Tutorial: acoustic full waveform inversion in time domain

Khalid Almuteri and Kris Innanen

ABSTRACT

Full waveform inversion (FWI) is an optimization problem that aims at extracting the earth's physical parameters, from recorded seismic data, through an iterative inversion process. This paper reviews the concept of FWI and provides an environment where a single-parameter acoustic FWI problem can be solved in time domain. We cover the wave equation discretization, using a second-order finite difference approximation, Clayton-Engquist absorbing boundary conditions, modeling stability and grid dispersion, and we derive the gradient needed to solve this optimization problem.

INTRODUCTION

Full waveform inversion (FWI) is a powerful technique that aims at reconstructing the physical parameters of the subsurface, by iteratively minimizing the difference between modeled data and observed data (Lailly, 1983; Tarantola, 1984; Virieux and Operto, 2009). The rise of FWI as a velocity model building tool is due to the high-resolution models that it provides (Pan and Innanen, 2015).

This paper aims at providing the necessary tools to solve a single-parameter acoustic FWI problem in a self-sufficient manner. The purpose of self-sufficiency is to address introductory level, yet essential, aspects associated with FWI, creating an environment where advanced topics in FWI can be investigated.

This paper is organized as follows. First, we briefly review numerical modeling of the acoustic wave equation. We then introduce the FWI problem. Afterward, we cover the line search approach, which is a strategy for solving optimization problems. Finally, we discuss some computational optimization technics.

BACKGROUND

Wave quation

The 2-D acoustic wave equation is given by

$$\frac{\partial^2 u}{\partial t^2} = \frac{k^2}{\rho^2}\left[\rho \bigtriangledown \cdot \left(\frac{1}{\rho} \bigtriangledown u\right)\right] + f \tag{1}$$

where $\rho = \rho(x, z)$ is the density, $k = k(x, z)$ is the bulk modulus, $u = u(x, z, t)$ is the pressure field, and f is the source function; it expands to

$$\frac{\partial^2 u}{\partial t^2} = v^2\left[\frac{-1}{\rho}\left(\frac{\partial \rho}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial \rho}{\partial z}\frac{\partial u}{\partial z}\right) + \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2}\right)\right] + f \tag{2}$$

where $v = v(x,z) = \sqrt{\frac{\rho(x,z)}{k(x,z)}}$ is the compressional wave velocity. In this reposrt, we assume a constant density model; hence, equation (2) simplifies to

$$\frac{\partial^2 u}{\partial t^2} = v^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2} \right) + f \tag{3}$$

Absorbing boundary conditions

The absorbing boundary conditions (ABC) that we present here is the Clayton-Engquist boundary conditions (Clayton and Engquist, 1977). It is based on the paraxial approximation of the scalar wave equation that separates the wavefield into up-going/down-going and right-going/left-going waves (Claerbout, 1985).

To obtain the paraxial approximation, the first step is to Fourier transform the homogeneous wave equation

$$\frac{1}{v^2}\frac{\partial^2 u}{\partial t^2} = \bigtriangledown^2 u \tag{4}$$

which yields the dispersion relation

$$k_x^2 + k_z^2 = \frac{\omega^2}{v^2}. \tag{5}$$

The dispersion relation enables us to separate the wavefield by solving for the appropriate wavenumber. For instance, the down-going wavefield is obtained by solving for the positive values of the wavenumber k_z

$$k_z = \sqrt{\frac{\omega^2}{v^2} - k_x^2} = \frac{\omega}{v}\sqrt{1 - \frac{v^2}{\omega^2}k_x^2} \tag{6}$$

$$\Rightarrow \frac{v}{\omega}k_z = \sqrt{1 - \frac{v^2}{\omega^2}k_x^2}. \tag{7}$$

The paraxial approximation is achieved by expanding the square root $\sqrt{1 - \frac{v^2}{\omega^2}k_x^2}$ and transforming the expansion to space-time domain. Higher order approximations using the Taylor expansion gives an unstable paraxial approximation (Engquist and Majda, 1977), while Padé expansion, (equation 8), is more stable and used instead.

$$a_j = 1 - \frac{\left(vk_x/\omega\right)^2}{1 + a_{j-1}}, \qquad a_1 = 1. \tag{8}$$

As a result, the first two orders of the paraxial approximation in Fourier domain are

$$First - order : k_z = \frac{\omega}{v} \tag{9a}$$

$$Second - order : k_z = \frac{\omega}{v}\left(1 - \frac{1}{2}\left(\frac{vk_x}{\omega}\right)^2\right) \tag{9b}$$

and the corresponding paraxial approximations in time-space domain are

$$First - order : \frac{\partial u}{\partial z} - \frac{1}{v}\frac{\partial u}{\partial t} = 0 \tag{10a}$$

$$Second - order : \frac{\partial^2 u}{\partial z \partial t} - \frac{1}{v}\frac{\partial^2 u}{\partial t^2} + \frac{v}{2}\frac{\partial^2 u}{\partial x^2} = 0. \tag{10b}$$

The first-order approximation, (equation 10a), is known as the 5° wave equation, and the second-order approximation, (equation 10b), is known as the 15° wave equation (Claerbout, 1985).

The 15° wave equation is discretized and solved to apply the ABC for the sides of the model. In this case, (equation 10b) is used to create an absorbing boundary along the top side of the model. For the corners, the ABC is implemented by rotating the first-order approximations by 45° (Clayton and Engquist, 1977).

