[COR: Number]

Cover Letter

formikoj: A flexible library for data management and processing in geophysics - Application for seismic refraction data

Matthias Steiner,Adrián Flores Orozco

Dear Editors-in-Chief,

we are submitting our manuscript "formikoj: A flexible library for data management and processing in geophysics - Application for seismic refraction data", which we consider is a suitable contribution for Computers & Geosciences. We confirm that the submission follows all the requirements and includes all the items of the submission checklist.

The manuscript presents the open-source python library formikoj for managing and processing geophysical data collected in environmental and engineering investigations. formikoj was specifically implemented for multi-platform usage to allow the efficient collaboration and exchange of data between different partners in research projects and academia. In this regard, we believe that this library aids in providing reproducible data and results as well as establishing and maintaining good research practices. Accordingly, we consider this manuscript relevant to the audience of Computers & Geosciences, and in general for geoscientists and practitioners working with geophysical methods.

We provide the source codes in a public repository with details listed in the section "Code availability".

We look forward to your decision.

Yours sincerely,

Matthias Steiner and Adrián Flores Orozco
Research Unit Geophysics, Department of Geodesy and Geoinformation, TU Wien; matthias.steiner@geo.tuwien.ac.at

Highlights

formikoj: A flexible library for data management and processing in geophysics - Application for seismic refraction data

Matthias Steiner,Adrián Flores Orozco

- flexible open-source and cross-platform library for managing and processing of geophysical data
- possibility to be deployed for different geophysical methods and/or instruments
- application for the modeling and processing of seismic refraction datasets
- applicable for seismic refraction data collected in 2D and 3D survey geometries
- easily scalable for custom requirements

formikoj: A flexible library for data management and processing in geophysics - Application for seismic refraction data

Matthias Steiner[a,*], Adrián Flores Orozco[a]

[a]Research Unit Geophysics, Department of Geodesy and Geoinformation, TU Wien

ARTICLE INFO

Keywords:
geophysical data processing
seismic refraction
first break picking
seismic waveform modeling
cross-platform application
geophysical python library
flexible open-source librarys
wave based methods

ABSTRACT

We introduce here the open-source library formikoj, which provides a flexible framework for managing and processing geophysical data collected in environmental and engineering investigations. To account for the substantial changes regarding the market shares of operating systems within the last two decades, the library is specifically implemented and tested for cross-plattform usage. We illustrate the applicability of the formikoj library to forward-model seismic refraction waveform data with the `SeismicWaveformModeler` based on a custom subsurface model and survey geometry. Moreover, we present two case studies from seismic refraction tomography field measurements to exemplify the wide range of possibilities provided by the formikoj library for the processing of real data. In particular, we explore different visualization of the seismic traveltime readings to enhance their consistency prior to the inversion. The low-level access provided by our library aims at giving the users the possibility to implement particular modeling, visualization and processing tools specifically designed for their objectives or other geophysical methods.

CRediT authorship contribution statement

Matthias Steiner: Conceptualization and implementation of the library, creating the figures, preparation of the manuscript. **Adrián Flores Orozco:** Conceptualization of the library, preparation of the manuscript.

1. Introduction

The acquisition of spatially quasi-continuous data in a non-invasive manner renders geophysical methods suitable for engineering and environmental investigations (e.g., Parsekian et al., 2015; Nguyen et al., 2018; Romero-Ruiz et al., 2018). However, the processing of geophysical data often relies on commercial software solutions and the associated licensing costs might render their use prohibitively expensive, which might be the case for academic projects or institutions. The most popular packages are Res2DInv[1] for electrical methods, Halliburton Landmark SeisSpace ProMAX[2] or ParkSeis [3] for seismic methods, or ReflexW[4] for ground-penetrating radar and seismic methods. A common limitation of the aforementioned software solutions refers to their specific platform requirements mainly related to the type and version of the operating system; moreover, the possibility to adapt the code are limited if possible at all. Considering the substantial changes regarding the market shares of operating systems within the last two decades, platform-specific software packages are becoming particularly obstructive for academic research and teaching. The

*Corresponding author
ORCID(s): 0000-0002-3595-3616 (M. Steiner); 0000-0003-0905-3718 (A. Flores Orozco)

[1]https://www.geometrics.com/software/res2dinv/, last accessed on February 28, 2023
[2]https://www.landmark.solutions/SeisSpace-ProMAX, last accessed on February 28, 2023
[3]https://www.parkseismic.com/parkseis/, last accessed on February 28, 2023
[4]https://www.sandmeier-geo.de/reflexw.html, last accessed on February 28, 2023

increasing popularity of the Python programming language led to the development of various cross-platform open-source software packages for processing, modeling and inverting geophysical data. Available packages can focus on specific geophysical methods, for instance, ResIPy (Blanchy et al., 2020) for electrical data, GPRPy (Plattner, 2020) for ground-penetrating radar data, or ObsPy (Beyreuther et al., 2010) and Pyrocko (Heimann et al., 2017) for seismological data. Other packages provide frameworks for the inversion and permit the inclusion of forward models for different geophysical methods, e.g., SimPEG (Cockett et al., 2015), Fatiando a Terra (Uieda et al., 2013) or pyGIMLi (Rücker et al., 2017).

The seismic refraction tomography (SRT) is a standard technique in environmental and engineering applications. Often applied together with other geophysical methods, the SRT is routinely used, e.g., in permafrost studies (e.g., Draebing, 2016; Steiner et al., 2021), for the investigation of landfills (e.g., Nguyen et al., 2018; Steiner et al., 2022), or for hydrogeological characterizations (e.g., Bücker et al., 2021). The market for seismic processing software has long been dominated by software packages designed for the processing of large datasets, e.g., associated to oil or gas exploration. Accordingly, these seismic processing solutions may not be suited for small-scale projects in environmental and engineering studies, or for teaching activities. ReflexW overcomes such limitations by providing processing tools specifically designed for near-surface investigations at substantially lower costs. In terms of licensing costs, Stockwell (1999) went a step further by making the Seismic Unix framework available entirely free of charge; whereas Guedes et al. (2022) recently presented RefraPy, a python processing tool for seismic refraction data. Implemented in python, RefraPy is potentially suitable for cross-platform usage, yet Guedes et al. (2022) developed and tested solely for Windows operating systems. Moreover, RefraPy does not offer the possibility to generate synthetic seismic waveform data, as required for survey design, as well as teaching and interpretation purposes.

The formikoj library presented here is an open-source framework for creating synthetic datasets, as well as for managing and processing numerical and field data independently from the operating system and without licensing costs; thus, overcoming limitations associated to existing solutions. The design of the library follows the multi-method concept of pyGIMLi and SimPEG, which allows for the implementation of custom designed tools for different geophysical methods. The usage of transparent file formats, e.g., the unified data format (udf[5]), and data management concepts (SQLite database) within the formikoj framework facilitates a simple data exchange between partners in research projects and academia, which is required to guarantee the repeatability of results and good research practices. Considering the diverse applications of the SR method we demonstrate the applicability of the proposed library based on tools implemented within the formikoj framework for the modeling and processing seismic waveform data. In particular, we present here a series of illustrative use cases based on the formikoj library referring to (i) the modeling of synthetic seismic refraction (SR) waveform data, (ii) the processing of a 2D SR field dataset collected with a roll-along

[5]http://resistivity.net/bert/data_format.html, last accessed on February 28, 2023

survey geometry, and (iii) the processing of a 3D SR field data set.

2. Design and structure of the `formikoj` library

As illustrated in Figure 1, the formikoj library comprises a modeling and a processing module, which both rely on a common utilities module. The `DataModeler` and the `MethodManager` class provide the basis to add modeling or processing functionalities for any kind of geophysical methods. In this manuscript, we present the application of the formikoj library for the modeling and processing of seismic refraction data based on the fundamental use cases presented in Figure 2 and 3, respectively. These diagrams aim at illustrating the corresponding workflows as well as the required interactions between the user, the formikoj library and third-party packages. The `SeismicWaveformModeler` and `SeismicRefractionManager` classes implement these fundamental use cases, yet their actual capabilities are continuously expanded, e.g., to address specific modeling or processing requirements as well as to enhance the user experience. Similar to RefraPy, these classes are built upon the functionalities of existing packages such as ObsPy for the processing of seismological data (Beyreuther et al., 2010) and pyGIMLi for the modeling and inversion of different geophysical data (Rücker et al., 2017). Other important third party dependencies refer to NumPy (Harris et al., 2020) and Pandas (McKinney, 2010) for general data handling, as well as matplotlib (Hunter, 2007) and PyVista (Sullivan and Kaszynski, 2019) for data visualization. In the current version, we implemented and tested formikoj primarily on Linux machines, yet the library has been successfully tested and used on all major operating systems, i.e., Linux, MacOS and Windows.

2.1. Generation of seismic waveform data for synthetic subsurface models

The SR method exploits the ground motion recorded by sensors installed in the surface (e.g., geophones) to characterize the propagation of seismic waves generated at well known locations (i.e., shot stations). The visualization of the ground motion as function of time yields a so-called seismogram for each geophone position. Accordingly, the `SeismicWaveformModeler` class provides a flexible way to generate synthetic seismic waveform data either in a python script, interactively in a jupyter notebook or an ipython shell. To create an instance of the class the user provides the absolute or relative path to the working directory as parameter to the constructor:

```
# Import the SeismicWaveformModeler from the formikoj library
from formikoj import SeismicWaveformModeler

# Create an instance of the SeismicWaveformModeler
swm = SeismicWaveformModeler('.')
```

```
INFO    : Created instance of SeismicWaveformModeler
```

Table 1
Description of the information to be provided in the columns of a measurement scheme in csv format.

Column	**Content**	**Data type**	**Description**
1	x coordinate	float	Station x coordinate, e.g., given in (m)
2	y coordinate	float	Station y coordinate, e.g., given in (m)
3	z coordinate	float	Station z coordinate, e.g., given in (m)
4	Geophone	bool	1 in case of a receiver station, 0 otherwise
5	Shot	bool	1 in case of a shot station, 0 otherwise

In the working directory, the required input files are provided via the subdirectory *in*, whereas the modeling output will be stored in the automatically created subdirectory *out*. The key input file is the measurement scheme as it contains information regarding the distribution of the shot and geophone stations. If provided in the unified data format, the measurement scheme is imported directly with pyGIMLi into a `DataContainer`. In case the measurement scheme is provided as a csv file, the `SeismicWaveformModeler` reads the information and writes it to a pyGIMLi `DataContainer` object. If provided as csv file, the measurement scheme contains a single line for each station in the survey layout, where a station either hosts a geophone or a shot, or both (see Table 1). The values provided in each line need to be separated by a unique delimiter, and the file must not contain a header.

