S33C–2790

Zephyr: Open Source Parallel Seismic Waveform Inversion in an Integrated Python-based Framework

Introduction

Zephyr is an open source Python framework that enables seismic waveform forward modelling and seismic waveform inversion. The present form of the project focusses on 2D and 2.5D frequency-domain waveform inversion with viscoacoustic and/or elastic anisotropic physics.

Zephyr is designed under an object-oriented paradigm, and makes extensive use of object class inheritance. This enables Zephyr objects to be used in a modular way, allowing the researcher to construct simulations and inverse problems by using a set of interchangeable building blocks.

Project goals

- Interactive research computing for algorithm design
- Noninteractive parallel & distributed computation for HPC
- Low-level forward modelling interface for embedding
- Replace & augment legacy Fortran waveform inversion code "FULLWV"
- Be accessible, convenient and free

Object attributes from key/value

Many of the programming constructs in the Zephyr implementation are subclasses of `AttributeMapper`, which automatically handles configuring new objects based on a global `systemConfig` dictionary (i.e., a key/value store). An `AttributeMapper` subclass defines a dictionary `initMap`, which includes keys for mappable inputs expected from the `systemConfig` parameter. Each object subscribes to the parameters it needs, including those needed by its ancestors. The `initMap` dictionary keys are inherited from the base classes.

```
class BaseModelDependent(AttributeMapper):

    initMap = {
    #   Argument        Required    Rename as ...   Store as type
        'nx':           (True,      None,           np.int64),
        'ny':           (False,     None,           np.int64),
        'nz':           (True,      None,           np.int64),
        'xorig':        (False,     '_xorig',       np.float64),
        'yorig':        (False,     '_xorig',       np.float64),
        'zorig':        (False,     '_zorig',       np.float64),
        'dx':           (False,     '_dx',          np.float64),
        'dy':           (False,     '_dx',          np.float64),
        'dz':           (False,     '_dz',          np.float64),
        'freeSurf':     (False,     '_freeSurf',    tuple),
    }

    def __init__(self, systemConfig, *args, **kwargs):
        ...
```

Each value in the `initMap` dictionary is a tuple, which determines how to process the corresponding entry in the dictionary-like `systemConfig`. The `BaseModelDependent` class is used in all classes that access model geometry.

1. An exception will be raised if the first element in the tuple is `True`, but the corresponding key is not in `systemConfig`.
2. If the second entry is `None`, the key will be set in the subclass's attribute dictionary unmodified, whereas if the second element is a string then that overrides the key.
3. If the third element is `None`, the input argument will be set in the subclass dictionary unmodified. If the third element is a callable then it will be applied to the element (e.g., to allow copying and/or typecasting of inputs).

Most of this behaviour is handled by `zephyr.backend.meta.AMMetaClass`, which is the metaclass for `AttributeMapper` and its descendants. All of this happens before the call to `AttributeMapper.__init__`. This enables the `initMap` class attribute to be merged automatically and inherit keys from the subclass's bases. Almost all runtime functionality is defined in methods and properties. Other key/value stores such as HDF5 files are also supported (e.g., via h5py).

← **Website**
zephyr.space

GitHub →
github.com/uwoseis/zephyr

Example: Crosshole survey

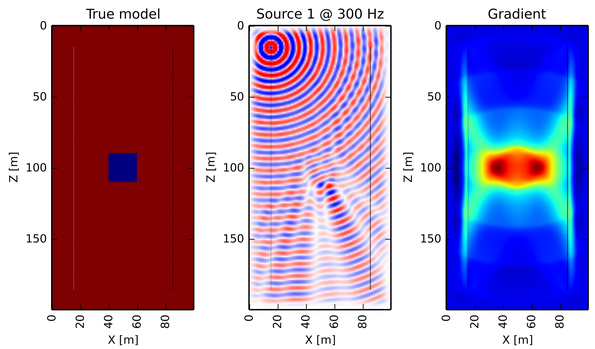

Fig. 1: *The "true" model for this example is presented in the left panel, with red representing the background acoustic velocity of 2500 m/s and blue representing an anomaly with velocity 2000 m/s. Data and wavefields are simulated for each of the 171 sources (white) and 171 receivers (black). The real part of the 300 Hz wavefield for the first source is shown in the middle panel. The right panel shows the first waveform inversion gradient, computed with all 29241 source/receiver pairs at 8 frequencies.*

Seismic waveform inversion involves reconstructing the features of an earth model from waveform measurements collected in the field. In order to demonstrate some of the features of Zephyr, we have designed a simple synthetic problem. **Fig. 1** shows the simple test model, and the wavefield resulting from a single time-harmonic source at location (15,15).

