Accelerating innovation with software abstractions for scalable computational geophysics

Mathias Louboutin[*,1], *Philipp Witte*[2], *Ali Siahkoohi*[1], *Gabrio Rizzuti*[3], *Ziyi Yin*[1], *Rafael Orozco*[1] *and Felix J. Herrmann*[1]
[1] *Georgia Institute of Technology,* [2] *Microsoft Research,* [3] *Utrecht University*

SUMMARY

We present the SLIM open-source software framework for computational geophysics, and more generally, inverse problems based on the wave-equation (e.g., medical ultrasound). We developed a software environment aimed at scalable research and development by designing multiple layers of abstractions. This environment allows the researchers to easily formulate their problem in an abstract fashion, while still being able to exploit the latest developments in high-performance computing. We illustrate and demonstrate the benefits of our software design on many geophysical applications, including seismic inversion and physics-informed machine learning for geophysics(e.g., loop unrolled imaging, uncertainty quantification), all while facilitating the integration of external software.

INTRODUCTION

Software development for exploration geophysics has been traditionally driven by performance. Though this has resulted in very efficient code, the usability and portability of these codes has been compromised as a result of the absence of user-level design. Abstractions and user interfaces at a high level have been developed in recent decades to facilitate research and development. A number of such interfaces are available, ranging from programming languages, e.g., Python (van Rossum and Drake, 2009), Julia (Bezanson et al., 2017), and Matlab, to domain-specific languages (DSLs), including RVL (Padula et al., 2009), Firedrake (Rathgeber et al., 2016), and Devito (Louboutin et al., 2019; Luporini et al., 2020). Additionally, there has been a community initiative towards open-source codes and reproducibility, led by Madagascar (Madagascar, 2017). These efforts have brought attention to the need for user abstraction to easily translate mathematics into code without having to manually refactor thousands of lines of code.

Motivated by this background, we introduce our fully open-source software SLIM framework based on high-level abstractions and separation of concerns. Our software design relies on three principles: (1) A high-level abstraction that represents the mathematical problem at hand; (2) Scalability with vertical integration of high-performance software and DSLs; (3) Interoperability through the adoption of language standards (object oriented, multiple dispatch, inheritence, ...). In the following, we will detail these three points, highlighting their importance and implementation. We will follow with some illustrations of the integration with external software to demonstrate how research and development can be eased and potentially accelerated by these design principles.

SOFTWARE DESIGN

The fundamental objective of a scientific software framework is to provide users with an interface that allows them to express their own scientific problems. In particular, the interface should provide abstractions to define the problem as closely as possible to its mathematical formulation. With this design philosophy, research and development turnaround time can be drastically reduced with quick prototyping and testing of new ideas and algorithms. Our software framework is grounded in this idea and led to the development of legacy software (Lin and Herrmann, 2015; Silva and Herrmann, 2019) adopted by industry for real-world application. Based on these ideas and modern programming paradigms, we designed a high-level framework that encapsulates the mathematical definition of geoscientific problems at every level. These different levels are handled through the vertical integration of domain-specific languages (DSLs). At the lowest level, Devito provides a symbolic DSL for the definition of wave-equation and a just-in-time compiler to target the available hardware with near-optimal performance. On top of Devito, we developed JUDI, a linear algebra DSL for wave-equation based modeling and inversion. Finally, we created a machine learning framework that allows us to integrate Devito propagators and JUDI Linear Operators into machine learning (ML) frameworks (PyTorch and Flux) opening the door to ML-augmented geophysical inverse problems and uncertainty quantification (UQ). Finally, throughout these different layers, we committed to follow language standards allowing easy interfacing and integration with external software and frameworks.

WAVE-EQUATION BASED INVERSION

Linear operators are at the core of applied geophysics. Some relevant examples include data filtering tools such as Fourier-based f-k filters, the Radon transform, NMO corrections to traditional imaging operator such as Kirchhoff migration, post-stack migration and wave-equation based inversion where the discrete wave-equation is a linear operator. One of the first frameworks for abstract matrix-free linear algebra was sPOT(van den Berg and Friedlander, 2009), later extended to wave-equation based inversion and distributed computing for wavefield reconstruction (Kumar, 2010). While its adoption was limited by its implementation in Matlab, such user oriented abstraction laid the ground for modern abstracted frameworks such as `JOLI`(Modzelewski et al., 2022) in Julia and `pyLops` in Python (Ravasi and Vasconcelos, 2020).

