Yet Another Wavelet Toolbox
Reference Guide

Version 0.1.1

The YAW Toolbox Team

November 18, 2003

Abstract

The aim of this toolbox is to provide a clear and well documented implementation in Matlab of some time-frequency and time-scale transformations like the well known continuous/discrete wavelet transforms.

In this guide each function of the Yet Another Wavelet Toolbox is documented.

Contents

1 History and contributors

The YAWTb project started on December 2000 few days after Laurent Jacques' decisive E-Mail. Alain Coron, Laurent Jacques, Attilio Rivoldini, and Pierre Vandergheynst make up the core of the team. Later Laurent Demanet joined the team.

The software is distributed under GNU General Public License. So your contributions are welcome. You may join the two following mailing lists:

- yawtb-devel@lists.sourceforge.net is devoted to the development. We discuss on features that we plan to add and how to code them.
- yawtb-code@lists.sourceforge.net is devoted to the users. They could send remarks, submit patches or new M-file.

2 How to contribute ?

Writing the Matlab headers

- First of all, you should use the template that you will find in `tools/devel/xemacs_macro/mfile_header_part.txt`.
- A `\YAWTB` command exist to denote the YAWTb toolbox
- Short description are better. We do not need to say the one function compute or return something. For example,
  - cwt1d: 1D continuous wavelet transform
  - gauss1d: 1D Gaussian wavelet in the frequency domain
- Some commands need and underscore _. In LaTeX the _ has a special meaning. Protect this character.
- Use `\libfun{function}` if you means the function `function`
- Use `\libvar{variable}` if you means the variable `variable`
- In ths Sections "Input Data" and "Output Data" we indicate the type of the variable. We already use the following types:

  - BOOLEAN : A real number with two possible values 0 or 1
  - COMPLEX MATRIX
  - CPLX a complex
  - CPLX MATRIX
  - CPLX VECTOR a vector of complexes
  - MATRIX : the matrix may be real or complex
  - MISC when the type depends on the context
  - REAL
  - REAL SCALAR
  - REAL MATRIX
  - REAL VECTOR
  - SCALAR : real or complex scalar
  - STRING
  - STRUCT a Matlabstructure
  - VECTOR
  - YAWTB OBJECT a structure that stores the output of functions like `cwt1d`

  Some are redundant. I guess that some should be considered as obsolete.

- The environments

  ```
  \begin{latexonly} ... \end{latexonly}
  ```

  or

  ```
  \begin{htmlonly} ... \end{htmlonly}
  ```

  enable you to include different comments in the printed and htmlize versions.

- Each function will get its own label during processing the doc. This label is build with the string `cmd:` followed by the name of the Matlab function. For example the label of the `cauchy2d` command is `cmd:cauchy2d`. Even the subsections of each command have their own labels. So you can reference the functions with the command
  `htmlref`. This command has two mandatory argument, the text that will appeak as a link and the reference of the link.

- Fill in the "See Also" sections with Perl regular expression separated by spaces. The regular expression should match the name of files (without the Matlab extension) in the tree of the toolbox on a Unix machine. Those regular will be expanded accordingly by a Perl script.

  For example, if you wish to add the function cwt2d, you may add `^cwt2d$` or `^continuous/.*/cwt2d$`. If you wish to add all the functions under the continuous directory, type `^continuous/.*`.

  If the regular expression does not match any name in the YAWTB tree, then the leading ^ (if any) and the terminated $ (if any) of the expression are removed. The remaining expression is added to the list of files. So you may add the `fft` command by typing `^fft$`.

TODO

I list a lot of possible improvements:

- Parse the Matlab header files and build a tree to be able to perform easy transformation. Having a tree should help at applying different type of transformation (depending on the context) on the different section (Example section, See Also section). But building a tree may require a lot of memory.

- Fill in the Section "Writing the Matlab headers"
- Code example Section: We should be able to extract the example code from the documentation, run separately those pieces of code and include the textual output as well as the figures in the final documentation. There should be a line-numbering system of the code and an automatic labelling of the code and the figure.

  How could we implement this?

  - Maybe with a code environement like this:

    ```
    \begin{code}[number=<number_of_figures>,
                 label=<label>]
    \end{code}
    ```

  - I suggest to internally construct the label of the code with `cmd:<cmd_name>:code:<label>`
  - The label of the associated figures could be `cmd:<cmd_name>:code:<label>:fig<fig_number>`
  - The Matlab command line input/output `<cmd_name>:code:<label>:Matlab.txt`. Which directory?
  - The figures could be saved under the name `<cmd_name>:code:<label>:fig<fig_number>.<extension>`. Which directory?

3 The commands by category

Contiuous Wavelet Transform

1D

Command	Description
cgt1d	1D continuous Gabor transform
cwt1d	1D continuous wavelet transform
dgauss1d	Compute the n order derivative of Gaussian in frequency
gauss1d	1D Gaussian window in the frequency domain
morlet1d	1D Morlet wavelet in the frequency domain
sdog1d	1D Scaling Difference Of Gaussian wavelet in the frequency domain
yashow_cgt1d	Display the result of `cgt1d`. Automatically called by `yashow`!
yashow_cwt1d	Display the result of `cwt1d`. Automatically called by `yashow`!

1D+T

Command	Description
cwt1dt	1D+T continuous wavelet transform
mexican1dt	1D+T Mexican Wavelet in wave-number/frequency domain
morlet1dt	1D+T Morlet Wavelet in wave-number/frequency domain
yashow_cwt1dt	Display the result of cwt1dt. Automatically called by `yashow`!

2D

Command	Description
cauchy2d	Compute the 2D Cauchy Wavelet in frequency plane
cwt2d	Compute several 2D continuous wavelet transforms
dergauss2d	Compute the 2D multiple derivative of Gaussian
dog2d	Compute the 2D DoG Wavelet in frequency plane
endstop1	Compute the single 2D EndStop Wavelet in frequency plane
endstop2	Compute the double 2D EndStop Wavelet in frequency plane
es2cauchy2d	Compute an second endstop wavelet from the 2D Cauchy Wavelet in frequency plane
es2morlet2d	Compute the 2D Morlet Wavelet in frequency plane
esmex2d	Compute the endstop 2D Mexican Wavelet in frequency plane
gabor2d	Compute the 2D Gabor Wavelet in frequency plane
gauss2d	Compute the bidimensionnal Gaussian
gaussx2d	Compute the 2D x order derivative of Gaussian
gaussz2d	Compute the 2D z order derivative of Gaussian
isdog2d	Compute the Inverse 2D Scaling difference of Gaussian Wavelet
isomor2d	Compute the isotropical Morlet Wavelet in frequency plane
mexican2d	Compute the 2D Mexican Wavelet in frequency plane
mexican2d_ctr	Compute the 2D Mexican Wavelet in frequency plane
morlet2d	Compute the 2D Morlet Wavelet in frequency plane
pethat2d	Compute the 2D Pet Hat wavelet in frequency plane
rmorlet2d	Compute the Real 2D Morlet Wavelet in frequency plane
samcwt2d	Compute the 2D CWT scale-angle measure

sarcwt2d	Compute the CWT scale (angle) representation of mat
	
	`Implement this program without cwt2d, directly with the`
	`scalar product of mat and the wavelet.`
	`This will of course decrease the computing time!`

sdog2d	Compute the 2D Scaling difference of Gaussian Wavelet
sqdog2d	Compute the square of 2D Scaling difference of Gaussian Wavelet
wheel2d	Compute the 2D Wheel wavelet in frequency plane
yashow_cwt2d	Display the result of cwt2d. Automatically called by yashow!
yashow_samcwt2d	Display the samcwt2d results (internal use)

3D

Command	Description
cwt3d	Compute several 3D continuous wavelet transforms
mexican3d	Compute the 3D Mexican Wavelet in frequency plane
morlet3d	Compute the 3D Morlet Wavelet in frequency plane
yashow_cwt3d	Display the result of cwt3d. Automatically called by yashow!

Sphere

Command	Description
cwtsph	Compute the spherical wavelet transform
dogsph	Spherical Difference Of Gaussians (DOG) wavelet
fcwtsph	Compute the fast spherical wavelet transform
morletsph	Spherical Morlet wavelet.
yashow_cwtsph	Display the result of cwtsph. Automatically called by yashow!

Discrete Wavelet Transform

Frames 2D

Command	Description
fwt2d	2D framed wavelet transform
fwt2d_allwav	
fwt2d_app	
fwt2d_app_filter	
fwt2d_dapp_filter	
fwt2d_dhigh_filter	
fwt2d_dwav_filter	
fwt2d_high	
fwt2d_high_filter	
fwt2d_init	
fwt2d_thresh	
fwt2d_wav	
fwt2d_wav_filter	
ifwt2d	2D inverse framed wavelet transform
ifwt2d_app	
ifwt2d_high	
ifwt2d_wav	

Spherical Frames

Command	Description
fwtsph	Framed Wavelet Transforn on the sphere
ifwtsph	Inverse Framed Wavelet Transform approximation on the sphere
sphcg	Spherical Conjugate Gradient Reconstruction
sphrichardson	

Packet

2d

Command	Description
iwpck2d	Compute the reconstruction of the wavelet packets transform
phin	The scaling function associated to the pseudo-diff operator
pmexican2d	Return the 2D mexican wavelet or scaling function packets
pseudiff	Return the 2D pseudo differential operator k^n wavelet
wpck2d	Compute the packet wavelet transform (details and approximation)
yashow_wpck2d	Display the output of cwt2d. Automatically called by yashow!

Interfaces

SpharmonicKit

Command	Description
convsph	Fast spherical convolution
convsph_semi	Fast spherical convolution for bandwidth ¡ 64
fst	Fast spherical harmonic transform
fzt	Fast spherical harmonic transform
ifst	Fast spherical harmonic transform
ilmshape	Reshape the output of fst in a l-m-readable matrix
lmshape	Reshape the output of fst in a l-m-readable matrix

Sample

Command	Description
cube	Generate a 3D sample cube.
linearchirp	Return a complex linear chirp and its theoretical instantaneous frequency
movgauss	Create a three moving Gaussians on the line.
the3dL	Generate a the 3D version of the L shape.
theL	A 2D academic example: the letter L in a NxN matrix
thedisk	Create a disk in a square image.
yachirp	Compute a typical chirp cosine

Help, help !

Command	Description
yademo	Execute the demo associates to a yawtb file
yahelp	Display the help associated to a yawtb file.

Tools

Command	Description
fftsph	Spherical Harmonic Transform
getopts	Return the value of the 'OptionName' variable inside list 'OptionList'.
getyawtbprefs	Get the YAWtb preferences
ifftsph	Inverse Spherical Harmonic Transform
inputeof	Work like input(msg,'s') but ends by a EOF
list_elem	Return the k-th element of rec if it exists.
normopts	Reject to the end of list the options charaterized by a string.
pgmread	Read an image in the raw PGM format (8bits).
pgmwrite	Write an image in the raw PGM format.
regrep	Perform a substring replacement on string using regular expression.
rgray	This function return the reverse gray colormap
setyawtbprefs	Set a YAWtb preference
sphgrid	Spherical grid
sphweight	Clenshaw-Curtis weights for spherical quadrature on grid (Odd x Even)
vect	Compute vector of several shape (linear,exponential, ...)
whatin	Return information about a matrix
yabeta	Profile function beta
yadiro	Dilation and rotation on grids of positions
yahist	Computes 1D and 2D histograms.
yaload	Load the YAWtb path into memory
yamake	Create all the mex files needed by yawtb
yamax	Determines the regional maxima of a real matrix
yamse	Mean square error between signals or images
yapbar	This function create a figure with a progress bar.
yapsnr	Peak Signal to Noise Ratio of signals or images
yapuls	Pulsation vector
yapuls2	Pulsation matrix
yasave	save and ask for informations to include in data
yashow	Display the result of any transform defined in YAWTB
yashow_matrix	display a matrix
yashow_spheric	Display a matrix onto a sphere
yashow_timeseq	Display a time sequence
yashow_volume	Display a volume
yasnr	Signal to Noise Ratio of signals or images
yaspharm	Generate a spherical harmonic on a regular spherical grid
yaspline	Yet another cardinal B-spline computation
yasthresh	Softly threshold a matrix.
yastrfind	Find one string within another
yathresh	Hardly threshold a matrix.
yawopts	Return the options to give to a yawtb mfile (internal use)
yawtbprefs	Set all the preferences of YAWtb

4 The commands in alphabetical order

aspline2d_app

Syntax

[] = aspline2d_app()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/frame_defs/aspline2d_app.m

aspline2d_high

Syntax

[] = aspline2d_app()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/frame_defs/aspline2d_high.m

aspline2d_info

Syntax

[] = aspline2d_info()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/frame_defs/aspline2d_info.m

aspline2d_wav

2D Angular Spline Wavelet

Syntax

[] = aspline_app()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/frame_defs/aspline2d_wav.m

box2d_app

Approximation function of the "Box" frame

Syntax

out = box2d_app(kx, ky)

Description

Compute the approximation function of the "Box" frame, namely the isotropic function which takes 1 if $(kx^2 + ky^2) < pi/2$, and 0 elsewhere.

Input Data

kx [MATRIX]: the horizontal frequencies;

ky [MATRIX]: the vertical frequencies;

Output Data

out [MATRIX]: the computed function on the frequency plane.

Location

discrete/frames/2d/frame_defs/box2d_app.m

box2d_wav

Wavelet of the "Box" frame.

Syntax

out = box2d_wav(kx, ky, nang)

Description

Compute the wavelet of the "Box" frame, namely the angular sector which take 1 if $pi/2 <= abs(K) < pi$, $-pi/nang < arg(K) <= pi/nang$, with $K = (kx, ky)$, and 0 elsewhere.

Input Data

kx [MATRIX]: the horizontal frequencies;

ky [MATRIX]: the vertical frequencies;

nang [INTEGER]: the number of angular sectors on which the frame decomposition is computed.

Output Data

out [MATRIX]: the computed function on the frequency plane.

Example(s)

>>

References

See Also

Location

discrete/frames/2d/frame_defs/box2d_wav.m

cauchy2d

Compute the 2D Cauchy Wavelet in frequency plane

Syntax

[out] = cauchy2d(kx,ky,apert,sigma,l,m)

Description

This function computes the 2D Cauchy wavelet in frequency plane. That is, the wavelet given by:

```
                  -
                 |(E(apert-pi/2) . K)^l
                 |           * (E(-apert+pi/2) . K)^m
                 |           * exp( - 0.5 * sigma * |K-K_0|^2 )
PSIHAT(kx,ky) = <                      INSIDE C(-apert,apert)
                 |
                 |0                    OUTSIDE C
                 `-
where: E(alpha) = (cos(alpha), sin(alpha))
       K        = (kx, ky)
       K_0      = (l+m)^0.5 * (sigma - 1)/sigma * (1,0)
       C(-apert,apert) = the cone supported by E(apert)
                         and E(-apert)
```

The wavelet parameters are thus:

- apert : the half aperture of the cone;
- sigma : the frequency spread of the wavelet;
- l,m : its vanishing moments.

This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

apert [REAL SCALAR]: The aperture of the cone (in gradiant)

sigma [REAL SCALAR]: The frequency spread of the wavelet

l,m [INTEGERS]: The vanishing moments

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = cauchy2d(kx,ky,pi/6,1,4,4);
>> imagesc(wav);
```

References

See Also

cauchy2d, cwt2d, dergauss2d, dog2d, es2cauchy2d, es2morlet2d, esmex2d, gabor2d, gauss2d, gaussx2d, gaussz2d, isdog2d, isomor2d, meshgrid, mexican2d, morlet2d, pethat2d, rmorlet2d, samcwt2d, sarcwt2d, sdog2d, sqdog2d, wheel2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/cauchy2d.m

cauchy2d_app

The approximation function of the Cauchy frame.

Syntax

out = cauchy2d_app(kx, ky)

Description

Compute the approximation function of the Cauchy frame, that is a Gaussian centered on the frequency origin and of spread equals to pi/2.

Input Data

kx [MATRIX]: the horizontal frequencies;

ky [MATRIX]: the vertical frequencies;

Output Data

out [MATRIX]: the computed function on the frequency plane.

See Also

Location

discrete/frames/2d/frame_defs/cauchy2d_app.m

cauchy2d_wav

The wavelet of the Cauchy frame.

Syntax

out = cauchy2d_wav(kx, ky, nang, sigma, l, m)

Description

Compute the wavelet of the Cauchy frame according the Cauchy wavalet given by cauchy2d().

Input Data

kx [MATRIX]: the horizontal frequencies;

ky [MATRIX]: the vertical frequencies;

nang [INTEGER]: the number of angular sectors on which the frame decomposition is computed.

sigma [REAL]: the radial spread of the wavelet.

l [INTEGER]: the moment of the wavelet on the edges of its cone.

Output Data

out [MATRIX]: the computed function on the frequency plane.

Example(s)

>>

References

See Also

cauchy2d, cauchy2d_app, cauchy2d_app, cauchy2d_wav, es2cauchy2d

Location

discrete/frames/2d/frame_defs/cauchy2d_wav.m

cgt1d

1D continuous Gabor transform

Syntax

out = cgt1d(fsig, freqs [,'Sigma', sigma] ... [, 'Window', winname ... [,WindowParameter]])

out = cgt1d(fsig, freqs [,'Sigma', sigma] ... [, 'Window', winname ... ['WindowOptionName', WindowOptionValue]])

Description

`cgt1d` computes the Continuous Gabor Transform (or Windowed Fourier Transform) of a signal. The window used is the Gaussian window but this can be modified with the option 'Window' followed by the name of the desired window. This one must correspond to a mfile defined in the subdirectory 'win_defs'.

Input Data

fsig [CPLX VECTOR]: the Fourier transform of the input signal;

freqs [REAL VECTOR]: the frequencies of the transform;

sigma [REAL]: the size of the window;

winame [STRING]: the name of the window to use if the gaussian is not wanted.

WindowParameter [MISC]: one window parameter. Its type depends on the window. Refer to the corresponding M-file (inside 'win_defs') for a description of the available parameters;

WindowOptionName, WindowOptionValue [STRING, MISC]: Another way of writing window parameters. The window parameter name (a string) is followed by its value. See the corresponding window M-file (inside 'win_defs') for a description of the parameter or just type the name of the window with [] as argument on the Matlab command line;

Output Data

out [STRUCT]: the output of the transform. It is a structure with the following fields:

- `out.data` [MATRIX] : the Gabor coefficients
- `out.type` (the type of the gabor transform)
- `out.win` [STRING] : the window name
- `out.para` [STRUCT] : the extra parameters given to cgt1d
- `out.freqs` [VECTOR]: the frequencies
- `out.fsig` [VECTOR]: the Fourier transform of the input signal

Example(s)

