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

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

Demonstração ao vivo

Sua saída seria -

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

Exemplo 2

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

Demonstração ao vivo

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

Demonstração ao vivo

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

Agora, a saída seria -

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