SeisFlows 用户手册

Release 0.1

Feb 03, 2021

CONTENTS

本文档基于普林斯顿大学 Ryan Modrak 编写的 SeisFlows。

本手册的更新与维护将会被 commit 到以下 GitHub 目录：

git clone https://github.com/niyiyu2316/seisflows.git

引用：

Ryan Modrak, Dmitry Borisov, Matthieu Lefebvre, Jeroen Tromp; SeisFlows—Flexible waveform inversion software, Computers & Geosciences, Volume 115, June 2018, Pages 88-95, https://doi.org/10.1016/j.cageo.2018.02.004

Ryan Modrak, Jeroen Tromp; Seismic waveform inversion best practices: regional, global and exploration test cases, Geophysical Journal International, Volume 206, Issue 3, 1 September 2016, Pages 1864–1889, https://doi.org/10.1093/gji/ggw202

CHAPTER

ONE

开始

1.1 安装

首先从 GitHub 下载 Seisflows 代码:

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

设置环境变量。将以下代码加入 `.bashrc` (如果使用其他 shell，相应地改变):

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

根据 SeisFlow 的位置，`/path/to/seisflows/scripts` 与 `/path/to/seisflows` 需要进行对应的调整。重新登陆，或使用:

```
$ source ~/.bashrc
```

应用环境变量更改。应用后，请确保 SeisFlow 目录不再发生移动。

1.2 依赖软件

SeisFlows 需要 Python 2.7 以及相应的 NumPy、SciPy 和 ObsPy 库支持。运行 SeisFlows 前需要提前安装这些库。SeisFlows 提供了若干程序测试，为确保其所需的依赖库能正常运行，这些测试必须通过，参见*测试*

正演程序需要提前编译，参考*正演程序配置*。

1.3 任务提交

每一个 SeisFlows 的任务需要有其独立的目录，目录下必须有用户参数输入文档 “paths.py“ 和 `parameters.py` 。在命令行该目录下输入 `sfrun` 即可开始执行一个任务。

1.4 正演程序配置

SeisFlows 包含为 SPECFEM2D/3D/3D_GLOBE 提供的 python 接口，而 SEM 的源代码则必须额外下载。

首先，应当运行 configure 命令配置合适的 Makefile。以下命令指定 gfortran 和 gcc 分别作为 fortran 和 c 的编译器，并提供 mpi 支持:

```
$ ./configure FC=gfortran CC=gcc --with-mpi
```

如果采用 mpi，请确认 mpif90 命令调用与 FC 对应的编译器:

```
$ mpif90 --version
GNU Fortran (Homebrew GCC 10.2.0) 10.2.0
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

否则，请先将 mpi 调用进行变更:

```
$ export OMPI_FC=gfortran
```

configure 输出以下内容则成功:

```
## ---------------- ##
## Specfem 2D 7.0.0 ##
## ---------------- ##

./configure has completed and set up a default configuration to build.

You may wish to modify the following files before running a simulation:
DATA/Par_file           Set parameters affecting the simulation.
DATA/SOURCE             Set the source parameters before running the solver.
```

编译 specfem2d 主程序:

```
$ make all -j 10
```

完成编译后，检查检查程序编译是否成功:

