PyFWI: A Python package for full-waveform inversion and reservoir monitoring

Amir Mardan[a], Bernard Giroux[a], Gabriel Fabien-Ouellet[b]

[a] *INRS-ETE, 490 Rue de la Couronne, Québec, QC, G1K 9A9, Canada. E-mail: amirhossein.mardan@inrs.ca (corresponding author); bernard.giroux@inrs.ca.*
[b] *Polytechnique Montréal, 2900 Boulevard Edouard-Montpetit, Montréal, Quebec H3T 1J4, Canada. Email: gabriel.fabien-ouellet@polymtl.ca.*

Abstract

Full-waveform inversion (FWI) of seismic data is a technique that can be used to image the subsurface as well as to monitor time-lapse changes in the subsurface (TL-FWI). `PyFWI` is a package that has been designed to carry out FWI and TL-FWI efficiently on GPU for research purposes. Several time-lapse strategies are implemented in `PyFWI`, such as parallel, double-difference, cascaded, central-difference, cross-updating, simultaneous, and weighted-average. An important challenge of TL-FWI is the crosstalk between parameters across different vintages. To alleviate this problem, `PyFWI` allows using different parameterizations. `PyFWI` is written in Python and relies on OpenCL for enabling calculations on GPUs, which leads to significant reduction of computation time compared to CPU implementation. Using OpenCL makes `PyFWI` portable across systems built with GPUs from different manufacturers.

Keywords: Full-waveform inversion, Reservoir monitoring, Python, GPU programming

1. Motivation and significance

Seismic methods are used to image and characterize the subsurface. They rely on mechanical waves to probe the Earth [1]. These waves propagate based on the (visco)elastic properties of the Earth. Full-waveform inversion (FWI) has shown its ability to extract high resolution image of the (visco)elastic properties in the subsurface directly from the recorded wavefield [2, 3]. Time-lapse FWI (TL-FWI) has also shown promising results to monitor property changes in the subsurface, which is important for CO_2 sequestration and oil and gas applications [4, 5, 6].

FWI allows obtaining a model of the subsurface by minimizing the residuals between data estimated from an initial model and the observed data at the receiver's locations [7]. The minimization is carried out with local gradient descent algorithms. To estimate the gradient of the misfit function, the adjoint state method [8] is used which requires one more modeling run. As FWI is computationally expensive, substantial work has been done to adapt this method to GPU programming [9, 10]. `PyFWI` relies on OpenCL [11] in its core to accelerate the computation by taking the advantage of GPU programming. As `PyFWI` is written in Python, this integration is happening through PyOpenCL [12]. Contrary to CUDA, OpenCL is compatible with the majority of existing processors [10] and this makes `PyFWI` a portable option.

In addition to computation cost, FWI is an ill-posed problem [13]. An additional challenge is the crosstalk between the parameters [14, 3]. To make the problem better posed, four common regularization methods such as Tikhonov, total-variation, prior-information, and parameters relationship [15, 4, 16, 17] are integrated into this package. The most effective methods to mitigate the problem of crosstalk are performing the inversion using an optimal parameterization [18, 14] and using the Hessian for optimization [19, 20]. To model isotropic elastic Earth, three parameters are required [18]. Depending on the problem at hand, different parameterizations (all linked to the elastic properties) can be used to perform the calculations. In the current version, `PyFWI` can be used to perform TL-FWI in four different parameterizations such as DV (density, P-wave velocity, and S-wave velocity), PCS (porosity, clay content, and water saturation), LMD (Lamé moduli and density), and KMD (bulk modulus, shear modulus, and density). It should be noted that users can easily augment this list with custom-defined parameterizations. In addition to conjugate gradient, the ℓ-BFGS method is provided to minimize the loss by taking the approximate of Hessian into account [21]. In addition to the mentioned techniques, gradient scaling ability [22] is integrated into this package to better address the problem of crosstalk.

`PyFWI` is a versatile and flexible package to perform TL-FWI. Different methods have been proposed to provide accurate time-lapse estimates of the subsurface using TL-FWI. `PyFWI` allows geophysicists to conduct a time-lapse study using seven methods such as parallel, double-difference [23], cascaded [24], cross-updating [25], simultaneous [25], central-difference [5], and weighted-average inversion [6]. These methods are discussed in more detail in [26].

2. Software description

`PyFWI` is a Python package first published in January 2022 on the Python Package Index (PyPi) under the GNU General Public License (GPLv3). The source code is hosted on GitHub and open to contributions.

