Computers and Geosciences

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

--Manuscript Draft--

Manuscript Number:	CAGEO-D-22-00399R1
Article Type:	Research Paper
Section/Category:	Data Processing
Keywords:	geophysical data processing; seismic refraction; seismic waveform modeling; cross-platform application; flexible open-source library; wave based methods
Corresponding Author:	Matthias Steiner TU Wien Research Unit of Geophysics Vienna, AUSTRIA
First Author:	Matthias Steiner
Order of Authors:	Matthias Steiner
	Adrián Flores Orozco
Abstract:	p, li { white-space: pre-wrap; }
Suggested Reviewers:	Leonardo Uieda Leonardo.Uieda@liverpool.ac.uk
	Alain Plattner amplattner@ua.edu
Response to Reviewers:	

[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 Editor-in-Chief, dear reviewers

thank you very much for the positive evaluation of our manuscript as well as for providing numerous constructive comments and suggestions for improvement. In the revised version of our manuscript "formikoj: A flexible library for data management and processing in geophysics - Application for seismic refraction data", we have attempted to implement every suggestion and address all comments.

In particular, we present theoretical details regarding the considered and implement methodologies, provide a self-contained synthetic study demonstrating the validity of the forward modeled seismic waveform data, and the fundamental processing capabilities of the proposed library for seismic refraction data. Moreover, we discuss a single field data application in order to provide a more concise demonstration of the library functionalities with regard to the processing of real seismic survey data.

Moreover, we modified the library according to the suggestions provided by the reviewers and provide the revised source codes in a public repository with details listed in the section "Code availability".

We hope that our revisions and replies alleviate the concerns of the reviewers and that you find our study suitable for publication in Computers & Geosciences. We look forward to your decision.

Yours sincerely,

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

Declaration of interests

☒The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Editor

E:

Similarly to the reviewers, I also liked the effort of making available open-source geophysical data processing tools. However, many packages already exist. What are the main differences between yours and existing packages like pygimli? The reviewers have also other valuable comments that are compulsory to acknowledge for the paper to be publishable in Computers & Geosciences.

Thank you very much for the positive evaluation of our manuscript as well as for providing numerous constructive comments and suggestions for improvement. In the revised version of our manuscript "formikoj: A flexible library for data management and processing in geophysics - Application for seismic refraction data", we have attempted to implement every suggestion and address all comments.

In particular, we present theoretical details regarding the considered and implement methodologies, provide a self-contained synthetic study demonstrating the validity of the forward modeled seismic waveform data, and the fundamental processing capabilities of the proposed library for seismic refraction data. Moreover, we discuss a single field data application in order to provide a more concise demonstration of the library functionalities with regard to the processing of real seismic survey data.

Moreover, we modified the library according to the suggestions provided by the reviewers and provide the revised source codes in a public repository with details listed in the section "Code availability".

We hope that our revisions and replies alleviate the concerns of the reviewers and that you find our study suitable for publication in Computers \& Geosciences. We look forward to your decision.

Reviewer 1

R1.1:

Really like the effort to be cross-platform and open-source. The authors try to fill the gap for data management for pygimli, but it was hard to acknowledge the incremental contribution from the pygimli library. It is also available on all platforms. I was not able to validate differences between the SeismicRefractionManager found in pygimli and the one in the formikoj library. Perphaps, there is some major differences and incremental contributions, but I would suggest that the authors make this clear throughout their revised manuscript. Other than that, a new package can be useful it brings new features and capabilities. I would recommend the authors make this clear before the manuscript is published.

A:

In the revised manuscript, we provide flow charts describing the fundamental workflows associated with the modeling and inversion of seismic refraction waveform data (Figures 2 and 3). These new figures illustrate the differences between the capabilities of the formikoj library and third-party libraries such as pyGIMLi.

In particular, the SeismicRefractionManager presented in our manuscript was developed for the picking of P-wave travel times; i.e., the SeismicRefractionManager provides the input data for the TravelTimeManager in pyGIMLi, which is used for the inversion of P-wave travel times. In the revised manuscript, we address and describe this fundamental differences between these two Manager classes on page 10 lines 329 to 338:

"*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. The first break traveltimes determined with the SeismicRefractionManager represent the input data, e.g., for the TravelTimeManager of pyGIMLi (Rücker et al., 2017). This is particularly relevant as the TravelTimeManager provides an inversion framework for first break traveltimes, yet not a framework for the processing of seismic waveform data. For the demonstration of the fundamental*

SeismicRefractionManager capabilities we consider the synthetic seismic data created with the SeismicWaveformModeler above, which illustrates that both classes can be combined in subsequent workflows."

R1.2:

The library seems to be built soundly with error and log messages. On the other hand, the design and workflow to go from database to result is not clearly outlined. It really shows in the manuscript as the steps are not clearly put in a logical order and equally detailed. This is another major revision I would advise for before publication.

A:

In the revised manuscript, we provide flow charts describing the fundamental workflows associated with the modeling and inversion of seismic refraction waveform data (Figures 2 and 3).

R1.3:

Two study cases are presented. One is in 3D and both are presented with good data visualization. All the figures in the manuscript are of great quality with clear axis and clear captions. I would have liked to see the end result with an inversion or at least some information about this process and how it links to this library. The author mention it is not in the scope, I understand but it would have been nice in a case study.

A:

In the revised manuscript, we present further information about the inversion algorithm implemented in pyGIMLi, which is used in our study to solve for subsurface models expressed in terms of the seismic P-wave velocity. In particular, we present a 2D inversion result obtained for the synthetic seisimic refraction data generated in our manuscript, as well as a 3D inversion result obtained from a 3D field dataset.

R1.4:

Line 67 : I wouldn't list those packages. This is highly subjective and doesn't help the reader.

Line 86: Same comment

A:

We agree with the reviewer that listing specific software packages as provided in Line 67 of the initially submitted manuscript is subjective and of no further benefit for the reader of this manuscript and removed this list in the revised manuscript.
In case of Lines 83 to 92 of the submitted manuscript, we review existing software solutions to demonstrate that the authors are aware of the multitude of seismic processing tools available under commercial and open-source licenses. Moreover, the list intends to illustrate that existing solutions address certain points of criticism postulated by the authors, yet none of the respective packages meets the requirements for establishing transparent, reproducible and cross-platform workflows for both modeling and processing of seismic refraction data.

R1.5:

Line 98-99: This is interesting contribution and should be valued more

In Lines 113 to 124 of the revised manuscript, we provide more information regarding the benefits of building the data management of the formikoj library on the SQLite database engine:
"*The formikoj library provides a well thought out data management concept based on the SQLite database engine that does not require a separate server process 2. In particular, the information for each project, such as survey geometry and first break travel times, are stored in a SQLite database file. Using such an application file format facilitates the cross-platform design of the formikoj library, and provides fast I/O operations through concise SQL queries. The portability of the application file allows for an easy exchange of projects between partners across institutions relying on different IT infrastructures. Accordingly, the formikoj library aims at providing a transparent and customizable framework for the collaborative design of reproducible workflows. The comprehensive usage of clear error and log messages aims at supporting the user throughout the modeling and processing workflows. A meticulous exception handling ensures that the data stored in the SQLite project database is not corrupted in case of erroneous input. Moreover, documenting the user input and the respective responses of the formikoj library with the python logging module provides a timestamped command history that further enhances the transparency and repeatability of the conducted workflows.*"

We present in Figure 5 of the revised manuscript the entity-relationship diagram of the SQLite database used for data managing associated with the processing of the seismic refraction data and provide a description of the respective database tables and processes in Lines 378 to 388:

"*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 dreceivers, 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.*"

R1.6:

Line 108-109: What's the difference with Pygimli. Should be detailed in the manuscript

A:

We consider this comment to be addressed by our reply to comment R1.1.

R1.7:

Line 222: What is the idea behind using string arguments ? I am not sure this is great for accessibility

We agree with the reviewer that string argument are not the most intuitive way for passing parameters. We revised the formikoj library accordingly.

R1.8:

General observations about the ergonomy of the library:

Standardizing data input geometry is a key step and the author propose a good framework. However, I am not sure that making the argument that minimum interaction with the user is a strong point. One could regard this feature as a lack of flexibility.

We rephrased the argument to emphasize that standardizing the geometry input in combination with a thorough sanity check reduces the risk of errors in the data base in Lines 572 to 579 of the revised manuscript:

"*By standardizing the data input in combination with a thorough sanity check aims at reducing the risk of corrupting the information stored in the project database. Moreover, crucial processing steps are automatized within the SeismicWaveformModeler and SeismicRefractionManager facilitated by 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.*"

R1.9:

Processing methods like srm.filter('some string about bp or what not') or srm.plot('traveltimes') should be used with proper arguments. For example, it would be muche preferable to call a filter like: srm.filter(type='bp',lb=10,ub=100). This is a bit longer but more explicit and intuitive.

We agree with the reviewer and modified the formikoj library accordingly to support the more explicit and intuitive way of passing parameters to class methods.

Reviewer 2

R2.1:

The manuscript is a software manual rather than a scientific paper. A scientific paper about a software needs to go further than explaining its functionalities. It must present theoretical details about the physical processes the software aims at simulating, and how they are numerically presented in the software. The submitted manuscript needs to further develop these aspects before it can be considered for publication. What assumptions are used for seismic forward modelling: pressure wave equation, so it is elastic isotropic only for P-waves. It considers 1D variations of velocity only, with constant thickness? These assumptions and limits of the software need to be clearly stated so potential users know what to expect.

A:

In the revised manuscript, we address this comment of the reviewer in different ways.

1. We substantially revised the structure and content of the manuscript to present a scientific paper about the formikoj library instead of a software manual for potential users.
2. We present the theoretical concepts and assumptions behind the forward modeling, processing and inversion capabilities throughout the revised manuscript.
3. We redesigned the synthetic study to demonstrate the forward modeling of seismic refraction waveform data for arbitrary synthetic subsurface models. We use first break traveltimes determined from the forward modeled seismograms to solve for the corresponding seismic P-wave velocity distribution. By comparing the known synthetic subsurface model with the subsurface model resolved through the inversion of the first break traveltimes we demonstrate the validity of the synthetic seismograms forward modeled with the formokoj library.
4. We state that the formikoj library provides a framework for the processing of seismic refraction data, where further algorithms can be implemented by the user, e.g., for the automatic processing of field surveys. The main advantage of our formikoj library is that it permits the design of experiments based on synthetic data sets (forward modeled over any subsurface geometry) that allow for testing processing strategies developed by the user. Such possibilities are not available in other open-source libraries like RefraPy.

R2.2:

Validation of your code to prove its reliability are needed in the manuscript, with analytical solutions for simple cases or benchmarking with existing software for more complex cases. They are found only in the acknowledgment section in the submitted manuscript but should be in the main text. Benefits from formikoj over existing software are not substantial or do not stand out the way they are presented. Your section about forward modeling should better present the potential of your software, by showing figures of results obtained for simple to complex geology. You claim you code is "more" than existing codes because it also does forward modelling in addition to interpretation, but you don't present many results on its specific application.

We redesigned the synthetic study to demonstrate the forward modeling of seismic refraction waveform data for arbitrary synthetic subsurface models. We use first break traveltimes determined from the forward modeled seismograms to solve for the corresponding seismic P-wave velocity distribution. By comparing the known synthetic subsurface model with the subsurface model resolved through the inversion of the first break traveltimes we demonstrate the validity of the synthetic seismograms forward modeled with the formokoj library (Line 141 to 320).

R2.3:

One of your critics of existing geophysical software is that they are not fully tested across multiple platforms. You mentioned in section 2 (lines 114-116) that your code is implemented and tested on Linux but is used on all platforms. Is your code tested on different platforms, or does it just happen to work for the time being? To claim your code to fully multi-platform, you'll need to provide a proof that it works and will be tested on all platforms.

In the revised manuscript, we provide information about the operating systems, i.e., distribution and corresponding version, the formikoj library is primarily developed on. Moreover, we provide this information also for all platforms we routinely and successfully test the formikoj library by conducting the fundamental workflows presented in Figures 2 and 3 of the revised manuscript (Line 137 to 140):

"*The formikoj library is primarily developed on machines running Linux Mint 20 or Kubuntu 22.4, respectively. Testing of the library refers to carrying out the fundamental use cases for modeling and processing of seismic refraction data presented in Figure 2 and 3. To ensure the cross-platform applicability of the formikoj library we test these fundamental use cases also on machines running on Windows 10, Windows 11 as well as macOS versions 12 and 13.*"

R2.4:

The snippets of codes found in the manuscript are not always explained, which makes their comprehension difficult. Snapshots showing the progress of the algorithm doesn't add much to the manuscript and can not be considered as a proof the algorithm gives reliable results.

In the revised manuscript, we present a single workflow for the synthetic study spanning from the forward modeling of the seismograms to the seismic refraction processing. Moreover, we restructured and rephrased the affected manuscript sections and paragraphs, respectively, and provide comments for each code snippet.

R2.5:

You discuss about the add-ons of supplementary functionalities, such as automatic first break picking algorithm. First break picking is a critical path in seismic refraction processing, more details are needed about the algorithm you used for automation.

We agree with the reviewer, that the automatic first break picking needs to be presented in more detail if used as an example in the manuscript. Our intention was to highlight the possibility to expand the functionality of the library based on a common use case, i.e., the automation of the first break picking. However, constraints regarding the length of the manuscript do not allow for a more detailed discussion of the first break picking algorithm, which is also not a main part of the manuscript as it was only intended as an illustrative example.

The revised manuscript provides a new section “Expanding the capabilities of the formikoj library” where we present details about the implemented automatic first break picking algorithm, i.e., the relevant equations as well as assumptions and limitations of the considered approach (Line 473 to 511).

R2.6:

The examples are interesting, but the authors need to build a bigger and better story around them. Conventional processing is presented for the moment, which can be done with free version of commercial software. The use of pseudo-section for QaQc is interesting, it needs to be further developed.

In the revised manuscript, we present examples based on a single site instead of presented two different field data sets. In this way, we intend to make the structure of the manuscript easier to follow and present a single story for both the synthetic as well as the field data application.

We appreciate the positive feedback regarding the use of pseudosections for QaQc. In the revised manuscript, we provide more details about the theory behind these data illustration technique and demonstrate their abilities and limitations based on the presented synthetic subsurface models (Line 287 to 298).

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

[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,~~ Editor-in-Chief, dear reviewers

~~we are submitting our manuscript~~ thank you very much for the positive evaluation of our manuscript as well as for providing numerous constructive comments and suggestions for improvement. In the revised version of 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~~we have attempted to implement every suggestion and address all comments.

~~The manuscript presents the open-source python library formikoj for managing and processing geophysical datacollected 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 dataand 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.~~ In particular, we present theoretical details regarding the considered and implement methodologies, provide a self-contained synthetic study demonstrating the validity of the forward modeled seismic waveform data, and the fundamental processing capabilities of the proposed library for seismic refraction data. Moreover, we discuss a single field data application in order to provide a more concise demonstration of the library functionalities with regard to the processing of real seismic survey data.

~~We provide the~~ Moreover, we modified the library according to the suggestions provided by the reviewers and provide the revised source codes in a public repository with details listed in the section "Code availability".

We hope that our revisions and replies alleviate the concerns of the reviewers and that you find our study suitable for publication in Computers & Geosciences. We look forward to your decision.

Yours sincerely,

Matthias Steiner and Adrián Flores Orozco
Research Unit of 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 of 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~~ for the forward modeling of 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~~ We use these synthetic seismic data set to demonstrate the fundamental seismic refraction processing capabilities of the `SeismicRefractionManager`; thus, illustrating the ability to combine modeling and processing tasks in a single workflow. Based on a 3D field dataset we present the available range of possibilities provided by the formikoj library for the processing of ~~real~~ seismic refraction survey data. In particular, we explore different visualization techniques of the seismic traveltime readings to enhance their consistency prior to the inversion with the third-party library pyGIMLi. The low-level access provided by ~~our~~ the formikoj library aims at ~~giving the users the possibility to implement particular~~ enabling users to implement novel modeling, visualization and processing tools specifically designed for their objectives ~~or~~ as well as 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[1] or ParkSeis [1] for seismic methods, or ReflexW[1] for ground-penetrating radar and seismic methods~~, and even for teaching in developing countries. A common limitation of ~~the aforementioned~~ existing software solutions refers to their specific platform requirements mainly related to the type and version of the operating system; moreover, 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~~

[1] ~~https://www.landmark.solutions/SeisSpace-ProMAX, last accessed on~~

[1] ~~https://www.parkseismic.com/parkseis/, last accessed on~~

[1] ~~https://www.sandmeier-geo.de/reflexw.html, last accessed on~~

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 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 for teaching and interpretation purposes, testing of research hypotheses, evaluating the accuracy of different measurement configurations or inversion strategies.

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[1]), 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~~ SRT method we demonstrate the applicability of the proposed library

[1] http://resistivity.net/bert/data_format.html, last accessed on March 11, 2023

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~~ a carefully designed synthetic study highlighting the capabilities of 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 survey geometry, and (iii) the processing of a~~ for the forward modeling and processing of 2D seismic refraction datasets. Based on a 3D field dataset we illustrate that the formikoj library also allows for the processing of complex survey geometries, where the obtained first break traveltimes can be inverted with third party open-source libraries to solve for 3D ~~SR field data sets~~subsurface models expressed in terms of the seismic P-wave velocity.

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 ~~specific~~ any kind of geophysical methods. The formikoj library provides a well thought out data management concept based on the SQLite database engine that does not require a separate server process [2]. In particular, ~~we present here the `SeismicWaveformModeler` and `SeismicRefractionManager` classes implemented within the formikoj framework, which aim at creating and processing seismic waveform data~~ the information for each project, such as survey geometry and first break traveltimes, are stored in a SQLite database file. Using such an application file format facilitates the cross-platform design of the formikoj library, and provides fast I/O operations through concise SQL queries. The portability of the application file allows for an easy exchange of projects between partners across institutions relying on different IT infrastructures. Accordingly, the formikoj library aims at providing a transparent and customizable framework for the collaborative design of reproducible workflows. The comprehensive usage of clear error and log messages aims at supporting the user throughout the modeling and processing workflows. A meticulous exception handling ensures that the data stored in the SQLite project database is not corrupted in case of erroneous input. Moreover, documenting the user input and the respective responses of the formikoj library with the python logging module provides a timestamped command history that further enhances the transparency and repeatability of the conducted workflows.

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 Figure 3, respectively. ~~Similar to RefraPy,~~ These flow charts illustrate the corresponding workflows as well as the required interactions between the user, the formikoj library and third-party packages. As can be seen from Figure 2 and Figure 3, the formikoj library acts as an interface between the user and more complex functionalities of third-party libraries, such as pyGIMLi for modeling and inversion of

[2] https://www.sqlite.org/index.html, last accessed on March 11, 2023

seismic refraction data. 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. To avoid redundancies in the implementation we built 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 used on all major operating systems, i.e., Linux, MacOS and Windows~~

The formikoj library is primarily developed on machines running Linux Mint 20 or Kubuntu 22.4, respectively. Testing of the library refers to carrying out the fundamental use cases for modeling and processing of seismic refraction data presented in Figure 2 and 3. To ensure the cross-platform applicability of the formikoj library we test these fundamental use cases also on machines running on Windows 10, Windows 11 as well as macOS versions 12 and 13.

~~General architecture of the formikoj library comprising a utility, modeling and processing module. The base classes `DataModeler` and `MethodManager` can be used to build tools for specific geophysical methods, e.g., seismic refraction.~~

2.1. Generation of seismic waveform data for synthetic subsurface models

2.2. ~~Generation of seismic waveform data for synthetic subsurface models: The `SeismicWaveformModeler`~~

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 for P-wave refraction modeling 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 ~~is provided~~ as parameter to the constructor: ~~The working directoryneeds to contain a subdirectory *in*, whereas the output directory *out* will be created automatically:~~

```
# Import the SeismicWaveformModeler from the formikoj library
from formikoj import SeismicWaveformModeler
%DIF <
# Create an instance of the SeismicWaveformModeler
swm = 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

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

~~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 The~~ In the working directory, the required input files ~~need to be~~ are provided via the subdirectory *in*~~of the working directory~~, whereas the modeling output will be stored in the automatically created subdirectory *out*. The key input file is the measurement scheme ~~, which~~ 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` ~~. In the csv format~~ 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 ~~waveforms~~waveform data, the parameters ~~for~~ 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: ~~In the exemplary configurationfile shown above~~

```
wavelet:
    length: 1.024
    frequency: 100
    sampling_rate: 2000
    pretrigger: 0.02
%DIF >
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
%DIF >
dataset:
    number: 1
    names: [syn_data]
    noise: 1
    noise_level: 1e-4
    missing_shots: 1
    broken_geophones: 1
    wrong_polarity: 1
%DIF >
traveltimes:
    noise_relative: 0.
    noise_absolute: 0.
```

In this exemplary configuration, the first block ~~contains information regarding the~~ (`wavelet`) contains the parameterization of the base wavelet, which controls the modeling of the seismic waveforms ~~as described in~~ (see Table 2~~. In the second block~~). The second block (`model`) contains information regarding the synthetic subsurface model. Here, the user can define simple models with all layers considered to be parallel to the surface topography, which is inferred from the station geometry provided in the measurement scheme. The seismic velocity values for the different model regions (`velmap`) and the corresponding thickness of the different geological units (`layers`) have to be explicitly defined in the configuration file. The remaining parameters (`quality`, ~~we~~ `area`, `smooth` and `sec_nodes`) refer to the properties of the mesh that is eventually used for the forward modeling of the seismic waveform data and corresponding first break traveltimes (we refer to the respective pyGIMLi resources[3] 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 ~~and~~ (the number of datasets is automatically determined~~. Alternatively, it is possible to~~), or set the number of datasets to be created and the dataset names are automatically generated with the prefix *dataset_*. ~~Various parameters~~ The remaining parameters provided in the `dataset` block control the random error (`noise` ~~) and systematic errors~~ and `noise_level`) as well as systematic errors (`missing_shots`, `broken_geophones` and `wrong_polarity`) in the modeled seismic waveform data (see Table 2 for a detailed description). The number and position of the shot and geophone stations affected by the systematic errors are randomly chosen with a maximum of ~~5%~~ 5 % of the total number of stations in order to avoid a high number of invalid trace data. The ~~third block contains information regarding the synthetic subsurface model. For each layer the corresponding velocity~~ (final block of the configuration file (~~`velmap`~~`traveltimes`) ~~and layer thickness (`layers`) need to be provided and all layers are considered to be parallel to the surface topography~~

[3] https://www.pygimli.org/pygimliapi/_generated/pygimli.meshtools.html#pygimli.meshtools.createMesh, last accessed March 11, 2023

~~(geometrical information regarding the stations in the measurement scheme). The remaining parameters, namely `quality, area, smooth`~~ contains data-error parameters for both the absolute error (e_{abs}) and ~~`sec_nodes`, define the properties of the mesh to be used for the forward modeling (we refer to the respective pyGIMLi resources[4] for further information). Alternatively, the user can provide a more complex mesh in the binary mesh format (~~the relative error (e_{rel}) defined by the user. The data error model is then estimated as

$$\boldsymbol{e} = e_{abs} + \boldsymbol{d}\, e_{rel}\,, \tag{1}$$

where $\boldsymbol{d}$ denotes the forward modeled data not affected by noise, i.e., ~~a bms file): The parameters in the final block control the forward modeling of the corresponding seismic traveltimes (see Table 2). A configuration file located in the input directory can be imported through the `load` method:~~ here the first break traveltimes obtained through the Dijkstra algorithm (Dijkstra, 1959). These computed error values are subsequently added to the data to obtain the noisy data as

$$\boldsymbol{d}_{noise} = \boldsymbol{d}\,(1 + \boldsymbol{e})\,. \tag{2}$$

~~Once the~~

The configuration file needs to be provided via the input directory from where it can be imported through the `load` method of the `SeismicWaveformModeler`~~instance is parameterized the synthetic~~ :

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

```
INFO    : Configuration loaded
```

To demonstrate the ability to model seismic waveform data ~~can be created~~ for arbitrary subsurface conditions we do not define a simple subsurface model in the configuration file but provide a more complex model in the binary mesh format (e.g., a bms file). 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 corresponding python script can be found 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: ~~The~~

```
# Load the mesh into the workflow
swm.load(type='mesh')
```

[4] ~~https://www.pygimli.org/pygimliapi/_generated/pygimli.meshtools.html#pygimli.meshtools.createMesh, last accessed~~

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

```
INFO    : Mesh loaded
```

The forward modeling process generating the seismic waveform data is then invoked through the `create` method:

```
# Start the modeling of the seismic waveform data
swm.create(type='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 ~~and creates the velocity model used~~ 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 considered for the waveform modeling ~~.~~ ~~Based on the wavelet properties a Ricker wavelet is generated~~ in this study (see Figure 4a). The model 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 $2500\,\mathrm{ms}^{-1}$, i.e., the model features vertical and lateral variations in the seismic velocity distribution. The third step refers to the generation of a Ricker wavelet through the pyGIMLi function `ricker` ~~. Subsequently, mesh~~as

$$u = \left(1 - 2\pi^2 f^2 t^2\right) e^{-\pi^2 f^2 (t - t_0)^2} \tag{3}$$

based on the wavelet properties provided in the configuration file. In Equation 3, f is the frequency of the wavelet given in Hz, t refers to the time base definition, i.e., the length and resolution of the wavelet in the time domain, and t_0 is the time offset of the wavelet.

