Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

SeisFlows—Flexible waveform inversion software

Ryan T. Modrak[a,*], Dmitry Borisov[a], Matthieu Lefebvre[b], Jeroen Tromp[a,c]

[a] Princeton University, Department of Geosciences, United States
[b] Princeton University, Institute for Computational Science and Engineering, United States
[c] Princeton University, Program in Applied and Computational Mathematics, United States

ARTICLE INFO

Keywords:
Full-waveform inversion
High-performance computing
Seismology
Ultrasound
Exploration geophysics

ABSTRACT

SeisFlows is an open source Python package that provides a customizable waveform inversion workflow and framework for research in oil and gas exploration, earthquake tomography, medical imaging, and other areas. New methods can be rapidly prototyped in SeisFlows by inheriting from default inversion or migration classes, and code can be tested on 2D examples before application to more expensive 3D problems. Wave simulations must be performed using an external software package such as SPECFEM3D. The ability to interface with external solvers lends flexibility, and the choice of SPECFEM3D as a default option provides optional GPU acceleration and other useful capabilities. Through support for massively parallel solvers and interfaces for high-performance computing (HPC) systems, inversions with thousands of seismic traces and billions of model parameters can be performed. So far, SeisFlows has run on clusters managed by the Department of Defense, Chevron Corp., Total S.A., Princeton University, and the University of Alaska, Fairbanks.

1. Introduction

Waveform inversion is a powerful and computationally expensive method for estimating an object's material properties and imaging its internal structure (Virieux and Optero, 2009; Burstedde and Ghattas, 2009). Whether applied to a human body, a hydrocarbon reservoir, or a continental craton, careful integration of software components, including wave-equation solvers, nonlinear optimization libraries, and signal processing routines, is required.

Many waveform inversion packages are proprietary codes developed by oil and gas companies. Usually, such software is not available to independent researchers.

Outside of industry, a number of open source waveform inversion packages have been developed. MADAGASCAR, a widely-used exploration geophysics package, provides a variety of capabilities, including some waveform inversion functions (Fomel et al., 2012). SEISCOPE, another widely used tool, provides FORTRAN 90 nonlinear optimization routines rather than a complete inversion workflow (Métivier and Brossier, 2016). Three more recent packages, PySIT, LASIF, and SimPEG use Python for software integration and data processing tasks. PySIT delivers powerful waveform inversion capabilities in a toolbox format (Hewett and Demanet, 2013). LASIF provides a useful system for organizing and visualizing earthquake tomography results (Krischer et al., 2015a). SimPEG provides a broad framework for geophysical inversion with accompanying simulation tools based on the finite volume method (Cockett et al., 2015).

SeisFlows differs from previous software in providing a high-level inversion workflow emphasizing both flexibility and HPC portability. To be of use to a wide community and remain relevant in an evolving research field, software must be flexible and portable across hardware and software environments. For methodological research, being able to prototype on a workstation or small cluster is desirable. For large-scale 3D applications, running in massively parallel environments is essential. Portability can be especially difficult in large-scale applications due to lack of standardization between clusters.

To help meet these design goals, SeisFlows abstracts the inversion problem into six parts: (1) solver, (2) preprocessing, (3) postprocessing, (4) nonlinear optimization, (5) system, and (6) workflow. The source code underlying SeisFlows is structured in a modular way based on these categories, and users are offered various choices for each one. For example, if the study area in an earthquake tomography project expands, users can trade a Cartesian solver for a spherical whole-earth solver. If a PBS cluster goes offline and a SLURM cluster comes online to replace it, users can trade the PBS system interface for a SLURM system interface. If desired functionality is missing from the main package, users can contribute their own classes or overload default ones.

* Corresponding author.
E-mail address: rmodrak@princeton.edu (R.T. Modrak).

https://doi.org/10.1016/j.cageo.2018.02.004
Received 9 January 2017; Received in revised form 17 January 2018; Accepted 15 February 2018
Available online 19 February 2018

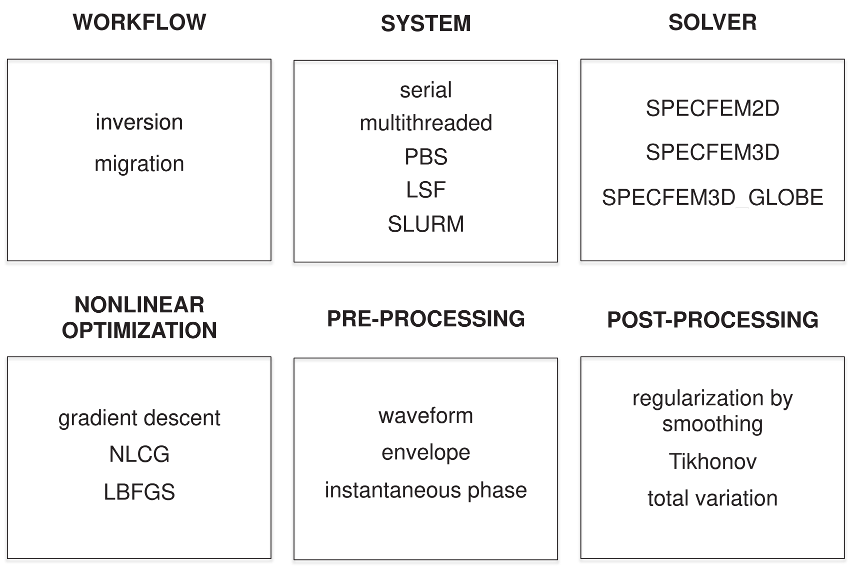

Fig. 1. Structure of the SeisFlows package. A selection of main options or settings are listed for each of the six parts that comprise the package.