Similarly, wave-equation based inversion can be trivially formulated as simple least-squares optimization problem for the non-linear wave-equation represented as an abstract matrix $\mathbf{A}(\mathbf{m})$ parametrized by the model parameter $\mathbf{m}$:

$$\underset{\mathbf{m}}{\text{minimize}} \quad \Phi(\mathbf{m}) = \sum_{i=1}^{n_s} ||\mathbf{P}_r \mathbf{A}(\mathbf{m})^{-1} \mathbf{P}_s^\top \mathbf{q}_i - \mathbf{d}_i||_2^2,$$

Building on modern software solutions JUDI (Witte et al., 2019a; Louboutin et al., 2022a) provides a high-level interface allowing to define a solver for this problem with a few lines of code. For example, FWI using Gauss-Newton updates at each iteration can be summarized in as little as five lines (Listing 1).

```
# Gauss-Newton method
F = judiModeling(model, srcGeom, recGeom)
J = judiJacobain(F, q)
for j=1:maxiter
    d_pred = F*q
    fhistory_GN[j] = .5f0*norm(d_pred - d_obs)^2
    # Gauss-Newton update
    p = lsqr(J, d_pred - d_obs)
    model0.m .= proj(model0.m .- reshape(p, model0.n))
end
```

Listing 1: FWI with Gauss-Newton updates using JUDI. The complete example is available and reproducible in the JUDI github repository.

In addition to offering a high-level, mathematical interface to the wave-equation and related linear operators, JUDI builds on multiple layers of abstractions while still providing computational performance that is comparable to the state-of-the-art(Luporini et al., 2020). JUDI integrates Devito as a backend for the actual wave-equation solves. Devito is a stencil DSL (Louboutin et al., 2019; Luporini et al., 2020) that offers a symbolic user-level interface to the underlying numerical solver. This symbolic interface allows the user to easily and mathematically define generic partial differential equations (PDEs), such as the acoustic, tilted-transverse-isotropic (TTI), or elastic wave-equations. The underlying software then provides a code generation framework than can generate highly-performant code for numerous architectures, (CPU, GPUs, ARM, POWER). Its ease of use and high-level interface have contributed to its adoption in the oil-and-gas industry for production and research at scale (Washbourne et al., 2021). We summarize the vertical integration of the different technologies (compiler, Devito, JUDI, . . .) in Figure 1, which highlights the different levels of abstraction. The 2007 BP TTI dataset, for example, can be setup and processed with RTM within days. The results are shown in Figure 2.

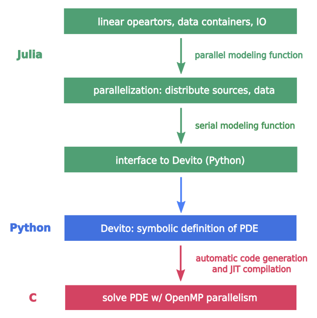
linear opeartors, data containers, IO
Julia
parallel modeling function
parallelization: distribute sources, data
serial modeling function
interface to Devito (Python)
Python
Devito: symbolic definition of PDE
automatic code generation and JIT compilation
C
solve PDE w/ OpenMP parallelism

Figure 1: Schematics of the vertical integration in JUDI that enables at scale inversion through layers of abstractions.

