

---

# **xarray***extrasDocumentation*

***Release 0.2.0***

**xarray***extrasDevelopers*

**2018-07-15**



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>1</b>
1.1	Required dependencies . . . . .	1
1.2	Testing . . . . .	1
<b>2</b>	<b>What's New</b>	<b>3</b>
2.1	v0.2.0 (2018-07-15) . . . . .	3
2.2	v0.1.0 (2018-05-19) . . . . .	3
<b>3</b>	<b>API Reference</b>	<b>5</b>
3.1	csv . . . . .	5
3.2	cumulatives . . . . .	6
3.3	interpolate . . . . .	6
3.4	numba_extras . . . . .	7
3.5	sort . . . . .	8
<b>4</b>	<b>License</b>	<b>11</b>
	<b>Python Module Index</b>	<b>13</b>



### 1.1 Required dependencies

- Python 3.5 or 3.6
- `scipy`
- `xarray`
- `dask`
- `numba`

### 1.2 Testing

To run the test suite after installing `xarray_extras`, first install (via pypi or conda)

- `py.test`: Simple unit testing library

and run `py.test --pyargs xarray_extras`.



### 2.1 v0.2.0 (2018-07-15)

- New function `xarray_extras.csv.to_csv()`
- Speed up interpolation for `k=2` and `k=3`
- CI: Rigorous tracking of minimum dependency versions
- CI: Explicit support for Python 3.7

### 2.2 v0.1.0 (2018-05-19)

Initial release.





## 3.1 csv

CSV file type support

`xarray_extras.csv.to_csv(x, path_or_buf, **kwargs)`  
Print DataArray to CSV.

When `x` has numpy backend, this function is equivalent to:

```
x.to_pandas().to_csv(path_or_buf, **kwargs)
```

When `x` has dask backend, this function returns a dask delayed object which will write to the disk only when its `.compute()` method is invoked.

Formatting and optional compression are parallelised across all available CPUs, using one dask task per chunk on the first dimension. Chunks on other dimensions will be merged ahead of computation.

### Parameters

- **x** – `xarray.DataArray` with one or two dimensions
- **path\_or\_buf** – File path or file-like object
- **kwargs** – Passed verbatim to `pandas.DataFrame.to_csv()` or `pandas.Series.to_csv()`

### Limitations

- When `x` has dask backend, `path_or_buf` must be a file path. Fancy URIs are not (yet) supported.
- When `x` has dask backend, `compression='zip'` is not supported. All other compression methods (gzip, bz2, xz) are supported.

### Distributed

This function supports `dask distributed`, with the caveat that all workers must write to the same shared mount-point and that the shared filesystem must strictly guarantee **close-open coherency**, meaning that one must be

able to call `write()` and then `close()` on a file descriptor from one host and then immediately afterwards `open()` from another host and see the output from the first host. Note that, for performance reasons, most network filesystems do not enable this feature by default.

Alternatively, one may write to local mountpoints and then manually collect and concatenate the partial outputs.

## 3.2 cumulatives

## 3.3 interpolate

xarray interpolation functions

`xarray_extras.interpolate.splrep(a, dim, k=3)`

Calculate the univariate B-spline for an N-dimensional array

### Parameters

- **a** (`xarray.DataArray`) – any `DataArray`
- **dim** – dimension of `a` to be interpolated. `a.coords[dim]` must be strictly monotonic ascending. All int, float (not complex), or datetime dtypes are supported.
- **k** (`int`) – B-spline order:

k	interpolation kind
0	nearest neighbour
1	linear
2	quadratic
3	cubic

**Returns** `Dataset` with `t`, `c`, `k` (knots, coefficients, order) variables, the same shape and coords as the input, that can be passed to `splev()`.

Example:

```
>>> x = np.arange(0, 120, 20)
>>> x = xarray.DataArray(x, dims=['x'], coords={'x': x})
>>> s = xarray.DataArray(np.linspace(1, 20, 5), dims=['s'])
>>> y = np.exp(-x / s)
>>> x_new = np.arange(0, 120, 1)
>>> tck = splrep(y, 'x')
>>> y_new = splev(x_new, tck)
```

### Features

- Interpolate a ND array on any arbitrary dimension
- dask supported on both on the interpolated array and `x_new`
- Supports ND `x_new` arrays
- The CPU-heavy interpolator generation (`splrep()`) is executed only once and then can be applied to multiple `x_new` (`splev()`)
- memory-efficient
- Can be pickled and used on dask distributed

### Limitations

- Chunks are not supported along dim on the interpolated dimension.

`xarray_extras.interpolate.splev(x_new, tck, extrapolate=True)`  
Evaluate the B-spline generated with `splrep()`.

