
multitaper

Release 1.1.3

German A. Prieto

Mar 02, 2022

CONTENTS:

1	Installation	3
2	mtspec module	5
3	mtcross module	13
4	utils module	19
5	Indices and tables	33
	Python Module Index	35
	Index	37

multitaper v.1.1.3

Germán A. Prieto Departamento de Geociencias, Universidad Nacional de Colombia

A collection of modules for spectral analysis using the multitaper algorithm. The modules not only includes power spectral density (PSD) estimation with confidence intervals, but also multivariate problems including coherence, dual-frequency, correlations, and deconvolution estimation. Implementations of the sine and quadratic multitaper methods are also available.

Github

<https://github.com/gaprieto/multitaper>

INSTALLATION

The multitaper package is composed of a number of Python modules. As of January 2022, multitaper can be installed using conda. *pip* installation is also available. You can also simply download the folder and install and add to your Python path.

Dependencies You will need Python 3.8+. The following packages are automatically installed:

numpy

scipy

Optional dependencies for plotting and example Notebooks:

jupyter

matplotlib

I recommend creating a virtual environment before installing:

```
>> conda create --name mtspec python=3.8
>> conda activate mtspec
```

Install with Conda:

```
>> conda install -c gprieto multitaper
```

Install with pip:

```
>> pip install multitaper
```

Local install: Download a copy of the codes from github

```
git clone https://github.com/gprieto/multitaper.git
```

or simply download ZIP file from <https://github.com/gprieto/multitaper> and navigate to the directory and type

```
>> pip install .
```


MTSPEC MODULE

Module with routines for univariate multitaper spectrum estimation (1D). Contains the main MTSpec and MTSine classes where the estimates are made and stored.

See module mtcross for bi-variate problems

Classes

- MTSine - A class to represent Sine Multitaper estimates
- MTSpec - A class to represent Thomson's multitaper estimates

Functions

- spectrogram - Computes a spectrogram with consecutive multitaper estimates.

class mtspec.**MTSine**(*x*, *ntap*=0, *ntimes*=0, *fact*=1.0, *dt*=1.0)

Bases: object

A class for sine multitaper estimates

Attributes

Parameters

npts [int] number of points of time series

nfft [int] number of points of FFT. $nfft = 2 * npts$

Time series

x [ndarray [npts]] time series

xvar [float] variance of time series

dt [float] sampling interval

Frequency vector

nf [int] number of unique frequency points of spectral estimate, assuming real time series

freq [ndarray [nfft]] frequency vector in Hz

df [float] frequency sampling interval

Method

ntap [int] fixed number of tapers if $ntap < 0$, use $kopt$

kopt [ndarray [nfft,1]] number of tapers at each frequency

ntimes [int] number of max iterations to perform

ireal [int] 0 - real time series 1 - complex time series

Spectral estimates

spec [ndarray [nfft,1]] multitaper estimate

err [ndarray [nfft,2]] 1-std confidence interval of spectral estimate simple dof estimate

Notes

The class is in charge of estimating the adaptive sine multitaper as in Riedel and Sidorenko (1995). This is done by performing a MSE adaptive estimation. First a pilot spectral estimate is used, and S'' is estimated, in order to get te number of tapers to use, using (13) of R & S for a min square error spectrum.

Unlike the prolate spheroidal multitapers, the sine multitaper adaptive process introduces a variable resolution and error in the frequency domain. Complete error information is contained in the output variables file as the corridor of 1-standard-deviation errors, and in K, the number of tapers used at each frequency. The errors are estimated in the simplest way, from the number of degrees of freedom (two per taper), not by jack-knifing. The frequency resolution is found from $K \cdot f_N / N_f$ where f_N is the Nyquist frequency and N_f is the number of frequencies estimated. The adaptive process used is as follows. A quadratic fit to the log PSD within an adaptively determined frequency band is used to find an estimate of the local second derivative of the spectrum. This is used in an equation like R & S (13) for the MSE taper number, with the difference that a parabolic weighting is applied with increasing taper order. Because the FFTs of the tapered series can be found by resampling the FFT of the original time series (doubled in length and padded with zeros) only one FFT is required per series, no matter how many tapers are used. This makes the program fast. Compared with the Thomson multitaper programs, this code is not only fast but simple and short. The spectra associated with the sine tapers are weighted before averaging with a parabolically varying weight. The expression for the optimal number of tapers given by R & S must be modified since it gives an unbounded result near points where S'' vanishes, which happens at many points in most spectra. This program restricts the rate of growth of the number of tapers so that a neighboring covering interval estimate is never completely contained in the next such interval.

This method SHOULD not be used for sharp cutoffs or deep valleys, or small sample sizes. Instead use Thomson multitaper in mtspec in this same library.

References

Riedel and Sidorenko, IEEE Tr. Sig. Pr, 43, 188, 1995

Based on Bob Parker psd.f codes. Most of the comments come from his documentation as well.