Based on the mesh, the velocity model and ~~Ricker wavelet are used~~ the Ricker wavelet we use pyGIMLi to solve the pressure wave equation for each shot station defined in the measurement scheme~~with the pyGIMLi function `solvePressureWave`~~. The resultant ~~waveforms~~ seismograms are extracted at the corresponding geophone stations~~are extracted and stored~~ , gathered for each shot position in an ObsPy `Stream` ~~data structure. A directory for each dataset defined in the configuration file will be created in the~~ object and saved in the output directory (*out*) ~~:~~

~~In~~ 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
```

In particular, the subdirectory *data* ~~, the synthetic seismic waveforms are stored in a separate shot file for each shot position~~ contains a separate file in the miniseed format (Ahern et al., 2012; Ringler and Evans, 2015) ~~with~~ for each shot

position, where the file extension *syn* ~~to identify the forward modeled shot files , e. g., Shot_1001.syn.~~ indicates 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~~required for the processing of the seismic waveforms, namely~~ , e.g., sampling rate, recording length, number of geophones and geophone spacing. Moreover, the protocol associates each shot file of the dataset ~~to~~ with a specific location ~~within~~ in the survey geometry ~~, i.e.,~~ with respect to the geophone positions~~:~~ , e.g.:

```
 ############################
   Line: SYN_syn_data
   Sampling rate: 2000 Hz
   Recording length: 0.512 s
   Number of geophones: 48
   Geophone spacing: 2 m
 ############################
%DIF >
     File number | Station
          1001    |   G001
            :     |     :
          1048    |   G048
```

The auxiliary file info.txt ~~provided in~~ exported to the dataset directory summarizes the parameters from the configuration file ~~and~~ as well as information regarding the simulated systematic errors in the synthetic seismic waveform data~~:~~ ~~Synthetic traveltimes (`swm.create('traveltimes')`) are stored in the dataset directory as udf files,~~ , e.g.:

```
Number of geophones: 48
Number of shots: 48
Recording length (s): 0.512
Sampling frequency (Hz): 2000
Wavelet type: Ricker
Frequency of the wavelet (Hz): 100
%DIF >
Missing shot(s): 11
Broken geophone(s): 13
Wrong polarity geophone(s): 26
```

Figure 4b presents the seismic waveform data forward modeled for a shot point located in the center of the profile. The seismograms are shown as curves, whereas the strength of the amplitudes is added as color-coded information. In the seismograms we see clear first onsets along the entire profile, yet receivers 3 and 45 were modeled as broken geophones, i.e., the corresponding seismograms contain solely noise. 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, to demonstrate how such outliers can be identified in a so-called pseudosection.

Pseudosections are a useful tool for the analysis of the data quality by presenting the data based on the positions of shots and receivers. Such a visualization allows for the detection of outliers and their spatial distribution as necessary to understand possible sources of error. In case of the SRT, a pseudosection as presented in Figure 4c, illustrates apparent velocity (v_{app}) values computed as

$$v_{app} = \frac{aoffset}{tt}, \tag{4}$$

where $aoffset$ refers 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., ~~dataset1_tt.pck.The file extension *pck* is an abbreviation of the word 'pick' and~~ 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. The inversion of first break traveltimes aims at resolving a model of the P-wave velocity of the subsurface materials, where systematic errors in the data would lead to the creation of artifacts in the imaging results or hinder the convergence of the inversion. Considering the multitude of available inversion frameworks we do not include a new inversion strategy in the formikoj library. Instead the processed data can be exported in any format required for the use of commercial inversion algorithms, or can be linked directly to other open-source libraries. In our case, we refer to the modeling and inversion capabilities of pyGIMLi (Rücker et al., 2017) . pyGIMLi uses a generalized Gauss-Newton method to solve the inversion problem through the minimization of an objective function given as:

$$\| \boldsymbol{W}_d \left(\mathcal{F}(\boldsymbol{m}) - \boldsymbol{d} \right) \|_2^2 + \lambda \| \boldsymbol{W}_m \left(\boldsymbol{m} - \boldsymbol{m}_0 \right) \|_2^2 \rightarrow \min \tag{5}$$

The first term on the right-hand side of Equation 5 refers to the data misfit. $\boldsymbol{W}_d$ is the data weighting matrix holding the reciprocals of the data errors, $\boldsymbol{d}$ denotes the data vector holding the input data (first break traveltimes~~saved in the file.~~), $\boldsymbol{m}$ is the model vector representing the target parameter (seismic P-wave velocity values), and $\mathcal{F}(\boldsymbol{m})$ is the model response. The second term denotes represents the regularization, where $\boldsymbol{W}_m$ and $\boldsymbol{m}_0$ are the model constraint matrix and the reference model, respectively. The regularization parameter λ balances the influence of the data misfit and 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, which is given in terms of the spatial variations of the seismic P-wave velocity, i.e., reflecting the associated lateral and vertical variations. Depending on the survey geometry the inversion can resolve subsurface models in both 2D or 3D. 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); yet, the exploration of different inversion strategies is beyond the scope of this study. 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 `SeismicRefractionManager`~~

The ~~SR~~ seismic refraction (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 the seismic velocity model obtained through the inversion. Accordingly, a proper enhancement of the perceptibility of the first onsets is crucial.

~~Accordingly, the~~ 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. The first break traveltimes determined with the `SeismicRefractionManager` represent the input data, e.g., for the `TravelTimeManager` of pyGIMLi (Rücker et al., 2017)

. This is particularly relevant as the `TravelTimeManager` provides an inversion framework for first break traveltimes, yet not a framework for the processing of seismic waveform data. For the demonstration of the fundamental `SeismicRefraction` capabilities we consider the synthetic seismic data created with the `SeismicWaveformModeler` above, which illustrates that both classes can be combined in subsequent workflows.

2.2.1. Compiling the survey information and creating a project

~~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, (ii)~~ To create a new ~~project, or (iii)~~ or load an existing ~~project from disk.~~

~~The data preview mode is primarily initiated if the provided working directory contains seismic shot files. To create or load a project~~`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. For the synthetic dataset considered in this study, the

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

~~Col~~ 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)

2D station coordinates are provided in the geometry file, whereas the z coordinate is assumed to be 0 m along the entire profile, as illustrated in Table 4; thus, reflecting the lack of surface topography in the synthetic model (see Figure 4a).

Table 4
Excerpt from the geometry file describing the survey geometry of the synthetic dataset generated in this study.

x (m)	y (m)	z (m)	Geo	Shot	1st Geo	# Geo
0.0	0.0	0.0	1	1001	1	48
2.0	0.0	0.0	1	1002	1	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
20.0	0.0	0.0	1	-1	1	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
24.0	0.0	0.0	0	1013	1	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
92.0	0.0	0.0	1	1047	1	48
94.0	0.0	0.0	1	1048	1	48

In the column **Geo** the geometry file contains the value 1 (True) for each station except the station located at 24 m referring to the broken geophone modeled by the `SeismicWaveformModeler`. When compiling the geometry file we also need to take into account the missing shot at station 11, i.e., the station located at 20 m along profile direction, and set the corresponding value to −1. The column **1st Geo** indicates the first active geophone along the profile for each shot file, which for the synthetic dataset considered here is the geophone deployed at the first station along the entire profile. In particular, the column **1st Geo** allows the reproduction of roll-along survey geometries, where in the first segment the first active geophone is always the geophone at station 1. Considering 48 geophones deployed in each segment, and an overlap of 50 %, the first geophone for the consecutive segments would be 25, 49, 73, 97, and so on.

An instance of the `SeismicRefractionManager` can be created by providing the path to the working directory as parameter to the constructor. Depending 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: ~~In particular~~

```
# Import the SeismicRefractionManager from the formikoj library
from formikoj import SeismicRefractionManager
%DIF >
# 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~~ *prj.db* in the working directory ~~) to store the geometry information, whereby the stations are consecutively numbered~~ 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 ~~through~~ for the user the `SeismicRefractionManager` ~~links~~ creates database tables *shots* an d*receivers*, which link the station numbers to shot index numbers (SIN) and receiver index numbers (RIN)~~assigned to shot and receiver stations, respectively(see Figure 6). Based on this information, the geometry is applied~~, 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 ~~database tables required for the processing of the seismic waveform data are created. In particular,~~ geometry is applied. In the last step, the database table *fbpicks* is created, which stores the first break traveltimes for each SIN-RIN pair ~~are stored in a dedicated database table *fbpicks*~~ together with the name of the corresponding pickset, i.e., ~~the~~ 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.

~~If a database file is already present in the working directory, the project information, the seismic waveforms as well as the default pickset 'picks' are automatically loaded:~~

~~For a project with a successfully applied geometry ,~~

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 ~~waveforms~~ waveform data based on a common ~~shot~~ (absolute offset (~~sin~~`aoffset`), a common ~~receiver~~ RIN (`rin`)~~or the common absolute offset (~~, or a common SIN (~~`aoffset`~~`sin`) ~~:~~

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

```
INFO    : 88 traces selected
```

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

```
INFO    : 48 traces selected
```

```
# Select traces with receiver
srm.select(by='sin', num=23)
```

```
INFO    : 48 traces selected
```

2.2.3. ~~*Visualization and enhancement of the seismic waveform data*~~

~~Calling the~~ Figure 7 shows the frequency spectrum of the currently selected traces, which can be computed and visualized through the `plot` method~~without passing any parameter allows for the visualization of the~~ :

```
# Plot the frequency spectrum
srm.plot(type='spectrum')
```

The `SeismicRefractionManager` resolves the frequency spectrum by computing the stacking the power spectral density (psd) of the seismograms $s\,(t)$ as

$$psd = 10\log_{10}\left(\frac{|\mathrm{fft}\,(s\,(t))|^2}{N}\right), \tag{6}$$

where fft refers to the fast fourier transformation (FFT) and N is the number of the considered seismograms. The frequency spectrum provides information regarding the amplitudes associated to different frequencies in the seismograms, which can be use for the identification of the dominating frequencies as required for the selection and application of adequate filters such as lowpass, highpass, bandpass and bandstop implemented in ObsPy (Beyreuther et al., 2010). To enhance the signal-to-noise ratio of the seismograms the user can apply the respective filters through the `filter` method, which utilizes the frequency filters implemented in the ObsPy package (lowpass, highpass, bandpass and bandstop; Beyr , e.g.:

```
# Apply bandpass filter
srm.filter(type='bandpass', freqmin=10, freqmax=120)
```

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

In this example, the filter is solely applied to the currently selected traces~~: Once opened~~, yet setting the parameter `onhold` as `True` automatically filters all subsequently selected traces with the same filter settings, e.g.:

```
# Apply lowpass filter
srm.filter(type='lowpass', freq=200, onhold=True)
```

```
INFO    : Applied 200.0 Hz lowpass filter
INFO    : Set filter hold on
```

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

```
# Plot selected traces
srm.plot()
```

By default, 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 modethe~~ mode, with the latter reflecting 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 8).

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 9). The initial processing mode is 'Fb pick', i.e., first break picking is possible. ~~The user can switch between the different modesby pressing specific keys on the keyboard.~~ Additional modes, accessible through the keyboard, allow for an enhanced visualization of the seismograms, e.g., associated to broken geophones or seismograms with wrong polarity. 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 9).

~~Through the `plot` method the frequency spectrum of the currently selected trace data can be visualized: A frequency spectrum as shown in Figure 7 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. 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. 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., 20 , e.g.: 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:

```
# Plot traveltime diagram
srm.plot(type='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')~~ respective data point 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`: 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: 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~~ Once the user is satisfied with the quality of the first break traveltimes, the data points of the pickset can be exported in the unified data format, which is compatible with the pyGIMLi framework. In particular, the udf file is stored in ~~*03_proc/picks* subdirectory~~ the subdirectory *03_proc/picks* with the current timestamp as suffix:

```
# Export first break traveltimes
srm.picksets(do='export', name='pick')
```

```
INFO    : Pickset 'pick' saved to pick_20230303T140447.pck'
```

3. ~~Exemplary use cases~~

2.1. ~~Modeling and use~~ Expanding the capabilities of ~~synthetic seismic waveform data~~the formikoj library

~~To demonstrate the applicability of the `SeismicWaveformModeler` class, we generate two synthetic seismic waveform datasets assuming two horizontal layers in a half-space without topography and an interface between these layers with a constant depth. Table **??** summarizes the parameterization used for the forward modeling of one dataset without noise (Dataset_1) and another dataset containing random noise and systematic errors (Dataset_2). For the visualization of both synthetic datasets, we provide the corresponding geometry files to the `SeismicRefractionManager` in order to have full control regarding data selection and processing capabilities.~~

~~The synthetic seismic waveforms for SIN 24 of Dataset_1 presented in Figure **??** reveal clear negative first onsets and allow the identification of crossover points, i.e., inflexion points between the first arrivals from the first layer (RIN 17 to 30) and the first onsets associated to the second layer (RIN 1 to 16 and RIN 31 to 48). In contrast the signal-to-noise ratio of Dataset_2 (see Figure **??**) is a function of the offset between shot and geophone position, i.e., traces farther away from the shot contain a higher level of random noise. Moreover, Dataset_2 also contains systematic errors, where RIN 6 and 14 refer to broken geophones, and RIN 35 is an example for readings with a wrong polarity. We believe that such synthetic datasets might be useful for investigating the effect of complex survey geometries on seismic data as well as the development and evaluation of processing strategies. Measurement scheme and parameters provided in the yaml files used to create synthetic seismic waveform datasets with (Dataset_1) and without added noise (Dataset_2). **Measurement scheme** Number of stations 48 Station spacing 2 m Number of geophones 48 Number of shots 48 **Model** *Layer 1 Layer 2* Thickness 3 m 10 m Velocity 750 m/s 3000 m/s **Dataset** *Dataset_1 Dataset_2* Noise False True Noise level 0 1e-5 Missing shots False True Broken geophones False True Wrong polarity False True~~

~~**Wavelet** Length 1.024 s Frequency 100 Hz Sampling rate 2000 Hz Synthetic seismic waveform data without random or systematic errors created with the `SeismicWaveformModeler()` class for a shot position in the center of the survey layout.~~

~~Synthetic seismic waveform data with added noise created with the `SeismicWaveformModeler()` class for a shot position in the center of the survey layout. The random noise refers to an offset dependent decrease of the signal-to-noise ratio, while the systematic broken geophones and wrong polarity are systematic errors.~~

~~Accordingly, the concept of the formikoj library allows the implementation of supplementary functionalities. Such~~ Making the formikoj library publicly available under an open-source license allows the addition of supplementary functionalities tailored for the specific requirements of the users. The concept of formikoj suggest that such custom extensions should be implemented either as internal methods or as functions in the utilities module, which are then executed through the `compute` method~~with a custom keyword. As an illustrative example, we implemented~~.

We illustrate this possibility for customization by implementing a simplified version of an automatic first break picking algorithm based on the short- and long-time window average ratio (STA/LTA) method (Allen, 1978), which determines the traveltimes ~~based on~~ following the energy ratio ~~method (e.g., Earle and Shearer, 1994). The~~ approach (e.g., Earle and Shearer, 1994). In particular, our implementation computes the envelope $E(t)$ of the seismogram $s(t)$ as (e.g., Duan and Zhang, 2020)

$$E(t) = \left(s(t)^2 + \tilde{s}(t)\right)^{1/2}, \tag{7}$$

where $\tilde{s}(t)$ is the Hilbert transform of $s(t)$. The energy ratio ER is then computed as $ER = STA/LTA$ with

$$STA(i) = \frac{1}{n_{STA}} \sum_{j=i-n_{STA}}^{i} E(j), \tag{8}$$

and

$$LTA(i) = \frac{1}{n_{LTA}} \sum_{j=i-n_{STA}}^{i} E(j), \tag{9}$$

where n_{STA} and n_{LTA} refer to the number of data points in $E(t)$ considered for the short- and long-time windows, respectively. For this exemplary implementation we determine the first break traveltimes in the seismograms as the

position of the maximum in the ER function, which is automatically saved in the project database.

The autopicking algorithm is added to the `SeismicRefractionManager` in form of two internal methods `_manage_autopicking` and `_compute_autopicks`, respectively. ~~The autopicking process can be started by passing the new keyword~~ To invoke the autopicking process we modified the `compute` method, which now accepts the custom-defined keyword ~~`autopick`~~`autopicking`~~as the first parameter to the~~. Additionally, the autopicking requires values to be passed for the parameters ~~`compute`~~`pick` ~~method. The second parameter defines whether traveltimes should be determined for the entire dataset (~~and `pickset`. The former accepts the values `all` ~~) or solely~~ or `cur` to determine first break traveltimes for all traces in the dataset or solely the currently selected traces~~(`cur`). The third parameter provides~~, respectively. The latter defines the name of the ~~corresponding pickset in the project database; thus, making the traveltimes available for visualization and processing with existing functionalities or further custom implementations.~~ picksets in which the automatically determined traveltimes should be saved to. A typical use case involves conducting the autopicking and exporting the determined traveltimes:

```
# Automatically pick first break traveltimes
srm.compute(do='autopicking', pick='all', pickset='autopicks')
srm.picksets(do='export', name='autopicks')
```

```
INFO    : Created new pickset 'autopicks'
INFO    : Pickset 'autopicks' loaded
INFO    : 'autopicks' set at active pickset
Progress <==============================> 100.0% completed
INFO    : Pickset 'autopicks' saved to autopicks_20230303T140834.pck
```

2.2. ~~First break travel time picking for a 2D roll-along field data set: the Danube island example~~

~~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, we compare the automatically determined first break picks with the forward modeled traveltimes computed above to allow for a basic evaluation of autopicking performance. The inset plot in Figure 11 presents the histogram of the autopicked traveltimes, which shows that three traveltimes should be considered outliers, and thus are removed from the dataset. After the outlier removal the correlation coefficient between forward modeled and autopicked traveltimes is 0.99 suggesting that the implemented autopicking algorithm performs well for the synthetic seismic waveform data. However, the observed deviation from the perfect correlation (i.e., the 1:1 line in Figure **??**~~.~~11) indicates that autopicking and forward modeling algorithm might be sensitive to different seismic wave phases. Further improvements in terms of the autopicking process could incorporate, for example, machine learning approaches as the method proposed by Duan and Zhang (2020), which can be easily implemented as an additional functionality in the

SeismicRefractionManager. We point out here, that following the same procedure, the user can implement further autopicking algorithms as well as other data processing strategies and have a simple framework for the evaluation of their performance through the analysis of synthetic data (e.g., clean and contaminated with Gaussian noise).

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

To demonstrate the applicability of the SeismicRefractionManager for the processing of real field data, we present here the analysis and inversion inversion results for a seismic refraction survey conducted in a soda lake located in the Nationalpark Neusiedler See–Seewinkel close to Vienna. The seismic survey aims at solving for the geometry of the confined aquifer below this soda lake with the required information referring to the ~~survey geometry refers to a roll-along geometry, i. e.,~~ thickness of the aquifer and 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.~~ depth of the groundwater table. As presented in Figure 12, 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 to obtain a sufficient coverage.

As in case of the synthetic dataset presented above, the geometry file contains the coordinates of the survey stations, yet for the soda lake dataset 3D coordinates we provided as illustrated in Table 5. Since geophones were not deployed at each survey station the column **Geo** contains the value 1 (True) for the first 96 stations and 0 (False) for the remaining 20 stations.

~~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 **??**. 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.~~

~~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.~~

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 5
Extract Excerpt from the roll-along 3D survey geometry fileshowing 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	
0.0 40336.056	0.0 292470.692	163.5 120.0	1	-1 -1 -1 2.0 0.0 163.5 0 1001	
⋮ 40334.751	⋮ 292469.177	⋮ 120.0	⋮ 1	⋮ 1002	⋮
98.0 0.0 163.5 0 1037 25 48 ⋮	⋮	⋮	⋮	⋮	
194.0 40284.472	0.0 292455.431	163.5 120.0	0 1	1049 1058	
196.0 40285.896	0.0 292454.027	163.5 120.0	1	-1 1059	-1 -1 198.0
202.0 0.0 163.5 0 1050 25 48 ⋮	⋮	⋮	⋮	⋮	
286.0 40339.421	0.0 292402.681	163.5 120.0	0 1	1084 1096	
290.0 40305.228	0.0 292445.050	163.5 120.0	0	1097	
⋮	⋮	⋮	⋮	⋮	
386.0 40265.415	0.0 292431.957	163.5 120.0	0	1109 1115	
390.0 40255.453	0.0 292431.395	163.5 120.0	0	2025 1116	97 48 ⋮⋮⋮⋮⋮⋮⋮570.0

offset stack (COS): The COS for the Danube island dataset presented in Figure ?? 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: 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 ??, the first onsets are easily identifiable despite the substantial seismic background noise at large offsets (RIN 73 to 90). 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. A pseudosection provides an illustration of the corresponding apparent seismic velocity values computed from the picked traveltimes and the distance between shots and geophones:

In a pseudosection, as presented in Figure ?? 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 ?? 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. 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).

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: Figure ?? 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.

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.

3.1. Processing of a 3D seismic refraction dataset: the soda lake example

The `SeismicRefractionManager` allows also the visualization and processing of data collected with a , automatically infers the 3D survey layout . To illustrate these capabilities, we present in Figure 12 the geometry of a from the 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 geometry, 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 geometryshown in Figure 12 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 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.~~

~~The geometry file contains the 3D coordinates for each shot or receiver station, and thus the~~`SeismicRefractionManager` ~~automatically~~ thus configures the project for 3D processing. ~~Figure 13 presents the~~ Then we can select seismograms the same way as for the synthetic data presented above. For this example we select seismic waveform data recorded for SIN 1, i.e., the shot position co-located with the first geophone (Station 01 in Figure 12)~~. The~~ :