2.1. Software functionality

The main functionality of `PyFWI` is providing the required tools to perform FWI and TL-FWI with a variety of techniques, considering seismic wave propagation in an elastic medium. FWI is a local minimization of residuals between the observed and estimated wavefields at the receiver's locations [13]. Taking the regularization term (χ_{REG}) into the consideration, the cost function of FWI problem can be written as

$$\chi(\mathbf{m}) = \frac{1}{2}\|\mathbf{W}_d\left(\mathbf{R}F(\mathbf{m}) - \mathbf{d}\right)\|_2^2 + \chi_{REG}, \tag{1}$$

where $\mathbf{R}$ samples the seismic wavefield at the receiver's locations. Vectors $\mathbf{m}$ and $\mathbf{d}$ represent the model parameters and observed data, respectively, along with $\mathbf{W}_d$ which is a weighting operator of the data misfit. The wavefield $\mathbf{u}$ is the solution of the partial differential operator, $F(\mathbf{m})$. In the time domain, this operator can be written for the 2D elastic case as

$$\begin{cases} \rho\frac{\partial v_x}{\partial t} = \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xz}}{\partial z} + s_{v_x}, \\ \rho\frac{\partial v_z}{\partial t} = \frac{\partial \tau_{zz}}{\partial z} + \frac{\partial \tau_{xz}}{\partial x} + s_{v_z}, \\ \frac{\partial \tau_{xx}}{\partial t} = (\lambda + 2\mu)\frac{\partial v_x}{\partial x} + \lambda\frac{\partial v_z}{\partial z} + s_{\tau_x}, \\ \frac{\partial \tau_{zz}}{\partial t} = \lambda\frac{\partial v_x}{\partial x} + (\lambda + 2\mu)\frac{\partial v_z}{\partial z} + s_{\tau_z}, \\ \frac{\partial \tau_{xz}}{\partial t} = \mu(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x}), \end{cases} \tag{2}$$

where s is the source function and where the particle velocities in x- and z-directions (v_x and v_z) plus normal (τ_{xx} and τ_{zz}) and shear stresses (τ_{xz}) build the wavefield vector, $\mathbf{u}$. At time $t = 0$, we have

$$v_i^{t=0} = \tau_{ij}^{t=0} = 0, \tag{3}$$

as the initial condition of this problem. For boundary condition, perfectly matched layer [27] (PML) is used to reduce the reflected energy from the edges of the model. To estimate the gradient of the cost function of a PDE-constrained optimization problem such as FWI, the adjoint state method can be used [8]. The adjoint of wavefield, $\tilde{\mathbf{u}}$, written as,

$$\tilde{\mathbf{u}} = [\tilde{v}_x, \tilde{v}_z, \tilde{\tau}_{xx}, \tilde{\tau}_{zz}, \tilde{\tau}_{xz}]^T, \tag{4}$$

is the solution of the adjoint of equation 2, presented as

$$\begin{cases} \frac{\partial \tilde{v}_x}{\partial t} = -\frac{\partial(\lambda+2\mu)\tilde{\tau}_{xx}}{\partial x} - \frac{\partial \lambda \tilde{\tau}_{zz}}{\partial x} - \frac{\partial \mu \tilde{\tau}_{xz}}{\partial z} - r_{v_x}, \\ \frac{\partial \tilde{v}_z}{\partial t} = -\frac{\partial \lambda \tilde{\tau}_{xx}}{\partial z} - \frac{\partial(\lambda+2\mu)\tilde{\tau}_{zz}}{\partial z} - \frac{\partial \mu \tilde{\tau}_{xz}}{\partial x} - r_{v_z}, \\ \frac{\partial \tilde{\tau}_{xx}}{\partial t} = -\frac{\partial \rho^{-1} \tilde{v}_x}{\partial x} - r_{\tau_{xx}}, \\ \frac{\partial \tilde{\tau}_{zz}}{\partial t} = -\frac{\partial \rho^{-1} \tilde{v}_z}{\partial z} - r_{\tau_{zz}}, \\ \frac{\partial \tilde{\tau}_{xz}}{\partial t} = -\frac{\partial \rho^{-1} \tilde{v}_x}{\partial z} - \frac{\partial \rho^{-1} \tilde{v}_z}{\partial x} - r_{\tau_{xz}}, \end{cases} \tag{5}$$

where ^{T} is the transpose operator and r represents the residuals between the components of seismic data. The gradient of the cost function can be obtained by

$$\nabla_{\mathbf{m}} \chi = \left\langle \tilde{\mathbf{u}}, \frac{\partial F(\mathbf{m})}{\partial \mathbf{m}} \mathbf{u} \right\rangle, \tag{6}$$

