Three Station Waveform Method of Computing Phase Velocity and its Confidence
Bill Menke, November 12, 2025

The phase method does not require the phase velocity function $v(f)$ to vary smoothly with frequency, despite it being smooth for most Earth structures. Smoothness constitutes prior information that, when incorporated into the estimation process, improves the confidence of the phase velocity and azimuth estimates. Prior information can be introduced by parameterizing differential travel time in such a way that it is constrained to be smoothly varying. Thus, we write travel time as $\tau(\omega, \mathbf{m})$, where $\mathbf{m}$ is a vector of model parameters and to estimate $\mathbf{m}$ using standard inverse theory by minimizing the error $E(\mathbf{m})$ of the fit between pairs of seismic waveforms. This process couples observations at different frequencies, allowing the prior information of smoothness to be enforced. Remarkably, and as we will show below, the derivatives needed to minimize $E(\mathbf{m})$ with respect to $\mathbf{m}$ can be calculated semi-analytically.

Consider a real phase delay filter $f(t)$ with Fourier transform $\tilde{f}(\omega) = \exp\{-i\omega\tau\ (\omega, \mathbf{m})\}$, where the differential travel time τ depends on initially unknown parameters $\mathbf{m}$, of length N_m. The objective is to estimate $\mathbf{m}$ using standard inverse theory. Note that $\tilde{f}_R(\omega) = \cos\big(-\omega\tau(\omega, \mathbf{m})\big)$ and $\tilde{f}_I(\omega) = \sin\big(-\omega\tau(\omega, \mathbf{m})\big)$. The frequency-domain filter has unit amplitude; that is, $\left|\tilde{f}(\omega)\right|^2 = \tilde{f}(\omega)\tilde{f}^*(\omega) = 1$. For $f(t)$ to be real, $\tilde{f}(-\omega) = \tilde{f}^*(\omega)$ so $\tau(-\omega, \mathbf{m}) = \tau(\omega, \mathbf{m})$.

The model that we consider is that $u_2(t)$ is a dispersively-delayed version of $u_1(t)$; that is:

$$u_2(t) = u_1(t) * f(t) \tag{40}$$

where * signifies convolution. The L_2 prediction error is defined as:

$$E \equiv \int [u_2(t) - u_1(t) * f(t)]^2\ dt = \int [u_1(t) * f(t)]^2\ dt + \int u_2^2(t)\ dt - 2\int [u_1(t) * f(t)]\, u_2(t)\, dt \tag{41}$$

Parseval's theorem for real time series $a(t)$ and $b(t)$ states:

$$\int_{-\infty}^{+\infty} a(t)b(t)\ dt = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \tilde{a}(\omega)\tilde{b}^*(\omega)\ d\omega \tag{42}$$

A real function, say $h(t)$, has a Fourier transform with symmetry $\tilde{h}(-\omega) = \tilde{h}^*(\omega)$. Consequently:

$$\int_{-\infty}^{+\infty} h(\omega)\ d\omega = \int_{0}^{+\infty} h(\omega)\, d\omega + \int_{-\infty}^{0} h(\omega)\, d\omega = \int_{0}^{+\infty} h(\omega)\, d\omega + \int_{+\infty}^{0} h(-\omega)\, d(-\omega) =$$

$$= \int_{0}^{+\infty} h(\omega)\, d\omega + \int_{0}^{+\infty} h^*(\omega)\, d\omega = 2\int_{0}^{+\infty} \mathrm{Re}\, h(\omega)\, d\omega$$

(43)

In a discrete implementation in which the integrals are approximated by Reiman sums, the zero-frequency value is being counted once in the $(-\infty, +\infty)$ summation but twice in the $(-\infty, 0)$ plus $(0, +\infty)$ summations. Thus, $h(\omega = 0)$ must be subtracted from the latter to match the former within the computer code that implement the algorithm.

Applying these results:

$$\int_{-\infty}^{+\infty} [u_1(t) * f(t)]^2\, dt = \frac{1}{2\pi}\int_{-\infty}^{+\infty} u_1(\omega)\tilde{A}^*(\omega)\,\tilde{f}(\omega)\tilde{f}^*(\omega)d\omega =$$

$$\frac{1}{2\pi}\int_{-\infty}^{+\infty} |\tilde{u}_1(\omega)|^2\, d\omega = \frac{1}{\pi}\int_{0}^{+\infty} |\tilde{u}_1(\omega)|^2\, d\omega$$

$$\int_{-\infty}^{+\infty} [u_2(t)]^2\, dt = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \tilde{u}_2(\omega)\tilde{u}_2^*(\omega)\, d\omega =$$

$$\frac{1}{2\pi}\int_{-\infty}^{+\infty} |\tilde{u}_2(\omega)|^2\, d\omega = \frac{1}{\pi}\int_{0}^{+\infty} |\tilde{u}_2(\omega)|^2\, d\omega$$

$$\int_{-\infty}^{+\infty} [u_1(t) * f(t)]\, u_2(t)\, dt = \frac{1}{\pi}\int_{0}^{+\infty} \mathrm{Re}\left[\tilde{u}_1(\omega)\tilde{u}_2^*(\omega)\tilde{f}(\omega)\right] d\omega$$

(44)

The product of three complex numbers has real and imaginary parts

$$(a + ib)(c + id)(e + if) = (ace - bde - adf - bcf) + i(ade + bce + acf - bdf)$$

(45)

so

$$\mathrm{Re}\left[\tilde{u}_1(\omega)\tilde{u}_2^*\tilde{f}(\omega)\right] = \tilde{u}_{1R}\tilde{u}_{2R}^*\tilde{f}_R - \tilde{u}_{1I}\tilde{u}_{2I}^*\tilde{f}_R - \tilde{u}_{R1}\tilde{u}_{2I}^*\tilde{f}_I - \tilde{u}_{1I}\tilde{u}_{2R}^*\tilde{f}_I = b_R\tilde{f}_R + b_I\tilde{f}_I$$

(46)

Putting all this together, we can write the error as

$$E = \frac{1}{\pi}\int_0^{\infty} H\, d\omega \quad \text{with} \quad H \equiv b_0 + b_R\tilde{f}_R + b_I\tilde{f}_I$$

with

$$b_0 \equiv |\tilde{u}_1(\omega)|^2 + |\tilde{u}_2(\omega)|^2$$

$$b_R \equiv -2(\tilde{u}_{1R}\tilde{u}_{2R} + \tilde{u}_{1I}\tilde{u}_{2I}) \quad \text{and} \quad b_I \equiv -2(\tilde{u}_{1R}\tilde{u}_{2I} - \tilde{u}_{1I}\tilde{u}_{2R})$$

(47)

We now differentiate E with respect to m_i. As the bs are not functions of m_i,

$$\frac{\partial E}{\partial m_i} = \frac{1}{\pi}\int_0^{+\infty} \frac{\partial H}{\partial m_i} d\omega \quad \text{with} \quad \frac{\partial H}{\partial m_i} = b_R\frac{\partial \tilde{f}_R}{\partial m_i} + b_I\frac{\partial \tilde{f}_I}{\partial m_i}$$

(48)

Similarly, the second derivative is:

$$\frac{\partial E^2}{\partial m_i \partial m_j} = \frac{1}{\pi}\int_0^{+\infty} \frac{\partial^2 H}{\partial m_i \partial m_j} d\omega \quad \text{with} \quad \frac{\partial^2 H}{\partial m_i \partial m_j} = b_R\frac{\partial^2 \tilde{f}_R}{\partial m_i \partial m_j} + b_I\frac{\partial \tilde{f}_I}{\partial m_i \partial m_j}$$

(49)

By applying the chain rule, we find that the first derivatives are:

$$\frac{\partial \tilde{f}_R}{\partial m_i} = \frac{\partial}{\partial m_i}\cos\big(-\omega T(\omega, \mathbf{m})\big) = \omega \sin(-\omega T)\frac{\partial \tau}{\partial m_i} = \omega \tilde{f}_I(\omega)\frac{\partial \tau}{\partial m_i}$$

$$\frac{\partial \tilde{f}_I}{\partial m_i} = \frac{\partial}{\partial m_i}\sin\big(-\omega T(\omega, \mathbf{m})\big) = -\omega \cos(-\omega T)\frac{\partial \tau}{\partial m_i} = -\omega \tilde{f}_R(\omega)\frac{\partial \tau}{\partial m_i}$$

(50)

and that the second derivatives are:

$$\frac{\partial^2 \tilde{f}_R}{\partial m_i \partial m_j} = \omega\frac{\partial}{\partial m_j}\left[\tilde{f}_I(\omega)\frac{\partial \tau}{\partial m_i}\right] = \omega\frac{\partial \tilde{f}_I(\omega)}{\partial m_j}\frac{\partial \tau}{\partial m_i} + \omega \tilde{f}_I(\omega)\frac{\partial^2 \tau}{\partial m_i \partial m_j}$$

$$\frac{\partial^2 \tilde{f}_I}{\partial m_i \partial m_j} = -\omega\frac{\partial}{\partial m_j}\left[\tilde{f}_R(\omega)\frac{\partial \tau}{\partial m_i}\right] = -\omega\frac{\partial \tilde{f}_R(\omega)}{\partial m_j}\frac{\partial \tau}{\partial m_i} - \omega \tilde{f}_R(\omega)\frac{\partial^2 \tau}{\partial m_i \partial m_j}$$

