hw7

March 13, 2023

0.1 A.)

Estimate the attenuation for waves of 30 s period (0.033 Hz) for PcP ray paths at vertical incidence. Consider only the intrinsic attenuation along the ray paths; do not include the reflection coefficient at the core–mantle boundary or geometrical spreading. Be sure to count both the surface-to- CMB and CMB-to-surface legs. You can get the PREM P velocities from Figure 1.1 (Copied below) or the values tabulated in Appendix A. You may estimate the travel times through the different Q layers in PREM by assuming a fixed velocity within each layer (e.g., use the average velocity between 3 and 80 km, the average between 80 and 220 km, etc.). Note that in this case, the "surface" is assumed to be the bottom of the ocean at 3 km depth. Compute tstar for PcP. Finally, give the attenuated amplitude of the PcP ray, assuming the ray had an initial amplitude of one.

[1]:
```
# PcP = Surface through mantle, off core, to surface

# calculate attenuation using eq. 6.86
# A(w) = A_0(w) * exp((-w*tstar) / 2)
# tstar = int(dt / Q(r)) = 1/Q(r) * int_path(dt)
```

[2]:
```
import numpy as np

# depth of different Q layers
depth = np.array([
    (80-3),
    (220-80),
    (670-220),
    (2891-670),
    (2891-670),
    (670-220),
    (220-80),
    (80-3),
])

# calculate mean alpha for each velocity change in Q layer
alpha = np.array([
    np.mean((5.8, 6.8, 8.11, 8.08)),
    np.mean((8.08, 8.02, 7.99)),
    np.mean((8.56, 8.66, 8.85, 8.91, 9.13, 9.5, 10.01, 10.16, 1.16, 10.27)),
    np.mean(
```

```
        (10.75, 11.07, 11.24, 11.42, 11.58, 11.73, 11.88, 12.02, 12.1, 12.29, 12.
↪42,
        12.54, 12.67, 12.78, 12.9, 13.02, 13.13, 13.25, 13.36, 13.48, 13.60, 13.
↪68,
        13.69, 13.71, 13.72)
    ),

    np.mean(
        (10.75, 11.07, 11.24, 11.42, 11.58, 11.73, 11.88, 12.02, 12.1, 12.29, 12.
↪42,
        12.54, 12.67, 12.78, 12.9, 13.02, 13.13, 13.25, 13.36, 13.48, 13.60, 13.
↪68,
        13.69, 13.71, 13.72)
    ),
    np.mean((8.56, 8.66, 8.85, 8.91, 9.13, 9.5, 10.01, 10.16, 1.16, 10.27)),
    np.mean((8.08, 8.02, 7.99)),
    np.mean((5.8, 6.8, 8.11, 8.08)),

])

# Q values to outer core from PREM
Q = np.array([600, 80,143, 312, 312, 143, 80, 600])
```

[3]:
```
# v = d / t -> t = d / v
dt = depth / alpha
tstar = dt / Q
```

[4]:
```
w = 2 * np.pi * 0.033
A_0 = 1
A = []
for i in range(0,8,1):
    A_0 = A_0 * np.exp(-w * tstar[i] / 2)
    A.append(A_0)
att_amp = A[-1]
print(f'The final attenuated amplited for the PcP wave is: {att_amp}')
```

```
The final attenuated amplited for the PcP wave is: 0.784300449848276
```

0.2 B.)

Using the approximation t=$4t$, compute the attenuated amplitude of ScS at 30 s period.

[5]:
```
tstar_b = 4 * tstar
```

[6]:
```
w = (2 * np.pi) / 30
A_0 = 1
A = []
```

```
for i in range(0,8,1):
    A_0 = A_0 * np.exp(-w * tstar_b[i] / 2)
    A.append(A_0)
att_amp = A[-1]
print(f'The final attenuated amplited for the ScS wave is: {att_amp}')
```

```
The final attenuated amplited for the ScS wave is: 0.37468518513264615
```

0.3 C.)

Repeat (a) and (b) for 1 s period (1 Hz) PcP and ScS waves.