This paper is divided into four sections. First, we elaborate on the motivation behind SeisFlows. Second, we briefly review the theory of waveform inversion. Third, we describe each of the six component parts and how they work together to mange the complexity of an inversion. Finally, we illustrate use of the package in an HPC context through an exploration geophysics example.

2. Motivation

About five years ago the seismology group at Princeton University was engaged in several waveform inversion projects, but differences between applications and the rapid pace of development led our codes to become fragmented. This in turn led to duplication of effort, susceptibility to bugs, and difficulty bringing new members up to speed. A framework for streamlining our software efforts was needed.

Besides the need for a flexible research tool, we also faced challenges involving portability. Our local cluster was too small to perform all the 3D inversions we were on working at the time, and other clusters were configured differently than our own. When starting out on a new HPC system, sometimes it took weeks of set-up work to simply run our codes. Although SeisFlows started as a research tool, some of the principles used to lend research flexibility also helped provide portability across computing environments.

Below, we discuss these flexibility and portability design goals in further details before moving on to a more comprehensive package overview and a practical demonstration with an industry benchmark.

2.1. *Flexibility*

A system in which new functionality can be added without compromising maintainability helps a package remain useful over time, even despite rapidly evolving research ideas or a high-turnover development team. Software engineering practices involving modular design and object-oriented programming are helpful for developing flexible software (e.g., Gamma et al., 1995). Such principles can be difficult to apply to scientific projects, however, because of challenges posed by legacy codes. An workable response, we found, was to develop a modular Python framework in which either legacy executables or native Python code can be used for key components. Besides giving options for integrating legacy solvers, Python also provides ease of use for domain scientists, powerful object-oriented capabilities, and a growing collection of scientific tools, including numpy for linear algebra (Walt et al., 2011) and obspy for seismic data processing (Krischer et al., 2015b).

2.2. *Portability*

For HPC applications, portability can be a daunting goal because cluster environments are extremely varied. An ability to work with different processor architectures, filesystems, memory configurations and job schedulers may be required for an HPC code to be portable.

The approach in SeisFlows is to provide an interface layer through which the workflow interacts with the system resources. To launch a set of forward simulations, for example, the user invokes the forward method of the solver via the run method of the system interface. By isolating environment-dependent attributes, the system interface provides a consistent command set across different computing environments.

3. Theory

As a way of introducing terminology used in the subsequent sections, we briefly review the theory of waveform inversion (also known as full-waveform inversion or FWI).

Waveform inversion is a data-fitting procedure in which comparisons between observed and synthetic data are used to image the internal structure of an object. Through a quantitative measure of fit between observations and synthetics, the inversion problem can be explicitly cast as a misfit function minimization problem. A common choice for the misfit function is

$$\chi(m) = \frac{1}{2}\sum_{i=1}^{N}\int\left|F[s_i(m;t)] - F[d_i(t)]\right|^2 \,\mathrm{d}t, \tag{1}$$

where s are synthetic data, d are observed data, F is a data processing operator, and the sum is taken over various sources and receivers ($i = 1, \ldots, N$). We generically call F a “preprocessing” operator to distinguish it from R in eq (2).

Given a model m of the material properties of the object of interest, it is the task of the wave-equation solver to generate the synthetic data s by numerically simulating wave propagation within the object (see Tromp, 2015 for an introduction).

Observed and synthetic data typically require some kind of signal processing to make meaningful comparisons. Often the operator F is simply a bandpass filter, but sometimes additional steps are required such as normalization, muting, denoising, or redatuming (see Yilmaz, 1987 for a comprehensive overview).

To iteratively update the model and improve the fit between observed and synthetic data, we apply a nonlinear optimization algorithm to the misfit function $\chi(m)$. With each iteration, a new model is generated from the previous model via

$$m_{k+1} = m_k + \alpha_k H_k^{-1} R[-g_k], \tag{2}$$

where g is the gradient vector of first-order derivatives of $\chi(m)$, and H is the Hessian matrix of second order derivatives of $\chi(m)$, or some approximation to the Hessian that depends on the chosen nonlinear optimization algorithm (see Nocedal and Wright, 2006 for a more thorough description). Finally, α is a step length determined by line search and R is a regularization, smoothing, or image processing operator that we generically refer to as a "postprocessing" operator.

4. Package organization

As illustrated in Fig. 1, SeisFlows consists of six components. While this structure is hardwired, the individual components themselves are highly customizable.

4.1. Solver

A tool for simulating wave propagation, the solver is the main computational engine underlying an inversion. In the course of a model update, the solver is first used to generate synthetic data s in Eq. (1). Then by backprojecting data residuals (e.g., Tromp et al., 2005; Fichtner et al., 2006), the solver is used to compute g in Eq. (2).

Choices for acoustic or elastic wave simulation in heterogeneous media include finite-difference, finite-element, and spectral-element numerical schemes. The type of solver best-suited to a particular inversion depends on whether the data involve body waves, surface waves or both; whether the topographic or bathymetric variations are small or large; and whether the material properties vary continuously or discontinuously within the target structure.

While in principle SeisFlows can interface with all the types of solvers mentioned above, to date, only spectral-element solver interfaces are included in the main repository. Interface classes for the **SPECFEM2D**, **SPECFEM3D** and **SPECFEM3D_GLOBE** packages are provided, supporting a wide variety of applications. All solver interfaces inherit from a common base class to avoid code duplication. Some researchers use SeisFlows in combination with finite-difference solvers (Gian Matharu, personal communication), but such functionality is not available yet in the main package.

4.2. Preprocessing

In our terminology, preprocessing refers to operations carried out on seismic traces, encompassing both $\chi(m)$ and $F[\cdot]$ in Eq (1). The name reflects the fact that such operations are usually performed prior to data residual backprojection.

The default preprocessing class works with time-domain data and relies on obspy for reading and writing data and signal processing. If a file format is not supported by obspy, users can contribute their own reading and writing utilities through a simple plugin system.