which is the zero-lag cross-correlation between the forward wavefield and the adjoint wavefield multiplied by the scattering matrix, $\frac{\partial F(\mathbf{m})}{\partial \mathbf{m}}$. Equation 2 is used to perform forward propagation and it is solved forward in time ($t : 0 \rightarrow T$) where T is the recording time. In contrast, equation 5 performs backward propagation and is solved in reverse of time ($t : 0 \leftarrow T$). Hence, the forward wavefield, $\mathbf{u}$, should be available while solving equation 5 to calculate the gradient of the cost function using equation 6. In PyFWI, wave equation is solved in time domain and $\mathbf{u}$ should be stored at each time step of forward modeling to be used after obtaining the adjoint wavefield at the corresponding time step. Although this technique is simple, the storage and input/output requirements of this data volume is demanding and needs a more efficient strategy. Different strategies are discussed in [28]. In `PyFWI`, a checkpointing strategy is employed where the forward wavefield is stored at some time steps and reconstructed at other time steps. `PyFWI` allows users to specify what percentage of forward wavefield should be stored.

Due to its ease of implementation and great compatibility with GPU programming, the finite-difference method (FDM) is used to solve equation 2 and equation 5. These equations are discretized in space using a staggered grid [29] as shown in Figure 1a. In time, derivatives are estimated using the leapfrog method where velocities and their adjoints are updated at integer time steps, and stresses and their adjoints are updated at half-time steps. Figure 1b shows the temporal discretizing scheme for wave propagation.

2.2. Software architecture

Algorithm 1 describes the process of computing the gradient of the cost function (equation 1) with respect to model parameters using the adjoint state method. This algorithm is implemented as the core of `PyFWI` in a

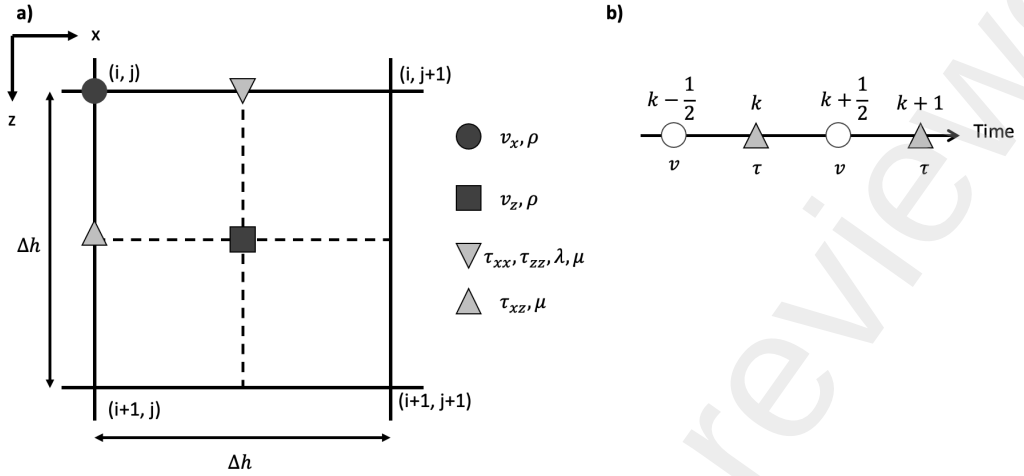

Figure 1: Geometry of a grid used for discretizing equation 2 and equation 5. (a) The staggered grid is used for spatial discretizing. Temporal discretizing for (b) forward and (c) adjoint modeling where k is a time step.

module named `wave_propagation` (Figure 2). This module allows users to perform the wave propagation (forward and backward) using the class `WavePropagator`. `WavePropagator` consists of two main methods, `forward_modeling` for performing the forward propagation (step 3-13) and `gradient` for performing the backward propagation and computing the gradient (step 15 - 28).

To estimate the gradient using `PyFWI`, it is first required to input the acquisition geometry and define parameters such as location of sources and receivers and the source function. This step is performed with the aid of the module `acquisition`. These parameters, in addition to the observed data and an initial model are the inputs for computing the gradient. `PyFWI` prepares the buffers for forward and backward wave propagation. In the next step, the software calls OpenCL kernels that apply finite-difference stencils to offload the computation to the GPUs. Forward wave propagation is performed on GPU. In the next step, the waveform is sampled at the receiver's locations (d_{est}). To compute the gradient of the cost function, the OpenCL kernels are called for performing the back propagation and the gradient is computed using the adjoint-state method. At the end, the gradients are transferred to CPU and the minimization step is performed in Python. After estimating the gradient at step 28 of algorithm 1, the module `grad_switcher` can be used to provide the gradient of the cost function in terms of different parameterizations. The computation cost for such an operation is negligible.

