EIP Sponsors User Guide no. 63, 20-06-2024

Geophysical Inversion using Parametric Variational Inference

GeoPVI User Guide

Xuebin Zhao[1] and Andrew Curtis[1]

[1] School of GeoSciences, University of Edinburgh, United Kingdom

E-mail: *xuebin.zhao@ed.ac.uk, andrew.curtis@ed.ac.uk*

SUMMARY

This user guide describes how to use *GeoPVI*, a Python package for **Geo**physical inversion using **P**arametric **V**ariational **I**nference methods. *GeoPVI* provides a suite of tools to facilitate variational Bayesian inversion, allowing users to efficiently model uncertainties in their geophysical parameter estimates. It differs from other non-parametric variational methods, such as Stein variational gradient descent (SVGD) and its variants, in the sense that the posterior probability distribution is parametrised explicitly and (semi-)analytically, which allows the posterior probability value for any model sample to be evaluated efficiently (without invoking Bayes' rule). This parametric feature can be used to vary prior information and test different prior hypotheses post Bayesian inversion, using a variational prior replacement methodology. *GeoPVI* currently features both mean field and full rank automatic differentiation variational inference (ADVI), physically structured variational inference (PSVI), normalising flows, and boosting variational inference (BVI). Future updates will expand this package to incorporate other parametric variational methods that have been tested in geophysics.

1 INTRODUCTION

Many geophysical problems can be cast as inverse problems that estimate a set of model parameters from observed geophysical data. Given that geophysical inversion often yields non-unique solutions, it is important to find all possible parameter values that fit observed data to within the range of possible data errors and estimate the corresponding uncertainties.

This can be achieved under a probabilistic framework by calculating the so-called *posterior* probability distribution function (pdf) $p(\mathbf{m}|\mathbf{d}_{obs})$ using Bayes' rule – a problem often referred to as Bayesian inference

$$p(\mathbf{m}|\mathbf{d}_{obs}) = \frac{p(\mathbf{d}_{obs}|\mathbf{m})p(\mathbf{m})}{p(\mathbf{d}_{obs})} \tag{1}$$

Here, vectors $\mathbf{m}$ and $\mathbf{d}_{obs}$ stand for model parameter and observed data, respectively, term $p(\mathbf{m})$ describes the *prior* information available on $\mathbf{m}$, $p(\mathbf{d}_{obs}|\mathbf{m})$ is the *likelihood* function, and $p(\mathbf{d}_{obs})$ is a normalisation constant called the *evidence*.

Monte Carlo sampling methods are often used to solve such problems by generating an ensemble of posterior samples that embody the Bayesian solution. These methods can sometimes be computationally expensive, especially for large-scale problems that contain thousands, even millions of unknown model parameters, so there is interest in testing the efficiency of other methods for particular classes of problems.

Variational inference solves Bayesian inverse problems under an optimisation framework, by seeking one optimal approximation to the true posterior pdf from a family of predefined and known distributions. Variational methods therefore convert a random sampling problem into numerical optimisations, while still providing fully probabilistic results. Depending on the problem, these methods can be more efficient than random sampling and provide better scaling to higher dimensions (more model parameters) (Bishop 2006; Blei et al. 2017; Zhang et al. 2018a). However, for some extreme cases such as 3D FWI, both variational methods and Monte Carlo sampling might not work due to the strong non-linearity and high dimensionality of the problems. Zhang et al. (2023) explained how certain variational methods are related to novel Monte Carlo type algo-

rithms, showing that a spectrum of techniques might be constructed that combine the strengths of both approaches, enabling complex problems to be solved.

Many variational methods have been applied to geophysical inverse problems and have proved to be more efficient than Monte Carlo methods. Currently, these methods can be categorised into two groups based on how they represent an approximate posterior distribution. The first group generates a set of posterior samples to approximate statistical information of the posterior distribution implicitly, such as Stein variational gradient descent (SVGD – Zhang & Curtis 2020; Zhang et al. 2023). The second group directly models a parametric representation (an explicit mathematical expression) for the posterior pdf (Zhao et al. 2021; Zhao & Curtis 2024c,d). This parametric (closed form) nature enables efficient evaluation of the posterior probability of any model sample once the inversion process is finished, without performing forward simulation or evaluating Bayes rule. We can further make use of this feature to update prior information post Bayesian inversion using a variational prior replacement method Zhao & Curtis (2024b).

