PyYAWT Documentation

Release 0.1.2.dev0+f4f6416

Holger Nahrstaedt

March 04, 2016

Contents

CHAPTER 1

PyYAWT - Yet Another Wavelet Toolbox in Python

PyYAWT is a free Open Source wavelet toolbox for Python programming language.

This toolbox is aimed to mimic matlab wavelet toolbox. Most of the functions are similiar to their counterparts in Matlab equivalents.

```
>>> import pyyawt
>>> cA, cD = pyyawt.dwt([1, 2, 3, 4], 'db1')
```

1.1 Main features

The main features of PyYAWT are:

- 1D, 2D and 3D Forward and Inverse Discrete Wavelet Transform (DWT and IDWT)
- 1D and 2D Stationary Wavelet Transform (Undecimated Wavelet Transform)
- Continuous wavelet transform
- Dualtree real and complex wavelet transform
- Double precision calculations
- Results compatible with Matlab Wavelet Toolbox (TM)

1.2 Requirements

PyYAWT is a package for the Python programming language. It requires:

- Python 2.7 or >=3.3
- Numpy >= 1.6.2

1.3 Download

The most recent *development* version can be found on GitHub at https://github.com/holgern/pyyawt.

Latest release, including source and binary package for Windows, is available for download from the Python Package Index or on the Releases Page.

1.4 Install

In order to build PyYAWT from source, a working C compiler (GCC or MSVC) and a recent version of Cython is required.

- Install PyYAWT with `pip install pyyawt`.
- To build and install from source, navigate to downloaded PyYAWT source code directory and type `python setup.py install`.

Prebuilt Windows binaries and source code packages are also available from Python Package Index.

See also:

Development notes section contains more information on building and installing from source code.

1.5 Documentation

Documentation with detailed examples and links to more resources is available online at http://pyyawt.readthedocs.org.

For more usage examples see the demo directory in the source package.

1.6 State of development & Contributing

PyYAWT started in 2009 as a scilab toolbox and was maintained until 2009 by its original developer. Authors were * Professor Mei, Supervisor * Roger Liu * Isaac Zhi * Jason Huang * Du HuiQian

In 2010, maintenance was taken over in a new repo) by Holger Nahrstaedt. Daubechies wavelets coefficents DB2 - DB50 were calculated by Bob Strunz - University of Limerick, Ireland

Finally, all the c-source files from the SWT-Toolbox are forked into this python toolbox pyawt)

Contributions recarding bug reports, bug fixes and new features are welcome.

1.7 Python 3

Python 3.x is fully supported from release v0.0.1 on.

1.8 Contact

Use GitHub Issues to post your comments or questions.

1.9 License

PyYAWT is a free Open Source software released under the GPL license.

1.10 Contents

1.10.1 Development notes

This section contains information on building and installing PyYAWT from source code as well as instructions for preparing the build environment on Windows and Linux.

Preparing Windows build environment

To start developing PyYAWT code on Windows you will have to install a C compiler and prepare the build environment.

Installing Microsoft Visual C++ Compiler for Python 2.7

Downloading Microsoft Visual C++ Compiler for Python 2.7 from https://www.microsoft.com/en-us/download/details.aspx?id=44266.

After installing the Compiler and before compiling the extension you have to configure some environment variables.

For build execute the `util/setenv_win.bat` script in the cmd window:

```
rem Configure the environment for builds.
rem Convince setup.py to use the SDK tools.
set MSSdk=1
set DISTUTILS_USE_SDK=1
```

Next steps

After completing these steps continue with *Installing build dependencies*.

Preparing Linux build environment

There is a good chance that you already have a working build environment. Just skip steps that you don't need to execute.

Installing basic build tools

Note that the example below uses `aptitude` package manager, which is specific to Debian and Ubuntu Linux distributions. Use your favourite package manager to install these packages on your OS.

```
aptitude install build-essential gcc python-dev git-core
```

Next steps

After completing these steps continue with *Installing build dependencies*.

Installing build dependencies

Setting up Python virtual environment

A good practice is to create a separate Python virtual environment for each project. If you don't have virtualenv yet, install and activate it using:

```
curl -O https://raw.github.com/pypa/virtualenv/master/virtualenv.py
python virtualenv.py <name_of_the_venv>
. <name_of_the_venv>/bin/activate
```

Installing Cython

Use `pip` (http://pypi.python.org/pypi/pip) to install Cython:

```
pip install Cython>=0.16
```

Installing numpy

Use `pip` to install numpy:

```
pip install numpy
```

It takes some time to compile numpy, so it might be more convenient to install it from a binary release.

Note: Installing numpy in a virtual environment on Windows is not straightforward.

It is recommended to download a suitable binary `.exe` release from http://www.scipy.org/Download/ and install it using `easy_install` (i.e. `easy_install numpy-1.6.2-win32-superpack-python2.7.exe`).

Note: You can find binaries for 64-bit Windows on http://www.lfd.uci.edu/~gohlke/pythonlibs/.

Installing Sphinx

Sphinx is a documentation tool that converts reStructuredText files into nicely looking html documentation. Install it with:

```
pip install Sphinx
```

Building and installing PyYAWT

Installing from source code

Go to https://github.com/holgern/pyyawt GitHub project page, fork and clone the repository or use the upstream repository to get the source code:

```
git clone https://github.com/holgern/pyyawt.git pyyawt
```

Activate your Python virtual environment, go to the cloned source directory and type the following commands to build and install the package:

```
python setup.py build
python setup.py install
```

To verify the installation run the following command:

```
python setup.py test
```

To build docs:

```
cd doc
make html
```

Installing from source code in Windows

Go to https://github.com/holgern/pyyawt GitHub project page, fork and clone the repository or use the upstream repository to get the source code:

```
git clone https://github.com/holgern/pyyawt.git pyyawt
```

Install Microsoft Visual C++ Compiler for Python 2.7 from https://www.microsoft.com/en-us/download/details.aspx?id=44266

Activate your Python virtual environment, go to the cloned source directory and type the following commands to build and install the package:

```
    util\setenv_win.bat
python setup.py build_ext --inplace
python setup.py install --user
```

To verify the installation run the following command:

```
python runtests.py
```

To build docs:

```
cd doc
make html
```

Installing a development version

You can also install directly from the source repository:

```
pip install -e git+https://github.com/holgern/pyyawt.git#egg=pyyawt
```

or:

```
pip install pyyawt==dev
```

Installing a regular release from PyPi

A regular release can be installed with pip or easy_install:

```
pip install pyyawt
```

Testing

Continous integration with Travis-CI

The project is using Travis-CI service for continous integration and testing.

Current build status is: If you are submitting a patch or pull request please make sure it does not break the build.

Running tests locally

Tests are implemented with nose, so use one of:

$ nosetests pyyawt

```
>>> pywt.test()
```

Running tests with Tox

There's also a config file for running tests with Tox (`pip install tox`). To for example run tests for Python 2.7 and Python 3.4 use:

```
tox -e py27,py34
```

For more information see the Tox documentation.

Something not working?

If these instructions are not clear or you need help setting up your development environment, go ahead and open a ticket on GitHub.

1.10.2 Resources

Code

The GitHub repository is now the main code repository.

If you are using the Mercurial repository at Bitbucket, please switch to Git/GitHub and follow for development updates.