```
>> load superpos
>> fsig = fft(sig);
>> freqs = 0.005: (0.12 - 0.005)/127: 0.12;
>> wsig = cgt1d(fsig, freqs, 'sigma', 50);
>> yashow(wsig);
```

References

See Also

cgt1d, cwt1d, gauss1d, yashow

Location

continuous/1d/cgt1d.m

convsph

Fast spherical convolution

Syntax

out = convsph(data, filter)

Description

This mex file computes a fast spherical convolution of spherical data `data` with the zonal filter `filter` using the classical spherical convolution theorem

$\hat{c}(l,m) = \frac{4\pi}{2l+1}\hat{f}(l,m)\hat{h}(l,0)$

with $\hat{c}$ the spherical transform of $c = f \star h$, $\hat{f}$ and $\hat{h}$ the spherical transform respectively of f (the data) and h (the zonal filter).

To perform this spherical convolution, `convsph` program uses the C functions of the (GPL) SpharmonicKit [1] based on the work of Driscoll, Healy and Rockmore [2] about fast spherical transforms.

Input Data

data [REAL MATRIX]: The spherical data described on a equi-angular (2B*2B) spherical grid (θ_i, ϕ_j) with $\phi_j = j\frac{2\pi}{2B}$ $(j = 0..2B-1)$ and $\theta_i = (2i+1)\frac{\pi}{2B}$ $(i = 0..2B-1)$. Notice that B must be a power of 2.

filter [REAL MATRIX]: the zonal filter, that is invariant under rotations around the North pole, defined on the same 2B*2B spherical equi-angular grid.

Output Data

out [REAL MATRIX]: the convolution defined on the same spherical equi-angular grid

Example(s)

```
>> load world2; figure; yashow(mat, 'spheric');
>> [phi,theta] = sphgrid(size(mat,1));
>> filter = exp(-(tan(theta/2)/0.05).^2);
>> out = convsph(double(mat), filter);
>> figure; yashow(out, 'spheric', 'mode','real'); colorbar;
```

References

[1] SpharmonicKit: http://www.cs.dartmouth.edu/ geelong/sphere/. Developed by Sean Moore, Dennis Healy, Dan Rockmore, Peter Kostelec.

[2] D. Healy Jr., D. Rockmore, P. Kostelec and S. Moore, "FFTs for the 2-Sphere - Improvements and Variations", Journal of Fourier Analysis and Applications, 9:4 (2003), pp. 341 - 385.

See Also

fst, fzt, ifst, ifst, ilmshape, ilmshape, lmshape

Location

interfaces/spharmonickit/convsph.m

convsph_semi

Fast spherical convolution for bandwidth ¡ 64

Syntax

out = convsph_semi(data, filter)

Description

This mex file computes a fast spherical convolution of spherical data `data` of bandwith ¡ 64 (i.e. on grid of size lesser than 128) with the zonal filter `filter` using the classical spherical convolution theorem

$\hat{c}(l,m) = \frac{4\pi}{2l+1}\hat{f}(l,m)\hat{h}(l,0)$

with $\hat{c}$ the spherical transform of $c = f \star h$, $\hat{f}$ and $\hat{h}$ the spherical transform respectively of f (the data) and h (the zonal filter).

To perform this spherical convolution, `convsph_semi` program uses the C functions of the (GPL) SpharmonicKit [1] based on the work of Driscoll, Healy and Rockmore [2] about fast spherical transforms.

Input Data

data [REAL MATRIX]: The spherical data described on a equi-angular (2B*2B) spherical grid (θ_i, ϕ_j) with $\phi_j = j\frac{2\pi}{2B}$ $(j = 0..2B-1)$ and $\theta_i = (2i+1)\frac{\pi}{2B}$ $(i = 0..2B-1)$. Notice that B must be a power of 2.

filter [REAL MATRIX]: the zonal filter, that is invariant under rotations around the North pole, defined on the same 2B*2B spherical equi-angular grid.

Output Data

out [REAL MATRIX]: the convolution defined on the same spherical equi-angular grid

Example(s)

```
>> load world2; mat=mat(1:4:end,1:4:end);
>> figure; yashow(mat, 'spheric');
>> [phi,theta] = sphgrid(size(mat,1));
>> filter = exp(-(tan(theta/2)/0.05).^2);
>> out = convsph\_semi(double(mat), filter);
>> figure; yashow(out, 'spheric', 'mode','real'); colorbar;
```

References

[1] SpharmonicKit: http://www.cs.dartmouth.edu/ geelong/sphere/. Developed by Sean Moore, Dennis Healy, Dan Rockmore, Peter Kostelec.

[2] D. Healy Jr., D. Rockmore, P. Kostelec and S. Moore, "FFTs for the 2-Sphere - Improvements and Variations", Journal of Fourier Analysis and Applications, 9:4 (2003), pp. 341 - 385.

See Also

fst, fzt, ifst, ifst, ilmshape, ilmshape, lmshape

Location

interfaces/spharmonickit/convsph_semi.m

cube

Generate a 3D sample cube.

Syntax

[out] = cube(n [,ncube])

Description

Input Data

n [INTEGER]: The size of the cubic domain where the cube is defined.

ncube [INTEGER]: the size of the cube set to n/2 by default.

Output Data

out [ARRAY]: the volume describing the cube (1 inside & 0 outside)

Example(s)

```
>> cub = cube(64);
>> yashow(cub);
```

References

See Also

Location

sample/3d/cube.m

cwt1d

1D continuous wavelet transform

Syntax

out = cwt1d(fsig, wavname, scales [,WaveletParameter]... [,'Norm',NormValue] [,'export'])

out = cwt1d(fsig, wavname, scales [,'WaveletOptionName', WaveletOptionValue]... [,'Norm',NormValue] [,'export'])

Description

`cwt1d` computes and returns the 1D continuous wavelet transform of a signal. The wavelet may be chosen among the ones defined in the subdirectory 'wave_defs' (see the 'README' file to write your own wavelet).

Input Data

fsig [CPLX VECTOR]: the Fourier transform of the input signal;

wavname [STRING]: the name of the wavelet;

scales [REAL VECTOR]: the scales of the transform;

WaveletParameter [MISC]: one wavelet parameter. Its type depends on the wavelet. Refer to the corresponding M-file (inside 'wave_defs') for a description of the available parameters;

WaveletOptionName, WaveletOptionValue [STRING, MISC]: Another way of writing wavelet parameters. The wavelet parameter name (a string) is followed by its value. See the corresponding wavelet M-file (inside 'wave_defs') for a description of the parameter, or just type the name of the wavelet with '[]' as argument on the Matlab command line;

NormValue ['l1'—'l2']: normalization of the wavelet transform, namely L^1 or L^2. The default is L^2.

'export' [BOOL]: if mentioned, this options force cwt1d to return only the wavelet coefficients directly in out. These are normally present in out.data

Output Data

out [MATRIX—STRUCT]: If 'export' option is not set, the output of the transform. It is a structure with the following fields:

- `out.data` [MATRIX] : wavelet coefficients
- `out.type` [STRING] : transform type ('cwt1d')
- `out.wav` [STRING] : name of the wavelet
- `out.para` [MATRIX] : extra parameters given to cwt1d
- `out.sc` [VECTOR]: scales of the wavelet transform
- `out.fsig` [VECTOR]: Fourier transform of the input signal

If 'export' is set, `out` is a matrix corresponding a `out.data`.

Example(s)

```
>> t    = 1:1024;
>> sig  = sin(2*pi*t/30./(1+t/1000));
>> fsig = fft(sig);
>> s = 20:140;
>> wsig = cwt1d(fsig, 'morlet', s);
>> yashow(wsig);
```

returns the 1D Morlet wavelet transform of a chirp signal for scales between 20 and 140. The implicit values are the Morlet wavelet parameters: `k0`=6; `sigma`=1. For other values, type something like

```
>> wsig  = cwt1d(fsig, 'morlet', s, 7, 2);
```

or,

```
>> wsig  = cwt1d(fsig, 'morlet', s, 'k_0', 7 , 'sigma', 2);
```

It changes the values of `k0` and `sigma` respectively to 7 and 2. Note that the first example is order dependant and not the second.

Finally, we change to a L^1 normalization of the cwt with the following command:

```
>> wsig  = cwt1d(fsig, 'morlet', s, 'k_0', 7 , 'sigma', 2, ...
                 'norm','l1');
```

References

See Also

cgt1d, cwt1d, dgauss1d, morlet1d, sdog1d, yashow

Location

continuous/1d/cwt1d.m

cwt1dt

1D+T continuous wavelet transform

Syntax

out = cwt1dt(fsig, wavname, scales, velos [,WaveletParameter] ... [,'Time',TimeValue])
out = cwt1dt(fsig, wavname, scales, velos [,'WaveletOptionName', ... WaveletOptionValue][,'Time',TimeValue])

Description

`cwt1dt` computes and returns the 1D + T continuous wavelet transform of a 1D+T signal. The wavelet may be chosen among the one defined in subdirectory 'wave_defs' (see the README to write your own wavelet)

Input Data

fsig [CPLX MATRIX]: the 2D Fourier transform of the 1D+T signal `sig` computed with fft2.

Be Careful: The time index of `sig` varies along the first dimension (columnwise/vertically), and spatial index is along the second dimension (rowwise/horizontally).

wavname [STRING]: the name of the wavelet

scales, velos [VECTOR]: the scales and the velocities of the transform

WaveletParameter [MISC]: one wavelet parameter. Its type depends on the wavelet. Refer to the corresponding M-file (inside 'wave_defs') for a description of the available parameters;

WaveletOptionName, WaveletOptionValue [STRING, MISC]: Another way of writing wavelet parameters. The wavelet parameter name (a string) is followed by its value. See the corresponding wavelet M-file (inside wave_defs) for a description of the parameter, or just type the name of the wavelet with '[]' as argument on the Matlab command line.

TimeValue [INTEGER]: The wavelet transform is only computed for this Time i.e. frame number

Output Data

out [STRUCT]: the output of the transform. A structured data with the following fields:

- `out.data` [3D or 4D MATRIX]: wavelet coefficients
- `out.type` [STRING] : transform type ('cwt1dt')
- `out.wav` [STRING] : name of the wavelet
- `out.para` [MATRIX] : extra parameters given to `cwt1dt`
- `out.sc` [REAL VECTOR] : scales of the wavelet transform
- `out.vel` [REAL VECTOR] : velocity
- `out.ftime` [VECTOR] : this field exists only if the optional argument 'Time' is specified. Then it is equal to the `TimeValue`th frame of the signal in the time domain.

Example(s)

```
%% load the 3 moving Gaussians sample.
>> mat = movgauss;

%% Computes the CWT1DT of this signal.
>> wav = cwt1dt(fft2(mat),'morlet',4,vect(-2,2,128),'Time',10);

%% Display at time 10 the velocity-position representation of the CWT.
>> yashow(wav);
```

References

[1]: "Spatio-Temporal Wavelet Transform for Motion Tracking", J.-P. Leduc, F. Mujica, R. Murenzi, and M. J. T. Smith, presented in ICASP'97, Munich, Germany, Apr. 1997

See Also

cwt1dt, mexican1dt, morlet1dt, movgauss, yashow, yashow_cwt1dt

Location

continuous/1dt/cwt1dt.m

cwt2d

Compute several 2D continuous wavelet transforms

Syntax

out = cwt2d(fimg, wavname, scales, angles [,WaveletParameter] [,'Norm',NormValue] [,'Export'] [,'Contrast'] [,'Pos', pos [,'fig',selfig]] [,'Exec', exec] [,'NoPBar'])

out = cwt2d(fimg, wavname, scales, angles [,'WaveletOptionName', WaveletOptionValue] [,'Norm',NormValue] [,'Export'] [,'Contrast'] ['Pos', pos [,'fig',selfig]] [,'Exec', exec] [,'NoPBar'])

Description

This function computes the 2d continuous wavelet transform of an image. Wavelets are taken inside the sub directory 'wave_defs' (see the README to know how to write your own wavelet)

Input Data

fimg [CPLX MATRIX]: the fourier transform of the image;

wavname [STRING]: the name of the wavelet to use;

scales, angles [REALS]: contains the scales and the angles of the transform. The 'angles' parameter is needed but not use in the case of istropic wavelet.

WaveletParameter [MISC]: a wavelet parameter. His type depend of the wavelet used. See the corresponding wavelet mfile (inside wave_defs) for the correct parameters.

WaveletOptionName, WaveletOptionValue [STRING, MISC]: Another way of writting wavelet parameters. The wavelet parameter name (a string) is followed by its value. See the corresponding wavelet mfile (inside wave_defs) for the parameter to enter.

NormValue ['l0'—'l1'—'l2']: is a string which describes the normalization of the wavelet transform, namely the 'L0', 'L1' or 'L2' normalization. The 'L2' is taken by default.

'Export' [BOOLEAN]: tell to cwt2d if the output must be just a matrix. If the keyword 'Export' is missing, the output is a yawtb object.

'Contrast' [BOOLEAN]: if implemented, normalized the CWT by the convolution of the image with a kernel of the same geometry than the wavelet. The mfile implementing this kernel has a name of the form <wavname>_ctr.m. See [1] for explanations.

pos [ARRAY 2x1—'inter']: gives the position vector b (in (x,y)==(j,i) format) where to compute the CWT. You can also select it interactively on the figure by entering the keyword 'inter' instead of the position vector;

selfig [INTEGER]: In the interactive 'Pos' mode, you can specify the number of the figure where to select the position with selfig.

exec [STRING]: execute a command specified by the string 'exec' on each result of the cwt2d transform of each iteration of the scale-angle loop.

'out.data' is then a cell array of size length(scales)*length(angles) which contains the result of the application of 'exec' on each CWT of fimg.

Syntax of 'exec': '<iter>' or '<init>;<iter>'

The '<init>' string contains some commands which have to be executed before the loop.

The '<iter>' string is the command to execute at each iteration.

Special keywords of '<iter>':

- $cwt can be used to specified the current CWT coefficients;
- $last the preceeding stored result (you have to initialize $rec to use it);

Special keywords of '<iter>' and '<init>':

- $rec represent the cell array where each computation is stored;
- $fimg represent the FFT of the image.

NoPBar [BOOLEAN]: Disable the loopbar in case of several scales and angles.

Output Data

out [STRUCT]: the output of the transform. If the keyword 'Export' is not present, out is a structured data with the following fields:

- "out.data" (the resulting matrix),
- "out.type" (the type of the transform),

else, out is just a matrix containing the CWT coefficients.

Example(s)

```
>> [x,y] = meshgrid(-64:64);
>> img   = max( abs(x), abs(y) ) < 30;
>> fimg  = fft2(img);
>> wimg  = cwt2d(fimg, 'morlet', 2, 0);
>> yashow(wimg);
```

Give the 2D Morlet wavelet transform of a 64 pixel width square for a scale equal to 2 and angle equal to 0. The implicit values are the Morlet wavelet parameters: w0=6; sigma=1. For other values, you can type something like

```
>> wimg  = cwt2d(fimg, 'morlet', 2, 0, 7, 2);
```

or,

```
>> wimg  = cwt2d(fimg, 'morlet', 2, 0, 'w0', 7 , 'sigma', 2);
```

This change values of w0 and sigma respectively to 7 and 2. Note that the first example is order dependant and not the second.

Finally, you can change the normalization of the cwt with the following command:

```
>> wimg  = cwt2d(fimg, 'morlet', 2, 0, 'w0', 7 , 'sigma', 2, ...
                 'norm','l1');
```

for the L1 normalization (and 'l2' for L2).

References

[1]: M. Duval-Destin, M.A. Muschietti and B. Torresani, Continous wavelet decompositions, multiresolution and contrast analysis" SIAM J. Math Anal. 24 (1993).

See Also

cauchy2d, cwt2d, dergauss2d, dog2d, es2cauchy2d, es2morlet2d, esmex2d, gabor2d, gauss2d, gaussx2d, gaussz2d, isdog2d, isomor2d, mexican2d, morlet2d, pethat2d, rmorlet2d, samcwt2d, sarcwt2d, sdog2d, sqdog2d, wheel2d, yashow, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/cwt2d.m

cwt3d

Compute several 3D continuous wavelet transforms

Syntax

out = cwt3d(fvol, wavname, scale, angles, [,WaveletParameter] [,'Norm',NormValue])
out = cwt3d(fvol, wavname, scale, angles, [,'WaveletOptionName', WaveletOptionValue] [,'Norm',NormValue])

Description

This function computes the 3d continuous wavelet transform of an image. Wavelets are taken inside the sub directory 'wave_defs' (see the README to know how to write your own wavelet)

Input Data

fvol [CPLX MATRIX]: the fourier transform of the image;

wavname [STRING]: the name of the wavelet to use;

scale [REALS]: contains the scale of the transform.

angles [2 REALS]: a 1x2 vector containing the two angles of the cwt, that is tha angles (theta,phi).

WaveletParameter [MISC]: a wavelet parameter. His type depend of the wavelet used. See the corresponding wavelet mfile (inside wave_defs) for the correct parameters.

WaveletOptionName, WaveletOptionValue [STRING, MISC]: Another way of writting wavelet parameters. The wavelet parameter name (a string) is followed by its value. See the corresponding wavelet mfile (inside wave_defs) for the parameter to enter.

NormValue ['l1'—'l2']: is a string which describes the normalization of the wavelet transform, namely the 'L1' or 'L2' normalization. The 'L2' is taken by default.

Output Data

out [STRUCT]: the output of the transform. A structured data with the following fields:

- "out.data" (the resulting matrix)
- "out.type" (the type of the transform)

Example(s)

```
>> vol  = cube(64);
>> yashow(vol,'fig',1),
>> fvol = fftn(vol);
>> wvol = cwt3d(fvol, 'mexican', 2, [0 0]);
>> yashow(wvol,'fig',2);
```

This code gives the 3D Mexican Hat wavelet transform of a 64 pixel width cube for a scale equal to 2. The angles here are useless because of the MH isotropicity. The implicit values are the Mexcian wavelet parameters: sigma=1. For other values, you can type something like

```
>> wvol  = cwt3d(fvol, 'mexican', 2, [0 0], 2);
```

or,

```
>> wvol  = cwt3d(fvol, 'mexican', 2, [0 0], 'sigma', 2);
```

This change values of sigma respectively to 2. Note that the first example is order dependant and not the second.

Notice that you can change the normalization of the cwt with the following command:

```
>> wvol  = cwt3d(fvol, 'mexican', 2, [0 0], 'sigma', 2, 'norm','l1');
```

for the L1 normalization (and 'l2' for L2).

Finally, you can visualize several transparent layers of the result by specifying the number of levels to show:

```
>> yashow(wvol,'levels',3)
```

References

See Also

cwt3d, mexican3d, morlet3d, yashow, yashow_cwt3d

Location

continuous/3d/cwt3d.m

cwtsph

Compute the spherical wavelet transform

Syntax

out = wavspheric(img, wavname, scales [,WaveletParameter])
out = wavspheric(img, wavname, scales [,'WaveletOptionName', WaveletOptionValue])

Description

This function performs a spherical wavelet transform on a sphere in spherical coordinates. The algorithm is based on the use of FFT according phi coordinates (phi and theta are of course defined on a cartezian grid). The computation time is of order $N^3 \log(N)$ if mat is an $N \times N$ matrix. Wavelets are taken inside the sub directory 'wave_defs' (see the README to know how to write your own wavelet).

Input Data

mat [DOUBLE MATRIX] : the input matrix in spherical coordinates.

wavname [STRING]: the name of the wavelet to use (see yawtb/continuous/sphere/wave_defs for existing wavelet);

scales [DOUBLE VECTOR] : the interval of scales where you want to compute a spherical CWT.

angles [DOUBLE VECTOR] : the interval of angles where you want to compute a spherical CWT.

WaveletParameter [MISC]: a wavelet parameter. His type depend of the wavelet used. See the corresponding wavelet mfile (inside wave_defs) for the correct parameters.

WaveletOptionName,WaveletOptionValue [STRING, MISC]: Another way of writting wavelet parameters. The wavelet parameter name (a string) is followed by its value. See the corresponding wavelet mfile (inside wave_defs) for the parameter to enter.

Output Data

out [YAWTB OBJECT]: contains the different results of cwtsph

Example(s)