A variety of data processing options are included in the default class. Choices are provided for highpass, lowpass and bandpass filtering; trace-by-trace or record section-by-record section normalization; and muting early or late arrivals.

Finally, machinery for generating adjoint traces is provided for use with "optimize-then-discretize" solvers. Adjoint trace generators corresponding to waveform difference, traveltime, envelope, and instantaneous phase objective functions are provided. Support for other objective functions can be added, again through a plugin system.

4.3. Postprocessing

In our terminology, "postprocessing" refers to image processing operations on models, sensitivity kernels, or migrated images. The name reflects the fact that such operations are usually performed after data backprojection.

A wide variety of operations fall into this category. Techniques including smoothing, basis projection, and Tikhonov and total variation regularization all involve operations on sensitivity kernels and haven been implemented using SeisFlows (see Modrak and Tromp, 2016 for an overview). Treatment of site effects or numerical artifacts around sources or receivers also falls under postprocessing, as does spatial filtering or sharpening performed in oil and gas exploration contexts.

Because postprocessing operations are sometimes performed directly on the model or kernels as expressed in the finite-difference, finite-element or spectral element basis used by the solver, there may be some overlap between the solver and postprocessing component. In SeisFlows we use standalone utilities included in the **SPECFEM2D**, **SPECFEM3D** and **SPECFEM3D_GLOBE** packages for smoothing operations on spectral-element bases, rather than reimplementing these operations in Python.

4.4. Nonlinear optimization

What drives the progress of an inversion, both in terms of generating model updates and reducing data misfit, is the nonlinear optimization procedure. The rate of convergence in waveform inversion depends on the particular nonlinear optimization algorithm chosen.

In the main package, available nonlinear optimization algorithms include gradient descent, nonlinear conjugate gradient (NLCG) and quasi-Newton (QN) algorithms, with bracketing and backtracking line searches and restart safeguards included in the implementation (see Modrak and Tromp, 2016 for an overview). Default numerical settings should work well for a wide range inversions, but tuning parameters are also provided. To improve performance beyond convergence offered by NLCG and QN, preconditioners can be loaded through a plugin system. Finally, a integration test is provided that checks the nonlinear optimization machinery using the Rosenbrock test case (Rosenbrock, 1960).

While far less than the memory used by full Newton or Gauss–Newton methods, conjugate gradient and quasi-Newton algorithms use roughly 2 to 10 times more storage than a simple gradient descent method. In other words, the memory required by NLCG or QN is 2–10 times the memory required to store a single gradient vector. Although solver operations can be performed in parallel over any number of nodes, nonlinear

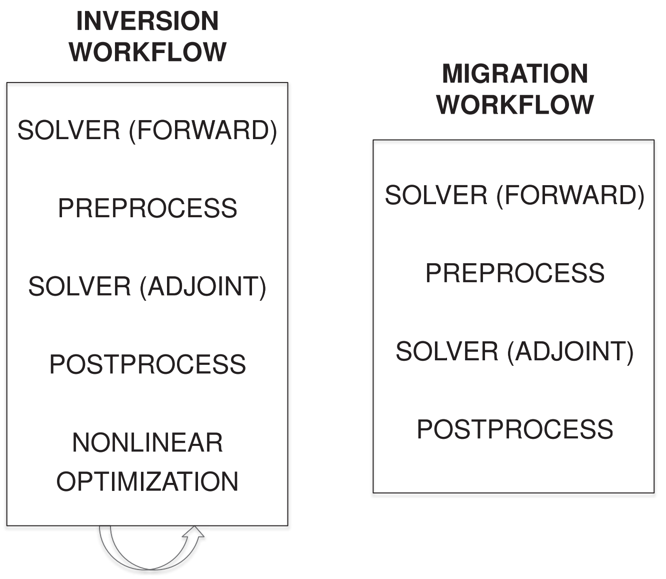

Fig. 2. Default inversion and migration workflows provided by SeisFlows. Because each component part is highly customizable, only a very schematic representation is shown above.

optimization operations using SeisFlows are currently performed on only a single node. If the nonlinear optimization storage cost exceeds the memory available on the node, vectors are stored in virtual memory using numpy.memmap. Parallelizing the nonlinear optimization routines could improve performance in very large inversions, but for reasons described in Sec. 4.5, this is not currently a priority.

4.5. *System*

As the software layer through which the solver and other workflow components interact with the system resources, the "system" component provides interoperability from one environment to another. A selection of interfaces is provided in the main package, including some for workstations and others for LSF, SLURM and PBS clusters. In all cases, the idea of the system component is to provide default configuration that allows for customization in case anything about the user's environment requires special handling. Examples of this type customization are available in github.com/rmodrak/seisflows-hpc.

Take the system.run function, used to execute embarrassingly parallel tasks, as an example. When running on a computer with a single available CPU core, users should select the serial system interface. The causes seisflows.system.serial.run, which executes tasks one at a time within a loop, to be loaded at runtime. Alternatively, when running, say, a small inversion on a SLURM cluster, users should choose the slurm_sm system interface. This causes seisflows.system.slurm_sm.run, which executes tasks simultaneously using resources allocated at the beginning of the

