xr-scipy Documentation

Release 0.0.1+dev.8ff001d

xr-scipy Developers

2019-11-01

EXAMPLES

xr-scipy is a thin wrapper of scipy for xarray eco-system.

Many scipy functions, such as *scipy.integrate.trapz*, requires coordinate array as an argument. Since xarray objects possess their coordinate values in it, there must be simpler api for such functions.

xr-scipy wraps them to use the native coordinate objects of xarray and returns an xarray object with the computed data. This enables more xarray-oriented data analysis.

CHAPTER

ONE

DOCUMENTATION

Examples

- *Gradient and Integration*
- *Fourier Transform*
- Interpolation (scipy.interpolate)
- *Digital filters*
- *Spectral (FFT) analysis*

1.1 Gradient and Integration

xr-scipy wraps `scipy.gradient` and some of `scipy.integrate` functions. Let's create a simple example DataArray:

```
In [1]: arr = xr.DataArray(np.sin(np.linspace(0, 6.28, 30)) ** 2,
   ...:                    dims=('x'), coords={'x': np.linspace(0, 5, 30)})
   ...:

In [2]: arr
Out[2]:
<xarray.DataArray (x: 30)>
array([0.00000000e+00, 4.61661813e-02, 1.76139460e-01, 3.65918356e-01,
       5.80457402e-01, 7.80138804e-01, 9.28088451e-01, 9.96985262e-01,
       9.74106426e-01, 8.63676857e-01, 6.86089001e-01, 4.74137072e-01,
       2.66961112e-01, 1.02819215e-01, 1.20225984e-02, 1.13381944e-02,
       1.00892388e-01, 2.64147680e-01, 4.70956575e-01, 6.83128766e-01,
       8.61483534e-01, 9.73085045e-01, 9.97324436e-01, 9.29725546e-01,
       7.82771507e-01, 5.83599545e-01, 3.68989697e-01, 1.78572829e-01,
       4.75122222e-02, 1.01461475e-05])
Coordinates:
  * x        (x) float64 0.0 0.1724 0.3448 0.5172 ... 4.483 4.655 4.828 5.0
```

Our *`gradient()`* takes an xarray object (possibly high dimensional) and a coordinate name which direction we compute the gradient of the array,

```
In [3]: grad = xrscipy.gradient(arr, 'x')

In [4]: grad
Out[4]:
<xarray.DataArray (x: 30)>
```

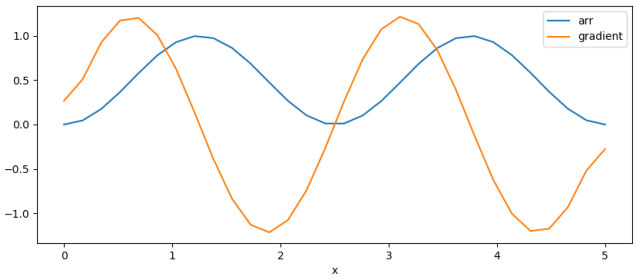
(continues on next page)

(continued from previous page)

```
array([ 0.26776385,  0.51080443,  0.92728131,  1.17252203,  1.2012393 ,
        1.00813004,  0.62885473,  0.13345213, -0.38659437, -0.83525053,
       -1.12966538, -1.21547088, -1.07682178, -0.73932169, -0.26529496,
        0.25772239,  0.73314751,  1.07318614,  1.21504515,  1.13252818,
        0.84087321,  0.39393861, -0.12574255, -0.62220349, -1.0037654 ,
       -1.19996725, -1.17457748, -0.93228468, -0.51783178, -0.27551204])
Coordinates:
  * x        (x) float64 0.0 0.1724 0.3448 0.5172 ... 4.483 4.655 4.828 5.0
```

The return type is also a DataArray with coordinates.

```
In [5]: arr.plot(label='arr')
Out[5]: [<matplotlib.lines.Line2D at 0x7f04e702b190>]

In [6]: grad.plot(label='gradient')
Out[6]: [<matplotlib.lines.Line2D at 0x7f04e6b8b490>]

In [7]: plt.legend()
Out[7]: <matplotlib.legend.Legend at 0x7f04e6ab9150>

In [8]: plt.show()
```

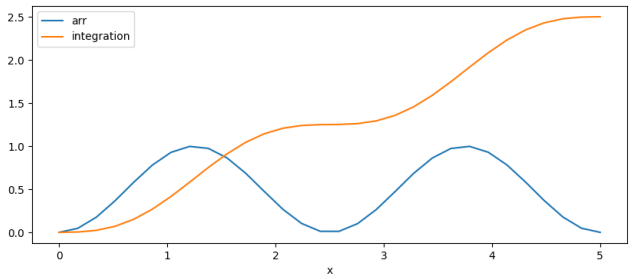


The other options (edge_order) are the same to `numpy.gradient`. See *`gradient()`*.

Similar to *`gradient()`*, xr-scipy wraps some functions in `scipy.integrate` module. Our integration function also takes an xarray object and coordinate name along which the array to be integrated. The return type is also a DataArray,

```
# trapz computes definite integration
In [9]: xrscipy.integrate.trapz(arr, coord='x')
Out[9]:
<xarray.DataArray ()>
array(2.50124814)

# cumurative integration returns a same shaped array
In [10]: integ = xrscipy.integrate.cumtrapz(arr, 'x')

In [11]: integ
Out[11]:
<xarray.DataArray (x: 30)>
array([0.        , 0.00397984, 0.02314412, 0.06987324, 0.15145736,
       0.26875014, 0.41601111, 0.58196574, 0.75188744, 0.91031703,
       1.04391753, 1.14393702, 1.2078248 , 1.23970241, 1.24960257,
       1.25161643, 1.26129148, 1.29276045, 1.35613151, 1.45562162,
       1.58877786, 1.74693032, 1.91679321, 2.08291821, 2.23054726,
```

```
       2.34833787, 2.43045763, 2.4776613 , 2.49715139, 2.50124814])
Coordinates:
  * x        (x) float64 0.0 0.1724 0.3448 0.5172 ... 4.483 4.655 4.828 5.0

In [12]: arr.plot(label='arr')
Out[12]: [<matplotlib.lines.Line2D at 0x7f04e6d95e90>]

In [13]: integ.plot(label='integration')
Out[13]: [<matplotlib.lines.Line2D at 0x7f04e6b81b90>]

In [14]: plt.legend()
Out[14]: <matplotlib.legend.Legend at 0x7f04e6c91950>

In [15]: plt.show()
```



See *trapz()* for other options.

Note: There are slight difference from the original implementations. Our *gradient()* does not accept multiple coordinates. Our *cumtrapz()* always assume `initial=0`.

1.2 Fourier Transform

xr-scipy wraps `numpy.fft`, for more convenient data analysis with xarray. Let us consider an example DataArray

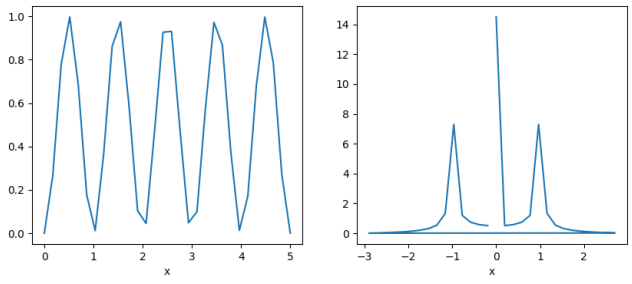
```
In [1]: arr = xr.DataArray(np.sin(np.linspace(0, 15.7, 30)) ** 2,
   ...:                    dims=('x'), coords={'x': np.linspace(0, 5, 30)})
   ...:

In [2]: arr
Out[2]:
<xarray.DataArray (x: 30)>
array([0.00000000e+00, 2.65553207e-01, 7.80138804e-01, 9.97157371e-01,
       6.86089001e-01, 1.77354508e-01, 1.13381944e-02, 3.64384719e-01,
       8.61483534e-01, 9.74609903e-01, 5.83599545e-01, 1.03788678e-01,
       4.48385590e-02, 4.69366760e-01, 9.26433982e-01, 9.30537558e-01,
       4.77318618e-01, 4.81921327e-02, 9.89817591e-02, 5.75738105e-01,
       9.72044464e-01, 8.66939139e-01, 3.72066354e-01, 1.30863288e-02,
       1.71312250e-01, 6.78674515e-01, 9.96246419e-01, 7.86699011e-01,
       2.72616236e-01, 6.34122960e-05])
Coordinates:
  * x        (x) float64 0.0 0.1724 0.3448 0.5172 ... 4.483 4.655 4.828 5.0
```

Our *fft()* takes an xarray object (possibly high dimensional) and a coordinate name which direction we compute the Fourier transform.

```
In [3]: fft = xrscipy.fft.fft(arr, 'x')

In [4]: fft
Out[4]:
<xarray.DataArray (x: 30)>
array([ 1.45066531e+01+0.j        , -5.08969487e-01-0.05277866j,
       -5.64134191e-01-0.11825163j, -6.97777394e-01-0.22340666j,
       -1.09493558e+00-0.47978099j, -6.34538952e+00-3.59910605j,
        1.08016447e+00+0.76901603j,  4.01404487e-01+0.35285112j,
        2.07930614e-01+0.22420945j,  1.19289778e-01+0.15804858j,
        7.02940829e-02+0.11554961j,  4.06345993e-02+0.08441817j,
        2.20354770e-02+0.05946099j,  1.05946593e-02+0.03803481j,
        4.35028404e-03+0.01857428j,  2.36237341e-03+0.j        ,
        4.35028404e-03-0.01857428j,  1.05946593e-02-0.03803481j,
        2.20354770e-02-0.05946099j,  4.06345993e-02-0.08441817j,
        7.02940829e-02-0.11554961j,  1.19289778e-01-0.15804858j,
        2.07930614e-01-0.22420945j,  4.01404487e-01-0.35285112j,
        1.08016447e+00-0.76901603j, -6.34538952e+00+3.59910605j,
       -1.09493558e+00+0.47978099j, -6.97777394e-01+0.22340666j,
       -5.64134191e-01+0.11825163j, -5.08969487e-01+0.05277866j])
Coordinates:
  * x        (x) float64 0.0 0.1933 0.3867 0.58 ... -0.58 -0.3867 -0.1933
```

The coordinate *x* is also converted to frequency.

```
In [5]: plt.figure(figsize=(10, 4))
Out[5]: <Figure size 1000x400 with 0 Axes>

In [6]: plt.subplot(1, 2, 1)
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f04e79ce450>

In [7]: arr.plot()
Out[7]: [<matplotlib.lines.Line2D at 0x7f04e7d9a4d0>]

In [8]: plt.subplot(1, 2, 2)
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f04e76daa50>

In [9]: np.abs(fft).plot()
Out[9]: [<matplotlib.lines.Line2D at 0x7f04e76adbd0>]

In [10]: plt.show()
```



Note: The coordinate values must be evenly spaced for FFT.

1.3 Multidimensional Fourier transform

xr-scipy also wraps the multidimensional Fourier transform, such as *rfftn()*

Their usage is very similar to the above, where we just need to specify coordinates.

```
In [11]: arr = xr.DataArray(np.random.randn(30, 20) ** 2,
   ....:                    dims=('x', 'y'),
   ....:                    coords={'x': np.linspace(0, 5, 30),
   ....:                            'y': np.linspace(0, 5, 20)})
   ....:

In [12]: fftn = xrscipy.fft.fftn(arr, 'x', 'y')

In [13]: fftn
Out[13]:
<xarray.DataArray (x: 30, y: 20)>
array([[ 6.80147367e+02+0.00000000e+00j, -5.40888999e+01-1.48914052e+01j,
         2.24071878e+01-6.83472672e+00j, -1.05248506e+01-1.98086495e+01j,
         3.30186673e+01+2.47699470e+01j, -4.10255035e+01+1.25478518e+01j,
         2.07742440e+01-2.10698675e+01j,  1.12280731e+01+8.16514918e+00j,
        -6.15766306e+01+1.53389009e+01j,  1.68000520e+01+3.08873987e+01j,
         6.91096924e+01+0.00000000e+00j,  1.68000520e+01-3.08873987e+01j,
        -6.15766306e+01-1.53389009e+01j,  1.12280731e+01-8.16514918e+00j,
         2.07742440e+01+2.10698675e+01j, -4.10255035e+01-1.25478518e+01j,
         3.30186673e+01-2.47699470e+01j, -1.05248506e+01+1.98086495e+01j,
         2.24071878e+01+6.83472672e+00j, -5.40888999e+01+1.48914052e+01j],
       [-9.78304942e+00-2.00060691e+01j, -3.01214418e+01-6.86851691e-01j,
         3.65704485e+01+2.64031486e+01j, -1.65402379e+01+5.18006959e+00j,
         5.75308280e+00+2.73910712e+01j, -2.48612360e+01-1.19071034e+01j,
        -1.90295095e+01+5.25959400e+01j, -7.66929151e+00-4.16255367e+00j,
        -3.41801537e+01+4.29905149e+01j,  1.23190320e+01+1.63252604e+01j,
        -4.84907294e+01-4.16601947e+01j,  1.07657876e+01-1.41656999e+01j,
        -3.39807068e+01+9.65027977e+00j,  5.89818080e+01-1.58963695e+00j,
        -2.24569277e+01+4.78807494e+00j,  2.29408613e+00-2.10730557e+01j,
         1.20560230e+00+2.45535230e+01j, -2.64074189e+01-1.52314832e+01j,
        -3.75456761e+00+6.28466305e+01j,  1.28963968e+01-3.01513717e+01j],
       [ 1.18390109e+01+1.69615481e+01j, -1.03891185e+01+3.07147965e+01j,
         3.48168720e+00+3.93232365e+01j,  1.51573868e+01+8.87107641e-02j,
        -3.26843493e+01+2.78910788e+01j, -7.38998869e+00+5.92937093e+00j,
         1.99909029e+01-2.37322857e+01j,  6.89571773e+00-7.65203935e+00j,
        -2.41167969e+01+8.25647322e+00j,  2.60820207e+01-1.08850196e+01j,
        -2.91364141e+01-4.57237154e+01j,  4.48110583e+01+2.09499730e+00j,
        -5.77461281e+00-2.23819916e+01j,  2.27092738e+01+1.09230345e+01j,
        -3.63069591e+01+2.63605337e+01j,  1.49904457e+01+1.62863784e+00j,
         5.18844566e+01-3.92437600e+01j, -6.17481319e+00+1.43282178e+01j,
         8.93965329e+00+1.28033963e+01j,  2.23641363e+01-2.98647685e-02j],
       [-3.00756470e+01-2.02311653e+01j, -1.52199079e+01+6.12960304e+01j,
         4.71762598e+01-8.72739590e+00j,  3.95568373e+01-8.28074518e+00j,
         2.05435690e+00+1.04214009e+01j,  1.26375702e+00+5.25047677e+00j,
        -2.30835820e+01+2.65438675e+01j, -2.16237997e+01-1.94507161e+01j,
         1.62248018e+01+4.73648432e+00j,  3.91626381e+00-1.85662042e+00j,
```

(continues on next page)

(continued from previous page)

```
        -5.02314992e+01-1.11478363e+01j,  1.44147644e+01+2.88680317e+01j,
         4.52215055e+01-3.12928417e+01j, -1.10414273e+01-7.78928074e+00j,
         1.17154656e+01-3.47452516e+01j, -2.36212076e+01+5.93467176e+01j,
        -1.80884532e+00-3.35852750e+01j,  3.10571817e+00+4.66607518e+01j,
         2.03781124e+01-1.49895411e+01j, -2.13363900e+01+1.58769170e+01j],
       [ 7.15150886e+00+1.25220177e+01j, -1.64656200e+01-8.80436886e+00j,
         2.35781124e+01-3.03692665e+01j, -2.21190855e+01-4.34572687e-02j,
         1.81568112e+01+1.23120934e+01j,  4.94153737e+01-9.78063668e+00j,
        -7.49789987e+00+1.57377970e+01j, -2.29754969e+01-1.78710672e+01j,
        -1.92307775e+00-2.94975177e+01j, -3.11790029e+01-2.11839622e+01j,
         3.04358949e+01+6.36528460e+01j,  5.50489436e+00-1.22197283e+01j,
         4.22534702e+01-1.06789983e+01j, -7.06272713e+01-1.11875490e+01j,
         3.27265622e+01+2.80609377e+00j, -3.95286687e+01-9.60118722e+00j,
         5.75318857e+00-2.22803596e+00j, -4.14294443e+01-1.48474337e+01j,
         3.23407481e+01-6.68717454e+01j, -3.92473335e+01+2.68505554e+01j],
       [ 1.06523934e+01+1.73673192e+01j, -1.95164660e+01-7.43979407e+01j,
        -4.93971463e+01+2.09266660e+01j,  2.60177333e+01+1.53278026e+01j,
        -3.60650217e+01+6.05873460e-01j,  8.12464287e+00+2.22765043e+00j,
        -1.40264564e+01+4.29802397e+01j,  4.70004035e+01-1.22464039e+01j,
        -2.85868251e+01+3.59036353e+01j, -3.44036489e+01-3.06603442e+01j,
         4.30603867e+00+2.01927339e+00j, -1.20910217e+01-4.36106228e+01j,
        -3.22237081e+01+2.25641449e+00j,  2.88057312e+01+2.37735569e+00j,
         3.53326051e+01+1.81494418e+01j, -2.47154470e+00-2.07686708e+01j,
         2.58992505e+00+2.03285531e+01j, -2.42399929e+00+3.57250326e+01j,
         2.18422788e+01+3.21744430e+00j, -1.65841241e+01-2.38798773e+01j],
       [-1.07818296e+00-1.47849666e+01j,  4.47886848e+01+2.34594148e+01j,
         3.79211155e-01+1.61643017e+01j, -4.39314574e+01-2.09381347e+01j,
        -1.52437785e+01-8.43433973e+00j,  2.35262485e+01+4.98147526e+01j,
        -6.11206007e+00+2.25673610e+01j,  3.30809693e+01-9.34850620e+00j,
         1.71673780e+01-3.47696707e+01j,  3.68696602e+01+3.59515289e+01j,
         1.65210470e+01-7.33123761e+01j,  1.02898229e+01-1.29941840e+01j,
        -2.82417266e+01-3.39720714e+01j,  3.41384511e+00+2.09842327e+01j,
        -3.62093944e+01-8.93102919e+01j, -1.47845148e+01-2.05218191e+01j,
         4.05592205e+00-2.11454331e+01j,  3.89740324e+01-3.17500007e+01j,
         6.45552327e+00+2.69100506e+01j,  5.70284748e+01+3.43547254e+01j],
       [-5.58110611e+00+4.40670582e+01j, -2.94205956e+00-4.12300389e+01j,
        -2.31106044e+01-1.65055492e+00j,  4.56543207e+01-1.42979937e+01j,
        -2.17746984e+01+7.18592405e+01j, -4.78639712e+01+4.14072434e-01j,
         1.98169103e+00-7.22053259e+00j, -4.08850966e+01-2.60055911e+00j,
        -6.06693803e+00+2.07656749e+01j,  5.41040083e-01-1.81124148e+01j,
         1.80817697e+01-2.95532249e+01j, -2.25934944e+01+2.84674795e+01j,
         2.43658059e+01-2.55621553e+01j,  7.95868211e+00+5.65582314e+01j,
        -1.43962466e+01+2.65526852e+01j, -1.73257338e+01+5.30225228e+01j,
         4.85320648e+01+5.60596237e+00j, -3.19274861e+01+7.80041966e+00j,
        -2.34627949e+00-8.89046635e+00j,  1.54083249e+01+2.00649186e+01j],
       [ 1.18824392e+01-3.22445034e+01j, -1.90338418e+01+1.51946428e+01j,
        -5.37257684e+00+4.98413122e+01j, -9.94112937e+00-8.26233410e+01j,
        -2.64544431e+01+3.55722370e+01j, -8.81439084e-01-1.06270574e+01j,
        -2.69432497e+01-3.13403552e+00j, -1.79272987e+01-5.37861863e+01j,
        -1.16715093e+01+2.20421542e+01j,  2.45644090e+01+3.12855452e+01j,
         1.06994721e+01-5.91144258e+01j, -1.83299526e+01+1.41597367e+01j,
         3.02689172e+01-1.20605584e+00j, -1.71615097e+01-2.65530672e+01j,
         7.12788455e+00-1.01466482e+01j,  8.95293832e+00+3.02611311e+01j,
         4.82976884e+01-5.07935313e+00j,  1.56397075e+00-5.98568234e+00j,
        -1.69562981e+01+2.69981446e+01j, -1.63233034e+01+3.07002636e+00j],
       [-3.07316501e+01+3.93892519e+01j,  4.37522412e+01-2.58679396e+01j,
         8.14578399e+00+3.67631795e+01j, -3.66511012e+00-1.63464859e+01j,
```