```
$ ls ./bin/
xadj_seismogram               xcombine_sem                 xmeshfem2D                 ␣
↪ xspecfem2D
xcheck_quality_external_mesh  xconvolve_source_timefunction xsmooth_sem                ␣
↪ xsum_kernels
```

1.5 开发参考

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.

CHAPTER

TWO

可执行程序

SeisFlows 的 `scripts/` 目录下提供了若干可执行 Python 脚本。由于环境变量中已经添加了该目录的位置，这些命令可以在任何目录下被执行。

> 而 `tests/` 目录则提供了若干测试，如 py 模块导入、系统测试等. 在开始进行设计项目前，必须提前通过测试。

2.1 脚本

- sfrun (sfsubmit): 提交任务脚本

  --workdir 指定工作目录，默认为 ${pwd}

  --parameter_file 指定 `parameter.py` 文件，默认为 `./parameter.py`

  --path_file 指定 `path.py` 文件，默认为 `./path.py`

- plot2model: 绘制二进制模型文件的脚本

  --nproc 并行核心数目

  --coordir 指定二进制坐标文件目录，默认为 `../model_init`

  --modeldir 指定二进制模型文件目录，默认为 `./`

  --para 指定需要绘图的模型参数，默认为 [‘vp’]

- ascii2bin: 将 dat 模型文件快速转换为 SEM2D 二进制模型文件的脚本

  --nproc 并行核心数目

  --dir 指定包含所有 dat 模型文件的目录，默认为 `./`

- sfclean: 删除目录下 **所有**反演输出
- sfresume: 继续当前提交的工作. 参考 `sfrun`.
- sfexample: 运行 SeisFlows 示例. **当前不可用**.

2.2 测试

- run_test_system: 测试并行环境
- run_test_import: 测试 python 模块导入
- run_test_optimize:

程序会使用 NBFGS 和 NLCG 方法运行对 $f_{Rosenbrock} = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$ 优化问题的求解

- run_test_preprocess: 测试包含 reader, writer, filter, muting, misfit 和 adjoint 在内的预处理模块
- run_test_tools: 测试其他 python 工具

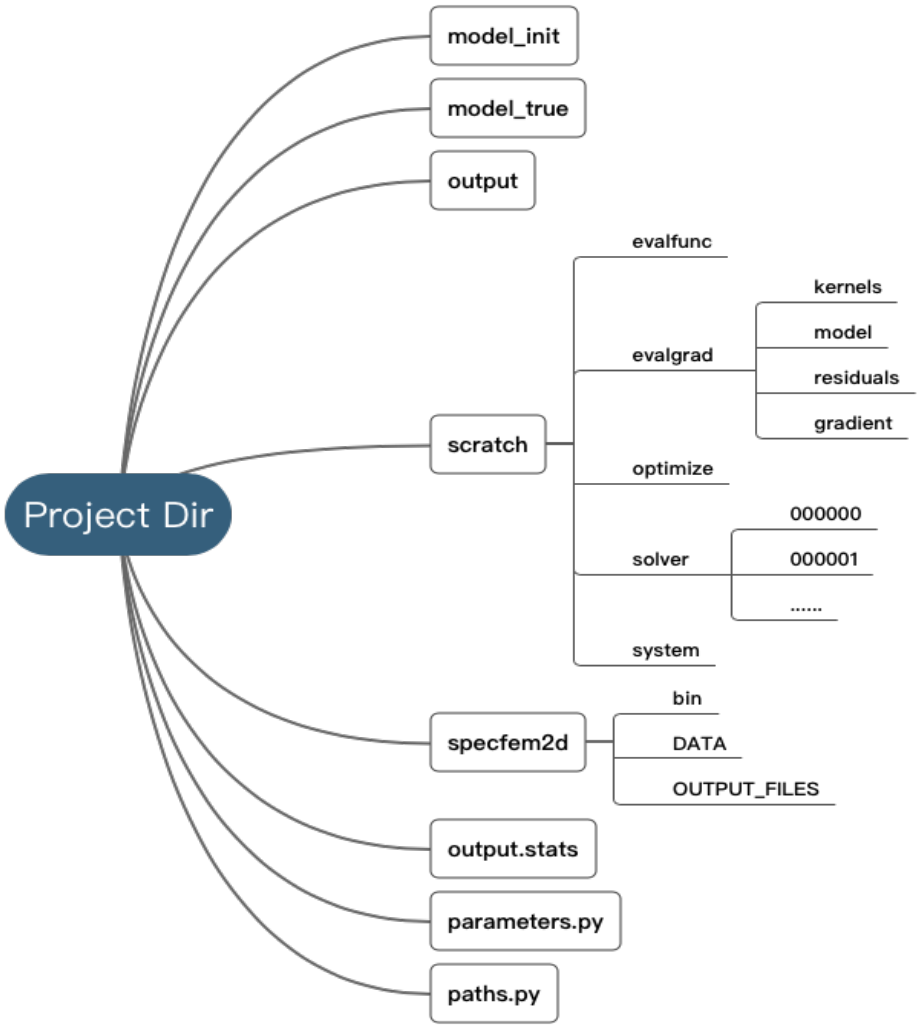
CHAPTER

THREE

设计项目

3.1 项目目录架构

SeisFlows 工作目录包含正演程序、模型和参数文件。下图展示了一个典型的 SeisFlows 项目目录架构:



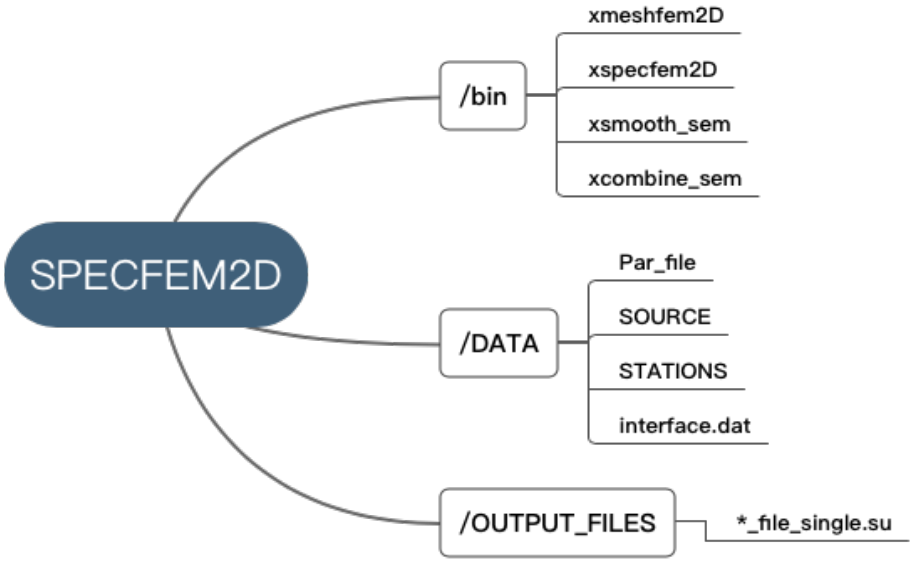
- model_init: 初始模型目录;

- model_true: 真实模型目录；
- output: 输出文件（如迭代模型与梯度）；
- output.stats: 统计参数输出文件；
- scratch: 进行正演、数据处理、敏感核处理的工作目录；
- specfem2d: 正演程序初始目录，参考*正演程序目录: specfem2d*
- parameters.py: 反演参数文件，参考*参数文件: parameters.py*
- paths.py: 路径参数文件，参考 path

`parameters.py` 和 `path.py` 中定义了所有 SeisFlows 的反演参数。在命令行中运行 `sfrun` 或 `sfsubmit` 时，程序会首先从这两个文件中读取参数，所以在提交任务给服务器前，这两个参数需要被仔细地检查以减少不必要的计算资源浪费。

3.2 正演程序目录: `specfem2d`

下图展示了 SPECFEM2D 正演程序目录架构:

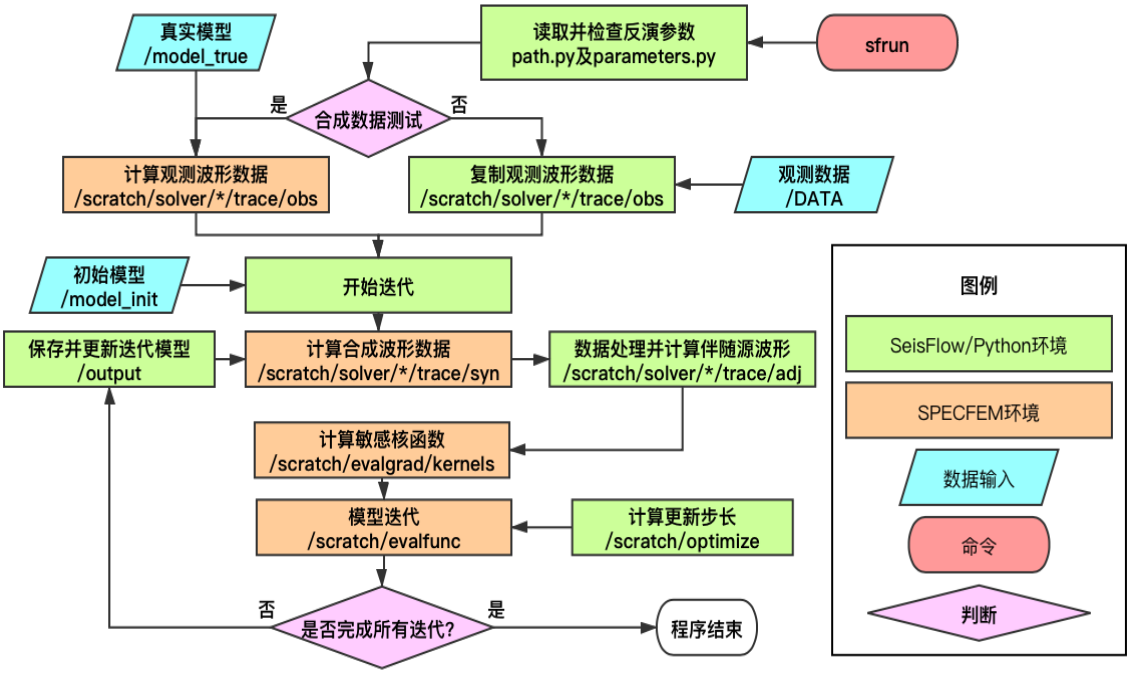


- /bin:
- /bin/xmeshfem2d: 模型网格剖分程序
- /bin/xspecfem2d: 正演计算程序
- /bin/xsmooth_sem: 敏感核平滑程序
- /bin/xcombine_sem: 敏感核求和叠加程序
- /DATA: 正演参数目录
- /DATA/Par_file: 正演参数
- /DATA/SOURCE: 震源参数
- /DATA/STATIONS: 台站参数

- /DATA/interface.dat 正演介质层面参数
- /OUTPUT_FILES: 输出文件目录

3.3 工作流程

SeisFlows 的工作流程图如下所示：



3.4 参数文件: parameters.py

`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.

`General`

`TITLE` Project title.

`WORKFLOW` Workflow specified for seisflows. ‘inversion’ and ‘migration’ are currently supported.

`SOLVER` Time domain solver specified for seisflows. 参考*正演程序配置* and *seisflows.solver package*

`SYSTEM` System type supported for seisflows. 参考 system and *seisflows.system package*

`OPTIMIZE` Optimization method used for inversion. 参考*seisflows.optimize package*

`PREPROCESS` Preprocessing workflow specified. 参考*seisflows.preprocess package*

`POSTPROCESS` Postprocessing workflow specified. ‘base’ needs to be specified.

`MISFIT` Type of misfit for evaluation. 参考*seisflows.plugins.misfit module*

MATERIALS Materials of simulation domain. ‘Elastic’ and ‘Acoustic’ are currently supported. 参考 seisflows.solver.base

`Workflow`

BEGIN First iteration index.

END Last iteration index.

NREC Number of receivers.

NSRC Number of sources. SEM source file needs to be stored with a six-digit index suffix.

SAVEMODEL Frequency of saving model. `1` by default.

SAVEGRADIENT Frequency of saving gradient. `1` by default.

SAVEKERNELS Frequency of saving kernels. `0` by default.

SAVETRACES Frequency of saving traces. `0` by default.

SAVERESIDUALS Frequency of saving residuals. `0` by default.

`Preprocessing`

FORMAT Data file format.

CHANNELS Data channels. Currenly, ‘su’, or ‘SU’ need to be specified.

NORMALIZE Apply normalization for traces. ‘NormalizeEventsL1’, ‘NormalizeEventsL2’, ‘NormalizeTracesL1’, ‘NormalizeTracesL2’ are currently supported. 参考 seisflows.preprocess

`Filter`

BANDPASS Boolean type bandpass switch for traces.

FILTER Type of filter used. 参考*seisflows.preprocess package*

FREQMIN Low frequency corner.

FREQMAX High frequency corner.

`Mute`

MUTE List type switch for trace mute. *seisflows.preprocess package* for supported options.

`Postprocessing`

SMOOTH Smoothing radius. 参考 `xsmooth_sem`

`Optimization`

PRECOND Preconditioner type. 参考 path 和*seisflows.plugins.preconds.diagonal module*

STEPMAX Maximum trial steps

`Solver`

NT Number of time steps defined in Par_file.

DT Time step defined in Par_file.

F0 Dominant frequency defined in SOURCE.

`System`

NTASK Number of tasks submitted. Currently, **NTASK** must satisfy 1 <= NTASK <= NSRC.

NPROC Number of processors.

`MPIEXEC` MPI executable prefix, e.g., `mpirun -np 13`. Note for a space at the end of the string, as seisflows concatenates the prefix with SPECFEM executable command.

3.5 路径文件: `paths.py`

`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.

`DATA` PATH contains seismic data if field data is used for inversion. Data of difference sources should be stored in separate folder. If `DATA` directory does not exist, seisflows would automatically generate synthetic data using model from `MODEL_TRUE`.

`MODEL_INIT` PATH contains model file for initial iteraion.

`MODEL_TRUE` PATH contains true model for generating synthetic data.

`PRECOND` PATH to user supplied diagonal preconditioner. Seisflows will rescale model parameters based on user supplied weights. 参考*seisflows.plugins.preconds.diagonal module*

`MASK` PATH to mask file for gradient scaling. Mask needs to be stored mimicking the file format in which models are stored.

`SPECFEM_DATA` PATH to SPECFEM `DATA` directory which contians Par_file, SOURCE, and other necessary inputs.

`SPECFEM_BIN` PATH to SPECFEM `bin` directory which contains binary executable command of SPECFEM solver.

CHAPTER

FOUR

示例

4.1 本地测试：Checkerboard 测试

SeisFlows 目录 `/path/to/seisflows/example` 下包含一个本地可运行的 SPECFEM2D 棋盘测试项目（**NGPU=1**）。程序在命令行下进入目录，完成参数检查后输入:

```
sfrun
```

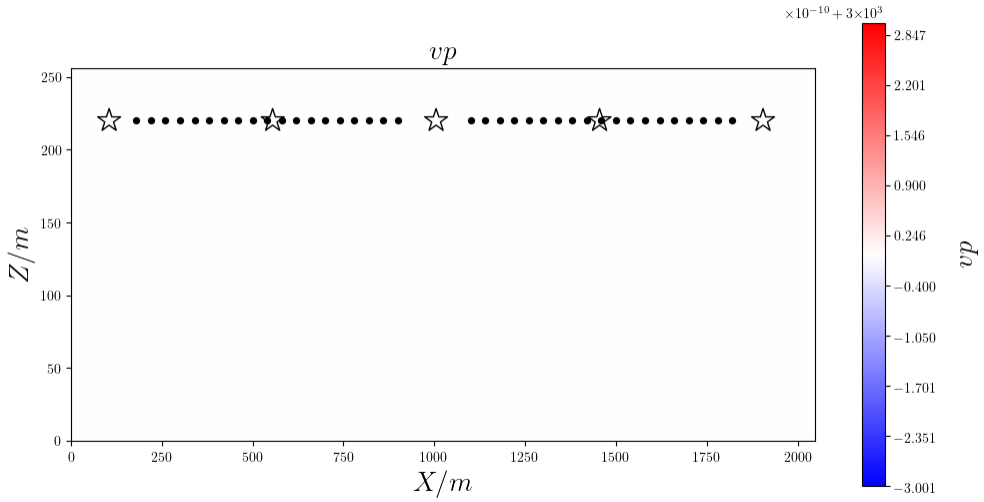
即开始运行 SeisFlows 反演。

4.1.1 震源台站配置

所有震源与台站文件保存于 `./specfem2d/DATA` 中，其中震源文件格式为 **SOURCE_xxxxxx**，多个震源单独保存；台站文件为 **STATIONS**，包含所有台站信息。

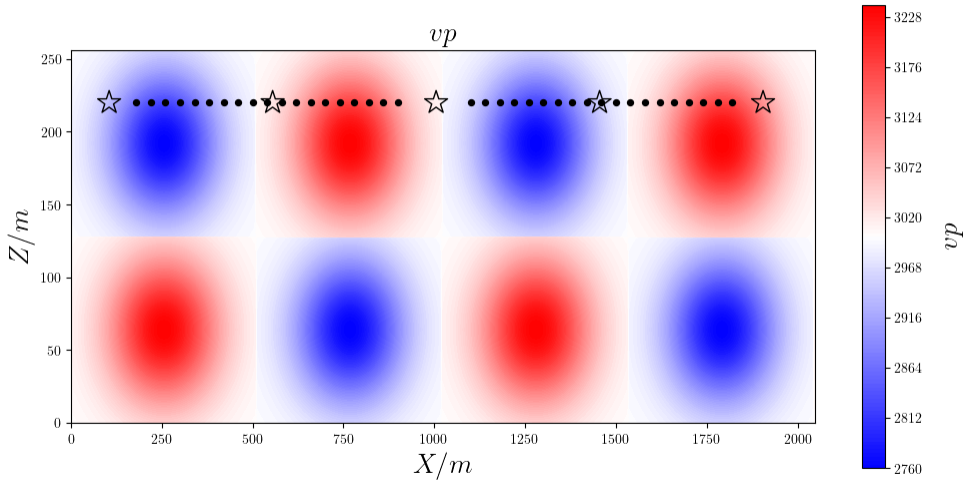
4.1.2 初始模型

反演初始模型保存于目录 `model_init` 中，为 $\rho = 2000kg/m^3, v_p = 3000m/s$ 的均匀 Acoustic 模型。模型长 2048m 米，高 256 米，震源台站设置和 P 波结构如下图所示:



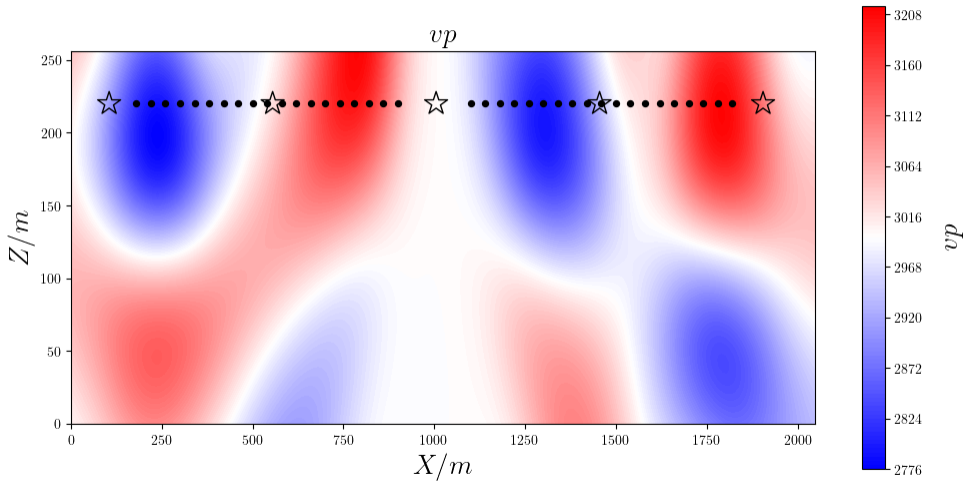
4.1.3 棋盘模型

当 `paths.py` 文件中未给定 **DATA** 目录时，Seisflows 会利用 `model_true` 目录下给定的模型计算合成数据并从初始模型开始进行反演。棋盘模型长 2048m 米，高 256 米，震源台站设置和 P 波结构如下图所示：

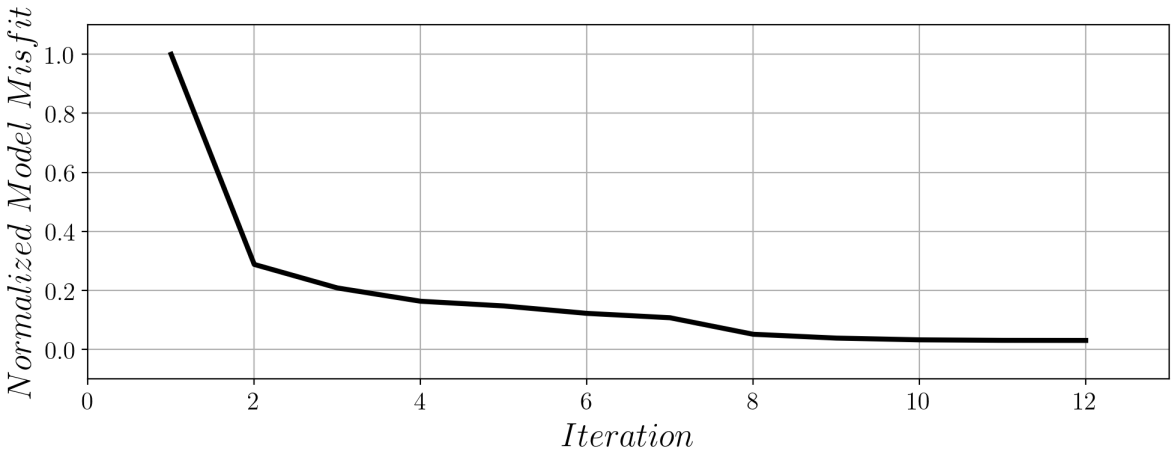


4.1.4 反演结果

反演参数定义于 `parameters.py` 内，采用最速下降法进行 11 次迭代后模型如下图所示：



归一化目标函数下降曲线如下图所示：


Normalized Model Misfit
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
Iteration

CHAPTER

FIVE

模块

5.1 seisflows.system package

5.1.1 seisflows.system.base module

class seisflows.system.base.**base**

Bases: object

Abstract base class

check()

Checks parameters and paths

checkpoint(*path*, *classname*, *method*, *args*, *kwargs*)

Writes information to disk so tasks can be executed remotely

run(*classname*, *method*, **args*, ***kwargs*)

Runs task multiple times

run_single(*classname*, *method*, **args*, ***kwargs*)

Runs task a single time

submit()

Submits workflow

taskid()

Provides a unique identifier for each running task

5.1.2 seisflows.system.lsf_lg module

class seisflows.system.lsf_lg.**lsf_lg**

Bases: *seisflows.system.base.base*

An interface through which to submit workflows, run tasks in serial or parallel, and perform other system functions.

By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

Intermediate files are written to a global scratch path PATH.SCRATCH, which must be accessible to all compute nodes.

Optionally, users can provide a local scratch path PATH.LOCAL if each compute node has its own local filesystem.

For important additional information, please see http://seisflows.readthedocs.org/en/latest/manual/manual.html#system-configuration

check()
Checks parameters and paths

job_id_list(*stdout*)

job_status(*classname*, *method*, *jobs*)

mpiexec()
Specifies MPI executable used to invoke solver

run(*classname*, *method*, *hosts='all'*, ***kwargs*)
Runs task multiple times in embarrassingly parallel fasion

Executes classname.method(*args, **kwargs) NTASK times, each time on NPROC cpu cores

run_single(*classname*, *method*, *hosts='all'*, ***kwargs*)
Runs task multiple times in embarrassingly parallel fasion

Executes classname.method(*args, **kwargs) NTASK times, each time on NPROC cpu cores

save_kwargs(*classname*, *method*, *kwargs*)

submit(*workflow*)
Submits workflow

taskid()
Provides a unique identifier for each running task

timestamp()

5.1.3 seisflows.system.lsf_sm module

5.1.4 seisflows.system.multicore module

class seisflows.system.multicore.**multicore**
Bases: *seisflows.system.serial.serial*

An interface through which to submit workflows, run tasks in serial or parallel, and perform other system functions.

By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

For important additional information, please see http://seisflows.readthedocs.org/en/latest/manual/manual.html#system-configuration

check()
Checks parameters and paths

run(*classname*, *method*, **args*, ***kwargs*)
Runs task multiple times in embarrassingly parallel fasion

Executes classname.method(*args, **kwargs) NTASK times, each time on NPROC cpu cores

run_single(*classname*, *method*, **args*, ***kwargs*)
Runs task a single time

save_kwargs(*classname*, *method*, *kwargs*)

submit(*workflow*)
Submits job

5.1.5 seisflows.system.multithreaded module

class seisflows.system.multithreaded.**multithreaded**
Bases: *seisflows.system.multicore.multicore*

An interface through which to submit workflows, run tasks in serial or parallel, and perform other system functions.

By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

For important additional information, please see http://seisflows.readthedocs.org/en/latest/manual/manual.html#system-configuration

check()
Checks parameters and paths

5.1.6 seisflows.system.pbs_lg module

class seisflows.system.pbs_lg.**pbs_lg**
Bases: *seisflows.system.base.base*

An interface through which to submit workflows, run tasks in serial or parallel, and perform other system functions.

By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

Intermediate files are written to a global scratch path PATH.SCRATCH, which must be accessible to all compute nodes.

Optionally, users can provide a local scratch path PATH.LOCAL if each compute node has its own local filesystem.

For important additional information, please see http://seisflows.readthedocs.org/en/latest/manual/manual.html#system-configuration

check()
Checks parameters and paths

job_array_args(*hosts*)

job_array_cmd(*classname*, *method*, *hosts*)

job_array_status(*classname*, *method*, *jobs*)
Determines completion status of one or more jobs

mpiexec()
Specifies MPI executable used to invoke solver

run(*classname*, *method*, *hosts='all'*, ***kwargs*)
Runs task multiple times in embarrassingly parallel fasion

Executes classname.method(*args, **kwargs) NTASK times, each time on NPROC cpu cores

save_kwargs(*classname*, *method*, *kwargs*)

submit(*workflow*)
Submits workflow

submit_job_array(*classname*, *method*, *hosts='all'*)

taskid()
Provides a unique identifier for each running task

5.1.7 seisflows.system.pbs_sm module

class seisflows.system.pbs_sm.**pbs_lg**
Bases: *seisflows.system.base.base*

An interface through which to submit workflows, run tasks in serial or parallel, and perform other system functions.

By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

Intermediate files are written to a global scratch path PATH.SCRATCH, which must be accessible to all compute nodes.

Optionally, users can provide a local scratch path PATH.LOCAL if each compute node has its own local filesystem.

For important additional information, please see http://seisflows.readthedocs.org/en/latest/manual/manual.html#system-configuration

check()
Checks parameters and paths

hostlist()
Generates list of allocated cores

mpiexec()
Specifies MPI executable used to invoke solver

run(*classname*, *method*, *hosts='all'*, ***kwargs*)
Runs embarrassingly parallel tasks

Executes the following multiple times: classname.method(*args, **kwargs)

system.taskid serves to provide each running task a unique identifier

save_kwargs(*classname*, *method*, *kwargs*)

submit(*workflow*)
Submits workflow

taskid()
Provides a unique identifier for each running task

5.1.8 seisflows.system.serial module

class seisflows.system.serial.**serial**
Bases: *seisflows.system.base.base*

An interface through which to submit workflows, run tasks in serial or parallel, and perform other system functions.

By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

For important additional information, please see http://seisflows.readthedocs.org/en/latest/manual/manual.html#system-configuration

check()
Checks parameters and paths

mpiexec()
Specifies MPI executable used to invoke solver

progress(*taskid*)
Provides status update

run(*classname*, *method*, *hosts='all'*, ***kwargs*)
Executes method from classname multiple times in serial taskid is used to identified a given task (one source)

run_single(*classname*, *method*, **args*, ***kwargs*)
Runs task a single time

submit(*workflow*)
Submits job

taskid()
Return the value of the environment variable SEISFLOWS_TASKID which provides a unique identifier for each running task

5.1.9 seisflows.system.slurm_lg module

class seisflows.system.slurm_lg.**slurm_lg**
Bases: *seisflows.system.base.base*

An interface through which to submit workflows, run tasks in serial or parallel, and perform other system functions.

By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

Intermediate files are written to a global scratch path PATH.SCRATCH, which must be accessible to all compute nodes.

Optionally, users can provide a local scratch path PATH.LOCAL if each compute node has its own local filesystem.

For important additional information, please see http://seisflows.readthedocs.org/en/latest/manual/manual.html#system-configuration

check()
Checks parameters and paths

job_array_status(*classname*, *method*, *jobs*)
Determines completion status of job or job array

job_id_list(*stdout*, *ntask*)
Parses job id list from sbatch standard output

job_status(*job*)
Queries completion status of a single job

mpiexec()
Specifies MPI executable used to invoke solver

run(*classname*, *method*, **args*, ***kwargs*)
Runs task multiple times in embarrassingly parallel fasion

Executes classname.method(*args, **kwargs) NTASK times, each time on NPROC cpu cores

run_single(*classname*, *method*, **args*, ***kwargs*)
Runs task a single time

Executes classname.method(*args, **kwargs) a single time on NPROC cpu cores

submit(*workflow*)
Submits workflow

taskid()
Provides a unique identifier for each running task

5.1.10 seisflows.system.slurm_sm module

class seisflows.system.slurm_sm.**slurm_sm**
Bases: *seisflows.system.base.base*

An interface through which to submit workflows, run tasks in serial or parallel, and perform other system functions.

By hiding environment details behind a python interface layer, these classes provide a consistent command set across different computing environments.

Intermediate files are written to a global scratch path PATH.SCRATCH, which must be accessible to all compute nodes.

Optionally, users can provide a local scratch path PATH.LOCAL if each compute node has its own local filesystem.

For important additional information, please see http://seisflows.readthedocs.org/en/latest/manual/manual.html#system-configuration