(51)

Computational efficiency can be gained by limiting the frequency-integrals to the $(0, \omega_2)$ interval, where ω_2 is the largest significant frequency in the data. In the special case of the linear model $\tau(\omega_k, \mathbf{m}) = \sum_n G_{kn} m_n$ (where $\mathbf{G}$ is an $N_\omega \times N_m$ matrix):

$$\left.\frac{\partial \tau}{\partial m_i}\right|_{\omega_k} = G_{ki} \quad \text{and} \quad \left.\frac{\partial^2 \tau}{\partial m_i \partial m_j}\right|_{\omega_k} = 0$$

(52)

Eqns. (2)-(11) have been verified by numerically calculation. Only the first derivatives (Eqns. 8 and 10) are needed for a steepest-descent inversion for **m** (Menke, Geophysical Data Analysis, 2024, Sec. 11.9). For a Newton's method inversion (Menke, Geophysical Data Analysis, 2024, Eqn. 11.76), the second derivatives (Eqns. 9 and 11) are needed as well.

For three stations, say (i, j, k), the total waveform error can be defined as:

$$E_T \equiv \frac{1}{\pi}\int_0^{+\infty} \begin{bmatrix}\tilde{e}^{(ij)} \\ \tilde{e}^{(ik)}\end{bmatrix}^H \tilde{\mathbf{C}}^{inv} \begin{bmatrix}\tilde{e}^{(ij)} \\ \tilde{e}^{(ik)}\end{bmatrix} d\omega \quad \text{with} \quad \tilde{\mathbf{C}}^{inv} \equiv \begin{bmatrix}\tilde{C}_{ij}^{inv}(\omega) & \tilde{C}_{ijk}^{inv}(\omega) \\ \tilde{C}_{ijk}^{inv*}(\omega) & \tilde{C}_{ik}^{inv}(\omega)\end{bmatrix}$$

$$\text{and} \quad \begin{bmatrix}\tilde{e}^{(ij)} \\ \tilde{e}^{(ik)}\end{bmatrix} \equiv \begin{bmatrix}\tilde{u}_j - \tilde{u}_i \tilde{f}^{(ij)} \\ \tilde{u}_k - \tilde{u}_i \tilde{f}^{(ik)}\end{bmatrix} \quad \text{and} \quad \begin{bmatrix}\tilde{f}^{(ij)} \\ \tilde{f}^{(ik)}\end{bmatrix} = \begin{bmatrix}\exp\{-i\omega\tau^{(ij)}(\mathbf{m}^{(ij)})\} \\ \exp\{-i\omega\tau^{(ik)}(\mathbf{m}^{(ik)})\}\end{bmatrix}$$

(53)

Here, *H* is the Hermitian transpose. For reasons that will be apparent, below, we introduce the quantity:

$$\tilde{f}^{(ijk)} \equiv \tilde{f}^{(ij)*}\tilde{f}^{(ik)} = \exp\{-i\omega\tau^{(ijk)}\} \text{ with } \tau^{(ijk)} \equiv \tau^{(ik)}(\mathbf{m}^{(ik)}) - \tau^{(ij)}(\mathbf{m}^{(ij)})$$

(54)

The matrix $\mathbf{C}^{inv}$acts to weight the errors, in the same way that, in generalized least squares, the inverse of covariance matrix $\mathbf{C}_e$ acts to weight individual errors $\mathbf{e}$ in the error formula $E = \mathbf{e}^T \mathbf{C}_e^{-1} \mathbf{e}$. The choice $\mathbf{C}^{inv} = \mathbf{I}$, which implies uncorrelated error, is common in the seismological literature and allows the two sets of model parameters, $\mathbf{m}^{(ij)}$ and $\mathbf{m}^{(ik)}$, to be inverted separately. Nevertheless, any diagonal $\mathbf{C}^{inv}$ is deficient in the statistical sense because it does not account for correlation of error between seismograms, which is known through Aki's (1957) work to be significant. We consider the $\mathbf{C}^{inv} \neq \mathbf{I}$ case here.

Following the general practice in generalized least squares, the weighting matrix $\tilde{\mathbf{C}}^{inv}$ is the inverse of the covariance matrix $\mathbf{C} \equiv \text{cov}(\tilde{e}_{ij}, \tilde{e}_{ik})$. However, we make the simplifying assumption that, at a given frequency, the $\tilde{e}^{(ij)}$ and $\tilde{e}^{(ik)}$ are correlated, as in Aki's (1957) noise model, but that errors at different frequencies are uncorrelated. This assumption allows us to consider $\mathbf{C}$ and $\tilde{\mathbf{C}}^{inv}$ to be 2×2 matrices that vary with frequency.

Following a procedure similar to the one we used in the two-station case, we rewrite the integrand in Eqn. (53) to bring out the $\tilde{f}$s, which are the only functions dependent upon the model parameters:

$$E_T = \frac{1}{\pi}\int_0^{+\infty} H\, d\omega$$

$$H \equiv c_0 + c_{ijR}\tilde{f}_R^{(ij)} + c_{ijR}\tilde{f}_I^{(ij)} + a_{ikR}\tilde{f}_R^{(ik)} + c_{ijR}\tilde{f}_I^{(ik)} + a_{ijkR}\tilde{f}_R^{(ijk)} + a_{ijkI}\tilde{f}_I^{(ijk)}$$

$$c_0 \equiv a_0 + b_0 \quad \text{and} \quad c_{ijR} = a_{ijR} + b_{ijR} \quad \text{and} \quad c_{ijI} = a_{ijI} + b_{ijI} \quad \text{and}$$

$$c_{ikR} = a_{ikR} + b_{ikR} \quad \text{and} \quad c_{ikI} = a_{ikI} + b_{ikI}$$

$$a_0 \equiv 2\left(\tilde{C}_{ijkR}^{inv}\tilde{u}_{jR}\tilde{u}_{kR} + \tilde{C}_{ijkI}^{inv}\tilde{u}_{jI}\tilde{u}_{kR} + \tilde{C}_{ijkR}^{inv}\tilde{u}_{jI}\tilde{u}_{kI} - \tilde{C}_{ijkI}^{inv}\tilde{u}_{jR}\tilde{u}_{kI}\right)$$

$$a_{ijR} \equiv 2\left(-\tilde{C}_{ijkR}^{inv}\tilde{u}_{iR}\tilde{u}_{kR} - \tilde{C}_{ijkI}^{inv}\tilde{u}_{iI}\tilde{u}_{kR} - \tilde{C}_{ijkR}^{inv}\tilde{u}_{iI}\tilde{u}_{kI} + \tilde{C}_{ijkI}^{inv}\tilde{u}_{iR}\tilde{u}_{kI}\right)$$

$$a_{ijI} \equiv 2\left(\tilde{C}_{ijkR}^{inv}\tilde{u}_{iI}\tilde{u}_{kR} - \tilde{C}_{ijkI}^{inv}\tilde{u}_{iR}\tilde{u}_{kR} - \tilde{C}_{ijkR}^{inv}\tilde{u}_{iR}\tilde{u}_{kI} - \tilde{C}_{ijkI}^{inv}\tilde{u}_{iI}\tilde{u}_{kI}\right)$$

$$a_{ikR} \equiv 2\left(-\tilde{C}_{ijkR}^{inv}\tilde{u}_{iR}\tilde{u}_{jR} - \tilde{C}_{ijkI}^{inv}\tilde{u}_{iR}\tilde{u}_{jI} - \tilde{C}_{ijkR}^{inv}\tilde{u}_{iI}\tilde{u}_{jI} - \tilde{C}_{ijkI}^{inv}\tilde{u}_{iI}\tilde{u}_{jR}\right)$$

$$a_{ikI} \equiv 2\left(\tilde{C}_{ijkR}^{inv}\tilde{u}_{iR}\tilde{u}_{jI} - \tilde{C}_{ijkI}^{inv}\tilde{u}_{iR}\tilde{u}_{jR} - \tilde{C}_{ijkR}^{inv}\tilde{u}_{iI}\tilde{u}_{jR} - \tilde{C}_{ijkI}^{inv}\tilde{u}_{iI}\tilde{u}_{jI}\right)$$

$$a_{ijkR} \equiv 2\tilde{C}_{ijkR}^{inv}|\tilde{u}_i|^2 \quad \text{and} \quad a_{ijkI} \equiv -2\tilde{C}_{ijkI}^{inv}|\tilde{u}_i|^2$$

$$b_0 \equiv -2\tilde{C}_{ij}^{inv}\left(|\tilde{u}_i|^2 + |\tilde{u}_j|^2\right) - 2\tilde{C}_{ik}^{inv}\left(|\tilde{u}_i|^2 + |\tilde{u}_k|^2\right)$$

$$b_{ijR} \equiv -2\tilde{C}_{ij}^{inv}\left(\tilde{u}_{iR}\tilde{u}_{jR} + \tilde{u}_{iI}\tilde{u}_{jI}\right) \quad \text{and} \quad b_{ijI} \equiv -2\tilde{C}_{ij}^{inv}\left(\tilde{u}_{iR}\tilde{u}_{jI} + \tilde{u}_{iI}\tilde{u}_{jR}\right)$$

$$b_{ikR} \equiv -2\tilde{C}_{ij}^{inv}(\tilde{u}_{iR}\tilde{u}_{kR} + \tilde{u}_{iI}\tilde{u}_{kI}) \quad \text{and} \quad b_{ijI} \equiv -2\tilde{C}_{ik}^{inv}(\tilde{u}_{iR}\tilde{u}_{kI} + \tilde{u}_{iI}\tilde{u}_{kR})$$

