Seismic Toolbox

Software Design Document

Parallel Programming Team

September 5, 2021

Contents

Chapter 1

Introduction

1.1 Purpose

This software design document describes the architecture and system design for the Seismic Toolbox repository 1.
Its target audience are the current and possible developers of this Software.

1.2 Scope

Seismic Toolbox contains all different seismology algorithms (Reverse time migration (RTM) currently). Algorithms are computationally intensive processes which requires propagating wave in 2D model using time domain finite differences wave equation solvers.

During the imaging process a forward-propagated source wave field is combined at regular time steps with a back-propagated receiver wave field. Traditionally, synchronization of both wave fields result in a very large volume of I/O, disrupting the efficiency of typical supercomputers. Moreover, the wave equation solvers are memory bandwidth bound due to low flop-per-byte ratio and non-contiguous memory access, resulting hence in a low utilization of available computing resources.

Alternatively, approaches to reduce the IO bottleneck or remove it completely to fully utilize the processing power are usually explored and utilized such as the use of compression to reduce the I/O volume.Another approach that eliminates the need for I/O would be to add another propagation in reverse-time to the forward propagated source wave field.

1.3 Reference Material

The source for all information in this document is the SeismicToolbox repository (1).

1.4 Definitions and Acronyms

This section provides definitions of all terms, acronyms, and abbreviations that might exist to properly interpret the SDD.

Term	Definition
Software Design Document (SDD)	Used as the primary medium for communicating software design information.
Reverse time migration (RTM)	A seismic imaging method to map the subsurface reflectively using recorded seismic waveforms.
Full waveform inversion (FWI)	A seismic imaging technique for extracting physical parameters of the medium using seismic waves
Pre-Stack Time Migration (PSTM)	A robust model building workflow.
Pre-Stack Depth Migration (PSDM)	A robust model building workflow.
ZFP compression (ZFP)	A compressor for integer and floating-point data stored in multidimensional arrays.
Open Multi-Processing (OpenMp)	API supports multi-platform shared-memory parallel programming.
Intel oneAPI (oneAPI)	A cross-industry, open, standards-based unified programming model that delivers a common developer experience across accelerator architecture for faster application performance, more productivity, and greater innovation.
Compute Unified Device Architecture (CUDA)	A parallel computing platform and application programming interface (API) model created by Nvidia.
Message Passing Interface standard (MPI)	API standardized and portable message-passing standard designed to function on parallel computing architectures.
JavaScript Object Notation (JSON)	A lightweight data-interchange format that easy for humans to read and write aslo for machines to parse and generate.
Commas Separated Values (CSV)	CSV file is a delimited text file that uses a comma to separate values.
Convectional Perfectly Matched Layer (CPML)	Artificial boundary condition 8.5.

Chapter 2

System Overview

This chapter gives a general description of the different functionalities of our project.

1. Support the following boundary conditions 8.5:
   - CPML
   - Sponge
   - Random
2. Support the following stencil 8.16 orders:
   - O(2)–> half length =1
   - O(4)–> half length =2
   - O(8)–> half length =4
   - O(12)–> half length =6
   - O(16)–> half length =8
3. Support 2D modeling and imaging.
4. Support the following algorithmic approaches:
   - Two propagation 8.6.1.
   - Two propagation with ZFP compression 8.6.2.
   - Three propagation 8.6.3
5. Support solving the equation system in:
   - Second Order 8.8.1
   - Staggered First Order 8.8.2
   - Vertical Transverse Isotropic (VTI) 8.9.1
   - Tilted Transverse Isotropic (TTI) 8.9.2
6. Support manual cache blocking 8.17.

Chapter 3

System Architecture

3.1 Architectural Design

This section contains a modular program structure and explains the relationships between the modules to achieve the complete functionality of the system.

This is a high level overview of how responsibilities of the system were partitioned and then assigned to subsystems.

The main purpose of this diagram is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together.

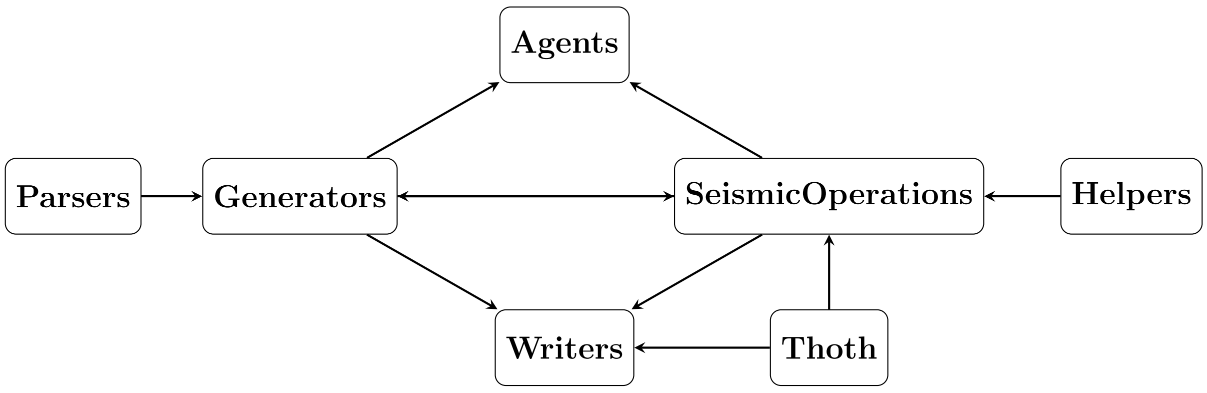


Figure 3.1: Architectural Design

The **Parsers** module is responsible for parsing the json files and make a json map, which will be used by the **Generators** module to generate the **Writers** and **Agents** modules and different components of the **SeismicOperations** library.

The **SeismicOperations** library is used by the **Agents**, **Generators** and **Writers** modules and is using the different functionalities of the **Helpers** and **Thoth** I/O library.

The **Thoth** I/O library is also used by the **Writers** module.

3.2 Decomposition Description

Provide a decomposition of the subsystems in the architectural design.

We are using object oriented description(OO).

The Decomposition Description of each module is in the following chapters.

Chapter 4

Main modules

This chapter includes the Decomposition Description of the different modules described in Figure 3.1 which are **Parsers**, **Generators**, **Agents** and **Writers**.

4.1 Parsers

The parsers module is responsible for parsing files.
It mainly consists of 2 submodules.

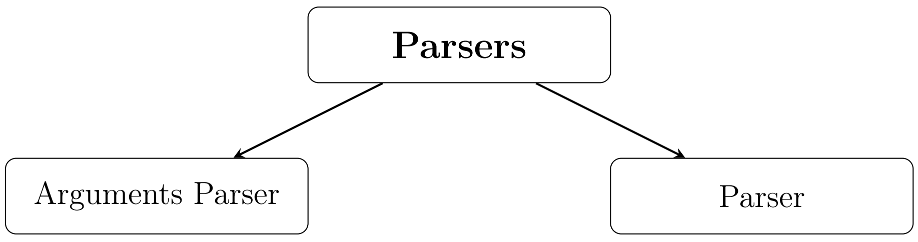


Figure 4.1: Parsers module

4.1.1 ArgumentsParser

This submodule is responsible for parsing the command line arguments to the appropriate targets.

4.1.2 Parser

This submodule is responsible for registering either the json files to be parsed or a whole folder to parse all json files inside.

4.2 Generators

This module is responsible for generating the different objects of the SeismicToolbox.
It has the following types:

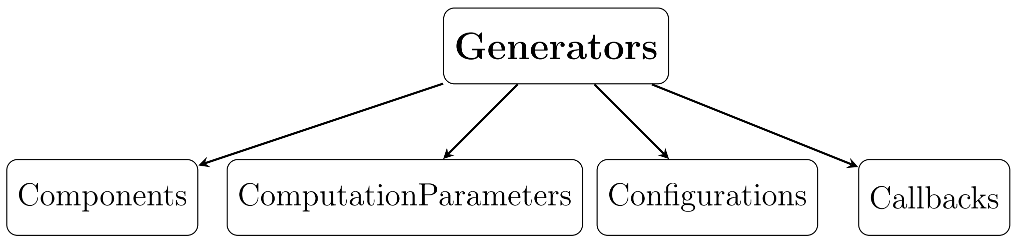


Figure 4.2: Generators module

4.2.1 Components Generator

This submodule is responsible for generating the different components of the Seismic Operation module 5.6.

4.2.2 Computation Parameters Generator

This submodule is responsible for generating the Computation Parameters that will be used like boundary length, source frequency, stencil-order, window parameters and cache blocking parameters.
It is also responsible for generating the computation parameter object 5.5.1.

4.2.3 Configurations Generator

This submodule is responsible for generating the different configurations to be used in the seismology algorithm like wave physics and approximation, equation order, grid sampling, model and traces files and different traces configurations.

4.2.4 Callbacks Generator

This submodule is responsible for generating the callback collection component of the SeismicOperations library 5.4.1 .

4.3 Agents

This module is responsible to preform RTMEngine 5.3.3 tasks according to the specifies running approach to be used, either it is with or without MPI shot distribution 8.7.
It cab be of the following types:

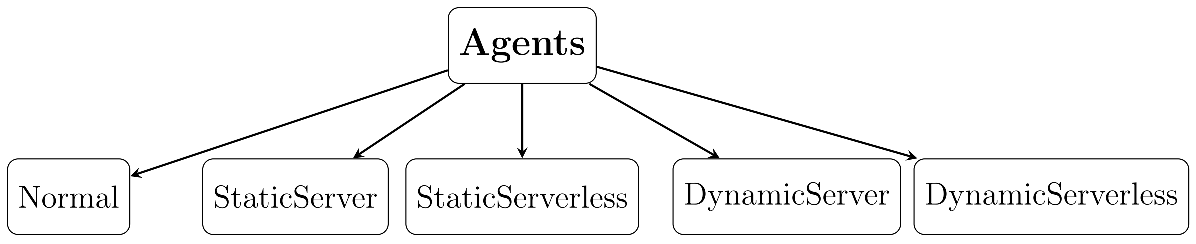


Figure 4.3: Agents module

4.3.1 Normal Agent

This is the normal agent without using MPI shot distribution.

4.3.2 StaticServer Agent

This is agent using MPI shot distribution with StaticServer approach 8.7.1

4.3.3 StaticServerless Agent

This is agent using MPI shot distribution with StaticServerless approach 8.7.1

4.3.4 DynamicServer Agent

This is agent using MPI shot distribution with DynamicServer approach 8.7.2

4.3.5 DynamicServerless Agent

This is agent using MPI shot distribution with DynamicServerless approach 8.7.2

4.4 Writers

This module is responsible for writing various types of formats for a given migration data.
It can be of the following types:

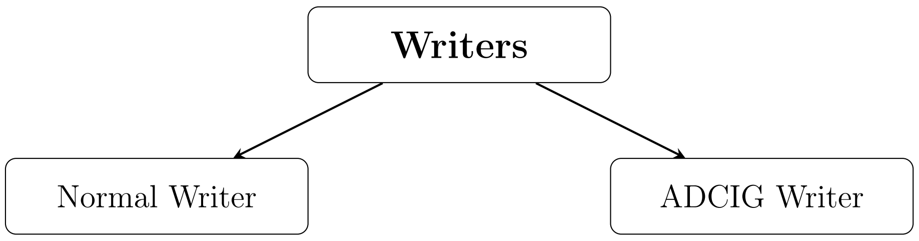


Figure 4.4: Writers module

4.4.1 Normal writer

This submodule is responsible for writing of results if the ADCIG imaging condition approach 8.13.2 is not used.

4.4.2 ADCIG writer

This submodule is responsible for writing of results if the ADCIG imaging condition approach 8.13.2 is used.

Chapter 5

SeismicOperations library

This chapter includes the Decomposition Description of the Seismic Operations library which provides interfaces, null concrete implementations and documentation of the interfaces and API of the different seismic components.

It also provides the implementation of the engine that will use the components for the full algorithm cycle.

This should bee used as a base for any engine oriented variation.

It consists of different modules as follows:

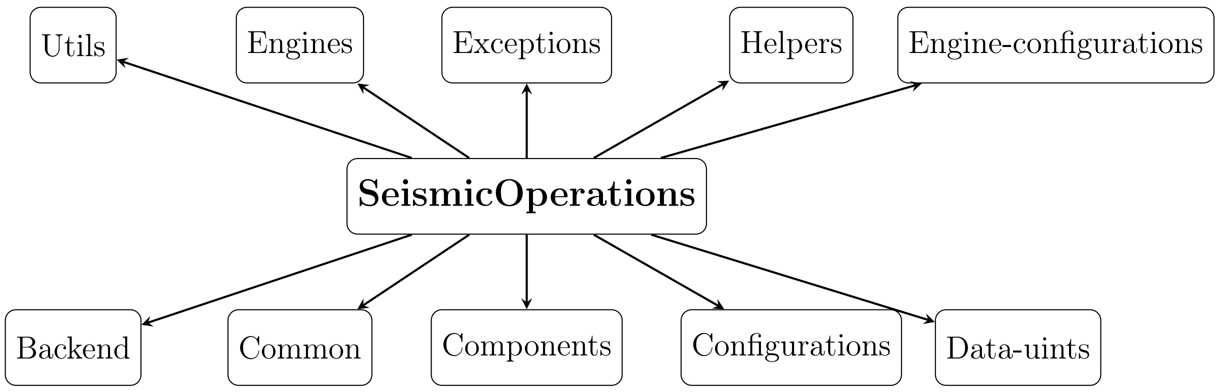


Figure 5.1: SeismicOperations library

5.1 Utils

This module contains different utilization components that can be used all over the SeismicToolbox.