```
# Select traces with receiver
srm.select(by='sin', num=1)
```

```
INFO    : 96 traces selected
```

In Figure 13, we can see that 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~~ for assessing the quality of the first break traveltimes. ~~Hence~~However, the `SeismicRefractionManager` ~~automatically switches to a~~ project is configured for 3D ~~representation where~~ processing, and thus 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 14 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.

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

```
# Plot the picking percentage
srm.plot(type='pickperc')
```

Figure 15 presents the picking percentage visualized separately for each SIN. Accordingly, a low picking percentage indicates shots affected by a low signal-to-noise ratio, whereas clear first onsets yield a correspondingly high picking percentage. For the soda lake dataset we observe a consistently high picking percentage for all shot positions; thus, indicating a good data quality. Moreover, the picking percentage plot can be used to track the picking progress, for instance, in case the traveltimes cannot be determined in frame of 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.

Once the first break picking is finished, the corresponding pickset can be exported for the inversion. ~~The inversion results and their interpretation are not~~ We inverted the first break traveltimes with pyGIMLi and present the resolved 3D subsurface model in Figure 16a. From this representation we can see, that the inversion solves for low seismic velocities (600 to 800 ms^{-1}) in the near-surface in the center of the investigated area, which corresponds to the still intact part of the soda lake, i.e., the part that is covered by water on a seasonal basis. Seismic velocity values above 800 ms^{-1} are resolved at depth as well as outside of the soda lake. To facilitate the interpretation of the resolved seismic velocity distribution in terms of the aquifer geometry we show the two vertical slices in Figure 16b and c, respectively. In particular, we relate seismic velocity values $>$ 800 ms^{-1} to the transition from the unsaturated to the saturated zone due to the higher seismic velocity of water ($\approx$ 1500 ms^{-1}) compared to air ($\approx$ 340 ms^{-1}) filling the pore space. Accordingly, our 3D subsurface model indicates the depth of the groundwater table at a depth of approximately 8 m. A more detailed interpretation is beyond the scope of this manuscript, yet ~~Figures 13 and 14~~ the presented figures 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~~ By standardizing the data input in combination with a thorough sanity check aims at reducing the risk of corrupting the information stored in the project database. Moreover, crucial processing steps are automatized within the `SeismicWaveformModeler` and the `SeismicRefractionManager` ~~based on~~ classes facilitated by 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~~ using and testing the formikoj library ~~against established software packages~~ in frame of their research activities, and for providing valuable suggestions for improvement. 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~~ datasets are available for downloading at the link:

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

The ~~orthophotos used in Figures ?? and 12 were~~ orthophoto used in Figure 12 was 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
%DIF >
# 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)
%DIF >
# - 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)
%DIF >
# - anomaly/infill
infill = mt.createPolygon([[20, -2], [72, -3.5],
                           [70, -6], [22, -6]],
                          isClosed=True, marker=2, area=0.1)
%DIF >
```

```
geom = top + infill + bottom
%DIF >
# Define shot and receiver stations and create corresponding nodes
nstats = 48
spc = 2
%DIF >
stations = np.vstack((np.linspace(0, (nstats-1)*spc, nstats),
                      np.zeros(nstats))).T
%DIF >
for p in stations:
    geom.createNode(p)
%DIF >
# Create mesh for the finite element modeling
mesh = mt.createMesh(geom, quality=34)
%DIF >
# 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.

Allen, R.V., 1978. Automatic earthquake recognition and timing from single traces. Bulletin of the seismological society of America 68, 1521–1532. doi:10.1785/bssa0680051521.

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.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik , 269–271doi:10.1007/bf01386390.