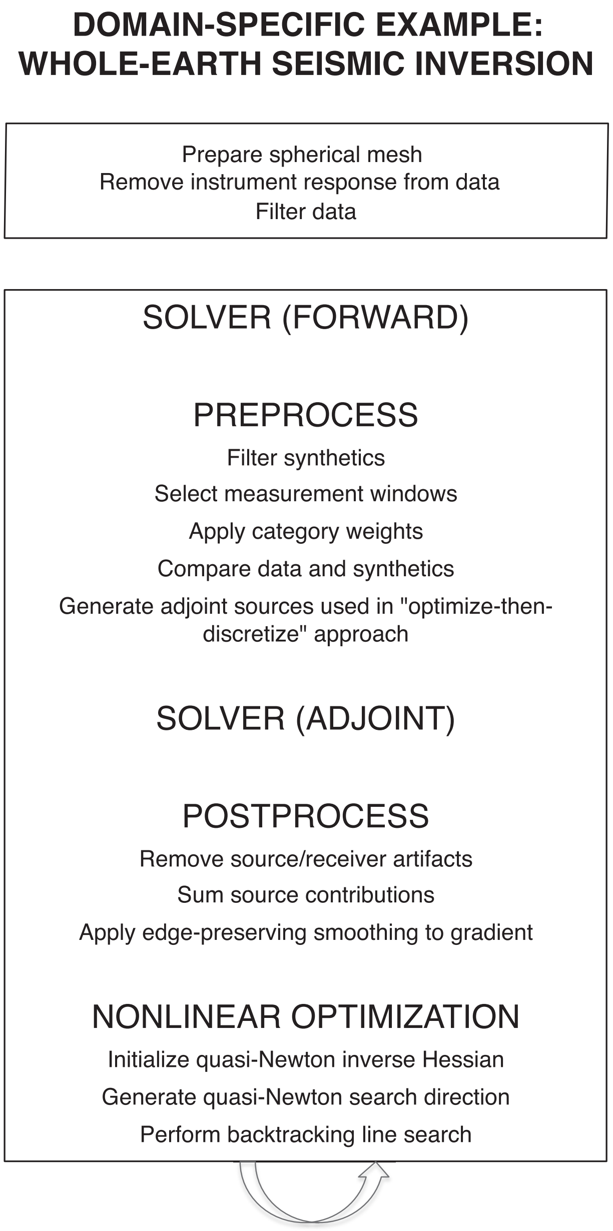
DOMAIN-SPECIFIC EXAMPLE:
WHOLE-EARTH SEISMIC INVERSION
Prepare spherical mesh
Remove instrument response from data
Filter data
SOLVER (FORWARD)
PREPROCESS
Filter synthetics
Select measurement windows
Apply category weights
Compare data and synthetics
Generate adjoint sources used in "optimize-then-discretize" approach
SOLVER (ADJOINT)
POSTPROCESS
Remove source/receiver artifacts
Sum source contributions
Apply edge-preserving smoothing to gradient
NONLINEAR OPTIMIZATION
Initialize quasi-Newton inverse Hessian
Generate quasi-Newton search direction
Perform backtracking line search

Fig. 3. One example of a domain-specific application that can be implemented in SeisFlows by overloading default classes. Special gradient smoothing operations can be used to address spatially uneven distribution of seismic stations and earthquakes in global seismology. To this end, users can choose from existing options within the postprocess category or provide their own custom postprocessing class. Steps listed in the upper box are carried out just once at the beginning of an inversion. Steps listed in the lower box are repeated until obtaining the desired level of misfit reduction.

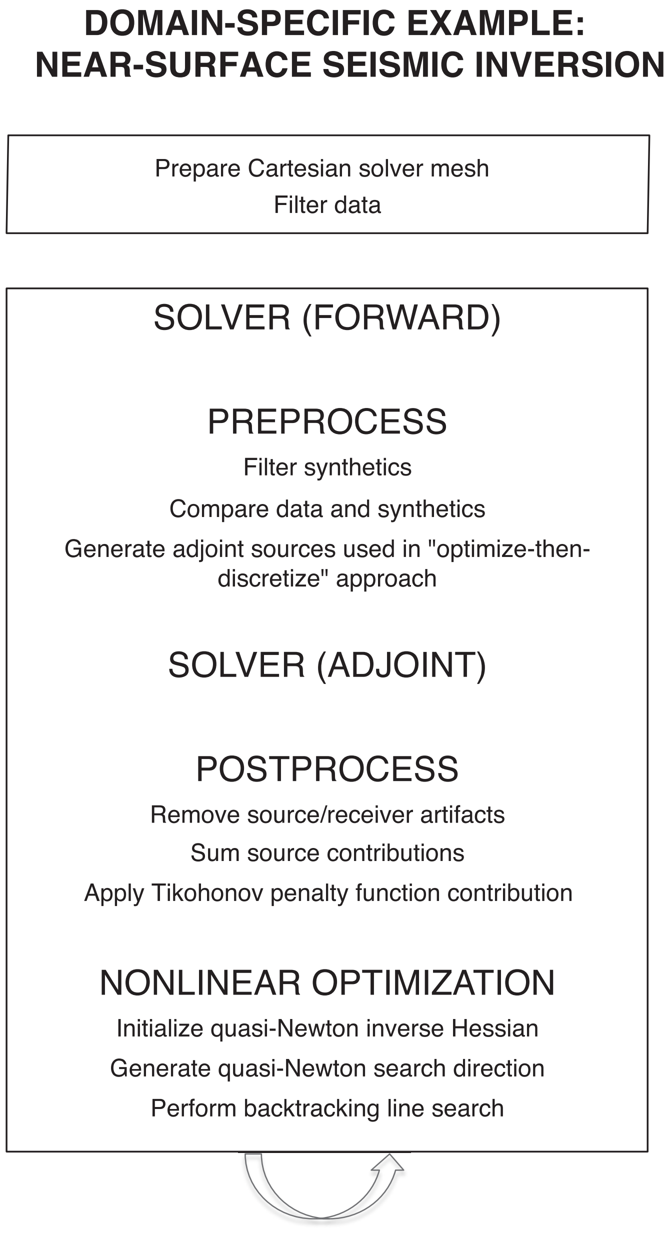
DOMAIN-SPECIFIC EXAMPLE:
NEAR-SURFACE SEISMIC INVERSION
Prepare Cartesian solver mesh
Filter data
SOLVER (FORWARD)
PREPROCESS
Filter synthetics
Compare data and synthetics
Generate adjoint sources used in "optimize-then-discretize" approach
SOLVER (ADJOINT)
POSTPROCESS
Remove source/receiver artifacts
Sum source contributions
Apply Tikohonov penalty function contribution
NONLINEAR OPTIMIZATION
Initialize quasi-Newton inverse Hessian
Generate quasi-Newton search direction
Perform backtracking line search