Questions and bug reports

Use GitHub Issues to post questions and open tickets.

Articles

1.10.3 pyyawt

pyyawt package

Submodules

pyyawt.cowt module

cowt function for wavelet denoising

pyyawt.cowt.**FSfarras**(**args*)

pyyawt.cowt.**dualfilt1**(**args*)

pyyawt.cowt.**dualtree**(**args*)

pyyawt.cowt.**idualtree**(**args*)

pyyawt.cowt.**dualtree2D**(**args*)

pyyawt.cowt.**idualtree2D**(**args*)

pyyawt.cowt.**cplxdual2D**(**args*)

pyyawt.cowt.**icplxdual2D**(**args*)

pyyawt.cwt module

cwt function for wavelet denoising

pyyawt.cwt.**sinus**(**args*)

pyyawt.cwt.**poisson**(**args*)

pyyawt.cwt.**mexihat**(**args*)

pyyawt.cwt.**morlet**(**args*)

pyyawt.cwt.**DOGauss**(**args*)

pyyawt.cwt.**Gauswavf**(**args*)

pyyawt.cwt.**shanwavf**(**args*)

pyyawt.cwt.**cmorlet**(**args*)

pyyawt.cwt.**fbspwavf**(**args*)

pyyawt.cwt.**cauchy**(**args*)

pyyawt.cwt.**cgauss**(**args*)

pyyawt.cwt.**meyer**(**args*)

pyyawt.cwt.**meyeraux**(**args*)

pyyawt.cwt.**wavefun**(**args*)

pyyawt.cwt.**wavefun2**(**args*)

pyyawt.denoising module

dwt1d function for wavelet

`pyyawt.denoising.`**`wnoisest`**(*C*, *L=None*, *S=None*)

estimates of the detail coefficients' standard deviation for levels contained in the input vector S

C: array_like coefficent array

L: array_like coefficent array

S: array_like estimate noise for this decompostion levels

STDC: array_like STDC[k] is an estimate of the standard deviation of C[k]

[c,l] = wavedec(x,2,'db3') wnoisest(c,l,[0,1])

`pyyawt.denoising.`**`wden`**(**args*)

wden performs an automatic de-noising process of a one-dimensional signal using wavelets. Calling Sequence ——————- [XD,CXD,LXD] = wden(X,TPTR,SORH,SCAL,N,wname) [XD,CXD,LXD] = wden(C,L,TPTR,SORH,SCAL,N,wname) Parameters ———- x: array_like

input vector

C: array_like coefficent array

L: array_like coefficent array

TPTR: str threshold selection rule 'rigrsure' uses the principle of Stein's Unbiased Risk. 'heursure' is an heuristic variant of the first option. 'sqtwolog' for universal threshold 'minimaxi' for minimax thresholding

SORH: str ('s' or 'h') soft or hard thresholding

SCAL: str 'one' for no rescaling 'sln' for rescaling using a single estimation of level noise based on first-level coefficients 'mln' for rescaling done using level-dependent estimation of level noise

N: int N: decompostion level

wname: str wavelet name

XD: array_like de-noised signal

CXD: array_like de-noised coefficent array

LXD: array_like de-noised length array

[xref,x] = wnoise(3,11,3) level = 4 xd = wden(x,'heursure','s','one',level,'sym8')

`pyyawt.denoising.`**`thselect`**(*X*, *TPTR*)

Threshold selection for de-noising. The algorithm works only if the signal X has a white noise of N(0,1). Dealing with unscaled or nonwhite noise can be handled using rescaling of the threshold.

X: array input vector with scaled white noise (N(0,1))

TPTR: str 'rigrsure': adaptive threshold selection using principle of Stein's Unbiased Risk Estimate. 'heursure': heuristic variant of the first option. 'sqtwolog': threshold is sqrt(2*log(length(X))). 'minimaxi': minimax thresholding.

THR: float threshold X-adapted value using selection rule defined by string TPTR

x = np.random.randn(1000) thr = thselect(x,'rigrsure') `pyyawt.denoising.`**`ValSUREThresh`**`(`*X*`)`

Adaptive Threshold Selection Using Principle of SURE

X: array Noisy Data with Std. Deviation = 1

tresh: float Value of Threshold

`pyyawt.denoising.`**`dyadlength`**`(`*x*`)`

Find length and dyadic length of array

X: array array of length n = 2^J (hopefully)

n: int length(x)

J: int least power of two greater than n

`pyyawt.denoising.`**`wthresh`**`(`*X*, *SORH*, *T*`)`

doing either hard (if SORH = 'h') or soft (if SORH = 's') thresholding

X: array input data (vector or matrix)

SORH: str 's': soft thresholding 'h' : hard thresholding

T: float threshold value

Y: array_like output

y = np.linspace(-1,1,100) thr = 0.4 ythard = wthresh(y,'h',thr) ytsoft = wthresh(y,'s',thr)

`pyyawt.denoising.`**`wnoise`**`(`*FUN*, *N*, *SQRT_SNR=1*`)`

Noisy wavelet test data

FUN: str / int 1 or 'blocks' 2 or 'bumps' 3 or 'heavy sine' 4 or 'doppler' 5 or 'quadchirp' 6 or 'mishmash'

N: int vector length of X = 2^N

SQRT_SNR: float standard deviation of added noise

X: array_like test data

XN: array_like noisy test data (rand(1,N,'normal') is added!)

[x,noisyx] = wnoise(4,10,7);

pyyawt.dwt module

Helper function for wavelet denoising

`pyyawt.dwt.`**`orthfilt`**`(`*w*`)`

orthfilt is an utility function for obtaining analysis and synthesis filter set of given orthogonal wavelets including haar, daubechies, coiflets and symlets

w: array_like scaling filter

Lo_D: array_like lowpass analysis filter

Hi_D: array_like highpass analysis filter

Lo_R: array_like lowpass synthesis filter

Hi_R: array_like highpass synthesis filter

F = dbwavf("db2") [lo_d,hi_d,lo_r,hi_r]=orthfilt(F)

`pyyawt.dwt.`**`biorfilt`**(*df*, *rf*)

biorfilt is an utility function for obtaining analysis and synthesis filter set of given bi-orthogonal spline wavelets. DF and RF should be output of biorfilt result with the same length.

df: array_like analysis scaling filter

rf: array_like synthesis scaling filter

Lo_D: array_like lowpass analysis filter

Hi_D: array_like highpass analysis filter

Lo_R: array_like lowpass synthesis filter

Hi_R: array_like highpass synthesis filter

RF,DF = biorwavf('bior3.3') [lo_d,hi_d,lo_r,hi_r]=biorfilt(DF,RF)

`pyyawt.dwt.`**`dbwavf`**(*wname*)

dbwavf is an utility function for obtaining scaling filter of daubechies wavelet.

wname: str wavelet name, 'db1' to 'db36'

F: array_like scaling filter

F = dbwavf("db2")

`pyyawt.dwt.`**`coifwavf`**(*wname*)

coifwavf is an utility function for obtaining scaling filter of coiflets wavelet.

wname: str wavelet name, 'coif1' to 'coif5'

F: array_like scaling filter

F = coifwavf('coif3')

`pyyawt.dwt.`**`symwavf`**(*wname*)

symwavf is an utility function for obtaining scaling filter of symlets wavelet.