(continues on next page)

(continued from previous page)

```
       -2.58192580e+01-6.89101643e+00j,  4.91499643e+01+1.65418913e+01j,
       -5.77409216e+01+1.60540597e+01j, -6.73505251e+00+1.98416837e+00j,
        2.35079633e+01+7.50280198e+00j, -1.39586815e+01-1.93893701e+01j,
        3.33511180e+00+1.94216403e+01j,  6.37599026e+01-4.38710211e+00j,
        4.12779068e+00+2.93916104e+00j, -2.00550737e+01-9.79395414e+00j,
        2.26950448e+01+1.38349923e+01j,  1.54237181e+01-8.05481446e+00j,
       -4.53686973e+00+1.45217067e+01j, -5.63117259e+01+1.11153064e+01j,
       -1.81651373e+01+3.26587070e+01j, -8.70958377e-01+9.02082150e-01j],
      [-1.70526169e+00+1.92163372e+01j,  2.97034824e+01-2.79547749e+01j,
       -1.85369575e+01-2.35003882e+01j, -6.64633008e+00-4.38510142e+00j,
        4.97460634e+01-2.32829438e+01j, -4.05256849e+01-1.34379916e+01j,
       -2.21996880e+01-5.72145390e+01j,  7.13148018e+01+3.63742781e+01j,
       -5.86853908e+01-4.98184864e+00j,  1.30386761e+01+1.50867127e+01j,
       -1.79552477e+01+1.19034968e+00j,  3.70133743e+01+5.49322284e+01j,
       -1.89551153e+01-2.96675488e+00j,  1.69927163e+01-5.46139595e+00j,
       -2.67284300e+00-3.00440969e+01j, -2.74936090e+01+1.67932310e+01j,
       -4.84984905e+01+4.44450945e+00j, -3.25168875e+01+3.05116544e+01j,
        2.77952349e+01+7.37147443e+00j, -4.67295936e+01-1.68476993e+01j],
      [ 3.20962594e-01-5.06835211e+00j,  2.19888129e+00-3.84687654e+01j,
        6.29856157e+01+5.11739533e+00j, -1.24628564e+01+1.01651712e+01j,
       -1.55673493e+01+2.14876071e+01j, -2.42030910e+00+7.35438393e+01j,
        3.15126654e+01-1.06852728e+01j, -1.26190710e+01+5.65750088e+00j,
        4.85206624e+01-5.74968092e+01j, -3.06979577e+01-1.49209380e+01j,
       -5.81901559e+01-1.92046694e+01j,  9.83142621e+00+3.45841898e+00j,
        1.65440668e+01+1.20444372e+01j, -3.08074399e+01+1.70928994e+01j,
        2.41874283e+00+8.27550100e+00j, -5.74958635e+00+5.93517320e+01j,
       -2.22563818e+01+4.74525652e+01j,  2.22153223e+01-4.70607883e+01j,
       -1.80254890e+01+1.68658665e+01j, -3.79819554e+01-1.09610326e+01j],
      [-5.53260189e-01-3.33642332e+00j, -3.36865318e+01-5.67699005e+01j,
       -2.61418425e+01+6.74386221e+00j, -8.56095964e+00-8.15590602e+00j,
        9.83735030e+00-3.69410893e+00j,  3.95641923e+01-1.82508937e+01j,
        4.82363622e+01+6.80687760e+01j,  5.85350225e+00-1.55203019e+01j,
       -4.18423517e+01+7.67380731e+00j,  3.72751167e+00-4.04047009e+00j,
        3.52998209e+01-9.15896876e+00j, -5.08422850e+01+7.73216775e+00j,
       -5.75851523e+01-1.61377363e+01j,  7.72403168e+00-2.65696480e+01j,
        2.33187751e+00+1.53364881e+01j,  4.29016562e+01-2.77208110e+01j,
        2.40399719e+01+1.60354310e+01j,  3.99123661e+01-1.40791608e+01j,
       -7.79934151e+00+3.38929659e+01j,  5.17194527e+01+5.66418081e-01j],
      [-2.42445557e+00-1.52927059e+01j, -1.74865873e+01+3.95124060e+01j,
        1.50253414e+01+4.68022072e+01j,  4.83083325e+01+3.88306999e+01j,
       -1.93474045e+01+3.09792374e+01j, -9.14912144e+00-2.14527482e+00j,
       -1.67661069e+00-1.34034137e+01j,  1.10542525e+01-4.34099936e+00j,
        2.51257491e-01+4.18618670e+00j,  3.64910931e+00+3.90753262e-01j,
        4.15379418e+01+1.21990746e+00j, -2.57463428e+01+1.07618398e+01j,
       -7.84762169e+00+3.58774714e+01j,  2.23354935e+01-1.18006751e+01j,
        1.22580500e+01-3.25180329e+00j,  1.32792638e+01+8.14105169e+00j,
       -1.29989388e+01-9.99264531e+00j,  8.74145820e-01-2.16671479e+00j,
       -1.13623186e+01+1.46572477e+01j,  2.06068442e+01-4.91914400e+00j],
      [-5.24122435e+01-2.57484452e+01j,  7.88456072e+00+1.36192049e+01j,
       -4.69476119e+01-4.19065872e+01j, -2.51337731e+01+5.04668321e+01j,
       -5.27983284e+01-2.20977238e+01j,  2.83922223e+01-1.59345822e+01j,
        3.99972852e+00-1.14365015e+01j, -6.67169831e+01+6.45536117e+01j,
       -1.85524904e+01+2.25721542e+01j,  5.71189562e+01-2.12050702e+01j,
       -3.62001794e+01-9.38292577e+00j, -1.82433660e+01-5.44240126e+01j,
        2.37075183e+00-1.96054108e+01j, -3.85231231e+00+1.79861160e+01j,
       -2.25027517e+01+2.48013014e+01j,  4.26317210e+01-1.00091783e+01j,
       -1.69716555e+01-4.53089458e+00j, -2.56628360e+01+5.40797707e+01j,
```

(continues on next page)

(continued from previous page)

```
       -7.58375000e+00+1.48726709e+01j,  1.49239103e+01+7.38038464e+00j],
      [ 1.40162448e+01-3.55271368e-15j, -1.71506677e+01+2.17743955e+01j,
        3.20915788e+01-4.21927434e+00j,  1.65585338e-02-9.93002000e+00j,
       -1.16510556e+01-1.77476509e+01j, -5.16221513e+01+9.35723183e+00j,
        2.25243755e+01-2.38435501e+01j,  4.77310092e+01+5.20077481e+01j,
        4.86248231e+01-1.74894997e+00j,  1.66786364e+01+1.33522786e+01j,
       -8.69681785e+00+3.55271368e-15j,  1.66786364e+01-1.33522786e+01j,
        4.86248231e+01+1.74894997e+00j,  4.77310092e+01-5.20077481e+01j,
        2.25243755e+01+2.38435501e+01j, -5.16221513e+01-9.35723183e+00j,
       -1.16510556e+01+1.77476509e+01j,  1.65585338e-02+9.93002000e+00j,
        3.20915788e+01+4.21927434e+00j, -1.71506677e+01-2.17743955e+01j],
      [-5.24122435e+01+2.57484452e+01j,  1.49239103e+01-7.38038464e+00j,
       -7.58375000e+00-1.48726709e+01j, -2.56628360e+01-5.40797707e+01j,
       -1.69716555e+01+4.53089458e+00j,  4.26317210e+01+1.00091783e+01j,
       -2.25027517e+01-2.48013014e+01j, -3.85231231e+00-1.79861160e+01j,
        2.37075183e+00+1.96054108e+01j, -1.82433660e+01+5.44240126e+01j,
       -3.62001794e+01+9.38292577e+00j,  5.71189562e+01+2.12050702e+01j,
       -1.85524904e+01-2.25721542e+01j, -6.67169831e+01-6.45536117e+01j,
        3.99972852e+00+1.14365015e+01j,  2.83922223e+01+1.59345822e+01j,
       -5.27983284e+01+2.20977238e+01j, -2.51337731e+01-5.04668321e+01j,
       -4.69476119e+01+4.19065872e+01j,  7.88456072e+00-1.36192049e+01j],
      [-2.42445557e+00+1.52927059e+01j,  2.06068442e+01+4.91914400e+00j,
       -1.13623186e+01-1.46572477e+01j,  8.74145820e-01+2.16671479e+00j,
       -1.29989388e+01+9.99264531e+00j,  1.32792638e+01-8.14105169e+00j,
        1.22580500e+01+3.25180329e+00j,  2.23354935e+01+1.18006751e+01j,
       -7.84762169e+00-3.58774714e+01j, -2.57463428e+01-1.07618398e+01j,
        4.15379418e+01-1.21990746e+00j,  3.64910931e+00-3.90753262e-01j,
        2.51257491e-01-4.18618670e+00j,  1.10542525e+01+4.34099936e+00j,
       -1.67661069e+00+1.34034137e+01j, -9.14912144e+00+2.14527482e+00j,
       -1.93474045e+01-3.09792374e+01j,  4.83083325e+01-3.88306999e+01j,
        1.50253414e+01-4.68022072e+01j, -1.74865873e+01-3.95124060e+01j],
      [-5.53260189e-01+3.33642332e+00j,  5.17194527e+01-5.66418081e-01j,
       -7.79934151e+00-3.38929659e+01j,  3.99123661e+01+1.40791608e+01j,
        2.40399719e+01-1.60354310e+01j,  4.29016562e+01+2.77208110e+01j,
        2.33187751e+00-1.53364881e+01j,  7.72403168e+00+2.65696480e+01j,
       -5.75851523e+01+1.61377363e+01j, -5.08422850e+01-7.73216775e+00j,
        3.52998209e+01+9.15896876e+00j,  3.72751167e+00+4.04047009e+00j,
       -4.18423517e+01-7.67380731e+00j,  5.85350225e+00+1.55203019e+01j,
        4.82363622e+01-6.80687760e+01j,  3.95641923e+01+1.82508937e+01j,
        9.83735030e+00+3.69410893e+00j, -8.56095964e+00+8.15590602e+00j,
       -2.61418425e+01-6.74386221e+00j, -3.36865318e+01+5.67699005e+01j],
      [ 3.20962594e-01+5.06835211e+00j, -3.79819554e+01+1.09610326e+01j,
       -1.80254890e+01-1.68658665e+01j,  2.22153223e+01+4.70607883e+01j,
       -2.22563818e+01-4.74525652e+01j, -5.74958635e+00-5.93517320e+01j,
        2.41874283e+00-8.27550100e+00j, -3.08074399e+01-1.70928994e+01j,
        1.65440668e+01-1.20444372e+01j,  9.83142621e+00-3.45841898e+00j,
       -5.81901559e+01+1.92046694e+01j, -3.06979577e+01+1.49209380e+01j,
        4.85206624e+01+5.74968092e+01j, -1.26190710e+01-5.65750088e+00j,
        3.15126654e+01+1.06852728e+01j, -2.42030910e+00-7.35438393e+01j,
       -1.55673493e+01-2.14876071e+01j, -1.24628564e+01-1.01651712e+01j,
        6.29856157e+01-5.11739533e+00j,  2.19888129e+00+3.84687654e+01j],
      [-1.70526169e+00-1.92163372e+01j, -4.67295936e+01+1.68476993e+01j,
        2.77952349e+01-7.37147443e+00j, -3.25168875e+01-3.05116544e+01j,
       -4.84984905e+01-4.44450945e+00j, -2.74936090e+01-1.67932310e+01j,
       -2.67284300e+00+3.00440969e+01j,  1.69927163e+01+5.46139595e+00j,
       -1.89551153e+01+2.96675488e+00j,  3.70133743e+01-5.49322284e+01j,
       -1.79552477e+01-1.19034968e+00j,  1.30386761e+01-1.50867127e+01j,
```

(continues on next page)

(continued from previous page)

```
        -5.86853908e+01+4.98184864e+00j,  7.13148018e+01-3.63742781e+01j,
        -2.21996880e+01+5.72145390e+01j, -4.05256849e+01+1.34379916e+01j,
         4.97460634e+01+2.32829438e+01j, -6.64633008e+00+4.38510142e+00j,
        -1.85369575e+01+2.35003882e+01j,  2.97034824e+01+2.79547749e+01j],
       [-3.07316501e+01-3.93892519e+01j, -8.70958377e-01-9.02082150e-01j,
        -1.81651373e+01-3.26587070e+01j, -5.63117259e+01-1.11153064e+01j,
        -4.53686973e+00-1.45217067e+01j,  1.54237181e+01+8.05481446e+00j,
         2.26950448e+01-1.38349923e+01j, -2.00550737e+01+9.79395414e+00j,
         4.12779068e+00-2.93916104e+00j,  6.37599026e+01+4.38710211e+00j,
         3.33511180e+00-1.94216403e+01j, -1.39586815e+01+1.93893701e+01j,
         2.35079633e+01-7.50280198e+00j, -6.73505251e+00-1.98416837e+00j,
        -5.77409216e+01-1.60540597e+01j,  4.91499643e+01-1.65418913e+01j,
        -2.58192580e+01+6.89101643e+00j, -3.66511012e+00+1.63464859e+01j,
         8.14578399e+00-3.67631795e+01j,  4.37522412e+01+2.58679396e+01j],
       [ 1.18824392e+01+3.22445034e+01j, -1.63233034e+01-3.07002636e+00j,
        -1.69562981e+01-2.69981446e+01j,  1.56397075e+00+5.98568234e+00j,
         4.82976884e+01+5.07935313e+00j,  8.95293832e+00-3.02611311e+01j,
         7.12788455e+00+1.01466482e+01j, -1.71615097e+01+2.65530672e+01j,
         3.02689172e+01+1.20605584e+00j, -1.83299526e+01-1.41597367e+01j,
         1.06994721e+01+5.91144258e+01j,  2.45644090e+01-3.12855452e+01j,
        -1.16715093e+01-2.20421542e+01j, -1.79272987e+01+5.37861863e+01j,
        -2.69432497e+01+3.13403552e+00j, -8.81439084e-01+1.06270574e+01j,
        -2.64544431e+01-3.55722370e+01j, -9.94112937e+00+8.26233410e+01j,
        -5.37257684e+00-4.98413122e+01j, -1.90338418e+01-1.51946428e+01j],
       [-5.58110611e+00-4.40670582e+01j,  1.54083249e+01-2.00649186e+01j,
        -2.34627949e+00+8.89046635e+00j, -3.19274861e+01-7.80041966e+00j,
         4.85320648e+01-5.60596237e+00j, -1.73257338e+01-5.30225228e+01j,
        -1.43962466e+01-2.65526852e+01j,  7.95868211e+00-5.65582314e+01j,
         2.43658059e+01+2.55621553e+01j, -2.25934944e+01-2.84674795e+01j,
         1.80817697e+01+2.95532249e+01j,  5.41040083e-01+1.81124148e+01j,
        -6.06693803e+00-2.07656749e+01j, -4.08850966e+01+2.60055911e+00j,
         1.98169103e+00+7.22053259e+00j, -4.78639712e+01-4.14072434e-01j,
        -2.17746984e+01-7.18592405e+01j,  4.56543207e+01+1.42979937e+01j,
        -2.31106044e+01+1.65055492e+00j, -2.94205956e+00+4.12300389e+01j],
       [-1.07818296e+00+1.47849666e+01j,  5.70284748e+01-3.43547254e+01j,
         6.45552327e+00-2.69100506e+01j,  3.89740324e+01+3.17500007e+01j,
         4.05592205e+00+2.11454331e+01j, -1.47845148e+01+2.05218191e+01j,
        -3.62093944e+01+8.93102919e+01j,  3.41384511e+00-2.09842327e+01j,
        -2.82417266e+01+3.39720714e+01j,  1.02898229e+01+1.29941840e+01j,
         1.65210470e+01+7.33123761e+01j,  3.68696602e+01-3.59515289e+01j,
         1.71673780e+01+3.47696707e+01j,  3.30809693e+01+9.34850620e+00j,
        -6.11206007e+00-2.25673610e+01j,  2.35262485e+01-4.98147526e+01j,
        -1.52437785e+01+8.43433973e+00j, -4.39314574e+01+2.09381347e+01j,
         3.79211155e-01-1.61643017e+01j,  4.47886848e+01-2.34594148e+01j],
       [ 1.06523934e+01-1.73673192e+01j, -1.65841241e+01+2.38798773e+01j,
         2.18422788e+01-3.21744430e+00j, -2.42399929e+00-3.57250326e+01j,
         2.58992505e+00-2.03285531e+01j, -2.47154470e+00+2.07686708e+01j,
         3.53326051e+01-1.81494418e+01j,  2.88057312e+01-2.37735569e+00j,
        -3.22237081e+01-2.25641449e+00j, -1.20910217e+01+4.36106228e+01j,
         4.30603867e+00-2.01927339e+00j, -3.44036489e+01+3.06603442e+01j,
        -2.85868251e+01-3.59036353e+01j,  4.70004035e+01+1.22464039e+01j,
        -1.40264564e+01-4.29802397e+01j,  8.12464287e+00-2.22765043e+00j,
        -3.60650217e+01-6.05873460e-01j,  2.60177333e+01-1.53278026e+01j,
        -4.93971463e+01-2.09266660e+01j, -1.95164660e+01+7.43979407e+01j],
       [ 7.15150886e+00-1.25220177e+01j, -3.92473335e+01-2.68505554e+01j,
         3.23407481e+01+6.68717454e+01j, -4.14294443e+01+1.48474337e+01j,
         5.75318857e+00+2.22803596e+00j, -3.95286687e+01+9.60118722e+00j,
```