GeoPVI implements a suite of methods that construct parametric posterior pdf's, currently including automatic differentiation variational inference (ADVI – Kucukelbir et al. 2017), physically structured variational inference (PSVI – Zhao & Curtis 2024d), normalising flows (Rezende & Mohamed 2015), and boosting variational inference (BVI – Guo et al. 2016; Miller et al. 2017). Future updates will try to incorporate other effective parametric variational inference methods.

2 INSTALLATION

2.1 Requirements

GeoPVI uses `NumPy` to perform basic numerical implementations, and uses automatic differentiation library provided by `PyTorch` to optimise variational parameters. `SciPy` is used to implement some scientific calculations, such as sparse matrix representation and inversion of large-scale matrix. In addition, `multiprocessing` package is used to parallelise forward simulations for different model samples. Alternatively, `Dask` can be used for this purpose, which provides an effective interface to use multiple CPU cores from modern high performance computing clusters. In a 2D

FWI example provided below, forward simulation code is written by C and Cython. These packages can be installed using the Python's package-management system pip:

```
$ pip install -r requirements.txt
```

2.2 Installation

Once you have downloaded the GeoPVI package and installed the above required packages, type

```
$ cd GeoPVI
$ sh setup.sh install
```

We recommend to install GeoPVI in an editable mode. Alternatively, if you do not want to install the package, simply do

```
$ sh setup.sh
```

If you do the latter, you need to tell scripts which use the GeoPVI package where the package is. For example, simply run a script with

```
$ PYTHONPATH=/your/GeoPVI/path python fwi2d.py
```

3 IMPLEMENTATION

In variational inference, a family of distributions (called the variational family) $\mathcal{Q}(\mathbf{m}) = \{q(\mathbf{m})\}$ is defined, from which an optimal member $q^*(\mathbf{m})$ is selected to best approximate the true posterior distribution $p(\mathbf{m}|\mathbf{d}_{obs})$. This optimal distribution can be found by maximising the following *evidence lower bound* (ELBO$[q(\mathbf{m})]$) (Zhao & Curtis 2024d):

$$\text{ELBO}[q(\mathbf{m})] = \mathbb{E}_{q(\mathbf{m})}[\log p(\mathbf{m}, \mathbf{d}_{obs})] - \mathbb{E}_{q(\mathbf{m})}[\log q(\mathbf{m})] \tag{2}$$

The calculation of the objective function in equation 2 requires $\log p(\mathbf{m}, \mathbf{d}_{obs})$ and $\log q(\mathbf{m})$ to be calculated, which can be constructed separately using GeoPVI. In the following subsections we first discuss implementations of the forward simulation which is required to calculate $p(\mathbf{m}, \mathbf{d}_{obs})$, after which we explain how the various variational approximations $q(\mathbf{m})$ to the posterior distribution are represented.

3.1 Forward simulations

The calculation of $\mathbb{E}_{q(\mathbf{m})}[\log p(\mathbf{m}, \mathbf{d}_{obs})]$ in equation 2 requires the logarithmic joint probability value $\log p(\mathbf{m}, \mathbf{d}_{obs})$, which is the summation of the logarithmic likelihood and prior values, according to Bayes' rule (equation 1). In GeoPVI, this can be calculated by defining a function that takes model parameter $\mathbf{m}$ as input and calculates $\log p(\mathbf{m}, \mathbf{d}_{obs})$. For example,

```
def log_prob(m):
    # input has a shape of (num_of_samples, dim)
    # Call a forward solver that takes x as input and outputs the modelled data
    # then calculate the log-likelihood and log-prior probability values
    data_syn = your_forward_solver(m)
    logp = log_prior(m) + log_like(data_syn, data_obs)
    return logp
```

where synthetic data `data_syn` are obtained by calling a user-defined forward simulation code `your_forward_solver`. Terms `log_prior` and `log_like` are functions which calculates the logarithmic prior and likelihood values given model sample `m`, synthetic and observed data `data_syn` and `data_obs`, respectively.

Currently, GeoPVI supports Normal and Uniform prior distributions. User-specified prior distributions can also be defined using a template in *geopvi/prior.py*. To use the Normal and Uniform prior pdfs provided in GeoPVI:

```
from geopvi.prior import Uniform, Normal
prior = Uniform(lowerbound, upperbound, smooth_matrix = None)
```

This defines a Uniform prior distribution bounded by a `lowerbound` and `upperbound`. If a smoothed prior distribution is desired, then pass a finite-difference smoothing matrix `L` to `smooth_matrix` (functions to define such a smoothing matrix for different situations are provided in *geopvi/utils.py*). Otherwise, it can be set as `None` to represent prior information without smoothing.