Modified

September 22 2005

Calls

utils.quick, utils.adapt

```
class mtspec.MTSpec(x, nw=4, kspec=0, dt=1.0, nfft=0, iadapt=0, vn=None, lamb=None)
```

Bases: object

A class for univariate Thomson multitaper estimates

Attributes

Parameters

npts [int] number of points of time series
nfft [int] number of points of FFT. Default adds padding.
nw [float] time-bandwidth product
kspec [int] number of tapers to use

Time series

x [ndarray [npts]] time series
xvar [float] variance of time series
dt [float] sampling interval

Frequency vector

nf [int] number of unique frequency points of spectral estimate, assuming real time series
freq [ndarray [nfft]] frequency vector in Hz
df [float] frequency sampling interval

Method

iadapt [int] defines method to use 0 - adaptive multitaper 1 - unweighted, wt = 1 for all tapers 2 - wt by the eigenvalue of DPSS
ireal [int] 0 - real time series 1 - complex time series

DPSS tapers and eigenvalues

vn [ndarray [npts,kspec]] Slepian sequences
lamb [ndarray [kspec]] Eigenvalues of Slepian sequences

Spectral estimates

yk [complex ndarray [nfft,kspec]] eigencoefficients, fft of tapered series
sk [ndarray [nfft,kspec]] eigenspectra, power spectra for each yk
spec [ndarray [nfft,1]] multitaper estimate
se [ndarray [nfft,1]] degrees of freedom of estimate
wt [ndarray [nfft,kspec]] weights for each eigencoefficient at each frequency

Methods

- **rspec** : returns the positive frequency of the spectra only
- **reshape** : reshape yk's based on F-test of line components
- **jackspec** : estimate 95% confidence interval of multitaper estimate
- **qiinv** : the quadratic inverse spectral estimate
- **ftest** : F-test of line components in the spectra
- **df_spec** : dual-frequency autospectra

References

Based on David J. Thomson's codes, Alan Chave and Thomson's Codes and partial codes from EIS-PACK, Robert L. Parker and Glenn Ierley from Scripps Institution of Oceanography. And my own Fortran90 library.

Notes

The class is in charge of estimating the adaptive weighed multitaper spectrum, as in Thomson 1982. This is done by estimating the dpss (discrete prolate spheroidal sequences), multiplying each of the kspec tapers with the data series, take the fft, and using the adaptive scheme for a better estimation.

As a by product of the spectrum (spec), all intermediate steps are retained, which can be used for bi-variate analysis, deconvolutuion, returning to the time domain, etc. By-products include the complex information in yk, the eigenspectra sk, the jackknife 95% confidence intervals (spec_ci), the degrees of freedom (se) and the weigths wt(nf,kspec) used. See below for a complete list.

Modified

January 2022 (German Prieto)

df_spec()

Performs the dual-frequency spectrum of a signal with itself.

Returns

df_spec [ndarray complex, 2D (nf,nf)] the complex dual-frequency cross-spectrum. Not normalized

df_cohe [ndarray, 2D (nf,nf)] MSC, dual-freq coherence matrix. Normalized (0.0,1.0)

df_phase [ndarray, 2D (nf,nf)] the dual-frequency phase

Calls

utils.df_spec

ftest()

Performs the F test for a line component

Computes F-test for single spectral line components at the frequency bins given in the MTSpec class.

Returns

F : ndarray [nfft] vector of f test values, real p : ndarray [nfft] vector with probability of line component

Calls

utils.f_test

jackspec()

code to calculate adaptively weighted jackknifed 95% confidence limits

Returns

spec_ci [ndarray [nfft,2]] real array of jackknife error estimates, with 5 and 95% confidence intervals of the spectrum.

Calls

utils.jackspec

qiinv()

Function to calculate the Quadratic Spectrum using the method developed by Prieto et al. (2007).

The first 2 derivatives of the spectrum are estimated and the bias associated with curvature (2nd derivative) is reduced.

Calculate the Stationary Inverse Theory Spectrum. Basically, compute the spectrum inside the innerband.

This approach is very similar to D.J. Thomson (1990).

Returns

qispec [ndarray [nfft,0]] the QI spectrum estimate

ds [ndarray [nfft,0]] the estimate of the first derivative

dds [ndarray [nfft,0]] the estimate of the second derivative

References

G. A. Prieto, R. L. Parker, D. J. Thomson, F. L. Vernon, and R. L. Graham (2007), Reducing the bias of multitaper spectrum estimates, *Geophys. J. Int.*, 171, 1269-1281. doi: 10.1111/j.1365-246X.2007.03592.x.

Calls

utils.qiinv

reshape(*fcrit=0.95, p=None*)

Reshape eigenft's (yk) around significant spectral lines The "significant" means above fcritical probability (0.95) If probability is large at neighbouring frequencies, I will only remove the largest probability energy.

Returns recalculated yk, sk, spec, wt, and se

Parameters

fcrit [float optional] Probability value over which to reshape, default = 0.95

p [ndarray optional [nfft]] F-test probabilities to find fcritical If None, it will be calculated

Returns

respec [ndarray [nfft]] The reshaped PSD estimate

spec [ndarray [nfft]] the PSD without the line components

yk [ndarray [nfft,kspec]] the eigenft's without line components

sline [ndarray [nfft]] the PSD of the line components only

Calls

utils.yk_reshape

rspec(*args)

Returns the spectra at positive frequencies, checking that a real input signal was used.