For the modeling of the seismic waveform data, the parameters characterizing the base wavelet, the synthetic subsurface model and the resulting waveform datasets are provided (see Table 2) in a configuration file following the yaml format:

```
wavelet:
    length: 1.024
    frequency: 100
    sampling_rate: 2000
    pretrigger: 0.02

model:
    velmap: [[1, 750], [2, 2500], [3, 4000]]
    layers: [[1, 3], [2, 5], [3, 15]]
    quality: 32
    area: 10
    smooth: [1, 10]
    sec_nodes: 3

dataset:
    number: 1
    names: [syn_data]
    noise: 1
```

```
    noise_level: 1e-4
    missing_shots: 1
    broken_geophones: 1
    wrong_polarity: 1

traveltimes:
    noise_relative: 0.
    noise_absolute: 0.
```

In this exemplary configuration, the first block (`wavelet`) contains the parameterization of the base wavelet, which controls the modeling of the seismic waveforms (see Table 2). The second block (`model`) contains information regarding the synthetic subsurface model. In the configuration file, the user can define simple models where all layers are considered to be parallel to the surface topography. The topography is inferred from the station geometry provided in the measurement scheme, while the seismic velocity (`velmap`) and thickness (`layers`) of the different layers are defined in the configuration file. The remaining parameters (`quality`, `area`, `smooth` and `sec_nodes`) refer to the properties of the mesh that is generated for the forward modeling (we refer to the respective pyGIMLi resources[6] for further information). In the third block of the exemplary configuration file (`dataset`), the user can set specific names for the datasets to be created (the number of datasets is automatically determined), or set the number of datasets to be created and the dataset names are automatically generated with the prefix *dataset_*. The remaining parameters in this block control the random error (`noise`) and systematic errors in the modeled seismic waveform data (see Table 2). The number and position of the shot and geophone stations affected by the systematic errors are randomly chosen with a maximum of 5 % of the total number of stations in order to avoid a high number of invalid trace data. The parameters in the final block (`traveltimes`) control the forward modeling of the corresponding seismic traveltimes (see Table 2). In particular, the user can set the relative and absolute noise that is added to the synthetic seismic traveltimes computed for the given subsurface model.

If stored in the input directory such a configuration file can be imported through the `load` method of the `SeismicWaveformMo`

```
# Load and parse the configuration file
swm.load('config')
```

```
INFO    : Configuration loaded
```

Instead of defining as simple subsurface model in the configuration file, we provide a more complex model in the binary mesh format (e.g., a bms file). In particular, we prepare the model and the corresponding forward modeling mesh based on the mesh tools provided by pyGIMLi and save the mesh in the bms format (a commented version of the

[6] https://www.pygimli.org/pygimliapi/_generated/pygimli.meshtools.html#pygimli.meshtools.createMesh, last accessed February 28, 2023

Table 2
Description of the parameters, which can be defined in a configuration file used for the modeling of the synthetic seismic data.

Parameter	Unit/ Data type	Description
`wavelet`		
length	s	Length of the base wavelet, which also defines the length of the synthetic seismic waveform data
frequency	Hz	Frequency of the base wavelet
sampling_rate	Hz	Defines temporal resolution of the seismic waveforms
pretrigger	s	Add buffer to the seismic waveforms before the onset of the actual data
`dataset`		
number	int	Number of datasets to be created
names	list (string)	Names of the datasets
noise	bool	1 in case noise should be added to the synthetic waveforms, 0 otherwise
noise_level	-	Level of the seismic background noise
missing_shots	bool	1 in case the datasets should be affected by missing shot files, 0 otherwise
broken_geophones	bool	1 in case the datasets should comprise broken geophones (i.e., no data in the corresponding seismograms), 0 otherwise
wrong_polarity	bool	1 in case the datasets should contain traces with inverse polarity, 0 otherwise
`model`		
velmap	list (float/int)	For each layer the first value defines the marker and the second one the seismic velocity within the layer
layers	list (float/int)	For each layer the first value defines the marker and the second one the thickness
`traveltimes`		
noise_relative	%/100	Relative noise to be added to the forward modeled seismic traveltimes
noise_absolute	s	Absolute noise to be added to the forward modeled seismic traveltimes

python script is presented in the Appendix). Similar to the configuration file, a bms file stored in the input directory can be imported into the workflow through the `load` method as follows:

```
8 # Load the mesh into the workflow
9 swm.load('mesh')
```

```
INFO    : Mesh loaded
```

The forward modeling process generating the seismic waveform data is then invoked by passing the string `'waveforms'` as parameter to the `create` method:

```
# Start the modeling of the seismic waveform data
swm.create('waveforms')
```

```
INFO    : Measurement scheme loaded
INFO    : Velocity model created
INFO    : Wavelet created
[++++++++++++++ 100% ++++++++++++++] 2048 of 2048 complete
      ...
[++++++++++++++ 100% ++++++++++++++] 2048 of 2048 complete
INFO    : Dataset 'syn_data' created
```

As can be seen from the log messages, the `SeismicWaveformModeler` first loads the measurement scheme into the workflow. In the second step, the provided mesh and the seismic velocity values defined in the configuration file are combined to create the seismic velocity model to be used for the waveform modeling. Figure 4a presents the subsurface model considered in this study, which consists of a top and a bottom layer with varying thickness characterized by seismic velocity values of $750\,\mathrm{ms}^{-1}$ and $4000\,\mathrm{ms}^{-1}$, respectively. In the center, the model contains an irregularly shaped anomaly associated with a seismic velocity of $2500\,\mathrm{ms}^{-1}$.

The third step refers to the generation of a Ricker wavelet based on the wavelet properties provided through the configuration file. In particular, the `SeismicWaveformModeler` uses the pyGIMLi function `ricker`, which creates the Ricker wavelet given as

$$u = \left(1 - 2\pi^2 f^2 t^2\right) e^{-\pi^2 f^2 (t-t_0)^2} . \tag{1}$$

In Equation 1, f is the frequency of the wavelet given in Hz, t refers to the time base definition, i.e., the length and resolution in the time domain, and t_0 is the time offset of the wavelet.

Subsequently, mesh, velocity model and Ricker wavelet are used to solve the pressure wave equation for each shot station defined in the measurement scheme with the pyGIMLi function `solvePressureWave`. The resultant waveforms at the corresponding geophone stations are extracted and saved in the output directory (*out*) as shown here:

```
working_directory
├── in
└── out
    └── syn_data
        ├── data
        │   ├── protocol.txt
        │   ├── station_coords.csv
        │   ├── Shot_1001.syn
        │   ├── ...
        │   └── Shot_10nn.syn
        ├── syn_data_tt.pck
        └── info.txt
```

The subdirectory *data* contains a separate file in the miniseed format (Ahern et al., 2012; Ringler and Evans, 2015) for each shot position, where the file extension *syn* to indicate that the shot files contain forward modeled seismic waveform data. The measurement protocol (protocol.txt) and the station coordinates provided as a csv file (station_coords.csv) are also stored in this directory. The header of the measurement protocol contains the survey parameters, e.g., sampling rate, recording length, number of geophones and geophone spacing. Moreover, the protocol associates each shot file of the dataset with a specific location in the survey geometry with respect to the geophone positions, e.g.:

```
 ############################
   Line: SYN_syn_data
   Sampling rate: 2000 Hz
   Recording length: 1.024 s
   Number of geophones: 48
   Geophone spacing: 2 m
 ############################

     File number | Station
         1001    |   G001
           :     |     :
         1048    |   G048
```

The auxiliary file info.txt provided in the dataset directory summarizes the parameters from the configuration file and information regarding the simulated systematic errors in the synthetic seismic waveform data:

```
Number of geophones: 48
Number of shots: 48
Recording length (s): 1.024
Sampling frequency (Hz): 2000
Wavelet type: Ricker
Frequency of the wavelet (Hz): 100

Missing shot(s): 32, 18
Broken geophone(s): 45, 3
Wrong polarity geophone(s): 34
```

In Figure 4b, we present the seismic waveform data forward modeled for a shot point located in the center of the profile. This plot visualizes the seismograms by combining the wiggle trace mode, i.e., the data are shown as curves, and the variable density mode, where the strength of the amplitudes is additionally reflected by the color saturation, i.e., high amplitudes refer to a stronger shade than low amplitudes. The presented seismograms show clear first onsets along the entire profile, yet receivers 3 and 45 were modeled to be broken. Crosses overlaid on the valid seismograms at the respective first onset indicate the first break traveltimes *tt* between the shot and the receiver. In addition to the

missing first break traveltimes for the broken receivers, we manually added a systematic error, i.e., we set an erroneous traveltime for receiver 36.

A pseudosection as presented in Figure 4c, illustrates apparent velocity (v_{app}) values computed as

$$v_{app} = \frac{aoffset}{tt}\,, \tag{2}$$

with $aoffset$ referring to the absolute offset between shot and receiver. The v_{app} values are plotted at pseudolocations, where the location along profile direction is defined by the midpoint of the corresponding shot-receiver pair and the pseudodepth is computed as $1/3$ of $aoffset$. As demonstrated in Figure 4c, a pseudosection allows for the identification of missing data (e.g., receiver 3 and 45) as well as systematic errors or outliers, i.e., velocities erroneously influenced by a single shot or receiver (see receiver 36). The main assumption here is that the pseudosection should reveal smooth transitions between lateral and vertical neighbors, considering that the data were collected with gradual changes in the position of the source (i.e., hammer blow) and the receiver (i.e., geophone). The pseudosection will reveal large variations in case of abrupt changes in the topography or the geometry of the array, yet this can be taken into account by the user during the identification of outliers and possible systematic errors.

Based on pseudosections erroneous traveltimes can be identified and subsequently removed or corrected, which is a critical processing step prior to the inversion of the data. In particular, the inversion of first break traveltimes aims at resolving a model of the P-wave velocity of the subsurface materials. Since the inversion of geophysical data is not within the scope of the formikoj library we rely for this purpose on the modeling and inversion capabilities of pyGIMLi (Rücker et al., 2017). The default inversion framework of pyGIMLi uses a generalized Gauss-Newton method to solve the inversion problem through the minimization of an objective function given as:

$$\Psi(\boldsymbol{m}) = \Psi_d(\boldsymbol{m}) + \lambda\Psi_m(\boldsymbol{m})\,. \tag{3}$$

The first term on the right-hand side of Equation3 refers to the data misfit, whereas the second term denotes represents the model constraints, i.e., the regularization. The regularization parameter λ controls the influence of the regularization term on the inversion process.

The inversion of the synthetic first break traveltimes considered here resolves the subsurface model presented in Figure 4d. To aid in the evaluation of this inversion result we superimposed the known interfaces between the different subsurface unit of the synthetic model. As can be seen from this plot, the imaging result resolves the fundamental structural features and reflects the P-wave velocity distribution of the synthetic subsurface model. Deviations from the true velocity model are due to the smoothness-constraint inversion scheme applied by pyGIMLi. To obtain sharper contrasts between the different subsurface units in the imaging result structural constraints could be incorporated in

the inversion, e.g., as demonstrated by (Steiner et al., 2021). Nonetheless, we consider Figure 4 to demonstrate the applicability of the `SeismicWaveformModeler` class for generating synthetic seismic waveform data to be used in numerical P-wave refraction seismic investigations.