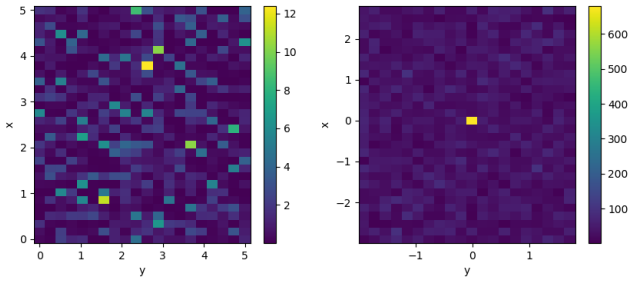
(continues on next page)

(continued from previous page)

```
         3.27265622e+01-2.80609377e+00j, -7.06272713e+01+1.11875490e+01j,
         4.22534702e+01+1.06789983e+01j,  5.50489436e+00+1.22197283e+01j,
         3.04358949e+01-6.36528460e+01j, -3.11790029e+01+2.11839622e+01j,
        -1.92307775e+00+2.94975177e+01j, -2.29754969e+01+1.78710672e+01j,
        -7.49789987e+00-1.57377970e+01j,  4.94153737e+01+9.78063668e+00j,
         1.81568112e+01-1.23120934e+01j, -2.21190855e+01+4.34572687e-02j,
         2.35781124e+01+3.03692665e+01j, -1.64656200e+01+8.80436886e+00j],
       [-3.00756470e+01+2.02311653e+01j, -2.13363900e+01-1.58769170e+01j,
         2.03781124e+01+1.49895411e+01j,  3.10571817e+00-4.66607518e+01j,
        -1.80884532e+00+3.35852750e+01j, -2.36212076e+01-5.93467176e+01j,
         1.17154656e+01+3.47452516e+01j, -1.10414273e+01+7.78928074e+00j,
         4.52215055e+01+3.12928417e+01j,  1.44147644e+01-2.88680317e+01j,
        -5.02314992e+01+1.11478363e+01j,  3.91626381e+00+1.85662042e+00j,
         1.62248018e+01-4.73648432e+00j, -2.16237997e+01+1.94507161e+01j,
        -2.30835820e+01-2.65438675e+01j,  1.26375702e+00-5.25047677e+00j,
         2.05435690e+00-1.04214009e+01j,  3.95568373e+01+8.28074518e+00j,
         4.71762598e+01+8.72739590e+00j, -1.52199079e+01-6.12960304e+01j],
       [ 1.18390109e+01-1.69615481e+01j,  2.23641363e+01+2.98647685e-02j,
         8.93965329e+00-1.28033963e+01j, -6.17481319e+00-1.43282178e+01j,
         5.18844566e+01+3.92437600e+01j,  1.49904457e+01-1.62863784e+00j,
        -3.63069591e+01-2.63605337e+01j,  2.27092738e+01-1.09230345e+01j,
        -5.77461281e+00+2.23819916e+01j,  4.48110583e+01-2.09499730e+00j,
        -2.91364141e+01+4.57237154e+01j,  2.60820207e+01+1.08850196e+01j,
        -2.41167969e+01-8.25647322e+00j,  6.89571773e+00+7.65203935e+00j,
         1.99909029e+01+2.37322857e+01j, -7.38998869e+00-5.92937093e+00j,
        -3.26843493e+01-2.78910788e+01j,  1.51573868e+01-8.87107641e-02j,
         3.48168720e+00-3.93232365e+01j, -1.03891185e+01-3.07147965e+01j],
       [-9.78304942e+00+2.00060691e+01j,  1.28963968e+01+3.01513717e+01j,
        -3.75456761e+00-6.28466305e+01j, -2.64074189e+01+1.52314832e+01j,
         1.20560230e+00-2.45535230e+01j,  2.29408613e+00+2.10730557e+01j,
        -2.24569277e+01-4.78807494e+00j,  5.89818080e+01+1.58963695e+00j,
        -3.39807068e+01-9.65027977e+00j,  1.07657876e+01+1.41656999e+01j,
        -4.84907294e+01+4.16601947e+01j,  1.23190320e+01-1.63252604e+01j,
        -3.41801537e+01-4.29905149e+01j, -7.66929151e+00+4.16255367e+00j,
        -1.90295095e+01-5.25959400e+01j, -2.48612360e+01+1.19071034e+01j,
         5.75308280e+00-2.73910712e+01j, -1.65402379e+01-5.18006959e+00j,
         3.65704485e+01-2.64031486e+01j, -3.01214418e+01+6.86851691e-01j]])
Coordinates:
  * x        (x) float64 0.0 0.1933 0.3867 0.58 ... -0.58 -0.3867 -0.1933
  * y        (y) float64 0.0 0.19 0.38 0.57 0.76 ... -0.76 -0.57 -0.38 -0.19
```

```
In [14]: plt.figure(figsize=(10, 4))
Out[14]: <Figure size 1000x400 with 0 Axes>

In [15]: plt.subplot(1, 2, 1)
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f04e770ea90>

In [16]: arr.plot()
Out[16]: <matplotlib.collections.QuadMesh at 0x7f04e749a0d0>

In [17]: plt.subplot(1, 2, 2)
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f04e749a350>

In [18]: np.abs(fftn.sortby('x').sortby('y')).plot()
Out[18]: <matplotlib.collections.QuadMesh at 0x7f04e73d8710>
```

```
In [19]: plt.show()
```



1.4 Digital filters

`xr-scipy` wraps some of SciPy functions for constructing frequency filters using functions such as `scipy.signal.firwin()` and `scipy.signal.iirfilter()`. Wrappers for convenient functions such as `scipy.signal.decimate()` and `scipy.signal.savgol_filter()` are also provided. For convenience, the `xrscipy.signal` namespace will be imported under the alias `dsp`

```
In [1]: import xrscipy.signal as dsp
```

1.4.1 Frequency filters

The main wrapper for frequency filters is the *`frequency_filter()`* wrapper. It's many arguments enable one to specify the type of filter, e.g. frequency band, FIR or IIR, response type family, filter order, forward-backward filtering, etc. By default a Butterworth IIR 4-th order filter with second-order-series (numerically stable even for high orders) forward-backward application (zero phase shift, but double order) is used, because such a filter typically offers a good performance for most time-series analysis applications.

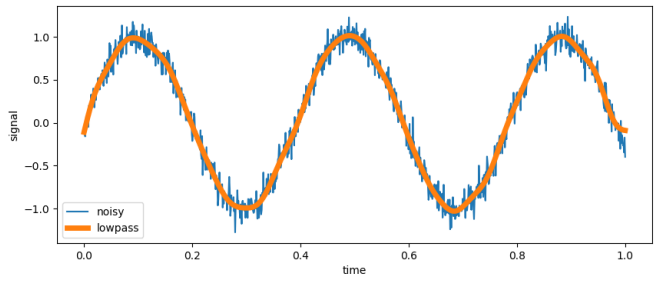
Convenience functions such as *`lowpass()`*, *`highpass()`* and *`bandpass()`* are provided which wrap *`frequency_filter()`* with a predefined response type and offer a more convenient interface for the cutoff frequency specification.

The cutoff frequency is specified in the inverse usint to the filtered dimension's coordinates (typically time). The wrapper automatically checks the sampling of those coordinates and normalizes the supplied frequency by the Nyquist frequency.

In the following example a simple low-pass filter is applied to a noisy signal. Because the cutoff frequency is close to 0 (relative to the Nyquist frequency) a high filter order is used for sufficient noise attenuation.

```
In [2]: t = np.linspace(0, 1, 1000)  # seconds

In [3]: sig = xr.DataArray(np.sin(16*t) + np.random.normal(0, 0.1, t.size),
   ...:                    coords=[('time', t)], name='signal')
   ...:

In [4]: sig.plot(label='noisy')
Out[4]: [<matplotlib.lines.Line2D at 0x7f04e7155850>]

In [5]: low = dsp.lowpass(sig, 20, order=8)  # cutoff at 20 Hz

In [6]: low.plot(label='lowpass', linewidth=5)
Out[6]: [<matplotlib.lines.Line2D at 0x7f04e715b210>]
```

```
In [7]: plt.legend()
Out[7]: <matplotlib.legend.Legend at 0x7f04e7456b50>

In [8]: plt.show()
```



1.4.2 Decimation

To demonstrate basic functionality of *decimate()*, let's create a simple example DataArray:

```
In [9]: arr = xr.DataArray(np.sin(np.linspace(0, 6.28, 300)) ** 2,
   ...:                    dims=('x'), coords={'x': np.linspace(0, 5, 300)})
   ...:

In [10]: arr
Out[10]:
<xarray.DataArray (x: 300)>
array([0.00000000e+00, 4.41075615e-04, 1.76352427e-03, 3.96501276e-03,
       7.04165700e-03, 1.09880289e-02, 1.57971657e-02, 2.14605829e-02,
       2.79682883e-02, 3.53088004e-02, 4.34691683e-02, 5.24349947e-02,
       6.21904612e-02, 7.27183561e-02, 8.40001050e-02, 9.60158036e-02,
       1.08744253e-01, 1.22162995e-01, 1.36248356e-01, 1.50975485e-01,
       1.66318399e-01, 1.82250028e-01, 1.98742264e-01, 2.15766010e-01,
       2.33291231e-01, 2.51287007e-01, 2.69721587e-01, 2.88562448e-01,
       3.07776350e-01, 3.27329391e-01, 3.47187076e-01, 3.67314370e-01,
       3.87675760e-01, 4.08235325e-01, 4.28956790e-01, 4.49803597e-01,
       4.70738966e-01, 4.91725960e-01, 5.12727552e-01, 5.33706689e-01,
       5.54626357e-01, 5.75449647e-01, 5.96139821e-01, 6.16660376e-01,
       6.36975107e-01, 6.57048172e-01, 6.76844156e-01, 6.96328134e-01,
       7.15465730e-01, 7.34223179e-01, 7.52567388e-01, 7.70465991e-01,
       7.87887410e-01, 8.04800909e-01, 8.21176647e-01, 8.36985732e-01,
       8.52200273e-01, 8.66793426e-01, 8.80739444e-01, 8.94013722e-01,
       9.06592841e-01, 9.18454608e-01, 9.29578094e-01, 9.39943674e-01,
       9.49533061e-01, 9.58329335e-01, 9.66316978e-01, 9.73481896e-01,
       9.79811449e-01, 9.85294470e-01, 9.89921285e-01, 9.93683730e-01,
       9.96575168e-01, 9.98590497e-01, 9.99726161e-01, 9.99980157e-01,
       9.99352038e-01, 9.97842910e-01, 9.95455437e-01, 9.92193830e-01,
       9.88063845e-01, 9.83072767e-01, 9.77229403e-01, 9.70544062e-01,
       9.63028539e-01, 9.54696093e-01, 9.45561427e-01, 9.35640654e-01,
       9.24951281e-01, 9.13512164e-01, 9.01343487e-01, 8.88466719e-01,
       8.74904578e-01, 8.60680991e-01, 8.45821055e-01, 8.30350985e-01,
       8.14298077e-01, 7.97690651e-01, 7.80558010e-01, 7.62930379e-01,
       7.44838859e-01, 7.26315369e-01, 7.07392591e-01, 6.88103910e-01,
       6.68483356e-01, 6.48565547e-01, 6.28385623e-01, 6.07979188e-01,
       5.87382245e-01, 5.66631134e-01, 5.45762465e-01, 5.24813057e-01,
```

(continues on next page)

(continued from previous page)

```
       5.03819871e-01, 4.82819946e-01, 4.61850332e-01, 4.40948025e-01,
       4.20149904e-01, 3.99492663e-01, 3.79012747e-01, 3.58746289e-01,
       3.38729045e-01, 3.18996332e-01, 2.99582964e-01, 2.80523193e-01,
       2.61850645e-01, 2.43598265e-01, 2.25798254e-01, 2.08482019e-01,
       1.91680109e-01, 1.75422169e-01, 1.59736883e-01, 1.44651923e-01,
       1.30193905e-01, 1.16388337e-01, 1.03259576e-01, 9.08307848e-02,
       7.91238919e-02, 6.81595518e-02, 5.79571089e-02, 4.85345633e-02,
       3.99085393e-02, 3.20942558e-02, 2.51054996e-02, 1.89546009e-02,
       1.36524118e-02, 9.20828690e-03, 5.63006706e-03, 2.92406530e-03,
       1.09505583e-03, 1.46265586e-04, 7.93685134e-05, 8.94482641e-04,
       2.59016986e-03, 5.16343847e-03, 8.60974844e-03, 1.29230194e-02,
       1.80956415e-02, 2.41184887e-02, 3.09809348e-02, 3.86708724e-02,
       4.71747340e-02, 5.64775165e-02, 6.65628067e-02, 7.74128112e-02,
       8.90083874e-02, 1.01329077e-01, 1.14353143e-01, 1.28057606e-01,
       1.42418289e-01, 1.57409853e-01, 1.73005850e-01, 1.89178764e-01,
       2.05900061e-01, 2.23140239e-01, 2.40868881e-01, 2.59054708e-01,
       2.77665637e-01, 2.96668830e-01, 3.16030761e-01, 3.35717269e-01,
       3.55693622e-01, 3.75924575e-01, 3.96374434e-01, 4.17007120e-01,
       4.37786231e-01, 4.58675106e-01, 4.79636890e-01, 5.00634601e-01,
       5.21631192e-01, 5.42589620e-01, 5.63472906e-01, 5.84244207e-01,
       6.04866876e-01, 6.25304528e-01, 6.45521104e-01, 6.65480938e-01,
       6.85148813e-01, 7.04490030e-01, 7.23470464e-01, 7.42056629e-01,
       7.60215733e-01, 7.77915737e-01, 7.95125414e-01, 8.11814401e-01,
       8.27953252e-01, 8.43513495e-01, 8.58467677e-01, 8.72789412e-01,
       8.86453435e-01, 8.99435637e-01, 9.11713113e-01, 9.23264203e-01,
       9.34068527e-01, 9.44107023e-01, 9.53361980e-01, 9.61817069e-01,
       9.69457373e-01, 9.76269412e-01, 9.82241168e-01, 9.87362105e-01,
       9.91623187e-01, 9.95016898e-01, 9.97537249e-01, 9.99179794e-01,
       9.99941634e-01, 9.99821427e-01, 9.98819383e-01, 9.96937271e-01,
       9.94178411e-01, 9.90547671e-01, 9.86051457e-01, 9.80697701e-01,
       9.74495849e-01, 9.67456842e-01, 9.59593101e-01, 9.50918498e-01,
       9.41448338e-01, 9.31199330e-01, 9.20189556e-01, 9.08438441e-01,
       8.95966716e-01, 8.82796387e-01, 8.68950689e-01, 8.54454050e-01,
       8.39332047e-01, 8.23611360e-01, 8.07319725e-01, 7.90485884e-01,
       7.73139539e-01, 7.55311293e-01, 7.37032600e-01, 7.18335711e-01,
       6.99253611e-01, 6.79819967e-01, 6.60069067e-01, 6.40035756e-01,
       6.19755380e-01, 5.99263719e-01, 5.78596927e-01, 5.57791467e-01,
       5.36884044e-01, 5.15911547e-01, 4.94910977e-01, 4.73919386e-01,
       4.52973809e-01, 4.32111201e-01, 4.11368368e-01, 3.90781909e-01,
       3.70388143e-01, 3.50223052e-01, 3.30322213e-01, 3.10720737e-01,
       2.91453207e-01, 2.72553616e-01, 2.54055309e-01, 2.35990924e-01,
       2.18392330e-01, 2.01290577e-01, 1.84715838e-01, 1.68697356e-01,
       1.53263391e-01, 1.38441175e-01, 1.24256858e-01, 1.10735466e-01,
       9.79008540e-02, 8.57756665e-02, 7.43812960e-02, 6.37378457e-02,
       5.38640937e-02, 4.47774605e-02, 3.64939774e-02, 2.90282592e-02,
       2.23934776e-02, 1.66013384e-02, 1.16620606e-02, 7.58435858e-03,
       4.37542673e-03, 2.04092654e-03, 5.84976768e-04, 1.01461475e-05])
Coordinates:
  * x        (x) float64 0.0 0.01672 0.03344 0.05017 ... 4.95 4.967 4.983 5.0
```

Our `decimate()` takes an xarray object (possibly high dimensional) and a dimension name (if not 1D) along which the signal should be decimated. Decimation means

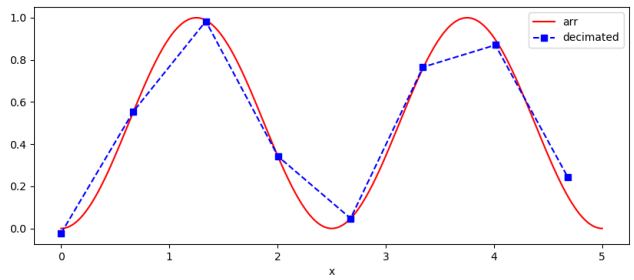
1. Apply a lowpass filter to remove frequencies above the new Nyquist frequency in order to prevent aliasing
2. Take every q-th point

```
In [11]: arr_decimated = dsp.decimate(arr, q=40)

In [12]: arr_decimated
Out[12]:
<xarray.DataArray (x: 8)>
array([-0.02324268,  0.55546594,  0.98349812,  0.34108733,  0.04773383,
        0.76581061,  0.87046882,  0.2435029 ])
Coordinates:
  * x        (x) float64 0.0 0.6689 1.338 2.007 2.676 3.344 4.013 4.682
```

An alternative parameter to `q` is `target_fs` which is the new target sampling frequency to obtain, `q = np.rint(current_fs / target_fs)`.

The return type is also a DataArray with coordinates.

```
In [13]: arr.plot(label='arr', color='r')
Out[13]: [<matplotlib.lines.Line2D at 0x7f04e74ffc50>]

In [14]: arr_decimated.plot.line('s--', label='decimated', color='b')
Out[14]: [<matplotlib.lines.Line2D at 0x7f04e76d4750>]

In [15]: plt.legend()
Out[15]: <matplotlib.legend.Legend at 0x7f04e71552d0>

In [16]: plt.show()
```



The other keyword arguments are passed on to *lowpass()*.

1.4.3 Savitzky-Golay LSQ filtering

The Savitzky-Golay filter as a special type of a FIR filter which is equivalent to replacing filtered values by leas-square fits of polynomials (or their derivatives) of a given order within a rolling window. For details see their Wikipedia page Such a filter is very useful when temporal or spatial features in the signal are of greater interest than frequency or wavenumber bands, respectively.