Algorithm 1 Pseudo-code for computation of the gradient using the adjoint state method where N_s, N_t, d_{est}, and *checkpoints* are number of sources, number of time samples, estimated data, and a list of time steps at which the forward wavefield should be stored.

Input: Observed data (d_{obs}), initial model (m_0), and acquisition parameters
Initialize OpenCL
$s \leftarrow 1$
while $s \leq N_s$ **do**
    $t \leftarrow 1$
    while $t \leq N_t$ **do**
        Call kernel_updatev for v_i
        Call kernel_updatet for τ_{ij}
        if t in *checkpoints* **then**
            store v_i and τ_{ij}
        Increment t
    Increment s
Record d_{est}
Compute residuals $d_{est} - d_{obs}$
while $1 \leq s \leq N_s$ **do**
    $t \leftarrow N_t$
    while $1 \leq t \leq N_t$ **do**
        if t in *checkpoints* **then**
            read v_i and τ_{ij}
        else
            Call kernel_updatet for τ_{ij}
            Call kernel_updatev for v_i
        Call kernel_updatet for $\tilde{\tau_{ij}}$
        Call kernel_updatev for $\tilde{v_i}$
        Call kernel_updateg for *gradients*
        Decrement t
    Decrement s
    Compute *gradients*
    Switch *gradients*

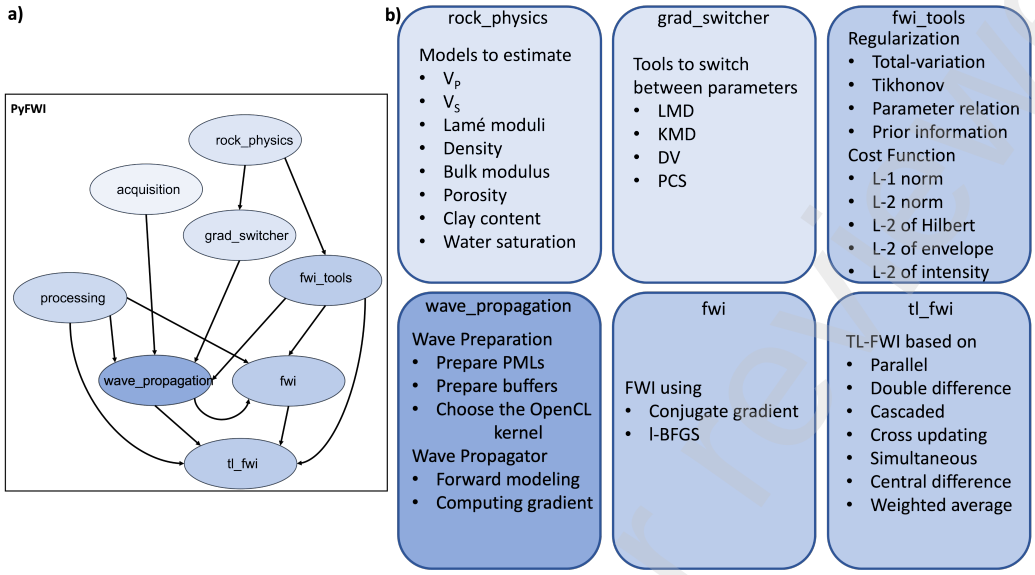

Figure 2: (a) Structure of PyFWI. `PyFWI` has `wave_propagation` at its core to perform wave propagation and compute the gradient. This module is used in `fwi` and `tl_fwi` to perform FWI and TL-FWI. (b) Applications of each module where `acquisition` and `processing` are used to define the acquisition and perform gain on in the current version.

Hence, changing the parameterization is performed on CPU.

On top of `wave_propagation`, `PyFWI` consists of modules `fwi` and `tl_fwi` to perform full-waveform inversion and time-lapse full-waveform inversion, respectively. Forward and backward propagations are the essence of performing these inversions. The class `TimeLapse` in module `tl_fwi` is derived from class `WavePropagator`, and thus benefits from the latter to simulate wave propagation. The class `TimeLapse`, also uses the class `FWI` from module `fwi` in its computation. The minimization of the cost function occurs mostly in class `FWI`. The optimization procedure can be regularized using `Regularization`, defined in the module `fwi_tools`.

