Source code for particleman.plotting

```
"""
Plotting functions for Stockwell transforms and normalized inner-product (NIP)
filtering.

These plotting routines, particularly `tile_comparison` and `NIP_filter_plot`,
are useful to see how much surface wave energy is in a packet, or how much
off-great-circle propagation there is.

These functions plot a number of configurations of "tiles." Each tile
consists of the Stockwell transform on top and an aligned time-series waveform
below.  The transform may have a hatching overlay that would normally
correspond to a NIP filter, and the time-series axis may have a reference
(gray) trace overlayed, which normally corresponds to the unfiltered trace.


## Channel nomenclature:

```
[nevrt][sd][f]
```

* n : north component
* e : east
* v : vertical
* s : scalar rotation (great circle)
* d : dynamic rotation (NIP estimate)
* f : NIP filtered

"""
import matplotlib.pyplot as plt
from matplotlib import gridspec
from mpl_toolkits.axes_grid1 import make_axes_locatable
from matplotlib.ticker import FormatStrFormatter
import numpy as np

TODO: make a **tile_kwargs argument in all calling signatures using plot_tile,
which would include arrivals, flim, clim, dlim, tlim, hatch, hatchlim
TODO: make a plot_image function, and use it as an argument in the plot_tile
function.

def _strip_zero_freq(T, F, S):
    """
    Removes the zero-frequency rows from T, F, and S, to facilitate log plotting.
    """
    if np.allclose(F[0], np.zeros_like(F[0])):
        F = F[1:]
        T = T[1:]
        S = S[1:]

    return T, F, S

def plot_tile(fig, ax1, T, F, S, ax2, d1, label1, color1='k', d2=None,   [docs]
              label2=None, arrivals=None, flim=None, clim=None, hatch=None,
              hatchlim=None, dlim=None, amp_fmt='%.2e', cmap=None, alpha=None):
    """
    Plot time-frequency pcolormesh tiles above time-aligned aligned time-series.

    Parameters
    ----
    fig : matplotlib.Figure
    ax1 : matplotlib.axis.Axis
        Axis for time-frequency pcolormesh tile and optional hatching.
    ax2 : matplotlib.axis.Axis
        Axis for time-series plot.
    T, F, S : numpy.ndarray (ndim 2)
```

```
        Time, frequency, S-transform tiles from stockwell.stransform
    d1, d2 : numpy.ndarray (ndim 1)
        Time-series, plotted black.  Optional d2 plotted gray. These need to
        be registered in time to T.
    color1 : str
        Color of plotted d1 line.
    label1, label2 : str
        Time-series legend label strings.
    arrivals : dict
        Sequence of arrivals to plot, of the form {label: time_in_seconds, ...}
    dlim : 2-tuple of floats
        Limits on the time-series amplitudes (y axis limits).
    hatch : numpy.ndarray (ndim 2)
        Optional tile used for hatch mask.
    hatchlim : tuple
        Hatch range used to display mask.  2-tuple of floats (hmin, hmax).
    amp_fmt : str
        Matplotlib format string used for amplitudes in colorbars and axes.
    cmap : matplotlib.colors.Colormap
    alpha: float
        Optionally, use a white transparency mask for overlying the hatch, with
        the given alpha value.

    Returns
    -------
    matplotlib.collections.QuadMesh
        The ax1 image from pcolormesh.

    Examples
    --------
    # filtered versus unfiltered radial, and set color limits
    >>> plot_tile(fig, ax21, T, F, Srs, ax22, rs, 'unfiltered', rsf, 'NIP filtered',
        arrivals=arrivals, flim=(0.0, fmax), clim=(0.0, 5e-5), hatch=sfilt, hatchlim=(0.0, 0.8))
    # scalar versus dynamic rotated radial
    >>> plot_tile(fig, ax21, T, F, Srs, ax22, rs, 'scalar', rd, 'dynamic', arrivals
        flim=(0.0, fmax), clim=(0.0, 5e-5), hatch=dfilt, hatchlim=(0.0, 0.8))

    """
    if not fig:
        fig = plt.figure()

    sciformatter = FormatStrFormatter(amp_fmt)

    # grab a time vector
    tm = T[0]