To demonstrate basic functionality of *savgol_filter()*, let's create a simple example DataArray of the quadratic shape and add some noise:

```
In [17]: arr = xr.DataArray(np.linspace(0, 5, 50) ** 2,
   ....:                    dims=('x'), coords={'x': np.linspace(0, 5, 50)})
   ....:

In [18]: noise = np.random.normal(0,3,50)

In [19]: arr_noisy = arr + noise
```

(continues on next page)

(continued from previous page)

```
In [20]: arr
Out[20]:
<xarray.DataArray (x: 50)>
array([0.00000000e+00, 1.04123282e-02, 4.16493128e-02, 9.37109538e-02,
       1.66597251e-01, 2.60308205e-01, 3.74843815e-01, 5.10204082e-01,
       6.66389005e-01, 8.43398584e-01, 1.04123282e+00, 1.25989171e+00,
       1.49937526e+00, 1.75968347e+00, 2.04081633e+00, 2.34277384e+00,
       2.66555602e+00, 3.00916285e+00, 3.37359434e+00, 3.75885048e+00,
       4.16493128e+00, 4.59183673e+00, 5.03956685e+00, 5.50812162e+00,
       5.99750104e+00, 6.50770512e+00, 7.03873386e+00, 7.59058726e+00,
       8.16326531e+00, 8.75676801e+00, 9.37109538e+00, 1.00062474e+01,
       1.06622241e+01, 1.13390254e+01, 1.20366514e+01, 1.27551020e+01,
       1.34943773e+01, 1.42544773e+01, 1.50354019e+01, 1.58371512e+01,
       1.66597251e+01, 1.75031237e+01, 1.83673469e+01, 1.92523948e+01,
       2.01582674e+01, 2.10849646e+01, 2.20324865e+01, 2.30008330e+01,
       2.39900042e+01, 2.50000000e+01])
Coordinates:
  * x        (x) float64 0.0 0.102 0.2041 0.3061 ... 4.694 4.796 4.898 5.0
```

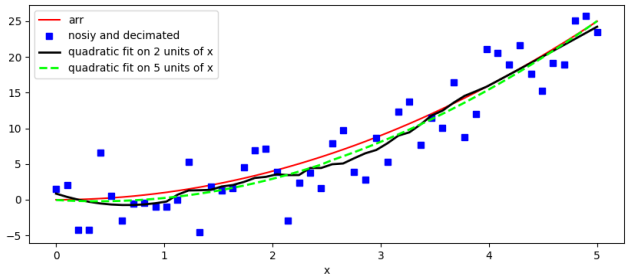
Our *savgol_filter()* takes an xarray object (possibly high dimensional) and a dimension name (if not 1D) along which the signal should be filtered. The window length is given in the units of the dimension coordinates.

```
In [21]: arr_savgol2 = dsp.savgol_filter(arr_noisy, 2, 2)

In [22]: arr_savgol5 = dsp.savgol_filter(arr_noisy, 5, 2)

In [23]: arr_savgol2
Out[23]:
<xarray.DataArray (x: 50)>
array([ 8.56203341e-01,  3.96395647e-01,  1.41459548e-02, -2.90545737e-01,
       -5.17679429e-01, -6.67255121e-01, -7.39272812e-01, -7.33732502e-01,
       -6.50634192e-01, -4.89977882e-01, -2.51763571e-01,  7.02067790e-01,
        1.30020637e+00,  1.33302851e+00,  1.38572699e+00,  1.85653379e+00,
        2.06074739e+00,  2.49014897e+00,  3.05217772e+00,  3.20485447e+00,
        3.56841209e+00,  3.44974397e+00,  3.46125936e+00,  4.44487308e+00,
        4.44708328e+00,  4.96773137e+00,  5.08244353e+00,  5.81362471e+00,
        6.50226254e+00,  6.97100619e+00,  7.89901971e+00,  8.98203322e+00,
        9.44445152e+00,  1.05323974e+01,  1.17935064e+01,  1.24788400e+01,
        1.35918883e+01,  1.45515981e+01,  1.51849573e+01,  1.58435727e+01,
        1.66685321e+01,  1.74966544e+01,  1.83279396e+01,  1.91623877e+01,
        1.99999988e+01,  2.08407728e+01,  2.16847097e+01,  2.25318095e+01,
        2.33820722e+01,  2.42354978e+01])
Coordinates:
  * x        (x) float64 0.0 0.102 0.2041 0.3061 ... 4.694 4.796 4.898 5.0

In [24]: arr_savgol5
Out[24]:
<xarray.DataArray (x: 50)>
array([-3.32337380e-02, -1.15598437e-01, -1.72851319e-01, -2.04992385e-01,
       -2.12021634e-01, -1.93939066e-01, -1.50744681e-01, -8.24384804e-02,
        1.09795374e-02,  1.29509372e-01,  2.73151023e-01,  4.41904491e-01,
        6.35769775e-01,  8.54746876e-01,  1.09883579e+00,  1.36803653e+00,
        1.66234908e+00,  1.98177345e+00,  2.32630963e+00,  2.69595763e+00,
        3.09071745e+00,  3.51058909e+00,  3.95557254e+00,  4.42566781e+00,
```

```
        4.92087489e+00,  5.54817753e+00,  6.09173275e+00,  6.65861004e+00,
        7.24880940e+00,  7.86233084e+00,  8.49917436e+00,  9.15933995e+00,
        9.84282761e+00,  1.05496373e+01,  1.12797692e+01,  1.20332230e+01,
        1.28099990e+01,  1.36100970e+01,  1.44335171e+01,  1.52802593e+01,
        1.61503236e+01,  1.70437099e+01,  1.79604183e+01,  1.89004488e+01,
        1.98638013e+01,  2.08504759e+01,  2.18604726e+01,  2.28937914e+01,
        2.39504322e+01,  2.50303952e+01])
Coordinates:
  * x        (x) float64 0.0 0.102 0.2041 0.3061 ... 4.694 4.796 4.898 5.0
```

The return type is also a DataArray with coordinates.

```
In [25]: arr.plot(label='arr', color='r')
Out[25]: [<matplotlib.lines.Line2D at 0x7f04e7104ad0>]

In [26]: arr_noisy.plot.line('s', label='nosiy and decimated', color='b')
Out[26]: [<matplotlib.lines.Line2D at 0x7f04e74e4ad0>]

In [27]: arr_savgol2.plot(label='quadratic fit on 2 units of x', color='k',␣
↪linewidth=2)
Out[27]: [<matplotlib.lines.Line2D at 0x7f04e7178310>]

In [28]: arr_savgol5.plot.line('--',label='quadratic fit on 5 units of x',␣
↪linewidth=2, color='lime')
Out[28]: [<matplotlib.lines.Line2D at 0x7f04e710ed10>]

In [29]: plt.legend()
Out[29]: <matplotlib.legend.Legend at 0x7f04e7178590>

In [30]: plt.show()
```



The other options (polynomial and derivative order) are the same as for `scipy.signal.savgol_filter()`, see *`savgol_filter()`* for details.

1.5 Spectral (FFT) analysis

xr-scipy wraps some of scipy spectral analysis functions such as `scipy.signal.spectrogram()`, `scipy.signal.csd()` etc. For convenience, the `xrscipy.signal` namespace will be imported under the alias `dsp`

```
In [1]: import xrscipy.signal as dsp
```

To demonstrate the basic functionality, let's create two simple example DataArray at a similar frequency but one with a frequency drift and some noise:

```
In [2]: time_ax = np.arange(0,100,0.01)

In [3]: sig_1 = xr.DataArray(np.sin(100 * time_ax) + np.random.rand(len(time_ax))*3,
   ...:                      coords=[("time", time_ax)], name='sig_1')
   ...:

In [4]: sig_2 = xr.DataArray((np.cos(100 * time_ax) + np.random.rand(len(time_ax))*3 +
   ...:                       3*np.sin(30 * time_ax**1.3)),
   ...:                      coords=[("time", time_ax)], name='sig_2')
   ...:
```

1.5.1 Power spectra

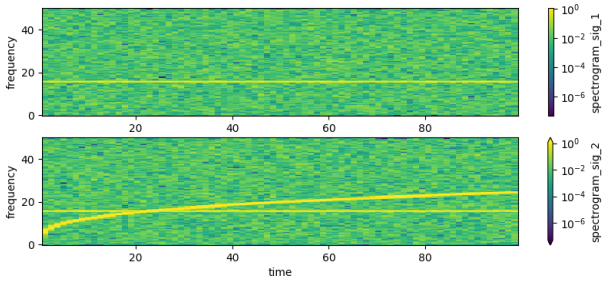
The *spectrogram()* function can be used directly on an xarray DataArray object. The returned object is again an `xarray.DataArray` object.

```
In [5]: spec_1 = dsp.spectrogram(sig_1)

In [6]: spec_2 = dsp.spectrogram(sig_2)

In [7]: spec_2
Out[7]:
<xarray.DataArray 'spectrogram_sig_2' (frequency: 129, time: 77)>
array([[2.44424801e-05, 5.22947169e-03, 2.89224670e-04, ...,
        1.49873467e-03, 7.65608700e-04, 9.59582636e-05],
       [1.75153585e-02, 1.89154990e-02, 7.80560915e-03, ...,
        1.44778153e-03, 1.09230319e-02, 9.86477734e-03],
       [2.67981444e-02, 1.72320528e-03, 1.77923215e-02, ...,
        3.47197400e-03, 7.94489056e-05, 1.24722011e-02],
       ...,
       [7.31455285e-03, 1.98518741e-02, 5.07390687e-03, ...,
        6.44165409e-03, 5.43732551e-03, 7.00189408e-03],
       [3.07786421e-03, 1.69232295e-03, 1.04356585e-03, ...,
        2.04653030e-02, 5.20056837e-04, 1.61887611e-02],
       [2.69532456e-03, 1.81679619e-05, 2.76618829e-03, ...,
        5.93410480e-03, 6.43149358e-03, 1.45961774e-02]])
Coordinates:
  * time       (time) float64 1.28 2.56 3.84 5.12 6.4 ... 94.72 96.0 97.28 98.56
  * frequency  (frequency) float64 0.0 0.3906 0.7812 1.172 ... 49.22 49.61 50.0
```

The `frequency` dimension coords are based on the transformed dimension (`time` in this case) coords sampling (i.e. inverse units). When the signal is 1D, the dimension does not have to be provided.

```
In [8]: norm = plt.matplotlib.colors.LogNorm()

In [9]: plt.subplot(211)
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f04e6f29990>

In [10]: spec_1.plot(norm=norm)
Out[10]: <matplotlib.collections.QuadMesh at 0x7f04e74e46d0>

In [11]: plt.subplot(212)
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f04ebdcdc90>

In [12]: spec_2.plot(norm=norm)
```

```
Out[12]: <matplotlib.collections.QuadMesh at 0x7f04e6f49250>

In [13]: plt.show()
```



These routines calculate the FFT on segments of the signal of a length controlled by `nperseg` and `nfft` parameters. The routines here offer a convenience parameter `seglen` which makes it possible to specify the segment length in the units of the transformed dimension's coords. If `seglen` is specified, `nperseg` is then calculated from it and `nfft` is set using `scipy.fftpack.next_fast_len` (or to closest higher power of 2). A desired frequency resolution spacing `df` can be achieved by specifying `seglen=1/df`.

Another convenience parameter is `overlap_ratio` which calculates the `noverlap` parameter (by how many points the segments overlap) as `noverlap = np.rint(overlap_ratio * nperseg)`

For example, these parameters calculate the spectrogram with a higher frequency resolution and try to make for the longer segments by overlapping them by 75%.

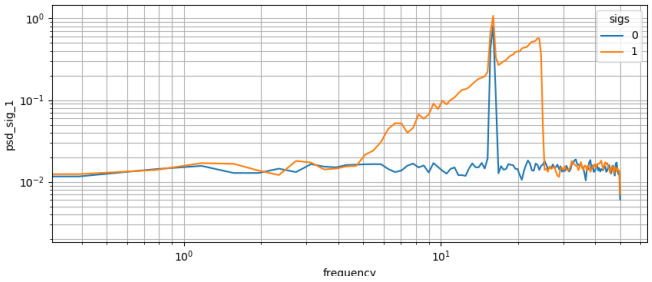
```
In [14]: dsp.spectrogram(sig_1, seglen=1, overlap_ratio=0.75)
Out[14]:
<xarray.DataArray 'spectrogram_sig_1' (frequency: 51, time: 397)>
array([[3.99319244e-04, 5.72385049e-06, 4.63213189e-04, ...,
        1.30700319e-03, 8.42935876e-04, 2.13630973e-03],
       [2.36519351e-03, 2.63675999e-03, 2.70759396e-03, ...,
        1.35248503e-02, 1.52467384e-02, 1.52477065e-02],
       [1.14508286e-03, 6.57120911e-03, 1.59951326e-02, ...,
        2.59556948e-02, 2.86589719e-02, 2.80344654e-02],
       ...,
       [3.22485304e-02, 5.98876777e-03, 4.75008719e-03, ...,
        1.27105913e-02, 1.10435949e-02, 2.60625766e-03],
       [6.48727646e-03, 4.24654407e-03, 5.71875070e-03, ...,
        3.54197551e-03, 4.97810155e-02, 5.84757962e-02],
       [7.82088430e-04, 3.02992866e-04, 8.29188527e-03, ...,
        1.46543043e-02, 4.78417261e-02, 3.80860346e-02]])
Coordinates:
  * time       (time) float64 0.5 0.75 1.0 1.25 1.5 ... 98.75 99.0 99.25 99.5
  * frequency  (frequency) float64 0.0 1.0 2.0 3.0 4.0 ... 47.0 48.0 49.0 50.0
```

All the functions can be calculated on N-dimensional signals if the dimension is provided. Here the power spectral density (PSD) P_{xx} is calculated using Welch's method (i.e. time average of the spectrogram) is shown

```
In [15]: sig_2D = xr.concat([sig_1,sig_2], dim="sigs")

In [16]: psd_2D = dsp.psd(sig_2D, dim="time")
```

```
In [17]: psd_2D.plot.line(x='frequency')
Out[17]:
[<matplotlib.lines.Line2D at 0x7f04e7036e50>,
 <matplotlib.lines.Line2D at 0x7f04e6ab4110>]
```

```
In [18]: plt.loglog()
Out[18]: []

In [19]: plt.grid(which='both')

In [20]: plt.show()
```



1.5.2 Cross-coherence and correlation

The same windowed FFT approach is also used to calculate the cross-spectral density P_{xy} (using *xrscipy.signal.csd()*) and from it coherency γ as

$$\gamma = \frac{\langle P_{xy} \rangle}{\sqrt{\langle P_{xx} \rangle \langle P_{yy} \rangle}}$$

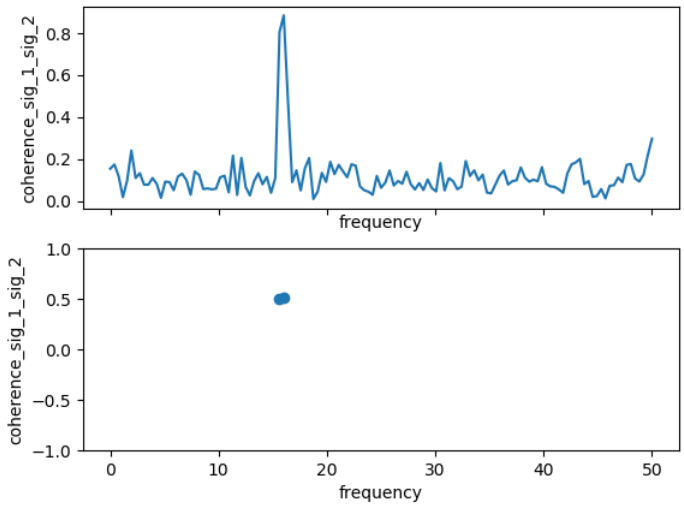
where $\langle \dots \rangle$ is an average over the FFT windows, i.e. ergodicity is assumed.

```
In [21]: coher_12 = dsp.coherence(sig_1, sig_2)

In [22]: coher_12[:10]
Out[22]:
<xarray.DataArray 'coherence_sig_1_sig_2' (frequency: 10)>
array([ 0.15445836+0.j        ,  0.09626349+0.14572021j,
        0.01668929+0.1171455j , -0.01160877+0.0149668j ,
       -0.01835304-0.09690598j,  0.05856481-0.23405385j,
       -0.10611867-0.03007782j, -0.08271153-0.10426975j,
        0.01914803-0.07631421j,  0.0691541 -0.03660832j])
Coordinates:
  * frequency  (frequency) float64 0.0 0.3906 0.7812 1.172 ... 2.734 3.125 3.516
```

The returned γ `DataArray` is complex (because so is P_{xy}) and the modulus is what is more commonly called coherence and the angle is the phase shift.

```
In [23]: coh = np.abs(coher_12)

In [24]: xphase = xr.apply_ufunc(np.angle, coher_12) / np.pi

In [25]: fig, axs = plt.subplots(2, 1, sharex=True)

In [26]: coh.plot(ax=axs[0])
Out[26]: [<matplotlib.lines.Line2D at 0x7f04e6aa9510>]

In [27]: xphase.where(coh > 0.6).plot.line('o--', ax=axs[1])
Out[27]: [<matplotlib.lines.Line2D at 0x7f04e6bebd50>]
```

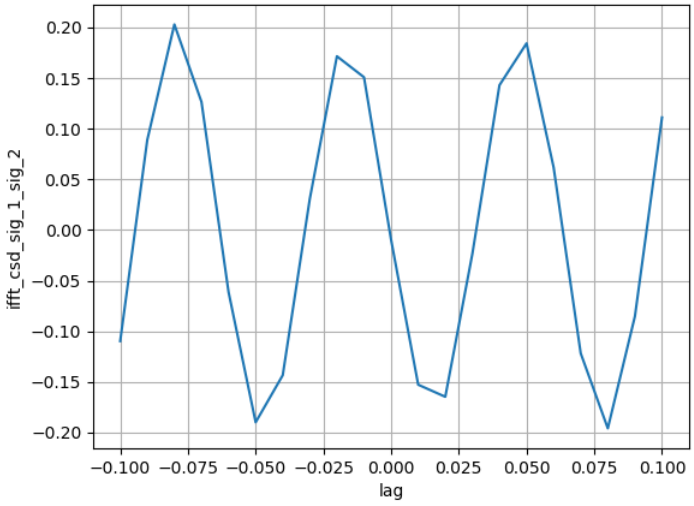
```
In [28]: axs[1].set(yticks=[-1, -0.5, 0, 0.5, 1]);

In [29]: plt.show()
```



In the future more convenient wrappers returning the coherence magnitude and cross-phase might be developed.

