SeisFlows Documentation

Release 0.1

Princeton University

Jan 26, 2018

Contents

We have prepared a 2D waveform inversion example that is inexpensive enough to run on almost any laptop, desktop, or cluster. To run this simple checkboard inversion, see these step-by-step instructions.

Some additional examples are available for download. Please review the instructions for the 2D checkerboard test case to get a sense for how to run these other inversions.

Some 2D examples based on the Marmousi model are available here.

A 3D Cartesian checkerboard example is available here.

A 3D global 1-chunk example is available here. Please note, the compressed archive for this example is very large (> 0.5 GB). *[No longer available because of file size.]*

At a minimimum, one processer is required for the 2D Marmousi examples, 16 processors are required for the 3D Cartesian example, and 64 processors are required for the global 1-chunk example. See here for more information about running inversions in parallel.

Note: File hosting services are provided by my alma mater. The download server may become temporarily unavailable due to system maintenance or permanently unavailable due to expiration of my account.

CHAPTER 1

Examples: Available Locally

Users with accounts on "tiger.princeton.edu" can run the following inversions without having to download files or recompile executables.

2D Regional and Global

- North America
- Southern California
- Global
- Deep Earth

2D Near Surface

- Marmousi offshore
- Marmousi onshore
- overthrust offshore
- overthrust onshore
- BP anticline
- BP salt diapir

3D Cartesian

- checkerboard

3D Global

- mideast

See these instructions for running inversions on our local cluster.

1.1 Documentation

1.1.1 Usage

Overview

SeisFlows is a Python waveform inversion package with a growing user base in academia and industry. So far, the package has been used for production runs with a billion or so model parameters and for research on oil and gas exploration, earthquake seismology, and general nonlinear optimization problems.

To provide flexibility, SeisFlows is very modular. Users are offered choices in each of the following categories:

- workflow
- system
- solver
- pre-processing
- post-processing
- nonlinear optimization

The thing that ties everything together is the workflow. Execution of a workflow is equivalent to stepping through the code contained in `workflow.main`. Users are free to customize the default 'inversion' and 'migration' workflows from the main package.

A number of options exist for system and solver. By isolating system and solver machinery, users can switch from one application to another with relative ease. For example, if the study area in an earthquake tomography project expands, users can trade a regional Cartesian solver for a global solver. If a PBS cluster goes offline and a SLURM cluster comes online to replace it, users can trade the PBS system interface for a SLURM system interface.

A selection of ready-to-go system and solver interfaces is provided in the main package. Through these interfaces, SeisFlows (or prototypes of it) have run on clusters managed by the Department of Defense, Chevron Corp., Total S.A., Princeton University and other universities and institutions.

Users can also choose from various pre-processing and post-processing options. In our terminology, pre-processing consists of signal processing operations performed on seismic traces prior to the gradient computation. Post-processing consists of regularization or image processing operations carried out after the gradient computation.

If desired functionality is missing from the main package, users can contribute their own classes or overload default ones.

Installation

To install Seisflows, first clone the repository:

```
git clone https://github.com/rmodrak/seisflows.git
```

Then set environment variables. Add the following lines to `.bashrc` (or modify accordingly if using a shell other than bash):

```
export PATH=$PATH:/path/to/seisflows/scripts
export PYTHONPATH=$PYTHONPATH:/path/to/seisflows
```

Software Prerequisites

SeisFlows requires Python 2.7, Numpy, Scipy and Obspy. Users will need to install these packages before being able to use SeisFlows.

Forward modeling software is also a prerequisite; see *Solver Configuration* for more information.

Hardware Prerequisites

Access to a computer cluster is required for most applications. Base classes are provided for several common cluster configurations, including PBS and SLURM. Nonstandard configurations can often be accommodated through modifications to one of the base classes; see *System Configuration* for details.

Job Submission

Each job must have it own *working directory* within which users must supply two input files, `paths.py` and `parameters.py`.

To begin executing a workflow, simply type `sfrun` within a working directory. If an `inversion` workflow and `serial` system configuration, for example, are specified in the parameters file, the inversion will begin executing immediately in serial. If a PBS, SLURM, or LSF system configuration is specified instead, execution may wait until required resources become available.

Once the workflow starts running, status information is displayed to the terminal or to the file `output.log`. By default, updated models and other inversion results are output to the working directory.

To get a sense for how it all works, try following the step by step instructions included here.

Solver Configuration

SeisFlows includes Python interfaces for SPECFEM2D, SPECFEM3D, and SPECFEM3D_GLOBE. While the Python interfaces are part of the SeisFlows package, the solver source code must be downloaded separately through GitHub.

After downloading the solver source code, users must configure and compile it, following the instructions in the solver user manual. Summarized briefly, the configuration and compilation procedure is:

Prior to compilation, users need to run the `configure` script and prepare input files such as

- parameter file
- source file
- stations file.

To successfully run the `configure`, you may need to install compilers, libraries, and other software in your environment.

The result of compilation is a set of binary executables, such as

- mesher
- solver
- smoothing utility
- summing utility.

Note that if solver input files change, solver executables may need to be recompiled.

After compilation, solver input files must be gathered together in one directory and solver executables in another. The absolute paths to the directories containing input files and executables must be given in `paths.py` as follows:

```
SPECFEM_DATA = '/path/to/spcefem/input/files'
SPECFEM_BIN = '/path/to/specfem/executable/files'
```

Writing Custom Solver Interfaces

Besides SPECFEM2D, SPECFEM3D, and SPECFEM3D_GLOBE, SeisFlows can interface with other solvers capable of running forward and adjoint simulations. Users unaffiliated with the main SeisFlows developers have succeeded in interfacing with, for example, their own finite difference solvers. For information about writing custom solver interfaces, see *Developer Reference*.