Fig. 4. Another example of a domain-specific application that can be implemented in SeisFlows by overloading default classes. Many specialized data preprocessing operations have been developed for near-surface data, which lack the well-behaved crust, mantle and core phases of whole-earth data. To this end, users can choose from existing options within the preprocess category or provide their own custom preprocessing class.

inversion, to be loaded. For larger inversions, users should select system interfaces ending with the lg suffix, which causes resources to be allocated on an as-needed basis rather than all at once at the beginning.

4.6. *Workflow*

Finally, the thing that ties everything together is the "workflow." Default inversion and migration workflows are provided that can be used directly or as a base class on top of which specialized strategies can be implemented.

In practice, execution of a workflow is equivalent to stepping through the code contained in workflow.main. Users are free to modify the default inversion and migration workflows, which are designed to be easily customizable. For example, by including initialize and finalize methods that can be used for any necessary setup or cleanup work before or after a model update.

Fig. 2 depicts default inversion and migration workflows. Because each component part is highly customizable, only a very schematic representation is provided. To give a sense for the type of domain-specific applications possible through SeisFlows, examples of more specialized whole-earth and near-surface inversion workflows are depicted in Figs. 3 and 4.

Reflecting our group's use of SeisFlows as a research tool, a number of custom strategies have been implemented by overloading the inversion and migration classes. Stochastic inversion, and double-difference waveform inversion, and full-waveform ambient noise inversion, for example, could all be implemented in this manner. Finally, in addition to carrying out inversions and migrations, users can script entirely new workflows by invoking the solver, preprocessing, postprocessing, and nonlinear optimization components in any desired sequence. Velocity analysis or uncertainty quantification tasks could be implemented in this way, to give some examples.

4.7. *Miscellaneous*

Here, we discuss miscellaneous aspects of package organization that fall outside the above categories.

4.7.1. *Tests*

Seisflows/tests contains a set of directly-invokable test scripts. run_test_system tests the system interface with a simple "hello" message, run_test_optimize solves an inexpensive nonlinear optimization problem, and run_test_tools executes the seisflows/tools unit tests.

4.7.2. *Source code repository*

Code is hosted and collaboratively developed using GitHub (github.com/rmodrak/seisflows). Following a pull request, the above tests are automatically run using Travis continuous integration (travis-ci.org). Documentation is viewable at seisflows.readthedocs.org.

4.7.3. *Plugins*

Each file or subdirectory in seisflows/plugins contains a set of functions that share exactly the same inputs and outputs syntax. New "plugins" can be added by creating a new function using the established syntax. For example, a new data misfit function can be added to seisflows/plugins/misfit.py by mimicking the inputs and outputs of the existing misfit functions. Because the syntax is kept very simple, adding new plugins generally requires little effort.

4.7.4. *Parameters and paths*

When submitting a workflow, users must specify parameter and path input files. Values are checked so that errors can be detected without loss of queue time or run time. Parameters and paths are stored in a dictionary that is accessible anywhere in the Python code.

5. *Example: onshore oil and gas exploration problem based on the SEAM foothills model*

Onshore seismic exploration has increased significantly in the last decade, but modeling wave propagation in complex environments remains difficult even for state-of-the art solvers (Oristaglio, 2012). With extreme topography typical of mountainous thrust zones and strong lateral and vertical velocity variations, the SEAM II foothills model is one of three benchmarks created by an industry consortium to address onshore modeling challenges (Oristaglio, 2013).

Below we present inversion results obtained using foothills data. Some of these results were previously published in Borisov et al. (2018); here we focus on computational cost, fault tolerance, and software flexibility issues not previously discussed.

5.1. *Methods*

To reduce the overall cost of the experiment, we selected a 25 km^2 portion of the original model. The chosen volume, shown in Fig. 5, is 7 $\times$ 3.5 $\times$ 3 km in the *x*-, *y*- and *z*-directions, respectively. Working in the 1–15 Hz range, some 10^8 numerical grid points were required for accurate wave simulation. Fig. 4 shows the model and numerical mesh. Topographic variations of as much as 0.9 km lead to significant scattering, focusing, and defocusing effects. The target model contains strong variations in *P*- and *S*-wavespeeds and density, with discontinuities in *S*-