[7]:
```
w = 2 * np.pi

A_0 = 1
A = []
for i in range(0,8,1):
    A_0 = A_0 * np.exp(-w * tstar[i] / 2)
    A.append(A_0)
att_amp = A[-1]
print(f'The final attenuated amplited for the PcP wave is: {att_amp}')

A_0 = 1
A = []
for i in range(0,8,1):
    A_0 = A_0 * np.exp(-w * tstar_b[i] / 2)
    A.append(A_0)
att_amp = A[-1]
print(f'The final attenuated amplited for the ScS wave is: {att_amp}')
```

```
The final attenuated amplited for the PcP wave is: 0.0006345983116549022
The final attenuated amplited for the ScS wave is: 1.6217938504235532e-13
```

0.4 D.)

Repeat your calculations for the PcP and ScS attenuated amplitudes at 1 Hz, but this time, use the Warren and Shearer (2000) Qα values from the figure below (Figure 6.17). How do the predicted amplitudes compare to the PREM model predictions? What do these results predict for the observability of teleseismic S arrivals at 1 Hz?

[8]:
```
w = 2 * np.pi

Q = [1/0.0043,1/0.0007,1/0.0007,1/0.0007,1/0.0007,1/0.0007,1/0.0007,1/0.0043]
tstar = dt / Q
tstar_b = 4 * tstar

A_0 = 1
A = []
```

```
for i in range(0,8,1):
    A_0 = A_0 * np.exp(-w * tstar[i] / 2)
    A.append(A_0)
att_amp = A[-1]
print(f'The final attenuated amplited for the PcP is: {att_amp}')

A_0 = 1
A = []
for i in range(0,8,1):
    A_0 = A_0 * np.exp(-w * tstar_b[i] / 2)
    A.append(A_0)
att_amp = A[-1]
print(f'The final attenuated amplited for the ScS wave is: {att_amp}')
```

```
The final attenuated amplited for the PcP is: 0.25267159680837875
The final attenuated amplited for the ScS wave is: 0.004075920455865417
```

The values from Warren and Shearer are much higher, and will drastically increase the observability of teleseismic S arrivals at 1 Hz

1 2.

This question relates to chapter 12.2 in noise seismology. It is modified from question 2 of chapter 12. Simulate the summations of cross-correlation functions from uniformly distributed noise sources