check()
Checks parameters and paths

hostlist()
Generates list of allocated cores

mpiexec()
Specifies MPI executable used to invoke solver

run(*classname*, *method*, **args*, ***kwargs*)
Runs task multiple times in embarrassingly parallel fasion

run_single(*classname*, *method*, **args*, ***kwargs*)
Runs task a single time

save_kwargs(*classname*, *method*, *kwargs*)

submit(*workflow*)
Submits workflow

taskid()
Provides a unique identifier for each running task

5.1.11 seisflows.system.tiger_lg module

class seisflows.system.tiger_lg.tiger_lg
Bases: *seisflows.system.slurm_lg.slurm_lg*

Specially designed system interface for tiger.princeton.edu

See parent class for more information.

check()
Checks parameters and paths

submit(*args, **kwargs)
Submits job

5.1.12 seisflows.system.tiger_sm module

class seisflows.system.tiger_sm.tiger_sm
Bases: *seisflows.system.slurm_sm.slurm_sm*

Specially designed system interface for tiger.princeton.edu

See parent class for more information.

check()
Checks parameters and paths

submit(*args, **kwargs)
Submits job

5.2 seisflows.solver package

5.2.1 seisflows.solver.base module

class seisflows.solver.base.base
Bases: object

Provides an interface through which solver simulations can be set up and run and a parent class for SPECFEM2D, SPECFEM3D and SPECFEM3D_GLOBE subclasses

This class supports only acoustic and isotropic elastic inversions. For additional options, see github.com/rmodrak/seisflows-multiparameter

eval_func, eval_grad, apply_hess These methods deal with evaluation of the misfit function or its derivatives. Together, they provide the primary interface through which SeisFlows interacts with SPECFEM2D/3D

forward, adjoint These methods allow direct access to low-level SPECFEM2D/3D components, providing an alternative interface through which to interact with the solver

setup, generate_data, generate_model One-time operations performed at beginning of an inversion or migration

initialize_solver_directories, initialize_adjoint_traces

SPECFEM2D/3D requires a particular directory structure in which to run and particular file formats for models, data, and parameter files. These methods help put in place all these prerequisites

load, save For reading and writing SPECFEM2D/3D models and kernels. On the disk, models and kernels are stored as binary files, and in memory, as dictionaries with different keys corresponding to different material parameters

split, merge Within the solver routines, it is natural to store models as dictionaries. Within the optimization routines, it is natural to store models as vectors. Two methods, 'split' and 'merge', are used to convert back and forth between these two representations

combine, smooth Utilities for combining and smoothing kernels

`adjoint()`
Calls adjoint solver

`apply_hess(`*path=''*`)`
Computes action of Hessian on a given model vector. (A gradient evaluation must have already been carried out.)

Input path : directory to which output files are exported

`check()`
Checks parameters and paths

`check_mesh_properties(`*path=None*`)`
path contains binary files such as: proc000000_z.bin, proc000000_x.bin, proc000000_vs.bin proc000000_vp.bin, proc000000_rho.bin, proc000001_z.bin, proc000001_x.bin, proc000001_vs.bin ⋯These will be read to get the number of processors used, the number of gll points and the coordinates of those points

`check_solver_parameter_files()`

`check_source_names()`
Determines names of sources by applying wildcard rule to user- supplied input files If source_prefix is 'SOURCE' and that in specfem DATA folder are the files SOURCE_00001, SOURCE_00002, SOURCE_00003, ⋯Then this will build the list names = ['00001','00002','00003',⋯] If, for ex, taskid is 1 the function returns ['00001', '00002']

`clean()`

`combine(`*input_path=''*, *output_path=''*, *parameters=[]*`)`
Sums individual source contributions. Wrapper over xcombine_sem utility.

`cwd`

`data_filenames`

`eval_func(`*path=''*, *export_traces=False*, *write_residuals=True*`)`
Performs forward simulations needed for misfit function evaluation

Input path : directory from which model is imported

Input export_traces : save or discard traces?

`eval_grad(`*path=''*, *export_traces=False*`)`
Evaluates gradient by carrying out adjoint simulations. (A function evaluation must already have been carried out.)

Input path : directory from which model is imported

Input export_traces : save or discard traces?

`export_kernels(`*path*`)`

`export_model(`*path, parameters=['rho', 'vp', 'vs']*`)`

`export_residuals`(*path*)

`export_traces`(*path*, *prefix='traces/obs'*)

`forward`()
Calls forward solver

`generate_data`(**args*, ***kwargs*)
Generates data

`generate_mesh`(**args*, ***kwargs*)
Performs meshing and database generation

`import_model`(*path*)

`import_traces`(*path*)

`initialize_adjoint_traces`()
Puts in place "adjoint traces" expected by SPECFEM Adjoint traces are initialized by writing zeros for all channels. (even the ones that are not actually in use, as required by specfem) Ex: Ux, Uy, Uz even if only Uy is used Channels actually in use during an inversion or migration will be overwritten with nonzero values later on.

`initialize_solver_directories`()
Creates directory structure expected by SPECFEM3D, copies executables, and prepares input files. Executables must be supplied by user as there is currently no mechanism for automatically compiling from source.

`io`
Solver IO module

`kernel_databases`

`load`(*path*, *parameters=[]*, *prefix=''*, *suffix=''*)
Loads SPECFEM2D/3D models or kernels

Input path : directory from which model is read

Input parameters : list of material parameters to be read (if empty, defaults to self.parameters)

Input prefix : optional filename prefix

Input suffix : optional filename suffix, eg '_kernel'

Output dict : model or kernels indexed by material parameter and processor rank, ie dict[parameter][iproc]

`merge`(*model*, *parameters=[]*)
Converts model from dictionary to vector representation

`mesh_properties`

`model_databases`

`rename_data`(*path*)
Works around conflicting data filename conventions

`rename_kernels`()
Works around conflicting kernel filename conventions

`save`(*dict, path, parameters=['vp', 'vs', 'rho'], prefix='', suffix=''*)
Saves SPECFEM2D/3D models or kernels

Input dict : model stored as a dictionary or Container

Input path : directory to which model is written

Input parameters : list of material parameters to be written

Input prefix : optional filename prefix

Input suffix : optional filename suffix, eg '_kernel'

`setup()`
Prepares solver for inversion or migration Sets up directory structure expected by SPECFEM and copies or generates seismic data to be inverted or migrated

`smooth(input_path='', output_path='', parameters=[], span=0.0)`
Smooths kernels by convolving them with a Gaussian. Wrapper over xsmooth_sem utility.

`source_name`

`source_names`

`source_prefix`

`split(m, parameters=[])`
Converts model from vector to dictionary representation

`taskid`

5.2.2 seisflows.solver.specfem2d module

`class seisflows.solver.specfem2d.specfem2d`
Bases: *seisflows.solver.base.base*

Python interface for SPECFEM2D

See base class for method descriptions

`adjoint()`
Calls SPECFEM2D adjoint solver

`check()`
Checks parameters and paths

`check_solver_parameter_files()`
Checks solver parameters

`data_filenames`

`export_model(path)`

`forward(path='traces/syn')`
Calls SPECFEM2D forward solver

`generate_data(**model_kwargs)`
Generates data (perform meshing and database generation first)

`generate_mesh(model_path=None, model_name=None, model_type='gll')`
Performs meshing and database generation

`import_model(path)`

`initialize_adjoint_traces()`
Puts in place "adjoint traces" expected by SPECFEM Adjoint traces are initialized by writing zeros for all channels. (even the ones that are not actually in use, as required by specfem) Ex: Ux, Uy, Uz even if only Uy is used Channels actually in use during an inversion or migration will be overwritten with nonzero values later on.

`kernel_databases`

model_databases

source_prefix

5.2.3 seisflows.solver.specfem3d module

class seisflows.solver.specfem3d.**specfem3d**

Bases: *seisflows.solver.base.base*

Python interface for SPECFEM3D

See base class for method descriptions

adjoint()
Calls SPECFEM3D adjoint solver

check()
Checks parameters and paths

check_solver_parameter_files()
Checks solver parameters

data_filenames

data_wildcard

eval_func(**args*, ***kwargs*)
Performs forward simulations needed for misfit function evaluation

Input path : directory from which model is imported

Input export_traces : save or discard traces?

forward(*path='traces/syn'*)
Calls SPECFEM3D forward solver

generate_data(***model_kwargs*)
Generates data

generate_mesh(*model_path=None*, *model_name=None*, *model_type='gll'*)
Performs meshing and database generation

initialize_adjoint_traces()
Puts in place "adjoint traces" expected by SPECFEM Adjoint traces are initialized by writing zeros for all channels. (even the ones that are not actually in use, as required by specfem) Ex: Ux, Uy, Uz even if only Uy is used Channels actually in use during an inversion or migration will be overwritten with nonzero values later on.

kernel_databases

model_databases

rename_data()
Works around conflicting data filename conventions

source_prefix

write_parameters()

write_receivers()

write_sources()

5.2.4 seisflows.solver.specfem3d_globe module

class seisflows.solver.specfem3d_globe.**specfem3d_globe**

Bases: *seisflows.solver.base.base*

Python interface for SPECFEM3D_GLOBE

See base class for method descriptions

adjoint()