(55)

The first derivatives are

$$\frac{\partial E_T}{\partial m_p^{(ij)}} = \frac{1}{\pi}\int_0^{+\infty} \frac{\partial H}{\partial m_p^{(ij)}}\, d\omega \quad \text{and} \quad \frac{\partial E_T}{\partial m_p^{(ik)}} = \frac{1}{\pi}\int_0^{+\infty} \frac{\partial H}{\partial m_p^{(ik)}}\, d\omega \quad \text{with}$$

$$\frac{\partial H}{\partial m_p^{(ij)}} = c_{ijR}\frac{\partial \tilde{f}_R^{(ij)}}{\partial m_p^{(ij)}} + c_{ijR}\frac{\partial \tilde{f}_I^{(ij)}}{\partial m_p^{(ij)}} + a_{ijkR}\frac{\partial \tilde{f}_R^{(ijj)}}{\partial m_p^{(ij)}} + a_{ijkI}\frac{\partial \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)}}$$

$$\frac{\partial H}{\partial m_p^{(ik)}} = c_{ikR}\frac{\partial \tilde{f}_R^{(ik)}}{\partial m_p^{(ik)}} + c_{ijR}\frac{\partial \tilde{f}_I^{(ij)}}{\partial m_p^{(ik)}} + a_{ijkR}\frac{\partial \tilde{f}_R^{(ijj)}}{\partial m_p^{(ik)}} + a_{ijkI}\frac{\partial \tilde{f}_R^{(ijk)}}{\partial m_p^{(ik)}}$$

$$\frac{\partial \tilde{f}_R^{(ij)}}{\partial m_p^{(ij)}} = \omega\tilde{f}_I^{(ij)}\frac{\partial \tau^{(ij)}}{\partial m_p^{(ij)}} \quad \text{and} \quad \frac{\partial \tilde{f}_I^{(ij)}}{\partial m_p^{(ij)}} = -\omega\tilde{f}_R^{(ij)}\frac{\partial \tau^{(ij)}}{\partial m_p^{(ij)}}$$

$$\frac{\partial \tilde{f}_R^{(ik)}}{\partial m_p^{(ik)}} = \omega \tilde{f}_I^{(ik)} \frac{\partial \tau^{(ik)}}{\partial m_p^{(ik)}} \quad \text{and} \quad \frac{\partial \tilde{f}_I^{(ik)}}{\partial m_p^{(ik)}} = -\omega \tilde{f}_R^{(ik)} \frac{\partial \tau^{(ik)}}{\partial m_p^{(ik)}}$$

$$\frac{\partial \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)}} = \omega \tilde{f}_I^{(ijk)} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ij)}} \quad \text{and} \quad \frac{\partial \tilde{f}_I^{(ijk)}}{\partial m_p^{(ij)}} = -\omega \tilde{f}_R^{(ijk)} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ij)}}$$

$$\frac{\partial \tilde{f}_R^{(ijk)}}{\partial m_p^{(ik)}} = \omega \tilde{f}_I^{(ijk)} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ik)}} \quad \text{and} \quad \frac{\partial \tilde{f}_I^{(ijk)}}{\partial m_p^{(ik)}} = -\omega \tilde{f}_R^{(ik)} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ik)}}$$

$$\frac{\partial \tau^{(ijk)}}{\partial m_p^{(ij)}} = -\frac{\partial \tau^{(ij)}}{\partial m_p^{(ij)}} \quad \text{and} \quad \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ik)}} = \frac{\partial \tau^{(ik)}}{\partial m_p^{(ij)}} \tag{56}$$

The second derivatives are:

$$\frac{\partial^2 H}{\partial m_p^{(ij)} \partial m_q^{(ij)}} = c_{ijR} \frac{\partial^2 \tilde{f}_R^{(ij)}}{\partial m_p^{(ij)} \partial m_q^{(ij)}} + c_{ijR} \frac{\partial^2 \tilde{f}_I^{(ij)}}{\partial m_p^{(ij)} \partial m_q^{(ij)}} + a_{ijkR} \frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)} \partial m_q^{(ij)}} + a_{ijkI} \frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)} \partial m_q^{(ij)}}$$

$$\frac{\partial^2 H}{\partial m_p^{(ik)} \partial m_q^{(ik)}} = c_{ijR} \frac{\partial^2 \tilde{f}_R^{(ik)}}{\partial m_p^{(ik)} \partial m_q^{(ik)}} + c_{ijR} \frac{\partial^2 \tilde{f}_I^{(ik)}}{\partial m_p^{(ik)} \partial m_q^{(ik)}} + a_{ijkR} \frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)} \partial m_q^{(ij)}} + a_{ijkI} \frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)} \partial m_q^{(ij)}}$$

$$\frac{\partial^2 H}{\partial m_p^{(ij)} \partial m_q^{(ik)}} = a_{ijkR} \frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)} \partial m_q^{(ij)}} + a_{ijkI} \frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)} \partial m_q^{(ij)}}$$

$$\frac{\partial^2 \tilde{f}_R^{(iu)}}{\partial m_p^{(iu)} \partial m_q^{(iu)}} = \omega \frac{\partial \tilde{f}_I^{(iu)}}{\partial m_p^{(iu)}} \frac{\partial \tau^{(iu)}}{\partial m_q^{(iu)}} + \omega \tilde{f}_I^{(iu)} \frac{\partial^2 \tau^{(iu)}}{\partial m_p^{(iu)} \partial m_q^{(iu)}} \quad \text{with } u = (j,k)$$

$$\frac{\partial^2 \tilde{f}_I^{(iu)}}{\partial m_p^{(iu)} \partial m_q^{(iu)}} = -\omega \frac{\partial \tilde{f}_R^{(iu)}}{\partial m_p^{(iu)}} \frac{\partial \tau^{(iu)}}{\partial m_q^{(iu)}} - \omega \tilde{f}_R^{(iu)} \frac{\partial^2 \tau^{(iu)}}{\partial m_p^{(iu)} \partial m_q^{(iu)}} \quad \text{with } u = (j,k)$$

$$\frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)} m_q^{(ij)}} = \omega \frac{\partial \tilde{f}_I^{(ijk)}}{\partial m_p^{(ij)}} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ij)}} + \omega \tilde{f}_I^{(ijk)} \frac{\partial^2 \tau^{(ijk)}}{\partial m_p^{(ij)} \partial m_p^{(ij)}}$$

$$\frac{\partial^2 \tilde{f}_I^{(ijk)}}{\partial m_p^{(ij)} m_q^{(ij)}} = -\omega \frac{\partial \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)}} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ij)}} - \omega \tilde{f}_R^{(ijk)} \frac{\partial^2 \tau^{(ijk)}}{\partial m_p^{(ij)} \partial m_p^{(ij)}}$$

$$\frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ik)} m_q^{(ik)}} = \omega \frac{\partial \tilde{f}_I^{(ijk)}}{\partial m_p^{(ik)}} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ik)}} + \omega \tilde{f}_I^{(ijk)} \frac{\partial^2 \tau^{(ijk)}}{\partial m_p^{(ik)} \partial m_p^{(ik)}}$$

$$\frac{\partial^2 \tilde{f}_I^{(ijk)}}{\partial m_p^{(ik)} m_q^{(ik)}} = -\omega \frac{\partial \tilde{f}_R^{(ijk)}}{\partial m_p^{(ik)}} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ik)}} - \omega \tilde{f}_R^{(ijk)} \frac{\partial^2 \tau^{(ijk)}}{\partial m_p^{(ik)} \partial m_p^{(ik)}}$$

$$\frac{\partial^2 \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)} m_q^{(ik)}} = \omega \frac{\partial \tilde{f}_I^{(ijk)}}{\partial m_p^{(ij)}} \frac{\partial \tau^{(ijk)}}{\partial m_{pq}^{(ik)}} \quad \text{and} \quad \frac{\partial^2 \tilde{f}_I^{(ijk)}}{\partial m_p^{(ij)} m_q^{(ik)}} = -\omega \frac{\partial \tilde{f}_R^{(ijk)}}{\partial m_p^{(ij)}} \frac{\partial \tau^{(ijk)}}{\partial m_p^{(ik)}}$$

(57)

For a linear travel time model with a matrix G_{nq}

$$\frac{\partial \tau^{(ij)}(\omega_n)}{\partial m_q^{(ij)}} = \frac{\partial \tau_n^{(ik)}(\omega_n)}{\partial m_q^{(ik)}} = G_{nq}$$

(58)

and all second derivatives of the τs are zero. All derivatives have been checked numerically.