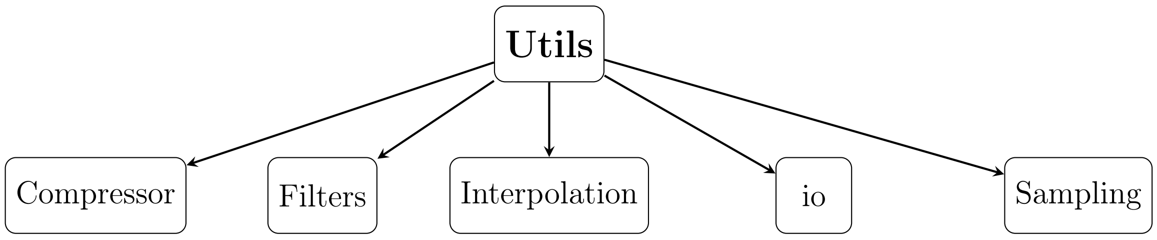


Figure 5.2: Utils module

5.1.1 Compressor

This submodule is the main entry point for all compression and decompression algorithms.

5.1.2 Filters

This submodule is responsible for noise filtering of the results.

5.1.3 Interpolation

This submodule is responsible for traces 8.1.1 interpolation.

5.1.4 io

This submodule includes utilization components for the reading and writing operations.

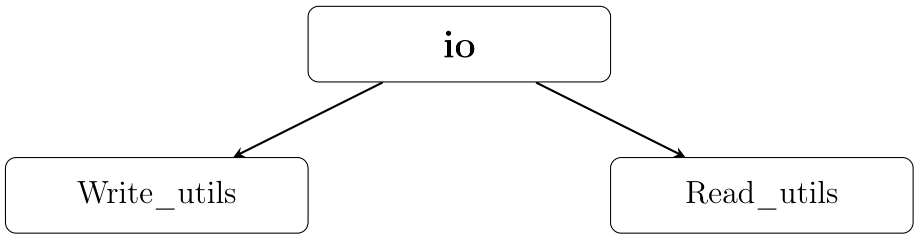


Figure 5.3: io submodule

Write_utils:

This is a write utility that is responsible for transforming an array of data into a gather of traces 8.1.1 for IO.

Read_utils:

This is a read utility that is responsible for parsing gathers into traces 8.1.1.

5.1.5 Sampling

This submodule is responsible for Grid sampling and resizing.

5.2 Exceptions

This module contains handling of different exceptions.
We are handling the coming exceptions:

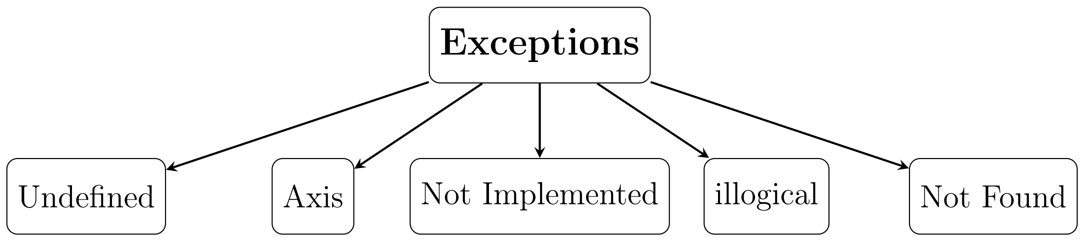


Figure 5.4: Exceptions module

5.2.1 Undefined Exception

This submodule is responsible for handling the undefined features.

5.2.2 Axis Exception

This submodule is responsible for handling the axis that is out of the defined dimensions for example Checks if the provided axe is within the x,y,z range.

5.2.3 Not Implemented Exception

This submodule is responsible for handling the unimplemented functionalities.

5.2.4 Illogical Exception

This submodule is responsible for handling the illogical values.

5.2.5 Not Found Exception

This submodule is responsible for handling the not found keys.

5.3 Engines

This module contains the concrete implementations of using the different components to apply the desired framework engine.
Framework engine can be of the following types:

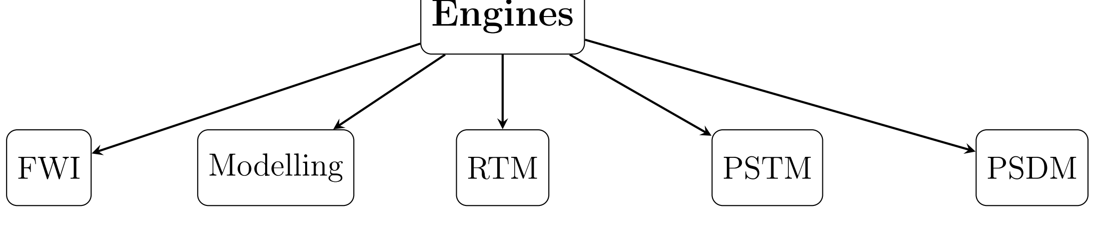


Figure 5.5: Engine module

5.3.1 FWI Engine

It is responsible of using the different components to apply the full-wave inversion (FWI).
Note: This approach is not implemented yet.

5.3.2 Modelling Engine

It is responsible of using the different components to model the needed algorithm.

5.3.3 RTM Engine

It is responsible of using the different components to apply the reverse time migration (RTM) imaging method.

5.3.4 PSTM Engine:

It is responsible of using the different components to apply the pre-stack time migration (PSTM) imaging method.
Note: This approach is not implemented yet.

5.3.5 PSDM Engine:

It is responsible of using the different components to apply the pre-stack depth migration (PSDM) imaging method.
Note: This approach is not implemented yet.

5.4 Helpers

This module includes helper components for the SeismicOperations library.
It consists of the followings:

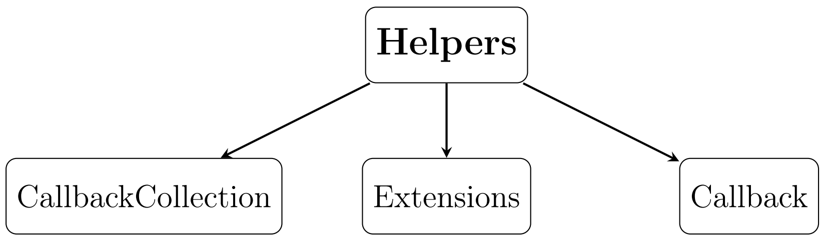


Figure 5.6: Helpers module

5.4.1 Callback collection

This submodule is responsible for registering or generating the different callbacks 5.4.3.

5.4.2 Extension

This submodule includes the different available extensions for the reading and writing formats.

5.4.3 Callback

This submodule includes all the different available callbacks.
It can be of the following types:

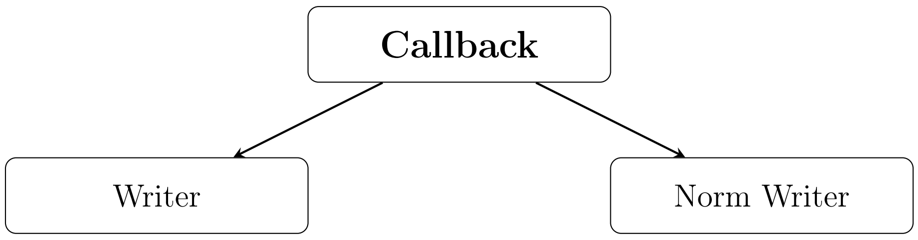


Figure 5.7: Callback submodule

Writer callback:

This callback is responsible for preparing all parameters, configurations and needed directories for writing the results in the different stages of the SeismicToolbox.

Norm Writer callback:

This callback is responsible for calculating and writing the pressure norm of the forward, backward and reversed(only in case of three propagation 8.6.3) wave fields.

5.5 Common

It is general module that includes some generic interfaces that can be used by any module in the SeismicToolbox.

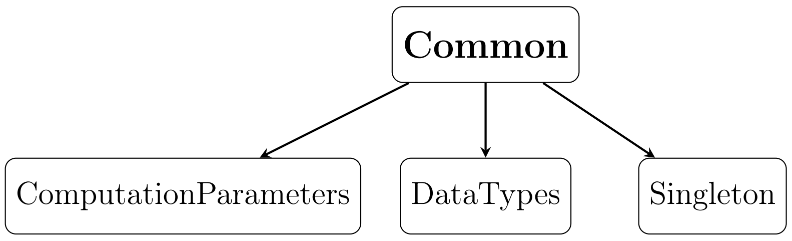


Figure 5.8: Common module

5.5.1 ComputationParameters

Includes parameters of the simulation independent from the block.

5.5.2 DataTypes

Generic customized data types for the SeismicToolbox.

5.5.3 Singleton

Template class for any singleton class to inherit from.

5.6 Components

This module includes the different components of the desired framework engine.
It consists of the following:

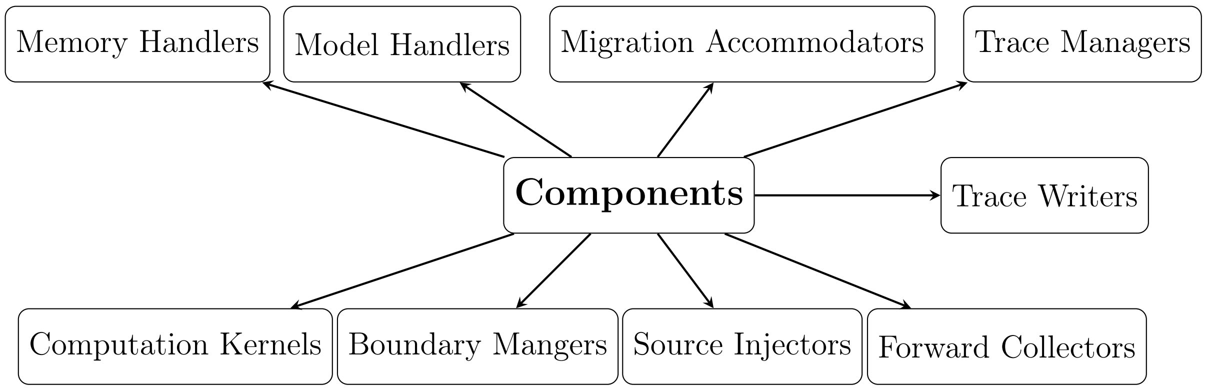


Figure 5.9: Components module

5.6.1 Memory handlers

This component is responsible for Applying first touch policy 8.12 with zero initialization for any pointer.

5.6.2 Model Handlers

All concrete techniques for loading or setting models should be implemented using this interface.
We are supporting till now the Seismic Model handler.

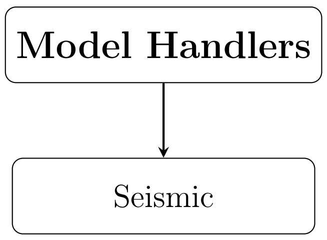


Figure 5.10: Model Handlers submodule

Seismic Model Handler:

This component is responsible for loading or setting seismic models.

5.6.3 Trace Managers

All concrete techniques for reading, pre-processing and injecting the traces 8.1.1 corresponding to a model should be implemented using this submodule.
We are supporting till now the Seismic Trace Manager.

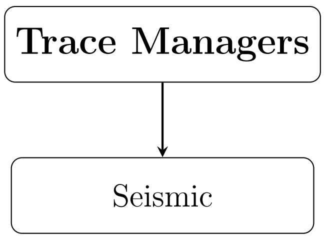


Figure 5.11: Trace Managers submodule

Seismic Trace Manager:

This component is responsible for reading, pre-processing and injecting the seismic traces 8.1.1 corresponding to the seismic model.

5.6.4 Trace Writers

This submodule is used to record at each time step the pressure at the surface in the places we want to have receivers on.
We are supporting till now the Seismic Trace Writer.

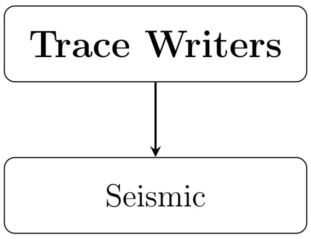


Figure 5.12: Trace Writers submodule

Seismic Trace Writer:

This component is used to record at each time step the seismic pressure at the surface in the places we want to have receivers on, it can be considered as a hydrophone.

5.6.5 Migration Accommodators

All concrete techniques for the imaging conditions techniques should be implemented using this submodule.
It consists of the following types.

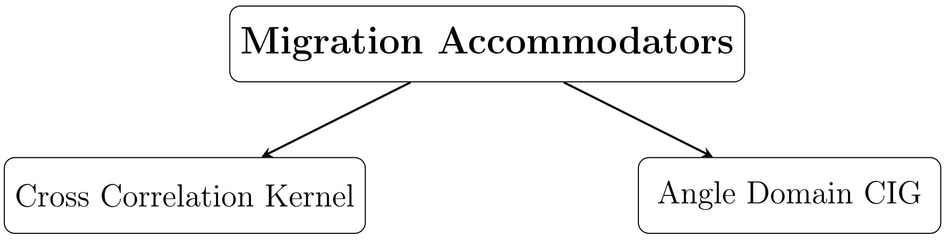


Figure 5.13: Migration Accommodators submodule

Cross Correlation Kernel:

This component is responsible for implementing the Migration Accommodator using cross correlation imaging condition 8.13.1.