1.1 A.)

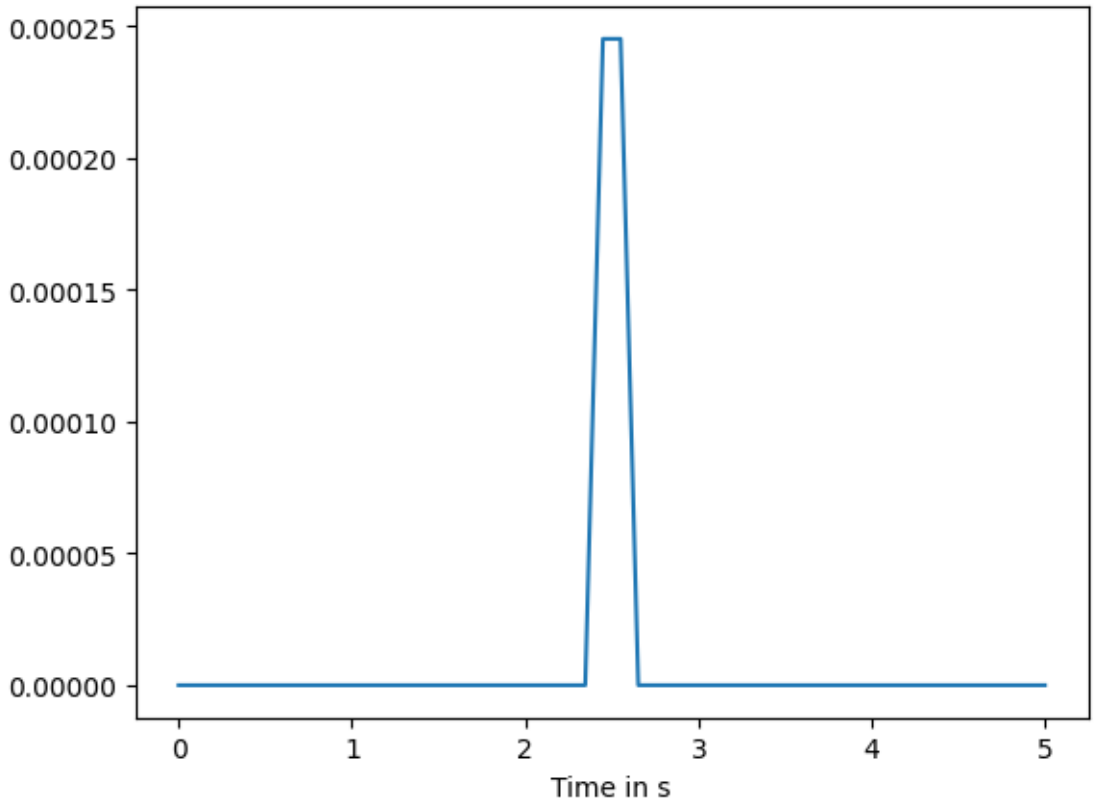
Produce your own version of the 2D example of Figure 12.3c by summing contributions from cross-correlations of a uniform azimuthal distribution of plane-wave sources. You should assume the cross-correlation functions are simple spike functions, as in Figure 12.3b. You will need to choose $d\vartheta$ small enough that you get a smooth v(t) function, given your chosen sample rate for v(t).

[9]:
```
from scipy import signal
import matplotlib.pyplot as plt
```

[10]:
```
# Sample rate
fs = 10

# window length
twin = 5

# points
t = np.linspace(0, twin, int(twin*fs))


# a single Ricker wavelet
a = 0.1# width of the Ricker wavelet in s
# ta = np.linspace(0,a,int(a*fs))
```

```
s = signal.ricker(len(t), a)
plt.plot(t,-s)
plt.xlabel('Time in s')
```

[10]: Text(0.5, 0, 'Time in s')



[11]:
```
# initialize the seismograms with zeros
n_1_x = np.zeros(len(t))
n_1_y = np.zeros(len(t))

# seismic wavespeed
c = 3

# station location
x = [-5.,0.]
y = [5.,0]
r = 10. # radius of the circle that has the noise sources


# assume a theta (in radian)
theta_1 = np.pi/3
```

