Earth and Planetary Physics
6: 487–494, 2022
doi: 10.26464/epp2022044

mFAST: A MATLAB toolbox for ocean bottom seismometer refraction first-arrival traveltime tomography

Bin Liu[1,2*]

[1]Southern Marine Science and Engineering Guangdong Laboratory (Guangzhou), Guangzhou 511458, China;

[2]Guangzhou Marine Geological Survey, Guangzhou 511458, China

Key Points:

- We develop an open-source MATLAB toolbox for first-arrival traveltime tomography.
- We implement the traveltime tomography as an efficient linearized inversion.
- We validate the code by using ocean bottom seismometer (OBS) seismic survey data.

Citation: Liu, B. (2022). mFAST: A MATLAB toolbox for ocean bottom seismometer refraction first-arrival traveltime tomography. *Earth Planet. Phys.*, *6*(5), 487–494. http://doi.org/10.26464/epp2022044

Abstract: First-arrival seismic traveltime tomography (FAST) is a well-established technique to estimate subsurface velocity structures. Although several existing open-source packages are available for first-arrival traveltime tomography, most were written in compiled languages and lack sufficient extendibility for new algorithms and functionalities. In this work, we develop an open-source, self-contained FAST package based on MATLAB, one of the most popular interpreted scientific programming languages, with a focus on ocean bottom seismometer refraction traveltime tomography. Our package contains a complete traveltime tomography workflow, including ray-tracing-based first-arrival traveltime computation, linearized inversion, quality control, and high-quality visualization. We design the package as a modular toolbox, making it convenient to integrate new algorithms and functionalities as needed. At the current stage, our package is most efficient for performing FAST for two-dimensional ocean bottom seismometer surveys. We demonstrate the efficacy and accuracy of our package by using a synthetic data example based on a modified Marmousi model.

Keywords: ocean bottom seismometer (OBS); first-arrival traveltime tomography; open source; MATLAB

1. Introduction

First-arrival seismic traveltime tomography (FAST) is a classical method of deriving subsurface velocity models by using the first-arrival traveltime information of seismic wavefields (Aki and Lee, 1976; White, 1989; Hole et al., 1992; Korenaga et al., 1997; Zelt and Barton, 1998; Zhang XY et al., 2018; Guo B et al., 2019). Unlike more modern tomography methods, such as full-waveform inversion, FAST is relatively easy to implement. More important, identification of the first-arrival traveltime from a waveform is, in most cases, straightforward and error-proof because first-arrival waveforms usually come with a strong amplitude, particularly for ocean seismic surveys.

Several open-source packages have been developed to realize FAST. White (1989) used a two-point ray-tracing algorithm and solved a damped least-squares problem during the implementation of FAST. Zelt and Smith (1992) developed one of the most widely used FAST packages, rayinvr. Other widely used refraction traveltime tomography packages include FAST (Zelt and Barton, 1998), tomo2d and tomo3d (Korenaga et al., 1997; Meléndez et al., 2015, 2019), jive3D (Hobro et al., 2003), and Profit (Koulakov et al., 2010). These packages adopt different strategies in model parameterization, ray-tracing algorithms, preconditioning, and regularization. Numerous synthetic and field data examples have demonstrated the efficacy and accuracy of these packages.

However, through extensive validations and comparisons, we find that these existing packages have some notable issues. First, most of the packages were developed by using compiled programming languages, such as C, C++, or Fortran; thus, code extendibility can be a nontrivial issue. Second, most of these packages adopt sophisticated strategies and algorithms to mitigate the need to solve a large, sparse linear system. Although these strategies work for the defined problem, they may further contribute to the code extendibility issue. Third, because of the compiled-language nature of these packages, they seldom come with internal visualization and quality control functionalities, which are sometimes crucial for preparing, performing, and interpreting a traveltime-based inversion. Modern FAST packages are emerging, such as Refrapy, a Python package for seismic refraction data analysis (Guedes et al., 2022). Refrapy is a functional package but is not self-contained, as it relies on a third-party tool, pyGIMLi (Rücker et al., 2017), to perform the most critical part, the traveltime inversion.

In this work, we develop a self-contained, open-source, MATLAB-based package to achieve FAST. We choose MATLAB for several reasons: (1) MATLAB is a high-level, interpreted scientific

Correspondence to: B. Liu, liugele@163.com
Received 10 APR 2022; Accepted 15 JUN 2022.
Accepted article online 11 AUG 2022.

programming language and is therefore convenient to tune and extend; (2) MATLAB provides native, high-quality visualization functionalities that can be conveniently integrated with computationally intensive routines; and (3) MATLAB is one of the most popular scientific programming languages and has a vast global community for knowledge sharing. MATLAB has proved to be a highly effective and efficient tool in Earth sciences (e.g., Marcotte, 1991; Tian D et al., 1993; Yeung and Chakrabarty, 1993). In seismic exploration, MATLAB has also been widely adopted to achieve complex functionalities, such as DSISoft for vertical source profiling data (Beaty et al., 2002), and SWIP for Love- and Rayleigh-wave inversions based on land seismic data (e.g., Pasquet and Bodet, 2017; Haney and Tsai, 2017, 2020), to name two.

At the current stage, our code is made particularly suitable for ocean bottom seismometer (OBS) surveys (see Figure 1). This choice is made in light of two important considerations. First, OBS surveys are becoming a common practice in the study of deep Earth structures beneath the ocean. Second, OBS is usually deployed with a large node spacing; therefore, the associated computational cost is substantially smaller compared with that of dense geometry. Therefore, it is computationally feasible to perform FAST for an OBS survey within MATLAB even without sophisticated parallelism. For other types of observation geometry, such as that for land near-surface imaging, the computational cost when using our mFAST (and in fact any ray-tracing-based FAST) can be high, and a more efficient tomography algorithm should be used.

