ArrayFunc

Authors: Michael Griffin
Version: 8.1.1 for 2022-03-31

Language: Python 3.6 or later

Table of Contents

Introduction

The ArrayFunc module provides high speed array processing functions for use with the standard Python array module. These functions are patterned after the functions in the standard Python Itertools module together with some additional ones from other sources.

The purpose of these functions is to perform mathematical calculations on arrays significantly faster than using native Python.

Function Summary

The functions fall into several categories.

Filling Arrays

Function	Description
count	Fill an array with evenly spaced values using a start and step values.
cycle	Fill an array with evenly spaced values using a start, stop, and step values, and repeat until the array is filled.
repeat	Fill an array with a specified value.

Filtering Arrays

Function	Description
afilter	Select values from an array based on a boolean criteria.
compress	Select values from an array based on another array of boolean values.
dropwhile	Select values from an array starting from where a selected criteria fails and proceeding to the end.
takewhile	Like dropwhile, but starts from the beginning and stops when the criteria fails.

Examining and Searching Arrays

Function	Description
findindex	Returns the index of the first value in an array to meet the specified criteria.
findindices	Searches an array for the array indices which meet the specified criteria and writes the results to a second array. Also returns the number of matches found.

Summarising Arrays

Function	Description
aany	Returns True if any element in an array meets the selected criteria.
aall	Returns True if all element in an array meet the selected criteria.

amax	Returns the maximum value in the array.
amin	Returns the minimum value in the array.
asum	Calculate the arithmetic sum of an array.

Data Conversion

Function	Description
convert	Convert arrays between data types. The data will be converted into the form required by the output array.

Mathematical operator functions

Function	Equivalent to
abs_	[abs(x) for x in array1]
add	[x + param for x in array1]
floordiv	[x // param for x in array1]
mod	[x % param for x in array1]
mul	[x * param for x in array1]
neg	[-x for x in array1]
pow	[x ** param for x in array1]
pow2	[x * x for x in array1]
pow3	[x * x * x for x in array1]
sub	[x - param for x in array1]
truediv	[x / param for x in array1]

Comparison operator functions

Function	Equivalent to
eq	all([x == param for x in array1])
ge	all([x >= param for x in array1])
gt	all([x > param for x in array1])
le	all([x <= param for x in array1])
lt	all([x < param for x in array1])
ne	all([x != param for x in array1])

Bitwise operator functions

Function	Equivalent to
and_	[x x & y param for x in array1]
invert	[~x for x in array1]
lshift	[x x << y param for x in array1]

or_	[x x | y param for x in array1]
rshift	[x x >> y param for x in array1]
xor	[x x ^ y param for x in array1]

Power and logarithmic functions

Function	Equivalent to
exp	[math.exp(x) for x in array1]
expm1	[math.expm1(x) for x in array1]
log	[math.log(x) for x in array1]
log10	[math.log10(x) for x in array1]
log1p	[math.log1p(x) for x in array1]
log2	[math.log2(x) for x in array1]
sqrt	[math.sqrt(x) for x in array1]

Hyperbolic functions

Function	Equivalent to
acosh	[math.acosh(x) for x in array1]
asinh	[math.asinh(x) for x in array1]
atanh	[math.atanh(x) for x in array1]
cosh	[math.cosh(x) for x in array1]
sinh	[math.sinh(x) for x in array1]
tanh	[math.tanh(x) for x in array1]

Trigonometric functions

Function	Equivalent to
acos	[math.acos(x) for x in array1]
asin	[math.asin(x) for x in array1]
atan	[math.atan(x) for x in array1]
atan2	[atan2(x, param) for x in array1]
cos	[math.cos(x) for x in array1]
hypot	[hypot(x, param) for x in array1]
sin	[math.sin(x) for x in array1]
tan	[math.tan(x) for x in array1]

Angular conversion

Function	Equivalent to

degrees	[math.degrees(x) for x in array1]
radians	[math.radians(x) for x in array1]

Number-theoretic and representation functions

Function	Equivalent to
ceil	[math.ceil(x) for x in array1]
copysign	[copysign(x, param) for x in array1]
fabs	[math.fabs(x) for x in array1]
factorial	[math.factorial(x) for x in array1]
floor	[math.floor(x) for x in array1]
fmod	[fmod(x, param) for x in array1]
isfinite	all([isfinite(x) for x in array1])
isinf	any([isinf(x) for x in array1])
isnan	any([isnan(x) for x in array1])
ldexp	math.ldexp(x, y)
trunc	[math.trunc(x) for x in array1]

Special functions

Function	Equivalent to
erf	[math.erf(x) for x in array1]
erfc	[math.erfc(x) for x in array1]
gamma	[math.gamma(x) for x in array1]
lgamma	[math.lgamma(x) for x in array1]

Additional functions

Function	Equivalent to
fma	[(x * param2 + param3) for x in array1]

Array Limit Attributes

In addition to functions, a set of attributes are provided representing the platform specific maximum and minimum numerical values for each array type. These attributes are part of the "arraylimits" module.

Searching and Summarising Arrays.

Comparison Operators

Some functions use comparison operators. These are unicode strings containing the Python compare operators and include following:

Operator	Description
'<'	Less than.
'<='	Less than or equal to.
'>'	Greater than.
'>='	Greater than or equal to.
'=='	Equal to.
'!='	Not equal to.

All comparison operators must contain only the above characters and may not include any leading or trailing spaces or other characters.

Description

aall

Calculate aall over the values in an array.

Equivalent to:	all([(x > param) for x in array])
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = aall(opstr, array, param)
result = aall(opstr, array, param, maxlen=y)
result = aall(opstr, array, param, nosimd=False)
```

- **opstr - The arithmetic comparison operation as a string.**

  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If any comparison operations result in true, the return value will be true. If all of them result in false, the return value will be false.

aany

Calculate aany over the values in an array.

Equivalent to:	any([(x > param) for x in array])
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = aany(opstr, array, param)
result = aany(opstr, array, param, maxlen=y)
result = aany(opstr, array, param, nosimd=False)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

afilter

Select values from an array based on a boolean criteria.

Equivalent to:	filter(lambda x: x < param, array)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = afilter(opstr, array, outparray, param)
result = afilter(opstr, array, outparray, param, maxlen=y)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- outparray - The output array.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - An integer count of the number of items filtered into outparray.

amax

Calculate amax over the values in an array.

Equivalent to:	max(x)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = amax(array)
result = amax(array, maxlen=y)
result = amax(array, nosimd=False)
```

- array - The input data array to be examined.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result = The maximum of all the values in the array.

amin

Calculate amin over the values in an array.

Equivalent to:	min(x)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = amin(array)
result = amin(array, maxlen=y)
result = amin(array, nosimd=False)
```

- array - The input data array to be examined.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result = The minimum of all the values in the array.

asum

Calculate the arithmetic sum of an array.

Equivalent to:	sum()
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = asum(array)
result = asum(array, maxlen=y)
result = asum(array, nosimd=False)
result = asum(array, matherrors=False)
```

- array - The input data array to be examined.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.

- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- matherrors - If True, checks for numerical errors including integer overflow are ignored.
- result - The sum of the array.

compress

Select values from an array based on another array of integers values. The selector array is interpreted as a set of boolean values, where any value other than *0* causes the value in the input array to be selected and copied to theoutput array, while a value of *0* causes the value to be ignored.

