bruges Documentation

Release 0.4.3

Evan Bianco, Ben Bougher, Matt Hall

Jun 30, 2021

Table of Contents

CHAPTER 1

bruges is a

bucket of really useful geophysical equations and stuff

In other words, it's just a load of functions that implement important equations in (mostly seismic) geophysics, from Aki-Richards to Zoeppritz.

1.1 Status

The bruges module contains several common geophysics functions
used for modelling and post-processing seismic reflection data.

1.2 Installation

Install with

```
pip install bruges
```

bruges requires NumPy and SciPy.

1.2.1 Content

bruges is a

bucket of really useful geophysical equations and stuff

In other words, it's just a load of functions that implement important equations in (mostly seismic) geophysics, from Aki-Richards to Zoeppritz.

Installation

Install with

```
pip install bruges
```

bruges requires NumPy and SciPy.

bruges

bruges package

Subpackages

bruges.attribute package

Submodules

bruges.attribute.dipsteer module

A dip attribute, probably most useful for guiding other attributes.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.attribute.dipsteer.**dipsteer**(*data*, *window_length*, *stepout*, *maxlag*, *overlap=1*, *dt=1*, *return_correlation=False*)

Calculates a dip field by finding the maximum correlation between adjacent traces.

Parameters

- **(ndarray)** (*data*) – A 2D seismic section (samples,traces) used to calculate dip.
- **(float)** (*maxlag*) – The length [in ms] of the window to use.
- **(int)** (*stepout*) – The number of traces on either side of each point to average when calculating the dip.
- **(float)** – The maximum amount time lag to use when correlating the traces.

Keyword Arguments

- **(float)** (*dt*) – The fractional overlap for each window. A value of 0 uses no redudant data, a value of 1 slides the dip correlator one sample at a time. Defaults to 1.
- **(float)** – The time sample interval in ms.

- **(bool)** (*return_correlation*) – Whether to return the correlation coefficients. If you choose True, you'll get a tuple, not an ndarray.

Returns a dip field [samples/trace] of the same shape as the input data (and optionally correlation coefficients, in which case you'll get a tuple of ndarrays back).

bruges.attribute.energy module

Mean-squared energy measurement.

copyright 2019 Agile Geoscience

license Apache 2.0

bruges.attribute.energy.**energy**(*traces*, *duration*, *dt=1*)

Compute an mean-squared energy measurement on seismic data.

The attribute is computed over the last dimension. That is, time should be in the last dimension, so a 100 inline, 100 crossline seismic volume with 250 time slices would have shape (100, 100, 250).

Parameters

- **traces** (*ndarray*) – The data array to use for calculating energy.
- **duration** (*float*) – the time duration of the window (in seconds), or samples if dt=1.
- **dt** (*float*) – the sample interval of the data (in seconds). Defaults to 1 so duration can be in samples.

Returns An array the same dimensions as the input array.

Return type ndarray

bruges.attribute.similarity module

A variance method to compute similarity.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.attribute.similarity.**similarity**(*traces*, *duration*, *dt=1*, *step_out=1*, *lag=0*)

Compute similarity for each point of a seismic section using a variance method between traces.

For each point, a kernel of n_samples length is extracted from a trace. The similarity is calculated as a normalized variance between two adjacent trace sections, where a value of 1 is obtained by identical if the traces are identical. The step out will decide how many adjacent traces will be used for each kernel, and should be increased for poor quality data. The lag determines how much neighbouring traces can be shifted when calculating similiarity, which should be increased for dipping data.

Parameters

- **traces** – A 2D numpy array arranged as [time, trace].
- **duration** – The length in seconds of the window trace kernel used to calculate the similarity.

Keyword Arguments

- **(default=1)** (*step_out*) – The number of adjacent traces to the kernel to check similarity. The maximum similarity value will be chosen.

- **(default=0)** (*lag*) – The maximum number of time samples adjacent traces can be shifted by. The maximum similarity of will be used.
- **(default=1)** (*dt*) – The sample interval of the traces in sec. (eg. 0.001, 0.002, ...). Will default to one, allowing duration to be given in samples.

bruges.attribute.spectraldecomp module

Spectral decomposition

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.attribute.spectraldecomp.**spectraldecomp**(*data*, *f=(0.1*, *0.25*, *0.4)*, *window_length=32*, *dt=1*, *window_type='hann'*, *overlap=1*, *normalize=False*)

Uses the STFT to decompose traces into normalized spectra. Only frequencies defined by the user will be output. Using 3 frequencies will work for RGB color plots.

Parameters **data** – A 1/2D array (samples, traces) of data that will be decomposed.

Keyword Arguments

- **f** – A list of frequencies to select from the spectra.
- **window_length** – The length of fft window to use for the STFTs. Defaults to 32. Can be provided in seconds if dt is specified.
- **dt** – The time sample interval of the traces.
- **window_type** – The type of window to use for the STFT. The same input as scipy.signal.get_window.
- **overlap** – The fraction of overlap between adjacent STFT windows
- **normalize** – Normalize the energy in each STFT window

Returns an array of shape (samples, traces, f)

bruges.attribute.spectrogram module

Spectrogram.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.attribute.spectrogram.**spectrogram**(*data*, *window_length*, *dt=1.0*, *window_type='boxcar'*, *overlap=0.5*, *normalize=False*, *zero_padding=0*)

Calculates a spectrogram using windowed STFTs.

Parameters

- **data** – 1D numpy array to process into spectra.
- **window_length** – The length of the window in seconds if dt is set, otherwise in samples. Will zero pad to the closest power of two.

Keyword Arguments

- **window_type** – A string specifying the type of window to use for the STFT. The same input as scipy.signal.get_window
- **dt** – The time sample interval of the trace. Defaults to 1, which allows window_length to be in seconds.
- **overlap** – The fractional overlap for each window. A value of 0 uses no redudant data, a value of 1 slides the STFT window one sample at a time. Defaults to 0.5
- **normalize** – Normalizes the each spectral slice to have unity energy.
- **zero_padding** – The amount of zero padding to when creating the spectra.

Returns

A spectrogram of the data ([time, freq]). (2D array for 1D input)

See Also

spectraldecomp : Spectral decomposition

Module contents

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.filters package

Submodules

bruges.filters.anisodiff module

copyright Alistair Muldal

license Unknown, shared on StackOverflow and Pastebin

Reference: P. Perona and J. Malik. Scale-space and edge detection using ansotropic diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(7):629-639, July 1990. <http://www.cs.berkeley.edu/~malik/papers/MP-aniso.pdf>

Original MATLAB code by Peter Kovesi School of Computer Science & Software Engineering The University of Western Australia pk @ csse uwa edu au <http://www.csse.uwa.edu.au>

Translated to Python and optimised by Alistair Muldal Department of Pharmacology University of Oxford <alistair.muldal@pharm.ox.ac.uk>

June 2000 original version. March 2002 corrected diffusion eqn No 2. July 2012 translated to Python

bruges.filters.anisodiff.**anisodiff**(*img*, *niter=1*, *kappa=50*, *gamma=0.1*, *step=(1.0, 1.0)*, *option=1*)

Anisotropic diffusion.

Usage: imgout = anisodiff(im, niter, kappa, gamma, option)

Parameters

- **– input image** (*img*) –
- **– number of iterations** (*niter*) –
- **– conduction coefficient 20-100 ?** (*kappa*) –

- **– max value of .25 for stability** (*gamma*) –
- **– tuple, the distance between adjacent pixels in** (*step*) –
- **– 1 Perona Malik diffusion equation No 1** (*option*) – 2 Perona Malik diffusion equation No 2

Returns imgout - diffused image.

kappa controls conduction as a function of gradient. If kappa is low small intensity gradients are able to block conduction and hence diffusion across step edges. A large value reduces the influence of intensity gradients on conduction.

gamma controls speed of diffusion (you usually want it at a maximum of 0.25)

step is used to scale the gradients in case the spacing between adjacent pixels differs in the x and y axes

Diffusion equation 1 favours high contrast edges over low contrast ones. Diffusion equation 2 favours wide regions over smaller ones.

bruges.filters.anisodiff.**anisodiff3**(*stack*, *niter=1*, *kappa=50*, *gamma=0.1*, *step=(1.0*, *1.0*, *1.0)*, *option=1*)

3D Anisotropic diffusion.

Usage: stackout = anisodiff(stack, niter, kappa, gamma, option)

Parameters

- **– input stack** (*stack*) –
- **– number of iterations** (*niter*) –
- **– conduction coefficient 20-100 ?** (*kappa*) –
- **– max value of .25 for stability** (*gamma*) –
- **– tuple, the distance between adjacent pixels in** (*step*) –
- **– 1 Perona Malik diffusion equation No 1** (*option*) – 2 Perona Malik diffusion equation No 2

Returns stackout - diffused stack.

kappa controls conduction as a function of gradient. If kappa is low small intensity gradients are able to block conduction and hence diffusion across step edges. A large value reduces the influence of intensity gradients on conduction.

gamma controls speed of diffusion (you usually want it at a maximum of 0.25)

step is used to scale the gradients in case the spacing between adjacent pixels differs in the x,y and/or z axes

Diffusion equation 1 favours high contrast edges over low contrast ones. Diffusion equation 2 favours wide regions over smaller ones.

bruges.filters.convolve module

Convolution in n-dimensions.

copyright 2019 Agile Geoscience

license Apache 2.0

bruges.filters.convolve.**apply_along_axis**(*func_1d*, *arr*, **args*, ***kwargs*)

bruges.filters.convolve.**convolve**(*reflectivity*, *wavelet*)
Convolve n-dimensional reflectivity with a 1D wavelet or 2D wavelet bank.

Args reflectivity (ndarray): The reflectivity trace, or 2D section, or volume. wavelet (ndarray): The wavelet, must be 1D function or a 2D wavelet 'bank'.

If a wavelet bank, time should be on the last axis.

bruges.filters.filters module

Smoothers.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.filters.filters.**conservative**(*arr*, *size=5*, *supercon=False*)
Conservative, a nonlinear n-D despiking filter. Very conservative! Only changes centre value if it is outside the range of all the other values in the kernel. Read http://subsurfwiki.org/wiki/Conservative_filter

Parameters

- **arr** (*ndarray*) – an n-dimensional array, such as a seismic horizon.
- **size** (*int*) – the kernel size, e.g. 5 for 5x5 (in a 2D arr). Should be odd, rounded up if not.
- **supercon** (*bool*) – whether to be superconservative. If True, replaces pixel with min or max of kernel. If False (default), replaces pixel with mean of kernel.

Returns the resulting smoothed array.

Return type ndarray

bruges.filters.filters.**kuwahara**(*arr*, *size=5*)
Kuwahara, a nonlinear 2D smoothing filter. http://subsurfwiki.org/wiki/Kuwahara_filter

Parameters

- **arr** (*ndarray*) – a 2D array, such as a seismic horizon.
- **size** (*int*) – the kernel size, e.g. 5 for 5x5. Should be odd, rounded up if not.

Returns the resulting smoothed array.

Return type ndarray

bruges.filters.filters.**mean**(*arr*, *size=5*)
A linear n-D smoothing filter. Can be used as a moving average on 1D data.

Parameters

- **arr** (*ndarray*) – an n-dimensional array, such as a seismic horizon.
- **size** (*int*) – the kernel size, e.g. 5 for 5x5. Should be odd, rounded up if not.

Returns the resulting smoothed array.

Return type ndarray

bruges.filters.filters.**median**(*arr*, *size=5*)
A nonlinear n-D edge-preserving smoothing filter.

Parameters

- **arr** (*ndarray*) – an n-dimensional array, such as a seismic horizon.

- **size** (*int*) – the kernel size, e.g. 5 for 5x5. Should be odd, rounded up if not.

Returns the resulting smoothed array.

Return type ndarray

bruges.filters.filters.**mode**(*arr*, *size=5*, *tie='smallest'*)

A nonlinear n-D categorical smoothing filter. Use this to filter non- continuous variables, such as categorical integers, e.g. to label facies.

Parameters

- **arr** (*ndarray*) – an n-dimensional array, such as a seismic horizon.
- **size** (*int*) – the kernel size, e.g. 5 for 5x5. Should be odd, rounded up if not.
- **tie** (*str*) – *'smallest'* or *'largest'*. In the event of a tie (i.e. two or more values having the same count in the kernel), whether to give back the smallest of the tying values, or the largest.

Returns the resulting smoothed array.

Return type ndarray

bruges.filters.filters.**rms**(*arr*, *size=5*)

A linear n-D smoothing filter. Can be used as a moving average on 1D data.

Parameters

- **arr** (*ndarray*) – an n-dimensional array, such as a seismic horizon.
- **size** (*int*) – the kernel size, e.g. 5 for 5x5. Should be odd, rounded up if not.

Returns the resulting smoothed array.

Return type ndarray

bruges.filters.filters.**rotate_phase**(*s*, *phi*, *degrees=False*)

Performs a phase rotation of wavelet or wavelet bank using:

$$A = w(t)\cos(\phi) - h(t)\sin(\phi)$$

where w(t) is the wavelet and h(t) is its Hilbert transform.

The analytic signal can be written in the form S(t) = A(t)exp(j*theta(t)) where A(t) = magnitude(hilbert(w(t))) and theta(t) = angle(hilbert(w(t)) then a constant phase rotation phi would produce the analytic signal S(t) = A(t)exp(j*(theta(t) + phi)). To get the non analytic signal we take real(S(t)) == A(t)cos(theta(t) + phi) == A(t)(cos(theta(t))cos(phi) - sin(theta(t))sin(phi)) <= trig identity == w(t)cos(phi) - h(t)sin(phi)

Parameters

- **w** (*ndarray*) – The wavelet vector, can be a 2D wavelet bank.
- **phi** (*float*) – The phase rotation angle (in radians) to apply.
- **degrees** (*bool*) – If phi is in degrees not radians.

Returns The phase rotated signal (or bank of signals).

bruges.filters.filters.**snn**(*arr*, *size=5*, *include=True*)

Symmetric nearest neighbour, a nonlinear 2D smoothing filter. http://subsurfwiki.org/wiki/Symmetric_nearest_neighbour_filter

Parameters

- **arr** (*ndarray*) – a 2D array, such as a seismic horizon.

- **size** (*int*) – the kernel size, e.g. 5 for 5x5. Should be odd, rounded up if not.
- **include** (*bool*) – whether to include the central pixel itself.

Returns the resulting smoothed array.

Return type ndarray

bruges.filters.kernels module

2D kernels for image processing.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.filters.kernels.**gaussian**(*size*, *size_y=None*)
2D Gaussian Kernel

Parameters

- **size** (*int*) – the kernel size, e.g. 5 for 5x5 (in a 2D arr). Should be odd, rounded up if not.
- **size_y** (*int*) – similar to size. If not provided, uses size as default.

