cwave Documentation

Release 1.0

Davide Albanese

Mar 28, 2017

Contents:

1 Financial Contributions **3**
1.1 Quickstart 3
1.2 Examples 4
1.3 API 7

2 Indices and tables **21**

Python Module Index **23**

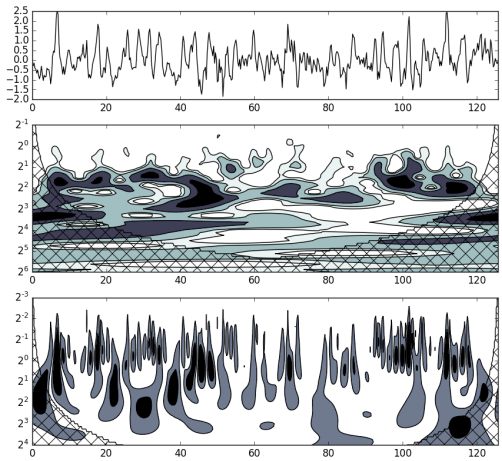
cwave is a Python2/3 library implementing the Continuous Wavelet Transform (CWT) using FFT as in *A Practical Guide to Wavelet Analysis* by C. Torrence and G.P. Compo. cwave is an open-source, GPLv3-licensed software.

- Homepage and documentation
- Issues
- Github page

CHAPTER 1

Financial Contributions

Computational Biology Unit - Research and Innnovation Center at Fondazione Edmund Mach



Quickstart

Install

The easiest way to install `cwave` is using pip:

```
$ pip install cwave
```

In Mac OS X/MacOS, we recommend to install Python from Homebrew.

You can also install `cwave` from source (with the command `python setup.py install`).

Using cwave

```
$ python
```

```
>>> import cwave
>>> import numpy as np
>>> x = np.random.normal(2, 1, 16)
>>> x
array([ 1.6960901 ,  3.27146653,  2.55896222,  2.39484518,  2.34977766,
        3.48575552,  1.7372688 , -0.21329766,  3.5425618 ,  3.34657898,
        1.54359934,  2.96181542,  2.43294205,  1.65980233,  3.44710306,
        1.69615204])
>>> dt=1
>>> T = cwave.cwt(cwave.Series(x, dt), cwave.DOG())
>>> sr = cwave.icwt(T)
>>> sr.x
array([ 1.64067578,  3.28517018,  2.78434897,  2.38949828,  2.58014315,
        3.52751356,  1.34776275, -0.41078628,  3.3648406 ,  3.56461166,
        1.7286081 ,  2.88596331,  2.40275636,  1.81964648,  3.28932397,
        1.7113465 ])
```