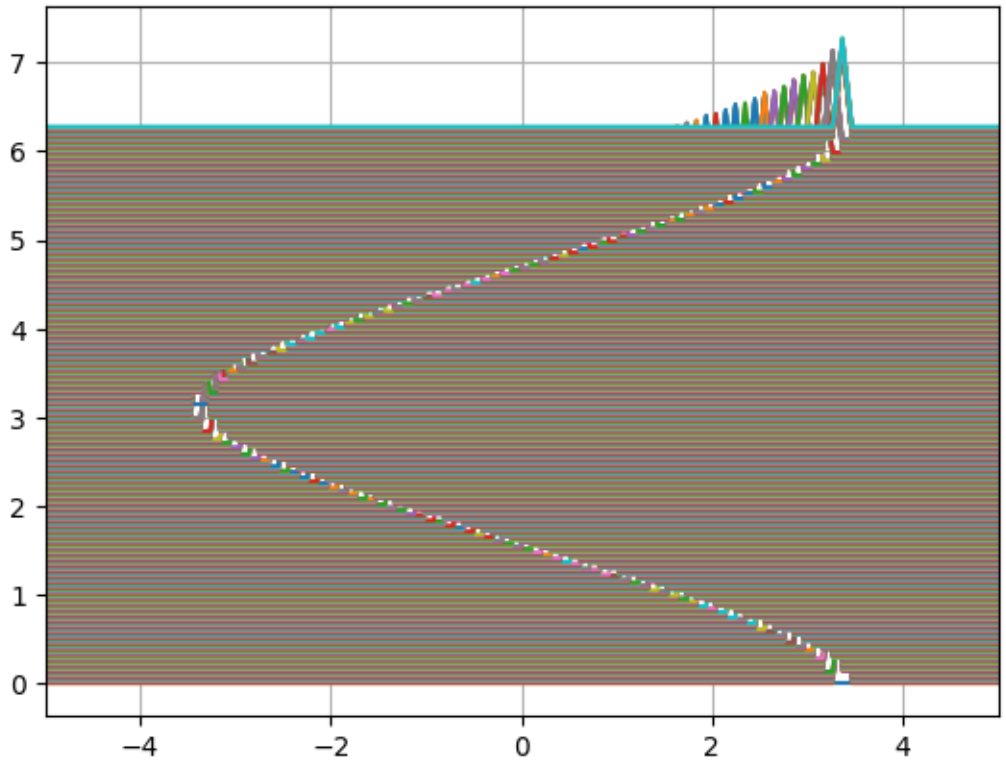
[12]:
```
# calculate the distance and travel times
dt_1_x = np.sqrt(
    (r*np.cos(theta_1) - x[0])**2 + (r*np.sin(theta_1) - x[1])**2
)/c

dt_1_y = np.sqrt(
    (r*np.cos(theta_1) - y[0])**2 + (r*np.sin(theta_1) - y[1])**2
)/c

print("travel time between sources and receivers ",dt_1_x," s",dt_1_y," s")
# indexes where the arrival time will be centered
# shift also with the ricker wavelet width to avoid acausal signals
# (ricker wavelet is centered at a)
i_1_x = int(dt_1_x*fs)
i_1_y = int(dt_1_y*fs)
print(i_1_x,i_1_y)
```

```
travel time between sources and receivers  4.409585518440984  s
2.8867513459481287  s
44 28
```

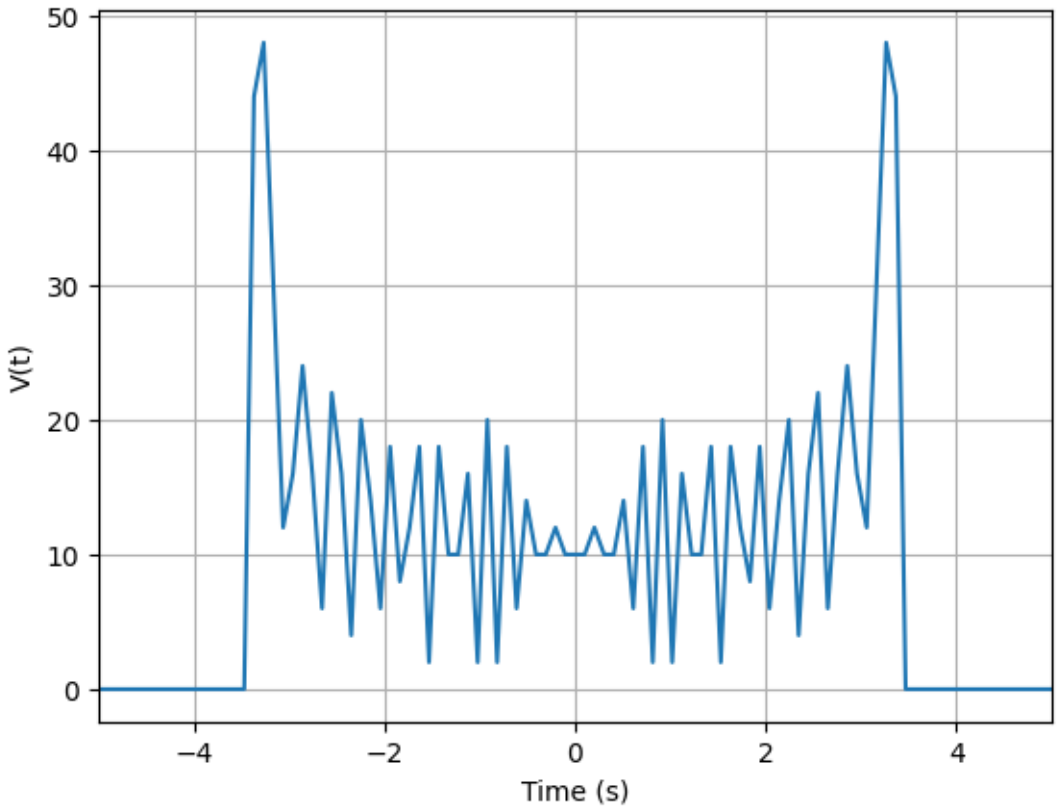
[13]:
```
tcorr = np.linspace(-twin,twin,int(2*len(t)-1))
```