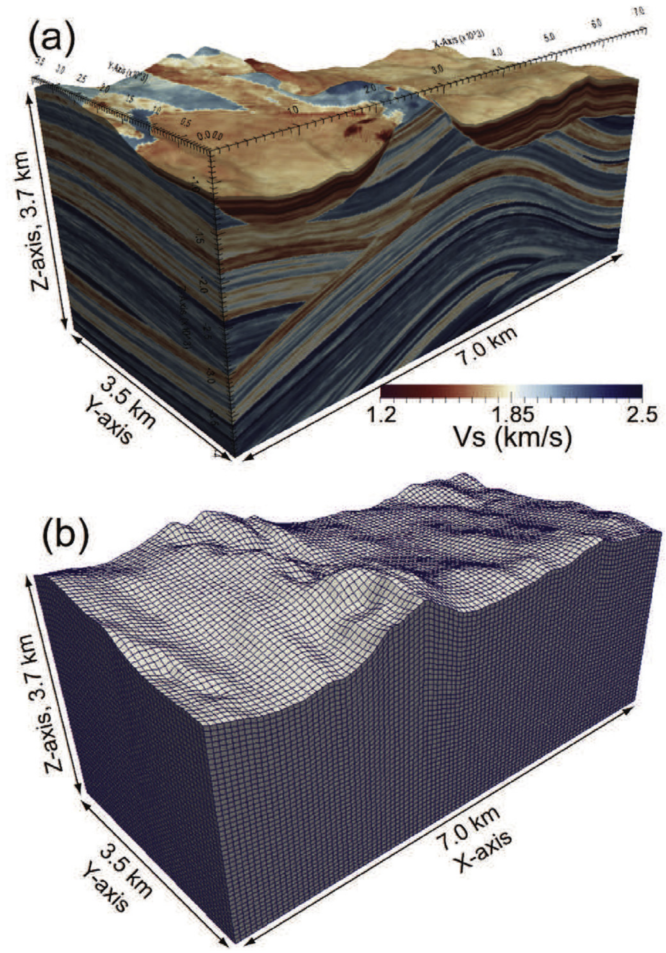

Fig. 5. Near-surface seismic inversion example. (a) Target *S*-wavespeed model. (b) Numerical mesh. Adding to the challenge and computational expense of the inversion, the target model contains large topographic variations and complex folded and faulted structures. The unstructured spectral-element numerical mesh contains on the order of 10^8 integration points.

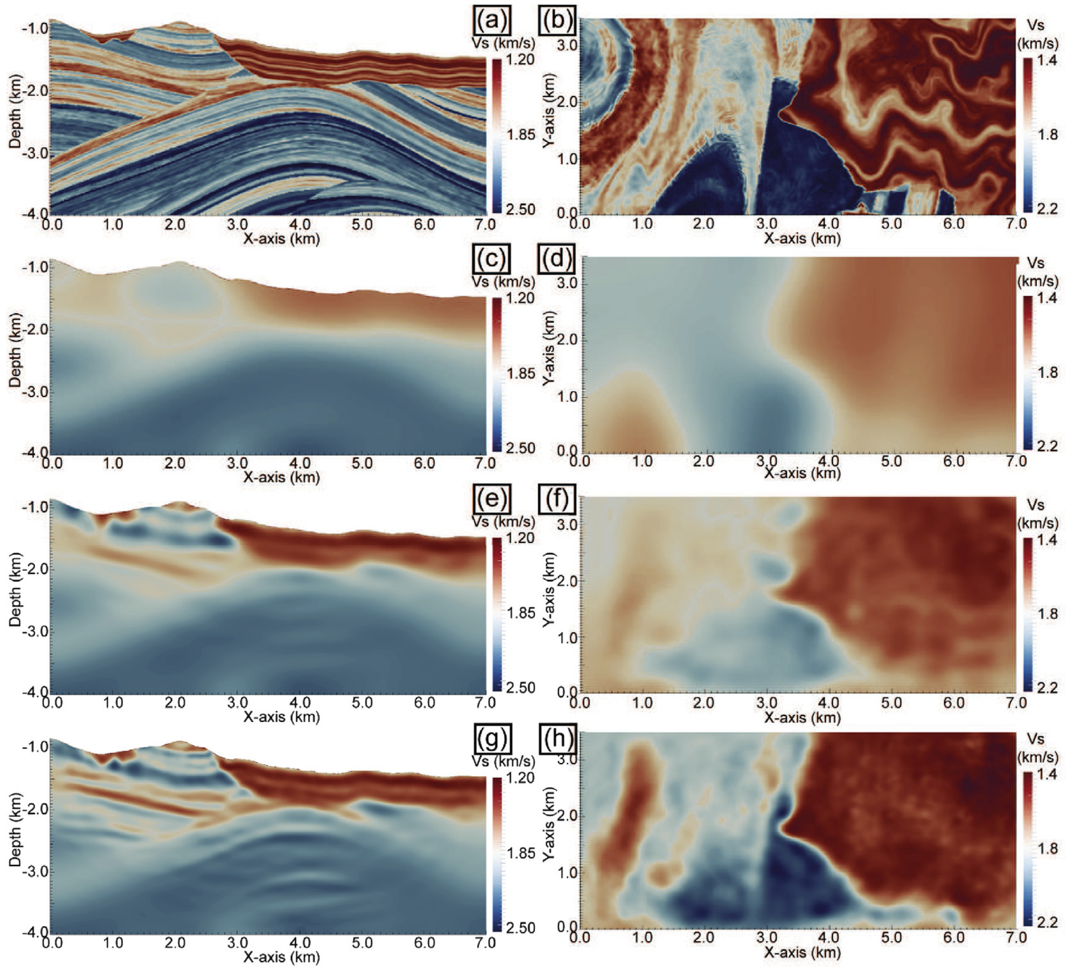

Fig. 6. Near-surface seismic inversion results. Vertical slices (left) and horizontal slices (right). (a,b) true model; (c,d) initial model; (e,f) result of envelope-difference inversion; (g,h) result of envelope-difference inversion followed by waveform-difference inversion.

wavespeed in some places of more than 1 km/s. The extreme topography of the foothills model has been shown to cause problems for finite-difference schemes, while finite-element and spectral-element codes such as SPECFEM3D have generally performed much better (Oristaglio, 2012).

In onshore seismic exploration, traces recorded at the surface are dominated by high-amplitude dispersive surface waves. Rather than removing such information from the data, surface waves can be used as an additional constraint on shallow structure. Unfortunately, several major obstacles, including difficulties in the numerical simulation of surface waves and problematic effects of near-surface complexity, make this approach difficult.

To make use of surface waves, flexibility is required in the inversion procedure. Most inversion packages employ waveform-difference measures of fit, but these are not well-suited to surface wave inversions in near-surface complexity exacerbates cycle skipping. To reduce cycle skipping, envelope-misfit functions have been proposed by Bozdag et al. (2011). As with many new techniques, envelope objective functions required fine-tuning before large-scale production. SeisFlows provided a useful framework for such testing.