```
>> load world;
>> wav = cwtsph(mat,'dog',0.05,0);
>> yashow(wav);
```

References

[1] : "Ondelettes directionnelles et ondelettes sur la sphre", P. Vandergheynst, Thse, Universit Catholique de Louvain, 1998

See Also

cwtsph_yashow

Location

continuous/sphere/cwtsph.m

dergauss2d

Compute the 2D multiple derivative of Gaussian

Syntax

[out] = polgauss2d(kx,ky,orderx,ordery)

Description

This function computes the 2D multiple derivative of Gaussian. That is, the wavelet given by

```
PSIHAT (kx,ky) = (i*kx).^orderx .* (i*ky)^ordery .* ...
                              exp( - (kx.^2 + ky.^2) / 2 )
```

where PSIHAT is the Fourier transform of PSI;

This wavelet depends of two parameters: orderx and ordery. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx,ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

orderx, ordery [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = dergauss2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/dergauss2d.m

dgauss1d

Compute the n order derivative of Gaussian in frequency

Syntax

[out] = dgauss1d(k,order,sigma)

Description

This function computes the n order derivative of Gaussian in frequency. That is, the wavelet given by

```
PSIHAT (k) = - (i*k).^order .* exp( - (sigma*k).^2 / 2 )
```

where PSIHAT is the Fourier transform of PSI;

This wavelet depends of two parameters: order, sigma. This function is used by the cwt1d routine which computes continuous wavelet transform in 1D.

Input Data

k [REAL VECTOR]: The frequency vector.

order, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency.

Example(s)

```
>> k = fftshift(yapuls(128));
>> wav = dgauss1d(k,2,1);
>> plot(k,-wav)
```

References

See Also

cwt1d, cwt1dt, meshgrid, yashow_cwt1d, yashow_cwt1dt

Location

continuous/1d/wave_defs/dgauss1d.m

dog2d

Compute the 2D DoG Wavelet in frequency plane

Syntax

[out] = dog2d(kx,ky,order,sigma)

Description

This function computes the 2D DoG (Derivative of Gaussian) wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky) = (i*kx).^order * exp( - sigma^2 * ( kx.^2 + ky.^2 ) )
```

where PSIHAT is the Fourier transform of PSI. This wavelet depends of two parameters: order and sigma. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx,ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = dog2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/dog2d.m

dogsph

Spherical Difference Of Gaussians (DOG) wavelet

Syntax

[out] = dogsph(X,Y,Z,x,y,z,sc,ang,alpha)

Description

This function returns the DOG (Difference Of Gaussians) wavelet [1] which is given by:

```
   Psi_a(theta,phi) =
      lambda(theta,a)^0.5 * exp(- tan(theta/2)^2/a^2 )
                 -  1/alf * lambda(theta,a*alf)^0.5 * exp(- tan(theta/2)^2/(a*alf)^2 )

                                 4 * a^2
where lambda =      -----------------------------------
                    ( (a^2-1)*cos(theta) +  (a^2+1) )^2.
```

This function is implemented in C.

Input Data

X, Y, Z [MATRICES]: 3D coordinates of a regular spherical grid on an unit sphere ;

x,y,z [SCALARS]: 3D coordinates of the wavelet center on the sphere

sc,ang [SCALARS]: Scale and angle of the wavelet. For the zonal DOG wavelet, the angle has no influence

alpha [DOUBLE SCALAR]: Scale interval between the two Gaussians that define the wavelet.

Output Data

out [DOUBLE MATRIX]: Wavelet on a spherical grid.

Example(s)

References

[1] : "Ondelettes directionnelles et ondelettes sur la sphre", P. Vandergheynst, Thse, Universit Catholique de Louvain, 1998

[2] : Jean-Pierre Antoine, L. Jacques and P. Vandergheynst, Wavelets on the sphere : Implementation and approximations. submit to Applied and Computational Harmonic Analysis (2001)

See Also

cwtsph, dogsph, fcwtsph, morletsph, yashow, yashow_cwtsph

Location

continuous/sphere/wave_defs/dogsph.m

endstop1

Compute the single 2D EndStop Wavelet in frequency plane

Syntax

[out] = endstop1(kx,ky,k_0,sigma)

Description

This function computes the single 2D EndStop wavelet in frequency plane. This wavelet is by the derivation according to y of the Morlet wavelet of main frequency vector (k0,0).

```
PSIHAT (kx,ky) =
    (i*ky) * exp( - sigma^2 * ( (kx-k_0).^2 + (ky/epsilon).^2 ) / 2)
```

where PSIHAT is the Fourier transform of PSI.

This wavelet depends of two parameters: k_0 and sigma. This function is used by the cwt2d routine which computes continuous wavelet transform in 2D.

Input Data

kx,ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

sigma [REAL SCALAR]: The wavelet spread factor.

epsilon [REAL SCALAR]: The wavelet anisotropy factor.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = endstop1(3*kx,3*ky,6,1,1);
>> yashow(wav);
```

References

[1] Sushil Kumar BHATTACHARJEE, "A Computational Approach to Image Retrieval", PhD Thesis, EPFL, Lausanne, 1999.

See Also

cwt2d, endstop2, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/endstop1.m

endstop2

Compute the double 2D EndStop Wavelet in frequency plane

Syntax

[out] = endstop2(kx,ky,k_0,sigma)

Description

This function computes the double 2D EndStop wavelet in frequency plane. This wavelet is given by

```
PSIHAT (kx,ky) =
    -4*pi/sigma^2 * (2*ky.^2 - sigma^2) * exp( - ( kx.^2 + ky.^2 ) / sigma^2)
        * exp( - ( (kx-k_0).^2 + ky.^2 ) / 2)
```

where PSIHAT is the Fourier transform of PSI. It results from the frequency multiplication [1] of a Morlet wavelet (k_0,0) with an y double derivative of Gaussian.

This wavelet depends of two parameters: k_0 and sigma. This function is used by the cwt2d routine which computes continuous wavelet transform in 2D.

Input Data

kx,ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = endstop1(kx,ky,6,1);
>> imagesc(wav);
```

References

[1] Sushil Kumar BHATTACHARJEE, "A Computational Approach to Image Retrieval", PhD Thesis, EPFL, Lausanne, 1999.

See Also

cwt2d, endstop1, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/endstop2.m

es2cauchy2d

Compute an second endstop wavelet from the 2D Cauchy Wavelet in frequency plane

Syntax

[out] = es2cauchy2d(kx,ky,apert,sigma,l,m)

Description

This function computes the 2D Cauchy wavelet in frequency plane. That is, the wavelet given by:

```
                  -
                 |(E(apert-pi/2) . K)^l
                 |           * (E(-apert+pi/2) . K)^m
                 |           * exp( - 0.5 * sigma * |K-K_0|^2 )
PSIHAT(kx,ky) = <                      INSIDE C(-apert,apert)
                 |
                 |0                    OUTSIDE C
                  `-
where: E(alpha) = (cos(alpha), sin(alpha))
       K        = (kx, ky)
       K_0      = (l+m)^0.5 * (sigma - 1)/sigma * (1,0)
       C(-apert,apert) = the cone supported by E(apert)
                         and E(-apert)
```

The wavelet parameters are thus:

- apert : the half aperture of the cone;
- sigma : the frequency spread of the wavelet;
- l,m : its vanishing moments.

This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

apert [REAL SCALAR]: The aperture of the cone (in gradiant)

sigma [REAL SCALAR]: The frequency spread of the wavelet

l,m [INTEGERS]: The vanishing moments

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = cauchy2d(kx,ky,pi/6,1,4,4);
>> imagesc(wav);
```

References

See Also

cauchy2d, cwt2d, dergauss2d, dog2d, es2cauchy2d, es2morlet2d, esmex2d, gabor2d, gauss2d, gaussx2d, gaussz2d, isdog2d, isomor2d, meshgrid, mexican2d, morlet2d, pethat2d, rmorlet2d, samcwt2d, sarcwt2d, sdog2d, sqdog2d, wheel2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/es2cauchy2d.m

es2morlet2d

Compute the 2D Morlet Wavelet in frequency plane

Syntax

[out] = morlet2d(kx,ky,k_0,sigma)

Description

This function computes the 2D Morlet wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky) = exp( - sigma^2 * ( (kx - k0).^2 + (epsilon*ky).^2 ) / 2)
```

where PSIHAT is the Fourier transform of PSI. This wavelet depends of two parameters: k0 and sigma. It is not truly admissible but for sigma*K_0>5.5, it is considered numerically admissible. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

k_0, sigma, epsilon [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = morlet2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/es2morlet2d.m

esmex2d

Compute the endstop 2D Mexican Wavelet in frequency plane

Syntax

[out] = esmex2d(kx,ky,sigma,epsilon)

Description

This function computes the 2D Mexican wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky) = |K|^2 * exp( - (sigma*K)^2 / 2 )
                  * A * exp(- A^2 / (2 epsilon^2) );
```

where,

- PSIHAT is the Fourier transform of PSI;
- $K = (kx, ky)$;
- A =arg(K) =atan(ky/kx);
- epsilon is the inverse of the angular selectivity of PSI.

This wavelet depends on two parameters: sigma, epsilon. These are respectively the radial spread and the angular selectivity.

This function is used by the cwt2d routine which computes continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

sigma, epsilon [REAL SCALARS]: The wavelet parameters. By default, sigmax = sigmay = sigma (isotropic case).

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = esmex2d(kx,ky,1,2);
>> yashow(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/esmex2d.m

estimnoise2d

Estimate the std. dev. of a white noise in an image.

Syntax

sigma = estimnoise(tmat [, 'cutoff'])

Description

Estimate the standard devation of a white noise in an image using a median filter and the Mexican Hat.

Input Data

tmat [MATRIX]: the FFT of the noisy image.

cutoff [REAL]: the cutoff in frequency below which the filter is zero.

Output Data

sigma [REAL]: estimation of the standard deviation of the noise.

Example(s)

```
>> %% Loading a simple disk image
>> X = thedisk(128, 'smooth');
>> figure; yashow(X,'cmap','rgray');
>> %% Creating the noisy image (sigma = 0.5)
>> [nrow,ncol]=size(X);
>> nX = X + 0.5 *randn(nrow,ncol);
>> figure; yashow(nX,'cmap','rgray');
>> %% Estimating sigma
>> sigma = estimnoise2d(fft2(nX))
```

References

See Also

Location

demos/denoising/2d/estimnoise2d.m

fcwtsph

Compute the fast spherical wavelet transform

Syntax

out = fcwtsph(fimg, wavname, scales [,WaveletParameter])
out = fcwtsph(fimg, wavname, scales [,'WaveletOptionName', WaveletOptionValue])

Description

This function performs a spherical wavelet transform on a sphere in spherical coordinates. The algorithm is based on the use of FFT according phi coordinates (phi and theta are of course defined on a cartezian grid). The computation time is of order $N^3 \log(N)$ if mat is an $N \times N$ matrix. Wavelets are taken inside the sub directory 'wave_defs' (see the README to know how to write your own wavelet).

Input Data

fimg [DOUBLE MATRIX] : the fast spherical transform of the data (cfr fst)

wavname [STRING]: the name of the wavelet to use (see yawtb/continuous/sphere/wave_defs for existing wavelet);

scales [DOUBLE VECTOR] : the interval of scales where you want to compute a spherical CWT.

angles [DOUBLE VECTOR] : the interval of angles where you want to compute a spherical CWT.

WaveletParameter [MISC]: a wavelet parameter. His type depend of the wavelet used. See the corresponding wavelet mfile (inside wave_defs) for the correct parameters.

WaveletOptionName,WaveletOptionValue [STRING, MISC]: Another way of writting wavelet parameters. The wavelet parameter name (a string) is followed by its value. See the corresponding wavelet mfile (inside wave_defs) for the parameter to enter.

Output Data

out [YAWTB OBJECT]: contains the different results of cwtsph

Example(s)

```
>> load world;
>> mat=1*(mat(1:256,1:2:end));
>> fmat = fst(mat);
>> wav = fcwtsph(fmat,'dog',0.05,0);
>> yashow(wav);
```

References

[1] : "Ondelettes directionnelles et ondelettes sur la sphre", P. Vandergheynst, Thse, Universit Catholique de Louvain, 1998

See Also

cwtsph_yashow

Location

continuous/sphere/fcwtsph.m

fftsph

Spherical Harmonic Transform

Syntax

out = fftsph(mat, [,'L', L] [,'axisym'] [,'cache'])

Description

Compute the Spherical Harmonic transform.

Input Data

mat [MATRIX]: the function to transform on an equiangular latitude-longitude spherical grid of size (2M+1)xN. The number of latitudes must be odd!

L [INT]: the order where to stop the transform. Default value: M.

'axisym' [BOOL]: tells to fftsph if mat represent an axisymmetric function, that is a function symmetric around the poles.

'cache' [BOOL]: tells to fftsph to cache spherical harmonics in tempdir (normally /tmp/) for further uses. If previous caches are detected, fftsph load the previous result instead of computing once again the spherical harmonics. This caching becomes intersting only from M greater than 64 in the axisymmetric case.

Remark: The call of fftsph([], 'clearcache') clear the cache.

Output Data

out [MATRIX]: the resulting coefficients. The size of out is 1x(L+1) if mat is axisymmetric, and (L+1)x(2L+1) if not. In the last case, for each l¡=L, coefficents are sorted in out(l,:) as -l, -l+1, ..., l-1, l, 0, ..., 0 with 2L+1 - (2l+1) zeros at the end.

Example(s)

```
>> %% Creating the equiangular spherical grid
>> [phi, theta] = meshgrid(vect(0,2*pi,16,'open'), vect(0,pi,9));
>> %% Creating a spherical function, here its is just Y43
>> mat = yaspharm(theta, phi, 4, 3);
>> figure; yashow(mat, 'spheric', 'relief', 'mode', 'real');
>> %% Computing the SH transform
>> tmat = fftsph(mat);
>> %% Finding important values (> 10*eps ~10^-15)
>> abs(tmat) > 10*eps
>> %% Not zero value in (5,8), i.e. l=4 and m=3 (8 - l - 1)
>> tmat(5,8)
```

References

See Also

ifftsph,, sphweight,, yaspharm

Location

tools/misc/fftsph.m

freq_delta

Syntax

[] = freq_delta()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/frame_defs/freq_delta.m

fst

Fast spherical harmonic transform

Syntax

coeffs = fst(data)

Description

This mex file computes a fast spherical harmonic transform of the spherical data `data` defined on a spherical grid 2B*2B where B is the frequency bandwidth of the data.

To perform this spherical transform, `fst` program uses the C functions of the (GPL) SpharmonicKit [1] based on the work of Driscoll, Healy and Rockmore [2] about fast spherical transforms.

Input Data

data [REAL MATRIX]: The spherical data described on a equi-angular (2B*2B) spherical grid (θ_i, ϕ_j) with $\phi_j = j\frac{2\pi}{2B}$ $(j = 0..2B-1)$ and $\theta_i = (2i+1)\frac{\pi}{2B}$ $(i = 0..2B-1)$. Notice that B must be a power of 2.

Output Data

coeff [COMPLEX MATRIX]: The spherical harmonic coefficients. Those are defined in a matrix of size B*B (with B define above). This matrix is formed by the concataining of the coefficents C(m,l) ($|m| \leq l$) in the following order: C(0,0) C(0,1) C(0,2) ... ... C(0,B-1) C(1,1) C(1,2) ... ... C(1,B-1) etc. C(B-2,B-2) C(B-2,B-1) C(B-1,B-1) C(-(B-1),B-1) C(-(B-2),B-2) C(-(B-2),B-1) etc. C(-2,2) ... ... C(-2,B-1) C(-1,1) C(-1,2) ... ... C(-1,B-1)

This only requires an array of size (B*B). Use lmshape to turn this representation in a more human readable form. ilmshape may next come back to the original representation.

Example(s)

```
>> [phi,theta] = sphgrid(2*128);
>> mat = exp(-tan(theta/2).^2) .* ( phi == phi(1,10));
>> fmat = fst(mat);
>> figure; yashow(fmat);
>> figure; yashow(lmshape(fmat));
```

References

[1] SpharmonicKit: http://www.cs.dartmouth.edu/ geelong/sphere/. Developed by Sean Moore, Dennis Healy, Dan Rockmore, Peter Kostelec.

[2] D. Healy Jr., D. Rockmore, P. Kostelec and S. Moore, "FFTs for the 2-Sphere - Improvements and Variations", Journal of Fourier Analysis and Applications, 9:4 (2003), pp. 341 - 385.

See Also

convsph, convsph_semi, fzt, ifst, ilmshape, ilmshape, lmshape

Location

interfaces/spharmonickit/fst.m

fwt2d

2D framed wavelet transform

Syntax

obj = fwt2d(tI, framename, J, K, ... [, 'scbase', scbase] [, 'scfirst', scfirst] ... [, 'sc', sc], [, 'ang', ang] ... [, 'FrameOptionName', FrameOptionValue] ... [, 'export', export])

Description

`fwt2d` computes and returns the 2D framed wavelet transform, or frame decomposition, of an image, that is the coefficients:

```
App      =  I * PHI[a\_J]
Wav[j,k] =  I * PSI[a\_j, \theta\_k]
```

where * is the convolution operator, $j = 1..J$, $k = 1..K$, $a_j = a_0 * p^(j - 1)$ for a first scale a_0 and a base p, $\theta_k = k * 2 * pi/K$.

The frame scheme, that is the set of of functions PHI, PSI and CHI, may be chosen among the ones defined in the subdirectory 'frame_defs'. Notice that in this directory, to one kind of frame correspond always three functions:

- '¡framename¿_app' for the low frequency approximation
- '¡framename¿_wav' for the wavelet definition the decomposition.

Input Data

tI [CPLX ARRAY]: the Fourier transform of the image to analyze;

framename [STRING]: the name of the frame to use;

J [INTEGER]: the number of scales on which the frame is based;

K [INTEGER—VECTOR]: in case of a directional frame, the number of sectors to use. If this number is different for each scale, it is allowed to enter a vector of length `J` representing the number of sectors for each scale from the first to the last one;

scbase [REAL]: the base 'p' in the power law rule that governs the scales, i.e. $a_j = a_0 * p^(j - 1)$;

scfirst [REAL]: the first scale a_0 in the scale rule above. By default, this a_0 is set to 1/2 giving a complete covering of the frequency plane if the wavelet is contained in the ring -pi/2 ¡= abs(k) ¡ pi;

sc [VECTOR]: vector of scale indices between 1 and 'J' in which the decomposition must be restricted;

ang [VECTOR]: vector of angle indices between 1 and 'K' in which the decomposition must be restricted;

export [STRING]: could be 'app', 'wav', 'allwav', or 'rem' and involves respectively the computation of only the approximation, the wavelet coefficients or the high frequency remainder.

Output Data

out [STRUCT]: the output of the transform. It is a structure with the following fields:

- `out.type` [STRING]: transform type ('fwt2d')
- `out.app` [MATRIX]: the approximation coefficients;
- `out.wav` [CELL]: 2D list of 2D matrices where the wavelet coefficients are stored.
- `out.rem` [MATRIX]: matrix storing the high frequency remains coefficients

Example(s)

