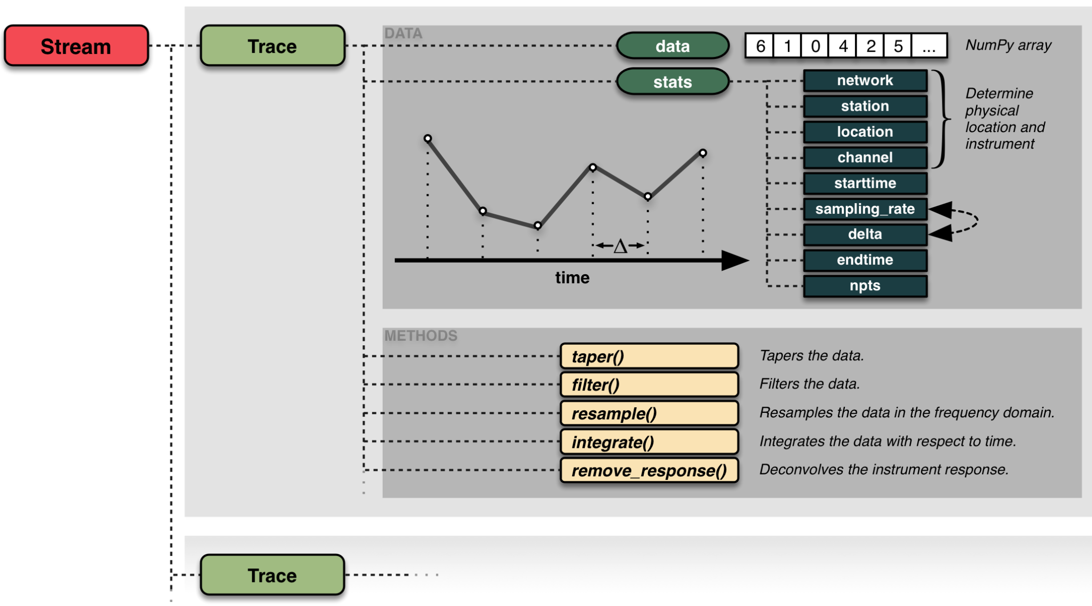
7. Process Visualization

January 24, 2024

1 Process Visualization

A series of scripts illustrating how to use infrapy subroutines as stand-alone modules

1.1 Exploratory Data Analysis



[ ]:
```
import obspy
from obspy import read

st = read('/run/media/viblab/Markov2/Haykal/AnakKrakatauEWS/data/raw/I06AU/
↪I06AU_SAC/IM.I06H*..BDF__20180709T000000Z__20180710T000000Z.sac')
print(st)
```

```
8 Trace(s) in Stream:
IM.I06H1..BDF | 2018-07-09T00:00:00.000000Z - 2018-07-10T00:00:16.100000Z | 20.0
Hz, 1728323 samples
IM.I06H2..BDF | 2018-07-09T00:00:15.100000Z - 2018-07-10T00:00:15.800000Z | 20.0
Hz, 1728015 samples
IM.I06H3..BDF | 2018-07-09T00:00:13.100000Z - 2018-07-10T00:00:00.000000Z | 20.0
```

```
Hz, 1727739 samples
IM.I06H4..BDF | 2018-07-09T00:00:00.000000Z - 2018-07-10T00:00:15.600000Z | 20.0
Hz, 1728313 samples
IM.I06H5..BDF | 2018-07-09T00:00:15.350000Z - 2018-07-10T00:00:16.200000Z | 20.0
Hz, 1728018 samples
IM.I06H6..BDF | 2018-07-09T00:00:16.550000Z - 2018-07-10T00:00:00.000000Z | 20.0
Hz, 1727670 samples
IM.I06H7..BDF | 2018-07-09T00:00:00.000000Z - 2018-07-10T00:00:16.650000Z | 20.0
Hz, 1728334 samples
IM.I06H8..BDF | 2018-07-09T00:00:16.149999Z - 2018-07-10T00:00:16.049999Z | 20.0
Hz, 1727999 samples
```

