The Differential Phase Method for Estimating Surface Wave Phase Velocity

Bill Menke, June 12, 2025

This is a review of a well-known technique.

Consider a plane wave with angular frequency $\omega \equiv 2\pi f$ and vector phase slowness (s_x, s_y). The scalar phase slowness is $s = \left(s_x^2 + s_y^2\right)^{1/2}$, the scalar phase velocity is $v = 1/s$ and the propagation angle from the positive y-axis is $\vartheta = \tan^{-1}(s_x/s_y)$. The vertical displacement u_i at position (x_i, y_i) is assumed to be the plane wave:

$$u_i = A_i \exp\{i\omega s_x x_i + i\omega s_y y_i - i\omega t\}$$

Here, the amplitude A_i and the slowness (s_x, s_y) are frequency dependent. The minus sign in the $-i\omega t$ time dependence is used because the waves then propagate in the positive direction when slowness is positive. The corresponding Fourier transform has a $+i\omega t$ time dependence. Unfortunately, Numpy's sign convection is reversed, so as we will point out below, a complex conjugate needs to be introduced into one of the equations when the algorithm is implemented in Python.

The Fourier transform of the displacement is

$$\tilde{u}_i = A_i \exp\{i\omega s_x x_i + i\omega s_y y_i\}$$

Given pair of stations (i, j) the displacement ratio is

$$R_{ij} \equiv \frac{\tilde{u}_i}{\tilde{u}_j} = \frac{A_i}{A_j} \exp\{i\omega s_x \Delta x_{ij} + i\omega s_y \Delta y_{ij}\} \equiv \frac{A_i}{A_j} \exp\{i\Delta\varphi_{ij}\}$$

where $\Delta\varphi_{ij} \equiv \omega s_x \Delta x_{ij} + \omega s_y \Delta y_{ij}$ is the differential phase. By Euler's formula

$$\mathrm{Re}\, R_{ij} + i\, \mathrm{Im}\, R_{ij} = \frac{A_i}{A_i} \cos\{\Delta\varphi_{ij}\} + i \frac{A_i}{A_i} \sin\{\Delta\varphi_{ij}\}$$

from whence

$$\frac{A_i}{A_i} \cos\{\Delta\varphi_{ij}\} = \mathrm{Re}\, R_{ij} \quad \text{and} \quad \frac{A_i}{A_i} \sin\{\Delta\varphi_{ij}\} = \mathrm{Im}\, R_{ij}$$

So,

$$\Delta\varphi_{ij} = \tan^{-1}\left\{\frac{-\mathrm{Im}\, R_{ij}}{\mathrm{Re}\, R_{ij}}\right\} + 2\pi n_{ij}$$

Here, the minus sign accounts for Numpy's sign convention, and n_{ij} is a yet-to-be-determined. frequency-dependent integer with the interpretation of "cycle skips". For very low frequency, we expect that $n_{ij} = 0$. It's value at a higher frequency can be determined from the requirement that

$\Delta\varphi_{ij}(\omega)$ is a continuous function of frequency. Starting with $k = 1$, we compare neighboring points, $\Delta\varphi_{ij}(\omega_{k-1})$ and $\Delta\varphi_{ij}(\omega_k)$ in the frequency series for the differential phase, and subtract 2π from all $\Delta\varphi_{ij}(\omega_p), p \geq k$ when $\Delta\varphi_{ij}(\omega_k) - \Delta\varphi_{ij}(\omega_{k-1})$ is greater than π, and add 2π when it is less than π. The Numpy `unwrap()` method implements this algorithm.

By measuring two pairs of stations, (i, j) and (i, k) we have two equations in two unknowns:

$$\omega^{-1}\Delta\varphi_{ij} = \Delta x_{ij} s_x + \Delta y_{ij} s_y$$

$$\omega^{-1}\Delta\varphi_{ik} = \Delta x_{ik} s_x + \Delta y_{ij} s_y$$

which can be solved by standard matrix methods.

In the following example, the source function is a Gaussion pulse with negligible energy above a frequency of 25 Hz. Consequently, estimates of phase velocity and azimuth are good only for frequencies lower than that value.