Interoperability

While a proper separation of concerns can lead to highly usable, portable, and performant software, another important aspect to consider is interoperability, namely the ability to integrate software from different sources and organizations. We now describe how the design of our software framework allows for easy interfacing with external frameworks. One major drawback of proprietary and low-level software is the limited capability to interface and interact with external software. Through open-source code development, and committing to language-specific standards (such as proper usage of types, hierarchy, inheritance, etc.) in Julia and Python, we enable seamless interoperability with external software and frameworks. This greatly increases the ability to perform research, which involves being able to easily test and combine different ideas. For instance, we combine core Julia packages, our numerical optimization toolbox SlimOptim.jl (Louboutin and Herrmann, 2021a), a constrained optimization software SetIntersectionProjection.jl (Peters and Herrmann, 2019), and COFII's wave-equation propagators (Washbourne et al., 2021), in order to setup a full waveform inversion (FWI) exercise. We were able to perform FWI, in parallel, on the Marmousi-II model by taking advantage of the best technology available. Additionally, this demonstrative example, and in general JUDI (or COFII), can trivially be deployed in the cloud using once again dedicated software abstractions (Washbourne et al., 2021; Louboutin et al., 2022b).

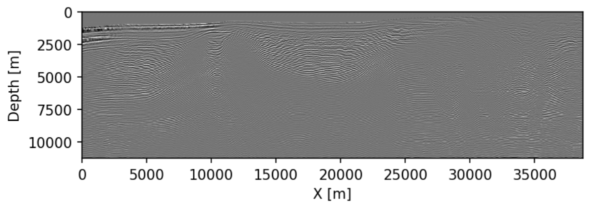

Figure 2: 2D RTM on the 2007 BP TTI model with a marine acquisition. This RTM was run on a GPU without moving off to the CPU at any time using randomized trace estimation as an extension of JUDI (Louboutin and Herrmann, 2021b; Louboutin, 2021).

```
function objective(F, velocity, d_obs)
    J = jacobian(F, velocity)
    d_pred = F * velocity # Forward modeling with COFFI
    G = J' * ({d_pred.- d_obs) # Gradient with COFII
    f = .5f0*norm(d_pred .- d_obs)^2 # Misfit
    return f, G
end
# Projection on TV+bounds, SetIntersectionProjection.jl
prj = setup_projection(...)
# Projected Quasi-Newton (l-BFGS) with SlimOptim.jl
sol = pqn(x->objective(F, x, d_obs), vec(v2), prj);
```

Listing 2: Projected Quasi-Newton FWI with multiple software packages. The complete example can be found here

Since different software modules can be pieced together with minimal effort, it greatly aids the development and testing of new research. This flexibility allows, for instance, the integration of machine learning into conventional algorithm pipelines in a plug-and-play fashion. In the following section, we provide some notable examples of this approach.

MACHINE LEARNING

Thanks to recent theoretical and practical advances, deep learning has seen a wide adoption in various geophysical applications ranging from processing (e.g., denoising, multiple removal) to inverse problems and seismic interpretation (e.g., Zhang et al., 2018; Wang et al., 2019; Yang and Ma, 2019). One of the core challenges in the adoption of deep learning for seismic applications is the integration of existing codes (e.g. seismic propagators) into deep-learning frameworks based on automatic differentiation (AD). To be able to differentiate through networks that contain both standard PyTorch/Tensorflow layers, as well as third party functions, such as a forward modeling kernel, the third party code must have manually implemented gradients and must be properly integrated into the deep learning framework. Our high-level linear algebra abstraction framework, which sits on top of our Devito-based propagators, facilitates this integration, as deep learning frameworks like PyTorch or Flux are already able to backpropagate through a linear operator (namely by applying its adjoint to the data residual). This allows us to implement deep neural networks in Flux, in which we can combine standard deep learning layers with external third party functions, e.g. convolutional layers with migration/demigration operators. This capability has enabled various research projects in our group, including surface-related multiple elimination (Siahkoohi et al., 2019c), dispersion attenuation (Siahkoohi et al., 2019a), and ML-augmented imaging and uncertainty quantification (Siahkoohi et al., 2019b, 2020). We show in the following two examples of machine learning for exploration geophysics that take advantage of these high-level abstractions to interface PDE solvers (JUDI, Devito) with deep learning frameworks.