wname: str wavelet name, 'sym2' to 'sym20'

F: array_like scaling filter

F = symwavf('sym7')

`pyyawt.dwt.`**`legdwavf`**(*wname*)

legdwavf is an utility function for obtaining scaling filter of legendre wavelet.

wname: str wavelet name, 'legd1' to 'legd9'

F: array_like scaling filter

F = legdwavf('sym7')

`pyyawt.dwt.`**`biorwavf`**(*wname*)

biorwavf is an utility function for obtaining twin scaling filters of bi-orthogonal spline wavelet including bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8, bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4, bior5.5 and bior6.8. Although the twin filters have different length, zeros has been fed to keep two filters the same length.

wname: str wavelet name, 'bior1.1' to 'bior6.8'

RF: array_like synthesis scaling filter

DF: array_like analysis scaling filter

RF,DF = biorwavf('bior3.3');

`pyyawt.dwt.rbiorwavf(wname)`

rbiorwavf is an utility function for obtaining twin scaling filters of bi-orthogonal spline wavelet including bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8, bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4, bior5.5 and bior6.8. Although the twin filters have different length, zeros has been fed to keep two filters the same length. rbiorwavf is reversing the results of biorwavf.

wname: str wavelet name, 'rbior1.1' to 'rbior6.8'

RF: array_like synthesis scaling filter

DF: array_like analysis scaling filter

[RF,DF]=rbiorwavf('rbior3.3')

`pyyawt.dwt.wfilters(wname, filterType=None)`

wfilters is an utility function for obtaining analysis and synthesis filter set.

[Lo_D,Hi_D,Lo_R,Hi_R]=wfilters(wname) [Lo_D,Hi_D]=wfilters(wname,'d') [Lo_R,Hi_R]=wfilters(wname,'r') [Lo_D,Lo_R]=wfilters(wname,'l') [Hi_D,Hi_R]=wfilters(wname,'h')

wname: str wavelet name, wavelet name, haar("haar"), daubechies ("db1" to "db20"), coiflets ("coif1" to "coif5"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8"

Lo_D: array_like lowpass analysis filter

Hi_D: array_like highpass analysis filter

Lo_R: array_like lowpass synthesis filter

Hi_R: array_like highpass synthesis filter

[lo_d,hi_d,lo_r,hi_r]=wfilters('db2')

`pyyawt.dwt.wmaxlev(signalLength, wname)`

wmaxlev is the maximum decompostion level calculation utility.

signalLength: int signal length

wname: str wavelet name, wavelet name, haar("haar"), daubechies ("db1" to "db20"), coiflets ("coif1" to "coif5"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8"

L: int decomposition level

L=wmaxlev(100,'db5')

`pyyawt.dwt.dwtmode(mode=None, nodisp=None)`

dwtmode is to display or change extension mode.

mode: str 'symh'('sym'), 'symw', 'asymh', 'asymw', 'zpd', 'zpd', 'per', 'ppd'.

nodisp: str None or 'nodisp'

mode: str extension mode

dwtmode() dwtmode('status') mode=dwtmode('status','nodisp') dwtmode(mode)

pyyawt.dwt1d module

Helper function for wavelet denoising

`pyyawt.dwt1d.`**`dwt`**`(`*x*, **args*`)`

[cA,cD]=dwt(x,wname,['mode',extMethod]) [cA,cD]=dwt(x,Lo_D,Hi_D,['mode',extMethod]) ——— Parameters ——— wname: str

wavelet name, wavelet name, haar("haar"), daubechies ("db1" to "db20"), coiflets ("coif1" to "coif5"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8"

x [array_like] input vector

Lo_D [array_like] lowpass analysis filter

Hi_D [array_like] highpass analysis filter

extMethod [str] extension mode, 'zpd' for example

cA: array_like approximation coefficent

cD: array_like detail coefficent

cA,cD=dwt(x,'db2','mode','asymh')

`pyyawt.dwt1d.`**`idwt`**`(`*cA*, *cD*, **args*`)`

Inverse Discrete Fast Wavelet Transform

X=idwt(cA,cD,wname,[L],['mode',extMethod]) X=idwt(cA,cD,Lo_R,Hi_R,[L],['mode',extMethod])

wname: wavelet name, haar("haar"), daubechies ("db1" to "db36"), coiflets ("coif1" to "coif17"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" x : reconstructed vector Lo_R: lowpass synthesis filter Hi_R: highpass syntheis filter L : restruction length cA: approximation coefficent cD: detail coefficent

idwt is for inverse discrete fast wavelet transform. Coefficent could be void vector as '[]' for cA or cD.

x=np.random.rand(1,100) [cA,cD]=dwt(x,'db2','mode','asymh') x0=idwt(cA,cD,'db2',100)

`pyyawt.dwt1d.`**`wavedec`**`(`*x*, *N*, **args*`)`

Multiple level 1-D discrete fast wavelet decomposition

[C,L]=wavedec(X,N,wname) [C,L]=wavedec(X,N,Lo_D,Hi_D)

wname : wavelet name, haar("haar"), daubechies ("db1" to "db36"), coiflets ("coif1" to "coif17"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" X : signal vector N : decompostion level Lo_D : lowpass analysis filter Hi_D : highpass analysis filter C : coefficient vector L : length vector

wavedec can be used for multiple-level 1-D discrete fast wavelet decompostion using a specific wavelet name wname or wavelet decompostion filters Lo_D and Hi_D. Such filters can be generated using wfilters.

The global extension mode which can be change using dwtmode is used.

The coefficient vector C contains the approximation coefficient at level N and all detail coefficient from level 1 to N

The first entry of L is the length of the approximation coefficent, then the length of the detail coefficients are stored and the last value of L is the length of the signal vector.

The approximation coefficient can be extracted with C(1:L(1)). The detail coefficients can be obtained with C(L(1):sum(L(1:2))), C(sum(L(1:2)):sum(L(1:3))),.... until C(sum(L(1:length(L)-2)):sum(L(1:length(L)-1)))

X = wnoise(4,10,0.5); //doppler with N=1024 [C,L]=wavedec(X,3,'db2')

`pyyawt.dwt1d.`**`waverec`**(*C*, *L*, **args*)

Multiple level 1-D inverse discrete fast wavelet reconstruction

x0=waverec(C,L,wname) x0=waverec(C,L,Lo_R,Hi_R)

wname : wavelet name, haar("haar"), daubechies ("db1" to "db36"), coiflets ("coif1" to "coif17"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" x0 : reconstructed vector Lo_R : lowpass synthesis filter Hi_R : highpass synthesis filter C : coefficent array L : length array

waverec can be used for multiple-level 1-D inverse discrete fast wavelet reconstruction.

waverec supports only orthogonal or biorthogonal wavelets.

X = wnoise(4,10,0.5); //doppler with N=1024 [C,L]=wavedec(X,3,'db2'); x0=waverec(C,L,'db2'); err = sum(abs(X-x0))

`pyyawt.dwt1d.`**`wrcoef`**(*approx_or_detail*, *C*, *L*, **args*)

Restruction from single branch from multiple level decomposition

X=wrcoef(type,C,L,wname,[N]) X=wrcoef(type,C,L,Lo_R,Hi_R,[N])

type : approximation or detail, 'a' or 'd'. wname : wavelet name X : vector of reconstructed coefficents Lo_R : lowpass synthesis filter Hi_R : highpass syntheis filter C : coefficent array L : length array N : restruction level with length(L)-2>=N

wrcoef is for reconstruction from single branch of multiple level decomposition from 1-D wavelet coefficients. Extension mode is stored as a global variable and could be changed with dwtmode. If N is omitted, maximum level (length(L)-2) is used.