    # TODO: remove fig from signature?
    im = plot_image(T, F, np.abs(S), hatch=hatch, hatchlim=hatchlim, flim=flim,
                    clim=clim, fig=fig, ax=ax1, cmap=cmap, alpha=alpha)
    cbar = plt.colorbar(im, fraction=0.05, pad=0.01, ax=[ax1, ax2],
                        format=amp_fmt)

    # waves and arrivals
    if d1 is not None:
        ax2.plot(tm, d1, color1, label=label1)
        # ax2.set_ylabel('amplitude')
        # set view limits
        dmx = d1.max()
        dmn = d1.min()
    else:
        # XXX
        dmx = d2.max()
        dmn = d2.min()

    if d2 is not None:
        ax2.plot(tm, d2, 'gray', label=label2, zorder=1)
        dmx = max([dmx, d2.max()])
        dmn = min([dmn, d2.min()])
    if not dlim:
        dlim = (dmn, dmx)
    ax2.set_ylim(dlim)
    ax2.set_xlabel('time [seconds]')
    ax2.set_xlim(tm[0], tm[-1])

    leg = ax2.legend(loc='lower right', frameon=False, fontsize=14)
    for legobj in leg.legendHandles:
        legobj.set_linewidth(2.0)

    if arrivals:
        plot_arrivals(ax2, arrivals, dmn, dmx)

    #ax2.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
```

```
    #ax2.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
    ax2.yaxis.set_major_formatter(sciformatter)
    fig.add_subplot(ax2)

    return im



def plot_arrivals(ax, arrivals, dmin, dmax):                                    [docs]
    """
    arrivals : dict
        are a dict of {name: seconds}.
    dmin, dmax : float
        y value in axis "ax" at which labels are plotted.
    """
    for arr, itt in arrivals.items():
        ax.vlines(itt, dmin, dmax, 'k', linestyle='dashed')
        ax.text(itt, dmax, arr, fontsize=12, horizontalalignment='left',
                va='top')


def make_tiles(fig, gs0, full=None):                                            [docs]
    """
    Give a list of (ax_top, ax_bottom) axis tuples for each SubPlotSpec in gs0.

    full : list
        Integer subplotspec numbers for which the ax_top is to take up the
        whole tile, so no ax_bottom is to be created.  Returns these tiles'
        axis handles as (ax_top, None).

    """
    if not full:
        full = []

    axes = []
    for i, igs in enumerate(gs0):
        iigs = gridspec.GridSpecFromSubplotSpec(3, 1, subplot_spec=igs,
                                                hspace=0.0)
        ax1 = plt.Subplot(fig, iigs[:-1, :])
        if i in full:
            ax2 = None
        else:
            ax2 = plt.Subplot(fig, iigs[-1, :], sharex=ax1)
        axes.append((ax1, ax2))

    return axes




def plot_image(T, F, C, hatch=None, hatchlim=None, flim=None, clim=None,        [docs]
               fig=None, ax=None, cmap=None, alpha=None):
    """
    Plot a fime-frequency image, optionally with a hatched mask.

    T, F, C : numpy.ndarray (ndim 2)
        Time and frequency domain arrays, and data arrays.
    hatch : numpy.ndarray (rank 2)
        Optional array mask for hatching or for opacity.
    hatchlim : float
        Optional hatch value cutoff for masking, above which masking is not
        done.  If hatch is provided but hatchlim is not, hatch will be used as
        an opacity mask, and it values are assumed to be between 0 and  1.