2. Methodology

We briefly describe the algorithm of FAST associated with our implementation. Our mFAST largely implements and improves the FAST algorithm developed by Aldridge and Oldenburg (1993).

FAST uses first-arrival traveltime information to estimate a subsurface velocity model. The first-arrival traveltime for a source–receiver pair can be computed by

$$t^i = \int_{L_i} s(x,z)\mathrm{d}s, \tag{1}$$

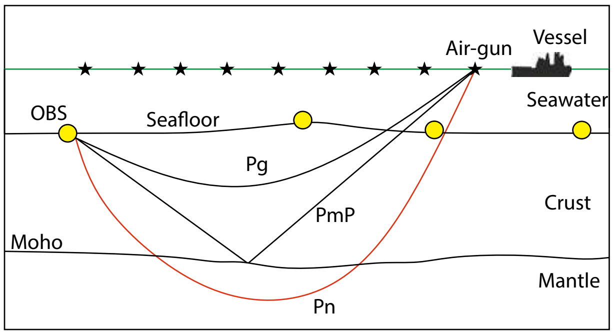

Figure 1. Schematic diagram of a typical ocean bottom seismometer (OBS) survey for studying deep Earth structures. PmP denotes the reflections from the Moho boundary. Pn and Pg are refraction waves from the upper mantle and crust, respectively. Phases Pg and Pn usually dominate the first-arrival traveltimes, but depending on the geometry and the actual geological model, the PmP phase may also appear in the first-arrival traveltimes. Note that in FAST, we consider only first-arrival traveltimes.

where $s(x,z)$ is the spatially varying slowness of a subsurface model, index i denotes a specific source–receiver pair, and L_i is the ray path associated with this source–receiver pair. Parameter L_i depends on both the source and receiver positions and the slowness model.

FAST is a nonlinear inversion problem, which is usually hindered by nonuniqueness and ill-posedness owing to insufficient data coverage and an inaccurate initial model. To mitigate this issue, many strategies have been proposed. The most common approach is the so-called local linearization and iteration (Aldridge and Oldenburg 1993; Korenaga et al., 1997; Zelt and Barton, 1998; Koulakov et al., 2010), which linearizes the nonlinear optimization in Equation (1) with respect to an initial model. The nonlinear optimization problem is then transformed into solving a large, sparse linear system associated with a perturbation with respect to the initial model. An optimal model is then obtained iteratively until the difference between the observed first-arrival traveltimes and the synthetic traveltimes is sufficiently small. Some studies have solved the problem in a completely nonlinear way, such as that based on the nonlinear conjugate gradient method (Zhang J et al., 1998). Taillandier et al. (2009) developed an adjoint method to solve FAST based on the eikonal equation and its associated adjoint-state equation, which is particularly efficient for dense observations. Recently, with the rapid improvements in computing power, a Bayesian-based method (Ryberg and Haberland, 2018) and a machine-learning-based method (Guo R et al., 2019) were developed to tackle the nonlinear FAST problem.

Our package adopts a linearized strategy to solve FAST. Our inversion begins from a smooth initial slowness model, and it generates ray paths and the associated first-arrival traveltimes for each source–receiver pair in the observation system via

$$t_0^i = \int_{L_i^0} s_0(x,z)\mathrm{d}s. \tag{2}$$

Assuming that the ray path calculated in the initial model is close to the real ray path, we relate the traveltime difference to the slowness perturbation by

$$\begin{aligned}\Delta t^i = t^i - t_0^i &= \int_{L_i} s(x,z)\mathrm{d}s - \int_{L_i^0} s_0(x,z)\mathrm{d}s\\ &\approx \int_{L_i^0} s(x,z)\mathrm{d}s - \int_{L_i^0} s_0(x,z)\mathrm{d}s\\ &= \int_{L_i^0} s(x,z) - s_0(x,z)\mathrm{d}s\\ &= \int_{L_i^0} \Delta s(x,z)\mathrm{d}s.\end{aligned} \tag{3}$$

By numerically calculating the integration, we transform FAST into a large sparse linear system

$$\Delta t^i = \sum_j^N L_{i,j}\Delta s_j, \tag{4}$$

which is then solved by minimizing the least-squares misfit function:

$$J(\Delta s) = \frac{1}{2}\|\boldsymbol{L}\Delta s - \Delta t\|^2. \tag{5}$$

In most cases, some type of smoothing or regularization is necessary to stabilize the inversion and to avoid artifacts in the resulting slowness perturbation. The optimization problem of FAST can then be written as

$$J(\Delta s) = \frac{1}{2}||\boldsymbol{L}\Delta s - \Delta t||^2 + \mu||\boldsymbol{R}\Delta s||^2 + \lambda||\boldsymbol{A}\Delta s||^2, \tag{6}$$

where μ, λ are hyperparameters, $\boldsymbol{R}$ is a regularization matrix, and $\boldsymbol{A}$ is a smoothing matrix. The new functional in Equation (6) corresponds to an augmented system of linear equations that can be written compactly as

$$\begin{Bmatrix} \boldsymbol{L} \\ \mu\boldsymbol{R} \\ \lambda\boldsymbol{A} \end{Bmatrix} \Delta s = \begin{Bmatrix} \Delta t \\ 0 \\ 0 \end{Bmatrix}. \tag{7}$$

The subsurface slowness model can then be updated by simply adding the slowness perturbation to the initial model $s_0(x, z)$. Theoretically, an optimal model can be obtained with only a single inversion. In practice, however, several iterations may be needed to produce an accurate result because of the sparsity of the observation system and traveltime pick errors.