[14]:
```
N = 1000 # we will create N noise sources
# initialize the cross correlations
Corr = np.zeros(shape=(N,len(tcorr)))
for i in range(N):
    theta = 2*np.pi/N*i
    # calculate the distance between each noise source and the receivers
    dt_x = np.sqrt(
      (r*np.cos(theta) - x[0])**2 + (r*np.sin(theta) - x[1])**2
    )/c

    dt_y = np.sqrt(
      (r*np.cos(theta) - y[0])**2 + (r*np.sin(theta) - y[1])**2
    )/c

    i_x = int(dt_x*fs)
    i_y = int(dt_y*fs)

    n_x= np.zeros(len(t))
    n_x[ i_x:i_x+1]=1

    n_y= np.zeros(len(t))
    n_y[ i_y:i_y+1]=1
    Corr[i,:] = np.correlate(n_x,n_y,'full')
    plt.plot(
```

```
        tcorr,
        Corr[i,:]+theta
    )
    plt.xlim([-5,5])
plt.grid()
```



[15]:
```
plt.plot(tcorr,np.sum(Corr,axis=0))
plt.xlim(-5,5)
plt.xlabel('Time (s)')
plt.ylabel('V(t)')
plt.grid()
```



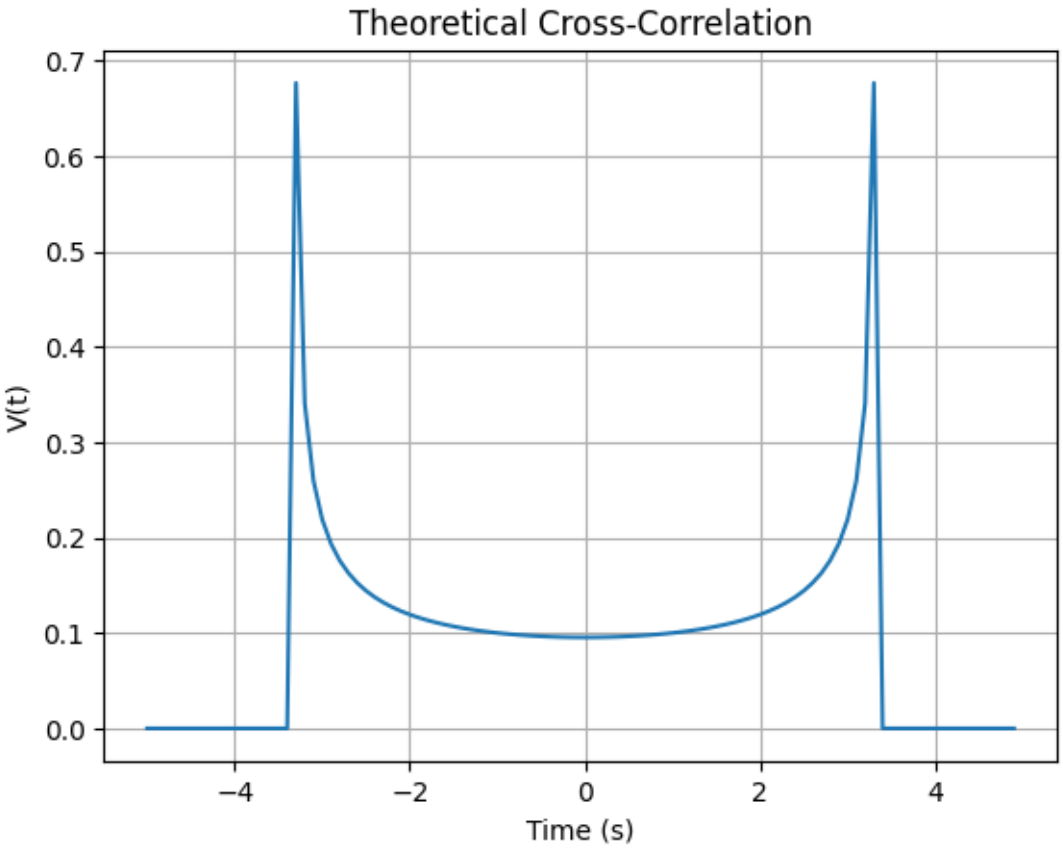
1.2 B.)