Calls SPECFEM3D_GLOBE adjoint solver

check()
Checks parameters and paths

check_mesh_properties(*path=None*, *parameters=None*)
path contains binary files such as: proc000000_z.bin, proc000000_x.bin, proc000000_vs.bin proc000000_vp.bin, proc000000_rho.bin, proc000001_z.bin, proc000001_x.bin, proc000001_vs.bin ⋯These will be read to get the number of processors used, the number of gll points and the coordinates of those points

data_filenames

forward(*path='traces/syn'*)
Calls SPECFEM3D_GLOBE forward solver

generate_data(***model_kwargs*)
Generates data

generate_mesh(*model_path=None*, *model_name=None*, *model_type='gll'*)
Performs meshing and database generation

initialize_adjoint_traces()
Puts in place "adjoint traces" expected by SPECFEM Adjoint traces are initialized by writing zeros for all channels. (even the ones that are not actually in use, as required by specfem) Ex: Ux, Uy, Uz even if only Uy is used Channels actually in use during an inversion or migration will be overwritten with nonzero values later on.

kernel_databases

load(*path*, *prefix='reg1_'*, *suffix=''*, *verbose=False*)
reads SPECFEM model or kernel

Models are stored in Fortran binary format and separated into multiple files according to material parameter and processor rank.

model_databases

rename_data()
Works around conflicting data filename conventions

save(*path*, *model*, *prefix='reg1_'*, *suffix=''*)
writes SPECFEM3D_GLOBE transerverly isotropic model

source_prefix

5.3 seisflows.workflow package

5.3.1 seisflows.workflow.base module

class seisflows.workflow.base.base
Bases: object

Workflow abstract base class

check()
Checks parameters and paths

checkpoint()
Writes information to disk so workflow can be resumed following a break

main()
Main routine

Execution of a workflow is equivalent to stepping through workflow.main

5.3.2 seisflows.workflow.inversion module

class seisflows.workflow.inversion.inversion
Bases: *seisflows.workflow.base.base*

Waveform inversion base class

Peforms iterative nonlinear inversion and provides a base class on top of which specialized strategies can be implemented.

To allow customization, the inversion workflow is divided into generic methods such as ‘initialize’, ‘finalize’, ‘evaluate_function’, ‘evaluate_gradient’, which can be easily overloaded.

Calls to forward and adjoint solvers are abstracted through the ‘solver’ interface so that various forward modeling packages can be used interchangeably.

Commands for running in serial or parallel on a workstation or cluster are abstracted through the ‘system’ interface.

check()
Checks parameters and paths

checkpoint()
Writes information to disk so workflow can be resumed following a break

clean()
Cleans directories in which function and gradient evaluations were carried out

compute_direction()
Computes search direction

evaluate_function()
Performs forward simulation to evaluate objective function

evaluate_gradient()
Performs adjoint simulation to evaluate gradient

finalize()
Saves results from current model update iteration

`initialize()`
Prepares for next model update iteration

`line_search()`
Conducts line search in given search direction

Status codes status > 0 : finished status == 0 : not finished status < 0 : failed

`main()`
Carries out seismic inversion

`save_gradient()`

`save_kernels()`

`save_model()`

`save_residuals()`

`save_traces()`

`setup()`
Lays groundwork for inversion. Runs setup method for preprocess, postprocess, optimize and also for the solver through the system: system.run('solver','setup') This will copy user supplied data or generate it from a 'real' model supplied This will setup the mesh for the initial model supplied If path LOCAL is supplied this is done even at iteration > 1

`write_gradient(`*path=''*, *suffix=''*`)`
Writes gradient in format expected by nonlinear optimization library

`write_misfit(`*path=''*, *suffix=''*`)`
Writes misfit in format expected by nonlinear optimization library

`write_model(`*path=''*, *suffix=''*`)`
Writes model in format expected by solver

5.3.3 seisflows.workflow.migration module

`class seisflows.workflow.migration.migration`
Bases: *seisflows.workflow.base.base*

Migration base class.

In the terminology of seismic exploration, implements a 'reverse time migration'.

`check()`
Checks parameters and paths

`main()`
Migrates seismic data

`prepare_model()`

`save_kernels()`

`save_kernels_sum()`

`save_traces()`

5.3.4 seisflows.workflow.test_adjoint module

seisflows.workflow.test_adjoint.**DotProductLHS**(*keys*, *x*, *y*)

seisflows.workflow.test_adjoint.**DotProductRHS**(*keys*, *x*, *y*)

class seisflows.workflow.test_adjoint.**test_adjoint**

Bases: *seisflows.workflow.base.base*

check()

Checks parameters and paths

event

main()

Main routine

Execution of a workflow is equivalent to stepping through workflow.main

prepare_model()

5.3.5 seisflows.workflow.test_forward module

class seisflows.workflow.test_forward.**test_forward**

Bases: *seisflows.workflow.base.base*

Tests solver by running forward simulation

check()

Checks parameters and paths

main()

Generates seismic data

5.3.6 seisflows.workflow.test_optimize module

class seisflows.workflow.test_optimize.**test_optimize**

Bases: *seisflows.workflow.base.base*

Optimization unit test.

Tests nonlinear optimization procedure with inexpensive test function.

check()

Checks parameters and paths

compute_direction()

compute_direction_newton()

evaluate_function()

evaluate_gradient()

finalize()

line_search()

main()

Main routine

Execution of a workflow is equivalent to stepping through workflow.main

setup()

status(*m_new*, *m_old*)

5.3.7 seisflows.workflow.test_postprocess module

class seisflows.workflow.test_postprocess.**test_postprocess**

Bases: *seisflows.workflow.base.base*

Postprocessing class

check()
Checks parameters and paths

main()
Writes gradient of objective function

5.3.8 seisflows.workflow.test_preprocess module

class seisflows.workflow.test_preprocess.**test_preprocess**

Bases: *seisflows.workflow.base.base*

Signal processing integration test

check()
Checks parameters and paths

main()
Tests data processing methods

save(*data*, *filename*)

test_adjoint(*dat*, *syn*)

test_filter(*dat*)

test_misfit(*dat*, *syn*)

test_mute(*dat*)

test_normalize(*dat*)

test_reader()

test_writer(*data*)

5.3.9 seisflows.workflow.test_system module

class seisflows.workflow.test_system.**test_system**

Bases: *seisflows.workflow.base.base*

Tests system interface

check()
Checks parameters and paths

hello(*msg='Hello from %d'*)
Prints hello message

main()
Main routine

Execution of a workflow is equivalent to stepping through workflow.main

5.3.10 seisflows.workflow.thrifty_inversion module

class seisflows.workflow.thrifty_inversion.**thrifty_inversion**
Bases: *seisflows.workflow.inversion.inversion*

Thrifty inversion subclass

Provides savings over conventional inversion by carrying over forward simulations from line search

The results of ‘inversion’ and ‘thrifty_inversion’ should be exactly the same

clean()
Cleans directories in which function and gradient evaluations were carried out

initialize()
Prepares for next model update iteration

status = 0

update_status()

5.4 seisflows.preprocess package

5.4.1 seisflows.preprocess.base module

class seisflows.preprocess.base.**base**
Bases: object

Data preprocessing class

Provides data processing functions for seismic traces, with options for data misfit, filtering, normalization and muting

apply_csg_mute(*traces*)

apply_filter(*traces*)

apply_filter_backwards(*traces*)

apply_mute(*traces*)

apply_normalize(*traces*)

check()
Checks parameters and paths

check_filter()
Checks filter settings

check_mute()
Checks mute settings

check_normalize()

csg_mute(*seismo_w*, *T*, *T1*)

get_network_size(*traces*)

`get_receiver_coords`(*traces*)

`get_source_coords`(*traces*)

`get_time_scheme`(*traces*)
FIXME: extract time scheme from trace headers rather than parameters file. Note from Alexis Bottero : it is actually better like this in my opinion because this allows for longer traces to be processed. Indeed, in su format only 2 bytes are dedicated to the number of samples which is supposed to be stored as an unsigned int. The maximum NT which can be stored in the header is then 32762 whereas there is no limit in principle.

`prepare_eval_grad`(*path='.'*)
Prepares solver for gradient evaluation by writing residuals and adjoint traces

Input path directory containing observed and synthetic seismic data

`setup`()
Sets up data preprocessing machinery

`sum_residuals`(*files*)
Sums squares of residuals

Input files list of single-column text files containing residuals

Output total_misfit sum of squares of residuals

`write_adjoint_traces`(*path*, *syn*, *obs*, *channel*)
Writes "adjoint traces" required for gradient computation

Input path location "adjoint traces" will be written

Input syn obspy Stream object containing synthetic data

Input obs obspy Stream object containing observed data

Input channel channel or component code used by writer

`write_residuals`(*path*, *syn*, *obs*)
Computes residuals

Input path location "adjoint traces" will be written

Input syn obspy Stream object containing synthetic data

Input obs obspy Stream object containing observed data

5.4.2 seisflows.preprocess.default module

`class seisflows.preprocess.default.default`
Bases: *seisflows.preprocess.base.base*

Default preprocesing class

Provides data processing functions for seismic traces, with options for data misfit, filtering, normalization and muting

5.4.3 seisflows.preprocess.double_difference module

`class seisflows.preprocess.double_difference.double_difference`