Parameters

args [ndarray] another array to return the positive frequencies. Could be qispec, spec_ci, etc.

mtspec.spectrogram(data, dt, twin, olap=0.5, nw=3.5, kspec=5, fmin=0.0, fmax=- 1.0, iadapt=0)

Computes a spectrogram with consecutive multitaper estimates. Returns both Thomson's multitaper and the Quadratic multitaper estimate

Parameters

data [array_like (npts,)] Time series or sequence

dt [float] Sampling interval in seconds of the time series.

twin [float] Time duration in seconds of each segment for a single multitaper estimate.

olap [float, optional] Overlap requested for the segment in building the spectrogram. Defaults = 0.5, values must be (0.0 - 0.99). Overlap rounds to the nearest integer point.

nw [float, optional] Time-bandwidth product for Thomson's multitaper algorithm. Default = 3.5

kspec [int, optional] Number of tapers for averaging the multitaper estimate. Default = 5

fmin [float, optional] Minimum frequency to estimate the spectrogram, otherwise returns the entire spectrogram matrix. Default = 0.0 Hz

fmax [float, optional] Maximum frequency to estimate the spectrogram, otherwise returns the entire spectrogram matrix. Default = 0.5/dt Hz (Nyquist frequency)

iadapt [integer, optional] User defined, determines which method for multitaper averaging to use. Default = 0
0 - Adaptive multitaper 1 - Eigenvalue weights 2 - Constant weighting

Returns

f [ndarray] Array of sample frequencies.

t [ndarray] Array of segment times.

Quad [ndarray] Spectrogram of x using the quadratic multitaper estimate.

MT [ndarray] Spectrogram of x using Thomson's multitaper estimate.

By default, the last axis of Quad/MT corresponds to the segment times.

See Also

MTSpec: Multitaper estimate of a time series.

Notes

The code assumes a real input signals and thus mainly returns the positive frequencies. For a complex input signals, code could require adaptation.

References

Prieto, G.A. (2022). The multitaper spectrum analysis package in Python. Seism. Res. Lett In review.

Examples

To do

MTCROSS MODULE

Module that contains all multivariate-multitaper codes.

Contains:

- `mt_cohe`
- `mt_deconv`
- To do: `wv_spec`

Module with routines for bi-variate multitaper spectrum estimation. Contains the main `MTCross` and `SineCross` classes where the estimates are made and stored.

It takes univariate classes `MTSpec` and `MTSine` for estimating coherence tranfer functions, etc.

See module `mtspec` for univariate problems

Classes

- `MTCross` - A class to represent Thomson's multitaper cross-spectra
- `SineCross` - A class to represent Sine Multitaper cross-spectra

Functions None

class `mtcross.MTCross`(*x*, *y*, *nw*=4, *kspec*=0, *dt*=1.0, *nfft*=0, *iadapt*=0, *wl*=0.0)

Bases: `object`

A class for bi-variate Thomson multitaper estimates. It performs main steps in bi-variate multitaper estimation, including cross-spectrum, coherency and transfer function.

Attributes

Parameters

npts [int] number of points of time series

nfft [int] number of points of FFT. Dafault adds padding.

nw [flaot] time-bandwidth product

kspec [int] number of tapers to use

Time series

x [ndarray [npts]] time series

xvar [float] variance of time series

dt [float] sampling interval

Frequency vector

nf [int] number of unique frequency points of spectral estimate, assuming real time series

freq [ndarray [nfft]] frequency vector in Hz

df [float] frequency sampling interval

Method

iadapt [int] defines method to use 0 - adaptive multitaper 1 - unweighted, wt =1 for all tapers 2 - wt by the eigenvalue of DPSS

wl [float, optional] water-level for stabilizing deconvolution (transfer function). defined as proportion of mean power of *Syy*

Spectral estimates

Sxx [ndarray [nfft]] Power spectrum of *x* time series

Syy [ndarray [nfft]] Power spectrum of *y* time series

Sxy [ndarray, complex [nfft]] Cross-spectrum of *x*, *y* series

cohe [ndarray [nfft]] MSC, freq coherence. Normalized (0.0,1.0)

phase [ndarray [nfft]] the phase of the cross-spectrum

cohy [ndarray, complex [nfft]] the complex coherency, normalized cross-spectrum

trf [ndarray, complex [nfft]] the transfer function $S_{xy}/(S_{yy_wl})$, with water-level optional

se [ndarray [nfft,1]] degrees of freedom of estimate

wt [ndarray [nfft,kspec]] weights for each eigencoefficient at each frequency

Methods

- **init** : Constructor of the *MTCross* class
- **mt_deconv** : Perform the deconvolution from the *self.trf*, by iFFT
- **mt_corr** [compute time-domain via iFFT of cross-spectrum,] coherency, and transfer function

Modified

German Prieto January 2022

mt_corr()

Compute time-domain via iFFT of cross-spectrum, coherency, and transfer function

Cross spectrum, coherency and transfer function already pre-computed in *MTCross*.

Returns

xcorr [ndarray [nfft]] time domain of the transfer function.

dcohy [ndarray [nfft]] time domain of the transfer function.

dfun [ndarray [nfft]] time domain of the transfer function.

Delay time $t=0$ is centered in the middle.

Notes

The three correlation-based estimates in the time domain

- correlation (cross-spectrum)
- deconvolution (transfer function)
- norm correlation (coherency)

Correlation:

- $S_{xy} = S_x * \text{conj}(S_y)$

Deconvolution:

- $S_{xy}/S_y = S_x * \text{conj}(S_y)/S_y^2$

Coherency

- $S_{xy}/\sqrt{S_x * S_y}$

mt_deconv()

Generate a deconvolution between two time series, returning the time-domain signal.

MTCross has already pre-computed the cross-spectrum and the transfer function.

Returns

dfun [ndarray [nfft]] time domain of the transfer function. delay time $t=0$ is centered in the middle.