Equivalent to:	itertools.compress(inparray, selectorarray)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
x = compress(inparray, outparray, selectorarray)
x = compress(inparray, outparray, selectorarray, maxlen=y)
```

- inparray - The input data array to be filtered.
- outparray - The output array.
- selectorarray - The selector array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- x - An integer count of the number of items filtered into outparray.

convert

Convert arrays between data types. The data will be converted into the form required by the output array. If any values in the input array are outside the range of the output array type, an exception will be raised. When floating point values are converted to integers, the value will be truncated.

Equivalent to:	[x for x in inputarray]
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
convert(inparray, outparray)
convert(inparray, outparray, maxlen=y)
```

- inparray - The input data array to be filtered.
- outparray - The output array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.

count

Fill an array with evenly spaced values using a start and step values.

Equivalent to:	itertools.count(start, len(array))

or	itertools.count(start, len(array), step)

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
count(array, start, step).
```

- array - The output array.
- start - The numeric value to start from.
- step - The value to increment by when creating each element. This parameter is optional. If it is omitted, a value of 1 is assumed. A

cycle

Fill an array with a series of values, repeating as necessary.

Equivalent to:	itertools.cycle(itertools.count(start, len(array)))
or	itertools.cycle(itertools.count(start, len(array), step))

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
cycle(array, start, stop, step)
```

- array - The output array.
- start - The numeric value to start from.
- stop - The value at which to stop incrementing. If stop is less than start, cycle will count down.
- step - The value to increment by when creating each element. This parameter is optional. If it is omitted, a value of 1 is assumed. The

dropwhile

Select values from an array starting from where a selected criteria fails and proceeding to the end.

Equivalent to:	itertools.dropwhile(lambda x: x < param, array)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = dropwhile(opstr, array, outparray, param)
result = dropwhile(opstr, array, outparray, param, maxlen=y)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- outparray - The output array.
- param - A non-array numeric parameter.

- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - An integer count of the number of items filtered into outparray.

findindex

Calculate findindex over the values in an array.

Equivalent to:	[x for x,y in enumerate(array) if y > param][0]
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = findindex(opstr, array, param)
result = findindex(opstr, array, param, maxlen=y)
result = findindex(opstr, array, param, nosimd=False)
```

- **opstr - The arithmetic comparison operation as a string.**

  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - The resulting index. This will be negative if no match was found.

findindices

Searches an array for the array indices which meet the specified criteria and writes the results to a second array. Also returns the number of matches found.

Equivalent to:	[x for x,y in enumerate(inparray) if y == param]
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = findindices(opstr, array, arrayout, param)
result = findindices(opstr, array, arrayout, param, maxlen=y)
```

- **opstr - The arithmetic comparison operation as a string.**

  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- arrayout - The output array. This must be an integer array of array type 'q' (signed long long).
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.

- result - An integer indicating the number of matches found.

repeat

Fill an array with a specified value.

Equivalent to:	itertools.repeat(value)

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
repeat(array, value)
```

- array - The output array.

takewhile

Select values from an array starting from the beginning and stopping when the criteria fails.

Equivalent to:	itertools.takewhile(lambda x: x < param, array)
Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = takewhile(opstr, array, outparray, param)
result = takewhile(opstr, array, outparray, param, maxlen=y)
```

- **opstr - The arithmetic comparison operation as a string.**
  These are: '==', '>', '>=', '<', '<=', '!='.
- array - The input data array to be examined.
- outparray - The output array.
- param - A non-array numeric parameter.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - An integer count of the number of items filtered into outparray.

arraylimits attributes

A set of attributes are provided representing the platform specific maximum and minimum numerical values for each array type. These attributes are part of the "arraylimits" module.

Array integer sizes may differ on 32 versus 64 bit versions, plus other platform characteristics may also produce differences.

Array Type Code	Description	Min Value	Max Value
b	signed char	b_min	b_max
B	unsigned char	B_min	B_max
h	signed short	h_min	h_max
H	unsigned short	H_min	H_max

i	signed int	i_min	i_max
I	unsigned int	I_min	I_max
l	signed long	l_min	l_max
L	unsigned long	L_min	L_max
q	signed long long	q_min	q_max
Q	unsigned long long	Q_min	Q_max
f	float	f_min	f_max
d	double	d_min	d_max

example:

```
import arrayfunc
from arrayfunc import arraylimits

arrayfunc.arraylimits.b_min
==> -128
arrayfunc.arraylimits.b_max
==> 127
arrayfunc.arraylimits.f_min
==> -3.4028234663852886e+38
arrayfunc.arraylimits.f_max
==> 3.4028234663852886e+38
```

Mathematical Functions

Description

Mathematical functions provide similar functionality to the functions of the same name in the standard library "math" and "operator" modules, but operate over whole arrays instead of on a single value.

Mathematical functions can accept a variety of different combinations of array and numerical parameters. Each function will automatically detect the category of parameter and adjust its behaviour accordingly.

Output can be either into a separate output array, or in-place (into the original array) if no output array is provided.

Parameter Forms

This example will subtract 10 from each element of array 'x', replacing the original data.:

```
x = array.array('b', [20,21,22,23,24,25])
arrayfunc.sub(x, 10)
```

This example will do the same, but place the results into array 'z', leaving the original array unchanged.:

```
x = array.array('b', [20,21,22,23,24,25])
z = array.array('b', [0] * len(x))
arrayfunc.sub(x, 10, z)
```

This is similar to the first one, but performs the calculation of '10 - x' instead of 'x - 10'.:

```
x = array.array('b', [20,21,22,23,24,25])
arrayfunc.sub(10, x)
```

This example takes each element of array 'x', adds the corresponding element of array 'y', and puts the result in array 'z'.:

```
x = array.array('b', [20,21,22,23,24,25])
y = array.array('b', [10,5,55,42,42,0])
z = array.array('b', [0] * len(x))
arrayfunc.add(x, y, z)
```

Parameter Type Consistency

Unless otherwise noted, all array and numeric parameters must be of the same type when calling a mathematical function. That is, you may not mix integer and floating point, or different integer sizes in the same calculation. Failing to use consistent parameters will result in an exception being raised.

Using Less than the Entire Array

If the size of the array is larger than the desired length of the calculation, it may be limited to the first part of the array by using the 'maxlen' parameter. In the following example only the first 3 array elements will be operated on, with the following ones left unchanged.:

```
x = array.array('b', [20,21,22,23,24,25])
arrayfunc.add(x, 10, maxlen=3)
```

Suppressing or Ignoring Math Errors

Functions can be made to ignore some mathematical errors (e.g. integer overflow) by setting the 'matherrors' keyword parameter to True.:

```
x = array.array('b', [20,21,22,23,24,25])
arrayfunc.add(x, 235, matherrors=True)
```

However, not all math errors can be suppressed, only those which would not otherwise cause a fatal error (e.g. division by zero).

Ignoring errors may be desirable if the side effect (e.g. the result of an integer overflow) is the intended effect, or for reasons of a minor performance improvement in some cases. Note that any such performance improvement will vary greatly depending upon the specific function and array type. Benchmark your calculation before deciding if this is worth while.

Differences with Native Python

In many cases the Python 'math' module functions are thin wrappers around the underlying C library, as is 'arrayfunc'.

However, in some cases 'arrayfunc' will not produce exactly the same result as Python. There are several reasons for this, the primary one being that arrayfunc operates on different underlying data types. Specifically, arrayfunc uses the platform's native integer and floating point types as exposed by the array module. For example, Python integers are of arbitrary size and can never overflow (Python simply expands the word size indefinitely), while arrayfunc integers will overflow the same as they would with programs written in C.