To define a Normal prior distribution:

```
prior = Normal(loc, std = None, covariance = None, scale_tril = None, precision = None)
```

where `loc` is the mean vector of the Normal distribution. The covariance matrix can be represented by one of the following four options: `std` (1D array representing standard deviations), `covariance` (positive-definite covariance matrix), `scale_tril` (lower-triangular factor – Cholesky

decomposition – of the covariance matrix, with positive-valued diagonal entities), or `precision` (positive-definite precision matrix). Note that only one of the four options can be specified, otherwise this will raise a Python `ValueError`.

The log-prior probability value for a given model sample $\mathbf{m}$ can be calculated using the GeoPVI built-in function:

```
log_prior = prior.log_prior(m)
```

If the likelihood function is assumed to be a Gaussian distribution, then it can be calculated by the misfit value between observed data and synthetic data generated by a forward modeller. Note that if the forward simulation code is written using PyTorch's built-in functions, the computational graph is already able to be recorded by PyTorch `autograd` package, such that the gradient of the likelihood value with respect to input sample $\mathbf{m}$ can be calculated directly using automatic differentiation. Otherwise, if the forward modeller itself is not auto-differentiable or relies on non-PyTorch libraries (e.g., NumPy or C), users will need to wrap it to interface with the `autograd` engine. Below is an example for this purpose:

```
from torch.autograd import Function


class ForwardModel(Function):
    # This class is used to register the output of the external (non-PyTorch) code
    # into the automatic differentiation engine
    @staticmethod
    def forward(ctx, input, your_external_solver):
        # forward simulation using external solver
        # returns simulation output (log-likelihood value in this case)
        # also registers its gradient w.r.t. input, and saves for back propagation
        output, grad = your_external_solver(input)
        ctx.save_for_backward(input, torch.tensor(grad))
        return torch.tensor(output)
    @staticmethod
    def backward(ctx, grad_output):
        # this function returns the gradient of loss function w.r.t the input tensor
        # therefore, the return shape should be the same as the shape of input tensor
        input, grad = ctx.saved_tensors
        grad_input = (grad_output[...,None] * grad)
```

```
        return grad_input, None

    def your_external_solver(m):
        # forward simulation of m
        # return log-likelihood value and its gradient w.r.t m
        # grad should have the same shape as that of m
        return log_like, grad

    def log_like_for_back_propagation(m):
        # m has a shape of (num_of_samples, dim)
        # forward simulation of m: your_external_solver(m)
        log_like = ForwardModel.apply(m, your_external_solver)
        return log_like
```

where `your_external_solver` is an external (non-PyTorch) Python function that performs forward simulation for input sample $\mathbf{m}$. It returns log-likelihood value `log_like` and its gradient with respect to $\mathbf{m}$. By calling function `log_like_for_back_propagation`, the calculated log-likelihood value is registered in the automatic differentiation graph and can be back propagated during the training process. More details can be found in a 2D FWI example provided below and on the PyTorch website.

GeoPVI provides two forward functions used typically in seismic imaging problems: the fast marching method (FMM) for travel time tomography, and 2D acoustic full waveform simulation for full waveform inversion (FWI). More details can be found in *geopvi/forward/tomo2d/posterior.py* and *geopvi/forward/fwi2d/posterior.py*.

3.2 Variational distributions

The simplest method to define a parametric variational pdf $q(\mathbf{m})$ is to use a factorised probability distribution, such as that defined in mean field automatic differentiation variational inference (MF-ADVI – Kucukelbir et al. 2017) where a Gaussian distribution with a diagonal covariance matrix is employed. That method is easy to implement as only the mean and standard deviation vectors of the Gaussian distribution need to be optimised during inversion. However, many previous studies showed that the method tends to strongly underestimate the posterior uncertainties since

all correlations between model parameters are ignored (Zhang & Curtis 2020; Zhao et al. 2021). In GeoPVI, we provide three different approaches to improve the performance of MF-ADVI, as illustrated in Figure 1.

The first approach is to increase the complexity of the diagonal Gaussian distribution (mean field ADVI) using more sophisticated methods, such as physically structured variational inference (PSVI – Zhao & Curtis 2024d) and full rank ADVI (FR-ADVI – Kucukelbir et al. 2017), where correlations between pairs of model parameters are treated as learnable parameters to be optimised. PSVI considers dominant parameter correlations only, whereas FR-ADVI models all correlation information (details can be found in Zhao & Curtis (2024d)).