3. Code Implementation

Our mFAST consists of several parts: (1) the data structure, (2) model parameterization, (3) the ray-tracing algorithm, (4) construction of the Jacobian matrix, (5) regularization and smoothing, and (6) solving the associated large, sparse linear system.

3.1 Data Structure

Developing a toolbox usually requires one or several suitable data structures. MATLAB can handle external files in many formats, while internally, MATLAB can work with scalars, vectors, multidimensional matrices, cell arrays, and structured variables. The data involved in a typical FAST may include the traveltime table, the initial velocity (or slowness) model, ray paths, and one or several hyperparameters. We use different data types to store different types of data involved in our mFAST.

The current version of our mFAST requires a traveltime table provided by the user in ASCII format. The traveltime table should contain the source positions in meters, the receiver positions in meters, and the observed traveltimes in seconds. See Table 1 for an example of such a traveltime table. The values in the five columns are the *x* and *z* coordinates of the source point, the *x* and *z* coordinates of the OBS point, and the first-arrival traveltime, respectively.

We load such a traveltime table into MATLAB and use a structure array to store all the information. The structure has multiple fields, including *obs_x*, *obs_z*, *shots_x*, *shots_z*, *t_obs*, *num_picks*, and *record_num*. We describe the meaning of these fields in Table 2.

The initial model can be either imported from an external file or specified internally within the MATLAB programming environment. The initial model is specified by the lateral and vertical coordinates, along with an array representing velocity values at these spatial points. All this information is stored in a structure that has multiple fields, including *x*, *z*, *nx*, *nz*, and *vel*. We also include the seafloor depth as an independent field in this structure.

Efficiently manipulating ray paths is critical to FAST. Whereas most of the existing FAST packages store ray paths in an external ASCII file (e.g., Koulakov et al., 2010), we use a structure array to store all the ray paths to avoid intensive and thus slow inputs/outputs (I/O), as described in Table 3. Each ray path contains the information regarding the source–receiver pair, the computed traveltime, the observed traveltime, and the spatial coordinates of the ray-path points, as well as an index indicating whether the ray is successfully traced.

Table 1. An example traveltime table as the input to our mFAST.

Ray	*x* coordinate of source (m)	*z* coordinate of source (m)	*x* coordinate of OBS (m)	*z* coordinate of OBS (m)	Picked travel time (s)
1	1,000	10	1,000	1,500	0.9270
2	1,500	10	1,000	1,500	0.9138
3	2,000	10	1,000	1,500	0.9011
4	2,500	10	1,000	1,500	0.8890
5	3,000	10	1,000	1,500	0.8775
6	3,500	10	1,000	1,500	0.8666
7	4,000	10	1,000	1,500	0.8564
8	4,500	10	1,000	1,500	0.8469

Table 2. Fields and their associated physical meanings in a mFAST structure array for storing the geometry and traveltime information.

Field name	Description
obs_x	The *x* coordinate of the OBS in meters
obs_z	The *z* coordinate of the OBS in meters
shots_x	The *x* coordinates of the shots in meters
shots_z	The *z* coordinates of the shots in meters
t_obs	The picked traveltime in seconds
num_picks	The number of picks for the current OBS
record_num	The index of the current OBS

Table 3. Description of the structure array associated with a ray path in mFAST.

Field name	Description
obs_x	The *x* coordinate of the OBS in meters
obs_z	The *z* coordinate of the OBS in meters
xs	The *x* coordinates of the shot in meters
zs	The *z* coordinates of the shot in meters
xr	The *x* coordinate of a point in the ray path in meters
zr	The *z* coordinate of a point in the ray path in meters
t_cal	The computed traveltime in seconds
t_obs	The picked (observed) traveltime in seconds
t_success	The index indicating a successful ray tracing

3.2 Model Parameterization

Our mFAST allows the user to parameterize a velocity model in two forms: a node-based model, as displayed in Figure 2a, and a cell-based model, as displayed in Figure 2b. For both types of models, we use regular grids to discretize the model. For the ray-tracing problem, a node-based model is used, and the velocity values are defined on the regularly spaced nodes (see Figure 2a). For the inversion problem, a cell-based model is used, and the unknown slowness values are defined within the cells (Figure 2b).

In mFAST, we invert for slowness rather than velocity for convenience. Specifically, our mFAST allows the user to use a larger grid spacing in the inversion process to reduce the number of inversion parameters compared with that in the ray-tracing (forward-modeling) problem. Therefore, the grid spacings in the inversion, *DX* and *DZ*, are usually much larger than the grid spacings in the forward modeling, *dx* and *dz*. Because forward ray tracing and inversion run alternately within a complete FAST process, a conversion between the node-based model and the cell-based model is required. We have developed two auxiliary functions to perform this conversion: *cells_model2nodes_model.m* and *nodes_model2cells_model.m*.

3.3 Ray Tracing

Ray tracing is the process of computing the ray path between a source–receiver pair and its corresponding traveltime. The computation of first-arrival traveltimes and ray paths are key parts of FAST.