Bases: *seisflows.preprocess.base.base*

Double-difference data processing class

Adds double-difference data misfit functions to base class

adjoint_dd(*si*, *sj*, *t0*, *nt*, *dt*)
Returns contribution to adjoint source from a single double difference measurement

apply_weights(*traces*)

check()
Checks parameters, paths, and dependencies

distance(*x1*, *y1*, *x2*, *y2*)

load_weights()

shift(*v*, *it*)
Shifts time series a given number of steps

sum_residuals(*files*)
Sums squares of residuals

write_adjoint_traces(*path*, *syn*, *dat*, *channel*)
Computes adjoint traces from observed and synthetic traces

write_residuals(*path*, *syn*, *dat*)
Computes residuals from observations and synthetics

5.5 seisflows.postprocess package

5.5.1 seisflows.postprocess.base module

class seisflows.postprocess.base.**base**
Bases: object

Regularization, smoothing, sharpening, masking and related operations on models or gradients

check()
Checks parameters and paths

process_kernels(*path*, *parameters*)
Sums kernels from individual sources, with optional smoothing

Input path directory containing sensitivity kernels

Input parameters list of material parameters e.g. [‘vp’,’vs’]

setup()
Placeholder for initialization or setup tasks

write_gradient(*path*)
Combines contributions from individual sources and material parameters to get the gradient, and optionally applies user-supplied scaling

Input path directory from which kernels are read and to which gradient is written

5.5.2 seisflows.postprocess.default module

class seisflows.postprocess.default.**default**
Bases: *seisflows.postprocess.base.base*

Default postprocesing option

Provides default image processing and regularization functions for models or gradients

5.6 seisflows.optimize package

5.6.1 seisflows.optimize.base module

class seisflows.optimize.base.base

Bases: object

Nonlinear optimization abstract base class

Base class on top of which steepest descent, nonlinear conjugate, quasi- Newton and Newton methods can be implemented. Includes methods for both search direction and line search.

To reduce memory overhead, vectors are read from disk rather than passed from calling routines. For example, at the beginning of compute_direction the current gradient is read from 'g_new' and the resulting search direction is written to 'p_new'. As the inversion progresses, other information is stored as well.

Variables m_new - current model m_old - previous model m_try - line search model f_new - current objective function value f_old - previous objective function value f_try - line search function value g_new - current gradient direction g_old - previous gradient direction p_new - current search direction p_old - previous search direction

check()
Checks parameters, paths, and dependencies

compute_direction()
Computes search direction

dot(*x*, *y*)
Computes inner product between vectors

finalize_search()
Prepares algorithm machinery and scratch directory for next model upate

initialize_search()
Determines first step length in line search

load(*filename*)

loadtxt(*filename*)

restart()
Restarts nonlinear optimization algorithm

Keeps current position in model space, but discards history of nonlinear optimization algorithm in an attempt to recover from numerical stagnation

retry_status()
Determines if restart is worthwhile

After failed line search, determines if restart is worthwhile by checking, in effect, if search direction was the same as gradient direction

save(*filename*, *array*)

savetxt(*filename*, *scalar*)

setup()
Sets up nonlinear optimization machinery

`update_search()`
Updates line search status and step length

Status codes status > 0 : finished status == 0 : not finished status < 0 : failed

5.6.2 seisflows.optimize.LBFGS module

`class seisflows.optimize.LBFGS.LBFGS`
Bases: *seisflows.optimize.base.base*

Limited memory BFGS algorithm

`check()`
Checks parameters, paths, and dependencies

`compute_direction()`
Computes search direction

`restart()`
Restarts nonlinear optimization algorithm

Keeps current position in model space, but discards history of nonlinear optimization algorithm in an attempt to recover from numerical stagnation

`setup()`
Sets up nonlinear optimization machinery

5.6.3 seisflows.optimize.NLCG module

`class seisflows.optimize.NLCG.NLCG`
Bases: *seisflows.optimize.base.base*

Nonlinear conjugate gradient method

`check()`
Checks parameters, paths, and dependencies

`compute_direction()`
Computes search direction

`restart()`
Restarts nonlinear optimization algorithm

Keeps current position in model space, but discards history of nonlinear optimization algorithm in an attempt to recover from numerical stagnation

`setup()`
Sets up nonlinear optimization machinery

5.6.4 seisflows.optimize.steepest_descent module

`class seisflows.optimize.steepest_descent.steepest_descent`
Bases: *seisflows.optimize.base.base*

Steepest descent method

`check()`
Checks parameters, paths, and dependencies

`compute_direction()`
Computes search direction

`restart()`
Restarts nonlinear optimization algorithm

Keeps current position in model space, but discards history of nonlinear optimization algorithm in an attempt to recover from numerical stagnation

`restarted = False`

`setup()`
Sets up nonlinear optimization machinery

5.7 seisflows.plugins package

5.7.1 seisflows.plugins.line_search package

seisflows.plugins.line_search.backtrack module

`class seisflows.plugins.line_search.backtrack.Backtrack(step_count_max=10, step_len_max=inf, path='/Users/niyiyu/Documents/GitHub/seisflows/docs')`

Bases: *seisflows.plugins.line_search.bracket.Bracket*

Implements backtracking linesearch

Variables x - list of step lenths from current line search f - correpsonding list of function values gtg - dot product of gradient with itself gtp - dot product of gradient and search direction

Status codes status > 0 : finished status == 0 : not finished status < 0 : failed

`calculate_step()`
Determines step length and search status

seisflows.plugins.line_search.base module

`class seisflows.plugins.line_search.base.Base(step_count_max=10, step_len_max=inf, path='/Users/niyiyu/Documents/GitHub/seisflows/docs')`

Bases: `object`

Abstract base class for line search

Variables x - list of step lenths from current line search f - correpsonding list of function values m - how many step lengths in current line search? n - how many model updates in optimization problem? gtg - dot product of gradient with itself gtp - dot product of gradient and search direction

Status codes status > 0 : finished status == 0 : not finished status < 0 : failed

`calculate_step()`

`clear_history()`
Clears line search history

`initialize(step_len, func_val, gtg, gtp)`

`search_history(sort=True)`
A convenience function, collects information needed to determine search status and calculate step length

update(*step_len*, *func_val*)

class seisflows.plugins.line_search.base.Writer(*path='./output.optim'*)
Bases: object

Utility for writing one or more columns to text file

newline()

write_header()

seisflows.plugins.line_search.bracket module

class seisflows.plugins.line_search.bracket.Bracket(*step_count_max=10*, *step_len_max=inf*, *path='/Users/niyiyu/Documents/GitHub/seisflows/docs'*)
Bases: *seisflows.plugins.line_search.base.Base*

Implements bracketing line search

Variables x - list of step lenths from current line search f - correpsonding list of function values gtg - dot product of gradient with itself gtp - dot product of gradient and search direction

Status codes status > 0 : finished status == 0 : not finished status < 0 : failed

calculate_step()
Determines step length and search status

5.7.2 seisflows.plugins.optimize package

seisflows.plugins.optimize.LBFGS module

class seisflows.plugins.optimize.LBFGS.LBFGS(*path='.'*, *load=<function loadnpy>*, *save=<function savenpy>*, *memory=5*, *thresh=0.0*, *maxiter=inf*, *precond=None*)
Bases: object

Limited-memory BFGS algorithm

Includes optional safeguards: periodic restarting and descent conditions.

To conserve memory, most vectors are read from disk rather than passed from a calling routine.

apply(*q*, *S=[]*, *Y=[]*)
Applies L-BFGS inverse Hessian to given vector

check_status(*g*, *r*)

restart()
Discards history and resets counters

update()
Updates L-BFGS algorithm history

seisflows.plugins.optimize.LCG module

class seisflows.plugins.optimize.LCG.LCG(*path*, *load=<function loadnpy>*, *save=<function savenpy>*, *thresh=inf*, *maxiter=inf*, *precond=None*)
Bases: object

CG solver

apply_precond(*r*)

check_status(**args*, ***kwargs*)

initialize()

update(*ap*)

seisflows.plugins.optimize.NLCG module

class seisflows.plugins.optimize.NLCG.**NLCG**(*path='.'*, *load=<function loadnpy>*, *save=<function savenpy>*, *thresh=1.0*, *maxiter=inf*, *precond=None*)

Nonlinear conjugate gradient method

restart()

Restarts algorithm

seisflows.plugins.optimize.NLCG.**check_conjugacy**(*g_new*, *g_old*)

seisflows.plugins.optimize.NLCG.**check_descent**(*p_new*, *g_new*)

seisflows.plugins.optimize.NLCG.**fletcher_reeves**(*g_new*, *g_old*, *precond=<function <lambda>>*)

seisflows.plugins.optimize.NLCG.**pollak_ribere**(*g_new*, *g_old*, *precond=<function <lambda>>*)

seisflows.plugins.optimize.PLCG module

class seisflows.plugins.optimize.PLCG.**LBFGS_**(*path='.'*, *load=<function loadnpy>*, *save=<function savenpy>*, *memory=5*, *thresh=0.0*, *maxiter=inf*, *precond=None*)

Bases: *seisflows.plugins.optimize.LBFGS.LBFGS*

Adapts L-BFGS from nonlinear optimization to preconditioning

class seisflows.plugins.optimize.PLCG.**PLCG**(*path*, *eta=1.0*, ***kwargs*)

Bases: *seisflows.plugins.optimize.LCG.LCG*

