# Numpy | Mathematical Function

NumPy contains a large number of various mathematical operations. NumPy provides standard trigonometric functions, functions for arithmetic operations, handling complex numbers, etc.

**Trigonometric Functions –**

NumPy has standard trigonometric functions which return trigonometric ratios for a given angle in radians.

**numpy.sin(x[, out]) = ufunc 'sin') :** This mathematical function helps user to calculate trignmetric sine for all x(being the array elements).

```
# Python program explaining
# sin() function

import numpy as np
import math

in_array = [0, math.pi / 2, np.pi / 3, np.pi]
print ("Input array : \n", in_array)

Sin_Values = np.sin(in_array)
print ("\nSine values : \n", Sin_Values)
```

Run on IDE

**Output :**

```
Input array :
 [0, 1.5707963267948966, 1.0471975511965976, 3.141592653589793]

Sine values :
 [  0.00000000e+00   1.00000000e+00   8.66025404e-01   1.22464680e-16]
```

**numpy.cos(x[, out]) = ufunc 'cos') :** This mathematical function helps user to calculate trignmetric cosine for all x(being the array elements).

```
# Python program explaining
# cos() function

import numpy as np
```

```
cos_Values = np.cos(in_array)
print ("\nCosine values : \n", cos_Values)
```

Run on IDE

**Output :**

```
Input array :
 [0, 1.5707963267948966, 1.0471975511965976, 3.141592653589793]

Cosine values :
 [  1.00000000e+00   6.12323400e-17   5.00000000e-01  -1.00000000e+00]
```

| FUNCTION | DESCRIPTION |
|---|---|
| **tan()** | Compute tangent element-wise. |
| **arcsin()** | Inverse sine, element-wise. |
| **arccos()** | Trigonometric inverse cosine, element-wise. |
| **arctan()** | Trigonometric inverse tangent, element-wise. |

| | |
|---|---|
| **rad2deg()** | Convert angles from radians to degrees. |
| **deg2rad** | Convert angles from degrees to radians. |
| **radians()** | Convert angles from degrees to radians. |
| **hypot()** | Given the “legs” of a right triangle, return its hypotenuse. |
| **unwrap()** | Unwrap by changing deltas between values to 2*pi complement. |

**Hyperbolic Functions –**

**numpy.sinh(x[, out]) = ufunc ‘sin’) :** This mathematical function helps user to calculate hyperbolic sine for all x(being the array elements).

Equivalent to `1/2 * (np.exp(x) - np.exp(-x)) or -1j * np.sin(1j*x)`.

```
# Python3 program explaining
# sinh() function

import numpy as np
import math

in_array = [0, math.pi / 2, np.pi / 3, np.pi]
print ("Input array : \n", in_array)

Sinh_Values = np.sinh(in_array)
print ("\nSine Hyperbolic values : \n", Sinh_Values)
```

Run on IDE

**Output :**

```
Input array :
 [0, 1.5707963267948966, 1.0471975511965976, 3.141592653589793]

Sine Hyperbolic values :
 [  0.          2.3012989    1.24936705  11.54873936]
```

**numpy.cosh(x[, out]) = ufunc ‘cos’) :** This mathematical function helps user to calculate hyperbolic

```
# Python3 program explaining
# cosh() function

import numpy as np
import math

in_array = [0, math.pi / 2, np.pi / 3, np.pi]
print ("Input array : \n", in_array)

cosh_Values = np.cosh(in_array)
print ("\ncosine Hyperbolic values : \n", cosh_Values)
```

Run on IDE

**Output :**

```
Input array :
 [0, 1.5707963267948966, 1.0471975511965976, 3.141592653589793]

cosine Hyperbolic values :
 [  1.          2.50917848   1.60028686  11.59195328]
```

| FUNCTION | DESCRIPTION |
|---|---|
| **tanh()** | Compute hyperbolic tangent element-wise. |
| **arcsinh()** | Inverse hyperbolic sine element-wise. |
| **arccosh()** | Inverse hyperbolic cosine, element-wise. |
| **arctanh()** | Inverse hyperbolic tangent element-wise. |

**Functions for Rounding –**

**numpy.around(arr, decimals = 0, out = None) :** This mathematical function helps user to evenly round array elements to the given number of decimals.

```
# Python program explaining
# around() function

import numpy as np
```

```
round_off_values = np.around(in_array)
print ("\nRounded values : \n", round_off_values)


in_array = [.53, 1.54, .71]
print ("\nInput array : \n", in_array)

round_off_values = np.around(in_array)
print ("\nRounded values : \n", round_off_values)

in_array = [.5538, 1.33354, .71445]
print ("\nInput array : \n", in_array)

round_off_values = np.around(in_array, decimals = 3)
print ("\nRounded values : \n", round_off_values)
```