We start our analysis of the confidence of the method with a derivation that establishes a relationship between covariance and the second derivative, which follows a similar one by Menke (Geophysical Data Analysis, 2024, Eqn. 4.76). In a linear generalized least squares problem, the weighted least squares error is defined as $E \equiv \mathbf{e}^T \mathbf{C}_e^{-1} \mathbf{e}$ with individual $\mathbf{e} \equiv \mathbf{d} - \mathbf{Km}$, where $\mathbf{d}$ are observed data and $\mathbf{K}$ is a data kernel**.** Inserting the second expression into the first and multiplying out terms yields:

$$E = (\mathbf{d} - \mathbf{Km})^T \mathbf{C}_e^{-1} (\mathbf{d} - \mathbf{Km}) = \mathbf{d}^T \mathbf{C}_e^{-1} \mathbf{d} - 2\mathbf{m}^T \mathbf{K}^T \mathbf{C}_e^{-1} \mathbf{d} + \mathbf{m}^T \mathbf{K}^T \mathbf{C}_e^{-1} \mathbf{Km}$$

(59)

The derivatives of the error with respect to the model parameters are

$$\frac{\partial E}{\partial \mathbf{m}} = -2\mathbf{K}^T \mathbf{C}_e^{-1} \mathbf{d} + 2\mathbf{K}^T \mathbf{C}_e^{-1} \mathbf{Gm} \quad \text{and} \quad \frac{\partial^2 E}{\partial \mathbf{m} \partial \mathbf{m}} = 2\mathbf{K}^T \mathbf{C}_e^{-1} \mathbf{K}$$

(60)

The generalizes least squares solution is obtained by setting the first derivative to zero, which yields

$$\mathbf{m} = \mathbf{K}^{-g} \mathbf{d} \quad \text{with} \quad \mathbf{K}^{-g} \equiv [\mathbf{K}^T \mathbf{C}_e^{-1} \mathbf{K}]^{-1} \mathbf{K}^T \mathbf{C}_e^{-1}$$

(61)

The covariance of the solution is obtained by standard error propagation

$$\operatorname{cov} \mathbf{m} = \mathbf{K}^{-g} \mathbf{C}_e \{\mathbf{K}^{-g}\}^T = [\mathbf{G}^T \mathbf{C}_e^{-1} \mathbf{G}]^{-1}$$

(62)

By comparing Eqns. (60) and (62), it is evident that the covariance is related to the second derivative

$$\operatorname{cov} \mathbf{m} = 2 \left[\frac{\partial^2 E}{\partial \mathbf{m} \partial \mathbf{m}} \right]^{-1}$$

(63)

Although this relationship is exact only in the linear case, we assume it holds approximately in nonlinear cases like the one considered here, too. Comparing the generalized least squares error E with the waveform error E_T (Eqn. 53), we find

$$\mathbf{m} \equiv \begin{bmatrix} \mathbf{m}^{(ij)} \\ \mathbf{m}^{(ik)} \end{bmatrix} \quad \mathbf{e} \equiv \begin{bmatrix} \tilde{e}^{(ij)} \\ \tilde{e}^{(ik)} \end{bmatrix} \quad \text{and} \quad \mathbf{C}_e^{-1} = \tilde{\mathbf{C}}^{inv} \quad \text{and} \quad E = \frac{\pi}{\Delta\omega} E_T \quad \text{and} \quad \operatorname{cov} \mathbf{m} = \frac{2\Delta\omega}{\pi} \left[\frac{\partial^2 E_T}{\partial \mathbf{m} \partial \mathbf{m}} \right]^{-1} \tag{64}$$

Written in more detail, the latter result is:

$$\begin{bmatrix} \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ij)}\right) & \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ik)}\right) \\ \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ik)}\right) & \operatorname{cov}\left(\mathbf{m}^{(ik)}, \mathbf{m}^{(ik)}\right) \end{bmatrix} = \frac{2\Delta\omega}{\pi} \begin{bmatrix} \dfrac{\partial^2 E_T}{\partial \mathbf{m}^{(ij)} \mathbf{m}^{(ij)}} & \dfrac{\partial^2 E_T}{\partial \mathbf{m}^{(ij)} \mathbf{m}^{(ik)}} \\ \dfrac{\partial^2 E_T}{\partial \mathbf{m}^{(ij)} \mathbf{m}^{(ik)}} & \dfrac{\partial^2 E_T}{\partial \mathbf{m}^{(ij)} \mathbf{m}^{(ik)}} \end{bmatrix}^{-1} \tag{65}$$

where the derivatives have been evaluated at the estimated solution. The travel times can be aggregated into a single equation

$$\boldsymbol{\tau} \equiv \left[\tau^{(ij)}(\omega_0) \cdots \tau^{(ij)}(\omega_N), \tau^{(ik)}(\omega_0) \cdots \tau^{(ij)}(\omega_N) \right]^T = \begin{bmatrix} \mathbf{G} & 0 \\ 0 & \mathbf{G} \end{bmatrix} \begin{bmatrix} \mathbf{m}^{(ij)} \\ \mathbf{m}^{(ik)} \end{bmatrix} \tag{66}$$

Then, by standard error propagation

$$\operatorname{cov}(\boldsymbol{\tau}) = \begin{bmatrix} \mathbf{G} & 0 \\ 0 & \mathbf{G} \end{bmatrix} \begin{bmatrix} \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ij)}\right) & \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ik)}\right) \\ \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ik)}\right) & \operatorname{cov}\left(\mathbf{m}^{(ik)}, \mathbf{m}^{(ik)}\right) \end{bmatrix} \begin{bmatrix} \mathbf{G} & 0 \\ 0 & \mathbf{G} \end{bmatrix}^T =$$

$$\begin{bmatrix} \mathbf{G} \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ij)}\right) \mathbf{G}^T & \mathbf{G} \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ik)}\right) \mathbf{G}^T \\ \mathbf{G} \operatorname{cov}\left(\mathbf{m}^{(ij)}, \mathbf{m}^{(ik)}\right) \mathbf{G}^T & \mathbf{G} \operatorname{cov}\left(\mathbf{m}^{(ik)}, \mathbf{m}^{(ik)}\right) \mathbf{G}^T \end{bmatrix} \tag{67}$$

At any given frequency, say ω_0, the slowness vector is $\left[s_x(\omega), s_y(\omega) \right]^T = \mathbf{M}^{-1} \left[\tau^{(ij)}(\omega_0), \tau^{(ij)}(\omega_0) \right]^T$, where the matrix is given by Eqn. (7). The aggregated form of this equation is

$$\mathbf{s} \equiv \left[s_x(\omega_0) \cdots s_x(\omega_N), s_y(\omega_0) \cdots s_y(\omega_N) \right]^T = \mathbf{W} \boldsymbol{\tau}$$

$$\operatorname{cov} \mathbf{s} = \mathbf{W} \operatorname{cov}(\boldsymbol{\tau}) \mathbf{W}^T \tag{68}$$

The matrix $\mathbf{W}$, which is sparse, contains two elements of $\mathbf{M}^{-1}$ on each of its rows:

$$\mathbf{W} \equiv \begin{bmatrix} M_{11}^{-1}\mathbf{I} & M_{12}^{-1}\mathbf{I} \\ M_{21}^{-1}\mathbf{I} & M_{22}^{-1}\mathbf{I} \end{bmatrix}$$

(69)

Arguably, it is sufficient to determine only the variance of the scalar slowness and direction (or alternatively, scalar velocity and direction), in which case Eqns. (37)-(39) can be applied separately at each frequency.