2.2. Processing of seismic refraction datasets

The SR method is based on the measurement of the traveltimes of seismic waves determined from the the first onset of the seismic energy recorded by geophones. In the SRT, the inversion of traveltimes gathered from tens to hundreds of seismograms permits the computation of variations in the seismic velocities in an imaging framework. Measuring the traveltimes in seismograms (i.e., first break picking) is commonly done manually (or semi-automatically) in an iterative process. The signal-to-noise ratio in the recorded seismograms substantially influences the quality of the traveltimes picked in the seismograms, and thus a proper enhancement of the perceptibility of the first onsets is crucial. Accordingly, the `SeismicRefractionManager` class provides functionalities that permit the processing of seismic waveform data for a refraction seismic analysis. These functionalities involve the reading of seismic waveform data, combining the data with information about the survey geometry, processing of the waveforms as well as the picking of first break traveltimes. Since the first break picking is a highly interactive process, the `SeismicRefractionManager` is designed primarily for usage from within an ipython shell.

For the demonstration of the fundamental `SeismicRefractionManager` capabilities we consider the synthetic seismic data created with the `SeismicWaveformModeler` above. This allows us to show that both classes can be combined in subsequent workflows, and to present the forward modeled seismic waveform data, the synthetic first break traveltimes as well as the seismic P-wave velocity model obtained through the inversion of these traveltimes.

2.2.1. Compiling the survey information and creating a project

To create a new or load an existing `SeismicRefractionManager` project, the working directory needs to contain specific subdirectories:

```
working_directory
├── 01_data
│   └── raw
├── 02_geom
└── 03_proc
```

In this directory structure, the seismic shot files are stored in *01_data/raw* and the geometry file (*geometry.csv*) is provided in *02_geom*. The geometry file is a csv file that provides an abstract representation of the survey layout and must not contain a header. The fundamental element for the description of the survey layout is the station, which refers either to a geophone position, a shot position or a position with co-located shot and geophone. For each station the geometric and semantic information described in Table 3 are provided column-wise, i.e., each line in the geometry file corresponds to a single station with a unique position within the survey layout.

Table 3
Description of the information to be provided in the columns of the geometry file.

Col	Content	Data type	Description
1	x coordinate	float	Station x coordinate, e.g., given in (m)
2	y coordinate	float	Station y coordinate, e.g., given in (m)
3	z coordinate	float	Station z coordinate, e.g., given in (m)
4	Geophone	bool	1 if a geophone was deployed at station, 0 otherwise
5	Shot	int	The numerical part of the file name, e.g., 1001, if a shot was conducted at this station, -1 otherwise
6	First geophone	int	First active geophone (-1 if no shot was conducted at the station)
7	Number of geophones	int	Number of active geophones (-1 if no shot was conducted at the station)

An instance of the `SeismicRefractionManager` can be created by providing the path to the working directory as parameter to the constructor. Based on the content of the working directory, the `SeismicRefractionManager` automatically decides whether (i) to start in the data preview mode (first break picking not possible), (ii) create a new project, or (iii) load an existing project from disk. In case the shot files as well as the geometry file are provided and a basic sanity check of the geometry file was successful, the `SeismicRefractionManager` creates a new project:

```
# Import the SeismicRefractionManager from the formikoj library
from formikoj import SeismicRefractionManager

# Create an instance of the SeismicRefractionManager
srm = SeismicRefractionManager('.')
```

```
INFO    : Read geometry information from file
INFO    : Extracted shot geometry
INFO    : Extracted receiver geometry
INFO    : Applied geometry
INFO    : Standard pickset 'picks' created
INFO    : Pickset 'picks' loaded
INFO    : 'picks' set as active pickset
Progress <==============================> 100.0% completed
INFO    : Read 48 files
```

In a first step, the `SeismicRefractionManager` creates an SQLite database *prj.db* in the working directory based on the entity-relationship diagram shown in Figure 5. The geometry information is then read from the geometry file and stored in the database table *geometry* with consecutively numbered stations (see Figure 6). To allow for an efficient data selection for the user the `SeismicRefractionManager` creates database tables *shots* an d*receivers*, which link the station numbers to shot index numbers (SIN) and receiver index numbers (RIN), respectively. For each shot-receiver pair the corresponding SIN and RIN are stored in the table *applied_geometry* together with the absolute offset and midpoint between these stations, i.e., the geometry is applied. In the last step, the database table *fbpicks* is created,

which stores the first break traveltimes for each SIN-RIN pair together with the name of the corresponding pickset, i.e., a common label for an entire set of first break traveltimes. By default, each project contains the default pickset 'picks', which is loaded and activated on startup. Once the database is initialized, the waveform data are read from disk and the project is ready for processing.

2.2.2. Selecting and visualizing seismic waveform data

Once the geometry is applied the `select` method of the `SeismicRefractionManager` allows to gather the seismic waveform data based on a common absolute offset (`aoffset`)

```
# Select traces with common absolute offset
srm.select(by='aoffset', num=6)
```

```
INFO    : 90 traces selected
```

a common RIN (`rin`),

```
# Select traces with receiver
srm.select(by='rin', num=10)
```

```
INFO    : 24 traces selected
```

or a common SIN (`sin`)

```
# Select traces with receiver
srm.select(by='sin', num=24)
```

```
INFO    : 24 traces selected
```

The selected traces can be visualized by calling the `plot` method without passing any parameter:

```
srm.plot()
```

Once opened, the seismogram plot visualizes the seismic waveforms in a combination of wiggle trace and variable area mode, i.e., the trace data are shown as curves. The area of the curves is colored red for negative and blue for positive amplitudes, respectively (not shown for brevity). Pressing the up or down arrow key on the keyboard toggles the visualization mode between variable area and variable density. In variable density mode the strength of the amplitudes is additionally reflected by the color saturation, i.e., high amplitudes refer to a stronger shade than low amplitudes (see Figure 7).

The active processing mode and data scaling mode are reported together with the traveltime at the current cursor position in the status bar of the seismogram plot window (see Figure 8). The initial processing mode is 'Fb pick', i.e., first break picking is possible. The user can switch between the different modes by pressing specific keys on the

keyboard. The 'm' key activates the trace mute mode ('Trc mute'), which allows to set the amplitude of a trace to zero by clicking with the left mouse button; clicking again on the same trace restores the amplitude information. The trace reverse mode ('Trc rev') is activated by pressing the 'r' key and enables the user to toggle the polarity of a trace by clicking on it with the left mouse button. The default data scaling mode is 'Zoom', which allows the scaling of the y-axis by turning the mouse wheel. By pressing the 'a' key the amplitude scaling mode ('Amp scal') is activated. Turning the mouse wheel increases or decreases the amplitudes of the traces currently shown in the seismogram plot, and thus might help to enhance the perceptibility of the first onsets. By pressing the key of the currently active mode again, the `SeismicRefractionManager` returns to the default mode; yet, the different modes can be activated in any arbitrary order (as illustrated in Figure 8).

Through the `plot` method the frequency spectrum of the currently selected trace data can be visualized:

```
srm.plot('spectrum')
```

A frequency spectrum as shown in Figure 9 can be used to discriminate the dominating signal frequencies from those associated to the background noise, and thus allows for the definition of adequate filter settings. To improve the signal-to-noise ratio it is possible to apply filters on the selected traces through the `filter` method, which utilizes the frequency filters implemented in the ObsPy package (lowpass, highpass, bandpass and bandstop; Beyreuther et al., 2010), e.g.:

```
srm.filter('lp 120')
```

```
INFO    : Applied 120.0 Hz lowpass filter
```

```
srm.filter('bp 10 120')
```

```
INFO    : Applied bandpass filter (10.0 to 120.0 Hz)
```

```
srm.filter('hold on')
```

```
INFO    : Set filter hold on
```

By default, filters are solely applied to the currently selected traces, yet setting the filter on hold allows the filtering of all subsequently selected traces with the same filter settings. In case the seismogram plot is opened, the effect of the applied filter on the seismic waveforms is interactively visualized.

2.2.3. Analysis of the seismic waveforms and first break traveltime picking

In the seismogram plot, the waveforms can be analyzed and processed to obtain information about the subsurface conditions. Activating the velocity estimation mode ('Vel est') by pressing the 'v' key on the keyboard, enables the user to estimate velocities for different wave phases (e.g., originating from a refractor) by pressing the left mouse button and moving the cursor. Once the left mouse button is released, a line between the start and end point is drawn and labeled with the corresponding velocity (estimates can be deleted by clicking with the right mouse button).

If the processing mode 'Fb pick' is activated, picking of first break traveltimes is done individually by clicking

with the left mouse button on the respective trace. Clicking again on the same trace will set the first break pick to the new location as there can only be one traveltime for each SIN-RIN pair; whereas clicking with the right mouse button deletes the pick. Alternatively, first break picks can be set for multiple traces by pressing the left mouse button and moving the cursor. Once the left mouse button is released, first break picks are defined at the intersections between the line and the seismograms. In the same way, multiple picks can be deleted if a line is drawn with the right mouse button pressed. The first break traveltimes determined in the seismogram plot are automatically written to the project database when the window is closed or another set of traces is loaded by pressing the 'left' or 'right' arrow keys on the keyboard.

The traveltime diagram for the currently active pickset can be created through the `plot` method:

```
srm.plot('traveltimes')
```

Figure 10 presents an exemplary traveltime diagram, which is a common way to examine the quality of the first break picking. Such illustration of the traveltimes can be used to identify outliers or erroneous measurements, which are commonly associated to traveltimes with substantial deviations from those observed at adjacent stations such as the first break pick for the SIN-RIN pair (23, 11). Outliers can be removed by clicking on the corresponding symbol ('x') in the traveltime diagram, which is instantly synchronized with the project database. If the seismogram plot and the traveltime diagram are used side-by-side, changes made to the first break picks in one window will interactively trigger an update of the other one and vice versa.