Differential operators

A second-order central finite difference approximation scheme is commonly used to numerically modeling the wave equation. The first partial derivative operator is defined as

$$\frac{\partial u}{\partial x} \approx \frac{u(x+h,z,t) - u(x-h,z,t)}{2h} \tag{11}$$

where h is the grid points spacing (dt is used for the time step). Equation 11 in the indices notation is

$$\frac{\partial u}{\partial x} = u_x|_{x=i} \approx \frac{u^n_{i+1,j} - u^n_{i-1,j}}{2h} \tag{12}$$

where we use subscript i for offset (x), subscript j for depth (z), and subscript n for time (t). The second partial derivative operator is defined as

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+h,z,t) - 2u(x,z,t) + u(x-h,z,t)}{h^2} \tag{13}$$

Equation (13) in the indices notation is

$$\frac{\partial^2 u}{\partial x^2} = u_{xx}|_{x=i} \approx \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{h^2} \tag{14}$$

The discretized form of equation (3) is given by

$$\frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{dt^2} \approx v^2_{i,j}\frac{u^n_{i+1,j} + u^n_{i,j+1} - 4u^n_{i,j} + u^n_{i-1,j} + u^n_{i,j-1}}{h^2} + f \tag{15}$$

where we are interested in solving for $u^{n+1}_{i,j}$.

Stability and grid dispersion

Wave equation discretization causes two issues to the numerical solution, grid dispersion, and instability of the forward modeling. Grid dispersion causes different frequencies to travel at different velocities, whereas instability of the finite-difference process results in accumulation of errors and failure of the numerical solution. As a result, two necessary criteria need to be met to limit the consequences of the discretization.

The first criterion relates the grid dispersion to the spatial sampling h. If h is not small enough, a dispersion will occur. For a second-order finite difference, at least, ten grid points are required for the upper half-power wavelength of the source (Alford et al., 1974). The equation that governs the relationship is

$$h \leq \frac{v_{min}(x,z)}{10 f_\circ} \tag{16}$$

where $f_\circ$ is the upper half-power frequency (Figure 1).

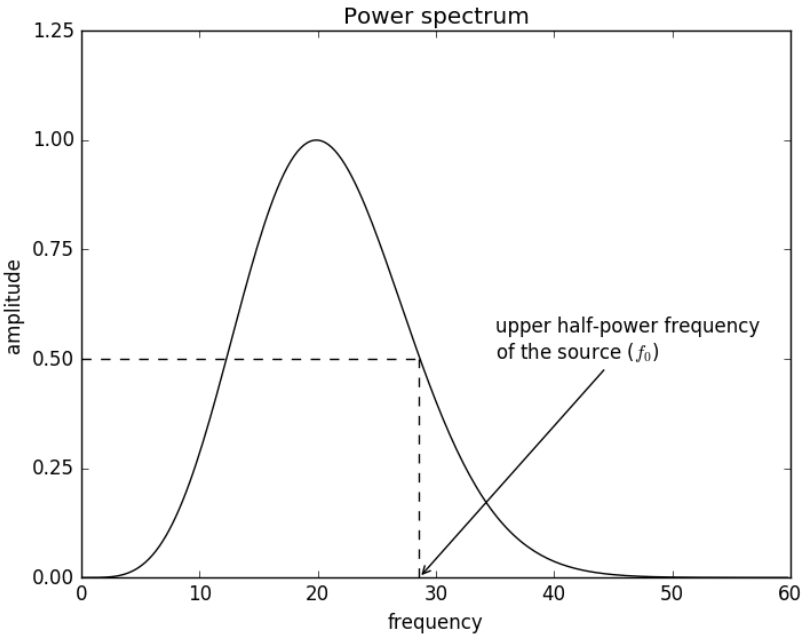

FIG. 1. Power spectrum of source wavelet [figure 3a of (Alford et al., 1974)].

The second criterion relates the stability to the temporal sampling dt (Lines et al., 1999). The criterion for a second-order scheme is

$$\frac{v_{max}(x,z)dt}{h} \leq \frac{1}{\sqrt{2}} \tag{17}$$

FULL WAVEFORM INVERSION

We define the data residual as

$$\Delta\mathbf{d} = \mathbf{d}_{obs} - \mathbf{d}_{cal}(\mathbf{m}) \tag{18}$$

where $\mathbf{d}_{obs}$ is the recorded seismic data and $\mathbf{d}_{cal}(\mathbf{m})$ is the modeled data, where $\mathbf{m}$ is p-wave velocity. FWI tries to minimize an objective function $E(\mathbf{m})$, which is the $\boldsymbol{L}_2$-norm of the data residual

$$E(\mathbf{m}) = \frac{1}{2}\left\|\Delta\mathbf{d}\right\|^2. \tag{19}$$

The least-squares norm E can be viewed as a function that maps $\boldsymbol{M}$ to $\boldsymbol{R}$ ($E : \boldsymbol{M} \rightarrow \boldsymbol{R}$), where $\boldsymbol{M}$ is a set of all geologically acceptable velocity models and $\boldsymbol{R}$ is a set of all residuals. The model $\mathbf{m}$ can be written as a sum of a known reference model $\mathbf{m}_0$ and a perturbation model $\delta\mathbf{m}$ ($\mathbf{m}_{n+1} = \mathbf{m}_n + \delta\mathbf{m}_n$). In FWI we are trying to find a model update $\delta\mathbf{m}_n$ so that $\mathbf{m}_{n+1}$ satisfies

$$\arg\min_{\mathbf{m}\in\boldsymbol{M}} E(\mathbf{m}_{n+1}) := \{\mathbf{m} \,|\, \mathbf{m} \in \boldsymbol{M} \ni \forall\mathbf{s} \in \boldsymbol{M} : E(\mathbf{m}) \leq E(\mathbf{s})\}. \tag{20}$$