Returns: a Gaussian kernel.

bruges.filters.kernels.**gaussian_kernel**(*size*, *size_y=None*)
2D Gaussian Kernel

Parameters

- **size** (*int*) – the kernel size, e.g. 5 for 5x5 (in a 2D arr). Should be odd, rounded up if not.
- **size_y** (*int*) – similar to size. If not provided, uses size as default.

Returns: a Gaussian kernel.

bruges.filters.wavelets module

Seismic wavelets.

copyright 2021 Agile Geoscience

license Apache 2.0

bruges.filters.wavelets.**berlage**(*duration*, *dt*, *f*, *n=2*, *alpha=180*, *phi=-1.5707963267948966*, *t=None*, *return_t=False*, *sym=None*)
Generates a Berlage wavelet with a peak frequency f. Implements

$$w(t) = AH(t)t^n \mathrm{e}^{-lphat} \cos(2\pi f_0 t + \phi_0)$$

as described in Aldridge, DF (1990), The Berlage wavelet, GEOPHYSICS 55 (11), p 1508-1511. Berlage wavelets are causal, minimum phase and useful for modeling marine airgun sources.

If you pass a 1D array of frequencies, you get a wavelet bank in return.

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.

- **dt** (*float*) – The sample interval in seconds (often one of 0.001, 0.002, or 0.004).
- **f** (*array-like*) – Centre frequency of the wavelet in Hz. If a sequence is passed, you will get a 2D array in return, one row per frequency.
- **n** (*float*) – The time exponent; non-negative and real.
- **alpha** (*float*) – The exponential decay factor; non-negative and real.
- **phi** (*float*) – The phase.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.
- **sym** (*bool*) – If True (default behaviour before v0.5) then the wavelet is forced to have an odd number of samples and the central sample is at 0 time.

Returns

ndarray. Berlage wavelet(s) with centre frequency f sampled on t. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

bruges.filters.wavelets.**cosine**(*duration*, *dt*, *f*, *t=None*, *return_t=False*, *taper='gaussian'*, *sigma=None*, *sym=None*)

With the default Gaussian window, equivalent to a 'modified Morlet' also sometimes called a 'Gabor' wavelet. The *bruges.filters.gabor* function returns a similar shape, but with a higher mean frequancy, somewhere between a Ricker and a cosine (pure tone).

If you pass a 1D array of frequencies, you get a wavelet bank in return.

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.
- **dt** (*float*) – The sample interval in seconds (often one of 0.001, 0.002, or 0.004).
- **f** (*array-like*) – Dominant frequency of the wavelet in Hz. If a sequence is passed, you will get a 2D array in return, one row per frequency.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.
- **taper** (*str or function*) – The window or tapering function to apply. To use one of NumPy's functions, pass 'bartlett', 'blackman' (the default), 'hamming', or 'hanning'; to apply no tapering, pass 'none'. To apply your own function, pass a function taking only the length of the window and returning the window function.
- **sigma** (*float*) – Width of the default Gaussian window, in seconds. Defaults to 1/8 of the duration.

Returns

ndarray. sinc wavelet(s) with centre frequency f sampled on t. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

bruges.filters.wavelets.**gabor**(*duration*, *dt*, *f*, *t=None*, *return_t=False*, *sym=None*)

Generates a Gabor wavelet with a peak frequency f0 at time t.

https://en.wikipedia.org/wiki/Gabor_wavelet

If you pass a 1D array of frequencies, you get a wavelet bank in return.

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.
- **dt** (*float*) – The sample interval in seconds (often one of 0.001, 0.002, or 0.004).
- **f** (*array-like*) – Centre frequency of the wavelet in Hz. If a sequence is passed, you will get a 2D array in return, one row per frequency.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.

Returns

ndarray. Gabor wavelet(s) with centre frequency f sampled on t. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

bruges.filters.wavelets.**generalized**(*duration*, *dt*, *f*, *u=2*, *t=None*, *return_t=False*, *imag=False*, *sym=None*)

Wang's generalized wavelet, of which the Ricker is a special case where u = 2. The parameter u is the order of the time-domain derivative, which can be a fractional derivative.

As given by Wang (2015), Generalized seismic wavelets. GJI 203, p 1172-78. DOI: https://doi.org/10.1093/gji/ggv346. I am using the (more accurate) frequency domain method (eq 4 in that paper).

Parameters

- **duration** (*float*) – The length of the wavelet, in s.
- **dt** (*float*) – The time sample interval in s.
- **f** (*float or array-like*) – The frequency or frequencies, in Hertz.
- **u** (*float or array-like*) – The fractional derivative parameter u.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – Whether to return the time basis array.
- **center** (*bool*) – Whether to center the wavelet on time 0.
- **imag** (*bool*) – Whether to return the imaginary component as well.
- **sym** (*bool*) – If True (default behaviour before v0.5) then the wavelet is forced to have an odd number of samples and the central sample is at 0 time.

Returns

ndarray. If f and u are floats, the resulting wavelet has duration/dt = A samples. If you give f as an array of length M and u as an array of length N, then the resulting wavelet bank will have shape (M, N, A). If f or u are floats, their size will be 1, and they will be squeezed out: the bank is always squeezed to its minimum number of dimensions. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

bruges.filters.wavelets.**klauder**(*duration*, *dt*, *f*, *autocorrelate=True*, *t=None*, *return_t=False*, *taper='blackman'*, *sym=None*, ***kwargs*)

By default, gives the autocorrelation of a linear frequency modulated wavelet (sweep). Uses scipy.signal.chirp, adding dimensions as necessary.

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.
- **dt** (*float*) – is the sample interval in seconds (usually 0.001, 0.002, or 0.004)
- **f** (*array-like*) – Upper and lower frequencies. Any sequence like (f1, f2). A list of lists will create a wavelet bank.
- **autocorrelate** (*bool*) – Whether to autocorrelate the sweep(s) to create a wavelet. Default is *True*.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.
- **taper** (*str or function*) – The window or tapering function to apply. To use one of NumPy's functions, pass 'bartlett', 'blackman' (the default), 'hamming', or 'hanning'; to apply no tapering, pass 'none'. To apply your own function, pass a function taking only the length of the window and returning the window function.
- **sym** (*bool*) – If True (default behaviour before v0.5) then the wavelet is forced to have an odd number of samples and the central sample is at 0 time.
- ****kwargs** – Further arguments are passed to scipy.signal.chirp. They are *method* ('linear','quadratic','logarithmic'), *phi* (phase offset in degrees), and *vertex_zero*.

Returns

The waveform. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

Return type ndarray

bruges.filters.wavelets.**ormsby**(*duration*, *dt*, *f*, *t=None*, *return_t=False*, *sym=None*)

The Ormsby wavelet requires four frequencies which together define a trapezoid shape in the spectrum. The Ormsby wavelet has several sidelobes, unlike Ricker wavelets.

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.
- **dt** (*float*) – The sample interval in seconds (usually 0.001, 0.002, or 0.004).
- **f** (*array-like*) – Sequence of form (f1, f2, f3, f4), or list of lists of frequencies, which will return a 2D wavelet bank.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.
- **sym** (*bool*) – If True (default behaviour before v0.5) then the wavelet is forced to have an odd number of samples and the central sample is at 0 time.

Returns

A vector containing the Ormsby wavelet, or a bank of them. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

Return type ndarray

bruges.filters.wavelets.**ormsby_fft**(*duration*, *dt*, *f*, *P=(0, 0)*, *return_t=True*, *sym=True*)

Non-white Ormsby, with arbitary amplitudes.

Can use as many points as you like. The power of f1 and f4 is assumed to be 0, so you only need to provide p2 and p3 (the corners). (You can actually provide as many f points as you like, as long as there are n - 2 matching p points.)

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.
- **dt** (*float*) – The sample interval in seconds (usually 0.001, 0.002, or 0.004).
- **f** (*array-like*) – Sequence of form (f1, f2, f3, f4), or list of lists of frequencies, which will return a 2D wavelet bank.
- **P** (*tuple*) – The power of the f2 and f3 frequencies, in relative dB. (The magnitudes of f1 and f4 are assumed to be -∞ dB, i.e. a magnitude of 0.) The default power values of (0, 0) results in a trapezoidal spectrum and a conventional Ormsby wavelet. Pass, e.g. (0, -15) for a 'pink' wavelet, with more energy in the lower frequencies.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.
- **sym** (*bool*) – If True (default behaviour before v0.5) then the wavelet is forced to have an odd number of samples and the central sample is at 0 time.

Returns

A vector containing the Ormsby wavelet, or a bank of them. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

Return type ndarray

bruges.filters.wavelets.**ricker**(*duration*, *dt*, *f*, *t=None*, *return_t=False*, *sym=None*)

Also known as the mexican hat wavelet, models the function:

$$A = (1 - 2\pi^2 f^2 t^2) e^{-\pi^2 f^2 t^2}$$

If you pass a 1D array of frequencies, you get a wavelet bank in return.

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.
- **dt** (*float*) – The sample interval in seconds (often one of 0.001, 0.002, or 0.004).
- **f** (*array-like*) – Centre frequency of the wavelet in Hz. If a sequence is passed, you will get a 2D array in return, one row per frequency.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.
- **sym** (*bool*) – If True (default behaviour before v0.5) then the wavelet is forced to have an odd number of samples and the central sample is at 0 time.

Returns

ndarray. Ricker wavelet(s) with centre frequency f sampled on t. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

bruges.filters.wavelets.**rotate_phase**(*w*, *phi*, *degrees=False*)

Performs a phase rotation of wavelet or wavelet bank using:

$$A = w(t)\cos(\phi) - h(t)\sin(\phi)$$

where w(t) is the wavelet and h(t) is its Hilbert transform.

The analytic signal can be written in the form S(t) = A(t)exp(j*theta(t)) where A(t) = magnitude(hilbert(w(t))) and theta(t) = angle(hilbert(w(t)) then a constant phase rotation phi would produce the analytic signal S(t) = A(t)exp(j*(theta(t) + phi)). To get the non analytic signal we take real(S(t)) == A(t)cos(theta(t) + phi) == A(t)(cos(theta(t))cos(phi) - sin(theta(t))sin(phi)) <= trig identity == w(t)cos(phi) - h(t)sin(phi)

Parameters

- **w** (*ndarray*) – The wavelet vector, can be a 2D wavelet bank.
- **phi** (*float*) – The phase rotation angle (in radians) to apply.
- **degrees** (*bool*) – If phi is in degrees not radians.

Returns The phase rotated signal (or bank of signals).

bruges.filters.wavelets.**sinc**(*duration*, *dt*, *f*, *t=None*, *return_t=False*, *taper='blackman'*, *sym=None*)

sinc function centered on t=0, with a dominant frequency of f Hz.

If you pass a 1D array of frequencies, you get a wavelet bank in return.

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.
- **dt** (*float*) – The sample interval in seconds (often one of 0.001, 0.002, or 0.004).
- **f** (*array-like*) – Dominant frequency of the wavelet in Hz. If a sequence is passed, you will get a 2D array in return, one row per frequency.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.
- **taper** (*str or function*) – The window or tapering function to apply. To use one of NumPy's functions, pass 'bartlett', 'blackman' (the default), 'hamming', or 'hanning'; to apply no tapering, pass 'none'. To apply your own function, pass a function taking only the length of the window and returning the window function.

Returns

ndarray. sinc wavelet(s) with centre frequency f sampled on t. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

bruges.filters.wavelets.**sweep**(*duration*, *dt*, *f*, *autocorrelate=True*, *t=None*, *return_t=False*, *taper='blackman'*, *sym=None*, ***kwargs*)

By default, gives the autocorrelation of a linear frequency modulated wavelet (sweep). Uses scipy.signal.chirp, adding dimensions as necessary.

Parameters

- **duration** (*float*) – The length in seconds of the wavelet.
- **dt** (*float*) – is the sample interval in seconds (usually 0.001, 0.002, or 0.004)
- **f** (*array-like*) – Upper and lower frequencies. Any sequence like (f1, f2). A list of lists will create a wavelet bank.

- **autocorrelate** (*bool*) – Whether to autocorrelate the sweep(s) to create a wavelet. Default is *True*.
- **t** (*array-like*) – The time series to evaluate at, if you don't want one to be computed. If you pass *t* then *duration* and *dt* will be ignored, so we recommend passing *None* for those arguments.
- **return_t** (*bool*) – If True, then the function returns a tuple of wavelet, time-basis.
- **taper** (*str or function*) – The window or tapering function to apply. To use one of NumPy's functions, pass 'bartlett', 'blackman' (the default), 'hamming', or 'hanning'; to apply no tapering, pass 'none'. To apply your own function, pass a function taking only the length of the window and returning the window function.
- **sym** (*bool*) – If True (default behaviour before v0.5) then the wavelet is forced to have an odd number of samples and the central sample is at 0 time.
- ****kwargs** – Further arguments are passed to scipy.signal.chirp. They are *method* ('linear','quadratic','logarithmic'), *phi* (phase offset in degrees), and *vertex_zero*.

Returns

The waveform. If you passed *return_t=True* then a tuple of (wavelet, t) is returned.

Return type ndarray

Module contents

bruges.models package

Submodules

bruges.models.panel module

bruges.models.panel.**interpolate**(**arrays*, *num=50*, *dists=None*, *kind='linear'*)

Linear interpolation between 1D arrays of the same length.

Parameters

- **arrays** (*ndarray*) – The 1D arrays to interpolate. All must be the same length. You can use the *reconcile()* function to produce them.
- **num** (*int*) – The number of steps to take, so will be the width (number of cols) of the output array.
- **dists** (*array-like*) – A list or tuple or array of the distances (any units) between the arrays in the real world.
- **kind** (*str*) – Will be passed to scipy.interpolate.interp1d, which does the lateral interpolation between samples.

Returns

ndarray. The result, with *num* columns. The number of rows is the same as the number of samples in the input arrays.

Example

```
>>> a = np.array([2, 6, 7, 7, 3])
>>> b = np.array([3, 7, 7, 3, 3])
>>> interp = interpolate(a, b, num=10)
>>> interp.shape
(5, 10)
```

bruges.models.panel.**panel**(**arrays*, *num=50*, *dists=None*, *order=0*, *kind='linear'*)

Interpolate an arbitrary collection of 1D arrays.

Parameters

- **num** (*int*) – The number of steps to take, so will be the width (number of cols) of the output array.
- **dists** (*array-like*) – The relative distances between the profiles in the array. Sum used to calculate the output width in pixels if the width argument is None. If not given, the distances are assumed to be equal.
- **order** (*int*) – The order of the interpolation, passed to scipy.ndimage.zoom. Suggestion: 0 for integers and 1 for floats.
- **kind** (*str*) – Will be passed to scipy.interpolate.interp1d, which does the lateral interpolation between samples.