```
>> %% Create the woman example
>> load woman; tX = fft2(X);
>> figure; yashow(X,'square','cmap','gray');
>> %% Decompose mat on the Angular Spline frame of 3 scales and 6 angles
>> fc = fwt2d(tX,'aspline',2,9);
>> %% Displaying the approximation
>> figure; yashow(fc.app, 'square');
>> %% Displaying the wavelet coefficient for sc=1 and ang=2
>> yashow(fc.wav{1,2}, 'square');
>> %% Displaying the summation of all the frequency masks
>> yashow(fftshift(fc.allwav), 'square');
>> %% Rebuilding the matrix without the approximation
>> fc.app = zeros(size(X));
>> nX=ifwt2d(fc);
>> %% Showing the result
>> yashow(nX, 'square', 'cmap', 'gray');
```

References

See Also

aspline2d_app, aspline2d_high, aspline2d_info, aspline2d_wav

Location

discrete/frames/2d/fwt2d.m

fwt2d_allwav

Syntax

[] = fwt2d_allwav()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_allwav.m

fwt2d_app

Syntax

[] = fwt2d_app()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_app.m

fwt2d_app_filter

Syntax

[] = fwt2d_app_filter()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_app_filter.m

fwt2d_dapp_filter

Syntax

[] = fwt2d_dapp_filter()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_dapp_filter.m

fwt2d_denoise

Image denoising by 2D directional framed wavelet thresholding

Syntax

obj = fwt2d_denoise(tI, framename, J, K, ... [,'sigma', sigma] [, 'level', level] [, 'soft'] ... [, 'highcorr', highcorr], ... [, 'scbase', scbase] [, 'scfirst', scfirst] ... [, 'sc', sc], [, 'ang', ang] ... [, 'FrameOptionName', FrameOptionValue])

Description

Denoise an image using thresholding of the coefficient of a directional frame before the rebuilding. Hard (default) and soft thresholding are available.

Input Data

tI [CPLX ARRAY]: the Fourier transform of the image to analyze;

framename [STRING]: the name of the frame to use. See the subdirectory in yawtb/discrete/2d/frame_defs to see what are the frames availables;

J [INTEGER]: the number of scales on which the frame is based;

K [INTEGER]: in case of a directional frame, the number of sectors to use.

sigma [REAL]: the std. dev. of the noise in the noisy image. If unknown, an estimation of the noise is computed using `estimnoise2d`.

level [REAL]: the level of thresholding to perform. Default value: $\sqrt(2 * log(n))$ with n the number if pixels in the image;

'soft' [BOOL]: perform a soft thresholding instead of the default hard thresholding.

highcorr [REAL]: correction to apply to the normal thresholding level for the high frequency coefficient of the frame;

scbase [REAL]: the base 'p' in the power law rule that governs the scales, i.e. $a_j = a_0 * p^(j - 1)$;

scfirst [REAL]: the first scale a_0 in the scale rule above. By default, this a_0 is set to 1/2 giving a complete covering of the frequency plane if the wavelet is contained in the ring -pi/2 ¡= abs(k) ¡ pi;

sc [VECTOR]: vector of scale indices between 1 and 'J' in which the decomposition must be restricted;

ang [VECTOR]: vector of angle indices between 1 and 'K' in which the decomposition must be restricted;

Output Data

out [CPLX MATRIX]: The denoised image.

Example(s)

```
>> %% Denoising of Lena (256x256 256 gray levels)
>> load lena256
>> %% Adding noise (PSNR ~20 dB)
>> nX = double(X) + 255/10*randn(256,256); yapsnr(X,nX)
>> %% Denoising using the Meyer frame, 4 scale level and 8 orientations
>> rX = fwt2d_denoise(fft2(nX), 'meyer', 4, 8, 'sigma', 255/10);
>> %% The gain
```

```
>> yapsnr(X,rX)
>> %% The results
>> figure;yashow(X,'cmap','gray');
>> figure;yashow(X,'cmap','gray');
>> figure;yashow(rX,'cmap','gray');
\subsubsection{\textcolor{dark-blue}{References}}
\label{cmd:fwt2d_denoise:Ref}


\subsubsection{\textcolor{dark-blue}{See Also}}
\label{cmd:fwt2d_denoise:See}



\subsubsection{\textcolor{dark-blue}{Location}}
demos/denoising/2d/fwt2d\_denoise.m
\clearpage


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


\subsection{\textcolor{red}{fwt2d\_dhigh\_filter}}
\label{cmd:fwt2d_dhigh_filter}

\subsubsection{\textcolor{dark-blue}{Syntax}}
\label{cmd:fwt2d_dhigh_filter:Syn}


[] = fwt2d\_dhigh\_filter()

\subsubsection{\textcolor{dark-blue}{Description}}
\label{cmd:fwt2d_dhigh_filter:Des}


\paragraph{\textcolor{dark-blue}{Input Data}}
\label{cmd:fwt2d_dhigh_filter:InpD}

\begin{description}
\item[] []:
\end{description}

\paragraph{\textcolor{dark-blue}{Output Data}}
\label{cmd:fwt2d_dhigh_filter:OutD}

\begin{description}
\item[] []:
\end{description}

\subsubsection{\textcolor{dark-blue}{Example(s)}}
\label{cmd:fwt2d_dhigh_filter:Exa}

\begin{verbatim}
```

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_dhigh_filter.m

fwt2d_dwav_filter

Syntax

[] = fwt2d_dwav_filter()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_dwav_filter.m

fwt2d_high

Syntax

[] = fwt2d_high()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_high.m

fwt2d_high_filter

Syntax

[] = fwt2d_high_filter()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_high_filter.m

fwt2d_init

Syntax

[] = fwt2d_init()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_init.m

fwt2d_sthresh

Syntax

[] = fwt2d_sthresh()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

demos/denoising/2d/fwt2d_sthresh.m

fwt2d_thresh

Syntax

[] = fwt2d_thresh()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_thresh.m

fwt2d_wav

Syntax

[] = fwt2d_wav()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/fwt2d_wav.m

fwt2d_wav_filter

Syntax

[] = fwt2d_wav_filter()

Description

Input Data

[]:

Output Data

[]:

Example(s)

>>

References

See Also

Location

discrete/frames/2d/fwt2d_wav_filter.m

fwtsph

Framed Wavelet Transforn on the sphere

Syntax

out = fwtsph(mat, wavname, ... [,'WaveletOptionName', WaveletOptionValue], ... [,'a0', a0], [,'tg'], [,'voice', voice]);

Description

Input Data

mat [REAL MATRIX]: The spherical signal on an equiangular spherical grid. Its size must be $2^M + 1x2^N$ with M and N positive integer.

wavname [STRING]: The name of the spherical wavelet (for instance 'dog').

WaveletOptionName, WaveletOptionValue [STRING, MISC]: The wavelet parameter name (a string) is followed by its value. See the corresponding wavelet mfile (inside continuous/sphere/wav_defs) to know the parameters to enter.

a0 [REAL]: the scale to start de scale sequence according to the rule selected ('tangential' or 'dyadic'). Default value: $a_0 = 4$.

tg [BOOL]: Use a tangential scale sequence, that is $a_j = a_0 tan((\pi/4)2^{-j})$, instead of the default dyadic one, i.e. $a_j = a_0 2^{-j}$

voice [INT]: Number of voice in each scale octave. Default value: 1.

Output Data

out [SRUCT]: Structure containing the frame coefficients. Imortant fields are:

- **out.img** [REAL MATRIX]: the orignal matrix;
- **out.wavname** [STRING]: the name of wavelet;
- **out.wavopts** [MISC]: the parameters of the wavelet;
- **out.J** [INT]: the number of scale equals to $M - 1$;
- **out.jv** [INT VECTOR]: scale index;
- **out.a0** [REAL]: the initial scale where the scale sequence is starded;
- **out.a** [REAL VECTOR]: the scale sequence;
- **out.nth** [INT]: number of theta values in the spherical grid;
- **out.nph** [INT]: number of phi values in the spherical grid;
- **out.lth** [INT]: log2(out.nth -1);
- **out.lph** [INT]: log2(out.nph);
- **out.th_step** [INT VECTOR]: number of points between two adjacent theta angles in function of scale;
- **out.ph_step** [INT VECTOR]: number of points between two adjacent phi angles in function of scale;

Example(s)

```
>> load world
>> yashow(mat,'spheric','fig',1);
>> wav=fwtsph(mat,'dog'); %% It's time to drink a cup of coffee!
>> yashow(wav.data{1},'spheric','fig',2);
>> yashow(wav.data{2},'spheric','fig',3);
>> yashow(wav.data{3},'spheric','fig',4);
```

```
>> yashow(wav.data{4},'spheric','fig',5);
>> yashow(wav.data{5},'spheric','fig',6);
>> yashow(wav.data{6},'spheric','fig',7);
>> yashow(wav.data{7},'spheric','fig',8);
>> yashow(wav.data{8},'spheric','fig',9);
```

References

See Also

Location

discrete/frames/sphere/fwtsph.m

fzt

Fast spherical harmonic transform

Syntax

coeffs = fzt(data)

Description

This mex file computes a fast spherical zonal (harmonic) transform of the spherical data `data` assuming it zonal, i.e. invariant under rotation around the north pole, and defined on a spherical grid 2B*2B where B is the frequency bandwidth of the data.

To perform this zonal transform, `fzt` program uses the C functions of the (GPL) SpharmonicKit [1] based on the work of Driscoll, Healy and Rockmore [2] about fast spherical transforms.

Input Data

data [REAL MATRIX]: The spherical data described on a equi-angular (2B*2B) spherical grid (θ_i, ϕ_j) with $\phi_j = j\frac{2\pi}{2B}$ $(j = 0..2B-1)$ and $\theta_i = (2i+1)\frac{\pi}{2B}$ $(i = 0..2B-1)$. Notice that B must be a power of 2.

The fact that data are zonal suppose that there is no dependence in phi.

Output Data

coeff [COMPLEX MATRIX]: The spherical harmonic coefficients. Those are defined in a matrix of size B*B (with B define above). This matrix is formed by the concataining of the coefficents C(m,l) $(|m| \leq l)$ in the following order: C(0,0) C(0,1) C(0,2) ... ... C(0,B-1) C(1,1) C(1,2) ... ... C(1,B-1) etc. C(B-2,B-2) C(B-2,B-1) C(B-1,B-1) C(-(B-1),B-1) C(-(B-2),B-2) C(-(B-2),B-1) etc. C(-2,2) ... ... C(-2,B-1) C(-1,1) C(-1,2) ... ... C(-1,B-1)

This only requires an array of size (B*B). Use lmshape to turn this representation in a more human readable form. ilmshape may next come back to the original representation.

Example(s)

```
>> [phi,theta] = sphgrid(2*128);
>> mat = exp(-(tan(theta/2)/0.1).^2);
>> fmat = fzt(mat);
>> figure; yashow(fmat);
>> l=0:127; figure; plot(l,fmat(1,:));
```

References

[1] SpharmonicKit: http://www.cs.dartmouth.edu/ geelong/sphere/. Developed by Sean Moore, Dennis Healy, Dan Rockmore, Peter Kostelec.

[2] D. Healy Jr., D. Rockmore, P. Kostelec and S. Moore, "FFTs for the 2-Sphere - Improvements and Variations", Journal of Fourier Analysis and Applications, 9:4 (2003), pp. 341 - 385.

See Also

convsph, convsph_semi, fst, ifst, ifst, ilmshape, ilmshape, lmshape

Location

interfaces/spharmonickit/fzt.m

gabor2d

Compute the 2D Gabor Wavelet in frequency plane

Syntax

[out] = gabor2d(kx,ky,k_0,sigma)

Description

This function computes the 2D Gabor wavelet in frequency plane. This wavelet is identical to the 2D Morlet wavelet (see morlet2d.m), that is, the wavelet given by

```
PSIHAT (kx,ky) = exp( - sigma^2 * ( (kx - k0).^2 + ky.^2 ) / 2)
```

where PSIHAT is the Fourier transform of PSI. This wavelet depends of two parameters: k0 and sigma. It is not truly admissible but for sigma*K_0>5.5, it is considered numerically admissible. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

k_0, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = gabor2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, es2morlet2d, meshgrid, morlet2d, rmorlet2d, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/gabor2d.m

gabor2d_app

The approximation function of the Gabor frame.

Syntax

out = gabor2d_app(kx, ky)

Description

Compute the approximation function of the Gabor frame, that is a Gaussian centered on the frequency origin and of spread equals to pi/2.

Input Data

kx [MATRIX]: the horizontal frequencies;

ky [MATRIX]: the vertical frequencies;

Output Data

out [MATRIX]: the computed function on the frequency plane.

See Also

Location

discrete/frames/2d/frame_defs/gabor2d_app.m

gabor2d_wav

The wavelet of the Gabor frame.

Syntax

out = gabor2d_app(kx, ky, nang, sigma, autospread)

Description

Compute the wavelet of the Gabor frame, that is a Gabor function given by

```
PSIHAT(kx,ky) = exp( - 0.5 * [ ((kx - k0)/wx)^2 + (ky/wy)^2 ] ),
```

where k0 = 3*pi/4, `wx` and 'wy' are respectively the horizontal and the vertical spread. 'wx' is always equal to 1/`sigma`. In the case where autospread is set to 1, 'wy' is computed to obtain a complete angular covering according to the number of sectors `nang`. If `autospread` is set to 0 (default), then 'wy' equals 'wx', that is 1/`sigma`.

Input Data

kx [MATRIX]: the horizontal frequencies;

ky [MATRIX]: the vertical frequencies;

nang [INTEGER]: the number of angular sectors on which the frame decomposition is computed.

sigma [REAL]: the horizontal spread of the gabor function in position.

autospread [BOOLEAN]: 1 if 'wy' must be set to obtain a complete angular covering with 'nang' rotated wavelets, 0 if not.

Output Data

out [MATRIX]: the computed function on the frequency plane.

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/frame_defs/gabor2d_wav.m

gauss1d

1D Gaussian window in the frequency domain

Syntax

[out] = gauss1d(k,sigma)

Description

This function returns `out` the 1D Gaussian in the frequency domain defined in that domain by:

```
out = exp( - sigma^2 * k.^2/2 )
```

Input Data

k [REAL MATRIX]: The frequency. You can use the YAWTBfunction `vect` to create it (see example below)

sigma [REAL SCALAR]: The spread of the Gaussian.

Output Data

out [REAL MATRIX]: The Gaussian in frequency.

Example(s)

```
>> step = 2*pi/128;
>> k =  -pi : step : (pi-step);
>> wav = gauss1d(k,6);
>> plot(k,wav);
```

Note that the two first lines are implemented by the YAWTB`vect` function:

```
>> k = vect(-pi,pi,128,'open');
```

References

See Also

cgt1d, gauss1d, vect

Location

continuous/1d/win_defs/gauss1d.m

gauss2d

Compute the bidimensionnal Gaussian

Syntax

[out] = gauss2d(kx,ky,sigma)

Description

This function computes the bidimensionnal Gaussian. This is not truly a wavelet but this function can be usefull for approximation calculation.

Input Data

kx,ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

sigma [REAL SCALARS]: The spread of the Gaussian.

Output Data

out [REAL MATRIX]: The Gaussian in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = gauss2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/gauss2d.m

gaussx2d

Compute the 2D x order derivative of Gaussian

Syntax

[out] = gaussx2d(kx,ky,order,sigma[,sigmax][,sigmay])

Description

This function computes the 2D x order derivative of Gaussian. That is, the wavelet given by

```
PSIHAT (kx,ky) = kx.^order .* exp( - (A*K).^2 / 2 )

where PSIHAT is the Fourier transform of PSI;
      K = (kx,ky);
      A = diag(sigmax,sigmay).
```

This wavelet depends of three parameters: order, sigmax, sigmay. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx,ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma, sigmax, sigmay [REAL SCALARS]: The wavelet parameters. By default, sigmax = sigmay = sigma (isotropic case).

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = gaussx2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/gaussx2d.m

gaussz2d

Compute the 2D z order derivative of Gaussian

Syntax

[out] = gaussz2d(kx,ky,order,sigma[,sigmax][,sigmay])

Description

This function computes the 2D x order derivative of Gaussian. That is, the wavelet given by

```
PSIHAT (kx,ky) = i*(kx + i*ky).^order .* exp( - (A*K).^2 / 2 )

where PSIHAT is the Fourier transform of PSI;
      K = (kx,ky);
      A = diag(sigmax,sigmay).
```

Notice that for real images, inside a CWT computed with the gaussz2d wavelet: - order = 1: gives the gradient of the smoothed analyzed image in complex representation - order = 2: gives the maximum curvature of this smoothed image vector in complex coordinates.

This wavelet depends of three parameters: order, sigmax, sigmay. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx,ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma, sigmax, sigmay [REAL SCALARS]: The wavelet parameters. By default, sigmax = sigmay = sigma (isotropic case).

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = gaussz2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/gaussz2d.m

getopts

Return the value of the 'OptionName' variable inside list 'OptionList'.

Syntax

[val, NewOptionList] = getops(OptionList, 'OptionName' ... [, OptionDefaultValue [,HasNoValue]])

Description

Return the value of the 'OptionName' variable inside list 'OptionList' and return a list NewOptionList without this option.

If this variable is not present, val is empty unless a OptionDefaultValue is given.

If HasNoValue is set to 1, val is set to 1 if OptionName is find, and 0 if not.

Input Data

OptionList [LIST] The list of options and value. Its syntax follows the pattern: 'OPT1', VAL1, 'OPT2', VAL2, ...;

OptionName [STRING] The option to seek inside OptionList;

OptionDefaultValue [MISC] The default value to give at val if none is detected.

HasNoValue [BOOL] Flag set to 1 if OptionName has no value (flag case).

Output Data

val [MISC] the output value.

NewOptionList [LIST] the new list of options without the option 'OptionName'.

Example(s)

```
>> [val,list] = getopts({'sigma', 1, 'rho', 2.3},'rho',7)
>> val = getopts({'sigma', 1, 'radian'},'radian',[],1)
```

References

See Also

Location

tools/misc/getopts.m

getyawtbprefs

Get the YAWtb preferences

Syntax

getyawtbpref properties = getyawtbpref value = getyawtbpref('property_name')

Description

Get the YAWtb preferences. This command display all the YAWtb properties getyawtbprefs The following command return all the pereferences in 'properties' properties = getyawtbpref To select only one property enter value = getyawtbpref('property_name') Value is set to [] if the property doesn't exist.

Input Data

property_name [STRING]: the name of the desired property.

Output Data

properties [STRUCT]: a structure with all the properties for fields and their corresponding values.

value [MISC]: the value of the property (empty if not existing)

Example(s)