References

The code more or less follows the paper Receiver Functions from multiple-taper spectral correlation estimates. J. Park and V. Levin., BSSA 90#6 1507-1520

It also uses the code based on dual frequency I created in GA Prieto, Vernon, FL, Masters, G, and Thomson, DJ (2005), Multitaper Wigner-Ville Spectrum for Detecting Dispersive Signals from Earthquake Records, Proceedings of the Thirty-Ninth Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA., pp 938-941.

class mtcross.SineCross(*x*, *y*, *ntap*=0, *ntimes*=0, *fact*=1.0, *dt*=1.0, *p*=0.95)

Bases: object

A class for bi-variate Sine multitaper estimates. Performs the coherence and cross-spectrum estimation using the sine multitaper method.

Attributes

Parameters

- npts** [int] number of points of time series
- nfft** [int] number of points of FFT. $nfft = 2 * npts$

Time series

x [ndarray [npts]] time series x
xvar [float] variance of x time series
y [ndarray [npts]] time series y
yvar [float] variance of y time series
dt [float] sampling interval

Frequency vector

nf [int] number of unique frequency points of spectral estimate, assuming real time series
freq [ndarray [nfft]] frequency vector in Hz
df [float] frequency sampling interval

Method

ntap [int] fixed number of tapers if ntap<0, use kopt
kopt [ndarray [nfft,1]] number of tapers at each frequency
ntimes [int] number of max iterations to perform
ireal [int] 0 - real time series 1 - complex time series

Spectral estimates

cspec [ndarray, complex [nfft]] Coss-spectrum of x, y series
sxy [ndarray, complex [nfft]] Coss-spectrum of x, y series
cohe [ndarray [nfft]] MSC, freq coherence. Normalized (0.0,1.0)
phase [ndarray [nfft]] the phase of the cross-spectrum
gain [ndarray [nfft]] the gain for the two spectra
cohy [ndarray, complex [nfft]] the complex coherency, normalized cross-spectrum
trf [ndarray, complex [nfft]] the transfer function $S_{xy}/(S_{yy_wl})$, with water-level optional
se [ndarray [nfft,1]] degrees of freedom of estimate
conf [ndarray [nfft,]] confidence in cross-spectrum at each frequency

Methods

- **init** : Constructor of the SineCross class
- **mt_deconv** : Perform the deconvolution from the self.trf, by iFFT
- **mt_corr** [compute time-domain via iFFT of cross-spectrum,] coherency, and transfer function

References

Riedel and Sidorenko, IEEE Tr. Sig. Pr, 43, 188, 1995

Based on Bob Parker psd.f and cross.f codes. Most of the comments come from his documentation as well.

Modified

January 2022, German A. Prieto

mt_corr()

Compute time-domain via iFFT of cross-spectrum, coherency, and transfer function

Cross spectrum, coherency and transfer function already pre-computed in SineCross class.

Returns

xcorr [ndarray [nfft]] time domain of the transfer function.

dcohy [ndarray [nfft]] time domain of the transfer function.

dfun [ndarray [nfft]] time domain of the transfer function.

Delay time $t=0$ in centered in the middle.

Notes**The three correlation-based estimates in the time domain**

- correlation (cross-spectrum)
- deconvolution (transfer function)
- norm correlation (coherency)

Correlation:

- $S_{xy} = S_x \cdot \text{conj}(S_y)$

Deconvolution:

- $S_{xy}/S_y = S_x \cdot \text{conj}(S_y)/S_y^2$

Coherency

- $S_{xy}/\sqrt{S_x \cdot S_y}$

mt_deconv()

Generate a deconvolution between two time series, returning the time-domain signal.

SineCross has already pre-computed the cross-spectrum and the transfer function.

Returns

dfun [ndarray [nfft]] time domain of the transfer function. delay time $t=0$ in centered in the middle.

UTILS MODULE

Module with all the definitions (routines) of general use of the multitaper routines.

Contains:

- `set_xint` - setup Ierly's quadrature
- `xint` - Quadrature by Ierley's method of Chebychev sampling.
- `dpss_ev` - Recalculate the DPSS eigenvalues using Quadrature
- `dpss` - calculate the DPSS for given NW, NPTS
- `eigenspec` - calculate eigenspectra using DPSS sequences.
- `adaptspec` - calculate adaptively weighted power spectrum
- `jackspec` - calculate adaptively weighted jackknifed 95% confidence limits
- `qiinv` - calculate the Stationary Inverse Theory Spectrum.
- `ftest` - performs the F-test for a line component
- `yk_reshape` - reshape eigenft's around significant spectral lines
- `wt2dof` - calculate the d.o.f. of the multitaper
- `df_spec` - Dual frequency spectrum, using two MTSPEC classes to compute.
- `sft` - the slow Fourier transform
- `squick` - for sine multitaper, constructs average multitaper
- `squick2` - for sine multitaper, constructs average multitaper, 2 signals
- `sadapt` - for sine multitaper, adaptive estimation of # of tapers
- `sadapt2` - for sine multitaper, same but for 2 signals
- `north` - for sine multitaper, derivatives of spectrum
- `curb` - for sine multitaper, clips # of tapers
- `get_data` - download data and load into numpy array

`utils.adaptspec(yk, sk, lamb, iadapt=0)`

Calculate adaptively weighted power spectrum Options for non-adaptive estimates are possible, with optional parameter `iadapt`, using average of `sk`'s or weighted by eigenvalue.

Parameters

yk [complex ndarray [nfft,kspec]] complex array of kspec eigencefficients

sk [ndarray [nfft,kspec]] array containing kspe power spectra

lamb [ndarray [kspec]] eigenvalues of tapers

iadapt [int] defines methos to use, default = 0 0 - adaptive multitaper 1 - unweighted, wt =1 for all tapers 2 - wt by the eigenvalue of DPSS

Returns

spec [ndarray [nfft]] real vector containing adaptively weighted spectrum

se [ndarray [nfft]] real vector containing the number of degrees of freedom for the spectral estimate at each frequency.

wt [ndarray [nfft,kspec]] real array containing the ne weights for kspec eigenspectra normalized so that if there is no bias, the weights are unity.