Returns

ndarray. The interpolated panel. Contains NaNs if sizes are non-uniform.

bruges.models.panel.**reconcile**(**arrays*, *order=0*)

Make sure 1D arrays are the same length. If not, stretch them to match the longest.

Parameters

- **arrays** (*ndarray*) – The input arrays.
- **order** (*int*) – The order of the interpolation, passed to scipy.ndimage.zoom. Suggestion: 0 for integers and 1 for floats.

Returns

tuple of ndarrays. The reconciled arrays — all of them are now the same length.

Example

```
>>> a = np.array([2, 6, 7, 7, 3])
>>> b = np.array([3, 7, 3])
>>> reconcile(a, b, order=0)
(array([2, 6, 7, 7, 3]), array([3, 7, 7, 3, 3]))
```

bruges.models.panel.**unreconcile**(*arr*, *sizes*, *dists=None*, *order=0*)

Opposite of reconcile. Restores the various profiles (the reference arrays, e.g. wells) to their original lengths.

Parameters

- **sizes** (*int*) – The relative lengths of the profiles in the array. Default returns the input array.

- **dists** (*array-like*) – The relative distances between the profiles in the array. Sum used to calculate the output width in pixels if the width argument is None. If not given, the distances are assumed to be equal.
- **order** (*int*) – The order of the spline interpolation, from 0 to 3. The default is 0, which gives nearest neighbour interpolation. 1 gives linear interpolation, etc. Use 0 for ints and 1-3 for floats.

Returns

ndarray. The resulting ndarray. The array contains NaNs where there is no interpolated data.

bruges.models.wedge module

bruges.models.wedge.**get_conforming**(*strat*, *thickness*, *conformance*)
Function to deal with top and bottom conforming wedges.

bruges.models.wedge.**get_strat**(*strat*, *thickness*, *kind='nearest'*, *position=1*, *wedge=None*, *zoom_mode='nearest'*)
Take a 'stratigraphy' (either an int, a tuple of ints, or a list-like of floats) and expand or compress it to the required thickness.

kind can be 'nearest', 'linear', 'quadratic', or 'cubic'.

bruges.models.wedge.**get_subwedges**(*target*, *breadth*)
For a binary target (the reference trace of a wedge), create the range of net:gross subwedges. We do this with binary morphologies in the following way:

- Erode the 'secondary' component (whatever appears second in the target array) one step at a time, until there is nothing left and the resulting trace contains only the primary.
- Dilate the secondary component until there is nothing left of the primary and the resulting trace contains only the secondary.
- Arrange the traces in order, starting with all primary, and ending in all secondary. The target trace will be somewhere in between, but not necessarily in the middle.

Returns a 2D array with one target wedge trace per section.

Parameters

- **target** (*array*) – A 1D array length N, the 'reference' trace for the wedge. The reference trace has thickness '1' in the wedge model. This trace must be 'binary' — i.e. it must contain exactly 2 unique values.
- **breadth** (*int*) – How many new reference traces should be in the output.

Returns

tuple (ndarray, ndarray, int). The ndarray has shape N x breadth. It represents one target wedge trace per section in 'breadth'. The integer is the position of the target trace in the ndarray's second dimension.

bruges.models.wedge.**pad_func**(*before*, *after*)
Padding function. Operates on vector *in place*, per the np.pad documentation.

bruges.models.wedge.**wedge**(*depth=(30, 40, 30)*, *width=(10, 80, 10)*, *breadth=None*, *strat=(0, 1, 2)*, *thickness=(0.0, 1.0)*, *mode='linear'*, *conformance='both'*)
Generate a wedge model.

Parameters

- **depth** (*int or tuple*) – The vertical size of the model. If a 3-tuple, then each element corresponds to a layer. If an integer, then each layer of the model will be 1/3 of the thickness. Note that if the 'right' wedge thickness is more than 1, then the total thickness will be greater than this value.
- **width** (*int or tuple*) – The width of the model. If a 3-tuple, then each element corresponds to a 'zone' (left, middle, right). If an integer, then the zones will be 10%, 80% and 10% of the width, respectively.
- **breadth** (*None or int*) – Not implemented. Raises an error.
- **strat** (*tuple*) – Stratigraphy above, in, and below the wedge. This is the 'reference' stratigraphy. If you give integers, you get 'solid' layers containing those numbers. If you give arrays, you will get layers of those arrays, expanded or squeezed into the layer thicknesses implied in *depth*.
- **thickness** (*tuple*) – The wedge thickness on the left and on the right. Default is (0.0, 1.0) so the wedge will be thickness 0 on the left and the wedge thickness specified in the depth argument on the right. If the thickness are equal, you'll have a flat, layer-cake model.
- **mode** (*str or function*) – What kind of interpolation to use. Default: 'linear'. Other options are 'sigmoid', which makes a clinoform-like body, 'root', which makes a steep-sided bowl shape like the edge of a channel, and 'power', which makes a less steep-sided bowl shape. If you pass a function, give it an API like np.linspace: f(start, stop, num), where start is the left thickness, stop is the right thickness, and num is the width of the middle (wedge) 'zone'.
- **conformance** (*str*) – 'top', 'bottom', or 'both' (the default). How you want the layers inside the wedge to behave. For top and bottom conformance, if the layer needs to be thicker than the reference.

Returns

A tuple containing the 2D wedge model, the top 'horizon', the base 'horizon', and the position at which the wedge has thickness 1 (i.e. is the thickness specfied by the middle layer depth and/or strat).

Return type namedtuple[ndarray, ndarray, ndarray, int]

Module contents

bruges.noise package

Submodules

bruges.noise.noise module

Noise.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.noise.noise.**noise_db**(*a*, *snr*)

Takes an array of seismic amplitudes and SNR in dB.

Args: a (array) : seismic amplitude array. snr (int): signal to noise ratio.

Returns: Noise array, the same shape as the input.

Note: it does *not* return the input array with the noise added.

Module contents

bruges.petrophysics package

Submodules

bruges.petrophysics.petrophysics module

petrophysics.py

copyright 2018 Agile Geoscience

license Apache 2.0

bruges.petrophysics.petrophysics.**density_to_porosity**(*rho*, *rho_matrix*, *rho_fluid*)
Get density from a porosity log. Typical values:

rho_matrix (sandstone) : 2650 kg/m^3 rho_matrix (limestome): 2710 kg/m^3 rho_matrix (dolomite): 2876 kg/m^3 rho_matrix (anyhydrite): 2977 kg/m^3 rho_matrix (salt): 20320 kg/m^3

rho_fluid (fresh water): 1000 kg/m^3 rho_fluid (salt water): 1100 kg/m^3

See wiki.aapg.org/Density-neutron_log_porosity.

Parameters

- **rho** (*ndarray*) – The bulk density log or RHOB.
- **rho_matrix** (*float*) –
- **rho_fluid** (*float*) –

Returns Estimate of porosity as a volume fraction.

bruges.petrophysics.petrophysics.**density_to_velocity**(*rho*, *alpha=310*, *beta=0.25*, *fps=False*)
Computes Gardner's density prediction from P-wave velocity.

Parameters

- **rho** (*ndarray*) – Density in kg/m^3.
- **alpha** (*float*) – The factor, 310 for m/s and 230 for fps.
- **beta** (*float*) – The exponent, usually 0.25.
- **fps** (*bool*) – Set to true for FPS and the equation will use the typical value for alpha. Overrides value for alpha, so if you want to use your own alpha, regardless of units, set this to False.

Returns Vp estimate in m/s.

Return type ndarray

bruges.petrophysics.petrophysics.**error_flag**(*pred*, *actual*, *dev=1.0*, *method=1*)
Calculate the difference between a predicted and an actual curve and return a log flagging large differences based on a user-defined distance (in standard deviation units) from the mean difference

Matteo Niccoli, October 2018

Parameters

- **predicted** (*ndarray*) –
- **actual** (*ndarray*) –
- **dev** (*float*) –
- **calcluation method** (*error*) – 1: difference between curves larger than mean difference plus dev 2: curve slopes have opposite sign 3: curve slopes of opposite sign OR difference larger than mean plus dev

Returns: flag (ndarray) = error flag curve

bruges.petrophysics.petrophysics.**gardner**(*vp*, *alpha=310*, *beta=0.25*, *fps=False*)

Computes Gardner's density prediction from P-wave velocity.

Parameters

- **vp** (*ndarray*) – P-wave velocity in m/s.
- **alpha** (*float*) – The factor, 310 for m/s and 230 for ft/s.
- **beta** (*float*) – The exponent, usually 0.25.
- **fps** (*bool*) – Set to true for FPS and the equation will use the typical value for alpha. Overrides value for alpha, so if you want to use you own alpha, regardless of units, set this to False.

Returns RHOB estimate in kg/m^3.

Return type ndarray

bruges.petrophysics.petrophysics.**gardner_param**(*vp*, *rhob*)

Finds optimal alpha and beta parameters for the gardner

Volodymyr Vragov, October 2018

Parameters

- **rho** (*ndarray*) – Density.
- **vp** (*ndarray*) – P-wave velocity.

Returns The factor. beta (float): The exponent, usually 0.25.

Return type alpha (float)

bruges.petrophysics.petrophysics.**inverse_gardner**(*rho*, *alpha=310*, *beta=0.25*, *fps=False*)

Computes Gardner's density prediction from P-wave velocity.

Parameters

- **rho** (*ndarray*) – Density in kg/m^3.
- **alpha** (*float*) – The factor, 310 for m/s and 230 for fps.
- **beta** (*float*) – The exponent, usually 0.25.
- **fps** (*bool*) – Set to true for FPS and the equation will use the typical value for alpha. Overrides value for alpha, so if you want to use your own alpha, regardless of units, set this to False.

Returns Vp estimate in m/s.

Return type ndarray

bruges.petrophysics.petrophysics.**optimize_inverse_gardner**(*rho*, *alpha*, *beta*)
Wrapper function to pass inverse_gardner to scipy.curve_fit to get optimal alpha and beta parameters

Matteo Niccoli and Volodymyr Vragov, October 2018

Parameters

- **rho** (*ndarray*) – Density.
- **alpha** (*float*) – The factor.
- **beta** (*float*) – The exponent.

Returns this is passed to scipy.curve_fit as, for example: popt_synt, pcov = scipy.curve_fit(optimize_inverse_gardner, rho, vp) For a ull example, please read: mycarta.wordpress.com/2018/10/28/geophysics-python-sprint-2018-day-2-and-beyond-part-i

Return type inverse_gardner

bruges.petrophysics.petrophysics.**porosity_to_density**(*phi*, *rho_matrix*, *rho_fluid*)
Get density from a porosity log. Typical values:

rho_matrix (sandstone) : 2650 kg/m^3 rho_matrix (limestome): 2710 kg/m^3 rho_matrix (dolomite): 2876 kg/m^3 rho_matrix (anyhydrite): 2977 kg/m^3 rho_matrix (salt): 20320 kg/m^3

rho_fluid (fresh water): 1000 kg/m^3 rho_fluid (salt water): 1100 kg/m^3

See wiki.aapg.org/Density-neutron_log_porosity.

Parameters

- **phi** (*ndarray*) – The porosity log.
- **rho_matrix** (*float*) –
- **rho_fluid** (*float*) –

Returns Estimate of bulk density, rho.

bruges.petrophysics.petrophysics.**slowness_to_velocity**(*slowness*)
Convert a slowness log in μs per unit depth, to velocity in unit depth per second.

Parameters **slowness** (*ndarray*) – A value or sequence of values.

Returns The velocity.

Return type ndarray

bruges.petrophysics.petrophysics.**velocity_to_density**(*vp*, *alpha=310*, *beta=0.25*, *fps=False*)
Computes Gardner's density prediction from P-wave velocity.

Parameters

- **vp** (*ndarray*) – P-wave velocity in m/s.
- **alpha** (*float*) – The factor, 310 for m/s and 230 for ft/s.
- **beta** (*float*) – The exponent, usually 0.25.
- **fps** (*bool*) – Set to true for FPS and the equation will use the typical value for alpha. Overrides value for alpha, so if you want to use you own alpha, regardless of units, set this to False.

Returns RHOB estimate in kg/m^3.

Return type ndarray

bruges.petrophysics.petrophysics.**velocity_to_slowness**(*slowness*)
Convert a slowness log in μs per unit depth, to velocity in unit depth per second.

Parameters **slowness** (*ndarray*) – A value or sequence of values.

Returns The velocity.

Return type ndarray

Module contents

bruges.reflection package

Submodules

bruges.reflection.reflection module

Various reflectivity algorithms.

copyright 2018 Agile Geoscience

license Apache 2.0

bruges.reflection.reflection.**acoustic_reflectivity**(*vp*, *rho*)
The acoustic reflectivity, given Vp and RHOB logs.

Parameters

- **vp** (*ndarray*) – The P-wave velocity.
- **rho** (*ndarray*) – The bulk density.

Returns The reflectivity coefficient series.

Return type ndarray

bruges.reflection.reflection.**akirichards**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*)
The Aki-Richards approximation to the reflectivity.

This is the formulation from Avseth et al., Quantitative seismic interpretation, Cambridge University Press, 2006. Adapted for a 4-term formula. See http://subsurfwiki.org/wiki/Aki-Richards_equation.

Returns the complex reflectivity.

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.
- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.
- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.
- **terms** (*bool*) – Whether or not to return a tuple of the terms of the equation. The first term is the acoustic impedance.

Returns

ndarray. The Aki-Richards approximation for P-P reflectivity at the interface. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m array (for float inputs and one angle), or an n x m array (for array inputs and an array of angles).

bruges.reflection.reflection.**akirichards_alt**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*)

This is another formulation of the Aki-Richards solution. See http://subsurfwiki.org/wiki/Aki-Richards_equation

Returns the complex reflectivity.

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.
- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.
- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.
- **terms** (*bool*) – Whether or not to return a tuple of the terms of the equation. The first term is the acoustic impedance.

Returns

ndarray. The Aki-Richards approximation for P-P reflectivity at the interface. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m array (for float inputs and one angle), or an n x m array (for array inputs and an array of angles).

bruges.reflection.reflection.**blangy**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *d1=0*, *e1=0*, *d2=0*, *e2=0*, *theta1=0*)

Implements the Blangy equation with the same interface as the other reflectivity equations. Wraps bruges.anisotropy.blangy(), which you may prefer to use directly.

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.
- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.
- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **d1** (*ndarray*) – The upper delta; float or 1D array length m.
- **e1** (*ndarray*) – The upper epsilon; float or 1D array length m.
- **d2** (*ndarray*) – The lower delta; float or 1D array length m.
- **e2** (*ndarray*) – The lower epsilon; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.