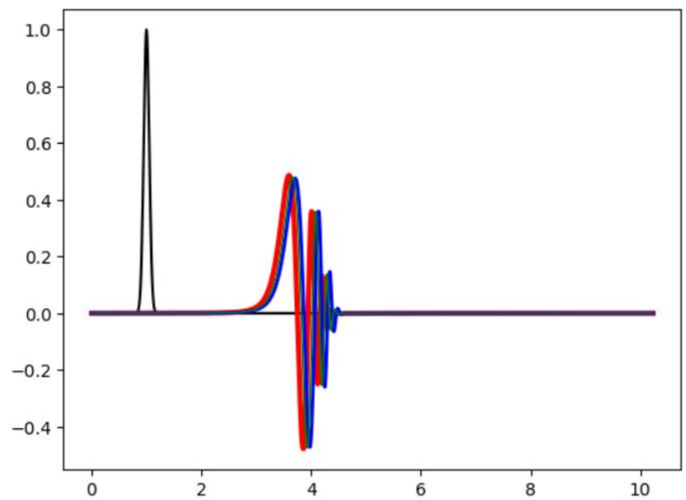
```
# two dimensional version, 3 stations in xy-plane

%reset -f
from math import exp, pi, sin, cos, tan, asin, acos, atan, atan2, sqrt, floor, ceil
import numpy as np
import scipy.linalg as la
from matplotlib import pyplot as plt

# station positions
x1 = 10.0114; y1 = 5.01;
x2 = 10.0114; y2 = 5.51;
x3 = 10.5114; y3 = 5.01;
Dx21 = x2-x1; Dy21 = y2-y1;
Dx31 = x3-x1; Dy31 = y3-y1;
theta = 30.8;

# time axis
N=1024;
No2 = int(N/2);
Dt = 0.01;
tmin=0.0;
t = np.zeros((N,1));
t[0:N,0] = Dt*np.linspace(0,N-1,N);
tmin = t[0,0];
tmax = t[N-1,0];

# generic frequency setup
fmax=1/(2.0*Dt);  # nyquist frequency
```

```
Df=fmax/No2;      # frequency sampling
Nf=No2+1;         # number of non-negative frequencies
# f is full freqeuncy vector:
# f = [0, Df, 2Df ... fmax, -fmax+Df, ... -Df]
f = np.zeros((N,1));
f[0:N,0] = Df*np.concatenate(
   (np.linspace(0,No2,Nf),np.linspace(-No2+1,-1,No2-1)),
   axis=0 );
Dw=2*pi*Df;       # angular frequency sampling
w=2*pi*f;         # full angular frequency vector
Nw=Nf;            # number of non-negative angular f's
fpos = np.zeros((Nf,1));
fpos[0:Nf,0] = Df * np.linspace(0,No2,Nf); # non-neg f's
wpos=2*pi*fpos;   # non-negative angular frequencies

# phase velocity
f0 = fmax/10.0;
v = 3.0*np.ones((N,1))+1.0*np.exp( -np.abs(f)/f0 );
s = np.reciprocal(v);
sx = s*sin((pi/180.0)*theta);
sy = s*cos((pi/180.0)*theta);
wsx = np.multiply(w,sx);
wsy = np.multiply(w,sy);

# positive frequencies only
fp = f[1:Nf,0:1];
wp = w[1:Nf,0:1];
Np, i = np.shape(wp);
vp = v[1:Nf,0:1];
sp = s[1:Nf,0:1];
spx = sx[1:Nf,0:1];
spy = sy[1:Nf,0:1];
wspx = wsx[1:Nf,0:1];
wspy = wsy[1:Nf,0:1];

# forward problem: predict seismogram from dispersion curve

# source pulse at (x,y)=(0,0)and time t0
t0 = 1.0;
sd0 = 0.05;
u0 = np.exp( -0.5*np.power(t-t0*np.ones((N,1)),2) / (sd0**2) );
u0t = np.fft.fft(u0,axis=0);

# station 1
```