Angle Domain CIG:

This component is responsible for implementing the Migration Accommodator using Angle Domain Common Image Gathering (CIG) imaging condition 8.13.2 (This is only supported for the OpenMp technology).

5.6.6 Computation Kernels

This includes all the different implementations of the computation kernels.

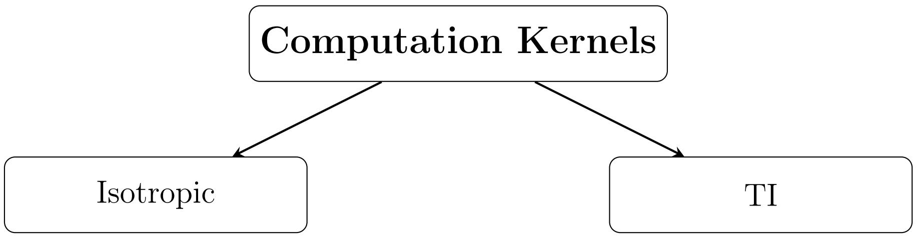


Figure 5.14: Computation Kernels submodule

Isotropic:

Consists of the following types:

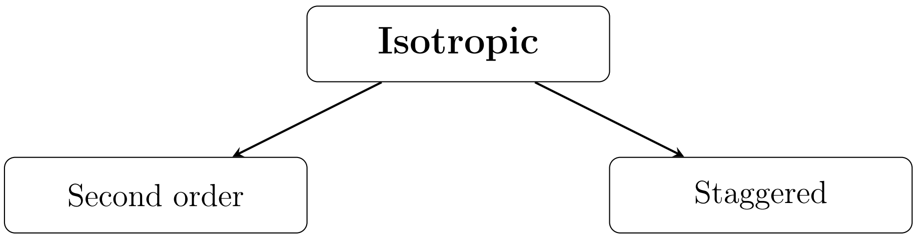


Figure 5.15: Isotropic component

Second order: the computation kernel is using second order wave equation 8.8.1.

Staggered: the computation kernel is using Staggered waveform 8.8.2.

TI:

Only available for the OpenMp technology.
It consists of the following types:

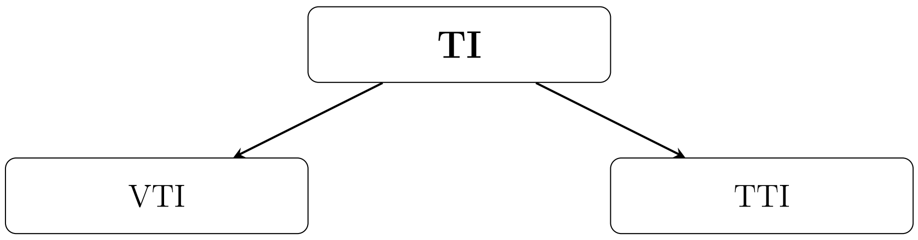


Figure 5.16: TI component

VTI: the computation kernel is using Vertical Transverse Isotropic 8.9.1.

TTI: the computation kernel is using Tilted Transverse Isotropic 8.9.2.

5.6.7 Boundary Managers:

All concrete techniques for absorbing boundary conditions 8.5 should be implemented using this interface.
It can be of the following types:

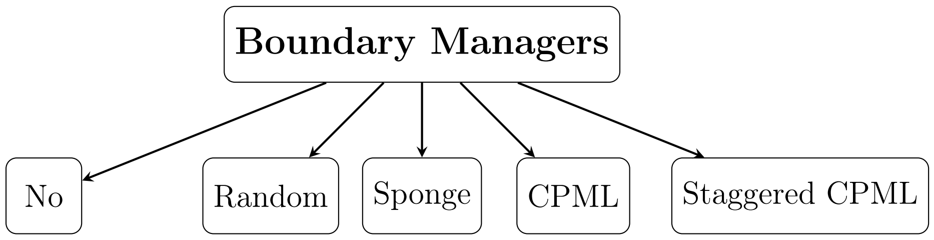


Figure 5.17: Boundary Managers submodule

No boundary manager:

Without using any absorbing boundary condition.

Random boundary manager:

Using the Random physical boundary condition 8.5.2 (this boundary condition is not supported for CUDA technology yet).

Sponge boundary manager:

Using the Sponge artificial boundary condition 8.5.1.

CPML boundary manager:

Using the CPML artificial boundary condition 8.5.1 (this boundary condition is not supported for CUDA technology yet).

Staggered CPML boundary manager:

Using the CPML artificial boundary condition 8.5.1 with the staggered waveform 8.8.2 (this boundary condition is not supported for CUDA technology yet).

5.6.8 Source Injectors

All concrete techniques for source injection should be implemented using this submodule.
Till now we are supporting Ricker source injector.

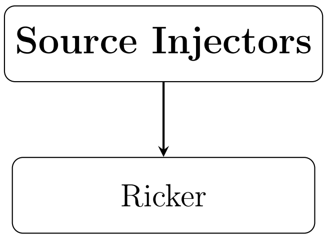


Figure 5.18: Source Injectors submodule

Ricker Source Injector:

Source injection is using the Ricker wavelet 8.11.

5.6.9 Forward Collectors

All concrete techniques for saving the forward propagation or restoring the results from it should be implemented using this submodule.
It has the following types:

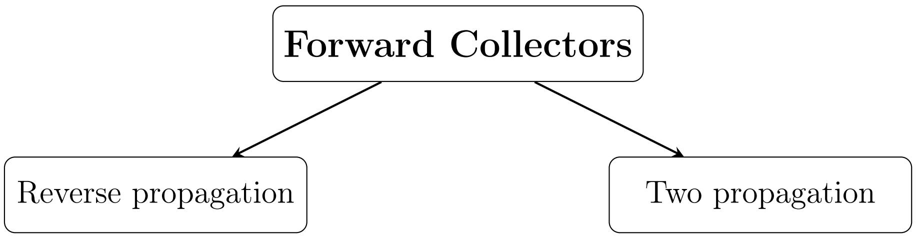


Figure 5.19: Forward Collectors submodule

Reverse propagation:

The forward collector is using the three propagation approach 8.6.3.

Two propagation:

The forward collector is using the two propagation approach 8.6.1 or two propagation with ZFP compression approach8.6.2 (according to user choice).

5.7 Engine-configurations:

This module contains pointers to concrete implementations of each component to be used in the desired framework engine.
It can be of the following types:

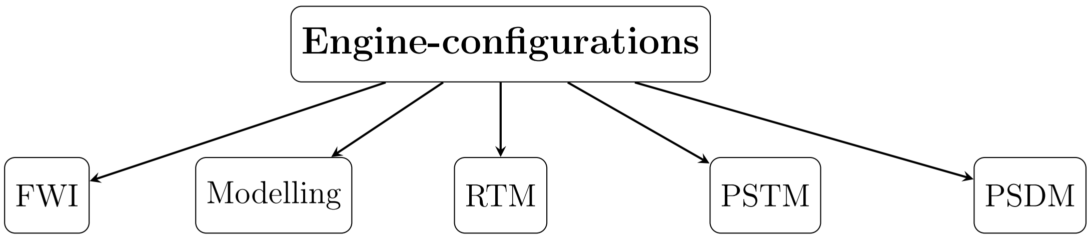


Figure 5.20: Engine-configurations module

5.7.1 FWI Engine-configurations:

It contains pointers to the concrete implementations of each component to be used in the FWI framework engine.

5.7.2 Modelling Engine-configurations:

It contains pointers to the concrete implementations of each component to be used in modelling engine.
Modelling Engine-Configuration is only used for modelling and it doesn't have a forward collector submodule 5.6.9 because we don't need to store the forward propagation.
If we are modeling then we only do forward and store the traces 8.1.1 while propagating using the Trace Writers submodule 5.6.4.
We store the traces for each shot 8.1.2 in the trace file.(each shot's traces in a different file)

5.7.3 RTM Engine-configurations:

It contains pointers to concrete implementations of each component to be used in the RTM framework engine.
In case of one shot used: take the output of the modelling engine (trace file) and the velocity file as inputs and generates the results.
In case of different shots used: take the output of the modelling engine (trace files) as a std::vector and the velocity file as inputs and generates the results.

5.7.4 PSTM Engine-configurations:

It contains pointers to the concrete implementations of each component to be used in the PSTM framework engine.

5.7.5 PSDM Engine-configurations:

It contains pointers to the concrete implementations of each component to be used in the PSDM framework engine.

5.8 Backend

This module includes the vari backends for all computations in the different technologies supported 8.19.
Till now we are only supporting OneAPI backend.

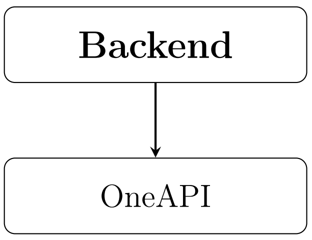


Figure 5.21: Backend module

5.8.1 OneAPI backend

This submodule includes some backend functionalities for the OneAPI technology 6 like setting the device queue and the SYCL 8.18 algorithm.

5.9 Configurations

This module includes different configuration objects.
It consists of the followings:

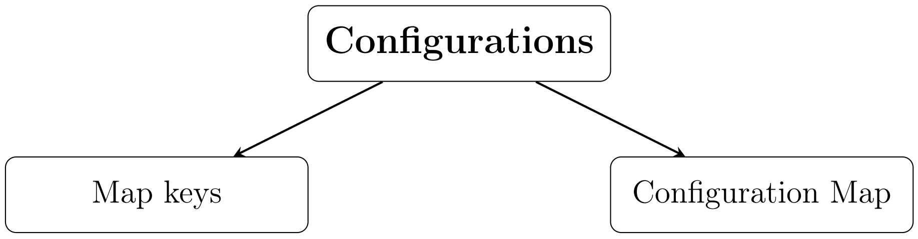


Figure 5.22: Configurations module

5.9.1 Map keys

It is a map that contains the different json parsed parameters and its associated keys in the SeismicToolbox.

5.9.2 Configuration Map

A configuration map containing a subset of properties, it is organized as subsections, with each subsection containing a list of properties.
Till now we are only supporting JSON Configuration Map.

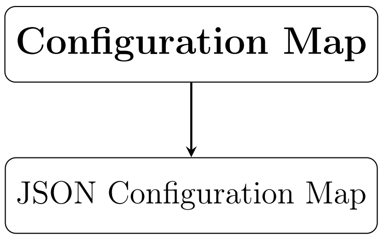


Figure 5.23: Configuration Map submodule

JSON Configuration Map:

A JSON implementation of the configuration map submodule.

5.10 Data units

This module includes some primitive objects that can be used all over the Seismic-Toolbox.
It consists of the following components.

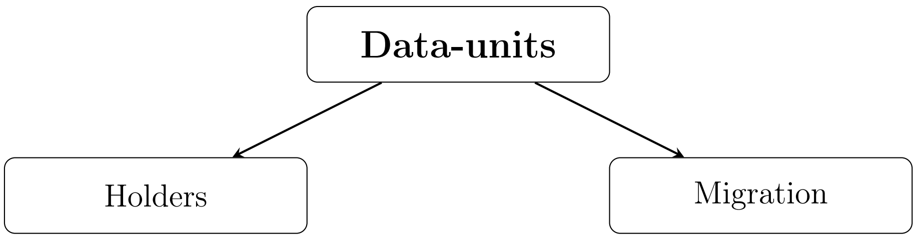


Figure 5.24: Data-units module

5.10.1 Holders

This includes holders for the basic data units.
It has the following types:

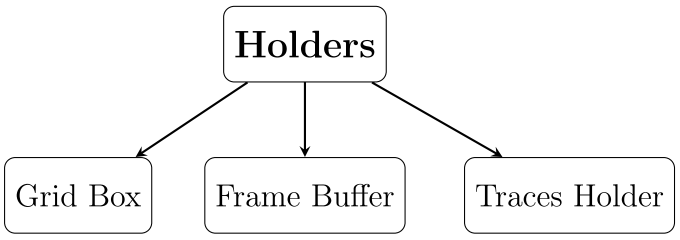


Figure 5.25: Holders submodule

Grid Box:

It holds all the meta data (i.e. dt, nt, grid size, window size, window start, reference point and cell dimensions) needed by all components.
It also holds all wave fields parameters and window parameters.

Frame Buffer:

This component is responsible for encapsulating memory accessing and memory transfers based on the technology used 8.19.

For example, CUDA technology requires cudamalloc to allocate the memory "in GPU RAM", cudamemcpy to copy data either device to host, host to device or device to device copy, and cudamemset to set the memory also, for the memory transfers between nodes and for I/O transfers.

Traces Holder:

This component contains the available traces 8.1.1 information.

5.10.2 Migration

This submodule includes basic data units for migration.