The wavelet coefficents C and L can be generated using wavedec.

x=rand(1,100) [C,L]=wavedec(x,3,'db2') x0=wrcoef('a',C,L,'db2',2)

`pyyawt.dwt1d.`**`appcoef`**(*C*, *L*, **args*)

1-D approximation coefficients extraction

A=appcoef(C,L,wname,[N]) A=appcoef(C,L,Lo_R,Hi_R,[N])

wname : wavelet name, haar("haar"), daubechies ("db1" to "db20"), coiflets ("coif1" to "coif5"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" A : extracted approximation coefficients Lo_R : lowpass synthesis filter Hi_R : highpass syntheis filter C : coefficent array L : length array N : restruction level with N<=length(L)-2

appcoef can be used for extraction or reconstruction of approximation coefficents at level N after a multiple level decompostion. Extension mode is stored as a global variable and could be changed with dwtmode. If N is omitted, the maximum level (length(L)-2) is used.

The length of A depends on the level N. C and L can be generated using wavedec.

X = wnoise(4,10,0.5) [C,L]=wavedec(X,3,'db2') A2=appcoef(C,L,'db2',2)

`pyyawt.dwt1d.`**`detcoef`**`(`*C*, *L*, *N=None*`)`

1-D detail coefficients extraction

D=detcoef(C,L,[N])

D : reconstructed detail coefficient C : coefficent array L : length array N : restruction level with N<=length(L)-2 Description ——— detcoef is for extraction of detail coeffient at different level after a multiple level decompostion. Extension mode is stored as a global variable and could be changed with dwtmode. If N is omitted, the detail coefficients will extract at the maximum level (length(L)-2).

The length of D depends on the level N.

C and L can be generated using wavedec.

X = wnoise(4,10,0.5); //doppler with N=1024 [C,L]=wavedec(X,3,'db2'); D2=detcoef(C,L,2)

`pyyawt.dwt1d.`**`wenergy`**`(`*C*, *L*`)`

Energy Statistics from multiple level decompostion

[Ea,Ed]=wenergy(c,l)

Ea : energy percentage of approximation coefficent Ed : energy percentage of detail coefficent, vector c : coefficent array l : length array

wenergy is to calculate the energy percentage of approximation and detail coefficent.

x=rand(1,100) [C,L]=wavedec(x,3,'db2') [Ea,Ed]=wenergy(C,L)

`pyyawt.dwt1d.`**`upcoef`**`(`*aprox_or_detail*, *x*, **args*`)`

Direct Restruction

Y=upcoef(type,x,wname,[N],[L]) Y=upcoef(type,x,Lo_R,Hi_R,[N],[L])

type: approximation or detail, 'a' or 'd'. x: input vector wname: wavelet name, haar("haar"), daubechies ("db1" to "db36"), coiflets ("coif1" to "coif17"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" X: reconstruction Lo_R: lowpass synthesis filter Hi_R: highpass syntheis filter N: restruction level L: desired output length

upcoef is for upward reconstruction from any desired input vector.

x=rand(1,100); [cA,cD]=dwt(x,'db2') Y=upcoef('a',cA,'db2',1) Z=upcoef('a',cA,'db2',3)

`pyyawt.dwt1d.`**`upwlev`**`(`*C*, *L*, **args*`)`

Single Level Reconstruction from multiple level decompostion

[NC,NL,CA]=upwlev(c,l,wname) [NC,NL,CA]=upwlev(c,l,Lo_R,Hi_R)

wname: wavelet name, haar("haar"), daubechies ("db1" to "db36"), coiflets ("coif1" to "coif17"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" NC: upward level coefficent array NL: upward level length array CA: approximation coeffient at the last level Lo_R: lowpass synthesis filter Hi_R: highpass syntheis filter c: coefficent array l: length array

upwlev is for single level reconstruction.

x=rand(1,100) [C,L]=wavedec(x,3,'db2') [NC,NL,CA3]=upwlev(C,L,'db2')

pyyawt.dwt2d module

dwt2d function for wavelet denoising

`pyyawt.dwt2d.`**`dwt2`**(*x*, **args*)

`pyyawt.dwt2d.`**`idwt2`**(**args*)

`pyyawt.dwt2d.`**`wavedec2`**(**args*)

`pyyawt.dwt2d.`**`waverec2`**(**args*)

`pyyawt.dwt2d.`**`wenergy2`**(**args*)

`pyyawt.dwt2d.`**`detcoef2`**(**args*)

`pyyawt.dwt2d.`**`appcoef2`**(**args*)

`pyyawt.dwt2d.`**`wrcoef2`**(**args*)

`pyyawt.dwt2d.`**`upcoef2`**(**args*)

`pyyawt.dwt2d.`**`upwlev2`**(**args*)

pyyawt.dwt3d module

dwt3d function for wavelet

`pyyawt.dwt3d.`**`dwt3`**(*cA*, *cD*, **args*)

`pyyawt.dwt3d.`**`idwt3`**(*cA*, *cD*, **args*)

pyyawt.setup module

`pyyawt.setup.`**`configuration`**(*parent_package=''*, *top_path=None*)

pyyawt.swt module

swt function for wavelet

`pyyawt.swt.`**`swt`**(**args*)

`pyyawt.swt.`**`iswt`**(**args*)

`pyyawt.swt.`**`swt2`**(**args*)

`pyyawt.swt.`**`iswt2`**(**args*)

pyyawt.utility module

Helper function for pyyawt

`pyyawt.utility.`**`conv`**(*a*, *b*)

`pyyawt.utility.`**`iconv`**(**args*)

`pyyawt.utility.`**`wrev`**(**args*)

`pyyawt.utility.qmf(x, even_odd=None)`
quadrature mirror

Y=qmf(x,[EVEN_ODD])

x: double vector EVEN_ODD: even or odd integer

Y: quadrature mirror

qmf is a quadrature mirror utility function on time domain. If EVEN_ODD is an even integer, output would be reversed version of input with even index entries sign changed. Otherwise, odd index entries will be changed. Default is even.

a=np.random.rand(3) Y=qmf(a)

`pyyawt.utility.dyaddown(x, *args)`
dyadic downsampling

Y=dyaddown(x,[EVEN_ODD]) Y=dyaddown(M,[EVEN_ODD],[type]) Y=dyaddown(M,[type],[EVEN_ODD])

x : double vector M : double matrix EVEN_ODD : even or odd integer type : downsampling manner, 'r' for row, 'c' for column, and 'm' for row and column simutaneously. Y : downsampling result

dyaddown is an utility function for dyadic downsampling. if EVEN_ODD is even, even index entries of input will be kept. Otherwise, odd index entries will be kept. Default is even. Optional argumet type is especially for matrix input downsampling.

a=np.random.rand((1,100)) Y=dyaddown(a) b=np.random.rand((25,25)) Y=dyaddown(b,'r',0)