Show that your numerical integration results are consistent with equation 12.13 by plotting these analytical functions on the same plots. The equations in the books are:

[16]:
```
v_t = []
for t in np.arange(-5, 5, 0.1):
    if abs(t) >= r/c:
        v = 0
    else:
        v = 1 / (np.pi * np.sqrt((r**2 / c**2) - t**2))
    v_t.append(v)
    # break
```

[17]:
```
plt.plot(
    np.arange(-5,5,0.1),
    v_t
)
plt.title('Theoretical Cross-Correlation')
plt.xlabel('Time (s)')
plt.ylabel('V(t)')
```

```
plt.grid()
```



1.3 C.)

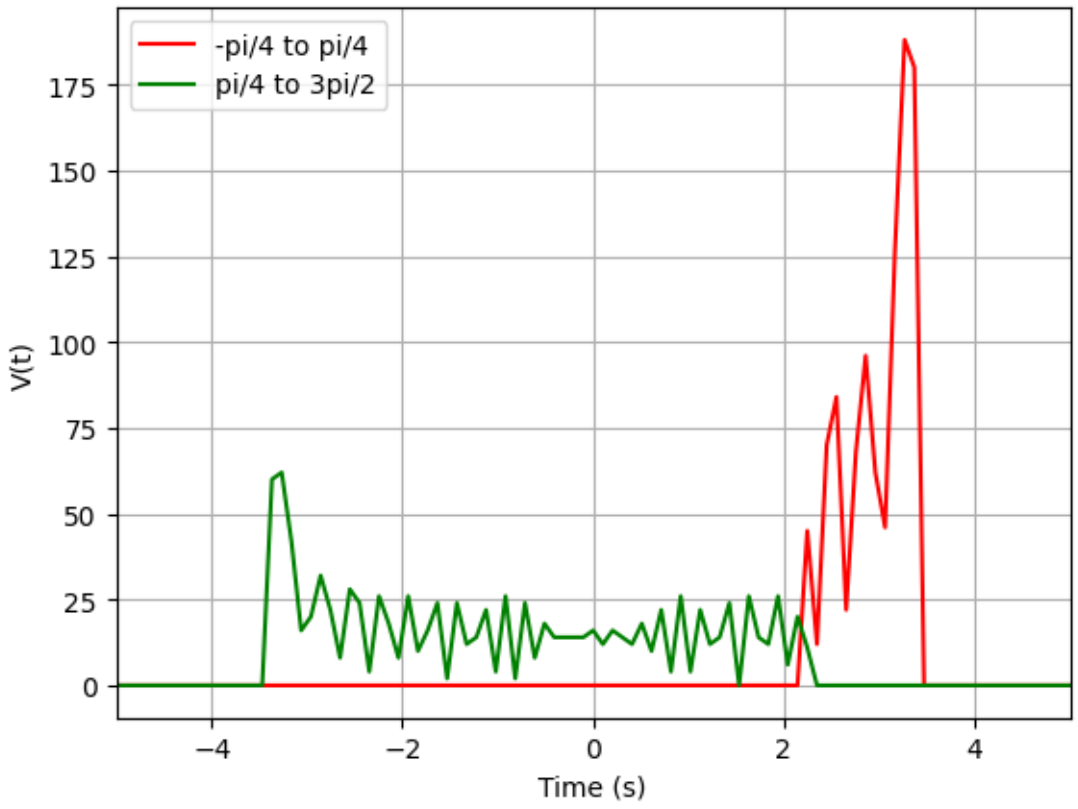
Perform partial integrations to look at the effects of imperfect noise source distributions. Only integrate from -pi/4 to pi/4. This assumes that the sources are only from one side of the station axis. Can you comment on the cross-correlation, and discuss the possible issues that will arise in practical examples? Try with angles from pi/4 to 3pi/2, what do you notice?