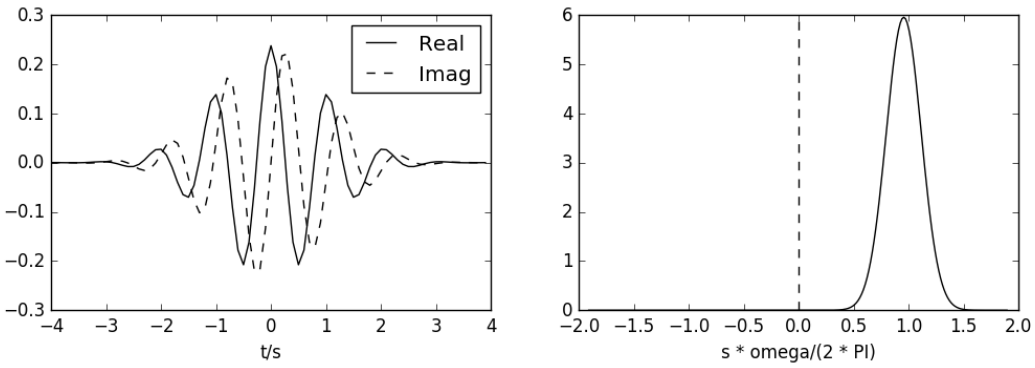
Examples

Wavelet functions

The example is located in `examples/wavelet_fun.py`.

```
# This code reproduces the figure 2a in (Torrence, 1998).
# The plot on the left give the real part (solid) and the imaginary part
# (dashed) for the Morlet wavelet in the time domain. The plot on the right
# give the corresponding wavelet in the frequency domain.
# Change the wavelet function in order to obtain the figures 2b 2c and 2d.

from __future__ import division

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

import cwave


wavelet = cwave.Morlet() # Paul(), DOG() or DOG(m=6)
dt = 1
scale = 10*dt
t = np.arange(-40, 40, dt)
omega = np.arange(-1.2, 1.2, 0.01)

psi = wavelet.time(t, scale=scale, dt=dt)
psi_real = np.real(psi)
psi_imag = np.imag(psi)
psi_hat = wavelet.freq(omega, scale=scale, dt=dt)

fig = plt.figure(1)
```

```
ax1 = plt.subplot(121)
plt.plot(t/scale, psi_real, "k", label=("Real"))
plt.plot(t/scale, psi_imag, "k--", label=("Imag"))
plt.xlabel("t/s")
plt.legend()

ax2 = plt.subplot(122)
plt.plot((scale*omega)/(2*np.pi), psi_hat, "k")
plt.vlines(0, psi_hat.min(), psi_hat.max(), linestyles="dashed")
plt.xlabel("s * omega/(2 * PI)")

plt.show()
```