In the second approach we construct a 'deeper' learning model by applying a series of invertible and differentiable transforms (called the flows) to a simple and known base probability distribution (e.g., a standard Normal or a Uniform distribution), rather than employing one single parametric distribution as considered in the first approach. The resulting method is referred to as normalising flows (Rezende & Mohamed 2015; Zhao et al. 2021).

The third approach builds a 'broader' model using a method called boosting variational inference (BVI – Guo et al. 2016; Miller et al. 2017; Zhao & Curtis 2024c). BVI uses a mixture of simple probability distributions to construct a complex variational pdf. Each component distribution is optimised sequentially using a greedy algorithm to avoid mode collapse where one component dominants the mixture distribution. In principle, we can use all methods in the first row in Figure 1 to model component distributions, then use BVI to build a mixture distribution. Currently, GeoPVI features boosting MF-ADVI, PSVI and FR-ADVI, as illustrated below each method in the bottom row of Figure 1.

The above methods are all implemented using either a flows-based or a BVI-based model (top and bottom rows in Figure 1, respectively). In the next two subsections we show how to build a variational pdf and perform inference using these two strategies.

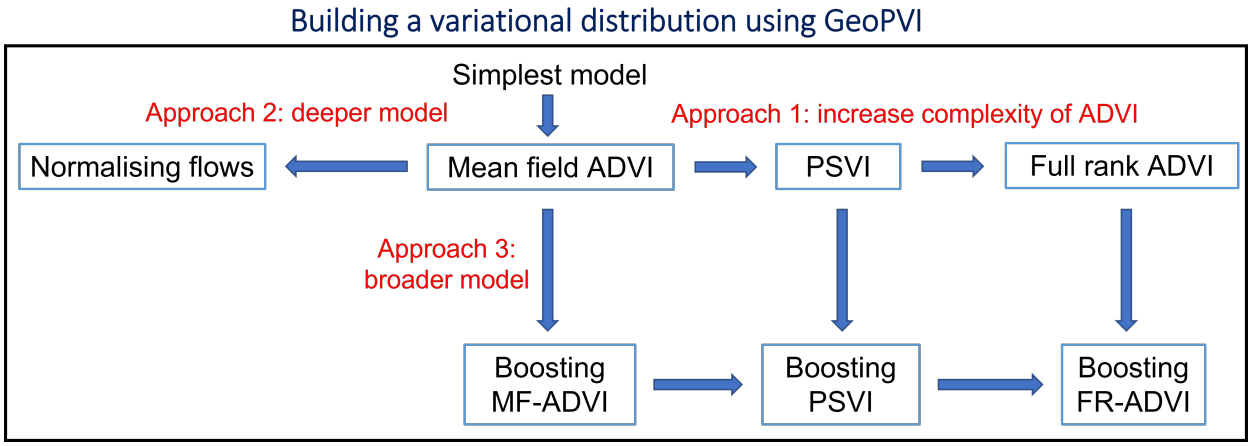

Figure 1. Structure of the GeoPVI package for building a variational distribution. Starting from the simplest case of mean field ADVI, three different approaches are employed to construct an improved variational pdf: either increase complexity of the ADVI covariance matrix, or implement a flows-based or boosting-based model. Arrows between different methods represent increasing complexity of the variational pdf.

3.3 Flows-based variational distribution

In GeoPVI we can build a parametric variational distribution $q(\mathbf{m})$ using a normalising flows model (Rezende & Mohamed 2015; Zhao et al. 2021). Normalising flows provide a flexible way to construct a parametric probability distribution by passing a base probability distribution through a sequence of invertible transforms (flows). To construct such a model:

```
from geopvi.nfvi.models import FlowsBasedDistribution
```

Users can choose specific flows based on problems at hand. Python script `geopvi/nfvi/flows.py` provides several commonly used flows in literature. Below is an example to construct a flows-based model to perform automatic differentiation variational inference (ADVI) and physically structured variational inference (PSVI):

```
from geopvi.nfvi.flows import Linear, Real2Constr
flows = [Linear(dim, kernel = 'structured', mask = off_diag_mask)]
flows.append(Real2Constr(lower = lowerbound, upper = upperbound))
variational_pdf = FlowsBasedDistribution(flows, base = 'Normal')
```