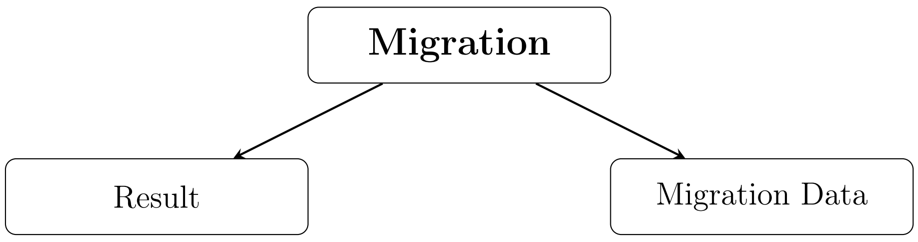


Figure 5.26: Migration submodule

Result:

Includes the different functionalities for handling results' data.

Migration Data:

It holds the data needed for migration.

Chapter 6

Helpers

The Helpers library is a bunch of utilities modules that can be applied to any project.
It consists of the following modules:

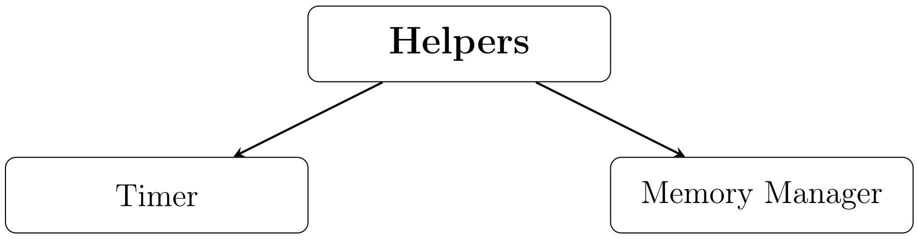


Figure 6.1: Helpers module

6.1 Timer

This module is responsible for calculating the timings and performance metrics of the different functions of the SeismicToolbox.

6.2 Memory Manager

This module is responsible for memory handling operations.
It consists of the following:

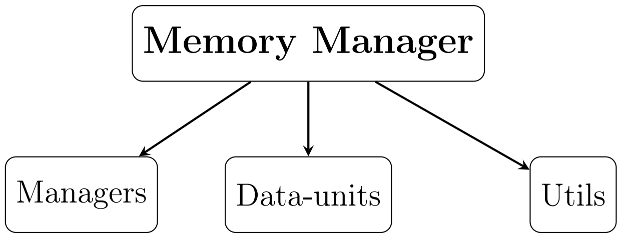


Figure 6.2: Memory Manager module

6.2.1 Managers

This submodule contains the following:

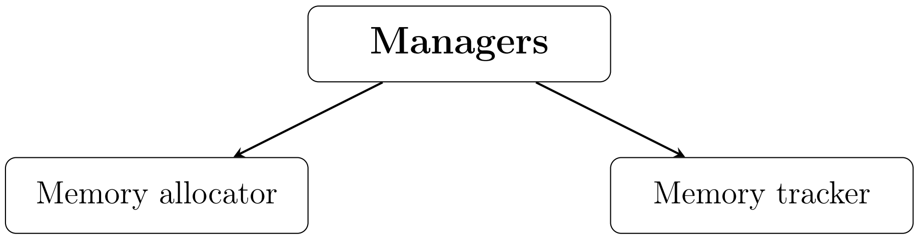


Figure 6.3: Managers submodule

Memory allocator:

This component is responsible for the aligned memory allocation and freeing.

Memory tracker:

This component is responsible for tracking memory operations.
Note: currently not used.

6.2.2 Data-units

When allocating in memory, we have a map of strings with its associated pointers to memory positions, so this submodule is responsible for handling this map.
It contains the following:

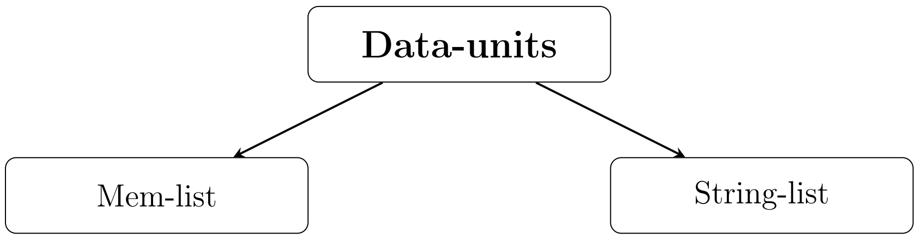


Figure 6.4: Data-units submodule

Mem-list

It is responsible for handling the pointers part of the map.

String-list

It is responsible for handling the strings part of the map.

6.2.3 Utils

This submodule includes the following:

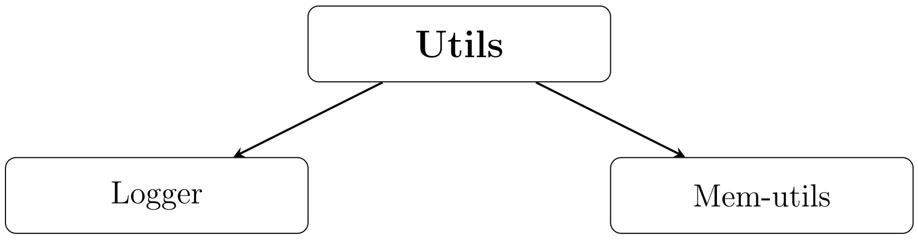


Figure 6.5: Utils submodule

Logger:

It is responsible for handling logging the information on the output file.

Mem-utils:

It includes some utilization functions for memory management.

Chapter 7

Thoth

The Thoth is a General I/O library for parsing, manipulating, creating, editing and visualizing different types of seismic file formats for developers to be used in different seismic applications.
It consists of different modules as follows:

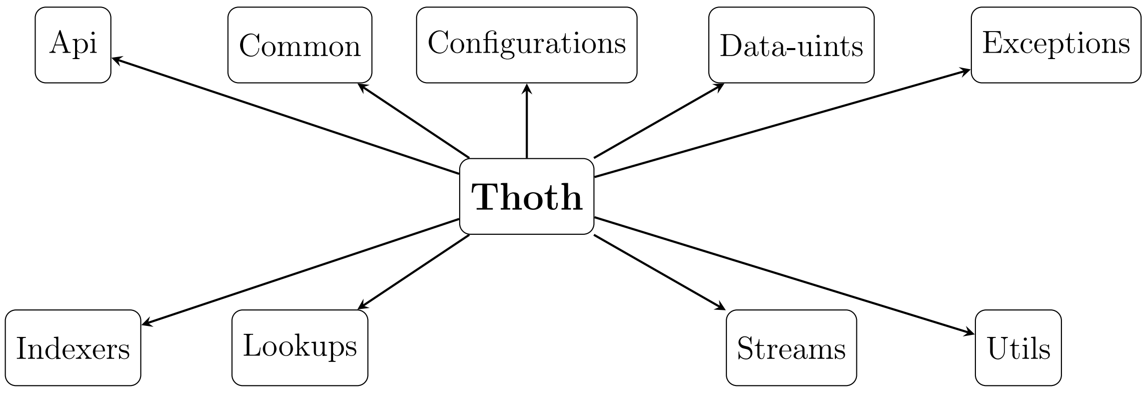


Figure 7.1: Thoth library

7.1 Api

It is considered as the Application Programming Interface (API) for the Thoth I/O library as it includes reference to tHe main header files of the library.

7.2 Common

It is general module that includes some generic interfaces that can be used by any module in the SeismicToolbox.

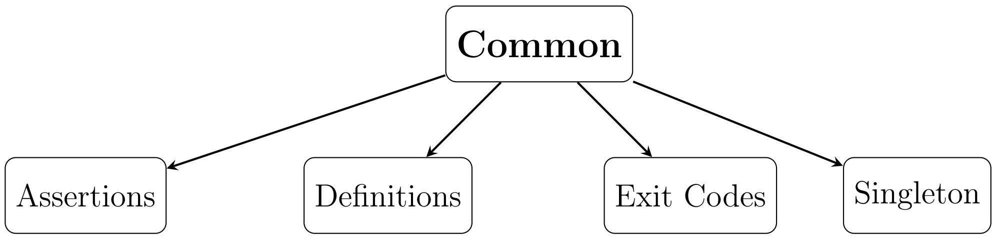


Figure 7.2: Common module

7.2.1 Assertions

It includes some Aseertions for the library which are predefined macros using which we can test certain assumptions that are set in the program.

7.2.2 Definitions

It includes Thoth I/O library common definitions like copy writes.

7.2.3 Exit Codes

It includes the exit codes for the Thoth I/O library.

7.2.4 Singleton

A Template class for any singleton class to inherit from.

7.3 Configurations

This module includes different configuration objects.
It consists of the followings:

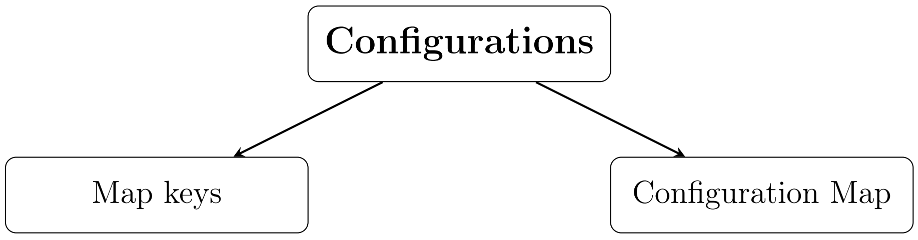


Figure 7.3: Configurations module

7.3.1 Map keys

It is a map that contains different json parsed parameters and its associated keys in the SeismicToolbox , also includes different file formats supported and their associated extensions.

7.3.2 Configuration Map

A configuration map containing a subset of properties, it is organized as subsections, with each subsection containing a list of properties.
Till now we are only supporting JSON Configuration Map.

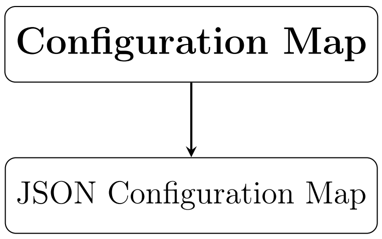


Figure 7.4: Configuration Map submodule

JSON Configuration Map:

A JSON implementation of the configuration map submodule.

7.4 Data units

This module includes some primitive objects that can be used all over the Seismic-Toolbox.
It consists of the following components.

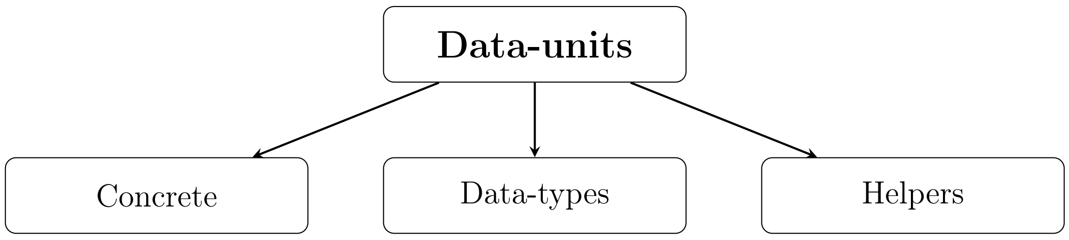


Figure 7.5: Data-units module

7.4.1 Concrete

This submodule consists of the following:

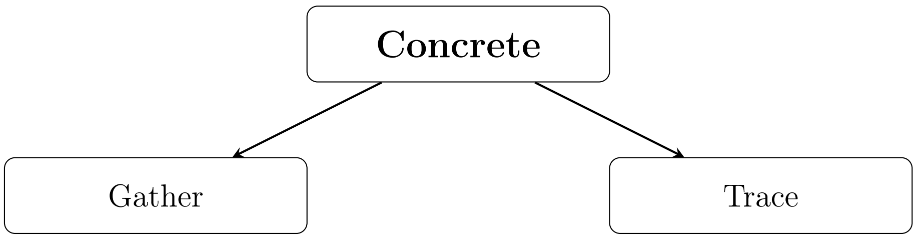


Figure 7.6: Concrete submodule

Gather:

The Gather is acollection of traces so this component is responsible for gather operations like addition and removal of a trace, getting number of traces, traces sorting and selection.

Trace

Trace Object contains Trace Header Key represented as a map and is responsible for trace operations like setting and getting of trace data, getting number of samples per trace and coordinates handling.

7.4.2 Data-types

It only includes the Trace Header Key.

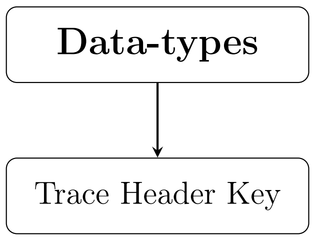


Figure 7.7: Data-types submodule

Trace Header Key:

It includes different trace header seismic keys, also it handles the different operations on them and keys mapping to their associated values in the SeismicToolbox.

7.4.3 Helpers

It only includes the Trace helper.

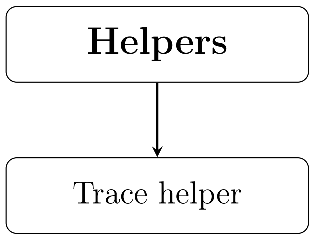


Figure 7.8: Helpers submodule

Trace helper:

Trace helper component is responsible for taking any trace data unit and helps manipulate or get any regarded meta data from it.