Returns

ndarray. The Blangy approximation for P-P reflectivity at the interface. Wraps *anisotropy.blangy()*.

bruges.reflection.reflection.**bortfeld**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*)

Compute Bortfeld approximation with three terms. http://sepwww.stanford.edu/public/docs/sep111/marie2/paper_html/node2.html Real numbers only.

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.
- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.
- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.
- **terms** (*bool*) – Whether or not to return a tuple of the terms of the equation. The first term is the acoustic impedance.

Returns

ndarray. The 3-term Bortfeld approximation for P-P reflectivity at the interface. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m array (for float inputs and one angle), or an n x m array (for array inputs and an array of angles).

bruges.reflection.reflection.**bortfeld2**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*)

The 2-term Bortfeld approximation for ava analysis. Wraps *shuey()*. Deprecated, use *bortfeld()* instead.

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.
- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.
- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.
- **terms** (*bool*) – Whether or not to return a tuple of the terms of the equation. The first term is the acoustic impedance.

Returns

ndarray. The 2-term Bortfeld approximation for P-P reflectivity at the interface. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m array (for float inputs and one angle), or an n x m array (for array inputs and an array of angles).

bruges.reflection.reflection.**bortfeld3**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*)

bruges.reflection.reflection.**critical_angles**(*vp1*, *vp2*, *vs2=None*)
Compute critical angle at an interface

Given the upper (vp1) and lower (vp2) velocities at an interface. If you want the PS-wave critical angle as well, pass vs2 as well.

Parameters

- **vp1** (*ndarray*) – Upper layer P-wave velocity.
- **vp2** (*ndarray*) – Lower layer P-wave velocity.
- **vs2** (*ndarray*) – Lower layer S-wave velocity (optional).

Returns

The first and second critical angles at the interface, in degrees. If there isn't a critical angle, it returns np.nan.

Return type tuple

bruges.reflection.reflection.**fatti**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*)
Compute reflectivities with Fatti's formulation of the Aki-Richards equation, which does not account for the critical angle. See Fatti et al. (1994), Geophysics 59 (9). Real numbers only.

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.
- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.
- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.
- **terms** (*bool*) – Whether or not to return a tuple of the terms of the equation. The first term is the acoustic impedance.

Returns

ndarray. The Fatti approximation for P-P reflectivity at the interface. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m array (for float inputs and one angle), or an n x m array (for array inputs and an array of angles).

bruges.reflection.reflection.**hilterman**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*)
Not recommended, only seems to match Zoeppritz to about 10 deg.

Hilterman (1989) approximation from Mavko et al. Rock Physics Handbook. According to Dvorkin: "arguably the simplest and a very convenient [approximation]." At least for small angles and small contrasts. Real numbers only.

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.

- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.
- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.
- **terms** (*bool*) – Whether or not to return a tuple of the terms of the equation. The first term is the acoustic impedance.

Returns

ndarray. The Hilterman approximation for P-P reflectivity at the interface. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m array (for float inputs and one angle), or an n x m array (for array inputs and an array of angles).

bruges.reflection.reflection.**preprocess**(*func*)

Decorator to preprocess arguments for the reflectivity equations.

Takes a reflectivity function requiring Vp, Vs, and RHOB for 2 rocks (upper and lower), plus incidence angle theta, plus kwargs. Returns that function with some arguments transformed.

bruges.reflection.reflection.**reflection_phase**(*reflectivity*)

Compute the phase of the reflectivity. Returns an array (or float) of the phase, in positive multiples of 180 deg or pi rad. So 1 is opposite phase. A value of 1.1 would be +1.1 imes pi rad.

Parameters **reflectivity** (*ndarray*) – The reflectivity, eg from *zoeppritz()*.

Returns The phase, strictly positive

Return type ndarray

bruges.reflection.reflection.**reflectivity**(*vp*, *vs*, *rho*, *theta=0*, *method='zoeppritz_rpp'*)

Offset reflectivity, given Vp, Vs, rho, and offset.

Computes 'upper' and 'lower' intervals from the three provided arrays, then passes the result to the specified method to compute reflection coefficients.

For acoustic reflectivity, either use the *acoustic_reflectivity()* function, or call *reflectivity()* passing any log as Vs, e.g. just give the Vp log twice (it won't be used anyway):

reflectivity(vp, vp, rho)

For anisotropic reflectivity, use either *anisotropy.blangy()* or *anisotropy.ruger()*.

Parameters

- **vp** (*ndarray*) – The P-wave velocity; float or 1D array length m.
- **vs** (*ndarray*) – The S-wave velocity; float or 1D array length m.
- **rho** (*ndarray*) – The density; float or 1D array length m.
- **theta** (*ndarray*) – The incidence angle; float or 1D array length n.
- **method** (*str*) – The reflectivity equation to use; one of:
  - 'scattering_matrix': scattering_matrix
  - 'zoeppritz_element': zoeppritz_element
  - 'zoeppritz': zoeppritz
  - 'zoeppritz_rpp': zoeppritz_rpp
  - 'akirichards': akirichards
  - 'akirichards_alt': akirichards_alt

- 'fatti': fatti
- 'shuey': shuey
- 'bortfeld': bortfeld
- 'hilterman': hilterman

- **Notes** –
  - scattering_matrix gives the full solution
  - zoeppritz_element gives a single element which you specify
  - zoeppritz returns RPP element only; use zoeppritz_rpp instead
  - zoeppritz_rpp is faster than zoeppritz or zoeppritz_element

Returns

ndarray. The result of running the specified method on the inputs. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m-1 array (for float inputs and one angle), or an m-1 x n array (for array inputs and an array of angles).

bruges.reflection.reflection.**scattering_matrix**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*)

Full Zoeppritz solution, considered the definitive solution. Calculates the angle dependent p-wave reflectivity of an interface between two mediums.

Originally written by: Wes Hamlyn, vectorized by Agile.

Returns the complex reflectivity.

Parameters

- **vp1** (*float*) – The upper P-wave velocity.
- **vs1** (*float*) – The upper S-wave velocity.
- **rho1** (*float*) – The upper layer's density.
- **vp2** (*float*) – The lower P-wave velocity.
- **vs2** (*float*) – The lower S-wave velocity.
- **rho2** (*float*) – The lower layer's density.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.

Returns

ndarray. The exact Zoeppritz solution for all modes at the interface. A 4x4 array representing the scattering matrix at the incident angle theta1.

bruges.reflection.reflection.**shuey**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*, *return_gradient=False*)

Compute Shuey approximation with 3 terms. http://subsurfwiki.org/wiki/Shuey_equation

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.
- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.

- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.
- **terms** (*bool*) – Whether or not to return a tuple of the terms of the equation. The first term is the acoustic impedance.
- **return_gradient** (*bool*) – Whether to return a tuple of the intercept and gradient (i.e. the second term divided by sin^2(theta)).

Returns

ndarray. The Aki-Richards approximation for P-P reflectivity at the interface. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m array (for float inputs and one angle), or an n x m array (for array inputs and an array of angles).

bruges.reflection.reflection.**shuey2**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*)
Compute Shuey approximation with 2 terms. Wraps *shuey()*. Deprecated, use *shuey()* instead.

bruges.reflection.reflection.**shuey3**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *terms=False*)
Compute Shuey approximation with 3 terms. Wraps *shuey()*. Deprecated, use *shuey()* instead.

bruges.reflection.reflection.**vectorize**(*func*)
Decorator to make sure the inputs are arrays. We also add a dimension to theta to make the functions work in an 'outer product' way.

Takes a reflectivity function requiring Vp, Vs, and RHOB for 2 rocks (upper and lower), plus incidence angle theta, plus kwargs. Returns that function with the arguments transformed to ndarrays.

bruges.reflection.reflection.**zoeppritz**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*)
Returns the PP reflection coefficients from the Zoeppritz scattering matrix. Wraps zoeppritz_element().

Returns the complex reflectivity.

Parameters

- **vp1** (*float*) – The upper P-wave velocity.
- **vs1** (*float*) – The upper S-wave velocity.
- **rho1** (*float*) – The upper layer's density.
- **vp2** (*float*) – The lower P-wave velocity.
- **vs2** (*float*) – The lower S-wave velocity.
- **rho2** (*float*) – The lower layer's density.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.

Returns

ndarray. Array length n of the exact Zoeppritz solution for the specified modes at the interface, at the incident angle theta1.

bruges.reflection.reflection.**zoeppritz_element**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*, *element='PdPu'*)
Returns any mode reflection coefficients from the Zoeppritz scattering matrix. Pass in the mode as element, e.g. 'PdSu' for PS.

Wraps scattering_matrix().

Returns the complex reflectivity.

Parameters

- **vp1** (*float*) – The upper P-wave velocity.
- **vs1** (*float*) – The upper S-wave velocity.
- **rho1** (*float*) – The upper layer's density.
- **vp2** (*float*) – The lower P-wave velocity.
- **vs2** (*float*) – The lower S-wave velocity.
- **rho2** (*float*) – The lower layer's density.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.
- **element** (*str*) – The name of the element to return, must be one of: 'PdPu', 'SdPu', 'PuPu', 'SuPu', 'PdSu', 'SdSu', 'PuSu', 'SuSu', 'PdPd', 'SdPd', 'PuPd', 'SuPd', 'PdSd', 'SdSd', 'PuSd', 'SuSd'.

Returns

ndarray. Array length n of the exact Zoeppritz solution for the specified modes at the interface, at the incident angle theta1.

bruges.reflection.reflection.**zoeppritz_rpp**(*vp1*, *vs1*, *rho1*, *vp2*, *vs2*, *rho2*, *theta1=0*)

Exact Zoeppritz from expression.

This is useful because we can pass arrays to it, which we can't do to scattering_matrix().

Dvorkin et al. (2014). Seismic Reflections of Rock Properties. Cambridge.

Returns the complex reflectivity.

Parameters

- **vp1** (*ndarray*) – The upper P-wave velocity; float or 1D array length m.
- **vs1** (*ndarray*) – The upper S-wave velocity; float or 1D array length m.
- **rho1** (*ndarray*) – The upper layer's density; float or 1D array length m.
- **vp2** (*ndarray*) – The lower P-wave velocity; float or 1D array length m.
- **vs2** (*ndarray*) – The lower S-wave velocity; float or 1D array length m.
- **rho2** (*ndarray*) – The lower layer's density; float or 1D array length m.
- **theta1** (*ndarray*) – The incidence angle; float or 1D array length n.

Returns

ndarray. The exact Zoeppritz solution for P-P reflectivity at the interface. Will be a float (for float inputs and one angle), a 1 x n array (for float inputs and an array of angles), a 1 x m array (for float inputs and one angle), or an n x m array (for array inputs and an array of angles).

Module contents

bruges.rockphysics package

Submodules

bruges.rockphysics.anisotropy module

Anisotropy effects.

Backus anisotropy is from thin layers.

Hudson anisotropy is from crack defects.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.rockphysics.anisotropy.**backus**(*vp*, *vs*, *rho*, *lb*, *dz*)

Backus averaging. Using Liner's algorithm (2014; see Notes).

Parameters

- **vp** (*ndarray*) – P-wave interval velocity.
- **vs** (*ndarray*) – S-wave interval velocity.
- **rho** (*ndarray*) – Bulk density.
- **lb** (*float*) – The Backus averaging length in m.
- **dz** (*float*) – The depth sample interval in m.

Returns

the smoothed logs: vp, vs, plus rho. Useful for computing other elastic parameters at a seismic scale.

Return type namedtuple

Notes

Liner, C (2014), Long-wave elastic attenuation produced by horizontal layering. The Leading Edge, June 2014, p 634-638.

bruges.rockphysics.anisotropy.**backus_parameters**(*vp*, *vs*, *rho*, *lb*, *dz*)

Intermediate parameters for Backus averaging. This is expected to be a private function. You probably want backus() and not this.

Parameters

- **vp** (*ndarray*) – P-wave interval velocity.
- **vs** (*ndarray*) – S-wave interval velocity.
- **rho** (*ndarray*) – Bulk density.
- **lb** (*float*) – The Backus averaging length in m.
- **dz** (*float*) – The depth sample interval in m.

Returns Liner's 5 intermediate parameters: A, C, F, L and M.

Return type tuple

Notes

Liner, C (2014), Long-wave elastic attenuation produced by horizontal layering. The Leading Edge, June 2014, p 634-638.

bruges.rockphysics.anisotropy.**backus_quality_factor**(*vp*, *vs*, *rho*, *lb*, *dz*)
Compute Qp and Qs from Liner (2014) equation 10.

Parameters

- **vp** (*ndarray*) – P-wave interval velocity.
- **vs** (*ndarray*) – S-wave interval velocity.
- **rho** (*ndarray*) – Bulk density.
- **lb** (*float*) – The Backus averaging length in m.
- **dz** (*float*) – The depth sample interval in m.

Returns Qp and Qs.

Return type namedtuple

bruges.rockphysics.anisotropy.**blangy**(*vp1*, *vs1*, *rho1*, *d1*, *e1*, *vp0*, *vs0*, *rho0*, *d0*, *e0*, *theta*)
Blangy, JP, 1994, AVO in transversely isotropic media-An overview. Geophysics 59 (5), 775-781. DOI: 10.1190/1.1443635

Provide Vp, Vs, rho, delta, epsilon for the upper and lower intervals, and theta, the incidence angle.

Parameters

- **vp1** – The p-wave velocity of the upper medium.
- **vs1** – The s-wave velocity of the upper medium.
- **rho1** – The density of the upper medium.
- **d1** – Thomsen's delta of the upper medium.
- **e1** – Thomsen's epsilon of the upper medium.
- **vp0** – The p-wave velocity of the lower medium.
- **vs0** – The s-wave velocity of the lower medium.
- **rho0** – The density of the lower medium.
- **d0** – Thomsen's delta of the lower medium.
- **e0** – Thomsen's epsilon of the lower medium.
- **theta** – A scalar [degrees].

Returns the isotropic and anisotropic reflectivities in a tuple. The isotropic result is equivalent to Aki-Richards.

TODO Use rocks.

bruges.rockphysics.anisotropy.**crack_density**(*porosity*, *aspect*)
Returns crack density from porosity and aspect ratio, phi and alpha respectively in the unnumbered equation between 15.40 and 15.41 in Dvorkin et al. 2014.

Parameters

- **porosity** (*float*) – Fractional porosity.
- **aspect** (*float*) – Aspect ratio.

Returns Crack density.

Return type float

bruges.rockphysics.anisotropy.**dispersion_parameter**(*qp*)

Kjartansson (1979). Journal of Geophysical Research, 84 (B9), 4737-4748. DOI: 10.1029/JB084iB09p04737.

bruges.rockphysics.anisotropy.**hudson_delta_G**(*porosity*, *aspect*, *mu*, *lam=None*, *pmod=None*)

The approximate reduction in shear modulus G (or mu) in the direction normal to a set of aligned cracks. Eqn 15.42 in Dvorkin et al (2014).