The `SeismicRefractionManager` handles first break picks in picksets, which can be organized and manipulated through the `picksets` method of the `SeismicRefractionManager`:

```
srm.picksets()
```

```
pickset         loaded          active
-----------------------------------------
picks           Y               Y
```

```
srm.picksets('import picking_part1.pck pck_p1')
```

```
INFO    : Created new pickset 'pck_p1'
INFO    : Pickset 'pck_p1' loaded
INFO    : 'pck_p1' set as active pickset
INFO    : Imported 'picking_part1.pck' to pickset 'pck_p1'
```

```
srm.picksets('copy pck_p1 pck_all')
```

```
INFO    : Created new pickset 'pck_all'
INFO    : Pickset 'pck_all' loaded
INFO    : 'pck_all' set as active pickset
INFO    : Copied pickset 'pck_p1' to 'pck_all'
```

```
srm.picksets('unload pck_p1 picks')
```

```
INFO    : Pickset 'pck_p1' removed from workflow
INFO    : Pickset 'picks' removed from workflow
```

```
srm.picksets('delete pck_p1')
```

```
INFO    : Pickset 'pck_p1' deleted
```

```
srm.picksets()
```

```
pickset         loaded          active
-----------------------------------------
picks           N               N
pck_all         Y               Y
```

Calling the `pickset` method without parameters shows the status of all picksets in the project. From the above use case, we see that by default, the pickset 'picks' is loaded and activated, i.e., modifications of first breaks are stored in this pickset. First break picks provided by another source (as udf file) can be imported from *03_proc/picks*. For the first break picking, it is sufficient to keep only one pickset in the workflow, i.e., not required picksets can be unloaded. A pickset currently not loaded can be deleted permanently, i.e., the corresponding traveltimes are removed from the database. The `picksets` method can also be used to load picksets from the database:

```
srm.picksets('load picks')
```

```
INFO    : Pickset 'picks' loaded
```

```
srm.picksets()
```

```
pickset         loaded          active
-----------------------------------------
picks           Y               N
pck_all         Y               Y
```

```
srm.picksets('use picks')
```

```
INFO    : Pickset 'picks' loaded
INFO    : 'picks' set as active pickset
```

```
srm.picksets()
```

```
pickset         loaded          active
-----------------------------------------
picks           Y               Y
pck_all         Y               N
```

When a pickset is loaded from the database, it does not become the active pickset automatically; whereas by using the parameter `use` the corresponding pickset is loaded and also becomes the active pickset. The first break traveltimes of a pickset can be exported to an udf file that is stored in *03_proc/picks* subdirectory with the current timestamp as suffix:

```
srm.picksets('export pck_all')
```

```
INFO    : pickset 'pck_all' saved to pck_all_20220428T145648.pck
```

3. Application to field data: Processing a 3D seismic refraction dataset

The seismic data used in this example were collected at the Danube island (Vienna) in June 2021, using 48 geophones deployed with 2 m spacing between them and shot locations located between the geophone positions. As illustrated in Figure 11, the survey geometry refers to a roll-along geometry, i.e., the geophone spread was moved along a profile with 50% overlap yielding a total of five segments. The objective of the survey was to define the contact between different sediments within the tertiary and quarternary deposits used to build the man-made Danube island. Additionally, the survey aimed to identify lateral changes that might indicate the position of a fault, which has been inferred from sediments recovered from drillings.

In the field, each segment was measured separately, yet for the processing all measurements are combined in a single profile. We can implement such measurement layout in a geometry file as illustrated in Table 4. The key parameter is '1st Geo' as it indicates the first active geophone along the profile for each shot file. For the first segment in a roll-along survey geometry, the first active geophone is always geophone 1. Considering the number of geophones used in each segment, and an overlap of 50%, the first geophone for segments two to five are 25, 49, 73 and 97, respectively.

Based on the shot files and the geometry file stored in the required directory structure the `SeismicRefractionManager` creates the project database. A first illustration of the subsurface conditions can be obtained by computing a common

Table 4
Extract from the roll-along survey geometry file showing how the information regarding the first geophone assigns the traces in the shot files to the correct stations.

x (m)	y (m)	z (m)	Geo	Shot	1st Geo	# Geo
0.0	0.0	163.5	1	-1	-1	-1
2.0	0.0	163.5	0	1001	**1**	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
94.0	0	163.5	0	1024	**1**	48
96.0	0	163.5	1	-1	-1	-1
98.0	0.0	163.5	0	1037	**25**	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
194.0	0.0	163.5	0	1049	**25**	48
196.0	0.0	163.5	1	-1	-1	-1
198.0	0.0	163.5	0	1073	**49**	48
200.0	0.0	163.5	1	-1	-1	-1
202.0	0.0	163.5	0	1050	**25**	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
286.0	0.0	163.5	0	1084	**49**	48
288.0	0.0	163.5	1	-1	-1	-1
290.0	0.0	163.5	0	1097	**73**	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
386.0	0.0	163.5	0	1109	**73**	48
388.0	0.0	163.5	1	-1	-1	-1
390.0	0.0	163.5	0	2025	**97**	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
570.0	0.0	163.5	0	2048	**97**	48
572.0	0.0	163.5	1	-1	-1	-1

offset stack (COS):

```
srm.compute('cos')
```

```
Progress <==============================> 100.0% completed
INFO    : Computed the common offset stack
```

The COS for the Danube island dataset presented in Figure 12 shows first onsets for absolute offsets up to approximately 150 m, which suggest a two-layered subsurface model. By using the velocity estimation functionality we can approximate the seismic velocity in the corresponding layers.

For the first break picking a set of traces is selected, filtered if necessary and visualized:

```
srm.select('sin 99')
```

```
INFO    : 48 traces selected
```

```
srm.filter('lp 120')
```

```
INFO    : Applied 120.0 Hz lowpass filter
```

```
srm.plot()
```

In this example, the trace data for SIN 99 are used, which refers to a shot position located in segment four. As can be seen in Figure 13, the first onsets are easily identifiable despite the substantial seismic background noise at large offsets (RIN 73 to 90). A pseudosection provides an illustration of the corresponding apparent seismic velocity values

computed from the picked traveltimes and the distance between shots and geophones:

```
srm.plot('pseudosection')
```

In a pseudosection, as presented in Figure 14 for the Danube island dataset, the apparent velocity values are assigned to pseudolocations, with the corresponding x- and z-coordinates determined as the midpoint and as $1/3$ of the absolute offset between the shot and geophone, respectively. Accordingly, such plot allows for the identification of outliers in the data, e.g., stark velocity contrasts for adjacent points, or systematic errors, e.g., velocities erroneously influenced by a single shot or receiver. The main assumption here is that the pseudosection should reveal smooth transitions between lateral and vertical neighbors, considering that the data were collected with gradual changes in the position of the source (i.e., hammer blow) and the receiver (i.e., geophone). The pseudosection will reveal large variations in case of abrupt changes in the topography or the geometry of the array, yet this can be taken into account by the user during the identification of outliers and possible systematic errors. For the Danube island dataset, the pseudosection suggests an increase in the seismic velocity along profile direction in deeper subsurface regions (i.e., larger pseudodepth). Such pattern could be related to a fault expected in this area of the Danube island, thus indicating that the geophysical survey was sufficiently designed to detect such feature. Moreover, the pseudosection presented in Figure 14 shows a low number of data points in the first segment of the Danube island survey. This lack of data points at large pseudodepths is due to a low number of picked traveltimes at large offsets, and thus might indicate a low signal-noise ratio.

To review the data quality along the entire profile it is possible to visualize the picking percentage, i.e., the ratio of actually picked traveltimes and total number of SIN-RIN pairs:

```
srm.plot('pickperc')
```

Figure 15 shows that the picking percentage is visualized separately for each SIN. In this way, a low picking percentage indicates shots affected by a low signal-to-noise ratio. For the Danube island dataset, the picking percentage is low in the first segment, which confirms the lack of datapoints observed in the pseudosection. Moreover, the picking percentage plot can be used to track the picking progress, for instance if the traveltimes cannot be determined in one session or to identify single shots that might have been forgotten during the first break picking. Accordingly, it is advisable to check the picking percentage prior to exporting the traveltimes for the inversion.

The `SeismicRefractionManager` allows also the visualization and processing of data collected with a 3D survey layout. To illustrate these capabilities, we present in Figure 16 the geometry of a 3D survey conducted in a soda lake located close to Vienna. The soda lake corresponds to quarternary sediments where capillary forces have developed a low permeable layer close to the surface (between 50 and 100 cm) with a high clay and salt content. The seismic survey aims to support the interpretation of the electrical and electromagnetic models obtained in a monitoring framework. Accordingly, the survey geometry shown in Figure 16 was specified by previously conducted electrical measurements

with electrodes arranged along two perpendicular lines. The seismic data were collected with 48 geophones deployed along the North-East to South-West oriented line, and 48 geophones along the North-West to South-East oriented line, with a spacing of 2 m between the geophones. Shots were generated with an 8 kg sledgehammer at the geophone positions as well as at positions along the diagonals.

The geometry file contains the 3D coordinates for each shot or receiver station, and thus the `SeismicRefractionManager` automatically configures the project for 3D processing. Figure 17 presents the seismic waveform data recorded for SIN 1, i.e., the shot position co-located with the first geophone (Station 01 in Figure 16). The data for RIN 1 to 48 appear familiar as the corresponding SIN-RIN layout refers to the one of conventional 2D profiles; whereas the seismic waveforms for RIN 49 to 96 show an entirely different pattern. To understand such visualization, we need to take into account that RIN 49 to 96 are deployed perpendicular to the direction of propagation of the wavefront originating from SIN 1. Accordingly, the observed curvature in the first onsets is due to the varying offset of the different SIN-RIN pairs.

Due to the 3D survey geometry, a 2D pseudosection is not suitable to illustrate the apparent velocity values obtained from the picked first break traveltimes. Hence, the `SeismicRefractionManager` automatically switches to a 3D representation where the apparent velocity values are illustrated in an interactive 3D pseudosection. This plot can be rotated and the image section can be zoomed and panned allowing the user to easily investigate the data quality for 3D geometries. Figure 18 shows a screenshot of the 3D pseudosection for the salt lake dataset; yet, such screenshot cannot reveal the full capabilities implemented in the `SeismicRefractionManger` for the interactive analysis and visualization of 3D pseudosections.

Once the first break picking is finished, the corresponding pickset can be exported for the inversion. The inversion results and their interpretation are not the scope of this manuscript, yet Figures 17 and 18 reveal the capabilities provided by the proposed framework for the visualization and processing of data collected in 3D survey geometries.

4. Conclusions and Outlook

We have presented formikoj, a flexible open-source library enabling the development of modeling and processing tools for geophysical data. Implemented in python and tested on all major operating systems (Linux/Unix, MacOS, Windows), formikoj is suitable for multi- and cross-platform applications; thus, allowing collaboration between users free from licensing costs and platform requirements.

We demonstrated the capabilities of the formikoj framework to develop versatile and easily scalable classes for the modeling and processing of waveforms in seismic refraction surveys. The required interaction with the user is reduced to a minimum as crucial processing steps are automatized within the `SeismicWaveformModeler` and `SeismicRefractionManager` based on efficient data input strategies, for instance the preparation and import of the

geometry file or the keyboard-based interaction related to the first break picking. In this regard, the user controls the formikoj library by providing text-based commands preferably through an ipython shell to exploit the full interactive potential modeling and processing tools. However, applications of the formikoj library can also be automatized by implementing workflows in python scripts or jupyter notebooks.