Think of arrayfunc as exposing C style semantics in a form convenient to use in Python. Some convenience which Python provides (e.g. no limit to the size of integers) is traded off for large performance increases.

However, Arrayfunc does implement the mod or '%' operator in a manner which is compatible with Python, not 'C'. The C method will produce mathematically incorrect answers under some ranges of values (as will many other programming languages as well as some popular spreadsheets which use the C compiler without correction). Python implements this in a mathematically correct manner in all cases, and Arrayfunc follows suit.

Arrayfunc diverges from Python in the following areas:

- The handling of non-finite floating point values such as 'NaN' (not-a-number) and +/-Inf in calculations may not always be compatible.
- The 'floor' function will return a floating point value when floating point arrays are used, rather than an integer. This is necessary to maintain compatibility with the array parameters.
- Floordiv does not behave the same as '//' when working with infinity. When dividing positive or negative infinity by any number, the arrayfunc version of floordiv will return +/- infinity, while the Python '//' operator will return 'NaN' (not-a-number) in each case.
- Binary operations such as shift and invert will operate according to their native array data types, which may differ from Python's own integer implementation. This is necessary because the array integer is of fixed size (Python integers can be infinitely large) and has both signed and unsigned types (Python integers are signed only).
- "Mod" does not behave exactly as "%" does for floating point. X % inf and x % -inf will return nan rather than +/- inf.
- The type of exception raised when an error is encountered in Python versus arrayfunc may not be the same in all cases.

Other Notes

- Ldexp only accepts an integer number as the second parameter, not an array.
- Math.pow is not implemented because it duplicates the operator pow (and the names would collide in arrayfunc).
- Fma is not part of the Python standard library, but has been offered here as an additional feature.

Mathematical operator functions

*abs_*

Calculate abs_ over the values in an array.

Equivalent to:	[abs(x) for x in array1]

Array types supported:	b, h, i, l, q, f, d
Exceptions raised:	OverflowError

Call formats:

```
abs_(array1)
abs_(array1, outparray)
abs_(array1, maxlen=y)
abs_(array1, matherrors=False))
abs_(array1, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

add

Calculate add over the values in an array.

Equivalent to:	[x + param for x in array1]
or	[param + y for y in array2]
or	[x + y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
add(array1, param)
add(array1, param, outparray)
add(param, array1)
add(param, array1, outparray)
add(array1, array2)
add(array1, array2, outparray)
add(array1, param, maxlen=y)
add(array1, param, matherrors=False)
add(array, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is

floordiv

Calculate floordiv over the values in an array.

Equivalent to:	[x // param for x in array1]
or	[param // y for y in array2]

or	[x // y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError, ZeroDivisionError

Call formats:

```
floordiv(array1, param)
floordiv(array1, param, outparray)
floordiv(param, array1)
floordiv(param, array1, outparray)
floordiv(array1, array2)
floordiv(array1, array2, outparray)
floordiv(array1, param, maxlen=y)
floordiv(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

mod

Calculate mod over the values in an array.

Equivalent to:	[x % param for x in array1]
or	[param % y for y in array2]
or	[x % y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError, ZeroDivisionError

Call formats:

```
mod(array1, param)
mod(array1, param, outparray)
mod(param, array1)
mod(param, array1, outparray)
mod(array1, array2)
mod(array1, array2, outparray)
mod(array1, param, maxlen=y)
mod(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.

- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

mul

Calculate mul over the values in an array.

Equivalent to:	[x * param for x in array1]
or	[param * y for y in array2]
or	[x * y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
mul(array1, param)
mul(array1, param, outparray)
mul(param, array1)
mul(param, array1, outparray)
mul(array1, array2)
mul(array1, array2, outparray)
mul(array1, param, maxlen=y)
mul(array1, param, matherrors=False)
mul(array, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is

neg

Calculate neg over the values in an array.

Equivalent to:	[-x for x in array1]

Array types supported:	b, h, i, l, q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
neg(array1)
neg(array1, outparray)
neg(array1, maxlen=y)
neg(array1, matherrors=False))
neg(array1, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

pow

Calculate pow over the values in an array.

Equivalent to:	[x ** param for x in array1]
or	[param ** y for y in array2]
or	[x ** y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError, ValueError

Call formats:

```
pow(array1, param)
pow(array1, param, outparray)
pow(param, array1)
pow(param, array1, outparray)
pow(array1, array2)
pow(array1, array2, outparray)
pow(array1, param, maxlen=y)
pow(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.

- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

pow2

Calculate pow2 over the values in an array.

Equivalent to:	[x * x for x in array1]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError, ValueError

Call formats:

```
pow2(array1)
pow2(array1, outparray)
pow2(array1, maxlen=y)
pow2(array1, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

pow3

Calculate pow3 over the values in an array.

Equivalent to:	[x * x * x for x in array1]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError, ValueError

Call formats:

```
pow3(array1)
pow3(array1, outparray)
pow3(array1, maxlen=y)
pow3(array1, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

sub

Calculate sub over the values in an array.

Equivalent to:	[x - param for x in array1]
or	[param - y for y in array2]
or	[x - y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError

Call formats:

```
sub(array1, param)
sub(array1, param, outparray)
sub(param, array1)
sub(param, array1, outparray)
sub(array1, array2)
sub(array1, array2, outparray)
sub(array1, param, maxlen=y)
sub(array1, param, matherrors=False)
sub(array, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is

truediv

Calculate truediv over the values in an array.

Equivalent to:	[x / param for x in array1]
or	[param / y for y in array2]
or	[x / y for x, y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d
Exceptions raised:	OverflowError, ArithmeticError, ZeroDivisionError

Call formats:

```
truediv(array1, param)
truediv(array1, param, outparray)
truediv(param, array1)
```

```
truediv(param, array1, outparray)
truediv(array1, array2)
truediv(array1, array2, outparray)
truediv(array1, param, maxlen=y)
truediv(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Comparison operator functions

eq

Calculate eq over the values in an array.

Equivalent to:	all([x == param for x in array1])
or	all([param == x for x in array1])
or	all([x == y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = eq(array1, param)
result = eq(param, array1)
result = eq(array1, array2)
result = eq(array1, param, maxlen=y)
result = eq(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).

- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

ge

Calculate ge over the values in an array.

Equivalent to:	all([x >= param for x in array1])
or	all([param >= x for x in array1])
or	all([x >= y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = ge(array1, param)
result = ge(param, array1)
result = ge(array1, array2)
result = ge(array1, param, maxlen=y)
result = ge(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

gt

Calculate gt over the values in an array.

Equivalent to:	all([x > param for x in array1])
or	all([param > x for x in array1])
or	all([x > y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = gt(array1, param)
result = gt(param, array1)
result = gt(array1, array2)
```

```
result = gt(array1, param, maxlen=y)
result = gt(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

le

Calculate le over the values in an array.

Equivalent to:	all([x <= param for x in array1])
or	all([param <= x for x in array1])
or	all([x <= y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = le(array1, param)
result = le(param, array1)
result = le(array1, array2)
result = le(array1, param, maxlen=y)
result = le(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

lt

Calculate lt over the values in an array.

Equivalent to:	all([x < param for x in array1])
or	all([param < x for x in array1])
or	all([x < y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = lt(array1, param)
result = lt(param, array1)
result = lt(array1, array2)
result = lt(array1, param, maxlen=y)
result = lt(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

ne

Calculate ne over the values in an array.