Modified

German Prieto, Aug 2006

Corrected the estimation of the dofs se (sum of squares of wt is 1.0) maximum wt = 1

German Prieto, October 2007 Added the an additional subroutine noadaptspec to calculate a simple non-adaptive multitaper spectrum. This can be used in transfer functions and deconvolution, where adaptive methods might not be necessary.

February 2022. Now calculating adapt weights without for loop.

Calls

nothing

`utils.copy_examples(path='./multitaper-examples')`

Copy the examples folder, so the user can have access to the Notebooks and .py files

Use `multitaper.utils.copy_examples()` function to copy all Notebooks and .py example files to local directory

Install the examples for multitaper in the given location.

WARNING: If the path exists, files will be overwritten. Default path is `./multitaper-examples/` to avoid potential overwrite of common folder names. Dependencies for the notebooks include - `matplotlib` - `scipy` - `numpy` These need to be available in the enviroment used.

References

Codes based on an example from Ben Mather, Robert Delhay, within the PyCurious package.

`utils.curb(n, v_in)`

Takes n-long vector `v[n]` and rewrites it so that all points lie below the piece-wise linear function $v(k) + \text{abs}(j-k)$, where `v(k)` is a local minimum in the original `v`.

Effectively clips strong peaks and keeps slopes under 1 in magnitude.

Parameters

v_in [ndarray [n]] vector to be clipped, n-long

Returns

v [ndarray [n]] clipped vector

`utils.df_spec(x, y=None, fmin=None, fmax=None)`

Dual frequency spectrum using one/two MTSPEC classes. For now, only positive frequencies are studied

Construct the dual-frequency spectrum from the yk's and the weights of the usual multitaper spectrum estimation.

Parameters

x [MTSpec class] variable with the multitaper information (yk's)

y [MTSpec class, optional] similar to x for a second time series if y is None, auto-dual frequency is calculated.

fmin [float, optional] minimum frequency to calculate the DF spectrum

fmax [float, optional] minimum frequency to calculate the DF spectrum

Returns

df_spec [ndarray complex, 2D (nf,nf)] the complex dual-frequency cross-spectrum. Not normalized

df_cohe [ndarray, 2D (nf,nf)] MSC, dual-freq coherence matrix. Normalized (0.0,1.0)

df_phase [ndarray, 2D (nf,nf)] the dual-frequency phase

Notes

both x and y need the same parameters (npts, kspec, etc.)

Modified

German Prieto, September 2005

German A. Prieto, September 2007

Slight rewrite to adjust to newer mtspec codes.

German Prieto, February 2022

Speed up by simplifying for loops and using np.tensordot

Calls

np.tensordot

`utils.df_spec_old(x, y=None, fmin=None, fmax=None)`

Dual frequency spectrum using one/two MTSPEC classes. For now, only positive frequencies are studied

Construct the dual-frequency spectrum from the yk's and the weights of the usual multitaper spectrum estimation.

Parameters

x [MTSpec class] variable with the multitaper information (yk's)

y [MTSpec class, optional] similar to x for a second time series if y is None, auto-dual frequency is calculated.

fmin [float, optional] minimum frequency to calculate the DF spectrum

fmax [float, optional] minimum frequency to calculate the DF spectrum

Returns

df_spec [ndarray complex, 2D (nf,nf)] the complex dual-frequency cross-spectrum. Not normalized

df_cohe [ndarray, 2D (nf,nf)] MSC, dual-freq coherence matrix. Normalized (0.0,1.0)

df_phase [ndarray, 2D (nf,nf)] the dual-frequency phase

Notes

both x and y need the same parameters (npts, kspec, etc.)

Modified

German Prieto, September 2005

German A. Prieto, September 2007

Slight rewrite to adjust to newer mtspec codes.

Calls

Nothing

`utils.dpss(npts, nw, kspec=None)`

Calculation of the Discrete Prolate Spheroidal Sequences, and the correspondent eigenvalues.

- Slepian, D. 1978 Bell Sys Tech J v57 n5 1371-1430
- Thomson, D. J. 1982 Proc IEEE v70 n9 1055-1096

Parameters

npts [int] the number of points in the series

nw [float] the time-bandwidth product (number of Rayleigh bins)

kspec [int] Optional, the desired number of tapers default = $2 \cdot nw - 1$

Returns

v [ndarray (npts,kspec)] the eigenvectors (tapers) are returned in v[npts,nev]

lamb [ndarray (kspec)] the eigenvalues of the v's

Notes

In SCIPY the codes are already available to calculate the DPSS. Eigenvalues are calculated using Chebeshev Quadrature. Code also performs interpolation if $NPTS > 1e5$

Also, define DPSS to be positive-standard, meaning vn's always start positive, whether symmetric or not.

Modified

December 2020

February 2022 - Changed a for loop for a direct np.sum().

Calls

scipy.signal.windows.dpss dpss_ev

utils.dpss2(*npts, nw, nev=None*)

This is a try to compute the DPSS using the original Thomson approach. It reduces the problem to half the size and inverts independently for the even and odd functions.

This is work in progress and not used.