Seismic imaging with deep priors and uncertainty quantification

Since we can integrate Devito's highly optimized wave-equation solvers with the PyTorch deep learning library, we propose to use deep priors (Lempitsky et al., 2018) to regularize seismic imaging, where we reparameterize the seismic image as the output of an untrained convolutional neural network (CNN). This approach acts as a regularization in the image space (Lempitsky et al., 2018; Cheng et al., 2019; Dittmer et al., 2020), which exploits the inductive bias of the CNN in representing images without noisy artifacts (Lempitsky et al., 2018). We perform Bayesian inference via a gradient-based Markov chain Monte Carlo (MCMC) algorithm (Welling and Teh, 2011), where each iteration requires differentiating the action of the linearized Born scattering operator on the CNN output with respect to the CNN's weights. In order to have access to the automatic differentiation utilities of PyTorch, we expose Devito's matrix-free implementations for the migration operator and its adjoint to PyTorch via Devito4PyTorch (Siahkoohi and Louboutin, 2021). This is exemplified in Listing 3. Figure 3 summarizes the results of seismic imaging and uncertainty quantification with this approach, which are borrowed from Siahkoohi et al. (2021).

```
def deep_prior_imaging(d_obs, n, maxiter=1000):
    # PyTorch Integrated Devito Jacobian.
    J = ForwardBornLayer(model, geometry)
    # Initialize deep prior CNN with random input.
    G = deep_prior_net()
    z = torch.randn([1, 1, n[1], n[2]])
    # Setup pSGLD MCMC sampler.
    optim, samples = pSGLD(G.parameters(), lr=0.001), []
    for i in range(maxitr):
        # Compute predicted perturbation model and data.
        x = G(z)
        d_pred = J(x)
        # Compute the negative log-posterior.
        nlp = n_log_posterior(d_pred, d_obs, G.↩
            parameters())
        # Compute the gradient w.r.t. the CNN weights,
        nlp.backward()
        # Sample with pSGLD.
        optim.step()
        samples.append(x.detach().numpy())
    return samples
```

Listing 3: Seismic imaging and uncertainty quantification with PyTorch and Devito as a wave-equation solver.

Loop-unrolled seismic imaging

Another set of applications that are enabled by the proper integration of abstract seismic modeling operators into deep learning frameworks are loop-unrolled optimization algorithms such as the learned primal-dual reconstruction (Andrychowicz et al., 2016; Adler and Öktem, 2018). These are special types of networks that follows the general structure of gradient-based optimization algorithms, in which conventional gradients are augmented by additional neural network layers. Using our integration of JUDI operators into Flux via the JUDI4Flux package (Witte et al., 2019b), we apply the loop-unrolled network architecture from (Adler and Öktem, 2017) to seismic imaging. Every iteration of the loop-unrolled algorithm consists of computing the (conventional) gradient of the LS-RTM objective function g, using the forward and adjoint linearized Born scattering operator, followed by the application of a shallow CNN (Listing 4). The network takes a single simultaneous (super-) shot record as the input and predicts an LS-RTM (or true) image. We train the network in a supervised fashion, in which we minimize the misfit between the predicted image and the true image (whereas in reality, one would train with available LS-RTM images). Figure 4 shows the output of the loop-unrolled gradient descent algorithm, Listing 4, after training the network for 2000 iterations on a training dataset of 2,000 data-image pairs. The example is not meant to represent a realistic scenario (as the true images which were used in the training process are obviously unknown), but serve as a proof of concept on how to augment physical models with data-driven approaches. The complete example and additional variations are available at LoopUnrolledSeismicImaging.

```
function loop_unrolled_lsrtm(d_obs, n; maxiter=10)
    x = randn(Float32, n[1], n[2], 1, 1)
    nx, ny, nc, nb = size(x)
    s = randn(Float32, nx, ny, 5, nb) # memory term
    for j=1:maxiter
        g = adjoint(J)*(J*vec(x) - d_obs)
        g = reshape(Flux.normalise(g), nx, ny, 1, 1)
        u = cat(x, g, s, dims=3)
        u = bn1(conv1(u)); u = bn2(conv2(u)); u = bn3(↩
            conv3(u));
        s = relu.(u[:, :, 2:6, :])
        dx = u[:, :, 1:1, :]
        x += dx
    end
    return vec(x)
end
```

Listing 4: Example of a physics-augmented neural network for seismic imaging. The network consists of 10 iterations of a loop-unrolled gradient descent algorithm, in which the conventional LS-RTM gradient g is augmented through convolution layers. The input into the network is the observed seismic data and the output is the predicted image.

