EIP Sponsors User Guide no. 43, 08-23-2022

Variational Inversion Package

VIP User Guide

Xin Zhang[1] and Andrew Curtis[1]

[1] School of GeoSciences, University of Edinburgh,

James Hutton Road, Edinburgh, *EH9 3FE*, United Kingdom

E-mail: *x.zhang2@ed.ac.uk, andrew.curtis@ed.ac.uk*

SUMMARY

This manual describes how to use the Variational Inversion Package (VIP). This software deploys variational inference methods to solve seismic full-waveform inversion and travel time tomography problems. It provides complete implementations of 2D travel time tomography and 2D full-waveform inversion, and a guide to implement 3D full-waveform inversion given a suitably efficient 3D forward and adjoint wavefield modelling solver (not included). The package includes three different variational inference methods: automatic differential variational inference (ADVI), Stein variational gradient descent (SVGD), and stochastic SVGD (sSVGD).

1 PACKAGE OVERVIEW

Geophysicists may need to characterize properties of the Earth's interior using observed seismic, gravitational or electromagnetic data. This is usually achieved by solving a highly nonunique inverse problem. It is therefore important to quantify uncertainties in the solution, in other words to invert for the family of all models that are consistent both with the data and with other information or expectations about the subsurface. A model that most closely represents the true Earth properties must lie within that family.

Bayesian inference is a method to solve inverse problems and quantify uncertainties. The method updates a *prior* probability density function (pdf) that describes what is already known about the model with new information contained in the data, to construct a probability distribution post inversion called the *posterior* pdf; this describes the above family of models, and their relative probability of being close to the true Earth structure. Variational inference is one efficient method to solve Bayesian inference problems.

Variational inference methods seek an optimal approximation to the posterior pdf within a predefined family of (simplified) pdfs by using optimization. The method can be more efficient than Monte Carlo sampling methods and can provide better scaling to higher dimensionality (Bishop 2006; Blei et al. 2017). It can be applied to large datasets by dividing the dataset into minibatches and using stochastic optimization, and can also be parallelized at the individual sample level. This makes the method a sensible choice for solving large scale inverse problems by taking advantage of modern high performance computational facilities.

VIP is a Python package which solves seismic inverse problems using variational inference methods, including automatic differential variational inference (ADVI, Kucukelbir et al. 2017), Stein variational gradient descent (SVGD, Liu & Wang 2016) and stochastic SVGD (sSVGD, Gallego & Insua 2018). The package provides a scalable implementation which can be deployed on a desktop as well as modern high performance computational facilities. As examples we provide complete implementations of 2D full-waveform inversion and 2D travel time tomography, and a guide to implement 3D full-waveform inversion given a suitably efficient 3D forward and adjoint wavefield modelling solver (not included).

2 INSTALLATION

2.1 Requirements

For efficiency and scalability, parts of the code are implemented using Fortran and Cython. In addition, we use Dask to parallelize the computation and H5py to store the final samples. The package therefore requires Cython, Dask and H5py, which can be installed easily

```
pip install Cython
pip install dask
pip install h5py
```

2.2 Installation

Once you have downloaded the package

```
cd VIP
sh setup.sh install
```

If you do not want to install the package, simply do

```
sh setup.sh
```

In this way the package is not installed in your Python environment. However, by doing this you need to tell scripts which use the VIP package where the package is. For example, when running a script

```
PYTHONPATH=/your/VIP/path python vip_example.py
```

3 VARIATIONAL INVERSION

Variational inference is a method which uses optimization to solve Bayesian inference problems. The VIP package implements three different variational methods: ADVI, SVGD and sSVGD, to estimate posterior probability distributions for geophysical inverse problems. ADVI uses a (transformed) Gaussian probability distribution to approximate the posterior pdf, whereas SVGD and sSVGD use a set of models (called particles) to represent the posterior pdf. An overview of these methods and Bayesian inference in general can be found in Zhang et al. (2021) and EIP report 51 (Zhang et al. 2022).

