NumPy - Guia rápido

NumPy - Introdução

NumPy é um pacote Python. Significa 'Python Numérico'. É uma biblioteca que consiste em objetos array multidimensionais e uma coleção de rotinas para processamento de array.

Numeric , o ancestral do NumPy, foi desenvolvido por Jim Hugunin. Outro pacote Numarray também foi desenvolvido, possuindo algumas funcionalidades adicionais. Em 2005, Travis Oliphant criou o pacote NumPy incorporando os recursos do Numarray ao pacote Numérico. Existem muitos contribuidores para este projeto de código aberto.

Operações usando NumPy

Usando NumPy, um desenvolvedor pode realizar as seguintes operações -

- Operações matemáticas e lógicas em arrays.
- Transformadas de Fourier e rotinas para manipulação de formas.
- Operações relacionadas à álgebra linear. NumPy possui funções integradas para álgebra linear e geração de números aleatórios.

NumPy – um substituto para MatLab

NumPy é frequentemente usado junto com pacotes como **SciPy** (Scientific Python) e **Mat-plotlib** (biblioteca de plotagem). Essa combinação é amplamente utilizada como substituto do MatLab, uma plataforma popular para computação técnica. No entanto, a alternativa Python ao MatLab é agora vista como uma linguagem de programação mais moderna e completa.

É de código aberto, o que é uma vantagem adicional do NumPy.

NumPy - Meio Ambiente

A distribuição padrão do Python não vem com o módulo NumPy. Uma alternativa leve é instalar o NumPy usando o popular instalador de pacotes Python, **pip** .

```
pip install numpy
```

A melhor maneira de habilitar o NumPy é usar um pacote binário instalável específico p seu sistema operacional. Esses binários contêm pilha SciPy completa (incluindo NumPy, SciPy,

matplotlib, IPython, SymPy e pacotes nose junto com o núcleo do Python).

janelas

Anaconda (de https://www.continuum.io) é uma distribuição Python gratuita para pilha SciPy. Também está disponível para Linux e Mac.

Canopy (https://www.enthought.com/products/canopy/) está disponível como distribuição gratuita e comercial com pilha SciPy completa para Windows, Linux e Mac.

Python (x,y): É uma distribuição Python gratuita com pilha SciPy e Spyder IDE para sistema operacional Windows. (Pode ser baixado em https://www.python-xy.github.io/)

Linux

Os gerenciadores de pacotes das respectivas distribuições Linux são usados para instalar um ou mais pacotes na pilha SciPy.

Para Ubuntu

```
sudo apt-get install python-numpy
python-scipy python-matplotlibipythonipythonnotebook python-pandas
python-sympy python-nose
```


Para Fedora

```
sudo yum install numpyscipy python-matplotlibipython
python-pandas sympy python-nose atlas-devel
```

Construindo a partir da fonte

Core Python (2.6.x, 2.7.x e 3.2.x em diante) deve ser instalado com distutils e o módulo zlib deve estar habilitado.

O compilador C GNU gcc (4.2 e superior) deve estar disponível.

Para instalar o NumPy, execute o seguinte comando.

```
Python setup.py install
```

Para testar se o módulo NumPy está instalado corretamente, tente importá-lo do prompt do Python.

```
import numpy
```

Se não estiver instalado, a seguinte mensagem de erro será exibida.

```
Traceback (most recent call last):
   File "<pyshell#0>", line 1, in <module>
      import numpy
ImportError: No module named 'numpy'
```

Alternativamente, o pacote NumPy é importado usando a seguinte sintaxe -

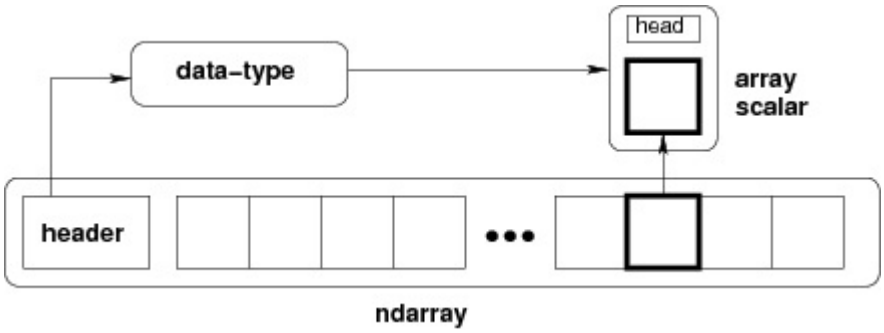
```
import numpy as np
```

NumPy - Objeto Ndarray

O objeto mais importante definido em NumPy é um tipo de array N-dimensional chamado **ndarray** . Descreve a coleção de itens do mesmo tipo. Os itens da coleção podem ser acessados usando um índice baseado em zero.

Cada item em um ndarray ocupa o mesmo tamanho de bloco na memória. Cada elemento em ndarray é um objeto do tipo de dados (chamado **dtype**).

Qualquer item extraído do objeto ndarray (por fatiamento) é representado por um objeto Python de um dos tipos escalares de array. O diagrama a seguir mostra um relacionamento entre ndarray, objeto de tipo de dados (dtype) e tipo escalar de array -



Uma instância da classe ndarray pode ser construída por diferentes rotinas de criação de array descritas posteriormente neste tutorial. O ndarray básico é criado usando uma função de array em NumPy da seguinte forma -

```
numpy.array
```

Ele cria um ndarray a partir de qualquer objeto que exponha a interface do array ou de qualquer método que retorne um array.

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

O construtor acima usa os seguintes parâmetros -

Sr. Não.	Parâmetro e Descrição
1	**objeto** Qualquer objeto que exponha o método de interface de array retorna um array ou qualquer sequência (aninhada).
2	**tipo d** Tipo de dados desejado da matriz, opcional
3	**cópia de** Opcional. Por padrão (true), o objeto é copiado
4	**ordem** C (linha principal) ou F (coluna principal) ou A (qualquer) (padrão)
5	**subok** Por padrão, o array retornado é forçado a ser um array de classe base. Se for verdade, as subclasses passaram
6	**ndmin** Especifica as dimensões mínimas da matriz resultante

Dê uma olhada nos exemplos a seguir para entender melhor.

Exemplo 1

```
import numpy as np
a = np.array([1,2,3])
print a
```

Demonstração ao vivo

A saída é a seguinte -

```
[1, 2, 3]
```

Exemplo 2

```
# more than one dimensions
import numpy as np
a = np.array([[1, 2], [3, 4]])
print a
```

Demonstração ao vivo

A saída é a seguinte -

```
[[1, 2]
 [3, 4]]
```

Exemplo 3

```
# minimum dimensions
import numpy as np
a = np.array([1, 2, 3,4,5], ndmin = 2)
print a
```

Demonstração ao vivo

A saída é a seguinte -

```
[[1, 2, 3, 4, 5]]
```

Exemplo 4

```
# dtype parameter
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print a
```

Demonstração ao vivo

A saída é a seguinte -

```
[ 1.+0.j,  2.+0.j,  3.+0.j]
```

O objeto **ndarray** consiste em um segmento unidimensional contíguo de memória do computador, combinado com um esquema de indexação que mapeia cada item para um local no bloco de memória. O bloco de memória contém os elementos em uma ordem de linha principal (estilo C) ou em uma ordem de coluna principal (estilo FORTRAN ou MatLab).

NumPy - Tipos de dados

NumPy suporta uma variedade muito maior de tipos numéricos do que Python. A tabela a seguir mostra diferentes tipos de dados escalares definidos em NumPy.

Sr. Não.	Tipos e descrição de dados
1	**bool_** Booleano (True ou False) armazenado como um byte
2	**int_** Tipo inteiro padrão (igual a C long; normalmente int64 ou int32)
3	**interno** Idêntico a C int (normalmente int32 ou int64)
4	**interno** Inteiro usado para indexação (igual a C ssize_t; normalmente int32 ou int64)
5	**int8** Bytes (-128 a 127)
6	**int16** Inteiro (-32768 a 32767)
7	**int32** Inteiro (-2147483648 a 2147483647)
8	**int64**

	Inteiro (-9223372036854775808 a 9223372036854775807)
9	**uint8** Inteiro sem sinal (0 a 255)
10	**uint16** Inteiro sem sinal (0 a 65535)
11	**uint32** Inteiro sem sinal (0 a 4294967295)
12	**uint64** Inteiro sem sinal (0 a 18446744073709551615)
13	**flutuador_** Abreviação de float64
14	**float16** Flutuação de meia precisão: bit de sinal, expoente de 5 bits, mantissa de 10 bits
15	**float32** Flutuação de precisão única: bit de sinal, expoente de 8 bits, mantissa de 23 bits
16	**float64** Flutuação de precisão dupla: bit de sinal, expoente de 11 bits, mantissa de 52 bits
17	**complexo_** Abreviação de complexo128
18	**complexo64** Número complexo, representado por dois floats de 32 bits (componentes reais e imaginários)
19	**complexo128** Número complexo, representado por dois floats de 64 bits (componentes reais e imaginários)

Os tipos numéricos NumPy são instâncias de objetos dtype (tipo de dados), cada um com características únicas. Os dtypes estão disponíveis como np.bool_, np.float32, etc.

Objetos de tipo de dados (dtype)

Um objeto de tipo de dados descreve a interpretação de um bloco fixo de memória correspondente a um array, dependendo dos seguintes aspectos -

- Tipo de dados (inteiro, flutuante ou objeto Python)

- Tamanho dos dados
- Ordem de bytes (little-endian ou big-endian)
- No caso do tipo estruturado, os nomes dos campos, tipo de dados de cada campo e parte do bloco de memória ocupado por cada campo.
- Se o tipo de dados for uma submatriz, sua forma e tipo de dados

A ordem dos bytes é decidida prefixando '<' ou '>' ao tipo de dados. '<' significa que a codificação é little-endian (o menos significativo é armazenado no menor endereço). '>' significa que a codificação é big endian (o byte mais significativo é armazenado no menor endereço).

Um objeto dtype é construído usando a seguinte sintaxe -

```
numpy.dtype(object, align, copy)
```

Os parâmetros são -

- **Object** - Para ser convertido em tipo de dados objeto
- **Align** - Se verdadeiro, adiciona preenchimento ao campo para torná-lo semelhante ao C-struct
- **Copy** - Faz uma nova cópia do objeto dtype. Se for falso, o resultado será uma referência ao objeto de tipo de dados integrado

Exemplo 1

Demonstração ao vivo

```
# using array-scalar type
import numpy as np
dt = np.dtype(np.int32)
print dt
```

A saída é a seguinte -

```
int32
```

Exemplo 2

Demonstração ao vivo

```
#int8, int16, int32, int64 can be replaced by equivalent string 'i1', 'i2','i4', etc.
import numpy as np

dt = np.dtype('i4')
print dt
```

A saída é a seguinte -

```
int32
```

Exemplo 3

```
# using endian notation
import numpy as np
dt = np.dtype('>i4')
print dt
```

Demonstração ao vivo

A saída é a seguinte -

```
>i4
```

Os exemplos a seguir mostram o uso do tipo de dados estruturados. Aqui, o nome do campo e o tipo de dados escalar correspondente devem ser declarados.

Exemplo 4

```
# first create structured data type
import numpy as np
dt = np.dtype([('age',np.int8)])
print dt
```

Demonstração ao vivo

A saída é a seguinte -

```
[('age', 'i1')]
```

Exemplo 5

```
# now apply it to ndarray object
import numpy as np

dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype = dt)
print a
```

Demonstração ao vivo

A saída é a seguinte -

```
[(10,) (20,) (30,)]
```

Exemplo 6

```
# file name can be used to access content of age column
import numpy as np

dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype = dt)
print a['age']
```

Demonstração ao vivo

A saída é a seguinte -

```
[10 20 30]
```

Exemplo 7

Os exemplos a seguir definem um tipo de dados estruturado chamado **estudante** com um campo de string 'nome', um **campo inteiro** 'idade' e um **campo flutuante** 'marcas'. Este dtype é aplicado ao objeto ndarray.

```
import numpy as np
student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
print student
```

Demonstração ao vivo

A saída é a seguinte -

```
[('name', 'S20'), ('age', 'i1'), ('marks', '<f4')])
```

Exemplo 8

```
import numpy as np

student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype = student)
print a
```

Demonstração ao vivo

A saída é a seguinte -

```
[('abc', 21, 50.0), ('xyz', 18, 75.0)]
```

Cada tipo de dados integrado possui um código de caracteres que o identifica exclusivamente.

- **'b'** - booleano
- **'i'** - inteiro (assinado)
- **'u'** - inteiro sem sinal
- **'f'** - ponto flutuante
- **'c'** - ponto flutuante complexo
- **'m'** – timedelta
- **'M'** - data e hora
- **'O'** - objetos (Python)
- **'S', 'a'** - (byte-)string
- **'U'** - Unicode
- **'V'** - dados brutos (vazio)

NumPy - Atributos da matriz

Neste capítulo, discutiremos os vários atributos de array do NumPy.

ndarray.forma

Este atributo de array retorna uma tupla que consiste em dimensões de array. Também pode ser usado para redimensionar o array.

Exemplo 1

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print a.shape
```

Demonstração ao vivo

A saída é a seguinte -

```
(2, 3)
```

Exemplo 2

```
# this resizes the ndarray
import numpy as np

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

Demonstração ao vivo

A saída é a seguinte -

```
[[1, 2]
 [3, 4]
 [5, 6]]
```

Exemplo 3

NumPy também fornece uma função de remodelação para redimensionar um array.