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.

Duan, X., Zhang, J., 2020. Multitrace first-break picking using an integrated seismic and machine learning methodpicking based on machine learning. Geophysics 85, WA269–WA277. doi:10.1190/GEO2019-0422.1.

Earle, P.S., Shearer, P.M., 1994. Characterization of global seismograms using an automatic-picking algorithm. Bulletin of the Seismological Society of America 84, 366–376.

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

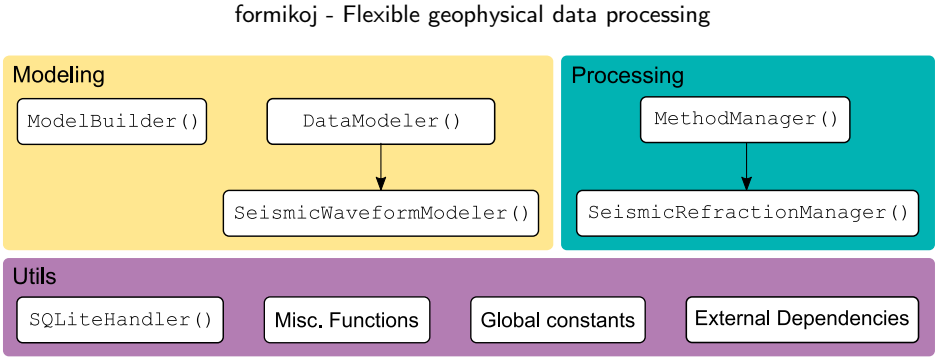


Figure 1: Fundamental use case illustrating the modeling of seismic refraction data within the formikoj ecosystem.

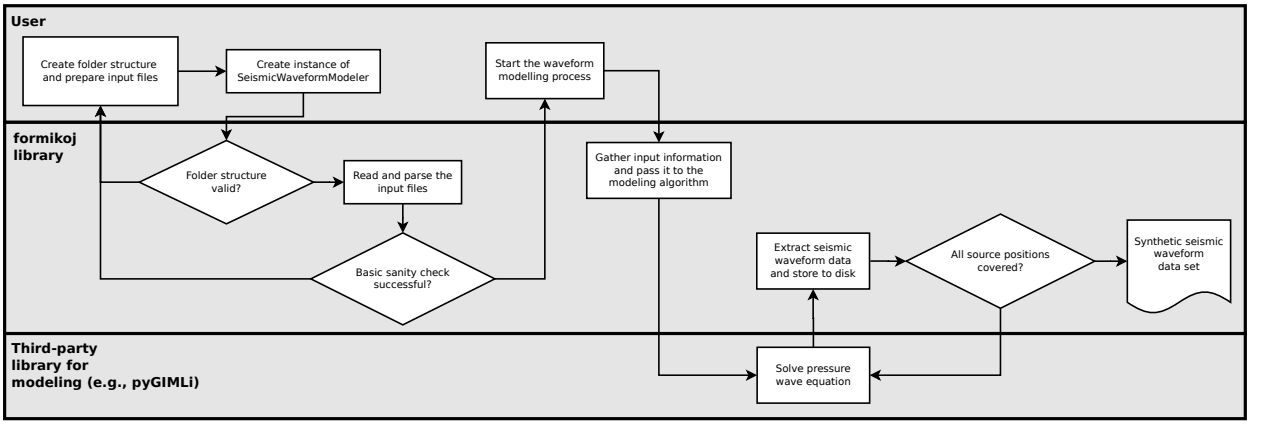


Figure 2: Fundamental use case illustrating the processing of seismic refraction data within the formikoj ecosystem.

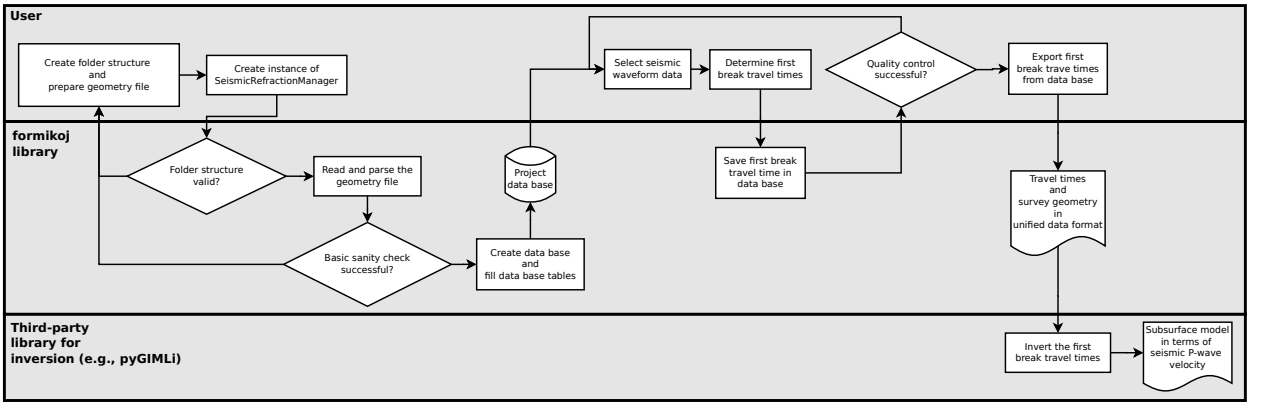


Figure 3: Typical formikoj workflow for the picking of first break traveltimes from seismic waveform data.

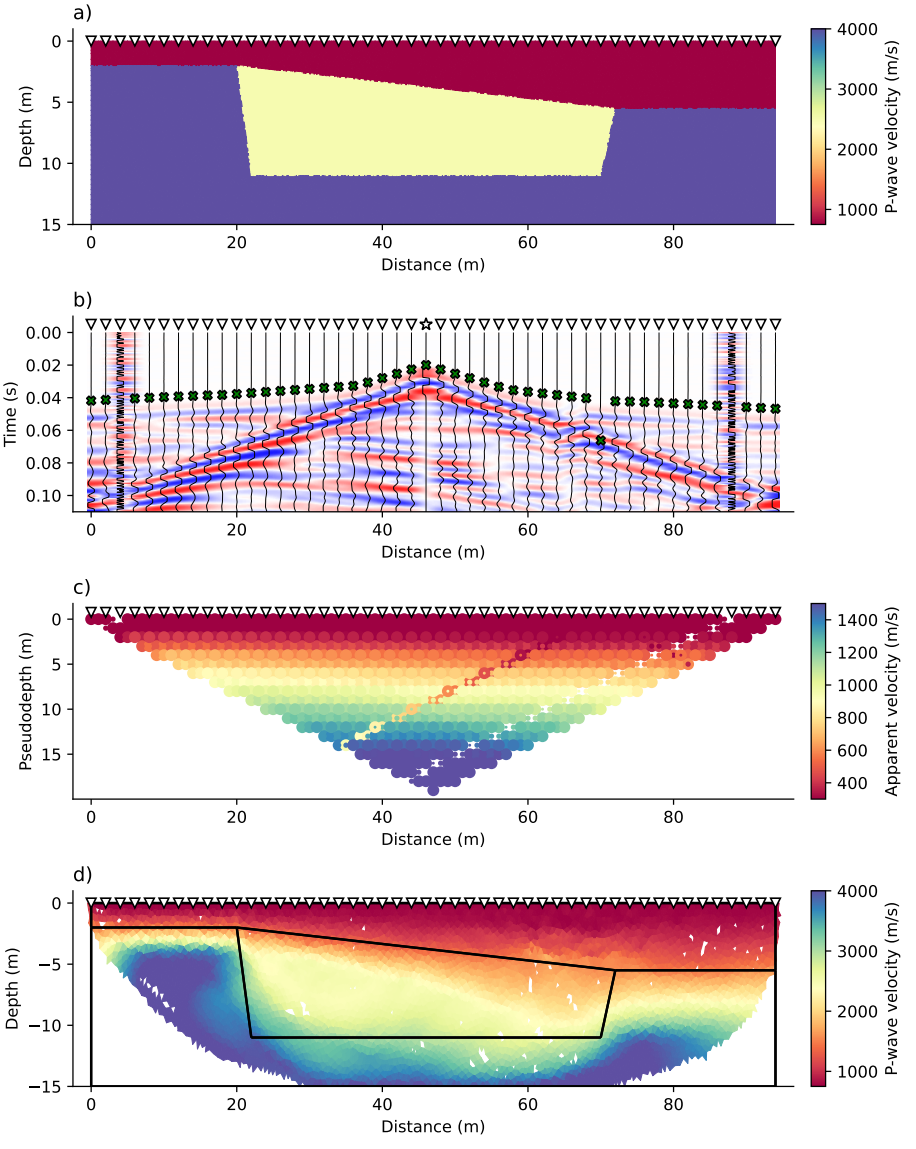


Figure 4: Synthetic seismic data generated with the `SeismicWaveformModeler`:
(a) Two-layered subsurface model without topography characterized by a distinct anomaly in the center. Triangles along the surface indicate the positions of geophones.
(b) Synthetic seismic waveform data computed from the subsurface 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 traveltimes based on the survey geometry.
(d) Subsurface model of the seismic P-wave velocity distribution obtained through the inversion of the synthetic P-wave traveltimes.

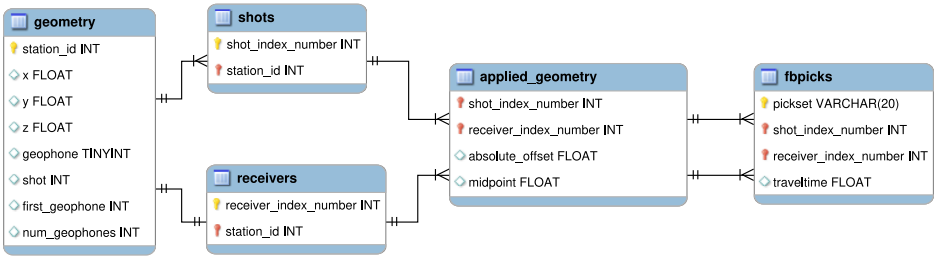


Figure 5: Entity-relationship diagram illustrating the SQLite database used in a `SeismicRefractionManager` project to store an manage processing related information.

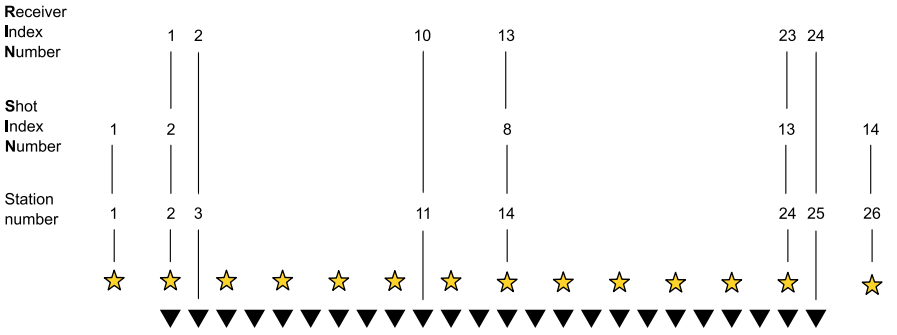


Figure 6: The `SeismicRefractionManager` addresses the stations through consecutive station numbers based on the sort order in the geometry file. The shot index numbers (SIN) and receiver index numbers (RIN) are assigned to the shot and receiver stations separately to allow for an intuitive data handling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

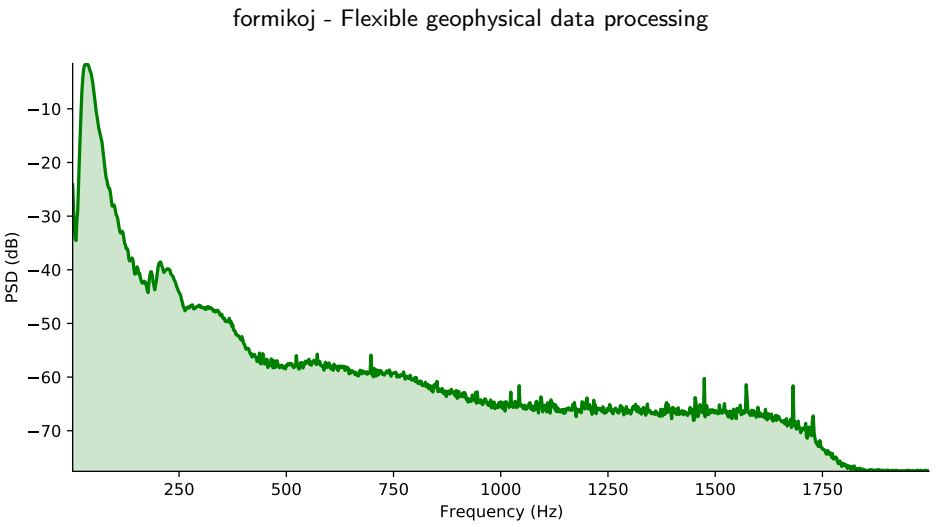


Figure 7: The frequency spectrum illustrates the frequency content of the currently selected traces, which allows for the identification of frequency ranges associated to noise allowing for the definition of corresponding frequency filter settings.

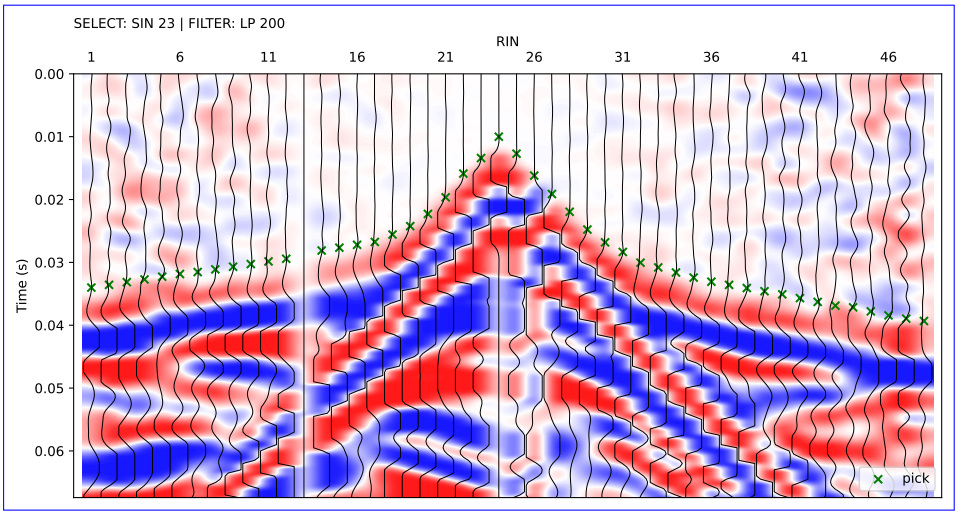


Figure 8: The seismogram plot presents the currently selected traces along the x axis, where the sort order is determined through the geometry. The corresponding trace data are illustrated as a function of time along the y axis (solid curves), with positive and negative amplitudes depicted in blue and 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.

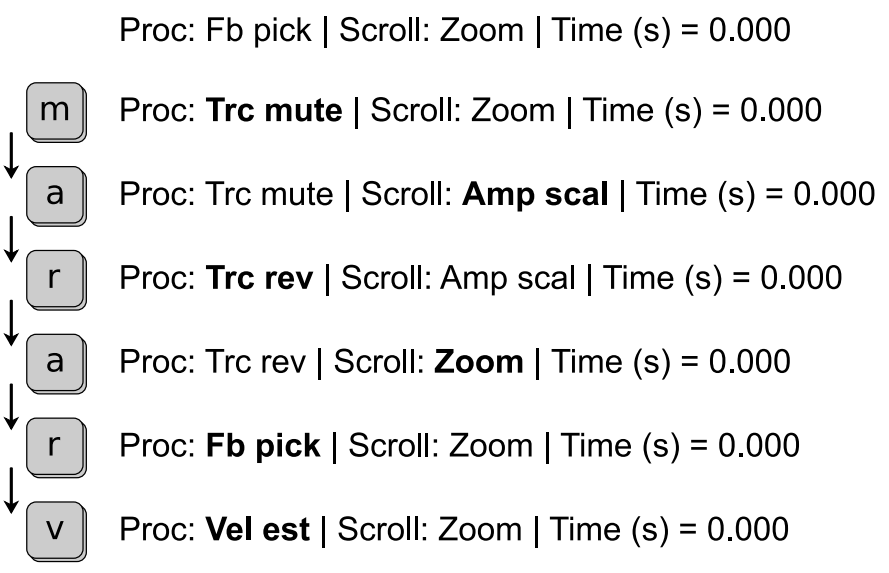


Figure 9: The status bar in the interactive seismogram plot displays the currently active processing and data scaling modes as well as the time (in seconds) at the current cursor position. By pressing the keys 'a', 'm', 'r', 'v' on the keyboard the different modes can be activated.

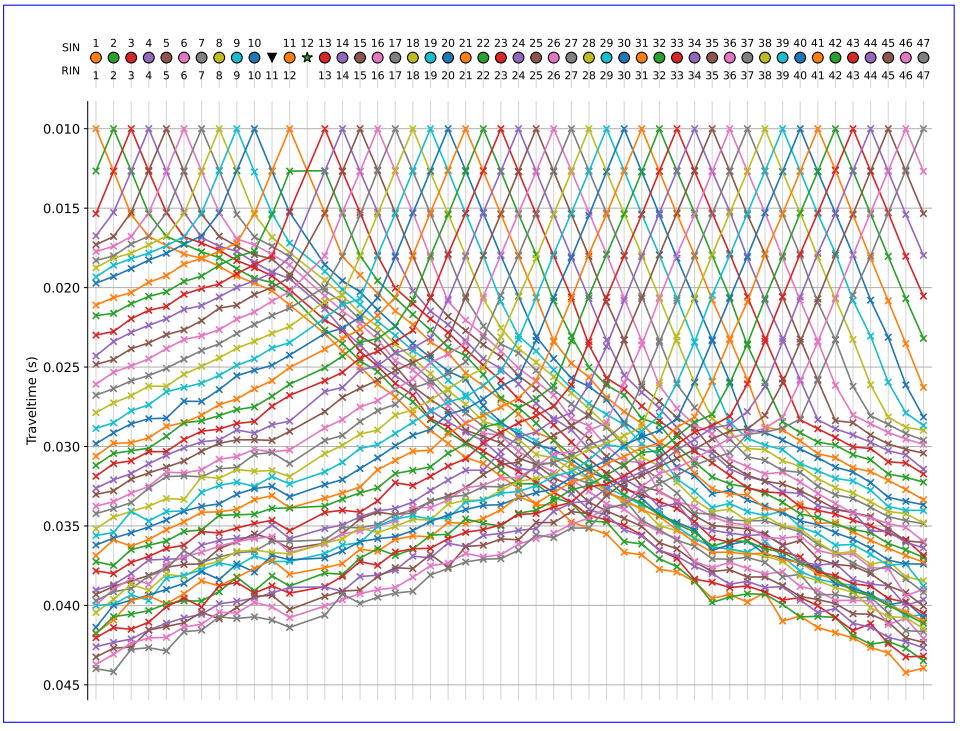


Figure 10: The traveltime diagram shows the first break traveltimes stored in the currently active pickset along the y axis (x symbols), where solid lines connect traveltimes assigned to a common shot. The sort order of the stations along the x axis reflects the geometry. Filled circles indicate stations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

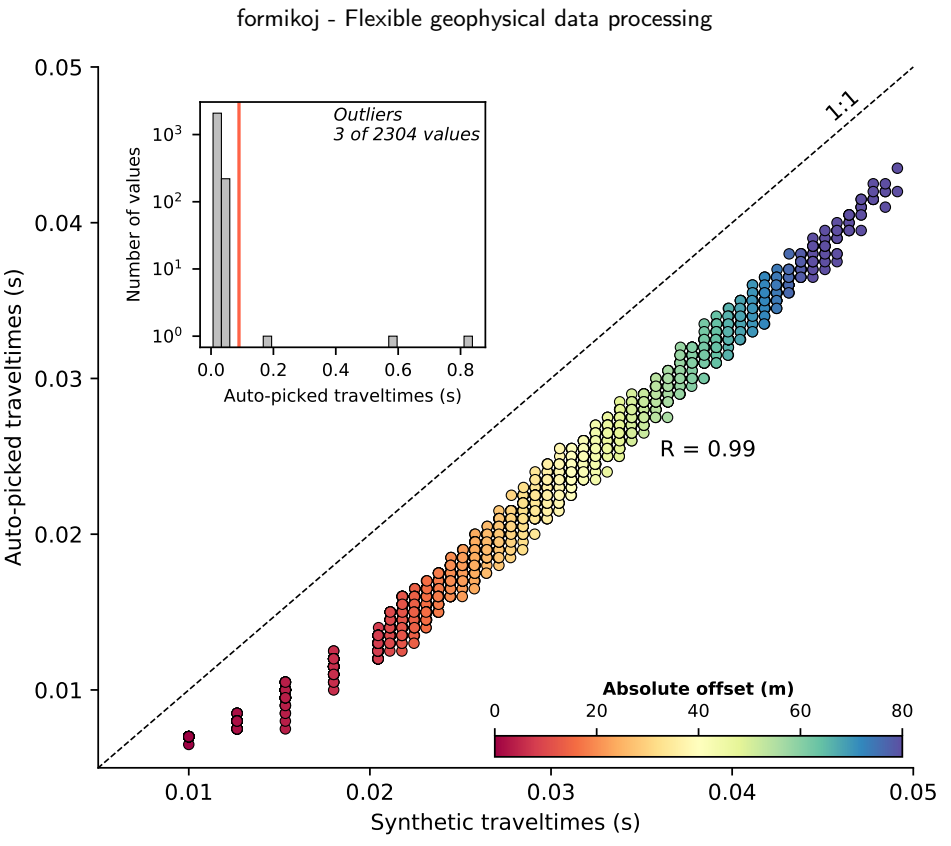


Figure 11: Comparison of forward modeled synthetic and automatically picked first break traveltimes for the synthetic seismic waveform data created in this study.

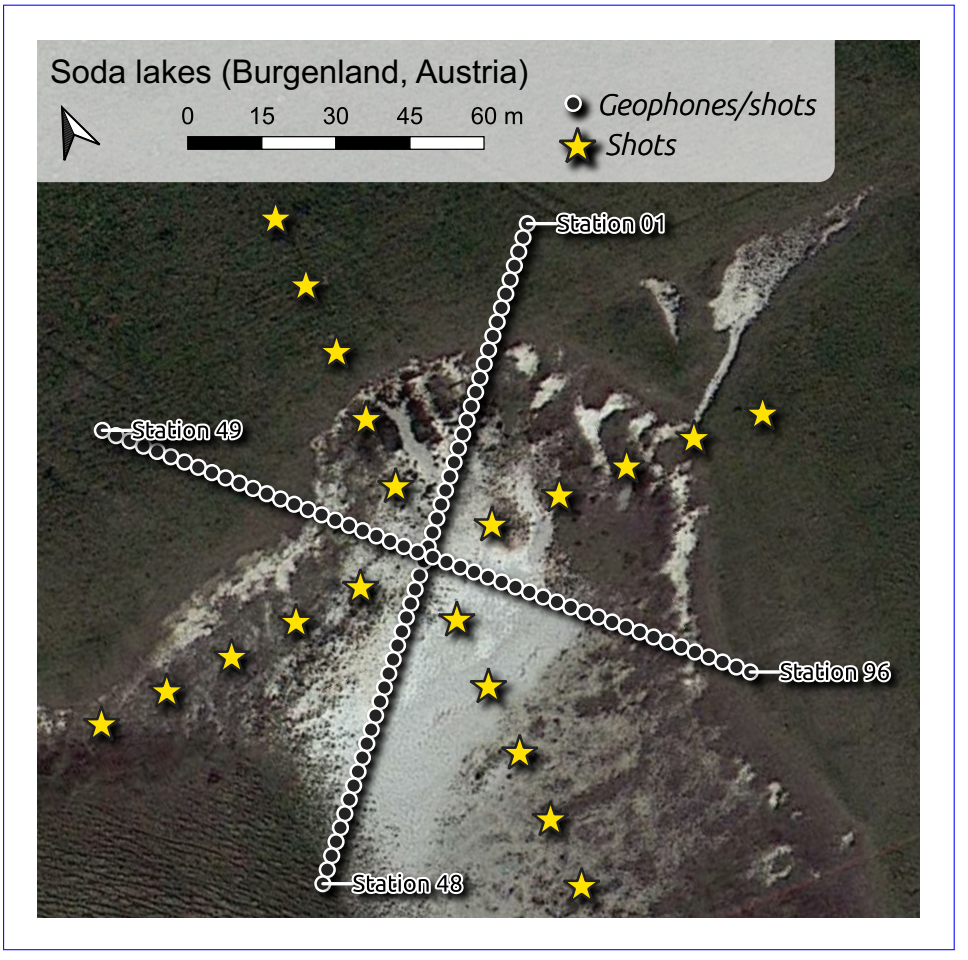


Figure 12: The ~~Danube island~~ soda lakes field dataset was collected ~~along~~ in a ~~single line with a roll-along~~ 3D survey layout ~~; the~~ with stations (co-located geophones and shots indicated by filled ~~triangles indicate the direction~~ dots) deployed in form of ~~the measurements~~a cross. ~~The five segments have an overlap~~ Additional shots (yellow stars) were conducted in from of ~~50%~~ a cross rotated by 45° to ~~ensure an adequate data~~ increase the coverage ~~along~~ of the ~~entire profile~~dataset.

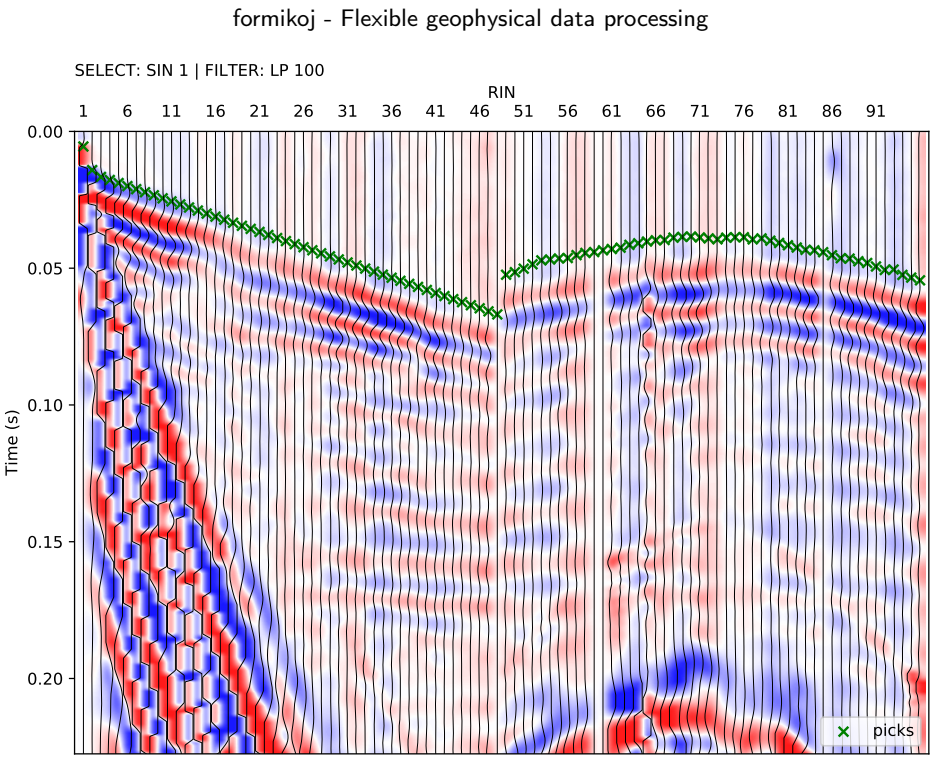