To use these methods,

```
from vip.pyvi.svgd import SVGD, sSVGD
from vip.pyvi.advi import ADVI
```

All methods require a function that takes model parameters as input and calculates the gradients of the logarithm posterior pdf with respect to model parameters. For example, define a function

```
def dlnprob(theta):
  # theta has a shape of (num_of_particles, num_of_parameters)
  # some calculation of theta
  return loss, grad, None
```

where `loss` is the misfit value (or negative logarithm of posterior pdf), `grad` contains the gradients. The third output is used to return other auxiliary values (e.g., a mask array), and can be ignored safely by setting it to `None`. Thereafter, we can create a class for each method, for example

```
svgd = SVGD(dlnprob, kernel='rbf', out='samples.hdf5')
```

This initialises a class of SVGD method that uses a radial basis function (rbf) kernel and saves results in `samples.hdf5`. To sample the posterior pdf, do

```
losses = svgd.sample(x0, n_iter=1000, stepsize=0.01, optimizer='sgd')
```

where `x0` is a variable of shape (`num_of_particles`, `num_of_parameters`) which contains starting particles, and `losses` is a vector containing misfit values for all iterations. This runs the SVGD algorithm for 1,000 iterations using the stochastic gradient descent (sgd) algorithm with a step length of 0.01. The final particles are written into the file `samples.hdf5`. For optimization the supported algorithms include `sgd`, `adagrad` (Duchi et al. 2011), `adadelta` (Zeiler 2012) and `adam` (Kingma & Ba 2014). The supported kernel function options are `rbf` and `diagonal`. The `diagonal` option uses a matrix-valued kernel

$$\mathbf{K}(\mathbf{m}',\mathbf{m}) = \mathbf{Q}^{-1}\mathrm{exp}(-\frac{1}{2\sigma^2}||\mathbf{m}-\mathbf{m}'||^2_{\mathbf{Q}}) \tag{1}$$

where $\mathbf{Q}$ is a positive definite diagonal matrix, $||\mathbf{m}-\mathbf{m}'||^2_{\mathbf{Q}} = (\mathbf{m}-\mathbf{m}')^{\mathrm{T}}\mathbf{Q}(\mathbf{m}-\mathbf{m}')$ and σ is a scaling parameter. The elements of $\mathbf{Q}$ can be set as the inverse of the variance calculated across particles. For example, one can do

```
svgd = SVGD(dlnprob, kernel='diagonal', weight='var', out='samples.hdf5')
```

To use sSVGD algorithm,

```
ssvgd = sSVGD(dlnprob, kernel='rbf', out='samples.hdf5')
losses = ssvgd.sample(x0, n_iter=2000, stepsize=0.01, burn_in=1000)
```

This runs the sSVGD algorithm for 2,000 iterations with a burn-in period of 1,000. Note that sSVGD also supports the diagonal matrix kernel function defined in equation 1. To use ADVI,

```
advi = ADVI(dlnprob, kernel='meanfield')
phi, losses = advi.sample(n_iter=2000, stepsize=0.01, optimizer='adam')
```

This runs ADVI for 2,000 iterations using the `adam` optimization algorithm. The vector `phi` contains the mean (first half) and the logarithm of the standard deviation (second half) of the final Gaussian distribution. The initial distribution is set to be a standard Normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ by default. To use the full-rank ADVI, set the kernel to *fullrank*.

4 EXAMPLES

In this section we show examples of using the VIP package to solve seismic inverse problems, including 2D travel time tomography and 2D full-waveform inversion. In addition we give a guide for implementing 3D variational full-waveform inversion.

4.1 2D travel time tomography

For 2D travel time tomography we use a fast marching method to model the travel time data (Rawlinson & Sambridge 2004), which can be found in *vip/tomo* folder. The implementation details of variational travel time tomography can be found in *tests/tomo2d/tomo2d.py*. To use the code, simply do

python tomo2d.py

The input and output of the code are described below.

4.1.1 Input

config.ini – This file contains the control parameters for the package. An example can be found below, in which every line beginning with ”#” is a comment.