Eqn (7) indicates that a perturbation is the slowness vector is given by $\Delta\mathbf{s} = \mathbf{M}^{-1}\Delta\boldsymbol{\tau}$. Standard error propagation then yields

$$\mathrm{cov}(\mathbf{s}) = \mathbf{M}^{-1}\,\mathrm{cov}(\boldsymbol{\tau})\,\mathbf{M}^{-1T}$$

(36)

The magnitude s and azimuth θ of the slowness are given by

$$s \equiv \left[s_x^2 + s_y^2\right]^{1/2} \quad \text{and} \quad \theta \equiv \tan^{-1}(r) \quad \text{with} \quad r \equiv \frac{s_y}{s_x}$$

(37)

We collect these quantities in the vector $\mathbf{w} \equiv [s, \theta]^T$ where $s = \left[s_{0x}^2 + s_{0y}^2\right]^{1/2}$ and $\theta = \tan^{-1}(s_y/s_x)$. Taylor's theorem then is used to construct the perturbation $\Delta\mathbf{w}$ around a reference value, say $\mathbf{w}_0$:

$$\Delta\mathbf{w} \equiv \begin{bmatrix} \Delta s \\ \Delta\theta \end{bmatrix} = \mathbf{Q}_w \Delta\mathbf{s} \quad \text{with}$$

$$\mathbf{Q}_w \equiv \begin{bmatrix} -s_{0x}\left[s_{0x}^2 + s_{0y}^2\right]^{-1/2} & -s_{0y}\left[s_{0x}^2 + s_{0y}^2\right]^{-1/2} \\ -s_{0y}\left(\dfrac{1}{s_{0x}^2 + s_{0y}^2}\right) & s_{0x}\left(\dfrac{1}{s_{0x}^2 + s_{0y}^2}\right) \end{bmatrix}$$

$$\mathrm{cov}(\mathbf{w}) = \mathbf{Q}_w\,\mathrm{cov}(\mathbf{s})\,\mathbf{Q}_w^T$$

(38)

Alternatively, phase velocity v can be used in place of slowness. Defining a vector $\mathbf{z} \equiv [v, \theta]^T$, where $v = \left[s_{0x}^2 + s_{0y}^2\right]^{-1/2}$ and $\theta = \tan^{-1}(s_y/s_x)$.we have

$$\Delta\mathbf{z} \equiv \begin{bmatrix} \Delta v \\ \Delta\theta \end{bmatrix} = \mathbf{Q}_v \Delta\mathbf{s}$$

$$\mathbf{Q}_z \equiv \begin{bmatrix} -s_{0x}\left[s_{0x}^2 + s_{0y}^2\right]^{-3/2} & -s_{0y}\left[s_{0x}^2 + s_{0y}^2\right]^{-3/2} \\ -s_{0y}\left(\dfrac{1}{s_{0x}^2 + s_{0y}^2}\right) & s_{0x}\left(\dfrac{1}{s_{0x}^2 + s_{0y}^2}\right) \end{bmatrix}$$

$$\mathrm{cov}(\mathbf{z}) = \mathbf{Q}_z\,\mathrm{cov}(\mathbf{s})\,\mathbf{Q}_z^T$$

For simplicity, we assume that the three stations are close enough together that the noise on each has the same variance:

$$\sigma_u^2 \equiv \mathrm{var}(\tilde{u}_i) = \mathrm{var}\big(\tilde{u}_j\big) = \mathrm{var}(\tilde{u}_k) \tag{70}$$

The weighting matrix $\tilde{\mathbf{C}}^{inv}$ that appears Equation (53) is the inverse of the covariance matrix of the observations, and has elements:

$$\mathbf{C} \equiv \begin{bmatrix} C_{ij} & C_{ijk} \\ C_{ijk}^* & C_{ik} \end{bmatrix}$$

$$C_{ij} \equiv \mathrm{var}\big(\tilde{e}_{ij}\big) = \mathrm{var}\big(\tilde{u}_j - \tilde{u}_i \tilde{f}^{(ij)}\big) = 2\sigma_u^2\big(1 - \mathrm{real}\big(\tilde{f}^{(ij)}\big)\,\mathrm{cov}\big(\tilde{u}_i, \tilde{u}_j\big)\big)$$

$$= 2\sigma_u^2\left(1 - \cos\big(\omega\tau^{(ij)}\big) J_0\big(\omega\tau^{(ij)}\big)\right)$$

$$C_{ik} \equiv \mathrm{var}(\tilde{e}_{ik}) = \mathrm{var}\big(\tilde{u}_k - \tilde{u}_i \tilde{f}^{(ik)}\big) = 2\sigma_u^2\left(1 - \mathrm{real}\big(\tilde{f}^{(ik)}\big) J_0\big(\omega\tau^{(ik)}\big)\right)$$

$$= 2\sigma_u^2\left(1 - \cos\big(\omega\tau^{(ik)}\big) J_0\big(\omega\tau^{(ik)}\big)\right)$$

$$C_{ijk} \equiv \mathrm{cov}\big(\tilde{e}_{ij}, \tilde{e}_{ik}\big) = \mathrm{cov}\big(\tilde{u}_j - \tilde{u}_i \tilde{f}^{(ij)}, \tilde{u}_k - \tilde{u}_i \tilde{f}^{(ik)}\big)$$

$$= \mathrm{cov}\big(\tilde{u}_j, \tilde{u}_k\big) - \mathrm{cov}\big(\tilde{u}_j, \tilde{u}_i \tilde{f}^{(ik)}\big) - \mathrm{cov}\big(\tilde{u}_i \tilde{f}^{(ij)}, \tilde{u}_k\big) + \mathrm{cov}\big(\tilde{u}_i \tilde{f}^{(ij)}, \tilde{u}_i \tilde{f}^{(ik)}\big)$$

$$= \mathrm{cov}\big(\tilde{u}_j, \tilde{u}_k\big) - \tilde{f}^{(ik)*}\,\mathrm{cov}\big(\tilde{u}_j, \tilde{u}_i\big) - \tilde{f}^{(ij)}\,\mathrm{cov}(\tilde{u}_i, \tilde{u}_k) + \tilde{f}^{(ij)} \tilde{f}^{(ik)*}\,\mathrm{cov}(\tilde{u}_i, \tilde{u}_i)$$

$$= \sigma_u^2\big(J_0\big(\omega\tau^{(jk)}\big) - \exp\big\{i\omega\tau^{(ik)}\big\} J_0\big(\omega\tau^{(ij)}\big) - \exp\big\{-i\omega\tau^{(ij)}\big\} J_0\big(\omega\tau^{(ik)}\big) + \exp\big\{i\omega\tau^{(ijk)}\big\}\big) \tag{71}$$

The weight matrix $\tilde{\mathbf{C}}^{inv}$ can then be found by inverting $\mathbf{C}$ at each frequency value.

One issue that arises is that $\tilde{\mathbf{C}}^{inv}$ is a function of the travel time functions, which are initially unknown. Our practice is to use the $\tilde{\mathbf{C}}^{inv}$ implied by the starting guess in the iterative inversion. Another possible strategy is to update $\tilde{\mathbf{C}}^{inv}$ after each iteration.

As an example, we apply the method to a synthetic dataset similar to the one described in the previous section. The travel time model is linear, with $M = 2$ model parameters for each pair of seismograms. A Newton method inversion converges to the correct solution in 15 iterations and leads to well-aligned seismograms (not shown). As intended, the parameterized travel time functions average data from different frequencies, leading to confidence intervals that are smaller than those provided by the phase method for a given signal-to-noise ratio (Fig. 8). However, the improvement in confidence is deceptive, in sense that it is dependent on the assumption that $v(f)$ varies smoothly varies with frequency, whereas in some cases it may not.

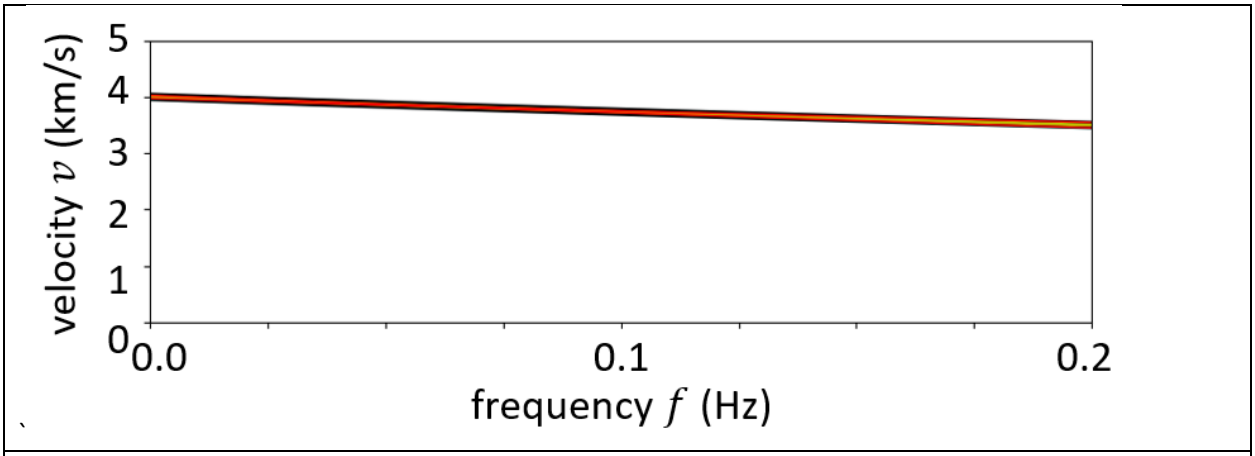

Fig. 8. Phase velocity estimated by the phase method applied to the exemplary seismograms shown in Fig. 6. True curve (black), estimated curve (yellow), and 95% confidence intervals at a signal-to-noise ration of $R = 5$.