```
>> getyawtbprefs
>> getyawtbprefs('yapbarVisible')
```

References

See Also

setyawtbprefs

Location

tools/misc/getyawtbprefs.m

ifftsph

Inverse Spherical Harmonic Transform

Syntax

[out] = ifftsph(fmat, theta, phi)

Description

Compute the inverse Spherical Harmonics transform.

Input Data

fmat [MATRIX]: The SPherical Harmonics coeffcients, for instance the result of fftsph.

theta, phi [MATRICES]: the equiangular grid in latitude-longitude coordinate where the function is defined. The size of this grid must be (2M+1)xN with an odd number of latitudes.

Output Data

out [MATRIX]: the rebuilt function.

Example(s)

```
>> %% Creating the equiangular spherical grid
>> [phi, theta] = meshgrid(vect(0,2*pi,32,'open'), vect(0,pi,17));
>> %% Creating a function
>> mat = yaspharm(theta, phi, 3,2) + yaspharm(theta, phi, 5,0);
>> figure; yashow(mat, 'spheric', 'relief', 'mode', 'real');
>> %% Computing the SH transform
>> tmat = fftsph(mat);
>> nmat = ifftsph(tmat, theta, phi);
>> yapsnr(mat,nmat)
>> figure; yashow(nmat, 'spheric', 'relief', 'mode', 'real');
```

References

See Also

fftsph,, sphweight,, yaspharm.

Location

tools/misc/ifftsph.m

ifst

Fast spherical harmonic transform

Syntax

data = ifst(coeffs)

Description

This mex file computes a fast inverse spherical harmonic transform of the spherical coefficients `coeffs` defined on a grid B*B where B is the frequency bandwidth of the data. Be carreful to either use the output of `fst` for these coefficients, ot to form them by using `ilmshape`.

To perform this spherical convolution, `ifst` program uses the C functions of the (GPL) SpharmonicKit [1] based on the work of Driscoll, Healy and Rockmore [2] about fast spherical transforms.

Input Data

coeff [COMPLEX MATRIX]: The spherical harmonic coefficients. Those are defined in a matrix of size B*B (with B the spherical bandwidth such that the parameter l¡=B). This matrix is formed by the concataining of the coefficents C(m,l) ($|m| \leq l$) in the following order: C(0,0) C(0,1) C(0,2) ... ... C(0,B-1) C(1,1) C(1,2) ... ... C(1,B-1) etc. C(B-2,B-2) C(B-2,B-1) C(B-1,B-1) C(-(B-1),B-1) C(-(B-2),B-2) C(-(B-2),B-1) etc. C(-2,2) ... ... C(-2,B-1) C(-1,1) C(-1,2) ... ... C(-1,B-1)

This only requires an array of size (B*B). Use lmshape to turn this representation in a more human readable form. ilmshape may next come back to the original representation.

Output Data

data [REAL MATRIX]: The spherical data described on a equi-angular (2B*2B) spherical grid (θ_i, ϕ_j) with $\phi_j = j\frac{2\pi}{2B}$ $(j = 0..2B-1)$ and $\theta_i = (2i+1)\frac{\pi}{2B}$ $(i = 0..2B-1)$. Notice that B must be a power of 2.

Example(s)

```
>> load world2; yashow(mat, 'spheric', 'fig', 10); colorbar; %% 256x256 array
>> nmat = ifst(fst(double(mat)));
>> yashow(nmat, 'spheric', 'fig', 11); colorbar;
>> %% Gibs-like oscillating are due to the frequency cutoff at bandwith 128
```

References

[1] SpharmonicKit: http://www.cs.dartmouth.edu/ geelong/sphere/. Developed by Sean Moore, Dennis Healy, Dan Rockmore, Peter Kostelec.

[2] D. Healy Jr., D. Rockmore, P. Kostelec and S. Moore, "FFTs for the 2-Sphere - Improvements and Variations", Journal of Fourier Analysis and Applications, 9:4 (2003), pp. 341 - 385.

See Also

convsph, convsph_semi, fst, fzt, ifst, ilmshape, ilmshape, lmshape

Location

interfaces/spharmonickit/ifst.m

ifwt2d

2D inverse framed wavelet transform

Syntax

out = fwt2d(yastruct, ['import', import]);

Description

`fwt2d` computes the 2D inverse Framed Wavelet Transform on the basis of the fwt2d's result.

Input Data

yastruct [STRUCT]: the result of `fwt2d`

import [STRING]: can be 'app', 'wav' or 'rem' and allow `ifwt2d` to rebuild only respectively the approximation, the wavelet coefficient or the high frequency remainder.

Output Data

out [MATRIX]: the reconstructed image.

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/ifwt2d.m

ifwt2d_app

Syntax

[] = fwt2d_app()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/ifwt2d_app.m

ifwt2d_high

Syntax

[] = fwt2d_high()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/ifwt2d_high.m

ifwt2d_wav

Syntax

[] = fwt2d_wav()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/ifwt2d_wav.m

ifwtsph

Inverse Framed Wavelet Transform approximation on the sphere

Syntax

out = ifwtsph(fwt)

Description

Compute an approximation of the rebuilding of the spherical data using the same wavelet for the reconstruction than for the analysis.

Input Data

fwt [STRUCT]: the result of the framed wavelet tranform obtain with fwtsph.

Output Data

out [MATRIX]: the rebuilt spherical signal

Example(s)

```
>> load world
>> yashow(mat, 'spheric', 'fig', 1);
>> wav = fwtsph(mat, 'dog'); %% It's time to drink a cup of coffee!
>> nmat = ifwtsph(wav);
>> yashow(nmat, 'spheric', 'fig', 2);
```

References

See Also

Location

discrete/frames/sphere/ifwtsph.m

ilmshape

Reshape the output of fst in a l-m-readable matrix

Syntax

[coeffs] = ilmshape(lmcoeffs)

Description

Reshape the (2B-1)*B matrix output of lmshape ins the B*B matrix format used by the fast forward and inverse spherical transform `fst` and `ifst`.

Input Data

lmcoeffs [COMPLEX MATRIX]: matrix of size (2*B-1)*B in which the coefficient C(m,l) are organized as follow C(0,0) C(0,1) C(0,2) ... ... C(0,B-1) 0 C(1,1) C(1,2) ... ... C(1,B-1) ... ... C(.,B-1) 0 ... 0 C(B-2,B-2) C(B-2,B-1) 0 ... 0 C(B-1,B-1) 0 0 C(-(B-1),B-1) 0 0 C(-(B-2),B-2) C(-(B-2),B-1) ... ... C(-.,B-1) 0 0 C(-2,2) ... ... C(-2,B-1) 0 C(-1,1) C(-1,2) ... ... C(-1,B-1) which is more readable when visualize through yashow(lmcoeff) for instance.

Output Data

coeffs [COMPLEX MATRIX]: The spherical harmonic coefficients format used by `fst` and `ifst`. Those are defined in a matrix of size B*B (with B define above). This matrix is formed by the concataining of the coefficents C(m,l) ($|m| \leq l$) in the following order: C(0,0) C(0,1) C(0,2) ... ... C(0,B-1) C(1,1) C(1,2) ... ... C(1,B-1) etc. C(B-2,B-2) C(B-2,B-1) C(B-1,B-1) C(-(B-1),B-1) C(-(B-2),B-2) C(-(B-2),B-1) etc. C(-2,2) ... ... C(-2,B-1) C(-1,1) C(-1,2) ... ... C(-1,B-1)

This only requires an array of size (B*B) but is very difficult to read. So, `ilmshape` remove just the zeros padding the lmcoeffs representation.

Example(s)

```
>> %% Determines the spatial shape of C(l,m)=(l<10)
>> l = 0:127;
>> fmat = [l<10; zeros(2*128-2,128)]; size(fmat)
>> mat = ifst(ilmshape(double(fmat)));
>> figure; yashow(mat,'spheric','fig',1,'relief','mode','real','cmap','jet'); colorbar
```

See Also

convsph, convsph_semi, fst, fzt, ifst, ifst, ilmshape

Location

interfaces/spharmonickit/ilmshape.m

inputeof

Work like input(msg,'s') but ends by a EOF

Syntax

out = inputeof(msg)

Description

Input Data

msg [STRING]: your prompt message

Output Data

out [STRING]: the text entered before the closing EOF

Example(s)

```
>> s=inputeof(['Enter something and terminate by typing ...
EOF<enter> on a new line']);
```

References

See Also

Location

tools/misc/inputeof.m

isdog2d

Compute the Inverse 2D Scaling difference of Gaussian Wavelet

Syntax

[out] = isdog2d(kx,ky,alpha)

Description

This function computes the 2D Scaling difference of Gaussian Wavelet in frequency plane. This wavelet given by

```
PSIHAT (kx,ky) = N*exp( - K.^2/2 ) - N*alpha^2 * exp ( - alpha^2 * K.^2/2)
```

where: $N = (alpha^2 - 1)^{-1}$, a normalizing term, and PSIHAT is the Fourier transform of PSI and K = (kx,ky). This wavelet depends of the alpha parmeter. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = isdog2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/isdog2d.m

isomor2d

Compute the isotropical Morlet Wavelet in frequency plane

Syntax

[out] = morlet2d(kx,ky,k_0,sigma)

Description

This function computes the isotropical 2D Morlet wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky) = - exp( - sigma^2 * (|K| - k0).^2 / 2)
```

where PSIHAT is the Fourier transform of PSI. This wavelet depends of two parameters: k0 and sigma. It is not truly admissible but for sigma*K_0>5.5, it is considered numerically admissible. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

k_0, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = isomor2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/isomor2d.m

iwpck2d

Compute the reconstruction of the wavelet packets transform

Syntax

[img] = iwpck2d(yawpck)

Description

This function computes the reconstruction of the wavelet packets transform from the output of the 'wpck2d' function which calculates the wavelet packet decomposition.

Input Data

yawpck [STRUCT]: The output of 'wpck2d'

Output Data

img [COMPLEX MATRIX]: the rebuilded image.

Example(s)

```
>> mat = theL(128);
>> fmat = fft2(mat);
>> sc = linspace(0.3,10,12);
>> ypck = wpck2d(fmat,'pmexican',sc);
>> figure; imagesc(real(ypck.approx));
>> figure; imagesc(real(ypck.details(:,:,1)));
>> figure; imagesc(real(iwpck2d(ypck)));
```

Compute the wavelet packet decomposition of a 'L' picture (see 'theL'), show the approximation at the last scale and the first details and finally display the rebuilded image.

References

See Also

iwpck2d, pmexican2d, theL, wpck2d, yashow_wpck2d

Location

discrete/packet/2d/iwpck2d.m

linearchirp

Return a complex linear chirp and its theoretical instantaneous frequency

Syntax

[out,nu] = linearchirp([n[,nu_0[,nu_1[,phi]]]])

Description

This function computes and returns a complex linear chirp of n points whose instantaneous frequency starts at nu_0 ends at nu_1 with phi as the phase at the origin.

The default values are: n = 256, nu_0 = 0, nu_1 = 0.4, phi = 0

Input Data

n [INTEGER]: the number of points.

nu_0 [REAL abs(nu_0)¡0.5]: Instantaneous frequency of the first point.

nu_1 [REAL abs(nu_1)¡0.5]: Instantaneous frequency of the last point.

phi [REAL]: Phase of the signal at the origin.

Output Data

out :The complex linear chirp.

nu :The instantaneous frequency of the linear chirp.

Example(s)

```
>> [signal,nu] = linearchirp(256, 0, 0.1 )
>> plot(real(signal))
```

References

See Also

Location

sample/1d/linearchirp.m

list_elem

Return the k-th element of rec if it exists.

Syntax

out = list_elem(rec,k,def)

Description

Return the k-th element of rec if exists. If not, default value def is returned.

Input Data

rec [LIST] the input list;

k [INTEGER] the element index;

def [MISC] default value.

Output Data

out the k-th element.

Example(s)

References

See Also

Location

tools/misc/list_elem.m

lmshape

Reshape the output of fst in a l-m-readable matrix

Syntax

[lmcoeffs] = lmshape(coeffs)

Description

Reshape the B*B matrix output of the fast spherical transform `fst` in a (2B-1)*B size "l-m-readable" matrix.

Input Data

coeffs [COMPLEX MATRIX]: The spherical harmonic coefficients. Those are defined in a matrix of size B*B (with B define above). This matrix is formed by the concataining of the coefficents C(m,l) ($|m| \leq l$) in the following order: C(0,0) C(0,1) C(0,2) ... ... C(0,B-1) C(1,1) C(1,2) ... ... C(1,B-1) etc. C(B-2,B-2) C(B-2,B-1) C(B-1,B-1) C(-(B-1),B-1) C(-(B-2),B-2) C(-(B-2),B-1) etc. C(-2,2) ... ... C(-2,B-1) C(-1,1) C(-1,2) ... ... C(-1,B-1)

This only requires an array of size (B*B) but is very difficult to read.

Output Data

lmcoeffs [COMPLEX MATRIX]: matrix of size (2*B-1)*B in which the coefficient C(m,l) are organized as follow C(0,0) C(0,1) C(0,2) ... ... C(0,B-1) 0 C(1,1) C(1,2) ... ... C(1,B-1) ... ... C(.,B-1) 0 ... 0 C(B-2,B-2) C(B-2,B-1) 0 ... 0 C(B-1,B-1) 0 0 C(-(B-1),B-1) 0 0 C(-(B-2),B-2) C(-(B-2),B-1) ... ... C(-.,B-1) 0 0 C(-2,2) ... ... C(-2,B-1) 0 C(-1,1) C(-1,2) ... ... C(-1,B-1) which is more readable when visualize through yashow(ncoeff) for instance.

Example(s)

```
>> yademo fst %% Contains a lmshape use
```

See Also

convsph, convsph_semi, fst, fzt, ifst, ifst, ilmshape

Location

interfaces/spharmonickit/lmshape.m

mexican1dt

1D+T Mexican Wavelet in wave-number/frequency domain

Syntax

out = mexican1dt(k,w,order,sigma[,sigmax][,sigmat])

Description

This function returns `out` the 1D+T Mexican wavelet in the wave-number/frequency domain defined in that domain by:

```
out = |K|.^order .* exp( - (A*K).^2 / 2 )
```

with

```
K = (k,w);
A = diag(sigmax,sigmat).
```

This wavelet depends on three parameters: `order`, `sigmax`, `sigmat`. This function is used by the `cwt1dt` routine which computes 1D+T continuous wavelet transform.

Input Data

k, w [REAL MATRICES]: The wave-number/frequency domain. Use meshgrid to create them.

order, sigma, sigmax, sigmat [REAL SCALARS]: The wavelet parameters. By default, `sigmax` = `sigmat` = `sigma` (isotropic case).

Output Data

out [REAL MATRIX]: The wavelet in wave-number/frequency domain.

Example(s)

References

See Also

cwt1dt, meshgrid, mexican1dt, morlet1dt, yashow_cwt1dt

Location

continuous/1dt/wave_defs/mexican1dt.m

mexican2d

Compute the 2D Mexican Wavelet in frequency plane

Syntax

[out] = mexican2d(kx,ky,order,sigma[,sigmax][,sigmay])

Description

This function computes the 2D Mexican wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky) = - |K|.^order .* exp( - (A*K).^2 / 2 )
```

where PSIHAT is the Fourier transform of PSI;

```
K = (kx,ky);
A = diag(sigmax,sigmay)
```

This wavelet depends of three parameters: order, sigmax, sigmay. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma, sigmax, sigmay [REAL SCALARS]: The wavelet parameters. By default, sigmax = sigmay = sigma (isotropic case).

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = mexican2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/mexican2d.m

mexican2d_ctr

Compute the 2D Mexican Wavelet in frequency plane

Syntax

[out] = mexican2d(kx,ky,order,sigma[,sigmax][,sigmay])

Description

This function computes the 2D Mexican wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky) = |K|.^order .* exp( - (A*K).^2 / 2 )
```

where PSIHAT is the Fourier transform of PSI;

```
K = (kx,ky);
A = diag(sigmax,sigmay).
```

This wavelet depends of three parameters: order, sigmax, sigmay. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma, sigmax, sigmay [REAL SCALARS]: The wavelet parameters. By default, sigmax = sigmay = sigma (isotropic case).

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = mexican2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/mexican2d_ctr.m

mexican3d

Compute the 3D Mexican Wavelet in frequency plane

Syntax

[out] = mexican3d(kx,ky,kz,order,sigma)

Description

This function computes the 3D Mexican wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky,kz) = |K|.^order .* exp( - (sigma*K).^2 / 2 )
```

where PSIHAT is the Fourier transform of PSI;

```
K = (kx,ky,kz);
```

This wavelet depends of two parameters: order, sigma. This function is used by the cwt3d routine which compute continuous wavelet transform in 3D.

Input Data

kx, ky, kz [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in the 3D frequency plane.

Example(s)

```
>> [kx,ky,kz] = meshgrid( vect(-pi,pi,64) );
>> wav = mexican3d(kx,ky,kz,2,1);
>> imagesc(wav);
```

References

See Also

cwt3d, meshgrid, yashow_cwt3d

Location

continuous/3d/wave_defs/mexican3d.m

meyer2d_app

Syntax

out = meyer2d_app()

Description

Input Data

[]:

Output Data

[]:

Example(s)

>>

References

See Also

Location

discrete/frames/2d/frame_defs/meyer2d_app.m

meyer2d_high

Syntax

out = meyer2d_high()

Description

Input Data

[]:

Output Data

[]:

Example(s)

>>

References

See Also

Location

discrete/frames/2d/frame_defs/meyer2d_high.m

meyer2d_info

Syntax

[] = aspline2d_info()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/frame_defs/meyer2d_info.m

meyer2d_wav

Syntax

out = meyer2d_wav()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/2d/frame_defs/meyer2d_wav.m

morlet1d

1D Morlet wavelet in the frequency domain

Syntax

[out] = morlet1d(k,k_0,sigma)

Description

This function returns `out` the 1D Morlet wavelet in the frequency domain defined in that domain by:

```
out = exp( - sigma^2 * (k - k_0).^2 / 2 )
```

This wavelet depends on two parameters, `k_0` and `sigma`. It is not truly admissible but for `sigma`*`k_0`¿5.5, it is considered numerically admissible.

Input Data

k [REAL MATRIX]: The frequency. You can use the YAWTBfunction `vect` to create it (see example below) ;

k_0, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency.

Example(s)

```
>> step = 2*pi/128;
>> k = -pi : step : (pi-step);
>> wav = morlet1d(k,1,6);
>> plot(k,abs(wav));
```

Note that the two first lines are implemented by the YAWTB`vect` function:

```
>> k = vect(-pi,pi,128,'open');
```

References

See Also

cwt1d, dgauss1d, morlet1d, sdog1d, vect

Location

continuous/1d/wave_defs/morlet1d.m

morlet1dt

1D+T Morlet Wavelet in wave-number/frequency domain

Syntax

out = morlet1dt(k,w,k_0,w_0)

Description

This function returns `out` the 1D+T Morlet wavelet in the wave-number/frequency domain defined in that domain by:

```
out = exp( - ( (k - k0).^2 + (w - w_0).^2 ) / 2).
```

This wavelet depends on two parameters: `k_0` and `w_0`. It is not truly admissible but for `k_0`>5.5 and `w_0`>5.5, it is considered as numerically admissible. This function is used by the `cwt1dt` routine which computes 1D+T continuous wavelet transform.

Input Data

k, w [REAL MATRICES]: The wave-number/frequency domain. Use meshgrid to create them.

k_0, w_0 [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in wave-number/frequency domain.

Example(s)

References

See Also

cwt1dt, meshgrid, mexican1dt, morlet1dt, yashow_cwt1dt

Location

continuous/1dt/wave_defs/morlet1dt.m

morlet2d

Compute the 2D Morlet Wavelet in frequency plane

Syntax

[out] = morlet2d(kx,ky,k_0,sigma)

Description

This function computes the 2D Morlet wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky) = exp( - sigma^2 * ( (kx - k0).^2 + (epsilon*ky).^2 ) / 2)
```

where PSIHAT is the Fourier transform of PSI. This wavelet depends of two parameters: k0 and sigma. It is not truly admissible but for sigma*K_0>5.5, it is considered numerically admissible. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

k_0, sigma, epsilon [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = morlet2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/morlet2d.m

morlet3d

Compute the 3D Morlet Wavelet in frequency plane

Syntax

[out] = morlet3d(kx,ky,kz,k_0,sigma)

Description

This function computes the 3D Morlet wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky,kz) = exp( - sigma^2 * ( kx.^2 + ky.^2 + (kz - k\_0).^2) / 2)
```

where PSIHAT is the Fourier transform of PSI. This wavelet depends of two parameters: k_0 and sigma. It is not truly admissible but for sigma*K_0>5.5, it is considered numerically admissible. This function is used by the cwt3d routine which compute continuous wavelet transform in 3D.

Input Data

kx, ky, kz [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

k_0, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> [kx,ky,kz] = meshgrid(vect(-pi,pi,128,'open'));
>> wav = morlet3d(kx,ky,kz,6,1);
>> yashow(wav);
```

References

See Also

cwt3d, meshgrid, yashow_cwt3d

Location

continuous/3d/wave_defs/morlet3d.m

morletsph

Spherical Morlet wavelet.