config.ini

```
[tomo]
# the working directory
basepath = /home/xzhang/vfwi/tests/tomo2d
# source file
srcfile = ${basepath}/input/sources.txt
# receiver file
recfile = ${basepath}/input/receivers.txt
# data file
datafile = ${basepath}/input/ttimes.txt

# other arguments for tomo2d code
# grid points in x direction (nx)
nx = 21
# grid points in y direction (ny)
ny = 21
# minimum x
xmin = -5.0
# minimum y
ymin = -5.0
# Grid spacing in x direction (dx)
dx = 0.5
# Grid spacing in y direction (dy)
dy = 0.5
# upsampling factor for grid spacings in x direction (gdx)
gdx = 2
# upsampling factor for grid spacings in y direction (gdy)
gdy = 2
# source refinement factor (upsampling grid spacings around source)
sdx = 4
# source refinement area in grid points, that is, the upsampling around source
# is conducted from source location to sext number of grid cells
sext = 4
```

```
[svgd]

# 'svgd' or 'ssvgd' is supported

method = svgd

# kernel function, only 'rbf' or 'diagonal' supported

kernel = rbf

# how to calculate the elements for diagonal kernel

# 'var' uses variance across all particles

diag = var

# optimizer, sgd, adagrad, adadelta or adam

optimizer = sgd

# using transform (true) or not (false) when using a Uniform prior.

# If true, transform the variable to an unconstained space and perform sampling in that space.

# If false, no transform is performed and parameters are clipped to within the lower and upper bounds.

# This option does not affect other priors, e.g., a Gaussian prior.

transform = true

# prior type, 'Uniform' or 'Gaussian'

priortype = Uniform

# file that contains hyperparameters for prior pdf

# If Uniform prior, the first column contains the lower boundary for each parameter, and

# the second column contains the upper boundary

# If Gaussian prior, the first column contains the mean and the second column contains

# the standard deviation

prior = ${tomo:basepath}/input/prior.txt

# number of particles

nparticles = 50

# number of iterations

iter = 200

# burn_in period, only used for ssvgd

burn_in = 100

# thining of the chain, only used for ssvgd

thin = 2

# step length for svgd and ssvgd

stepsize = 0.1

# decay the stepsize exponentially, the final stepsize will be stepsize*final_decay

final_decay = 1.0

# noise level, currently only support a constant number

sigma = 0.05

# smoothness constraint in x direction, 0 for no smoothness

smoothx = 0
```

```
# smoothness constraint in y direction, 0 for no smoothness
smoothy = 0
# output directory
outpath = ${tomo:basepath}/results
```

---

prior.txt – This is the hyperparameter file for the prior distribution whose name is given by `prior` in *config.ini*. If a Uniform prior distribution is used, the file contains the lower (first column) and upper (second column) bounds for each parameter. If a Gaussian prior distribution is used, the file contains the mean (first column) and standard deviation (second column) for each parameter.

sources.txt and *receivers.txt* – Source and receiver coordinates files. Each row contains the (x, y) location for a source (or a receiver). The file names are given by `srcfile` and `recfile` in *config.ini*.

ttimes.txt – Travel time data file, whose name is given by `datafile` in *config.ini*. Each row contains a travel time. The order is first source and first receiver, first source and second receiver, ..., first source and last receiver; the next line has 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 given by `outpath` in *config.ini*. The samples are stored in *samples.hdf5* which is a self-explanatory HDF5 file, and can be read using the package H5py. The misfit values of all iterations are written in *misfits.txt* file.

4.1.3 Examples

A simple synthetic example described in Zhang & Curtis (2020) is provided in the folder *tests/-tomo2d*.

4.2 2D full-waveform inversion

As an example we implement a 2D acoustic variational full-waveform inversion. 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. The implementation details can be found in *vip/fwi2d*. The example code that uses variational inference to solve this 2D FWI problem is in *tests/fwi2d/vfwi2d.py*. To run the code, simply do

python vfwi2d.py

The input and output of the code are described below.

4.2.1 Input

config.ini – This file contains the control parameters. An example is shown below

config.ini

```
[FWI]
# the working directory
basepath = /home/xzhang/vfwi/tests/fwi2d
# configuration file for the 2d forward modelling code. Just give a name, the contents of the file
# is automatically generated by the code
configfile = ${basepath}/input/input_params.txt
# data file
datafile = ${basepath}/input/waveform.npy

# other arguments for forward modelling code
# grid points in x direction (nx)
nx = 200
# grid points in z direction (nz)
nz = 120
# perfect matched layer (pml) points (pml0)
pml = 10
# Finite difference order (Lc)
Lc = 3
# Total number of sources (ns)
ns = 10
# Total time steps (nt)
nt = 2501
```

```
# Shot interval in grid points (ds)

ds = 20

# Grid number of the first shot to the left of the model (ns0)

ns0 = 10

# Depth of source in grid points (depths)

depths = 1

# Depth of receiver in grid points (depthr)

depthr = 17

# Receiver interval in grid points (dr)

dr = 1

# Time step interval of saved wavefield during forward modelling (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


[svgd]

# 'svgd' or 'ssvgd' is supported

method = svgd

# kernel function, only 'rbf' or 'diagonal' supported

kernel = rbf

# how to calculate the elements for diagonal kernel

# 'var' uses variance across all particles

diag = var

# optimizer, sgd, adagrad, adadelta, and adam

optimizer = sgd

# using transform (true) or not (false)

transform = true

# prior type, 'Uniform' or 'Gaussian'

priortype = Uniform

# file that contains hyperparameters for prior pdf

prior = ${FWI:basepath}/input/prior.txt

# number of particles

nparticles = 200
```

```
# number of iterations
iter = 2000
# burn_in period, only used for ssvgd
burn_in = 1000
# thining of the chain, only used for ssvgd
thin = 2
# step length for svgd and ssvgd
stepsize = 0.2
# decay the stepsize exponentially, the final stepsize will be stepsize*final_decay
final_decay = 0.2
# noise level, currently only support a constant number
sigma = 1e-1
# smoothness in x direction, 0 for no smoothness
smoothx = 0
# smoothness in z direction, 0 for no smoothness
smoothz = 0
# output directory
outpath = ${FWI:basepath}/results

[dask]
# number of dask workers, each worker performs the forward modelling for one particle at once
nworkers = 20
# number of threads for each dask worker, usually set to one to let the forward modelling
# code decide the number of threads, e.g. by using omp_num_threads
ph = 1
# directory for dask
daskpath = ${FWI:basepath}/dask
```