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print b
```

Demonstração ao vivo

A saída é a seguinte -

```
[[1, 2]
 [3, 4]
 [5, 6]]
```

ndarray.ndim

Este atributo de array retorna o número de dimensões do array.

Exemplo 1

```
# an array of evenly spaced numbers
import numpy as np
a = np.arange(24)
print a
```

Demonstração ao vivo

A saída é a seguinte -

```
[0 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 17 18 19 20 21 22 23]
```

Exemplo 2

```
# this is one dimensional array
import numpy as np
a = np.arange(24)
a.ndim

# now reshape it
b = a.reshape(2,4,3)
print b
# b is having three dimensions
```

Demonstração ao vivo

A saída é a seguinte -

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

numpy.itemsize

Este atributo array retorna o comprimento de cada elemento do array em bytes.

Exemplo 1

```
# dtype of array is int8 (1 byte)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.int8)
print x.itemsize
```

Demonstração ao vivo

A saída é a seguinte -

```
1
```

Exemplo 2

```
# dtype of array is now float32 (4 bytes)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.float32)
print x.itemsize
```

Demonstração ao vivo

A saída é a seguinte -

```
4
```

numpy.flags

O objeto ndarray possui os seguintes atributos. Seus valores atuais são retornados por esta função.

Sr. Não.	Atributo e descrição
1	**C_CONTÍGUO (C)** Os dados estão em um único segmento contíguo estilo C
2	**F_CONTÍGUO (F)**

	Os dados estão em um único segmento contíguo no estilo Fortran
3	**PRÓPRIOS DADOS (O)** A matriz possui a memória que usa ou a empresta de outro objeto
4	**ESCRITÁVEL (W)** A área de dados pode ser gravada. Definir isso como False bloqueia os dados, tornando-os somente leitura
5	**ALINHADO (A)** Os dados e todos os elementos estão alinhados adequadamente para o hardware
6	**ATUALIZAÇÃOIFCOPY (U)** Este array é uma cópia de algum outro array. Quando este array for desalocado, o array base será atualizado com o conteúdo deste array

Exemplo

O exemplo a seguir mostra os valores atuais dos sinalizadores.

```
import numpy as np
x = np.array([1,2,3,4,5])
print x.flags
```

Demonstração ao vivo

A saída é a seguinte -

```
C_CONTIGUOUS : True
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False
```

NumPy - Rotinas de criação de array

Um novo objeto **ndarray** pode ser construído por qualquer uma das seguintes rotinas de criação de array ou usando um construtor ndarray de baixo nível.

numpy.vazio

Ele cria uma matriz não inicializada de formato e tipo especificados. Ele usa o seguinte construtor -

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

O construtor usa os seguintes parâmetros.

Sr. Não.	Parâmetro e Descrição
1	**Forma** Forma de um array vazio em int ou tupla de int
2	**Tipo D** Tipo de dados de saída desejado. Opcional
3	**Ordem** 'C' para matriz de linha principal estilo C, 'F' para matriz de coluna principal estilo FORTRAN

Exemplo

O código a seguir mostra um exemplo de uma matriz vazia.

```
import numpy as np
x = np.empty([3,2], dtype = int)
print x
```

Demonstração ao vivo

A saída é a seguinte -

```
[[22649312   1701344351]
 [1818321759  1885959276]
 [16779776   156368896]]
```

Note - Os elementos de um array mostram valores aleatórios porque não são inicializados.

numpy.zeros

Retorna uma nova matriz de tamanho especificado, preenchida com zeros.

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

O construtor usa os seguintes parâmetros.

Sr. Não.	Parâmetro e Descrição
1	**Forma** Forma de um array vazio em int ou sequência de int
2	**Tipo D** Tipo de dados de saída desejado. Opcional
3	**Ordem** 'C' para matriz de linha principal estilo C, 'F' para matriz de coluna principal estilo FORTRAN

Exemplo 1

```
# array of five zeros. Default dtype is float
import numpy as np
x = np.zeros(5)
print x
```

Demonstração ao vivo

A saída é a seguinte -

```
[ 0.  0.  0.  0.  0.]
```

Exemplo 2

```
import numpy as np
x = np.zeros((5,), dtype = np.int)
print x
```

Demonstração ao vivo

Agora, a saída seria a seguinte -

```
[0  0  0  0  0]
```

Exemplo 3

```
# custom type
import numpy as np
```

Demonstração ao vivo

```
x = np.zeros((2,2), dtype = [('x', 'i4'), ('y', 'i4')])
print x
```

Deve produzir a seguinte saída -

```
[[(0,0)(0,0)]
 [(0,0)(0,0)]]
```

numpy.ones

Retorna uma nova matriz de tamanho e tipo especificados, preenchida com uns.

```
numpy.ones(shape, dtype = None, order = 'C')
```

O construtor usa os seguintes parâmetros.

Sr. Não.	Parâmetro e Descrição
1	**Forma** Forma de um array vazio em int ou tupla de int
2	**Tipo D** Tipo de dados de saída desejado. Opcional
3	**Ordem** 'C' para matriz de linha principal estilo C, 'F' para matriz de coluna principal estilo FORTRAN

Exemplo 1

Demonstração ao vivo

```
# array of five ones. Default dtype is float
import numpy as np
x = np.ones(5)
print x
```

A saída é a seguinte -

```
[ 1.  1.  1.  1.  1.]
```

Exemplo 2

```
import numpy as np
x = np.ones([2,2], dtype = int)
print x
```

Demonstração ao vivo

Agora, a saída seria a seguinte -

```
[[1  1]
 [1  1]]
```

NumPy - Matriz de dados existentes

Neste capítulo, discutiremos como criar um array a partir de dados existentes.

numpy.asarray

Esta função é semelhante a numpy.array, exceto pelo fato de possuir menos parâmetros. Esta rotina é útil para converter a sequência Python em ndarray.

```
numpy.asarray(a, dtype = None, order = None)
```

O construtor usa os seguintes parâmetros.

Sr. Não.	Parâmetro e Descrição
1	**a** Insira dados em qualquer formato, como lista, lista de tuplas, tuplas, tupla de tuplas ou tupla de listas
2	**tipo d** Por padrão, o tipo de dados de entrada é aplicado ao ndarray resultante
3	**ordem** C (linha maior) ou F (coluna maior). C é o padrão

Os exemplos a seguir mostram como você pode usar a função **asarray** .

Exemplo 1

```
# convert list to ndarray
import numpy as np

x = [1,2,3]
a = np.asarray(x)
print a
```

Demonstração ao vivo

Sua saída seria a seguinte -

```
[1  2  3]
```

Exemplo 2

```
# dtype is set
import numpy as np

x = [1,2,3]
a = np.asarray(x, dtype = float)
print a
```

Demonstração ao vivo

Agora, a saída seria a seguinte -

```
[ 1.  2.  3.]
```

Exemplo 3

```
# ndarray from tuple
import numpy as np

x = (1,2,3)
a = np.asarray(x)
print a
```

Demonstração ao vivo

Sua saída seria -

```
[1  2  3]
```

Exemplo 4

```
# ndarray from list of tuples
import numpy as np

x = [(1,2,3),(4,5)]
a = np.asarray(x)
print a
```

Demonstração ao vivo

Aqui, a saída seria a seguinte -

```
[(1, 2, 3) (4, 5)]
```

numpy.frombuffer

Esta função interpreta um buffer como um array unidimensional. Qualquer objeto que exponha a interface do buffer é usado como parâmetro para retornar um **ndarray** .

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

O construtor usa os seguintes parâmetros.

Sr. Não.	Parâmetro e Descrição
1	**amortecedor** Qualquer objeto que exponha a interface do buffer
2	**tipo d** Tipo de dados do ndarray retornado. O padrão é flutuar
3	**contar** O número de itens a serem lidos, o padrão -1 significa todos os dados
4	**desvio** A posição inicial a partir da qual ler. O padrão é 0

Exemplo

Os exemplos a seguir demonstram o uso da função **frombuffer** .

Demonstração ao vivo

```
import numpy as np
s = 'Hello World'
a = np.frombuffer(s, dtype = 'S1')
print a
```

Aqui está o resultado -

```
['H'  'e'  'l'  'l'  'o'  ' '  'W'  'o'  'r'  'l'  'd']
```

numpy.fromiter

Esta função cria um objeto **ndarray** a partir de qualquer objeto iterável. Uma nova matriz unidimensional é retornada por esta função.

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

Aqui, o construtor usa os seguintes parâmetros.

Sr. Não.	Parâmetro e Descrição
1	**iterável** Qualquer objeto iterável
2	**tipo d** Tipo de dados da matriz resultante
3	**contar** O número de itens a serem lidos do iterador. O padrão é -1, o que significa que todos os dados serão lidos

Os exemplos a seguir mostram como usar a função interna **range()** para retornar um objeto de lista. Um iterador desta lista é usado para formar um objeto **ndarray** .

Exemplo 1

```
# create list object using range function
import numpy as np
list = range(5)
print list
```

Demonstração ao vivo

Sua saída é a seguinte -

```
[0, 1, 2, 3, 4]
```

Exemplo 2

Demonstração ao vivo

```
# obtain iterator object from list
import numpy as np
list = range(5)
it = iter(list)

# use iterator to create ndarray
x = np.fromiter(it, dtype = float)
print x
```

Agora, a saída seria a seguinte -

```
[0.  1.  2.  3.  4.]
```

NumPy - Matriz de intervalos numéricos

Neste capítulo, veremos como criar um array a partir de intervalos numéricos.

numpy.arange

Esta função retorna um objeto **ndarray** contendo valores espaçados uniformemente dentro de um determinado intervalo. O formato da função é o seguinte -

```
numpy.arange(start, stop, step, dtype)
```

O construtor usa os seguintes parâmetros.

Sr. Não.	Parâmetro e Descrição
1	**começar** O início de um intervalo. Se omitido, o padrão é 0
2	**parar** O final de um intervalo (não incluindo este número)

3	**etapa** Espaçamento entre valores, o padrão é 1
4	**tipo d** Tipo de dados do ndarray resultante. Se não for fornecido, o tipo de dados de entrada será usado

Os exemplos a seguir mostram como você pode usar esta função.

Exemplo 1

```
import numpy as np
x = np.arange(5)
print x
```

Demonstração ao vivo

Sua saída seria a seguinte -

```
[0  1  2  3  4]
```

Exemplo 2

```
import numpy as np
# dtype set
x = np.arange(5, dtype = float)
print x
```

Demonstração ao vivo

Aqui, a saída seria -

```
[0.  1.  2.  3.  4.]
```

Exemplo 3

```
# start and stop parameters set
import numpy as np
x = np.arange(10,20,2)
print x
```

Demonstração ao vivo

Sua saída é a seguinte -

```
[10  12  14  16  18]
```

numpy.linspace

Esta função é semelhante à função **arange()** . Nesta função, em vez do tamanho do passo, é especificado o número de valores espaçados uniformemente entre o intervalo. O uso desta função é o seguinte -

```
numpy.linspace(start, stop, num, endpoint, retstep, dtype)
```

O construtor usa os seguintes parâmetros.

Sr. Não.	Parâmetro e Descrição
1	**começar** O valor inicial da sequência
2	**parar** O valor final da sequência, incluído na sequência se o ponto final estiver definido como verdadeiro
3	**número** O número de amostras uniformemente espaçadas a serem geradas. O padrão é 50
4	**ponto final** True por padrão, portanto, o valor stop é incluído na sequência. Se for falso, não está incluído
5	**retrocesso** Se verdadeiro, retorna amostras e alterna entre os números consecutivos
6	**tipo d** Tipo de dados de saída **ndarray**

Os exemplos a seguir demonstram a função use **linspace** .

Exemplo 1

Demonstração ao vivo

```
import numpy as np
x = np.linspace(10,20,5)
print x
```

Sua saída seria -

```
[10.  12.5  15.  17.5  20.]
```

Exemplo 2

Demonstração ao vivo

```
# endpoint set to false
import numpy as np
x = np.linspace(10,20, 5, endpoint = False)
print x
```

A saída seria -

```
[10.  12.  14.  16.  18.]
```

Exemplo 3

Demonstração ao vivo

```
# find retstep value
import numpy as np

x = np.linspace(1,2,5, retstep = True)
print x
# retstep here is 0.25
```

Agora, a saída seria -

```
(array([ 1.  ,  1.25,  1.5 ,  1.75,  2.  ]), 0.25)
```

numpy.logspace

Esta função retorna um objeto **ndarray** que contém os números espaçados uniformemente em uma escala logarítmica. Os pontos finais inicial e final da escala são índices da base, geralmente 10.