[ ]:
```
tr = st[0]
tr.data
```

[ ]:
```
array([-441., -565., -661., …, 6904., 6678., 6811.], dtype=float32)
```

[ ]:
```
tr = st[1]
tr.data
```

[ ]:
```
array([6568., 6521., 6538., …, 4796., 5026., 5125.], dtype=float32)
```

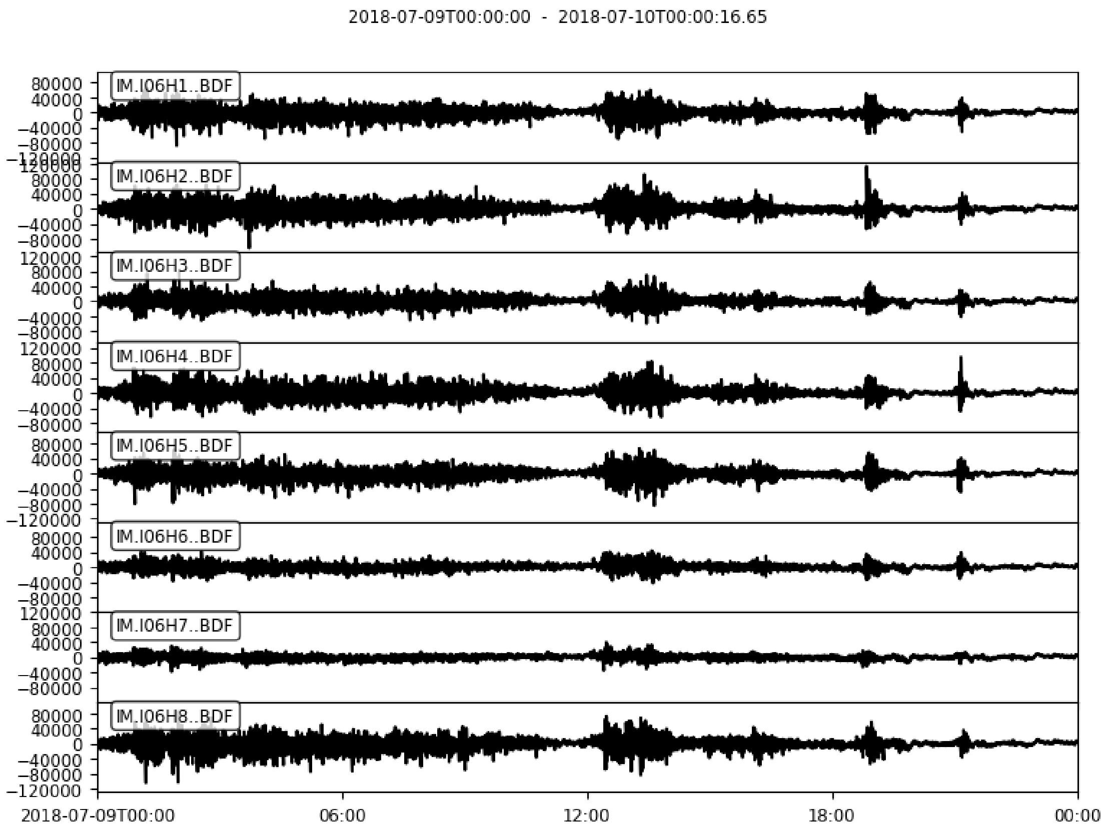
[ ]:
```
tr = st[2]
tr.data
```

[ ]:
```
array([3399., 3222., 3168., …, 5327., 5284., 5076.], dtype=float32)
```

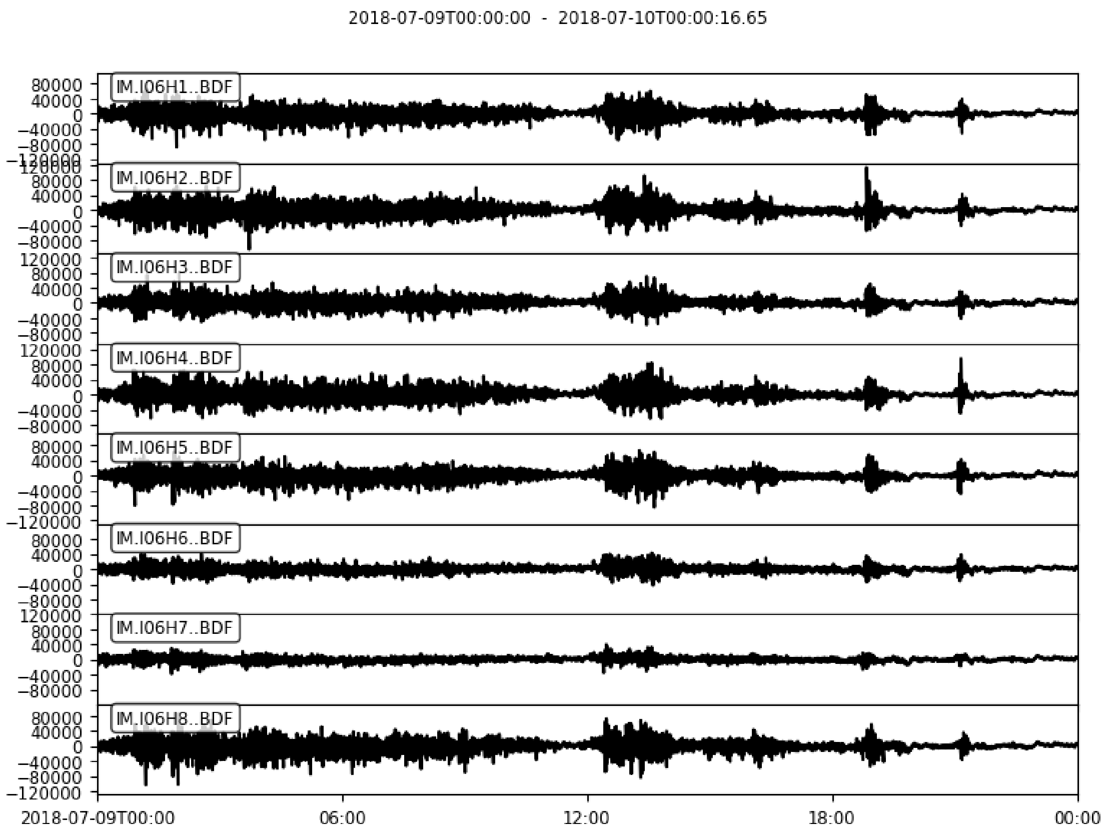
[ ]:
```
tr = st[2]
tr.stats
```

[ ]:
```
         network: IM
         station: I06H3
        location:
         channel: BDF
       starttime: 2018-07-09T00:00:13.100000Z
         endtime: 2018-07-10T00:00:00.000000Z
   sampling_rate: 20.0
           delta: 0.05
            npts: 1727739
           calib: 1.0
         _format: SAC
             sac: AttribDict({'delta': 0.05, 'depmin': -58468.0, 'depmax':
80937.0, 'b': 0.0, 'e': 86386.9, 'stla': -12.14509, 'stlo': 96.81774, 'stel':
20.8, 'depmen': -241.49185, 'nzyear': 2018, 'nzjday': 190, 'nzhour': 0, 'nzmin':
0, 'nzsec': 13, 'nzmsec': 100, 'nvhdr': 6, 'npts': 1727739, 'iftype': 1,
'leven': 1, 'lpspol': 0, 'lovrok': 1, 'lcalda': 1, 'kstnm': 'I06H3', 'kcmpnm':
'BDF', 'knetwk': 'IM'})
```