The misfit function (equation 19), in the vicinity of the model $\mathbf{m}_n$, can approximated, using Taylor expansion, as

$$E(\mathbf{m}_n + \delta\mathbf{m}_n) \approx E(\mathbf{m}_n) + \frac{\partial E(\mathbf{m}_n)}{\partial\mathbf{m}_n}\delta\mathbf{m}_n \tag{21}$$

taking the derivative with respect to the model parameter $\mathbf{m}$ yields

$$\frac{\partial E(\mathbf{m}_n + \delta\mathbf{m}_n)}{\partial\mathbf{m}_n} \approx \frac{\partial E(\mathbf{m}_n)}{\partial\mathbf{m}_n} + \frac{\partial^2 E(\mathbf{m}_n)}{\partial\mathbf{m}_n^2}\delta\mathbf{m}_n. \tag{22}$$

The minimum of the misfit function $\boldsymbol{E}(\mathbf{m}_n)$ is attained when $\frac{\partial\boldsymbol{E}(\mathbf{m}_n)}{\partial\mathbf{m}_n} = 0$; hence, the perturbation model is

$$\delta\mathbf{m}_n = -\left[\frac{\partial^2 E(\mathbf{m}_n)}{\partial\mathbf{m}_n^2}\right]^{-1}\frac{\partial E(\mathbf{m}_n)}{\partial\mathbf{m}_n} \tag{23}$$

where

$$\mathbf{g} = \frac{\partial E(\mathbf{m}_n)}{\partial\mathbf{m}_n} \tag{24}$$

is the gradient and

$$\boldsymbol{H} = \frac{\partial^2 E(\mathbf{m}_n)}{\partial\mathbf{m}_n^2} \tag{25}$$

is the Hessian[1].

[1]For a detailed derivation of the perturbation model and for an interpretation of the Hessian the reader is referred to Margrave et al. (2011).

To understand the physical interpretation of the gradient (equation 24), we introduce two equations [2]

$$\left[\bigtriangledown^2 + \omega^2 \mathbf{m}_0\right] G_0(\mathbf{r}_g, \mathbf{r}_s, \omega) = \delta(\mathbf{r}_g - \mathbf{r}_s) \tag{26}$$

$$\left[\bigtriangledown^2 + \omega^2 \mathbf{m}\right] G(\mathbf{r}_g, \mathbf{r}_s, \omega) = \delta(\mathbf{r}_g - \mathbf{r}_s) \tag{27}$$

where G_0 is the modeled wavefield, G is the actual wavefield, $\mathbf{m}_0$ is the known reference medium, $\mathbf{m}$ is the actual medium, and $\mathbf{m} = \mathbf{m}_0 + \delta\mathbf{m}$. Also, we redefine the the data residual (equation 18) as

$$\delta G(\mathbf{r}_g, \mathbf{r}_s, \omega|\mathbf{m}_0) = G(\mathbf{r}_g, \mathbf{r}_s, \omega) - G_0(\mathbf{r}_g, \mathbf{r}_s, \omega|\mathbf{m}_0). \tag{28}$$

First, we take the derivative of equation (19) with respect to the model parameters $\mathbf{m}$ (using equation 28 instead of equation 18), which will give us

$$\begin{aligned} \frac{\partial E(\mathbf{m}_0)}{\partial \mathbf{m}_0(\mathbf{r})} &= -\sum_{\mathbf{r}_s,\mathbf{r}_g} \int d\omega\, \Re\left\{ \frac{\partial G_0(\mathbf{r}_s, \mathbf{r}_g, \omega|\mathbf{m}_0)}{\partial \mathbf{m}_0(\mathbf{r})} \delta G^*(\mathbf{r}_s, \mathbf{r}_g, \omega|\mathbf{m}_0) \right\} \\ \Rightarrow \frac{\partial E(\mathbf{m}_0)}{\partial \mathbf{m}_0} &= -\Re\left\{ \frac{\partial \boldsymbol{G}_0^T(\mathbf{r}, \omega|\mathbf{m}_0)}{\partial \mathbf{m}_0} \delta \boldsymbol{G}^*(\mathbf{r}, \omega|\mathbf{m}_0) \right\} = -\Re\left\{ \mathbf{J}^T \delta \boldsymbol{G}^* \right\} \end{aligned} \tag{29}$$

where $\mathbf{J}$ is the sensitivity or Fréchet derivative. In a matrix form equation (29) can be written as

$$\frac{\partial E(\mathbf{m}_0)}{\partial \mathbf{m}_0} = -\Re\left\{ \begin{bmatrix} \frac{\partial u_1}{m_1} & \cdots & \frac{\partial u_k}{m_1} & \cdots & \frac{\partial u_n}{m_1} \\ \frac{\partial u_1}{m_2} & \cdots & \frac{\partial u_k}{m_2} & \cdots & \frac{\partial u_n}{m_2} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial u_1}{m_n} & \cdots & \frac{\partial u_k}{m_n} & \cdots & \frac{\partial u_n}{m_n} \end{bmatrix} \begin{bmatrix} \delta G_1^* \\ \vdots \\ \delta G_k^* \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right\} \tag{30}$$

where the node points are ordered so that the first k columns corresponds to the nodes of the receivers locations. Each column in $\mathbf{J}^T$ describes the change of behavior of the wavefield at a node point when there is a change in the model parameters. For a point diffractor model, the quantity $\Re\left\{\mathbf{J}^T \delta \boldsymbol{G}^*\right\}$ attains its extremum at the location of the point diffractor (Pratt et al., 1998). This corresponds to a perturbation in the wavefield due to a variation between the current velocity model $\mathbf{m}_0$ and the actual velocity $\mathbf{m}$, $(\delta\mathbf{m} \neq 0)$ (Virieux and Operto, 2009); hence, the opposite direction to the gradient is the update direction that should minimize the misfit function.