3. Illustrative Examples

In this section, we present a simple synthetic example that shows the main functionalities of `PyFWI`. In this example, we simulate two common scenarios, and make use of the Marmousi model [30], which is a well-known model for analyzing the efficiency of FWI algorithms [25, 4, 6]. Figure 3a-3c show the properties of the true baseline model and Figure 3d-3f show the properties of the true monitor model. The Marmousi model contains two sandstone reservoirs. To create the baseline model, we changed the V_P and V_S in the

reservoirs for their respective background velocity. This means the velocity decreases in time in one reservoir (mimicking CO_2 injection in the reservoir) and increases in another reservoir (simulating petroleum production). The true time-lapse changes are presented in Figure 3g- 3i, in which the reservoirs can be easily detected. The seismic data modeled with the baseline and monitor models and their difference are presented in Figure 4. The time-lapse data (Figure 4c) are amplified by a factor of 5 for better visualization. This figure illustrates a shot gather for the middle source, in an acquisition where 7 isotropic explosive sources are deployed. For these sources, the Ricker wavelet with a peak frequency of 30 Hz is employed. We also use PML around the model to avoid reflections from the edges of the model.

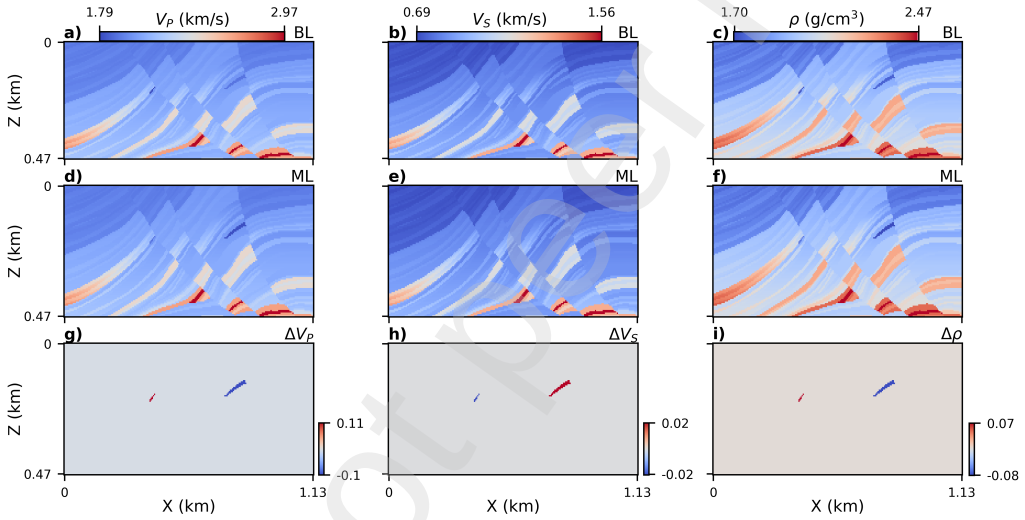

Figure 3: Marmousi model used as an example for this study. True (a-c) baseline, (d-f) monitor models, and (g-i) time-lapse changes. From left to right, the columns show V_P, V_S, and ρ, respectively.

To perform TL-FWI, we used frequencies of [10, 20, 25, 35, 40, 55] Hz in a multi-scale strategy with 10 iterations at each frequency (listing 1). The multi-scale strategy is provided to mitigate the problem of cycle-skipping and consists in performing the inversion from lower frequencies toward higher ones. The simultaneous time-lapse algorithm is used for this study, which minimizes the difference between the observed and estimated data from different vintages by considering a penalty term for difference between the baseline and monitor models, such that

$$\begin{aligned}\chi(\mathbf{m}_b, \mathbf{m}_m) =& \lambda_b \|\mathbf{R}F(\mathbf{m}_b, \mathbf{u}_b) - \mathbf{d}_b\|_2^2 + \lambda_m \|\mathbf{R}F(\mathbf{m}_m, \mathbf{u}_m) - \mathbf{d}_m\|_2^2 \\ &+ \lambda_\Delta \|\mathbf{m}_m - \mathbf{m}_b\|_2^2,\end{aligned} \tag{7}$$