[ ]:
```
st.plot(size=(800,600))
```



[ ]:



1.2 Beamforming:

Run Bartlett, Capon or Generalized Least Squares beamforming processes on an hour-long dataset from the BRP array in Utah

[ ]:
```
import numpy as np
from multiprocess import Pool
import matplotlib.pyplot as plt
import matplotlib.cm as cm
palette = cm.jet
import matplotlib.ticker as mtick
from obspy.core import read
from scipy import signal
from infrapy.detection import beamforming_new

import warnings
warnings.filterwarnings("ignore")
```

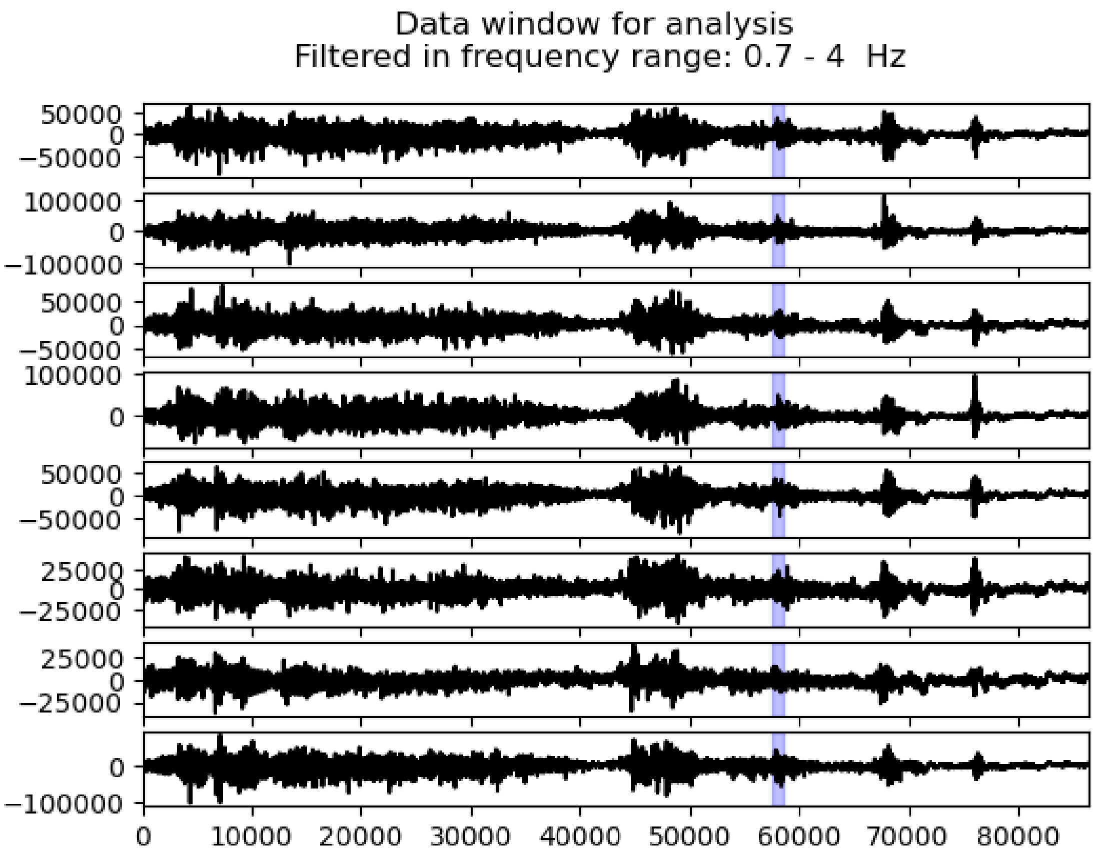
[ ]:
```
# ######################### #
#     Define Parameters     #
# ######################### #
```

```
sac_glob = "/run/media/viblab/Markov2/Haykal/AnakKrakatauEWS/data/raw/I06AU/
 ↪I06AU_SAC/IM.I06H*..BDF__20180709T000000Z__20180710T000000Z.sac" ## load in␣
 ↪SAC files for processing

freq_min, freq_max = 0.7, 4 ## define frequency band of interest
window_length, window_step = 10.0, 2.5 ## define window length and window step␣
 ↪for beamforming

ns_start, ns_end = 100.0, 400.0 ## define noise window (in sec); only needed␣
 ↪for GLS processing
sig_start, sig_end = 57600, 58600 ## define signal window [time window in sec␣
 ↪used for analysis]

back_az_vals = np.arange(-180.0, 180.0, 1.5)
trc_vel_vals = np.arange(300.0, 600.0, 2.5)

method="bartlett" ## beamforming method; options are bartlett, capon, GLS

p = Pool(10) ## define number of CPUs used for processing
```

[ ]:
```
# ######################### #
#  Read, Shift Start Time,  #
#      and Filter Data      #
# ######################### #
x, t, t0, geom = beamforming_new.stream_to_array_data(read(sac_glob))
M, N = x.shape
```