Modified from F90 library: German Prieto December 2020

The tapers are the eigenvectors of the tridiagonal matrix $\sigma(i,j)$ [see Slepian(1978) eq 14 and 25.] They are also the eigenvectors of the Toeplitz matrix eq. 18. We solve the tridiagonal system in

scipy.linalg.eigh_tridiagonal

(real symmetric tridiagonal solver) for the tapers and use them in the integral equation in the frequency domain (dpss_ev subroutine) to get the eigenvalues more accurately, by performing Chebychev Gaussian Quadrature following Thomson's codes.

First, we create the main and off-diagonal vectors of the tridiagonal matrix. We compute separately the even and odd tapers, by calling eigh_tridiagonal from SCIPY.

We, refine the eigenvalues, by computing the inner bandwidth energy in the frequency domain (eq. 2.6 Thomson). Also the "leakage" (1 - eigenvalue) is estimated, independently if necessary.

In SCIPY the code are already available to calculate the DPSS. Eigenvalues are calculated using Chebeshev Quadrature.

Code also performs interpolation if $NPTS > 1e5$ Also, define DPSS to be positive-standard, meaning v_n 's always start positive, whether symmetric or not.

Calls

To do

utils.dpss_ev(*vn, w, atol=1e-14*)

Recalculate the DPSS eigenvalues, performing the integration in the -W:W range, using Quadrature.

computes eigenvalues for the discrete prolate spheroidal sequences in efn by integration of the corresponding squared discrete prolate spheroidal wavefunctions over the inner domain. Due to symmetry, we perform integration from zero to w.

We use Chebychev quadrature for the numerical integration.

Parameters

vn [ndarray [npts,kspec]] DPSS to calculate eigenvalues

w [float] the bandwidth (= time-bandwidth product/ndata)

atol [float, optional] absolute error tolerance for the integration. this should be set to 10^{-n} , where n is the number of significant figures that can be represented on the machine. default = $1e-14$

Returns

lamb [ndarray [kspec]] vector of length `vn.shape[1]`, contains the eigenvalues

Modified

November 2004 (German A. Prieto)

Calls

xint

`utils.eigenspec(x, vn, lamb, nfft)`

Calculate eigenspectra using DPSS sequences. Gets y_k 's from Thomson (1982).

Parameters

x [ndarray [npts,0]] real vector with the time series

vn [ndarray [npts,kspec]] the different tapers computed in dpss

lambda [ndarray [kspec]] the eigenvalues of the tapers vn

nfft [int] number of frequency points (inc. positive and negative frequencies)

Returns

yk [complex ndarray [kspec,nfft]] complex array with kspec fft's of tapered data. Regardless of real/complex input data all frequencies are stored. Good for coherence, deconvolution, etc.

sk [ndarray [kspec,nfft]] real array with kspec eigenspectra

Modified

February 2022. Changed a for loop for xtap

German Prieto, November 2004

Notes

Computes eigen-fft's by windowing real data with dpss and taking ffts Note that fft is unnormalized and window is such that its sum of squares is one, so that $psd=yk^{**2}$.

The fft's are computed using SCIPY FFT codes, and parallel FFT can potentially speed up the calculation. Up to KSPEC works are sent. The y_k 's are saved to get phase information. Note that tapers are applied to the original data (npts long) and the FFT is zero padded up to NFFT points.

Calls

scipy.fft.fft

utils.ftest(*vn*, *yk*)

Performs the F test for a line component

Compute F-test for single spectral line components at the frequency bins given by the mtspec routines.

Parameters

vn [ndarray [npts,kspec]] Slepian sequences real

yk [ndarray, complex [nfft,kspec]] multitaper eigencoefficients, complex kspec fft's of tapered data series

Returns

F : ndarray [nfft] vector of f-test values, real p : ndarray [nfft] vector with probability of line component

Calls

scipy.stats.f.cdf, scipy.stats.f.cdf

utils.get_data(*fname*)

Utility function to download the data from the Zenodo repository with the direct URL path (fixed).

Parameters

fname [char] filename of the data to download

Returns

data [ndarray] numpy array with the downloaded data In case of error, data = 0 is returned

utils.jackspec(*spec*, *sk*, *wt*, *se*)

code to calculate adaptively weighted jackknifed 95% confidence limits

Parameters

spec [ndarray [nfft]] real vector containing adaptively weighted spectrum

sk [ndarray [nfft,kspec]] array with kth power spectra

wt [ndarray [nfft,kspec]] real array containing the ne weights for kspec eigenspectra normalized so that if there is no bias, the weights are unity.

se [ndarray [nfft]] real vector containing the number of degrees of freedom for the spectral estimate at each frequency.

Returns

spec_ci [ndarray [nfft,2]] real array of jackknife error estimates, with 5 and 95% confidence intervals of the spectrum.

Calls

scipy.stats.t.ppf

Modified

German Prieto, Aug 2006

German Prieto, March 2007

Changed the Jackknife to be more efficient.

utils.north(*n, il, i2, s*)

Performs LS fit to *s* by a degree-two polynomial in an orthogonal basis. Function to be run with the Sine multitaper codes.

Returns

ds [float] estimate of 1st derivative ds/dn at center of record

dds [float] estimate of 2nd derivative

utils.qiinv(*spec, yk, wt, vn, lamb, nw*)

Function to calculate the Quadratic Spectrum using the method developed by Prieto et al. (2007).

The first 2 derivatives of the spectrum are estimated and the bias associated with curvature (2nd derivative) is reduced.

Calculate the Stationary Inverse Theory Spectrum. Basically, compute the spectrum inside the innerband.

This approach is very similar to D.J. Thomson (1990).