#### Parameters

- **x\_new** – Any `DataArray` with any number of dims, not necessarily the original interpolation dim. Alternatively, it can be any 1-dimensional array-like; it will be automatically converted to a `DataArray` on the interpolation dim.
- **tck** (`xarray.Dataset`) – As returned by `splrep()`. It can have been:
  - transposed (not recommended, as performance will drop if c is not C-contiguous)
  - sliced, reordered, or (re)chunked, on any dim except the interpolation dim
  - computed from dask to numpy backend
  - round-tripped to disk
- **extrapolate** –
  - True** Extrapolate the first and last polynomial pieces of b-spline functions active on the base interval
  - False** Return NaNs outside of the base interval
  - 'periodic'** Periodic extrapolation is used
  - 'clip'** Return `y[0]` and `y[-1]` outside of the base interval

**Returns** `DataArray` with all dims of the interpolated array, minus the interpolation dim, plus all dims of `x_new`

See `splrep()` for usage example.

## 3.4 numba\_extras

Extensions to numba

`xarray_extras.numba_extras.guvectorize(signature, layout, **kws)`

Convenience wrapper around `numba.guvectorize()`. Generate signature for all possible data types and set a few healthy defaults.

#### Parameters

- **signature** (`str`) – numba signature, containing `{T}`
- **layout** (`str`) – as in `numba.guvectorize()`
- **kws** – passed verbatim to `numba.guvectorize()`. This function changes the default for `cache` from `False` to `True`.

example:

```
guvectorize("{T}[:, :], {T}[:, :]", "(i)->(i)")
```

Is the same as:

```
numba.guvectorize([
    "float32[:, float32[:,",
    "float64[:, float64[:,",
    ...
], "(i)->(i)", cache=True)
```

---

**Note:** Discussing upstream fix; see <https://github.com/numba/numba/issues/2936>.

---

## 3.5 sort

Sorting functions

`xarray_extras.sort.topk(a, k, dim, split_every=None)`

Extract the *k* largest elements from *a* on the given dimension, and return them sorted from largest to smallest. If *k* is negative, extract the *-k* smallest elements instead, and return them sorted from smallest to largest.

This assumes that *k* is small. All results will be returned in a single chunk along the given axis.

`xarray_extras.sort.argtopk(a, k, dim, split_every=None)`

Extract the indexes of the *k* largest elements from *a* on the given dimension, and return them sorted from largest to smallest. If *k* is negative, extract the *-k* smallest elements instead, and return them sorted from smallest to largest.

This assumes that *k* is small. All results will be returned in a single chunk along the given axis.

`xarray_extras.sort.take_along_dim(a, ind, dim)`

Use the output of `argtopk()` to pick points from *a*.

### Parameters

- **a** – any xarray object
- **ind** – array of ints, as returned by `argtopk()`
- **dim** – dimension along which `argtopk` was executed

An example that uses all of the above functions is *source attribution*. Given a generic function  $y = f(x_0, x_1, \dots, x_i)$ , which is embarrassingly parallel along a given dimension, one wants to find:

- the top *k* elements of *y* along the dimension
- the elements of all *x*'s that generated the top *k* elements of *y*

```
>>> from xarray import DataArray
>>> from xarray_extras.sort import *
>>> x = DataArray([[5, 3, 2, 8, 1],
>>>                [0, 7, 1, 3, 2]], dims=['x', 's'])
>>> y = x.sum('x') # y = f(x), embarrassingly parallel among dimension 's'
>>> y
<xarray.DataArray (s: 5)>
array([ 5, 10,  3, 11,  3])
Dimensions without coordinates: s
>>> top_y = topk(y, 3, 's')
>>> top_y
<xarray.DataArray (s: 3)>
array([11, 10,  5])
Dimensions without coordinates: s
```

(continues on next page)

(continued from previous page)

```
>>> top_x = take_along_dim(x, argtopk(y, 3, 's'), 's')
>>> top_x
<xarray.DataArray (x: 2, s: 3)>
array([[8, 3, 5],
       [3, 7, 0]])
Dimensions without coordinates: x, s
```



## CHAPTER 4

---

### License

---

xarray\_extras is available under the open source [LGPL License](#).





### X

- `xarray_extras.csv`, 5
- `xarray_extras.interpolate`, 6
- `xarray_extras.numba_extras`, 7
- `xarray_extras.sort`, 8



### A

`argtopk()` (in module `xarray_extras.sort`), 8

### G

`guvectorize()` (in module `xarray_extras.numba_extras`), 7

### S

`splev()` (in module `xarray_extras.interpolate`), 7

`splrep()` (in module `xarray_extras.interpolate`), 6

### T

`take_along_dim()` (in module `xarray_extras.sort`), 8

`to_csv()` (in module `xarray_extras.csv`), 5

`topk()` (in module `xarray_extras.sort`), 8

### X

`xarray_extras.csv` (module), 5

`xarray_extras.interpolate` (module), 6

`xarray_extras.numba_extras` (module), 7

`xarray_extras.sort` (module), 8