[ ]:
```
# ######################### #
#         View Data         #
# ######################### #
plt.figure()
for m in range(M):
    plt.subplot(M, 1, m + 1)
    plt.xlim([0, t[-1]])
    plt.plot(t, x[m], 'k-')
    plt.axvspan(xmin = sig_start , xmax = sig_end, alpha = 0.25, color = 'blue')
    if method == "gls":
        plt.axvspan(xmin = ns_start , xmax = ns_end, alpha = 0.25, color =␣
 ↪'red')
    if m < (M - 1) : plt.setp(plt.subplot(M, 1, m + 1).get_xticklabels(),␣
 ↪visible=False)

if method == "gls":
    plt.suptitle("Data windows for signal (blue) and noise (red) \n Filtered in␣
 ↪frequency range: " + str(freq_min) + " - " + str(freq_max) + "  Hz \n ")
else:
```

```
    plt.suptitle("Data window for analysis \n Filtered in frequency range: " +
 ↪str(freq_min) + " - " + str(freq_max) + "  Hz \n ")

plt.show(block=False)
plt.pause(0.1)
```



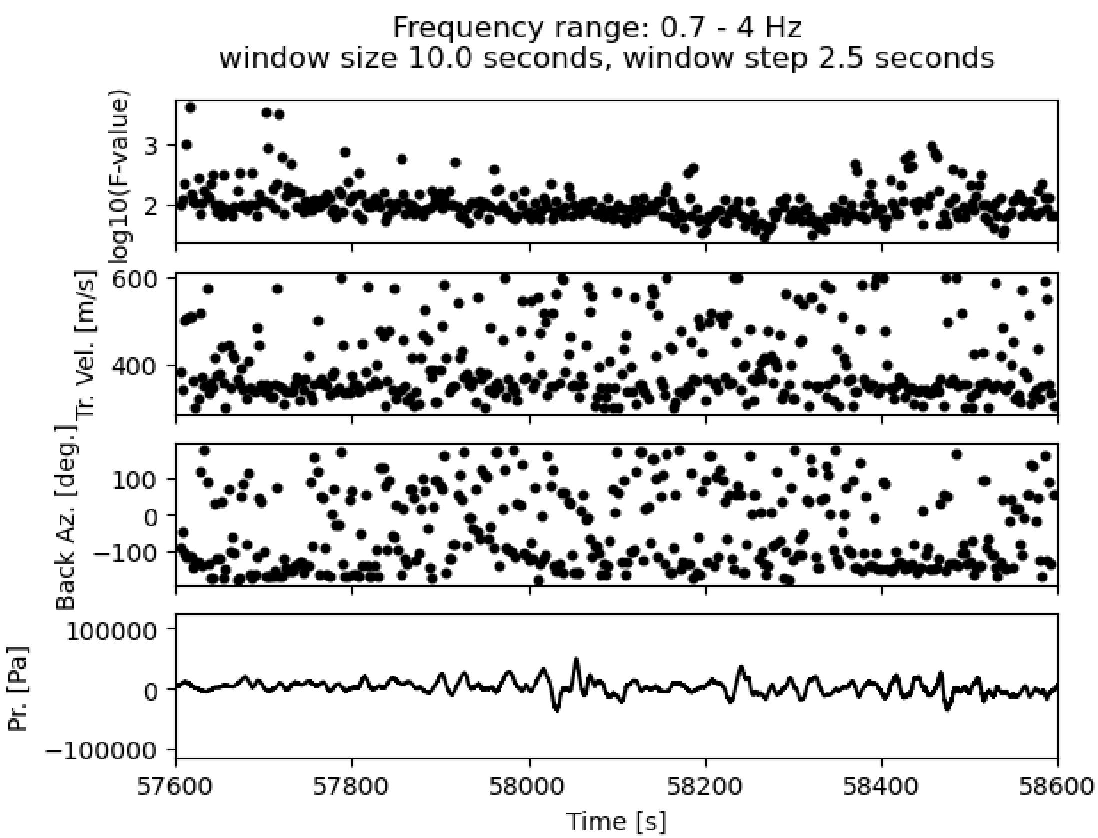
[ ]:

```
# ######################### #
#        Run Methods        #
# ######################### #

# define slowness and delays
slowness = beamforming_new.build_slowness(back_az_vals, trc_vel_vals)
delays = beamforming_new.compute_delays(geom, slowness)

# define the noise covariance if using generalized least squares method
if method == "gls":
    _, S, _ = beamforming_new.fft_array_data(x, t, window=[ns_start, ns_end],
 ↪sub_window_len=window_length)
```

```
    ns_covar_inv = np.empty_like(S)
    for n in range(S.shape[2]):
        S[:, :, n] += 1.0e-3 * np.mean(np.diag(S[:, :, n])) * np.eye(S.shape[0])
        ns_covar_inv[:, :, n] = np.linalg.inv(S[:, :, n])
else:
    ns_covar_inv = None



# Run beamforming in windowed data and write to file
times, beam_results = [],[]
for window_start in np.arange(sig_start, sig_end, window_step):
    if window_start + window_length > sig_end:
        break

    times = times + [[t0 + np.timedelta64(int(window_start), 's')]]
    X, S, f = beamforming_new.fft_array_data(x, t, window=[window_start,␣
 ↪window_start + window_length])
    beam_power = beamforming_new.run(X, S, f, geom, delays, [freq_min,␣
 ↪freq_max], method="bartlett", pool=p, normalize_beam=True,␣
 ↪ns_covar_inv=ns_covar_inv)
    peaks = beamforming_new.find_peaks(beam_power, back_az_vals, trc_vel_vals,␣
 ↪signal_cnt=1)
    beam_results = beam_results + [[peaks[0][0], peaks[0][1], peaks[0][2] / (1.
 ↪0 - peaks[0][2]) * (x.shape[0] - 1)]]

times = np.array(times)[:, 0]
beam_results = np.array(beam_results)
```