This initialises a variational model using one `Linear` flow and one `Real2Constr` flow, applied on a standard normal base probability distribution. `Real2Constr` transforms model parameters from the space of real numbers into a constrained space bounded by hyper-parameters `lowerbound` and `upperbound`. If model parameters do not have lower/upper bounds, the corresponding parameter

can be set to `None`. `Linear` flow performs a linear transform on the input vector $\mathbf{x}$

$$\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\mathbf{x} \tag{3}$$

The variational parameters are $\boldsymbol{\mu}$ and $\mathbf{L}$ in this case. If $\mathbf{x}$ follows a standard Normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, one single `Linear` flow essentially transforms $\mathcal{N}(\mathbf{0}, \mathbf{I})$ into any normal distribution defined by $\mathcal{N}(\boldsymbol{\mu}, \mathbf{L}\mathbf{L}^{\mathbf{T}})$. In `Linear` flow, `dim` is the dimensionality (number of unknown parameters to be inferred) of vector $\mathbf{x}$. Parameter `kernel` controls different methods used to construct matrix $\mathbf{L}$, and its available options include `'diagonal'`, `'fullrank'` and `'structured'`, each corresponding to a Gaussian distribution with a diagonal, full, and structured (desired correlation information - see details in Zhao & Curtis 2024d) covariance matrix, respectively. These three options can be used to perform mean field ADVI, full rank ADVI, and PSVI, followed by a `Real2Constr` flow. If `'structured'` is used, a mask defining the desired correlation structure (`off_diag_mask`) is required (see details in examples provided in GeoPVI). Otherwise it can safely be ignored by setting it to `None`.

Perform

```
x = variational_pdf.sample_from_base(100) # x has a shape of (100, dim)
m, logq = variational_pdf(x)
```

which generates 100 samples x from base distribution of the flows-based model. These samples are then transformed through the flows-based model to obtain samples $\mathbf{m}$ distributed according to the constructed variational distribution $q(\mathbf{m})$ and their corresponding logarithmic probability values $\log q(\mathbf{m})$.

After obtaining the two ingredients $\log q(\mathbf{m})$ and $\log p(\mathbf{m}, \mathbf{d}_{obs})$, we can estimate the ELBO in equation 2 using Monte Carlo integration with n random samples drawn from $q(\mathbf{m})$

$$\text{ELBO}[q(\mathbf{m})] \approx \frac{1}{n}\sum_{i}(\log p(\mathbf{m}, \mathbf{d}_{obs}) - \log q(\mathbf{m}_i)) \tag{4}$$

Define a loss function to be the negative ELBO, which can be calculated by performing

```
x = variational_pdf.sample_from_base(100) # x has a shape of (100, dim)
m, logq = variational_pdf(x)
logp = log_prob(m)
```

```
# -ELBO can be estimated using Monte Carlo integration
negative_elbo = -torch.mean(logp - logq)
```

The optimal variational distribution can be found by iteratively minimising this loss function until convergence. This training process can be accomplished by calling a `VariationalInversion` class defined in GeoPVI:

```
from geopvi.nfvi.models import VariationalInversion
inversion = VariationalInversion(variationalDistribution = variational_pdf,
                                 log_posterior = log_prob)
```

Then, variational inversion can be performed by:

```
negative_elbo = inversion.update(n_iter = 1000, nsample = 10,
                                 optimizer = 'torch.optim.Adam')
```

which updates the flows-based variational model using the `Adam` optimizer for 1000 iterations with $n = 10$ samples per iteration for Monte Carlo integration in equation 4. This function returns the `negative_elbo` value within each iteration. Once the training process is finished, posterior samples can be obtained by sampling from the variational distribution

```
samples = variational_pdf.sample(1000)
```

3.4 BVI-based variational distributions

GeoPVI provides a second approach to build a variational distribution using boosting variational inference (BVI), in which a mixture (sum) of simple distributions is used to approximate the posterior probability distribution. BVI trains and adds one component distribution to the mixture at a time using a greedy algorithm

$$q^t(\mathbf{m}) = (1 - w_t)q^{t-1}(\mathbf{m}) + w_t g_t(\mathbf{m}) \tag{5}$$

where $q^t(\mathbf{m})$ is a mixture of t distributions, obtained by combining the previous mixture distribution $q^{t-1}(\mathbf{m})$, weighted by $(1 - w_t)$, with the new component $g_t(\mathbf{m})$ weighted by w_t. The new component distribution $g_t(\mathbf{m})$ is found by maximising a so-called residual-ELBO (RELBO):

$$\text{RELBO}[g_t(\mathbf{m})] = \mathbb{E}_{g_t(\mathbf{m})}[\log p(\mathbf{m}, \mathbf{d}_{obs})] - \mathbb{E}_{g_t(\mathbf{m})}[\log q^{t-1}(\mathbf{m})] - \lambda \mathbb{E}_{g_t(\mathbf{m})}[\log g_t(\mathbf{m})] \tag{6}$$

More theoretical details can be found in Zhao & Curtis (2024c).