---

prior.txt – This file contains the hyperparameters of the prior distribution whose name is given by `datafile` in *config.ini*. 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 Gaussian 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 waveform data. The file name is given by `datafile` in *config.ini*. 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

Similarly as above, the outputs are stored in the directory specified by *outpath* in *config.ini*. The samples are stored in *samples.hdf5* and misfit values are stored in *misfits.txt*.

4.2.3 Examples

A synthetic example that uses a part of the Marmousi model can be found in the folder *tests/fwi2d*. The details of this example are described in Zhang & Curtis (2021).

4.3 3D full-waveform inversion

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 VIP package to solve large scale problems.

Throughout this suite of methods, the forward modelling computation of each particle can be computed independently which makes it easy to parallelize. For example, one can use the Dask package to parallelize the computation.

```
from dask.distributed import Client

def fwi_i(model):
  # forward modelling for the ith particle
  loss, grad = external_fwi_code(model)
  return loss, grad

def fwi(models):
  # forward modelling for all particles
  # create a dask client
  client = Client()

  # submit computation of each particle 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)
    loss = np.zeros((models.shape[0],))
    grad = np.zeros_like(models)
    for i in range(models.shape[0]):
      loss[i] = results[i][0]
      grad[i,:] = results[i][1]

    return loss, grad, None
```

The function `fwi` can thereafter be used with the VIP package to implement 3D variational FWI. For example,

```
  ssvgd = sSVGD(fwi, kernel='rbf', out='samples.hdf5')
  losses = ssvgd.sample(x0, n_iter=2000, stepsize=0.01, burn_in=1000)
```

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 *vip/fwi* and *tests/fwi3d/vfwi3d.py*.

TERMS OF USE AND DISCLAIMER

To the best of our knowledge this code is correct, error-free as of September 2022. 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 Xin Zhang at *x.zhang2@ed.ac.uk*, and/or Andrew Curtis at *Andrew.Curtis@ed.ac.uk*.

ACKNOWLEDGMENTS

The authors thank the Edinburgh Imaging Project sponsors (BP and Total) 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.

Duchi, J., Hazan, E., & Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization, *Journal of Machine Learning Research*, **12**(Jul), 2121–2159.

Gallego, V. & Insua, D. R., 2018. Stochastic gradient mcmc with repulsive forces, *arXiv preprint arXiv:1812.00071*.

Kingma, D. P. & Ba, J., 2014. Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980*.

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.

Liu, Q. & Wang, D., 2016. Stein variational gradient descent: A general purpose Byesian inference algorithm, in *Advances In Neural Information Processing Systems*, pp. 2378–2386.

Rawlinson, N. & Sambridge, M., 2004. Multiple reflection and transmission phases in complex layered media using a multistage fast marching method, *Geophysics*, **69**(5), 1338–1350.

Zeiler, M. D., 2012. Adadelta: an adaptive learning rate method, *arXiv preprint arXiv:1212.5701*.

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

Zhang, X. & Curtis, A., 2021. Bayesian full-waveform inversion with realistic priors, *Geophysics*, **86**(5), 1–20.

Zhang, X., Nawaz, M. A., Zhao, X., & Curtis, A., 2021. An introduction to variational inference in geophysical inverse problems, in *Advances in Geophysics*, vol. 62, pp. 73–140, Elsevier.

Zhang, X., Lomas, A., Zhou, M., Zheng, Y., & Curtis, A., 2022. 3D variational Bayesian full waveform inversion, *EIP Report*, **51**.