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.]
```