The cross-correlation is calculated similarly as γ, but with $\mathcal{F}^{-1}[\langle P_* \rangle]$, i.e. in the inverse-FFT domain. The `lag` coordinates are the inverse of the `frequency` coordinates.

```
In [30]: xcorr_12 = dsp.xcorrelation(sig_1, sig_2)

In [31]: xcorr_12.loc[-0.1:0.1].plot()
Out[31]: [<matplotlib.lines.Line2D at 0x7f04e51421d0>]

In [32]: plt.grid()

In [33]: plt.show()
```



A partially averaged counterpart to *coherence()* is *coherogram()* which uses a running average over `nrolling` FFT windows.

Help & reference

- *What's New*
- *API reference*

1.6 What's New

1.6.1 v0.0 (6 March 2018)

Initial release.

1.7 API reference

This page provides an auto-generated summary of xr-scipy's API. For more details and examples, refer to the relevant chapters in the main part of the documentation.

1.7.1 Top-level functions

gradient(f, coord[, edge_order])	Return the gradient of an xarray object.

xrscipy.gradient

xrscipy.**gradient**(*f*, *coord*, *edge_order=1*)

Return the gradient of an xarray object.

The gradient is computed using second order accurate central differences in the interior points and either first or second order accurate one-sides (forward or backwards) differences at the boundaries. The returned gradient hence has the same shape as the input array.

Parameters

- **f** (*xarray object*) – An xarray object containing samples of a scalar function.
- **coord** (*str*) – Coordinate along which the gradient is computed.
- **edge_order** (*{1, 2}, optional*) – Gradient is calculated using N-th order accurate differences at the boundaries. Default: 1.

Returns **gradient**

Return type xarray object

1.7.2 integrate

integrate.trapz(obj, coord)	Integrate along the given coordinate using the composite trapezoidal rule.

Continued on next page

Table 2 – continued from previous page

integrate.simps(obj, coord[, even])	Integrate y(x) using samples along the given coordinate and the composite Simpson's rule.
integrate.romb(obj, coord[, show])	Romberg integration using samples of a function.
integrate.cumtrapz(obj, coord)	Cumulatively integrate y(x) using the composite trapezoidal rule.

xrscipy.integrate.trapz

xrscipy.integrate.**trapz**(*obj*, *coord*)

Integrate along the given coordinate using the composite trapezoidal rule.

Integrate *y* (*x*) along given coordinate.

Parameters

- **obj** (*xarray object*) – Input array to integrate.
- **coord** (*string*) – The coordinate along which to integrate.

Returns **trapz** – Definite integral as approximated by trapezoidal rule.

Return type float

See also:

scipy.integrate.trapz() Original scipy implementation

References

xrscipy.integrate.simps

xrscipy.integrate.**simps**(*obj*, *coord*, *even='avg'*)

Integrate y(x) using samples along the given coordinate and the composite Simpson's rule. If x is None, spacing of dx is assumed.

If there are an even number of samples, N, then there are an odd number of intervals (N-1), but Simpson's rule requires an even number of intervals. The parameter 'even' controls how this is handled.

Parameters

- **obj** (*xarray object*) – Array to be integrated.
- **coord** (*string*) – The coordinate along which to integrate.
- **even** (*str {'avg', 'first', 'last'}, optional*) –

  'avg' [Average two results:1) use the first N-2 intervals with] a trapezoidal rule on the last interval and 2) use the last N-2 intervals with a trapezoidal rule on the first interval.

  'first' [Use Simpson's rule for the first N-2 intervals with] a trapezoidal rule on the last interval.

  'last' [Use Simpson's rule for the last N-2 intervals with a] trapezoidal rule on the first interval.

See also:

scipy.integrate.simps() Original scipy implementation

quad() adaptive quadrature using QUADPACK

romberg() adaptive Romberg quadrature

quadrature() adaptive Gaussian quadrature

fixed_quad() fixed-order Gaussian quadrature

dblquad() double integrals

tplquad() triple integrals

romb() integrators for sampled data

cumtrapz() cumulative integration for sampled data

ode() ODE integrators

odeint() ODE integrators

xrscipy.integrate.romb

xrscipy.integrate.**romb**(*obj*, *coord*, *show=False*)

Romberg integration using samples of a function.

Parameters

- **obj** (*xarray object*) – A vector of `2**k + 1` equally-spaced samples of a function.
- **coord** (*string*) – The coordinate along which to integrate.
- **show** (*bool, optional*) – When *y* is a single 1-D array, then if this argument is True print the table showing Richardson extrapolation from the samples. Default is False.

Returns **romb** – The integrated result for *coordiante*.

Return type xarray object

See also:

scipy.integrate.romb() Original scipy implementation

quad() adaptive quadrature using QUADPACK

romberg() adaptive Romberg quadrature

quadrature() adaptive Gaussian quadrature

fixed_quad() fixed-order Gaussian quadrature

dblquad() double integrals

tplquad() triple integrals

simps() integrators for sampled data

cumtrapz() cumulative integration for sampled data

ode() ODE integrators

odeint() ODE integrators

xrscipy.integrate.cumtrapz

xrscipy.integrate.**cumtrapz**(*obj*, *coord*)

Cumulatively integrate y(x) using the composite trapezoidal rule.

Parameters

- **obj** (*xarray object*) – Values to integrate.
- **coord** (*string*) – The coordinate along which to integrate.
- **initial** (*scalar, optional*) – If given, insert this value at the beginning of the returned result. Typically this value should be 0. Default is None, which means no value at `x[0]` is returned and *res* has one element less than *y* along the axis of integration.

Returns **res** – The result of cumulative integration of *y* along *coordiante*. If *initial* is None, the shape is such that the coordiante of integration has one less value than *y*. If *initial* is given, the shape is equal to that of *y*.

Return type xarray object

See also:

scipy.integrate.cumtrapz() Original scipy implementation

quad() adaptive quadrature using QUADPACK

romberg() adaptive Romberg quadrature

quadrature() adaptive Gaussian quadrature

fixed_quad() fixed-order Gaussian quadrature

dblquad() double integrals

tplquad() triple integrals

romb() integrators for sampled data

ode() ODE integrators

odeint() ODE integrators

1.7.3 interpolate

interpolate.interp1d(obj, coord[, kind, ...])	Interpolate a 1-D function.
interpolate.PchipInterpolator(obj, coord[, ...])	PCHIP 1-d monotonic cubic interpolation.
interpolate.Akima1DInterpolator(obj, coord)	Akima interpolator
interpolate.CubicSpline(obj, coord[, ...])	Cubic spline data interpolator.
interpolate.LinearNDInterpolator(obj, *coords)	Piecewise linear interpolant in N dimensions.
interpolate.NearestNDInterpolator(obj, *coords)	Nearest-neighbour interpolation in N dimensions.
interpolate.CloughTocher2DInterpolator(obj, ...)	Piecewise cubic, C1 smooth, curvature-minimizing interpolant in 2D.
interpolate.RegularGridInterpolator(obj, *coords)	Interpolation on a regular grid in arbitrary dimensions
interpolate.griddata(obj, coords, ...)	Wrapper for griddata.

xrscipy.interpolate.interp1d

xrscipy.interpolate.**interp1d**(*obj*, *coord*, *kind='linear'*, *copy=True*, *bounds_error=None*, *fill_value=nan*, *assume_sorted=False*)

Interpolate a 1-D function.

x and *y* are arrays of values used to approximate some function f: `y = f(x)`. This class returns a function whose call method uses interpolation to find the value of new points.

Note that calling *interp1d* with NaNs present in input values results in undefined behaviour.

Parameters

- **obj** (*xarray object*) –
- **coord** (*string*) – Coordinate along which to interpolate.
- **kind** (*str or int, optional*) – Specifies the kind of interpolation as a string ('linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'previous', 'next', where 'zero', 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of zeroth, first, second or third order; 'previous' and 'next' simply return the previous or next value of the point) or as an integer specifying the order of the spline interpolator to use. Default is 'linear'.
- **copy** (*bool, optional*) – If True, the class makes internal copies of x and y. If False, references to *x* and *y* are used. The default is to copy.
- **bounds_error** (*bool, optional*) – If True, a ValueError is raised any time interpolation is attempted on a value outside of the range of x (where extrapolation is necessary). If False, out of bounds values are assigned *fill_value*. By default, an error is raised unless `fill_value="extrapolate"`.
- **fill_value** (*array-like or (array-like, array_like) or "extrapolate", optional*) –
  - if a ndarray (or float), this value will be used to fill in for requested points outside of the data range. If not provided, then the default is NaN. The array-like must broadcast properly to the dimensions of the non-interpolation axes.
  - If a two-element tuple, then the first element is used as a fill value for `x_new < x[0]` and the second element is used for `x_new > x[-1]`. Anything that is not a 2-element tuple (e.g., list or ndarray, regardless of shape) is taken to be a single array-like argument meant to be used for both bounds as `below, above = fill_value, fill_value`.

    New in version 0.17.0.
  - If "extrapolate", then points outside the data range will be extrapolated.

    New in version 0.17.0.
- **assume_sorted** (*bool, optional*) – If False, values of *x* can be in any order and they are sorted first. If True, *x* has to be an array of monotonically increasing values.

See also:

scipy.interpolate.interp1d() Original scipy implementation

UnivariateSpline() An object-oriented wrapper of the FITPACK routines.

interp2d() 2-D interpolation

xrscipy.interpolate.**fill_value**

xrscipy.interpolate.**__call__**()

xrscipy.interpolate.PchipInterpolator

xrscipy.interpolate.**PchipInterpolator**(*obj*, *coord*, *extrapolate=None*)

PCHIP 1-d monotonic cubic interpolation.

x and y are arrays of values used to approximate some function f, with `y = f(x)`. The interpolant uses monotonic cubic splines to find the value of new points. (PCHIP stands for Piecewise Cubic Hermite Interpolating Polynomial).

Parameters

- **obj** (*xarray object*) –
- **coord** (*string*) – Coordinate along which to interpolate.
- **extrapolate** (*bool, optional*) – Whether to extrapolate to out-of-bounds points based on first and last intervals, or to return NaNs.

See also:

scipy.interpolate.PchipInterpolator() Original scipy implementation

xrscipy.interpolate.**__call__**()

xrscipy.interpolate.**derivative**()

xrscipy.interpolate.**antiderivative**()

xrscipy.interpolate.**roots**()

References

xrscipy.interpolate.Akima1DInterpolator

xrscipy.interpolate.**Akima1DInterpolator**(*obj*, *coord*)

Akima interpolator

Fit piecewise cubic polynomials, given vectors x and y. The interpolation method by Akima uses a continuously differentiable sub-spline built from piecewise cubic polynomials. The resultant curve passes through the given data points and will appear smooth and natural.

Parameters

- **obj** (*xarray object*) –
- **coord** (*string*) – Coordinate along which to interpolate.

See also:

scipy.interpolate.Akima1DInterpolator() Original scipy implementation

xrscipy.interpolate.**__call__**()

xrscipy.interpolate.**derivative**()

xrscipy.interpolate.**antiderivative**()

xrscipy.interpolate.**roots**()

References

[1] A new method of interpolation and smooth curve fitting based on local procedures. Hiroshi Akima, J. ACM, October 1970, 17(4), 589-602.

xrscipy.interpolate.CubicSpline

xrscipy.interpolate.**CubicSpline**(*obj*, *coord*, *bc_type='not-a-knot'*, *extrapolate=None*)

Cubic spline data interpolator.

Interpolate data with a piecewise cubic polynomial which is twice continuously differentiable[1]. The result is represented as a *PPoly* instance with breakpoints matching the given data.

Parameters

- **obj** (*xarray object*) –
- **coord** (*string*) – Coordinate along which to interpolate.
- **bc_type** (*string or 2-tuple, optional*) – Boundary condition type. Two additional equations, given by the boundary conditions, are required to determine all coefficients of polynomials on each segment[2].

  If *bc_type* is a string, then the specified condition will be applied at both ends of a spline. Available conditions are:

  – 'not-a-knot' (default): The first and second segment at a curve end are the same polynomial. It is a good default when there is no information on boundary conditions.

  – 'periodic': The interpolated functions is assumed to be periodic of period `x[-1] - x[0]`. The first and last value of *y* must be identical: `y[0] == y[-1]`. This boundary condition will result in `y'[0] == y'[-1]` and `y''[0] == y''[-1]`.

  – 'clamped': The first derivative at curves ends are zero. Assuming a 1D *y*, `bc_type=((1, 0.0), (1, 0.0))` is the same condition.

  – 'natural': The second derivative at curve ends are zero. Assuming a 1D *y*, `bc_type=((2, 0.0), (2, 0.0))` is the same condition.

  If *bc_type* is a 2-tuple, the first and the second value will be applied at the curve start and end respectively. The tuple values can be one of the previously mentioned strings (except 'periodic') or a tuple *(order, deriv_values)* allowing to specify arbitrary derivatives at curve ends:

  – *order*: the derivative order, 1 or 2.

  – *deriv_value*: array_like containing derivative values, shape must be the same as *y*, excluding `axis` dimension. For example, if *y* is 1D, then *deriv_value* must be a scalar. If *y* is 3D with the shape (n0, n1, n2) and axis=2, then *deriv_value* must be 2D and have the shape (n0, n1).
- **extrapolate** (*{bool, 'periodic', None}, optional*) – If bool, determines whether to extrapolate to out-of-bounds points based on first and last intervals, or to return NaNs. If 'periodic', periodic extrapolation is used. If None (default), `extrapolate` is set to 'periodic' for `bc_type='periodic'` and to True otherwise.

See also:

[1] Cubic Spline Interpolation on Wikiversity.
[2] Carl de Boor, "A Practical Guide to Splines", Springer-Verlag, 1978.

scipy.interpolate.CubicSpline() Original scipy implementation

xrscipy.interpolate.**x**
Breakpoints. The same `x` which was passed to the constructor.

Type ndarray, shape (n,)

xrscipy.interpolate.**c**
Coefficients of the polynomials on each segment. The trailing dimensions match the dimensions of *y*, excluding `axis`. For example, if *y* is 1-d, then `c[k, i]` is a coefficient for `(x-x[i])**(3-k)` on the segment between `x[i]` and `x[i+1]`.

Type ndarray, shape (4, n-1, ..)

xrscipy.interpolate.**axis**
Interpolation axis. The same axis which was passed to the constructor.

Type int

xrscipy.interpolate.**__call__**()

xrscipy.interpolate.**derivative**()

xrscipy.interpolate.**antiderivative**()

xrscipy.interpolate.**integrate**()

xrscipy.interpolate.**roots**()

References

xrscipy.interpolate.LinearNDInterpolator

xrscipy.interpolate.**LinearNDInterpolator**(*obj*, **coords*, *fill_value=np.nan*, *rescale=False*)
Piecewise linear interpolant in N dimensions.

New in version 0.9.

Parameters

- **obj** (*xarray object*) –
- ***coord** (*strings*) – Coordinates along which to interpolate.
- **fill_value** (*float, optional*) – Value used to fill in for requested points outside of the convex hull of the input points. If not provided, then the default is `nan`.
- **rescale** (*bool, optional*) – Rescale points to unit cube before performing interpolation. This is useful if some of the input dimensions have incommensurable units and differ by many orders of magnitude.

See also:

scipy.interpolate.LinearNDInterpolator() Original scipy implementation

xrscipy.interpolate.**__call__**()

Notes

The interpolant is constructed by triangulating the input data with Qhull[1], and on each triangle performing linear barycentric interpolation.

References

xrscipy.interpolate.NearestNDInterpolator

xrscipy.interpolate.**NearestNDInterpolator**(*obj*, **coords*)

Nearest-neighbour interpolation in N dimensions.

New in version 0.9.

Parameters

- **obj** (*xarray object*) –
- ***coord** (*strings*) – Coordinates along which to interpolate.
- **rescale** (*boolean, optional*) – Rescale points to unit cube before performing interpolation. This is useful if some of the input dimensions have incommensurable units and differ by many orders of magnitude.

  New in version 0.14.0.
- **tree_options** (*dict, optional*) – Options passed to the underlying `cKDTree`.

  New in version 0.17.0.

See also:

scipy.interpolate.NearestNDInterpolator() Original scipy implementation

xrscipy.interpolate.**__call__**()

Notes

Uses `scipy.spatial.cKDTree`

xrscipy.interpolate.CloughTocher2DInterpolator

xrscipy.interpolate.**CloughTocher2DInterpolator**(*obj*, **coords*, *fill_value=np.nan*, *tol=False*, *maxiter*, *rescale*)

Piecewise cubic, C1 smooth, curvature-minimizing interpolant in 2D.

New in version 0.9.

Parameters

- **obj** (*xarray object*) –
- ***coord** (*strings*) – Coordinates along which to interpolate.
- **fill_value** (*float, optional*) – Value used to fill in for requested points outside of the convex hull of the input points. If not provided, then the default is `nan`.
- **tol** (*float, optional*) – Absolute/relative tolerance for gradient estimation.

[1] http://www.qhull.org/

- **maxiter** (*int, optional*) – Maximum number of iterations in gradient estimation.
- **rescale** (*bool, optional*) – Rescale points to unit cube before performing interpolation. This is useful if some of the input dimensions have incommensurable units and differ by many orders of magnitude.

See also:

scipy.interpolate.CloughTocher2DInterpolator() Original scipy implementation

xrscipy.interpolate.**__call__**()

Notes

The interpolant is constructed by triangulating the input data with Qhull[1], and constructing a piecewise cubic interpolating Bezier polynomial on each triangle, using a Clough-Tocher scheme [CT]. The interpolant is guaranteed to be continuously differentiable.

The gradients of the interpolant are chosen so that the curvature of the interpolating surface is approximatively minimized. The gradients necessary for this are estimated using the global algorithm described in [Nielson83,Renka84]_.

References

xrscipy.interpolate.RegularGridInterpolator

xrscipy.interpolate.**RegularGridInterpolator**(*obj*, **coords*, *method='linear'*, *bounds_error=True*, *fill_value=nan*)

Interpolation on a regular grid in arbitrary dimensions

The data must be defined on a regular grid; the grid spacing however may be uneven. Linear and nearest-neighbour interpolation are supported. After setting up the interpolator object, the interpolation method (*linear* or *nearest*) may be chosen at each evaluation.

Parameters

- **obj** (*xarray object*) –
- ***coord** (*strings*) – Coordinates along which to interpolate.
- **method** (*str, optional*) – The method of interpolation to perform. Supported are "linear" and "nearest". This parameter will become the default for the object's __call__ method. Default is "linear".
- **bounds_error** (*bool, optional*) – If True, when interpolated values are requested outside of the domain of the input data, a ValueError is raised. If False, then *fill_value* is used.
- **fill_value** (*number, optional*) – If provided, the value to use for points outside of the interpolation domain. If None, values outside the domain are extrapolated.

See also:

scipy.interpolate.RegularGridInterpolator() Original scipy implementation

NearestNDInterpolator() Nearest neighbour interpolation on unstructured data in N dimensions

LinearNDInterpolator() Piecewise linear interpolant on unstructured data in N dimensions

[1] http://www.qhull.org/

xrscipy.interpolate.**__call__**()

Notes

Contrary to LinearNDInterpolator and NearestNDInterpolator, this class avoids expensive triangulation of the input data by taking advantage of the regular grid structure.