Preconditioned truncated-Newton CG solver

Adds preconditioning and adaptive stopping to LCG base class

apply_precond(*r*)

check_status(*ap*, *verbose=True*)

Checks Eisenstat-Walker termination status

5.7.3 seisflows.plugins.preconds package

seisflows.plugins.preconds.diagonal module

class seisflows.plugins.preconds.diagonal.**Diagonal**

Bases: `object`

User supplied diagonal preconditioner

Rescales model parameters based on user supplied weights

5.7.4 seisflows.plugins.solver package

seisflows.plugins.solver.specfem2d module

seisflows.plugins.solver.specfem2d.**smooth_legacy**(*input_path=''*, *output_path=''*, *parameters=[]*, *span=0.0*)

seisflows.plugins.solver.specfem2d.**write_receivers**(*coords*, *path='.'*)
Writes receiver information to text file

seisflows.plugins.solver.specfem2d.**write_sources**(*coords*, *path='.'*, *ws=1.0*, *suffix=''*)
Writes source information to text file TODO this has to be adapted for new versions of specfem because the source file format has changed

seisflows.plugins.solver.specfem3d module

seisflows.plugins.solver.specfem3d.**write_receivers**(*h*)
Writes receiver information to text file

seisflows.plugins.solver.specfem3d.**write_sources**(*PAR*, *h*, *path='.'*)
Writes source information to text file

seisflows.plugins.solver.specfem3d_globe module

seisflows.plugins.solver.specfem3d_globe.**write_parameters**(*par*, *version*)
Writes parameters to text file

seisflows.plugins.solver.specfem3d_globe.**write_receivers**(*h*)
Writes receiver information to text file

seisflows.plugins.solver.specfem3d_globe.**write_sources**(*PAR*, *h*, *path='.'*)
Writes source information to text file

5.7.5 seisflows.plugins.solver_io package

seisflows.plugins.solver_io.adios module

seisflows.plugins.solver_io.adios.**mread**(*path*, *parameters*, *iproc*, *prefix=''*, *suffix=''*)
Multiparameter read, callable by a single mpi process

seisflows.plugins.solver_io.adios.**read**(*path*, *parameter*, *iproc*)
Reads from ADIOS container

seisflows.plugins.solver_io.adios.**write**(*v*, *path*, *parameter*, *iproc*)
Writes to ADIOS container

seisflows.plugins.solver_io.fortran_binary module

seisflows.plugins.solver_io.fortran_binary.**copy_slice**(*src*, *dst*, *iproc*, *parameter*)
Copies SPECFEM model slice

seisflows.plugins.solver_io.fortran_binary.**read_slice**(*path*, *parameters*, *iproc*)
Reads SPECFEM model slice(s) Such as, for example : proc000005_vp.bin In that specific case it would be : read_slice(path, ‘vp’, 5)

seisflows.plugins.solver_io.fortran_binary.**write_slice**(*data*, *path*, *parameters*, *iproc*)
Writes SPECFEM model slice

5.7.6 seisflows.plugins.adjoint module

seisflows.plugins.adjoint.**Acceleration**(*syn*, *obs*, *nt*, *dt*)

seisflows.plugins.adjoint.**Amplitude**(*syn*, *obs*, *nt*, *dt*)
Cross correlation amplitude

seisflows.plugins.adjoint.**Displacement**(*syn*, *obs*, *nt*, *dt*)

seisflows.plugins.adjoint.**Envelope**(*syn*, *obs*, *nt*, *dt*, *eps=0.05*)
Envelope difference (Yuan et al 2015, eq 16)

seisflows.plugins.adjoint.**Envelope2**(*syn*, *obs*, *nt*, *dt*, *eps=0.0*)
Envelope amplitude ratio (Yuan et al 2015, eq B-2, B-3)

seisflows.plugins.adjoint.**Envelope3**(*syn*, *obs*, *nt*, *dt*, *eps=0.0*)
Envelope cross-correlation lag (Yuan et al 2015, eqs B-2, B-5)

seisflows.plugins.adjoint.**InstantaneousPhase**(*syn*, *obs*, *nt*, *dt*, *eps=0.05*)
Instantaneous phase (from Bozdag et al. 2011, eq 27)

seisflows.plugins.adjoint.**InstantaneousPhase2**(*syn*, *obs*, *nt*, *dt*, *eps=0.0*)

seisflows.plugins.adjoint.**Traveltime**(*syn*, *obs*, *nt*, *dt*)
Cross correlation traveltime (Tromp et al 2005, eq 45)

seisflows.plugins.adjoint.**TraveltimeInexact**(*syn*, *obs*, *nt*, *dt*)
Much faster (but possibly inaccurate) version of Traveltime function

seisflows.plugins.adjoint.**Velocity**(*syn*, *obs*, *nt*, *dt*)

seisflows.plugins.adjoint.**Waveform**(*syn*, *obs*, *nt*, *dt*)
Waveform difference (Tromp et al 2005, eq 9)

5.7.7 seisflows.plugins.misfit module

seisflows.plugins.misfit.**Acceleration**(*syn*, *obs*, *nt*, *dt*)

seisflows.plugins.misfit.**Amplitude**(*syn*, *obs*, *nt*, *dt*)
Cross correlation amplitude

seisflows.plugins.misfit.**Displacement**(*syn*, *obs*, *nt*, *dt*)

seisflows.plugins.misfit.**Envelope**(*syn*, *obs*, *nt*, *dt*, *eps=0.05*)
Envelope difference (Yuan et al 2015, eq 9)

seisflows.plugins.misfit.**Envelope2**(*syn*, *obs*, *nt*, *dt*, *eps=0.0*)
Envelope amplitude ratio (Yuan et al 2015, eq B-1)

seisflows.plugins.misfit.**Envelope3**(*syn*, *obs*, *nt*, *dt*, *eps=0.0*)
Envelope cross-correlation lag (Yuan et al 2015, eqs B-4)

seisflows.plugins.misfit.**InstantaneousPhase**(*syn*, *obs*, *nt*, *dt*, *eps=0.05*)
Instantaneous phase from Bozdag et al. 2011

seisflows.plugins.misfit.**InstantaneousPhase2**(*syn*, *obs*, *nt*, *dt*, *eps=0.0*)

seisflows.plugins.misfit.**Traveltime**(*syn*, *obs*, *nt*, *dt*)
Compute cross correlation traveltime between two traces suposing that they contain only one arrival

seisflows.plugins.misfit.**TraveltimeInexact**(*syn*, *obs*, *nt*, *dt*)
Much faster (but possibly inaccurate) version of Traveltime function

seisflows.plugins.misfit.**Velocity**(*syn*, *obs*, *nt*, *dt*)

seisflows.plugins.misfit.**Waveform**(*syn*, *obs*, *nt*, *dt*)
Waveform difference

5.7.8 seisflows.plugins.readers module

seisflows.plugins.readers.**ascii**(*path*, *filenames*)
Reads SPECFEM3D-style ascii data

seisflows.plugins.readers.**readBigSuFile**(*nameOfFile*, *nt*, *format='SU'*, *byteorder='<'*)
This function is a hack to read .su file containing too many samples per traces. In the su format only 2 bytes per trace are dedicated to encoding for the number of samples (as signed int, see: http://lists.swapbytes.de/archives/obspy-users/2017-March/002359.html). Even if it's an old format it's still extremely stupid. This prove the lack of vision the designer of this format had at that time. They could have chosen 8 bytes or 16 bytes it was no big deal…They've cost me a day's work. But let us forget about the past. This limits the size of the traces to 32768 samples (NSTEP beween -32768 to 32768). Let us now suppose that we have NSTEP = 80000 samples per trace. We still want to use Obspy. The problem is that the NSTEP written in the .su file does not make any sense anymore and it is read by the obspy.read function! We thus rewrote a quick version of this function replacing the number of point by PAR.NT It is mainly copy-pasted from Obspy source code.

seisflows.plugins.readers.**su**(*path*, *filename*)
Reads Seismic Unix files. Hardwired °

Function readBigSuFile is a hack to read su file containing too many samples per trace. In the su format only 2 bytes per trace are dedicated to encoding for the number of samples (as signed int, see: http://lists.swapbytes.de/archives/obspy-users/2017-March/002359.html). Even if it's an old format it's still extremely stupid. This proove the lack of vision the designer of this format had at that time. They could have chosen 8 bytes or 16 bytes it was no big deal…They've cost me a day's work. But let us forget about the past. This limits the size of the traces to 32768 samples (NSTEP beween -32768 to 32768). Let us now suppose that we have NSTEP = 40000 samples per trace. We still want to use Obspy. The problem is that the NSTEP written in the .su file does not make any sense anymore and it is read by the obspy.read function! We thus rewrote a quick version of this function from Obspy replacing the number of point by PAR.NT

5.7.9 seisflows.plugins.wavelets module

seisflows.plugins.wavelets.**gabor**(*nt*, *df*, *fp*)

seisflows.plugins.wavelets.**ricker**(*nt*, *dt*, *fp*)

5.7.10 seisflows.plugins.writers module

seisflows.plugins.writers.**ascii**(*stream*, *path*, *filenames*)
Write ascii signal file

seisflows.plugins.writers.**su**(*stream*, *path*, *filename*)
Write Seismic Unix files. Function writeBigSuFile is a hack to write a .su file when the number of samples per trace is two big. In the su format only 2 bytes per trace are dedicated to encoding for the number of samples (as signed int, see: http://lists.swapbytes.de/archives/obspy-users/2017-March/002359.html). Even if it's an old format it's still extremely stupid. This proove the lack of vision the designer of this format had at that time. They could have chosen 8 bytes or 16 bytes it was no big deal…They've cost me a day's work. But let us forget about the past. This limits the size of the traces in the header to maximum 32768. We use Obspy to write

the file with dummy values there instead of the real number of sample (that we now anyway : it is PAR.NT). We thus rewrote a quick version of this function from Obspy replacing the number of point by PAR.NT

seisflows.plugins.writers.**writeBigSuFile**(*stream*, *path*, *byteorder='<'*)
This function is a hack to write a .su file when the number of samples per trace is two big. In the su format only 2 bytes per trace are dedicated to encoding for the number of samples (as signed int, see: http://lists.swapbytes.de/archives/obspy-users/2017-March/002359.html). Even if it's an old format it's still extremely stupid. This proove the lack of vision the designer of this format had at that time. They could have chosen 8 bytes or 16 bytes it was no big deal…They've cost me a day's work. But let us forget about the past. This limits the size of the traces in the header to maximum 32768. We use Obspy to write the file with dummy values there instead of the real number of sample (that we now anyway : it is PAR.NT). We thus rewrote a quick version of this function from Obspy replacing the number of point by PAR.NT This is mostly copy-pastes from Obspy source code