[2]The derivations of the FWI equations will be carried in the frequency domain. For time domain derivation, the reader is referred to (Yang et al., 2015).

Although, equation (30) indicates that we only need to know the behavior of the wavefield at receivers locations, which reduces equation (30) to

$$\frac{\partial E(\mathbf{m}_0)}{\partial \mathbf{m}} = -\Re\left\{\begin{bmatrix} \frac{\partial u_1}{m_1} & \cdots & \frac{\partial u_k}{m_1} \\ \frac{\partial u_1}{m_2} & \cdots & \frac{\partial u_k}{m_2} \\ \vdots & \ddots & \vdots \\ \frac{\partial u_1}{m_n} & \cdots & \frac{\partial u_k}{m_n} \end{bmatrix} \begin{bmatrix} \delta G_1^* \\ \vdots \\ \delta G_k^* \end{bmatrix}\right\}. \tag{31}$$

The construction of the sensitivity matrix $\mathbf{J}$ in its full or reduced form is extremely expensive (Virieux and Operto, 2009); hence, another approach to compute the gradient is needed.

Using equations (26) and (27), one can compute the gradient with only two forward-modeling problems. First, we substitute $\mathbf{m}_0 + \delta\mathbf{m}$ for $\mathbf{m}$ in equation (27)

$$\left[\bigtriangledown^2 + \omega^2\mathbf{m}_0\right] G(\mathbf{r}_g, \mathbf{r}_s, \omega) = \delta(\mathbf{r}_g - \mathbf{r}_s) - \omega^2\delta\mathbf{m}(\mathbf{r}_g)G(\mathbf{r}_g, \mathbf{r}_s, \omega). \tag{32}$$

The solution of equation (32) in integral form is given by

$$G(\mathbf{r}_g, \mathbf{r}_s, \omega) = G_0(\mathbf{r}_g, \mathbf{r}_s, \omega|\mathbf{m}_0) - \omega^2 \int d\mathbf{r}' G_0(\mathbf{r}_g, \mathbf{r}', \omega|\mathbf{m}_0)\delta\mathbf{m}(\mathbf{r}')G(\mathbf{r}', \mathbf{r}_s, \omega). \tag{33}$$

Using born series to eliminate $G(\mathbf{r}', \mathbf{r}_s, \omega)$ yields

$$\begin{aligned} \delta G(\mathbf{r}_g, \mathbf{r}_s, \omega) &= -\omega^2 \int d\mathbf{r}' G_0(\mathbf{r}_g, \mathbf{r}', \omega)\delta\mathbf{m}(\mathbf{r}')G_0(\mathbf{r}', \mathbf{r}_s, \omega) \\ &+ \omega^4 \int d\mathbf{r}' G_0(\mathbf{r}_g, \mathbf{r}', \omega)\delta\mathbf{m}(\mathbf{r}') \int d\mathbf{r}'' G_0(\mathbf{r}', \mathbf{r}'', \omega)\delta\mathbf{m}(\mathbf{r}'')G_0(\mathbf{r}'', \mathbf{r}_s, \omega) \\ &+ \dots . \end{aligned} \tag{34}$$

Localizing the perturbation model $\delta\mathbf{m}(\mathbf{r})$ by introducing the delta function so that

$$\delta\mathbf{m}(\mathbf{r}') = \delta m\, \delta(\mathbf{r}' - \mathbf{r}), \tag{35}$$

and by substituting equation (35) into equation (34), we obtain

$$\begin{aligned} \delta G(\mathbf{r}_g, \mathbf{r}_s, \omega) &= -\omega^2 G(\mathbf{r}_g, \mathbf{r}, \omega)\delta m\, G(\mathbf{r}, \mathbf{r}_s, \omega) \\ &+ \omega^4 G(\mathbf{r}_g, \mathbf{r}, \omega)\delta m\, G(\mathbf{r}, \mathbf{r}, \omega)\delta m\, G(\mathbf{r}, \mathbf{r}_s, \omega) + \dots \\ &= -\omega^2 G_0(\mathbf{r}_g, \mathbf{r}, \omega)\delta m\, G(\mathbf{r}, \mathbf{r}_s, \omega)[1 - \omega^2\delta m\, G(\mathbf{r}, \mathbf{r}, \omega) + \dots]\,. \end{aligned} \tag{36}$$

Noting that the series expansion of $\frac{1}{1+x}$ is $(1 - x + x^2 - \ldots)$, equation (36) can be re-written as

$$\delta G(\mathbf{r}_g, \mathbf{r}_s, \omega) = -\frac{\omega^2 G(\mathbf{r}_g, \mathbf{r}, \omega)\delta m\, G(\mathbf{r}, \mathbf{r}_s, \omega)}{1 + \omega^2 \delta m\, G(\mathbf{r}, \mathbf{r}, \omega)}$$
$$\Rightarrow \frac{\delta G(\mathbf{r}_g, \mathbf{r}_s, \omega)}{\delta m} = -\frac{\omega^2 G(\mathbf{r}_g, \mathbf{r}, \omega) G(\mathbf{r}, \mathbf{r}_s, \omega)}{1 + \omega^2 \delta m\, G(\mathbf{r}, \mathbf{r}, \omega)}. \tag{37}$$