Equivalent to:	all([x != param for x in array1])
or	all([param != x for x in array1])
or	all([x != y for x,y in zip(array1, array2)])

Array types supported:	b, B, h, H, i, I, l, L, q, Q, f, d

Call formats:

```
result = ne(array1, param)
result = ne(param, array1)
result = ne(array1, array2)
result = ne(array1, param, maxlen=y)
result = ne(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.

- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled if present. The default is False (SIMD acceleration is enabled if present).
- result - A boolean value corresponding to the result of all the comparison operations. If all comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

Bitwise operator functions

*and_*

Calculate and_ over the values in an array.

Equivalent to:	[x x & y param for x in array1]
or	[param x & y x for x in array1]
or	[x x & y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
and_(array1, param)
and_(array1, param, outparray)
and_(param, array1)
and_(param, array1, outparray)
and_(array1, array2)
and_(array1, array2, outparray)
and_(array1, param, maxlen=y)
and_(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

invert

Calculate invert over the values in an array.

Equivalent to:	[~x for x in array1]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
invert(array1)
invert(array1, outparray)
invert(array1, maxlen=y)
invert(array1, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

lshift

Calculate lshift over the values in an array.

Equivalent to:	[x x << y param for x in array1]
or	[param x << y x for x in array1]
or	[x x << y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
lshift(array1, param)
lshift(array1, param, outparray)
lshift(param, array1)
lshift(param, array1, outparray)
lshift(array1, array2)
lshift(array1, array2, outparray)
lshift(array1, param, maxlen=y)
lshift(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.

- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

*or_*

Calculate or_ over the values in an array.

Equivalent to:	[x x | y param for x in array1]
or	[param x | y x for x in array1]
or	[x x | y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
or_(array1, param)
or_(array1, param, outparray)
or_(param, array1)
or_(param, array1, outparray)
or_(array1, array2)
or_(array1, array2, outparray)
or_(array1, param, maxlen=y)
or_(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

rshift

Calculate rshift over the values in an array.

Equivalent to:	[x x >> y param for x in array1]
or	[param x >> y x for x in array1]
or	[x x >> y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
rshift(array1, param)
rshift(array1, param, outparray)
rshift(param, array1)
rshift(param, array1, outparray)
rshift(array1, array2)
rshift(array1, array2, outparray)
rshift(array1, param, maxlen=y)
rshift(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

xor

Calculate xor over the values in an array.

Equivalent to:	[x x ^ y param for x in array1]
or	[param x ^ y x for x in array1]
or	[x x ^ y y for x,y in zip(array1, array2)]

Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	

Call formats:

```
xor(array1, param)
xor(array1, param, outparray)
xor(param, array1)
xor(param, array1, outparray)
xor(array1, array2)
xor(array1, array2, outparray)
xor(array1, param, maxlen=y)
xor(array1, param, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.

- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Power and logarithmic functions

exp

Calculate exp over the values in an array.

Equivalent to:	[math.exp(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
exp(array1)
exp(array1, outparray)
exp(array1, maxlen=y)
exp(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

expm1

Calculate expm1 over the values in an array.

Equivalent to:	[math.expm1(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
expm1(array1)
expm1(array1, outparray)
expm1(array1, maxlen=y)
expm1(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.

- matherrors - If true, arithmetic error checking is disabled. The default is false.

log

Calculate log over the values in an array.

Equivalent to:	[math.log(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
log(array1)
log(array1, outparray)
log(array1, maxlen=y)
log(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

log10

Calculate log10 over the values in an array.

Equivalent to:	[math.log10(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
log10(array1)
log10(array1, outparray)
log10(array1, maxlen=y)
log10(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

log1p

Calculate log1p over the values in an array.

Equivalent to:	[math.log1p(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
log1p(array1)
log1p(array1, outparray)
log1p(array1, maxlen=y)
log1p(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

log2

Calculate log2 over the values in an array.

Equivalent to:	[math.log2(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
log2(array1)
log2(array1, outparray)
log2(array1, maxlen=y)
log2(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

sqrt

Calculate sqrt over the values in an array.

Equivalent to:	[math.sqrt(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
sqrt(array1)
sqrt(array1, outparray)
sqrt(array1, maxlen=y)
sqrt(array1, matherrors=False))
sqrt(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Hyperbolic functions

acosh

Calculate acosh over the values in an array.

Equivalent to:	[math.acosh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
acosh(array1)
acosh(array1, outparray)
acosh(array1, maxlen=y)
acosh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

asinh

Calculate asinh over the values in an array.

Equivalent to:	[math.asinh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
asinh(array1)
asinh(array1, outparray)
asinh(array1, maxlen=y)
asinh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

atanh

Calculate atanh over the values in an array.

Equivalent to:	[math.atanh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
atanh(array1)
atanh(array1, outparray)
atanh(array1, maxlen=y)
atanh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

cosh

Calculate cosh over the values in an array.

Equivalent to:	[math.cosh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
cosh(array1)
cosh(array1, outparray)
```

```
cosh(array1, maxlen=y)
cosh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

sinh

Calculate sinh over the values in an array.

Equivalent to:	[math.sinh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
sinh(array1)
sinh(array1, outparray)
sinh(array1, maxlen=y)
sinh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

tanh

Calculate tanh over the values in an array.

Equivalent to:	[math.tanh(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
tanh(array1)
tanh(array1, outparray)
tanh(array1, maxlen=y)
tanh(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Trigonometric functions

acos

Calculate acos over the values in an array.

Equivalent to:	[math.acos(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
acos(array1)
acos(array1, outparray)
acos(array1, maxlen=y)
acos(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

asin

Calculate asin over the values in an array.

Equivalent to:	[math.asin(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
asin(array1)
asin(array1, outparray)
asin(array1, maxlen=y)
asin(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.

- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

atan

Calculate atan over the values in an array.

Equivalent to:	[math.atan(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
atan(array1)
atan(array1, outparray)
atan(array1, maxlen=y)
atan(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

atan2

Calculate atan2 over the values in an array.

Equivalent to:	[atan2(x, param) for x in array1]
or	[atan2(param, x) for x in array1]
or	[atan2(x, y) for x, y in zip(array1, array2)]

Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
atan2(array1, param)
atan2(array1, param, outparray)
atan2(param, array1)
atan2(param, array1, outparray)
atan2(array1, array2)
atan2(array1, array2, outparray)
atan2(array1, param, maxlen=y)
atan2(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

cos

Calculate cos over the values in an array.

Equivalent to:	[math.cos(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
cos(array1)
cos(array1, outparray)
cos(array1, maxlen=y)
cos(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

hypot

Calculate hypot over the values in an array.

Equivalent to:	[hypot(x, param) for x in array1]
or	[hypot(param, x) for x in array1]
or	[hypot(x, y) for x, y in zip(array1, array2)]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
hypot(array1, param)
hypot(array1, param, outparray)
hypot(param, array1)
hypot(param, array1, outparray)
```

```
hypot(array1, array2)
hypot(array1, array2, outparray)
hypot(array1, param, maxlen=y)
hypot(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

sin

Calculate sin over the values in an array.

Equivalent to:	[math.sin(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
sin(array1)
sin(array1, outparray)
sin(array1, maxlen=y)
sin(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

tan

Calculate tan over the values in an array.

Equivalent to:	[math.tan(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
tan(array1)
tan(array1, outparray)
tan(array1, maxlen=y)
tan(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Angular conversion

degrees

Calculate degrees over the values in an array.

Equivalent to:	[math.degrees(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
degrees(array1)
degrees(array1, outparray)
degrees(array1, maxlen=y)
degrees(array1, matherrors=False))
degrees(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

radians

Calculate radians over the values in an array.