Parameters

- **porosity** (*float*) – Fractional porosity, phi.
- **aspect** (*float*) – Aspect ratio, alpha.
- **mu** (*float*) – Shear modulus, sometimes called G.
- **lam** (*float*) – Lame's first parameter, lambda.
- **pmod** (*float*) – Compressional modulus, M.

Returns M_inf - M_0 = Delta c_11.

Return type float

bruges.rockphysics.anisotropy.**hudson_delta_M**(*porosity*, *aspect*, *mu*, *lam=None*, *pmod=None*)

The approximate reduction in compressional modulus M in the direction normal to a set of aligned cracks. Eqn 15.40 in Dvorkin et al (2014).

Parameters

- **porosity** (*float*) – Fractional porosity, phi.
- **aspect** (*float*) – Aspect ratio, alpha.
- **mu** (*float*) – Shear modulus, sometimes called G.
- **lam** (*float*) – Lame's first parameter.
- **pmod** (*float*) – Compressional modulus, M.

Returns M_inf - M_0 = Delta c_11.

Return type float

bruges.rockphysics.anisotropy.**hudson_inverse_Q_ratio**(*mu=None*, *pmod=None*, *pr=None*, *vp=None*, *vs=None*, *aligned=True*)

Dvorkin et al. (2014), Eq 15.44 (aligned) and 15.48 (not aligned). You must provide one of the following: *pr*, or *vp* and *vs*, or *mu* and *pmod*.

Parameters

- **mu** (*float*) – Shear modulus, sometimes called G.
- **pmod** (*float*) – Compressional modulus, M.
- **pr** (*float*) – Poisson's ratio, somtimes called v.
- **vp** (*ndarray*) – P-wave interval velocity.
- **vs** (*ndarray*) – S-wave interval velocity.
- **aligned** (*bool*) – Either treats cracks as alligned (Default, True) or assumes defects are randomly oriented (False)

Returns 2Q_s^-1

Return type float

bruges.rockphysics.anisotropy.**hudson_quality_factor**(*porosity*, *aspect*, *mu*, *lam=None*, *pmod=None*)

Returns Q_p and Q_s for cracked media. Equations 15.41 and 15.43 in Dvorkin et al. (2014).

Parameters

- **porosity** (*float*) – Fractional porosity, phi.
- **aspect** (*float*) – Aspect ratio, alpha.
- **mu** (*float*) – Shear modulus, sometimes called G.
- **lam** (*float*) – Lame's first parameter, lambda.
- **pmod** (*float*) – Compressional modulus, M.

Returns Q_p float: Q_s

Return type float

bruges.rockphysics.anisotropy.**ruger**(*vp1*, *vs1*, *rho1*, *d1*, *e1*, *vp2*, *vs2*, *rho2*, *d2*, *e2*, *theta*)

Coded by Alessandro Amato del Monte and (c) 2016 by him https://github.com/aadm/avo_explorer/blob/master/avo_explorer_v2.ipynb

Rüger, A., 1997, P -wave reflection coefficients for transversely isotropic models with vertical and horizontal axis of symmetry: Geophysics, v. 62, no. 3, p. 713–722.

Provide Vp, Vs, rho, delta, epsilon for the upper and lower intervals, and theta, the incidence angle.

Parameters

- **vp1** – The p-wave velocity of the upper medium.
- **vs1** – The s-wave velocity of the upper medium.
- **rho1** – The density of the upper medium.
- **d1** – Thomsen's delta of the upper medium.
- **e1** – Thomsen's epsilon of the upper medium.
- **vp0** – The p-wave velocity of the lower medium.
- **vs0** – The s-wave velocity of the lower medium.
- **rho0** – The density of the lower medium.
- **d0** – Thomsen's delta of the lower medium.
- **e0** – Thomsen's epsilon of the lower medium.
- **theta** – A scalar [degrees].

Returns anisotropic reflectivity.

bruges.rockphysics.anisotropy.**thomsen_parameters**(*vp*, *vs*, *rho*, *lb*, *dz*)

Liner, C, and T Fei (2006). Layer-induced seismic anisotropy from full-wave sonic logs: Theory, application, and validation. Geophysics 71 (6), p D183–D190. DOI:10.1190/1.2356997

Parameters

- **vp** (*ndarray*) – P-wave interval velocity.
- **vs** (*ndarray*) – S-wave interval velocity.
- **rho** (*ndarray*) – Bulk density.
- **lb** (*float*) – The Backus averaging length in m.

- **dz** (*float*) – The depth sample interval in m.

Returns delta, epsilon and gamma.

Return type namedtuple

bruges.rockphysics.bounds module

Bounds on effective elastic modulus. :copyright: 2015 Agile Geoscience :license: Apache 2.0

bruges.rockphysics.bounds.**hashin_shtrikman**(*f*, *k*, *mu*, *modulus='bulk'*)

Hashin-Shtrikman bounds for a mixture of two constituents. The best bounds for an isotropic elastic mixture, which give the narrowest possible range of elastic modulus without specifying anything about the geometries of the constituents.

Parameters

- **f** – list or array of volume fractions (must sum to 1.00 or 100%).
- **k** – bulk modulus of constituents (list or array).
- **mu** – shear modulus of constituents (list or array).
- **modulus** – A string specifying whether to return either the 'bulk' or 'shear' HS bound.

Returns The Hashin Shtrikman (lower, upper) bounds.

Return type namedtuple

Source Berryman, J.G., 1993, Mixture theories for rock properties Mavko, G., 1993, Rock Physics Formulas.

: Written originally by Xingzhou 'Frank' Liu, in MATLAB : modified by Isao Takahashi, 4/27/99, : Translated into Python by Evan Bianco

bruges.rockphysics.bounds.**hill_average**(*f*, *m*)

The Hill average effective elastic modulus, mh of a mixture of N material phases. This is defined as the simple average of the Reuss (lower) and Voigt (upper) bounds.

Parameters

- **f** – list or array of N volume fractions (must sum to 1 or 100).
- **m** – elastic modulus of N constituents (list or array).

Returns: mh: Hill average.

bruges.rockphysics.bounds.**reuss_bound**(*f*, *m*)

The lower bound on the effective elastic modulus of a mixture of N material phases. This is defined at the harmonic average of the constituents. Same as Wood's equation for homogeneous mixed fluids.

Parameters

- **f** – list or array of N volume fractions (must sum to 1 or 100).
- **m** – elastic modulus of N constituents (list or array).

Returns: mr: Reuss lower bound.

bruges.rockphysics.bounds.**voigt_bound**(*f*, *m*)

The upper bound on the effective elastic modulus, mv of a mixture of N material phases. This is defined at the arithmetic average of the constituents.

Parameters

- **f** – list or array of N volume fractions (must sum to 1 or 100).
- **m** – elastic modulus of N constituents (list or array).

Returns: mv: Voigt upper bound.

bruges.rockphysics.elastic module

Elastic impedance.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.rockphysics.elastic.**elastic_impedance**(*vp*, *vs*, *rho*, *theta1*, *k=None*, *normalize=False*, *constants=None*, *use_sin=False*, *rho_term=False*)

Returns the elastic impedance as defined by Connolly, 1999; we are using the formulation reported in Whitcombe et al. (2001). Inputs should be shape m x 1, angles should be n x 1. The result will be m x n.

Parameters

- **vp** (*ndarray*) – The P-wave velocity scalar or 1D array.
- **vs** (*ndarray*) – The S-wave velocity scalar or 1D array.
- **rho** (*ndarray*) – The bulk density scalar or 1D array.
- **theta1** (*ndarray*) – Incident angle(s) in degrees, scalar or array.
- **k** (*float*) – A constant, see Connolly (1999). Default is None, which computes it from Vp and Vs.
- **normalize** (*bool*) – if True, returns the normalized form of Whitcombe.
- **constants** (*tuple*) – A sequence of 3 constants to use for normalization. If you don't provide this, then normalization just uses the means of the inputs. If these are scalars, the result will be the acoustic impedance (see Whitcombe et al., 2001).
- **use_sin** (*bool*) – If True, use sin^2 for the first term, instead of tan^2 (see Connolly).
- **rho_term** (*bool*) – If True, provide alternative form, with Vp factored out; use in place of density in generating synthetics in other software (see Connolly). In other words, the result can be multipled with Vp to get the elastic impedance.

Returns The elastic impedance log at the specficied angle or angles.

Return type ndarray

bruges.rockphysics.fluids module

Fluid properties.

These functions implement equations from Batzle and Wang (1992), seismic properties of pore fluids. GEOPHYSICS, VOL. 57, NO. 11; P. 1396-1408,

copyright 2018 Agile Geoscience

license Apache 2.0

bruges.rockphysics.fluids.**rho_brine**(*temperature*, *pressure*, *salinity*)

The density of NaCl brine, given temperature, pressure, and salinity. The density of pure water is computed from rho_water(). Implements equation 27b from Batzle and Wang (1992).

Use scalars or arrays; if you use arrays, they must be the same size.

Parameters

- **temperature** (*array*) – The temperature in degrees Celsius.
- **pressure** (*array*) – The pressure in pascals.
- **salinity** (*array*) – The weight fraction of NaCl, e.g. 35e-3 for 35 parts per thousand, or 3.5% (the salinity of seawater).

Returns The density in kg/m3.

Return type array

bruges.rockphysics.fluids.**rho_gas**(*temperature*, *pressure*, *molecular_weight*)

Gas density, given temperature (in deg C), pressure (in Pa), and molecular weight.

Parameters

- **temperature** (*array*) – The temperature in degrees Celsius.
- **pressure** (*array*) – The pressure in pascals.
- **molecular_weight** (*array*) – The molecular weight.

Returns The density in kg/m3.

Return type array

bruges.rockphysics.fluids.**rho_water**(*temperature*, *pressure*)

The density of pure water, as a function of temperature and pressure. Implements equation 27a from Batzle and Wang (1992).

Use scalars or arrays; if you use arrays, they must be the same size.

Parameters

- **temperature** (*array*) – The temperature in degrees Celsius.
- **pressure** (*array*) – The pressure in pascals.

Returns The density in kg/m3.

Return type array

bruges.rockphysics.fluids.**v_brine**(*temperature*, *pressure*, *salinity*)

The acoustic velocity of brine, as a function of temperature (deg C), pressure (Pa), and salinity (weight fraction). Implements equation 29 from Batzle and Wang (1992).

Note that this function does not work at pressures above about 100 MPa.

Use scalars or arrays; if you use arrays, they must be the same size.

Parameters

- **temperature** (*array*) – The temperature in degrees Celsius.
- **pressure** (*array*) – The pressure in pascals.
- **salinity** (*array*) – The weight fraction of NaCl, e.g. 35e-3 for 35 parts per thousand, or 3.5% (the salinity of seawater).

Returns The velocity in m/s.

Return type array

bruges.rockphysics.fluids.**v_water**(*temperature*, *pressure*)

The acoustic velocity of pure water, as a function of temperature and pressure. Implements equation 28 from Batzle and Wang (1992), using the coefficients in Table 1.

Note that this function does not work at pressures above about 100 MPa.

Use scalars or arrays; if you use arrays, they must be the same size.

Parameters

- **temperature** (*array*) – The temperature in degrees Celsius.
- **pressure** (*array*) – The pressure in pascals.

Returns The velocity in m/s.

Return type array

bruges.rockphysics.fluids.**wood**(*Kf1*, *Kf2*, *Sf1*)

Wood's equation, per equation 35b in Batzle and Wang (1992).

bruges.rockphysics.fluidsub module

copyright 2015 Agile Geoscience

license Apache 2.0

fluidsub.py

Calculates various parameters for fluid substitution from Vp, Vs, and rho

Matt Hall, Evan Bianco, July 2014

Using http://www.subsurfwiki.org/wiki/Gassmann_equation

The algorithm is from Avseth et al (2006), per the wiki page.

Informed by Smith et al, Geophysics 68(2), 2003.

At some point we should do Biot too, per Russell... http://cseg.ca/symposium/archives/2012/presentations/Biot_Gassmann_and_me.pdf

bruges.rockphysics.fluidsub.**avseth_fluidsub**(*vp*, *vs*, *rho*, *phi*, *rhof1*, *rhof2*, *kmin*, *kf1*, *kf2*)

Naive fluid substitution from Avseth et al, section 1.3.1. Bulk modulus of fluid 1 and fluid 2 are already known, and the bulk modulus of the dry frame, Kmin, is known. Use SI units.

Parameters

- **vp** (*float*) – P-wave velocity
- **vs** (*float*) – S-wave velocity
- **rho** (*float*) – bulk density
- **phi** (*float*) – porosity (volume fraction, e.g. 0.20)
- **rhof1** (*float*) – bulk density of original fluid (base case)
- **rhof2** (*float*) – bulk density of substitute fluid (subbed case)
- **kmin** (*float*) – bulk modulus of solid mineral (s)

- **kf1** (*float*) – bulk modulus of original fluid
- **kf2** (*float*) – bulk modulus of substitute fluid

Returns Vp, Vs, and rho for the substituted case

Return type Tuple

bruges.rockphysics.fluidsub.**avseth_gassmann**(*ksat1*, *kf1*, *kf2*, *k0*, *phi*)

Applies the inverse Gassmann's equation to calculate the rock bulk modulus saturated with fluid with bulk modulus. Requires as inputs the insitu fluid bulk modulus, the insitu saturated rock bulk modulus, the mineral matrix bulk modulus and the porosity.

Parameters

- **ksat1** (*float*) – initial saturated rock bulk modulus.
- **kf1** (*float*) – initial fluid bulk modulus.
- **kf2** (*float*) – final fluid bulk modulus.
- **k0** (*float*) – mineral bulk modulus.
- **phi** (*float*) – porosity.

Returns final saturated rock bulk modulus.

Return type ksat2 (float)

bruges.rockphysics.fluidsub.**rhogas**(*gravity*, *temp*, *pressure*)

From http://www.spgindia.org/geohorizon/jan_2006/dhananjay_paper.pdf

bruges.rockphysics.fluidsub.**rhosat**(*phi*, *sw*, *rhomin*, *rhow*, *rhohc*)

Density of partially saturated rock.

bruges.rockphysics.fluidsub.**smith_fluidsub**(*vp*, *vs*, *rho*, *phi*, *rhow*, *rhohc*, *sw*, *swnew*, *kw*, *khc*, *kclay*, *kqtz*, *vclay*, *rhownew=None*, *rhohcnew=None*, *kwnew=None*, *khcnew=None*)

Naive fluid substitution from Smith et al. 2003. No pressure/temperature correction. Only works for SI units right now.