7.5 Exceptions

This module contains handling of different exceptions.
We are handling the coming exceptions:

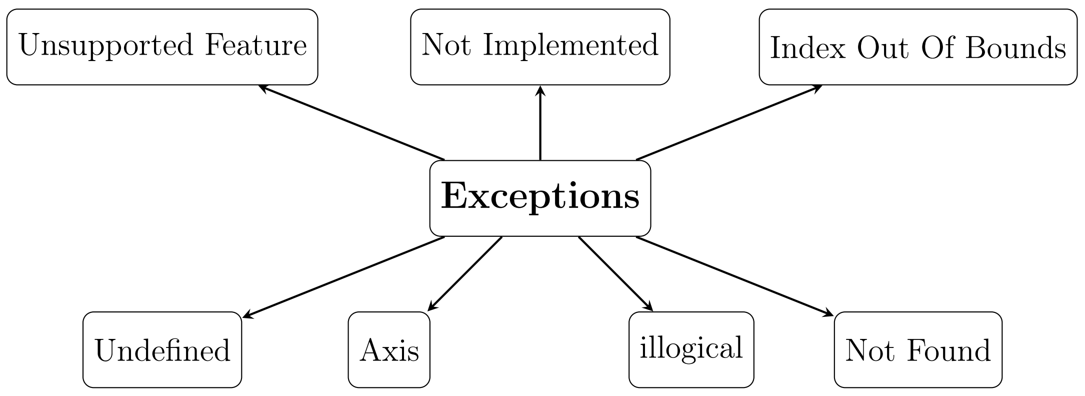


Figure 7.9: Exceptions module

7.5.1 Unsupported Feature Exception

This submodule is responsible for handling the unsupported features.

7.5.2 Not Implemented Exception

This submodule is responsible for handling the unimplemented functionalities.

7.5.3 Index Out Of Bounds Exception

This submodule is responsible for handling the out of range memory accesses.

7.5.4 Undefined Exception

This submodule is responsible for handling the undefined features.

7.5.5 Axis Exception

This submodule is responsible for handling the axis that is out of the defined dimensions for example Checks if the provided axe is within the x,y,z range.

7.5.6 Illogical Exception

This submodule is responsible for handling the illogical values.

7.5.7 Not Found Exception

This submodule is responsible for handling the not found keys.

7.6 Indexers

This module is responsible for handling indexes.
It consists of the following:

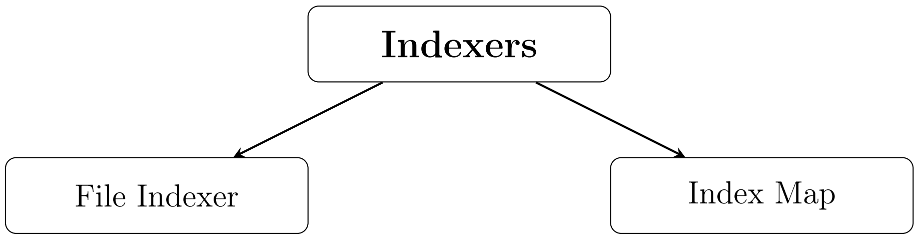


Figure 7.10: Indexers module

7.6.1 File Indexer

This submodule is reposnible for traces 8.1.1 indexing operations within a given file.

7.6.2 Index Map

The Index map submodule is a data unit for indexing purposes, after indexer does its work this submodule stores that work in index map to be used by readers then.

7.7 Lookups

This module consists of the following:

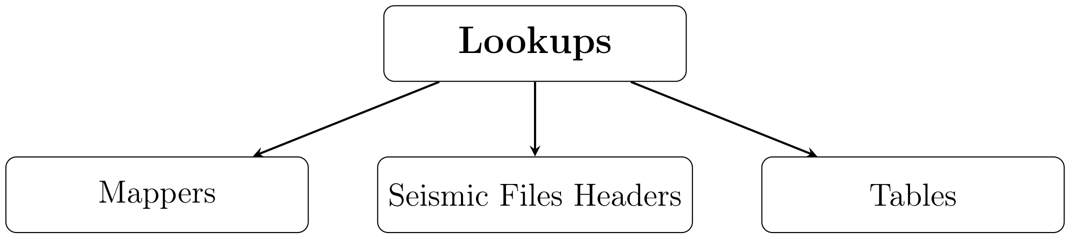


Figure 7.11: Lookups module

7.7.1 Mappers

This submodule has the following types:

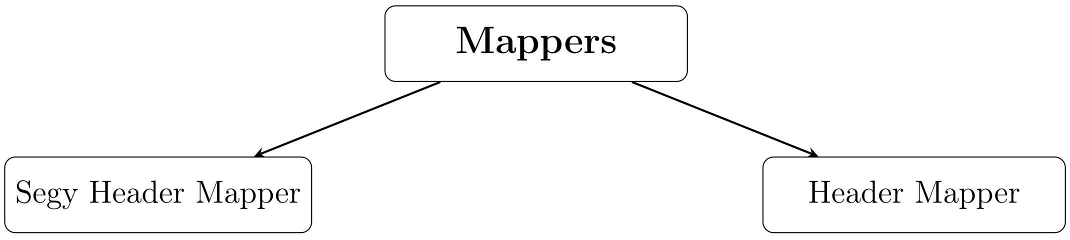


Figure 7.12: Mappers submodule

Segy Header Mapper:

it maps from Trace Header Key to their appropriate offset position in segy trace header and their type.

Header Mapper:

It is a utility class for header mapping between raw byte pointers, and the trace object 7.4.1.

7.7.2 Seismic Files Headers

This submodule contains a map of the trace header keys and their associated start position and offset length.

7.7.3 Tables

This submodule is responsible for handling the headers of SEG-Y file format 8.3. It consists of the following:

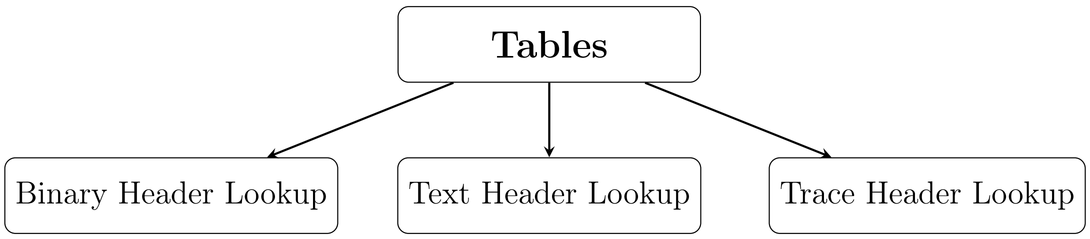


Figure 7.13: Tables submodule

Binary Header Lookup:

This component contains the size, start and end positions of the binary header.

It also contains the lookup keys of the binary header where variable types corresponds to the number of allocated byte(s) for this variable internally in the SEG-Y 8.3 or SU 8.4 file according to the general format.

All variables are stored as big endian 8.15, if the machine used is little endian 8.14, big endian to little endian conversion should take place.

Text Header Lookup:

It contains the size, start and end positions of the text header and extended text header(if available) of the SEG-Y file 8.3.

Trace Header Lookup:

This component contains the size and start positions of the trace header.

It also contains the lookup keys of the trace header where variable types corresponds to the number of allocated byte(s) for this variable internally in the SEG-Y 8.3 or SU 8.4 file according to the general format.

All variables are stored as big endian 8.15, if the machine used is little endian 8.14, big endian to little endian conversion should take place.

7.8 Streams

This modules is responsible for the different I/O streams.
It consists of the following:

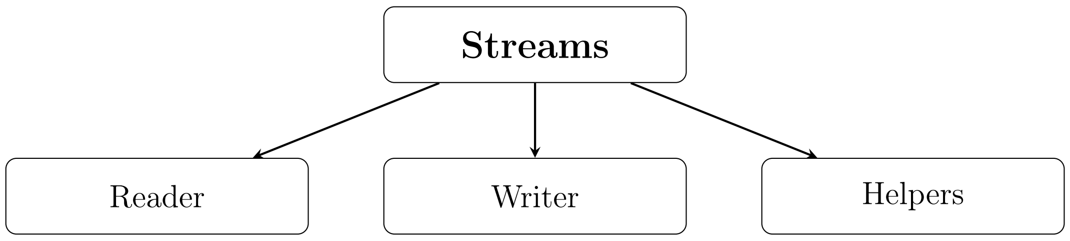


Figure 7.14: Streams module

7.8.1 Reader

This submodule is the Reader interface for seismic data.
This should be configurable to allow each reader type to extract its unique additional properties.
A simple flow should be like the following:

1. Reader.AcquireConfiguration(...);
2. Reader.Initialize(...);
3. Do all other operations afterwards.

We are supporting the following reader types:

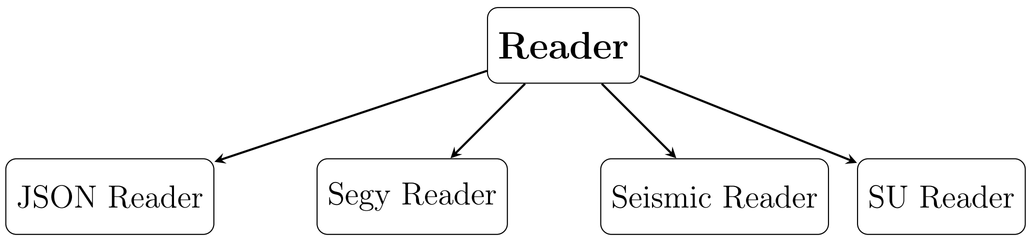


Figure 7.15: Reader submodule

JSON Reader:

It is the JSON file format reader.

Segy Reader:

It is the SEG-Y 8.3 file format reader.

Seismic Reader:

This component is a High-level wrapper for all seismic readers.

SU Reader:

It is the SU 8.4 file format reader.

7.8.2 Writer

This submodule is the writer interface for seismic data.
This should be configurable to allow each writer type to extract its unique additional properties.
A simple flow should be like the following:

1. Writer.AcquireConfiguration(...);
2. Writer.Initialize(...);
3. Do all other operations afterwards.

We are supporting following writer types:

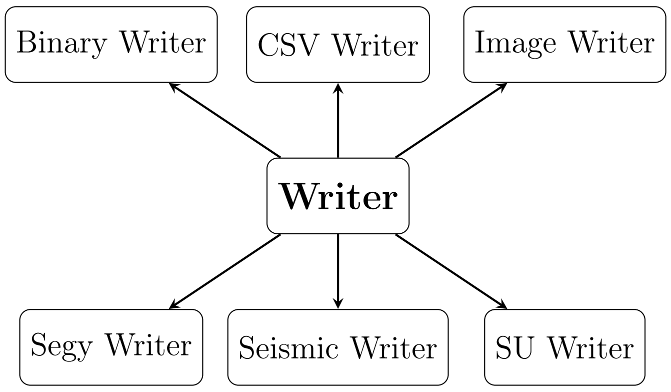


Figure 7.16: Writer submodule

Binary Writer:

It is the binary file format writer.

CSV Writer:

It is the CSV file format writer.

Image Writer:

It writes the results in png file format using the OpenCV library.

Segy Writer:

It is the segy 8.3 file format writer.

Seismic Writer:

It is a High-level wrapper for all seismic writers.

SU Writer

It is the SU 8.4 file format writer.

7.8.3 Helpers

It includes some helper objects for the I/O operations.
It consists of the following:

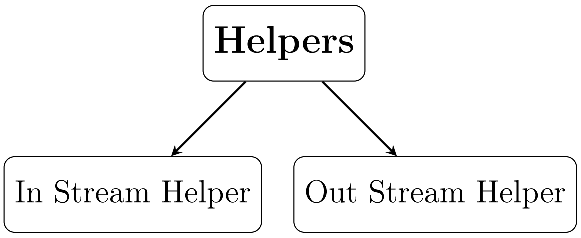


Figure 7.17: Helpers submodule

In Stream Helper:

This component is a file helper to take any stream and helps manipulate or get any regarded meta data from it.

Out Stream Helper:

This component is a file helper to take any stream and helps manipulate or write any regarded data to it.

7.9 Utils

This module contains different utilization submodules that can be used all over the SeismicToolbox.
It consists of the following:

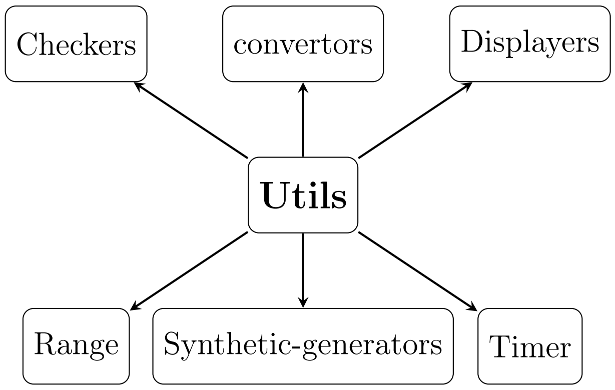


Figure 7.18: Utils module

7.9.1 Checkers

This submodule is responsible for checking if the variables are stored as big endian 8.15 or little endian 8.14.

7.9.2 Convertors

This submodule includes some type convertors.
We are supporting the following type converetors.