Equivalent to:	[math.radians(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
radians(array1)
radians(array1, outparray)
radians(array1, maxlen=y)
radians(array1, matherrors=False))
radians(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Number-theoretic and representation functions

ceil

Calculate ceil over the values in an array.

Equivalent to:	[math.ceil(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
ceil(array1)
ceil(array1, outparray)
ceil(array1, maxlen=y)
ceil(array1, matherrors=False))
ceil(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

copysign

Calculate copysign over the values in an array.

Equivalent to:	[copysign(x, param) for x in array1]
or	[copysign(param, x) for x in array1]
or	[copysign(x, y) for x, y in zip(array1, array2)]

Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
copysign(array1, param)
copysign(array1, param, outparray)
copysign(param, array1)
copysign(param, array1, outparray)
copysign(array1, array2)
copysign(array1, array2, outparray)
copysign(array1, param, maxlen=y)
copysign(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

fabs

Calculate fabs over the values in an array.

Equivalent to:	[math.fabs(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
fabs(array1)
fabs(array1, outparray)
fabs(array1, maxlen=y)
fabs(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

factorial

Calculate factorial over the values in an array.

Equivalent to:	[math.factorial(x) for x in array1]
Array types supported:	b, B, h, H, i, I, l, L, q, Q
Exceptions raised:	OverflowError

Call formats:

```
factorial(array1)
factorial(array1, outparray)
factorial(array1, maxlen=y)
factorial(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

floor

Calculate floor over the values in an array.

Equivalent to:	[math.floor(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
floor(array1)
floor(array1, outparray)
floor(array1, maxlen=y)
floor(array1, matherrors=False))
floor(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

fmod

Calculate fmod over the values in an array.

Equivalent to:	[fmod(x, param) for x in array1]
or	[fmod(param, x) for x in array1]
or	[fmod(x, y) for x, y in zip(array1, array2)]

Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
fmod(array1, param)
fmod(array1, param, outparray)
fmod(param, array1)
fmod(param, array1, outparray)
fmod(array1, array2)
fmod(array1, array2, outparray)
fmod(array1, param, maxlen=y)
fmod(array1, param, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- param - A non-array numeric parameter.
- array2 - A second input data array. Each element in this array is applied to the corresponding element in the first array.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

isfinite

Calculate isfinite over the values in an array.

Equivalent to:	all([isfinite(x) for x in array1])

========== ========== Array types supported: f, d Exceptions raised: ========== ==========

Call formats:

```
result = isfinite(array1)
result = isfinite(array1, maxlen=y)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.

- result - A boolean value corresponding to the result of all the comparison operations. If all of the comparison operations result in true, the return value will be true. If any of them result in false, the return value will be false.

isinf

Calculate isinf over the values in an array.

Equivalent to:	any([isinf(x) for x in array1])

========== ========== Array types supported: f, d Exceptions raised: ========== ==========

Call formats:

```
result = isinf(array1)
result = isinf(array1, maxlen=y)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - A boolean value corresponding to the result of all the comparison operations. If at least one comparison operation results in true, the return value will be true. If none of them result in true, the return value will be false.

isnan

Calculate isnan over the values in an array.

Equivalent to:	any([isnan(x) for x in array1])

========== ========== Array types supported: f, d Exceptions raised: ========== ==========

Call formats:

```
result = isnan(array1)
result = isnan(array1, maxlen=y)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- result - A boolean value corresponding to the result of all the comparison operations. If at least one comparison operation results in true, the return value will be true. If none of them result in true, the return value will be false.

ldexp

Calculate ldexp over the values in an array.

Equivalent to:	math.ldexp(x, y)
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
ldexp(array1, exp)
ldexp(array1, exp, outparray)
ldexp(array1, exp, maxlen=y)
ldexp(array1, exp, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- exp - The exponent to apply to the input array. This must be an integer.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

trunc

Calculate trunc over the values in an array.

Equivalent to:	[math.trunc(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
trunc(array1)
trunc(array1, outparray)
trunc(array1, maxlen=y)
trunc(array1, matherrors=False))
trunc(array, nosimd=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.
- nosimd - If True, SIMD acceleration is disabled. This parameter is optional. The default is FALSE.

Special functions

erf

Calculate erf over the values in an array.

Equivalent to:	[math.erf(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
erf(array1)
erf(array1, outparray)
erf(array1, maxlen=y)
erf(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

erfc

Calculate erfc over the values in an array.

Equivalent to:	[math.erfc(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
erfc(array1)
erfc(array1, outparray)
erfc(array1, maxlen=y)
erfc(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

gamma

Calculate gamma over the values in an array.

Equivalent to:	[math.gamma(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
gamma(array1)
gamma(array1, outparray)
gamma(array1, maxlen=y)
gamma(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

lgamma

Calculate lgamma over the values in an array.

Equivalent to:	[math.lgamma(x) for x in array1]
Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
lgamma(array1)
lgamma(array1, outparray)
lgamma(array1, maxlen=y)
lgamma(array1, matherrors=False))
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Additional functions

fma

Calculate fma over the values in an array.

Equivalent to:	[(x * param2 + param3) for x in array1]
or	[(x * y + param3) for x,y in zip(array1, array2)]

or	[(x * param2 + z) for x,z in zip(array1, array3)]
or	[(x * y + z) for x,y,z in zip(array1, array2, array3)]

Array types supported:	f, d
Exceptions raised:	ArithmeticError

Call formats:

```
fma(array1, array2, array3)
fma(array1, array2, array3, outparray)
fma(array1, array2, param3)
fma(array1, array2, param3, outparray)
fma(array1, param2, array3)
fma(array1, param2, array3, outparray)
fma(array1, param2, param3)
fma(array1, param2, param3, outparray)
fma(array1, array2, array3, maxlen=y)
fma(array1, array2, array3, matherrors=False)
```

- array1 - The first input data array to be examined. If no output array is provided the results will overwrite the input data.
- **array2 - A second input data array. Each element in this array is**
  applied to the corresponding element in the first array.
- **param2 - A non-array numeric parameter which may be used in place**
  of array2.
- array3 - A third input data array. Each element in this array is applied to the corresponding element in the first array.
- **param3 - A non-array numeric parameter which may be used in place**
  of array3.
- outparray - The output array. This parameter is optional.
- maxlen - Limit the length of the array used. This must be a valid positive integer. If a zero or negative length, or a value which is greater than the actual length of the array is specified, this parameter is ignored.
- matherrors - If true, arithmetic error checking is disabled. The default is false.

Option Flags and Parameters

Arithmetic Overflow Control

Many functions allow integer overflow detection to be turned off if desired. See the list of operators for which operators this applies to.

Integer overflow is when a number becomes too large to fit within the specified word size for that array data type. For example, an unsigned char has a range of 0 to 255. When a calculation overflows, it "wraps around" one or more times and produces an arithmetically invalid result.

If it is known in advance that overflow cannot occur (due to the size of the numbers), or if overflow is a desired side effect, then overflow checking may be disabled via the "matherrors" parameter. Setting

"matherrors" to true will *disable* overflow checking, while setting it to false will *enable* overflow checking. Checking is enabled by default, including when the "matherrors" parameter is not specified.

Disabling overflow checking can significantly increase the speed of calculation, with the amount of improvement depending on the type of calculation being performed and the data type used.

Using Only Part of an Array

The array math functions only use existing arrays that the user provides and do not create new arrays or resize existing ones. The reason for this is that when very large arrays are being used, continually allocating and de-allocating arrays can take too much time, plus this may result in problems controlling how much memory is used.