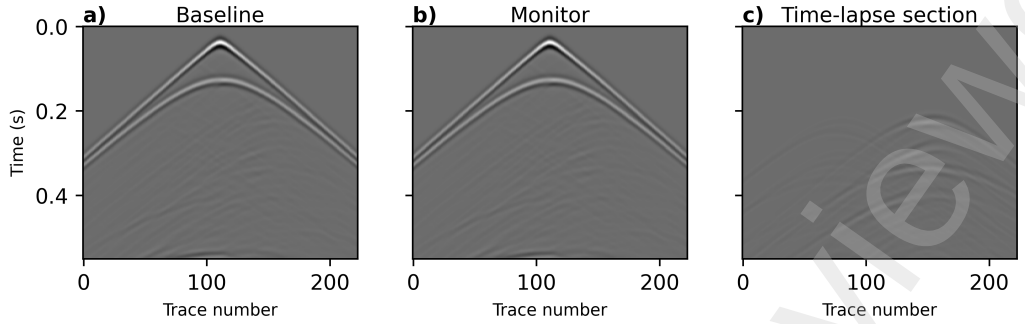

Figure 4: Seismic data from (a) baseline, (b) monitor models, and (c) time-lapse difference. The amplitude of the time-lapse data is amplified by a factor of 5 for better visualization.

where λ and the subscripts b and m represent the regularization weight for each term, baseline, and monitor models.

```
from PyFWI.tl_fwi import TimeLapse

inv_freqs = [10, 20, 25, 35, 40, 55]
iterations =[10 for freq in inv_freqs]

# Creating a TL-FWI object
tl = TimeLapse(db_obs, dm_obs, inpa, src, rec_loc,
    model_shape)

# Call the object for estimating the time-lapse changes by
    updating m0 as the initial model
dm, rms = tl(m0, iterations, inv_freqs, tl_method="sim",
            n_params=1, k_0=3, k_end=4)
```

Listing 1: Performing TL-FWI by creating the object `TL` and calling this object by providing the initial model, number of iteration for each frequency, and list of frequencies for performing multi-scale inversion.

TL-FWI results are presented in Figure 5 using two different parameterizations. Figure 5a-5c are obtained using the DV parameterization while Figure 5d-5f are estimated using the PCS parameterization. Comparing the estimated time-lapse changes (Figure 5) with the true changes (Figure 3g-3i) shows how changing the parameterization can lead to a more accurate estimate of changes in the elastic properties of the subsurface (11.7% higher accuracy). Although the estimates still have artifacts, `PyFWI` gives the user the possibility to choose a parameterization and a strategy that is appropriate for the project at hand.

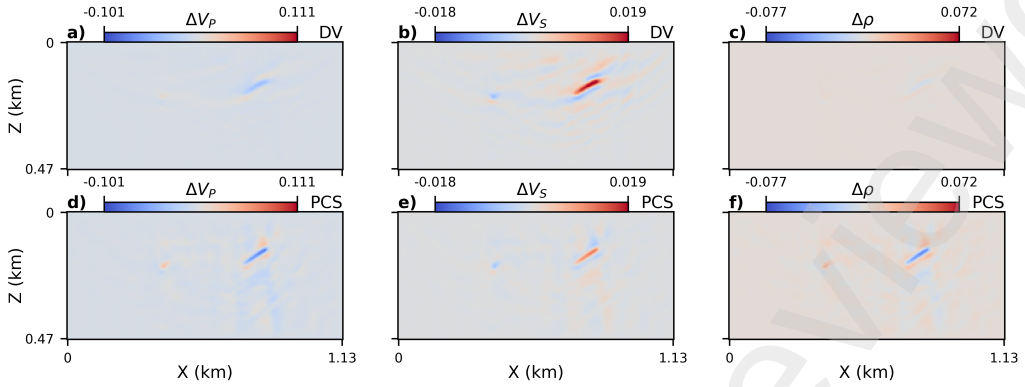

Figure 5: The estimated time-lapse image using (a-c) DV parameterization and (d-f) PCS parameterization. From left to right, the columns show the changes in V_P, V_S, and ρ, respectively. The PCS parameterization leads to 11.7% higher accuracy in this study.

4. Impact

`PyFWI` can be employed to perform full-waveform inversion and time-lapse full-waveform inversion. Using OpenCL allows this package to reduce the computation time by leveraging the compute power from GPU of most vendors. Although the authors are dedicated to further development of the package, the current version can be used only for 2D problems.

This package was first released in January 2022 and based on statistics from PePy website, it has been downloaded 13000 times. The authors used this package in a variety of studies and they published different papers [6, 31, 32, 33], in addition to the papers under revision or in preparation where `PyFWI` is presented. This should increase the visibility of this package outside of the research group of the authors.