```
numpy.logspace(start, stop, num, endpoint, base, dtype)
```

Os parâmetros a seguir determinam a saída da função **logspace** .

Sr. Não.	Parâmetro e Descrição
1	**começar** O ponto de partida da sequência é base start
2	**parar** O valor final da sequência é parada base
3	**número** O número de valores entre o intervalo. O padrão é 50
4	**ponto final** Se for verdade, stop é o último valor do intervalo
5	**base** Base do espaço de log, o padrão é 10
6	**tipo d** Tipo de dados da matriz de saída. Se não for fornecido, depende de outros argumentos de entrada

Os exemplos a seguir ajudarão você a entender a função **logspace** .

Exemplo 1

Demonstração ao vivo

```
import numpy as np
# default base is 10
a = np.logspace(1.0, 2.0, num = 10)
print a
```

Sua saída seria a seguinte -

```
[ 10.          12.91549665    16.68100537     21.5443469  27.82559402
  35.93813664   46.41588834    59.94842503     77.42636827   100.    ]
```

Exemplo 2

```
# set base of log space to 2
import numpy as np
a = np.logspace(1,10,num = 10, base = 2)
print a
```

Demonstração ao vivo

Agora, a saída seria -

```
[ 2.    4.    8.   16.   32.   64.  128.  256.   512.  1024.]
```

NumPy - Indexação e fatiamento

O conteúdo do objeto ndarray pode ser acessado e modificado por indexação ou fatiamento, assim como os objetos contêineres integrados do Python.

Conforme mencionado anteriormente, os itens no objeto ndarray seguem o índice baseado em zero. Três tipos de métodos de indexação estão disponíveis – **acesso de campo, fatiamento básico** e **indexação avançada** .

O fatiamento básico é uma extensão do conceito básico do Python de fatiamento em n dimensões. Um objeto de fatia Python é construído fornecendo parâmetros **start, stop** e **step à função de fatia** integrada . Este objeto de fatia é passado para o array para extrair uma parte do array.

Exemplo 1

```
import numpy as np
a = np.arange(10)
s = slice(2,7,2)
print a[s]
```

Demonstração ao vivo

Sua saída é a seguinte -

```
[2  4  6]
```

No exemplo acima, um objeto **ndarray** é preparado pela função **arange()** . Em seguida, um objeto de fatia é definido com valores de início, parada e etapa 2, 7 e 2, respectivamente. Quando esse objeto de fatia é passado para o ndarray, uma parte dele começando com o índice 2 até 7 com um passo de 2 é fatiada.

O mesmo resultado também pode ser obtido fornecendo os parâmetros de fatiamento separados por dois pontos: (start:stop:step) diretamente ao objeto **ndarray** .

Exemplo 2

```
import numpy as np
a = np.arange(10)
b = a[2:7:2]
print b
```

Demonstração ao vivo

Aqui, obteremos a mesma saída -

```
[2  4  6]
```

Se for colocado apenas um parâmetro, será retornado um único item correspondente ao índice. Se um : for inserido na frente dele, todos os itens desse índice em diante serão extraídos. Se dois parâmetros (com : entre eles) forem usados, os itens entre os dois índices (não incluindo o índice de parada) com a etapa um padrão serão fatiados.

Exemplo 3

```
# slice single item
import numpy as np

a = np.arange(10)
b = a[5]
print b
```

Demonstração ao vivo

Sua saída é a seguinte -

```
5
```

Exemplo 4

```
# slice items starting from index
import numpy as np
```

Demonstração ao vivo

```
a = np.arange(10)
print a[2:]
```

Agora, a saída seria -

```
[2 3 4 5 6 7 8 9]
```

Exemplo 5

Demonstração ao vivo

```
# slice items between indexes
import numpy as np
a = np.arange(10)
print a[2:5]
```

Aqui, a saída seria -

```
[2 3 4]
```

A descrição acima também se aplica ao ndarray multidimensional .

Exemplo 6

Demonstração ao vivo

```
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print a

# slice items starting from index
print 'Now we will slice the array from the index a[1:]'
print a[1:]
```

A saída é a seguinte -

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]

Now we will slice the array from the index a[1:]
```

```
[[3 4 5]
 [4 5 6]]
```

O fatiamento também pode incluir reticências (...) para criar uma tupla de seleção do mesmo comprimento que a dimensão de uma matriz. Se reticências forem usadas na posição da linha, ela retornará um ndarray composto por itens nas linhas.

Exemplo 7

Demonstração ao vivo

```
# array to begin with
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])

print 'Our array is:'
print a
print '\n'

# this returns array of items in the second column
print 'The items in the second column are:'
print a[...,1]
print '\n'

# Now we will slice all items from the second row
print 'The items in the second row are:'
print a[1,...]
print '\n'

# Now we will slice all items from column 1 onwards
print 'The items column 1 onwards are:'
print a[...,1:]
```

A saída deste programa é a seguinte -

```
Our array is:
[[1 2 3]
 [3 4 5]
 [4 5 6]]

The items in the second column are:
[2 4 5]

The items in the second row are:
```

```
[3 4 5]

The items column 1 onwards are:
[[2 3]
 [4 5]
 [5 6]]
```

NumPy - Indexação Avançada

É possível fazer uma seleção de ndarray que seja uma sequência não tupla, objeto ndarray de tipo de dados inteiro ou booleano ou uma tupla com pelo menos um item sendo um objeto de sequência. A indexação avançada sempre retorna uma cópia dos dados. Contra isso, o fatiamento apresenta apenas uma visão.

Existem dois tipos de indexação avançada - **Inteira** e **Booleana** .

Indexação inteira

Este mecanismo ajuda a selecionar qualquer item arbitrário em um array com base em seu índice N-dimensional. Cada matriz inteira representa o número de índices nessa dimensão. Quando o índice consiste em tantas matrizes inteiras quanto as dimensões do ndarray de destino, ele se torna simples.

No exemplo a seguir, um elemento da coluna especificada de cada linha do objeto ndarray é selecionado. Conseqüentemente, o índice da linha contém todos os números das linhas e o índice da coluna especifica o elemento a ser selecionado.

Exemplo 1

Demonstração ao vivo

```
import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2], [0,1,0]]
print y
```

Sua saída seria a seguinte -

```
[1  4  5]
```

A seleção inclui elementos em (0,0), (1,1) e (2,0) da primeira matriz.

No exemplo a seguir, os elementos colocados nos cantos de uma matriz 4X3 são selecionados. Os índices de linha de seleção são [0, 0] e [3,3], enquanto os índices de coluna são [0,2] e [0,2].

Exemplo 2

Demonstração ao vivo

```
import numpy as np
x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])

print 'Our array is:'
print x
print '\n'

rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]

print 'The corner elements of this array are:'
print y
```

A saída deste programa é a seguinte -

```
Our array is:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

The corner elements of this array are:
[[ 0  2]
 [ 9 11]]
```

A seleção resultante é um objeto ndarray contendo elementos de canto.

A indexação avançada e básica pode ser combinada usando uma fatia (:) ou reticências (…) com uma matriz de índice. O exemplo a seguir usa fatia para linha e índice avançado para coluna. O resultado é o mesmo quando slice é usado para ambos. Mas o índice avançado resulta em cópia e pode ter layout de memória diferente.

Exemplo 3

Demonstração ao vivo

Demonstração ao vivo

```
import numpy as np
x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])

print 'Our array is:'
print x
print '\n'

# slicing
z = x[1:4,1:3]

print 'After slicing, our array becomes:'
print z
print '\n'

# using advanced index for column
y = x[1:4,[1,2]]

print 'Slicing using advanced index for column:'
print y
```

O resultado deste programa seria o seguinte -

```
Our array is:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

After slicing, our array becomes:
[[ 4  5]
 [ 7  8]
 [10 11]]

Slicing using advanced index for column:
[[ 4  5]
 [ 7  8]
 [10 11]]
```

Indexação de matriz booleana

Este tipo de indexação avançada é usado quando o objeto resultante é o resultado de operações booleanas, como operadores de comparação.

Exemplo 1

Neste exemplo, os itens maiores que 5 são retornados como resultado da indexação booleana.

Demonstração ao vivo

```
import numpy as np
x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])

print 'Our array is:'
print x
print '\n'

# Now we will print the items greater than 5
print 'The items greater than 5 are:'
print x[x > 5]
```

O resultado deste programa seria -

```
Our array is:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

The items greater than 5 are:
[ 6  7  8  9 10 11]
```

Exemplo 2

Neste exemplo, os elementos NaN (Not a Number) são omitidos usando ~ (operador de complemento).

Demonstração ao vivo

```
import numpy as np
a = np.array([np.nan, 1,2,np.nan,3,4,5])
print a[~np.isnan(a)]
```

Sua saída seria -

```
[ 1.  2.  3.  4.  5.]
```

Exemplo 3

O exemplo a seguir mostra como filtrar os elementos não complexos de uma matriz.

```
import numpy as np
a = np.array([1, 2+6j, 5, 3.5+5j])
print a[np.iscomplex(a)]
```

Demonstração ao vivo

Aqui, a saída é a seguinte -

```
[2.0+6.j  3.5+5.j]
```

NumPy - Transmissão

O termo **radiodifusão** refere-se à capacidade do NumPy de tratar matrizes de diferentes formatos durante operações aritméticas. As operações aritméticas em arrays geralmente são feitas nos elementos correspondentes. Se duas matrizes tiverem exatamente o mesmo formato, essas operações serão executadas sem problemas.

Exemplo 1

```
import numpy as np

a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print c
```

Demonstração ao vivo

Sua saída é a seguinte -

```
[10  40  90  160]
```

Se as dimensões de duas matrizes forem diferentes, as operações elemento a elemento não serão possíveis. No entanto, operações em matrizes de formatos não semelhantes ainda são possíveis no NumPy, devido à capacidade de transmissão. A matriz menor é **transmitida** para o tamanho da matriz maior para que tenham formas compatíveis.

A transmissão é possível se as seguintes regras forem satisfeitas -

- Matriz com **ndim** menor que a outra é anexada a '1' em seu formato.
- O tamanho em cada dimensão da forma de saída é o máximo dos tamanhos de entrada nessa dimensão.
- Uma entrada pode ser usada no cálculo, se seu tamanho em uma dimensão específica corresponder ao tamanho de saída ou se seu valor for exatamente 1.
- Se uma entrada tiver tamanho de dimensão 1, a primeira entrada de dados nessa dimensão será usada para todos os cálculos nessa dimensão.

Diz-se que um conjunto de matrizes pode ser **transmitido** se as regras acima produzirem um resultado válido e uma das seguintes afirmações for verdadeira -

- As matrizes têm exatamente o mesmo formato.
- As matrizes têm o mesmo número de dimensões e o comprimento de cada dimensão é um comprimento comum ou 1.
- Uma matriz com poucas dimensões pode ter sua forma anexada a uma dimensão de comprimento 1, de modo que a propriedade declarada acima seja verdadeira.

O programa a seguir mostra um exemplo de transmissão.

Exemplo 2

Demonstração ao vivo

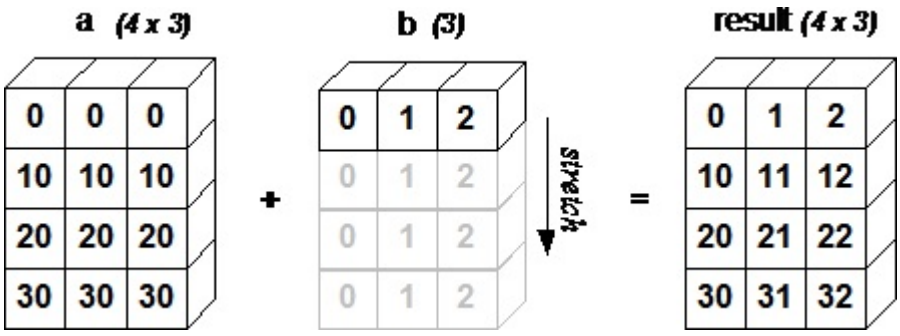
```
import numpy as np
a = np.array([[0.0,0.0,0.0],[10.0,10.0,10.0],[20.0,20.0,20.0],[30.0,30.0,30.0]])
b = np.array([1.0,2.0,3.0])

print 'First array:'
print a
print '\n'

print 'Second array:'
print b
print '\n'

print 'First Array + Second Array'
print a + b
```

O resultado deste programa seria o seguinte -

```
First array:
[[ 0. 0. 0.]
```

```
 [ 10. 10. 10.]
 [ 20. 20. 20.]
 [ 30. 30. 30.]]

Second array:
[ 1. 2. 3.]

First Array + Second Array
[[ 1. 2. 3.]
 [ 11. 12. 13.]
 [ 21. 22. 23.]
 [ 31. 32. 33.]]
```

A figura a seguir demonstra como o array **b** é transmitido para se tornar compatível com **a** .



NumPy - Iterando sobre array

O pacote NumPy contém um objeto iterador **numpy.nditer** . É um objeto iterador multidimensional eficiente com o qual é possível iterar sobre um array. Cada elemento de um array é visitado usando a interface Iterator padrão do Python.

Vamos criar um array 3X4 usando a função arange() e iterar sobre ele usando **nditer** .

Exemplo 1

Demonstração ao vivo

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)

print 'Original array is:'
print a
print '\n'

print 'Modified array is:'
```

```
for x in np.nditer(a):
   print x,
```

A saída deste programa é a seguinte -

```
Original array is:
[[ 0 5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Modified array is:
0 5 10 15 20 25 30 35 40 45 50 55
```

Exemplo 2

A ordem da iteração é escolhida para corresponder ao layout de memória de um array, sem considerar uma ordem específica. Isso pode ser visto iterando a transposição da matriz acima.

Demonstração ao vivo

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)

print 'Original array is:'
print a
print '\n'

print 'Transpose of the original array is:'
b = a.T
print b
print '\n'

print 'Modified array is:'
for x in np.nditer(b):
   print x,
```

A saída do programa acima é a seguinte -

```
Original array is:
[[ 0 5 10 15]
 [20 25 30 35]
 [40 45 50 55]]
```

```
Transpose of the original array is:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]

Modified array is:
0 5 10 15 20 25 30 35 40 45 50 55
```

Ordem de iteração

Se os mesmos elementos forem armazenados usando a ordem do estilo F, o iterador escolhe a maneira mais eficiente de iterar em uma matriz.