End-to-end inversion for geological carbon storage monitoring

Finally, we show that we can easily build a framework for seismic monitoring of geological carbon storage that integrates PDE solvers, deep learning models and physical constraints. These monitoring problems rely on three different types of

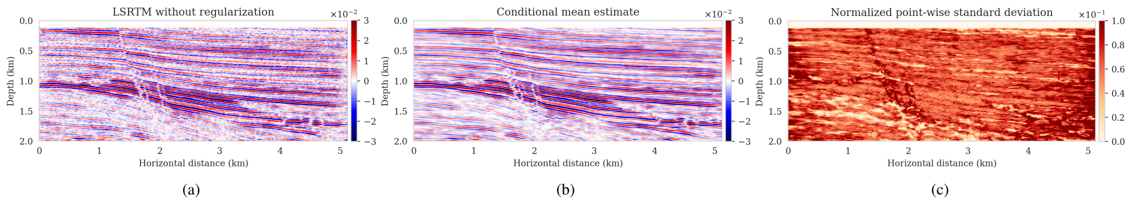
Figure 3: Imaging a 2D subset of the Parihaka (WesternGeco., 2012) dataset with deep priors (Siahkoohi et al., 2021). (a) LSRTM without any regularization. (b) The conditional (posterior) mean estimate, using the deep prior as regularization. (c) Normalized pointwise standard deviation.

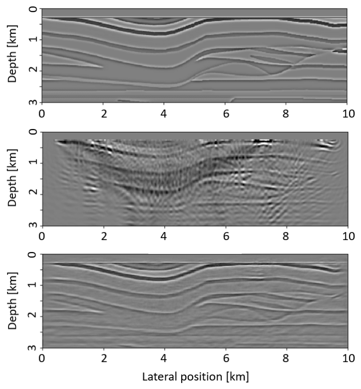
Figure 4: Loop-unrolled LSRTM on a test 2D slice of the 2D overthrust model. (a) True image, (b) RTM of simultaneous shot and (c) loop-unrolling of.

physics (Li et al., 2020a): fluid-flow physics, rock physics and wave physics. Given time-lapse seismic data collected over multiple years, we jointly invert for the rocks intrinsic permeability which can be used to recover the CO_2 concentration. By leveraging our high-level abstractions and AD, we can directly differentiate through all three physics solvers which map permeability to seismic data and minimize a fully coupled data misfit objective function. At each iteration, given an estimate of the permeability K, we model seismic data where the subsurface velocity is translated from the solution of the fluid-flow simulation. We can then compute the standard data misfit and use automatic differentiation to compute the permeability update. Because each step is abstracted, we can easily swap each physical solver with a different one, such as replacing the fluid-flow solver by a trained Fourier Neural Operator (FNO) for computational efficiency (Li et al., 2020b). We show in Figure 5 that we can recover the permeability from seismic measurements using a pre-trained FNO in place of a fluid-flow solver.

CONCLUSIONS

Through this paper, we have introduced a software design philosophy aimed at enabling research and development at scale in geoscience, with the potential to generalize to any inverse problem such as medical imaging. We demonstrated through carefully chosen example that high-level abstractions allow to express complex problem in a clear and representative way without incurring additional computational costs. Our software framework already allows for a wide range of application, including industry scale inversion and cutting-edge physics-informed deep learning for geophysics. We intend to build additional capabilities to tackle modern computing environment such as Cloud computing, task dedicated accelerators (i.e TPUs) and non-convex optimization.

```
opt = RMSprop()
for j=1:maxiter
    theta = params(K)
    grads = gradient(theta) do
        c = S(K); v = R(c); d_pred = F(v)
        return 0.5f0 * norm(d_pred-d_obs)^2f0
    end
    for p in theta
        update!(opt, p, grads[p])
    end
end
```

Listing 5: Example of an end-to-end coupled inversion for seismic monitoring of geological carbon storage. A pre-trained FNO S is used as a surrogate for fluid-flow simulation. In each iteration, we calculating the seismic data misfit, compute the gradient with respect to input permeability, K, via AD, and update it according to an optimizer *opt*.