5.8 seisflows.tools package

5.8.1 seisflows.tools.array module

seisflows.tools.array.**count_zeros**(*a*)
Counts number of zeros in a list or array

seisflows.tools.array.**grid2mesh**(*V*, *grid*, *mesh*)
Interpolates from structured coordinates (grid) to unstructured coordinates (mesh)

seisflows.tools.array.**gridsmooth**(*Z*, *span*)
Smooths values on 2D rectangular grid

seisflows.tools.array.**loadnpy**(*filename*)
Loads numpy binary file.

seisflows.tools.array.**mesh2grid**(*v*, *mesh*)
Interpolates from an unstructured coordinates (mesh) to a structured coordinates (grid)

seisflows.tools.array.**meshsmooth**(*v*, *mesh*, *span*)
Smooths values on 2D unstructured mesh

seisflows.tools.array.**savenpy**(*filename*, *v*)
Saves numpy binary file.

seisflows.tools.array.**sortrows**(*a*, *return_index=False*, *return_inverse=False*)
Sorts rows of numpy array

seisflows.tools.array.**stack**(**args*)

seisflows.tools.array.**uniquerows**(*a*, *sort_array=False*, *return_index=False*)
Finds unique rows of numpy array

5.8.2 seisflows.tools.err module

exception seisflows.tools.err.**ParameterError**(**args*)
Bases: exceptions.ValueError

5.8.3 seisflows.tools.graphics module

seisflows.tools.graphics.**get_regular_ticks**(*v*, *interval*)
Returns regular tick intervals.

seisflows.tools.graphics.**plot_gll**(*x*, *y*, *z*, *vmin=None*, *vmax=None*)
Plots values on 2D unstructured GLL mesh

seisflows.tools.graphics.**plot_many_gll**(*x*, *y*, *z*, *vmin=None*, *vmax=None*)
Plots values on big 2D unstructured GLL mesh (in that case tricontourf does not work)

seisflows.tools.graphics.**plot_section**(*stream*, *ax=None*, *cmap='seismic'*, *clip=100*, *title=''*, *x_interval=1.0*, *y_interval=1.0*)
Plots a seismic section from an obspy stream.

Parameters

- **stream** (*Obspy stream object*) – Obspy stream object created from a SU data file
- **ax** (*Matplotlib Axes object*) – Optional axis object
- **cmap** (*str*) – Matplotlib colormap option.
- **clip** (*float*) – Percentage value (0-100) for amplitude clipping
- **title** (*str*) – plot title
- **x_interval** (*float*) – Offset axis tick interval in km
- **y_interval** (*float*) – Time axis tick interval in km

Raises NotImplementedError – If stream object does not have SU format

seisflows.tools.graphics.**plot_vector**(*t*, *v*, *xlabel=''*, *ylabel=''*, *title=''*)
Plots a vector or time series.

Parameters

- **v** (*ndarray, ndims = 1/2*) – Vector or time series to plot
- **xlabel** (*str*) – x axis label
- **ylabel** (*str*) – y axis label
- **title** (*str*) – plot title

Raises ValueError – If dimensions of v are greater than 2

5.8.4 seisflows.tools.math module

seisflows.tools.math.**angle**(*x*, *y*)

seisflows.tools.math.**backtrack2**(*f0*, *g0*, *x1*, *f1*, *b1=0.1*, *b2=0.5*)
Safeguarded parabolic backtrack

seisflows.tools.math.**backtrack3**(*f0*, *g0*, *x1*, *f1*, *x2*, *f2*)
Safeguarded cubic backtrack

seisflows.tools.math.**dot**(*x*, *y*)

seisflows.tools.math.**gauss2**(*X*, *Y*, *mu*, *sigma*, *normalize=True*)
Evaluates Gaussian over points of X,Y

seisflows.tools.math.**grad**(*V*, *h=[]*)
Evaluates derivatives on a 2D rectangular grid

seisflows.tools.math.**hilbert**(*w*)

seisflows.tools.math.**lsq2**(*x*, *f*)
Parabolic least squares fit

seisflows.tools.math.**nabla**(*V*, *h=[]*)
Returns sum of first-order spatial derivatives of a function defined on a 2D rectangular grid; generalizes Laplacian

seisflows.tools.math.**nabla2**(*V*, *h=[]*)
Returns sum of second-order spatial derivatives of a function defined on a 2D rectangular grid; generalizes Laplacian

seisflows.tools.math.**polyfit2**(*x*, *f*)
Parabolic fit

seisflows.tools.math.**tv**(*Z*, *h=[]*, *epsilon=1e-06*)

5.8.5 seisflows.tools.msg module

5.8.6 seisflows.tools.seismic module

class seisflows.tools.seismic.**Container**
Bases: collections.defaultdict

Dictionary-like object for holding models or kernels

class seisflows.tools.seismic.**Minmax**
Bases: collections.defaultdict

Keeps track of min,max values of model or kernel

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class seisflows.tools.seismic.**Writer**(*path='./output.stat'*)
Bases: object

Utility for appending values to text files

seisflows.tools.seismic.**call_solver**(*mpiexec*, *executable*, *output='solver.log'*)
Calls MPI solver executable

A less complicated version, without error catching, would be subprocess.call(mpiexec +' '+ executable, shell=True)

seisflows.tools.seismic.**getpar**(*key*, *file='DATA/Par_file'*, *sep='='*, *cast=<type 'str'>*, *noOutput=False*)
Reads parameter from text file

seisflows.tools.seismic.**setpar**(*key*, *val*, *filename='DATA/Par_file'*, *path='.'*, *sep='='*)
Writes parameter to text file

5.8.7 seisflows.tools.signal module

seisflows.tools.signal.**correlate**(*u*, *v*)

seisflows.tools.signal.**mask**(*slope*, *const*, *offset*, *time_scheme*, *length=400*)
Constructs tapered mask that can be applied to trace to mute early or late arrivals.

seisflows.tools.signal.**mute_early_arrivals**(*traces*, *slope*, *const*, *time_scheme*, *s_coords*, *r_coords*)
Applies tapered mask to record section, muting early arrivals

Signals arriving before

SLOPE * || s - r || + CONST

are muted, where slope is has units of velocity**-1, CONST has units of time, and || s - r || is distance between source and receiver.

seisflows.tools.signal.**mute_late_arrivals**(*traces*, *slope*, *const*, *time_scheme*, *s_coords*, *r_coords*)
Applies tapered mask to record section, muting late arrivals

Signals arriving after

SLOPE * || s - r || + CONST

are muted, where SLOPE is has units of velocity**-1, CONST has units of time, and || s - r || is distance between source and receiver.

seisflows.tools.signal.**mute_long_offsets**(*traces*, *dist*, *s_coords*, *r_coords*)
Mutes traces having

|| s - r || > DIST

where || s - r || is the offset between source and receiver and DIST is a user-supplied cutoff

seisflows.tools.signal.**mute_short_offsets**(*traces*, *dist*, *s_coords*, *r_coords*)
Mutes traces having

|| s - r || < DIST

where || s - r || is the offset between source and receiver and DIST is a user-supplied cutoff

seisflows.tools.signal.**sconvolve**(*s*, *h*, *w*, *inplace=True*)

seisflows.tools.signal.**tukeywin**(*nt*, *imin*, *imax*, *alpha=0.05*)

5.8.8 seisflows.tools.tools module

class seisflows.tools.tools.**Struct**(**args*, ***kwargs*)
Bases: dict

seisflows.tools.tools.**call**(**args*, ***kwargs*)

seisflows.tools.tools.**diff**(*list1*, *list2*)
Difference between unique elements of lists

seisflows.tools.tools.**divides**(*i*, *j*)
True if j divides i

seisflows.tools.tools.**exists**(*names*)
Wrapper for os.path.exists

seisflows.tools.tools.**findpath**(*name*)
Resolves absolute path of module

seisflows.tools.tools.**getset**(*arg*)

seisflows.tools.tools.**iterable**(*arg*)

seisflows.tools.tools.**loadjson**(*filename*)
Load object using json

seisflows.tools.tools.**loadnpy**(*filename*)
Loads numpy binary file.

seisflows.tools.tools.**loadobj**(*filename*)
Load object using pickle

seisflows.tools.tools.**loadpy**(*filename*)

seisflows.tools.tools.**loadtxt**(*filename*)
Load scalar from text file

seisflows.tools.tools.**loadyaml**(*filename*)

seisflows.tools.tools.**module_exists**(*name*)

seisflows.tools.tools.**nproc**()

seisflows.tools.tools.**package_exists**(*name*)

seisflows.tools.tools.**pkgpath**(*name*)

seisflows.tools.tools.**savejson**(*filename*, *obj*)
Save object using json

seisflows.tools.tools.**savenpy**(*filename*, *v*)
Saves numpy binary file.

seisflows.tools.tools.**saveobj**(*filename*, *obj*)
Save object using pickle

seisflows.tools.tools.**savetxt**(*filename*, *v*)
Save scalar to text file

seisflows.tools.tools.**timestamp**()

5.8.9 seisflows.tools.unix module

seisflows.tools.unix.**cat**(*src*, **dst*)
Open a file and print it. If file dst is given the file is printed into dst

seisflows.tools.unix.**cd**(*path*)
Change current directory to the one given

seisflows.tools.unix.**cp**(*src=''*, *dst=''*)
Copy all (files or directories, as list or tupples) given in src to directory (or file) dst

seisflows.tools.unix.**hostname**()

seisflows.tools.unix.**ln**(*src*, *dst*)

seisflows.tools.unix.**ls**(*path*)

seisflows.tools.unix.**mkdir**(*dirs*)

seisflows.tools.unix.**mv**(*src=''*, *dst=''*)

seisflows.tools.unix.**rename**(*old*, *new*, *names*)

seisflows.tools.unix.**rm**(*path=''*)

seisflows.tools.unix.**select**(*items*, *prompt=''*)

seisflows.tools.unix.**touch**(*filename*, *times=None*)

seisflows.tools.unix.**which**(*name*)

5.9 seisflows.config module

class seisflows.config.**Dict**(*newdict*)
Bases: object

Dictionary-like object for holding parameters or paths

update(*newdict*)

class seisflows.config.**Null**(**args*, ***kwargs*)
Bases: object

Always and reliably does nothing

seisflows.config.**config**()
Instantiates SeisFlows objects and makes them globally accessible by registering them in sys.modules

seisflows.config.**custom_import**(**args*)
Imports SeisFlows module and extracts class of same name. For example,

custom_import('workflow', 'inversion')

imports 'seisflows.workflow.inversion' and, from this module, extracts class 'inversion'.

seisflows.config.**load**(*path*)
Imports session from disk

seisflows.config.**save**()
Exports session to disk Write files: seisflows_parameters.json, seisflows_paths.json, seisflows_system.p, seisflows_preprocess.p, seisflows_solver.p, seisflows_postprocess.p, seisflows_optimize.p, seisflows_workflow.p

seisflows.config.**tilde_expand**(*mydict*)
Expands tilde character in path strings

PYTHON MODULE INDEX

S

INDEX

D

E

F

G

H

I

J

K

L

M

N

T

U

V

W