Run on IDE

**Output :**

```
Input array :
 [0.5, 1.5, 2.5, 3.5, 4.5, 10.1]

Rounded values :
 [  0.   2.   2.   4.   4.  10.]

Input array :
 [0.53, 1.54, 0.71]

Rounded values :
 [ 1.  2.  1.]

Input array :
 [0.5538, 1.33354, 0.71445]

Rounded values :
 [ 0.554  1.334  0.714]
```

**numpy.round_(arr, decimals = 0, out = None) :** This mathematical function round an array to the given number of decimals.

```
# Python program explaining
# round_() function
import numpy as np

in_array = [.5, 1.5, 2.5, 3.5, 4.5, 10.1]
print ("Input array : \n", in_array)

round_off_values = np.round_(in_array)
print ("\nRounded values : \n", round_off_values)


in_array = [.53, 1.54, .71]
print ("\nInput array : \n", in_array)

round_off_values = np.round_(in_array)
print ("\nRounded values : \n", round_off_values)

in_array = [.5538, 1.33354, .71445]
print ("\nInput array : \n", in_array)

round_off_values = np.round_(in_array, decimals = 3)
print ("\nRounded values : \n", round_off_values)
```

Run on IDE

**Output :**

```
Input array :
 [0.5, 1.5, 2.5, 3.5, 4.5, 10.1]

Rounded values :
 [  0.   2.   2.   4.   4.  10.]

Input array :
 [0.53, 1.54, 0.71]
```

```
Input array :
 [0.5538, 1.33354, 0.71445]

Rounded values :
 [ 0.554  1.334  0.714]
```

| FUNCTION | DESCRIPTION |
|---|---|
| **rint()** | Round to nearest integer towards zero. |
| **fix()** | Round to nearest integer towards zero. |
| **floor()** | Return the floor of the input, element-wise. |
| **ceil()** | Return the ceiling of the input, element-wise. |
| **trunc()** | Return the truncated value of the input, element-wise. |

**Exponents and logarithms Functions –**

**numpy.exp(array, out = None, where = True, casting = 'same_kind', order = 'K', dtype = None) :** This mathematical function helps user to calculate exponential of all the elements in the input array.

```
# Python program explaining
# exp() function
import numpy as np

in_array = [1, 3, 5]
print ("Input array : ", in_array)

out_array = np.exp(in_array)
print ("Output array : ", out_array)
```

Run on IDE

**Output :**

```
Input array :  [1, 3, 5]
Output array :  [   2.71828183   20.08553692  148.4131591 ]
```

**logarithm of x** where x belongs to all the input array elements.

Natural logarithm log is the **inverse of the exp()**, so that `log(exp(x)) = x`. The natural logarithm is log in base e.

```
# Python program explaining
# log() function
import numpy as np

in_array = [1, 3, 5, 2**8]
print ("Input array : ", in_array)

out_array = np.log(in_array)
print ("Output array : ", out_array)


print("\nnp.log(4**4) : ", np.log(4**4))
print("np.log(2**8) : ", np.log(2**8))
```

Run on IDE

**Output :**

```
Input array :  [1, 3, 5, 256]
Output array :  [ 0.          1.09861229  1.60943791  5.54517744]

np.log(4**4) :  5.54517744448
np.log(2**8) :  5.54517744448
```

| FUNCTION | DESCRIPTION |
|---|---|
| **expm1()** | Calculate exp(x) – 1 for all elements in the array. |
| **exp2()** | Calculate 2**p for all p in the input array. |
| **log10()** | Return the base 10 logarithm of the input array, element-wise. |
| **log2()** | Base-2 logarithm of x. |
| **log1p()** | Return the natural logarithm of one plus the input array, element-wise. |
| | |

**Arithmetic Functions –**

**numpy.reciprocal(x, /, out=None, *, where=True) :** This mathematical function is used to calculate reciprocal of all the elements in the input array.