Numerous methods have been developed to solve the traveltime equation, such as the shortest path method (Moser, 1991), the bending method (Moser et al., 1992), and the shooting method. In mFAST, we adopt a two-step method to compute ray paths (Aldridge and Oldenburg, 1993). In this two-step method, we first compute the first-arrival traveltime field by using the shortest path method. We then trace a ray path from the receiver back to the source point along the direction of the negative gradient of the traveltime field. Such a strategy computes the first-arrival traveltimes in the entire model and therefore permits an arbitrary observation geometry. This flexibility is crucial to an OBS survey FAST because the distribution of source–receiver pairs is usually highly irregular. Note that in our mFAST, the source and receiver points are interchanged for the purpose of computational efficiency. That is, we consider the OBS position as the source point and the source point as the OBS point. The interchange of source and receiver points is a direct application of the well-established reciprocity principle in wave propagation theory.

The functionality of ray tracing in our mFAST is *mFAST_raytracing.m* and can be called via

[tcal, success_index, rays] = mFAST_raytracing(forward_model, rays_geom, raytracing_par, flag).

Figure 3 shows a ray-tracing result in a smoothed Marmousi model produced by using *mFAST_raytracing.m*. We display the traced ray paths in Figure 3a and the corresponding first-arrival traveltimes in Figure 3b. We display the synthetic waveforms computed using a finite-difference wave-equation solver in Figure 3c, and we overlay the computed traveltimes on the waveforms in Figure 3d. We observe a good match between the first-arrival traveltimes and the wave-equation-based waveforms.

3.4 Computation of the Jacobian Matrix

Computing the Jacobian matrix (i.e., $\boldsymbol{L}$ in Equation (4)) is the key to our mFAST. The Jacobian matrix $\boldsymbol{L}$ by definition is an N-by-M matrix, where N is the number of rays and M is the number of cells used to represent the model. An element in the Jacobian matrix, say, L_{ij}, represents the length of the segment of the ray path i within cell j. For example, in Figure 4, we use a total of 30 cells to represent a model. Assume a total of four rays are traced. The Jacobian matrix is then a 4-by-30 matrix, and $L_{3,9}$ represents the length of the red-colored segment within cell 9 associated with the ray connecting the OBS with source point 3.

The critical step in constructing the Jacobian matrix is accurately extracting a segment from the whole ray path for each cell, considering the fact that a ray is defined by a set of discrete points, indicated by the solid red circles in Figure 4. We provide two different methods of constructing the Jacobian matrix, which are *Bulid_jacobi_matrix_cells_simple.m* and *Bulid_jacobi_matrix_cells.m*:

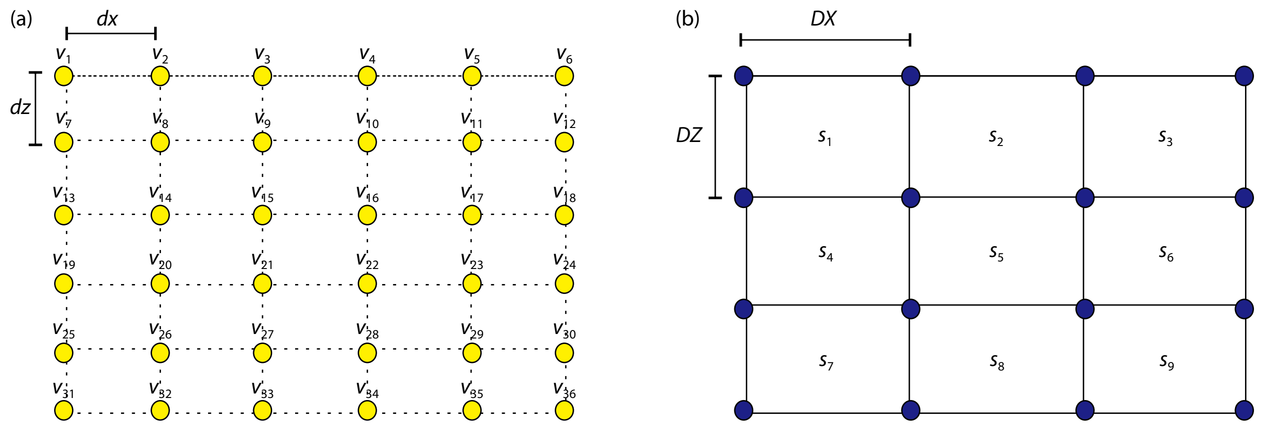
(a)
dx
dz
v_1 v_2 v_3 v_4 v_5 v_6
v_7 v_8 v_9 v_{10} v_{11} v_{12}
v_{13} v_{14} v_{15} v_{16} v_{17} v_{18}
v_{19} v_{20} v_{21} v_{22} v_{23} v_{24}
v_{25} v_{26} v_{27} v_{28} v_{29} v_{30}
v_{31} v_{32} v_{33} v_{34} v_{35} v_{36}
(b)
DX
DZ
s_1 s_2 s_3
s_4 s_5 s_6
s_7 s_8 s_9

Figure 2. Schematic diagram of the two model parameterization strategies in our mFAST. (a) The node-based model and (b) the cell-based model. For the node-based model, parameters such as velocities are assigned to each of the nodes. For the cell-based model, slowness values are assigned within the cells. Values v_1–v_{36} indicate 36 velocity values associated with the 36 distinct nodes, and values s_1–s_9 indicate nine slowness values associated with the nine distinct cells.

