Source code for particleman.core

```
"""
Core Stockwell transform and inverse transform functions.

"""
import numpy as np
from .st import st, ist

def _get_lo_hi(L, hp, lp, Fs):
    """Get context-appropriate representation of hp, lp.

    L : int
        Length of time series.


    """
    if Fs:
        # If the sample rate has been specified then
        # we low-pass at the nyquist frequency.
        if not lp:
            lp = Fs/2.0

        # Providing the sample rate also means that the
        # filter parameters are in Hz, so we convert
        # them to the appropriate number of samples
        low = int(np.floor(hp/(Fs/L)))
        high = int(np.ceil(lp/(Fs/L)))
    else:
        # Since we don't have a sampling rate then
        # everything will be expressed in samples
        if not lp:
            lp = L/2.0
        low = int(hp)
        high = int(lp)

    return low, high, lp


def get_TF_arrays(N, Fs=0, hp=0, lp=0):                                    [docs]
    """ Make the Stockwell time, frequency arrays for plotting.

    Parameters
    ----------
    N : int
        Number of samples in the time series.
    hp : float
        high-pass point in samples (if Fs is not specified) or in Hz (if Fs is specified)
    lp : float
        low-pass point in samples (if Fs is not specified) or in Hz (if Fs is specified)
    Fs : float
        sampling rate in Hz

    Returns
    -------
    T, F : numpy.ndarray (complex, rank 2)

    """
    # XXX: doesn't work yet.  still needs "S", the transform tile
    if Fs:
        t = 1.0/Fs # Length of one sample
        t = np.arange(N) * t # List of time values
        T, F = np.meshgrid(t, np.arange(hp, lp, (lp-hp) / (1.0 * S.shape[0])))
    else:
        t = np.arange(N)
        T, F = np.meshgrid(t, np.arange(int(hp), int(lp), int(lp-hp)/(1.0*S.shape[0])))

    return T, F
```

```
def stransform(x, Fs=0, hp=0, lp=0, return_time_freq=False):                    [docs]
    """Perform a Stockwell transform on a time-series.

    Returns the transform (S), and time (T) and frequency (F)
    matrices suitable for use with the contour/contourf functions.

    Parameters
    ----------
    x : numpy.ndarray
        array containing time-series data
    hp : float
        high-pass point in samples (if Fs is not specified) or in Hz (if Fs is specified)
    lp : float
        low-pass point in samples (if Fs is not specified) or in Hz (if Fs is specified)
    Fs : float
        sampling rate in Hz
    return_time_freq : bool
        If True, also return the correct-sized time and frequency domain tiles.

    Returns
    -------
    S : numpy.ndarray (numpy.complex128, rank 2)
        Stockwell transform (S) matrix
    T, F :  numpy.ndarray (float64, rank 2), optional
        Time (T) and frequency (F) matrices.

    Examples
    --------
    Transform a 100 Hz time series

    >>> S, T, F = stransform(data, Fs=100, return_time_freq=True)
    >>> plt.contourf(T, F, abs(S))

    References
    ----------
    * http://vcs.ynic.york.ac.uk/docs/naf/intro/concepts/timefreq.html
    * http://kurage.nimh.nih.gov/meglab/Meg/Stockwell

    """
    low, high, lp = _get_lo_hi(len(x), hp, lp, Fs)

    # The stockwell transform
    S = st(x, low, high)

    # Compute our time and frequency matrix with
    # the correct scaling for use with the
    # contour and contourf functions
    if return_time_freq:
        L = len(x)
        if Fs:
            t = 1.0/Fs # Length of one sample
            t = np.arange(L)*t # List of time values
            T, F = np.meshgrid(t, np.arange(hp, lp, (lp-hp)/(1.0*S.shape[0])))
        else:
            t = np.arange(L)
            T, F = np.meshgrid(t, np.arange(int(hp), int(lp), int(lp-hp)/(1.0*S.shape[0])))
        out = (S, T, F)
    else:
        out = S

    return out


def istransform(X, Fs=0, hp=0, lp=0):                                          [docs]
    """Perform inverse Stockwell transform
    """
    #XXX: untested
    low, high, lp = _get_lo_hi(X.shape[1], hp, lp, Fs)

    x = ist(X, low, high)

    return x
```