

```
np.logspace(-2,2,5)
# Out:
# array([  1.00000000e-02,   1.00000000e-01,   1.00000000e+00,
#          1.00000000e+01,   1.00000000e+02])
```

されたからをする

```
np.fromfunction(lambda i: i**2, (5,))
# Out:
# array([  0.,   1.,   4.,   9.,  16.])
np.fromfunction(lambda i,j: i**2, (3,3))
# Out:
# array([[ 0.,  0.,  0.],
#        [ 1.,  1.,  1.],
#        [ 4.,  4.,  4.]])
```

```
x = np.arange(4)
x
#Out:array([0, 1, 2, 3])
```

スカラーはにです

```
x+10
#Out: array([10, 11, 12, 13])
```

スカラーはにです

```
x*2
#Out: array([0, 2, 4, 6])
```

のはにです

```
x+x
#Out: array([0, 2, 4, 6])
```

のはにです

```
x*x
#Out: array([0, 1, 4, 9])
```

ドットまたは、よりにはは、

```
x.dot(x)
#Out: 14
```

Python 3.5では、 `@`がのとしてされました

```
x = np.diag(np.arange(4))
print(x)
'''
   Out: array([[0, 0, 0, 0],
   [0, 1, 0, 0],
   [0, 0, 2, 0],
   [0, 0, 0, 3]])
'''
print(x@x)
print(x)
'''
   Out: array([[0, 0, 0, 0],
   [0, 1, 0, 0],
   [0, 0, 4, 0],
   [0, 0, 0, 9]])
'''
```

します。をしたコピーをします。インプレースではありません。

```
#np.append(array, values_to_append, axis=None)
x = np.array([0,1,2,3,4])
np.append(x, [5,6,7,8,9])
# Out: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x
# Out: array([0, 1, 2, 3, 4])
y = np.append(x, [5,6,7,8,9])
y
# Out: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

hstack 。スタック。 カラムスタック
vstack 。スタック。 スタック

```
# np.hstack(tup), np.vstack(tup)
x = np.array([0,0,0])
y = np.array([1,1,1])
z = np.array([2,2,2])
np.hstack(x,y,z)
# Out: array([0, 0, 0, 1, 1, 1, 2, 2, 2])
np.vstack(x,y,z)
# Out: array([[0, 0, 0],
#             [1, 1, 1],
#             [2, 2, 2]])
```

アクセス

スライスシンタックスは`i:j:k`であり、 `i`はインデックスをむ、 `j`はインデックス、 `k`はステップサイズです。のPythonデータとに、ののインデックスは0です。

```
x = np.arange(10)
x[0]
# Out: 0

x[0:4]
# Out: array([0, 1, 2, 3])

x[0:4:2]
# Out:array([0, 2])
```

のはのからえられます。 `-1`はののにアクセスします

```
x[-1]
# Out: 9
x[-1:0:-1]
# Out: array([9, 8, 7, 6, 5, 4, 3, 2, 1])
```

は、をコンマでってすることによってアクセスできます。これまでのルーールはすべてされます。

```
x = np.arange(16).reshape((4,4))
x
# Out:
#     array([[ 0,  1,  2,  3],
#            [ 4,  5,  6,  7],
#            [ 8,  9, 10, 11],
#            [12, 13, 14, 15]])

x[1,1]
# Out: 5

x[0:3,0]
# Out: array([0, 4, 8])

x[0:3, 0:3]
# Out:
#     array([[ 0,  1,  2],
#            [ 4,  5,  6],
#            [ 8,  9, 10]])

x[0:3:2, 0:3:2]
# Out:
#     array([[ 0,  2],
#            [ 8, 10]])
```

の

```
arr = np.arange(10).reshape(2, 5)
```

`.transpose`メソッドをう

```
arr.transpose()
# Out:
#      array([[0, 5],
#            [1, 6],
#            [2, 7],
```

```
#            [3, 8],
#            [4, 9]])
```

`.T`メソッド

```
arr.T
# Out:
#     array([[0, 5],
#            [1, 6],
#            [2, 7],
#            [3, 8],
#            [4, 9]])
```

または`np.transpose`

```
np.transpose(arr)
# Out:
#     array([[0, 5],
#            [1, 6],
#            [2, 7],
#            [3, 8],
#            [4, 9]])
```

2の、これはのようになのとです。 nの、のをすることができます。デフォルトでは、これは`array.shape`にし`array.shape`