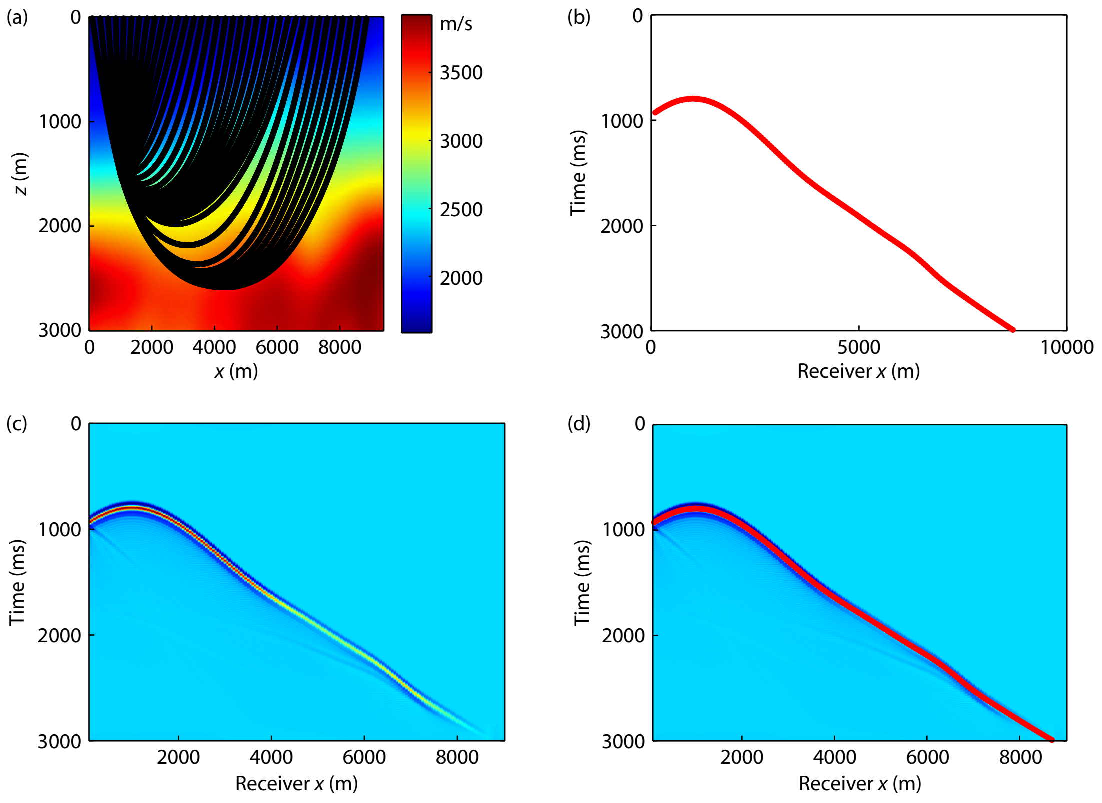

Figure 3. An example of ray tracing using mFAST. (a) Ray paths for several source–receiver pairs, (b) the corresponding first-arrival traveltimes, (c) synthetic waveforms computed by using a finite-difference-based wave-equation solver, and (d) the first-arrival traveltimes overlaid on the synthetic waveforms.

[jacobi_matrix]=Bulid_jacobi_matrix_cells_simple(rays, cells_model, success_index,flag);

[jacobi_matrix]=Bulid_jacobi_matrix_cells(rays, cells_model, success_index,flag).

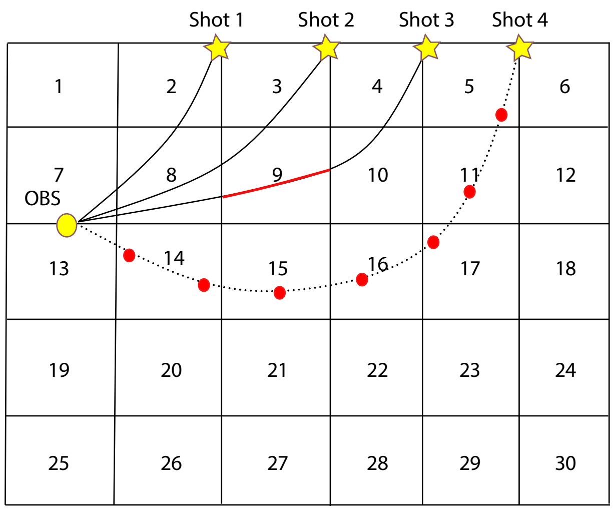

Figure 4. Schematic diagram of the elements within the Jacobian matrix ***L*** and the definition of a ray, which consists of a series of discrete spatial points. OBS, ocean bottom seismometer. The length of the red-colored segment within cell 9 associated with the ray connecting the OBS with source point 3 is the element $L_{3,9}$ of the Jacobian matrix ***L***.

For the simpler method of the two (*Bulid_jacobi_matrix_cells_simple*), we extract two successive points from the ray path and calculate the middle point between the two points. We then determine the cell in which the middle point is located. Finally, we assign to that cell the segment constituted by the two points. This method does not have particularly high accuracy and applies to cases in which the step used for ray tracing is sufficiently small (e.g., 10 m).

For the more accurate method (*Bulid_jacobi_matrix_cells*), all the intersections between the ray path and the grid lines that define the cell model are computed (see Figure 5) with a MATLAB toolbox named *grid_intersections*. The method works for both regular and

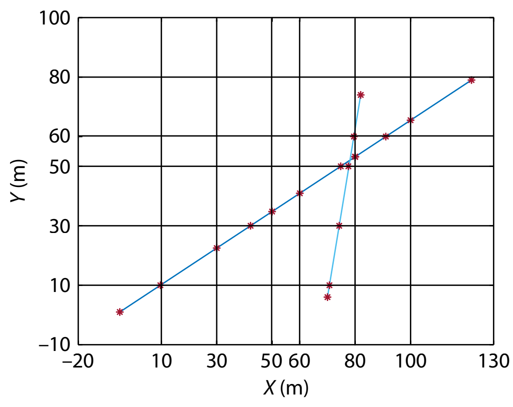

Figure 5. An illustration of 2-D rays intersecting with rectangular grids.

irregular grids. This accurate method applies to cases in which the step used in the ray tracing is large, yet it takes a longer time to construct the Jacobian matrix compared with the simpler method (*Bulid_jacobi_matrix_cells_simple*).