By taking the limit so that δm goes to zero we obtain

$$\frac{\partial G(\mathbf{r}_g, \mathbf{r}_s, \omega|\mathbf{m}_0)}{\partial \mathbf{m}_0(\mathbf{r})} = \lim_{\delta m \to 0} \frac{\delta G(\mathbf{r}_g, \mathbf{r}_s, \omega|\mathbf{m}_0)}{\delta m} = -\omega^2 G(\mathbf{r}_g, \mathbf{r}, \omega|\mathbf{m}_0) G(\mathbf{r}, \mathbf{r}_s, \omega|\mathbf{m}_0), \tag{38}$$

and substituting equation (38) into equation (29) gives us

$$g(\mathbf{r}) = \sum_{\mathbf{r}_s, \mathbf{r}_g} \int d\omega\, \omega^2\, G(\mathbf{r}, \mathbf{r}_s, \omega|\mathbf{m}_0) \times G(\mathbf{r}_g, \mathbf{r}, \omega|\mathbf{m}_0) \delta G^*(\mathbf{r}_s, \mathbf{r}_g, \omega) \tag{39}$$

where $G(\mathbf{r}, \mathbf{r}_s, \omega|\mathbf{m}_0)$ is the source wavefield forward-propagated in medium $\mathbf{m}_0$ and $G(\mathbf{r}_g, \mathbf{r}, \omega|\mathbf{m}_0)\delta G^*(\mathbf{r}_s, \mathbf{r}_g, \omega)$ is the back-propagated data residual in medium $\mathbf{m}_0$. Equation (39) is the gradient of the misfit function in frequency domain, where the model parameter $\mathbf{m}(\mathbf{r})$ is parametrized in terms of slowness squared $(\frac{1}{\mathbf{v}(\mathbf{r})^2})$.

The inverse of the hessian $\boldsymbol{H}$ can be replaced by a positive scalar α in equation (23) which gives us

$$\delta \mathbf{m} = -\alpha \frac{\partial E(\mathbf{m}_0)}{\partial \mathbf{m}_0} \tag{40}$$

that results in solving the FWI problem using a gradient method (Virieux and Operto, 2009).

The corresponding equation in time domain to equation (39) is given by

$$g(\mathbf{r}) = \frac{-2}{\boldsymbol{v}^3} \sum_{\mathbf{r}_s, \mathbf{r}_g} \int_0^T dt\, \frac{\partial^2 G(\mathbf{r}, \mathbf{r}_s, t)}{\partial t^2} \big[G(\mathbf{r}_g, \mathbf{r}, T - t) * \delta G(\mathbf{r}_s, \mathbf{r}_g, t)\big] \tag{41}$$

where $\frac{\partial^2 G(\mathbf{r}, \mathbf{r}_s, t)}{\partial t^2}$ is the second partial derivative in time of the forward-propagated source wavefield, $G(\mathbf{r}_g, \mathbf{r}, T - t) * \delta G(\mathbf{r}_s, \mathbf{r}_g, t)$ is the time-reversed back-propagated data residual wavefield, and $*$ denotes convolution. In equation (41) the parametrization has been changed, so that the model parameter $\mathbf{m}(\mathbf{r})$ is parametrized in terms of the velocity $(\mathbf{v}(\mathbf{r}))$, by applying the chain rule $\big[d\mathbf{E}/d\mathbf{v} = (d\mathbf{E}/d\frac{1}{\mathbf{v}^2})(d\frac{1}{\mathbf{v}^2}/d\mathbf{v}) = (\frac{-2}{\mathbf{v}^3}) d\mathbf{E}/d\mathbf{v})\big]$.

Algorithm 1: FWI - Gradient method

Initialization :

$\mathbf{d}_{obs}$, $\mathbf{m}_0$, num_iter;

Computation:

for *i = 0* **to** *num_iter* **do**

$compute$:

`source wavefield` $\frac{\partial^2 G(\mathbf{r},t;\mathbf{r}_s,\mathbf{m}_i)}{\partial t^2}$;

`modeled data` $\mathbf{d}_{cal}(\mathbf{m}_i)$;

`data residual` $\Delta\mathbf{d}$;

`data residual wavefield` $\delta G(\mathbf{r}, t; \mathbf{r}_s, \mathbf{m}_i)$;

`gradient` $\nabla E(\mathbf{m}_i)$;

$update\, model$:

`find` α `that minimizes` $E(\mathbf{m}_i - \alpha\nabla E)$;

$\mathbf{m}_{i+1} = \mathbf{m}_i - \alpha\nabla E$;

end

LINE SEARCH

The gradient (equation 24) relate changes in the data residual to variations in the velocity model, which is a relative quantity that needs to be scaled (Basker et al., 2016). Ideally, the Hessian (equation 25) is the appropriate scaling multiplier (Margrave et al., 2011); however, it is computationally expensive to compute and can be replaced with a scalar α for a single-parameter inversion.

In the process of line search, the goal is to find a step-length α that satisfies

$$E(\mathbf{m}_n + \alpha_n \mathbf{p}_n) < E(\mathbf{m}_n) \tag{42}$$

where $\mathbf{p}_n = -\nabla E(\mathbf{m}_n)$. Equation (42) imposes a simple reduction, however, such criterion is insufficient (Nocedal and Wright, 2006). An optimum α value is one that satisfies

$$\arg\min_{\alpha\in\mathbb{R}_{\geq 0}} E(\mathbf{m}+\alpha\mathbf{p}) := \{\alpha \,|\, \alpha \in \mathbb{R}_{\geq 0} \ni \forall\beta \in \mathbb{R}_{\geq 0} : E(\mathbf{m}+\alpha\mathbf{p}) \leq E(\mathbf{m}+\beta\mathbf{p})\} \tag{43}$$

A sufficient decrease can be achieved by using Armijo condition (equation 44), instead of equation (42)

$$E(\mathbf{m}_n + \alpha_n \mathbf{p}_n) \leq E(\mathbf{m}_n) + c\,\alpha_n \nabla E_n^T \mathbf{p}_n \tag{44}$$

where $c \in (0, 1)$.