Syntax

[out] = morletsph(X,Y,Z,x,y,z,sc,ang,k0)

Description

This function computes the Spherical Morlet wavelet [3] given by:

```
   Psi_a(theta,phi) =
      lambda(theta,a)^0.5 * exp(i*k0*2*tan(theta/2)/a*cos(phi-ang))
                          * exp(-2*tan(theta/2)^2/a^2)
                          * 0.5 * (1 + tan(theta/2)^2/a^2)

                                   4 * a^2
where lambda =     ----------------------------------
                   ( (a^2-1)*cos(theta) +  (a^2+1) )^2.
```

Warning : this wavelet should be complex-valued, but only the real part is computed here.

This function is implemented in C.

Input Data

X,Y,Z [DOUBLE MATRICES]: the 3D coordinates of a regular spherical grid on a unit sphere.

x,y,z [DOUBLE SCALARS]: the 3D coordinates of the wavelet center on the sphere.

sc,ang [DOUBLE SCALARS]: the scale and the angle of the wavelet.

k0 [DOUBLE SCALAR]: norm of the wave vector of the wavelet in the euclidean limit (accounts for the number of oscillations)

Output Data

out [DOUBLE MATRIX]: the wavelet on a spherical grid.

Example(s)

References

s [1] : "Ondelettes directionnelles et ondelettes sur la sphre", P. Vandergheynst, Thse, Universit catholique de Louvain, 1998

[2] : Jean-Pierre Antoine, L. Jacques and P. Vandergheynst, Wavelets on the sphere : Implementation and approximations. submitted to Applied and Computational Harmonic Analysis (2001)

[3] : "Ondelettes et Dtection de Sources Gamma dans l'Univers", L. Demanet, Mmoire, Universit catholique de Louvain, Juin 2001.

See Also

cwtsph, dogsph, fcwtsph, morletsph, yashow, yashow_cwtsph

Location

continuous/sphere/wave_defs/morletsph.m

movgauss

Create a three moving Gaussians on the line.

Syntax

[out] = movgauss([nt] [,nx])

Description

This function creates three moving Gaussians on the line with speeds equal to 1, 0.5, -1 pixel/frame.

Input Data

nt, nx [INTEGERS]: the number of times and positions (default 64×128).

Output Data

out [DOUBLE MATRIX]: a nt×nx matrix containing the three moving Gaussians.

Example(s)

```
>> mat = movgauss;
%% CWT Velocity-position representation on frame 10
>> wav=cwt1dt(fft2(mat),'morlet',6,vect(-2,2,128),'time',10);
>> yashow(wav);
```

References

See Also

Location

sample/1dt/movgauss.m

normopts

Reject to the end of list the options charaterized by a string.

Syntax

[out] = normopts(list)

Description

Reject to the end of list 'list' the options charaterized by a string. So 1,2, 'Opt', OptVal, 4,3 will be turn into 1,2,4,3, 'Opt', OptVal.

Input Data

list [LIST] the input options list.

Output Data

out [LIST] the output list.

Example(s)

```
>> out = normopts({'sigma',6, 1,pi})\\
out =
     [1] [3.1416] 'sigma' [6]
```

References

See Also

getopts

Location

tools/misc/normopts.m

pethat2d

Compute the 2D Pet Hat wavelet in frequency plane

Syntax

[out] = pethat2d(kx,ky)

Description

This function computes the Aurell's 2D Pet Hat wavelet in frequency plane. That is, the wavelet given by

```
                     _
                    |   - cos^2(pi/2 * log2( |K| / 2^0.5 ) )
PSIHAT (kx,ky) = <                 if 1/2^0.5 <= |K| < 2*2^0.5
                    |   0          elsewhere
                    `_
```

where PSIHAT is the Fourier transform of PSI;

```
K = (kx,ky);
```

Notice that we have scaled this wavelet from this original formulation to share the same maximum than the Mexican Hat (in $|K| = 2^{0.5}$)

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = pethat2d(kx,ky,6,1);
>> imagesc(wav);
```

References

P. Frick and Al., "Scaling and Correlation Analysis of Galactic Images", arXiv:astro-ph/0109017 v1, 3 Sep 2001

E. Aurell and Al, 1994, Physica D, 72, 95

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/pethat2d.m

pgmread

Read an image in the raw PGM format (8bits).

Syntax

[mat] = pgmread(fname)

Description

Input Data

fname [STRING]: the name of the file to read.

Output Data

out [ARRAY]: a matrix containing the image.

Example(s)

```
>>
```

References

Matthew Dailey's readpgm.m file (see http://ai.ucsd.edu/Tutorial/matlab.html)

See Also

pgmwrite

Location

tools/io/pgmread.m

pgmwrite

Write an image in the raw PGM format.

Syntax

pgmwrite(mat, fname)

Description

Write an image 'mat' in a file 'fname' in the raw PGM format.

Input Data

mat [ARRAY]: the origanl matrix. If this file is not in the uint8 format, it is converted.

fname [STRING]: the string containing the name of the file to write in the current dir.

Output Data

[]:

Example(s)

```
>>
```

References

Matthew Dailey's writepgm.m file (see http://ai.ucsd.edu/Tutorial/matlab.html)

See Also

pgmread

Location

tools/io/pgmwrite.m

phin

The scaling function associated to the pseudo-diff operator

Syntax

[out] = phin(k,n)

Description

this function computes the scaling function associated to the pseudo-diff operator k^n, n≥ 2 using the recursion formula:

```
phi_n(k) = 0.5*k^(n-2) + 0.5*(n-2)*phi_(n-2)(k)
```

Input Data

k [REAL MATRIX]: The radial frequency

n [INTEGER]: The order of the function (see formula above)

Output Data

out [REAL MATRIX]: the resulting function

Example(s)

References

See Also

iwpck2d, pseudiff,, wpck2d, yashow_wpck2d

Location

discrete/packet/2d/wpck_defs/phin.m

pmexican2d

Return the 2D mexican wavelet or scaling function packets

Syntax

out = pmexican2d(kx,ky,mode[,alpha])

Description

Return the 2D mexican wavelet or scaling function packets

Input Data

kx,ky [REAL MATRIX] the frequency plane

mode ['infinit'—'scaling'—'wavelet'—'dscaling'—'dwavelet']: specify the function to return through out. So, you can have the integrated scaling function ('scaling'), the wavelet packet ('wavelet'), the duals of these (respectively 'dscaling' and 'dwavelet') useful for the reconstruction, and finally the infinitesimal wavelet ('infinit'), that is the 2D mexican hat.

Output Data

out The frequency mask of the returned function.

Example(s)

References

See Also

gaussian2d, iwpck2d, mexican2d, mexican2d_ctr, pmexican2d, wpck2d, yashow_wpck2d

Location

discrete/packet/2d/wpck_defs/pmexican2d.m

pseudiff

Return the 2D pseudo differential operator k^n wavelet

Syntax

out = pseudiff(kx, ky, mode [, alpha])

Description

Return the 2D pseudiff wavelet or scaling function packets based on the recursive filter

```
phi_n(k) = 0.5*k^(n-2) + 0.5*(n-2)*phi_(n-2)(k)
```

(implemented in phin.m) So, the wavelet packet is

```
wpck = \sqrt(phi_n(k) - phi_n(alpha*k))
```

and the scaling function

```
wphi = \sqrt(phi_n(k))
```

Input Data

kx, ky [REAL MATRIX] the frequency plane

mode ['infinit'—'scaling'—'wavelet'—'dscaling'—'dwavelet']: specify the function to return through out. So, you can have the integrated scaling function ('scaling'), the wavelet packet ('wavelet'), the duals of these (respectively 'dscaling' and 'dwavelet') useful for the reconstruction, and finally the infinitesimal wavelet ('infinit'), that is the 2D mexican hat.

alpha [REAL]: the scale ratio defining the packet wavelet.

n [INTEGER]: the order of the pseudiff wavelet packet.

Output Data

out The frequency mask of the returned function.

Example(s)

References

P. Vandergheynst, "Ondelettes directionnelles et Ondelettes sur la Sphre", PhD Thesis, Louvain-la-Neuve, 1998.

See Also

gaussian2d, iwpck2d, mexican2d, mexican2d_ctr, phin, pmexican2d, wpck2d, yashow_wpck2d

Location

discrete/packet/2d/wpck_defs/pseudiff.m

regrep

Perform a substring replacement on string using regular expression.

Syntax

out = regrep(pattern,replace,string)

Description

This program performs the replacement of the substring 'pattern' into 'string' with 'replace'. The writting of the pattern follows the regular expression syntax. Remark: regrep uses the perl executable.

Input Data

pattern [STRING]: the pattern to search into string;

replace [STRING]: the replacing word;

string [STRING]: the string processed.

Output Data

out [STRING]: the resulting string.

Example(s)

```
>> regrep('([a-z])e','a','hello')
```

Location

tools/misc/regrep.m

rgray

This function return the reverse gray colormap

Syntax

[out] = rgray([n])

Description

This function returns the reverse gray colormap simply given by 1-gray. The number of gray level can optionnally be given through the n variable.

Input Data

n [INTEGER]: the number of gray level (64 by default)

Output Data

out [MATRIX]: the reverse gray colormap of nx3 size.

Example(s)

```
>> yashow(rand(128,128),'cmap','rgray');
```

References

See Also

yashow, yashow_cgt1d, yashow_cwt1d, yashow_cwt1dt, yashow_cwt2d, yashow_cwt3d, yashow_cwtsph, yashow_matrix, yashow_samcwt2d, yashow_spheric, yashow_timeseq, yashow_volume, yashow_wpck2d

Location

tools/cmap/rgray.m

rmorlet2d

Compute the Real 2D Morlet Wavelet in frequency plane

Syntax

[out] = rmorlet2d(kx,ky,k_0,sigma)

Description

This function computes the real 2D Morlet wavelet in frequency plane. That is, the wavelet given by

```
PSIHAT (kx,ky) = exp( - sigma^2 * ( (kx - k0).^2 +  (epsilon*ky).^2 ) / 2)
               + exp( - sigma^2 * ( (kx + k0).^2 +  (epsilon*ky).^2 ) / 2)
```

where PSIHAT is the Fourier transform of PSI. This wavelet depends of two parameters: k0 and sigma. It is not truly admissible but for sigma*K_0>5.5, it is considered numerically admissible. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

k_0, sigma, epsilon [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = rmorlet2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/rmorlet2d.m

samcwt2d

Compute the 2D CWT scale-angle measure

Syntax

sam = samcwt2d(fimg, wavname, scales, angles [, WaveletParameter] [,'Norm', NormValue])

sam = samcwt2d(fimg, wavname, scales, angles [,'WaveletOptionName', WaveletOptionValue] [,'Norm', NormValue])

Description

This function compute the scale-angle measure of an image through its Fourier transform 'fimg', that is the integration over all the position of the CWT of mat according the wavelet 'wavname'.

Input Data

fimg [COMPLEX MATRIX]: the Fourier transform of the image

wavname [STRING]: the name of the wavelet. See `yawtdir/continuous/2d/wave\_defs` to see which wavelets are available.

scales, angles [REAL VECTORS]: the scale and the angle vectors.

Output Data

out [YAWTB OBJECT]: the scale-angle measure.

Example(s)

References

See Also

Location

continuous/2d/samcwt2d.m

sarcwt2d

Compute the CWT scale (angle) representation of mat

```
Implement this program without cwt2d, directly with the
scalar product of mat and the wavelet.
This will of course decrease the computing time!
```

Syntax

coeff = sarcwt2d(mat,wavname,scales,angles [,'x',x,'y',y] [,'Show'] ['ShowIn',resfig] [,'Fig',fig] [,'WaveletParameterName',WaveletParameterValue'])

Description

Compute the scale angle continuous wavelet representation at the point (x:col, y:row) of the matrix mat. These coordinates can be either given to the program through with the ('x', x) and ('y', y) options, or directly selected with the mouse on a figure of mat. This figure is either display by the call of sarcwt2d, or taken in an existent figure if the fig option is set (through the command ('Fig',n) with 'n' the index of the current figure). As in cwt2d, the wavelet parameters can be introduced with the usual syntax (see cwt2d for explanations).

Input Data

mat [MATRIX]: the input image.

wavname [STRING]: the name of the wavelet. See cwt2d for an exact list of the available wavelets.

scales [REAL VECTOR]: the vector of scales.

angles [REAL VECTOR]: the vector of angles.

x, y [REAL]: the coordinates of the point (x:col, y:row), that is the fixed position b of the wavelet transform.

fig [INTEGER]: the index of the figure where to select the position b if not given through x and y.

WaveletParameterValue [MISC]: the parameters of the wavelet. See the corresponding wavelet code for explanation.

Output Data

out [VECTOR]: the vector of wavelet coefficient according scales.

Example(s)

References

See Also

cwt2d, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/sarcwt2d.m

sdog1d

1D Scaling Difference Of Gaussian wavelet in the frequency domain

Syntax

[out] = sdog1d(k,alpha)

Description

This function returns `out` the 1D Scaling Difference Of Gaussian wavelet in the frequency domain defined in that domain by:

```
out = exp( - k.^2 ) - exp ( - alpha^2 * k.^2)
```

Input Data

k [REAL MATRIX]: The frequency. You can use the YAWTBfunction `vect` to create it (see example below).

alpha [REAL SCALAR]: The wavelet parameter.

Output Data

out [REAL MATRIX]: The wavelet in frequency.

Example(s)

```
>> step = 2*pi/128;
>> k = -pi : step : (pi-step);
>> wav = sdog1d(freqs,6);
>> plot(k,wav);
```

Note that the two first lines are implemented by the YAWTB`vect` function:

```
>> k = vect(-pi,pi,128,'open');
```

References

See Also

cwt1d, dgauss1d, morlet1d, sdog1d, vect

Location

continuous/1d/wave_defs/sdog1d.m

sdog2d

Compute the 2D Scaling difference of Gaussian Wavelet

Syntax

[out] = sdog2d(kx,ky,alpha)

Description

This function computes the 2D Scaling difference of Gaussian Wavelet in frequency plane. This wavelet given by

```
PSIHAT (kx,ky) = exp( - K.^2/2 ) - exp ( - alpha^2 * K.^2/2)
```

where: PSIHAT is the Fourier transform of PSI and K = (kx,ky). This wavelet depends of the alpha parmeter. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = sdog2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/sdog2d.m

setyawtbprefs

Set a YAWtb preference

Syntax

setyawtbpref(property_name, value)

Description

Set the property named 'property_name' in YAWtb preference to 'value' This property must exist.

Input Data

property_name [STRING]: The name of the property

value [MISC]: The new value

See Also

getyawtbprefs

Location

tools/misc/setyawtbprefs.m

sphcg

Spherical Conjugate Gradient Reconstruction

Syntax

[out, err] = sphcg(wav, n [,'ref', ref], [,'approx', g])

Description

Iterativeley rebuild a spherical signal from the wavelet coefficients of a spherical frame.

Input Data

wav [STRUCT]: the frame coefficients obtained by fwtsph.

n [INT]: number of iteration to perform in the algorithm.

ref [REAL MATRIX]: the original spherical data if available. It is used to compute the rebuilding error at each iteration.

g [REAL MATRIX]: approximated rebuilding of the spherical data obtained with ifwtsph if available.

Output Data

out [REAL MATRIX]: approximated rebuilding the spherical data after n iteration of the conjugated gradient algorithm.

err [REAL VECTOR]: convergence error at each iteration.

Example(s)

```
>> load sector
>> mat=mat(1:2:end,1:4:end);
>> whos
>> yashow(mat,'spheric','fig',1);
>> wav=fwtsph(mat,'dog');
>> g=ifwtsph(wav);
>> yashow(g,'spheric','fig',2);
>> nmat=sphcg(wav,5,'ref',mat,'approx',g); %% It's time to take two pills of Temesta ;-)
>> yashow(nmat,'spheric','fig',3);
```

References

See Also

Location

discrete/frames/sphere/sphcg.m

sphgrid

Spherical grid

Syntax

[phi, theta] = sphgrid(nth [,nph] [,'withpoles'])

Description

Compute a equi-angular spherical grid. By default, this one is given by (theta_i,phi_j), i=0..nth-1, j=0..nph-1 with theta_i = (2*i+1)*pi/nth phi_j = j*2*pi/nph The size of this grid is nth*nph

If 'withpoles' is set, the grid is then defined by (theta_i,phi_j), i=0..nth, j=0..nph-1 with theta_i = i*pi/nth phi_j = j*2*pi/nph The size is then equal to (nth+1)*nph.

Input Data

nth [INT]: the number of angles theta

nph [INT]: the number of angles phi

'withpoles' [BOOL]: activate the second grid forming method described above.

Output Data

phi [REAL MATRIX]: the values of phi increasing with the column index only.

theta [REAL MATRIX]: the values of theta increasing with the row index.

Example(s)

```
>> [phi,theta] = sphgrid(256);
>> f=exp(-tan(theta/2).^2).*cos(6*phi);
>> yashow(f,'spheric','relief');
```

See Also

cwtsph,, fcwtsph, fst,, ifst,

Location

tools/misc/sphgrid.m

sphrichardson

Syntax

[] = sphrichardson()

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

discrete/frames/sphere/sphrichardson.m

sphweight

Clenshaw-Curtis weights for spherical quadrature on grid (Odd x Even)

Syntax

w = sphweight(Nth, Nph [,'nopoles'])

Description

Input Data

Nth [INT]: the height of the grid

Nph [INT]: the width of the grid.

'nopoles' [BOOL]: use the square grid with no poles described in [2]

Output Data

w [REAL MATRIX]: the Clenshaw-Curtis weights or the quadrature weight of the grid obtained with the `nopoles` flag (see above) of size (Nth x Nph).

Example(s)

```
>>
```

References

[1] J. P. Imhof, "On the Method for Numerical Integration of Clenshaw and Curtis", Numerische Mathematik 5, 138-141 (1963)

[2] J.R. Driscol and D. M. Healy. Computing fourier transforms and convolutions on the 2-sphere. Advances in Applied Mathematics, 15 :202-250, 1994.

See Also

fftshp,, ifftsph,, yaspharm

Location

tools/misc/sphweight.m

sqdog2d

Compute the square of 2D Scaling difference of Gaussian Wavelet

Syntax

[out] = sqdog2d(kx,ky,alpha)

Description

This function computes the square of 2D Scaling difference of Gaussian Wavelet in frequency plane. This wavelet given by

```
PSI(x,y)       = sdog2d(x,y,alpha).^2;
PSIHAT (kx,ky) = (1/2) * exp( - K.^2/4 ) ...
         + 1/(2*alpha^2) * exp ( - alpha^2 * K.^2 / 4) ...
         - 2/(alpha^2 + 1) * exp ( - alpha^2 * K.^2 / ...
                                   (2*(alpha^2 + 1)) )
```

where: PSIHAT is the Fourier transform of PSI and K = (kx,ky). This wavelet depends of the alpha parmeter. This function is used by the cwt2d routine which compute continuous wavelet transform in 2D.

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

order, sigma [REAL SCALARS]: The wavelet parameters.

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = sdog2d(kx,ky,6,1);
>> imagesc(wav);
```

References

See Also

cwt2d, meshgrid, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/sqdog2d.m

the3dL

Generate a the 3D version of the L shape.