If any of *points* have a dimension of size 1, linear interpolation will return an array of *nan* values. Nearest-neighbor interpolation will work as usual in this case.

New in version 0.14.

References

xrscipy.interpolate.griddata

xrscipy.interpolate.**griddata**(*obj*, *coords*, *new_coords*, ***kwargs*)

Wrapper for griddata. coords: sequence of strings. new_coords: the same length of xr.DataArrays.

1.7.4 fft

fft.fft(a, coord[, n, norm])	Compute the one-dimensional discrete Fourier Transform.
fft.ifft(a, coord[, n, norm])	Compute the one-dimensional inverse discrete Fourier Transform.
fft.rfft(a, coord[, n, norm])	Compute the one-dimensional discrete Fourier Transform for real input.
fft.irfft(a, coord[, n, norm])	Compute the inverse of the n-point DFT for real input.
fft.fftn(a, *coords[, shape, norm])	Compute the N-dimensional discrete Fourier Transform.
fft.ifftn(a, *coords[, shape, norm])	Compute the N-dimensional inverse discrete Fourier Transform.
fft.rfftn(a, *coords[, shape, norm])	Compute the N-dimensional discrete Fourier Transform for real input.
fft.irfftn(a, *coords[, shape, norm])	Compute the inverse of the N-dimensional FFT of real input.
fft.hfft(a, coord[, n, norm])	Compute the FFT of a signal which has Hermitian symmetry (real spectrum).
fft.ihfft(a, coord[, n, norm])	Compute the inverse FFT of a signal which has Hermitian symmetry.

xrscipy.fft.fft

xrscipy.fft.**fft**(*a*, *coord*, *n=None*, *norm=None*)

Compute the one-dimensional discrete Fourier Transform.

This function computes the one-dimensional *n*-point discrete Fourier Transform (DFT) with the efficient Fast Fourier Transform (FFT) algorithm [CT].

Parameters

- **a** (*xarray object*) – The data tp transform.coord : string The axis along which the transform is applied. . The coordinate must be evenly spaced.

- **n** (*int, optional*) – Length of the transformed axis of the output. If *n* is smaller than the length of the input, the input is cropped. If it is larger, the input is padded with zeros. If *n* is not given, the length of the input along the axis specified by *axis* is used.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coord indicated by *coord*, or the last one if *coord* is not specified.

Return type complex xarray object

See also:

numpy.fft.fft() Original numpy implementation

numpy.fft() for definition of the DFT and conventions used.

ifft() The inverse of *fft*.

fft2() The two-dimensional FFT.

fftn() The *n*-dimensional FFT.

rfftn() The *n*-dimensional FFT of real input.

fftfreq() Frequency bins for given FFT parameters.

Raises **IndexError** – if *axes* is larger than the last axis of *a*.

References

xrscipy.fft.ifft

xrscipy.fft.**ifft**(*a*, *coord*, *n=None*, *norm=None*)

Compute the one-dimensional inverse discrete Fourier Transform.

This function computes the inverse of the one-dimensional *n*-point discrete Fourier transform computed by *fft*. In other words, `ifft(fft(a)) == a` to within numerical accuracy. For a general description of the algorithm and definitions, see *numpy.fft*.

The input should be ordered in the same way as is returned by *fft*, i.e.,

- `a[0]` should contain the zero frequency term,
- `a[1:n//2]` should contain the positive-frequency terms,
- `a[n//2 + 1:]` should contain the negative-frequency terms, in increasing order starting from the most negative frequency.

Parameters

- **a** (*xarray object*) – The data tp transform.coord : string The axis along which the transform is applied. . The coordinate must be evenly spaced.
- **n** (*int, optional*) – Length of the transformed axis of the output. If *n* is smaller than the length of the input, the input is cropped. If it is larger, the input is padded with zeros. If *n* is not given, the length of the input along the axis specified by *axis* is used. See notes about padding issues.

- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coord indicated by *coord*, or the last one if *coord* is not specified.

Return type complex xarray object

See also:

numpy.fft.ifft() Original numpy implementation

numpy.fft() An introduction, with definitions and general explanations.

fft() The one-dimensional (forward) FFT, of which *ifft* is the inverse

ifft2() The two-dimensional inverse FFT.

ifftn() The n-dimensional inverse FFT.

Raises **IndexError** – If *axes* is larger than the last axis of *a*.

xrscipy.fft.rfft

xrscipy.fft.**rfft**(*a*, *coord*, *n=None*, *norm=None*)

Compute the one-dimensional discrete Fourier Transform for real input.

This function computes the one-dimensional *n*-point discrete Fourier Transform (DFT) of a real-valued array by means of an efficient algorithm called the Fast Fourier Transform (FFT).

Parameters

- **a** (*xarray object*) – The data tp transform.coord : string The axis along which the transform is applied. . The coordinate must be evenly spaced.
- **n** (*int, optional*) – Number of points along transformation axis in the input to use. If *n* is smaller than the length of the input, the input is cropped. If it is larger, the input is padded with zeros. If *n* is not given, the length of the input along the axis specified by *axis* is used.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coord indicated by *coord*, or the last one if *coord* is not specified. If *n* is even, the length of the transformed coord is `(n/2)+1`. If *n* is odd, the length is `(n+1)/2`.

Return type complex xarray object

See also:

numpy.fft.rfft() Original numpy implementation

numpy.fft() For definition of the DFT and conventions used.

irfft() The inverse of *rfft*.

fft() The one-dimensional FFT of general (complex) input.

fftn() The *n*-dimensional FFT.

rfftn() The *n*-dimensional FFT of real input.

Raises **IndexError** – If *axis* is larger than the last axis of *a*.

xrscipy.fft.irfft

xrscipy.fft.**irfft**(*a*, *coord*, *n=None*, *norm=None*)

Compute the inverse of the n-point DFT for real input.

This function computes the inverse of the one-dimensional *n*-point discrete Fourier Transform of real input computed by *rfft*. In other words, `irfft(rfft(a), len(a)) == a` to within numerical accuracy. (See Notes below for why `len(a)` is necessary here.)

The input is expected to be in the form returned by *rfft*, i.e. the real zero-frequency term followed by the complex positive frequency terms in order of increasing frequency. Since the discrete Fourier Transform of real input is Hermitian-symmetric, the negative frequency terms are taken to be the complex conjugates of the corresponding positive frequency terms.

Parameters

- **a** (*xarray object*) – The data tp transform.coord : string The axis along which the transform is applied. . The coordinate must be evenly spaced.
- **n** (*int, optional*) – Length of the transformed axis of the output. For *n* output points, `n//2+1` input points are necessary. If the input is longer than this, it is cropped. If it is shorter than this, it is padded with zeros. If *n* is not given, it is determined from the length of the input along the axis specified by *axis*.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coord indicated by *coord*, or the last one if *coord* is not specified. The length of the transformed coord is *n*, or, if *n* is not given, `2*(m-1)` where `m` is the length of the transformed coord of the input. To get an odd number of output points, *n* must be specified.

Return type xarray object

See also:

numpy.fft.irfft() Original numpy implementation

numpy.fft() For definition of the DFT and conventions used.

rfft() The one-dimensional FFT of real input, of which *irfft* is inverse.

fft() The one-dimensional FFT.

irfft2() The inverse of the two-dimensional FFT of real input.

irfftn() The inverse of the *n*-dimensional FFT of real input.

Raises **IndexError** – If *axis* is larger than the last axis of *a*.

xrscipy.fft.fftn

xrscipy.fft.**fftn**(*a*, **coords*, *shape=None*, *norm=None*)

Compute the N-dimensional discrete Fourier Transform.

This function computes the *N*-dimensional discrete Fourier Transform over any number of axes in an *M*-dimensional array by means of the Fast Fourier Transform (FFT).

Parameters

- **a** (*xarray object*) – Object which the transform is applied.
- **s** (*mapping from coords to size, optional*) – The shape of the result.coords : string Coordinates along which the transform is applied. The coordinate must be evenly spaced.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coords indicated by *coords*, or by a combination of *s* and *a*, as explained in the parameters section above.

Return type complex xarray object

See also:

numpy.fft.fftn() Original numpy implementation

numpy.fft() Overall view of discrete Fourier transforms, with definitions and conventions used.

ifftn() The inverse of *fftn*, the inverse *n*-dimensional FFT.

fft() The one-dimensional FFT, with definitions and conventions used.

rfftn() The *n*-dimensional FFT of real input.

fft2() The two-dimensional FFT.

fftshift() Shifts zero-frequency terms to centre of array

Raises

- **ValueError** – If *s* and *axes* have different length.
- **IndexError** – If an element of *axes* is larger than than the number of axes of *a*.

xrscipy.fft.ifftn

xrscipy.fft.**ifftn**(*a*, **coords*, *shape=None*, *norm=None*)

Compute the N-dimensional inverse discrete Fourier Transform.

This function computes the inverse of the N-dimensional discrete Fourier Transform over any number of axes in an M-dimensional array by means of the Fast Fourier Transform (FFT). In other words, `ifftn(fftn(a)) == a` to within numerical accuracy. For a description of the definitions and conventions used, see *numpy.fft*.

The input, analogously to *ifft*, should be ordered in the same way as is returned by *fftn*, i.e. it should have the term for zero frequency in all axes in the low-order corner, the positive frequency terms in the first half of all axes, the term for the Nyquist frequency in the middle of all axes and the negative frequency terms in the second half of all axes, in order of decreasingly negative frequency.

Parameters

- **a** (*xarray object*) – Object which the transform is applied.
- **s** (*mapping from coords to size, optional*) – The shape of the result.coords : string Coordinates along which the transform is applied. The coordinate must be evenly spaced.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coords indicated by *coords*, or by a combination of *s* or *a*, as explained in the parameters section above.

Return type complex xarray object

See also:

numpy.fft.ifftn() Original numpy implementation

numpy.fft() Overall view of discrete Fourier transforms, with definitions and conventions used.

fftn() The forward *n*-dimensional FFT, of which *ifftn* is the inverse.

ifft() The one-dimensional inverse FFT.

ifft2() The two-dimensional inverse FFT.

ifftshift() Undoes *fftshift*, shifts zero-frequency terms to beginning of array.

Raises

- **ValueError** – If *s* and *axes* have different length.
- **IndexError** – If an element of *axes* is larger than than the number of axes of *a*.

xrscipy.fft.rfftn

xrscipy.fft.**rfftn**(*a*, **coords*, *shape=None*, *norm=None*)

Compute the N-dimensional discrete Fourier Transform for real input.

This function computes the N-dimensional discrete Fourier Transform over any number of axes in an M-dimensional real array by means of the Fast Fourier Transform (FFT). By default, all axes are transformed, with the real transform performed over the last axis, while the remaining transforms are complex.

Parameters

- **a** (*xarray object*) – Object which the transform is applied.
- **s** (*mapping from coords to size, optional*) – The shape of the result.coords : string Coordinates along which the transform is applied. The coordinate must be evenly spaced.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coords indicated by *coords*, or by a combination of *s* and *a*, as explained in the parameters section above. The length of the last coord transformed will be `s[-1]//2+1`, while the remaining transformed coords will have lengths according to *s*, or unchanged from the input.

Return type complex xarray object

See also:

numpy.fft.rfftn() Original numpy implementation

irfftn() The inverse of *rfftn*, i.e. the inverse of the n-dimensional FFT of real input.

fft() The one-dimensional FFT, with definitions and conventions used.

rfft() The one-dimensional FFT of real input.

fftn() The n-dimensional FFT.

rfft2() The two-dimensional FFT of real input.

Raises

- **ValueError** – If *s* and *axes* have different length.
- **IndexError** – If an element of *axes* is larger than than the number of axes of *a*.

xrscipy.fft.irfftn

xrscipy.fft.**irfftn**(*a*, **coords*, *shape=None*, *norm=None*)

Compute the inverse of the N-dimensional FFT of real input.

This function computes the inverse of the N-dimensional discrete Fourier Transform for real input over any number of axes in an M-dimensional array by means of the Fast Fourier Transform (FFT). In other words, `irfftn(rfftn(a), a.shape) == a` to within numerical accuracy. (The `a.shape` is necessary like `len(a)` is for *irfft*, and for the same reason.)

The input should be ordered in the same way as is returned by *rfftn*, i.e. as for *irfft* for the final transformation axis, and as for *ifftn* along all the other axes.

Parameters

- **a** (*xarray object*) – Object which the transform is applied.
- **s** (*mapping from coords to size, optional*) – The shape of the result.coords : string Coordinates along which the transform is applied. The coordinate must be evenly spaced.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coords indicated by *coords*, or by a combination of *s* or *a*, as explained in the parameters section above. The length of each transformed coord is as given by the corresponding element of *s*, or the length of the input in every coord except for the last one if *s* is not given. In the final transformed coord the length of the output when *s* is not given is `2*(m-1)` where `m` is the length of the final transformed coord of the input. To get an odd number of output points in the final coord, *s* must be specified.

Return type xarray object

See also:

numpy.fft.irfftn() Original numpy implementation

rfftn() The forward n-dimensional FFT of real input, of which *ifftn* is the inverse.

fft() The one-dimensional FFT, with definitions and conventions used.

irfft() The inverse of the one-dimensional FFT of real input.

irfft2() The inverse of the two-dimensional FFT of real input.

Raises

- **ValueError** – If *s* and *axes* have different length.
- **IndexError** – If an element of *axes* is larger than than the number of axes of *a*.

xrscipy.fft.hfft

xrscipy.fft.**hfft**(*a*, *coord*, *n=None*, *norm=None*)

Compute the FFT of a signal which has Hermitian symmetry (real spectrum).

Parameters

- **a** (*xarray object*) – The data tp transform.coord : string The axis along which the transform is applied. . The coordinate must be evenly spaced.
- **n** (*int, optional*) – Length of the transformed axis of the output. For *n* output points, `n//2+1` input points are necessary. If the input is longer than this, it is cropped. If it is shorter than this, it is padded with zeros. If *n* is not given, it is determined from the length of the input along the axis specified by *axis*.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coord indicated by *coord*, or the last one if *coord* is not specified. The length of the transformed coord is *n*, or, if *n* is not given, `2*(m-1)` where `m` is the length of the transformed coord of the input. To get an odd number of output points, *n* must be specified.

Return type xarray object

See also:

numpy.fft.hfft() Original numpy implementation

rfft() Compute the one-dimensional FFT for real input.

ihfft() The inverse of *hfft*.

Raises **IndexError** – If *axis* is larger than the last axis of *a*.

xrscipy.fft.ihfft

xrscipy.fft.**ihfft**(*a*, *coord*, *n=None*, *norm=None*)

Compute the inverse FFT of a signal which has Hermitian symmetry.

Parameters

- **a** (*xarray object*) – The data tp transform.coord : string The axis along which the transform is applied. . The coordinate must be evenly spaced.
- **n** (*int, optional*) – Length of the inverse FFT. Number of points along transformation axis in the input to use. If *n* is smaller than the length of the input, the input is cropped. If it is larger, the input is padded with zeros. If *n* is not given, the length of the input along the axis specified by *axis* is used.
- **norm** (*{None, "ortho"}, optional*) – New in version 1.10.0.

  Normalization mode (see *numpy.fft*). Default is None.

Returns **out** – The truncated or zero-padded input, transformed along the coord indicated by *coord*, or the last one if *coord* is not specified. If *n* is even, the length of the transformed coord is `(n/2)+1`. If *n* is odd, the length is `(n+1)/2`.

Return type complex xarray object

See also:

numpy.fft.ihfft() Original numpy implementation

1.7.5 fftpack

fftpack.fft(obj, coord[, n])	Return discrete Fourier transform of real or complex sequence.
fftpack.ifft(obj, coord[, n])	Return discrete inverse Fourier transform of real or complex sequence.
fftpack.rfft(obj, coord[, n])	Discrete Fourier transform of a real sequence.
fftpack.irfft(obj, coord[, n])	Return inverse discrete Fourier transform of real sequence x.
fftpack.dct(obj, coord[, type, n, norm])	Return the Discrete Cosine Transform of arbitrary type sequence x.
fftpack.idct(obj, coord[, type, n, norm])	Return the Inverse Discrete Cosine Transform of an arbitrary type sequence.
fftpack.dst(obj, coord[, type, n, norm])	Return the Discrete Sine Transform of arbitrary type sequence x.
fftpack.idst(obj, coord[, type, n, norm])	Return the Inverse Discrete Sine Transform of an arbitrary type sequence.

xrscipy.fftpack.fft

xrscipy.fftpack.**fft**(*obj*, *coord*, *n=None*)

Return discrete Fourier transform of real or complex sequence.

The returned complex array contains `y(0), y(1),..., y(n-1)` where

`y(j) = (x * exp(-2*pi*sqrt(-1)*j*np.arange(n)/n)).sum()`.

Parameters

- **obj** (*xarray object*) – Array to Fourier transform.
- **coord** (*string*) – Coordinate along which the fft's are computed. The coordinate must be evenly spaced.
- **n** (*int, optional*) – Length of the Fourier transform. If `n < x.shape[axis]`, *x* is truncated. If `n > x.shape[axis]`, *x* is zero-padded. The default results in `n = x.shape[axis]`.

Returns

z –

with the elements:

```
[y(0),y(1),..,y(n/2),y(1-n/2),...,y(-1)]         if n is even
[y(0),y(1),..,y((n-1)/2),y(-(n-1)/2),...,y(-1)]  if n is odd
```

where:

```
y(j) = sum[k=0..n-1] x[k] * exp(-sqrt(-1)*j*k* 2*pi/n), j = 0..n-1
```

Return type complex xarray object

See also:

scipy.fftpack.fft() Original scipy implementation

ifft() Inverse FFT

rfft() FFT of a real sequence

xrscipy.fftpack.ifft

xrscipy.fftpack.**ifft**(*obj*, *coord*, *n=None*)

Return discrete inverse Fourier transform of real or complex sequence.

The returned complex array contains `y(0), y(1),..., y(n-1)` where

`y(j) = (x * exp(2*pi*sqrt(-1)*j*np.arange(n)/n)).mean().`

Parameters

- **obj** (*xarray object*) – Transformed data to invert.
- **coord** (*string*) – Coordinate along which the ifft's are computed. The coordinate must be evenly spaced.
- **n** (*int, optional*) – Length of the inverse Fourier transform. If `n < x.shape[axis]`, *x* is truncated. If `n > x.shape[axis]`, *x* is zero-padded. The default results in `n = x.shape[axis]`.

Returns **ifft** – The inverse discrete Fourier transform.

Return type xarray object of floats

See also:

scipy.fftpack.ifft() Original scipy implementation

fft() Forward FFT

xrscipy.fftpack.rfft

xrscipy.fftpack.**rfft**(*obj*, *coord*, *n=None*)

Discrete Fourier transform of a real sequence.

Parameters

- **obj** (*xarray object*) – The data to transform.
- **coord** (*string*) – The axis along which the transform is applied. The default is the
- – The coordinate must be evenly spaced.
- **n** (*int, optional*) – Defines the length of the Fourier transform. If *n* is not specified (the default) then `n = x.shape[axis]`. If `n < x.shape[axis]`, *x* is truncated, if `n > x.shape[axis]`, *x* is zero-padded.