Parameters

- **vp** (*float*) – P-wave velocity
- **vs** (*float*) – S-wave velocity
- **rho** (*float*) – bulk density
- **phi** (*float*) – porosity (v/v, fraction)
- **rhow** (*float*) – density of water
- **rhohc** (*float*) – density of HC
- **sw** (*float*) – water saturation in base case
- **swnew** (*float*) – water saturation in subbed case
- **kw** (*float*) – bulk modulus of water
- **khc** (*float*) – bulk modulus of hydrocarbon
- **kclay** (*float*) – bulk modulus of clay
- **kqtz** (*float*) – bulk modulus of quartz
- **vclay** (*float*) – Vclay, v/v

- **rhownew** (*float*) – density of water in subbed case (optional)
- **rhohcnew** (*float*) – density of HC in subbed case (optional)
- **kwnew** (*float*) – bulk modulus of water in subbed case (optional)
- **khcnew** (*float*) – bulk modulus of hydrocarbon in subbed case (optional)

Returns Vp, Vs, and rho for the substituted case.

Return type Tuple

bruges.rockphysics.fluidsub.**smith_gassmann**(*kdry*, *k0*, *kf*, *phi*)

Applies the direct Gassmann's equation to calculate the saturated rock bulk modulus from porosity and the dry-rock, mineral and fluid bulk moduli.

Parameters

- **kdry** (*float*) – dry-rock bulk modulus.
- **k0** (*float*) – mineral bulk modulus.
- **kf** (*float*) – fluid bulk modulus.
- **phi** (*float*) – Porosity.

Returns saturated rock bulk modulus.

Return type ksat (float)

bruges.rockphysics.fluidsub.**vels**(*Kdry*, *Gdry*, *K0*, *D0*, *Kf*, *Df*, *phi*)

Calculate velocities and densities of saturated rock via Gassmann equation. Provide all quantities in SI units.

Parameters

- **Gdry** (*Kdry,*) – Dry rock bulk & shear modulus in Pa.
- **G0** (*K0,*) – Mineral bulk & shear modulus in Pa.
- **Df** (*Kf,*) – Fluid bulk modulus in Pa and density in kg/m^3.
- **phi** (*float or array_like*) – Porosity, v/v.

Returns

- **vp, vs** (*float or array_like*) – Saturated rock P- and S-wave velocities in m/s.
- **rho** (*float or array_like*) – Saturated rock density in kg/m^3.
- **K** (*float or array_like*) – Saturated rock bulk modulus in Pa.

bruges.rockphysics.fluidsub.**vrh**(*kclay*, *kqtz*, *vclay*)

Voigt-Reuss-Hill average to find Kmatrix from clay and qtz components. Works for any two components, they don't have to be clay and quartz.

From Smith et al, Geophysics 68(2), 2003.

Parameters

- **kclay** (*float*) – K_clay.
- **kqtz** (*float*) – K_quartz.
- **vclay** (*float*) – V_clay.

Returns Kvrh, also known as Kmatrix.

bruges.rockphysics.moduli module

moduli.py

Converts between various acoustic/eslatic parameters, and provides a way to calculate all the elastic moduli from Vp, Vs, and rho.

Created June 2014, Matt Hall

Using equations http://www.subsurfwiki.org/wiki/Elastic_modulus from Mavko, G, T Mukerji and J Dvorkin (2003), The Rock Physics Handbook, Cambridge University Press.

bruges.rockphysics.moduli.**bulk**(*vp=None*, *vs=None*, *rho=None*, *mu=None*, *lam=None*, *youngs=None*, *pr=None*, *pmod=None*)

Computes bulk modulus given either Vp, Vs, and rho, or any two elastic moduli (e.g. lambda and mu, or Young's and P moduli). SI units only.

Parameters

- **vs, and rho**(*vp,*) –
- **any 2 from lam, mu, youngs, pr, and pmod**(*or*) –

Returns Bulk modulus in pascals, Pa

bruges.rockphysics.moduli.**lam**(*vp=None*, *vs=None*, *rho=None*, *pr=None*, *mu=None*, *youngs=None*, *bulk=None*, *pmod=None*)

Computes lambda given either Vp, Vs, and rho, or any two elastic moduli (e.g. bulk and mu, or Young's and P moduli). SI units only.

Parameters

- **vs, and rho**(*vp,*) –
- **any 2 from bulk, mu, youngs, pr, and pmod**(*or*) –

Returns Lambda in pascals, Pa

bruges.rockphysics.moduli.**moduli_dict**(*vp*, *vs*, *rho*)

Computes elastic moduli given Vp, Vs, and rho. SI units only.

Parameters **Vs, and rho**(*Vp,*) –

Returns A dict of elastic moduli, plus P-wave impedance.

bruges.rockphysics.moduli.**mu**(*vp=None*, *vs=None*, *rho=None*, *pr=None*, *lam=None*, *youngs=None*, *bulk=None*, *pmod=None*)

Computes shear modulus given either Vp, Vs, and rho, or any two elastic moduli (e.g. lambda and bulk, or Young's and P moduli). SI units only.

Parameters

- **vs, and rho**(*vp,*) –
- **any 2 from lam, bulk, youngs, pr, and pmod**(*or*) –

Returns Shear modulus in pascals, Pa

bruges.rockphysics.moduli.**pmod**(*vp=None*, *vs=None*, *rho=None*, *pr=None*, *mu=None*, *lam=None*, *youngs=None*, *bulk=None*)

Computes P-wave modulus given either Vp, Vs, and rho, or any two elastic moduli (e.g. lambda and mu, or Young's and bulk moduli). SI units only.

Parameters

- **vs, and rho** (*vp,*) –
- **any 2 from lam, mu, youngs, pr, and bulk** (*or*) –

Returns P-wave modulus in pascals, Pa

bruges.rockphysics.moduli.**pr**(*vp=None*, *vs=None*, *rho=None*, *mu=None*, *lam=None*, *youngs=None*, *bulk=None*, *pmod=None*)

Computes Poisson ratio given either Vp, Vs, and rho, or any two elastic moduli (e.g. lambda and mu, or Young's and P moduli). SI units only.

Parameters

- **vs, and rho** (*vp,*) –
- **any 2 from lam, mu, youngs, bulk, and pmod** (*or*) –

Returns Poisson's ratio, dimensionless

bruges.rockphysics.moduli.**vp**(*youngs=None*, *vs=None*, *rho=None*, *mu=None*, *lam=None*, *bulk=None*, *pr=None*, *pmod=None*)

Computes Vp given bulk density and any two elastic moduli (e.g. lambda and mu, or Young's and P moduli). SI units only.

Parameters

- **2 from lam, mu, youngs, pr, pmod, bulk** (*Any*) –
- **Rho** –

Returns Vp in m/s

bruges.rockphysics.moduli.**vs**(*youngs=None*, *vp=None*, *rho=None*, *mu=None*, *lam=None*, *bulk=None*, *pr=None*, *pmod=None*)

Computes Vs given bulk density and shear modulus. SI units only.

Parameters

- **Mu** –
- **Rho** –

Returns Vs in m/s

bruges.rockphysics.moduli.**youngs**(*vp=None*, *vs=None*, *rho=None*, *mu=None*, *lam=None*, *bulk=None*, *pr=None*, *pmod=None*)

Computes Young's modulus given either Vp, Vs, and rho, or any two elastic moduli (e.g. lambda and mu, or bulk and P moduli). SI units only.

Parameters

- **vs, and rho** (*vp,*) –
- **any 2 from lam, mu, bulk, pr, and pmod** (*or*) –

Returns Young's modulus in pascals, Pa

bruges.rockphysics.rockphysicsmodels module

rockphysicsmodels.py

A bunch of rock physics models. References are mentioned in docstrings of individual functions. Docstrings follow numpy/scipy convention.

Alessandro Amato del Monte, March 2019

bruges.rockphysics.rockphysicsmodels.**constant_cement**(*K0*, *G0*, *phi*, *phi_cem=0.38*, *phi_c=0.4*, *Cn=8.6*, *Kc=37*, *Gc=45*, *scheme=2*)

Constant cement model, Avseth et al. (2000). This model describes the elastic behaviour (K=bulk and G=shear moduli) of high porosity dry sand with a certain initial cementation by interpolating with the lower Hashin-Shtrikman bound the two end members at zero porosity and critical porosity. The zero porosity end member has K and G equal to mineral. The high porosity end member has K and G given by the Contact Cement model. It is assumed that the porosity reduction is due to non-cementing material filling in the available pore space.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **phi_cem** (*float, optional*) – Porosity at initial cementation. Default: 0.38.
- **phi_c** (*float, optional*) – Critical porosity. Default: 0.4.
- **Cn** (*float, optional*) – Coordination number. Default: 8.6.
- **Gc** (*Kc,*) – Cement bulk & shear modulus in GPa. Default: 37, 45.
- **scheme** (*int, optional*) – Cementation scheme, can be either 1 or 2: 1: cement deposited at grain contacts 2: cement as uniform layer around grains. Default: 2.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Dvorkin et al. (2014), Seismic Reflections of Rock Properties, Cambridge University Press (p.30-31)

bruges.rockphysics.rockphysicsmodels.**contact_cement**(*K0*, *G0*, *phi*, *phi_c=0.4*, *Cn=8.6*, *Kc=37*, *Gc=45*, *scheme=2*)

Contact cement or cemented sand model,. This model describes the elastic behaviour (K=bulk and G=shear moduli) of dry sand where cement is deposited at grain contacts. The cement properties can be modified as well as the type of cementation scheme.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **phi_c** (*float, optional*) – Critical porosity. Default: 0.4
- **Cn** (*float, optional*) – Coordination number Default: 8.6.
- **Gc** (*Kc,*) – Cement bulk & shear modulus in GPa. Default: 37, 45.
- **scheme** (*int, optional*) – Cementation scheme, can be either 1 or 2: 1: cement deposited at grain contacts 2: cement as uniform layer around grains. Default: 2.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Dvorkin-Nur (1996), Elasticity of High-Porosity Sandstones: Theory for Two North Sea Data Sets. Geophysics 61, no. 5 (1996). Mavko et al. (2009), The Rock Physics Handbook, Cambridge University Press (p.255)

bruges.rockphysics.rockphysicsmodels.**critical_porosity**(*K0*, *G0*, *phi*, *phi_c=0.4*)

Critical porosity model. This model describes the elastic behaviour (K=bulk and G=shear moduli) of dry sand for porosities below the critical porosity (phi_c). Above phi_c, the fluid phase is load-bearing, below phi_c the solid phase (mineral grains) is load-bearing. The equations here describe the variation of K and G for porosities below phi_c as straight lines. Critical porosity is usually 0.4 for sandstone, 0.7 for chalk, 0.02-0.03 for granites.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **phi_c** (*float, optional*) – Critical porosity. Default: 0.4

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Mavko et al. (2009), The Rock Physics Handbook, Cambridge University Press (p.370)

bruges.rockphysics.rockphysicsmodels.**hertz_mindlin**(*K0*, *G0*, *sigma*, *phi_c=0.4*, *Cn=8.6*, *f=1*)

Hertz-Mindlin model. This model describes the elastic behaviour (K=bulk and G=shear moduli) of a dry pack of spheres subject to a hydrostatic confining pressure.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **sigma** (*float*) – Effective stress in MPa.
- **phi_c** (*float, optional*) – Critical porosity. Default: 0.4
- **Cn** (*float, optional*) – Coordination number Default: 8.6.
- **f** (*float, optional*) – Shear modulus correction factor, f=1 for dry pack with perfect adhesion between particles and f=0 for dry frictionless pack.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Mavko et al. (2009), The Rock Physics Handbook, Cambridge University Press (p.246)

bruges.rockphysics.rockphysicsmodels.**increasing_cement**(*K0*, *G0*, *phi*, *phi_cem=0.38*, *phi_c=0.4*, *Cn=8.6*, *Kc=37*, *Gc=45*, *scheme=2*)

Increasing cement model (Modified Hashin-Shtrikman upper bound). This model describes the elastic behaviour (K=bulk and G=shear moduli) of a dry sand with a certain initial cementation by interpolating with the upper Hashin-Shtrikman bound the two end members at zero porosity and critical porosity. The zero porosity end member has K and G equal to mineral. The high porosity end member has K and G given by the Contact Cement model. Probably best to avoid using if for porosities>phi_cem. Need to check references.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.

- **phi** (*float or array_like*) – Porosity.
- **phi_cem** (*float, optional*) – Porosity at initial cementation. Default: 0.38.
- **phi_c** (*float, optional*) – Critical porosity. Default: 0.4.
- **Cn** (*float, optional*) – Coordination number. Default: 8.6.
- **Gc** (*Kc,*) – Cement bulk & shear modulus in GPa. Default: 37, 45.
- **scheme** (*int, optional*) – Cementation scheme, can be either 1 or 2: 1: cement deposited at grain contacts 2: cement as uniform layer around grains. Default: 2.

Returns **Kdry, Gdry** – dry rock bulk & shear modulus in GPa.

Return type float or array_like

bruges.rockphysics.rockphysicsmodels.**soft_sand**(*K0*, *G0*, *phi*, *sigma*, *phi_c=0.4*, *Cn=8.6*, *f=1*)

Soft sand, or friable sand or uncemented sand model. This model describes the elastic behaviour (K=bulk and G=shear moduli) of poorly sorted dry sand by interpolating with the lower Hashin-Shtrikman bound the two end members at zero porosity and critical porosity. The zero porosity end member has K and G equal to mineral. The end member at critical porosity has K and G given by Hertz-Mindlin model.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **sigma** (*float*) – Effective stress in MPa.
- **phi_c** (*float, optional*) – Critical porosity. Default: 0.4
- **Cn** (*float, optional*) – Coordination number Default: 8.6.
- **f** (*float, optional*) – Shear modulus correction factor, f=1 for dry pack with perfect adhesion between particles and f=0 for dry frictionless pack.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Mavko et al. (2009), The Rock Physics Handbook, Cambridge University Press (p.258)

bruges.rockphysics.rockphysicsmodels.**stiff_sand**(*K0*, *G0*, *phi*, *sigma*, *phi_c=0.4*, *Cn=8.6*, *f=1*)

Stiff sand model. This model describes the elastic behaviour (K=bulk and G=shear moduli) of stiff dry sands by interpolating with the upper Hashin-Shtrikman bound the two end members at zero porosity and critical porosity. The zero porosity end member has K and G equal to mineral. The end member at critical porosity has K and G given by Hertz-Mindlin model.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **sigma** (*float*) – Effective stress in MPa.
- **phi_c** (*float, optional*) – Critical porosity. Default: 0.4
- **Cn** (*float, optional*) – Coordination number Default: 8.6.