To run BVI,

```
from geopvi.boostingvi.component import GaussianComponent
gaussian = GaussianComponent(dim, kernel = 'diagonal', constrained = False)
```

This defines a Gaussian component distribution with a dimensionality of `dim` and a `diagonal` covariance kernel (similar to the `Linear` flow introduced above). In this case, for each component we need to optimise its mean and standard deviation vectors. If the model parameter $\mathbf{m}$ to be inverted is defined within a constrained space, then set parameter `constrained = True`, otherwise set it as `False`.

A BVI model built by boosting a set of Gaussian distributions can be constructed by

```
from geopvi.boostingvi.bvi import BoostingGaussian
bvi_pdf = BoostingGaussian(componentDistribution = gaussian, log_posterior = log_prob)
```

To train this BVI model, run

```
result = bvi_pdf.update(ncomponent = 5, n_iter = 1000, nsample = 10,
                            optimizer = 'torch.optim.Adam')
```

which builds a mixture of 5 Gaussian distributions, each updated for 1000 iterations and 10 samples per iteration to estimate the expectations in equation 6. After training, the inversion result is returned, defined by a Python dictionary containing optimised parameters for each component. Posterior samples can be obtained by

```
samples = bvi_pdf.sample(1000)
```

4 EXAMPLES

We now show examples of using the GeoPVI package to solve 2D travel time tomography, and 2D and 3D full waveform inversion (FWI) problems. Note that the implementation of 3D FWI requires a suitably efficient 3D forward and adjoint wavefields solver, which are not included in the current release of GeoPVI. In addition, we provide more example applications in Jupyter notebooks in *geopvi/examples/tutorials* folder.

4.1 2D travel time tomography

For 2D travel time tomography we use a fast marching method (FMM) to model the travel time data (Rawlinson & Sambridge 2005), which can be found in *geopvi/forward/tomo2d* folder. The implementation details of variational travel time tomography can be found in *examples/tomo2d*. To use the code,

```
$ cd examples/tomo2d
$ sh run.sh
```

The input and output of the code are described below. Note that hyperparameters for building a variational model and performing optimisation are defined as a Python `argparser.parse_args()` object in *examples/tomo2d/tomo2d.py*, and are self-explanatory.

4.1.1 Input

The following input files are stored in *examples/tomo2d/input*.

config.ini – This file contains nuisance parameters required by the 2D FMM code.

prior.txt – This file contains the hyperparameters of the prior distribution whose name is given by the argument `args.prior_param` in *tomo2d.py*. If a Uniform distribution is used, the file contains the lower (first column) and upper (second column) bounds for each parameter. In the case of a normal distribution, the file contains the mean (first column) and standard deviation (second column) for each parameter.

sources.txt and *receivers.txt* – Source and receiver coordinate files. Each row contains the (x, y) location for a source (or a receiver).

traveltime.txt – Travel time data file, whose name is given by the argument `args.datafile` in *tomo2d.py*. Each row contains a travel time. The order is first source and first receiver, first source and second receiver, ..., first source and last receiver; second source and first receiver, second source and second receiver, ... and so on. If the travel time for a given source-receiver pair is not available, set it to zero.

4.1.2 output

The outputs are stored in the directory *examples/tomo2d/output*.

**parameter.npy* – A NumPy NPY format file which contains the optimised variational parameters. If `Linear` flow is used, this output file includes the vector $\boldsymbol{\mu}$ (the first `dim` elements), diagonal elements of the matrix $\mathbf{L}$ (the second `dim` elements) and off-diagonal elements of the matrix $\mathbf{L}$ (the remaining elements). If mean field ADVI is performed, no off-diagonal elements are output. Parameters $\boldsymbol{\mu}$ and $\mathbf{L}$ can then be used to define a Gaussian variational distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{L}\mathbf{L}^{\mathbf{T}})$.

**model.pt* – A PyTorch .pt format file which stores the entire variational model using Python's pickle module. This file can be used to resume training (optimisation) process from a certain checkpoint, or to load the obtained variational model for post-inversion analysis.

**samples.npy* – A NumPy NPY format file which contains the posterior samples.

**loss.txt* – A text file which stores the negative ELBO values during training.

4.1.3 Examples

A synthetic example described in Zhao et al. (2021) can be found in the folder *examples/tomo2d*.