`pyyawt.utility.dyadup(x, *args)`
dyadic upsampling

Y=dyadup(x,[EVEN_ODD]) Y=dyadup(M,[EVEN_ODD],[type]) Y=dyadup(M,[type],[EVEN_ODD])

x : double vector M : double matrix EVEN_ODD : even or odd integer type : upsampling manner, 'r' for row, 'c' for column, and 'm' for row and column simutaneously. Y : upsampling result

dyadup is an utility function for dyadic upsampling. if EVEN_ODD is even, zeors will be put between input entries and output length will be two times input length minus one. Otherwise, additional two zeros will be put at the head and tail of output so the output length will be two times input length plus one. Default is odd. Optional argumet type is especially for matrix input upsampling.

a=rand(1,100) Y=dyadup(a) b=rand(25,25) Y=dyadup(b,'r',0)

`pyyawt.utility.wkeep(x, *args)`
signal extraction

Y=wkeep(x,L,[type]) Y=wkeep(x,L,[FIRST]) Y=wkeep(M,S,[indexVector])

x : double vector M : double matrix L : length integer type: extraction manner, 'l' for left, 'r' for right, and 'c' for center FIRST: index integer from which extraction starts. S : size integer vector containing row size and column size wanted indexVector : row and column index integer vector from which extraction starts. Y : extraction result

wkeep is an utility function for both vector and matrix extraction. For vector extraction, extractions will be aligned to the right, left or center based on optional argument type. So does matrix extraction.

a = np.linspace(1,8,8) X=np.dot(np.array([a]).T,np.array([a])) Y=wkeep(X,[4, 4])

`pyyawt.utility.wextend(dim, extMethod, x, L, typeString=None)`
signal extension

Y=wextend(onedim,extMode,x,L,[type]) Y=wextend(twodim,extMode,M,sizeVector,[typeStringVector]) Y=wextend(twodim,extMode,M,sizeVector,[typeString]) Y=wextend(twodim,extMode,M,L) Y=wextend(row_col,extMode,M,L,[type]) x : double vector M : double matrix L : length integer type : extraction manner, 'l' for left, 'r' for right, and 'b' for both left and right sizeVector : integer vector containing row and column size to extend typeString : string for extension, 'bb', 'll', 'rr', 'bl', 'lb', 'br', 'rb', 'lr', 'rl'. typeStringVector : string vector for extension, ['b' 'b'], ['l' 'l'], ['r' 'r'], ['b' 'l'], ['l' 'b'], ['b' 'r'], ['r' 'b'], ['r' 'l'], ['l' 'r']. extMode : extension method, 'symh'('sym'), 'symw', 'asymh', 'asymw', 'zpd', 'zpd', 'per', 'ppd'. row_col : adding row or adding column, 'ar' or 'addrow' for row, 'ac' or 'addcol' for column. onedim : one dimension indication, 1, '1', '1d' and '1D' twodim : two dimension indication, 2, '2', '2d' and '2D' Y : extension result

wextend is an utility function for signal extension.

a=rand(1,100); Y=wextend(1,'symh',a,5,'b'); b=rand(25,25); Y=wextend(2,'symh',b,[3,5],'lb'); Y=wextend('ar','symh',b,3,'r');

`pyyawt.utility.`**`wcodemat`**(**args*)

`pyyawt.utility.`**`mat3Dtran`**(**args*)

`pyyawt.utility.`**`wrev3`**(**args*)

`pyyawt.utility.`**`wrev2`**(**args*)

`pyyawt.utility.`**`wnorm`**(**args*)

`pyyawt.utility.`**`waveletfamilies`**(**args*)

pyyawt.version module

Module contents

wavelet toolbox

`pyyawt.`**`DOGauss`**(**args*)

`pyyawt.`**`FSfarras`**(**args*)

`pyyawt.`**`Gauswavf`**(**args*)

`pyyawt.`**`Tester`**
alias of `NoseTester`

`pyyawt.`**`ValSUREThresh`**(*X*)
Adaptive Threshold Selection Using Principle of SURE

X: array Noisy Data with Std. Deviation = 1

tresh: float Value of Threshold

`pyyawt.`**`appcoef`**(*C*, *L*, **args*)
1-D approximation coefficients extraction

A=appcoef(C,L,wname,[N]) A=appcoef(C,L,Lo_R,Hi_R,[N])

wname : wavelet name, haar("haar"), daubechies ("db1" to "db20"), coiflets ("coif1" to "coif5"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" A : extracted approximation coefficients Lo_R : lowpass synthesis filter Hi_R : highpass syntheis filter C : coefficent array L : length array N : restruction level with N<=length(L)-2 appcoef can be used for extraction or reconstruction of approximation coefficents at level N after a multiple level decompostion. Extension mode is stored as a global variable and could be changed with dwtmode. If N is omitted, the maximum level (length(L)-2) is used.

The length of A depends on the level N. C and L can be generated using wavedec.

X = wnoise(4,10,0.5) [C,L]=wavedec(X,3,'db2') A2=appcoef(C,L,'db2',2)

`pyyawt.`**`appcoef2`**`(`**args*`)`

`pyyawt.`**`biorfilt`**`(`*df*, *rf*`)`

biorfilt is an utility function for obtaining analysis and synthesis filter set of given bi-orthogonal spline wavelets. DF and RF should be output of biorfilt result with the same length.

df: array_like analysis scaling filter

rf: array_like synthesis scaling filter

Lo_D: array_like lowpass analysis filter

Hi_D: array_like highpass analysis filter

Lo_R: array_like lowpass synthesis filter

Hi_R: array_like highpass synthesis filter

RF,DF = biorwavf('bior3.3') [lo_d,hi_d,lo_r,hi_r]=biorfilt(DF,RF)

`pyyawt.`**`biorwavf`**`(`*wname*`)`

biorwavf is an utility function for obtaining twin scaling filters of bi-orthogonal spline wavelet including bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8, bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4, bior5.5 and bior6.8. Although the twin filters have different length, zeros has been fed to keep two filters the same length.

wname: str wavelet name, 'bior1.1' to 'bior6.8'

RF: array_like synthesis scaling filter

DF: array_like analysis scaling filter

RF,DF = biorwavf('bior3.3');

`pyyawt.`**`cauchy`**`(`**args*`)`

`pyyawt.`**`cgauss`**`(`**args*`)`

`pyyawt.`**`cmorlet`**`(`**args*`)`

`pyyawt.`**`coifwavf`**`(`*wname*`)`

coifwavf is an utility function for obtaining scaling filter of coiflets wavelet.

wname: str wavelet name, 'coif1' to 'coif5'

F: array_like scaling filter

F = coifwavf('coif3')

`pyyawt.`**`conv`**`(`*a*, *b*`)`

`pyyawt.`**`cplxdual2D`**`(`**args*`)`

`pyyawt.`**`dbwavf`**`(`*wname*`)`

dbwavf is an utility function for obtaining scaling filter of daubechies wavelet.

wname: str wavelet name, 'db1' to 'db36'

F: array_like scaling filter

F = dbwavf("db2")

`pyyawt.detcoef(C, L, N=None)`

1-D detail coefficients extraction

D=detcoef(C,L,[N])

D : reconstructed detail coefficient C : coefficent array L : length array N : restruction level with N<=length(L)-2 Description ——— detcoef is for extraction of detail coeffient at different level after a multiple level decompostion. Extension mode is stored as a global variable and could be changed with dwtmode. If N is omitted, the detail coefficients will extract at the maximum level (length(L)-2).