Based on three exemplary use cases, we illustrated the applicability of both the `SeismicWaveformModeler` and the `SeismicRefractionManager`. In the first use case, we showed the possibility to forward model seismic waveform data based on custom subsurface models and survey geometries with the `SeismicWaveformModeler`. Additionally, the resulting waveforms can be subjected to systematic and random noise sources. The capabilities of the `SeismicRefractionManager` were demonstrated through the processing of field datasets collected in complex survey layout, namely a roll-along and a 3D geometry. Moreover, we showed how the different data visualization options can assist during the data processing to ensure consistency in the first break traveltimes. In particular, we developed a visualization of the traveltimes by means of pseudosections illustrating the corresponding apparent seismic velocities. Such plots allow for a quick identification of systematic errors and outliers in both 2D and 3D datasets.

By making the source code of the formikoj library available under the MIT license we intend to spark the development of further modeling and processing tools for various geophysical models based on this framework. Our further efforts will focus on implementing tools for other wave-based geophysical methods used in frame of our research activities, such as multi-channel analysis of (seismic) surface waves or magnetotelluric surveys.

5. Acknowledgments

The authors are grateful to Nathalie Roser and Lukas Aigner for benchmarking the formikoj library against established software packages in frame of their research activities. Furthermore, we would like to thank Clemens Moser, Martin Mayr, Vinzenz Schichl and Harald Pammer for their constructive comments during first tests of the formikoj framework as well as for their help during the seismic surveys.

Code and data availability section

Name of the code/library: formikoj

Contact: matthias.steiner@geo.tuwien.ac.at

Hardware requirements: No specific requirements

Program language: Python

Software required: Anaconda/Miniconda recommended

Total program and dataset size: 250 MB

The source codes and exemplary data sets are available for downloading at the link:

https://git.geo.tuwien.ac.at/msteine1/formikoj.git

The orthophotos used in Figures 11 and 16 were published by geoland.at under a CC BY 4.0 license.

A. Source code for generating the subsurface model considered in this study

```
# Import required packages
import numpy as np
import pygimli as pg
import pygimli.meshtools as mt

# Create the model geometry
# - top layer
top = mt.createPolygon([[0, 0], [94, 0],
                        [94, -3.5], [72, -3.5],
                        [20, -2], [0, -2]],
                       isClosed=True, marker=1, area=0.1)

# - bottom layer
bottom = mt.createPolygon([[0, -2], [20, -2],
                           [22, -6], [70, -6],
                           [72, -3.5], [94, -3.5],
                           [94, -10], [0, -10]],
                          isClosed=True, marker=3, area=0.1)

# - anomaly/infill
infill = mt.createPolygon([[20, -2], [72, -3.5],
                           [70, -6], [22, -6]],
                          isClosed=True, marker=2, area=0.1)

geom = top + infill + bottom
```

```
# Define shot and receiver stations and create corresponding nodes
nstats = 48
spc = 2

stations = np.vstack((np.linspace(0, (nstats-1)*spc, nstats),
                      np.zeros(nstats))).T

for p in stations:
    geom.createNode(p)

# Create mesh for the finite element modeling
mesh = mt.createMesh(geom, quality=34)

# Save the mesh in the binary mesh format for later use
# with the SeismicWaveformModeler
mesh.save('mesh.bms')
```

References

Ahern, T., Casey, R., Barnes, D., Benson, R., Knight, T., Trabant, C., 2012. SEED Reference Manual, Version 2.4. URL: http://www.fdsn.org/pdf/SEEDManual_V2.4.pdf.

Beyreuther, M., Barsch, R., Krischer, L., Megies, T., Behr, Y., Wassermann, J., 2010. Obspy: A python toolbox for seismology. Seismological Research Letters 81, 530–533. doi:10.1785/gssrl.81.3.530.

Blanchy, G., Saneiyan, S., Boyd, J., McLachlan, P., Binley, A., 2020. Resipy, an intuitive open source software for complex geoelectrical inversion/modeling. Computers & Geosciences 137, 104423. URL: https://www.sciencedirect.com/science/article/pii/S0098300419308192, doi:https://doi.org/10.1016/j.cageo.2020.104423.

Bücker, M., Flores Orozco, A., Gallistl, J., Steiner, M., Aigner, L., Hoppenbrock, J., Glebe, R., Morales Barrera, W., Pita de la Paz, C., García García, C.E., et al., 2021. Integrated land and water-borne geophysical surveys shed light on the sudden drying of large karst lakes in southern mexico. Solid Earth 12, 439–461. doi:10.5194/se-12-439-2021.

Cockett, R., Kang, S., Heagy, L.J., Pidlisecky, A., Oldenburg, D.W., 2015. Simpeg: An open source framework for simulation and gradient based parameter estimation in geophysical applications. Computers & Geosciences 85, 142–154. URL: https://www.sciencedirect.com/science/article/pii/S009830041530056X, doi:https://doi.org/10.1016/j.cageo.2015.09.015.

Draebing, D., 2016. Application of refraction seismics in alpine permafrost studies: A review. Earth-Science Reviews 155, 136–152. doi:https://doi.org/10.1016/j.earscirev.2016.02.006.

Guedes, V.J.C.B., Maciel, S.T.R., Rocha, M.P., 2022. Refrapy: A python program for seismic refraction data analysis. Computers & Geosciences 159, 105020. URL: https://www.sciencedirect.com/science/article/pii/S0098300421003022, doi:https://doi.org/10.1016/j.cageo.2021.105020.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern,

R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585, 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2, doi:10.1038/s41586-020-2649-2.

Heimann, S., Kriegerowski, M., Isken, M., Cesca, S., Daout, S., Grigoli, F., Juretzek, C., Megies, T., Nooshiri, N., Steinberg, A., Sudhaus, H., Vasyura-Bathke, H., Willey, T., Dahm, T., 2017. Pyrocko - an open-source seismology toolbox and library doi:10.5880/GFZ.2.1.2017.001.

Hunter, J.D., 2007. Matplotlib: A 2d graphics environment. Computing in Science & Engineering 9, 90–95. doi:10.1109/MCSE.2007.55.

McKinney, W., 2010. Data Structures for Statistical Computing in Python, in: Stéfan van der Walt, Jarrod Millman (Eds.), Proceedings of the 9th Python in Science Conference, pp. 56 – 61. doi:10.25080/Majora-92bf1922-00a.

Nguyen, F., Ghose, R., Isunza Manrique, I., Robert, T., Dumont, G., 2018. Managing past landfills for future site development: A review of the contribution of geophysical methods, in: Proceedings of the 4th International Symposium on Enhanced Landfill Mining, pp. 27–36.

Parsekian, A., Singha, K., Minsley, B.J., Holbrook, W.S., Slater, L., 2015. Multiscale geophysical imaging of the critical zone. Reviews of Geophysics 53, 1–26. doi:10.1002/2014RG000465.

Plattner, A.M., 2020. Gprpy: Open-source ground-penetrating radar processing and visualization software. The Leading Edge 39, 332–337. doi:10.1190/tle39050332.1.

Ringler, A.T., Evans, J.R., 2015. A quick seed tutorial. Seismological Research Letters 86, 1717–1725. doi:10.1785/0220150043.

Romero-Ruiz, A., Linde, N., Keller, T., Or, D., 2018. A review of geophysical methods for soil structure characterization. Reviews of Geophysics 56, 672–697. doi:10.1029/2018RG000611.

Rücker, C., Günther, T., Wagner, F.M., 2017. pygimli: An open-source library for modelling and inversion in geophysics. Computers & Geosciences 109, 106–123. doi:10.1016/j.cageo.2017.07.011.

Steiner, M., Katona, T., Fellner, J., Flores Orozco, A., 2022. Quantitative water content estimation in landfills through joint inversion of seismic refraction and electrical resistivity data considering surface conduction. Waste Management 149, 21–32. URL: https://www.sciencedirect.com/science/article/pii/S0956053X22002793, doi:https://doi.org/10.1016/j.wasman.2022.05.020.

Steiner, M., Wagner, F.M., Maierhofer, T., Schöner, W., Flores Orozco, A., 2021. Improved estimation of ice and water contents in alpine permafrost through constrained petrophysical joint inversion: The hoher sonnblick case study. Geophysics 86, 1–84. doi:10.1190/geo2020-0592.1.

Stockwell, J.W., 1999. The cwp/su: Seismic unix package. Computers & Geosciences 25, 415–419. URL: https://www.sciencedirect.com/science/article/pii/S0098300498001459, doi:10.1016/S0098-3004(98)00145-9.

Sullivan, C.B., Kaszynski, A., 2019. PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK). Journal of Open Source Software 4, 1450. URL: https://doi.org/10.21105/joss.01450, doi:10.21105/joss.01450.

Uieda, L., Oliveira Jr, V.C., Barbosa, V.C., 2013. Modeling the earth with fatiando a terra, in: Proceedings of the 12th Python in Science Conference, pp. 96–103.

List of Figures

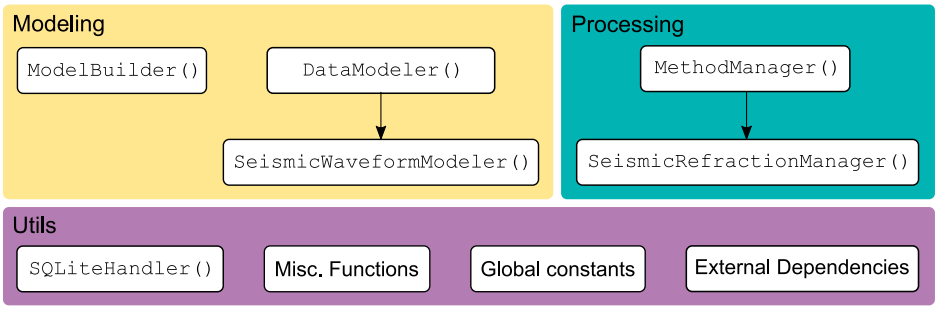

Figure 1: Fundamental use case illustrating the modeling of seismic refraction data within the formikoj ecosystem.

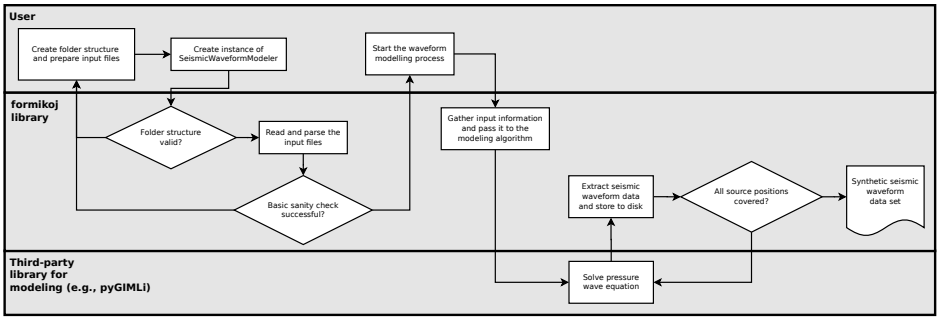

Figure 2: Fundamental use case illustrating the processing of seismic refraction data within the formikoj ecosystem.