```
a = np.arange(12).reshape((3,2,2))
a.transpose() # equivalent to a.transpose(2,1,0)
# Out:
#   array([[[ 0,  4,  8],
#           [ 2,  6, 10]],
#
#          [[ 1,  5,  9],
#           [ 3,  7, 11]]])
```

しかし、インデックスののがです

```
a.transpose(2,0,1)
# Out:
#    array([[[ 0,  2],
#            [ 4,  6],
#            [ 8, 10]],
#
#           [[ 1,  3],
#            [ 5,  7],
#            [ 9, 11]]])

a = np.arange(24).reshape((2,3,4))  # shape (2,3,4)
a.transpose(2,0,1).shape
# Out:
#    (4, 2, 3)
```

ブールインデックス

```
arr = np.arange(7)
print(arr)
# Out: array([0, 1, 2, 3, 4, 5, 6])
```

スカラーとのはbooleanをします

```
arr > 4
# Out: array([False, False, False, False, False,  True,  True], dtype=bool)
```

このをして、4よりきいのみをすることができます。

```
arr[arr>4]
# Out: array([5, 6])
```

ブールインデックスは、なるでできますする。

```
# Two related arrays of same length, i.e. parallel arrays
idxs = np.arange(10)
sqrs = idxs**2

# Retrieve elements from one array using a condition on the other
my_sqrs = sqrs[idxs % 2 == 0]
print(my_sqrs)
# Out: array([0, 4, 16, 36, 64])
```

の

`numpy.reshape` じ`numpy.ndarray.reshape` メソッドは、じサイズのをすが、しい

```
print(np.arange(10).reshape((2, 5)))
# [[0 1 2 3 4]
#  [5 6 7 8 9]]
```

しいをし、そのでしません

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

ただし、 `ndarray` `shape`をきすることは`ndarray`。

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

このははくかもしれませんが、 `ndarray`はしたメモリブロックにされ、その`shape`はこのデータス

トリームをオブジェクトとしてどのようにするかをするだけです。

`shape`タプルの1つののは`-1`ます。 `numpy`はあなたのためにこののさをします

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

または

```
a = np.arange(12)
print(a.reshape((3, 2, -1)))

# [[[ 0  1]
#   [ 2  3]]

#  [[ 4  5]
#   [ 6  7]]

#  [[ 8  9]
#   [10 11]]]
```

`a.reshape((3, -1, -1))`などののされていないディメンションはされず、 `ValueError`をスローします。

アレイのブロードキャスト

はNumpyでできます。じのの、するインデックスのでがされることをします。

```
# Create two arrays of the same size
a = np.arange(6).reshape(2, 3)
b = np.ones(6).reshape(2, 3)

a
# array([0, 1, 2],
#       [3, 4, 5])
b
# array([1, 1, 1],
#       [1, 1, 1])

# a + b: a and b are added elementwise
a + b
# array([1, 2, 3],
#       [4, 5, 6])
```

は、ナンシーNumpy によってなるのにしてすることもできる。に、のはののに「ブロードキャスト」されているので、のはするのサブにしてされます。

```
# Create arrays of shapes (1, 5) and (13, 1) respectively
a = np.arange(5).reshape(1, 5)
a
```

```
# array([[0, 1, 2, 3, 4]])
b = np.arange(4).reshape(4, 1)
b
# array([0],
#       [1],
#       [2],
#       [3])

# When multiplying a * b, slices with the same dimensions are multiplied
# elementwise. In the case of a * b, the one and only row of a is multiplied
# with each scalar down the one and only column of b.
a*b
# array([[ 0,  0,  0,  0,  0],
#        [ 0,  1,  2,  3,  4],
#        [ 0,  2,  4,  6,  8],
#        [ 0,  3,  6,  9, 12]])
```

これをさらにするために、2Dアレイと3Dアレイをのサブディメンションですることをしてください。

```
# Create arrays of shapes (2, 2, 3) and (2, 3) respectively
a = np.arange(12).reshape(2, 2, 3)
a
# array([[[ 0  1  2]
#         [ 3  4  5]]
#
#        [[ 6  7  8]
#         [ 9 10 11]]])
b = np.arange(6).reshape(2, 3)
# array([[0, 1, 2],
#        [3, 4, 5]])

# Executing a*b broadcasts b to each (2, 3) slice of a,
# multiplying elementwise.
a*b
# array([[[ 0,  1,  4],
#         [ 9, 16, 25]],
#
#        [[ 0,  7, 16],
#         [27, 40, 55]]])

# Executing b*a gives the same result, i.e. the smaller
# array is broadcast over the other.
```

はいつされますか

は、2つのアレイがのあるをするときにわれる。

シェイプは、のものからにコンポーネントごとにされます。 2つのディメンションは、じものか1つが$_1$にがあります。のがのよりもいをする、するはされない。

のあるの