Exemplo 1

Demonstração ao vivo

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print 'Original array is:'
print a
print '\n'

print 'Transpose of the original array is:'
b = a.T
print b
print '\n'

print 'Sorted in C-style order:'
c = b.copy(order='C')
print c
for x in np.nditer(c):
   print x,

print '\n'

print 'Sorted in F-style order:'
c = b.copy(order='F')
print c
for x in np.nditer(c):
   print x,
```

Sua saída seria a seguinte -

```
Original array is:
[[ 0 5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Transpose of the original array is:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]

Sorted in C-style order:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]
0 20 40 5 25 45 10 30 50 15 35 55

Sorted in F-style order:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]
0 5 10 15 20 25 30 35 40 45 50 55
```

Exemplo 2

É possível forçar o objeto **nditer** a usar uma ordem específica mencionando-a explicitamente.

Demonstração ao vivo

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)

print 'Original array is:'
print a
print '\n'

print 'Sorted in C-style order:'
for x in np.nditer(a, order = 'C'):
   print x,
print '\n'
```

```
print 'Sorted in F-style order:'
for x in np.nditer(a, order = 'F'):
   print x,
```

Sua saída seria -

```
Original array is:
[[ 0 5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Sorted in C-style order:
0 5 10 15 20 25 30 35 40 45 50 55

Sorted in F-style order:
0 20 40 5 25 45 10 30 50 15 35 55
```

Modificando Valores de Matriz

O objeto **nditer** possui outro parâmetro opcional chamado **op_flags** . Seu valor padrão é somente leitura, mas pode ser definido para modo leitura-gravação ou somente gravação. Isso permitirá a modificação de elementos da matriz usando este iterador.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print 'Original array is:'
print a
print '\n'

for x in np.nditer(a, op_flags = ['readwrite']):
   x[...] = 2*x
print 'Modified array is:'
print a
```

Sua saída é a seguinte -

```
Original array is:
[[ 0 5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Modified array is:
[[ 0 10 20 30]
 [ 40 50 60 70]
 [ 80 90 100 110]]
```

Loop Externo

O construtor da classe nditer possui um parâmetro **'flags'** , que pode assumir os seguintes valores -

Sr. Não.	Parâmetro e Descrição
1	**índice_c** O índice C_order pode ser rastreado
2	**índice_f** O índice Fortran_order é rastreado
3	**multi-índice** Tipos de índices com um por iteração podem ser rastreados
4	**loop_externo** Faz com que os valores fornecidos sejam matrizes unidimensionais com vários valores em vez de matrizes de dimensão zero

Exemplo

No exemplo a seguir, as matrizes unidimensionais correspondentes a cada coluna são percorridas pelo iterador.

Demonstração ao vivo

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)

print 'Original array is:'
print a
print '\n'
```

```
print 'Modified array is:'
for x in np.nditer(a, flags = ['external_loop'], order = 'F'):
   print x,
```

A saída é a seguinte -

```
Original array is:
[[ 0 5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Modified array is:
[ 0 20 40] [ 5 25 45] [10 30 50] [15 35 55]
```

Iteração de transmissão

Se duas matrizes forem **transmitidas** , um objeto **nditer** combinado será capaz de iterar sobre elas simultaneamente. Supondo que um array **a** tenha dimensão 3X4 e haja outro array **b** de dimensão 1X4, o iterador do seguinte tipo é usado (o array **b** é transmitido para o tamanho de **a**).

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)

print 'First array is:'
print a
print '\n'

print 'Second array is:'
b = np.array([1, 2, 3, 4], dtype = int)
print b
print '\n'

print 'Modified array is:'
for x,y in np.nditer([a,b]):
   print "%d:%d" % (x,y),
```

Sua saída seria a seguinte -

```
First array is:
[[ 0 5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Second array is:
[1 2 3 4]

Modified array is:
0:1 5:2 10:3 15:4 20:1 25:2 30:3 35:4 40:1 45:2 50:3 55:4
```

NumPy - Manipulação de array

Várias rotinas estão disponíveis no pacote NumPy para manipulação de elementos no objeto ndarray. Eles podem ser classificados nos seguintes tipos -

Mudando de forma

Sr. Não.	Forma e Descrição
1	**remodelar** Dá uma nova forma a um array sem alterar seus dados
2	**plano** Um iterador 1-D sobre o array
3	**achatar** Retorna uma cópia do array recolhido em uma dimensão
4	**desvendar** Retorna uma matriz achatada contígua

Transpor operações

Sr. Não.	Operação e Descrição
1	**transpor** Permuta as dimensões de uma matriz

2	**ndarray.T** O mesmo que self.transpose()
3	**eixo de rolagem** Rola o eixo especificado para trás
4	**swapaxes** Troca os dois eixos de um array

Alterando Dimensões

Sr. Não.	Dimensão e descrição
1	**transmissão** Produz um objeto que imita a transmissão
2	**transmitir_para** Transmite uma matriz para uma nova forma
3	**expand_dims** Expande a forma de uma matriz
4	**espremer** Remove entradas unidimensionais da forma de uma matriz

Unindo matrizes

Sr. Não.	Matriz e descrição
1	**concatenar** Une uma sequência de arrays ao longo de um eixo existente
2	**pilha** Une uma sequência de arrays ao longo de um novo eixo
3	**pilha** Empilha matrizes em sequência horizontalmente (em colunas)
4	**vstack** Empilha matrizes em sequência verticalmente (linha)

Divisão de matrizes

Sr. Não.	Matriz e descrição
1	**dividir** Divide um array em vários submatrizes
2	**hsplit** Divide uma matriz em várias submatrizes horizontalmente (em colunas)
3	**vsdividido** Divide uma matriz em várias submatrizes verticalmente (em linhas)

Adicionando/Removendo Elementos

Sr. Não.	Elemento e Descrição
1	**redimensionar** Retorna um novo array com a forma especificada
2	**acrescentar** Acrescenta os valores ao final de uma matriz
3	**inserir** Insere os valores ao longo do eixo fornecido antes dos índices fornecidos
4	**excluir** Retorna uma nova matriz com submatrizes ao longo de um eixo excluído
5	**exclusivo** Encontra os elementos únicos de um array

NumPy - Operadores Binários

A seguir estão as funções para operações bit a bit disponíveis no pacote NumPy.

Sr. Não.	Operação e Descrição
1	**bit a bit_e** Calcula a operação AND bit a bit dos elementos da matriz
2	**bit a bit_ou** Calcula a operação OR bit a bit dos elementos da matriz
3	**invertido**

	Calcula NOT bit a bit
4	**desvio à esquerda** Desloca bits de uma representação binária para a esquerda
5	**deslocamento para a direita** Desloca bits da representação binária para a direita

NumPy - Funções de String

As funções a seguir são usadas para executar operações de string vetorizadas para matrizes de dtype numpy.string_ ou numpy.unicode_. Eles são baseados nas funções de string padrão da biblioteca interna do Python.

Sr. Não.	Descrição da função
1	**adicionar()** Retorna concatenação de strings elemento a elemento para duas matrizes de str ou Unicode
2	**multiplicar()** Retorna a string com concatenação múltipla, elemento a elemento
3	**Centro()** Retorna uma cópia da string fornecida com elementos centralizados em uma string de comprimento especificado
4	**capitalizar()** Retorna uma cópia da string com apenas o primeiro caractere em maiúscula
5	**título()** Retorna a versão do título elemento a elemento da string ou unicode
6	**mais baixo()** Retorna um array com os elementos convertidos em letras minúsculas
7	**superior()** Retorna um array com os elementos convertidos para letras maiúsculas
8	**dividir()** Retorna uma lista de palavras na string, usando separatordelimiter
9	**linhas divididas()** Retorna uma lista das linhas do elemento, quebrando nos limites da linha
10	**faixa()**

	Retorna uma cópia com os caracteres iniciais e finais removidos
11	**juntar()** Retorna uma string que é a concatenação das strings na sequência
12	**substituir()** Retorna uma cópia da string com todas as ocorrências de substring substituídas pela nova string
13	**decodificar()** Chama str.decode elemento a elemento
14	**codificar()** Chama str.encode elemento a elemento

Essas funções são definidas na classe de array de caracteres (numpy.char). O pacote Numarray mais antigo continha a classe chararray. As funções acima na classe numpy.char são úteis na execução de operações de string vetorizadas.

NumPy - Funções Matemáticas

Compreensivelmente, o NumPy contém um grande número de diversas operações matemáticas. NumPy fornece funções trigonométricas padrão, funções para operações aritméticas, manipulação de números complexos, etc.

Funções trigonométricas

NumPy possui funções trigonométricas padrão que retornam razões trigonométricas para um determinado ângulo em radianos.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([0,30,45,60,90])

print 'Sine of different angles:'
# Convert to radians by multiplying with pi/180
print np.sin(a*np.pi/180)
print '\n'

print 'Cosine values for angles in array:'
print np.cos(a*np.pi/180)
print '\n'
```

```
print 'Tangent values for given angles:'
print np.tan(a*np.pi/180)
```

Aqui está o resultado -

```
Sine of different angles:
[ 0.        0.5       0.70710678  0.8660254   1.        ]

Cosine values for angles in array:
[  1.00000000e+00   8.66025404e-01   7.07106781e-01   5.00000000e-01
   6.12323400e-17]

Tangent values for given angles:
[  0.00000000e+00   5.77350269e-01   1.00000000e+00   1.73205081e+00
   1.63312394e+16]
```

As funções arcsin, arcos e **arctan** retornam o inverso trigonométrico de sen, cos e tan do ângulo fornecido. O resultado dessas funções pode ser verificado pela **função numpy.degrees()** convertendo radianos em graus.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([0,30,45,60,90])

print 'Array containing sine values:'
sin = np.sin(a*np.pi/180)
print sin
print '\n'

print 'Compute sine inverse of angles. Returned values are in radians.'
inv = np.arcsin(sin)
print inv
print '\n'

print 'Check result by converting to degrees:'
print np.degrees(inv)
print '\n'

print 'arccos and arctan functions behave similarly:'
cos = np.cos(a*np.pi/180)
print cos
print '\n'
```

```
print 'Inverse of cos:'
inv = np.arccos(cos)
print inv
print '\n'

print 'In degrees:'
print np.degrees(inv)
print '\n'

print 'Tan function:'
tan = np.tan(a*np.pi/180)
print tan
print '\n'

print 'Inverse of tan:'
inv = np.arctan(tan)
print inv
print '\n'

print 'In degrees:'
print np.degrees(inv)
```

Sua saída é a seguinte -

```
Array containing sine values:
[ 0.        0.5       0.70710678  0.8660254   1.        ]

Compute sine inverse of angles. Returned values are in radians.
[ 0.        0.52359878  0.78539816  1.04719755  1.57079633]

Check result by converting to degrees:
[  0.  30.  45.  60.  90.]

arccos and arctan functions behave similarly:
[  1.00000000e+00   8.66025404e-01   7.07106781e-01   5.00000000e-01
   6.12323400e-17]

Inverse of cos:
[ 0.        0.52359878  0.78539816  1.04719755  1.57079633]

In degrees:
[  0.  30.  45.  60.  90.]
```

```
Tan function:
[  0.00000000e+00   5.77350269e-01   1.00000000e+00   1.73205081e+00
  1.63312394e+16]

Inverse of tan:
[ 0.        0.52359878  0.78539816  1.04719755  1.57079633]

In degrees:
[  0.  30.  45.  60.  90.]
```

Funções para arredondamento

numpy.around()

Esta é uma função que retorna o valor arredondado para a precisão desejada. A função usa os seguintes parâmetros.

```
numpy.around(a,decimals)
```

Onde,

Sr. Não.	Parâmetro e Descrição
1	**a** Dados de entrada
2	**decimais** O número de casas decimais para arredondar. O padrão é 0. Se for negativo, o número inteiro será arredondado para a posição à esquerda da vírgula decimal

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([1.0,5.55, 123, 0.567, 25.532])

print 'Original array:'
print a
print '\n'

print 'After rounding:'
print np.around(a)
```

```
print np.around(a, decimals = 1)
print np.around(a, decimals = -1)
```

Ele produz a seguinte saída -

```
Original array:
[  1.     5.55  123.     0.567  25.532]

After rounding:
[  1.   6.  123.   1.  26. ]
[  1.   5.6 123.   0.6 25.5]
[  0.   10. 120.   0.  30. ]
```

numpy.floor()

Esta função retorna o maior número inteiro não maior que o parâmetro de entrada. O piso do **escalar x** é o maior **inteiro i** , tal que **i <= x** . Observe que em Python, o piso sempre é arredondado de 0.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])

print 'The given array:'
print a
print '\n'

print 'The modified array:'
print np.floor(a)
```

Ele produz a seguinte saída -

```
The given array:
[ -1.7  1.5 -0.2  0.6  10. ]

The modified array:
[ -2.  1. -1.  0.  10.]
```

numpy.ceil()

A função ceil() retorna o teto de um valor de entrada, ou seja, o teto do **escalar x** é o menor **inteiro i** , tal que **i >= x.**

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])

print 'The given array:'
print a
print '\n'

print 'The modified array:'
print np.ceil(a)
```

Ele produzirá a seguinte saída -

```
The given array:
[ -1.7  1.5 -0.2  0.6  10. ]

The modified array:
[ -1.  2. -0.  1. 10.]
```

NumPy - Operações Aritméticas

Matrizes de entrada para realizar operações aritméticas como add(), subtract(), multiplicar() e dividir() devem ter o mesmo formato ou devem estar em conformidade com as regras de transmissão de matrizes.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)

print 'First array:'
print a
print '\n'

print 'Second array:'
b = np.array([10,10,10])
print b
```

```
print '\n'

print 'Add the two arrays:'
print np.add(a,b)
print '\n'

print 'Subtract the two arrays:'
print np.subtract(a,b)
print '\n'

print 'Multiply the two arrays:'
print np.multiply(a,b)
print '\n'

print 'Divide the two arrays:'
print np.divide(a,b)
```

Ele produzirá a seguinte saída -

```
First array:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]

Second array:
[10 10 10]

Add the two arrays:
[[ 10. 11. 12.]
 [ 13. 14. 15.]
 [ 16. 17. 18.]]

Subtract the two arrays:
[[-10. -9. -8.]
 [ -7. -6. -5.]
 [ -4. -3. -2.]]

Multiply the two arrays:
[[ 0. 10. 20.]
 [ 30. 40. 50.]
 [ 60. 70. 80.]]