Parameters

spec [ndarray [nfft,0]] the adaptive multitaper spectrum (so far)

yk [ndarray, complex [npts,kspec]] multitaper eigencoefficients, complex

wt [ndarray [nf,kspec]] the weights of the different coefficients. input is the original multitaper weights, from the Thomson adaptive weighting. **vn** : ndarray [npts,kspec] the Slepian sequences

lambda [ndarray [kspec]] the eigenvalues of the Slepian sequences

nw [float] The time-bandwidth product

Returns

qispec [ndarray [nfft,0]] the QI spectrum estimate

ds [ndarray [nfft,0]] the estimate of the first derivative

dds [ndarray [nfft,0]] the estimate of the second derivative

References

G. A. Prieto, R. L. Parker, D. J. Thomson, F. L. Vernon, and R. L. Graham (2007), Reducing the bias of multitaper spectrum estimates, *Geophys. J. Int.*, 171, 1269-1281. doi: 10.1111/j.1365-246X.2007.03592.x.

Notes

In here I have made the Chebyshev polynomials unitless, meaning that the associated parameters ALL have units of the PSD and need to be normalized by $1/W$ for *lpha_1*, $1/W^{*2}$ for *lpha_2*, etc.

Modified

Nov 2021 (German A Prieto)

Major adjustment in the inverse problem steps. Now, the constant term is first inverted for, and then the 1st and 2nd derivative so that we obtain an independent 2nd derivative.

June 5, 2009 (German A. Prieto)

Major change, saving some important values so that if the subroutine is called more than once, with similar values, many of the variables are not calculated again, making the code run much faster.

Calls

scipy.optimize.nnls, scipy.linalg.qr, scipy.linalg.lstsq

`utils.sadapt(nptwo, fx, nf, df, initap, ntimes, fact)`

Performs the (sine multitaper) adaptive spectral estimation From a basic pilot estimate, computes S'' to be used in (13) of Riedel and Sidorenko (1995) for the MSE spectrum.

Parameters

nptwo [int] The twice signal length ($2 \cdot npts$)

fx [ndarray, complex [nptwo]] The FFT of the two signals (twice length)

nf [int] Number of frequency points for spec

df [float] Freq sampling

initap [int]

Number of tapers to use for pilot estimate Later we can add the spec result as test

ntimes [int] number of iterations for estimate

fact [float] degree of smoothing (def = 1.0)

Returns

spec [ndarray (nf)] the spectral estimate

kopt [ndarray, int [nf]] the number of tapers at each frequency.

References

Based on the sine multitaper code of R. L. Parker.

Calls

squick, north, curb

`utils.sadapt2(nptwo, fx, nf, df, initap, ntimes, fact)`

Performs the adaptive spectral estimation From a basic pilot estimate, computes S'' to be used in (13) of Riedel and Sidorenko (1995) for the MSE spectrum.

Parameters

nptwo [int] The twice signal length ($2 \cdot npts$)

fx [ndarray, complex [nptwo,2]] The FFT of the two signals (twice length)

nf [int] Number of frequency points for spec

df [float] Freq sampling

initap [int]

Number of tapers to use for pilot estimate Later we can add the spec result as test

ntimes [int] number of iterations for estimate

fact [float] degree of smoothing (def = 1.0)

Returns

spec [ndarray (nf,4)] the spectral estimate and coherence, phase

kopt [ndarray, int [nf]] the number of tapers at each frequency.

References

Based on the sine multitaper code of R. L. Parker.

Calls

squick, north, curb

Calls

squick2, orthog

utils.set_xint(ising)

Sets up weights and sample points for Ierley quadrature,

Slightly changed from original code, to avoid using common blocks. Also avoided using some go to statements, not needed.

Parameters

ising [integer]

ising=1 integrand is analytic in closed interval

ising=2 integrand may have bounded singularities at end points

Returns

w [ndarray (nomx,lomx+1)] weights

x [sample points (lomx+1)] sample points

lomx=number of samples = 2**nomx

Modified

November 2004 (German A. Prieto)

utils.sft(*x, om*)

calculates the (slow) fourier transform of real sequence $x(i), i=1, \dots, n$ at angular frequency *om* normalized so that $\text{nyquist}=\pi$. the sine transform is returned in *st* and the cosine transform in *ct*.

algorithm is that of goertzel with modifications by gentleman, comp.j. 1969

transform is not normalized

to normalize one-sided ft, divide by $\sqrt{\text{data length}}$ for positive *om*, the ft is defined as $\text{ct}-(0.,1.)\text{st}$ or like `slatec cfft`

Parameters

x [ndarray (n,)] time sequence $x[0], x[1], \dots$

om [float] angular frequency of interest, normalized such that $\text{Nyq} = \pi$

Modified

German Prieto November 2004

utils.squick(*nptwo, fx, nf, ntap=None, kopt=None*)

Sine multitaper routine. With a double length FFT constructs $\text{FT}[\sin(q*n)*x(n)]$ from $F[x(n)]$, that is constructs the FFT of the sine tapered signal.

The FFT should be performed previous to the call.

Parameters

nptwo [float] The twice signal length ($2*npts$)

fx [ndarray, complex] The FFT of the signal (twice length)

nf [int] Number of frequency points for spec

ntap [int, optional] Constant number of tapers to average from if *None*, *kopt* is used. if > 0 Constant value to be used if ≤ 0 Use the *kopt* array instead

ktop [ndarray, int [nf]] array of integers, with the number of tapers at each frequency.