    Returns
    -------
    matplotlib.collections.QuadMesh from pcolormesh

    """
    if not fig:
        fig = plt.figure()
    if not ax:
        ax = fig.gca()

    ax.axes.get_xaxis().set_visible(False)

    if not cmap:
        cmap = plt.cm.viridis
    im = ax.pcolormesh(T, F, C, cmap=cmap)

    if flim:
        ax.set_ylim(flim)
```

```python
    if clim:
        im.set_clim(clim)

    if (hatch is not None) and hatchlim:
        if alpha:
            ax.contourf(T, F, hatch, hatchlim, colors='w', alpha=alpha)
        else:
            ax.contourf(T, F, hatch, hatchlim, colors='k', linewidths=0.5, hatches=['x'], alpha=0.0)
            ax.contour(T, F, hatch, [max(hatchlim)], linewidths=0.5, colors='k')

    ax.set_ylabel('frequency [Hz]')
    ax.set_yscale('log')
    ax.get_yaxis().set_tick_params(direction='out', which='both')
    fig.add_subplot(ax)

    return im



def plot_instantaneous_azimuth(T, F, theta, fs=1.0, flim=None, dlim=None,
                               clim=None, fig=None, ax=None):
    """
    Plot the instantanous azimuth TF tile using imshow.

    Parameters
    ----------
    T, F, theta : numpy.ndarray (ndim 2)
        Time [sec], frequency [Hz], and instantaneous azimuth calculated by
        stockwell.filter.instantanous_azimuth
    fs : float
        Sampling rate of data used.
    flim, dlim, clim : tuple (min, max)
        Optional frequency [Hz], time [sec], or color min/max limits for plots.
    fig : matplotlib.Figure instance
    ax : matplotlib.axis.Axis instance

    Returns
    -------
    matplotlib.Figure

    """
    if not fig:
        f = plt.figure()

    # plt.imshow(theta, origin='lower', cmap=plt.cm.hsv, aspect='auto',
    #            extent=[0, theta.shape[1], 0, fs/2.0], interpolation='nearest')
    plt.pcolormesh(T, F, theta)
    plt.colorbar()
    plt.axis('tight')

    if flim:
        plt.ylim(flim)

    if dlim:
        plt.xlim(dlim)

    if not clim:
        mx = np.nanmax(theta)
        plt.clim(-mx, mx)

    return f
```

[docs]

```python
def tile_comparison(T, F, Sv, Srs, Srd, Sts, Std, v, rs, rd, ts, td,
                    arrivals, flim, clim, dlim, hatch=None, hatchlim=None,
                    fig=None, xlim=None):
    """
    Make a 6-panel side-by-side comparison of tiles, such as scalar versus
    dynamic rotations.

    Sv, Srs, Srd, Sts, Std : numpy.ndarray (ndim 2)
        The vertical, radial-scalar, radial-dynamic, transverse-scalar, and
        transverse-dynamic stockwell transforms.
    v, rs, rd, ts, td : numpy.ndarray (ndim 1)
        The corresponding vertical, radial-scalar, radial-dynamic,
        transverse-scalar, and transverse-dynamic time-series vectors.
    arrivals : sequence of (str, float) 2-tuples
        Sequence of arrivals to plot, of the form (label, time_in_seconds)
    flim, clim, dlim, xlim : tuple
        Frequency, stockwell amplitude, time-series amplitude, and time-series
```

[docs]

```
        time limits of display.  2-tuples of (min, max) floats.
    hatch : numpy.ndarray (ndim 2)
        Optional tile used for hatch mask.
    hatchlim : tuple
        Hatch range used to display mask.  2-tuple of floats (hmin, hmax).