Returns

z –

The returned real array contains:

```
[y(0),Re(y(1)),Im(y(1)),...,Re(y(n/2))]              if n is even
[y(0),Re(y(1)),Im(y(1)),...,Re(y(n/2)),Im(y(n/2))]   if n is odd
```

where:

```
y(j) = sum[k=0..n-1] x[k] * exp(-sqrt(-1)*j*k*2*pi/n)
j = 0..n-1
```

Return type real xarray object

See also:

scipy.fftpack.rfft() Original scipy implementation

xrscipy.fftpack.irfft

xrscipy.fftpack.**irfft**(*obj*, *coord*, *n=None*)

Return inverse discrete Fourier transform of real sequence x.

The contents of *x* are interpreted as the output of the *rfft* function.

Parameters

- **obj** (*xarray object*) – Transformed data to invert.
- **coord** (*string*) – Coordinate along which the ifft's are computed. The coordinate must be evenly spaced.
- **n** (*int, optional*) – Length of the inverse Fourier transform. If n < x.shape[axis], x is truncated. If n > x.shape[axis], x is zero-padded. The default results in n = x.shape[axis].

Returns **irfft** – The inverse discrete Fourier transform.

Return type xarray object of floats

See also:

scipy.fftpack.irfft() Original scipy implementation

xrscipy.fftpack.dct

xrscipy.fftpack.**dct**(*obj*, *coord*, *type=2*, *n=None*, *norm=None*)

Return the Discrete Cosine Transform of arbitrary type sequence x.

Parameters

- **obj** (*xarray object*) – The input array.
- **coord** (*string*) – Coordinate along which the dct is computed. The coordinate must be evenly spaced.
- **type** (*{1, 2, 3, 4}, optional*) – Type of the DCT (see Notes). Default type is 2.
- **n** (*int, optional*) – Length of the transform. If `n < x.shape[axis]`, *x* is truncated. If `n > x.shape[axis]`, *x* is zero-padded. The default results in `n = x.shape[axis]`.
- **norm** (*{None, 'ortho'}, optional*) – Normalization mode (see Notes). Default is None.

Returns **y** – The transformed input array.

Return type xarray object of real

See also:

scipy.fftpack.dct() Original scipy implementation

idct() Inverse DCT

References

xrscipy.fftpack.idct

xrscipy.fftpack.**idct**(*obj*, *coord*, *type=2*, *n=None*, *norm=None*)

Return the Inverse Discrete Cosine Transform of an arbitrary type sequence.

Parameters

- **obj** (*xarray object*) – The input array.
- **coord** (*string*) – Coordinate along which the idct is computed. The coordinate must be evenly spaced.
- **type** (*{1, 2, 3, 4}, optional*) – Type of the DCT (see Notes). Default type is 2.
- **n** (*int, optional*) – Length of the transform. If `n < x.shape[axis]`, *x* is truncated. If `n > x.shape[axis]`, *x* is zero-padded. The default results in `n = x.shape[axis]`.
- **norm** (*{None, 'ortho'}, optional*) – Normalization mode (see Notes). Default is None.

Returns **idct** – The transformed input array.

Return type xarray object of real

See also:

scipy.fftpack.idct() Original scipy implementation

dct() Forward DCT

xrscipy.fftpack.dst

xrscipy.fftpack.**dst**(*obj*, *coord*, *type=2*, *n=None*, *norm=None*)

Return the Discrete Sine Transform of arbitrary type sequence x.

Parameters

- **obj** (*xarray object*) – The input array.
- **coord** (*string*) – Coordinate along which the dst is computed. The coordinate must be evenly spaced.
- **type** (*{1, 2, 3, 4}, optional*) – Type of the DST (see Notes). Default type is 2.
- **n** (*int, optional*) – Length of the transform. If `n < x.shape[axis]`, *x* is truncated. If `n > x.shape[axis]`, *x* is zero-padded. The default results in `n = x.shape[axis]`.
- **norm** (*{None, 'ortho'}, optional*) – Normalization mode (see Notes). Default is None.

Returns **dst** – The transformed input array.

Return type xarray object of reals

See also:

`scipy.fftpack.dst()` Original scipy implementation

`idst()` Inverse DST

References

xrscipy.fftpack.idst

xrscipy.fftpack.**idst**(*obj*, *coord*, *type=2*, *n=None*, *norm=None*)

Return the Inverse Discrete Sine Transform of an arbitrary type sequence.

Parameters

- **obj** (*xarray object*) – The input array.
- **coord** (*string*) – Coordinate along which the idst is computed. The coordinate must be evenly spaced.
- **type** (*{1, 2, 3, 4}, optional*) – Type of the DST (see Notes). Default type is 2.
- **n** (*int, optional*) – Length of the transform. If `n < x.shape[axis]`, *x* is truncated. If `n > x.shape[axis]`, *x* is zero-padded. The default results in `n = x.shape[axis]`.
- **norm** (*{None, 'ortho'}, optional*) – Normalization mode (see Notes). Default is None.

Returns **idst** – The transformed input array.

Return type xarray object of real

See also:

`scipy.fftpack.idst()` Original scipy implementation

`dst()` Forward DST

1.7.6 Spectral (FFT) analysis

signal.csd(darray, other_darray[, fs, ...])	Estimate the cross power spectral density, Pxy, using Welch's method.
signal.psd(darray[, fs, seglen, ...])	Calculate the power spectral density.
signal.coherence(darray, other_darray[, fs, ...])	Calculate the coherence as <CSD> / sqrt(<PSD1> * <PSD2>)
signal.xcorrelation(darray, other_darray[, ...])	Calculate the crosscorrelation.
signal.crossspectrogram(darray, other_darray)	Calculate the cross spectrogram.
signal.spectrogram(darray[, fs, seglen, ...])	Calculate the spectrogram using crossspectrogram applied to the same data
signal.coherogram(darray, other_darray[, ...])	Calculate the coherogram
signal.hilbert(darray[, N, dim])	Compute the analytic signal, using the Hilbert transform.

xrscipy.signal.csd

xrscipy.signal.**csd**(*darray*, *other_darray*, *fs=None*, *seglen=None*, *overlap_ratio=0.5*, *window='hann'*, *nperseg=256*, *noverlap=None*, *nfft=None*, *detrend='constant'*, *return_onesided=True*, *dim=None*, *scaling='density'*, *mode='psd'*)

Estimate the cross power spectral density, Pxy, using Welch's method.

Parameters

- **darray** (*xarray*) – Series of measurement values
- **other_darray** (*xarray*) – Series of measurement values fs : float, optional Sampling frequency of the *darray* and *other_darray* (time) series. If not specified, will be calculated it from the sampling step of the specified (or only) dimension.
- **window** (*str or tuple or array_like, optional*) – Desired window to use. If *window* is a string or tuple, it is passed to *get_window* to generate the window values, which are DFT-even by default. See *get_window* for a list of windows and required parameters. If *window* is array_like it will be used directly as the window and its length must be nperseg. Defaults to a Hann window.
- **seglen** (*float, optional*) – Segment length (i.e. nperseg) in units of the used (e.g. time) dimension.
- **nperseg** (*int, optional*) – Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is array_like, is set to the length of the window.
- **noverlap** (*int, optional*) – Number of points to overlap between segments. If *None*, `noverlap = np.rint(nperseg * overlap_ratio)`. Defaults to *None*.
- **overlap_ratio** (*float, optional*) – Used to calculate noverlap, if it is not specified (see above). Defaults to 0.5.
- **nfft** (*int, optional*) – Length of the FFT used, if a zero padded FFT is desired. If *None*, the FFT length is *nperseg*. Defaults to *None*.
- **detrend** (str or function or *False*, optional) – Specifies how to detrend each segment. If *detrend* is a string, it is passed as the *type* argument to the *detrend* function. If it is a function, it takes a segment and returns a detrended segment. If *detrend* is *False*, no detrending is done. Defaults to 'constant'.
- **return_onesided** (*bool, optional*) – If *True*, return a one-sided spectrum for real data. If *False* return a two-sided spectrum. Defaults to *True*, but for complex data, a two-sided spectrum is always returned.
- **dim** (*str, optional*) – Dimension along which the FFT is computed and sampling step calculated. If the signal is 1D, uses the only dimension, otherwise must be specified.
- **scaling** (*{ 'density', 'spectrum' }, optional*) – Selects between computing the cross spectral density ('density') where *Pxy* has units of V**2/Hz and computing the cross spectrum ('spectrum') where *Pxy* has units of V**2, if *darray* and *other_darray* are measured in V and *fs* is measured in Hz. Defaults to 'density'.
- **mode** (*str*) – Defines what kind of return values are expected. Options are ['psd', 'complex', 'magnitude', 'angle', 'phase']. 'complex' is equivalent to the output of *stft* with no padding or boundary extension. 'magnitude' returns the absolute magnitude of the STFT. 'angle' and 'phase' return the complex angle of the STFT, with and without unwrapping, respectively.

Returns **Pxy** – Cross spectral density or cross power spectrum with frequency dimension.

Return type xarray.DataArray

Notes

By convention, Pxy is computed with the conjugate FFT of *darray* multiplied by the FFT of *other_darray*. If the input series differ in length, the shorter series will be zero-padded to match. An appropriate amount of overlap will depend on the choice of window and on your requirements. For the default Hann window an overlap of 50% is a reasonable trade off between accurately estimating the signal power, while not over counting any of the data. Narrower windows may require a larger overlap.

References

xrscipy.signal.psd

xrscipy.signal.**psd**(*darray*, *fs=None*, *seglen=None*, *overlap_ratio=0.5*, *window='hann'*, *nperseg=256*, *noverlap=None*, *nfft=None*, *detrend='constant'*, *return_onesided=True*, *scaling='density'*, *dim=None*, *mode='psd'*)

Calculate the power spectral density.

Parameters

- **darray** (*xarray*) – Series of measurement values fs : float, optional Sampling frequency of the *darray* and *other_darray* (time) series. If not specified, will be calculated it from the sampling step of the specified (or only) dimension.
- **window** (*str or tuple or array_like, optional*) – Desired window to use. If *window* is a string or tuple, it is passed to *get_window* to generate the window values, which are DFT-even by default. See *get_window* for a list of windows and required parameters. If *window* is array_like it will be used directly as the window and its length must be nperseg. Defaults to a Hann window.
- **seglen** (*float, optional*) – Segment length (i.e. nperseg) in units of the used (e.g. time) dimension.
- **nperseg** (*int, optional*) – Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is array_like, is set to the length of the window.
- **noverlap** (*int, optional*) – Number of points to overlap between segments. If *None*, `noverlap = np.rint(nperseg * overlap_ratio)`. Defaults to *None*.
- **overlap_ratio** (*float, optional*) – Used to calculate noverlap, if it is not specified (see above). Defaults to 0.5.
- **nfft** (*int, optional*) – Length of the FFT used, if a zero padded FFT is desired. If *None*, the FFT length is *nperseg*. Defaults to *None*.
- **detrend** (str or function or *False*, optional) – Specifies how to detrend each segment. If *detrend* is a string, it is passed as the *type* argument to the *detrend* function. If it is a function, it takes a segment and returns a detrended segment. If *detrend* is *False*, no detrending is done. Defaults to 'constant'.
- **return_onesided** (*bool, optional*) – If *True*, return a one-sided spectrum for real data. If *False* return a two-sided spectrum. Defaults to *True*, but for complex data, a two-sided spectrum is always returned.
- **dim** (*str, optional*) – Dimension along which the FFT is computed and sampling step calculated. If the signal is 1D, uses the only dimension, otherwise must be specified.
- **scaling** (*{ 'density', 'spectrum' }, optional*) – Selects between computing the cross spectral density ('density') where *Pxy* has units of V**2/Hz and computing

the cross spectrum ('spectrum') where *Pxy* has units of V**2, if *darray* and *other_darray* are measured in V and *fs* is measured in Hz. Defaults to 'density'.

- **mode** (*str*) – Defines what kind of return values are expected. Options are ['psd', 'complex', 'magnitude', 'angle', 'phase']. 'complex' is equivalent to the output of *stft* with no padding or boundary extension. 'magnitude' returns the absolute magnitude of the STFT. 'angle' and 'phase' return the complex angle of the STFT, with and without unwrapping, respectively.

Returns **Pxx** – Power spectrum density of 'darray'.

Return type xarray.DataArray

xrscipy.signal.coherence

xrscipy.signal.**coherence**(*darray*, *other_darray*, *fs=None*, *seglen=None*, *overlap_ratio=0.5*, *window='hann'*, *nperseg=256*, *noverlap=None*, *nfft=None*, *detrend='constant'*, *dim=None*)

Calculate the coherence as <CSD> / sqrt(<PSD1> * <PSD2>)

Parameters

- **darray** (*xarray*) – Series of measurement values
- **other_darray** (*xarray*) – Series of measurement values fs : float, optional Sampling frequency of the *darray* and *other_darray* (time) series. If not specified, will be calculated it from the sampling step of the specified (or only) dimension.
- **window** (*str or tuple or array_like, optional*) – Desired window to use. If *window* is a string or tuple, it is passed to *get_window* to generate the window values, which are DFT-even by default. See *get_window* for a list of windows and required parameters. If *window* is array_like it will be used directly as the window and its length must be nperseg. Defaults to a Hann window.
- **seglen** (*float, optional*) – Segment length (i.e. nperseg) in units of the used (e.g. time) dimension.
- **nperseg** (*int, optional*) – Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is array_like, is set to the length of the window.
- **noverlap** (*int, optional*) – Number of points to overlap between segments. If *None*, `noverlap = np.rint(nperseg * overlap_ratio)`. Defaults to *None*.
- **overlap_ratio** (*float, optional*) – Used to calculate noverlap, if it is not specified (see above). Defaults to 0.5.
- **nfft** (*int, optional*) – Length of the FFT used, if a zero padded FFT is desired. If *None*, the FFT length is *nperseg*. Defaults to *None*.
- **detrend** (str or function or *False*, optional) – Specifies how to detrend each segment. If *detrend* is a string, it is passed as the *type* argument to the *detrend* function. If it is a function, it takes a segment and returns a detrended segment. If *detrend* is *False*, no detrending is done. Defaults to 'constant'.
- **return_onesided** (*bool, optional*) – If *True*, return a one-sided spectrum for real data. If *False* return a two-sided spectrum. Defaults to *True*, but for complex data, a two-sided spectrum is always returned.
- **dim** (*str, optional*) – Dimension along which the FFT is computed and sampling step calculated. If the signal is 1D, uses the only dimension, otherwise must be specified.

Returns **coh** – Coherence of 'darray' and 'other_darray'. It is complex and abs(coh)**2 is the squared magnitude coherohram.

Return type xarray.DataArray, complex

xrscipy.signal.xcorrelation

xrscipy.signal.**xcorrelation**(*darray*, *other_darray*, *normalize=True*, *fs=None*, *seglen=None*, *overlap_ratio=0.5*, *window='hann'*, *nperseg=256*, *noverlap=None*, *nfft=None*, *detrend='constant'*, *dim=None*)

Calculate the crosscorrelation.

Parameters

- **darray** (*xarray*) – Series of measurement values
- **other_darray** (*xarray*) – Series of measurement values fs : float, optional Sampling frequency of the *darray* and *other_darray* (time) series. If not specified, will be calculated it from the sampling step of the specified (or only) dimension.
- **window** (*str or tuple or array_like, optional*) – Desired window to use. If *window* is a string or tuple, it is passed to *get_window* to generate the window values, which are DFT-even by default. See *get_window* for a list of windows and required parameters. If *window* is array_like it will be used directly as the window and its length must be nperseg. Defaults to a Hann window.
- **seglen** (*float, optional*) – Segment length (i.e. nperseg) in units of the used (e.g. time) dimension.
- **nperseg** (*int, optional*) – Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is array_like, is set to the length of the window.
- **noverlap** (*int, optional*) – Number of points to overlap between segments. If *None*, `noverlap = np.rint(nperseg * overlap_ratio)`. Defaults to *None*.
- **overlap_ratio** (*float, optional*) – Used to calculate noverlap, if it is not specified (see above). Defaults to 0.5.
- **nfft** (*int, optional*) – Length of the FFT used, if a zero padded FFT is desired. If *None*, the FFT length is *nperseg*. Defaults to *None*.
- **detrend** (str or function or *False*, optional) – Specifies how to detrend each segment. If *detrend* is a string, it is passed as the *type* argument to the *detrend* function. If it is a function, it takes a segment and returns a detrended segment. If *detrend* is *False*, no detrending is done. Defaults to 'constant'.
- **return_onesided** (*bool, optional*) – If *True*, return a one-sided spectrum for real data. If *False* return a two-sided spectrum. Defaults to *True*, but for complex data, a two-sided spectrum is always returned.
- **dim** (*str, optional*) – Dimension along which the FFT is computed and sampling step calculated. If the signal is 1D, uses the only dimension, otherwise must be specified.

Returns **xcorr** – Crosscorrelation of 'darray' and 'other_darray' with the given dimension switched to the lag.

Return type xarray

xrscipy.signal.crossspectrogram

xrscipy.signal.**crossspectrogram**(*darray*, *other_darray*, *fs=None*, *seglen=None*, *overlap_ratio=0.5*, *window='hann'*, *nperseg=256*, *noverlap=None*, *nfft=None*, *detrend='constant'*, *return_onesided=True*, *dim=None*, *scaling='density'*, *mode='psd'*)

Calculate the cross spectrogram.

Parameters

- **darray** (*xarray*) – Series of measurement values
- **other_darray** (*xarray*) – Series of measurement values fs : float, optional Sampling frequency of the *darray* and *other_darray* (time) series. If not specified, will be calculated it from the sampling step of the specified (or only) dimension.
- **window** (*str or tuple or array_like, optional*) – Desired window to use. If *window* is a string or tuple, it is passed to *get_window* to generate the window values, which are DFT-even by default. See *get_window* for a list of windows and required parameters. If *window* is array_like it will be used directly as the window and its length must be nperseg. Defaults to a Hann window.
- **seglen** (*float, optional*) – Segment length (i.e. nperseg) in units of the used (e.g. time) dimension.
- **nperseg** (*int, optional*) – Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is array_like, is set to the length of the window.
- **noverlap** (*int, optional*) – Number of points to overlap between segments. If *None*, `noverlap = np.rint(nperseg * overlap_ratio)`. Defaults to *None*.
- **overlap_ratio** (*float, optional*) – Used to calculate noverlap, if it is not specified (see above). Defaults to 0.5.
- **nfft** (*int, optional*) – Length of the FFT used, if a zero padded FFT is desired. If *None*, the FFT length is *nperseg*. Defaults to *None*.
- **detrend** (str or function or *False*, optional) – Specifies how to detrend each segment. If *detrend* is a string, it is passed as the *type* argument to the *detrend* function. If it is a function, it takes a segment and returns a detrended segment. If *detrend* is *False*, no detrending is done. Defaults to 'constant'.
- **return_onesided** (*bool, optional*) – If *True*, return a one-sided spectrum for real data. If *False* return a two-sided spectrum. Defaults to *True*, but for complex data, a two-sided spectrum is always returned.
- **dim** (*str, optional*) – Dimension along which the FFT is computed and sampling step calculated. If the signal is 1D, uses the only dimension, otherwise must be specified.
- **scaling** (*{ 'density', 'spectrum' }, optional*) – Selects between computing the cross spectral density ('density') where *Pxy* has units of V**2/Hz and computing the cross spectrum ('spectrum') where *Pxy* has units of V**2, if *darray* and *other_darray* are measured in V and *fs* is measured in Hz. Defaults to 'density'.
- **mode** (*str*) – Defines what kind of return values are expected. Options are ['psd', 'complex', 'magnitude', 'angle', 'phase']. 'complex' is equivalent to the output of *stft* with no padding or boundary extension. 'magnitude' returns the absolute magnitude of the STFT. 'angle' and 'phase' return the complex angle of the STFT, with and without unwrapping, respectively.