3.5 Linearized Inversion

We perform the traveltime tomography based on Equation (4). However, before solving Equation (4), some constraints should be considered. As we describe in Equations (5) and (6), regularization, in most cases, is critical to stabilizing the inversion procedure. In mFAST, this regularization is achieved via the matrix $\boldsymbol{R}$. For instance, a Laplacian operator can be provided as a regularization matrix with the proper coefficients in the rows and columns to represent the discrete second-order partial differential operator. Users can also define their own regularization matrix as needed. In addition, to ensure that the final result is artifact free, a smoothing operator can be provided to the inversion process as matrix $\boldsymbol{A}$. Once the regularization matrix and smoothing matrix are provided, the augmented matrix in Equation (7) can be straightforwardly built.

The slowness perturbation, Δs, is then obtained by calling the built-in function *lsqr.m*:

$$\Delta s = lsqr(\boldsymbol{D}, \Delta T), \tag{8}$$

where $\boldsymbol{D}$ is the assembled matrix of $\boldsymbol{L}$, $\mu\boldsymbol{R}$, and $\lambda\boldsymbol{A}$, and ΔT is the right term in Equation (7). Note that for Octave users, the MATLAB built-in function *lsqr.m* may not be available and should be properly replaced by some available least-squares linear solver in Octave.

One can perform additional smoothing on the slowness perturbation Δs before adding it to the initial model. The updated model can then serve as an initial model for the next iteration of the inversion.

3.6 Auxiliary Functionalities

Our mFAST includes some auxiliary functionalities for quality control, data and results visualization, and exporting high-quality figures for publication. Our package also provides functionalities to mask the seawater layer to keep the water layer velocity fixed during iterations. Nevertheless, the seawater layer can be updated in the inversion. In this case, the traveltimes of the direct arrivals should be included in the traveltime table.

4. Numerical Results

We use a synthetic data example based on a modified Marmousi model to demonstrate the efficacy and accuracy of our mFAST. The modified Marmousi model is 22,000 m in the horizontal direction and 7,500 m in the vertical direction, with a grid interval of 25 m in both the *x* and *z* directions. The seawater layer has a thickness varying from 900 to 1,600 m. A total of 22 OBS surveys are deployed on the seafloor from the horizontal direction of 200 to 21,200 m, with a regular spacing of 1,000 m. Eight hundred eighty-one sources are placed on the sea surface at a regular interval of 25 m. The model and the observation geometry are displayed in

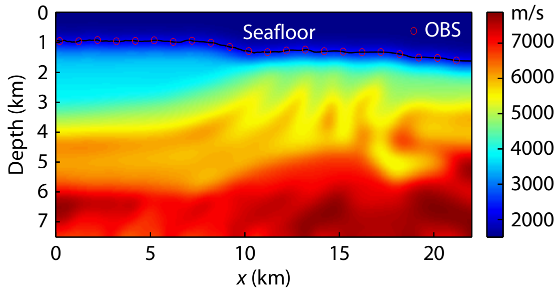

Figure 6. A modified Marmousi model and the observation geometry used in our synthetic data example. OBS, ocean bottom seismometer.

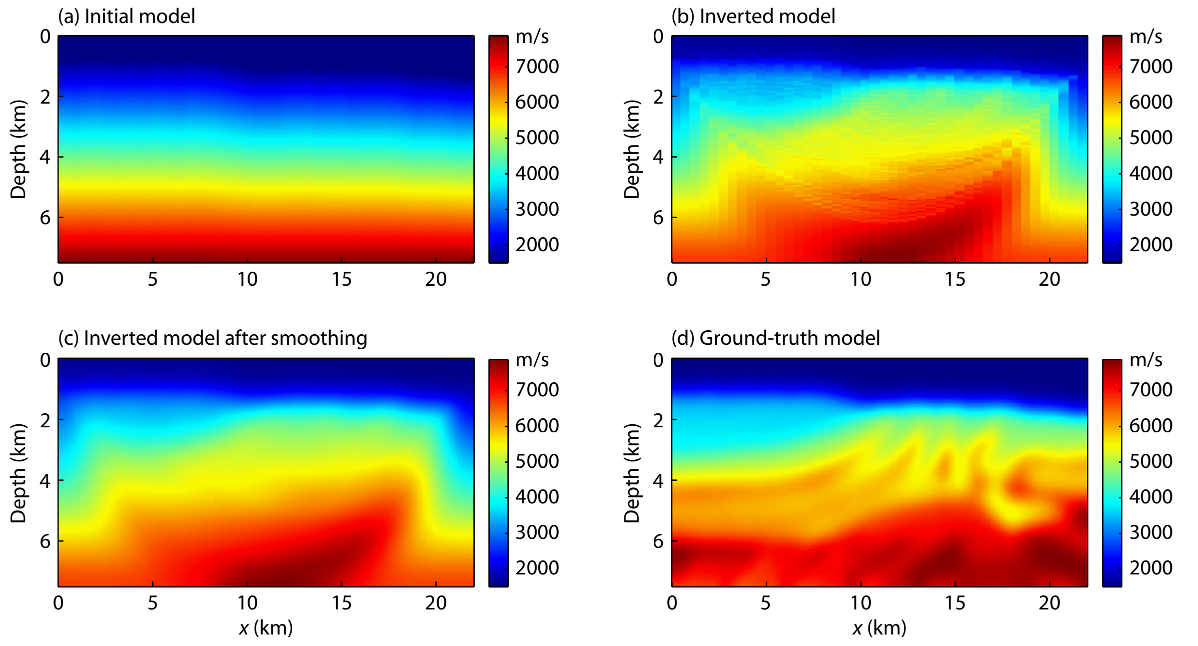

Figure 7. (a) A gradient model used as the initial model, (b) the inverted velocity model after five iterations, (c) a smoothed version of the inverted model displayed in panel (b), and (d) the ground-truth model as the reference.