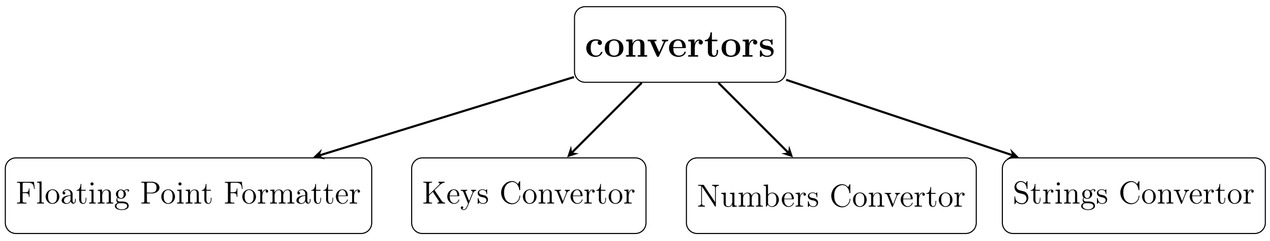


Figure 7.19: convertors submodule

Floating Point Formatter:

This component works as a convertor from any floating type representation to another.

Keys Convertor:

This component works as a convertor from string representation to Trace Header Key object 7.4.2 and vice versa.

Numbers Convertor:

This component works as a convertor from any representation type to another, like conversion from IBM to IEEE floating point format.

Strings Convertor

This component takes a string and change it to any representation type.

7.9.3 Displayers

This submodule Prints the text header extracted from the given SEG-Y 8.3 file in the SEG-Y community format.

7.9.4 Range

This submodule is responsible for ranges handing like offsetting and start positions.

7.9.5 Synthetic-generators

This submodule is responsible for the synthetic traces generation.
It consists of the following:

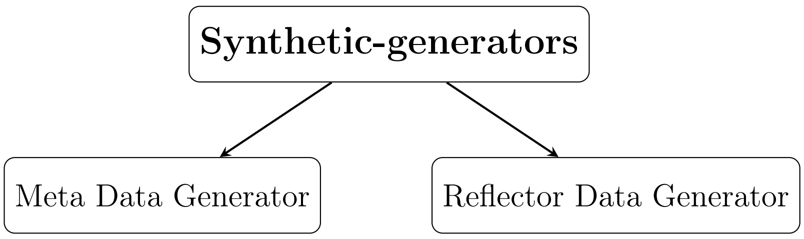


Figure 7.20: Synthetic-generators submodule

Meta Data Generator:

This component is an interface for all meta-data generators to generate traces with correct meta-data.
It consists of the following types:

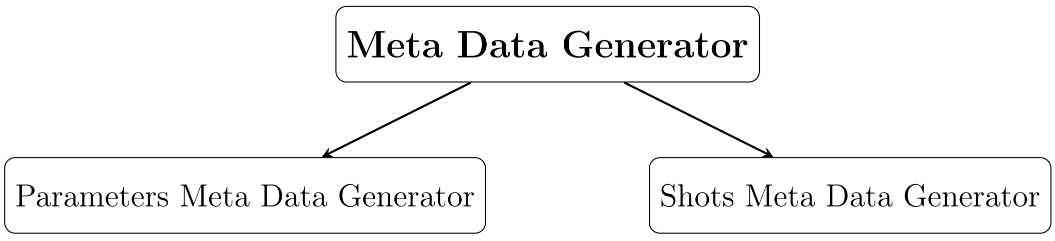


Figure 7.21: Meta Data Generator component

ParameterMetaDataGenerator and Plane Reflector Data Generator are both ParameterMetaDataGenerator for all the parameter file Plane Reflector Data Generator for layers

Parameters Meta Data Generator: It is a Meta data traces generator for all the parameter file.

Shots Meta Data Generator: It is a Meta data trace generator for the given shots.

Reflector Data Generator

This compoennet is an Interface for all the reflector data generators.
Till now we are supporting Plane Reflector Generator.

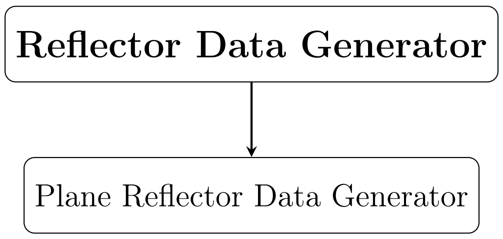


Figure 7.22: Reflector Data Generator component

Plane Reflector Data Generator It is a synthetic trace generator for layers.

7.9.6 Timer

Till now we are supporting Execution timer.

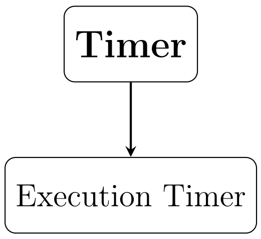


Figure 7.23: Timer submodule

Execution Timer

This component is a Function execution timer.
Note this component is currently not used.

Chapter 8

APPENDICES

This chapter provides supporting details that could aid in the understanding of the Software Design Document.

8.1 Seismic Terminologies

This sections includes some of the Seismic Terminologies that are used all over the document.

8.1.1 Trace

It is the values that are recorded by the microphone.
It can be represented as a 1 dimensional array of values.
The difference in time between each recorded sample is defined by the hardware used and called sampling interval.
Each trace might have some additional metadata eg: source location, receiver location, ...etc

8.1.2 Shot

A group of traces that are recorded with the same source location.

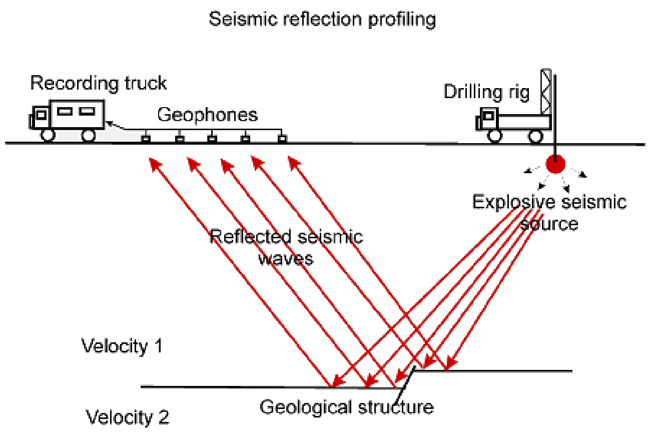


Figure 8.1: Group of traces

In another way, a shot is the result of a single explosion with many receivers.

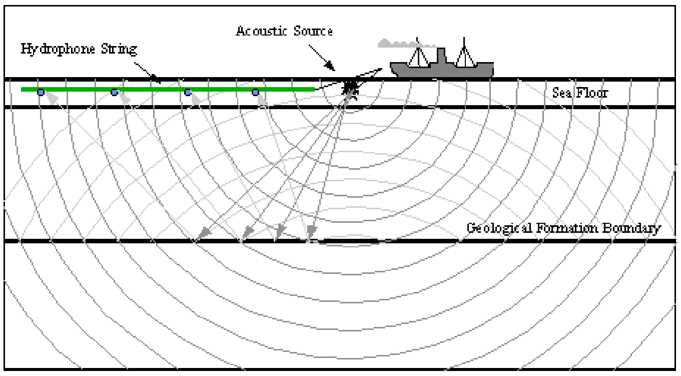


Figure 8.2: Single explosion, many receivers

8.2 Seismic Waves

It consists of P-wave and S-wave.

8.2.1 P-wave

Primary wave is the fastest wave to move in the ground.
They are pressure waves.
They depend on the different wave velocities(Vp) and densities of the ground.
They are longitudinal waves, which are waves in which the displacement of the medium is in the same direction as, or the opposite direction to, the direction of propagation of the wave.

8.2.2 S-wave

Secondary waves are the second waves in terms of speed after the primary waves.
They are shear waves.
They depend on the different wave velocities(Vs).
They are transverse wave is a moving wave whose oscillations are perpendicular to the direction of the wave.

8.3 SEG-Y Seismic Data Exchange Format

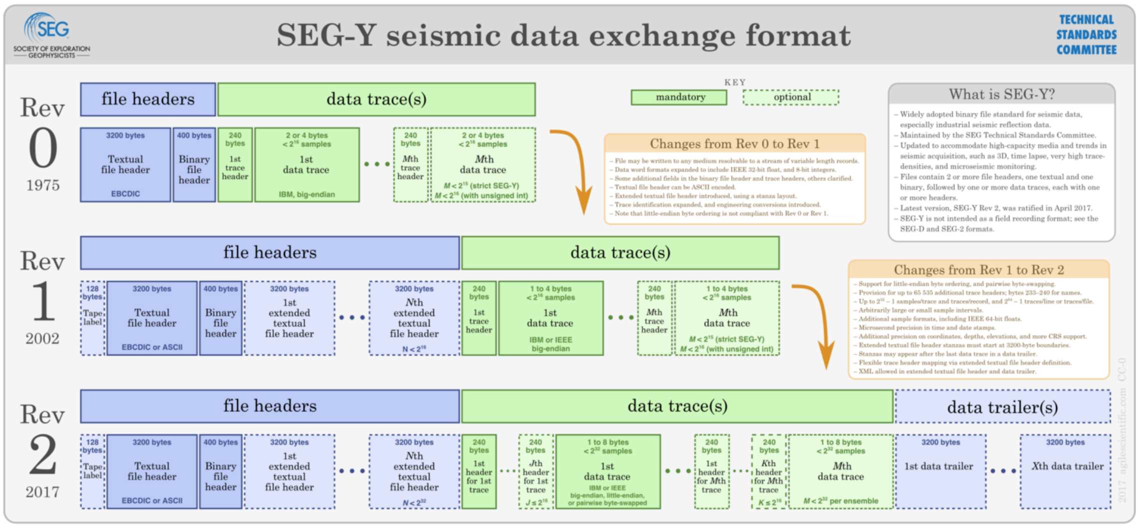
The SEG-Y (sometimes SEG Y) file format is one of several standards developed by the Society of Exploration Geophysicists (SEG) for storing geophysical data.

A SEG-Y file consists of a 3600 byte header; a number of extended textual headers; a number trace headers+data.

- A 3200 byte Textual File Header, ASCII or EBCDIC formated.
- A 400 byte Binary File Header

- A (optional) number of 'Extended Textual File Headers', 3200 bytes long, ASCII or EBCDIC formatted.
- A number of traces, separated into a 240 bytes long binary Trace Header, followed by the Trace Data, that can be formatted in a number of ways : IEEE, IBM Floating Point, 1,2 and 4 byte two's complement integers.

A useful chart which shows the differences between SEG-Y rev0, rev1 and rev2. Published in Agile's blog post by Matt Hall, SEGY-Y Rev 2 is in the following figure.



What is SEG-Y?

- Widely adopted binary file standard for seismic data, especially industrial seismic reflection data.
- Maintained by the SEG Technical Standards Committee.
- Updated to accommodate high-capacity media and trends in seismic acquisition, such as 3D, time lapse, very high trace-densities, and microseismic monitoring.
- Files contain 2 or more file headers, one textual and one binary, followed by one or more data traces, each with one or more headers.
- Latest version, SEG-Y Rev 2, was ratified in April 2017.
- SEG-Y is not intended as a field recording format; see the SEG-D and SEG-2 formats.

Changes from Rev 0 to Rev 1

- File may be written to any medium resolvable to a stream of variable length records.
- Data word formats expanded to include IEEE 32-bit float, and 8-bit integers.
- Some additional fields in the binary file header and trace headers, others clarified.
- Textual file header can be ASCII encoded.
- Extended textual file header introduced, using a stanza layout.
- Trace identification expanded, and engineering conversions introduced.
- Note that little-endian byte ordering is not compliant with Rev 0 or Rev 1.

Changes from Rev 1 to Rev 2

- Support for little-endian byte ordering, and pairwise byte-swapping.
- Provision for up to 65 535 additional trace headers; bytes 233–240 for names.
- Up to $2^{32}-1$ samples/trace and traces/record, and $2^{64}-1$ traces/line or traces/file.
- Arbitrarily large or small sample intervals.
- Additional sample formats, including IEEE 64-bit floats.
- Microsecond precision in time and date stamps.
- Additional precision on coordinates, depths, elevations, and more CRS support.
- Extended textual file header stanzas must start at 3200-byte boundaries.
- Stanzas may appear after the last data trace in a data trailer.
- Flexible trace header mapping via extended textual file header definition.
- XML allowed in extended textual file header and data trailer.

Figure 8.3: SEG-Y Seismic Data Exchange Format

For more details of the SEG-Y check this link: 8

8.4 SU file

A SU formatted file is just a simple version of a file, containing only trace information :

- No 3200 byte textual header and no extended textual headers.
- No binary header.
- The data must be formatted as IEEE.
- Data can be both little 8.14 and big endian 8.15 formatted.

8.5 Boundary Condition

In real life the wave goes to infinity, however in our RTM the domain is limited so if we didn't simulate the infinite behavior, the wave would not disappear at the end of the domain it will reflect and affect our simulation badly(wrong behavior).

The boundary condition is additional computation at the boundary of the domain to simulate the effect that when the wave reaches boundaries it will absorb the wave as much as it can so that the reflected part(if exist) will be small noise that does not hardly affect our simulation or migration results.

Boundary conditions are around the domain in 4 sides.(up,down,right,left)all boundaries.

We have artificial and physical boundary conditions.

8.5.1 Artificial boundary condition

If the wave values is 100 I need it to be 90 then 80 and so on (reduce the pressure values by external side (myself) like Sponge and CPML.