[ ]:
```
# Prep figure
f, a = plt.subplots(4, sharex=True)
plt.xlim([sig_start, sig_end])
a[3].set_xlabel("Time [s]")
a[3].set_ylabel("Pr. [Pa]")
a[2].set_ylabel("Back Az. [deg.]")
a[1].set_ylabel("Tr. Vel. [m/s]")
if method == "music":
    a[0].set_ylabel("Beam Power")
else:
    a[0].set_ylabel("log10(F-value)")

a[3].plot(t, x[1,:], '-k')
plt.suptitle("Frequency range: " + str(freq_min) + " - " + str(freq_max) + " Hz␣
 ↪\n window size " + str(window_length) + " seconds, window step " +␣
 ↪str(window_step) +  " seconds")

for aa in range(len(times)):
```

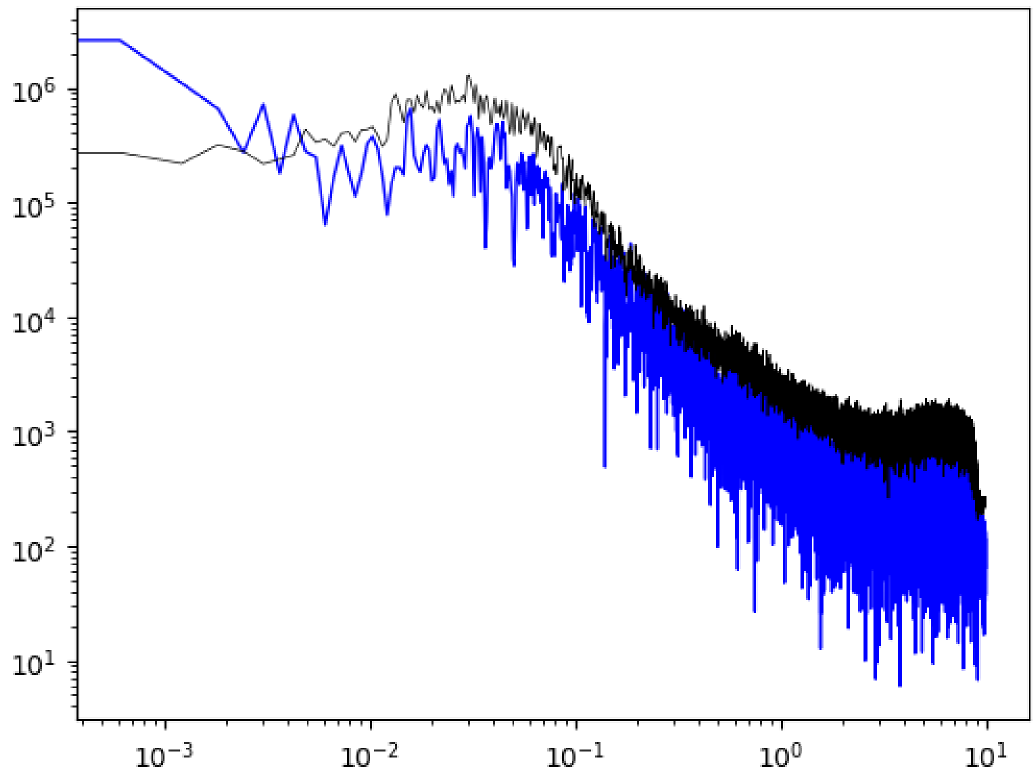
```
    dt = times[aa]-times[0]
    start = dt.item().total_seconds()
    start = start + sig_start
    if method == "music":
        a[2].plot([start + 1.0 / 2.0 * window_length], [beam_results[aa][0]],␣
↪'ok', markersize=3.3)
        a[1].plot([start + 1.0 / 2.0 * window_length], [beam_results[aa][1]],␣
↪'ok', markersize=3.3)
        a[0].plot([start + 1.0 / 2.0 * window_length], [beam_results[aa][2]],␣
↪'ok', markersize=3.3)
        plt.pause(0.1)
    else:
        a[2].plot([start + 1.0 / 2.0 * window_length], [beam_results[aa][0]],␣
↪'ok', markersize=3.3)
        a[1].plot([start + 1.0 / 2.0 * window_length], [beam_results[aa][1]],␣
↪'ok', markersize=3.3)
        a[0].plot([start + 1.0 / 2.0 * window_length], [beam_results[aa][2]],␣
↪'ok', markersize=3.3)
plt.show(block=False)
```