Divide the two arrays:
[[ 0. 0.1 0.2]
```

```
 [ 0.3 0.4 0.5]
 [ 0.6 0.7 0.8]]
```

Vamos agora discutir algumas das outras funções aritméticas importantes disponíveis no NumPy.

numpy.recíproco()

Esta função retorna o inverso do argumento, elemento a elemento. Para elementos com valores absolutos maiores que 1, o resultado é sempre 0 devido à maneira como o Python lida com a divisão inteira. Para o número inteiro 0, um aviso de overflow é emitido.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([0.25, 1.33, 1, 0, 100])

print 'Our array is:'
print a
print '\n'

print 'After applying reciprocal function:'
print np.reciprocal(a)
print '\n'

b = np.array([100], dtype = int)
print 'The second array is:'
print b
print '\n'

print 'After applying reciprocal function:'
print np.reciprocal(b)
```

Ele produzirá a seguinte saída -

```
Our array is:
[  0.25   1.33   1.    0.   100.  ]

After applying reciprocal function:
main.py:9: RuntimeWarning: divide by zero encountered in reciprocal
  print np.reciprocal(a)
[ 4.        0.7518797  1.          inf  0.01    ]
```

```
The second array is:
[100]

After applying reciprocal function:
[0]
```

numpy.power()

Esta função trata os elementos do primeiro array de entrada como base e os retorna elevado à potência do elemento correspondente no segundo array de entrada.

Demonstração ao vivo

```
import numpy as np
a = np.array([10,100,1000])

print 'Our array is:'
print a
print '\n'

print 'Applying power function:'
print np.power(a,2)
print '\n'

print 'Second array:'
b = np.array([1,2,3])
print b
print '\n'

print 'Applying power function again:'
print np.power(a,b)
```

Ele produzirá a seguinte saída -

```
Our array is:
[  10  100 1000]

Applying power function:
[   100   10000 1000000]

Second array:
[1 2 3]
```

```
Applying power function again:
[       10      10000 1000000000]
```

numpy.mod()

Esta função retorna o restante da divisão dos elementos correspondentes na matriz de entrada. A função **numpy.remainder()** também produz o mesmo resultado.

Demonstração ao vivo

```
import numpy as np
a = np.array([10,20,30])
b = np.array([3,5,7])

print 'First array:'
print a
print '\n'

print 'Second array:'
print b
print '\n'

print 'Applying mod() function:'
print np.mod(a,b)
print '\n'

print 'Applying remainder() function:'
print np.remainder(a,b)
```

Ele produzirá a seguinte saída -

```
First array:
[10 20 30]

Second array:
[3 5 7]

Applying mod() function:
[1 0 2]

Applying remainder() function:
[1 0 2]
```

As funções a seguir são usadas para realizar operações em array com números complexos.

- **numpy.real()** - retorna a parte real do argumento do tipo de dados complexo.
- **numpy.imag()** - retorna a parte imaginária do argumento do tipo de dados complexo.
- **numpy.conj()** - retorna o conjugado complexo, que é obtido alterando o sinal da parte imaginária.
- **numpy.angle()** - retorna o ângulo do argumento complexo. A função possui parâmetro de grau. Se for verdade, o ângulo será retornado em graus; caso contrário, o ângulo estará em radianos.

Demonstração ao vivo

```
import numpy as np
a = np.array([-5.6j, 0.2j, 11. , 1+1j])

print 'Our array is:'
print a
print '\n'

print 'Applying real() function:'
print np.real(a)
print '\n'

print 'Applying imag() function:'
print np.imag(a)
print '\n'

print 'Applying conj() function:'
print np.conj(a)
print '\n'

print 'Applying angle() function:'
print np.angle(a)
print '\n'

print 'Applying angle() function again (result in degrees)'
print np.angle(a, deg = True)
```

Ele produzirá a seguinte saída -

```
Our array is:
[ 0.-5.6j 0.+0.2j 11.+0.j 1.+1.j ]

Applying real() function:
[ 0. 0. 11. 1.]
```

```
Applying imag() function:
[-5.6 0.2 0. 1. ]

Applying conj() function:
[ 0.+5.6j 0.-0.2j 11.-0.j 1.-1.j ]

Applying angle() function:
[-1.57079633 1.57079633 0. 0.78539816]

Applying angle() function again (result in degrees)
[-90. 90. 0. 45.]
```

NumPy - Funções Estatísticas

NumPy tem algumas funções estatísticas úteis para encontrar mínimo, máximo, desvio padrão percentil e variância, etc., dos elementos fornecidos na matriz. As funções são explicadas a seguir -

numpy.amin() e numpy.amax()

Essas funções retornam o mínimo e o máximo dos elementos de uma determinada matriz ao longo do eixo especificado.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print 'Our array is:'
print a
print '\n'

print 'Applying amin() function:'
print np.amin(a,1)
print '\n'

print 'Applying amin() function again:'
print np.amin(a,0)
print '\n'

print 'Applying amax() function:'
```

```
print np.amax(a)
print '\n'

print 'Applying amax() function again:'
print np.amax(a, axis = 0)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[3 7 5]
[8 4 3]
[2 4 9]]

Applying amin() function:
[3 3 2]

Applying amin() function again:
[2 4 3]

Applying amax() function:
9

Applying amax() function again:
[8 7 9]
```

numpy.ptp()

A função **numpy.ptp()** retorna o intervalo (máximo-mínimo) de valores ao longo de um eixo.

Demonstração ao vivo

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print 'Our array is:'
print a
print '\n'

print 'Applying ptp() function:'
print np.ptp(a)
print '\n'

print 'Applying ptp() function along axis 1:'
print np.ptp(a, axis = 1)
```

```
print '\n'

print 'Applying ptp() function along axis 0:'
print np.ptp(a, axis = 0)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[3 7 5]
[8 4 3]
[2 4 9]]

Applying ptp() function:
7

Applying ptp() function along axis 1:
[4 5 7]

Applying ptp() function along axis 0:
[6 3 6]
```

numpy.percentil()

Percentil (ou percentil) é uma medida usada em estatísticas que indica o valor abaixo do qual cai uma determinada porcentagem de observações em um grupo de observações. A função **numpy.percentile()** recebe os seguintes argumentos.

```
numpy.percentile(a, q, axis)
```

Onde,

Sr. Não.	Argumento e Descrição
1	**a** Matriz de entrada
2	**q** O percentil a ser calculado deve estar entre 0 e 100
3	**eixo** O eixo ao longo do qual o percentil deve ser calculado

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[30,40,70],[80,20,10],[50,90,60]])

print 'Our array is:'
print a
print '\n'

print 'Applying percentile() function:'
print np.percentile(a,50)
print '\n'

print 'Applying percentile() function along axis 1:'
print np.percentile(a,50, axis = 1)
print '\n'

print 'Applying percentile() function along axis 0:'
print np.percentile(a,50, axis = 0)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[30 40 70]
 [80 20 10]
 [50 90 60]]

Applying percentile() function:
50.0

Applying percentile() function along axis 1:
[ 40. 20. 60.]

Applying percentile() function along axis 0:
[ 50. 40. 60.]
```

numpy.median()

A mediana é definida como o valor que separa a metade superior de uma amostra de dados da metade inferior. A função **numpy.median()** é usada conforme mostrado no programa a seguir.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[30,65,70],[80,95,10],[50,90,60]])

print 'Our array is:'
print a
print '\n'

print 'Applying median() function:'
print np.median(a)
print '\n'

print 'Applying median() function along axis 0:'
print np.median(a, axis = 0)
print '\n'

print 'Applying median() function along axis 1:'
print np.median(a, axis = 1)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[30 65 70]
 [80 95 10]
 [50 90 60]]

Applying median() function:
65.0

Applying median() function along axis 0:
[ 50. 90. 60.]

Applying median() function along axis 1:
[ 65. 80. 60.]
```

numpy.mean()

A média aritmética é a soma dos elementos ao longo de um eixo dividida pelo número de elementos. A função **numpy.mean()** retorna a média aritmética dos elementos do array. Se o eixo for mencionado, ele será calculado ao longo dele.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])

print 'Our array is:'
print a
print '\n'

print 'Applying mean() function:'
print np.mean(a)
print '\n'

print 'Applying mean() function along axis 0:'
print np.mean(a, axis = 0)
print '\n'

print 'Applying mean() function along axis 1:'
print np.mean(a, axis = 1)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[1 2 3]
 [3 4 5]
 [4 5 6]]

Applying mean() function:
3.66666666667

Applying mean() function along axis 0:
[ 2.66666667 3.66666667 4.66666667]

Applying mean() function along axis 1:
[ 2. 4. 5.]
```

numpy.average()

A média ponderada é uma média resultante da multiplicação de cada componente por um fator que reflete a sua importância. A função **numpy.average()** calcula a média ponderada

dos elementos em um array de acordo com seus respectivos pesos dados em outro array. A função pode ter um parâmetro de eixo. Se o eixo não for especificado, a matriz será nivelada.

Considerando uma matriz [1,2,3,4] e pesos correspondentes [4,3,2,1], a média ponderada é calculada somando o produto dos elementos correspondentes e dividindo a soma pela soma dos pesos.

Média ponderada = (1*4+2*3+3*2+4*1)/(4+3+2+1)

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([1,2,3,4])

print 'Our array is:'
print a
print '\n'

print 'Applying average() function:'
print np.average(a)
print '\n'

# this is same as mean when weight is not specified
wts = np.array([4,3,2,1])

print 'Applying average() function again:'
print np.average(a,weights = wts)
print '\n'

# Returns the sum of weights, if the returned parameter is set to True.
print 'Sum of weights'
print np.average([1,2,3, 4],weights = [4,3,2,1], returned = True)
```

Ele produzirá a seguinte saída -

```
Our array is:
[1 2 3 4]

Applying average() function:
2.5

Applying average() function again:
2.0
```

```
Sum of weights
(2.0, 10.0)
```

Em uma matriz multidimensional, o eixo para cálculo pode ser especificado.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.arange(6).reshape(3,2)

print 'Our array is:'
print a
print '\n'

print 'Modified array:'
wt = np.array([3,5])
print np.average(a, axis = 1, weights = wt)
print '\n'

print 'Modified array:'
print np.average(a, axis = 1, weights = wt, returned = True)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[0 1]
 [2 3]
 [4 5]]

Modified array:
[ 0.625 2.625 4.625]

Modified array:
(array([ 0.625, 2.625, 4.625]), array([ 8., 8., 8.]))
```

Desvio padrão

O desvio padrão é a raiz quadrada da média dos desvios quadrados da média. A fórmula para o desvio padrão é a seguinte -

```
std = sqrt(mean(abs(x - x.mean())**2))
```

Se a matriz for [1, 2, 3, 4], então sua média é 2,5. Portanto, os desvios quadrados são [2,25, 0,25, 0,25, 2,25] e a raiz quadrada de sua média dividida por 4, ou seja, sqrt (5/4) é 1,1180339887498949.

Exemplo

```
import numpy as np
print np.std([1,2,3,4])
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
1.1180339887498949
```

Variância

A variância é a média dos desvios quadrados, ou seja, **mean(abs(x - x.mean())**2)** . Em outras palavras, o desvio padrão é a raiz quadrada da variância.

Exemplo

```
import numpy as np
print np.var([1,2,3,4])
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
1.25
```

NumPy - Funções de classificação, pesquisa e contagem

Uma variedade de funções relacionadas à classificação estão disponíveis no NumPy. Essas funções de classificação implementam diferentes algoritmos de classificação, cada um deles caracterizado pela velocidade de execução, desempenho do pior caso, espaço de trabalho necessário e estabilidade dos algoritmos. A tabela a seguir mostra a comparação de três algoritmos de classificação.

tipo	velocidade	pior caso	área de trabalho	estábulo
'ordenação rápida'	1	O(n^2)	0	não
'mesclar classificação'	2	O(n*log(n))	~n/2	sim
'heapsort'	3	O(n*log(n))	0	não

numpy.sort()

A função sort() retorna uma cópia classificada do array de entrada. Possui os seguintes parâmetros -

```
numpy.sort(a, axis, kind, order)
```

Onde,

Sr. Não.	Parâmetro e Descrição
1	**a** Matriz a ser classificada
2	**eixo** O eixo ao longo do qual a matriz deve ser classificada. Se não houver, a matriz será nivelada, classificando no último eixo
3	**tipo** O padrão é classificação rápida
4	**ordem** Se a matriz contiver campos, a ordem dos campos a serem classificados

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[3,7],[9,1]])

print 'Our array is:'
print a
print '\n'

print 'Applying sort() function:'
```

```
print np.sort(a)
print '\n'

print 'Sort along axis 0:'
print np.sort(a, axis = 0)
print '\n'

# Order parameter in sort function
dt = np.dtype([('name', 'S10'),('age', int)])
a = np.array([("raju",21),("anil",25),("ravi", 17), ("amar",27)], dtype = dt)

print 'Our array is:'
print a
print '\n'

print 'Order by name:'
print np.sort(a, order = 'name')
```

Ele produzirá a seguinte saída -

```
Our array is:
[[3 7]
 [9 1]]

Applying sort() function:
[[3 7]
 [1 9]]

Sort along axis 0:
[[3 1]
 [9 7]]

Our array is:
[('raju', 21) ('anil', 25) ('ravi', 17) ('amar', 27)]

Order by name:
[('amar', 27) ('anil', 25) ('raju', 21) ('ravi', 17)]
```

numpy.argsort()

A função **numpy.argsort()** executa uma classificação indireta na matriz de entrada, ao longo de um determinado eixo e usando um tipo especificado de classificação para retornar a matriz de índices de dados. Esta matriz de índices é usada para construir a matriz classificada.

Exemplo

Demonstração ao vivo

```
import numpy as np
x = np.array([3, 1, 2])

print 'Our array is:'
print x
print '\n'

print 'Applying argsort() to x:'
y = np.argsort(x)
print y
print '\n'

print 'Reconstruct original array in sorted order:'
print x[y]
print '\n'

print 'Reconstruct the original array using loop:'
for i in y:
   print x[i],
```

Ele produzirá a seguinte saída -

```
Our array is:
[3 1 2]

Applying argsort() to x:
[1 2 0]

Reconstruct original array in sorted order:
[1 2 3]

Reconstruct the original array using loop:
1 2 3
```

numpy.lexsort()

A função executa uma classificação indireta usando uma sequência de chaves. As chaves podem ser vistas como uma coluna em uma planilha. A função retorna uma matriz de índices,

com os quais os dados classificados podem ser obtidos. Observe que a última chave é a chave primária do tipo.