Figure 13: Examplary seismic waveforms from the soda lakes dataset shown for shot index number (SIN) 1 with a 100 Hz lowpass filter applied to suppress high frequency noise. The recorded seismic waveforms clearly reflect the geometry of the geophones with RIN 1 to 48 deployed along the direction of wave propagation, while RIN 49 to 96 are deployed perpendicular to the propagating wavefront.

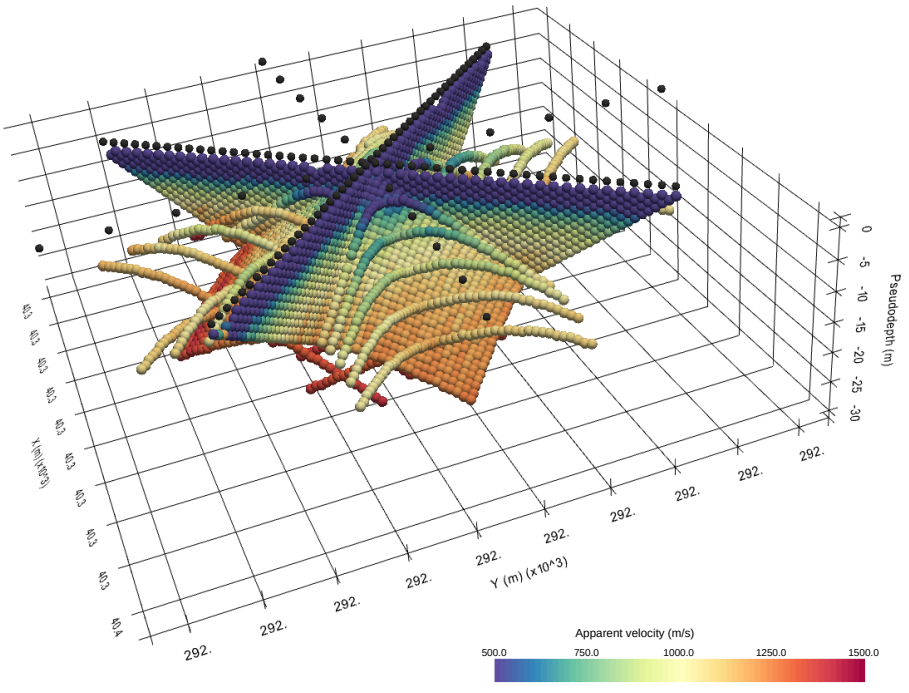


Figure 14: 3D pseudosection showing the apparent seismic velocities determined from the first break traveltimes obtained from the soda lakes dataset and the corresponding absolute offset between the shot and receiver stations. The apparent velocity for each shot-receiver pair is illustrated at the corresponding 2D midpoint and pseudodepth (1/3 of the absolute offset), thus yielding a 3D representation.

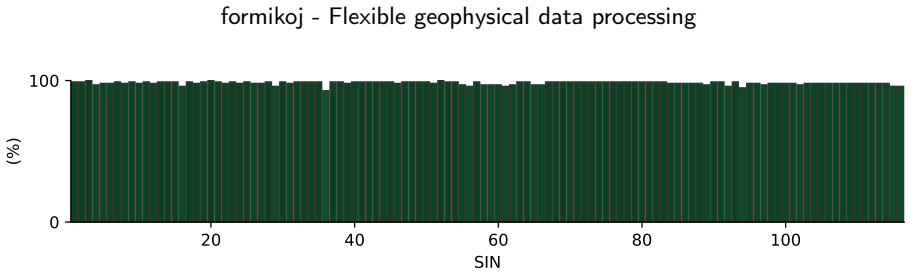


Figure 15: Picking percentage for each shot in the soda lakes dataset indicating a high signal-to-noise ratio allowing for the picking of first break traveltimes for nearly all shot-receiver pairs.

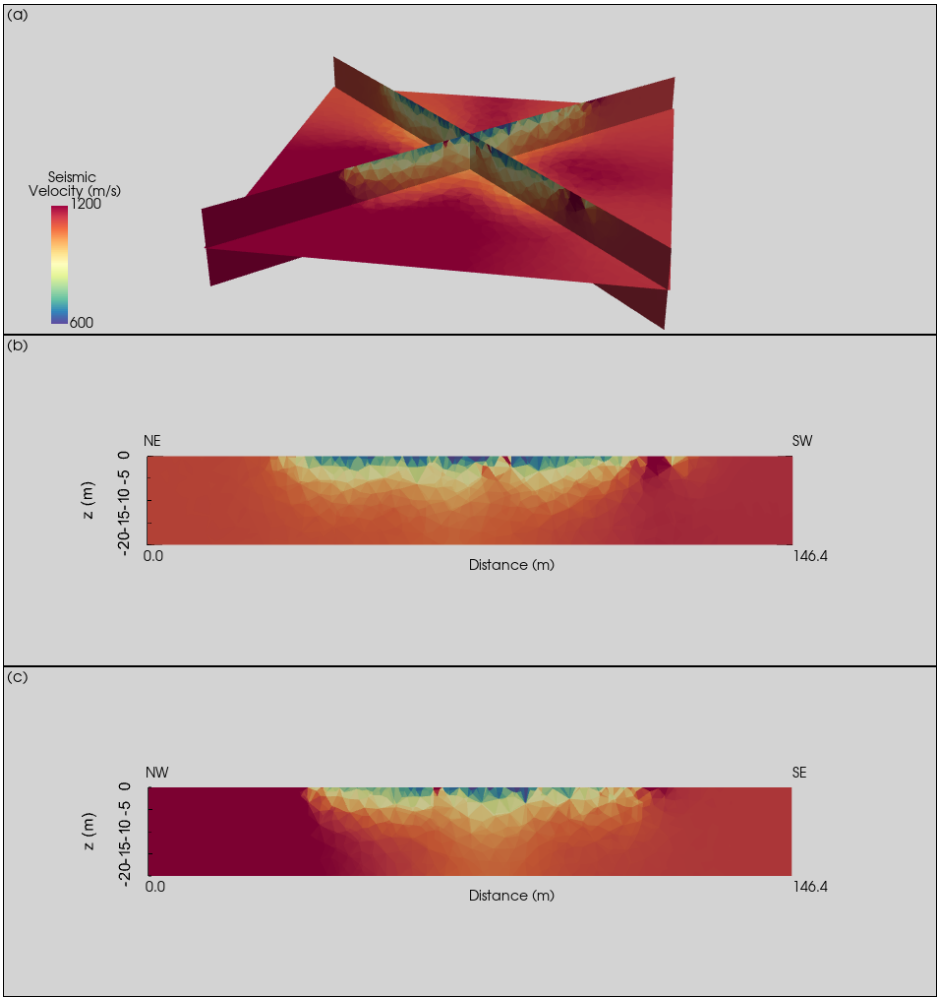


Figure 16: Subsurface model of the soda lake in terms of the seismic P-wave velocity. (a) 3D representation of orthogonal slices through the resolved subsurface model. (b) 2D representation of the NE–SW slice. (c) 2D representation of the NW–SE slice.

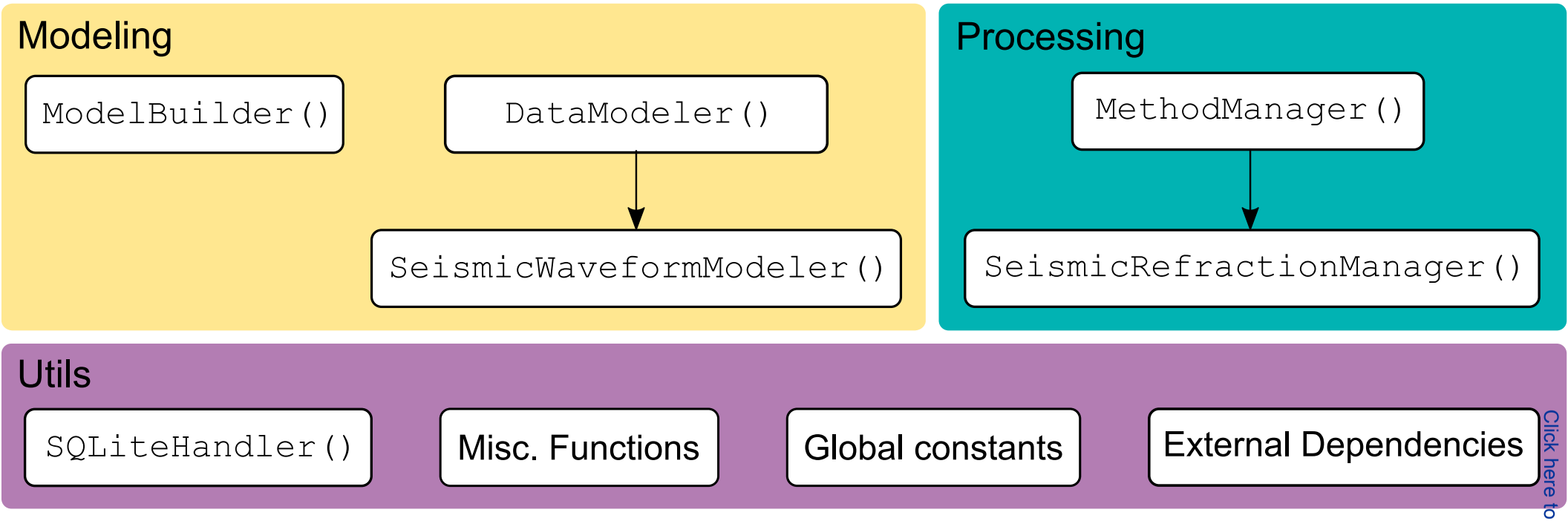


Figure 02

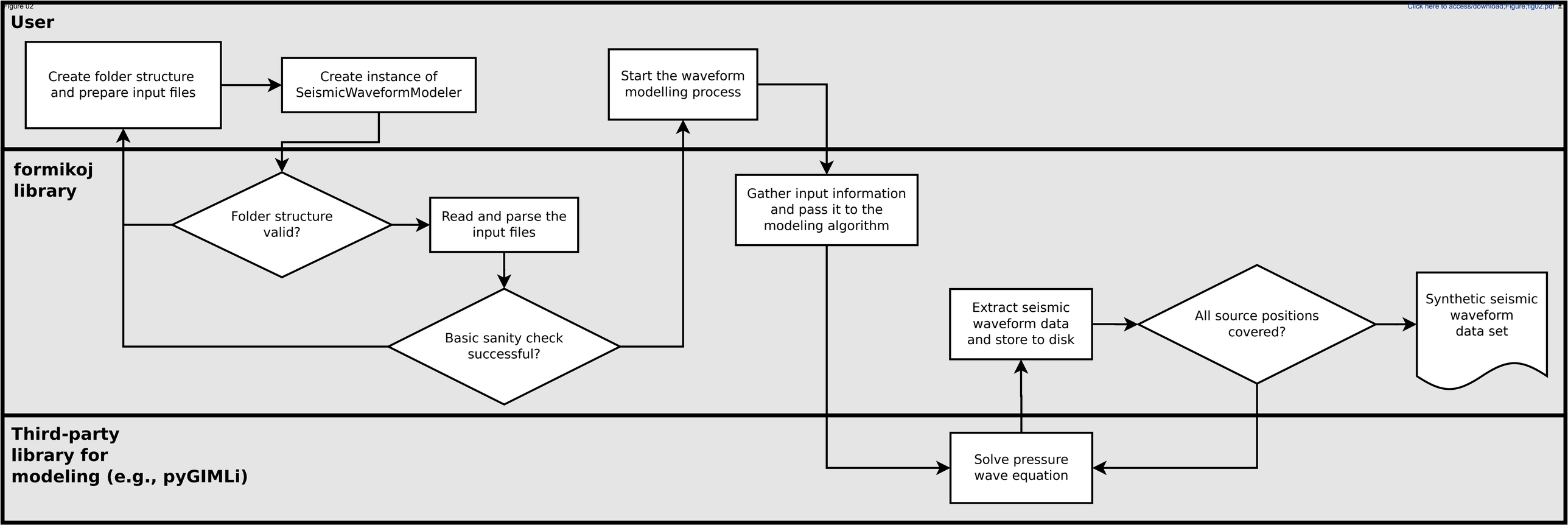


Figure 03

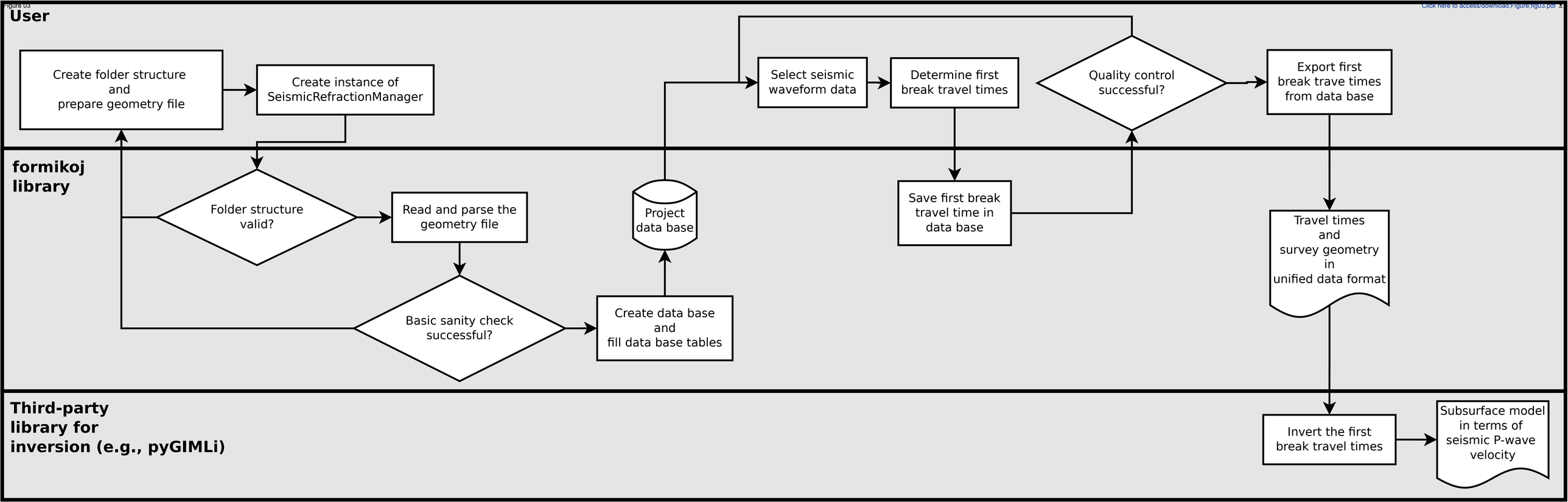


Figure 04

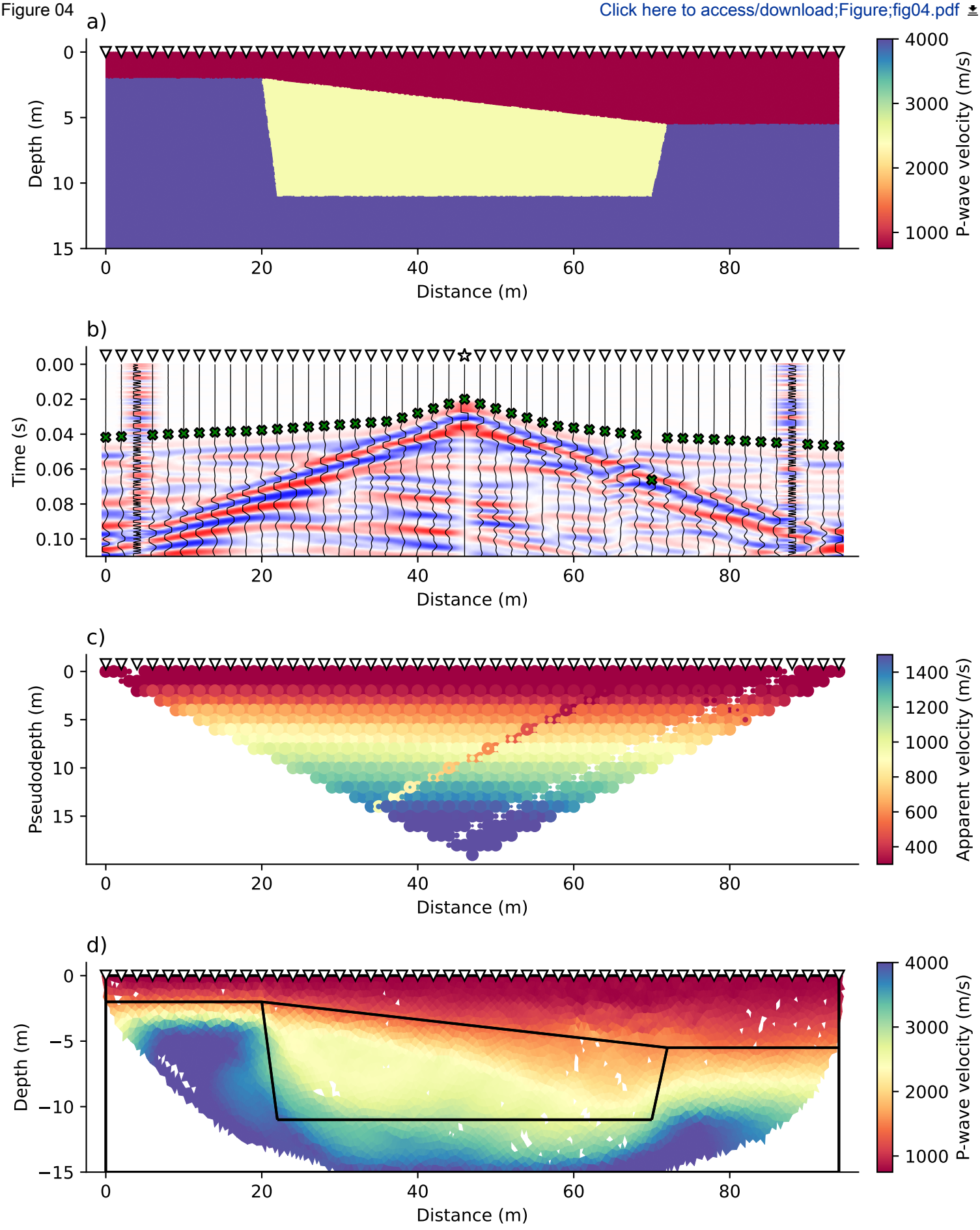


Figure 05

Click here to access/download;Figure;fig05.eps

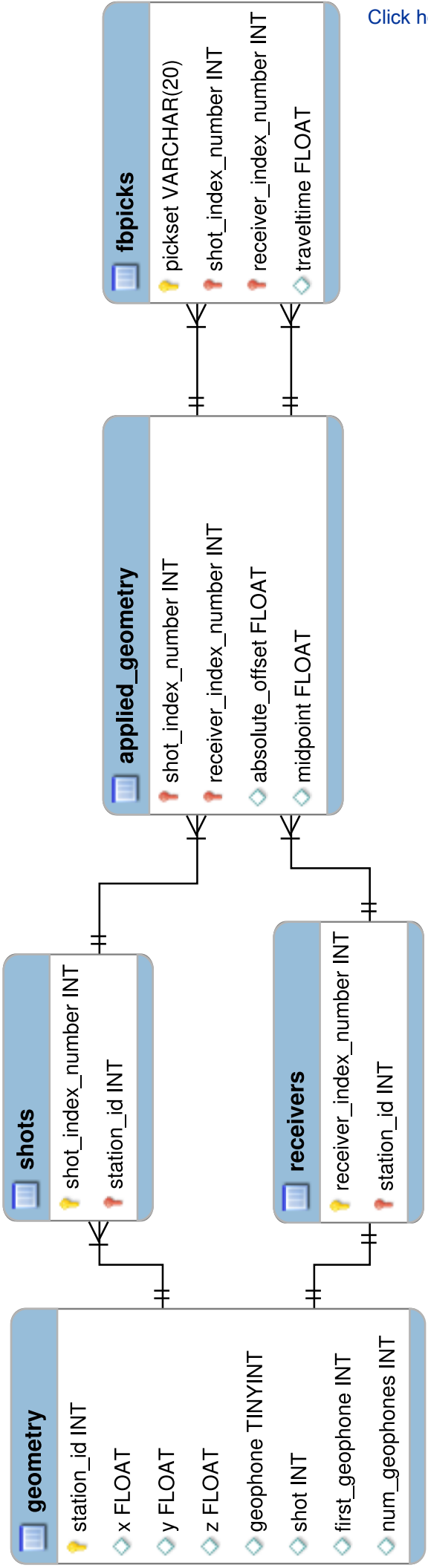


Figure 06

Click here to access/download;Figure;fig06.pdf

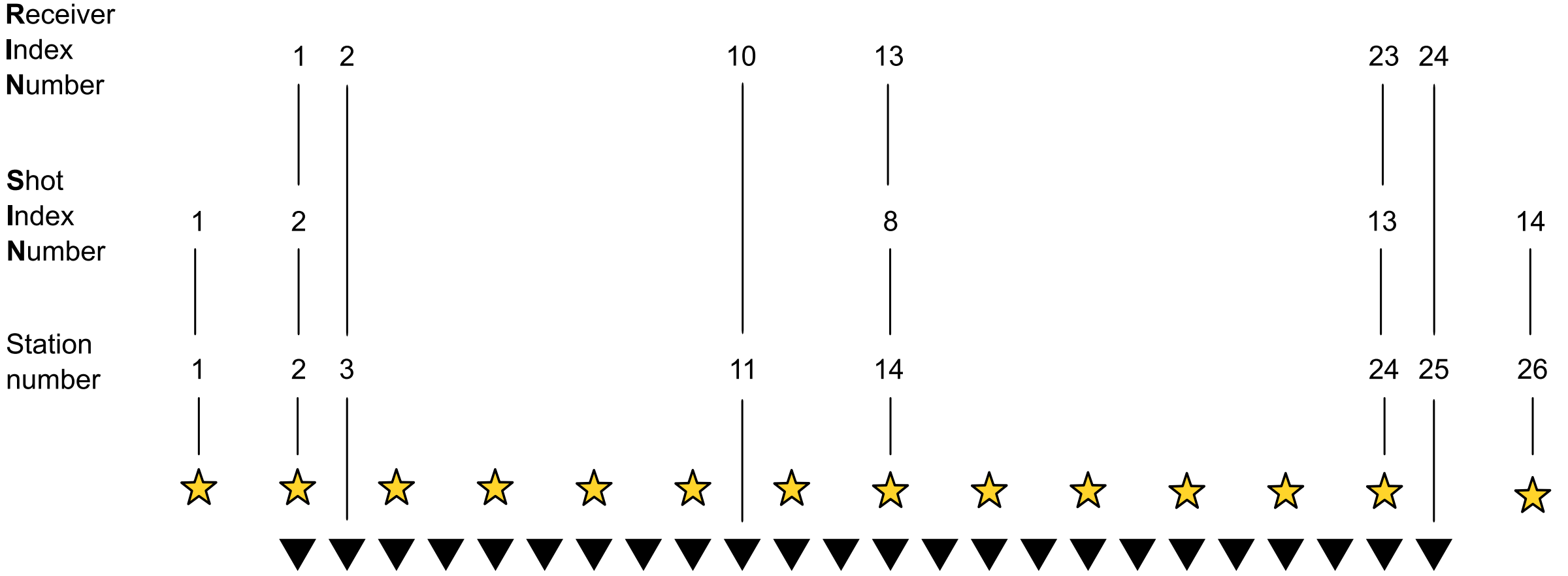


Figure 07

Click here to access/download;Figure;fig07.pdf

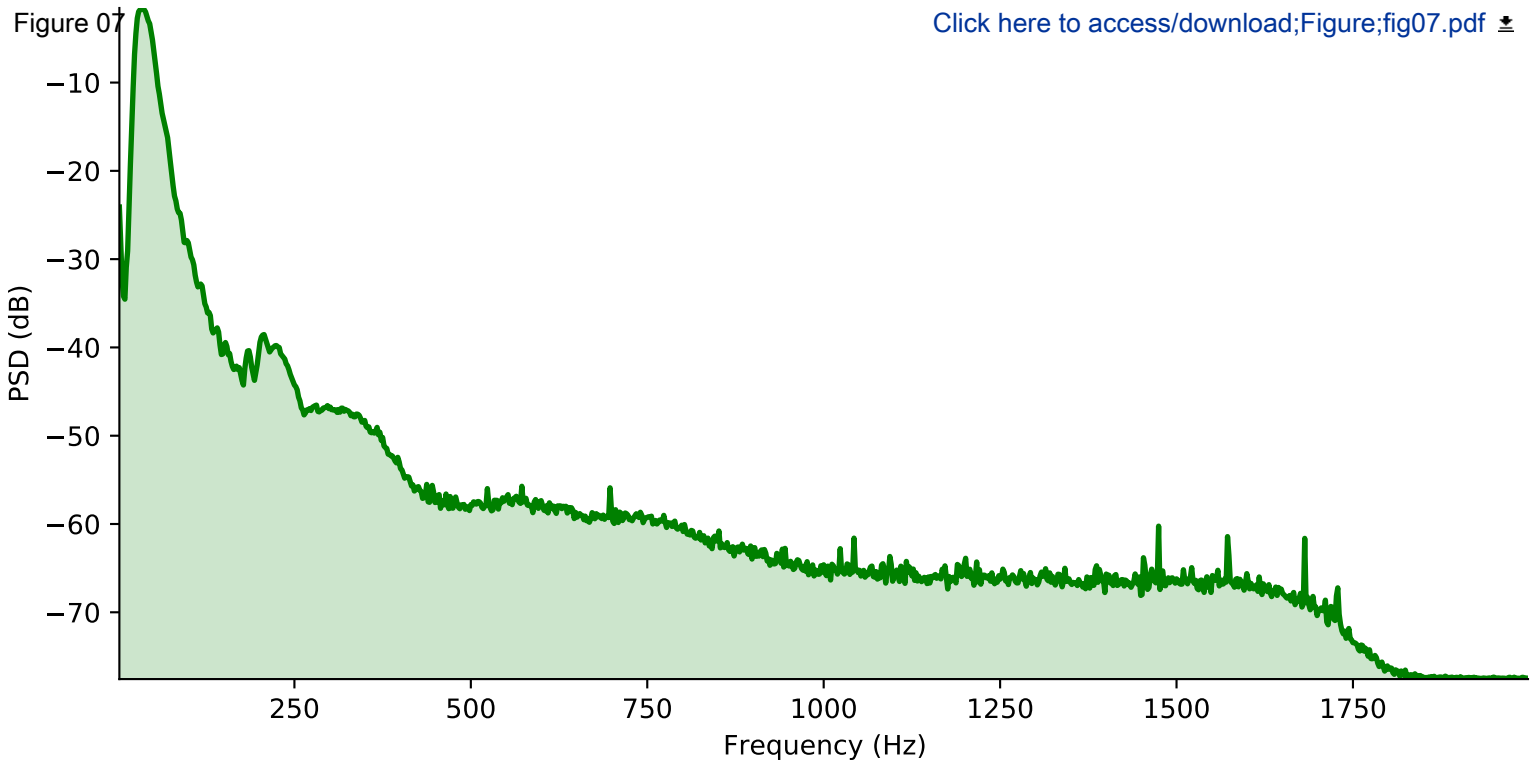
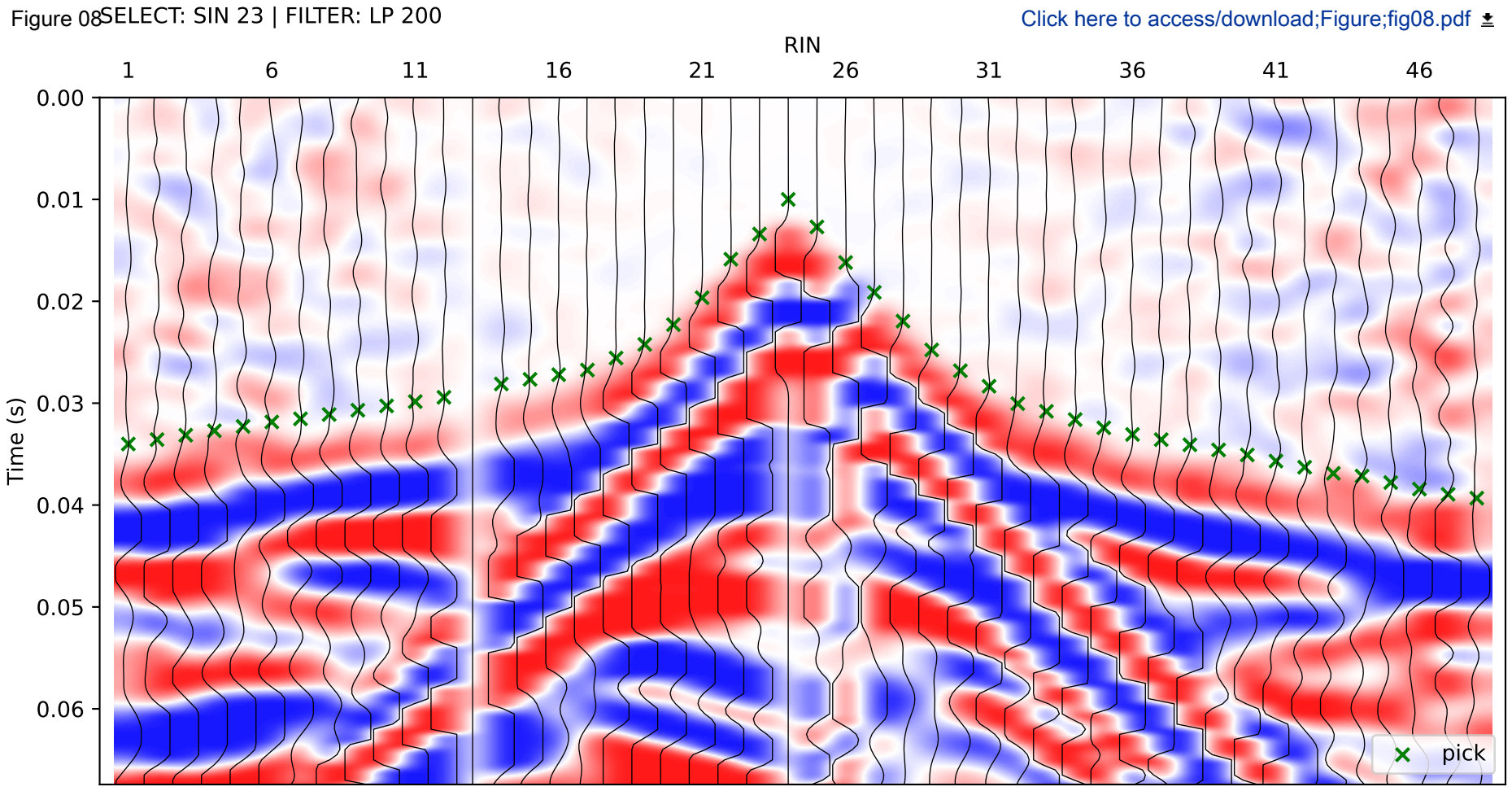


Figure 08



Click here to access/download;Figure;fig08.pdf

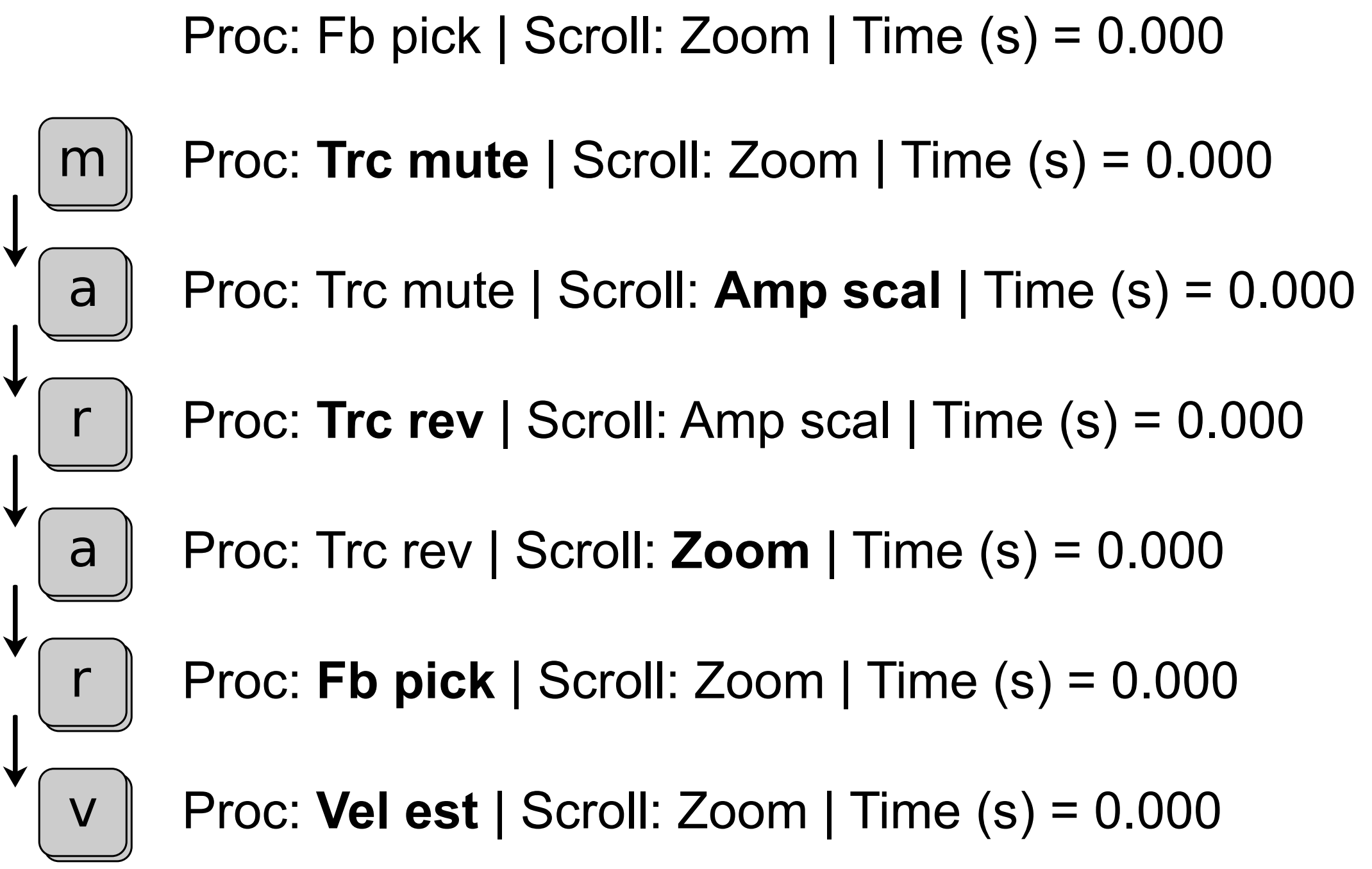


Figure 10

Click here to access/download;Figure;fig10.pdf

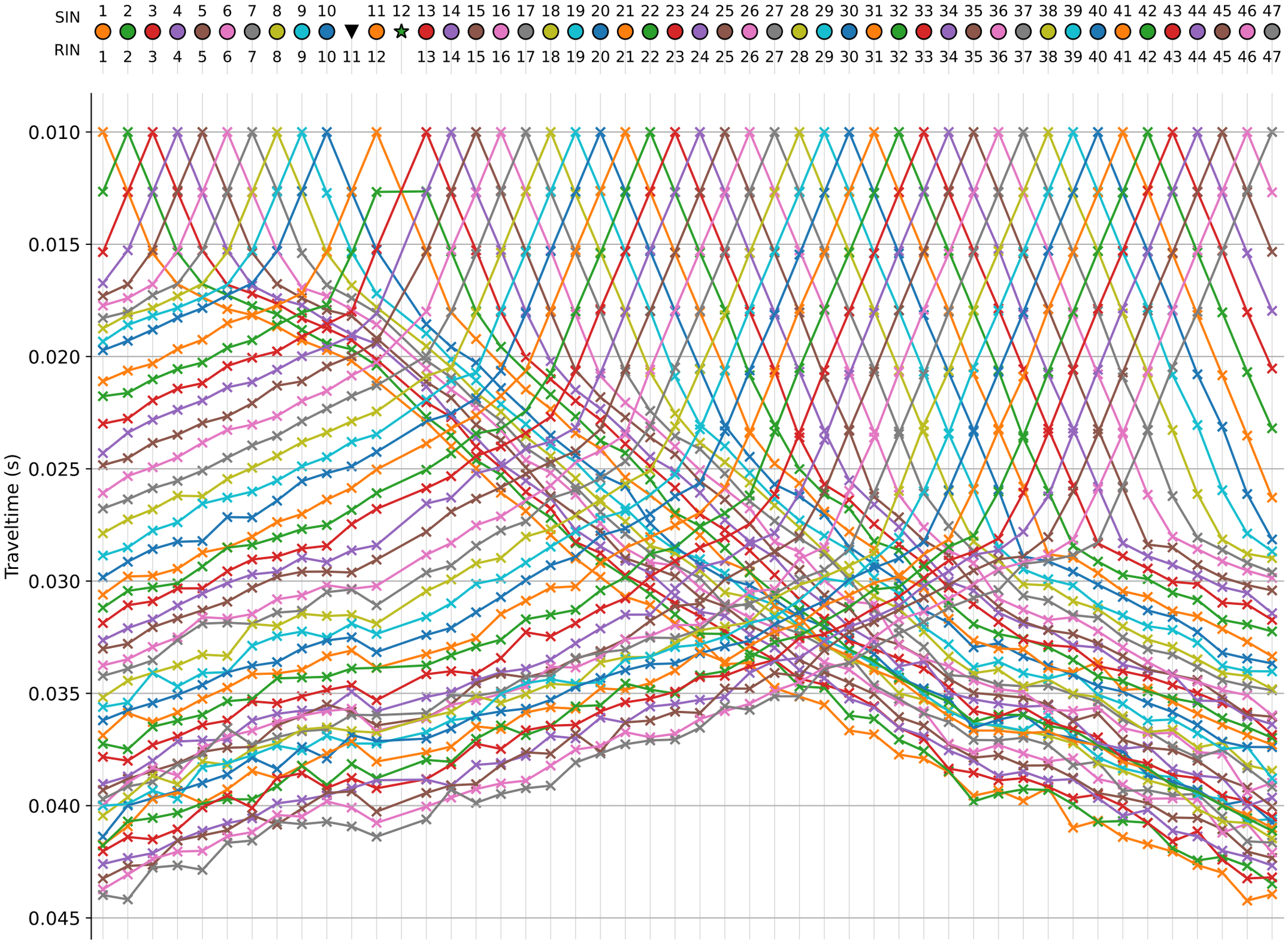


Figure 11

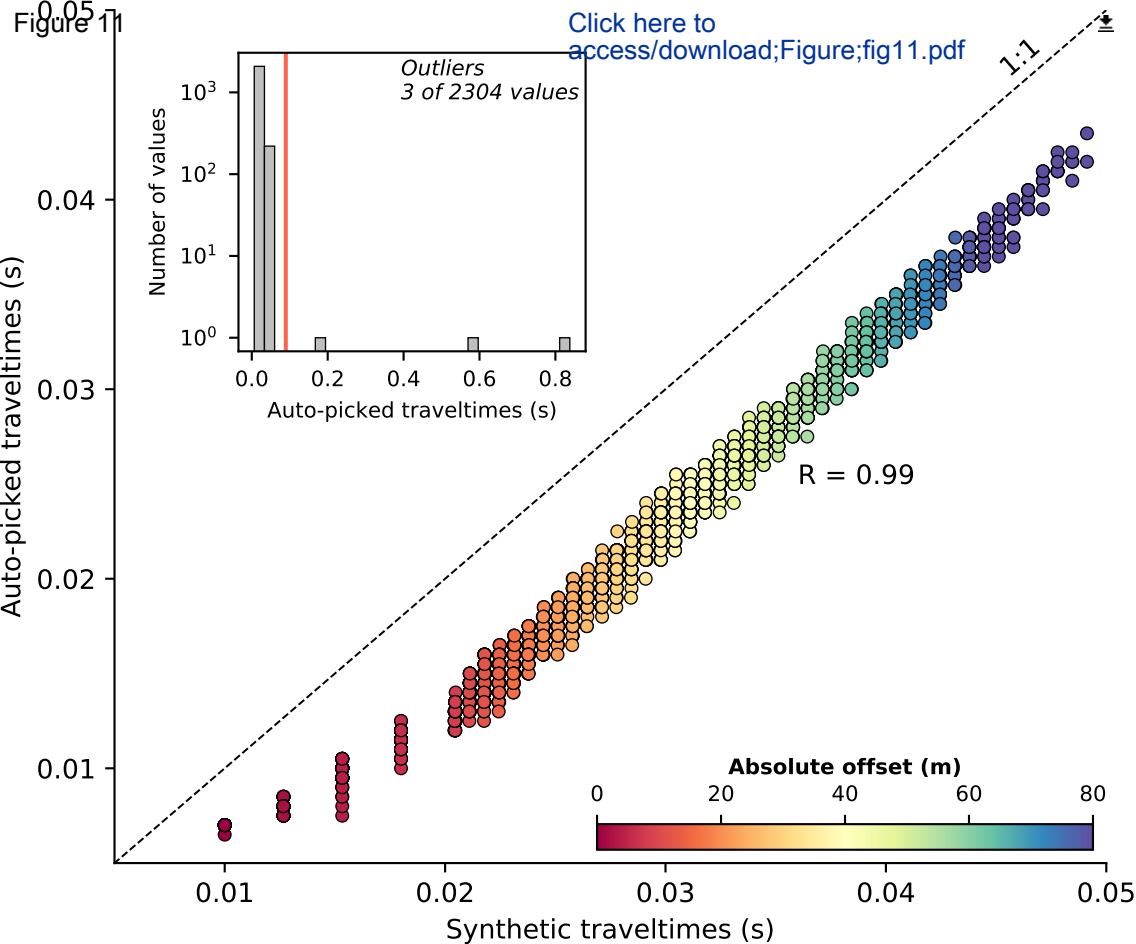


Figure 12

Click here to access/download;Figure;fig12.pdf

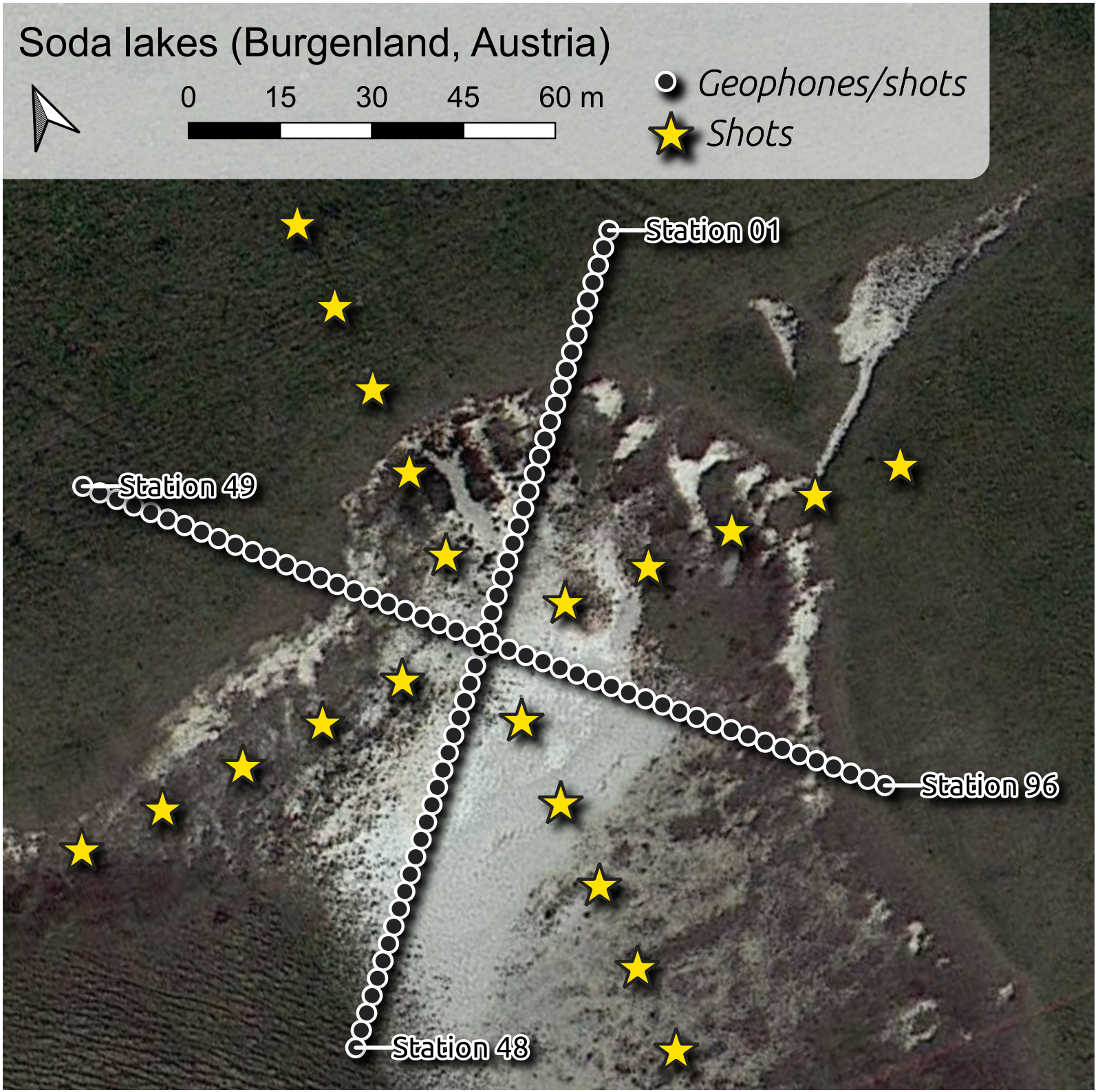
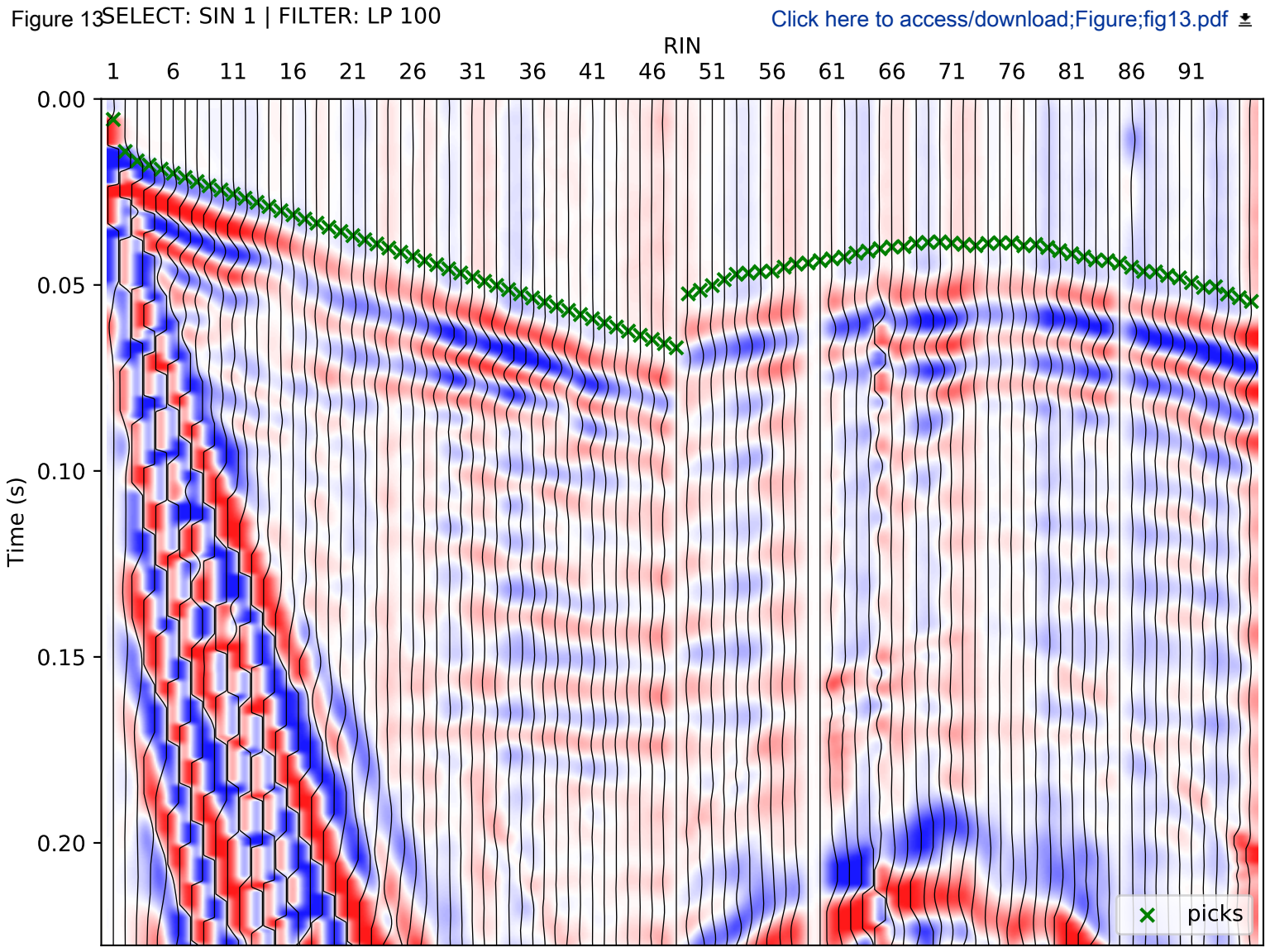


Figure 13



Click here to access/download;Figure;fig13.pdf

Figure 14

Click here to access/download;Figure;fig14.pdf

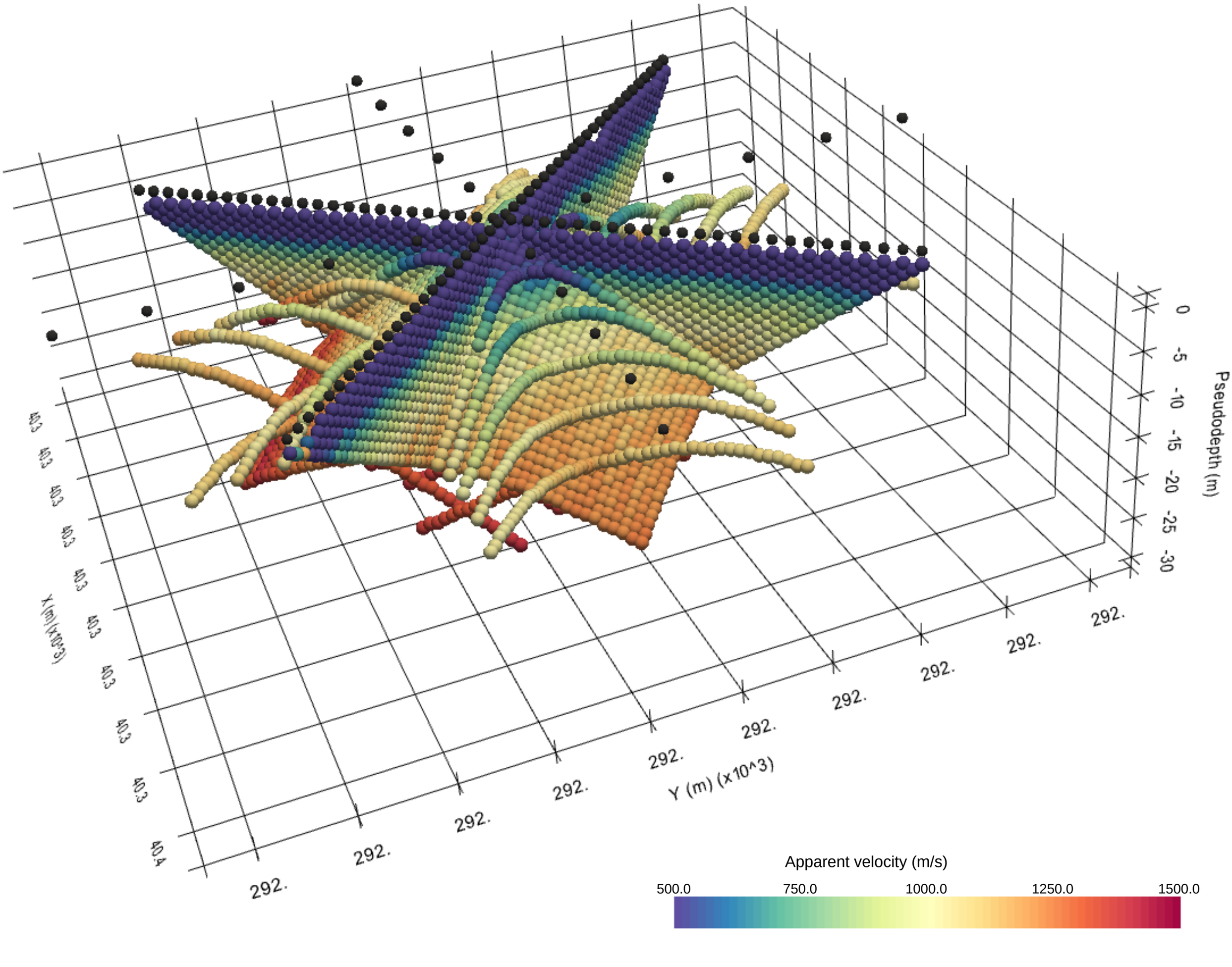


Figure 15

Click here to access/download;Figure;fig15.pdf

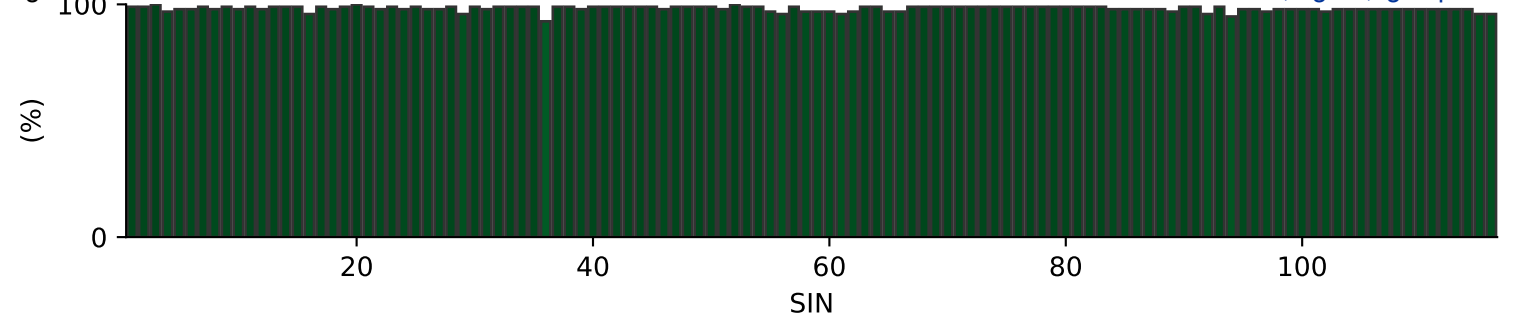
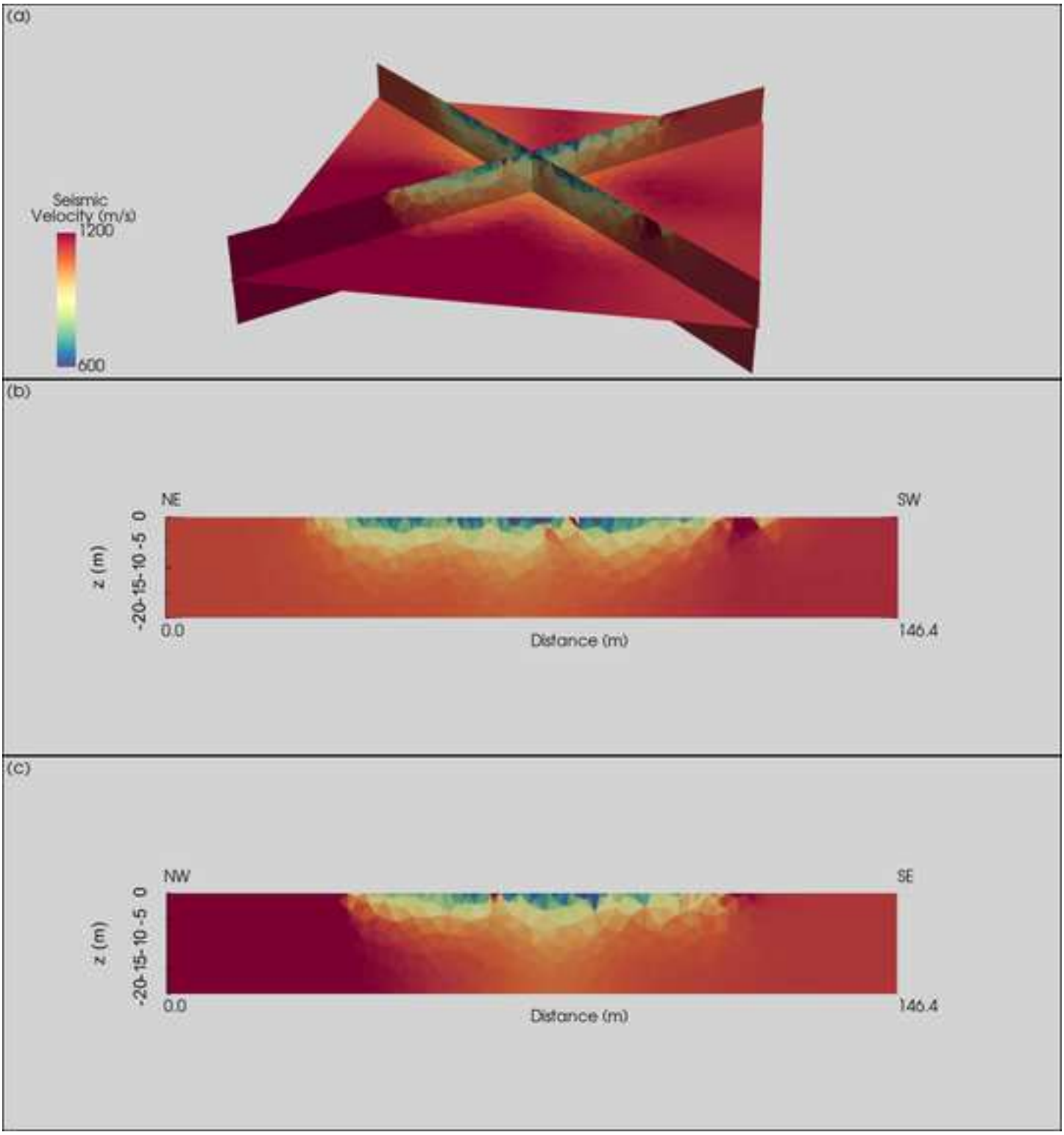


Figure 16

Click here to access/download;Figure;fig16.png



[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 Editor-in-Chief, dear reviewers

thank you very much for the positive evaluation of our manuscript as well as for providing numerous constructive comments and suggestions for improvement. In the revised version of our manuscript "formikoj: A flexible library for data management and processing in geophysics - Application for seismic refraction data", we have attempted to implement every suggestion and address all comments.

In particular, we present theoretical details regarding the considered and implement methodologies, provide a self-contained synthetic study demonstrating the validity of the forward modeled seismic waveform data, and the fundamental processing capabilities of the proposed library for seismic refraction data. Moreover, we discuss a single field data application in order to provide a more concise demonstration of the library functionalities with regard to the processing of real seismic survey data.

Moreover, we modified the library according to the suggestions provided by the reviewers and provide the revised source codes in a public repository with details listed in the section "Code availability".

We hope that our revisions and replies alleviate the concerns of the reviewers and that you find our study suitable for publication in Computers & Geosciences. We look forward to your decision.

Yours sincerely,

Matthias Steiner and Adrián Flores Orozco