Since the filter functions (or other data sources) may not use all of an output array, and the result may vary depending on the data, most functions provide an optional keyword parameter which limits the functions to part of the array. The "maxlen" parameter specifies the maximum number of array elements to use, starting from the beginning of the array.

For example, specifying a "maxlen" of 10 for a 20 element array will limit a function to using only the first 10 array elements and ignoring the rest of the array.

If the array length limit value is zero, negative, or greater than the actual size of the array, the length limit will be ignored and the entire array used. The default is to use the entire array.

SIMD Control

SIMD (Single Instruction Multiple Data) is a set of CPU features which allow multiple operations to take place in parallel. Some, but not all, functions will make use of these instructions to speed up execution.

Those functions which do support SIMD features will automatically make use of them by default unless this feature is disabled. There is normally no reason to disable SIMD, but should there be hardware related problems the function can be forced to fall back to conventional execution mode.

If the optional parameter "nosimd" is set to true ("nosimd=True"), SIMD execution will be disabled. The default is "False".

To repeat, there is normally no reason to wish to disable SIMD.

See the documentation section on SIMD support has more detail.

Data Types

Array Types

The following array types from the Python standard library are supported.

Array Type Code	Description
b	signed char
B	unsigned char
h	signed short
H	unsigned short
i	signed int
I	unsigned int
l	signed long

L	unsigned long
q	signed long long
Q	unsigned long long
f	float
d	double

Numeric Parameter Types

Python Type	Description
integer	Integral values such as 0, 1, 100, -99, etc.
floating point	Real numbers such as 0.0, 1.93, 3.1417, -5693.0, etc.

The numeric type must be compatible with the array type code.

The 'L' and 'Q' type parameters cannot be checked for integer overflow due to a mismatch between Python and 'C' language numeric limits.

Maximum Array Size

Arrays are limited to no more than the number of elements defined by the Python C API constant Py_ssize_t. The size of this will depend on your platform characteristics. However, it will normally allow for arrays larger than can be contained in memory for most computers.

When creating very large arrays, it is recommended to consider using itertools.repeat as an initializer or to use array.extend or array.append to add to an array rather than using a list as an initializer. Lists use much more memory than arrays (even for the same data type), and it is easy to run out of memory if you are not careful when creating very large arrays from lists.

Platform Compiler Support

Beginning with version 2.0 of ArrayFunc, versions compiled with the Microsoft MSVS compiler now has feature parity with the GCC version. This change is due to the Microsoft C compiler now supporting a new enough version of the 'C' standard.

Integer Error Checking

Error checking in integer operators is conducted as follows:

Error Categories

Operation	Result out of range	Divide by zero	Negate max. negative signed int	Parameter is negative
Addition (+)	X			
Subtraction (-)	X			
Modulus (%)		X	X	
Multiplication (*)	X			
Division (/, //)		X	X	
Negation (-)			X	
Absolute Value			X	

Factorial	X			X
Power (**)	X			X

- Negation of the maximum negative signed in (the most negative integer for that array type) can be caused by negation, absolute value, division, and modulus operations. Since signed integers do not have a symmetrical range (e.g. -128 to 127 for 8 bit sizes) anything which attempts to convert (in this example) -128 to +128 would cause an overflow back to -128.
- The factorial of negative numbers is undefined.
- Powers are not calculated for integers raised to negative powers, as integer arrays cannot contain fractional results.

Disabling Integer Division by Zero Checks

Division by zero cannot be disabled for integer division or modulus operations. Division by zero could cause seg faults (crashes), so this option is ignored for these functions.

Floating Point NaN and Infinity

Floating point numbers include three special values, NaN (Not a Number), and negative and positive infinity. Arrayfunc uses the platform C compiler to create executable code. Some compilers may produce different results than other compilers under certain conditions when operating on NaN and infinity values. In addition, the Arrayfunc results may differ from those in native Python on some platforms when using NaN and infinity as inputs.

However, since using NaN and infinity as numeric inputs is not a common operation, this is unlikely to be a serious problem when writing cross platform code in most cases.

Exceptions

Exceptions - General

The following exceptions apply to most functions.

Exception type	Text	Description
ArithmeticError	arithmetic error in calculation.	An arithmetic error occurred in a calculation.
ZeroDivisionError	zero division error in calculation.	A calculation attempted to divide by zero.
IndexError	array length error.	One or more arrays has an invalid length (e.g a length of zero).
IndexError	input array length error.	The input array has an invalid length.
IndexError	output length error.	The output array has an invalid length.
IndexError	array length mismatch.	Two or more arrays which are expected to be of equal length are not.
OverflowError	arithmetic overflow in calculation.	An arithmetic integer overflow occurred in a calculation.
OverflowError	arithmetic overflow in parameter.	The size or range of a non-array parameter was not compatible with the array parameters.

TypeError	array and parameter type mismatch.	A non-array parameter data type was not compatible with the array parameters.
TypeError	array type mismatch.	An array parameter is not compatible with another array parameter. For most functions, both arrays must be of the same type.
TypeError	unknown array type.	The array type is unknown.
TypeError	array.array expected.	A non-array parameter was found where an array parameter was expected.
ValueError	operator not valid for this function.	An operator parameter used was not valid for this function.
ValueError	operator not valid for this platform.	The operator used is not supported on this platform.
TypeError	parameter error.	An unspecified error occurred when parsing the parameters.
TypeError	parameter missing.	An expected parameter was missing.
ValueError	parameter not valid for this operation.	A value is not valid for this operation. E.g. attempting to perform a factorial on a negative number.
IndexError	selector length error.	The selector array length is incorrect.
ValueError	conversion not valid for this type.	The conversion attempted was invalid.
ValueError	cannot convert float NaN to integer.	Cannot convert NaN (Not A Number) floating point value in the input array to integer.
TypeError	output array type invalid.	The output array type is invalid.

Platform Oddities

As most operators are implemented using native behaviour, details of some operations may depend on the CPU architecture.

Lshift and rshift will exhibit a behaviour that depends on the CPU type whether it is 32 or 64 bit, and array size.

For 32 bit x86 systems, if the array word size is 32 bits or less, the shift is masked to 5 bits. That is, shift amounts greater than 32 will "roll over", repeating smaller shifts.

On 64 bit systems, this behaviour will vary depending on whether SIMD is used or not. This, arrays which are not even multiples of SIMD register sizes may exibit different behaviour at different array indexes (depending on whether SIMD or non-SIMD instructions were used for those parts of the array).

ARM does not display this roll-over behaviour, and so may give different results than x86. However, negative shift values may result in the shift operation being conducted in the opposite direction (e.g. right shift instead of left shift).

The conclusion is that bit shift operations which use a shift amount which is not in the range of 0 to "maximum number" may produce undefined results. So valid bit shift amounts should be 0 to 7, 0 to 15, 0 to 31 and 0 to 63, depending on the array type.

SIMD Support

General

SIMD (Single Instruction Multiple Data) is a set of CPU features which allow multiple operations to take place in parallel. Some, but not all, functions will make use of these instructions to speed up execution.

Those functions which do support SIMD features will automatically make use of them by default unless this feature is disabled. There is normally no reason to disable SIMD, but should there be hardware related problems the function can be forced to fall back to conventional execution mode.

Platform Support

SIMD instructions are presently supported only on the following:

- 64 bit x86 (i.e. AMD64) using GCC.
- 32 bit ARMv7 using GCC (tested on Raspberry Pi 3).
- 64 bit ARMv8 AARCH64 using GCC (tested on Raspberry Pi 4).