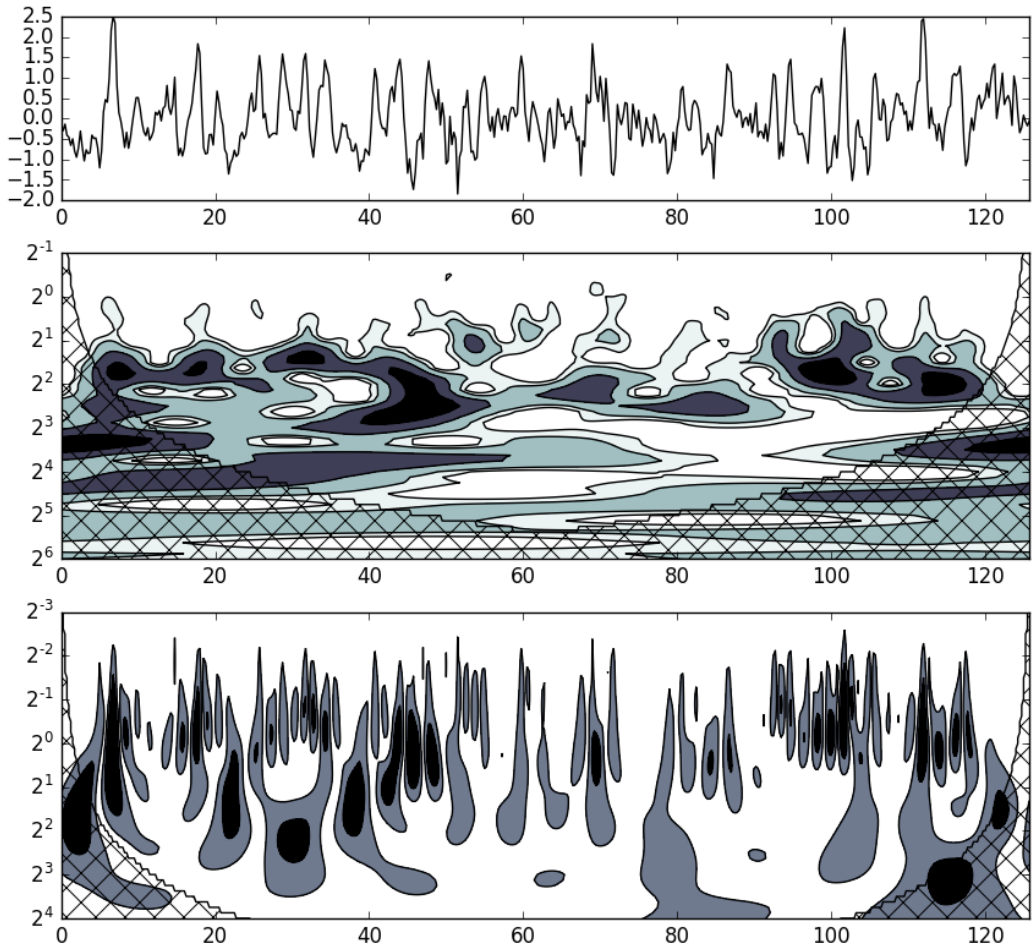
Nino3 SST time series

The example is located in `examples/nino.py`.

```
# This code reproduces the figure 1a in (Torrence, 1998).
# This example does not include the zero padding at the end of the series and
# the red noise analysis.

from __future__ import division

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

import cwave

dt = 0.25
dj = 0.125

x = np.loadtxt("sst_nino3.txt")
s = cwave.Series(x, dt)

var = np.var(s.x)
N = s.x.shape[0]
times = dt * np.arange(N)

## Morlet
```

```
w_morlet = cwave.Morlet()
T_morlet = cwave.cwt(s, w_morlet, dj=dj, scale0=2*dt)
Sn_morlet= T_morlet.S() / var
# COI
taus_morlet= [T_morlet.wavelet.efolding_time(scale) for scale in T_morlet.scales]
mask_morlet = np.zeros_like(T_morlet.W, dtype=np.bool)
for i in range(mask_morlet.shape[0]):
    w = times < taus_morlet[i]
    mask_morlet[i, w] = True
    mask_morlet[i, w[::-1]] = True

## DOG
w_dog = cwave.DOG()
T_dog = cwave.cwt(s, w_dog, dj=dj, scale0=dt/2)
Sn_dog = T_dog.S() / var
# COI
taus_dog= [T_dog.wavelet.efolding_time(scale) for scale in T_dog.scales]
mask_dog = np.zeros_like(T_dog.W, dtype=np.bool)
for i in range(mask_dog.shape[0]):
    w = times < taus_dog[i]
    mask_dog[i, w] = True
    mask_dog[i, w[::-1]] = True

## Plot

fig = plt.figure(1)
gs = gridspec.GridSpec(3, 1, height_ratios=[0.6, 1, 1])

# plot the series s
ax1 = plt.subplot(gs[0])
p1 = ax1.plot(times, s.x, "k")

# plot the wavelet power spectrum (Morlet)
ax2 = plt.subplot(gs[1], sharex=ax1)
X, Y =  np.meshgrid(times, T_morlet.scales)
plt.contourf(X, Y, Sn_morlet, [1, 2, 5, 10], origin='upper', cmap=plt.cm.bone_r,
    extend='both')
plt.contour(X, Y, Sn_morlet, [1, 2, 5, 10], colors='k', linewidths=1, origin='upper')

# plot COI (Morlet)
plt.contour(X, Y, mask_morlet, [0, 1], colors='k', linewidths=1, origin='upper')
plt.contourf(X, Y, mask_morlet, [0, 1], colors='none', origin='upper', extend='both',
    hatches=[None, 'x'])
ax2.set_yscale('log', basey=2)
plt.ylim(0.5, 64)
plt.gca().invert_yaxis()

# plot the wavelet power spectrum (DOG)
ax3 = plt.subplot(gs[2], sharex=ax1)
X, Y =  np.meshgrid(times, T_dog.scales)
plt.contourf(X, Y, Sn_dog, [2, 10], origin='upper', cmap=plt.cm.bone_r,
    extend='both')
plt.contour(X, Y, Sn_dog, [2, 10], colors='k', linewidths=1, origin='upper')

# plot COI (DOG)
plt.contour(X, Y, mask_dog, [0, 1], colors='k', linewidths=1, origin='upper')
plt.contourf(X, Y, mask_dog, [0, 1], colors='none', origin='upper', extend='both',
    hatches=[None, 'x'])
```

```
ax3.set_yscale('log', basey=2)
plt.ylim(0.125, 16)
plt.gca().invert_yaxis()

plt.show()
```



Filtering

TODO

API

Summary

Series and transformed data

`Series`(x[, dt])	Series.
`Trans`(W, scales, omega, wavelet[, x_mean, dt])	Continuous wavelet transformed series.

Functions

`cwt`(s, wavelet[, dj, scale0, scales])	Continuous wavelet transform.
`icwt`(T)	Inverse continuous wavelet transform.

