Addendum to How to Make Synthetic Micro-Seismic Noise

by Bill Menke, March 7, 2026

Here is a proof that the Fourier transformed signal consists of a cone in the (k_x, k_y, ω) plane. In developing it, I discovered that a normalization factor was missing from the code. The corrected code is attached.

Although Aki's (1957) method is often described as applying to the temporal cross-correlation between two signals measured at different positions, it can also be formulated in terms of the 3D autocorrelation of a wavefield that varies with both space and time. I take this latter approach here/

In the time domain, an ergodic random signal $u(x, y, t)$, where (x, y) is position cand an t is time, has an autocorrelation $C(\Delta x, \Delta y, \Delta t)$ where $(\Delta x, \Delta y)$ is spatial lag and Δt is temporal lag. It is defined as

$$C(\Delta x, \Delta y, \Delta t) = \lim_{V \to \infty} \frac{1}{V} \iiint u_V(x, y, t) u_V(x + \Delta x, y + \Delta y, t + \Delta t)\, dx\, dy\, dt \tag{1}$$

Here, $u_V(x, y, t)$ is windowed version of $u(x, y, t)$, using a boxcar window function of volume V in the space of (x, y, t). A well-known property of the autocorrelation $C(\Delta x, \Delta y, \Delta t)$ is that its Fourier transform is the power spectral density $S(k_x, k_y, \omega)$, and conversely that the inverse Fourier transform of the power spectral density is the autocorrelation. Using this second property, we write:

$$C(\Delta x, \Delta y, \omega) = \left(\frac{1}{2\pi}\right)^2 \iint S(k_x, k_y, \omega) e^{ik_x \Delta x + ik_y \Delta y}\, dk_x dk_y \tag{2}$$

Here, (k_x, k_y) is spatial wavenumber and ω is angular frequency. We use the convention of identifying transformed variables by their arguments, only; that is $a(\omega)$ is Fourier transform of $a(t)$. Now suppose radially symmetry, so that $C(\Delta x, \Delta y, \omega) = C(r, \omega)$ with $r^2 \equiv (\Delta x)^2 + (\Delta y)^2$ and $S(k_x, k_y, \omega) = S(k_r, \omega)$ with $k_r^2 \equiv k_x^2 + k_y^2$. Then we can write Eqn. (6) as its equivalent Hankel transform:

$$C(r, \omega) = \frac{1}{2\pi} \iint S(k_r, \omega) J_o(k_r r) k_r dk_r \tag{3}$$

Evidentially, the choice

$$S(k_r, \omega) = \frac{2\pi}{k_r} \delta\left(k_r - \frac{\omega}{c}\right) \tag{4}$$

leads to an Aki (1957) type autocorrelation:

$$C(r, \omega) = \frac{1}{2\pi} \iint \frac{2\pi}{k_r} \delta\left(k_r - \frac{\omega}{c}\right) J_o(k_r r) k_r dk_r = J_o\left(\frac{\omega r}{c}\right) \tag{5}$$

At constant ω, the Eqn. (4) implies that $S(k_r, \omega)$ is a circle in the (k_x, k_y) plane. As the total energy in the circle is proportional to its circumference $2\pi k_r$, the factor of k_r^{-1} implies that the total energy in the wavefield is independent of frequency ω. Eqn. (4) only places a condition on the amplitude of $u(k_x, k_y, \omega)$, allowing its phase to be randomly assigned. According to Eqn. (4), the amplitude is constant along the perimeter of the circle. However, this condition seems physically implausible, as it implies that waves from all directions have the same exact amplitude, whereas some random variation must occur in nature. We allow for such variation by randomly drawing both the real and imaginary parts of $u(k_x, k_y, \omega)$ from a Normal p.d.f. with zero mean and variance that is independent of (k_x, k_y). Finally, we note that introducing a "source" or "instrument" factor $A(\omega) > 0$ to Eqn. (4) leads to

$$S(k_r, \omega) = \frac{A(\omega)}{2\pi k_r} \delta\left(k_r - \frac{\omega}{c}\right) \quad \text{and} \quad C(r, \omega) = A(\omega)\, J_o\left(\frac{\omega r}{c}\right) \tag{6}$$

Thus, $C(r, \omega)$ varies as a Bessel function in r, but has a more complicated behavior in ω. Nevertheless, $C(r, \omega)$ is still oscillatory and the positions of its zeros are unaffected by the choice of $A(\omega)$.

I have patched the `synmicroseisms()` code to match Eqn. (6). The old version omitted the factor of k_r^{-1}. However, the change has very little effect on the result, except possibly at the lowest frequencies (Figs. 1 and 2).

Reference

Aki, K. (1957). Space and time spectra of stationary stochastic waves, with special reference to microtremors, Bulletin of the Earthquake Research Inst Institute 35, 415–457.

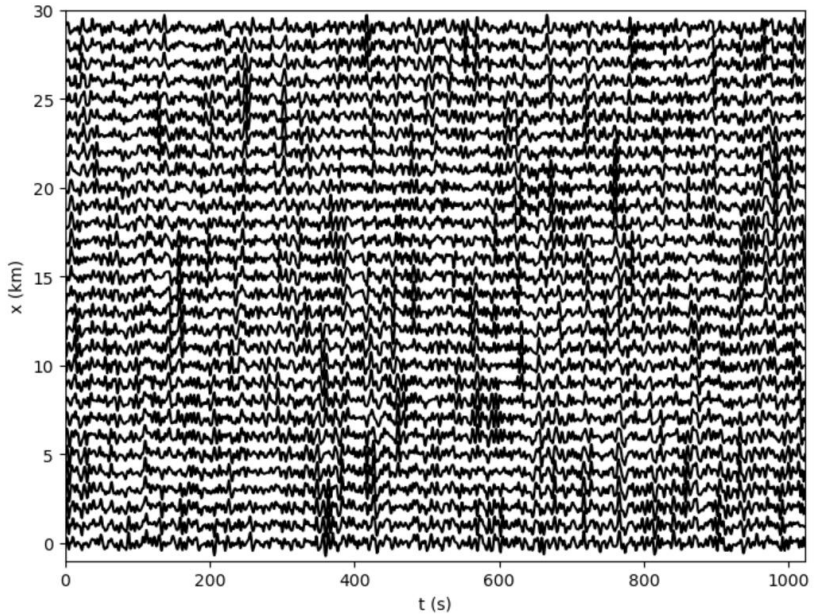

Fig. 1. Microseismic wavefield for stations with variable x-position and fixed y-position.

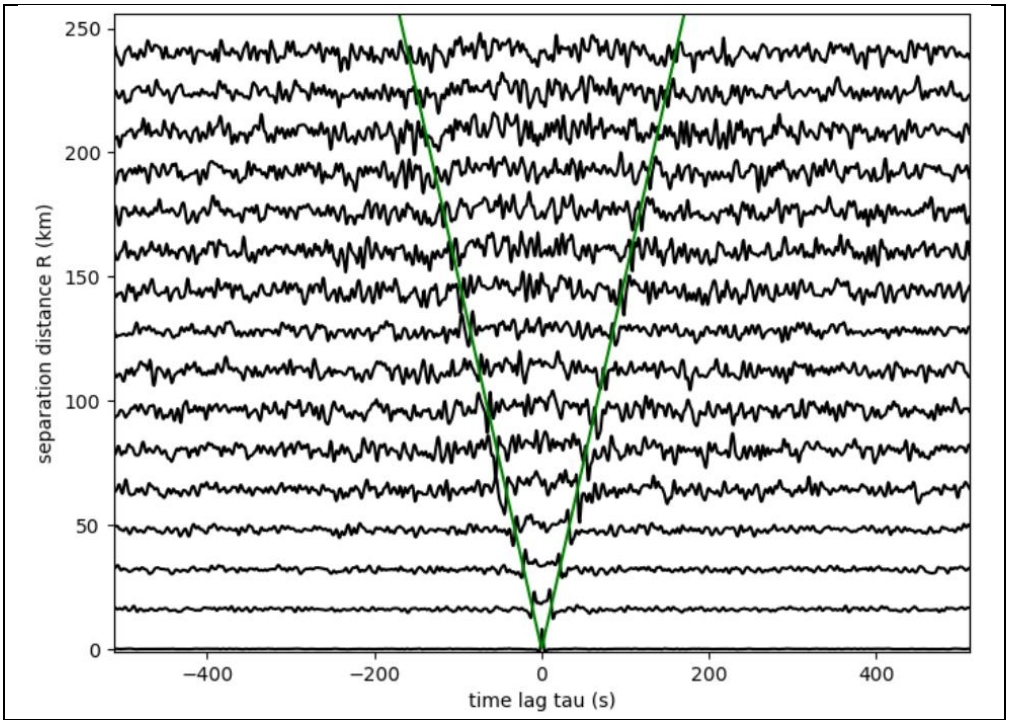

Fig. 2. Ensemble-averaged correlograms (black) for pairs of stations separated by distance R with phase velocity curve (green) superimposed.

```
import numpy as np;
import matplotlib.pyplot as plt;
from math import floor, pi, exp, sqrt;
import scipy.signal as sg;

def synmicroseisms( Dx, Nx, Dy, Ny, Dt, Nt, c, A, ixiy ):
    # by Bill Menke, December 2025; modified March 2026
    # makes synthetic seismograms of spatially-correlated microseismic noise
    # Dx, Nx:  grid spacing and number of gridpoints in position x
    # Dy, Ny:  grid spacing and number of gridpoints in position y
    #          notes: Dx=Dy and Nx=Ny are bes
    #                 Nx, Ny, Nt must be even and powers of 2 are best
    # Dt, Nt:  grid spacing and number of gridpoints in position t
    # c: Nw x 1 array of phase velocities, where Nw=Nt/2+1
    #    corresponding frequencies are from frequency 0 to 1/(2 Dt) Hz
    # A: Nw x 1 array of amplitudes, where Nw=Nt/2+1
    #    corresponding frequencies are from frequency 0 to 1/(2 Dt) Hz
    # ixiy: Nxy x 2 integer array of (ix,iy) grid nodes of seismograms
    #    that are to be returned; ix in range 0 to Nx-1, iy in range
    #    0 to Ny-1, Nxy can be any length

    Nxy, i = np.shape(ixiy);

    # x-axis and kx-axis
    Nxo2 = int(Nx/2);
    Nkx=Nxo2+1;  # number of non-negative frequencies
    kxmax = 2.0*pi/(2.0*Dx); # Nyquist frequency
    Dkx = kxmax/(Nx/2);  # frequency increment
    kxp = np.zeros((Nkx,1));
    kxp[0:Nkx,0] = np.linspace(0.0,kxmax,Nkx);
    kx = np.zeros((Nx,1));
    kx[0:Nx,0]=Dkx * np.concatenate(
```

```
    (np.linspace(0,Nxo2,Nkx), np.linspace(-Nxo2+1,-1,Nxo2-1) ), axis=0 );

# y-axis and ky-axis
Nyo2 = int(Ny/2);
Nky=Nyo2+1;  # number of non-negative frequencies
kymax = 2.0*pi/(2.0*Dy); # Nyquist frequency
Dky = kymax/(Ny/2);  # frequency increment
kyp = np.zeros((Nky,1));
kyp[0:Nky,0] = np.linspace(0.0,kymax,Nky);
ky = np.zeros((Ny,1));
ky[0:Ny,0]=Dky * np.concatenate(
    (np.linspace(0,Nyo2,Nky), np.linspace(-Nyo2+1,-1,Nyo2-1) ), axis=0 );

# t-axis and w-axis
tmax=Dt*(Nt-1);
t = np.zeros((Nt,1));
t[0:Nt,0] = Dt * np.linspace(0,Nt-1,Nt);
Nto2 = int(Nt/2);
Nw=Nto2+1;  # number of non-negative frequencies
fmax = 1.0/(2.0*Dt); # Nyquist frequency
wmax = 2.0*pi*fmax;
Dw = wmax/(Nt/2);  # frequency increment
wp = np.zeros((Nw,1));
wp[0:Nw,0] = np.linspace(0.0,wmax,Nw);
w = np.zeros((Nt,1));
w[0:Nt,0]=Dw * np.concatenate(
    (np.linspace(0,Nto2,Nw), np.linspace(-Nto2+1,-1,Nto2-1) ), axis=0 );

# fourier transform of selected seismograms
dtxy = np.zeros( (Nw,Nxy), dtype=complex );

# angular information
Nth = 2*(Nx+Ny);
thmax = 2.0*pi;
Dth = thmax/(Nth-1);
th = np.zeros((Nth,1));
th[0:Nth,0] = np.linspace(0.0,thmax,Nth);
sth = np.sin(th);
cth = np.cos(th);

# loop over frequencies
for iw in range(1,Nw):
    # Fourier transform at all wavenumbers and constant fquency
    dt = np.zeros((Nx,Ny),dtype=complex);
    wi = wp[iw,0];
    ci = c[iw,0];
    Ai = A[iw,0];
    # random phases
    nseR = np.random.normal( loc=0.0, scale=1.0, size=(Nth,1) );
    nseI = np.random.normal( loc=0.0, scale=1.0, size=(Nth,1) );
    # c = w/k so k = w/c
    kr0 = wi/ci;
    kx0 = kr0*sth;
    ky0 = kr0*cth;
    # populate circle in kn-ky plane with random numbers
    for ith in range(Nth):
        ikx0 = int(kx0[ith,0]/Dkx);
        iky0 = int(ky0[ith,0]/Dky);
        if( ikx0 < 0 ):
            ikx0 = Nx+ikx0;
        if( ikx0 < 0 ):
             ikx0 = 0;
        elif( ikx0 > (Nx-1) ):
```

```
                ikx0 = (Nx-1);
            if( iky0 < 0 ):
                iky0 = Ny+iky0;
            if( iky0 < 0 ):
                iky0=0;
            elif( iky0 > (Ny-1) ):
                ikx0 = (Ny-1);
            dtR = nseR[ith,0];
            dtI = nseR[ith,0];
            dt[ikx0, iky0] = complex( dtR, dtI );
        sqrtpower = sqrt(np.sum(np.power(np.abs(dt),2))); # added 03/07/2026
        dt = Ai*dt/sqrtpower;                             # added 03/07/2026
        d = np.fft.ifft2(dt); # inverse trasnform (kx,ky) to (x,y)
        # grab desired (x,y) positions
        for ixy in range(Nxy):
            dtxy[ iw, ixy ] = d[ ixiy[ixy,0], ixiy[ixy,1] ];

    # Fourier transform w to t
    d = np.zeros((Nt,Nxy));
    for ixy in range(Nxy):
        ut = dtxy[0:Nw, ixy:ixy+1];
        u = np.fft.irfft(ut,axis=0);
        d[0:Nt,ixy:ixy+1] = u;

    return(d);

# grid size
Nx = 512;
Dx = 1.0;
Ny = 512;
Dy = 1.0;
Nt = 1024;
Dt = 1.0;

# time/frequency axis (needed to compute phase velocity)
tmax=Dt*(Nt-1);
t = np.zeros((Nt,1));
t[0:Nt,0] = Dt * np.linspace(0,Nt-1,Nt);
Nto2 = int(Nt/2);
Nw=Nto2+1;  # number of non-negative frequencies
fmax = 1.0/(2.0*Dt); # Nyquist frequency
wmax = 2.0*pi*fmax;
Dw = wmax/(Nt/2);  # frequency increment
wp = np.zeros((Nw,1));
wp[0:Nw,0] = np.linspace(0.0,wmax,Nw);

# compute phase velocity c(w) and ampolitude A(w)
w0 = wmax/10.0;
wc = wmax/5.0;
sw = wc;
sw2 = sw**2;
clow=2.0;
chigh=1.5;
c = np.zeros((Nw,1));
A = np.zeros((Nw,1));
for iw in range(1,Nw):
    wi = wp[iw,0];
    ci = clow - (clow-chigh)*wi/w0;
    if( ci<chigh ):
         ci = chigh;
    c[iw,0] = ci;
    A[iw,0] = exp(-0.5*(wi-wc)**2/sw2 );
```

```
# select 30 stations with variable x on a single y-level
Nxy=30;
ixiy = np.zeros((Nxy,2),dtype=int);
for i in range(Nxy):
    ixiy[i,0] = i;
    ixiy[i,1] = int(Ny/2+1);

# compute wavefield
d=synmicroseisms( Dx, Nx, Dy, Ny, Dt, Nt, c, A, ixiy );

# plot wavefield
dmax = np.max(np.abs(d));
plt.figure(figsize=(8,6));
plt.subplot(1,1,1);
plt.xlabel("t (s)");
plt.ylabel("x (km)");
plt.axis( [0, tmax, -1, Nxy] );
for ixy in range(Nxy):
    plt.plot( t,ixy+d[0:Nt,ixy:ixy+1]/dmax, 'k-' );
plt.show();
print("Fig. 1. Microseismic wavefield for stations with variable x-position");
print(" and fixed y-position");

# select Nx stations with variable x on a single y-level
Nxy=Nx;
ixiy = np.zeros((Nxy,2),dtype=int);
for i in range(Nxy):
    ixiy[i,0] = i;
    ixiy[i,1] = int(Ny/2+1);

# compute wavefield
d=synmicroseisms( Dx, Nx, Dy, Ny, Dt, Nt, c, A, ixiy );

# compute ensemble-averaged correlogram
# note that this is only maginally enough data to get a good correlogram
Nlag=int(Nx/2);
xc = np.zeros((Nlag,Nt));
for ilag in range(Nlag):
    for ixy in range(Nxy):
        d1 = d[0:Nt,ixy];
        j = (ixy+ilag)%Nx;
        d2 = d[0:Nt,j];
        corr = sg.correlate( d1, d2, mode='same', method='fft' );
        xc[ilag,0:Nt] = xc[ilag,0:Nt] + corr;

# plot correlogram
xcmax = np.max(xc);
plt.figure(figsize=(8,6));
plt.subplot(1,1,1);
NPLOT=Nlag;
NSKIP = 16;
plt.axis( [-tmax/2, tmax/2, -1, NPLOT] );
plt.xlabel("time lag tau (s)");
plt.ylabel("separation distance R (km)");
for ix in range(0,NPLOT,NSKIP):
    xcmax = np.max(xc[ix,0:Nt]);
    plt.plot( t-tmax/2,ix+0.5*NSKIP*xc[ix,0:Nt]/xcmax, 'k-' );
plt.plot(t-tmax/2,np.abs(chigh*(t-tmax/2)),"g-");
plt.show();
print("Fig. 2. Ensemble-averaged correlagrams (black) for pairs of stations");
print("separated by distance R with phase velocity curve (green) superimposed");
```