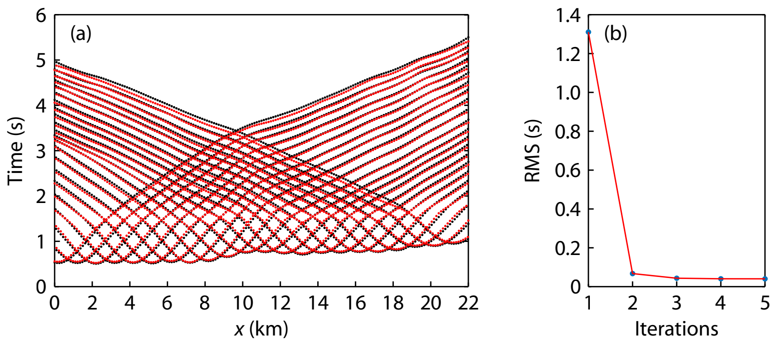

Figure 8. (a) Traveltime matching between the observed (red) and the synthetic (black) traveltimes for different shots. (b) Convergence of the root mean square (RMS) traveltime misfit over a total of five iterations.

Figure 6. We use the gradient velocity model displayed in Figure7a as the initial model.

Figure 7b shows the mFAST inversion result. We observe some artifacts in the inverted model, possibly caused by the sparse OBS geometry. After smoothing, the inverted model displayed in Figure 7c has a similar trend as the ground-truth model displayed in Figure 7d. Because mFAST is essentially a traveltime-based inversion, the small-scale heterogeneities in the horizontal position are not well-resolve from 10 to 18 km in the final inversion result.

We observe a good match between the synthetic and observed traveltimes, as shown in Figure 8a, and the root mean square of the traveltime difference decreases from approximately 1,300 ms to approximately 40 ms in the final iteration, as displayed in Figure 8b. The reduction in the traveltime misfit demonstrates the efficacy and accuracy of our mFAST.

5. Conclusions

We have developed an open-source package based on MATLAB for first-arrival seismic tomography (mFAST), with convenience, usability, and extendibility as the most important motivations and with the OBS survey as a main focus of application. The package contains a forward-modeling module based on a two-step shortest path ray-tracing algorithm, an inversion module based on a linearized traveltime tomography strategy, and regularization and several supporting functionalities. This self-contained package is flexible in terms of model parameterization, observation geometry setup, and hyperparameter tuning. Specifically, we have adopted and implemented a linearized strategy to perform the tomography, leading to fast convergence of the traveltime misfit even for complex models. We have validated our mFAST by using a modified Marmousi model. Our mFAST has been made publicly available as an open-source package and can be downloaded from https://www.mathworks.com/matlabcentral/fileexchange/105885-mfast or github.com/liugele/mFAST.

Acknowledgments

This study was financially supported by the Key Special Project for Introduced Talents Team of Southern Marine Science and Engineering Guangdong Laboratory (Guangzhou) (Grant No. GML2019ZD0207) and was further supported by the Guangzhou Municipal Science and Technology Bureau (Grant No. 202102021054). The author thanks Kai Lu, Donghui Bian, Yunxia Xu, and Wenbin Jiang for helpful discussions. The MATLAB toolbox used to compute the grid intersection can be downloaded from https://www.mathworks.com/matlabcentral/fileexchange/46796-grid_intersections-zp. The shortest path method code used to calculate the traveltime field can be downloaded from http://utam.gg.utah.edu/Inter.LAB1/CH2.lab/lab.mig.pre/lab.html and is also available at https://www.mathworks.com/matlabcentral/fileexchange/30585-large-data-in-matlab-a-seismic-data-processing-case-study. Octave, a well-known free, open-source alternative to the proprietary MATLAB, can be downloaded from https://www.gnu.org/software/octave/download.

References

Aki, K., and Lee, W. H. K. (1976). Determination of three-dimensional velocity anomalies under a seismic array using first P arrival times from local earthquakes: 1. A homogeneous initial model. *J. Geophys. Res.*, *81*(23), 4381–4399. https://doi.org/10.1029/JB081i023p04381

Aldridge, D. F., and Oldenburg, D. W. (1993). Two-dimensional tomographic inversion with finite-difference traveltimes. *J. Seismic Explor.*, *2*(3), 257–274.

Beaty, K. S., Perron, G., Kay, I., and Adam, E. (2002). DSISoft—A MATLAB VSP data processing package. *Comput. Geosci.*, *28*(4), 501–511. https://doi.org/10.1016/S0098-3004(01)00073-5

Guedes, V. J. C. B., Maciel, S. T. R., and Rocha, M. P. (2022). Refrapy: A Python program for seismic refraction data analysis. *Comput. Geosci.*, *159*, 105020. https://doi.org/10.1016/j.cageo.2021.105020

Guo, B., Chen, J. H., Liu, Q. Y., and Li, S. C. (2019). Crustal structure beneath the Qilian Orogen Zone from multiscale seismic tomography. *Earth Planet. Phys.*, *3*(3), 232–242. https://doi.org/10.26464/epp2019025