The length of D depends on the level N.

C and L can be generated using wavedec.

X = wnoise(4,10,0.5); //doppler with N=1024 [C,L]=wavedec(X,3,'db2'); D2=detcoef(C,L,2)

`pyyawt.detcoef2(*args)`

`pyyawt.dualfilt1(*args)`

`pyyawt.dualtree(*args)`

`pyyawt.dualtree2D(*args)`

`pyyawt.dwt(x, *args)`

[cA,cD]=dwt(x,wname,['mode',extMethod]) [cA,cD]=dwt(x,Lo_D,Hi_D,['mode',extMethod]) ——— Parameters ——— wname: str

> wavelet name, wavelet name, haar("haar"), daubechies ("db1" to "db20"), coiflets ("coif1" to "coif5"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8"

x [array_like] input vector

Lo_D [array_like] lowpass analysis filter

Hi_D [array_like] highpass analysis filter

extMethod [str] extension mode, 'zpd' for example

cA: array_like approximation coefficent

cD: array_like detail coefficent

cA,cD=dwt(x,'db2','mode','asymh')

`pyyawt.dwt2(x, *args)`

`pyyawt.dwt3(cA, cD, *args)`

`pyyawt.dwtmode(mode=None, nodisp=None)`

dwtmode is to display or change extension mode.

mode: str 'symh'('sym'), 'symw', 'asymh', 'asymw', 'zpd', 'zpd', 'per', 'ppd'.

nodisp: str None or 'nodisp'

mode: str extension mode

dwtmode() dwtmode('status') mode=dwtmode('status','nodisp') dwtmode(mode)

pyyawt.**dyaddown**(*x*, **args*)
dyadic downsampling

Y=dyaddown(x,[EVEN_ODD]) Y=dyaddown(M,[EVEN_ODD],[type]) Y=dyaddown(M,[type],[EVEN_ODD])

x : double vector M : double matrix EVEN_ODD : even or odd integer type : downsampling manner, 'r' for row, 'c' for column, and 'm' for row and column simutaneously. Y : downsampling result

dyaddown is an utility function for dyadic downsampling. if EVEN_ODD is even, even index entries of input will be kept. Otherwise, odd index entries will be kept. Default is even. Optional argumet type is especially for matrix input downsampling.

a=np.random.rand((1,100)) Y=dyaddown(a) b=np.random.rand((25,25)) Y=dyaddown(b,'r',0)

pyyawt.**dyadlength**(*x*)
Find length and dyadic length of array

X: array array of length n = 2^J (hopefully)

n: int length(x)

J: int least power of two greater than n

pyyawt.**dyadup**(*x*, **args*)
dyadic upsampling

Y=dyadup(x,[EVEN_ODD]) Y=dyadup(M,[EVEN_ODD],[type]) Y=dyadup(M,[type],[EVEN_ODD])

x : double vector M : double matrix EVEN_ODD : even or odd integer type : upsampling manner, 'r' for row, 'c' for column, and 'm' for row and column simutaneously. Y : upsampling result

dyadup is an utility function for dyadic upsampling. if EVEN_ODD is even, zeors will be put between input entries and output length will be two times input length minus one. Otherwise, additional two zeros will be put at the head and tail of output so the output length will be two times input length plus one. Default is odd. Optional argumet type is especially for matrix input upsampling.

a=rand(1,100) Y=dyadup(a) b=rand(25,25) Y=dyadup(b,'r',0)

pyyawt.**fbspwavf**(**args*)

pyyawt.**iconv**(**args*)

pyyawt.**icplxdual2D**(**args*)

pyyawt.**idualtree**(**args*)

pyyawt.**idualtree2D**(**args*)

pyyawt.**idwt**(*cA*, *cD*, **args*)
Inverse Discrete Fast Wavelet Transform

X=idwt(cA,cD,wname,[L],['mode',extMethod]) X=idwt(cA,cD,Lo_R,Hi_R,[L],['mode',extMethod])

wname: wavelet name, haar("haar"), daubechies ("db1" to "db36"), coiflets ("coif1" to "coif17"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" x : reconstructed vector Lo_R: lowpass synthesis filter Hi_R: highpass syntheis filter L : restruction length cA: approximation coefficent cD: detail coefficent

idwt is for inverse discrete fast wavelet transform. Coefficent could be void vector as '[]' for cA or cD.

x=np.random.rand(1,100) [cA,cD]=dwt(x,'db2','mode','asymh') x0=idwt(cA,cD,'db2',100) pyyawt.**idwt2**(**args*)

pyyawt.**idwt3**(*cA*, *cD*, **args*)

pyyawt.**iswt**(**args*)

pyyawt.**iswt2**(**args*)

pyyawt.**legdwavf**(*wname*)

legdwavf is an utility function for obtaining scaling filter of legendre wavelet.

wname: str wavelet name, 'legd1' to 'legd9'

F: array_like scaling filter

F = legdwavf('sym7')

pyyawt.**mat3Dtran**(**args*)

pyyawt.**mexihat**(**args*)

pyyawt.**meyer**(**args*)

pyyawt.**meyeraux**(**args*)

pyyawt.**morlet**(**args*)

pyyawt.**orthfilt**(*w*)

orthfilt is an utility function for obtaining analysis and synthesis filter set of given orthogonal wavelets including haar, daubechies, coiflets and symlets

w: array_like scaling filter

Lo_D: array_like lowpass analysis filter

Hi_D: array_like highpass analysis filter

Lo_R: array_like lowpass synthesis filter

Hi_R: array_like highpass synthesis filter

F = dbwavf("db2") [lo_d,hi_d,lo_r,hi_r]=orthfilt(F)

pyyawt.**poisson**(**args*)

pyyawt.**qmf**(*x*, *even_odd=None*)

quadrature mirror

Y=qmf(x,[EVEN_ODD])

x: double vector EVEN_ODD: even or odd integer

Y: quadrature mirror

qmf is a quadrature mirror utility function on time domain. If EVEN_ODD is an even integer, output would be reversed version of input with even index entries sign changed. Otherwise, odd index entries will be changed. Default is even.

a=np.random.rand(3) Y=qmf(a)

pyyawt.**rbiorwavf**(*wname*)

rbiorwavf is an utility function for obtaining twin scaling filters of bi-orthogonal spline wavelet including bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8, bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4, bior5.5 and bior6.8. Although the twin filters have different length, zeros has been fed to keep two filters the same length. rbiorwavf is reversing the results of biorwavf.

wname: str wavelet name, 'rbior1.1' to 'rbior6.8'

RF: array_like synthesis scaling filter

DF: array_like analysis scaling filter

[RF,DF]=rbiorwavf('rbior3.3')

`pyyawt.`**`shanwavf`**`(`**args*`)`

`pyyawt.`**`sinus`**`(`**args*`)`

`pyyawt.`**`swt`**`(`**args*`)`

`pyyawt.`**`swt2`**`(`**args*`)`

`pyyawt.`**`symwavf`**`(`*wname*`)`

symwavf is an utility function for obtaining scaling filter of symlets wavelet.

wname: str wavelet name, 'sym2' to 'sym20'

F: array_like scaling filter

F = symwavf('sym7')

`pyyawt.`**`thselect`**`(`*X*, *TPTR*`)`

Threshold selection for de-noising. The algorithm works only if the signal X has a white noise of N(0,1). Dealing with unscaled or nonwhite noise can be handled using rescaling of the threshold.