Wavefields (like the second panel of **Fig. 1**) are simulated for all 171 sources (markers in the left "borehole"), and data (**Fig. 2**) are extracted for all 171 receivers (markers in the right "borehole"), leading to a total of 29241 data for each of 8 temporal frequencies.

The data residuals represent the error between the "true" data in the model containing the square anomaly, and new "synthetic" data in a model with a constant acoustic velocity of 2500 m/s. From these data residuals, a gradient is calculated, which indicates how to update the model to better fit the data (**Fig. 1**, right panel).

The full source code for this example is presented in the right-hand panel.

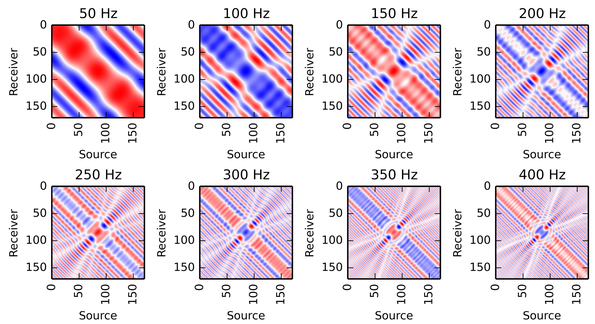

Fig. 2: *The data from the "true" model are shown for each source/receiver pair at 8 frequencies (50–450 Hz). Only the real part of the data are shown, though each datum is a complex number.*

Fig. 3 shows some example wavefields from three different Zephyr forward modelling kernels.

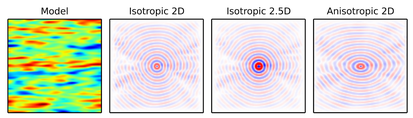

Fig. 3: *A simple heterogeneous model, and results from three different Zephyr modelling kernels (`MiniZephyr`, `MiniZephyr25D` and `Eurus`)*

Project structure

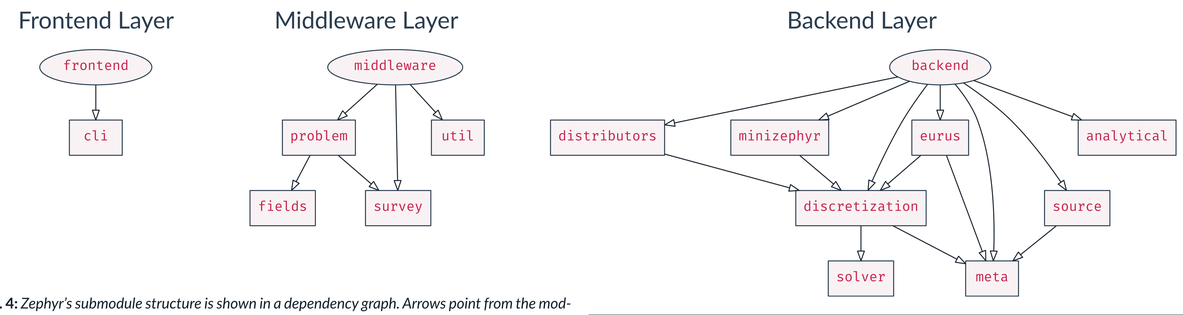

Fig. 4: *Zephyr's submodule structure is shown in a dependency graph. Arrows point from the module that uses a feature to the module that defines it.*

Zephyr is organized into three sub-packages:

- `zephyr.backend` Forward modelling, constructing sparse systems of matrix equations and RHS (source) vectors, solving for wavefields with parallel and distributed computations
- `zephyr.middleware` Survey geometry, data and field types, user-facing forward and inverse problems, optimization; interfaces tightly with SimPEG
- `zephyr.frontend` Application user interface for noninteractive research computation, teaching and cluster deployment; command-line (CLI) based

Full source code for example

```
import numpy as np
from zephyr.backend import MiniZephyr
from zephyr.middleware import HelmBaseProblem, HelmBaseSurvey
from SimPEG.DataMisfit import l2_DataMisfit

nx = 100
nz = 200
cBackground = 2500.
cAnomaly = -500.

cInitial = cBackground * np.ones((nz,nx))

cUpdate = np.zeros((nz,nx))
cUpdate[(nz/2)-10:(nz/2)+10,(nx/2)-10:(nx/2)+10] = cAnomaly

cTrue = cInitial + cUpdate

sz = np.arange(25, nz-24, 1)
sx = 25. * np.ones((len(sz),))
rz = np.arange(25, nz-24, 1)
rx = (nx - 25.) * np.ones((len(rz),))

geom = {
    'src':      np.vstack([sx, sz]).T,
    'rec':      np.vstack([rx, rz]).T,
}

systemConfig = {
    'dx':       1.,                     # m
    'dz':       1.,                     # m
    'c':        cTrue,                  # m/s
    'rho':      1.,                     # kg/m^3
    'nx':       nx,                     # count
    'nz':       nz,                     # count
    'freqs':    np.arange(50, 450, 50), # Hz
    'disc':     MiniZephyr,             # discretization
    'geom':     geom,                   # dictionary
}

problem = HelmBaseProblem(systemConfig)
survey  = HelmBaseSurvey(systemConfig)
problem.pair(survey)

dObs = survey.dpred()
survey.dobs = dObs
survey.std = 1.
misfit = l2_DataMisfit(survey)
misfit.Wd = 1.

resid = survey.residual(cInitial)
g = misfit.evalDeriv(cInitial)
```

Authorship & Contact info

Brendan R. Smithyman <bsmithym@uwo.ca>
Shaun Hadden <shadden2@uwo.ca>
R. Gerhard Pratt <gpratt2@uwo.ca>

Dept. of Earth Sciences, Western University
London, ON, Canada N6A 5B7

License

Zephyr is open source software released under the MIT License. For more information, please check the project website and/or consult the link below: https://opensource.org/licenses/MIT

Dependencies

Zephyr is written in Python, and tested using the CPython implementation on Linux and Mac OS.

Zephyr makes extensive use of NumPy and SciPy in order to support sparse matrix operations. At present (and by design), Zephyr's `backend` depends only on NumPy, SciPy and the Python standard library. Zephyr aims to be compatible with MUMPS, but this functionality is presently experimental.

The `frontend` and `middleware` layers of Zephyr depend on IPython for parallel computing, and on SimPEG, which is a flexible toolkit for solving geophysical inversion problems, written in Python.

The `frontend` command-line interface depends on click. Plotting and analysis tools make use of matplotlib, IPython and d3.js.

Documentation uses mkdocs and Sphinx.

References

Cockett, R., Kang, S., Heagy, L.J., Pidlisecky, A., Oldenburg, D.W., 2015. SimPEG: An open source framework for simulation and gradient based parameter estimation in geophysical applications. Comput. Geosci. 85, 142–154. doi:10.1016/j.cageo.2015.09.015

Operto, S., Virieux, J., Ribodetti, A., Anderson, J., 2009. Finite-difference frequency-domain modeling of viscoacoustic wave propagation in 2D tilted transversely isotropic (TTI) media. Geophysics 74, T75–T95. doi:10.1190/1.3157243

Pratt, R.G., 1999. Seismic waveform inversion in the frequency domain, Part 1: Theory and verification in a physical scale model. Geophysics 64, 888–901. doi:10.1190/1.1444597

Smithyman, B.R., Clowes, R.M., 2013. Waveform tomography in 2.5D: Parameterization for crooked-line acquisition geometry. J. Geophys. Res. Solid Earth 118, 2119–2137. doi:10.1002/jgrb.50100