    Returns
    -------
    matplotlib.Figure

    """
    # from http://matplotlib.org/1.3.1/users/gridspec.html
    if not fig:
        fig = plt.figure()

    gs0 = gridspec.GridSpec(3, 2)
    gs0.update(hspace=0.15, wspace=0.15, left=0.05, right=0.95, top=0.95,
               bottom=0.05)
    tile1, tile2, tile3, tile4, tile5, tile6 = make_tiles(fig, gs0)
    ax11, ax12 = tile1
    ax21, ax22 = tile2
    ax31, ax32 = tile3
    ax41, ax42 = tile4
    ax51, ax52 = tile5
    ax61, ax62 = tile6

    ax11.set_title('Vertical')
    plot_tile(fig, ax11, T, F, Sv, ax12, v, 'vertical', arrivals=arrivals,
              flim=flim, clim=clim, dlim=dlim)

    ax21.set_title('Vertical')
    plot_tile(fig, ax21, T, F, Sv, ax22, v, 'vertical', arrivals=arrivals,
              flim=flim, clim=clim, dlim=dlim)

    ax31.set_title('Radial, scalar')
    plot_tile(fig, ax31, T, F, Srs, ax32, rs, 'great circle', 'k', rd, 'dynamic',
              arrivals=arrivals, flim=flim, clim=clim, dlim=dlim, hatch=hatch,
              hatchlim=hatchlim)
    #ax31.contour(T, F, theta - az_prop, [20, 0.0, -20], linewidth=1.5,
    #             colors=['r','w','b'])
    #ax31.contour(T, F, theta - az_prop, [-40, 0, 40], cmap=plt.cm.seismic)

    ax41.set_title('Radial, dynamic')
    plot_tile(fig, ax41, T, F, Srd, ax42, rd, 'dynamic', 'k', rs, 'great circle',
              arrivals=arrivals, flim=flim, clim=clim, dlim=dlim, hatch=hatch,
              hatchlim=hatchlim)

    ax51.set_title('Transverse, scalar')
    plot_tile(fig, ax51, T, F, Sts, ax52, ts, 'great circle', 'k', td, 'dynamic',
              arrivals=arrivals, flim=flim, clim=clim, dlim=dlim, hatch=hatch,
              hatchlim=hatchlim)
    #ax51.contour(T, F, theta - az_prop, [40, 0.0, -40], linewidth=1.5,
    #             colors=['r','w','b'])
    #ax51.contour(T, F, theta - az_prop, [-40, 0, 40], cmap=plt.cm.seismic)

    ax61.set_title('Transverse, dynamic')
    plot_tile(fig, ax61, T, F, Std, ax62, td, 'dynamic', 'k', ts, 'great circle',
              arrivals=arrivals, flim=flim, clim=clim, dlim=dlim, hatch=hatch,
              hatchlim=hatchlim)

    if xlim:
        ax11.set_xlim(*xlim)
        ax21.set_xlim(*xlim)
        ax31.set_xlim(*xlim)
        ax41.set_xlim(*xlim)
        ax51.set_xlim(*xlim)
        ax61.set_xlim(*xlim)

    return fig



def check_filters(T, F, Sv, Srs, Sts, vsf, rsf, ts, arrivals, flim, clim,       [docs]
                  dlim, xlim, hatch=None, hatchlim=None, fig=None):
    """
    Parameters
    ----------
    T, F : numpy.ndarray (ndim 2)
        The time and freqency domain tiles/grids.
    Sv, Srs, Sts : numpy.ndarray (ndim 2)
        The vertical, scalar-rotated radial, and scalar-rotated transverse
        Stockwell transform tiles.
```

```
    vsf, rsf, ts : numpy.ndarray (ndim 1)
        The Stockwell NIP-filtered vertical and radial, and transverse
        time-series vectors.
    arrivals : sequence of (str, float) 2-tuples
        Sequence of arrivals to plot, of the form (label, time_in_seconds)
    dlim : 2-tuple of floats
        Limits on the time-series amplitudes (y axis limits).
    hatch : numpy.ndarray (ndim 2)
        Optional tile used for hatch mask.
    hatchlim : tuple
        Hatch range used to display mask.  2-tuple of floats (hmin, hmax).