4.2 2D FWI

The package includes a 2D FWI example, in which the forward modelling of waveform data is performed using a finite difference method, and the gradient of the misfit function with respect to velocity parameters is obtained using the adjoint method. Its implementation details can be found in *geopvi/forward/fwi2d*. The example code that uses variational inference to solve this 2D FWI problem is in *examples/fwi2d/fwi2d.py*. To run the code:

```
$ cd examples/fwi2d
$ sh run.sh
```

The input and output of the code are described below. Note that hyperparameters for building a variational model and performing optimisation are defined as a Python `argparser.parse_args()` object in *examples/fwi2d/fwi2d.py*, and are self-explanatory.

4.2.1 Input

The following input files are stored in *examples/fwi2d/input*.

config.ini – This file contains the control parameters to perform forward and adjoint simulation. An example is shown below, in which every line beginning with '#' is a comment:

```
# nuisance parameter for 2D fwi code
[FWI]
# grid points in x direction (nx)
nx = 250
# grid points in z direction (nz)
nz = 110
# depth of (water) layer where velocity is set as true value during inversion
layer_fixed = 10
# true velocity value in the layer with fixed velocity value
vel_fixed = 1950
# pml points (pml0)
pml = 10
# Finite difference order (Lc)
Lc = 3
# Method to calculate Laplace operator: 0 for FDM and 1 for PSM, (laplace_slover)
laplace_slover = 0
# Total number of sources (ns)
ns = 12
# Total time steps (nt)
nt = 2001
# Shot interval in grid points (ds)
ds = 20
# Grid number of the first shot to the left of the model (ns0)
ns0 = 15
# Depth of source in grid points (depths)
depths = 1
# Depth of receiver in grid points (depthr)
depthr = 9
# Receiver interval in grid points (dr)
dr = 1
# Time step interval of saved wavefield during forward (nt_interval)
nt_interval = 2
# Grid spacing in x direction (dx)
```

```
dx = 20.0
# Grid spacing in z direction (dz)
dz = 20.0
# Time step (dt)
dt = 0.002
# Donimate frequency (f0)
f0 = 10.0
```

prior.txt – This file contains the hyperparameters of the prior distribution whose name is given by the argument `args.prior_param` in *fwi2d.py*. If a Uniform distribution is used, the file contains the lower (first column) and upper (second column) bounds at each depth (Z direction). In the case of a normal distribution, the file contains the mean (first column) and standard deviation (second column) at each depth. The prior distribution is assumed to be laterally homogeneous.

waveform.npy – A NumPy NPY format file which contains the observed waveform data. The file name is given by the argument `args.datafile` in *fwi2d.py*. The data is stored in a NumPy array with a shape `(ns, nr, nt)` where `ns` is the number of sources, `nr` is the number of receivers and `nt` is the number of time points of each trace.

4.2.2 output

The outputs are stored in the directory *examples/fwi2d/output*.

**parameter.npy* – A NumPy NPY format file which contains the optimised variational parameters. If `Linear` flow is used, this output file includes the vector $\boldsymbol{\mu}$ (first `dim` elements), diagonal elements of the matrix $\mathbf{L}$ (second `dim` elements) and off-diagonal elements of the matrix $\mathbf{L}$ (remaining elements). If mean field ADVI is performed, no off-diagonal elements are output. Parameters $\boldsymbol{\mu}$ and $\mathbf{L}$ can then be used to define a Gaussian variational distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{L}\mathbf{L}^{\mathbf{T}})$.

**model.pt* – A PyTorch .pt format file which stores the entire variational model using Python's pickle module. This file can be used to resume training (optimisation) process from a certain checkpoint, or to load the obtained variational model for post-inversion analysis.

**samples.npy* – A NumPy NPY format file which contains the posterior samples.

**loss.txt* – A text file which stores the negative ELBO values during training.

4.2.3 Examples

A synthetic example that uses a part of the Marmousi model can be found in the folder *examples/fwi2d*. The details of this example are described in Zhao & Curtis (2024d).

4.3 3D FWI

In a range of geophysical applications one may need to solve a large scale inverse problem. This has become possible through the development of modern high performance computation (HPC) facilities. In this section we take 3D FWI as an example to show how to use the GeoPVI package to solve large scale problems.