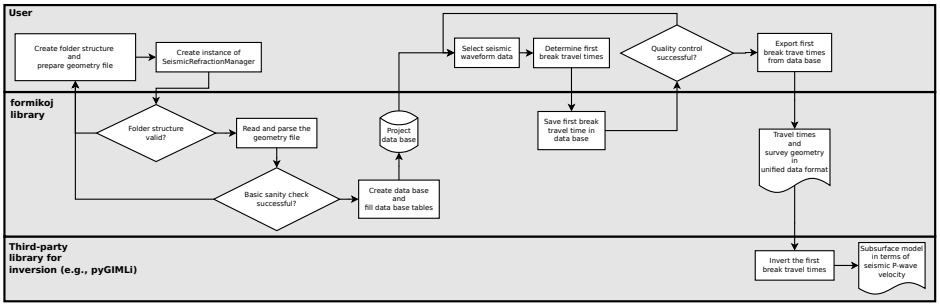

Figure 3: Typical formikoj workflow for the picking of first break traveltimes from seismic waveform data.

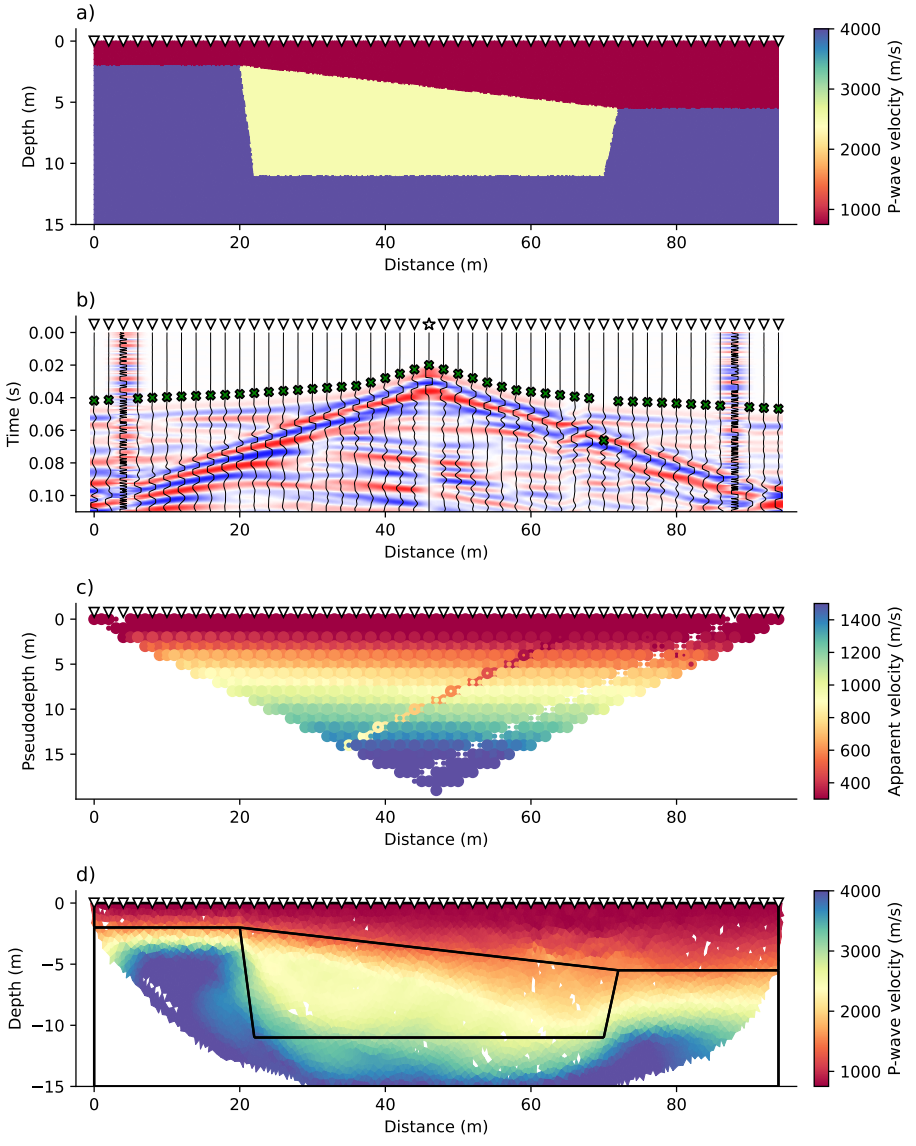

Figure 4: Synthetic seismic data generated with the `SeismicWaveformModeler`:
(a) Two-layered subsurface model without topography characterized by a distinct anomaly in the center. Triangles along the surface indicate the positions of geophones.
(b) Synthetic seismic waveform data computed from the subsurface model for a shot point (star-shaped symbol) located in the center of the profile.
(c) Pseudosection illustrating the apparent velocity computed from the seismic travel times based on the survey geometry.
(d) Subsurface model of the seismic P-wave velocity distribution obtained through the inversion of the synthetic P-wave travel times.

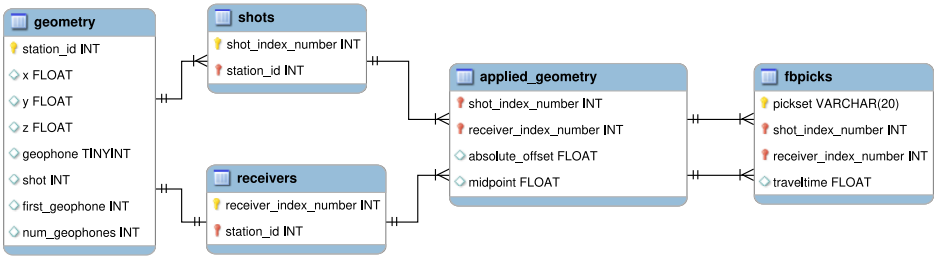

Figure 5: Entity-relationship diagram illustrating the SQLite database used in a `SeismicRefractionManager` project to store an manage processing related information.

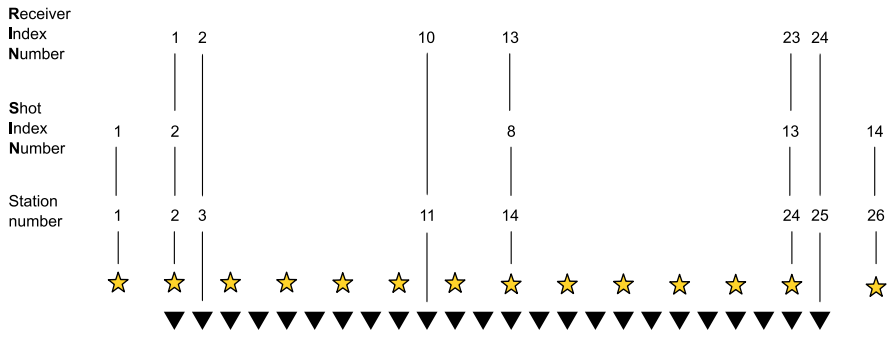

Figure 6: The `SeismicRefractionManager` addresses the stations through consecutive station numbers based on the sort order in the geometry file. The shot index numbers (SIN) and receiver index numbers (RIN) are assigned to the shot and receiver stations separately to allow for an intuitive data handling for the user.

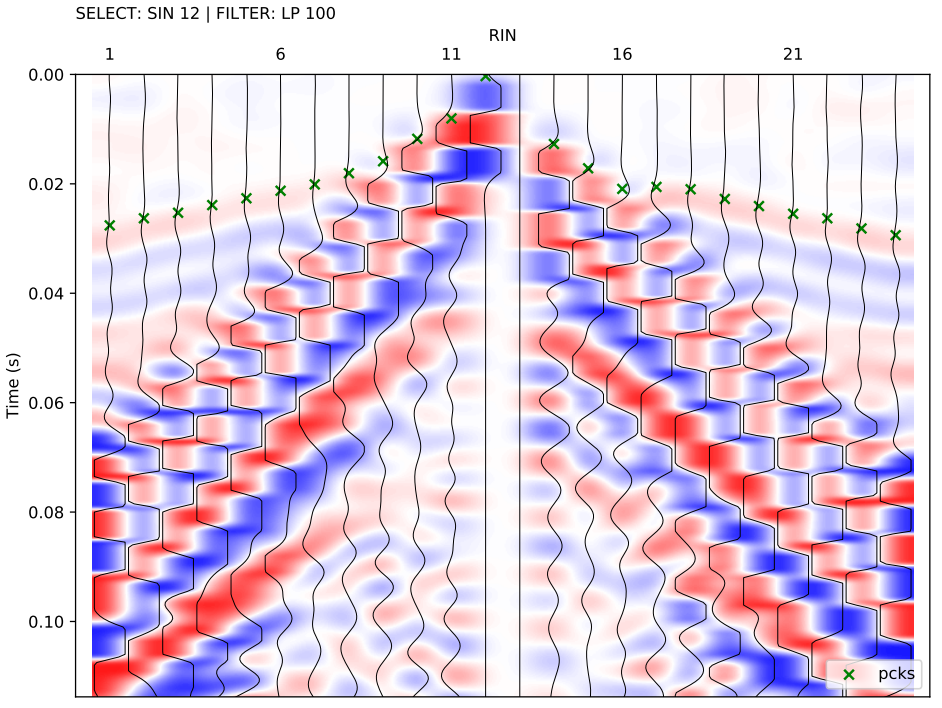

Figure 7: The seismogram plot presents the currently selected traces along the x axis, where the sort order is determined through the geometry. The corresponding trace data are illustrated as a function of time along the y axis (solid curves), with positive and negative amplitudes depicted in blue and red, respectively. The selection criterion and the applied filter are shown in the upper left corner of the plot. Green crosses refer to the picked traveltimes stored in the currently active pickset.

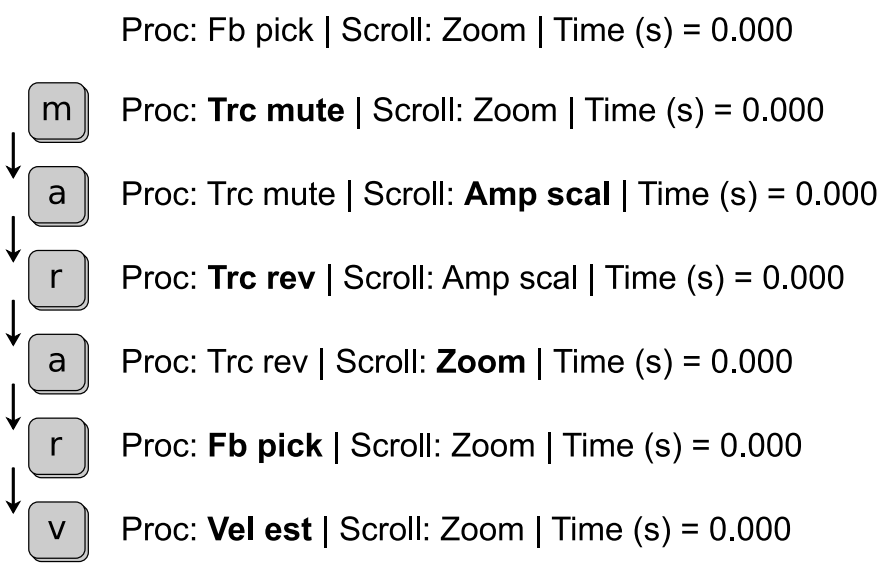

Figure 8: The status bar in the interactive seismogram plot displays the currently active processing and data scaling modes as well as the time (in seconds) at the current cursor position. By pressing the keys 'a', 'm', 'r', 'v' on the keyboard the different modes can be activated.