Returns

spec [ndarray (nf,)] the spectral estimate

References

Based on the sine multitaper code of R. L. Parker.

utils.squick2(*nptwo, fx, nf, ntap=None, kopt=None*)

Sine multitaper routine. With a double length FFT constructs $\text{FT}[\sin(q*n)*x(n)]$ from $F[x(n)]$, that is constructs the FFT of the sine tapered signal.

The FFT should be performed previous to the call.

Parameters

nptwo [float] The twice signal length ($2*npts$)

fx [ndarray, complex [nptwo,2]] The FFT of the two signals (twice length)

nf [int] Number of frequency points for spec

ntap [int, optional``] Constant number of tapers to average from if > 0 Constant value to be used if None kopt used if <= 0 Use the kopt array instead

kopt [ndarray, int [nf]] array of integers, with the number of tapers at each frequency.

Returns

spec [ndarray (nf,4)] the spectral estimates (first 2 columns) and the cross spectral estimates (last 2 columns)

References

Based on the sine multitaper code of R. L. Parker.

utils.wt2dof(wt)

Calculate the degrees of freedom of the multitaper based on the weights of the different tapers.

Parameters

wt [ndarray [nfft,kspec]] weights of the tapers at each frequency

Returns

se [ndarray [nfft]] degrees of freedom at each frequency

Modified

February 2022, changed a for loop for direct numpy sum.

utils.xint(a, b, tol, vn, npts)

Quadrature by Ierley's method of Chebychev sampling.

Parameters

a [float] upper limit of integration

b [float] upper limit of integration

tol [float] tolerance for integration

vn [ndarray] taper or Slepian sequence to convert-integrate

npts [int] number of points of tapers

Notes

This is a slight variation of Gleen Ierly's code. What was mainly done, was to avoid use of common blocks, defining all variables and performing the numerical integration inside (previously done by function pssevf).

Exponential convergence rate for analytic functions! Much faster than Romberg; competitive with Gauss integration, without awkward weights.

Integrates the function dpsw on (a, b) to absolute accuracy tol > 0.

the function in time is given by rpar with ipar points

I removed the optional printing routine part of the code, to make it easier to read. I also moved both nval, etol as normal variables inside the routine.

nval = number of function calls made by routine etol = approximate magnitude of the error of the result

NB: function set_xint is called once before xint to provide quadrature samples and weights.

I also altered the subroutine call, to get the weights and not save them in a common block, but get them directly back.

lomx=number of samples = 2**nomx

Modified

November 2004 (German A. Prieto)

Calls

utils.set_xint

utils.**yk_reshape**(yk_in, vn, p=None, fcrit=0.95)

reshape the yk's based on the F-test of line compenents

Reshape eigenft's around significant spectral lines The "significant" means above fcritical probability (def=0.95)
If probability is large at neighbouring frequencies, code will only remove the largest probability energy.

Parameters

yk [ndarray complex [nfft,kspec]] eigenft's

vn [ndarray [npts,kspec]] DPSS sequences

p [ndarray optional [nfft]] F-test probabilities to find fcritical In None, it will be calculated

fcrit [float optional] Probability value over which to reshape, default = 0.95

Returns

yk [ndarray, complex [nfft,kspec]] Reshaped eigenft's

sline [ndarray [nfft]] Power spetrum of line components only

Modified

April 2006 (German A. Prieto)

Calls

ftest - if P is not present scipy.fft.fft

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

m

[mtcross](#), 13

[mtspec](#), 5

u

[utils](#), 19

A

`adaptspec()` (in module *utils*), 19

C

`copy_examples()` (in module *utils*), 20

`curb()` (in module *utils*), 20

D

`df_spec()` (in module *utils*), 21

`df_spec()` (*mtspec.MTSpec* method), 8

`df_spec_old()` (in module *utils*), 21

`dpss()` (in module *utils*), 22

`dpss2()` (in module *utils*), 23

`dpss_ev()` (in module *utils*), 23

E

`eigenspec()` (in module *utils*), 24

F

`fctest()` (in module *utils*), 24

`fctest()` (*mtspec.MTSpec* method), 8

G

`get_data()` (in module *utils*), 25

J

`jackspec()` (in module *utils*), 25

`jackspec()` (*mtspec.MTSpec* method), 8

M

module

mtcross, 13

mtspec, 5

utils, 19

`mt_corr()` (*mtcross.MTCross* method), 14

`mt_corr()` (*mtcross.SineCross* method), 17

`mt_deconv()` (*mtcross.MTCross* method), 15

`mt_deconv()` (*mtcross.SineCross* method), 17

mtcross

 module, 13

MTCross (class in *mtcross*), 13

MTSine (class in *mtspec*), 5

mtspec

 module, 5

MTSpec (class in *mtspec*), 6

N

`north()` (in module *utils*), 26

Q

`qiinv()` (in module *utils*), 26

`qiinv()` (*mtspec.MTSpec* method), 9

R

`reshape()` (*mtspec.MTSpec* method), 9

`rspec()` (*mtspec.MTSpec* method), 10

S

`sadapt()` (in module *utils*), 27

`sadapt2()` (in module *utils*), 27

`set_xint()` (in module *utils*), 28

`sft()` (in module *utils*), 28

SineCross (class in *mtcross*), 15

`spectrogram()` (in module *mtspec*), 10

`squick()` (in module *utils*), 29

`squick2()` (in module *utils*), 29

U

utils

 module, 19

W

`wt2dof()` (in module *utils*), 30

X

`xint()` (in module *utils*), 30

Y

`yk_reshape()` (in module *utils*), 31