Wavelet functions

The The `Wavelet` abstract class

All the wavelet function classes inherit from the abstract class `Wavelet`. The `Wavelet` has the following abstract methods:

`Wavelet.time`(t[, scale, dt])	Wavelet function in the time domain psi0(t/s).
`Wavelet.freq`(omega[, scale, dt])	Wavelet function in the frequency domain psi0_hat(s*omega).
`Wavelet.fourier_period`(scale)	Computes the equivalent Fourier period (wavelength) from wavelet scale.
`Wavelet.efolding_time`(scale)	Returns the e-folding time tau_s.

Moreover, the `Wavelet` exposes the following methods (available to the subclasses):

`Wavelet.wavelet_scale`(lmbd)	Computes the wavelet scale from equivalent Fourier period (wavelength).
`Wavelet.efolding_time`(scale)	Returns the e-folding time tau_s.
`Wavelet.smallest_scale`(dt)	Returns the smallest resolvable scale.
`Wavelet.auto_scales`(dt, dj, N[, scale0])	Computes the equivalent Fourier period (wavelength) from wavelet scale.

Available wavelet classes

`Morlet`([omega0])	Morlet wavelet function.
`Paul`([m])	Paul Wavelet function.
`DOG`([m])	Derivative Of Gaussian (DOG) Wavelet function.

Classes and functions

class `cwave.`**`Series`**(*x*, *dt=1*)

Series.

Example

```
>>> import cwave
>>> import numpy as np
>>> x = np.random.sample(16)
>>> dt=2
>>> s = cwave.Series(x, dt)
```

dt
Get the time step (float).

x
Get the series (1d numpy array).

class `cwave.`**`Trans`**(*W*, *scales*, *omega*, *wavelet*, *x_mean=0*, *dt=1*)
Continuous wavelet transformed series.

S()
Returns the wavelet power spectrum abs(W)^2.

W
Get the transformed series (2d numpy array).

dt
Get the time step (float).

omega
Get the angular frequencies (1d numpy array).

scales
Get the wavelet scales (1d numpy array).

var()
Returns the variance (eq. 14 in Torrence 1998)

wavelet
Get the wavelet (Wavelet).

x_mean
Get the mean value of the original (non-transformed) series (float).

`cwave.`**`cwt`**(*s*, *wavelet*, *dj=0.25*, *scale0=None*, *scales=None*)
Continuous wavelet transform.

Parameters

- **s** (*`cwave.Series`* object) – series.
- **wavelet** (*`cwave.Wavelet`* object) – wavelet function.
- **dj** (*`float`*) – scale resolution (spacing between scales). A smaller dj will give better scale resolution. If scales is not None, this parameter is ignored.
- **scale0** (*`float`*) – the smallest scale. If scale0=None it is chosen so that the equivalent Fourier period is 2*dt. If scales is not None, this parameter is ignored.
- **scales** (*`None, float or 1d array_like object`*) – wavelet scale(s). If scales=None, the scales are automatically computed as fractional powers of two, with a scale resolution dj and the smallest scale scale0. If the parameter scales is provided, dj and scale0 are automatically ignored.

Returns **T** – continuous wavelet transformed series.

Return type *cwave.Trans* object

Example

```
>>> import cwave
>>> import numpy as np
>>> x = np.random.sample(8)
>>> dt=2
>>> T = cwave.cwt(cwave.Series(x, dt), cwave.DOG(), scales=[2, 4, 8])
>>> T.S()
array([[  1.19017195e-01,   1.54253543e-01,   9.07432163e-02,
          2.96227293e-02,   5.89519189e-04,   1.09486971e-01,
          1.15546678e-02,   3.93108223e-02],
       [  4.43627788e-01,   2.27266757e-01,   3.35649411e-05,
          1.86687786e-01,   3.78349904e-01,   2.24894861e-01,
          3.22011576e-03,   1.84538790e-01],
       [  8.01208492e-03,   4.40859411e-03,   1.92688516e-05,
          3.62275119e-03,   8.01206572e-03,   4.40859342e-03,
          1.92697929e-05,   3.62275056e-03]])
```

cwave.**icwt**(*T*)

Inverse continuous wavelet transform.

Parameters **T** (*cwave.Trans* object) – continuous wavelet transformed series.

Returns **s** – series.