Exemplo

Demonstração ao vivo

```
import numpy as np

nm = ('raju','anil','ravi','amar')
dv = ('f.y.', 's.y.', 's.y.', 'f.y.')
ind = np.lexsort((dv,nm))

print 'Applying lexsort() function:'
print ind
print '\n'

print 'Use this index to get sorted data:'
print [nm[i] + ", " + dv[i] for i in ind]
```

Ele produzirá a seguinte saída -

```
Applying lexsort() function:
[3 1 0 2]

Use this index to get sorted data:
['amar, f.y.', 'anil, s.y.', 'raju, f.y.', 'ravi, s.y.']
```

O módulo NumPy possui várias funções para pesquisar dentro de um array. Estão disponíveis funções para encontrar o máximo, o mínimo, bem como os elementos que satisfazem uma determinada condição.

numpy.argmax() e numpy.argmin()

Essas duas funções retornam os índices dos elementos máximo e mínimo, respectivamente, ao longo do eixo fornecido.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[30,40,70],[80,20,10],[50,90,60]])

print 'Our array is:'
```

```
print a
print '\n'

print 'Applying argmax() function:'
print np.argmax(a)
print '\n'

print 'Index of maximum number in flattened array'
print a.flatten()
print '\n'

print 'Array containing indices of maximum along axis 0:'
maxindex = np.argmax(a, axis = 0)
print maxindex
print '\n'

print 'Array containing indices of maximum along axis 1:'
maxindex = np.argmax(a, axis = 1)
print maxindex
print '\n'

print 'Applying argmin() function:'
minindex = np.argmin(a)
print minindex
print '\n'

print 'Flattened array:'
print a.flatten()[minindex]
print '\n'

print 'Flattened array along axis 0:'
minindex = np.argmin(a, axis = 0)
print minindex
print '\n'

print 'Flattened array along axis 1:'
minindex = np.argmin(a, axis = 1)
print minindex
```

Ele produzirá a seguinte saída -

```
Our array is:
[[30 40 70]
 [80 20 10]
```

```
 [50 90 60]]

Applying argmax() function:
7

Index of maximum number in flattened array
[30 40 70 80 20 10 50 90 60]

Array containing indices of maximum along axis 0:
[1 2 0]

Array containing indices of maximum along axis 1:
[2 0 1]

Applying argmin() function:
5

Flattened array:
10

Flattened array along axis 0:
[0 1 1]

Flattened array along axis 1:
[0 2 0]
```

numpy.diferente de zero()

A função **numpy.nonzero()** retorna os índices de elementos diferentes de zero na matriz de entrada.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[30,40,0],[0,20,10],[50,0,60]])

print 'Our array is:'
print a
print '\n'

print 'Applying nonzero() function:'
print np.nonzero (a)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[30 40 0]
 [ 0 20 10]
 [50 0 60]]

Applying nonzero() function:
(array([0, 0, 1, 1, 2, 2]), array([0, 1, 1, 2, 0, 2]))
```

numpy.where()

A função where() retorna os índices dos elementos em uma matriz de entrada onde a condição fornecida é satisfeita.

Exemplo

Demonstração ao vivo

```
import numpy as np
x = np.arange(9.).reshape(3, 3)

print 'Our array is:'
print x

print 'Indices of elements > 3'
y = np.where(x > 3)
print y

print 'Use these indices to get elements satisfying the condition'
print x[y]
```

Ele produzirá a seguinte saída -

```
Our array is:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]

Indices of elements > 3
(array([1, 1, 2, 2, 2]), array([1, 2, 0, 1, 2]))
```

```
Use these indices to get elements satisfying the condition
[ 4. 5. 6. 7. 8.]
```

numpy.extract()

A função **extract()** retorna os elementos que satisfazem qualquer condição.

Demonstração ao vivo

```
import numpy as np
x = np.arange(9.).reshape(3, 3)

print 'Our array is:'
print x

# define a condition
condition = np.mod(x,2) == 0

print 'Element-wise value of condition'
print condition

print 'Extract elements using condition'
print np.extract(condition, x)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]

Element-wise value of condition
[[ True False True]
 [False True False]
 [ True False True]]

Extract elements using condition
[ 0. 2. 4. 6. 8.]
```

NumPy - Troca de Bytes

Vimos que os dados armazenados na memória de um computador dependem da arquitetura que a CPU utiliza. Pode ser little-endian (o byte mais significativo é armazenado no menor

endereço) ou big-endian (o byte mais significativo no menor endereço).

numpy.ndarray.byteswap()

A função **numpy.ndarray.byteswap()** alterna entre as duas representações: bigendian e little-endian.

Demonstração ao vivo

```
import numpy as np
a = np.array([1, 256, 8755], dtype = np.int16)

print 'Our array is:'
print a

print 'Representation of data in memory in hexadecimal form:'
print map(hex,a)
# byteswap() function swaps in place by passing True parameter

print 'Applying byteswap() function:'
print a.byteswap(True)

print 'In hexadecimal form:'
print map(hex,a)
# We can see the bytes being swapped
```

Ele produzirá a seguinte saída -

```
Our array is:
[1 256 8755]

Representation of data in memory in hexadecimal form:
['0x1', '0x100', '0x2233']

Applying byteswap() function:
[256 1 13090]

In hexadecimal form:
['0x100', '0x1', '0x3322']
```

NumPy - Cópias e Visualizações

Ao executar as funções, algumas delas retornam uma cópia do array de entrada, enquanto outras retornam a visualização. Quando o conteúdo é armazenado fisicamente em outro local,

chama-se **Copiar** . Se, por outro lado, for fornecida uma visão diferente do mesmo conteúdo da memória, chamamos-lhe **View** .

Sem cópia

Atribuições simples não fazem a cópia do objeto array. Em vez disso, ele usa o mesmo id() do array original para acessá-lo. O **id()** retorna um identificador universal do objeto Python, semelhante ao ponteiro em C.

Além disso, quaisquer alterações em um deles são refletidas no outro. Por exemplo, a mudança na forma de um também mudará a forma do outro.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.arange(6)

print 'Our array is:'
print a

print 'Applying id() function:'
print id(a)

print 'a is assigned to b:'
b = a
print b

print 'b has same id():'
print id(b)

print 'Change shape of b:'
b.shape = 3,2
print b

print 'Shape of a also gets changed:'
print a
```

Ele produzirá a seguinte saída -

```
Our array is:
[0 1 2 3 4 5]

Applying id() function:
```

```
139747815479536

a is assigned to b:
[0 1 2 3 4 5]
b has same id():
139747815479536

Change shape of b:
[[0 1]
 [2 3]
 [4 5]]

Shape of a also gets changed:
[[0 1]
 [2 3]
 [4 5]]
```

Visualizar ou cópia superficial

NumPy possui o método **ndarray.view()** que é um novo objeto de array que analisa os mesmos dados do array original. Ao contrário do caso anterior, a alteração nas dimensões do novo array não altera as dimensões do original.

Exemplo

Demonstração ao vivo

```
import numpy as np
# To begin with, a is 3X2 array
a = np.arange(6).reshape(3,2)

print 'Array a:'
print a

print 'Create view of a:'
b = a.view()
print b

print 'id() for both the arrays are different:'
print 'id() of a:'
print id(a)
print 'id() of b:'
print id(b)
```

```
# Change the shape of b. It does not change the shape of a
b.shape = 2,3

print 'Shape of b:'
print b

print 'Shape of a:'
print a
```

Ele produzirá a seguinte saída -

```
Array a:
[[0 1]
 [2 3]
 [4 5]]

Create view of a:
[[0 1]
 [2 3]
 [4 5]]

id() for both the arrays are different:
id() of a:
140424307227264
id() of b:
140424151696288

Shape of b:
[[0 1 2]
 [3 4 5]]

Shape of a:
[[0 1]
 [2 3]
 [4 5]]
```

Uma fatia de um array cria uma visualização.

Exemplo

```
import numpy as np
a = np.array([[10,10], [2,3], [4,5]])
```

Demonstração ao vivo

```
print 'Our array is:'
print a

print 'Create a slice:'
s = a[:, :2]
print s
```

Ele produzirá a seguinte saída -

```
Our array is:
[[10 10]
 [ 2 3]
 [ 4 5]]

Create a slice:
[[10 10]
 [ 2 3]
 [ 4 5]]
```

Cópia profunda

A função **ndarray.copy()** cria uma cópia profunda. É uma cópia completa do array e de seus dados e não é compartilhada com o array original.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[10,10], [2,3], [4,5]])

print 'Array a is:'
print a

print 'Create a deep copy of a:'
b = a.copy()
print 'Array b is:'
print b

#b does not share any memory of a
print 'Can we write b is a'
print b is a
```

```
print 'Change the contents of b:'
b[0,0] = 100

print 'Modified array b:'
print b

print 'a remains unchanged:'
print a
```

Ele produzirá a seguinte saída -

```
Array a is:
[[10 10]
 [ 2 3]
 [ 4 5]]

Create a deep copy of a:
Array b is:
[[10 10]
 [ 2 3]
 [ 4 5]]
Can we write b is a
False

Change the contents of b:
Modified array b:
[[100 10]
 [ 2 3]
 [ 4 5]]

a remains unchanged:
[[10 10]
 [ 2 3]
 [ 4 5]]
```

NumPy - Biblioteca de Matrizes

O pacote NumPy contém uma biblioteca Matrix **numpy.matlib** . Este módulo possui funções que retornam matrizes em vez de objetos ndarray.

matlib.empty()

A função **matlib.empty()** retorna uma nova matriz sem inicializar as entradas. A função usa os seguintes parâmetros.

```
numpy.matlib.empty(shape, dtype, order)
```

Onde,

Sr. Não.	Parâmetro e Descrição
1	**forma** **int** ou tupla de **int** definindo a forma da nova matriz
2	**Tipo D** Opcional. Tipo de dados da saída
3	**ordem** C ou F

Exemplo

```
import numpy.matlib
import numpy as np

print np.matlib.empty((2,2))
# filled with random data
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
[[ 2.12199579e-314,  4.24399158e-314]
 [ 4.24399158e-314,  2.12199579e-314]]
```

numpy.matlib.zeros()

Esta função retorna a matriz preenchida com zeros.

```
import numpy.matlib
import numpy as np
print np.matlib.zeros((2,2))
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
[[ 0.  0.]
 [ 0.  0.]]
```

numpy.matlib.ones()

Esta função retorna a matriz preenchida com 1s.

```
import numpy.matlib
import numpy as np
print np.matlib.ones((2,2))
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
[[ 1.  1.]
 [ 1.  1.]]
```

numpy.matlib.eye()

Esta função retorna uma matriz com 1 ao longo dos elementos diagonais e zeros em outros lugares. A função usa os seguintes parâmetros.

```
numpy.matlib.eye(n, M,k, dtype)
```

Onde,

Sr. Não.	Parâmetro e Descrição
1	**n** O número de linhas na matriz resultante
2	**M** O número de colunas, o padrão é n
3	**k** Índice de diagonal
4	**tipo d** Tipo de dados da saída

Exemplo

```
import numpy.matlib
import numpy as np
print np.matlib.eye(n = 3, M = 4, k = 0, dtype = float)
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]]
```

numpy.matlib.identity()

A função **numpy.matlib.identity()** retorna a matriz de identidade do tamanho fornecido. Uma matriz identidade é uma matriz quadrada com todos os elementos diagonais como 1.

```
import numpy.matlib
import numpy as np
print np.matlib.identity(5, dtype = float)
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
```

numpy.matlib.rand()

A função **numpy.matlib.rand()** retorna uma matriz de um determinado tamanho preenchida com valores aleatórios.

Exemplo

Demonstração ao vivo

```
import numpy.matlib
import numpy as np
print np.matlib.rand(3,3)
```

Ele produzirá a seguinte saída -

```
[[ 0.82674464  0.57206837  0.15497519]
 [ 0.33857374  0.35742401  0.90895076]
 [ 0.03968467  0.13962089  0.39665201]]
```

Observe que uma matriz é sempre bidimensional, enquanto ndarray é uma matriz n-dimensional. Ambos os objetos são interconversíveis.

Exemplo

Demonstração ao vivo

```
import numpy.matlib
import numpy as np

i = np.matrix('1,2;3,4')
print i
```

Ele produzirá a seguinte saída -

```
[[1  2]
 [3  4]]
```

Exemplo

```
import numpy.matlib
import numpy as np

j = np.asarray(i)
print j
```

Ele produzirá a seguinte saída -

```
[[1  2]
 [3  4]]
```

Exemplo

```
import numpy.matlib
import numpy as np

k = np.asmatrix (j)
print k
```

Ele produzirá a seguinte saída -

```
[[1  2]
 [3  4]]
```

NumPy - Álgebra Linear

O pacote NumPy contém o módulo **numpy.linalg** que fornece todas as funcionalidades necessárias para álgebra linear. Algumas das funções importantes deste módulo estão descritas na tabela a seguir.

Sr. Não.	Descrição da função
1	**ponto** Produto escalar das duas matrizes
2	**ponto** Produto escalar dos dois vetores
3	**interno** Produto interno das duas matrizes
4	**matmul** Produto matricial das duas matrizes
5	**determinante** Calcula o determinante da matriz
6	**resolver** Resolve a equação da matriz linear
7	**inv.** Encontra o inverso multiplicativo da matriz

NumPy - Matplotlib

Matplotlib é uma biblioteca de plotagem para Python. Ele é usado junto com o NumPy para fornecer um ambiente que seja uma alternativa eficaz de código aberto para MatLab. Também pode ser usado com kits de ferramentas gráficas como PyQt e wxPython.

O módulo Matplotlib foi escrito pela primeira vez por John D. Hunter. Desde 2012, Michael Droettboom é o principal desenvolvedor. Atualmente, Matplotlib versão. 1.5.1 é a versão estável disponível. O pacote está disponível em distribuição binária, bem como na forma de código-fonte em www.matplotlib.org .