Code

```
def wfpredictedBC( A, B, C, Dt, mij, mik, wpos):
# compute predicted seismogram Bpre
    N, i = np.shape(A);
    M, i = np.shape(mij);
    Nw, i = np.shape(wpos);
    moB = mij;
    moC = mik;
    fmax = 1.0/(2.0*Dt);
    Df = fmax/(N/2);
    Nf = int(N/2)+1;
    wposfull = 2.0*pi*eda_cvec( Df*np.linspace(0,Nf-1,Nf) );
    TposB, dTdmiB = wfTandderivs3(wpos,moB,False);
    TposBfull = np.concatenate( (TposB,TposB[Nw-1,0]*np.ones((Nf-Nw,1))), axis=0 );
    TposC, dTdmiC = wfTandderivs3(wpos,moC,False);
    TposCfull = np.concatenate( (TposC,TposC[Nw-1,0]*np.ones((Nf-Nw,1))), axis=0 );
    w = np.concatenate( (wposfull, -np.flip(wposfull[1:Nf-1],axis=0)), axis=0 );
    TB = np.concatenate( (TposBfull, np.flip(TposBfull[1:Nf-1],axis=0)), axis=0 );
    TC = np.concatenate( (TposCfull, np.flip(TposCfull[1:Nf-1],axis=0)), axis=0 );
    ftpB = np.exp( -complex(0.0,1.0)*np.multiply(w,TB) );
    ftpC = np.exp( -complex(0.0,1.0)*np.multiply(w,TC) );
    At = np.fft.fft(A,axis=0);
    Bpre = np.real( np.fft.ifft(np.multiply(At,ftpB),axis=0) );
    Cpre = np.real( np.fft.ifft(np.multiply(At,ftpC),axis=0) );
    return Bpre, Cpre;

def wffouriersetup3( A, B, C, Dt, f0 ):
    # A, B, C: timeseries
    # Dt: time sampling
    # f0: maximum frequency to use in fitting
    # primary quantities
    N, i = np.shape(A);
    fmax = 1.0/(2.0*Dt);
    Df = fmax/(N/2);
    Dw = 2*pi*Df;
    Nf = int( f0/Df );
```

```
    wpos = 2.0*pi*eda_cvec( Df*np.linspace(0,Nf-1,Nf) );
    At = Dt*np.fft.fft( A, axis=0 );
    Bt = Dt*np.fft.fft( B, axis=0 );
    Ct = Dt*np.fft.fft( C, axis=0 );
    Atpos = At[0:Nf,0:1];
    Btpos = Bt[0:Nf,0:1];
    Ctpos = Ct[0:Nf,0:1];
    return Dw, wpos, Atpos, Btpos, Ctpos;

def wfderivssetup3( wpos, Atpos, Btpos, Ctpos, Cinv ):
    # wpos: non-negative angular frequencies, must start at 0, can
    #       end before Nyquist, say length is Nw
    # Atpos, Btpos, Ctpos: non-negative Fourier components of
    #       A(t), B(t), C(t) at angular frequencies wpos
    # Cinv Nwx2x2 inverse covariance matrix

    Nw, i = np.shape(wpos);

    # Hermetian correlation matrix
    Cij = np.real( Cinv[0:Nw,0:1,0] );
    Cik = np.real( Cinv[0:Nw,1:2,1] );
    CijkR = np.real( Cinv[0:Nw,0:1,1] );
    CijkI = np.imag( Cinv[0:Nw,0:1,1] );

    # real and imaginary parts of data
    uiR = np.real(Atpos);
    uiI = np.imag(Atpos);
    ujR = np.real(Btpos);
    ujI = np.imag(Btpos);
    ukR = np.real(Ctpos);
    ukI = np.imag(Ctpos);

    # weighted power in data
    uisq = np.power(uiR,2) + np.power(uiI,2);
    ujsq = np.power(ujR,2) + np.power(ujI,2);
    uksq = np.power(ukR,2) + np.power(ukI,2)
    Cijuisq = np.multiply( Cij, uisq );
    Cikuisq = np.multiply( Cik, uisq );
    Cijujsq = np.multiply( Cij, ujsq );
    Cikuksq = np.multiply( Cik, uksq );

    # energy in the timeseries
    CijPi = (1.0/pi)*Dw*( np.sum(Cijuisq) - 0.5*Cijuisq[0,0]);
    CikPi = (1.0/pi)*Dw*( np.sum(Cikuisq) - 0.5*Cikuisq[0,0]);
    CijPj = (1.0/pi)*Dw*( np.sum(Cijujsq) - 0.5*Cijujsq[0,0]);
    CikPk = (1.0/pi)*Dw*( np.sum(Cikuksq) - 0.5*Cikuksq[0,0]);

    # functions related to I1
    b0ij = Cijuisq + Cijujsq;
    bijR = -2.0*(np.multiply( Cij, np.multiply(uiR,ujR) + np.multiply(uiI,ujI)));
    bijI = -2.0*(np.multiply( Cij, np.multiply(uiR,ujI) - np.multiply(uiI,ujR)));

    # functions related to I2
    b0ik = Cikuisq + Cikuksq;
    bikR = -2.0*(np.multiply( Cik, np.multiply(uiR,ukR) + np.multiply(uiI,ukI)));
    bikI = -2.0*(np.multiply( Cik, np.multiply(uiR,ukI) - np.multiply(uiI,ukR)));

    # functions related to I3
```

```
    a0 = 2.0*(  np.multiply(CijkR,np.multiply(ujR,ukR)) +
np.multiply(CijkI,np.multiply(ujI,ukR))
          + np.multiply(CijkR,np.multiply(ujI,ukI)) -
np.multiply(CijkI,np.multiply(ujR,ukI)) );
    aijR = 2.0*( - np.multiply(CijkR,np.multiply(uiR,ukR)) -
np.multiply(CijkI,np.multiply(uiI,ukR))
             - np.multiply(CijkR,np.multiply(uiI,ukI)) +
np.multiply(CijkI,np.multiply(uiR,ukI)) );
    aijI = 2.0*(   np.multiply(CijkR,np.multiply(uiI,ukR)) -
np.multiply(CijkI,np.multiply(uiR,ukR))
             - np.multiply(CijkR,np.multiply(uiR,ukI)) -
np.multiply(CijkI,np.multiply(uiI,ukI)) );
    aikR = 2.0*( - np.multiply(CijkR,np.multiply(ujR,uiR)) -
np.multiply(CijkI,np.multiply(ujI,uiR))
             - np.multiply(CijkR,np.multiply(ujI,uiI)) +
np.multiply(CijkI,np.multiply(ujR,uiI)) );
    aikI = 2.0*( - np.multiply(CijkR,np.multiply(ujI,uiR)) +
np.multiply(CijkI,np.multiply(ujR,uiR))
             + np.multiply(CijkR,np.multiply(ujR,uiI)) +
np.multiply(CijkI,np.multiply(ujI,uiI)) );
    aijkR =  2.0*np.multiply(CijkR,uisq);
    aijkI = -2.0*np.multiply(CijkI,uisq);

    # only the a0 integral is independent of m and can be integrated at this point
    # Note: not sure about the normalization, need to check
    Ia0 = (1.0/pi)*Dw*( np.sum(a0) - 0.5*a0[0,0]);
    b0 = b0ij+b0ik;
    Ib0 = (1.0/pi)*Dw*( np.sum(b0) - 0.5*b0[0,0]);

    return Ib0, bijR, bijI, bikR, bikI, Ia0, aijR, aijI, aikR, aikI, aijkR, aijkI;

def wfTandderivs3(wpos, m, dosecond):
    # implements T(w,m)=Gm, dT/dm=G and d2T/dm2=0
    # computes and returns d2T/dm2 only when dosecond is True
    Nf,i = np.shape(wpos);
    M,i = np.shape(m);
    Dw = wpos[1,0]-wpos[0,0];
    # linear model, with G externally defined
    Tpos = np.matmul(G,m);
    dTdmi = G;
    if dosecond :
        d2Tdmi2 = np.zeros((Nf,M,M)); # is identically zero in this case
                                      # but in general, not zero
        return Tpos, dTdmi, d2Tdmi2;
    else:
        return Tpos, dTdmi;

def wfderivs3( wpos, Ib0, bijR, bijI, bikR, bikI,
               Ia0, afijR, afijI, afikR, afikI, afijkR, afijkI, mij, mik, dosecond ):

    # derivates with respect to model parameters
    M, i = np.shape(mij);
    Nf, i = np.shape(wpos);
    dEdmij = np.zeros((M,1));
    dEdmik = np.zeros((M,1));

    if dosecond:
        Tij, dTijdmij, d2Tijdmij2 = wfTandderivs3(wpos,mij,dosecond);
```

```
        Tik, dTikdmik, d2Tikdmik2 = wfTandderivs3(wpos,mik,dosecond);
    else:
        Tij, dTijdmij = wfTandderivs3(wpos,mij,dosecond);
        Tik, dTikdmik = wfTandderivs3(wpos,mik,dosecond);

    # total error ET associated with Integrals I1 and I2
    ET = Ia0+Ib0;
    wTij = np.multiply(wpos,Tij);
    wTik = np.multiply(wpos,Tik);
    wTijk = np.multiply(wpos,Tik-Tij);
    fijR = np.cos( -wTij );
    fijI = np.sin( -wTij );
    fikR = np.cos( -wTik );
    fikI = np.sin( -wTik );
    fijkR = np.cos( -wTijk );
    fijkI = np.sin( -wTijk );
    Va =
np.multiply(aijR,fijR)+np.multiply(aijI,fijI)+np.multiply(aikR,fikR)+np.multiply(aikI,
fikI);
    Va = Va + np.multiply(aijkR,fijkR)+np.multiply(aijkI,fijkI);
    Vb =
np.multiply(bijR,fijR)+np.multiply(bijI,fijI)+np.multiply(bikR,fikR)+np.multiply(bikI,
fikI);
    V = Va+Vb;
    ET = ET + (1.0/pi)*Dw*(np.sum(V)-0.5*V[0,0]);

    # w * f's
    wfijR = np.multiply(wpos,fijR);
    wfijI = np.multiply(wpos,fijI);
    wfikR = np.multiply(wpos,fikR);
    wfikI = np.multiply(wpos,fikI);
    wfijkR = np.multiply(wpos,fijkR);
    wfijkI = np.multiply(wpos,fijkI);

    dTijkdmij = -dTijdmij;
    dTijkdmik =  dTikdmik;

    cijR = aijR + bijR;
    cijI = aijI + bijI;
    cikR = aikR + bikR;
    cikI = aikI + bikI;

    if( dosecond ):
        d2Tijkdmij2 = -d2Tijdmij2;
        d2Tijkdmik2 =  d2Tikdmik2;
        d2Edmij2 = np.zeros((M,M));
        d2Edmik2 = np.zeros((M,M));
        d2Edmijmik = np.zeros((M,M));

    # do for each mi
    for p in range(M):
        dfijRdmij =   np.multiply( wfijI, dTijdmij[0:Nf,p:p+1]);
        dfijIdmij =  -np.multiply( wfijR, dTijdmij[0:Nf,p:p+1]);
        dfijkRdmij =  np.multiply( wfijkI, dTijkdmij[0:Nf,p:p+1]);
        dfijkIdmij = -np.multiply( wfijkR, dTijkdmij[0:Nf,p:p+1]);

        H1 = np.multiply( cijR, dfijRdmij );
        H2 = np.multiply( cijI, dfijIdmij );
```

```
        H3 = np.multiply( aijkR, dfijkRdmij );
        H4 = np.multiply( aijkI, dfijkIdmij );
        H = H1+H2+H3+H4;
        dEdmij[p,0] = (1/pi)*(Dw*np.sum(H) - 0.5*Dw*H[0,0]);

        dfikRdmik =   np.multiply( wfikI, dTikdmik[0:Nf,p:p+1]);
        dfikIdmik =  -np.multiply( wfikR, dTikdmik[0:Nf,p:p+1]);
        dfijkRdmik =  np.multiply( wfijkI, dTijkdmik[0:Nf,p:p+1]);
        dfijkIdmik = -np.multiply( wfijkR, dTijkdmik[0:Nf,p:p+1]);

        H1 = np.multiply( cikR, dfikRdmik );
        H2 = np.multiply( cikI, dfikIdmik );
        H3 = np.multiply( aijkR, dfijkRdmik );
        H4 = np.multiply( aijkI, dfijkIdmik );
        H = H1+H2+H3+H4;
        dEdmik[p,0] = (1/pi)*(Dw*np.sum(H) - 0.5*Dw*H[0,0]);

        if( dosecond ):
            for q in range(p,M):
                # fij ij-ij deriv
                P1 = np.multiply(wpos, np.multiply(dfijIdmij, dTijdmij[0:Nf,q:q+1]));
                P2 = np.multiply(wpos, np.multiply(fijI, d2Tijdmij2[0:Nf,p:p+1,q]));
                d2fijRdmijpdmijq = P1+P2;
                P1 = -np.multiply(wpos, np.multiply(dfijRdmij, dTijdmij[0:Nf,q:q+1]));
                P2 = -np.multiply(wpos, np.multiply(fijR, d2Tijdmij2[0:Nf,p:p+1,q]));
                d2fijIdmijpdmijq = P1+P2;

                # fik ik-ik deriv
                P1 = np.multiply(wpos, np.multiply(dfikIdmik, dTikdmik[0:Nf,q:q+1]));
                P2 = np.multiply(wpos, np.multiply(fikI, d2Tikdmik2[0:Nf,p:p+1,q]));
                d2fikRdmikpdmikq = P1+P2;
                P1 = -np.multiply(wpos, np.multiply(dfikRdmik, dTikdmik[0:Nf,q:q+1]));
                P2 = -np.multiply(wpos, np.multiply(fikR, d2Tikdmik2[0:Nf,p:p+1,q]));
                d2fikIdmikpdmikq = P1+P2;

                # fijk ij-ij deriv
                P1 = np.multiply(wpos, np.multiply(dfijkIdmij,
dTijkdmij[0:Nf,q:q+1]));
                P2 = np.multiply(wpos, np.multiply(fijkI, d2Tijkdmij2[0:Nf,p:p+1,q]));
                d2fijkRdmijpdmijq = P1+P2;
                P1 = -np.multiply(wpos, np.multiply(dfijkRdmij,
dTijkdmij[0:Nf,q:q+1]));
                P2 = -np.multiply(wpos, np.multiply(fijkR,
d2Tijkdmij2[0:Nf,p:p+1,q]));
                d2fijkIdmijpdmijq = P1+P2;

                # fijk ik-ik deriv
                P1 = np.multiply(wpos, np.multiply(dfijkIdmik,
dTijkdmik[0:Nf,q:q+1]));
                P2 = np.multiply(wpos, np.multiply(fijkI, d2Tijkdmik2[0:Nf,p:p+1,q]));
                d2fijkRdmikpdmikq = P1+P2;
                P1 = -np.multiply(wpos, np.multiply(dfijkRdmik,
dTijkdmik[0:Nf,q:q+1]));
                P2 = -np.multiply(wpos, np.multiply(fijkR,
d2Tijkdmik2[0:Nf,p:p+1,q]));
                d2fijkIdmikpdmikq = P1+P2;

                # fijk ij-ik deriv
```

```
                P1 = np.multiply(wpos, np.multiply(dfijkIdmij,
dTijkdmik[0:Nf,q:q+1]));
                d2fijkRdmijpdmikq = P1;
                P1 = -np.multiply(wpos, np.multiply(dfijkRdmij,
dTijkdmik[0:Nf,q:q+1]));
                d2fijkIdmijpdmikq = P1;

                # E ij-ij deriv
                H1 = np.multiply( cijR, d2fijRdmijpdmijq );
                H2 = np.multiply( cijI, d2fijIdmijpdmijq );
                H3 = np.multiply( aijkR, d2fijkRdmijpdmijq );
                H4 = np.multiply( aijkI, d2fijkIdmijpdmijq );
                H = H1+H2+H3+H4;
                d2Edmij2[p,q] = (1/pi)*(Dw*np.sum(H) - 0.5*Dw*H[0,0]);
                d2Edmij2[q,p]  =d2Edmij2[p,q]

                # E ik-ik deriv
                H1 = np.multiply( cikR, d2fikRdmikpdmikq );
                H2 = np.multiply( cikI, d2fikIdmikpdmikq );
                H3 = np.multiply( aijkR, d2fijkRdmikpdmikq );
                H4 = np.multiply( aijkI, d2fijkIdmikpdmikq );
                H = H1+H2+H3+H4;
                d2Edmik2[p,q] = (1/pi)*(Dw*np.sum(H) - 0.5*Dw*H[0,0]);
                d2Edmik2[q,p]=d2Edmik2[p,q];

                # E ij-ik deriv
                H1 = np.multiply( aijkR, d2fijkRdmijpdmikq );
                H2 = np.multiply( aijkI, d2fijkIdmijpdmikq );
                H = H1+H2;
                d2Edmijmik[p,q] = (1/pi)*(Dw*np.sum(H) - 0.5*Dw*H[0,0]);
                d2Edmijmik[q,p] = d2Edmijmik[p,q];

    if( dosecond ):
        return ET, Tij, Tik, dEdmij, dEdmik, d2Edmij2, d2Edmik2, d2Edmijmik;
    else:
        return ET, Tij, Tik, dEdmij, dEdmik;