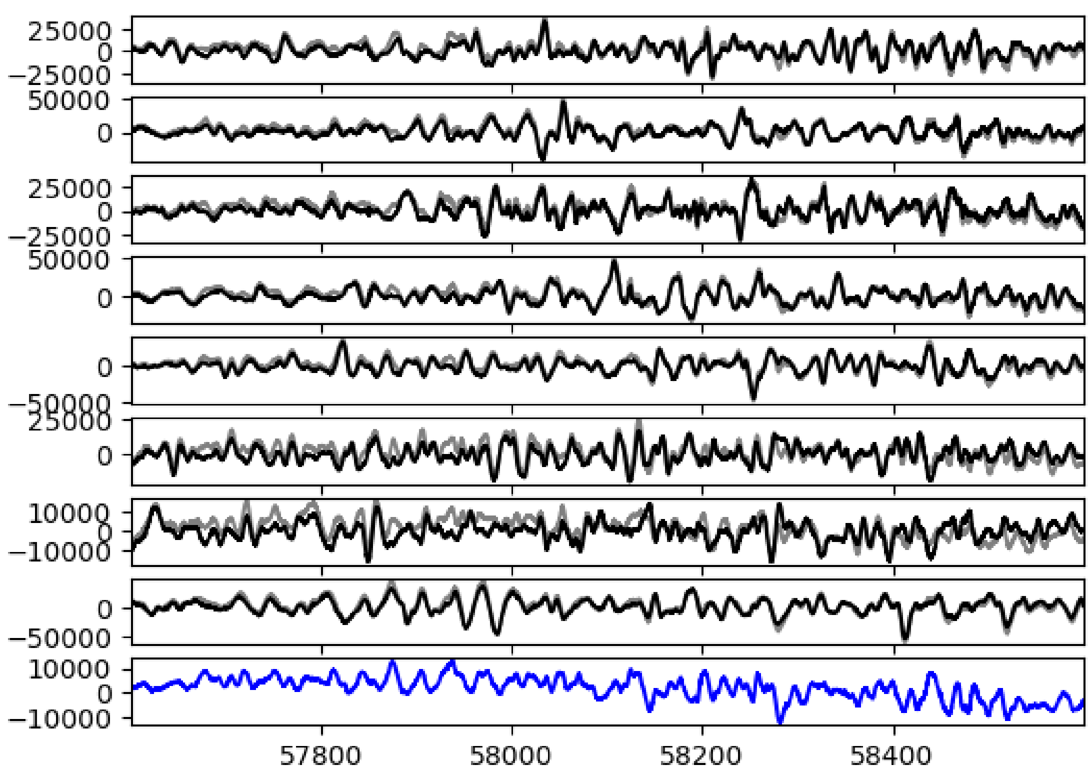
[ ]:
```python
# ########################## #
#        Save Results        #
# ########################## #

np.save("times", times)
np.save("beam_results", beam_results)
```

[ ]:
```python
# ####################################### #
#        Define Beam and Residuals        #
# ####################################### #
back_az = beam_results[np.argmax(beam_results[:, 2]), 0]
tr_vel = beam_results[np.argmax(beam_results[:, 2]), 1]

X, S, f = beamforming_new.fft_array_data(x, t, window=[sig_start, sig_end],␣
 ↪fft_window="boxcar")
sig_est, residual = beamforming_new.extract_signal(X, f, np.array([back_az,␣
 ↪tr_vel]), geom)

plt.figure()
plt.loglog(f, abs(sig_est), '-b', linewidth=1.0)
plt.loglog(f, np.mean(abs(residual), axis=0), '-k', linewidth=0.5)

signal_wvfrm = np.fft.irfft(sig_est) / (t[1] - t[0])
resid_wvfrms = np.fft.irfft(residual, axis=1) / (t[1] - t[0])
t_mask = np.logical_and(sig_start < t, t < sig_end)

plt.figure()
for m in range(M):
    plt.subplot(M + 1, 1, m + 1)
    plt.xlim([t[t_mask][0], t[t_mask][-1]])
    plt.plot(t[t_mask], x[m, t_mask], '0.5')
    plt.plot(t[t_mask], resid_wvfrms[m, :len(t[t_mask])], 'k-')
    plt.setp(plt.subplot(M + 1, 1, m + 1).get_xticklabels(), visible=False)
plt.subplot(M + 1, 1, M + 1)
plt.xlim([t[t_mask][0], t[t_mask][-1]])
plt.plot(t[t_mask], signal_wvfrm[:len(t[t_mask])], 'b-')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f435b46e640>]
```



10^6
10^5
10^4
10^3
10^2
10^1
10^{-3}
10^{-2}
10^{-1}
10^0
10^1



25000
0
−25000
50000
0
25000
0
−25000
50000
0
0
−50000
25000
0
10000
0
−10000
0
−50000
10000
0
−10000
57800
58000
58200
58400

1.3 Detection

Run detection on the series of beamforming results produced in the above step

[ ]:
```
# ######################### #
#     Define Parameters     #
# ######################### #

# Detection params
# times_file, beam_results_file = None, None
times_file, beam_results_file = "times.npy", "beam_results.npy"

det_win_len = 60 * 5
det_thresh = 0.99
min_seq = 5
det_method = "fstat"
TB_prod = 40 * 10
back_az_lim = 10
M=4
```

[ ]:
```
# ################################## #
#   Load data and prepare analysis   #
# ################################## #

if times_file and beam_results_file:
    times = np.load(times_file)
    beam_results = np.load(beam_results_file)
else:
    print('No beamforming input provided')
```

[ ]:
```
# ################################## #
#       Run detection analysis       #
# ################################## #

dets = beamforming_new.detect_signals(times, beam_results, det_win_len,␣
 ↪TB_prod, channel_cnt=M, det_thresh=det_thresh, min_seq=min_seq,␣
 ↪back_az_lim=back_az_lim)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[49], line 5
      1 # ################################## #
      2 #       Run detection analysis       #
      3 # ################################## #
```

```
----> 5 dets =␣
  ↪beamforming_new.detect_signals(times, beam_results, det_win_len, TB_prod, channel_cnt=M, de

TypeError: detect_signals() got an unexpected keyword argument 'det_thresh'
```