Return type *cwave.Series* object

Example

```
>>> import cwave
>>> import numpy as np
>>> x = np.random.normal(2, 1, 16)
>>> x
array([ 1.6960901 ,  3.27146653,  2.55896222,  2.39484518,  2.34977766,
        3.48575552,  1.7372688 , -0.21329766,  3.5425618 ,  3.34657898,
        1.54359934,  2.96181542,  2.43294205,  1.65980233,  3.44710306,
        1.69615204])
>>> dt=1
>>> T = cwave.cwt(cwave.Series(x, dt), cwave.DOG())
>>> sr = cwave.icwt(T)
>>> sr.x
array([ 1.64067578,  3.28517018,  2.78434897,  2.38949828,  2.58014315,
        3.52751356,  1.34776275, -0.41078628,  3.3648406 ,  3.56461166,
        1.7286081 ,  2.88596331,  2.40275636,  1.81964648,  3.28932397,
        1.7113465 ])
```

class cwave.**Wavelet**

The abstract wavelet base class.

auto_scales(*dt*, *dj*, *N*, *scale0=None*)

Computes the equivalent Fourier period (wavelength) from wavelet scale.

Parameters

- **dt** (*float*) – time step.

- **dj** (*float*) – scale resolution. For the Morlet wavelet, a dj of about 0.5 is the largest value that still gives adequate sampling in scale, while for the other wavelet functions al larger value can be used.
- **N** (*integer*) – number of data samples.
- **scale0** (*float*) – the smallest scale. If scale0=None it is chosen so that the equivalent Fourier period is 2*dt.

Returns **scale** – scales.

Return type 1d numpy array

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.auto_scales(dt=1, dj=0.125, N=64, scale0=1)
array([  1.        ,   1.09050773,   1.18920712,   1.29683955,
         1.41421356,   1.54221083,   1.68179283,   1.83400809,
         2.        ,   2.18101547,   2.37841423,   2.59367911,
         2.82842712,   3.08442165,   3.36358566,   3.66801617,
         4.        ,   4.36203093,   4.75682846,   5.18735822,
         5.65685425,   6.1688433 ,   6.72717132,   7.33603235,
         8.        ,   8.72406186,   9.51365692,  10.37471644,
         11.3137085 ,  12.3376866 ,  13.45434264,  14.67206469,
         16.        ,  17.44812372,  19.02731384,  20.74943287,
         22.627417  ,  24.67537321,  26.90868529,  29.34412938,
         32.        ,  34.89624745,  38.05462768,  41.49886575,
         45.254834  ,  49.35074641,  53.81737058,  58.68825877,  64.        ])
```

efolding_time(*scale*)

Returns the e-folding time tau_s.

Parameters **scale** (*float*) – wavelet scale.

Returns **tau** – e-folding time.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.efolding_time(1)
1.4142135623730951
```

fourier_period(*scale*)

Computes the equivalent Fourier period (wavelength) from wavelet scale.

Parameters **scale** (*float*) – wavelet scale.

Returns **lmbd** – equivalent Fourier period.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.fourier_period(scale=1)
1.0330436477492537
```

freq(*omega*, *scale=1*, *dt=None*)

Wavelet function in the frequency domain psi0_hat(s*omega).

Parameters

- **omega** (*float or 1d array_like object*) – angular frequency (length N).
- **scale** (*float*) – wavelet scale.
- **dt** (*None or float*) – time step. If dt is not None, the method returns the energy-normalized psi_hat (psi_hat0).

Returns **psi0_hat** – the wavelet in the frequency domain

Return type float/complex or 1D numpy array

Example

```
>>> import numpy as np
>>> import cwave
>>> w = cwave.Morlet()
>>> omega = 2 * np.pi * np.fft.fftfreq(32, 1)
>>> w.freq(omega, 10)
array([  0.00000000e+000,   2.17596717e-004,   8.76094852e-002,
         7.46634798e-001,   1.34686366e-001,   5.14277294e-004,
         4.15651521e-008,   7.11082035e-014,   2.57494732e-021,
         1.97367345e-030,   3.20214178e-041,   1.09967442e-053,
         7.99366604e-068,   1.22994640e-083,   4.00575772e-101,
         2.76147731e-120,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000])
```

smallest_scale(*dt*)

Returns the smallest resolvable scale. It is chosen so that the equivalent Fourier period is 2*dt.

Parameters **dt** (*float*) – time step.

Returns **scale0** – smallest scale.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.smallest_scale(dt=1)
1.9360266183887822
```

time(*t*, *scale=1*, *dt=None*)

Wavelet function in the time domain psi0(t/s).

