EIP Sponsors User Guide no. 61, 24-01-2024

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 manual describes how to use GeoPVI, a Python package designed for geophysical inversion using parametric variational inference methods, in which variational distribution is defined explicitly by parametric (analytic) representations. GeoPVI provides a suite of tools to facilitate Bayesian inversion processes, allowing users to efficiently construct a parametric posterior distribution and model uncertainties in their geophysical parameter estimations. It differs from other non-parametric variational methods, such as Stein variational gradient descent (SVGD), in the sense that the posterior distribution is parmetrised explicitly and (semi-)analytically, such that posterior probability value of any model sample can be evaluated efficiently after the inversion (without invoking Bayes' rule). GeoPVI currently features both mean field and full rank automatic differentiation variational inference (ADVI), physically structured variational inference (PSVI), and normalising flows. 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 typical inverse problems that estimate a set of latent model parameters from observed data. Given that geophysical inversion often yields non-unique solutions, it is important to find all possible parameters that are consistent with observed data to estimate the corresponding uncertainties.

This can be achieved under a probabilistic framework to calculate the so-called *posterior* probability distribution function (pdf) using Bayes' rule - a suite of methods referred to as Bayesian inference. Monte Carlo sampling methods are often used to solve Bayesian problems by generating an ensemble of posterior samples that embody the Bayesian solution. However, these methods can be computationally expensive, especially for large-scale problems that contain thousands, even millions of unknown parameters.

Variational inference solves Bayesian inference problems within 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 a numerical optimisation one, while still providing fully probabilistic results. Therefore, the method can be more efficient than random sampling methods and can provide better scaling to higher dimensional problems (Bishop 2006; Blei et al. 2017; Zhang et al. 2018).

Many variational methods have been applied to geophysical inverse problems and are proved to be far more efficient than Monte Carlo methods. Currently, these methods can be categorised into two groups based on their approaches to represent an approximation to the posterior distribution. The first group generates a set of posterior samples to implicitly approximate statistical information of the posterior distribution, such as Stein variational gradient descent (SVGD – Zhang & Curtis 2020; Zhang et al. 2023); and the second group directly models a parametric representation (expression) of the probability distribution (Zhao et al. 2021; Zhao & Curtis 2023, 2024). GeoPVI focuses on a suite of methods that optimise a parametric posterior pdf, currently including automatic differentiation variational inference (ADVI – Kucukelbir et al. 2017), physically structured variational inference (PSVI – Zhao & Curtis 2024), and normalising flows (Rezende & Mohamed 2015). Future updates will expand this package to incorporate other variational methods such as boosting variational inference.

2 INSTALLATION

2.1 Requirements

GeoPVI uses `NumPy` to perform basic numerical implementations, and use the automatic differentiation library provided by `PyTorch` to optimise variational parameters. `SciPy` is used to implement some scientific calculations, such as sparse matrix representation and inverse of large-scale matrix, In addition, `multiprocessing` package is used to parallelise the forward simulation 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

One you have downloaded the GeoPVI package and installed the above required packages

```
$ 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
```

Then, 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 2024):

$$\text{ELBO}[q(\mathbf{m})] = \mathbb{E}_{q(\mathbf{m})}[\log p(\mathbf{m}, \mathbf{d}_{obs})] - \mathbb{E}_{q(\mathbf{m})}[\log q(\mathbf{m})] \tag{1}$$

The calculation of the objective function in equation 1 requires to calculate $\log p(\mathbf{m}, \mathbf{d}_{obs})$ and $\log q(\mathbf{m})$, which can be constructed separately using GeoPVI.

3.1 Variational distribution

In GeoPVI, we build a parametric variational distribution using a normalising flows model (Rezende & Mohamed 2015; Zhao et al. 2021). Normalising flows provides a flexible way to construct a parametric probability distribution by passing a simple and known initial probability distribution (e.g., a standard Normal or a Uniform distribution) through a series of invertible and differentiable transforms, called the flows. To construct such a model:

```python
from geopvi.vi.models import VariationalModel
```

Users can choose specific transforms (flows) based on problems at hand. Python script `geopvi/vi/flows.py` provides several commonly used flows in literature (more available flows will be implemented in future release). Below is an example using two commonly used flows

```python
from geopvi.vi.flows import Linear, Real2Constr
flows = [Linear(dim, kernel = 'structured', mask = off_diag_mask) for _ in range(nflows)]
flows.append(Real2Constr(lower = lowerbound, upper = upperbound))
model = VariationalModel(flows)
```

This initialises a variational model using `nflows` `Linear` flows and one `Real2Constr` flow. `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 bound, the corresponding parameter can be set to `None`. `Linear` flow performs a linear transform on the input vector $\mathbf{x}$

$$\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\mathbf{x} \tag{2}$$

The variational parameters are $\boldsymbol{\mu}$ and $\mathbf{L}$. 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 of the model vector $\mathbf{m}$ (number of unknown parameters to be inferred). Parameter `kernel` controls different methods used to construct matrix $\mathbf{L}$, and its available options include `'diagonal'`, `'fullrank'` and `'structured'`, corresponding to a Gaussian distribution with a diagonal, full, and structured (desired correlation information - see details in Zhao & Curtis 2024) covariance matrices, 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 Zhao & Curtis 2024, and an example provided in GeoPVI). Otherwise it can be ignored safely by setting it to `None`.