Sponge:

In this boundary condition, values at boundaries are multiplied by 0.9 so it will be reduced while moving in the boundaries.

CPML:

In this boundary condition, changes to the wave equation itself is applied to add neutral attenuation without adding external factor.(perfectly matched layers)(no refraction from boundaries)as if you change the values of boundaries by yourself it will exist refraction as you change from outside but here is a neutral one integrated in the wave equation so it will absorb as much as it can(best case)

8.5.2 Physical boundary condition

It is a physical attenuation to the wave like Random boundaries.

Random:

It is to bound your domain by random points in velocity so the wave when it hits it will not give pure reflections at specific points as it crashes down to multi reflections that are not concentrated into one point, so random boundaries are add randomization to make the reflections random and if we use the multiple shots the stacking of the results of the shots will remove the random noise from the random boundaries.

8.6 Propagation approaches

In our project, we are supporting the following approaches:

8.6.1 Two propagation

An I/O intensive approach where you would store all of the calculated wave fields while performing the forward propagation, then read them while performing the backward propagation.

8.6.2 Two propagation with ZFP compression

Using the ZFP compression technique in the two-propagation workflow to reduce the volume of data in the I/O.

8.6.3 Three propagation

A computation intensive approach where you would calculate the forward propagation storing only the last two time steps. You would then do a reverse propagation, propagate the wave field stored from the forward backward in time alongside the backward propagation.

8.7 MPI shot distribution

MPI shot distribution is an approach used in HPC clusters for imaging techniques like Reverse Time Migration(RTM) to run it on multiple nodes, it mainly distributes the working shots among nodes of the clusters using the Message Passing Interface standard (MPI). In this approach each MPI process works on a number of shots and then an MPI communication happens among the processes for collection of results per process and generating the final result. There exist different approaches for MPI shot distribution, we mainly used static and dynamic approaches to achieve this:

8.7.1 Static

- Number of shots per process is defined from the beginning.
- Shots per process=number of shots/number of working processes
  - Remainder is handled

It has two approaches for implementation:

Static Server:

Static with server and clients:

1. The server obtains the shot IDs that can be processed.
   - Ends the RTM if there are no shots available.
2. Each client works in an equal number of shots.
   - Process 1 works on shots 0, n, 2n.
   - Process 2 works on shots 1, n+1, 2n+1.
   - And so on till there are no shots available.
3. An MPI reduction is done and the server (process with rank 0)has the final stacked correlation from all the clients.

Static Server-less:

Static with no server:

1. All processes obtain the shot IDs that can be processed
   - Ends the RTM if there are no shots available.
2. Each process works on an equal number of shots.
   - Process 0 works on shots 0, n, 2n.
   - Process 1 works on shots 1, n+1, 2n+1.
   - And so on till there are no shots available.
3. An MPI reduction is done and the process with rank 0 has the final stacked correlation from all other processes.

8.7.2 Dynamic

It has two approaches for implementation:

Dynamic Server:

Dynamic with server and clients:

1. The server obtains the shot IDs that can be processed.
   - Ends the RTM if there are no shots available.
2. The server sends one shot to each client process.
3. Each client works in the received shot.
4. The client sends to the server when it finishes that shot.
5. The server sends another shot to this client
   - Repeat steps 3, 4 and 5.
   - Each client does the accumulation of their own shots.
6. The server raises a flag when there are no more shots available.
7. An MPI reduction is done and the server has the final stacked correlation from all the clients.

Note: The case when number of shots < number of processes is handled.

Dynamic Serverless:

Dynamic with server working on shots as others:

1. The server obtains the shot IDs that can be processed.
    - Ends the RTM if there are no shots available.
2. The server allocates the first shot ID to himself.
3. The server sends one shot to each client process.
4. Each client works in the received shot.
    - The server works on the first shot ID at the same time.
5. The server works on the coming shot ID when it finishes.
6. The client sends to the server when it finishes that shot.
7. The server sends another shot to this client.
8. This client works in the received shot.
    - Repeat steps 5,6,7 and 8.
    - Each client does the accumulation of their own shots.
9. The server raises a flag when there are no more shots available.
10. An MPI reduction is done and the server has the final stacked correlation from all the clients.

Note: The case when number of shots $<$ number of processes is handled.

8.7.3 Static advantages and disadvantages:

Advantages:

- Simple in implementation.
- No communication overhead is required between server and clients.
- Faster in time.
- Suitable for homogeneous clusters.

Disadvantages:

- Not suitable for heterogeneous clusters where a node has a higher processing power than others.
    - A waiting time overhead will be added.

8.7.4 Dynamic advantages and disadvantages:

Advantages:

- Suitable for heterogeneous clusters.

Disadvantages:

- An overhead is added due to the server client communication.
- Slower than static approaches.
- Dynamic scheduling adds some overhead.
  - A waiting time overhead will be added.

8.8 Isotropic Wave forms

Isotropic refers to the properties of a material which is independent of the direction Some of different implementations of the computation kernel are in the coming subsections.

8.8.1 Second order

It is the solution of the partial derivative equation using second order finite difference concept 8.10, where p is pressure and c is wave velocity.

$$\frac{\partial^2 p}{\partial t^2} = c^2 \left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} \right) \tag{8.1}$$

Obtaining an approximation for the second derivative of pressure in time using discrete points is as follows (left hand side of equation 8.1):

$$\frac{\partial^2 p}{\partial t^2} = \frac{p_{t+1}[x,y,z] - 2p_t[x,y,z] + p_{t-1}[x,y,z]}{\Delta t^2} \tag{8.2}$$

Where obtaining the same approximating of pressure in space [x,y,z] using discrete points is as follows (right hand side of equation 8.1):

$$\frac{\partial^2 p}{\partial x^2} = FD_x = \frac{1}{\Delta x^2} \left(c_0 p_t[x,y,z] + \sum_{i=1}^{n} c_i \left(p_t[x+i,y,z] + p_t[x-i,y,z] \right) \right) \tag{8.3}$$

$$\frac{\partial^2 p}{\partial y^2} = FD_y = \frac{1}{\Delta y^2} \left(c_0 p_t[x,y,z] + \sum_{i=1}^{n} c_i \left(p_t[x,y+i,z] + p_t[x,y-i,z] \right) \right) \tag{8.4}$$

$$\frac{\partial^2 p}{\partial z^2} = FD_z = \frac{1}{\Delta z^2} \left(c_0 p_t[x,y,z] + \sum_{i=1}^{n} c_i \left(p_t[x,y,z+i] + p_t[x,y,z-i] \right) \right) \tag{8.5}$$

So merging equations 8.2, 8.3, 8.4 and 8.5 gives this equation:

$$\begin{aligned} p_{t+1}[x,y,z] &= 2p_t[x,y,z] - p_{t-1}[x,y,z] \\ &\quad + \Delta t^2 \cdot c^2 \left(FD_x + FD_y + FD_z\right) \end{aligned} \tag{8.6}$$

Which is quite simple to implement a working version and a little hard to implement an optimized and efficient version.

8.8.2 Staggered

It is the solution of the partial derivative equation using staggered-grid finite difference concept 8.10
Staggered-grid finite-difference (FD) methods are widely used in seismic modeling because of their high computational efficiency, lower memory requirement and high accuracy.
Acoustic wave equation in 2 dimensions can be written as:

$$\frac{\partial p}{\partial t} = -\frac{1}{k}\left(\frac{\partial v_z}{\partial z} + \frac{\partial v_x}{\partial x}\right) \tag{8.7}$$

$$\frac{\partial v_x}{\partial t} = -\frac{1}{\rho}\left(\frac{\partial p}{\partial x}\right) \tag{8.8}$$

$$\frac{\partial v_z}{\partial t} = -\frac{1}{\rho}\left(\frac{\partial p}{\partial z}\right) \tag{8.9}$$

where P is the acoustic pressure fluctuation, ρ is the density,

$$K = \rho v^2 \text{ (is bulk modulus)}$$

and v is the wave-propagation speed.

v_x and v_z are scaled particle-velocity components

Assuming a constant density , the staggered-grid FD scheme is as follows (M is assumed to be 7):

$$p_{i,j}^{k+1/2} = p_{i,j}^{k-1/2} - \frac{v^2\tau}{h}\left(\sum_{m=1}^{M} c_m\left(v^k_{x(i,j+m-1/2)} - v^k_{x(i,j-m+1/2)}\right) + \sum_{m=1}^{M} c_m\left(v^k_{z(i,j+m-1/2)} - v^k_{z(i,j-m+1/2)}\right)\right) \tag{8.10}$$

$$v^{k+1}_{z(i+1/2,j)} = v^k_{z(i+1/2,j)} - \frac{\tau}{h}\sum_{m=1}^{M} c_m\left(p^{k+1/2}_{i+m,j} - p^{k+1/2}_{i-m+1,j}\right) \tag{8.11}$$

$$v^{k+1}_{x(i,j+1/2)} = v^k_{x(i,j+1/2)} - \frac{\tau}{h}\sum_{m=1}^{M} c_m\left(p^{k+1/2}_{i,j+m} - p^{k+1/2}_{i,j-m+1}\right) \tag{8.12}$$

Where M is the length of the FD operator, cm are the staggered grid FD coefficients, h is the spatial-grid interval and τ is the time step.
Hereafter, we call the staggered-grid FD scheme in equations 8.10-8.12 the traditional staggered-grid FD scheme.

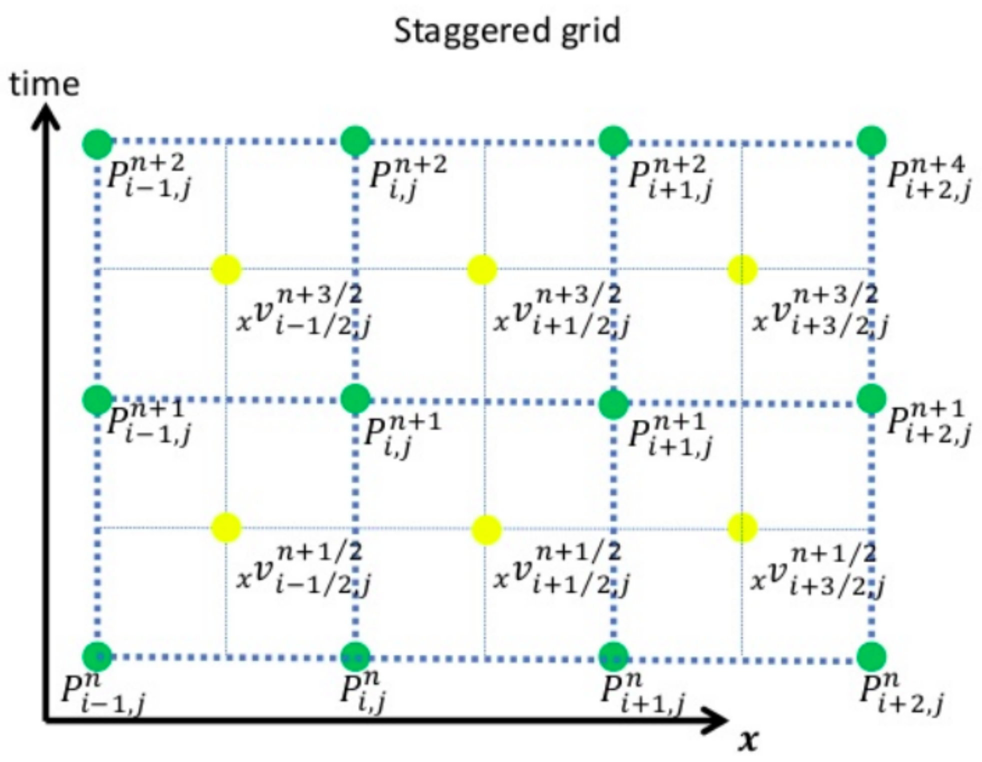


Figure 8.4: Staggered grid

8.9 Transverse Isotropy (TI)

It is an an isotropic wave forms which are direction-d It is the most typical anisotropic media to consider, where the velocity normal to the bedding is lower than along it, and which may be vertical (VTI) or tilted (TTI).