    fig : matplotlib.Figure

    Returns
    -------
    matplotlib.Figure

    """
    if not fig:
        fig = plt.figure()

    gs0 = gridspec.GridSpec(3, 1)
    gs0.update(hspace=0.15, wspace=0.15, left=0.05, right=0.95, top=0.95,
               bottom=0.05)

    tile1, tile2, tile3 = make_tiles(fig, gs0)
    ax11, ax12 = tile1
    ax21, ax22 = tile2
    ax31, ax32 = tile3

    ax11.set_title('Vertical, scalar rotation')
    plot_tile(fig, ax11, T, F, Sv, ax12, vsf, 'filtered', 'k', v, 'vertical',
              arrivals=arrivals, flim=flim, clim=clim, dlim=dlim, hatch=hatch,
              hatchlim=hatchlim)

    ax21.set_title('Radial, scalar rotation')
    plot_tile(fig, ax21, T, F, Srs, ax22, rsf, 'filtered', 'k', rs, 'radial',
              arrivals=arrivals, flim=flim, clim=clim, dlim=dlim, hatch=hatch,
              hatchlim=hatchlim)

    ax31.set_title('Transverse, scalar rotation')
    plot_tile(fig, ax31, T, F, Sts, ax32, ts, 'transverse', arrivals=arrivals,
              flim=flim, clim=clim, dlim=dlim, hatch=hatch, hatchlim=hatchlim)

    if xlim:
        ax11.set_xlim(*xlim)
        ax21.set_xlim(*xlim)
        ax31.set_xlim(*xlim)

    return fig



def plot_NIP(T, F, nips, fs=1.0, flim=None, fig=None, ax=None):        [docs]
    """
    Plot the normalized inner product tile.

    Parameters
    ----------
    T, F, nips : numpy.ndarray (ndim 2)
        Time, frequency, normalized inner-product tiles.
    fs : float
        Sampling frequency of the underlying time-series data.
    flim : tuple
        Frequency limits as (fmin, fmax) 2-tuple, in Hz.

    Returns
    -------
    matplotlib.Figure

    """
    if not fig:
        fig = plt.figure()

    # plt.imshow(nips, cmap=plt.cm.seismic, origin='lower',
    #            extent=[0,nips.shape[1], 0, fs/2], aspect='auto',
    #            interpolation='nearest')
    if ax:
        im = ax.pcolormesh(T, F, nip, cmap=plt.cm.seismic)
    else:
        im = plt.pcolormesh(T, F, nip, cmap=plt.cm.seismic)
    plt.colorbar()
```

```python
    plt.colorbar()
    plt.contour(T, F, nips, [0.8], linewidths=2.0, colors='k')
    plt.axis('tight')
    if flim:
        plt.ylim(flim)
    plt.ylabel('frequency [Hz]')
    plt.xlabel('time [sec]')

    return fig



def compare_waveforms(v, vsf, rs, rsf, ts, arrivals):                    [docs]
    """
    Compare the static and dynamically filtered waveforms.

    A 3-panel waveform plot of 3 components.  Unfiltered waves are in gray,
    filtered are overplotted in black.

    Parameters
    ----------
    v, vsf : numpy.ndarray (rank 1)
        Unfiltered and static-rotated filtered vertical waveform.
    rs, rsf : numpy.ndarray (rank 1)
        Unfiltered and static-rotated filtered radial waveform.
    ts: numpy.ndarray (rank 1)
        Unfiltered transverse waveform.

    """
    plt.subplot(311)
    plt.title('vertical')
    plt.plot(v, 'gray', label='original')
    plt.plot(vsf, 'k', label='NIP filtered')
    plt.legend(loc='lower left')

    plt.subplot(312)
    plt.title('radial')
    plt.plot(rs, 'gray', label='original')
    plt.plot(rsf, 'k', label='NIP filtered')
    plt.legend(loc='lower left')

    plt.subplot(313)
    plt.title('transverse')
    plt.plot(ts, 'gray', label='original')
    plt.legend(loc='lower left')

    # plot arrivals
    for arr, itt in arrivals.items():
        plt.vlines(itt, v.min(), v.max(), 'k', linestyle='dashed')
        plt.text(itt, v.max(), arr, fontsize=9, horizontalalignment='left',
                 va='top')



def NIP_filter_plots(T, F, theta, fs, Sr, St, Sv, r, t, v, rf, tf=None, vf=None,    [docs]
                     arrivals=None, flim=None, hatch=None, hatchlim=None, fig=None):
    """
def NIP_filter_plots(T, F, theta, fs, Sr, St, Sv, rf, r, vf, v, t, tf=None,
    Quad plot of NIP, and 3 tiles of Stockwell transform with NIP filter hatch
    and filtered+unfiltered time-series for each component.