Returns **Pxy** – Cross spectrogram of 'darray' and 'other_darray' with one new dimmension frequency and new coords for the specified dim.

Return type xarray.DataArray

Notes

By convention, Pxy is computed with the conjugate FFT of *darray* multiplied by the FFT of *other_darray*. If the input series differ in length, the shorter series will be zero-padded to match. An appropriate amount of overlap will depend on the choice of window and on your requirements. For the default Hann window an overlap of 50% is a reasonable trade off between accurately estimating the signal power, while not over counting any of the data. Narrower windows may require a larger overlap.

References

xrscipy.signal.spectrogram

xrscipy.signal.**spectrogram**(*darray*, *fs=None*, *seglen=None*, *overlap_ratio=0.5*, *window='hann'*, *nperseg=256*, *noverlap=None*, *nfft=None*, *detrend='constant'*, *return_onesided=True*, *dim=None*, *scaling='density'*, *mode='psd'*)

Calculate the spectrogram using crossspectrogram applied to the same data

Parameters

- **darray** (*xarray*) – Series of measurement values fs : float, optional Sampling frequency of the *darray* and *other_darray* (time) series. If not specified, will be calculated it from the sampling step of the specified (or only) dimension.
- **window** (*str or tuple or array_like, optional*) – Desired window to use. If *window* is a string or tuple, it is passed to *get_window* to generate the window values, which are DFT-even by default. See *get_window* for a list of windows and required parameters. If *window* is array_like it will be used directly as the window and its length must be nperseg. Defaults to a Hann window.
- **seglen** (*float, optional*) – Segment length (i.e. nperseg) in units of the used (e.g. time) dimension.
- **nperseg** (*int, optional*) – Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is array_like, is set to the length of the window.
- **noverlap** (*int, optional*) – Number of points to overlap between segments. If *None*, `noverlap = np.rint(nperseg * overlap_ratio)`. Defaults to *None*.
- **overlap_ratio** (*float, optional*) – Used to calculate noverlap, if it is not specified (see above). Defaults to 0.5.
- **nfft** (*int, optional*) – Length of the FFT used, if a zero padded FFT is desired. If *None*, the FFT length is *nperseg*. Defaults to *None*.
- **detrend** (str or function or *False*, optional) – Specifies how to detrend each segment. If *detrend* is a string, it is passed as the *type* argument to the *detrend* function. If it is a function, it takes a segment and returns a detrended segment. If *detrend* is *False*, no detrending is done. Defaults to 'constant'.
- **return_onesided** (*bool, optional*) – If *True*, return a one-sided spectrum for real data. If *False* return a two-sided spectrum. Defaults to *True*, but for complex data, a two-sided spectrum is always returned.

- **dim** (*str, optional*) – Dimension along which the FFT is computed and sampling step calculated. If the signal is 1D, uses the only dimension, otherwise must be specified.
- **scaling** (*{ 'density', 'spectrum' }, optional*) – Selects between computing the cross spectral density ('density') where *Pxy* has units of V**2/Hz and computing the cross spectrum ('spectrum') where *Pxy* has units of V**2, if *darray* and *other_darray* are measured in V and *fs* is measured in Hz. Defaults to 'density'.
- **mode** (*str*) – Defines what kind of return values are expected. Options are ['psd', 'complex', 'magnitude', 'angle', 'phase']. 'complex' is equivalent to the output of *stft* with no padding or boundary extension. 'magnitude' returns the absolute magnitude of the STFT. 'angle' and 'phase' return the complex angle of the STFT, with and without unwrapping, respectively.

Returns **Pxx** – Spectrogram of 'darray'.

Return type xarray.DataArray

xrscipy.signal.coherogram

xrscipy.signal.**coherogram**(*darray*, *other_darray*, *fs=None*, *seglen=None*, *overlap_ratio=0.5*, *nrolling=8*, *window='hann'*, *nperseg=256*, *noverlap=None*, *nfft=None*, *detrend='constant'*, *return_onesided=True*, *dim=None*)

Calculate the coherogram

The coherence (i.e. averaging of complex phasors) is done using a rolling average <...> of given size along the FFT windows and then coherogram = <crossspectrogram> / sqrt(<spectrogram1> * <spectrogram2>)

Parameters

- **darray** (*xarray*) – Series of measurement values
- **other_darray** (*xarray*) – Series of measurement values
- **nrolling** (*int, optional*) – Number of FFT windows used in the rolling average. fs : float, optional Sampling frequency of the *darray* and *other_darray* (time) series. If not specified, will be calculated it from the sampling step of the specified (or only) dimension.
- **window** (*str or tuple or array_like, optional*) – Desired window to use. If *window* is a string or tuple, it is passed to *get_window* to generate the window values, which are DFT-even by default. See *get_window* for a list of windows and required parameters. If *window* is array_like it will be used directly as the window and its length must be nperseg. Defaults to a Hann window.
- **seglen** (*float, optional*) – Segment length (i.e. nperseg) in units of the used (e.g. time) dimension.
- **nperseg** (*int, optional*) – Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is array_like, is set to the length of the window.
- **noverlap** (*int, optional*) – Number of points to overlap between segments. If *None*, `noverlap = np.rint(nperseg * overlap_ratio)`. Defaults to *None*.
- **overlap_ratio** (*float, optional*) – Used to calculate noverlap, if it is not specified (see above). Defaults to 0.5.
- **nfft** (*int, optional*) – Length of the FFT used, if a zero padded FFT is desired. If *None*, the FFT length is *nperseg*. Defaults to *None*.
- **detrend** (str or function or *False*, optional) – Specifies how to detrend each segment. If *detrend* is a string, it is passed as the *type* argument to the *detrend* function. If it is a function,

it takes a segment and returns a detrended segment. If *detrend* is *False*, no detrending is done. Defaults to 'constant'.

- **return_onesided** (*bool, optional*) – If *True*, return a one-sided spectrum for real data. If *False* return a two-sided spectrum. Defaults to *True*, but for complex data, a two-sided spectrum is always returned.
- **dim** (*str, optional*) – Dimension along which the FFT is computed and sampling step calculated. If the signal is 1D, uses the only dimension, otherwise must be specified.

Returns **coh** – Coherogram of 'darray' and 'other_darray'. It is complex and abs(coh)**2 is the squared magnitude coherohram.

Return type xarray.DataArray, complex

xrscipy.signal.hilbert

xrscipy.signal.**hilbert**(*darray*, *N=None*, *dim=None*)

Compute the analytic signal, using the Hilbert transform. The transformation is done along the selected dimension.

Parameters

- **darray** (*xarray*) – Signal data. Must be real.
- **N** (*int, optional*) – Number of Fourier components. Defaults to size along dim.
- **dim** (*string, optional*) – Axis along which to do the transformation. Uses the only dimension of darray is 1D.

Returns **darray** – Analytic signal of the Hilbert transform of 'darray' along selected axis.

Return type xarray

1.7.7 Digital filters

signal.frequency_filter(darray, f_crit[, ...])	Applies given frequency filter to a darray.
signal.lowpass(darray, f_cutoff, *args, **kwargs)	Applies lowpass filter to a darray.
signal.highpass(darray, f_cutoff, *args, ...)	Applies highpass filter to a darray.
signal.bandpass(darray, f_low, f_high, ...)	Applies bandpass filter to a darray.
signal.bandstop(darray, f_low, f_high, ...)	Applies bandstop filter to a darray.
signal.decimate(darray[, q, target_fs, dim])	Decimate signal by given (int) factor or to closest possible target_fs along the specified dimension.
signal.savgol_filter(darray, window_length, ...)	Apply a Savitzky-Golay filter to an array.

xrscipy.signal.frequency_filter

xrscipy.signal.**frequency_filter**(*darray*, *f_crit*, *order=None*, *irtype='iir'*, *filtfilt=True*, *apply_kwargs=None*, *in_nyq=False*, *dim=None*, ***kwargs*)

Applies given frequency filter to a darray.

This is a 1-d filter. If the darray is one dimensional, then the dimension along which the filter is applied is chosen automatically if not specified by *dim*. If *darray* is multi dimensional then axis along which the filter is applied has to be specified by *dim* string.

The type of the filter is chosen by *irtype* and then *filtfilt* states is the filter is applied both ways, forward and

backward. Additional parameters passed to filter function specified by *apply_kwargs*.

If 'iir' is chosen as *irtype*, then if *filtfilt* is True then the filter scipy.signal.filtfilt is used, if False scipy.signal.lfilter applies.

If 'fir' is chosen as *irtype*, then if *filtfilt* is True then the filter scipy.signal.sosfiltfilt is used, if False scipy.signal.sosfilt applies.

Parameters

- **darray** (*DataArray*) – An xarray type data to be filtered.
- **f_crit** (*array_like*) – A scalar or length-2 sequence giving the critical frequencies.
- **order** (*int, optional*) – The order of the filter. If Default then it takes order defaults from *_ORDER_DEFAULTS*, which is *irtype* specific. Default is None.
- **irtype** (*string, optional*) – A string specifying the impulse response of the filter, has to be either "fir" then finite impulse response (FIR) is used, or "iir" then infinite impulse response (IIR) filter is applied. ValueError is raised otherwise. Default is "iir".
- **filtfilt** (*bool, optional*) – When True the filter is applied both forwards and backwards, otherwise only one way, from left to right, is applied. Default is True.
- **apply_kwargs** (*dict, optional*) – Specifies kwargs, which are passed to the filter function given by *irtype* and *filtfilt*. Default is None.
- **in_nyq** (*bool, optional*) – If True, then the critical frequencies given by *f_crit* are normalized by Nyquist frequency. Default is False.
- **dim** (*string, optional*) – A string specifing the dimension along which the filter is applied. If *darray* is 1-d then the dimension is found if not specified by *dim*. For multi dimensional *darray* has to be specified, otherwise ValueError is raised. Default is None.
- **kwargs** – Arbitrary keyword arguments passed when the filter is being designed, either to scipy.signal.iirfilter if "iir" method for *irtype* is chosen, or scipy.signal.firwin.

xrscipy.signal.lowpass

xrscipy.signal.**lowpass**(*darray*, *f_cutoff*, **args*, ***kwargs*)

Applies lowpass filter to a darray.

This is a 1-d filter. If the darray is one dimensional, then the dimension along which the filter is applied is chosen automatically if not specified by an arg *dim*. If *darray* is multi dimensional then axis along which the filter is applied has to be specified by an additional argument *dim* string.

Parameters

- **darray** (*DataArray*) – An xarray type data to be filtered.
- **f_cutoff** (*array_like*) – A scalar specifying the cut-off frequency for the lowpass filter.
- **args** – Additional arguments passed to frequency_filter function to further specify the filter. The following parameters can be passed: (order, irtype, filtfilt, apply_kwargs, in_nyq, dim)
- **kwargs** – Arbitrary keyword arguments passed when the filter is being designed. See frequency_filter documentation for furhter information.

xrscipy.signal.highpass

xrscipy.signal.**highpass**(*darray*, *f_cutoff*, **args*, ***kwargs*)

Applies highpass filter to a darray.

This is a 1-d filter. If the darray is one dimensional, then the dimension along which the filter is applied is chosen automatically if not specified by an arg *dim*. If *darray* is multi dimensional then axis along which the filter is applied has to be specified by an additional argument *dim* string.

Parameters

- **darray** (*DataArray*) – An xarray type data to be filtered.
- **f_cutoff** (*array_like*) – A scalar specifying the cut-off frequency for the highpass filter.
- **args** – Additional arguments passed to frequency_filter function to further specify the filter. The following parameters can be passed: (order, irtype, filtfilt, apply_kwargs, in_nyq, dim)
- **kwargs** – Arbitrary keyword arguments passed when the filter is being designed. See frequency_filter documentation for furhter information.

xrscipy.signal.bandpass

xrscipy.signal.**bandpass**(*darray*, *f_low*, *f_high*, **args*, ***kwargs*)

Applies bandpass filter to a darray.

This is a 1-d filter. If the darray is one dimensional, then the dimension along which the filter is applied is chosen automatically if not specified by an arg *dim*. If *darray* is multi dimensional then axis along which the filter is applied has to be specified by an additional argument *dim* string.

Parameters

- **darray** (*DataArray*) – An xarray type data to be filtered.
- **f_low** (*array_like*) – A scalar specifying the lower cut-off frequency for the bandpass filter.
- **f_high** (*array_like*) – A scalar specifying the higher cut-off frequency for the bandpass filter.
- **args** – Additional arguments passed to frequency_filter function to further specify the filter. The following parameters can be passed: (order, irtype, filtfilt, apply_kwargs, in_nyq, dim)
- **kwargs** – Arbitrary keyword arguments passed when the filter is being designed. See frequency_filter documentation for furhter information.

xrscipy.signal.bandstop

xrscipy.signal.**bandstop**(*darray*, *f_low*, *f_high*, **args*, ***kwargs*)

Applies bandstop filter to a darray.

This is a 1-d filter. If the darray is one dimensional, then the dimension along which the filter is applied is chosen automatically if not specified by an arg *dim*. If *darray* is multi dimensional then axis along which the filter is applied has to be specified by an additional argument *dim* string.

Parameters

- **darray** (*DataArray*) – An xarray type data to be filtered.

- **f_low** (*array_like*) – A scalar specifying the lower cut-off frequency for the bandstop filter.
- **f_high** (*array_like*) – A scalar specifying the higher cut-off frequency for the bandstop filter.
- **args** – Additional arguments passed to frequency_filter function to further specify the filter. The following parameters can be passed: (order, irtype, filtfilt, apply_kwargs, in_nyq, dim)
- **kwargs** – Arbitrary keyword arguments passed when the filter is being designed. See frequency_filter documentation for furhter information.

xrscipy.signal.decimate

xrscipy.signal.**decimate**(*darray*, *q=None*, *target_fs=None*, *dim=None*, ***lowpass_kwargs*)

Decimate signal by given (int) factor or to closest possible target_fs along the specified dimension.

Decimation: lowpass to new nyquist frequency and then downsample by factor *q lowpass_kwargs* are given to the lowpass method.

If *q* is not given, it is approximated as the closest integer ratio of fs / *target_fs*, so *target_fs* must be smaller than current sampling frequency fs.

If $q < 2$, decimation is skipped and a DecimationWarning is emitted

Parameters

- **darray** (*DataArray*) – An xarray type data to be decimated.
- **q** (*array_like*) – A scalar specifying the factor by which the signal should be decimated. If not given, it is approximated as the closest integer ratio of fs / *target_fs*. If set lower than 2, decimation is skipped and a DecimationWarning is emitted. Default is None.
- **target_fs** (*array_like, optional*) – A scalar specifying target sampling frequency of returning data. Must be smaller than current sampling frequency. Default is None.
- **dim** (*string, optional*) – A string specifing the dimension along which the filter is applied. If *darray* is 1-d then the dimension is found if not specified by *dim*. For multi dimensional *darray* has to be specified, otherwise ValueError is raised. Default is None.
- **lowpass_kwargs** – Arbitrary keyword arguments passed to the lowpass method. See lowpass method for further details.

xrscipy.signal.savgol_filter

xrscipy.signal.**savgol_filter**(*darray*, *window_length*, *polyorder*, *deriv=0*, *delta=None*, *dim=None*, *mode='interp'*, *cval=0.0*)

Apply a Savitzky-Golay filter to an array.

This is a 1-d filter. If *darray* has dimension greater than 1, *dim* determines the dimension along which the filter is applied. :param darray: An xarray type data to be filtered. If values of *darray* are not

a single or double precision floating point array, it will be converted to type `numpy.float64` before filtering.

Parameters

- **window_length** (*int*) – The length of the filter window (i.e. the number of coefficients). *window_length* must be a positive odd integer. If *mode* is 'interp', *window_length* must be less than or equal to the size of *darray*.
- **polyorder** (*int*) – The order of the polynomial used to fit the samples. *polyorder* must be less than *window_length*.
- **deriv** (*int, optional*) – The order of the derivative to compute. This must be a nonnegative integer. The default is 0, which means to filter the data without differentiating.
- **delta** (*float, optional*) – The spacing of the samples to which the filter will be applied. This is only used if deriv > 0. Default is 1.0.
- **dim** (*string, optional*) – Specifies the dimension along which the filter is applied. For 1-d darray finds the only dimension, if not specified. For multi dimensional darray, the dimension for the filtering has to be specified, otherwise raises ValueError. Default is None.
- **mode** (*str, optional*) – Must be 'mirror', 'constant', 'nearest', 'wrap' or 'interp'. This determines the type of extension to use for the padded signal to which the filter is applied. When *mode* is 'constant', the padding value is given by *cval*. See the Notes for more details on 'mirror', 'constant', 'wrap', and 'nearest'. When the 'interp' mode is selected (the default), no extension is used. Instead, a degree *polyorder* polynomial is fit to the last *window_length* values of the edges, and this polynomial is used to evaluate the last *window_length // 2* output values.
- **cval** (*scalar, optional*) – Value to fill past the edges of the input if *mode* is 'constant'. Default is 0.0.

Returns **y** – The filtered data.

Return type DataArray, same shape as *darray*

CHAPTER

TWO

LICENSE

xr-scipy is available under the open source Apache License.

BIBLIOGRAPHY

[CT] See, for example, P. Alfeld, ''A trivariate Clough-Tocher scheme for tetrahedral data". Computer Aided Geometric Design, 1, 169 (1984); G. Farin, ''Triangular Bernstein-Bezier patches". Computer Aided Geometric Design, 3, 83 (1986).

[Nielson83] G. Nielson, ''A method for interpolating scattered data based upon a minimum norm network". Math. Comp., 40, 253 (1983).

[Renka84] R. J. Renka and A. K. Cline. ''A Triangle-based C1 interpolation method.", Rocky Mountain J. Math., 14, 223 (1984).

[CT] Cooley, James W., and John W. Tukey, 1965, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.* 19: 297-301.

INDEX

Symbols

A

B

C

D

F

G

H

I

L

N

P

R

S

T

X