**Note:** For integer arguments with absolute value larger than 1, the result is always zero because of the way Python handles integer division. For integer zero the result is an overflow.

```
# Python3 code demonstrate reciprocal() function

# importing numpy
import numpy as np

in_num = 2.0
print ("Input  number : ", in_num)

out_num = np.reciprocal(in_num)
print ("Output number : ", out_num)
```

Run on IDE

**Output :**

```
Input  number :  2.0
Output number :  0.5
```

. Array element from first array is divided by elements from second element (all happens element wise). Both arr1 and arr2 must have same shape and element in arr2 must not be zero; otherwise it will raise an error.

```
# Python program explaining
# divide() function
import numpy as np

# input_array
arr1 = [2, 27, 2, 21, 23]
arr2 = [2, 3, 4, 5, 6]
print ("arr1          : ", arr1)
print ("arr2          : ", arr2)

# output_array
out = np.divide(arr1, arr2)
print ("\nOutput array : \n", out)
```

Run on IDE

**Output :**

```
arr1          :  [2, 27, 2, 21, 23]
arr2          :  [2, 3, 4, 5, 6]

Output array :
 [ 1.          9.          0.5         4.2         3.83333333]
```

| FUNCTION | DESCRIPTION |
|---|---|
| **add()** | Add arguments element-wise. |
| **positive()** | Numerical positive, element-wise. |
| **negative()** | Numerical negative, element-wise. |
| **multiply()** | Multiply arguments element-wise. |
| **power()** | First array elements raised to powers from second array, element-wise. |
| **subtract()** | Subtract arguments, element-wise. |

| | |
|---|---|
| floor_divide() | Return the largest integer smaller or equal to the division of the inputs. |
| float_power() | First array elements raised to powers from second array, element-wise. |
| mod() | Return the element-wise remainder of division. |
| remainder() | Return element-wise remainder of division. |
| divmod() | Return element-wise quotient and remainder simultaneously. |

**Complex number Function –**

**numpy.isreal(array) :** Test element-wise whether it is a real number or not(not infinity or not Not a Number) and return the result as a boolean array.

```
# Python Program illustrating
# numpy.isreal() method

import numpy as geek

print("Is Real : ", geek.isreal([1+1j, 0j]), "\n")

print("Is Real : ", geek.isreal([1, 0]), "\n")
```

Run on IDE

**Output :**

```
 Is Real :  [False  True]

 Is Real :  [ True  True]
```

**numpy.conj(x[, out] = ufunc 'conjugate') :** This function helps the user to conjugate any complex number.

The conjugate of a complex number is obtained by changing the sign of its imaginary part. If the complex number is 2+5j then its conjugate is 2-5j.

```
# Python3 code demonstrate conj() function

#importing numpy
import numpy as np

in_complx1 = 2+4j
```

```
out_complx2 = np.conj(in_complx2)
print ("Output conjugated complex number of 5-8j: ", out_complx2)
```

Run on IDE

**Output :**

```
Output conjugated complex number of  2+4j :  (2-4j)
Output conjugated complex number of 5-8j:  (5+8j)
```

**Special functions –**

**numpy.cbrt(arr, out = None, ufunc 'cbrt') :** This mathematical function helps user to calculate cube root of x for all x being the array elements.

```
# Python program explaining
# cbrt () function

import numpy as np

arr1 = [1, 27000, 64, -1000]
print ("cbrt Value of arr1 : \n", np.cbrt(arr1))

arr2 = [1024 ,-128]
print ("\ncbrt Value of arr2 : ", np.cbrt(arr2))
```

Run on IDE

**Output :**

```
cbrt Value of arr1 :
 [  1.  30.   4. -10.]

cbrt Value of arr2 :  [ 10.0793684  -5.0396842]
```

**numpy.clip() :** This function is used to Clip (limit) the values in an array.
Given an interval, values outside the interval are clipped to the interval edges. For example, if an interval of [0, 1] is specified, values smaller than 0 become 0, and values larger than 1 become 1.

```
# Python3 code demonstrate clip() function

# importing the numpy
```

```
out_array = np.clip(in_array, a_min = 2, a_max = 6)
print ("Output array : ", out_array)
```

Run on IDE

**Output :**

```
Input array :  [1, 2, 3, 4, 5, 6, 7, 8]
Output array :  [2 2 3 4 5 6 6 6]
```

| FUNCTION | DESCRIPTION |
|---|---|
| **convolve()** | Returns the discrete, linear convolution of two one-dimensional sequences. |
| **sqrt()** | Return the non-negative square-root of an array, element-wise. |
| **square()** | Return the element-wise square of the input. |
| **absolute()** | Calculate the absolute value element-wise. |
| **fabs()** | Compute the absolute values element-wise. |
| **sign()** | Returns an element-wise indication of the sign of a number. |
| **interp()** | One-dimensional linear interpolation. |
| **maximum()** | Element-wise maximum of array elements. |
| **minimum()** | Element-wise minimum of array elements. |
| **real_if_close()** | If complex input returns a real array if complex parts are close to zero. |
| **nan_to_num()** | Replace NaN with zero and infinity with large finite numbers. |
| **heaviside()** | Compute the Heaviside step function. |

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.