    Parameters
    ----------
    T, F, theta: numpy.ndarray (ndim 2)
        Time, frequency, instantaneous azimuth tiles.
    fs : float
        Sampling rate of underlying time-series data.
    Sr, St, Sv : numpy.ndarray (ndim 2, complex)
        Stockwell transform of the radial, transverse, and vertical component data.
    r, t, v: numpy.ndarray (ndim 1)
        Unfiltered radial, transverse, and vertical component time-series.
    rf, tf, vf : numpy.ndarray (ndim 1)
        NIP-filtered radial, transverse, and vertical component time-series.
    arrivals : sequence of (str, float) 2-tuples
        Sequence of arrivals to plot, of the form (label, time_in_seconds)
    flim : tuple
        Frequency limits as (fmin, fmax) 2-tuple, in Hz.
    hatch : numpy.ndarray (ndim 2)
        Optional tile used for cross-hatch visual mask.
    hatchlim : tuple
        Hatch range used to display mask.  2-tuple of floats (hmin, hmax).
    fig : matplotlib.Figure
```

```
    Returns
    -------
    tiles : list
        List of two-tuples of axes objects (axis_top, axis_bottom) of each tile,
        clockwise from top-left.  Can be unpacked to get all axes as follows:

    >>> tiles = NIP_filter_plots(...)
    >>> (ax11, ax12), (ax21, ax22), (ax31, ax32), (ax41, ax42) = tiles

    """
    if not fig:
        fig = plt.figure()

    # 2x2 grid of tiles
    gs0 = gridspec.GridSpec(2, 2)
    gs0.update(hspace=0.10, wspace=0.12, left=0.04, right=0.96, top=0.95,
               bottom=0.05)

    tile1, tile2, tile3, tile4 = make_tiles(fig, gs0)
    ax11, ax12 = tile1 #top left
    ax21, ax22 = tile2 #top right
    ax31, ax32 = tile3 #bottom left
    ax41, ax42 = tile4 #bottom right


    # top left axes: Instantaneous and weighted mean azimuth
    mean_theta = np.ma.average(theta, axis=0, weights=hatch)
    # opacity_mask = np.array([np.abs(S) / np.abs(S).max() for S in (Sr, St, Sv)]).mean(axis=0)
    ax11.set_title('Instantaneous propagation azimuth')
    _ = plot_tile(fig, ax11, T, F, theta, ax12, mean_theta, 'filter-weighted mean',
                  'k', arrivals=arrivals, flim=flim,
                  dlim=[mean_theta.min(), mean_theta.max()], hatch=hatch,
                  hatchlim=hatchlim, amp_fmt='%d', cmap=plt.cm.nipy_spectral, alpha=1.0)

    # top right: Vertical
    # s transform and filter
    ax21.set_title('Vertical')
    _ = plot_tile(fig, ax21, T, F, Sv, ax22, vf, 'filtered', 'k', v, 'original',
                  arrivals, flim=flim, hatch=hatch, hatchlim=hatchlim)

    # bottom right: Radial
    # s transform and filter
    ax41.set_title('Radial')
    _ = plot_tile(fig, ax41, T, F, Sr, ax42, rf, 'filtered', 'k', r, 'original',
                  arrivals, flim=flim, hatch=hatch, hatchlim=hatchlim)

    # bottom left: Transverse
    # s transform and filter
    ax31.set_title('Transverse')
    _ = plot_tile(fig, ax31, T, F, St, ax32, tf, 'filtered', 'k', t, 'original',
                  arrivals=arrivals, flim=flim, hatch=hatch, hatchlim=hatchlim)

    return [tile1, tile2, tile3, tile4]
```