The Armijo condition can be implemented using a backtracking approach (Algorithm 2). In this method, an initial value of α is set to a large positive number ($\alpha = \bar{\alpha} \,|\, \bar{\alpha} > 0$) and will be checked whether or not it satisfies Armijo condition. If α meets the Armijo condition, then it has the appropriate value, and if it does not satisfy the condition, then α need to be decreased by a factor ρ ($\alpha \leftarrow \rho\alpha$). The advantage of the backtracking approach is that it ensures that the step length is either a fixed value ($\alpha = \bar{\alpha}$) or that it is sufficiently small to satisfy the Armijo condition, but not too small that it requires

many iterations for the problem to converge ($\bar{\alpha} > \alpha > \epsilon$) (Nocedal and Wright, 2006).

Algorithm 2: Backtracking line search

Initialization :

choose $\bar{\alpha} > 0, \rho \in (0,1), c \in (0,1)$;

$\alpha \leftarrow \bar{\alpha}$;

Computation:

while $E(\mathbf{m}_n + \alpha_n \mathbf{p}_n) > E(\mathbf{m}_n) + c\,\alpha_n \nabla E_n^T \mathbf{p}_n$ **do**

$\alpha \leftarrow \rho\alpha$;

end

$\alpha_n \leftarrow \alpha$;

OPTIMIZATION

In this section, we present a set of techniques to optimize the FWI problem solving, along with the necessary tools for a python implementation of the problem. First, we will point some of the freely available tools to solve the FWI problem. Afterward, we will discuss the optimization techniques.

Tools

Python

Python is an open-source, general-purpose, high-level programming language. It has a large number of specialized libraries, with a wide number of developers. Also, Python takes less time to develop compared to other languages. As a result, it is an ideal language to solve sophisticated and computationally-intensive problems for both educational purposes and industry usage.

Spyder

Spyder is an integrated development environment (IDE) for scientific programming in Python. Important libraries for scientific computing such as Numpy, Scipy, and Matplotlib are integrated within Spyder. Moreover, IPython, an interactive command shell, which enables a UNIX-based commands and Python scripts to be executed at the same place, and much more other features, is included in Spyder. All of which makes Spyder an attractive environment for scientific programming.

Joblib

Joblib is an open-source optimization library used to speed up execution of programs through processes such as memoization and/or parallelization. This library has an advantage of ease of parallelizing codes.

Memory management

FWI is a computationally expensive problem to solve. A major issue is memory requirement, as typically a significant amount of memory is required to store wavefields to compute the gradient. For instance, calculating a single wavefield of a model that consists of 501×301 grid points, a recording time of $4.5\,s\,(t_{max} = 4.5\,s)$, and sampling interval of $1\,ms\,(dt = 0.001\,s)$; the memory required is 5.056 GB. For a single shot in FWI, two wavefields are needed to compute the gradient. Also, for a parallel implementation of FWI; the memory required to solve the problem is tremendous. For the same model, utilizing ten cores to solve the problem would require 101.12 GB of memory. This might not be an issue when solving a single-parameter 2D acoustic FWI problem on a cluster, however, moving into a more complicated, FWI-based, problems such as reflection-based waveform inversion (RWI) or elastic FWI, the memory requirement increases drastically. As a result, memory management is a vital aspect in a successful parallel FWI implementation.

In here, we suggest an alternative approach to compute the wavefield by slicing it into sub-wavefields and saving every sub-wavefield to disk separately (Figure 2). In this way, the complete wavefield need not be stored in memory, but rather only a part of it need to be in memory and saved to disk before computing another segment of the wavefield. This method does not exhaust the memory and is faster than saving every time-step of the wavefield to disk separately, which is the least memory-exhaustive approach to follow. Algorithm (2) shows a method to compute time-steps for the sub-wavefields.

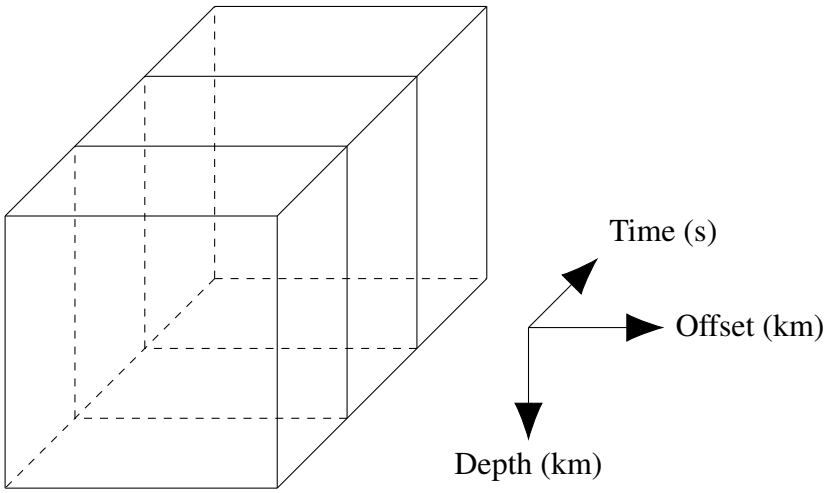

FIG. 2. Wavefield cube.