```
(7, 5, 3)    # compatible because dimensions are the same
```

```
(7, 5, 3)


(7, 5, 3)    # compatible because second dimension is 1
(7, 1, 3)

(7, 5, 3, 5) # compatible because exceeding dimensions are not compared
      (3, 5)

(3, 4, 5)    # incompatible
   (5, 5)

(3, 4, 5)    # compatible
   (1, 5)
```

にするのはのとおりです。

にCSVファイルのをする

```
filePath = "file.csv"
data = np.genfromtxt(filePath)
```

くのオプションがサポートされています。 ドキュメントをしてください

```
data = np.genfromtxt(filePath, dtype='float', delimiter=';', skip_header=1, usecols=(0,1,3) )
```

ナンパーのnndarray

numpyのとなるデータは、 `ndarray` nのです。 `ndarray`のは

- すなわち、それらはじデータのをむ
- サイズのをむ 、のサイズをするnののののタプルによってえられる

1

```
x = np.arange(15)
# array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
x.shape
# (15,)
```

2

```
x = np.asarray([[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14]])
x
# array([[ 0, 1, 2, 3, 4],
# [ 5, 6, 7, 8, 9],
# [10, 11, 12, 13, 14]])
x.shape
# (3, 5)
```

3

```
np.arange(12).reshape([2,3,2])
```

をせずにをするには、のようにします。

```
x = np.empty([2, 2])
# array([[ 0., 0.],
# [ 0., 0.]])
```

データとキャスティング

データはデフォルトでfloatにされています

```
x = np.empty([2, 2])
# array([[ 0., 0.],
# [ 0., 0.]])

x.dtype
# dtype('float64')
```

いくつかのデータがされている、numpyはデータをします

```
x = np.asarray([[1, 2], [3, 4]])
x.dtype
# dtype('int32')
```

をうとき、numpyは`ndarray`のデータにわせてにをキャストしようとします

```
x[1, 1] = 1.5 # assign a float value
x[1, 1]
# 1
# value has been casted to int
x[1, 1] = 'z' # value cannot be casted, resulting in a ValueError
```

アレイのブロードキャストもしてください。

```
x = np.asarray([[1, 2], [3, 4]])
# array([[1, 2],
#        [3, 4]])
y = np.asarray([[5, 6]])
# array([[5, 6]])
```

のでは、2x2と1x2ベクトルがあります。それでもたちはをうことができます

```
# x + y
array([[ 6, 8],
       [ 8, 10]])
```

これは、`y`が「びた」ためです

```
array([[5, 6],
       [5, 6]])
```

xのにわせて

リソース

- からのndarrayの Nndarray
- クラス ndarray

オンラインでをむ https://riptutorial.com/ja/numpy/topic/1296/

12: のとみみ

き

ナンシーアレイはさまざまなでしてロードすることができます。

Examples

numpy.saveとnumpy.loadの

np.saveとnp.loadは、のサイズのnumpyのとロードのためのいやすいフレームワークをします。

```
import numpy as np

a = np.random.randint(10,size=(3,3))
np.save('arr', a)

a2 = np.load('arr.npy')
print a2
```

オンラインでのとみみをむ https://riptutorial.com/ja/numpy/topic/10891/のとみみ

クレジット

S. No		Contributors
1	numpyをいめる	Andras Deak, Chris Mueller, Code-Ninja, Community, Dux, edwinksl, grovduck, Hammer, hashcode55, Joshua Cook, karel, Kersten, Mpaull, prasastoadi, rlee827, sebix, user2314737, Yassie
2	ndarrayをサブクラスする	Eric
3	np.linalgをった	Daniel, DataSwede, Fermi paradox, Mahdi, Sean Easter
4	numpy.cross	bob.sacamento, Mad Physicist
5	numpy.dot	bpachev, paddyg, Shubham Dang
6	numpyのファイルIO	Alex, atomh33ls, Sparkler
7	データのフィルタリング	Alex, farleytpm
8	ブールインデックス	Chris Mueller
9	ランダムデータの	amin, ayhan, B8vrede, Dux, Fermi paradox, user2314737
10	な	Alex
11		Andras Deak, ayhan, B Samedi, B8vrede, Benjamin, DataSwede, Dux, Gwen, Hamlet, Hammer, KARANJ, Keith L, pixatlazaki, Ryan, Sean Easter, Sparkler, The Hagen, TPVasconcelos, user2314737
12	のとみみ	obachtos