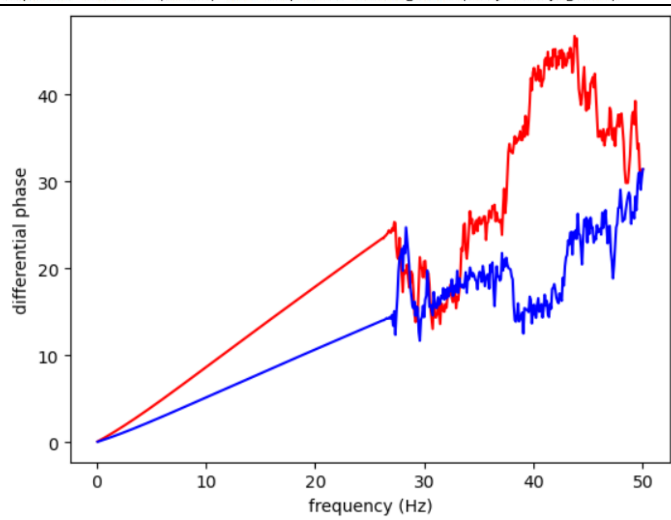
```
ph1 = np.exp( complex(0.0,-1.0)*(wsx*x1+wsy*y1) );
u1t = np.multiply( u0t, ph1 );
u1 = np.real( np.fft.ifft(u1t,axis=0) );

# station 2
ph2 = np.exp( complex(0.0,-1.0)*(wsx*x2+wsy*y2) );
u2t = np.multiply( u0t, ph2 );
u2 = np.real( np.fft.ifft(u2t,axis=0) );

# station 3
ph3 = np.exp( complex(0.0,-1.0)*(wsx*x3+wsy*y3) );
u3t = np.multiply( u0t, ph3 );
u3 = np.real( np.fft.ifft(u3t,axis=0) );

fig=plt.figure();
ax=plt.subplot(1,1,1);
plt.plot(t,u0,'k-');
plt.plot(t,u1,'r-', lw=3);
plt.plot(t,u2,'b-', lw=2);
plt.plot(t,u3,'g-', lw=1);
plt.show();
print('Caption: initial (black) and dispersed seismograms (red, blue)');

# inverse problem: calculate dispersion curve from seismogram

# take FFT of data and
# select only positive frequencies
u1t = np.fft.fft( u1, axis=0 );
u1tp = u1t[1:Nf,0:1];
u2t = np.fft.fft( u2, axis=0 );
u2tp = u2t[1:Nf,0:1];
u3t = np.fft.fft( u3, axis=0 );
u3tp = u3t[1:Nf,0:1];

# differential phase
R21p = np.copy(u2tp);
for i in range(Np):
   d = u1tp[i,0];
   if( d==0.0 ):
      R21p[i,0] = R21p[i,0] / 1E-15;
   else:
      R21p[i,0] = R21p[i,0] / d;
Dph21p = np.arctan2( -np.imag(R21p), np.real(R21p) );
Dph21p = np.unwrap( Dph21p, axis=0 );
```

```
R31p = np.copy(u3tp);
for i in range(Np):
    d = u1tp[i,0];
    if( d==0.0 ):
        R31p[i,0] = R31p[i,0] / 1E-15;
    else:
        R31p[i,0] = R31p[i,0] / d;
Dph31p = np.arctan2( -np.imag(R31p), np.real(R31p) );
Dph31p = np.unwrap( Dph31p, axis=0 );

fig=plt.figure();
ax=plt.subplot(1,1,1);
plt.plot(fp,Dph21p,'r-');
plt.plot(fp,Dph31p,'b-');
plt.xlabel('frequency (Hz)');
plt.ylabel('differential phase');
plt.show();
print('Caption: unwrapped differential phase (radians) for pairs 21 (red) and 31 (blue)');

# 2x2 equation for slowvess
sxpest = np.zeros((Np,1));
sypest = np.zeros((Np,1));
spest = np.zeros((Np,1));
vpest = np.zeros((Np,1));
thetapest = np.zeros((Np,1));
M = np.array( [ [Dx21, Dy21], [Dx31, Dy31]] );
RHS = np.zeros((2,1));
for i in range(Np):
    RHS[0,0] = Dph21p[i,0] / wp[i,0];
    RHS[1,0] = Dph31p[i,0] / wp[i,0];
    z = la.solve( M, RHS );
    sxpest[i,0] = z[0,0];
    sypest[i,0] = z[1,0];
    spest[i,0] = sqrt( sxpest[i,0]**2 + sypest[i,0]**2 );
    vpest[i,0] = 1.0 / spest[i,0];
    thetapest[i,0] = (180.0/pi) * atan2( sxpest[i,0], sypest[i,0] );

fig=plt.figure();
ax=plt.subplot(1,1,1);
plt.axis( [0.0, fp[Np-1,0], 0.0, 5.0] );
plt.plot(fp,vp,'k-');
plt.plot(fp,vpest,'r:');
plt.xlabel('frequency (Hz)');
plt.ylabel('velocity (s/kn)');
```