[ ]:
```
# ################################## #
#       Print Detection Summary      #
# ################################## #
print('\n' + "Detection Summary:")
for det in dets:
    print("Detection time:", det[0], '\t', "Rel. detection onset:", det[1],␣
 ↪'\t',"Rel. detection end:", det[2], '\t',end=' ')
    print("Back azimuth:", det[3], '\t', "Trace velocity:", det[4], '\t',␣
 ↪"F-stat:", det[5], '\t', "Array dim:", M)
```

```
Detection Summary:
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[21], line 5
      1 # ################################## #
      2 #       Print Detection Summary      #
      3 # ################################## #
      4 print('\n' + "Detection Summary:")
----> 5 for det in dets:
      6     print("Detection time:", det[0], '\t', "Rel. detection onset:",␣
  ↪det[1], '\t',"Rel. detection end:", det[2], '\t',end=' ')
      7     print("Back azimuth:", det[3], '\t', "Trace velocity:", det[4],␣
  ↪'\t', "F-stat:", det[5], '\t', "Array dim:", M)

NameError: name 'dets' is not defined
```

[ ]:
```
def find_nearest(a, a0):
    "Element in nd array `a` closest to the scalar value `a0`"
    idx = np.abs(a - a0).argmin()
    return a.flat[idx]
```

[ ]:
```
# ################################## #
#       Plot Detection Results       #
# ################################## #

plt.figure()
plt.suptitle("Detection results for analysis \n Filtered in frequency range: "␣
 ↪+ str(freq_min) + " - " + str(freq_max) + "  Hz \n ")
```

```
for det in range(len(dets)):
    dt = dets[det][0]-times[0]
    start = dt.item().total_seconds()
    ts = sig_start + start + dets[det][1]
    te = sig_start + start + dets[det][2]
    for m in range(M):
        plt.subplot(M, 1, m + 1)
        plt.xlim([sig_start, sig_end])
        plt.plot(t, x[m], 'k-')
        plt.axvspan(xmin = ts , xmax = te, alpha = 0.25, color = 'red')
        if m < (M - 1) : plt.setp(plt.subplot(M, 1, m + 1).get_xticklabels(),␣
 ↪visible=False)


f, a = plt.subplots(4, sharex=True)
plt.xlim([sig_start, sig_end])
a[3].set_xlabel("Time [s]")
a[3].set_ylabel("Pr. [Pa]")
a[2].set_ylabel("Back Az. [deg.]")
a[1].set_ylabel("Tr. Vel. [m/s]")
if method == "music":
    a[0].set_ylabel("Beam Power")
else:
    a[0].set_ylabel("log10(F-value)")

a[3].plot(t, x[1,:], '-k')
plt.suptitle("Detection Processing Results")

position = []
for det in range(len(dets)):
    dt = dets[det][0]-times[0]
    start = dt.item().total_seconds()
    ts = sig_start + start + dets[det][1]
    te = sig_start + start + dets[det][2]
    a[3].axvspan(xmin = ts , xmax = te, alpha = 0.25, color = 'red')

    duration = te-ts
    duration = duration/window_step

    for bb in range(0,int(duration),1):
        temp = dets[det][0]+np.timedelta64(int(dets[det][1]),'s')+np.
 ↪timedelta64(int(window_step*bb),'s')
        det_time=find_nearest(times, temp)
        det_times = np.where(times==det_time)
        pos = det_times[0][0]
        position.append(pos)
for aa in range(len(times)):
```

```python
    dt = times[aa]-times[0]
    start = dt.item().total_seconds()
    start = start + sig_start
    a[2].plot([start], [beam_results[aa][0]], 'ok', markersize=3.3)
    a[1].plot([start], [beam_results[aa][1]], 'ok', markersize=3.3)
    a[0].plot([start], [beam_results[aa][2]], 'ok', markersize=3.3)
for aa in position:
    dt = times[aa]-times[0]
    start = dt.item().total_seconds()
    start = start + sig_start
    a[2].plot([start], [beam_results[aa][0]], 'or', markersize=3.3)
    a[1].plot([start], [beam_results[aa][1]], 'or', markersize=3.3)
    a[0].plot([start], [beam_results[aa][2]], 'or', markersize=3.3)

plt.show(block=False)
```

[ ]:
```python
########################################################
##      Plot Detection Results in Slowness Space      ##
########################################################

for det in range(len(dets)):
    dt = dets[det][0]-times[0]
    start = dt.item().total_seconds()
    ts = sig_start + start + dets[det][1]
    te = sig_start + start + dets[det][2]
    X, S, f = beamforming_new.fft_array_data(x, t, window=[ts, te])
    beam_power = beamforming_new.run(X, S, f, geom, delays, [freq_min,␣
 ↪freq_max], method=method, signal_cnt=1, pool=p, ns_covar_inv=ns_covar_inv,␣
 ↪normalize_beam=True)

    avg_beam_power = np.average(beam_power, axis=0)
        #avg_beam_power = beamforming_new.multi_freq_beam(beam_power)
    print('Detection #' + str(det+1))
    plt.figure()
    plt.clf()
    plt.xlim([min(slowness[:, 0]), max(slowness[:, 0])])
    plt.ylim([min(slowness[:, 1]), max(slowness[:, 1])])
    if method == "bartlett_covar" or method == "bartlett" or method == "gls":
        plt.scatter(slowness[:, 0], slowness[:, 1], c=avg_beam_power,␣
 ↪cmap=palette, marker="o", s=[12.5] * len(slowness), edgecolor='none', vmin=0.
 ↪0, vmax=1.0)
    else:
        plt.scatter(slowness[:, 0], slowness[:, 1], c=avg_beam_power,␣
 ↪cmap=palette, marker="o", s=[12.5] * len(slowness), edgecolor='none', vmin=0.
 ↪0, vmax=np.max(avg_beam_power))
    plt.pause(1.0)
```

```
    # Compute back azimuth projection of distribution
    az_proj, tv_proj = beamforming_new.project_beam(beam_power, back_az_vals, trc_vel_vals, method="mean")

    plt.figure()
    plt.suptitle("Average Beam Power")

    plt.clf()
    plt.xlim([min(back_az_vals), max(back_az_vals)])
    plt.xlabel('Backazimuth')
    plt.ylabel('Avg. Beam Power')
    if method == "bartlett_covar" or method == "bartlett" or method == "gls":
        plt.ylim([0.0, 1.0])
    else:
        plt.ylim([0.0, np.max(avg_beam_power)])
    plt.plot(back_az_vals, az_proj, '-k', linewidth=2.5)
    plt.pause(0.2)
```