5. Conclusion

In this paper, `PyFWI` is presented which is a Python package for performing full-waveform inversion and time-lapse full-waveform inversion. To mitigate some challenges inherent to these problems, `PyFWI` provides different options such as different types of regularization, multi-scale strategy, gradient scaling, and different parameterizations. The most important aspect of this package is the ability to perform the inversion with different parameterizations which has significant effects on the results. Using OpenCL makes this package compatible with majority of GPU manufacturers. The source code is hosted on GitHub and is open to contribution and further development. `PyFWI` is also provided on the Python Package Index (PyPi) website which leads to easy installation using the package installer for Python (pip).

6. Acknowledgments

This project was supported by an NSERC Discovery Grant to B. Giroux (RGPIN-2017-06215). A. Mardan also was partially supported by SEG/-Landmark Scholarship.

References

[1] B. Giroux, ttcrpy: A python package for traveltime computation and raytracing, SoftwareX 16 (2021) 100834. doi:10.1016/j.softx.2021.100834.

[2] M. Maharramov, B. L. Biondi, M. A. Meadows, Time-lapse inverse theory with applications, Geophysics 81 (6) (2016) R485–R501. doi:10.1190/geo2016-0131.1.

[3] G. Fabien-Ouellet, E. Gloaguen, B. Giroux, Time domain viscoelastic full waveform inversion, Geophysical Journal International 209 (3) (2017) 1718–1734. doi:10.1093/gji/ggx110.

[4] A. Asnaashari, R. Brossier, S. Garambois, F. Audebert, P. Thore, J. Virieux, Time-lapse seismic imaging using regularized full-waveform inversion with a prior model: Which strategy?, Geophysical Prospecting 63 (1) (2015) 78–98. doi:10.1111/1365-2478.12176.

[5] W. Zhou, D. Lumley, Central-difference time-lapse 4D seismic full-waveform inversion, Geophysics 86 (2) (2021) R161–R172. doi:10.1190/geo2019-0834.1.

[6] A. Mardan, B. Giroux, G. Fabien-Ouellet, Time-lapse seismic full waveform inversion using improved cascaded method, in: Second EAGE Conference on Seismic Inversion, European Association of Geoscientists & Engineers, 2022, pp. 1–5. doi:10.3997/2214-4609.202229003.

[7] A. Tarantola, Inversion of seismic reflection data in the acoustic approximation, Geophysics 49 (8) (1984) 1259–1266. doi:10.1190/1.1441754.

[8] R. E. Plessix, A review of the adjoint-state method for computing the gradient of a functional with geophysical applications, Geophysical Journal International 167 (2) (2006) 495–503. doi:10.1111/j.1365-246X.2006.02978.x.

[9] J. Shin, W. Ha, H. Jun, D.-J. Min, C. Shin, 3D Laplace-domain full waveform inversion using a single GPU card, Computers & Geosciences 67 (2014) 1–13. doi:10.1016/j.cageo.2014.02.006.

[10] G. Fabien-Ouellet, E. Gloaguen, B. Giroux, Time-domain seismic modeling in viscoelastic media for full waveform inversion on heterogeneous computing platforms with OpenCL, Computers & Geosciences 100 (2017) 142 – 155. doi:10.1016/j.cageo.2016.12.004.

[11] J. E. Stone, D. Gohara, G. Shi, Opencl: A parallel programming standard for heterogeneous computing systems, Computing in Science Engineering 12 (3) (2010) 66–73. doi:10.1109/MCSE.2010.69.

[12] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, A. Fasih, PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation, Parallel Computing 38 (3) (2012) 157–174. doi:10.1016/j.parco.2011.09.001.

[13] J. Virieux, S. Operto, An overview of full-waveform inversion in exploration geophysics, Geophysics 74 (6) (2009) WCC1–WCC26. doi:10.1190/1.3238367.

[14] S. Operto, Y. Gholami, V. Prieux, A. Ribodetti, R. Brossier, L. Metivier, J. Virieux, A guided tour of multiparameter full-waveform inversion with multicomponent data: From theory to practice, The Leading Edge 32 (9) (2013) 1040–1054. doi:10.1190/tle32091040.1.

[15] A. Y. Anagaw, M. D. Sacchi, Edge-preserving seismic imaging using the total variation method, Journal of Geophysics and Engineering 9 (2) (2012) 138–146.

[16] E. Esser, L. Guasch, T. van Leeuwen, A. Y. Aravkin, F. J. Herrmann, Total variation regularization strategies in full-waveform inversion, SIAM Journal on Imaging Sciences 11 (1) (2018) 376–406. doi:10.1137/17M111328X.

[17] Q. Hu, K. Innanen, Elastic full-waveform inversion with rock-physics constraints, in: First International Meeting for Applied Geoscience & Energy, Society of Exploration Geophysicists, 2021, pp. 662–666. doi:10.1190/segam2021-3594471.1.