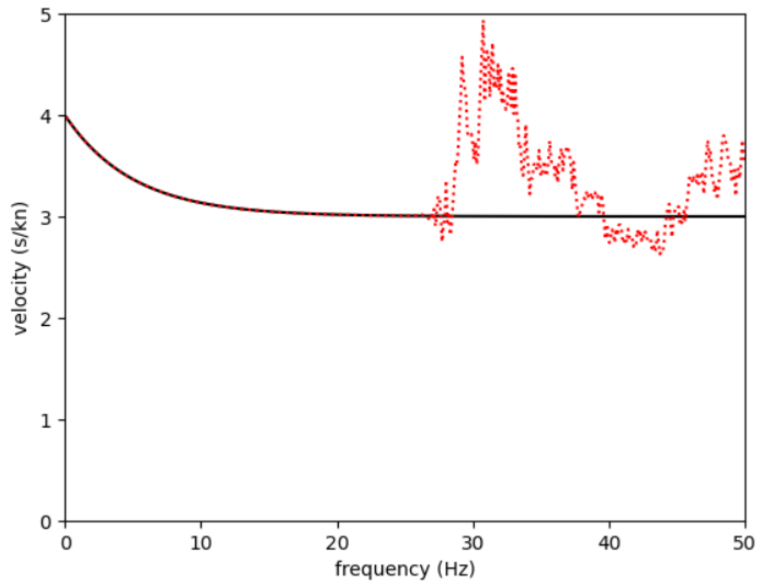
```
plt.show();
print('Caption: velocity (km/s), true (black) estimated (red)');

fig=plt.figure();
ax=plt.subplot(1,1,1);
plt.axis( [0.0, fp[Np-1,0], -180.0, 180.0] );
plt.plot(fp,theta*np.ones((Np,1)),'k-');
plt.plot(fp,thetapest,'r:');
plt.xlabel('frequency (Hz)');
plt.ylabel('azimuth (s/kn)');
plt.show();
print('Caption: azimuth (deg), true (black) estimated (red)');
```



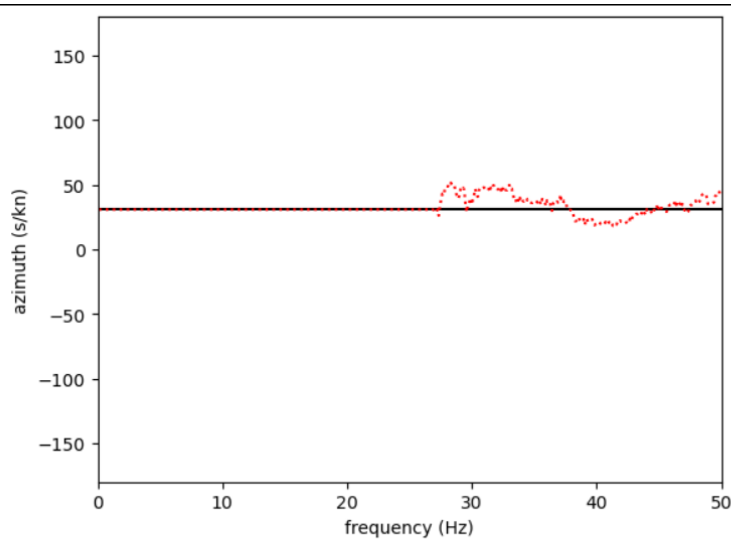
Caption: initial (black) and dispersed seismograms (red, blue, green)



Caption: unwrapped differential phase (radians) for pairs 21 (red) and 31 (blue)



Caption: velocity (km/s), true (black) estimated (red)



Caption: azimuth (deg), true (black) estimated (red)