1.4 Association

Associate a number of detections contained in a .dat file (/data/detection_set1.dat or /data/detection_set2.dat)

[ ]:
```
import numpy as np
from multiprocess import Pool

from infrapy.association import hjl
from infrapy.propagation import likelihoods as lklhds
```

[ ]:
```
########################
### Define parameters ###
########################

# Read in detections from file
det_list = lklhds.json_to_detection_list('../examples/data/detection_set1.json')

# define joint-likelihood calculation parameters
width = 10.0
rng_max = 3000.0

# define clustering parameters
dist_max = 10.0
clustering_threshold = 5.0
trimming_thresh = 3.0

pl = Pool(4)
```

```
######################
#### Run analysis ####
######################
labels, dists = hjl.run(det_list, clustering_threshold, dist_max=dist_max,␣
↪bm_width=width, rng_max=rng_max, trimming_thresh=trimming_thresh,␣
↪pool=pl,show_result=True)
```

```
############################
#### Summarize Clusters ####
############################
clusters, qualities = hjl.summarize_clusters(labels, dists)
for n in range(len(clusters)):
    print("Cluster:", clusters[n], '\t', "Cluster Quality:", 10.
↪0**(-qualities[n]))
```

1.5 Location

Test the Bayesian Infrasonic Source Localization (BISL) methodology using a set of provided detections (/data/detection_set1.dat or /data/detection_set2.dat). Location will be run twice, once assuming uniform atmospheric propagation and a second time applying provided atmospheric propagation priors for the Western US (see Blom et al., 2015 for further explanation)

```
import numpy as np

from infrapy.location import bisl
from infrapy.propagation import likelihoods as lklhds
from infrapy.propagation import infrasound as infsnd
```

```
# ######################### #
#       Define Inputs       #
# ######################### #

# Define ground_truth if known (41.131, -112.896 for UTTR; Test includes show␣
↪in June 2004)
grnd_trth = [41.131, -112.896, np.datetime64('2004-06-02T17:23:04.0')]

# Define localization parameters
bm_width = 12.5
rad_min, rad_max = 50.0, 500.0
rng_max = np.pi / 2.0 * 6370.0
resolution = int(np.sqrt(1e5))
```

```
# ################################# #
#       Define Detection List       #
# ################################# #

'''
```

```
# Define the list of detections (output from association)
# detection format: (lat, lon, arrival time, back az, F stat, elements)
# arrival time format: datetime.datetime(year, month, day, hour, minute, second)
det1 = lklhds.InfrasoundDetection(42.7668, -109.5939, np.
 ↪datetime64('2004-06-02T17:42:14.0'), -125.6, 75.0, 4)
det2 = lklhds.InfrasoundDetection(38.4296, -118.3036, np.
 ↪datetime64('2004-06-02T17:50:38.0'),   56.6, 75.0, 4)
det3 = lklhds.InfrasoundDetection(48.2641, -117.1257, np.
 ↪datetime64('2004-06-02T18:09:14.0'),  157.5, 75.0, 4)
det_list = [det1, det2, det3]
'''

# Load detection list from flat file
#det_list = lklhds.file2dets("data/detection_set2.dat")

# Load detection list from json file
det_list = lklhds.json_to_detection_list('../examples/data/detection_set2.json')
```

[ ]:
```
# ########################## #
#          Run BISL          #
#       in Verbose Mode      #
# ########################## #

# Run analysis without priors
result,pdf = bisl.run(det_list,
                    bm_width=bm_width,
                    rad_min=rad_min,
                    rad_max=rad_max,
                    rng_max=rng_max,
                    resol=resolution,angle=[-180,180])

summary = bisl.summarize(result)
```

[ ]:
```
# ########################## #
#     Display Results        #
# ########################## #

print('-' * 75)
print('BISL Summary\n')
print(summary)
print('\n' + '-'*75 + '\n')
```

[ ]:
```
# ########################## #
#          Define Priors,    #
#          Load from File    #
#           and Display      #
# ########################## #
```

```
model = infsnd.PathGeometryModel()
model.load("../infrapy/propagation/priors/UTTR_models/UTTR_06_1800UTC.pgm")
#model.display()
```

[ ]:
```
# ########################## #
#          Run BISL          #
#       in Verbose Mode      #
# .       With Priors .      #
# ########################## #

result,pdf = bisl.run(det_list,
                    bm_width=bm_width,
                    rad_min=rad_min,
                    rad_max=rad_max,
                    rng_max=rng_max,
                    resol=resolution,
                    path_geo_model=model,angle=[-180,180])

summary = bisl.summarize(result)
```

[ ]:
```
# ########################## #
#     Display Results       #
# ########################## #

print('-' * 75)
print('BISL Summary\n')
print(summary)
print('\n' + '-'*75 + '\n')
```

[ ]:
```

```