Parameters

- **t** (*float*) – time. If df is not None each element of t should be multiple of dt.
- **scale** (*float*) – wavelet scale.
- **dt** (*None or float*) – time step. If dt is not None, the method returns the energy-normalized psi (psi0).

Returns **psi0** – the wavelet in the time domain.

Return type float/complex or 1D numpy array

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> t = range(-8, 8)
>>> w.time(t, 10)
array([ 0.04772449+0.54333716j, -0.28822893+0.51240757j,
       -0.56261977+0.27763414j, -0.65623233-0.09354365j,
       -0.51129130-0.46835014j, -0.16314795-0.69929479j,
        0.26678672-0.68621589j,  0.61683875-0.42200209j,
        0.75112554+0.j        ,  0.61683875+0.42200209j,
        0.26678672+0.68621589j, -0.16314795+0.69929479j,
       -0.51129130+0.46835014j, -0.65623233+0.09354365j,
       -0.56261977-0.27763414j, -0.28822893-0.51240757j])
```

wavelet_scale(*lmbd*)

Computes the wavelet scale from equivalent Fourier period (wavelength).

Parameters **lmbd** (*float*) – equivalent Fourier period.

Returns **scale** – wavelet scale.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.wavelet_scale(lmbd=10)
9.6801330919439117
```

class cwave.**Morlet**(*omega0=6*)

Morlet wavelet function.

Example

```
>>> import cwave
>>> w = cwave.Morlet(omega0=6)
```

efolding_time(*scale*)

Returns the e-folding time tau_s.

Parameters **scale** (*float*) – wavelet scale.

Returns **tau** – e-folding time.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.efolding_time(1)
1.4142135623730951
```

fourier_period(*scale*)

Computes the equivalent Fourier period (wavelength) from wavelet scale.

Parameters **scale** (*float*) – wavelet scale.

Returns **lmbd** – equivalent Fourier period.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.fourier_period(scale=1)
1.0330436477492537
```

freq(*omega*, *scale=1*, *dt=None*)

Wavelet function in the frequency domain psi0_hat(s*omega).

Parameters

- **omega** (*float or 1d array_like object*) – angular frequency (length N).
- **scale** (*float*) – wavelet scale.
- **dt** (*None or float*) – time step. If dt is not None, the method returns the energy-normalized psi_hat (psi_hat0).

Returns **psi0_hat** – the wavelet in the frequency domain

Return type float/complex or 1D numpy array

Example

```
>>> import numpy as np
>>> import cwave
>>> w = cwave.Morlet()
>>> omega = 2 * np.pi * np.fft.fftfreq(32, 1)
>>> w.freq(omega, 10)
array([  0.00000000e+000,   2.17596717e-004,   8.76094852e-002,
         7.46634798e-001,   1.34686366e-001,   5.14277294e-004,
```

```
        4.15651521e-008,   7.11082035e-014,   2.57494732e-021,
        1.97367345e-030,   3.20214178e-041,   1.09967442e-053,
        7.99366604e-068,   1.22994640e-083,   4.00575772e-101,
        2.76147731e-120,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000])
```

omega0
Get the omega0 parameter

time(*t*, *scale=1*, *dt=None*)
Wavelet function in the time domain psi0(t/s).

Parameters

- **t** (*float*) – time. If df is not None each element of t should be multiple of dt.
- **scale** (*float*) – wavelet scale.
- **dt** (*None or float*) – time step. If dt is not None, the method returns the energy-normalized psi (psi0).

Returns **psi0** – the wavelet in the time domain.