Syntax

[out] = cube(n [,ncube])

Description

Input Data

n [INTEGER]: The size of the cubic domain where the L shape is defined.

ncube [INTEGER]: the size of the cube containing the L set to n/2 by default.

Output Data

out [ARRAY]: the volume describing the L shape (1 inside & 0 outside)

Example(s)

```
>> theL = the3dL(64);
>> yashow(theL);
```

References

See Also

Location

sample/3d/the3dL.m

theL

A 2D academic example: the letter L in a NxN matrix

Syntax

out = theL([N])

Description

Return a binary square matrix of N×N size with the letter L inside.

Input Data

N [INTEGER] the width of the returned square matrix (default 64)

Output Data

out [BINARY MATRIX] the resulting matrix.

Example(s)

```
>> mat = theL;
>> imagesc(theL);
>> colormap(gray);
```

References

See Also

Location

sample/2d/theL.m

thedisk

Create a disk in a square image.

Syntax

[out] = thedisk([n] ['Center', center] [,'Radius', radius] [,'Smooth', smooth])

Description

Input Data

n [INTEGER]: the size of the image (default 64);

center [REAL VECTOR]: specify the center of the disk in (x,y) coordinates knowing that the complete image is the domain [-1,1]x[-1,1]. 'center' is set to the origin [0 0] by default.

radius [REAL]: the radius of the disk relatively to the size of the output matrix (0: nothing, 1: the whole matrix).

smooth [REAL] specify if the disk must be smooth on the edge.

Output Data

out [BINARY MATRIX]:

Example(s)

```
>> mat= the disk;
>> yashwo(mat);
```

References

See Also

Location

sample/2d/thedisk.m

vect

Compute vector of several shape (linear,exponential, ...)

Syntax

[out] = vect(deb, fin, N [, typ] [, closeness])

Description

This function compute several kind of N points vectors of beginning 'deb' and end 'end'. It resumes the action of 'linspace' and 'logspace' and add some extra options.

Input Data

deb, fin [REALS] extremities of the vector

N [INTEGER] number of points inside the vector

typ [STRING] the type of the vector. It can be:

- **'log'** logarithmic spaced;
- **'sqr'** squared spaced;

closeness [STRING]: The type of interval to implement. Allowed type are:

- **'close'** linear spaced with step = (fin-deb)/(N-1)
- **'open'—'ropen'** linear spaced with step = (fin-deb)/N. Useful for cases where 'fin' is identified to 'deb' (e.g. frequencies with pi and -pi). It modelizes the right open interval [deb,fin[
- **'lopen'** same than 'open' but modelizes the left-open interval]deb,fin].
- **'rlopen'—'lropen'** modelizes the open interval]deb,fin[.

Output Data

out [REAL VECTOR] output vector.

Example(s)

References

See Also

linspace, logspace

Location

tools/misc/vect.m

whatin

Return information about a matrix

Syntax

whatin mat whatin(mat)

Description

Return the major information of a matrix mat.

Input Data

mat [MATRIX—STRING]: the input matrix or its name in the matlab workspace. Note: Don't use a name of variable inside a mfile because 'whatin' looks for the variable in the 'base' workspace.

Example(s)

```
>> whatin(rand(5,5) + i*rand(5,5))
```

References

See Also

Location

tools/misc/whatin.m

wheel2d

Compute the 2D Wheel wavelet in frequency plane

Syntax

[out] = pethat2d(kx,ky)

Description

This function is based on the Aurell's 2D Pet Hat wavelet in frequency plane. It is given by:

```
                      -
                     |   cos^2(pi/2 * log( |K| ) / log(sigma) )
PSIHAT (kx,ky) = <                  if 1/sigma <= |K| < sigma
                     |   0            elsewhere
                     `-
```

where PSIHAT is the Fourier transform of PSI;

```
K = (kx,ky);
```

Input Data

kx, ky [REAL MATRICES]: The frequency plane. Use meshgrid to create it.

sigma [REAL]: The spread of the wavelet in frequency (default, sigma=$2^{0.5}$)

Output Data

out [REAL MATRIX]: The wavelet in frequency plane.

Example(s)

```
>> step = 2*pi/128;
>> [kx,ky] = meshgrid( -pi : step : (pi-step) );
>> wav = wheel2d(kx,ky);
>> yashow(wav);
```

References

See Also

cwt2d, meshgrid, pethat2d, samcwt2d, sarcwt2d, yashow_cwt2d, yashow_samcwt2d

Location

continuous/2d/wave_defs/wheel2d.m

wpck2d

Compute the packet wavelet transform (details and approximation)

Syntax

out = wpck2d(fimg,wpckname [,'PckParameterName',PckParameterValue]...)

Description

This function computes the (isotropic) wavelet packets of an image given through its Fourier transform fimg. The approximation is calculated from the last scale of 'scales' (recorded inside out.approx) and all details define a volume 'out.details' within each z slide corresponds to a fixed scale.

Input Data

fimg [COMPLEX MATRIX]: The FFT of the original image.

wpckname [STRING]: the name of the wavelet/scaling packet to use (see pck_defs subdir for available packets).

scales [POSITIVE REAL VECTOR]: the vector of scales.

PckParameterValue [MISC]: the value of a precised packet parameter. See the code of the corresponding wavelet packet for explanations

Output Data

out [STRUCT] a yawtb object such that:

- out.approx [COMPLEX MATRIX]: contains the approximation at the last scale of the image.
- out.details [COMPLEX VOLUME]: contains the details at each scale (zslides correspond to fixed scale.
- out.scales [POSITIVE REAL VECTOR]: the scales.
- out.extra [LIST]: the extra parameter given to wpckname.
- out.wpckname [STRING]: the name of the packet.
- out.type [STRING]: the name of the mfile, that is wpck2d.

Example(s)

```
>> mat = theL;
>> tmat = fft2(mat);
>> scales = vect(0.5,10,12);
>> pwav = wpck2d(tmat,'pmexican',scales);
>> figure; imagesc(real(pwav.approx)); // TODO yashow(pwav)!!!
>> figure; imagesc(real(pwav.details(:,:,1)));
>> figure; imagesc(real(pwav.details(:,:,3)));
```

References

[1] Bruno Torrsani. Analyse Continue par Ondelettes, ???

See Also

iwpck2d, pmexican2d, theL

Location

discrete/packet/2d/wpck2d.m

yabeta

Profile function beta

Syntax

out = yabeta(t, m)

Description

Compute the profile function beta of order m (described in [1]) such that: $beta(t, m)^2 + beta(1 - t, m)^2 = 1 for tin [-1, 1]$
$beta(t, m + 1) = beta(sin(t * pi/2), m)$
$beta(t, 0) = sin((1 + t) * pi/4)$

Input Data

t [REAL VECTOR]: the positions on which to compute the beta function;

m [INT]: the order of the beta function

Output Data

out [REAL VECTOR]: the beta function computed on

Example(s)

```
>> t = vect(-2,2,128);
>> plot(t,yabeta(t,0),'b',t,yabeta(t,1),'r');set(gca,'ylim',[-.5,1.5]);
>> hold on;plot(t,yabeta(t,2),'g',t,yabeta(t,3),'k');hold off
```

References

[1] S. Mallat. A Wavelet Tour of Signal Processing.

See Also

yaspline

Location

tools/misc/yabeta.m

yachirp

Compute a typical chirp cosine

Syntax

[out] = yachirp([n])

Description

This function computes another typical chirp cosine on n points (the default is 256)

Input Data

n [INTEGER]: the number of points.

Output Data

out The output chirp.

Example(s)

```
>> ch = yachirp(256)
>> plot(ch)
```

References

See Also

Location

sample/1d/yachirp.m

yademo

Execute the demo associates to a yawtb file

Syntax

yademo(yafile) yademo yafile

Description

Input Data

yafile [STRING]: the name of this file.

Location

tools/help/yademo.m

yadiro

Dilation and rotation on grids of positions

Syntax

[nX, nY] = yadiro(X,Y,sc,ang [,'freq'])

Description

`yadiro` performs a dilation of 'sc' and a rotation of 'ang' on two grids 'X' and 'Y' build by meshgrid. So, $nX = sc^-1 * (cos(ang) * X - sin(ang) * Y)$
$nY = sc^-1 * (sin(ang) * X + cos(ang) * Y)$

This functions is used in almost all the 2D transforms of the yawtb.

Input Data

X [REAL MATRIX]: the matrix of 'x' (horizontal) positions;

Y [REAL MATRIX]: the matrix of 'y' (vertical) positions;

sc [REAL SCALAR]: the scale of dilation (must be positive);

ang [REAL SCALAR]: the angle of the rotation in radian;

'freq' [BOOLEAN]: inverse the dilation to correspond to its frequency action.

Output Data

nX [REAL ARRAY]: the dilated and rotated horizontal positions;

nY [REAL ARRAY]: the dilated and rotated vertical positions;

Example(s)

```
>> [x,y] = meshgrid(vect(-1,1,3), vect(-1,1,3))
>> [nx,ny] = yadiro(x,y,2,pi/2)
```

References

See Also

meshgrid

Location

tools/misc/yadiro.m

yahelp

Display the help associated to a yawtb file.

Syntax

yahelp yafile yahelp(yafile [,section])

Description

YAHELP display a userfriendly help of any yawtb function.

Input Data

yafile [STRING]: the name of this file.

section [SET OF STRING]: tells to yahelp which part of the help must be displayed. Type yahelp([]) to obtain a complete list of the available sections.

Example(s)

```
>> yahelp yashow
>> yahelp yashow syntax
```

Location

tools/help/yahelp.m

yahist

Computes 1D and 2D histograms.

Syntax

[out[,l]] = yahist(V,N [,'min',m] [,'max',M]) [out[,lx,ly]] = yahist(Vx,Vy,N [,'minx',mx] [,'miny', my] ... [,'maxx',Mx] [,'maxy', My])

Description

This function computes the 1D or 2D histograms of vectors or matrices data.

Input Data

V [VECTOR—ARRAY]: a vector or an array of values (an image) to transform into an histogram;

Vx,Vy [VECTORS—ARRAYS]: in the case of 2D histogram, vx and vy are two vectors or arrays of same sizes which values describe respectively the x and the y coordinates of the 2D histogram;

N [INTEGER—VECTOR]: N contains the number of desired levels into the output histogram. in the 2D case, N could be either a vector of length 2 containing the number of levels in x and y coordinates, or an integer assuming that these numbers of levels are equal.

m,M,mx,my,Mx,My [REALS]: user defined range of value instead of those defined by minimum and maximum of data.

Output Data

out [VECTOR—MATRIX]: a 1D or a 2D histogram, that is a vector or a matrix;

l,lx,ly [VECTORS]: the range of data in 1D, or in 2D for the x or y coordinates.

Example(s)

```
>> load woman;
>> [h,l] = yahist(X,20);
>> plot(l,h);
```

References

See Also

Location

tools/misc/yahist.m

yaload

Load the YAWtb path into memory

Syntax

yaload(['debug'])

Description

This function place all the YAWtb path into memory

Input Data

'debug' [BOOLEAN]: set on the debug mode and display the information about all the path addings.

Location

yaload.m

yamake

Create all the mex files needed by yawtb

Syntax

yamake(['debug'])

Description

This function creates all the mex files needed by yawtb. It parses all the yawtb subdirs and compiles C files like described into a file called ".to_compile" related to each subdir.

Input Data

'debug' [BOOLEAN]: set on the debug mode and display the information about all the compilations.

Location

yamake.m

yamax

Determines the regional maxima of a real matrix

Syntax

[out] = yamax(mat ['connect',connexion] [,'pos'] [,'thresh',T] ['dir',angle])

Description

This function determines the local maxima of the real matrix 'mat' with a specified pixel connection (4 or 8 neighbours).

Input Data

mat [REAL MATRIX]: the input matrix;

connexion [INTEGER]: the connection to use: 4 or 8.

'pos' [BOOLEAN]: tell to yamax to give the list of the row and column positions of the maxima as result.

'strict' [BOOLEAN]: if you want the strict maximums, that is, strictly greater than their neighbours.

T [REAL]: a threshold between 0 and 1 for the maxima detection. Only maxima greater than minmat + T*(maxmat - minmat) are conserved.

angle [REAL—REAL MATRIX]: determine the maxima of mat in the direction angle. This angle is either a scalar, or a matrix giving one angle for each element of mat. In the two cases, each element of angle must be inside the interval [-pi/2 pi/2].

Output Data

out [MISC]: If 'pos' is not specified, out is a binary matrix with 1 on maxima and 0 elsewhere. If 'pos' is specified, out is a struct array, with fields

- **out.i** the row positions of the maxima;
- **out.j** the column positions of the maxima;
- **out.lin** the positions in a linear mode such that, mat(out.lin) gives all the maxima of mat.

Example(s)

The center of a Gaussian

```
>> [x,y]=meshgrid(vect(-1,1,5));
>> g=exp(-x.^2-y.^2);
>> yamax(g)
```

Maxima of this Gaussian in the directions of 0 and pi/4 radian

```
>> yamax(g,'dir',0)
>> yamax(g,'dir',pi/4)
```

References

See Also

Location

tools/misc/yamax.m

yamse

Mean square error between signals or images

Syntax

mse = yamse(psig, nsig)

Description

This function computes the Mean Square Error (MSE) between a pure and a noisy signal/image. The formula used is
$MSE = |psig - nsig|_2$

Input Data

psig [REAL VECTOR—REAL MATRIX]: real vector or real matrix giving the base signal, generally the signal without noise

nsig [REAL VECTOR—REAL MATRIX]: real vector or real matrix giving the noisy signal.

Output Data

mse [REAL SCALAR]: The Mean Square Error

Example(s)

```
>> pmat = theL(256); yashow(pmat,'square');
>> nmat = pmat + randn(size(pmat))/5; yashow(nmat,'square');
>> yamse(pmat, nmat)
```

Location

tools/misc/yamse.m

yapbar

This function create a figure with a progress bar.

Syntax

oyapbar = yapbar([], lim) oyapbar = yapbar(oyapbar, iter) oyapbar = yapbar(oyapbar, 'Close');

Description

This function create a progress bar inside a figure. This is useful inside long computatino program.

Input Data

oyapbar [STRUCT]: progress bar object returned by the initialization with oyapbar set to [].

lim [INTEGER]: the limit of the progress bar.

iter [INTEGER— '++' — '–']: the iterator. In its numerical form, it must be greater than 0 and lesser than the original lim.

'Close' [BOOLEAN]: tell to yapbar to close the progress bar and set oyapbar to [].

Output Data

oyapbar [STRUCT]: the progress bar object.

Example(s)

Initialization of the yapbar object (10 is the number of steps):

```
>> oyap = yapbar([],10);
```

Progression of the yapbar:

```
>> for k=1:10, ...
   oyap = yapbar(oyap,'++'); ...
   end
```

Closing of this yapbar:

```
>> oyap = yapbar(oyap,'close');
```

References

See Also

Location

tools/display/yapbar.m

yapsnr

Peak Signal to Noise Ratio of signals or images

Syntax

psnr = yapsnr(psig, nsig)

Description

This function computes the Peak Signal to Noise Ratio (PSNR) between a pure and a noisy signal/image. The formula used is

$PSNR = 10 log10(M_p sig^2/MSE)$

where $M_p sig$ is the maximum absolute value of psig, and MSE is the Root Mean Square Error given by the square root of

$MSE = Sum_n (psig(n) - nsig(n))^2/N$

for vectors, and by the square root of

$MSE = Sum_n Sum_m (psig(n,m) - nsig(n,m))^2/(N * M)$

for matrices.

Notice that if 'psig' is of uint8 data type, $M_p sig$ is automatically set to 255.

Input Data

psig [REAL VECTOR—REAL MATRIX]: real vector or real matrix giving the base signal, generally the signal without noise

nsig [REAL VECTOR—REAL MATRIX]: real vector or real matrix giving the noisy signal.

Output Data

psnr [REAL SCALAR]: The Peak Signal to Noise ratio.

Example(s)

```
>> pmat = theL(256); yashow(pmat,'square');
>> nmat = pmat + randn(size(pmat))/5; yashow(nmat,'square');
>> yapsnr(pmat, nmat)
```

Location

tools/misc/yapsnr.m

yapuls

Pulsation vector

Syntax

puls = yapuls(npuls)

Description

`yapuls` returns a pulsation vector `puls` of size `npuls` which is the concatenation of two subvectors whose elements are respectively in $[0, \pi)$ and $[-\pi, 0)$. Useful function when computing wavelets directly in the Fourier domain.

Input Data

npuls [REAL SCALAR]: length of the pulsation vector

Output Data

puls [REAL VECTOR]: the pulsation vector

Example(s)

```
>> puls5 = yapuls(5)
>> puls6 = yapuls(6)
```

prints and returns a 5-length, and a 6-length pulsation vector.

References

See Also

cwt1d, cwt2d, cwt3d, vect

Location

tools/misc/yapuls.m

yapuls2

Pulsation matrix

Syntax

[pulsx,pulsy] = yapuls2(npulsx [,npulsy]) [pulsx,pulsy] = yapuls2(npuls)

Description

`yapuls2` returns two pulsation matrices `pulsx` and `pulsy` of size `npulsx`*`npulsy` such that, each row of `pulsx` or each column of `pulsy` is the concatenation of two subvectors whose elements are respectively in $[0, \pi)$ and $[-\pi, 0)$. Useful function when computing 2D wavelets directly in the Fourier domain.