X: array input vector with scaled white noise (N(0,1))

TPTR: str 'rigrsure': adaptive threshold selection using principle of Stein's Unbiased Risk Estimate. 'heursure': heuristic variant of the first option. 'sqtwolog': threshold is sqrt(2*log(length(X))). 'minimaxi': minimax thresholding.

THR: float threshold X-adapted value using selection rule defined by string TPTR

x = np.random.randn(1000) thr = thselect(x,'rigrsure')

`pyyawt.`**`upcoef`**`(`*aprox_or_detail*, *x*, **args*`)`

Direct Restruction

Y=upcoef(type,x,wname,[N],[L]) Y=upcoef(type,x,Lo_R,Hi_R,[N],[L])

type: approximation or detail, 'a' or 'd'. x: input vector wname: wavelet name, haar("haar"), daubechies ("db1" to "db36"), coiflets ("coif1" to "coif17"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8" X: reconstruction Lo_R: lowpass synthesis filter Hi_R: highpass syntheis filter N: restruction level L: desired output length

upcoef is for upward reconstruction from any desired input vector.

x=rand(1,100); [cA,cD]=dwt(x,'db2') Y=upcoef('a',cA,'db2',1) Z=upcoef('a',cA,'db2',3)

`pyyawt.`**`upcoef2`**`(`**args*`)`

`pyyawt.`**`upwlev`**`(`*C*, *L*, **args*`)`

Single Level Reconstruction from multiple level decompostion

[NC,NL,CA]=upwlev(c,l,wname) [NC,NL,CA]=upwlev(c,l,Lo_R,Hi_R) wname: wavelet name, haar(“haar”), daubechies (“db1” to “db36”), coiflets (“coif1” to “coif17”), symlets (“sym2” to “sym20”), legendre (“leg1” to “leg9”), bathlets(“bath4.0” to “bath4.15” and “bath6.0” to “bath6.15”), dmey (“dmey”), beyklin (“beylkin”), vaidyanathan (“vaidyanathan”), biorthogonal B-spline wavelets (“bior1.1” to “bior6.8”), “rbior1.1” to “rbior6.8” NC: upward level coefficent array NL: upward level length array CA: approximation coeffient at the last level Lo_R: lowpass synthesis filter Hi_R: highpass syntheis filter c: coefficent array l: length array

upwlev is for single level reconstruction.

x=rand(1,100) [C,L]=wavedec(x,3,’db2’) [NC,NL,CA3]=upwlev(C,L,’db2’)

pyyawt.**upwlev2**(**args*)

pyyawt.**wavedec**(*x*, *N*, **args*)

Multiple level 1-D discrete fast wavelet decomposition

[C,L]=wavedec(X,N,wname) [C,L]=wavedec(X,N,Lo_D,Hi_D)

wname : wavelet name, haar(“haar”), daubechies (“db1” to “db36”), coiflets (“coif1” to “coif17”), symlets (“sym2” to “sym20”), legendre (“leg1” to “leg9”), bathlets(“bath4.0” to “bath4.15” and “bath6.0” to “bath6.15”), dmey (“dmey”), beyklin (“beylkin”), vaidyanathan (“vaidyanathan”), biorthogonal B-spline wavelets (“bior1.1” to “bior6.8”), “rbior1.1” to “rbior6.8” X : signal vector N : decompostion level Lo_D : lowpass analysis filter Hi_D : highpass analysis filter C : coefficient vector L : length vector

wavedec can be used for multiple-level 1-D discrete fast wavelet decompostion using a specific wavelet name wname or wavelet decompostion filters Lo_D and Hi_D. Such filters can be generated using wfilters.

The global extension mode which can be change using dwtmode is used.

The coefficient vector C contains the approximation coefficient at level N and all detail coefficient from level 1 to N

The first entry of L is the length of the approximation coefficent, then the length of the detail coefficients are stored and the last value of L is the length of the signal vector.

The approximation coefficient can be extracted with C(1:L(1)). The detail coefficients can be obtained with C(L(1):sum(L(1:2))), C(sum(L(1:2)):sum(L(1:3))),.... until C(sum(L(1:length(L)-2)):sum(L(1:length(L)-1)))

X = wnoise(4,10,0.5); //doppler with N=1024 [C,L]=wavedec(X,3,’db2’)

pyyawt.**wavedec2**(**args*)

pyyawt.**wavefun**(**args*)

pyyawt.**wavefun2**(**args*)

pyyawt.**waveletfamilies**(**args*)

pyyawt.**waverec**(*C*, *L*, **args*)

Multiple level 1-D inverse discrete fast wavelet reconstruction

x0=waverec(C,L,wname) x0=waverec(C,L,Lo_R,Hi_R)

wname : wavelet name, haar(“haar”), daubechies (“db1” to “db36”), coiflets (“coif1” to “coif17”), symlets (“sym2” to “sym20”), legendre (“leg1” to “leg9”), bathlets(“bath4.0” to “bath4.15” and “bath6.0” to “bath6.15”), dmey (“dmey”), beyklin (“beylkin”), vaidyanathan (“vaidyanathan”), biorthogonal B-spline wavelets (“bior1.1” to “bior6.8”), “rbior1.1” to “rbior6.8” x0 : reconstructed vector Lo_R : lowpass synthesis filter Hi_R : highpass synthesis filter C : coefficent array L : length array

waverec can be used for multiple-level 1-D inverse discrete fast wavelet reconstruction.

waverec supports only orthogonal or biorthogonal wavelets.

X = wnoise(4,10,0.5); //doppler with N=1024 [C,L]=wavedec(X,3,’db2’); x0=waverec(C,L,’db2’); err = sum(abs(X-x0)) `pyyawt.`**`waverec2`**(**args*)

`pyyawt.`**`wcodemat`**(**args*)

`pyyawt.`**`wden`**(**args*)

wden performs an automatic de-noising process of a one-dimensional signal using wavelets. Calling Sequence ———————- [XD,CXD,LXD] = wden(X,TPTR,SORH,SCAL,N,wname) [XD,CXD,LXD] = wden(C,L,TPTR,SORH,SCAL,N,wname) Parameters ———- x: array_like

input vector

C: array_like coefficent array

L: array_like coefficent array

TPTR: str threshold selection rule 'rigrsure' uses the principle of Stein's Unbiased Risk. 'heursure' is an heuristic variant of the first option. 'sqtwolog' for universal threshold 'minimaxi' for minimax thresholding

SORH: str ('s' or 'h') soft or hard thresholding

SCAL: str 'one' for no rescaling 'sln' for rescaling using a single estimation of level noise based on first-level coefficients 'mln' for rescaling done using level-dependent estimation of level noise

N: int N: decompostion level

wname: str wavelet name

XD: array_like de-noised signal

CXD: array_like de-noised coefficent array

LXD: array_like de-noised length array

[xref,x] = wnoise(3,11,3) level = 4 xd = wden(x,'heursure','s','one',level,'sym8')