Other compilers or platforms will still run the same functions and should produce the same results, but they will not benefit from SIMD acceleration.

However, non-SIMD functions will still be much faster standard Python code. See the performance benchmarks to see what the relative speed differences are. With wider data types (e.g. double precision floating point) SIMD provides only marginal speed ups anyway.

Raspberry Pi 32 versus 64 bit

The Raspberry Pi uses an ARM CPU. This can operate in 32 or 64 bit mode. When in 32 bit mode, the Raspberry Pi 3 operates in ARMv7 mode. This has 64 bit ARM NEON SIMD vectors.

When in 64 bit mode, it acts as an ARMv8, with AARCH64 128 bit ARM NEON SIMD vectors.

The Raspbian Linux OS is 32 bit mode only. Other distros such as Ubuntu offer 64 bit versions.

The "setup.py" file uses platform detection code to determine which ARM CPU and mode it is running on. Due to the availability of hardware for testing, this code is tailored to the Raspberry Pi 3 and Raspberry Pi 4 and the operating systems listed. This code then selects the appropriate compiler arguments to pass to the setup routines to tell the compiler what mode to compile for.

If other ARM platforms are used which have different platform signatures or which require different compiler arguments, the "setup.py" file may need to be modified in order to use SIMD acceleration.

However, the straight 'C' code should still compile and run, and still provide performance many times faster than when using native Python.

Data Type Support

x86-64

The following table shows which array data types are supported by x86-64 SIMD instructions.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	X	X	X	X	X	X					X	X
aany	X	X	X	X	X	X					X	X
abs_	X		X		X							
add	X		X		X						X	X

amax	X	X	X	X	X	X					X	X
amin	X	X	X	X	X	X					X	X
and_	X	X	X	X	X	X						
asum											X	X
ceil											X	X
degrees											X	X
eq	X	X	X	X	X	X					X	X
findindex	X	X	X	X	X	X					X	X
floor											X	X
ge	X	X	X	X	X	X					X	X
gt	X	X	X	X	X	X					X	X
invert	X	X	X	X	X	X						
le	X	X	X	X	X	X					X	X
lshift	X	X	X	X	X	X						
lt	X	X	X	X	X	X					X	X
mul											X	X
ne	X	X	X	X	X	X					X	X
neg	X		X		X							
or_	X	X	X	X	X	X						
radians											X	X
rshift	X	X		X	X	X						
sqrt											X	X
sub	X		X		X						X	X
trunc											X	X
xor	X	X	X	X	X	X						

ARMv7

The following table shows which array data types are supported by ARMv7 SIMD instructions.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	X	X	X	X								
aany	X	X	X	X								
abs_	X		X								X	
add	X	X	X	X							X	
amax	X	X	X	X	X	X					X	
amin	X	X	X	X	X	X					X	
and_	X	X	X	X	X	X						
degrees											X	
eq	X	X	X	X								

findindex	X	X	X	X								
ge	X	X	X	X								
gt	X	X	X	X								
invert	X	X	X	X	X	X						
le	X	X	X	X								
lshift	X	X	X	X	X	X						
lt	X	X	X	X								
mul	X	X	X	X							X	
ne	X	X	X	X								
neg	X		X								X	
or_	X	X	X	X	X	X						
radians											X	
rshift	X	X	X	X	X	X						
sub	X	X	X	X							X	
xor	X	X	X	X	X	X						

ARMv8 AARCH64

The following table shows which array data types are supported by ARMv8 SIMD instructions.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	X	X	X	X	X	X					X	
aany	X	X	X	X	X	X					X	
abs_	X		X		X						X	
add	X	X	X	X	X	X					X	
amax	X	X	X	X	X	X					X	
amin	X	X	X	X	X	X					X	
and_	X	X	X	X	X	X						
degrees											X	
eq	X	X	X	X	X	X					X	
findindex	X	X	X	X	X	X					X	
ge	X	X	X	X	X	X					X	
gt	X	X	X	X	X	X					X	
invert	X	X	X	X	X	X						
le	X	X	X	X	X	X					X	
lshift	X	X	X	X	X	X						
lt	X	X	X	X	X	X					X	
mul	X	X	X	X	X	X					X	
ne	X	X	X	X	X	X					X	
neg	X		X		X						X	

or_	X	X	X	X	X	X						
radians											X	
rshift	X	X	X	X	X	X						
sub	X	X	X	X	X	X					X	
xor	X	X	X	X	X	X						

SIMD Support Attributes

There is an attribute which can be tested to detect if ArrayFunc is compiled with SIMD support and if the current hardware supports the required SIMD level.

arrayfunc.simdsupport.hassimd

The attribute "hassimd" will be True if the module supports SIMD.

example:

```
import arrayfunc
arrayfunc.simdsupport.hassimd
==> True
```

Performance

Variables affecting Performance

The purpose of the Arrayfunc module is to execute common operations faster than native Python. The relative speed will depend upon a number of factors:

- The function.
- The data type of the array.
- Function options. Turning checking off will result in faster performance.
- The data in the arrays and the parameters.
- The size of the array.
- The platform, including CPU type (e.g. x86 or ARM), operating system, and compiler.

The speeds listed below should be used as rough guidelines only. More exact results will require application specific testing. The numbers shown are the execution time of each function relative to native Python. For example, a value of '50' means that the corresponding Arrayfunc operation ran 50 times faster than the closest native Python equivalent.

Both relative performance (the speed-up as compared to Python) and absolute performance (the actual execution speed of Python and ArrayFunc) will vary significantly depending upon the compiler (which is OS platform dependent) and whether compiled to 32 or 64 bit. If your precise actual benchmark performance results matter, be sure to conduct your testing using the actual OS and compiler your final program will be deployed on. The values listed below were measured on x86-64 Linux compiled with GCC.

Note: Some more complex Arrayfunc functions do not work exactly the same way as the built-in or "itertools" Python equivalents. This means that the benchmark results should be taken as general guidelines rather than precise comparisons.

Typical Performance Readings

Default Performance

In this set of tests, all error checking was turned on and SIMD acceleration was enabled where this did not conflict with the preceding (the defaults in each case).

Relative Performance - Python Time / Arrayfunc Time.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall	116	99	67	50	34	21	7.8	8.1	8.2	7.6	54	29
aany	53	56	28	31	16	16	3.9	3.8	3.6	3.5	25	14
abs_	1286		704		419		101		101		161	111
acos											14	13
acosh											8.3	8.5
add	1249	110	663	119	342	99	86	67	99	63	331	119
afilter	94	92	87	86	86	86	87	87	87	86	87	87
amax	75	73	148	147	114	102	15	15	15	15	87	43
amin	68	65	129	131	101	89	14	14	13	13	71	34
and_	1801	1606	1027	866	437	344	103	104	140	86		
asin											16	14
asinh											6.9	7.6
asum	5.9	7.4	7.5	9.8	8.1	8.0	8.4	6.5	5.4	7.0	30	15
atan											14	11
atan2											7.3	6.5
atanh											7.0	7.4
ceil											770	229
compress	29	36	30	38	30	19	32	22	32	21	31	34
convert	285	285	178	274	263	252	144	134	121	155	152	141
copysign											190	143
cos											24	10
cosh											13	13
count	198	223	220	217	163	112	158	108	154	110	80	79
cycle	80	80	80	85	85	56	85	57	86	55	28	28
degrees											479	156
dropwhile	111	113	110	110	110	109	110	108	108	108	116	102
eq	985	1000	475	525	264	177	63	61	62	61	265	131
erf											10	10
erfc											6.1	6.3