Algorithm 3: Sub-wavefields computation

Initialization :

```
num_cores:  number of cores to be used;
mem_use  :  memory to be used;
nx       :  number of x − axis grid points;
nz       :  number of z − axis grid points;
nt       :  number of time steps;
```

Computation:

```
    // sub_nt  :  maximum time-steps for sub-wavefields
sub_nt   = [mem_use]/[(8)(num_cores)(nx)(nz)];
    // rem_proc:  time-steps for the last sub-wavefields
rem_proc = nt%sub_proc;
    // num_proc:  number of sub-wavefields
num_proc = [(nt-rem_proc)/sub_proc] + 1;
    // num_iter:  array of length equal to number of sub-wavefields
    //           with values corresponding to the time-steps for
    //           each sub-wavefield
```

for *i=1:(num_proc-1)* **do**

```
 | num_iter[i] = [sub_nt];
```

end

```
num_iter[num_proc] = [rem_proc];
```

Parallelization

In this section we show a simple example on how to use *joblib* package to parallelize a task (code 1), where the parallelization is carried by distributing sources over processors. In here, we use a function, *compute_observed*, that takes a set of parameters as an argument and generate a shot gather. From *joblib* we utilize two functions, *Parallel* and *delayed*. *Parallel* takes a keyword *n_jobs* to specify the number of cores to be used. The function *delayed* is decorator that separate a function from its passed arguments. This decorator takes a function and its arguments as an argument and returns the function and its arguments as a tuple, (*function*, *arguments*).

```
1  from joblib import Parallel, delayed
2
3  # Parallel: parallelization function from joblib
4  # delayed: decorator function from joblib
5  # xs: shots lateral postions
6  # i: index of shot (shot_number - 1)
7  # k: source lateral position (node position)
8  # zs: source depth
9  # xr: receivers lateral postions
10 # zr: receivers depths
11 # vel: velocity model
12 # density: density model
13 # nx: number of grid points along the x-axis
14 # nz: number of grid points along the z-axis
15 # tmax: maximum recording time
16 # src: source function (1-D array)
17 # dt: time step interval
```

```
# dx: grid points spacing
# compute_observed: a function that computes a single shot
# observed_data: if "compute_observed" returns somtheing it will be
    stored in compute_observed
observed_data = Parallel(n_jobs=num_cores)(delayed(compute_observed)(i,
    k, zs, xr, zr, vel, density, nx, nz, tmax, src, dt, dx) for i, k in
    enumerate(xs))
```

Code 1. An example of a parallelized code.

CONCLUSION

This tutorial briefly reviews the numerical aspects of wave equation modeling, using the finite difference method, and the concept of FWI. Also, simple optimization techniques to overcome some computational obstacles are presented. The motivation of this report is to provide the fundamental tools for an FWI implementation.

APPENDIX A: FINITE-DIFFERENCE ENGINE

The finite-difference approximation is the simplest and most widely used method to solve partial differential equations numerically. Numerical solutions provide an environment that helps in understanding physical phenomena, such as wave propagation in different media, as exact analytical solutions are usually unavailable. In Geophysics, the wave equation modeling is the fundamental block of processes such as reverse-time migration, and seismic modeling and inversion; hence, a proper implementation is vital. In this appendix, we present our implementation of the finite-difference engine in Python (code 2).

```
# import a copy function
from copy import deepcopy
# import numpy library
import numpy as np
# import boundary conditions function
from apply_boundary_condition import apply_boundary_condition

def d2x_d2z(u, p, dx):
    """
        This function computes [d^2u/x^2 + d^2u/z^2] (laplacian
    operator)

        p: density
    """
    a = (u[1:-1,2:]+u[1:-1,:-2]+u[2:,1:-1]+u[:-2,1:-1]-4.0*u
    [1:-1,1:-1])/(dx**2)
    b = (p[1:-1,2:]-p[1:-1,:-2])*(u[1:-1,2:]-u[1:-1,:-2])/(4.0*(dx**2)*
    p[1:-1,1:-1])
    c = (p[2:,1:-1]-p[:-2,1:-1])*(u[2:,1:-1]-u[:-2,1:-1])/(4.0*(dx**2)*
    p[1:-1,1:-1])
    d2 = a-b-c
    return d2

####

def wavefield(velocity, density, u0, u, u1, it, source, xs, zs, xr, zr,
    dx, dt, derivative=False):
```

```
    """
        This function computes the wavefield
    """
    # compute laplacian operator
    d = d2x_d2z(u, density, dx)

    # compute wave field at time t = n+1
    u1[1:-1,1:-1] = d * (dt**2) * (velocity[1:-1,1:-1]**2) + 2.0 * u
   [1:-1,1:-1] - u0[1:-1,1:-1]

    # inject source
    if len(source.shape) == 1:
        u1[zs, xs] = u1[zs, xs] + source[it] * (dt**2)
    elif len(source.shape) == 2:
        u1[zr, xr] = u1[zr, xr] + source[it,:] * (dt**2)

    # return wavefield and its second time derivative
    if derivative == 1:
        return u1, (u0-2.0*u+u1)/(dt**2)

   # return wavefield
    return u1

####

def afd(itr, dx, velocity, density, snap1, snap2, tmax, source, xs, zs,
    boundary_condition, dt, xr=[], zr=[], option=0, derivative=False):
    """
        This function manages the acoustic finite-difference scheme

        itr: number of time-steps to be computed
        dx: grid points spacing
        snap1: pressure field at time t = n-1   (past time)
        snap2: pressure field at time t = n     (current time)

        source: source array (1-D for a wavelet - 2-D for shot back-
    propagation)
        xs: x node of source
        zs: z node of source

        dt = time step

        xr: array of x nodes of receivers
        zr: array of z nodes of receivers

        option = 0: compute synthetic data and wavefield cube
        option = 1: compute synthetic data only
        option = -1: compute wavefield cube only

        derivative: to compute second time derivative of wavefield
    """
    # get model dimensions
    (nz,nx) = velocity.shape

    # pressure fields
    #### future (t=n+1) pressure field
    u1 = np.zeros((nz,nx))
    ##### current (t=n) pressure field
    u = deepcopy(snap2)
```

```
    #### past (t=n-1) pressure field
    u0 = deepcopy(snap1)

    # memory allocation for synthetic data - option = 1 or 0
    if option == 1 or option == 0:
        acq = np.zeros((itr, len(xr)))

    # memory allocation for wavefield - option = 1 and derivative = False
    if option != 1 and not derivative:
        out_u = np.zeros((itr, nz-2, nx-2))

    # memory allocation for wavefield - option = 1 and derivative = True
    if option != 1 and derivative:
        out_u = np.zeros((itr, nz-2, nx-2))
        out_u2 = np.zeros((itr, nz-2, nx-2))

    # computation that include wavefield
    if option != 1:
        for it in range(itr):
            if not derivative:
                # compute wavefield
                u1 = wavefield(velocity, density, u0, u, u1, it, source, xs, zs, xr, zr, dx, dt)
            else:
                # compute wavefield and its second time derivative
                u1, u2 = wavefield(velocity, density, u0, u, u1, it, source, xs, zs, xr, zr, dx, dt, True)

            # apply boundary conditions
            u1 = apply_boundary_condition(u1, u0, u, velocity, dx, dt, boundary_condition)

            # update wavefields
            u0 = deepcopy(u)
            u = deepcopy(u1)

            # store synthetic data
            if option != -1:
                acq[it, :]=u[zr, xr]

            # store wavefield
            if option != 1 and not derivative:
                out_u[it] = np.array([deepcopy(u1[1:-1,1:-1])])

            # store wavefield and its second time derivative
            if option != 1 and derivative:
                # in this statement out_u2 if d^2u/dt^2 and out_u is u
                out_u[it] = np.array([deepcopy(u1[1:-1,1:-1])])
                out_u2[it] = np.array([deepcopy(u2[1:-1,1:-1])])

    # computation that does not include wavefield
    else:
        for it in range(itr):
            # compute wavefield
            u1 = wavefield(velocity, density, u0, u, u1, it, source, xs, zs, xr, zr, dx, dt)