Research Unit of 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 of 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 for the forward modeling of seismic refraction waveform data with the `SeismicWaveformModeler` based on a custom subsurface model and survey geometry. We use these synthetic seismic data set to demonstrate the fundamental seismic refraction processing capabilities of the `SeismicRefractionManager`; thus, illustrating the ability to combine modeling and processing tasks in a single workflow. Based on a 3D field dataset we present the available range of possibilities provided by the formikoj library for the processing of seismic refraction survey data. In particular, we explore different visualization techniques of the seismic traveltime readings to enhance their consistency prior to the inversion with the third-party library pyGIMLi. The low-level access provided by the formikoj library aims at enabling users to implement novel modeling, visualization and processing tools specifically designed for their objectives as well as 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, and even for teaching in developing countries. A common limitation of existing 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 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.,

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

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 for teaching and interpretation purposes, testing of research hypotheses, evaluating the accuracy of different measurement configurations or inversion strategies.

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[1]), 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 SRT 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 a carefully designed synthetic study highlighting the capabilities of the formikoj library for the forward modeling and processing of 2D seismic refraction datasets. Based on a 3D field dataset we illustrate that the formikoj library also allows for the processing of complex survey geometries, where the obtained first break traveltimes can be inverted with third party open-source libraries to solve for 3D subsurface models expressed in terms of the seismic

[1] http://resistivity.net/bert/data_format.html, last accessed on March 11, 2023

P-wave velocity.

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. The formikoj library provides a well thought out data management concept based on the SQLite database engine that does not require a separate server process [2]. In particular, the information for each project, such as survey geometry and first break traveltimes, are stored in a SQLite database file. Using such an application file format facilitates the cross-platform design of the formikoj library, and provides fast I/O operations through concise SQL queries. The portability of the application file allows for an easy exchange of projects between partners across institutions relying on different IT infrastructures. Accordingly, the formikoj library aims at providing a transparent and customizable framework for the collaborative design of reproducible workflows. The comprehensive usage of clear error and log messages aims at supporting the user throughout the modeling and processing workflows. A meticulous exception handling ensures that the data stored in the SQLite project database is not corrupted in case of erroneous input. Moreover, documenting the user input and the respective responses of the formikoj library with the python logging module provides a timestamped command history that further enhances the transparency and repeatability of the conducted workflows.

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 Figure 3, respectively. These flow charts illustrate the corresponding workflows as well as the required interactions between the user, the formikoj library and third-party packages. As can be seen from Figure 2 and Figure 3, the formikoj library acts as an interface between the user and more complex functionalities of third-party libraries, such as pyGIMLi for modeling and inversion of seismic refraction data. 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. To avoid redundancies in the implementation we built these classes 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.

The formikoj library is primarily developed on machines running Linux Mint 20 or Kubuntu 22.4, respectively. Testing of the library refers to carrying out the fundamental use cases for modeling and processing of seismic refraction

[2] https://www.sqlite.org/index.html, last accessed on March 11, 2023

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

data presented in Figure 2 and 3. To ensure the cross-platform applicability of the formikoj library we test these fundamental use cases also on machines running on Windows 10, Windows 11 as well as macOS versions 12 and 13.

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 for P-wave refraction modeling 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
```