For the inversion, we used 72 sources and 2502 receivers regularly distributed on the surface. The distance between receivers was 50 m and 200 m in the *x*- and *y*-directions, respectively, while the distance between shots was 600 m in both directions. Each shot corresponded to a force applied in the vertical direction with a Ricker wavelet as a source time function. We used a relatively high frequency band (with a dominant frequency of 6 Hz); we did not use a multiscale approach in the manner of Bunks et al. (1995). Further, we used only the vertical component for all receivers. Search directions were computed using the L-BFGS algorithm, which in SeisFlows comes with numerical safeguards that provide stability (e.g., Dennis and Schnabel, 1996).

Despite cropping the model, the inversion still exceeded the capacity of our local allocation, so we ran it on a 2000-core LSF machine provided by Total S.A. instead. Only minor modification of the default LSF interface in SeisFlows was necessary, with changes involving specification of the resource queue and MPI library paths. Memory requirements for the mesh exceeded the capacity of a single node, so we used multiple nodes for each wavefield simulation. With 6 nodes per source and 16 CPU cores per node, 96 cores were required for each source. Using seisflows/system/lsf_lg.py, simulations were queued in job arrays, executing whenever resources became available.

Due to decreasing hardware reliability with age, the rate at which simulations failed grew from less than 1 percent to more than 5 percent over the two years we used the cluster. To provide fault tolerance, we modified the lsf_lg interface so that (1) simulations were delayed by a random fraction of a second, avoiding large numbers of simultaneous I/O operations whenever multiple simulations left the queue at the same time; and (2) the status of every simulation in the job array was checked every few seconds, and failed simulations were automatically resubmitted as standalone jobs. A simple example of our approach to failure detection and automatic resubmission can be found in github.com/rmodrak/seisflows-hpc (see modules with "fault_tolerance" or "FT" in the filename). Whereas before only one or two iterations could be reliably carried out in one try, with the above measures we were able to run

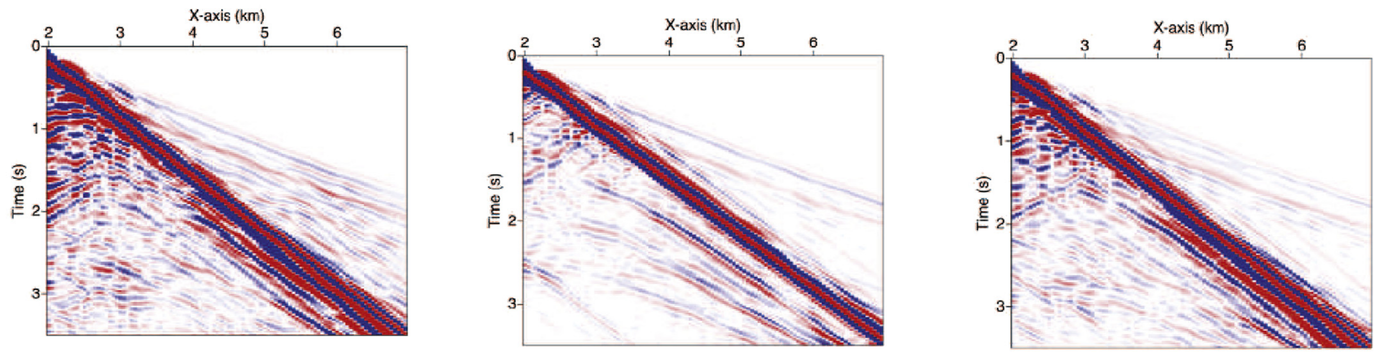

Fig. 7. A shot record of the SEAM Phase II foothills model, vertical component of displacement. *left:* observed traces, *center:* initial synthetics, *right:* synthetics after envelope FWI.

arbitrarily many iterations at a time.

5.2. Results

First, we used an envelope misfit function to simultaneously invert body and surface waves. After 30 iterations, accuracy of the S-wavespeed model increased significantly in the shallow subsurface (Fig. 6e and f). Then we used this result as input for a waveform-difference inversion, again using both body and surface waves. After a further 30 iterations shallow structure improved even more (Fig. 6g and h). With sensitivity concentrated at the free surface and with amplitudes many times greater than body waves, surface waves strongly dominate both the data and the model updates, so recovery of deep structures is not expected until surface waves are muted (Borisov et al., 2018). Having at this point recovered the shallow part of the model using body and surface waves, a conventional body wave-only inversion could be used to improve recovery at depth.

Record sections for the shot located at $x = 1$ km, $y = 1.75$ km are shown in Fig. 7. As the shallow part of the true model contains strong heterogeneities, the observed data are dominated by large-amplitude, dispersive Rayleigh waves. In contrast, the surface waves in the initial synthetics are much less dispersive because the starting model is relatively smooth. It is clear that the synthetic shot gather generated on a model inverted using envelope-based inversion agrees much more closely with the observed one.

6. Conclusions

Through the SEAM II foothills test case, we illustrated the use of SeisFlows on a difficult problem. Results in this example depended in part on the flexible design of the inversion software, which made experimentation with nonstandard objective functions possible. Success also depended on a portability between HPC environments, which allowed us to change over to supercomputer provided by Total S.A. after we had exceeded our local cluster allocation.

Besides the above oil and gas exploration benchmark, other uses of SeisFlows include detection of illicit tunnels (Smith et al., 2017); nondestructive testing of bridge deck delaminations (Nguyen and Modrak, 2018); medical imaging with ultrasonic waves (Bachmann, 2016); and oil and gas exploration with acoustic (Modrak and Tromp, 2015), isotropic elastic (Modrak et al., 2016) and transversely anisotropic models (Rusmanugroho et al., 2017). Earthquake tomography using data from Silwal and Tape (2016) is currently underway with the package.

