numpy.vectorize

class `numpy.` **`vectorize`**(*pyfunc*, *otypes=None*, *doc=None*, *excluded=None*, *cache=False*, *signature=None*) **[source] (http://github.com/numpy/numpy/blob/v1.14.0/numpy/lib/function_base.py#L2543-L2901)**

Generalized function class.

Define a vectorized function which takes a nested sequence of objects or numpy arrays as inputs and returns an single or tuple of numpy array as output. The vectorized function evaluates *pyfunc* over successive tuples of the input arrays like the python map function, except it uses the broadcasting rules of numpy.

The data type of the output of *vectorized* is determined by calling the function with the first element of the input. This can be avoided by specifying the *otypes* argument.

Parameters:

pyfunc *: callable*

A python function or method.

otypes *: str or list of dtypes, optional*

The output data type. It must be specified as either a string of typecode characters or a list of data type specifiers. There should be one data type specifier for each output.

doc *: str, optional*

The docstring for the function. If *None*, the docstring will be the `pyfunc.__doc__`.

excluded *: set, optional*

Set of strings or integers representing the positional or keyword arguments for which the function will not be vectorized. These will be passed directly to *pyfunc* unmodified.
New in version 1.7.0.

cache *: bool, optional*

If *True*, then cache the first function call that determines the number of outputs if *otypes* is not provided.

New in version 1.7.0.

signature *: string, optional*

Generalized universal function signature, e.g., `(m,n),(n)->(m)` for vectorized matrix-vector multiplication. If provided, `pyfunc` will be called with (and expected to return) arrays with shapes given by the size of corresponding core dimensions. By default, `pyfunc` is assumed to take scalars as input and output.
New in version 1.12.0.

Returns: **vectorized** *: callable*

Vectorized function.

See also:

`frompyfunc` **(numpy.frompyfunc.html#numpy.frompyfunc)**	Takes an arbitrary Python function and returns a ufunc

Notes

The `vectorize` function is provided primarily for convenience, not for performance. The implementation is essentially a for loop.

If *otypes* is not specified, then a call to the function with the first argument will be used to determine the number of outputs. The results of this call will be cached if *cache* is *True* to prevent calling the function twice. However, to implement the cache, the original function must be wrapped which will slow down subsequent calls, so only do this if your function is expensive.

The new keyword argument interface and *excluded* argument support further degrades performance.

References

[R293] NumPy Reference, section Generalized Universal Function API (http://docs.scipy.org/doc/numpy/reference/c-api.generalized-ufuncs.html).

Examples

```
>>> def myfunc(a, b):
...     "Return a-b if a>b, otherwise return a+b"
...     if a > b:
...         return a - b
...     else:
...         return a + b


>>> vfunc = np.vectorize(myfunc)
>>> vfunc([1, 2, 3, 4], 2)
array([3, 4, 1, 2])
```

The docstring is taken from the input function to `vectorize` unless it is specified:

```
>>> vfunc.__doc__
'Return a-b if a>b, otherwise return a+b'
>>> vfunc = np.vectorize(myfunc, doc='Vectorized `myfunc`')
>>> vfunc.__doc__
'Vectorized `myfunc`'
```

The output type is determined by evaluating the first element of the input, unless it is specified:

```
>>> out = vfunc([1, 2, 3, 4], 2)
>>> type(out[0])
<type 'numpy.int32'>
>>> vfunc = np.vectorize(myfunc, otypes=[float])
>>> out = vfunc([1, 2, 3, 4], 2)
>>> type(out[0])
<type 'numpy.float64'>
```

The *excluded* argument can be used to prevent vectorizing over certain arguments. This can be useful for array-like arguments of a fixed length such as the coefficients for a polynomial as in `polyval` (numpy.polyval.html#numpy.polyval):

```
>>> def mypolyval(p, x):
...     _p = list(p)
...     res = _p.pop(0)
...     while _p:
...         res = res*x + _p.pop(0)
...     return res
>>> vpolyval = np.vectorize(mypolyval, excluded=['p'])
>>> vpolyval(p=[1, 2, 3], x=[0, 1])
array([3, 6])
```

Positional arguments may also be excluded by specifying their position:

```
>>> vpolyval.excluded.add(0)
>>> vpolyval([1, 2, 3], x=[0, 1])
array([3, 6])
```

The *signature* argument allows for vectorizing functions that act on non-scalar arrays of fixed length. For example, you can use it for a vectorized calculation of Pearson correlation coefficient and its p-value:

```
>>> import scipy.stats
>>> pearsonr = np.vectorize(scipy.stats.pearsonr,
...                         signature='(n),(n)->(),()')
>>> pearsonr([[0, 1, 2, 3]], [[1, 2, 3, 4], [4, 3, 2, 1]])
(array([ 1., -1.]), array([ 0.,  0.]))
```

Or for a vectorized convolution:

```
>>> convolve = np.vectorize(np.convolve, signature='(n),(m)->(k)')
>>> convolve(np.eye(4), [1, 2, 1])
array([[ 1.,  2.,  1.,  0.,  0.,  0.],
       [ 0.,  1.,  2.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  2.,  1.,  0.],
       [ 0.,  0.,  0.,  1.,  2.,  1.]])
```

Methods

`__call__` (numpy.vectorize.__call__.html#numpy.vectorize.__call__)(*args, **kwargs)

Return arrays with the results of *pyfunc* broadcast (vectorized) over `args` and *kwargs* not in *excluded*.

Previous topic

numpy.apply_over_axes (numpy.apply_over_axes.html)

Next topic

numpy.vectorize.__call__ (numpy.vectorize.__call__.html)