Input Data

npulsx [REAL SCALAR]: length of the x-pulsation vector

npulsy [REAL SCALAR]: length of the y-pulsation vector equals to `npulsx` by default.

npuls [REAL VECTOR]: 2-length vector containing the values of npulsx and npulsy.

Output Data

pulsx [REAL MATRIX]: the x-pulsation matrix

pulsy [REAL MATRIX]: the y-pulsation matrix

Example(s)

```
>> [pulsx,pulsy] = yapuls2(5)
>> [pulsx,pulsy] = yapuls2([5 4])
>> [pulsx,pulsy] = yapuls2(5,4)
```

prints and returns a 5x5-size, and two 5x4-size pulsation matrix.

References

See Also

cwt1d, cwt2d, cwt3d, vect

Location

tools/misc/yapuls2.m

yasave

save and ask for informations to include in data

Syntax

yasave FILENAME data1 data2

Description

This function saves data exactly as the builtin save matlab function does, but in addition, prompts for a little explanation introduce in the data by the variable yasave_info.

Input Data

FILENAME [STRING]: the name of you file as for matlab save function

Example(s)

```
>> yasave something\_to\_delete
```

References

See Also

Location

tools/misc/yasave.m

yashow

Display the result of any transform defined in YAWTB

Syntax

yashow(yastruct ['Fig',fig] [,'Mode',mode [,ModeParam]] [,'CMap',cmap] ... [,'Square'] [,'Equal'] [,'HistEq'] [,'Normalize'] ... [,'Contour' [,nlevel]] ... [,'Contour' [,vlevel]] ... [,'Surf'] ... [,'Spheric'[,'Relief'[,'Ratio',ratio]]] ... [,'maxi'[,T]] [,'mini'[,T]])

Description

The aim of this function is to simplify the use of the Matlabvisualization tools for any YAWTBresult coming, for instance, from `cwt1d`, `cwt2d` and `cwtsph`.

Input Data

yastruct [STRUCT]: the result of a particular YAWTBtransform.

fig [POSITIVE INTEGER]: specifies the figure in which to display the input.

mode ['abs','angle', 'essangle', 'real','imag']: is a string which defines the display mode. Available modes for the data representation are:

- **'abs'** : the absolute value;
- **'angle'** : the complex argument;
- **'essangle'** : the essential argument, that is the argument thesholded by default at 1% of the modulus. You may change this value by putting a real number between 0 and 1 after 'essangle' in the yashow call.
- **'real'** : the real part;
- **'imag'** : the imaginary part.

cmap [STRING]: define the colormap to use in the display, e.g. 'jet' [default], 'hsv', 'gray'.

'Square' [BOOLEAN]: specifies if yashow must display a matrix like a square

'Equal' [BOOLEAN]: specifies if yashow must keep the square shape of the pixels (dx==dy).

'HistEq' [BOOLEAN]: specifies if yashow must equalize histogram of the display matrix.

'Normalize' [BOOLEAN]: specifies if the data must be normalized to [-1,1]. This can be useful in 1D, if you plot several signal with hold on and you are interested in their shape.

'Contour' [BOOLEAN]: display the contour of the matrix. Possible modifiers are:

- **nlevel** [INTEGER]: the number of levels to display;
- **vlevel** [1x2 VECTOR]: a vector containing twice the curve level to display, e.g. [0 0] to show the curve level of zero height.

'Surf' [BOOLEAN]: display the matrix as a 3D surface.

'maxi' [BOOLEAN]: specifies if crosses representing maxima must be added to the displayed image.

'mini' [BOOLEAN]: specifies if circles representing minima must be added to the displayed image.

T [REAL]: gives an eventual threshold for the maxima or the minima displayed by the two preceeding options. T belongs to the interval 0¡=T¡=1. The maxima shown are those which have a value greater than MIN + T*(MAX-MIN), where MIN is the global minimum of the data and MAX its global maximum. Similarly, the minima must be below MAX - T*(MAX-MIN) to be represented.

'Spheric' [BOOLEAN]: (for matrix only !) Specifies if the matrix must be mapped onto a sphere (useful for cwtsph & al.).

The suboptions of this mode are :

'Relief' [BOOLEAN]: the mapping is done in relief, that is, the values of the input matrix correspond to the variation of spherical radius around 1. The maximum absolute value of this matrix is set by default to 1/3 of the sphere radius. This can be changed by the ratio variable.

ratio [DOUBLE]: set the ratio between the highest absolute value of the input matrix and the sphere radius.

Example(s)

In its easiest form, yashow can be used like this:

```
>> [x,y] = meshgrid(-64:64);
>> square = max(abs(x), abs(y)) < 20;
>> fsquare = fft2(square);
>> wsquare = cwt2d(fsquare,'cauchy',2,0);
>> yashow(wsquare);
```

So, the absolute value the square 2D wavelet transform is shown. The last line can be change by

```
>> yashow(wsquare,'Mode','angle','CMap','gray(40)');
```

to display the argument of this transform in the colormap 'gray' of 40 levels.

See Also

cgt1d, cwt1d, cwt2d, cwtsph, wpck2d, yashow

Location

tools/display/yashow.m

yashow_cgt1d

Display the result of `cgt1d`. Automatically called by `yashow`!

Syntax

cgt1d_yashow(yawres[,'thresh',thresh])

Description

This function displays the result of the `cgt1d` function. It is automatically called by `yashow` with respect to the type of `yawres`, that is, the name of `yawres.type`.

Input Data

yawres [YAWTB OBJECT]: the input structure coming from `cgt1d`.

thresh [REAL SCALAR $\in [0, 100]$]: the thresholding of the displayed matrix. Default value is 5 %.

Output Data

Example(s)

```
>> load superpos
>> fsig = fft(sig);
>> freqs = 0.005: (0.12 - 0.005)/127: 0.12;
>> wsig = cgt1d(fsig, freqs, 'sigma', 50);
>> yashow(wsig);
```

References

See Also

cgt1d, gauss1d, yashow

Location

continuous/1d/yashow_cgt1d.m

yashow_cwt1d

Display the result of `cwt1d`. Automatically called by `yashow`!

Syntax

cwt1d_yashow(yawres[,'thresh',thresh] [,'edgeeffect'])

Description

This function displays the result of the `cwt1d` function. It is automatically called by `yashow` with respect to the type of `yawres`, that is, the name of `yawres.type`.

Input Data

yawres [YAWTB OBJECT]: the input structure coming from `cwt1d`.

thresh [REAL SCALAR $\in [0, 1]$]: the thresholding of the displayed matrix. Default value is 0.05 that is 5%.

'edgeeffect' [BOOL]: display the edge effects on the cwt.

Output Data

Example(s)

```
>> sig = yachirp;
>> sc = vect(4, 50, 128, 'log');
>> wav = cwt1d(fft(sig), 'morlet', sc);
>> yashow(wav, 'mode', 'angle');
```

References

See Also

cwt1d, dgauss1d, morlet1d, sdog1d, yashow

Location

continuous/1d/yashow_cwt1d.m

yashow_cwt1dt

Display the result of cwt1dt. Automatically called by `yashow`!

Syntax

cwt1dt_yashow(yawres[,'filter'])

Description

This function displays the result of the `cwt1dt` function which computes the 1D+T CWT of a space-time signal. It is automatically called by `yashow` with respect to the type of `yawres`, that is, the name (string) inside the `yawres.type` field.

Input Data

yawres [YAWTB OBJECT]: the structure returned by `cwt1dt`.

filter [BOOLEAN]: If this modifier is set, then the wavelet in wave-number/frequency domain is displayed.

Output Data

Example(s)

References

See Also

cwt1dt, mexican1dt, morlet1dt, yashow, yashow_cwt1dt

Location

continuous/1dt/yashow_cwt1dt.m

yashow_cwt2d

Display the result of cwt2d. Automatically called by yashow!

Syntax

cwt2d_yashow(yawres [,'filter' [,filter_type]])

Description

This function displays the result of the cwt2d function. It is automatically called by yashow with respect to the type of yawres, that is, the name of yawres.type.

Input Data

yawres [YAWTB OBJECT]: the input structure coming from cwt2d.

filter [BOOLEAN]: the presence of this modifier specify if we want to display the filter, that is, the wavelet in frequency.

filter_type ['freq'—'pos']: the type of filter to display: its frequency ('freq') or its positional ('pos') representation. By default, without any precision, the frequency part is drawn.

Output Data

Example(s)

References

See Also

Location

continuous/2d/yashow_cwt2d.m

yashow_cwt3d

Display the result of cwt3d. Automatically called by yashow!

Syntax

cwt3d_yashow(yawres [,'filter' [,filter_type]])

Description

This function displays the result of the cwt3d function. It is automatically called by yashow with respect to the type of yawres, that is, the name of yawres.type.

Input Data

yawres [YAWTB OBJECT]: the input structure coming from cwt3d.

filter [BOOLEAN]: the presence of this modifier specify if we want to display the filter, that is, the wavelet in frequency.

filter_type ['freq'—'pos']: the type of filter to display: its frequency ('freq') or its positional ('pos') representation. By default, without any precision, the frequency part is drawn.

Output Data

Example(s)

References

See Also

Location

continuous/3d/yashow_cwt3d.m

yashow_cwtsph

Display the result of cwtsph. Automatically called by yashow!

Syntax

cwtsph_yashow(yawres [,'fig',fig] [,'faces',N] [,'filter'] [,'relief'[,'ratio',ratio]])

Description

This function displays the result of the cwtsph function. It is automatically called by yashow with respect to the type of yawres, that is, the name of yawres.type. The data are mapped onto a sphere approximate by a N faces polygon (N=20 by default).

Input Data

yawres [YAWTB OBJECT]: the input structure coming from cwtsph.

fig [INTEGER]: the figure where to display result.

N [INTEGER]: change the number of faces for the sphere approximation.

filter [BOOLEAN]: the presence of this modifier specify if we want to display the filter, that is, the spherical wavelet.

relief [BOOLEAN]: this modifier change the display of the CWT (or its filter) from a simple spherical mapping to a true relief vision.

ratio [DOUBLE SCALAR]: in conjunction with the boolean 'relief', this parameter determines the ratio between the CWT highest value and the sphere radius. By default, it is set to 1/3.

Output Data

Example(s)

```
>> load world;
>> wav = cwtsph(mat,'dog',0.05,0);
>> yashow(wav);
```

References

See Also

cwtsph, fcwtsph, yashow_cwtsph

Location

continuous/sphere/yashow_cwtsph.m

yashow_matrix

display a matrix

Syntax

yashow_matrix(mat [,'x',x] [,'y',y] ... [,'CMap', cmap] ['Square'] ['Equal'] ... [,'Contour'[,n]] [,'Surf'] [,'HistEq'] ... [,'maxi' [,T]] [,'mini' [,T]] [,'Freq'])

Description

Specialization of yashow to the display of real matrices.

Input Data

mat [MATRIX]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

tools/display/yashow_matrix.m

yashow_samcwt2d

Display the samcwt2d results (internal use)

Syntax

msacwt2d_yashow(yawres)

Description

Input Data

Output Data

Example(s)

References

See Also

Location

continuous/2d/yashow_samcwt2d.m

yashow_spheric

Display a matrix onto a sphere

Syntax

yashow_spheric(mat, ['cmap', cmap] ... [,'faces', faces] [,'relief'] [,'ratio'])

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

tools/display/yashow_spheric.m

yashow_timeseq

Display a time sequence

Syntax

yashow_timeseq (seq [,'pause'])

Description

Input Data

[]:

Output Data

[]:

Example(s)

```
>>
```

References

See Also

Location

tools/display/yashow_timeseq.m

yashow_volume

Display a volume

Syntax

yashow_volume(vol [,'x', x] [,'y', y] [,'z', z] ... [,'square'] [,'equal'] ... [,'levels', levels] [,'color', color] [,'alpha', alpha])

Description

Input Data

[]:

Output Data

[]:

Example(s)

\>>

References

See Also

Location

tools/display/yashow_volume.m

yashow_wpck2d

Display the output of cwt2d. Automatically called by yashow!

Syntax

wpck2d_yashow(yawres [,'approx'] [,'details', index])

Description

Input Data

Output Data

Example(s)

References

See Also

Location

discrete/packet/2d/yashow_wpck2d.m

yasnr

Signal to Noise Ratio of signals or images

Syntax

snr = yasnr(psig, nsig)

Description

This function computes the Signal to Noise Ratio (SNR) between a pure and a noisy signal/image. The formula used is

$SNR = 20log10(std_psig/std_nsig)$

where std_v is the standart deviation of v given by

$std_v^2 = 1/Nsum_n(v[n] - mean_v)^2$
$mean_v = 1/Nsum_nv[n]$

if v is a vector and by

$std_ps^2 = 1/N^2sum_nsum_m(psig[n,m] - mean_ps)^2$
$mean_v = 1/N^2sum_nsum_mv[n,m]$

if v is a matrix.

Input Data

psig [REAL VECTOR—REAL MATRIX]: real vector or real matrix giving the base signal, generally the signal without noise

nsig [REAL VECTOR—REAL MATRIX]: real vector or real matrix giving the noisy signal.

Output Data

snr [REAL SCALAR]: The Signal to Noise ratio.

Example(s)

```
>> pmat = theL(256); yashow(pmat,'square');
>> nmat = pmat + randn(size(pmat))/5; yashow(nmat,'square');
>> yasnr(pmat, nmat)
```

Location

tools/misc/yasnr.m

yaspharm

Generate a spherical harmonic on a regular spherical grid

Syntax

Ylk = yaspharm(theta, phi, l, k)

Description

Input Data

theta [REAL MATRIX]: the grid of co-latitude angles.

phi [REAL MATRIX]: the grid of longitude angles.

l [INT]: the degree of the spherical harmonic.

k [INT]: the order of the spherical harmonic.

Output Data

Ylk [CPLX MATRIX]: the spherical harmonic computed on the spherical grid.

Example(s)

```
>> [phi, theta] = meshgrid( vect(0,2*pi,256,'open'), vect(0,pi,128));
>> Ylk = yaspharm(theta, phi, 6, 1);
>> yashow(Ylk, 'spheric', 'relief', 'mode', 'real');
>> yashow(Ylk, 'spheric', 'relief', 'mode', 'imag');
>> yashow(Ylk, 'spheric', 'relief', 'mode', 'abs');
```

References

[1] "Spherical Harmonics" on mathworld.com, http://mathworld.wolfram.com/SphericalHarmonic.html

See Also

Location

tools/misc/yaspharm.m

yaspline

Yet another cardinal B-spline computation

Syntax

out = yaspline(x, m)

Description

Compute the cardinal B-spline of order `m` with the recurrence formula $N(x,m) = x * N(x,m-1) + (m-x) * N(x-1,m-1)$

N(x,0) = 1 if x in [-1/2,1/2[0 elsewhere

Notice that yaspline recenters the resulting spline onto 0 instead of (m+1)/2.

Input Data

x [REAL VECTOR]: the vector of positions;

m [INTEGER]: the order of the spline (default: 2).

Output Data

out [REAL VECTOR]: the resulting spline

Example(s)

```
>> plot( yaspline(vect(-5,5,128), 6) );
```

References

Unser & al.

See Also

Location

tools/misc/yaspline.m

yasthresh

Softly threshold a matrix.

Syntax

[out] = yasthresh(mat,level [,'absolute'])

Description

Return the matrix 'mat' softly thresholded by 'level' percent.

Input Data

mat [VECTOR—MATRIX]: the input vector or matrix.

level [REAL]: If relative, the level of thresholding, that is 0<level<1, with 0 and 1 corresponding respectively to 0% and 100% of thresholding. If absolute, level is taken as a value.

'absolute' [BOOL]: determine if the level is relative or absolute.

Output Data

out [REAL VECTOR—REAL MATRIX]: the softly thresholded vector or matrix.

Example(s)

```
>> mat = rand(100,100);
>> nmat = yasthresh(mat,0.5);
>> figure; imagesc(mat);
>> figure; imagesc(nmat);
```

References

D. Donoho, "Denoising by soft-thresholding", IEEE Trans. Info. Theory, vol 43, pp. 613–627, 1995.

See Also

yathresh

Location

tools/misc/yasthresh.m

yastrfind

Find one string within another

Syntax

pos = yastrfind(str, substr)

Description

This function is only there for internal use in the YAWtb functions. It solves the problem between the existence of strfind or findstr matlab built-in functions. If your matlab doesn't recognize strfind, you have only to change the code of this file (check it, it's easy).

Input Data

str [STRING]: string where to search 'substr'

substr [STRING]: string to search in 'str'

Output Data

pos [INT VECTOR]: vector of positions of 'substr' in 'str' computing from the first character of 'substr'. 'pos' is empty if 'substr' is not found.

Example(s)

```
>> yastrfind('Hello world or hello planet?','or')
```

Location

tools/misc/yastrfind.m

yathresh

Hardly threshold a matrix.

Syntax

[out] = yathresh(mat,level,['absolute'])

Description

Return the matrix 'mat' hard thresholded by 'level'. This level can be realtive (in percent of maximum) or absolute.

Input Data

mat [MATRIX]: the input matrix;

level [REAL]: the level of thresholding. If relative, `level` is 0<level<1, with 0 and 1 corresponding respectively to 0% and 1% of thresholding. If absolute, `level` is considered as a value.

Output Data

out [MATRIX]: the thresholded matrix.

Example(s)

```
>> mat = rand(100,100);
>> nmat = yathresh(mat,0.5);
>> figure; imagesc(mat);
>> figure; imagesc(nmat);
```

References

See Also

Location

tools/misc/yathresh.m

yawopts

Return the options to give to a yawtb mfile (internal use)

Syntax

opts = yawopt(listopts,funcname)

Description

Return the options to give to a yawtb mfile (internal use) by comparing a list of parameters given in the list `listopts` with the default values returned by the 'funcname([])'.

The syntax of `listopts` is 'name_1', value_1, 'name_2', value_2, ... The syntax of the list returned by 'funcname([])' is the same with the following particularities:

- if a 'value_j' is a string '*name_l', it mean that this value is by default equal to this of 'name_l';
- if 'value_j' is a string identical to 'name_j', it means that 'name_j' is a boolean flag which is absent by default.

Input Data

listopts [CELL]: the list os the user parameter;

funcname [STRING]: the name of the function to test.

Output Data

opts [CELL]: the list of argument for funcname;

listopts [CELL]: the input `listopts` without the parameter of the function funcname.

See Also

getopts

Location

tools/misc/yawopts.m

yawtbprefs

Set all the preferences of YAWtb

Syntax

yawtbsetprefs()

Description

Determined all the user preferences of the YAWtb behaviour

See Also

getappdata, setappdata

Location

yawtbprefs.m

5 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR

A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and a brief idea of what it does.
`Copyright (C)` *yyyy name of author*

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published
by the Free Software Foundation; either version 2 of the License,
or (at your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software Foundation,
Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

`Gnomovision version 69, Copyright (C)` *year name of author*

```
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show
w’.
This is free software, and you are welcome to redistribute it under
certain conditions; type ‘show c’ for details.
```

The hypothetical commands @sampshow w and @sampshow c should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than @sampshow w and @sampshow c; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.
```

signature of Ty Coon`, 1 April 1989`
`Ty Coon, President of Vice`

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

6 GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise

Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ”or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A

copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.