```

```
            # apply boundary conditions
            u1 = apply_boundary_condition(u1, u0, u, velocity, dx, dt,
    boundary_condition)

            # update wavefields
            u0 = deepcopy(u)
            u = deepcopy(u1)

            # store synthetic data
            acq[it, :]=u[zr, xr]

    # return computed data
    if derivative and option != 1:
        if option == 0:
            return acq, out_u2, out_u, u0, u
        elif option == -1:
            return out_u2, out_u, u0, u
        else:
            return None

    elif not derivative and option != 1:
        if option == 0:
            return acq, out_u, u0, u
        elif option == -1:
            return out_u, u0, u
        else:
            return None
    else:
        return acq
```

Code 2. Finite-difference engine

ACKNOWLEDGEMENTS

We thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors and NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 461179-13.

REFERENCES

Alford, R., Kelly, K., and Boore, D. M., 1974, Accuracy of finite-difference modeling of the acoustic wave equation: Geophysics, **39**, No. 6, 834–842.

Basker, B., Rüger, A., Deng, L., and Jaramillo, H., 2016, Practical considerations and quality control for an FWI workflow: The Leading Edge, **35**, No. 2, 151–156.

Claerbout, J. F., 1985, Imaging the earth's interior: Blackwell scientific publications Oxford.

Clayton, R., and Engquist, B., 1977, Absorbing boundary conditions for acoustic and elastic wave equations: Bulletin of the Seismological Society of America, **67**, No. 6, 1529–1540.

Engquist, B., and Majda, A., 1977, Absorbing boundary conditions for numerical simulation of waves: Proceedings of the National Academy of Sciences, **74**, No. 5, 1765–1766.

Lailly, P., 1983, The seismic inverse problem as a sequence of before stack migrations, *in* Conference on Inverse Scattering: Theory and Application, Society of Industrial and Applied Mathematics, 206–220.

Lines, L. R., Slawinski, R., and Bording, R. P., 1999, A recipe for stability of finite-difference wave-equation computations: Geophysics, **64**, No. 3, 967–969.

Margrave, G., Yedlin, M., and Innanen, K., 2011, Full waveform inversion and the inverse hessian: CREWES Annual Report, **23**.

Nocedal, J., and Wright, S., 2006, Numerical optimization, Springer Series in Operations Research and Financial Engineering: Springer-Verlag New York, second edn.

Pan, W., and Innanen, K., 2015, Full-waveform inversion in the frequency-ray parameter domain: CREWES Annual Report, **27**.

Pratt, R. G., Shin, C., and Hick, G., 1998, Gauss-newton and full newton methods in frequency-space seismic waveform inversion: Geophysical Journal International, **133**, No. 2, 341–362.

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: Geophysics, **49**, No. 8, 1259–1266.

Virieux, J., and Operto, S., 2009, An overview of full-waveform inversion in exploration geophysics: Geophysics, **74**, No. 6, WCC127–WCC152.

Yang, P., Gao, J., and Wang, B., 2015, A graphics processing unit implementation of time-domain full-waveform inversion: Geophysics, **80**, No. 3, F31–F39.