Guo, R., Li, M. K., Yang, F., Xu, S. H., and Abubakar, A. (2019). First arrival traveltime tomography using supervised descent learning technique. *Inverse Probl.*, *35*(10), 105008. https://doi.org/10.1088/1361-6420/ab32f7

Haney, M. M., and Tsai, V. C. (2017). Perturbational and nonperturbational inversion of Rayleigh-wave velocities. *Geophysics*, *82*(3), F15–F28. https://doi.org/10.1190/GEO2016-0397.1

Haney, M. M., and Tsai, V. C. (2020). Perturbational and nonperturbational inversion of Love-wave velocities. *Geophysics*, *85*(1), F19–F26. https://doi.org/10.1190/geo2018-0882.1

Hobro, J. W. D., Singh, S. C., and Minshull, T. A. (2003). Three-dimensional tomographic inversion of combined reflection and refraction seismic traveltime data. *Geophys. J. Int.*, *152*(1), 79–93. https://doi.org/10.1046/j.1365-246X.2003.01822.x

Hole, J. A. (1992). Nonlinear high-resolution three-dimensional seismic travel time tomography. *J. Geophys. Res.*, *97*(B5), 6553–6562. https://doi.org/10.1029/92JB00235

Korenaga, J., Holbrook, W. S., Singh, S. C., and Minshull, T. A. (1997). Natural gas hydrates on the southeast US. margin: constraints from full waveform and travel time inversions of wide-angle seismic data. *J. Geophys. Res.*, *102*(B7), 15345–15365. https://doi.org/10.1029/97JB00725

Koulakov, I., Stupina, T., and Kopp, H. (2010). Creating realistic models based on combined forward modeling and tomographic inversion of seismic profiling data. *Geophysics*, *75*(3), B115–B136. https://doi.org/10.1190/1.3427637

Marcotte, D. (1991). Cokriging with matlab. *Comput. Geosci.*, *17*(9), 1265–1280. https://doi.org/10.1016/0098-3004(91)90028-C

Moser, T. J. (1991). Shortest path calculation of seismic rays. *Geophysics*, *56*(1), 59–67. https://doi.org/10.1190/1.1442958

Moser, T. J., Nolet, G., and Snieder, R. (1992). Ray bending revisited. *Bull. Seismol. Soc. Am.*, *82*(1), 259–288.

Meléndez, A., Estela Jiménez, C., Sallarès, V., and Ranero, C. R. (2019). Anisotropic P-wave travel-time tomography implementing Thomsen's weak approximation in TOMO3D. *Solid Earth*, *10*(6), 1857–1876. https://doi.org/10.5194/se-10-1857-2019

Meléndez, A., Korenaga, J., Sallarès, V., Miniussi, A., and Ranero, C. R. (2015). TOMO3D: 3-D joint refraction and reflection traveltime tomography parallel code for active-source seismic data-synthetic test. *Geophys. J. Int.*, *203*(1), 158–174. https://doi.org/10.1093/gji/ggv292

Pasquet, S., and Bodet, L. (2017). SWIP: an integrated workflow for surface-wave dispersion inversion and profiling. *Geophysics*, *82*(6), WB47–WB61. https://doi.org/10.1190/GEO2016-0625.1

Rücker, C., Günther, T., and Wagner, F. M. (2017). pyGIMLi: an open-source library for modelling and inversion in geophysics. *Comput. Geosci.*, *109*, 106–123. https://doi.org/10.1016/j.cageo.2017.07.011

Ryberg, T., and Haberland, C. (2018). Bayesian inversion of refraction seismic traveltime data. *Geophys. J. Int.*, *212*(3), 1645–1656. https://doi.org/10.1093/gji/ggx500

Taillandier C., Noble M., Chauris H., and Calandra H. (2009). First-arrival traveltime tomography based on the adjoint-state method. *Geophysics*, *74*, WCB1–WCB10. https://doi.org/10.1190/1.3250266

Tian, D., Sorochian, S., and Myers, D. E. (1993). Correspondence analysis with MATLAB. *Comput. Geosci.*, *19*(7), 1007–1022. https://doi.org/10.1016/0098-3004(93)90006-Q

White, D. J. (1989). Two-dimensional seismic refraction tomography. *Geophys. J. Int.*, *97*(2), 223–245. https://doi.org/10.1111/j.1365-246X.1989.tb00498.x

Yeung, K., and Chakrabarty, C. (1993). An algorithm for transient pressure analysis in arbitrarily shaped reservoirs. *Comput. Geosci.*, *19*(3), 391–397. https://doi.org/10.1016/0098-3004(93)90093-K

Zelt, C. A., and Barton, P. J. (1998). Three-dimensional seismic refraction tomography: a comparison of two methods applied to data from the Faeroe Basin. *J. Geophys. Res.*, *103*(B4), 7187–7210. https://doi.org/10.1029/97jb03536

Zelt, C. A., and Smith, R. B. (1992). Seismic traveltime inversion for 2-D crustal velocity structure. *Geophys. J. Int.*, *108*(1), 16–34. https://doi.org/10.1111/j.1365-246X.1992.tb00836.x

Zhang, J., and Toksöz, M. N. (1998). Nonlinear refraction traveltime tomography. *Geophysics*, *63*(5), 1726–1737. https://doi.org/10.1190/1.1444468

Zhang, X. Y., Bai, Z. M., Xu, T., Gao, R., Li, Q. S., Hou, J., and Badal, J. (2018). Joint tomographic inversion of first-arrival and reflection traveltimes for recovering 2-D seismic velocity structure with an irregular free surface. *Earth Planet. Phys.*, *2*, 220–230. https://doi.org/10.26464/epp2018021