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. Here, the user can define simple models with all layers considered to be parallel to the surface topography, which is inferred from the station geometry provided in the measurement scheme. The seismic velocity values for the different model regions (`velmap`) and the corresponding thickness of the different geological units (`layers`) have to be explicitly defined in the configuration file. The remaining parameters (`quality`, `area`, `smooth` and `sec_nodes`) refer to the properties of the mesh that is eventually used for the forward modeling of the seismic waveform data and corresponding first break traveltimes (we refer to the respective pyGIMLi resources[3] for further information). In the third block of the exemplary configuration file (`dataset`), the user can set specific

[3] https://www.pygimli.org/pygimliapi/_generated/pygimli.meshtools.html#pygimli.meshtools.createMesh, last accessed March 11, 2023

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 provided in the `dataset` block control the random error (`noise` and `noise_level`) as well as systematic errors (`missing_shots`, `broken_geophones` and `wrong_polarity`) in the modeled seismic waveform data (see Table 2 for a detailed description). 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 final block of the configuration file (`traveltimes`) contains data-error parameters for both the absolute error (e_{abs}) and the relative error (e_{rel}) defined by the user. The data error model is then estimated as

$$\boldsymbol{e} = e_{abs} + \boldsymbol{d}\, e_{rel}\,, \tag{1}$$

where $\boldsymbol{d}$ denotes the forward modeled data not affected by noise, i.e., here the first break traveltimes obtained through the Dijkstra algorithm (Dijkstra, 1959). These computed error values are subsequently added to the data to obtain the noisy data as

$$\boldsymbol{d}_{noise} = \boldsymbol{d}\,(1 + \boldsymbol{e})\,. \tag{2}$$

The configuration file needs to be provided via the input directory from where it can be imported through the `load` method of the `SeismicWaveformModeler`:

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

```
INFO    : Configuration loaded
```

To demonstrate the ability to model seismic waveform data for arbitrary subsurface conditions we do not define a simple subsurface model in the configuration file but provide a more complex model in the binary mesh format (e.g., a bms file). 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 corresponding python script can be found 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:

```
# Load the mesh into the workflow
swm.load(type='mesh')
```

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

```
INFO    : Mesh loaded
```

The forward modeling process generating the seismic waveform data is then invoked through the `create` method:

```
# Start the modeling of the seismic waveform data
swm.create(type='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 considered for the waveform modeling in this study (see Figure 4a). The model consists of a top and a bottom layer with varying thickness characterized by seismic velocity values of 750 ms^{-1} and 4000 ms^{-1}, respectively. In the center, the model contains an irregularly shaped anomaly associated with a seismic velocity of 2500 ms^{-1}, i.e., the model features vertical and lateral variations in the seismic velocity distribution. The third step refers to the generation of a Ricker wavelet through the pyGIMLi function `ricker` as

$$u = \left(1 - 2\pi^2 f^2 t^2\right) e^{-\pi^2 f^2 (t-t_0)^2} \tag{3}$$

based on the wavelet properties provided in the configuration file. In Equation 3, f is the frequency of the wavelet given in Hz, t refers to the time base definition, i.e., the length and resolution of the wavelet in the time domain, and t_0 is the time offset of the wavelet.

Based on the mesh, the velocity model and the Ricker wavelet we use pyGIMLi to solve the pressure wave equation for each shot station defined in the measurement scheme. The resultant seismograms are extracted at the corresponding geophone stations, gathered for each shot position in an ObsPy `Stream` object 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
```

In particular, 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* indicates 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: 0.512 s
  Number of geophones: 48
  Geophone spacing: 2 m
############################

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

The auxiliary file info.txt exported to the dataset directory summarizes the parameters from the configuration file as well as information regarding the simulated systematic errors in the synthetic seismic waveform data, e.g.:

```
Number of geophones: 48
Number of shots: 48
Recording length (s): 0.512
Sampling frequency (Hz): 2000
Wavelet type: Ricker
Frequency of the wavelet (Hz): 100

Missing shot(s): 11
Broken geophone(s): 13
Wrong polarity geophone(s): 26
```

Figure 4b presents the seismic waveform data forward modeled for a shot point located in the center of the profile. The seismograms are shown as curves, whereas the strength of the amplitudes is added as color-coded information. In the seismograms we see clear first onsets along the entire profile, yet receivers 3 and 45 were modeled as broken geophones, i.e., the corresponding seismograms contain solely noise. 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, to demonstrate how such outliers can be identified in a so-called pseudosection.

Pseudosections are a useful tool for the analysis of the data quality by presenting the data based on the positions of shots and receivers. Such a visualization allows for the detection of outliers and their spatial distribution as necessary to understand possible sources of error. In case of the SRT, a pseudosection as presented in Figure 4c, illustrates apparent velocity (v_{app}) values computed as

$$v_{app} = \frac{aoffset}{tt}, \tag{4}$$

where $aoffset$ refers 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. The inversion of first break traveltimes aims at resolving a model of the P-wave velocity of the subsurface materials, where systematic errors in the data would lead to the creation of artifacts in the imaging results or hinder the convergence of the inversion. Considering the multitude of available inversion frameworks we do not include a new inversion strategy in the formikoj library. Instead the processed data can be exported in any format required for the use of commercial inversion algorithms, or can be linked directly to other open-source libraries. In our case, we refer to the modeling and inversion capabilities of pyGIMLi (Rücker et al., 2017). pyGIMLi uses a generalized Gauss-Newton method to solve the inversion problem through the minimization of an objective function given as:

$$\| \boldsymbol{W}_d \left(\mathcal{F}(\boldsymbol{m}) - \boldsymbol{d} \right) \|_2^2 + \lambda \| \boldsymbol{W}_m \left(\boldsymbol{m} - \boldsymbol{m}_0 \right) \|_2^2 \rightarrow \min \tag{5}$$

The first term on the right-hand side of Equation 5 refers to the data misfit. $\boldsymbol{W}_d$ is the data weighting matrix holding the reciprocals of the data errors, $\boldsymbol{d}$ denotes the data vector holding the input data (first break traveltimes), $\boldsymbol{m}$ is the model vector representing the target parameter (seismic P-wave velocity values), and $\mathcal{F}(\boldsymbol{m})$ is the model response. The second term denotes represents the regularization, where $\boldsymbol{W}_m$ and $\boldsymbol{m}_0$ are the model constraint matrix and the reference model, respectively. The regularization parameter λ balances the influence of the data misfit and 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, which is given in terms of the spatial variations of the seismic P-wave velocity, i.e., reflecting the associated lateral and vertical variations. Depending on the survey geometry the inversion can resolve subsurface models in both 2D or 3D. 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); yet, the exploration of different inversion strategies is beyond the scope of this study. 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 seismic refraction (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 the seismic velocity model obtained through the inversion. Accordingly, a proper enhancement of the perceptibility of the first onsets is crucial.

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. The first break traveltimes determined with the `SeismicRefractionManager` represent the input data, e.g., for the `TravelTimeManager` of pyGIMLi (Rücker et al., 2017). This is particularly relevant as the `TravelTimeManager` provides an inversion framework for first break traveltimes, yet not a framework for the processing of seismic waveform data. For the demonstration of the fundamental `SeismicRefractionManager` capabilities we consider the synthetic seismic data created with the `SeismicWaveformModeler` above, which illustrates that both classes can be combined in subsequent workflows.

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. For the synthetic dataset considered in this study, the 2D station coordinates are provided in the geometry file, whereas the z coordinate is assumed to be 0 m along the entire profile, as illustrated in Table 4; thus, reflecting the lack of surface topography in the synthetic model (see Figure 4a). In the column **Geo** the geometry file contains the value 1 (True) for each station except the station located at 24 m referring to the broken geophone modeled by the `SeismicWaveformModeler`. When compiling the geometry file we also need to take into account the missing shot at station 11, i.e., the station located at 20 m along profile direction, and set the corresponding value to −1. The column **1st Geo** indicates the first active geophone along the profile for each shot file, which for the synthetic dataset considered here is the geophone deployed at the first station along the entire profile. In particular, the column **1st Geo** allows the reproduction of roll-along survey geometries, where in the first segment the first active geophone is always the geophone at station 1. Considering 48 geophones deployed in each segment, and an overlap of 50 %, the first geophone for the consecutive segments would be 25, 49, 73, 97, and so on.

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. Depending 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

Table 4
Excerpt from the geometry file describing the survey geometry of the synthetic dataset generated in this study.

x (m)	y (m)	z (m)	Geo	Shot	1st Geo	# Geo
0.0	0.0	0.0	1	1001	1	48
2.0	0.0	0.0	1	1002	1	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
20.0	0.0	0.0	1	-1	1	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
24.0	0.0	0.0	0	1013	1	48
⋮	⋮	⋮	⋮	⋮	⋮	⋮
92.0	0.0	0.0	1	1047	1	48
94.0	0.0	0.0	1	1048	1	48

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`), a common RIN (`rin`), or a common SIN (`sin`)

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

```
INFO    : 88 traces selected
```

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

```
INFO    : 48 traces selected
```

```
# Select traces with receiver
srm.select(by='sin', num=23)
```

```
INFO    : 48 traces selected
```

Figure 7 shows the frequency spectrum of the currently selected traces, which can be computed and visualized through the `plot` method:

```
# Plot the frequency spectrum
srm.plot(type='spectrum')
```

The `SeismicRefractionManager` resolves the frequency spectrum by computing the stacking the power spectral density (psd) of the seismograms $s(t)$ as

$$psd = 10 \log_{10} \left(\frac{|\mathrm{fft}(s(t))|^2}{N} \right), \tag{6}$$

where fft refers to the fast fourier transformation (FFT) and N is the number of the considered seismograms. The frequency spectrum provides information regarding the amplitudes associated to different frequencies in the seismograms, which can be use for the identification of the dominating frequencies as required for the selection and application of adequate filters such as lowpass, highpass, bandpass and bandstop implemented in ObsPy (Beyreuther et al., 2010). To enhance the signal-to-noise ratio of the seismograms the user can apply the respective filters 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.:

```
# Apply bandpass filter
srm.filter(type='bandpass', freqmin=10, freqmax=120)
```

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

In this example, the filter is solely applied to the currently selected traces, yet setting the parameter `onhold` as `True` automatically filters all subsequently selected traces with the same filter settings, e.g.:

```
# Apply lowpass filter
srm.filter(type='lowpass', freq=200, onhold=True)
```

```
INFO    : Applied 200.0 Hz lowpass filter
INFO    : Set filter hold on
```

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

```
# Plot selected traces
srm.plot()
```

By default, 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 between variable area and variable density mode, with the latter reflecting the strength of the amplitudes by the color saturation, i.e., high amplitudes refer to a stronger shade than low amplitudes (see Figure 8). 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 9). The initial processing mode is 'Fb pick', i.e., first break picking is possible. Additional modes, accessible through the keyboard, allow for an enhanced visualization of the seismograms, e.g., associated to broken geophones or seismograms with wrong polarity. 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 9).

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:

```
# Plot traveltime diagram
srm.plot(type='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 respective data point 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. Once the user is satisfied with the quality of the first break traveltimes, the data points of the pickset can be exported in the unified data format, which is compatible with the pyGIMLi framework. In particular, the udf file is stored in the subdirectory *03_proc/picks* with the current timestamp as suffix:

```
# Export first break traveltimes
srm.picksets(do='export', name='pick')
```

```
INFO    : Pickset 'pick' saved to pick_20230303T140447.pck'
```

2.3. Expanding the capabilities of the formikoj library

Making the formikoj library publicly available under an open-source license allows the addition of supplementary functionalities tailored for the specific requirements of the users. The concept of formikoj suggest that such custom extensions should be implemented either as internal methods or as functions in the utilities module, which are then executed through the `compute` method.

We illustrate this possibility for customization by implementing a simplified version of an automatic first break picking algorithm based on the short- and long-time window average ratio (STA/LTA) method (Allen, 1978), which determines the traveltimes following the energy ratio approach (e.g., Earle and Shearer, 1994). In particular, our implementation computes the envelope $E(t)$ of the seismogram $s(t)$ as (e.g., Duan and Zhang, 2020)

$$E(t) = \left(s(t)^2 + \tilde{s}(t)\right)^{1/2} , \tag{7}$$

where $\tilde{s}(t)$ is the Hilbert transform of $s(t)$. The energy ratio ER is then computed as $ER = STA/LTA$ with

$$STA(i) = \frac{1}{n_{STA}} \sum_{j=i-n_{STA}}^{i} E(j) , \tag{8}$$

and

$$LTA(i) = \frac{1}{n_{LTA}} \sum_{j=i-n_{STA}}^{i} E(j) , \tag{9}$$

where n_{STA} and n_{LTA} refer to the number of data points in $E(t)$ considered for the short- and long-time windows, respectively. For this exemplary implementation we determine the first break traveltimes in the seismograms as the position of the maximum in the ER function, which is automatically saved in the project database.

The autopicking algorithm is added to the `SeismicRefractionManager` in form of two internal methods `_manage_autopicking` and `_compute_autopicks`, respectively. To invoke the autopicking process we modified the `compute` method, which now accepts the custom-defined keyword `autopicking`. Additionally, the autopicking requires values to be passed for the parameters `pick` and `pickset`. The former accepts the values `all` or `cur` to determine first break traveltimes for all traces in the dataset or solely the currently selected traces, respectively. The latter defines the name of the picksets in which the automatically determined traveltimes should be saved to. A typical use case involves conducting the autopicking and exporting the determined traveltimes:

```
# Automatically pick first break traveltimes
srm.compute(do='autopicking', pick='all', pickset='autopicks')
srm.picksets(do='export', name='autopicks')
```

```
INFO    : Created new pickset 'autopicks'
INFO    : Pickset 'autopicks' loaded
INFO    : 'autopicks' set at active pickset
Progress <==============================> 100.0% completed
INFO    : Pickset 'autopicks' saved to autopicks_20230303T140834.pck
```

In Figure 11, we compare the automatically determined first break picks with the forward modeled traveltimes computed above to allow for a basic evaluation of autopicking performance. The inset plot in Figure 11 presents the histogram of the autopicked traveltimes, which shows that three traveltimes should be considered outliers, and thus are removed from the dataset. After the outlier removal the correlation coefficient between forward modeled and autopicked traveltimes is 0.99 suggesting that the implemented autopicking algorithm performs well for the synthetic seismic waveform data. However, the observed deviation from the perfect correlation (i.e., the 1:1 line in Figure 11) indicates that autopicking and forward modeling algorithm might be sensitive to different seismic wave phases. Further improvements in terms of the autopicking process could incorporate, for example, machine learning approaches as the method proposed by Duan and Zhang (2020), which can be easily implemented as an additional functionality in the `SeismicRefractionManager`. We point out here, that following the same procedure, the user can implement further autopicking algorithms as well as other data processing strategies and have a simple framework for the evaluation of their performance through the analysis of synthetic data (e.g., clean and contaminated with Gaussian noise).

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

To demonstrate the applicability of the `SeismicRefractionManager` for the processing of real field data, we present here the analysis and inversion inversion results for a seismic refraction survey conducted in a soda lake located in the Nationalpark Neusiedler See–Seewinkel close to Vienna. The seismic survey aims at solving for the geometry of the confined aquifer below this soda lake with the required information referring to the thickness of the aquifer and the depth of the groundwater table. As presented in Figure 12, 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 to obtain a sufficient coverage.

As in case of the synthetic dataset presented above, the geometry file contains the coordinates of the survey stations, yet for the soda lake dataset 3D coordinates we provided as illustrated in Table 5. Since geophones were not

Table 5
Excerpt from the 3D survey geometry file.

x (m)	y (m)	z (m)	Geo	Shot	1st Geo	# Geo
40336.056	292470.692	120.0	1	1001	1	96
40334.751	292469.177	120.0	1	1002	1	96
⋮	⋮	⋮	⋮	⋮	⋮	⋮
40284.472	292455.431	120.0	1	1058	1	96
40285.896	292454.027	120.0	1	1059	1	96
⋮	⋮	⋮	⋮	⋮	⋮	⋮
40339.421	292402.681	120.0	1	1096	1	96
40305.228	292445.050	120.0	0	1097	1	96
⋮	⋮	⋮	⋮	⋮	⋮	⋮
40265.415	292431.957	120.0	0	1115	1	96
40255.453	292431.395	120.0	0	1116	1	96

deployed at each survey station the column **Geo** contains the value 1 (True) for the first 96 stations and 0 (False) for the remaining 20 stations. Based on the shot files and the geometry file stored in the required directory structure the `SeismicRefractionManager` creates the project database, automatically infers the 3D survey layout from the 3D survey geometry, and thus configures the project for 3D processing. Then we can select seismograms the same way as for the synthetic data presented above. For this example we select seismic waveform data recorded for SIN 1, i.e., the shot position co-located with the first geophone (Station 01 in Figure 12):

```
# Select traces with receiver
srm.select(by='sin', num=1)
```

```
INFO    : 96 traces selected
```

In Figure 13, we can see that 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 for assessing the quality of the first break traveltimes. However, the `SeismicRefractionManager` project is configured for 3D processing, and thus 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 14 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.

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