Return type float/complex or 1D numpy array

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> t = range(-8, 8)
>>> w.time(t, 10)
array([ 0.04772449+0.54333716j, -0.28822893+0.51240757j,
       -0.56261977+0.27763414j, -0.65623233-0.09354365j,
       -0.51129130-0.46835014j, -0.16314795-0.69929479j,
        0.26678672-0.68621589j,  0.61683875-0.42200209j,
        0.75112554+0.j        ,  0.61683875+0.42200209j,
        0.26678672+0.68621589j, -0.16314795+0.69929479j,
       -0.51129130+0.46835014j, -0.65623233+0.09354365j,
       -0.56261977-0.27763414j, -0.28822893-0.51240757j])
```

class cwave.**Paul**(*m=4*)
Paul Wavelet function.

Example

```
>>> import cwave
>>> w = cwave.Paul(m=4)
```

efolding_time(*scale*)
Returns the e-folding time tau_s.

Parameters **scale** (*float*) – wavelet scale.

Returns **tau** – e-folding time.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.efolding_time(1)
1.4142135623730951
```

fourier_period(*scale*)

Computes the equivalent Fourier period (wavelength) from wavelet scale.

Parameters **scale** (*float*) – wavelet scale.

Returns **lmbd** – equivalent Fourier period.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.fourier_period(scale=1)
1.0330436477492537
```

freq(*omega*, *scale=1*, *dt=None*)

Wavelet function in the frequency domain psi0_hat(s*omega).

Parameters

- **omega** (*float or 1d array_like object*) – angular frequency (length N).
- **scale** (*float*) – wavelet scale.
- **dt** (*None or float*) – time step. If dt is not None, the method returns the energy-normalized psi_hat (psi_hat0).

Returns **psi0_hat** – the wavelet in the frequency domain

Return type float/complex or 1D numpy array

Example

```
>>> import numpy as np
>>> import cwave
>>> w = cwave.Morlet()
>>> omega = 2 * np.pi * np.fft.fftfreq(32, 1)
>>> w.freq(omega, 10)
array([  0.00000000e+000,   2.17596717e-004,   8.76094852e-002,
         7.46634798e-001,   1.34686366e-001,   5.14277294e-004,
         4.15651521e-008,   7.11082035e-014,   2.57494732e-021,
         1.97367345e-030,   3.20214178e-041,   1.09967442e-053,
         7.99366604e-068,   1.22994640e-083,   4.00575772e-101,
         2.76147731e-120,   0.00000000e+000,   0.00000000e+000,
```

```
        0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
        0.00000000e+000,   0.00000000e+000])
```

m

Get the m parameter (order)

time(*t*, *scale=1*, *dt=None*)

Wavelet function in the time domain psi0(t/s).

Parameters

- **t** (*float*) – time. If df is not None each element of t should be multiple of dt.
- **scale** (*float*) – wavelet scale.
- **dt** (*None or float*) – time step. If dt is not None, the method returns the energy-normalized psi (psi0).

Returns **psi0** – the wavelet in the time domain.

Return type float/complex or 1D numpy array

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> t = range(-8, 8)
>>> w.time(t, 10)
array([ 0.04772449+0.54333716j, -0.28822893+0.51240757j,
       -0.56261977+0.27763414j, -0.65623233-0.09354365j,
       -0.51129130-0.46835014j, -0.16314795-0.69929479j,
        0.26678672-0.68621589j,  0.61683875-0.42200209j,
        0.75112554+0.j        ,  0.61683875+0.42200209j,
        0.26678672+0.68621589j, -0.16314795+0.69929479j,
       -0.51129130+0.46835014j, -0.65623233+0.09354365j,
       -0.56261977-0.27763414j, -0.28822893-0.51240757j])
```

class cwave.**DOG**(*m=2*)

Derivative Of Gaussian (DOG) Wavelet function.

Example

```
>>> import cwave
>>> w = cwave.DOG(m=2)
```

efolding_time(*scale*)

Returns the e-folding time tau_s.

Parameters **scale** (*float*) – wavelet scale.

Returns **tau** – e-folding time.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.efolding_time(1)
1.4142135623730951
```

fourier_period(*scale*)

Computes the equivalent Fourier period (wavelength) from wavelet scale.

Parameters **scale** (*float*) – wavelet scale.

Returns **lmbd** – equivalent Fourier period.

Return type float

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> w.fourier_period(scale=1)
1.0330436477492537
```

freq(*omega*, *scale=1*, *dt=None*)

Wavelet function in the frequency domain psi0_hat(s*omega).