Acknowledgments

The authors thank TOTAL S.A. for providing computational resources; the SEAM consortium for making available the foothills model; and Etienne Bachmann, Wenjie Lei, Gian Matharu, Luan Nguyen, Youyi Ruan, Heru Rusmanugroho, Frederik Simons, James Smith, Yanhua Yuan, and two anonymous reviewers for very helpful discussions.

References

Bachmann, E., 2016. Imagerie Ultrasonore 2D et 3D sur GPU: Application au temps réel et à l'inversion de forme d'onde complète. Ph.D. thesis. Université Toulouse.
Borisov, D., Modrak, R., Gao, F., Tromp, J., 2018. 3D elastic full-waveform inversion of surface waves in the presence of irregular topography using an envelope-based misfit function. Geophysics 83 (1), R1–R11.
Bozdag, E., Trampert, J., Tromp, J., 2011. Misfit functions for full waveform inversion based on instantaneous phase and envelope measurements. Geophys. J. Int. 185 (2), 845–870.
Bunks, C., Fatimetou, M., Zaleski, S., Chavent, G., 1995. Multiscale seismic waveform inversion. Geophysics 60, 1457–1473.
Burstedde, C., Ghattas, O., 2009. Algorithmic strategies for full waveform inversion: 1D experiments. Geophysics 74, WCC37–W3346.
Cockett, Rowan, Kang, Seogi, Heagy, Lindsey J., Pidlisecky, Adam, Oldenburg, Douglas W., 2015. SimPEG: An open source framework for simulation and gradient based parameter estimation in geophysical applications. Comput. Geosci. 85, 142–154. https://doi.org/10.1016/j.cageo.2015.09.015.
Dennis, J., Schnabel, R., 1996. Numerical Methods for Unconstrained Optimization and Nonlinear Equations. SIAM.
Fichtner, A., Bunge, H.-P., Igel, H., 2006. The adjoint method in seismology: I. theory. Phys. Earth Planet. In. 157 (1), 86–104.
Fomel, S., Sava, P., Vlad, I., Liu, Y., Bashkardin, V., 2012. Madagascar: open-source software project for multidimensional data analysis and reproducible computational experiments. J. Open Res. Software (1).
Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley.
Hewett, R., Demanet, L., 2013. The Python Seismic Imaging Toolbox. pysit.org.
Krischer, L., Fichtner, A., Zukauskaite, S., Igel, H., 2015a. Largescale seismic inversion framework. Seismol Res. Lett. 86 (4), 1198.
Krischer, L., Megies, T., Barsch, R., Beyreuther, M., Lecocq, T., Caudron, C., Wassermann, J., 2015b. Obspy: a bridge for seismology into the scientific python ecosystem. Comput. Sci. Discov. 8 (1), 014003.
Métivier, L., Brossier, R., 2016. The seiscope optimization toolbox: a large-scale nonlinear optimization library based on reverse communication. Geophysics 81 (2), F1–F15.
Modrak, R., Tromp, J., 2015. Computational Efficiency of Full Waveform Inversion Algorithms. SEG Technical Program, pp. 4838–4842.
Modrak, R., Tromp, J., 2016. Seismic waveform inversion best practices: regional, global and exploration test cases. Geophys. J. Int. 206 (3), 1864–1889.
Modrak, R., Yuan, Y., Tromp, J., 2016. On the Choice of Material Parameters for Elastic Waveform Inversion. SEG Technical Program, pp. 1115–1119.
Nguyen, L.T., Modrak, R.T., 2018. Ultrasonic wavefield inversion and migration in complex heterogeneous structures: 2D numerical imaging and nondestructive testing experiments. Ultrasonics 82, 357–370.
Nocedal, J., Wright, S., 2006. Numerical Optimization. Springer.
Oristaglio, M., 2012. Seam phase iiland seismic challenges. Lead. Edge 31 (3), 264.
Oristaglio, M., 2013. Seam phase ii: the foothills modelseismic exploration in mountainous regions. Lead. Edge 32 (9), 1020–1024.
Rosenbrock, H., 1960. An automatic method for finding the greatest or least value of a function. Comput. J. 3, 175–184.
Rusmanugroho, H., Modrak, R., Tromp, J., 2017. Anisotropic full-waveform inversion with tilt-angle recovery. Geophysics 82 (3), R135–R151.
Silwal, V., Tape, C., 2016. Seismic moment tensors and estimated uncertainties in southern Alaska. J. Geophys. Res.: Solid Earth 121 (4), 2772–2797, 2015JB012588.
Smith, J., Borisov, D., Modrak, R., Tromp, J., Cudney, H., Moran, M., Sloan, S., Miller, R., Peterie, S., 2017. Near-surface Seismic Imaging of Tunnels Using 3D Elastic Full-waveform Inversion. SEG Technical Program, pp. 2637–2641.
Tromp, J., 2015. Forward modeling and synthetic seismograms: 3D numerical methods. In: Schubert, G. (Ed.), Treatise on Geophysics, second ed. Elsevier, Oxford, pp. 231–251.

Tromp, J., Tape, C., Liu, Q., 2005. Seismic tomography, adjoint methods, time reversal and banana-doughnut kernels. Geophys. J. Int. 160, 195–216.

Virieux, J., Optero, S., 2009. An overview of full-waveform inversion in exploration geophysics. Geophysics 74.

Walt, S.v.d., Colbert, S.C., Varoquaux, G., Mar. 2011. The numpy array: a structure for efficient numerical computation. Comp. Sci. Eng. 13 (2), 22–30.

Yilmaz, Ö., 1987. Seismic Data Processing. Society of Exploration Geophysicists.