Throughout this suite of methods, the forward simulation can be computed independently which makes it easy to parallelise. For example, one can use the `Dask` package to parallelise the computation using multiple CPU cores.

```
from dask.distributed import Client

def fwi_i(model):
    # forward midelling for the ith model sample
    log_like, grad = external_fwi_code(model)
    return log_like, grad

def fwi(models):
    # forward modelling for all model samples
    # First, create a dask client for cros-core parallisation
    client = Client()

    # submit simulation job to dask client
    futures = []
    for i in range(models.shape[0]):
        futures.append(client.submit(fwi_i, model, pure = False))

    # gather the computational results
    results = client.gather(futures)
    log_like = np.zeros((models.shape[0],))
    grad = np.zeros_like(models)
    for i in range(models.shape[0]):
```

```
        log_like[i] = results[i][0]
        grad[i] = results[i][1]

    return log_like, grad
```

The function `fwi` can thereafter be used with the GeoPVI package to implement 3D FWI, just as displayed above. In this way the code can use many cores that are available on modern HPC facilities to provide an efficient method. Note that the code can be further improved by combining the modern Distributed Resource Management Systems and by using minibatch data sets. A detailed example that uses a part of the 3D Overthrust model are described in Zhao & Curtis (2024a), and the corresponding code can be found in *geopvi/forward/fwi3d* and *examples/fwi3d/fwi3d.py*.

5 TERMS OF USE AND DISCLAIMER

To the best of our knowledge this code is correct, error-free as of June 2024. We can not guarantee its accuracy however. If after your own detailed investigation you believe that there are errors/bugs in the code or examples, please contact Xuebin Zhao at xuebin.zhao@ed.ac.uk, or Andrew Curtis at andrew.curtis@ed.ac.uk.

ACKNOWLEDGMENTS

We thank Edinburgh Imaging Project (EIP - `https://blogs.ed.ac.uk/imaging/`) sponsors (BP and TotalEnergies) for supporting this research.

REFERENCES

Bishop, C. M., 2006. *Pattern recognition and machine learning*, springer.

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D., 2017. Variational inference: A review for statisticians, *Journal of the American statistical Association*, **112**(518), 859–877.

Guo, F., Wang, X., Fan, K., Broderick, T., & Dunson, D. B., 2016. Boosting variational inference, *Advances in Neural Information Processing Systems*.

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., & Blei, D. M., 2017. Automatic differentiation variational inference, *The Journal of Machine Learning Research*, **18**(1), 430–474.

Miller, A. C., Foti, N. J., & Adams, R. P., 2017. Variational boosting: Iteratively refining posterior approximations, in *International Conference on Machine Learning*, pp. 2420–2429, PMLR.

Rawlinson, N. & Sambridge, M., 2005. The fast marching method: an effective tool for tomographic imaging and tracking multiple phases in complex layered media, *Exploration Geophysics*, **36**(4), 341–350.

Rezende, D. J. & Mohamed, S., 2015. Variational inference with normalizing flows, *arXiv preprint arXiv:1505.05770*.

Zhang, C., Bütepage, J., Kjellström, H., & Mandt, S., 2018a. Advances in variational inference, *IEEE transactions on pattern analysis and machine intelligence*, **41**(8), 2008–2026.

Zhang, X. & Curtis, A., 2020. Seismic tomography using variational inference methods, *Journal of Geophysical Research: Solid Earth*, **125**(4), e2019JB018589.

Zhang, X., Curtis, A., Galetti, E., & De Ridder, S., 2018b. 3-d monte carlo surface wave tomography, *Geophysical Journal International*, **215**(3), 1644–1658.

Zhang, X., Lomas, A., Zhou, M., Zheng, Y., & Curtis, A., 2023. 3D bayesian variational full waveform inversion, *Geophysical Journal International*, **234**(1), 546–561.

Zhao, X. & Curtis, A., 2024a. Efficient 3d bayesian fullwaveform inversion: an example discriminating prior hypotheses, *EIP Report 62*.

Zhao, X. & Curtis, A., 2024b. Variational prior replacement in Bayesian inference and inversion, *EIP Report 61*.

Zhao, X. & Curtis, A., 2024c. Bayesian inversion, uncertainty analysis and interrogation using boosting variational inference, *Journal of Geophysical Research: Solid Earth*, **129**(1), e2023JB027789.

Zhao, X. & Curtis, A., 2024d. Physically structured variational inference for Bayesian full waveform inversion, *ESS Open Archinve*.

Zhao, X., Curtis, A., & Zhang, X., 2021. Bayesian seismic tomography using normalizing flows, *Geophysical Journal International*, **228**(1), 213–239.