Design Philosophy

Integration of the solver with the other workflow components can be challenging. Here we try to give an idea of the issues involved from both a developer and a user standpoint.

- Solver computations account for most of the cost of an inversion. As a result, the solver must be written in an efficient compiled language, and wrappers must be written to integrate the compiled code with other software components.
- There is currently no mechanism for automatically compiling executables for SPECFEM2D, SPECFEM3D, or SPECFEM3D_GLOBE. Users must prepare their own SPECFEM input files and then follow the compilation procedure in the SPECFEM documentation.
- As described above, SeisFlows uses two input files, `paths.py` and `parameters.py`. Problems could arise if parameters from SeisFlows input files conflict with parameters from solver input file. Users must make sure that there are no conflicts between SeisFlows parameters and solver parameters.
- In the solver routines, it's natural to represent velocity models as dictionaries, with different keys corresponding to different material parameters. In the optimization routines, it's natural to represent velocity models as vectors. To convert back and forth between these two representations, a pair of utility functions–`split` and `merge`–are included in solver.base.

System Configuration

SeisFlows can run on SLURM, PBS, and LSF clusters, as well as, for very small problems, laptops or desktops. A list of available system interface classes follows. By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

PBS_SM - For small inversions on PBS clusters. All resources are allocated at the beginning and all simulations are run within a single job. Requires that individual wavefield simulations run each on a single core, making this option suitable for small 2D inversions only.

PBS_LG - For large inversions on PBS clusters. The work of the inversion is divided between multiple jobs that are coordinated by a single long-running master job. Resources are allocated on a per simulation basis. Suitable for small to medium 3D inversions in which individual wavefield simulation span several or more nodes.

SLURM_SM - For small inversions on SLURM clusters. All resources are allocated at the beginning and all simulations are run within a single job. Requires that each individual wavefield simulation runs only a single core, making this option suitable for small 2D inversions only.

SLURM_LG - For large inversions on SLURM clusters. The work of the inversion is divided between multiple jobs that are coordinated by a single long-running master job. Resources are allocated on a per simulation basis. Suitable for 3D inversions in which individual wavefield simulation span several or more nodes.

SERIAL - Tasks that are normally carried out in parallel are instead carried out one at a time. Useful for debugging, but not much else.

MULTITHREADED - On desktops or laptops with multiple cores, allows embarrassingly parallel tasks to be carried out several at a time, rather than one at a time. Can be used to run small 2D inversions on a laptop or desktop.

LSF_SM - Same as SLURM_SM and PBS_SM, except for LSF clusters.

LSF_LG - Same as SLURM_LG and PBS_LG, except for LSF clusters.

Writing Custom System Interfaces

If your needs are more specialized, please view `seisflows.system` source code to get a sense for how to write your own custom system interfaces. In our experience, system interfaces require no more than a few hundred lines of code, so writing your own is generally possible once you are familiar with the SeisFlows framework and your own cluster environment.

Design Philosophy

To make SeisFlows work across different environments, our approach is to wrap system commands with a thin Python layer. To handle job submission, for example, we wrap the PBS command `qsub` and the SLURM command `sbatch` with a python utility called *system.submit*. The result is a consistent python interface across different clusters.

Filesystem settings can be adjusted by modifying values in the `PATH` dictionary, which is populated from `paths.py`. Output files and temporary files, by default, are written to the working directory. If a value for `PATH.SCRATCH` is supplied, temporary files are written there instead. If each compute node has its own local filesystem, a value for `PATH.LOCAL` can be supplied so that temporary files required only for a local process need not be written to the global filesystem.

As the size of an inversion grows, scalability and fault tolerance become increasingly important. If a single forward simulation spans more than one node, users must select `pbs_lg` or `slurm_lg` system configurations in `parameters.py`. If a forward simulation fits onto a single node, users should select `pbs_sm` or `slurm_sm` instead.

In SeisFlows, the overall approach to solving system interface problems is to use lightweight Python wrappers. For complex cluster configurations, heavier-weight solutions may be required. Users are referred to SAGA or Pegasus projects for ideas.

Developer Reference

To allow classes to work with one another, each must conform to an established interface. This means certain classes must implement certain methods, with specified input and output. Required methods include

- `setup` methods are generic methods, called from the `main` workflow script and meant to provide users the flexibility to perform any required setup tasks.
- `check` methods are the default mechanism for parameter declaration and checking and are called just once, prior to a job being submitted through the scheduler.

Besides required methods, classes may include any number of private methods or utility functions.

Parameter Files

`parameters.py` contains a list of parameter names and values. Prior to a job being submitted, parameters are checked so that errors can be detected without loss of queue time or wall time. Parameters are stored in a dictionary that is accessible from anywhere in the Python code. By convention, all parameter names must be upper case. Parameter values can be floats, integers, strings or any other Python data type. Parameters can be listed in any order.

`paths.py` contains a list of path names and values. Prior to a job being submitted, paths are checked so that errors can be detected without loss of queue time or wall time. Paths are stored in a dictionary that is accessible from anywhere in the Python code. By convention, all names must be upper case, and all values must be absolute paths. Paths can be listed in any order.