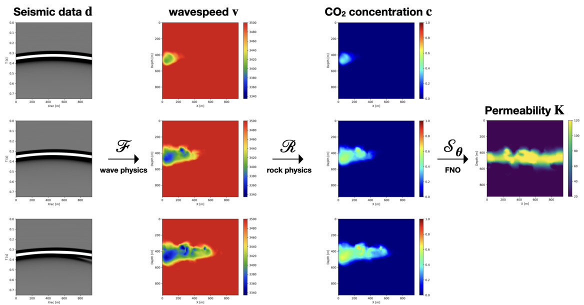

Figure 5: End-to-end inversion of CO_2 concentration from seismic measurements. In order to obtain this end-to-end result, the three operators representing the fluid-flow modeling, rock property modeling, wave modeling, and their derivatives are integrated by combining AD and manually defined gradients.

ACKNOWLEDGMENT

This research was carried out with the support of Georgia Research Alliance and partners of the ML4Seismic Center. We thank J. Washbourne at CVX for the fruitful discussions on open source software development and interoperability.

REFERENCES

Adler, J., and O. Öktem, 2017, Solving ill-posed inverse problems using iterative deep neural networks: Inverse Problems, **33**, 124007.

———, 2018, Learned primal-dual reconstruction: IEEE transactions on medical imaging, **37**, 1322–1332.

Andrychowicz, M., M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, 2016, Learning to learn by gradient descent by gradient descent: Advances in neural information processing systems, **29**.

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah, 2017, Julia: A fresh approach to numerical computing: SIAM Review, **59**, 65–98.

Cheng, Z., M. Gadelha, S. Maji, and D. Sheldon, 2019, A Bayesian Perspective on the Deep Image Prior: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 5443–5451.

Dittmer, S., T. Kluth, P. Maass, and D. Otero Baguer, 2020, Regularization by architecture: A deep prior approach for inverse problems: Journal of Mathematical Imaging and Vision, **62**, 456–470.

Kumar, N., 2010, Parallelizing operations with ease using parallel spot.

Lempitsky, V., A. Vedaldi, and D. Ulyanov, 2018, Deep Image Prior: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 9446–9454.

Li, D., K. Xu, J. M. Harris, and E. Darve, 2020a, Coupled time-lapse full-waveform inversion for subsurface flow problems using intrusive automatic differentiation: Water Resources Research, **56**, e2019WR027032.

Li, Z., N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, 2020b, Fourier neural operator for parametric partial differential equations.

Lin, T. T., and F. J. Herrmann, 2015, The student-driven hpc environment at slim: Presented at the Inaugural Full-Waveform Inversion Workshop. ((Natal, Brazil)).

Louboutin, M., 2021, slimgroup/timeprobeseismic.jl.

Louboutin, M., and F. Herrmann, 2021a, slimgroup/slimoptim.jl: v0.1.6.

Louboutin, M., and F. J. Herrmann, 2021b, Ultra-low memory seismic inversion with randomized trace estimation: SEG Technical Program Expanded Abstracts, 787–791. ((IMAGE, Denver)).

Louboutin, M., M. Lange, F. Luporini, N. Kukreja, P. A. Witte, F. J. Herrmann, P. Velesko, and G. J. Gorman, 2019, Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration: Geoscientific Model Development, **12**, 1165–1187.

Louboutin, M., P. Witte, Z. Yin, H. Modzelewski, and C. da Costa, 2022a, slimgroup/judi.jl: v2.6.4.

Louboutin, M., Z. F. Yin, and F. Herrmann, 2022b, slimgroup/judi4cloud.jl: First public release.