Convencionalmente, o pacote é importado para o script Python adicionando a seguinte instrução -

```
from matplotlib import pyplot as plt
```

Aqui, **pyplot()** é a função mais importante na biblioteca matplotlib, que é usada para plotar dados 2D. O script a seguir traça a equação **y = 2x + 5**

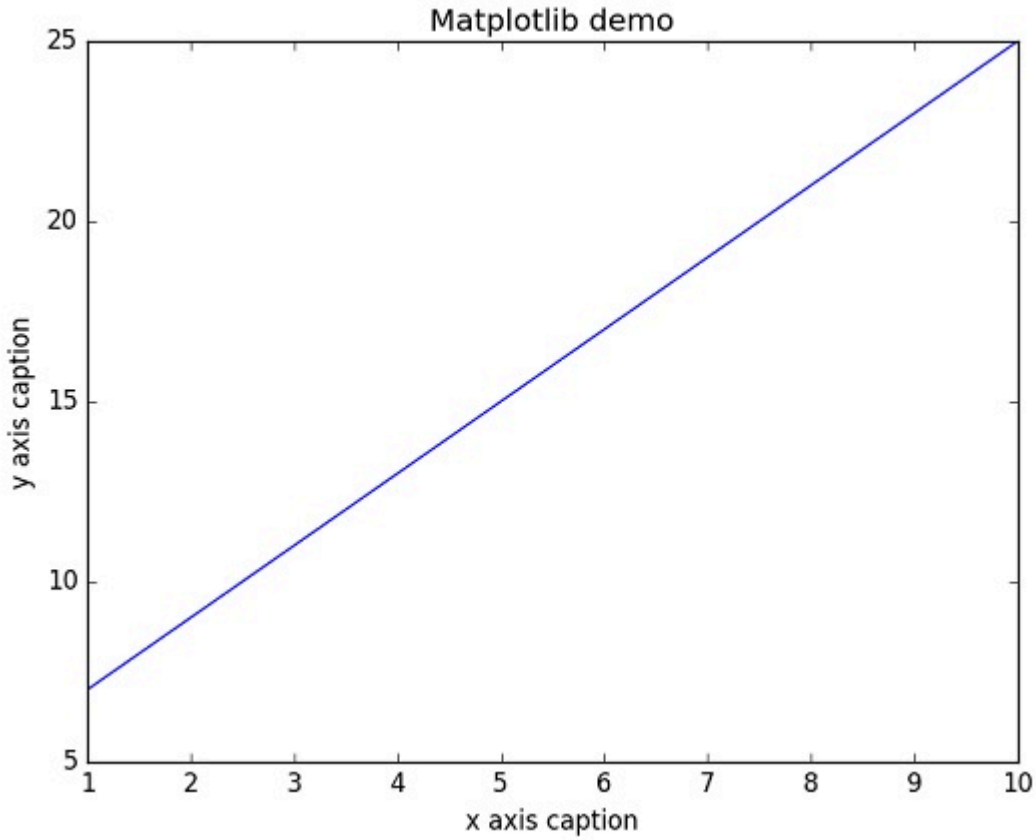
Exemplo

```
import numpy as np
from matplotlib import pyplot as plt

x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y)
plt.show()
```

Um objeto ndarray x é criado a partir **da função np.arange()** como os valores no **eixo x** . Os valores correspondentes no **eixo y** são armazenados em outro **objeto ndarray y** . Esses valores são plotados usando a função **plot()** do submódulo pyplot do pacote matplotlib.

A representação gráfica é exibida pela função **show()** .

O código acima deve produzir a seguinte saída -



Em vez do gráfico linear, os valores podem ser exibidos discretamente adicionando uma string de formato à função **plot()** . Os seguintes caracteres de formatação podem ser usados.

Sr. Não.	Descrição do personagem
1	**'-'** Estilo de linha sólida
2	**'--'** Estilo de linha tracejada
3	**'-.'** Estilo de linha traço-ponto
4	**':'** Estilo de linha pontilhada
5	**'.'** Marcador de ponto
6	**','** Marcador de pixels
7	**'o'** Marcador de círculo

8	**'v'** Marcador triângulo_para baixo
9	**'^'** Marcador Triangle_up
10	**'<'** Marcador triângulo_esquerdo
11	**'>'** Marcador triângulo_direito
12	**'1'** Marcador tri_down
13	**'2'** Marcador Tri_up
14	**'3'** Marcador tri_esquerdo
15	**'4'** Marcador tri_right
16	**'é'** Marcador quadrado
17	**'p'** Marcador do Pentágono
18	**'*'** Marcador de estrela
19	**'h'** Marcador Hexágono1
20	**'H'** Marcador Hexágono2
21	**'+'** Mais marcador
22	**'x'** Marcador X
23	**'D'** Marcador de diamante
24	**'d'**

	Marcador Thin_diamond
25	'|' Marcador de linha V
26	'_' Marcador de linha H

As seguintes abreviações de cores também são definidas.

Personagem	Cor
'b'	Azul
'g'	Verde
'r'	Vermelho
'c'	Ciano
'eu'	Magenta
'você'	Amarelo
'k'	Preto
'c'	Branco

Para exibir os círculos que representam pontos, em vez da linha no exemplo acima, use **"ob"** como string de formato na função plot().

Exemplo

```
import numpy as np
from matplotlib import pyplot as plt

x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y,"ob")
plt.show()
```

O código acima deve produzir a seguinte saída -

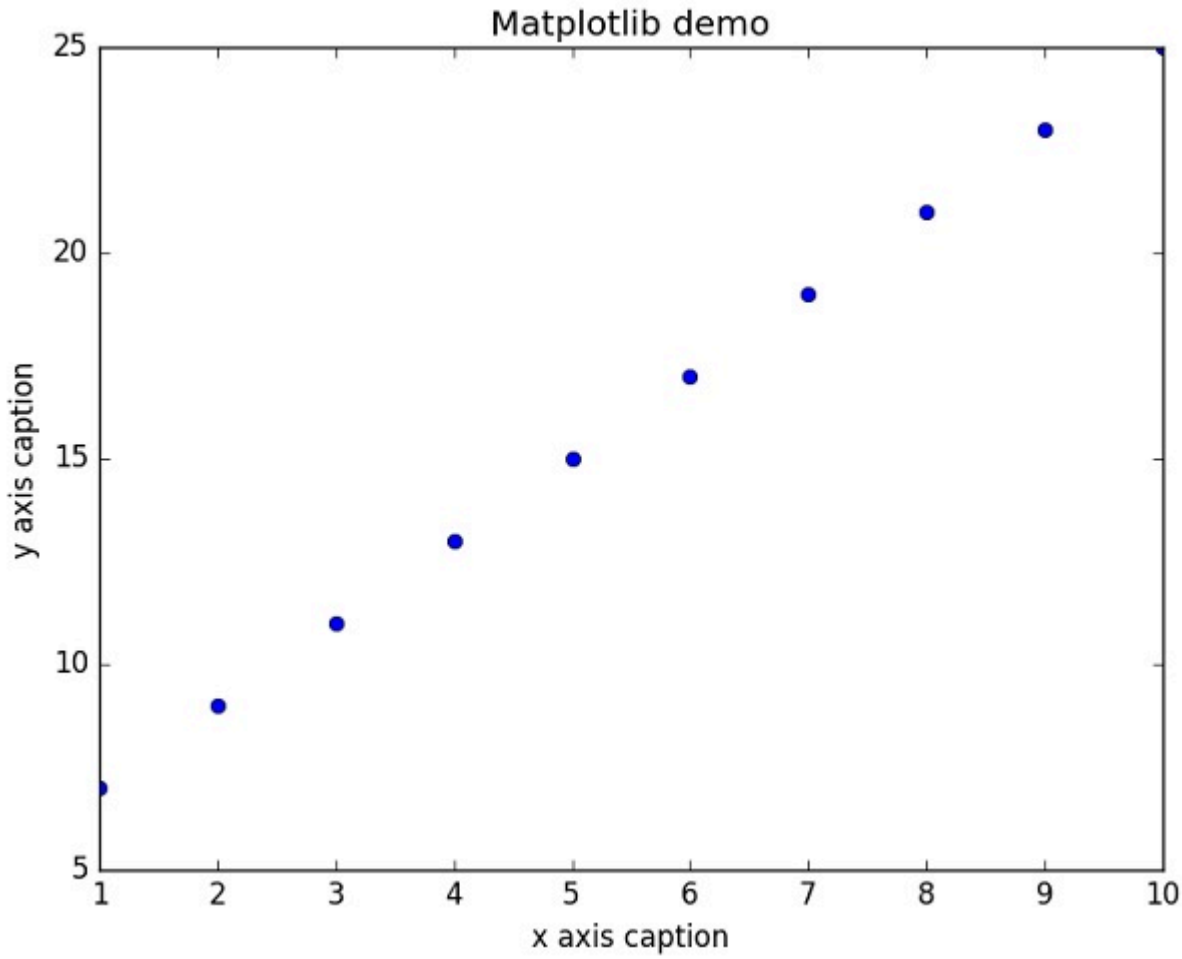
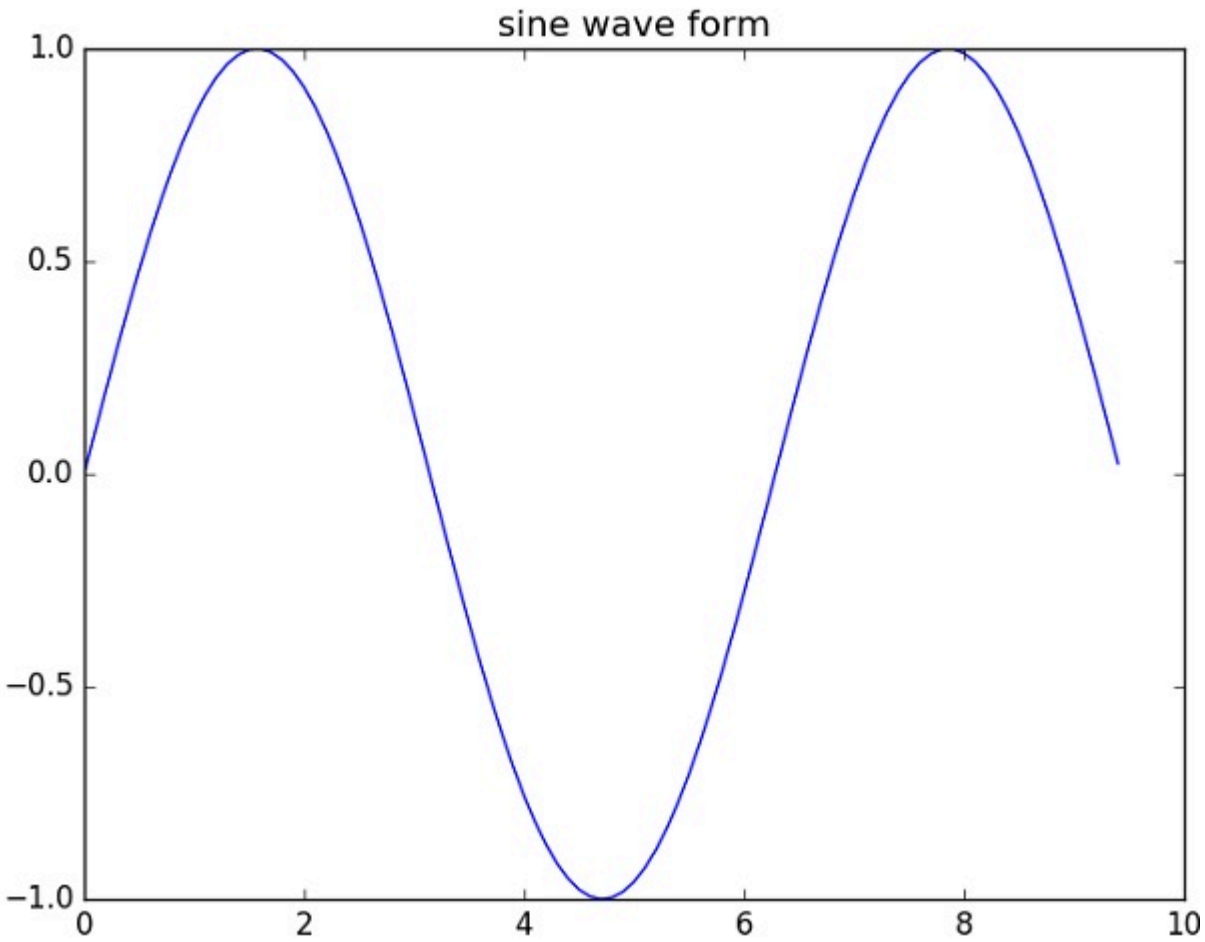


Gráfico de onda senoidal

O script a seguir produz o **gráfico de onda senoidal** usando matplotlib.