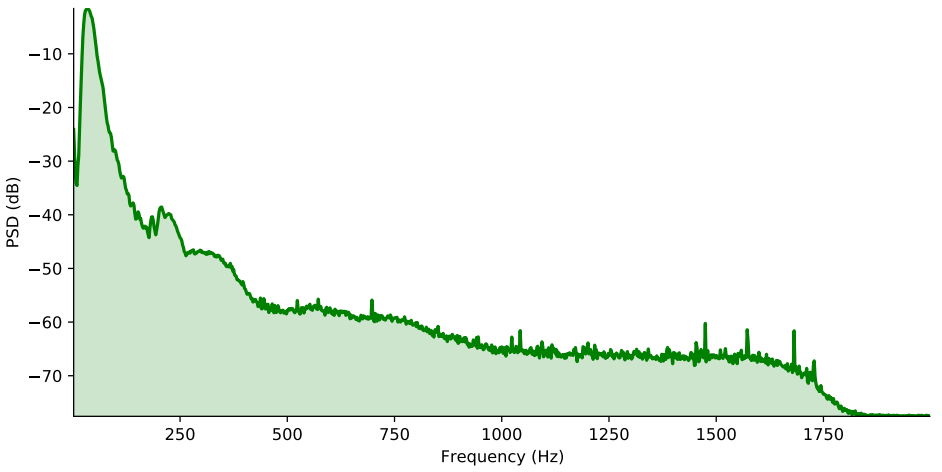

Figure 9: The frequency spectrum illustrates the frequency content of the currently selected traces, which allows for the identification of frequency ranges associated to noise, which can be omitted through the application of corresponding frequency filtering.

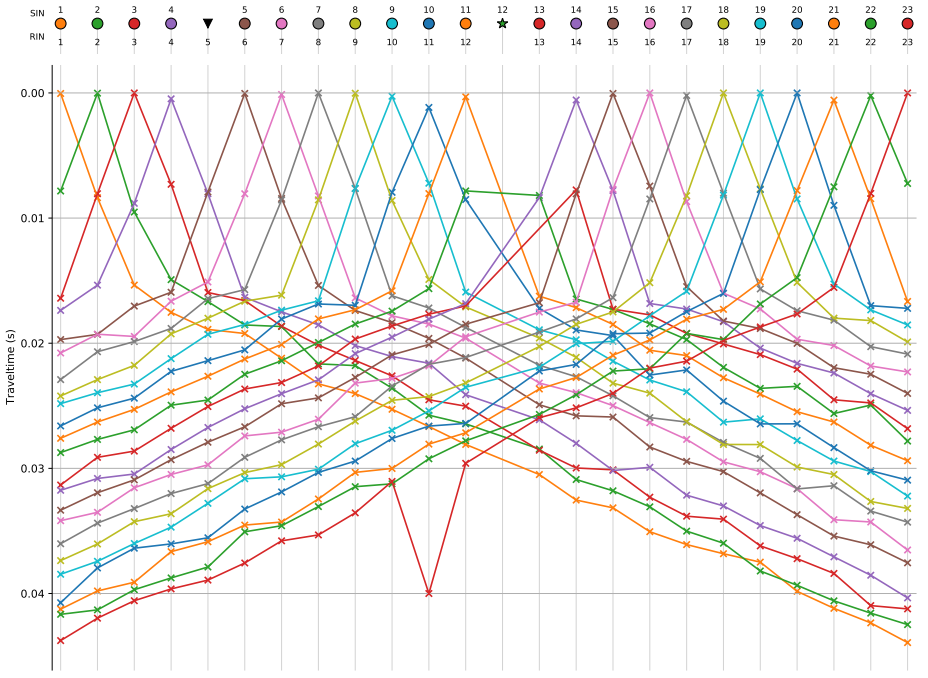

Figure 10: The traveltime diagram shows the first break traveltimes stored in the currently active pickset along the y axis (x symbols), where solid lines connect traveltimes assigned to a common shot. The sort order of the stations along the x axis reflects the geometry. Filled circles indicate stations with co-located shot and geophone (receiver), triangles refer to receiver stations (no shot) and stars refer to shot stations (no geophone).

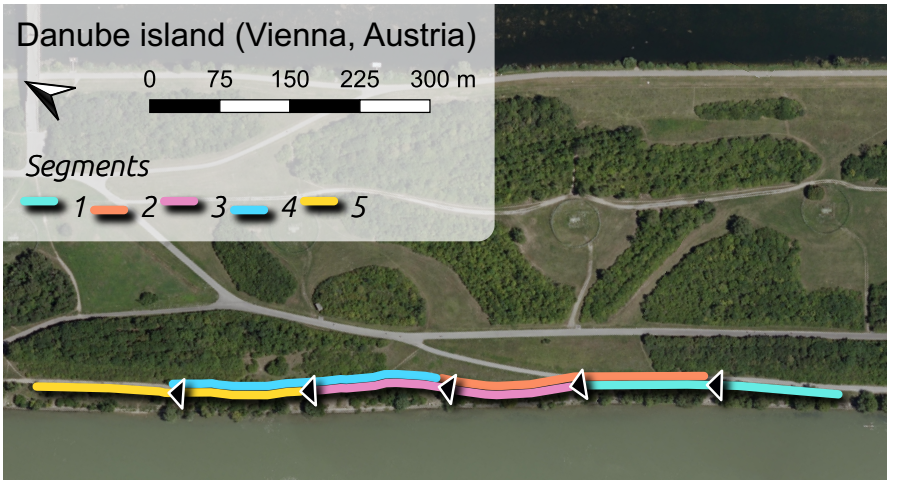

Figure 11: The Danube island field dataset was collected along a single line with a roll-along survey layout; the filled triangles indicate the direction of the measurements. The five segments have an overlap of 50% to ensure an adequate data coverage along the entire profile.

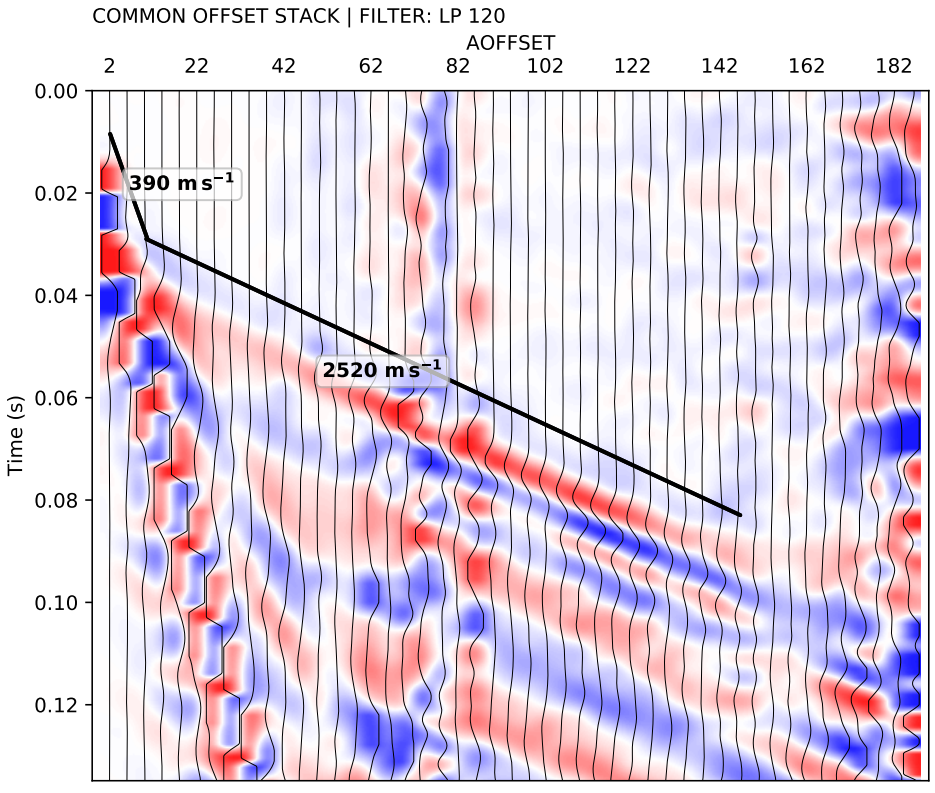

Figure 12: The common offset stack computed for the Danube island dataset clearly showing a two-layered subsurface. The seismic velocities within the layer can be estimated from the gradient of the lines drawn along the corresponding first onsets of the seismic waves.

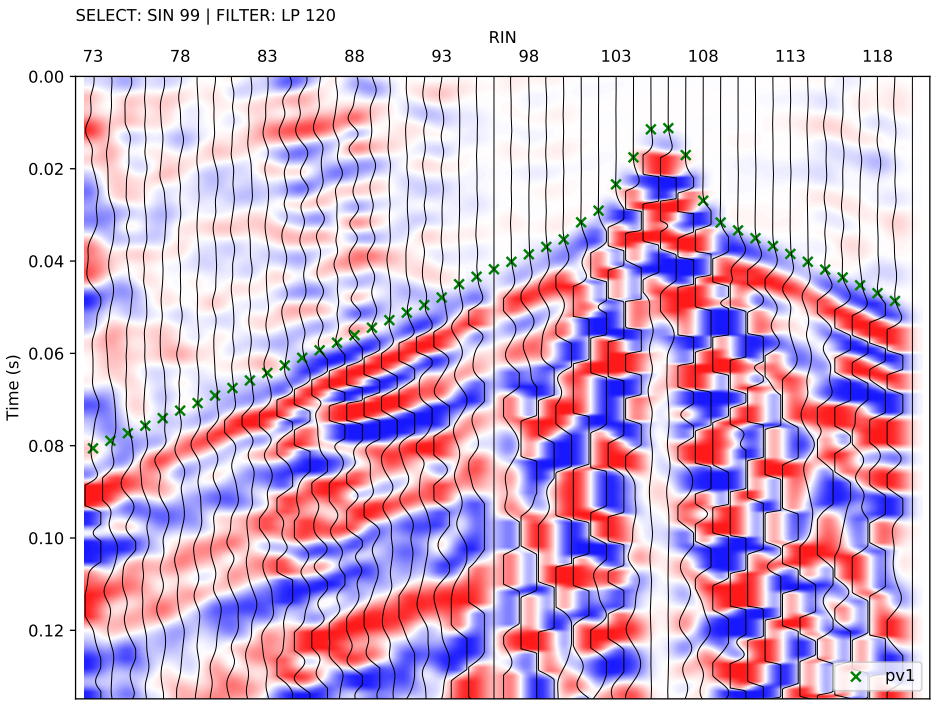

Figure 13: Examplary seismic waveforms from the Danube island dataset shown for shot index number (SIN) 99 with a 120 Hz lowpass filter applied to suppress high frequency noise. The receiver index numbers (RIN) start at 73, thus indicating that the data were collected with a roll-along survey geometry.