[18] A. Tarantola, A strategy for nonlinear elastic inversion of seismic reflection data, Geophysics 51 (10) (1986) 1893–1903. doi:10.1190/1.1442046.

[19] L. Métivier, R. Brossier, J. Virieux, S. Operto, Full waveform inversion and the truncated Newton method, SIAM Journal on Scientific Computing 35 (2) (2013) B401–B437. doi:10.1137/120877854.

[20] P. Yang, R. Brossier, L. Métivier, J. Virieux, W. Zhou, A time-domain preconditioned truncated Newton approach to visco-acoustic multiparameter full waveform inversion, SIAM Journal on Scientific Computing 40 (4) (2018) B1101–B1130. doi:10.1137/17M1126126.

[21] J. Nocedal, S. J. Wright, Numerical Optimization, Springer, New York, NY, 2006. doi:10.1007/978-0-387-40065-5_1.

[22] F. Lavoué, R. Brossier, L. Métivier, S. Garambois, J. Virieux, Two-dimensional permittivity and conductivity imaging by full waveform inversion of multioffset GPR data: A frequency-domain quasi-Newton approach, Geophysical Journal International 197 (1) (2014) 248–268. doi:10.1093/gji/ggt528.

[23] T. Watanabe, S. Shimizu, E. Asakawa, T. Matsuoka, Differential waveform tomography for time-lapse crosswell seismic data with application to gas hydrate production monitoring, Society of Exploration Geophysicists, 2005, pp. 2323–2326. doi:10.1190/1.1845221.

[24] P. Routh, G. Palacharla, I. Chikichev, S. Lazaratos, Full Wavefield Inversion of Time-Lapse Data for Improved Imaging and Reservoir Characterization, Society of Exploration Geophysicists, 2012, pp. 1–6. doi:10.1190/segam2012-1043.1.

[25] M. Maharramov, B. Biondi, Robust joint full-waveform inversion of time-lapse seismic data sets with total-variation regularization, arXiv: Geophysicsdoi:10.48550/arXiv.1408.0645.

[26] A. Mardan, B. Giroux, G. Fabien-Ouellet, Weighted-average time-lapse seismic full-waveform inversion, Geophysicsdoi:https://doi.org/10.1190/geo2022-0090.1.

[27] J.-P. Berenger, A perfectly matched layer for the absorption of electromagnetic waves, Journal of Computational Physics 114 (2) (1994) 185–200. doi:10.1006/jcph.1994.1159.

[28] B. D. Nguyen, G. A. McMechan, Five ways to avoid storing source wavefield snapshots in 2D elastic prestack reverse time migration, Geophysics 80 (1) (2015) S1–S18. doi:10.1190/geo2014-0014.1.

[29] J. Virieux, P-SV wave propagation in heterogeneous media; velocity-stress finite-difference method, Geophysics 51 (4) (1986) 889–901. doi:10.1190/1.1442147.

[30] A. Brougois, M. Bourget, M. Lailly, P. Ricarte, R. Versteeg, Marmousi, model and data (1990). doi:10.3997/2214-4609.201411190.

[31] A. Mardan, B. Giroux, G. Fabien-Ouellet, Time-lapse full-waveform inversion for monitoring the fluid saturation, in: Second EAGE Conference on Seismic Inversion, European Association of Geoscientists & Engineers, 2022, pp. 1–5. doi:10.3997/2214-4609.202210635.

[32] A. Mardan, B. Giroux, G. Fabien-Ouellet, Effects of nonrepeatability on time-lapse full-waveform inversion, in: 2^{nd} EAGE Conference on Seismic Inversion, European Association of Geoscientists & Engineers, 2022, pp. 1–5. doi:10.3997/2214-4609.202211009.

[33] A. Mardan, B. Giroux, G. Fabien-Ouellet, M. Saberi, Direct monitoring of fluid saturation using time-lapse full-waveform inversion, in: 2^{nd} International Meeting for Applied Geoscience & Energy Expanded Abstracts, Society of Exploration Geophysicists, 2022. doi:10.1190/image2022-3746685.1.

Required Metadata

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.1.9
C2	Permanent link to code/repository used for this code version	github.com/amirmardan/pyfwi
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	GPL-3.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	python, OpenCL
C7	Compilation requirements, operating environments & dependencies	
C8	If available Link to developer documentation/manual	https://pyfwi.readthedocs.io/en
C9	Support email for questions	mardan.amir.h@gmail.com

Table 1: Code metadata