- **f** (*float, optional*) – Shear modulus correction factor, f=1 for dry pack with perfect adhesion between particles and f=0 for dry frictionless pack.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa

Return type float or array_like

References

Mavko et al. (2009), The Rock Physics Handbook, Cambridge University Press (p.260)

bruges.rockphysics.rockphysicsmodels.**vernik_consol_sand**(*K0*, *G0*, *phi*, *sigma*, *b=10*)

Vernik & Kachanov Consolidated Sand Model. This model describes the elastic behaviour (K=bulk and G=shear moduli) of consolidated dry sand subject to a hydrostatic confining pressure as a continuous solid containing pores and cracks.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **sigma** (*float*) – Effective stress in MPa.
- **b** (*float, optional*) – Slope parameter in pore shape empirical equation, range: 8-12. Default: 10.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Vernik & Kachanov (2010), Modeling elastic properties of siliciclastic rocks, Geophysics v.75 n.6

bruges.rockphysics.rockphysicsmodels.**vernik_sand_diagenesis**(*K0*, *G0*, *phi*, *sigma*, *phi_c=0.36*, *phi_con=0.26*, *b=10*, *n=2.0*, *m=2.05*)

Vernik & Kachanov Sandstone Diagenesis Model. This model describes the elastic behaviour (K=bulk and G=shear moduli) of dry sand modeled as a continuous solid containing pores and cracks for porosities below phi_con (consolidation porosity) using Vernik's Consolidated Sand Model, and as a granular material for porosities above phi_con using Vernik's Soft Sand Model 1.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **sigma** (*float*) – Effective stress in MPa.
- **phi_c** (*float, optional*) – Critical porosity, range 0.30-0.42. Default: 0.36.
- **phi_con** (*float, optional*) – Consolidation porosity, range 0.22-0.30. Default: 0.26.
- **b** (*float, optional*) – Slope parameter in pore shape empirical equation, range: 8-12. Default: 10.
- **m** (*n,*) – Empirical factors. Default: 2.00, 2.05.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Vernik & Kachanov (2010), Modeling elastic properties of siliciclastic rocks, Geophysics v.75 n.6

bruges.rockphysics.rockphysicsmodels.**vernik_shale**(*vclay*, *phi*, *rhom=2.73*, *rhob=1*, *Mqz=96*, *c33_clay=33.4*, *A=0.00284*)

Vernik & Kachanov Shale Model. This model describes the elastic behaviour in terms of velocities and density of inorganic shales.

Parameters

- **vclay** (*float or array_like*) – Dry clay content volume fraction.
- **phi** (*float or array_like*) – Porosity, maximum 0.40.
- **rhom** (*float, optional*) – Shale matrix density in g/cc. Default: 2.73.
- **rhob** (*float, optional*) – Brine density in g/cc. Default: 1.
- **Mqz** (*float, optional*) – P-wave elastic modulus of remaining minerals in GPa Default: 96.
- **c33_clay** (*float, optional*) – Anisotropic clay constant in GPa. Default: 33.4.
- **A** (*float, optional*) – Empirical coefficient for Vs. Default is good for illite/smectite/chlorite, can be raised up to .006 for kaolinite-rich clays. Default: 0.00284.

Returns **vp, vs, density** – P- and S-wave velocities in m/s, density in g/cc.

Return type float or array_like

Notes

Shale matrix density (rhom) averages 2.73 +/- 0.03 g/cc at porosities below 0.25. It gradually varies with compaction and smectite-to-illite transition. A more accurate estimate can be calculated with this equation: rhom = 2.76+0.001*((rho-2)-230*np.exp(-4*(rho-2)))

References

Vernik & Kachanov (2010), Modeling elastic properties of siliciclastic rocks, Geophysics v.75 n.6

bruges.rockphysics.rockphysicsmodels.**vernik_soft_sand_1**(*K0*, *G0*, *phi*, *sigma*, *phi_c=0.36*, *phi_con=0.26*, *b=10*, *n=2.0*, *m=2.05*)

Vernik & Kachanov Soft Sand Model 1. This model describes the elastic behaviour (K=bulk and G=shear moduli) of dry sand modeled as a granular material. Only applicable for porosities between the low-porosity end-member (at the consolidation porosity phi_con) and the high-porosity end-member (at the critical porosity phi_c). The low-porosity end member is calculated with Vernik's Consolidated Sand Model.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.

- **sigma** (*float*) – Effective stress in MPa.
- **phi_c** (*float, optional*) – Critical porosity, range 0.30-0.42. Default: 0.36.
- **phi_con** (*float, optional*) – Consolidation porosity, range 0.22-0.30. Default: 0.26.
- **b** (*float, optional*) – Slope parameter in pore shape empirical equation, range: 8-12. Default: 10.
- **m** (*n,*) – Empirical factors. Default: 2.00, 2.05.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Vernik & Kachanov (2010), Modeling elastic properties of siliciclastic rocks, Geophysics v.75 n.6

bruges.rockphysics.rockphysicsmodels.**vernik_soft_sand_2**(*K0*, *G0*, *phi*, *p=20*, *q=20*)

Vernik & Kachanov Soft Sand Model 2. This model describes the elastic behaviour (K=bulk and G=shear moduli) of dry sand modeled as a granular material. Applicable in the entire porosity range.

Parameters

- **G0** (*K0,*) – Mineral bulk & shear modulus in GPa.
- **phi** (*float or array_like*) – Porosity.
- **q** (*p,*) – Pore shape factor for K and G, range: 10-45. Default: 20.

Returns **Kdry, Gdry** – Dry rock bulk & shear modulus in GPa.

Return type float or array_like

References

Vernik & Kachanov (2010), Modeling elastic properties of siliciclastic rocks, Geophysics v.75 n.6

bruges.rockphysics.rpm module

rpm.py

Rock physics models @author: Alessandro Amato del Monte

bruges.rockphysics.rpm.**contactcement**(*K0*, *G0*, *phi*, *phic=0.4*, *Cn=8.6*, *Kc=37*, *Gc=45*, *scheme=2*)

Contact cement (cemented sand) model, Dvorkin-Nur (1996) written by Alessandro Amato del Monte (2015) from Mavko et al., Rock Physics Handbook, p.255 INPUT K0, G0: mineral bulk & shear modulus in GPa phi: porosity phic: critical porosity (default 0.4) Cn: coordination nnumber (default 8.6) Kc, Gc: cement bulk & shear modulus in GPa (default 37, 45 i.e. quartz) scheme: 1=cement deposited at grain contacts, 2=in uniform layer around grains (default 2) OUTPUT K_DRY, G_DRY: dry rock bulk & shear modulus in GPa

bruges.rockphysics.rpm.**critpor**(*K0*, *G0*, *phi*, *phic=0.4*)

Critical porosity, Nur et al. (1991, 1995) written by Alessandro Amato del Monte (2015) from Mavko et al., Rock Physics Handbook, p.353 INPUT K0, G0: mineral bulk & shear modulus in GPa phi: porosity phic: critical porosity (default 0.4) OUTPUT K_DRY, G_DRY: dry rock bulk & shear modulus in GPa

bruges.rockphysics.rpm.**hertzmindlin**(*K0*, *G0*, *phi*, *phic=0.4*, *Cn=8.6*, *P=10*, *f=1*)
Hertz-Mindlin model written by Alessandro Amato del Monte (2015) from Mavko et al., Rock Physics Handbook, p.246 INPUT K0, G0: mineral bulk & shear modulus in GPa phi: porosity phic: critical porosity (default 0.4) Cn: coordination nnumber (default 8.6) P: confining pressure in MPa (default 10) f: shear modulus correction factor, f=1 for dry pack with perfect adhesion between particles and f=0 for dry frictionless pack OUTPUT K_DRY, G_DRY: dry rock bulk & shear modulus in GPa

bruges.rockphysics.rpm.**softsand**(*K0*, *G0*, *phi*, *phic=0.4*, *Cn=8.6*, *P=10*, *f=1*)
Soft-sand (uncemented) model written by Alessandro Amato del Monte (2015) from Mavko et al., Rock Physics Handbook, p.258 INPUT K0, G0: mineral bulk & shear modulus in GPa phi: porosity phic: critical porosity (default 0.4) Cn: coordination nnumber (default 8.6) P: confining pressure in MPa (default 10) f: shear modulus correction factor, f=1 for dry pack with perfect adhesion between particles and f=0 for dry frictionless pack OUTPUT K_DRY, G_DRY: dry rock bulk & shear modulus in GPa

bruges.rockphysics.rpm.**stiffsand**(*K0*, *G0*, *phi*, *phic=0.4*, *Cn=8.6*, *P=10*, *f=1*)
Stiff-sand model written by Alessandro Amato del Monte (2015) from Mavko et al., Rock Physics Handbook, p.260 INPUT K0, G0: mineral bulk & shear modulus in GPa phi: porosity phic: critical porosity (default 0.4) Cn: coordination nnumber (default 8.6) P: confining pressure in MPa (default 10) f: shear modulus correction factor, f=1 for dry pack with perfect adhesion between particles and f=0 for dry frictionless pack OUTPUT K_DRY, G_DRY: dry rock bulk & shear modulus in GPa

bruges.rockphysics.rpt module

rpt.py

Rock physics templates Uses rock physics models defined in rpm.py For Voigt-Reuss-Hill averages I define my own function here but I should really change it and use the bounds already defined in bruges. @author: Alessandro Amato del Monte

bruges.rockphysics.rpt.**rpt**(*model='soft'*, *vsh=0.0*, *fluid='gas'*, *phic=0.4*, *Cn=8*, *P=10*, *f=1*, *cement='quartz'*)

bruges.rockphysics.rpt.**vrh**(*volumes*, *k*, *mu*)
Calculates Voigt-Reuss-Hill bounds. INPUT volumes: array with volumetric fractions k: array with bulk modulus mu: array with shear modulus OUTPUT k_u, k_l: upper (Voigt) and lower (Reuss) average for k mu_u, mu_l: upper (Voigt) and lower (Reuss) average for mu k0, mu0: Hill average of k and mu

Module contents

bruges.transform package

Submodules

bruges.transform.coordinates module

Coordinate transformation. This module contains a class for converting between seismic survey inline-xline coordinates and real-world UTM coordinates.

copyright 2018 Agile Geoscience

license Apache 2.0

class bruges.transform.coordinates.**CoordTransform**(*ix*, *xy*)
Bases: object

A class for converting between seismic survey inline-xline coordinates and real-world UTM coordinates.

Instantiate with a pair of at least 3 coordinates mapping one space to the other. Provide two array-likes of shape (3, 2). See below for example.

After instantiation, the class is callable in the 'forward' (inline-xline to UTMx-UTMy) direction. You can also use `CoordTransform.forwrd()`. The `CoordTransform.reverse()` method converts in the other direction (UTMx-UTMy to inline-xline).

Example >>> corner_ix = [[0, 0], [0, 950], [650, 950]] >>> corner_xy = [[605835.5, 6073556.5],

[629576.3, 6074220.0], [629122.5, 6090463.2]]

```
>>> transform = bruges.transform.CoordTransform(corner_ix, corner_xy)
>>> transform([300, 400])
array([ 615622.18016194, 6081332.72995951])
>>> transform.forward([300, 400])
array([ 615622.18016194, 6081332.72995951])
>>> transform.reverse([ 615622.18016194, 6081332.72995951])
array([300, 400])
```

forward(*p*)

Convert inline-xline to UTMx-UTM-y.

Example >>> transform.forward([300, 400]) array([615622.18016194, 6081332.72995951])

reverse(*q*)

Convert UTMx-UTM-y to inline-xline.

Example >>> transform.reverse([615622.18016194, 6081332.72995951]) array([300, 400])

bruges.transform.cumavg module

Average velocity equations.

All from Chris Liner, Elements of 3D Seismology, PennWell Press, 2004.

bruges.transform.cumavg.**v_avg**(*v*, *depth=None*, *time=None*)

Cumulative average of a velocity log. You must provide either a depth or a time basis for the log.

Parameters

- **v** (*ndarray*) – The velocity log.
- **depth** (*ndarray*) – The depth values corresponding to the log.
- **time** (*ndarray*) – The time values corresponding to the log.

Returns The V_avg log.

Return type ndarray

bruges.transform.cumavg.**v_bac**(*v*, *rho*, *depth*)

Cumulative Backus average of a velocity log. You must provide either a depth or a time basis for the log.

For a non-cumulative version that can also provide sclaing for the V_s log, as well as quality factor, see bruges.anisotropy.backus.

Parameters

- **v** (*ndarray*) – The velocity log.
- **rho** (*ndarray*) – The density log.

- **depth** (*ndarray*) – The depth values corresponding to the logs.

Returns The V_bac log.

Return type ndarray

bruges.transform.cumavg.**v_rms**(*v*, *depth=None*, *time=None*)
Cumulative RMS mean of a velocity log. You must provide either a depth or a time basis for the log.

Parameters

- **v** (*ndarray*) – The velocity log.
- **depth** (*ndarray*) – The depth values corresponding to the log.
- **time** (*ndarray*) – The time values corresponding to the log.

Returns The V_rms log.

Return type ndarray

bruges.transform.nmo module

Normal moveout.

© 2017 Leo Uieda, licensed BSD 3-clause license. https://github.com/seg/tutorials-2017/tree/master/1702_Step_by_step_NMO

bruges.transform.nmo.**nmo_correction**(*cmp*, *dt*, *offsets*, *velocities*)
Performs NMO correction on the given CMP.

The units must be consistent. E.g., if dt is seconds and offsets is meters, velocities must be m/s.

Parameters

- **cmp** (*ndarray*) – The 2D array CMP gather that we want to correct.
- **dt** (*float*) – The sampling interval.
- **offsets** (*ndarray*) – A 1D array with the offset of each trace in the CMP.
- **velocities** (*ndarray*) – A 1D array with the NMO velocity for each time. Should have the same number of elements as the CMP has samples.

Returns The NMO corrected gather.

Return type ndrray

bruges.transform.nmo.**reflection_time**(*t0*, *x*, *vnmo*)
Calculate the travel-time of a reflected wave. Doesn't consider refractions or changes in velocity.

The units must be consistent. E.g., if t0 is seconds and x is metres, vnmo must be m/s.

Parameters

- **t0** (*float*) – The 0-offset (normal incidence) travel-time.
- **x** (*float*) – The offset of the receiver.
- **vnmo** (*float*) – The NMO velocity.

Returns The reflection travel-time.

Return type t (float)

bruges.transform.nmo.**sample_trace**(*trace*, *time*, *dt*)
Sample an amplitude at a given time using interpolation.

Parameters

- **trace** (*1D array*) – Array containing the amplitudes of a single trace.
- **time** (*float*) – The time at which I want to sample the amplitude.
- **dt** (*float*) – The sampling interval.

Returns The interpolated amplitude. Will be None if *time* is beyond the end of the trace or if there are fewer than two points between *time* and the end.