Luporini, F., M. Louboutin, M. Lange, N. Kukreja, P. Witte, J. Hückelheim, C. Yount, P. H. J. Kelly, F. J. Herrmann, and G. J. Gorman, 2020, Architecture and performance of devito, a system for automated stencil computation: ACM Trans. Math. Softw., **46**.

Madagascar, 2017.

Modzelewski, H., M. Louboutin, Z. F. Yin, and R. Orozco, 2022, slimgroup/joli.jl: v0.7.16.

Padula, A. D., S. D. Scott, and W. W. Symes, 2009, A software framework for abstract expression of coordinate-free linear algebra and optimization algorithms: ACM Trans. Math. Softw., **36**.

Peters, B., and F. J. Herrmann, 2019, Algorithms and software for projections onto intersections of convex and non-convex sets with applications to inverse problems: arXiv preprint arXiv:1902.09699.

Rathgeber, F., D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. Mcrae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, 2016, Firedrake: Automating the finite element method by composing abstractions: ACM Trans. Math. Softw., **43**.

Ravasi, M., and I. Vasconcelos, 2020, Pylops—a linear-operator python library for scalable algebra and optimization: SoftwareX, **11**, 100361.

Siahkoohi, A., and M. Louboutin, 2021, slimgroup/devito4pytorch: Initial release.

Siahkoohi, A., M. Louboutin, and F. J. Herrmann, 2019a, The importance of transfer learning in seismic modeling and imaging: Geophysics.

———, 2019b, Neural network augmented wave-equation simulation: Technical Report TR-CSE-2019-1, Georgia Institute of Technology.

Siahkoohi, A., G. Rizzuti, and F. J. Herrmann, 2020, A deep-learning based bayesian approach to seismic imaging and uncertainty quantification: Presented at the EAGE Annual Conference Proceedings.

———, 2021, Deep bayesian inference for seismic imaging with tasks: arXiv preprint arXiv:2110.04825.

Siahkoohi, A., D. J. Verschuur, and F. J. Herrmann, 2019c, Surface-related multiple elimination with deep learning: SEG Technical Program Expanded Abstracts, 4629–4634.

Silva, C. D., and F. J. Herrmann, 2019, A unified 2d/3d large scale software environment for nonlinear inverse problems: ACM Transactions on Mathematical Software.

van den Berg, E., and M. P. Friedlander, 2009, Spot: A linear-operator toolbox for matlab: Presented at the SCAIM, SCAIM Seminar, SCAIM Seminar.

van Rossum, G., and F. L. Drake, 2009, Python 3 reference manual: CreateSpace.

Wang, B., N. Zhang, W. Lu, and J. Wang, 2019, Deep-learning-based seismic data interpolation: A preliminary result: Geophysics, **84**, V11–V20.

Washbourne, J., S. Kaplan, M. Merino, U. Albertin, A. Sekar, C. Manuel, S. Mishra, M. Chenette, and A. Loddoch, 2021, *in* Chevron optimization framework for imaging and inversion (COFII) – An open source and cloud friendly Julia language framework for seismic modeling and inversion: 792–796.

Welling, M., and Y. W. Teh, 2011, Bayesian Learning via Stochastic Gradient Langevin Dynamics: Proceedings of the 28th International Conference on Machine Learning, Omnipress, 681–688.

WesternGeco., 2012, Parihaka 3D PSTM Final Processing Report: Technical Report New Zealand Petroleum Report 4582, New Zealand Petroleum & Minerals, Wellington.

Witte, P. A., M. Louboutin, N. Kukreja, F. Luporini, M. Lange,

G. J. Gorman, and F. J. Herrmann, 2019a, A large-scale framework for symbolic implementations of seismic inversion algorithms in julia: Geophysics, **84**, F57–F71. ((Geophysics)).

Witte, P. A., M. Louboutin, and F. J. Herrmann, 2019b, Judi4flux: Seismic modeling for deep learning.

Yang, F., and J. Ma, 2019, Deep-learning inversion: A next-generation seismic velocity model building method: Geophysics, **84**, R583–R599.

Zhang, G., Z. Wang, and Y. Chen, 2018, Deep learning for seismic lithology prediction: Geophysical Journal International, **215**, 1368–1387.