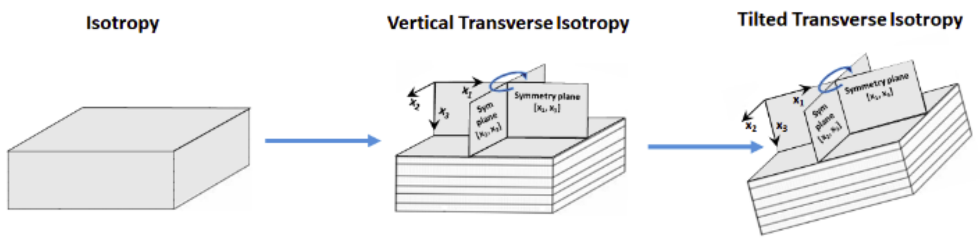


Figure 8.5: VTI vs TTI

For more details on the TI check these link: 9

8.9.1 VTI

Vertical transverse Isotropy equations:

$$\frac{\partial^2\sigma_H}{\partial t^2} = V_p^2\left((1+2\epsilon)\left(\frac{\partial^2\sigma_H}{\partial x^2}+\frac{\partial^2\sigma_H}{\partial y^2}\right)+\sqrt{1+2\delta}\frac{\partial^2\sigma_v}{\partial z^2}\right) \tag{8.13}$$

$$\frac{\partial^2 \sigma_v}{\partial t^2} = V_p^2 \left((1+2\epsilon) \left(\frac{\partial^2 \sigma_H}{\partial x^2} + \frac{\partial^2 \sigma_H}{\partial y^2} \right) + \sqrt{1+2\delta} \frac{\partial^2 \sigma_v}{\partial z^2} \right) \tag{8.14}$$

Where σ_H and σ_v correspond to the horizontal and vertical stress components
V_P is the P-wave 8.2.1 velocity in the direction of symmetry axis
ϵ and δ are Thomsen's anisotropy parameters (Thomsen, 1986)

8.9.2 TTI

Tilted Transverse Isotropy equations:

$$\begin{aligned}\frac{\partial^2}{\partial x^2} \equiv{}& \cos^2\theta\cos^2\phi\frac{\partial^2}{\partial x^2} + \cos^2\theta\sin^2\phi\frac{\partial^2}{\partial y^2} + \sin^2\theta\frac{\partial^2}{\partial z^2} \\ &+ \cos^2\theta\sin 2\phi\frac{\partial^2}{\partial x\partial y} - \sin 2\theta\cos\phi\frac{\partial^2}{\partial x\partial z} \\ &- \sin 2\theta\sin\phi\frac{\partial^2}{\partial y\partial z}\end{aligned} \tag{8.15}$$

$$\frac{\partial^2}{\partial y^2} \equiv \sin^2\phi\frac{\partial^2}{\partial x^2} + \cos^2\phi\frac{\partial^2}{\partial y^2} - \sin 2\phi\frac{\partial^2}{\partial x\partial y} \tag{8.16}$$

$$\begin{aligned}\frac{\partial^2}{\partial y^2} \equiv{}& \sin^2\theta\cos^2\phi\frac{\partial^2}{\partial x^2} + \sin^2\theta\sin^2\phi\frac{\partial^2}{\partial y^2} + \cos^2\theta\frac{\partial^2}{\partial z^2} \\ &+ \sin^2\theta\sin 2\phi\frac{\partial^2}{\partial x\partial y} + \sin 2\theta\cos\phi\frac{\partial^2}{\partial x\partial z} \\ &+ \sin 2\theta\sin\phi\frac{\partial^2}{\partial y\partial z}\end{aligned} \tag{8.17}$$

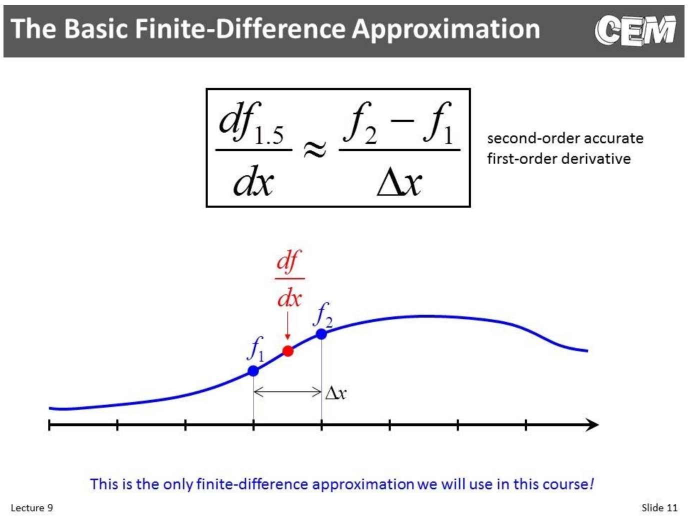
Where θ is the phase angle measured from the symmetry axis
And ϕ is the dip measured to the vertical.

8.10 Finite difference concept

It is a simple numerical method based on taylor expansion.

8.11 Ricker wavelet

A zero-phase wavelet, the second derivative of the Gaussian function or the third derivative of the normal-probability density function. A Ricker wavelet is often used as a zero-phase embedded wavelet in modeling and synthetic seismograph manufacture. See the below figure. Named for Norman H. Ricker (1896–1980), American geophysicist.
For more details check this link 2



The Basic Finite-Difference Approximation

$$\boxed{\frac{df_{1.5}}{dx} \approx \frac{f_2 - f_1}{\Delta x}}$$

second-order accurate first-order derivative

This is the only finite-difference approximation we will use in this course!

Lecture 9

Slide 11

Figure 8.6: Finite difference approximation

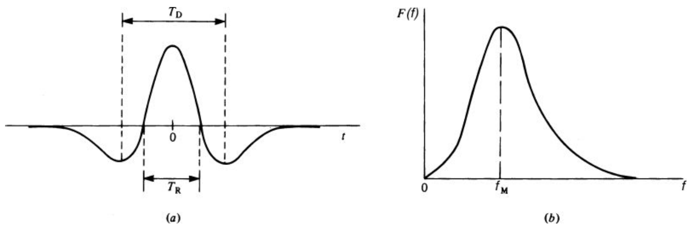


Figure 8.7: Ricker wavelet.(a) Time-domain (b) frequency-domain.

8.12 First touch policy

Specifying memory policies for a process or address range does not cause any allocation of memory, which is often confusing to newcomers.
Memory policies specify what should happen when the system needs to allocate memory for a virtual address.
Pages in a process's memory space that have not been touched or that are zero do not have memory assigned to them. The processor will generate a hardware fault when a process touches or writes to an address (page fault) that is not yet populated.
During page-fault handling by the kernel, the page is allocated. The instruction that caused the fault is then restarted and will be able to access the memory as needed. What matters, therefore, is the memory policy in effect when the allocation occurs. This is called the first touch.
The first-touch policy refers to the fact that a page is allocated based on the effective policy when some process first uses a page in some fashion. The effective memory policy on a page depends on memory policies assigned to a memory range or on a memory policy associated with a task. If a page is only in use by a single thread, then there is no ambiguity as to which policy will be followed. However, pages are often used by multiple threads. Any one of them may cause the page to be allocated. If the threads have different memory policies, then the page may as a result seem to be allocated in surprising ways for a process that also sees the same page later.

8.13 Imaging conditions

In our project we are supporting 2 imaging conditions cross correlation kernel and Angle Domain Common Image Gathering (ADCIG).

8.13.1 Cross Correlation Kernel

It is an efficient tool for detection and characterization of seismic signals, we use it to calculate the similarity or time alignment of forward-propagated source wave field and back-propagated receiver wave field.

In our project it can be of the following types:

1. **No compensation:** will provide normal cross correlation
2. **Source compensation:** will compensate the cross correlation stack for the source illumination effect.
3. **Receiver compensation:** will compensate the cross correlation stack for the receiver illumination effect.
4. **Combined compensation:** will compensate the cross correlation stack for both the source and the receiver illumination effect.

8.13.2 Angle Domain Common Image Gathering (ADCIG)

Angle-domain common-image gathers ADCIG have become a useful tool in image interpolation, migration velocity analysis MVA and amplitude-versus-angle AVA analysis, in RTM, we can produce angle-domain common image gathers in either the scattering-angle domain or the dip-angle domain 8.13.2.
We here obtain angle-domain common image gathers efficiently in acoustic RTM by using the normalized Poynting vectors 8.13.2 which is easy to compute during wavefield propagation.
For more details please check these links: 3 & 4

Poynting Vector:

In physics, the Poynting vector represents the directional energy flux (the energy transfer per unit area per unit time) of an electromagnetic field.
Using the same understanding, in the seismic waves we can identify the Poynting vector as an indicator to the direction of energy flux, this transforms the wave from scalar value to a vector with a defined direction in space, hence we can calculate the angle between the vectors representing the source wavefield and those representing the receiver wave-field.

Scatter angle and dip angle:

Relating to the following figure, the scattering angle δ is half the angle between the poynting vector of the source wave-field and the poynting vector of the receiver wave-field, the scattering angle spans between 0 and 90 with the sign information +/- ignored.
While the dip angle α represents the angle of the subsurface structures that reflect

the wavefield, the dip angle spans between -90 and 90 because different signs of dip vectors represent different orientations of subsurface structures.

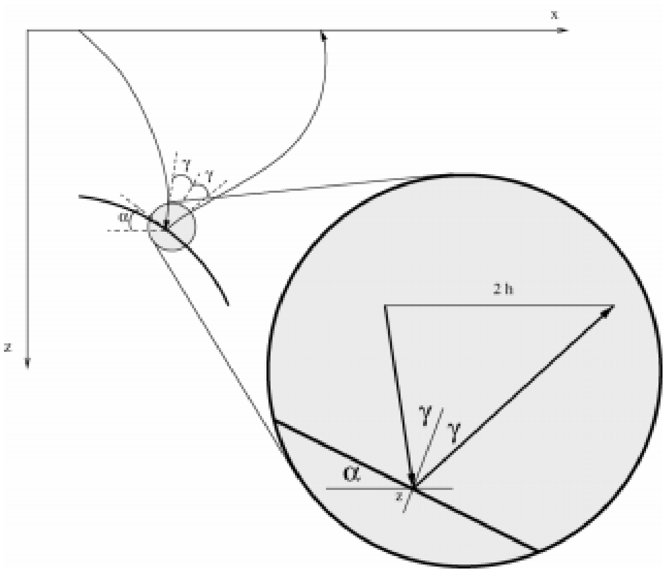


Figure 8.8: Scatter angle vs dip angle

8.14 Little endian

Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first. For example, in a little-endian computer, the two bytes required for the hexadecimal number 4F52 would be stored as 4F52 in storage(52 at address 1000, 4F at 1001).

8.15 Big endian

Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address)
For example, in a big-endian computer, the two bytes required for the hexadecimal number 4F52 would be stored as 4F52 in storage (if 4F is stored at storage address 1000, for example, 52 will be at address 1001).

8.16 Stencil

In mathematics especially the areas of numerical analysis, a stencil is a geometric arrangement of a nodal group that relate to the point of interest by using a numerical approximation routine.
For example a **Five-point stencil** means given grid in one dimension the five-point stencil of a point in the grid (the red point) is a stencil made up of the point itself together with its four "neighbors".

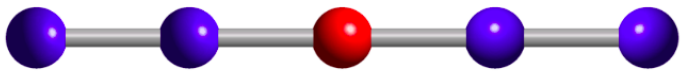


Figure 8.9: five-point stencil in one dimension

The same approach is applied for the different dimensions if a multidimensional grid is used.

8.16.1 Stencil order

It is the total number of neighbors for the point of interest.
For the one dimension **Five-point stencil**,the stencil order =4 (two points in the right direction and two points in the left direction)

8.16.2 Half-length

It is the total number of neighbors for the point of interest at only one direction.
For the one dimension **Five-point stencil**, the half-length =2 (two neighbors in the right direction or two neighbors in the left direction)

8.17 Cache blocking

Cache Blocking is a technique to rearrange data access to pull subsets (blocks) of data into cache and to operate on this block in order to avoid having to repeatedly fetch data from main memory.
It is possible to manually block loop data in such a way to reuse cache.
Manual cache blocking means to manually define the cache block size you need to use in your loop.

8.18 SYCL

SYCL is a higher-level programming model to improve programming productivity on various hardware accelerators.

It is a single-source domain-specific embedded language (DSEL) based on pure C++17. It is a standard developed by Khronos Group, announced in March 2014.

8.19 Technologies supported

In our project we are supporting these different technologies.

8.19.1 OpenMP technology

OpenMP (Open Multi-Processing) is a set of compiler directives, library procedures and environment variables intended for programming multi-threaded applications on multi-processor systems with shared memory (SMP-systems).

The first OpenMP standard was developed in 1997 as an API oriented on writing easy portable multi-threaded applications. At first, it was based on FORTRAN language but then included C and C++ as well.

OpenMP interface became one of the most popular parallel programming technologies. OpenMP is successfully exploited both in programming of supercomputer systems with many processors and in desktop user systems.

For more details check this link: 5

8.19.2 oneAPI technology

Intel oneAPI (oneAPI) is A cross-industry, open,standards-based unified programming model that delivers a common developer experience across accelerator architecture for faster application performance,more productivity, and greater innovation.

The oneAPI industry initiative encourages collaboration on the oneAPI specification and compatible oneAPI implementations across the ecosystem.

For more details check this link: 6

8.19.3 CUDA technology

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs).

With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs.

For more details check these slides:7

Chapter 9

References

1. Seismic Toolbox repository:
https://gitlab.brightskiesinc.com/parallel-programming/SeismicToolbox

2. Ricker wavelet:https://wiki.seg.org/wiki/Dictionary:Ricker_wavelet

3. Angle-domain common-image gathers in reverse-time migration:
https://onlinelibrary.wiley.com/doi/full/10.1111/1365-2478.12785

4. Efficient dip-angle angle-domain common-image gather:
https://onlinelibrary.wiley.com/doi/full/10.1111/1365-2478.12696

5. OpenMp:// https://pvs-studio.com/en/blog/posts/a0057/

6. oneAPI://https://www.oneapi.com/

7. CUDA:https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf

8. SEG-Y :https://cultpenguin.gitbooks.io/segymat/content/format.html

9. TI resources:

   - https://csim.kaust.edu.sa/files/report/11_seg/papr283.pdf
   - https://library.seg.org/doi/abs/10.1190/1.3059320