Then, $\log q(\mathbf{m})$ can be calculated by

```python
x = torch.from_numpy(np.random.normal(0., 1., size = (n, dim)))
z, logq = model(x)
```

3.2 Forward simulation

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

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

where `logp`, `log_like` and `log_prior` are the logarithmic joint probability, likelihood and prior values, respectively.

Currently, GeoPVI supports Normal and Uniform prior distributions:

```
from geopvi.prior import Uniform, Normal
prior = Uniform(lowerbound, upperbound, smooth_matrix = L)
```

This defines a Uniform prior distribution bounded by `lowerbound` and `upperbound`. If a smooth prior distribution is desired, then pass a smooth matrix `L` to `smooth_matrix`. 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 covariance, with positive-valued diagonal), or `precision` (positive-definite precision matrix). Note that only one of the four options can be specified.

The log-prior value of a given model sample `x` can be calculated using GeoPVI built-in functions:

```
log_prior = prior.log_prior(x)
```

If the likelihood function is assumed to be a Gaussian distribution, then it can be calculated from 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 samples `x` can be calculated directly through automatic differentiation. Otherwise, if the forward modeller itself is not auto-differentiable or rely 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 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 register its gradient w.r.t. input, and save 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(x):
    # forward simulation of x
    # return log-likelihood value and its gradient w.r.t x
    # grad should have the same shape as that of x
    return log_like, grad

def log_like_for_back_propagation(x):
```

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

where `your_external_solver` is an external (non-PyTorch) Python function that performs forward simulation for input sample. It returns log-likelihood value `log_like` and its gradient with respect to `x`. 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 training process. More details can be found in a 2D FWI example provided below and on the PyTorch website.

3.3 Variational Optimisation

After obtaining the two ingredients $\log q(\mathbf{m})$ and $\log p(\mathbf{m}, \mathbf{d}_{obs})$, we can estimate the ELBO in equation 1 using Monte Carlo integration with n random samples drawn from $q(\mathbf{m})$

$$\mathrm{ELBO}[q(\mathbf{m})] \approx \frac{1}{n} \sum_{i} (\log p(\mathbf{m}, \mathbf{d}_{obs}) - \log q(\mathbf{m}_i)) \tag{3}$$

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

```python
x = torch.from_numpy(np.random.normal(0., 1., size = (n, dim)))
z, logq = model(x)
logp = log_prob(z)
# -ELBO can be estimated using Monte Carlo integration
loss = -torch.mean(logp - logq)
```

The optimal variational distribution can be found by iteratively minimising this loss function until convergence.

4 EXAMPLES

In this section we show examples of using the GeoPVI package to solve 2D and 3D full waveform inversion (FWI) problems. Note that the implementation of 3D FWI requires a suitably efficient 3D forward and adjoint wavefield solver, which is not included in the current release of GeoPVI.

4.1 2D FWI

In a 2D FWI example, 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/fwi2d*. The example code that uses variational inference to solve this 2D FWI problem is in *examples/fwi2d/fwi2d.py*. To run the code, simply do

```
$ 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-explained.

4.1.1 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.

4.1.2 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 is 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 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 (2024).

4.2 3D FWI

In a range of geophysical applications one may need to solve a large scale inverse problem. This has become possible in the light of 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. For a detailed example that uses the SGE system, see *geopvi/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 March 2024. We can not guarantee its accuracy however. If after your own detailed investigation you believe 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) 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.

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.

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

Zhang, C., Bütepage, J., Kjellström, H., & Mandt, S., 2018. 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., Lomas, A., Zhou, M., Zheng, Y., & Curtis, A., 2023. 3-d bayesian variational full waveform inversion, *Geophysical Journal International*, **234**(1), 546–561.

Zhao, X. & Curtis, A., 2023. Bayesian inversion, uncertainty analysis and interrogation using boosting variational inference, *Journal of Geophysical Research: Solid Earth*, **129**(1), e2023JB027789.

Zhao, X. & Curtis, A., 2024. Physically structured variational inference for bayesian full waveform inversion, *EIP Report 61*.

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