Exemplo

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.title("sine wave form")

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```



subtrama()

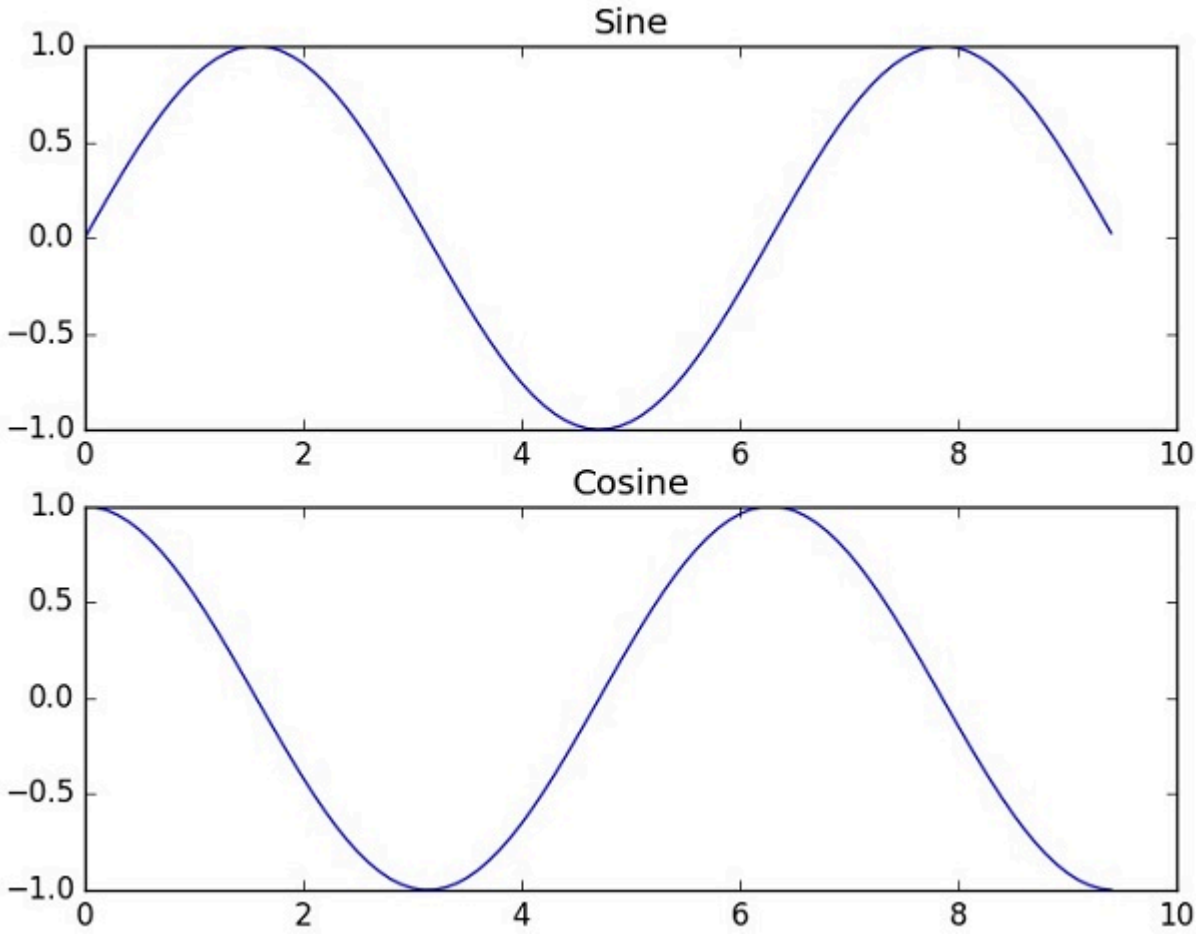
A função subplot() permite plotar coisas diferentes na mesma figura. No script a seguir, os valores de **seno** e **cosseno** são plotados.

Exemplo

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')
```

```
# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

O código acima deve produzir a seguinte saída -



bar()

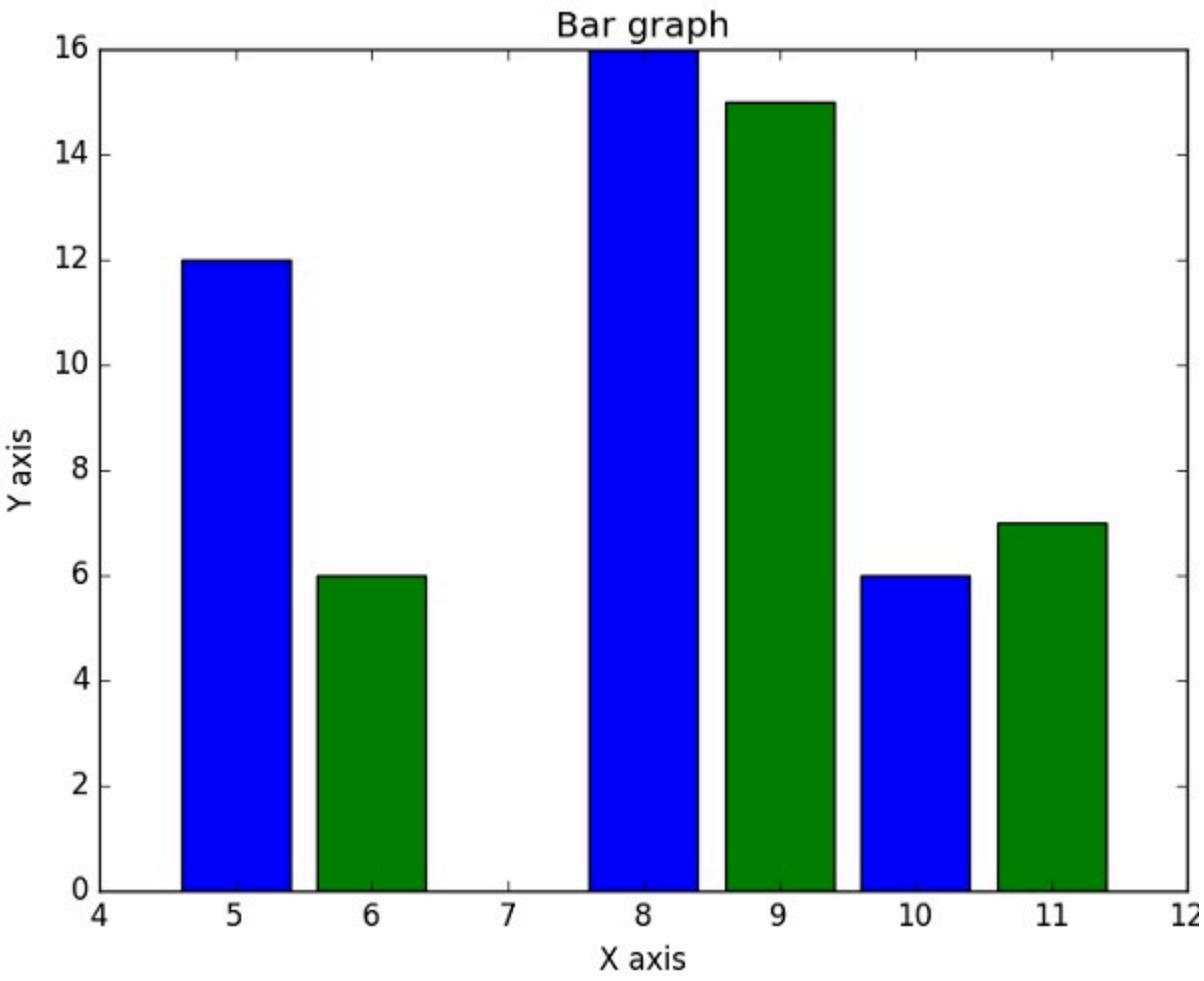
O **submódulo pyplot** fornece a função **bar()** para gerar gráficos de barras. O exemplo a seguir produz o gráfico de barras de dois conjuntos **de** matrizes **xey** .

Exemplo

```
from matplotlib import pyplot as plt
x = [5,8,10]
y = [12,16,6]

x2 = [6,9,11]
y2 = [6,15,7]
```

```
plt.bar(x, y, align = 'center')
plt.bar(x2, y2, color = 'g', align = 'center')
plt.title('Bar graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')

plt.show()
```

Este código deve produzir a seguinte saída -



NumPy - Histograma usando Matplotlib

NumPy possui uma função **numpy.histogram()** que é uma representação gráfica da distribuição de frequência dos dados. Retângulos de tamanho horizontal igual correspondente ao intervalo de classe denominado **bin** e **altura variável** correspondente à frequência.

numpy.histograma()

A função numpy.histogram() considera a matriz de entrada e os compartimentos como dois parâmetros. Os elementos sucessivos na matriz bin atuam como o limite de cada compartimento.

```
import numpy as np

a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
np.histogram(a,bins = [0,20,40,60,80,100])
hist,bins = np.histogram(a,bins = [0,20,40,60,80,100])
print hist
print bins
```

Ele produzirá a seguinte saída -

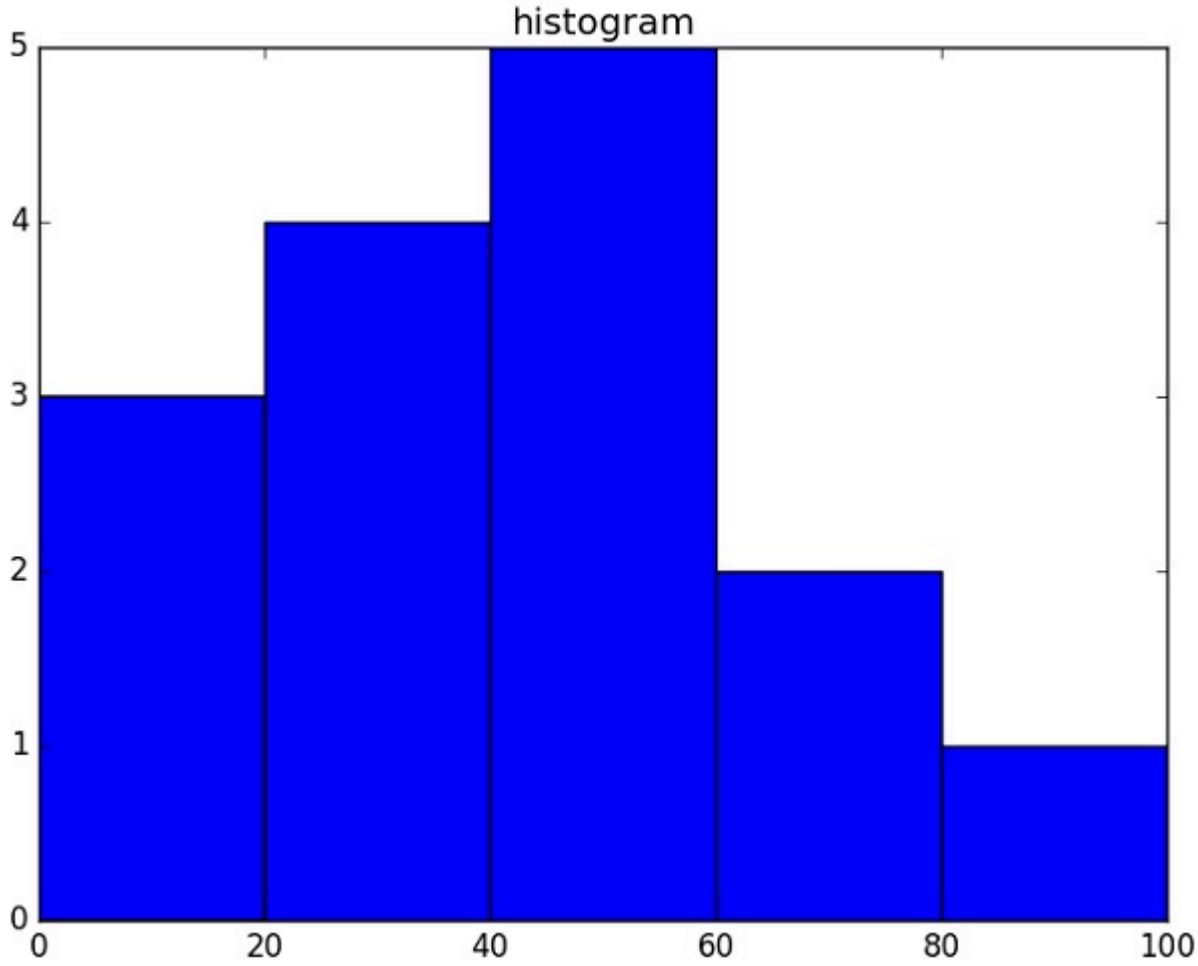
```
[3 4 5 2 1]
[0 20 40 60 80 100]
```

plt()

Matplotlib pode converter esta representação numérica do histograma em um gráfico. A **função plt()** do submódulo pyplot pega o array contendo os dados e o array bin como parâmetros e converte em um histograma.

```
from matplotlib import pyplot as plt
import numpy as np

a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,20,40,60,80,100])
plt.title("histogram")
plt.show()
```

Deve produzir a seguinte saída -



E/S com NumPy

Os objetos ndarray podem ser salvos e carregados nos arquivos do disco. As funções IO disponíveis são -

- **As funções load()** e **save()** lidam com arquivos binários /numPy (com extensão **npy)**
- **As funções loadtxt()** e **savetxt()** lidam com arquivos de texto normais

NumPy apresenta um formato de arquivo simples para objetos ndarray. Este arquivo **.npy** armazena dados, forma, tipo e outras informações necessárias para reconstruir o ndarray em um arquivo de disco, de modo que o array seja recuperado corretamente, mesmo se o arquivo estiver em outra máquina com arquitetura diferente.

numpy.save()

O arquivo **numpy.save()** armazena a matriz de entrada em um arquivo de disco com extensão **npy** .

```
import numpy as np
a = np.array([1,2,3,4,5])
```

```
np.save('outfile',a)
```

Para reconstruir o array de **outfile.npy** , use a função **load()** .

```
import numpy as np
b = np.load('outfile.npy')
print b
```

Ele produzirá a seguinte saída -

```
array([1, 2, 3, 4, 5])
```

As funções save() e load() aceitam um parâmetro booleano adicional **Allow_pickles** . Um pickle em Python é usado para serializar e desserializar objetos antes de salvar ou ler um arquivo em disco.

salvartxt()

O armazenamento e recuperação de dados de array em formato de arquivo de texto simples é feito com as funções **savetxt()** e **loadtxt()** .

Exemplo

```
import numpy as np

a = np.array([1,2,3,4,5])
np.savetxt('out.txt',a)
b = np.loadtxt('out.txt')
print b
```

Ele produzirá a seguinte saída -

```
[ 1.  2.  3.  4.  5.]
```

As funções savetxt() e loadtxt() aceitam parâmetros opcionais adicionais, como cabeçalho, rodapé e delimitador.