def wfinvert3newton(wpos, Ib0, bijR, bijI, bikR, bikI, Ia0, aijR, aijI, aikR, aikI,
aijkR, aijkI,
                    mijstart, mikstart, dmmax, Niter, DEstagnate):
    # This version uses Newton's method, utilizing both dEdm and d2Edm2
    # Ib0, bijR, bijI, bikR, bikI, Ia0, aijR, aijI, aikR, aikI, aijkR, aijkI
    #   from wfderivssetup3()
    # mstart: staring guess for solution
    # dm: maximum step size (I used 1.0);
    # Niter: maximum number of iterations (I used 100);
    # DEstagnate: change in relative error below which
    #    iterations aer terminated (I used 1e-5)

    M, i = np.shape(mijstart);

    # starting solution
    mijo = np.copy(mijstart);
    miko = np.copy(mikstart);
    mo = np.concatenate( (mijo,miko), axis=0 );
    # setup for iteration
    Eo, Tijo, Tiko, dEdmijo, dEdmiko, X1, X2, X3 = wfderivs3( wpos,
              Ib0, bijR, bijI, bikR, bikI, Ia0,
```

```
              aijR, aijI, aikR, aikI, aijkR, aijkI, mijo, miko, True);
    Estart = Eo;
    Elast = Eo;
    Tijstart = np.copy(Tijo);
    Tikstart = np.copy(Tiko);
    count=1;
    for k in range(Niter):

        dEdmo = np.concatenate( (dEdmijo,dEdmiko), axis=0 );
        d2Edm2o = np.concatenate(
(np.concatenate((X1,X3),axis=1),np.concatenate((X3.T,X2),axis=1)), axis=0 );
        dm = -la.solve( d2Edm2o, dEdmo );
        dmabsmax = np.max(np.abs(dm));
        if( dmabsmax > dmmax ):
            dm = dmmax*dm/dmabsmax;
        mo = mo + dm;
        mijo = mo[0:M,0:1];
        miko = mo[M:2*M,0:1];
        Eo, Tijo, Tiko, dEdmijo, dEdmiko, X1, X2, X3 = wfderivs3( wpos,
              Ib0, bijR, bijI, bikR, bikI, Ia0,
              aijR, aijI, aikR, aikI, aijkR, aijkI, mijo, miko, True);
        count=count+1;

        # reduction in error during this iteration
        if( k>3 ): # do at least 3 iterations
            if( Elast==0.0 ):
                break;
            DE = (Elast-Eo)/Elast;
            if( (DE>=0.0) and (DE<DEstagnate) ):
                break; # terminate iterations when change in solution is sufficiently
small
        Elast = Eo;
    return( Tijstart, Tikstart, Estart, mijo, miko, Tijo, Tijo, Eo, count );

# set up weight matrix
Tijtrue, tmp = wfTandderivs3(wpos,mijtrue,False);
Tiktrue, tmp = wfTandderivs3(wpos,miktrue,False);
MM = np.array( [ [DXij, DYij], [DXik, DYik] ] );
rhs = np.zeros((2,1));
strue = np.zeros((Nw,1));
vtrue = np.zeros((Nw,1));
Joij = np.zeros((Nw,));
Joik = np.zeros((Nw,));
Jojk = np.zeros((Nw,));
cosij = np.zeros((Nw,));
cosik = np.zeros((Nw,));
expij = np.zeros((Nw,),dtype=complex);
expik = np.zeros((Nw,),dtype=complex);
expijk = np.zeros((Nw,),dtype=complex);
vvarpair = np.zeros((Nw,1));
svarpair = np.zeros((Nw,1));
for i in range(Nw):
    rhs[0,0]=Tijtrue[i,0];
    rhs[1,0]=Tiktrue[i,0];
    sxy = la.solve( MM, rhs );
    strue[i,0] = sqrt( sxy[0,0]**2 + sxy[1,0]**2 );
    vtrue[i,0] = 1.0/strue[i,0];
    Joij[i] = special.j0( wpos[i,0]*Rij/vtrue[i,0] );
```

```
    Joik[i] = special.j0( wpos[i,0]*Rik/vtrue[i,0] );
    Jojk[i] = special.j0( wpos[i,0]*Rjk/vtrue[i,0] );
    cosij[i] = cos( wpos[i,0]*Rij/vtrue[i,0] );
    cosik[i] = cos( wpos[i,0]*Rik/vtrue[i,0] );
    expij[i] = np.exp( -complex(0.0,1.0)*wpos[i,0]*Rij/vtrue[i,0] );
    expik[i] = np.exp( complex(0.0,1.0)*wpos[i,0]*Rik/vtrue[i,0] ); # lack of muinus
sign intentional
    expijk[i] = np.exp( -complex(0.0,1.0)*wpos[i,0]*(Rij-Rik)/vtrue[i,0] );
    if( i>0 ):
        svarpair[i,0] = ((wpos[i,0]*Dx*snr)**-2)*(1.0-
cos(wpos[i,0]*Dx/vtrue[i,0])*special.j0(wpos[i,0]*Dx/vtrue[i,0]));
        vvarpair[i,0] = svarpair[i,0] * (vtrue[i,0]**4);
# correlation matrix
CC = np.zeros((Nw,2,2),dtype=complex);
Cinv = np.zeros((Nw,2,2),dtype=complex);

varu = (uimean/snr)**2;
if( FULLC ):
    CC[0:Nw,0,0] = 2.0*varu*(np.ones((Nw,))-np.multiply(cosij,Joij));
    CC[0:Nw,1,1] = 2.0*varu*(np.ones((Nw,))-np.multiply(cosik,Joik));
    myf = Jojk-np.multiply(expik,Joij)-np.multiply(expij,Joik)+expijk;
    CC[0:Nw,0,1] = varu*myf;
    CC[0:Nw,1,0] = np.conjugate( CC[0:Nw,0,1] );
else:
    CC[0:Nw,0,0] = 2.0*varu*np.ones((Nw,));
    CC[0:Nw,1,1] = 2.0*varu*np.ones((Nw,));

# fudge zero frequency case to the uncorrelated value
# else inverse will be singular
i=0;
CC[0,0,0] = 2.0*varu;
CC[0,1,1] = 2.0*varu;
CC[0,0,1] = 0.0;
CC[0,1,0] = 0.0;

for i in range(Nw):
    CCi = np.squeeze( CC[i,0:2,0:2] );
    CCinvi = la.inv( CCi );
    Cinv[i,0:2,0:2] = CCinvi;
    ev = la.eigh( CCinvi );
    if( ev[0][0] < 0.0 ):
        print("non-pos-def warning", i );
        print(ev[0] );
        print(CCi);
        print(CCinvi);

# its elements
Cij = np.real( Cinv[0:Nw,0:1,0] );
Cik = np.real( Cinv[0:Nw,1:2,1] );
Cijk = Cinv[0:Nw,0:1,1];
Cikj = Cinv[0:Nw,1:2,0];
CijkR = np.real( Cinv[0:Nw,0:1,1] );
CijkI = np.imag( Cinv[0:Nw,0:1,1] );


# variance
GG = np.concatenate(
    (np.concatenate((G,np.zeros((Nw,2))),axis=1),
```

```
     np.concatenate((np.zeros((Nw,2)),G),axis=1)), axis=0 );
covTT = np.matmul( np.matmul(GG,covm), GG.T );
print("min diag covTT", np.min(np.diag(covTT)) );
print("max diag covTT", np.max(np.diag(covTT)) );

MM = np.array( [ [DXij, DYij], [DXik, DYik] ] );
MMINV = la.inv(MM);
sx = np.zeros((Nw,1));
sy = np.zeros((Nw,1));
s = np.zeros((Nw,1));
svar = np.zeros((Nw,1));
v = np.zeros((Nw,1));
vvar = np.zeros((Nw,1));
rhs = np.zeros((2,1));
covT = np.zeros((2,2));
for i in range(Nw):
    rhs[0,0]=Tij[i,0];
    rhs[1,0]=Tik[i,0];
    sol = la.solve(MM,rhs);
    sx[i,0] = sol[0,0];
    sy[i,0] = sol[1,0];
    covT[0,0] = covTT[i,i];
    covT[1,1] = covTT[i+Nw,i+Nw];
    covT[0,1] = covTT[i,i+Nw];
    covT[1,0] = covTT[i+Nw,i];
    covsxy = np.matmul( np.matmul(MMINV,covT), MMINV.T);
    s[i,0] = sqrt( sx[i,0]**2 + sy[i,0]**2 );
    Q = np.zeros((2,1));
    Q[0,0] = sx[i,0]/s[i,0];
    Q[1,0] = sy[i,0]/s[i,0];
    svar[i,0] = np.matmul( np.matmul( Q.T, covsxy), Q )[0,0];
    if( svar[i,0] < 0.0 ):
        print("svar<0", i, svar[i,0] );
    v[i,0] = 1.0/s[i,0];
    vvar[i,0] = svar[i,0]/(s[i,0]**4);
    if( vvar[i,0] < 0.0 ):
        print("vvar<0", i, vvar[i,0] );
```