Parameters

- **omega** (*float or 1d array_like object*) – angular frequency (length N).
- **scale** (*float*) – wavelet scale.
- **dt** (*None or float*) – time step. If dt is not None, the method returns the energy-normalized psi_hat (psi_hat0).

Returns **psi0_hat** – the wavelet in the frequency domain

Return type float/complex or 1D numpy array

Example

```
>>> import numpy as np
>>> import cwave
>>> w = cwave.Morlet()
>>> omega = 2 * np.pi * np.fft.fftfreq(32, 1)
>>> w.freq(omega, 10)
array([  0.00000000e+000,   2.17596717e-004,   8.76094852e-002,
         7.46634798e-001,   1.34686366e-001,   5.14277294e-004,
         4.15651521e-008,   7.11082035e-014,   2.57494732e-021,
         1.97367345e-030,   3.20214178e-041,   1.09967442e-053,
         7.99366604e-068,   1.22994640e-083,   4.00575772e-101,
         2.76147731e-120,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000])
```

m

Get the m parameter (derivative)

time(*t*, *scale=1*, *dt=None*)

Wavelet function in the time domain psi0(t/s).

Parameters

- **t** (*float*) – time. If df is not None each element of t should be multiple of dt.
- **scale** (*float*) – wavelet scale.
- **dt** (*None or float*) – time step. If dt is not None, the method returns the energy-normalized psi (psi0).

Returns **psi0** – the wavelet in the time domain.

Return type float/complex or 1D numpy array

Example

```
>>> import cwave
>>> w = cwave.Morlet()
>>> t = range(-8, 8)
>>> w.time(t, 10)
array([ 0.04772449+0.54333716j, -0.28822893+0.51240757j,
       -0.56261977+0.27763414j, -0.65623233-0.09354365j,
       -0.51129130-0.46835014j, -0.16314795-0.69929479j,
        0.26678672-0.68621589j,  0.61683875-0.42200209j,
        0.75112554+0.j        ,  0.61683875+0.42200209j,
        0.26678672+0.68621589j, -0.16314795+0.69929479j,
       -0.51129130+0.46835014j, -0.65623233+0.09354365j,
       -0.56261977-0.27763414j, -0.28822893-0.51240757j])
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

cwave, 8

Index

A

auto_scales() (cwave.Wavelet method), 10

C

cwave (module), 8
cwt() (in module cwave), 9

D

DOG (class in cwave), 17
dt (cwave.Series attribute), 9
dt (cwave.Trans attribute), 9

E

efolding_time() (cwave.DOG method), 17
efolding_time() (cwave.Morlet method), 13
efolding_time() (cwave.Paul method), 15
efolding_time() (cwave.Wavelet method), 11

F

fourier_period() (cwave.DOG method), 18
fourier_period() (cwave.Morlet method), 14
fourier_period() (cwave.Paul method), 16
fourier_period() (cwave.Wavelet method), 11
freq() (cwave.DOG method), 18
freq() (cwave.Morlet method), 14
freq() (cwave.Paul method), 16
freq() (cwave.Wavelet method), 12

I

icwt() (in module cwave), 10

M

m (cwave.DOG attribute), 18
m (cwave.Paul attribute), 17
Morlet (class in cwave), 13

O

omega (cwave.Trans attribute), 9
omega0 (cwave.Morlet attribute), 15

P

Paul (class in cwave), 15

S

S() (cwave.Trans method), 9
scales (cwave.Trans attribute), 9
Series (class in cwave), 8
smallest_scale() (cwave.Wavelet method), 12

T

time() (cwave.DOG method), 19
time() (cwave.Morlet method), 15
time() (cwave.Paul method), 17
time() (cwave.Wavelet method), 12
Trans (class in cwave), 9

V

var() (cwave.Trans method), 9

W

W (cwave.Trans attribute), 9
Wavelet (class in cwave), 10
wavelet (cwave.Trans attribute), 9
wavelet_scale() (cwave.Wavelet method), 13

X

x (cwave.Series attribute), 9
x_mean (cwave.Trans attribute), 9