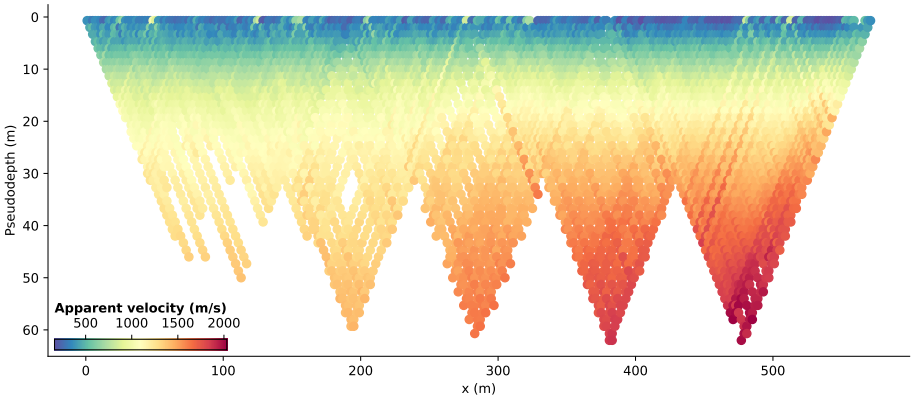

Figure 14: Pseudosection showing the apparent seismic velocities determined from the first break traveltimes obtained from the Danube island dataset and the corresponding absolute offset between the shot and receiver stations. The apparent velocity for each shot-receiver pair is illustrated at the corresponding midpoint and pseudodepth (1/3 of the absolute offset).

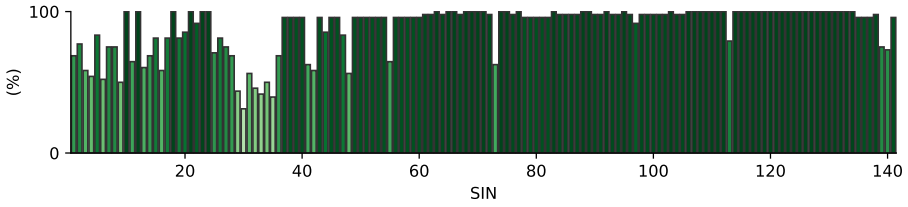

Figure 15: Picking percentage for each shot in the Danube island roll-along dataset. Low values in the first third of the dataset indicate a low signal-to-noise ratio in the seismograms hindering the picking of first break traveltimes for each shot-receiver pair.

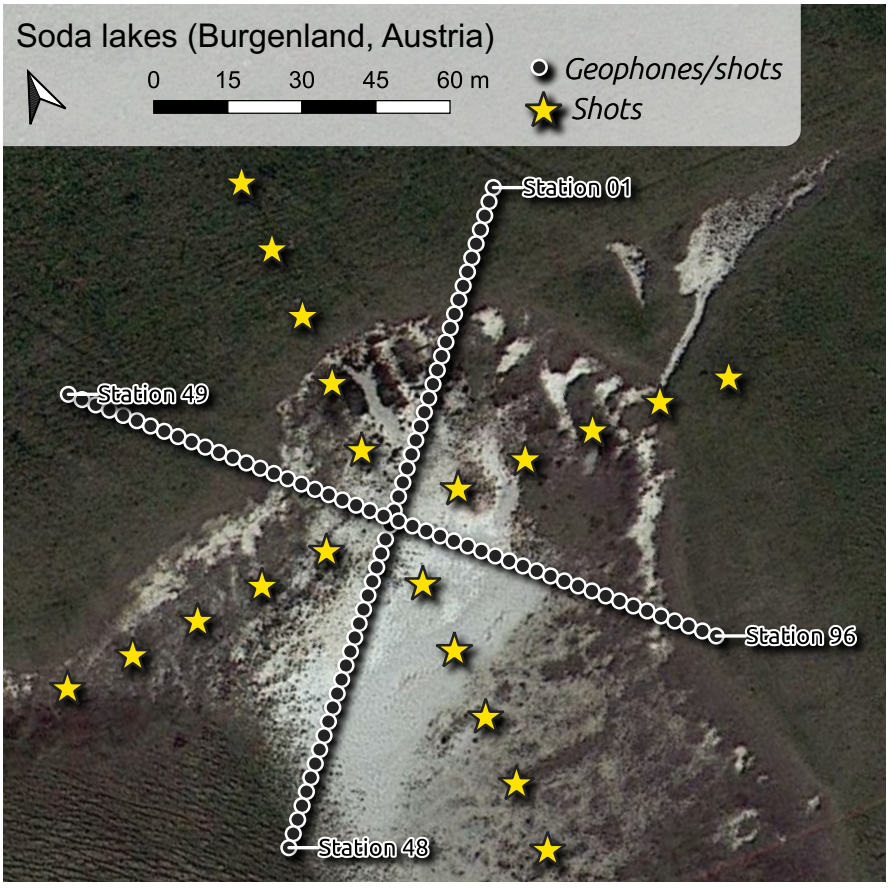

Figure 16: The soda lakes field dataset was collected in a 3D survey layout with stations (co-located geophones and shots indicated by filled dots) deployed in form of a cross. Additional shots (yellow stars) were conducted in from of a cross rotated by 45° to increase the coverage of the dataset.

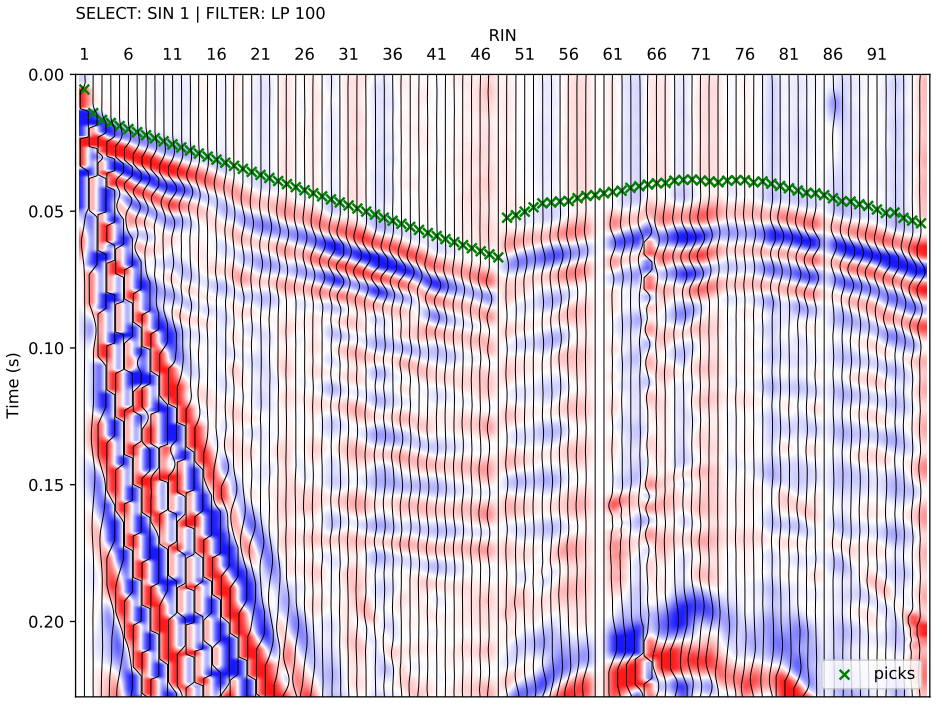

Figure 17: Examplary seismic waveforms from the soda lakes dataset shown for shot index number (SIN) 1 with a 100 Hz lowpass filter applied to suppress high frequency noise. The recorded seismic waveforms clearly reflect the geometry of the geophones with RIN 1 to 48 deployed along the direction of wave propagation, while RIN 49 to 96 are deployed perpendicular to the propagating wavefront.

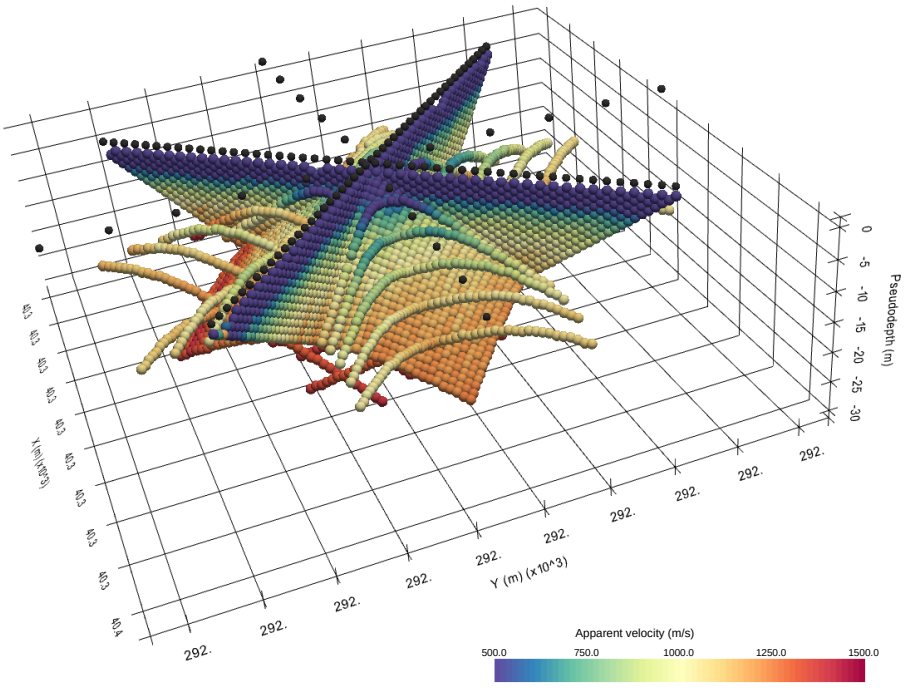

Figure 18: 3D pseudosection showing the apparent seismic velocities determined from the first break traveltimes obtained from the soda lakes dataset and the corresponding absolute offset between the shot and receiver stations. The apparent velocity for each shot-receiver pair is illustrated at the corresponding 2D midpoint and pseudodepth (1/3 of the absolute offset), thus yielding a 3D representation.