[18]:
```
N = 1000 # we will create N noise sources
# initialize the cross correlations
t = np.linspace(0,twin,int(twin*fs))

Corr = np.zeros(shape=(N,len(tcorr)))
for i in range(N):
    theta = -np.pi/(4)+i*1*np.pi/(2*N)
    # calculate the distance between each noise source and the receivers
    dt_x = np.sqrt(
      (r*np.cos(theta) - x[0])**2 + (r*np.sin(theta) - x[1])**2
    )/c
```

```
    dt_y = np.sqrt(
      (r*np.cos(theta) - y[0])**2 + (r*np.sin(theta) - y[1])**2
    )/c

    i_x = int(dt_x*fs)
    i_y = int(dt_y*fs)

    n_x = np.zeros(len(t))
    n_x[ i_x:i_x+1]=1

    n_y= np.zeros(len(t))
    n_y[ i_y:i_y+1]=1
    Corr[i,:] = np.correlate(n_x,n_y,'full')
plt.plot(
    tcorr,
    np.sum(Corr,axis=0),
    'r-',
    label='-pi/4 to pi/4')

N = 1000
for i in range(N):
    theta = +0.25*np.pi+i*3*np.pi/(2*N)
    # calculate the distance between each noise source and the receivers
    dt_x = np.sqrt(
      (r*np.cos(theta) - x[0])**2 + (r*np.sin(theta) - x[1])**2
    )/c

    dt_y = np.sqrt(
      (r*np.cos(theta) - y[0])**2 + (r*np.sin(theta) - y[1])**2
    )/c

    i_x = int(dt_x*fs)
    i_y = int(dt_y*fs)

    n_x = np.zeros(len(t))
    n_x[ i_x:i_x+1]=1

    n_y= np.zeros(len(t))
    n_y[ i_y:i_y+1]=1
    Corr[i,:] = np.correlate(n_x,n_y,'full')
plt.plot(
    tcorr,
    np.sum(Corr,axis=0),
    'g-',
    label='pi/4 to 3pi/2'
)
```

```
plt.xlim([-5,5])
plt.xlabel('Time (s)')
plt.ylabel('V(t)')
plt.grid()
plt.legend()
```

[18]: <matplotlib.legend.Legend at 0x7f971758c340>



The correlations seem to be strongly dependant on location relative to source.

[ ]: