How to Make Synthetic Micro-Seismic Noise
by Bill Menke, November 30, 2025

Synthetic noise is often used during the testing of seismic analysis software. Here, I demonstrate how a micro-seismic noise field with a realistic spatial correlation structure can be simulated.

I adopt Aki's (1957) model in which micro-seismic noise is composed of surface waves arriving from all directions. I consider here a vertical-component noise field $u(x, y, t)$ composed of dispersive Rayleigh waves with phase velocity $v(\omega)$. The noise field has 3-D Fourier transform $\tilde{u}(k_x, k_y, \omega)$. I construct the noise field on a discrete grid that is $N_x \times N_y \times N_t$ is size. As the wave field is real, its Fourier transform is on a discrete $N_x \times N_y \times \left((N_t/2) + 1\right)$ grid.

At fixed frequency ω_0 the Fourier transform $\tilde{u}(k_x, k_y, \omega_0)$ is non-zero only on a circle in the (k_x, k_y) plane. This circle is centered at the origin and has diameter $k_r = \omega_0 / v(\omega_0)$. It represents waves propagating in all directions at fixed phase velocity $v(\omega_0)$ (Fig. 1). I start with a grid that is everywhere zero and then sweep though N_θ angles in the range $0 \leq \theta \leq 2\pi$, finding the grid node closest to $(k_x, k_y) = (k_r \sin\theta\,, k_r \sin\theta)$ and setting it to a value $A(\omega)(r_R + i r_I)$. Here, $A(\omega_0)$ is a frequency-dependent amplitude and (r_R, r_I) are uncorrelated Normally distributed random numbers with zero mean and unit variance.

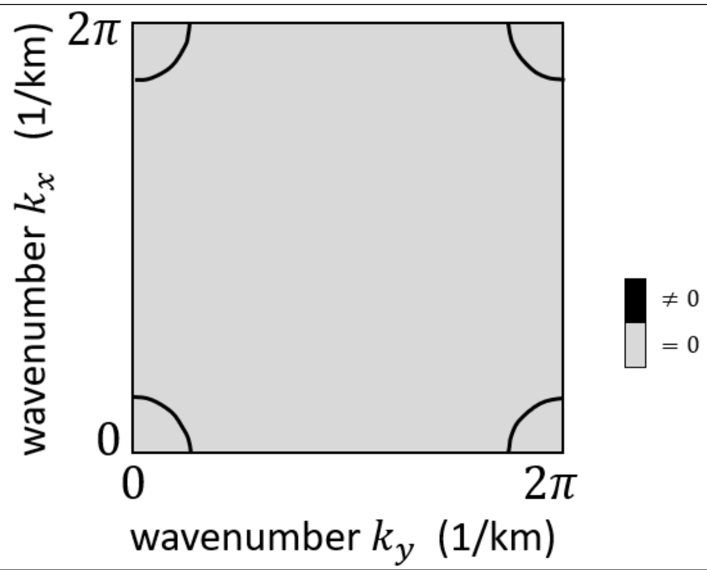

Fig. 1. The function $\tilde{u}(k_x, k_y, \omega_0)$ for variable wavenumber $(k_x, k_y,)$ and for fixed frequency ω_0. The (k_x, k_y) plane has $N_x \times N_y = 512 \times 512$ grid points. The values (k_x, k_y) for which $\tilde{u} \neq 0$ form a circle in the plane (black curves). It appears as four quarter-circles because of the peculiar ordering of values by the fast-Fourier transform algorithm. A total of $N_\theta = 360$ values of (k_x, k_y) are set for each ω_0.

The shape of $A(\omega)$ should be guided by the amplitude spectral density of the microseisms being modeled. I use a Gaussian $A(\omega)$ with center frequency ω_c and standard deviation σ_ω.

This process if repeated for every frequency in the grid. An inverse Fourier transform then takes $\tilde{u}(k_x, k_y, \omega)$ to $u(x, y, t)$.   For fixed (x_i, y_i), $u(x_i, y_i, t)$ is one realization of micro-seismic noise (Fig. 2).

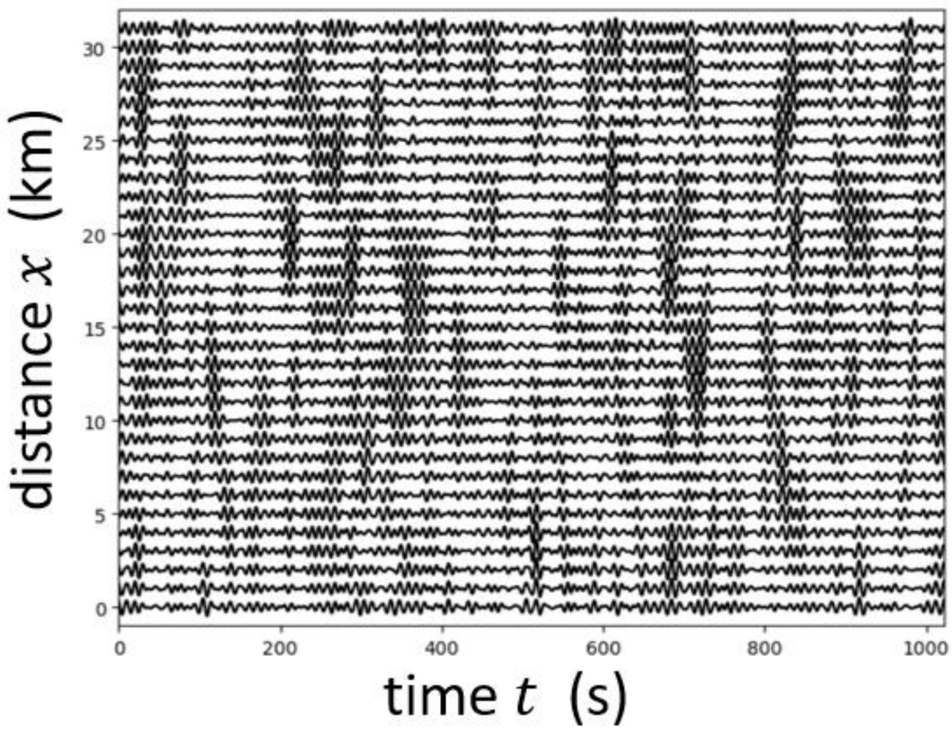

Fig. 2. Synthetic micro-seismic noise $u(x, y_i, t)$ for fixed y_i. The noise has center frequency $\omega_c/2\pi = 0.1$ Hz and standard deviation $\sigma_\omega/2\pi = 0.025$ Hz. The wavefield at neighboring xs is highly correlated.

In cases where seismograms at only selected (x_i, y_i) are required, the full transform $\tilde{u}(k_x, k_y, \omega)$ need not be calculated. Instead, the wavenumber sheets $\tilde{u}(k_x, k_y, \omega_0)$ are calculated for each frequency ω_0, separately transformed to $\tilde{u}(x, y, \omega_0)$, and the desired (x_i, y_i) values are retained. A final set of Fourier transforms then converts each $\tilde{u}(x_i, y_i, \omega)$ to $u(x_i, y_i, t)$.

Seismogram pairs collected at points (x_i, y_i) and (x_j, y_j) separated by a distance $d_{ij} = \left((x_j - x_i)^2 + (y_j - y_i)^2\right)^{½}$ are correlated. Their behavior is investigated by computing the correlogram $c_{ij}(t) \equiv u(x_i, y_i, t) \star u(x_j, y_j, t)$, where $\star$ denotes cross-correlation. This correlogram is then ensemble-averaged over all points with the same separation distance d_k yielding $c_k(t)$. In the limit of an indefinitely large ensemble, this correlogram is symmetric in time; that is, $c_k(t) = c_k(-t)$. In the case of a finite ensemble, this symmetry is introduced by defining a symmetrized correlogram $C(t, d_k) \equiv (c_k(t) + c_k(-t))/2$. As expected, this correlogram contains two Rayleigh wave trains propagating in opposed directions (Fig. 3).

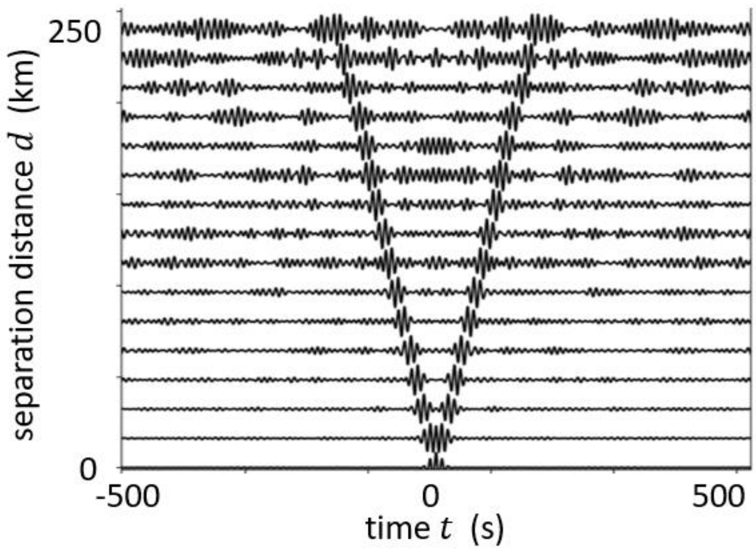

Fig. 3. Symmetrized correlogram $C(t,d)$, as a function of separation distance d and time t, for the seismograms in Fig. 2. Two Rayleigh wave trains, one forward in time and one backward in time, are evident.

As $C(t,d)$ is symmetric in time, its Fourier transform (the ensemble-averaged cross-spectra) $\tilde{C}(\omega,d)$ is purely real. Aki (1957) showed that

$$\frac{\tilde{C}(\omega,d)}{\tilde{C}(\omega,0)} = J_0\left(\frac{\omega d}{v(\omega)}\right)$$

Where $J_0(.)$ is the zeroth order Bessel function of the first kind. This behavior is reproduced by Fourier transforming $C(t,d)$ to $\tilde{C}(\omega,d)$ (Fig. 4).

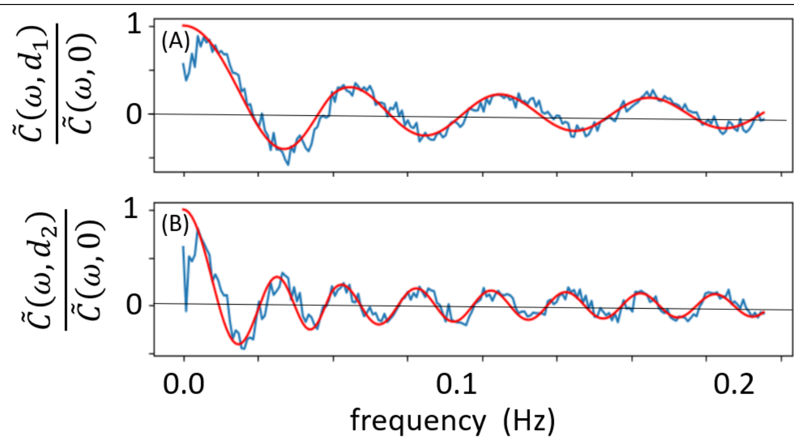

Fig. 4. Ensemble-averaged cross-spectra $\tilde{C}(\omega,d)$ (blue) for (A) $d = 30$ km and (B) $d = 60$ km. The predictions of the Aki (1957) model (red) are in good agreement in the $0.01 - 0.20$ Hz frequency interval in which the simulation has significant spectral power. The noise has center frequency $\omega_c/2\pi = 0.1$ Hz and standard deviation $\sigma_\omega/2\pi = 0.1$ Hz.

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.

Python code

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

def synmicroseisms( Dx, Nx, Dy, Ny, Dt, Nt, c, A, ixiy ):
    # by Bill Menke, December 2025
    # 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=Ai, size=(Nth,1) );
    nseI = np.random.normal( loc=0.0, scale=Ai, 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) ):
                 iky0 = (Ny-1);
            dtR = nseR[ith,0];
            dtI = nseR[ith,0];
            dt[ikx0, iky0] = complex( dtR, dtI );
        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");
```