Return type amplitude (float or None)

bruges.transform.timedepthconv module

Time-depth conversion.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.transform.timedepthconv.**depth_to_time**(*data*, *vmodel*, *dz*, *dt*, *twt=True*, *mode='nearest'*, *return_t=False*)

Converts data from the depth domain to the time domain given a velocity model.

Parameters

- **data** (*ndarray*) – The data to convert, will work with a 1 or 2D numpy numpy array. array(samples,traces).
- **vmodel** (*ndarray*) – P-wave interval velocity model that corresponds to the data. Must be the same shape as data.
- **dz** (*float*) – The sample interval of the input data [m].
- **dt** (*float*) – The sample interval of the output data [s].
- **twt** (*bool*) – Use twt travel time, defaults to true.
- **mode** (*str*) – What kind of interpolation to use, defaults to 'nearest'.
- **return_t** (*bool*) – Whether to also return the new time basis.

Returns The data resampled in the time domain.

bruges.transform.timedepthconv.**time_to_depth**(*data*, *vmodel*, *dt*, *dz*, *twt=True*, *mode='nearest'*, *return_z=False*)

Converts data from the time domain to the depth domain given a velocity model.

Parameters

- **data** (*ndarray*) – The data to convert, will work with a 1 or 2D numpy numpy array. array(samples,traces).
- **vmodel** (*ndarray*) – P-wave interval velocity model that corresponds to the data. Must be the same shape as data.
- **dt** (*float*) – The sample interval of the input data [s], or an array of times.
- **dz** (*float*) – The sample interval of the output data [m], or an array of depths.
- **twt** (*bool*) – Use twt travel time, defaults to true.
- **mode** (*str*) – What kind of interpolation to use, defaults to 'nearest'.
- **return_z** (*bool*) – Whether to also return the new time basis.

Returns ndarray: The data resampled in the depth domain.

Module contents

bruges.unit package

Submodules

bruges.unit.unit module

Unit conversion.

copyright 2015 Agile Geoscience

license Apache 2.0

bruges.unit.unit.**ft_m**(*x*)

bruges.unit.unit.**m_ft**(*x*)

Module contents

bruges.util package

Submodules

bruges.util.transformations module

Homogeneous Transformation Matrices and Quaternions.

A library for calculating 4x4 matrices for translating, rotating, reflecting, scaling, shearing, projecting, orthogonalizing, and superimposing arrays of 3D homogeneous coordinates as well as for converting between rotation matrices, Euler angles, and quaternions. Also includes an Arcball control object and functions to decompose transformation matrices.

Author Christoph Gohlke

Organization Laboratory for Fluorescence Dynamics, University of California, Irvine

Version 2017.02.17

Requirements

- CPython 2.7 or 3.5
- Numpy 1.11
- Transformations.c 2017.02.17 (recommended for speedup of some functions)

Notes

The API is not stable yet and is expected to change between revisions.

This Python code is not optimized for speed. Refer to the transformations.c module for a faster implementation of some functions.

Documentation in HTML format can be generated with epydoc.

Matrices (M) can be inverted using numpy.linalg.inv(M), be concatenated using numpy.dot(M0, M1), or transform homogeneous coordinate arrays (v) using numpy.dot(M, v) for shape (4, *) column vectors, respectively numpy.dot(v, M.T) for shape (*, 4) row vectors ("array of points").

This module follows the "column vectors on the right" and "row major storage" (C contiguous) conventions. The translation components are in the right column of the transformation matrix, i.e. M[:3, 3]. The transpose of the transformation matrices may have to be used to interface with other graphics systems, e.g. with OpenGL's glMultMatrixd(). See also [16].

Calculations are carried out with numpy.float64 precision.

Vector, point, quaternion, and matrix function arguments are expected to be "array like", i.e. tuple, list, or numpy arrays.

Return types are numpy arrays unless specified otherwise.

Angles are in radians unless specified otherwise.

Quaternions w+ix+jy+kz are represented as [w, x, y, z].

A triple of Euler angles can be applied/interpreted in 24 ways, which can be specified using a 4 character string or encoded 4-tuple:

Axes 4-string: e.g. 'sxyz' or 'ryxy'

- first character : rotations are applied to 's'tatic or 'r'otating frame
- remaining characters : successive rotation axis 'x', 'y', or 'z'

Axes 4-tuple: e.g. (0, 0, 0, 0) or (1, 1, 1, 1)

- inner axis: code of axis ('x':0, 'y':1, 'z':2) of rightmost matrix.
- parity : even (0) if inner axis 'x' is followed by 'y', 'y' is followed by 'z', or 'z' is followed by 'x'. Otherwise odd (1).
- repetition : first and last axis are same (1) or different (0).
- frame : rotations are applied to static (0) or rotating (1) frame.

Other Python packages and modules for 3D transformations and quaternions:

- **Transforms3d** includes most code of this module.
- Blender.mathutils
- numpy-dtypes

References

(1) Matrices and transformations. Ronald Goldman. In "Graphics Gems I", pp 472-475. Morgan Kaufmann, 1990.

(2) More matrices and transformations: shear and pseudo-perspective. Ronald Goldman. In "Graphics Gems II", pp 320-323. Morgan Kaufmann, 1991.

(3) Decomposing a matrix into simple transformations. Spencer Thomas. In "Graphics Gems II", pp 320-323. Morgan Kaufmann, 1991.

(4) Recovering the data from the transformation matrix. Ronald Goldman. In "Graphics Gems II", pp 324-331. Morgan Kaufmann, 1991.

(5) Euler angle conversion. Ken Shoemake. In "Graphics Gems IV", pp 222-229. Morgan Kaufmann, 1994.

(6) Arcball rotation control. Ken Shoemake. In "Graphics Gems IV", pp 175-192. Morgan Kaufmann, 1994.

(7) Representing attitude: Euler angles, unit quaternions, and rotation vectors. James Diebel. 2006.

(8) A discussion of the solution for the best rotation to relate two sets of vectors. W Kabsch. Acta Cryst. 1978. A34, 827-828.

(9) Closed-form solution of absolute orientation using unit quaternions. BKP Horn. J Opt Soc Am A. 1987. 4(4):629-642.

(10) Quaternions. Ken Shoemake. http://www.sfu.ca/~jwa3/cmpt461/files/quatut.pdf

(11) From quaternion to matrix and back. JMP van Waveren. 2005. http://www.intel.com/cd/ids/developer/asmo-na/eng/293748.htm

(12) Uniform random rotations. Ken Shoemake. In "Graphics Gems III", pp 124-132. Morgan Kaufmann, 1992.

(13) Quaternion in molecular modeling. CFF Karney. J Mol Graph Mod, 25(5):595-604

(14) New method for extracting the quaternion from a rotation matrix. Itzhack Y Bar-Itzhack, J Guid Contr Dynam. 2000. 23(6): 1085-1087.

(15) Multiple View Geometry in Computer Vision. Hartley and Zissermann. Cambridge University Press; 2nd Ed. 2004. Chapter 4, Algorithm 4.7, p 130.

(16) Column Vectors vs. Row Vectors. http://steve.hollasch.net/cgindex/math/matrix/column-vec.html

Examples

```
>>> alpha, beta, gamma = 0.123, -1.234, 2.345
>>> origin, xaxis, yaxis, zaxis = [0, 0, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1]
>>> I = identity_matrix()
>>> Rx = rotation_matrix(alpha, xaxis)
>>> Ry = rotation_matrix(beta, yaxis)
>>> Rz = rotation_matrix(gamma, zaxis)
>>> R = concatenate_matrices(Rx, Ry, Rz)
>>> euler = euler_from_matrix(R, 'rxyz')
>>> numpy.allclose([alpha, beta, gamma], euler)
True
>>> Re = euler_matrix(alpha, beta, gamma, 'rxyz')
>>> is_same_transform(R, Re)
True
>>> al, be, ga = euler_from_matrix(Re, 'rxyz')
>>> is_same_transform(Re, euler_matrix(al, be, ga, 'rxyz'))
True
>>> qx = quaternion_about_axis(alpha, xaxis)
>>> qy = quaternion_about_axis(beta, yaxis)
>>> qz = quaternion_about_axis(gamma, zaxis)
>>> q = quaternion_multiply(qx, qy)
>>> q = quaternion_multiply(q, qz)
>>> Rq = quaternion_matrix(q)
>>> is_same_transform(R, Rq)
True
```

(continues on next page)

(continued from previous page)

```
>>> S = scale_matrix(1.23, origin)
>>> T = translation_matrix([1, 2, 3])
>>> Z = shear_matrix(beta, xaxis, origin, zaxis)
>>> R = random_rotation_matrix(numpy.random.rand(3))
>>> M = concatenate_matrices(T, R, Z, S)
>>> scale, shear, angles, trans, persp = decompose_matrix(M)
>>> numpy.allclose(scale, 1.23)
True
>>> numpy.allclose(trans, [1, 2, 3])
True
>>> numpy.allclose(shear, [0, math.tan(beta), 0])
True
>>> is_same_transform(R, euler_matrix(axes='sxyz', *angles))
True
>>> M1 = compose_matrix(scale, shear, angles, trans, persp)
>>> is_same_transform(M, M1)
True
>>> v0, v1 = random_vector(3), random_vector(3)
>>> M = rotation_matrix(angle_between_vectors(v0, v1), vector_product(v0, v1))
>>> v2 = numpy.dot(v0, M[:3,:3].T)
>>> numpy.allclose(unit_vector(v1), unit_vector(v2))
True
```

bruges.util.util module

Utility functions.

copyright 2021 Agile Scientific

license Apache 2.0

bruges.util.util.**apply_along_axis**(*func_1d*, *arr*, *kernel*, ***kwargs*)

Apply 1D function across 2D slice as efficiently as possible.

Although *np.apply_along_axis* seems to do well enough, map usually seems to end up beig a bit faster.

Parameters **func_1d** (*function*) – the 1D function to apply, e.g. np.convolve. Should take 2 or more arguments: the

Example >>> apply_along_axes(np.convolve, reflectivity_2d, wavelet, mode='same')

bruges.util.util.**deprecated**(*instructions*)

Flags a method as deprecated. This decorator can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used. :param instructions: A human-friendly string of instructions, such

as: 'Please migrate to add_proxy() ASAP.'

Returns The decorated function.

bruges.util.util.**error_flag**(*pred*, *actual*, *dev=1.0*, *method=1*)

Calculate the difference between a predicted and an actual curve and return a log flagging large differences based on a user-defined distance (in standard deviation units) from the mean difference.

Parameters

- **predicted** (*ndarray*) – predicted log.

- **actual** (*ndarray*) – original log.
- **dev** (*float*) – standard deviations to use, default 1
- **calcluation method** (*error*) – default 1 1: difference between curves larger than mean difference plus dev 2: curve slopes have opposite sign. Will require depth log for

  .diff method

  3: curve slopes of opposite sign OR difference larger than mean plus dev

Returns flag (ndarray) = error flag curve

Author: Matteo Niccoli, 2018

bruges.util.util.**extrapolate**(*a*)

Extrapolate up and down an array from the first and last non-NaN samples.

E.g. Continue the first and last non-NaN values of a log up and down.

Parameters **a** (*ndarray*) – The array to treat.

Returns The treated array.

Return type ndarray

bruges.util.util.**moving_average**(*a*, *length*, *mode='same'*)

Computes the mean in a moving window using convolution. For an alternative, as well as other kinds of average (median, mode, etc.), see bruges.filters.

Example

```
>>> test = np.array([1,1,9,9,9,9,9,2,3,9,2,2,np.nan,1,1,1,1])
>>> moving_average(test, 5, mode='same')
array([ 2.2,  4. ,  5.8,  7.4,  9. ,  7.6,  6.4,  6.4,  5. ,  3.6,  nan,
        nan,  nan,  nan,  nan,  0.8,  0.6])
```

bruges.util.util.**moving_avg_conv**(*a*, *length*, *mode='same'*)

Moving average via convolution. Keeping it for now for compatibility.

bruges.util.util.**moving_avg_fft**(*a*, *length*, *mode='same'*)

Moving average via FFT convolution. Keeping it for now for compatibility.

bruges.util.util.**nearest**(*a*, *num*)

Finds the array's nearest value to a given num.

Parameters

- **a** (*ndarray*) – An array.
- **num** (*float*) – The value to find the nearest to.

Returns float. The normalized array.

bruges.util.util.**next_pow2**(*num*)

Calculates the next nearest power of 2 to the input. Uses 2**ceil(log2(num)).

Parameters **num** (*number*) – The number to round to the next power if two.

Returns number. The next power of 2 closest to num.

bruges.util.util.**normalize**(*a*, *new_min=0.0*, *new_max=1.0*)
Normalize an array to [0,1] or to arbitrary new min and max.

Parameters

- **a** (*ndarray*) – An array.
- **new_min** (*float*) – The new min to scale to, default 0.
- **new_max** (*float*) – The new max to scale to, default 1.

Returns ndarray. The normalized array.

bruges.util.util.**power**(*start*, *stop*, *num*)
Nonlinear space following a power function.

bruges.util.util.**rms**(*a*, *axis=None*)
Calculates the RMS of an array.

Parameters

- **a** (*ndarray*) –
- **axis** (*int*) – the RMS for the whole array is computed.

Returns The RMS of the array along the desired axis or axes.

Return type ndarray

bruges.util.util.**root**(*start*, *stop*, *num*)
Nonlinear space following a sqrt function.

bruges.util.util.**sigmoid**(*start*, *stop*, *num*)
Nonlinear space following a logistic function.

The function is asymptotic; the parameters used in the sigmoid gets within 0.5% of the target thickness in a wedge increasing from 0 to 2x the original thickness.

bruges.util.util.**top_and_tail**(**arrays*)
Top and tail all arrays to the non-NaN extent of the first array.

E.g. crop the NaNs from the top and tail of a well log.

Parameters **arrays** (*list*) – A list of arrays to treat.

Returns A list of treated arrays.

Return type list

Module contents

Submodules

bruges.bruges module

Generic error class.

copyright 2015 Agile Geoscience

license Apache 2.0

exception bruges.bruges.**BrugesError**
Bases: Exception

Generic error class.

bruges.get_bruges module

Make random text for Bruges.

bruges.get_bruges.**get_bruges**(*p*, *n*)

Module contents

Initialize the library.

copyright 2015 Agile Geoscience

license Apache 2.0

Links

- Documentation
- Project page
- Issue Tracker
- PyPi
- Agile Geoscience

Indices and tables

- genindex
- modindex
- search

1.3 Contributors

- Evan Bianco
- Ben Bougher
- Matt Hall
- Alessandro Amato del Monte
- Wes Hamlyn
- Sean Ross-Ross

1.4 Links

- Documentation
- Project page
- Issue Tracker
- PyPi
- Agile Geoscience

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y

Z