exp											18	17
expm1											7.1	7.6
fabs											163	116
factorial	136	195	130	135	135	136	110	101	125	101		
findindex	201	219	106	105	41	37	15	13	13	13	61	35
findindices	24	29	32	29	22	30	24	26	23	28	30	30
floor											729	248
floordiv	33	28	34	33	35	28	33	29	35	26	189	169
fma											88	83
fmod											9.2	11
gamma											1.5	1.6
ge	986	956	571	548	275	177	62	62	71	62	333	140
gt	1009	757	524	380	186	173	64	63	63	65	266	147
hypot											14	11
invert	2089	2444	1081	1389	590	554	158	180	151	181		
isfinite											107	119
isinf											117	124
isnan											142	121
ldexp											24	24
le	1048	972	528	536	188	182	71	63	62	65	263	130
lgamma											8.1	7.7
log											24	20
log10											13	12
log1p											8.4	8.5
log2											21	11
lshift	1527	1502	945	953	444	340	121	104	131	109		
lt	693	754	362	399	183	159	63	64	63	63	241	128
mod	28	26	22	28	31	22	31	22	32	25	72	67
mul	87	110	93	91	99	101	76	71	93	71	339	134
ne	1027	1020	493	536	278	186	67	68	65	64	296	138
neg	1226		538		280		93		103		139	85
or_	1588	1666	1004	855	434	342	108	83	114	99		
pow	246	282	226	229	297	264	232	196	199	185	7.9	6.9

pow2	229	228	261	294	262	208	199	190	202	206	206	116
pow3	215	208	229	222	215	193	136	125	127	119	159	141
radians											525	193
repeat	134	123	132	146	141	42	120	38	132	38	118	101
rshift	1213	1440	191	805	419	328	115	101	133	107		
sin											20	11
sinh											5.4	5.8
sqrt											233	96
sub	1072	87	594	137	348	67	104	64	91	81	343	101
takewhile	157	159	156	157	161	108	137	102	146	80	200	110
tan											8.3	5.9
tanh											7.0	6.9
truediv	53	44	53	49	52	44	53	43	51	45	134	123
trunc											730	252
xor	1604	1593	982	894	444	359	116	110	117	84		

Stat	Value
Average:	206
Maximum:	2444
Minimum:	1.5
Array size:	100000

Optimised Performance (with SIMD)

In this set of tests, all arithmetic error checking was disabled (not the default state) and SIMD acceleration was enabled (the normal default). The values here are relative to the default (see the above table), where values less than 1 are slower, and values above 1 are faster.

Floating point SIMD operations are only enabled when error checking is disabled. This data may be of some use when estimating if any useful performance gains can be made in your specific application by disabling error checking. It is not recommended to disable math error checking without good reason.

It will be noted that some integer operations which use SIMD are also slightly faster when error checking is disabled due to reduced checking overhead.

Effect of turning error checking off and leaving SIMD on for functions with both.

function	b	B	h	H	i	I	l	L	q	Q	f	d
aall												
aany												
abs_	1.2		1.4		1.4							
acos												
acosh												

add	1.3		1.3		1.1						1.1	1.1
afilter												
amax												
amin												
and_												
asin												
asinh												
asum											1.0	1.0
atan												
atan2												
atanh												
ceil											1.4	1.1
compress												
convert												
copysign												
cos												
cosh												
count												
cycle												
degrees											1.1	1.0
dropwhile												
eq												
erf												
erfc												
exp												
expm1												
fabs												
factorial												
findindex												
findindices												
floor											1.4	1.1
floordiv												
fma												
fmod												
gamma												
ge												
gt												
hypot												

invert												
isfinite												
isinf												
isnan												
ldexp												
le												
lgamma												
log												
log10												
log1p												
log2												
lshift												
lt												
mod												
mul											1.1	1.2
ne												
neg	1.3		1.3		1.3							
or_												
pow												
pow2												
pow3												
radians											1.1	1.0
repeat												
rshift												
sin												
sinh												
sqrt											1.0	1.0
sub	1.2		1.5		1.1						1.4	1.1
takewhile												
tan												
tanh												
truediv												
trunc											1.4	1.1
xor												

Stat	Value
Average:	1
Maximum:	1

Minimum:	1.0
Array size:	100000

Array Size Versus Performance

The following shoes the effects of array size on a selected arrayfunc function benchmark.

As array size increases, function call overhead decreases as a proportion of total run time.

Declines in performance when the array exceeds a certain size may be related to hardware cache effects. Arrayfunc functions together with their data may be able to reside entirely in cache, but larger arrays may require repeated cache reloads. This threshold will depend upon the particular hardware being used.

Add constant to array - times faster than Python, default settings.

Array size	b	B	h	H	i	I	l	L	q	Q	f	d
10	1.7	1.6	1.6	1.5	1.4	1.2	1.5	1.1	1.5	1.0	1.2	1.2
100	13	11	11	12	12	8.3	11	9.4	10	7.5	9.8	9.3
1000	118	60	107	59	66	43	54	40	56	42	62	54
10000	636	99	414	100	258	93	108	82	113	96	247	140
100000	1239	111	639	112	354	101	129	91	129	113	350	162
1000000	559	113	233	109	117	86	60	45	65	48	111	58
10000000	407	135	216	113	111	83	39	44	57	45	107	56

Xor an array by a constant - times faster than Python, default settings.

Array size	b	B	h	H	i	I	l	L	q	Q	f	d
10	1.8	1.7	1.6	1.7	1.5	1.3	1.5	1.3	1.6	1.3		
100	16	15	16	14	15	10	13	10	13	9.6		
1000	141	137	126	122	91	69	64	55	64	56		
10000	781	794	592	556	382	269	134	104	145	111		
100000	1611	1622	999	933	457	381	101	108	170	127		
1000000	845	821	263	280	132	105	64	51	68	55		
10000000	504	534	257	248	128	102	63	50	70	54		

Platform Effects

The platform, including CPU, OS, compiler, and compiler version can affect performance, and this influence can change significantly for different functions.

If your application requires exact performance data, then benchmark your application in the specific platform (hardware, OS, and compiler) that you will be using.

Platform support

Arrayfunc is written in 'C' and uses the standard C libraries to implement the underlying math functions. Arrayfunc has been tested on the following platforms.

OS	Hardware	Bits	Compiler	Python Version
Ubuntu 20.04 LTS	x86_64	64	GCC	3.8
Ubuntu 21.10	x86_64	64	GCC	3.9
Debian 11	i686	32	GCC	3.9
Debian 11	x86_64	64	GCC	3.9
OpenSuse 15.3	x86_64	64	GCC	3.6
Alma 8.5	x86_64	64	GCC	3.6
FreeBSD 13	x86_64	64	LLVM	3.8
OpenBSD 7.0	x86_64	64	LLVM	3.8
MS Windows 10	x86_64	64	MS VS C v.1929	3.10
MS Windows 11	x86_64	64	MS VS C v.1929	3.10
Raspberry Pi 2022-04-04	RPi 3	32	GCC	3.9
Ubuntu 20.04	RPi 4	64	GCC	3.8

- The Rasberry Pi 3 tests were conducted on a Raspberry Pi 3 ARM CPU running in 32 bit mode.
- The Ubuntu ARM tests were conducted on a Raspberry Pi 4 ARM CPU running in 64 bit mode.
- All others were conducted using VMs running on x86 hardware.