`pyyawt.`**`wenergy`**(*C*, *L*)

Energy Statistics from multiple level decompostion

[Ea,Ed]=wenergy(c,l)

Ea : energy percentage of approximation coefficent Ed : energy percentage of detail coefficent, vector c : coefficent array l : length array

wenergy is to calculate the energy percentage of approximation and detail coefficent.

x=rand(1,100) [C,L]=wavedec(x,3,'db2') [Ea,Ed]=wenergy(C,L)

`pyyawt.`**`wenergy2`**(**args*)

`pyyawt.`**`wextend`**(*dim*, *extMethod*, *x*, *L*, *typeString=None*)

signal extension

Y=wextend(onedim,extMode,x,L,[type]) Y=wextend(twodim,extMode,M,sizeVector,[typeStringVector]) Y=wextend(twodim,extMode,M,sizeVector,[typeString]) Y=wextend(twodim,extMode,M,L) Y=wextend(row_col,extMode,M,L,[type])

x : double vector M : double matrix L : length integer type : extraction manner, 'l' for left, 'r' for right, and 'b' for both left and right sizeVector : integer vector containing row and column size to extend typeString : string for extension, 'bb', 'll', 'rr', 'bl', 'lb', 'br', 'rb', 'lr', 'rl'. typeStringVector : string vector for extension, ['b' 'b'], ['l' 'l'], ['r' 'r'], ['b' 'l'], ['l' 'b'], ['b' 'r'], ['r' 'b'], ['r' 'l'], ['l' 'r']. extMode : extension method, 'symh'('sym'), 'symw', 'asymh', 'asymw', 'zpd', 'zpd', 'per', 'ppd'. row_col : adding row or adding column, 'ar' or 'addrow' for row, 'ac' or 'addcol' for column. onedim : one dimension indication, 1, '1', '1d' and '1D' twodim : two dimension indication, 2, '2', '2d' and '2D' Y : extension result wextend is an utility function for signal extension.

a=rand(1,100); Y=wextend(1,'symh',a,5,'b'); b=rand(25,25); Y=wextend(2,'symh',b,[3,5],'lb'); Y=wextend('ar','symh',b,3,'r');

`pyyawt.`**`wfilters`**(*wname*, *filterType=None*)

wfilters is an utility function for obtaining analysis and synthesis filter set.

[Lo_D,Hi_D,Lo_R,Hi_R]=wfilters(wname) [Lo_D,Hi_D]=wfilters(wname,'d') [Lo_R,Hi_R]=wfilters(wname,'r') [Lo_D,Lo_R]=wfilters(wname,'l') [Hi_D,Hi_R]=wfilters(wname,'h')

wname: str wavelet name, wavelet name, haar("haar"), daubechies ("db1" to "db20"), coiflets ("coif1" to "coif5"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8"

Lo_D: array_like lowpass analysis filter

Hi_D: array_like highpass analysis filter

Lo_R: array_like lowpass synthesis filter

Hi_R: array_like highpass synthesis filter

[lo_d,hi_d,lo_r,hi_r]=wfilters('db2')

`pyyawt.`**`wkeep`**(*x*, **args*)

signal extraction

Y=wkeep(x,L,[type]) Y=wkeep(x,L,[FIRST]) Y=wkeep(M,S,[indexVector])

x : double vector M : double matrix L : length integer type: extraction manner, 'l' for left, 'r' for right, and 'c' for center FIRST: index integer from which extraction starts. S : size integer vector containing row size and column size wanted indexVector : row and column index integer vector from which extraction starts. Y : extraction result

wkeep is an utility function for both vector and matrix extraction. For vector extraction, extractions will be aligned to the right, left or center based on optional argument type. So does matrix extraction.

a = np.linspace(1,8,8) X=np.dot(np.array([a]).T,np.array([a])) Y=wkeep(X,[4, 4])

`pyyawt.`**`wmaxlev`**(*signalLength*, *wname*)

wmaxlev is the maximum decompostion level calculation utility.

signalLength: int signal length

wname: str wavelet name, wavelet name, haar("haar"), daubechies ("db1" to "db20"), coiflets ("coif1" to "coif5"), symlets ("sym2" to "sym20"), legendre ("leg1" to "leg9"), bathlets("bath4.0" to "bath4.15" and "bath6.0" to "bath6.15"), dmey ("dmey"), beyklin ("beylkin"), vaidyanathan ("vaidyanathan"), biorthogonal B-spline wavelets ("bior1.1" to "bior6.8"), "rbior1.1" to "rbior6.8"

L: int decomposition level

L=wmaxlev(100,'db5')

`pyyawt.`**`wnoise`**(*FUN*, *N*, *SQRT_SNR=1*)

Noisy wavelet test data

FUN: str / int 1 or 'blocks' 2 or 'bumps' 3 or 'heavy sine' 4 or 'doppler' 5 or 'quadchirp' 6 or 'mishmash'

N: int vector length of X = 2^N

SQRT_SNR: float standard deviation of added noise

X: array_like test data

XN: array_like noisy test data (rand(1,N,'normal') is added!)

[x,noisyx] = wnoise(4,10,7);

`pyyawt.`**`wnoisest`**(*C*, *L=None*, *S=None*)

estimates of the detail coefficients' standard deviation for levels contained in the input vector S

C: array_like coefficent array

L: array_like coefficent array

S: array_like estimate noise for this decompostion levels

STDC: array_like STDC[k] is an estimate of the standard deviation of C[k]

[c,l] = wavedec(x,2,'db3') wnoisest(c,l,[0,1])

`pyyawt.`**`wnorm`**(**args*)

`pyyawt.`**`wrcoef`**(*approx_or_detail*, *C*, *L*, **args*)

Restruction from single branch from multiple level decomposition

X=wrcoef(type,C,L,wname,[N]) X=wrcoef(type,C,L,Lo_R,Hi_R,[N])

type : approximation or detail, 'a' or 'd'. wname : wavelet name X : vector of reconstructed coefficents Lo_R : lowpass synthesis filter Hi_R : highpass syntheis filter C : coefficent array L : length array N : restruction level with length(L)-2>=N

wrcoef is for reconstruction from single branch of multiple level decomposition from 1-D wavelet coefficients. Extension mode is stored as a global variable and could be changed with dwtmode. If N is omitted, maximum level (length(L)-2) is used.

The wavelet coefficents C and L can be generated using wavedec.

x=rand(1,100) [C,L]=wavedec(x,3,'db2') x0=wrcoef('a',C,L,'db2',2)

`pyyawt.`**`wrcoef2`**(**args*)

`pyyawt.`**`wrev`**(**args*)

`pyyawt.`**`wrev2`**(**args*)

`pyyawt.`**`wrev3`**(**args*)

`pyyawt.`**`wthresh`**(*X*, *SORH*, *T*)

doing either hard (if SORH = 'h') or soft (if SORH = 's') thresholding

X: array input data (vector or matrix)

SORH: str 's': soft thresholding 'h' : hard thresholding

T: float threshold value

Y: array_like output

y = np.linspace(-1,1,100) thr = 0.4 ythard = wthresh(y,'h',thr) ytsoft = wthresh(y,'s',thr)

Contents:

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

L

M

O

P

Q

R

S

T

U

V

W