```
# Plot the picking percentage
srm.plot(type='pickperc')
```

Figure 15 presents the picking percentage visualized separately for each SIN. Accordingly, a low picking percentage indicates shots affected by a low signal-to-noise ratio, whereas clear first onsets yield a correspondingly high picking percentage. For the soda lake dataset we observe a consistently high picking percentage for all shot positions; thus, indicating a good data quality. Moreover, the picking percentage plot can be used to track the picking progress, for instance, in case the traveltimes cannot be determined in frame of 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.

Once the first break picking is finished, the corresponding pickset can be exported for the inversion. We inverted the first break traveltimes with pyGIMLi and present the resolved 3D subsurface model in Figure 16a. From this representation we can see, that the inversion solves for low seismic velocities (600 to 800 ms^{-1}) in the near-surface in the center of the investigated area, which corresponds to the still intact part of the soda lake, i.e., the part that is covered by water on a seasonal basis. Seismic velocity values above 800 ms^{-1} are resolved at depth as well as outside of the soda lake. To facilitate the interpretation of the resolved seismic velocity distribution in terms of the aquifer geometry we show the two vertical slices in Figure 16b and c, respectively. In particular, we relate seismic velocity values $> 800\,\mathrm{ms}^{-1}$ to the transition from the unsaturated to the saturated zone due to the higher seismic velocity of water ($\approx 1500\,\mathrm{ms}^{-1}$) compared to air ($\approx 340\,\mathrm{ms}^{-1}$) filling the pore space. Accordingly, our 3D subsurface model indicates the depth of the groundwater table at a depth of approximately 8 m. A more detailed interpretation is beyond the scope of this manuscript, yet the presented figures 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. By standardizing the data input in combination with a thorough sanity check aims at reducing the risk of corrupting the information stored in the project database. Moreover, crucial processing steps are automatized within the `SeismicWaveformModeler` and the `SeismicRefractionManager` classes facilitated by 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.

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 using and testing the formikoj library in frame of their research activities, and for providing valuable suggestions for improvement. 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 datasets are available for downloading at the link:

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

The orthophoto used in Figure 12 was 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.

Allen, R.V., 1978. Automatic earthquake recognition and timing from single traces. Bulletin of the seismological society of America 68, 1521–1532. doi:10.1785/bssa0680051521.

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.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik , 269–271doi:10.1007/bf01386390.

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.

Duan, X., Zhang, J., 2020. Multitrace first-break picking using an integrated seismic and machine learning methodpicking based on machine

learning. Geophysics 85, WA269–WA277. doi:10.1190/GEO2019-0422.1.

Earle, P.S., Shearer, P.M., 1994. Characterization of global seismograms using an automatic-picking algorithm. Bulletin of the Seismological Society of America 84, 366–376.

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

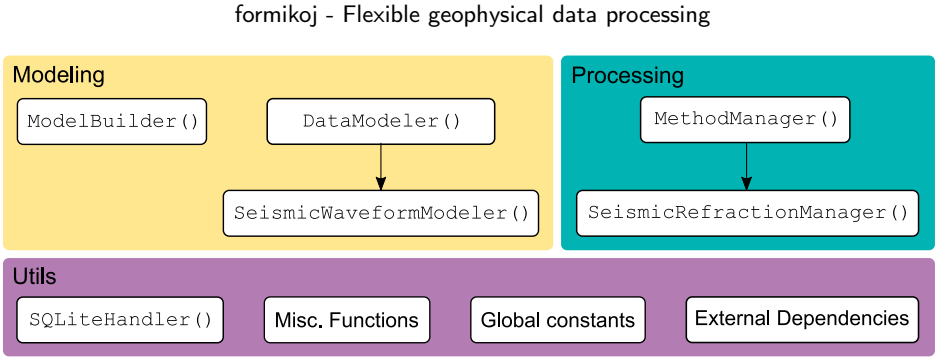


Figure 1: Fundamental use case illustrating the modeling of seismic refraction data within the formikoj ecosystem.

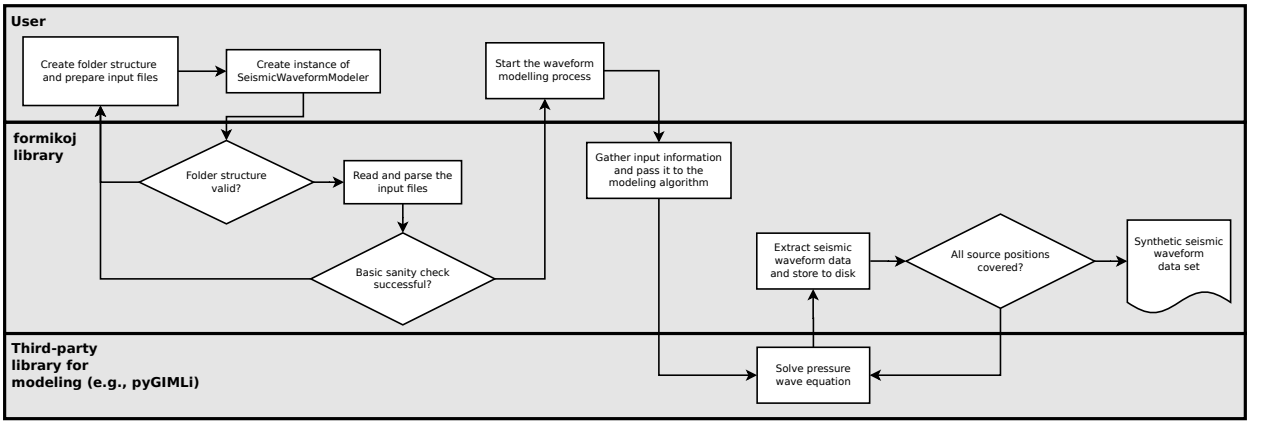


Figure 2: Fundamental use case illustrating the processing of seismic refraction data within the formikoj ecosystem.

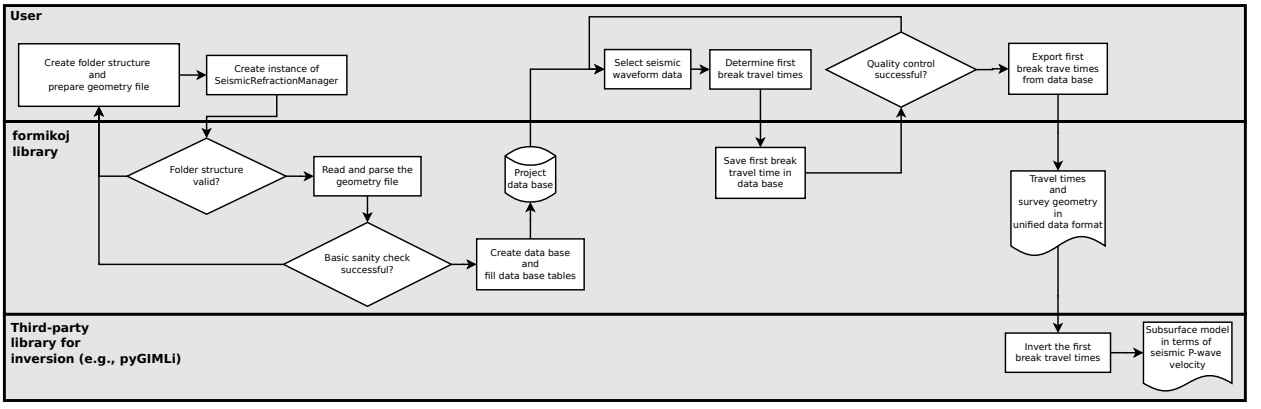


Figure 3: Typical formikoj workflow for the picking of first break traveltimes from seismic waveform data.

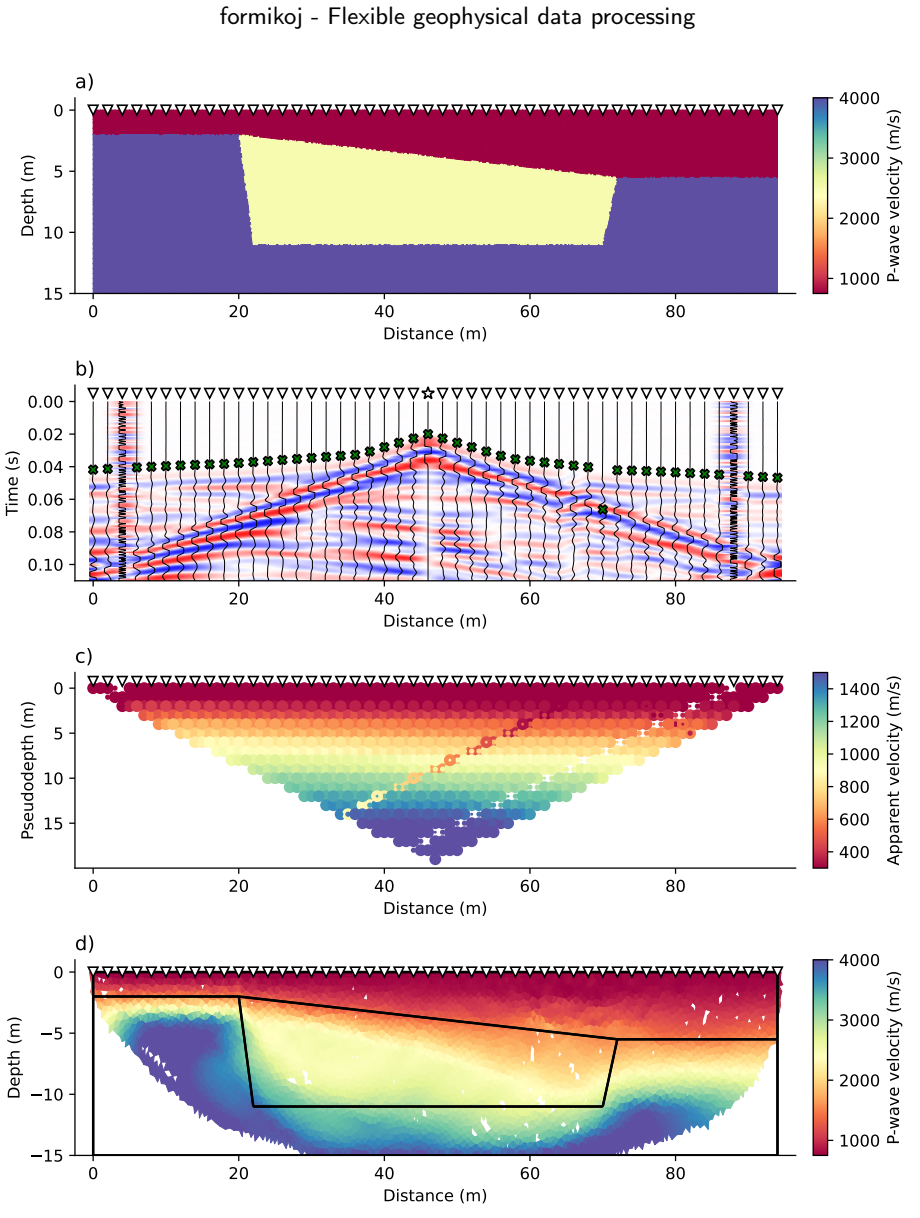


Figure 4: Synthetic seismic data generated with the `SeismicWaveformModeler`:
(a) Two-layered subsurface model without topography characterized by a distinct anomaly in the center. Triangles along the surface indicate the positions of geophones.
(b) Synthetic seismic waveform data computed from the subsurface 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 traveltimes based on the survey geometry.
(d) Subsurface model of the seismic P-wave velocity distribution obtained through the inversion of the synthetic P-wave traveltimes.

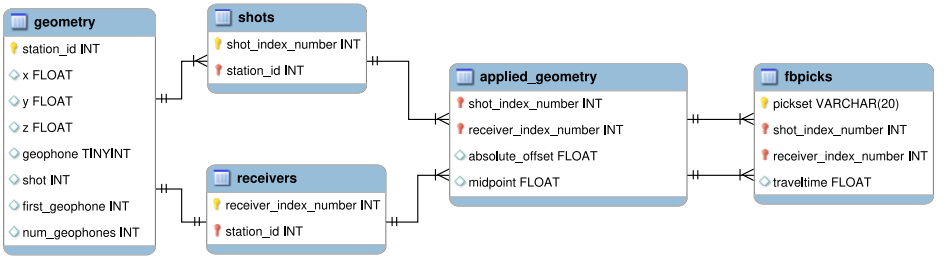


Figure 5: Entity-relationship diagram illustrating the SQLite database used in a `SeismicRefractionManager` project to store an manage processing related information.

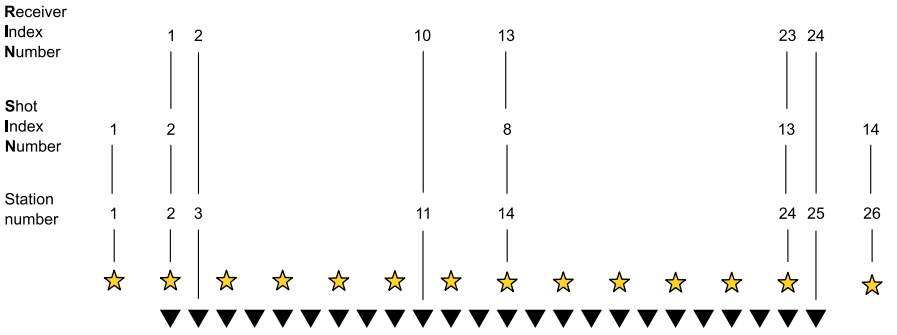


Figure 6: The `SeismicRefractionManager` addresses the stations through consecutive station numbers based on the sort order in the geometry file. The shot index numbers (SIN) and receiver index numbers (RIN) are assigned to the shot and receiver stations separately to allow for an intuitive data handling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

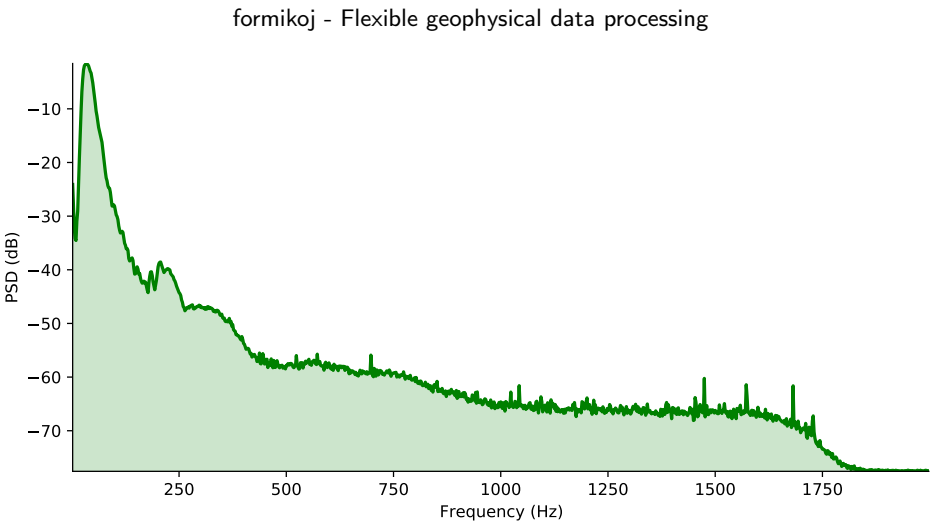


Figure 7: The frequency spectrum illustrates the frequency content of the currently selected traces, which allows for the identification of frequency ranges associated to noise allowing for the definition of corresponding frequency filter settings.

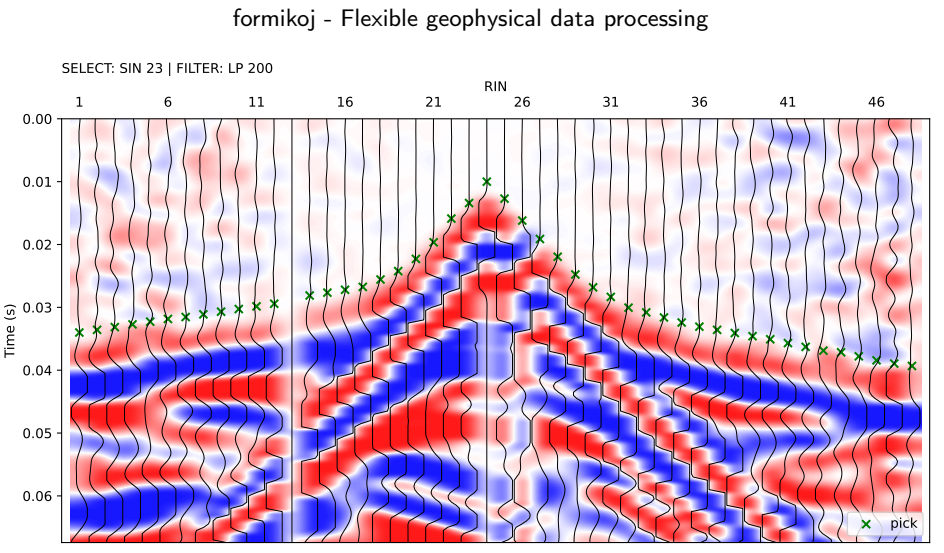


Figure 8: The seismogram plot presents the currently selected traces along the x axis, where the sort order is determined through the geometry. The corresponding trace data are illustrated as a function of time along the y axis (solid curves), with positive and negative amplitudes depicted in blue and 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.

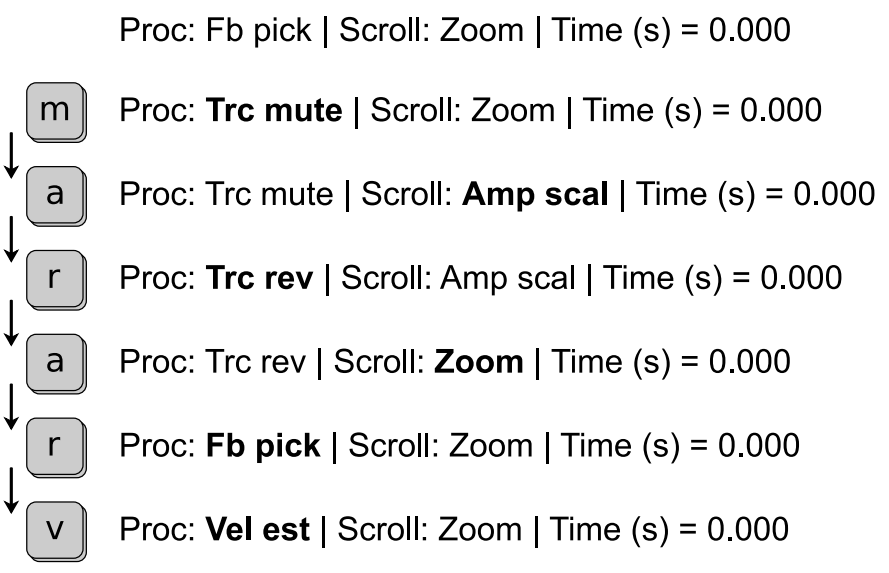


Figure 9: The status bar in the interactive seismogram plot displays the currently active processing and data scaling modes as well as the time (in seconds) at the current cursor position. By pressing the keys 'a', 'm', 'r', 'v' on the keyboard the different modes can be activated.

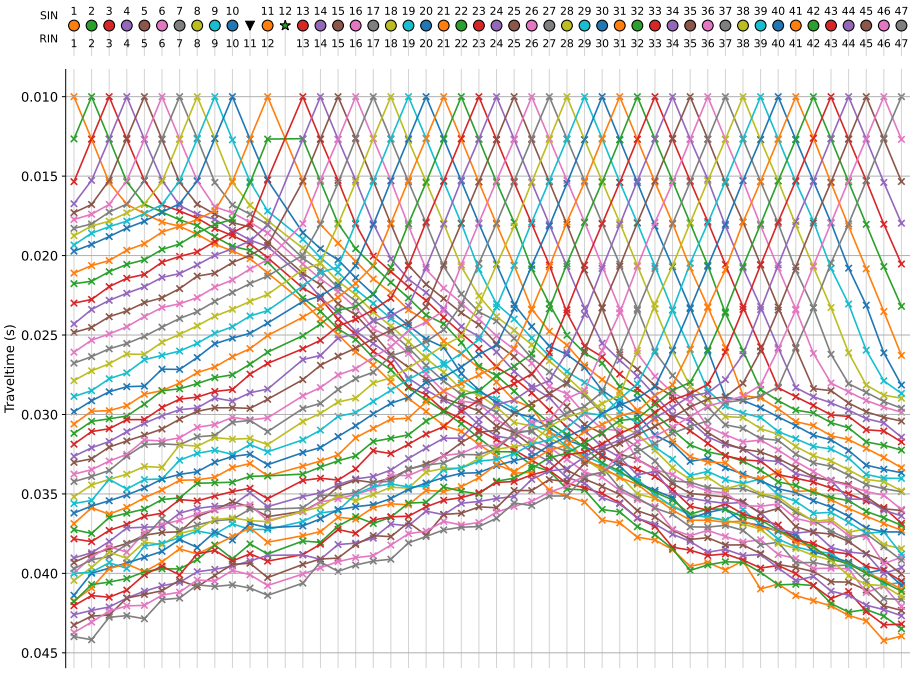


Figure 10: The traveltime diagram shows the first break traveltimes stored in the currently active pickset along the y axis (x symbols), where solid lines connect traveltimes assigned to a common shot. The sort order of the stations along the x axis reflects the geometry. Filled circles indicate stations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

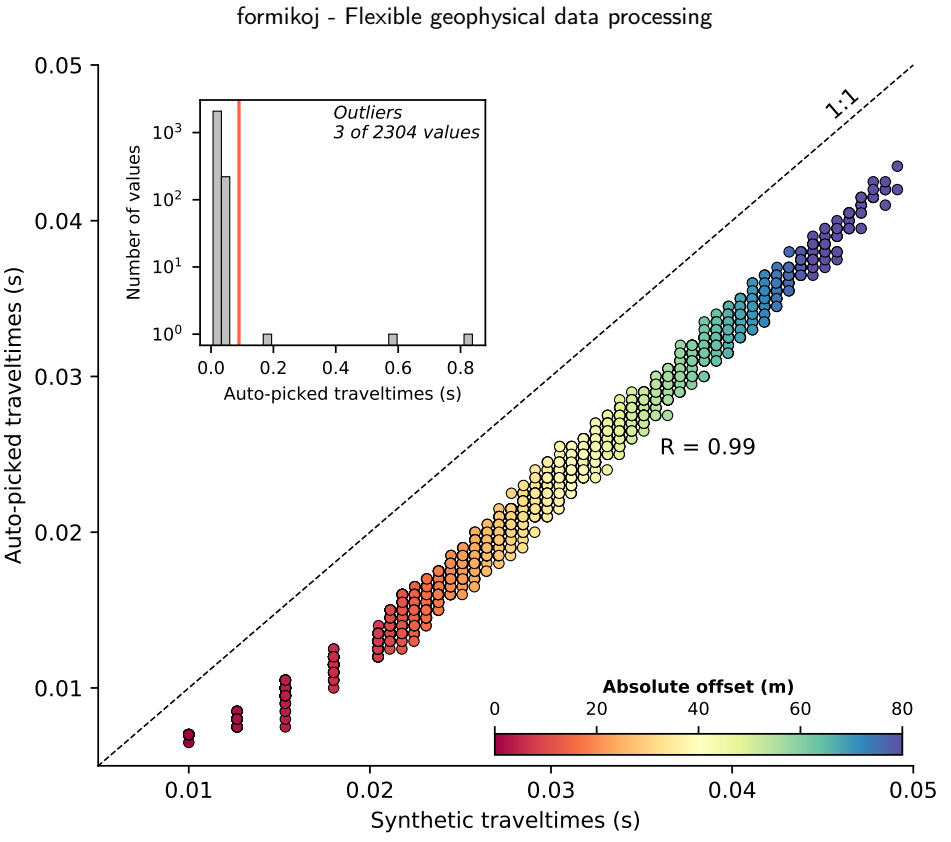


Figure 11: Comparison of forward modeled synthetic and automatically picked first break traveltimes for the synthetic seismic waveform data created in this study.

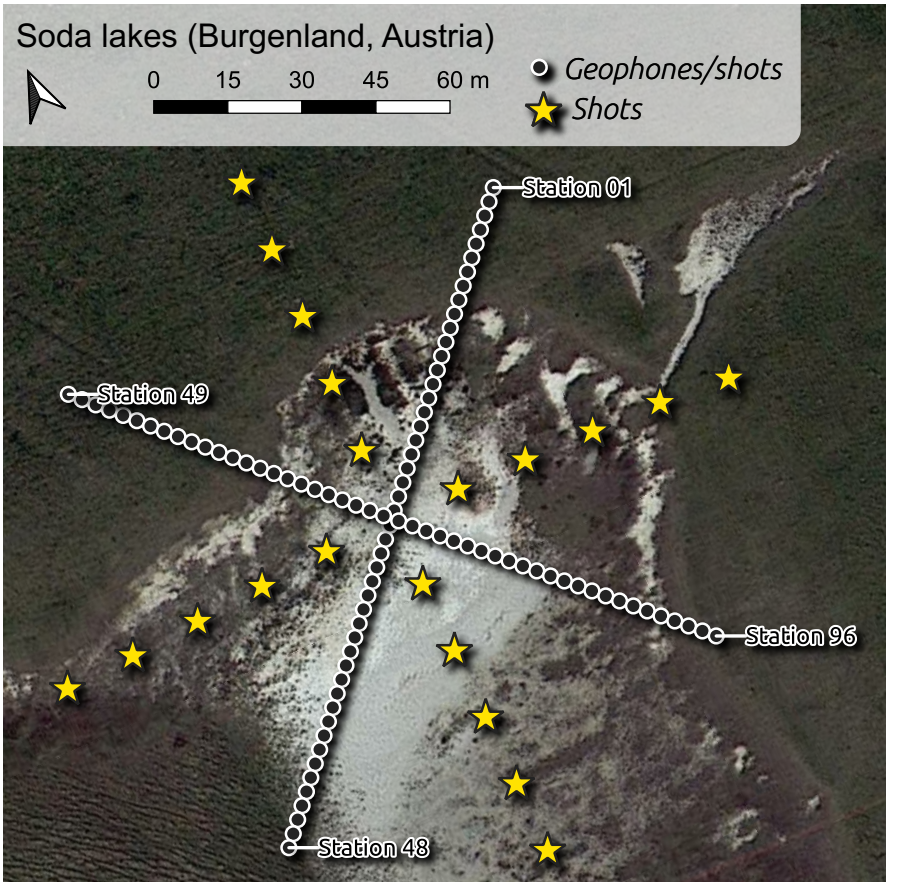


Figure 12: The soda lakes field dataset was collected in a 3D survey layout with stations (co-located geophones and shots indicated by filled dots) deployed in form of a cross. Additional shots (yellow stars) were conducted in from of a cross rotated by 45° to increase the coverage of the dataset.

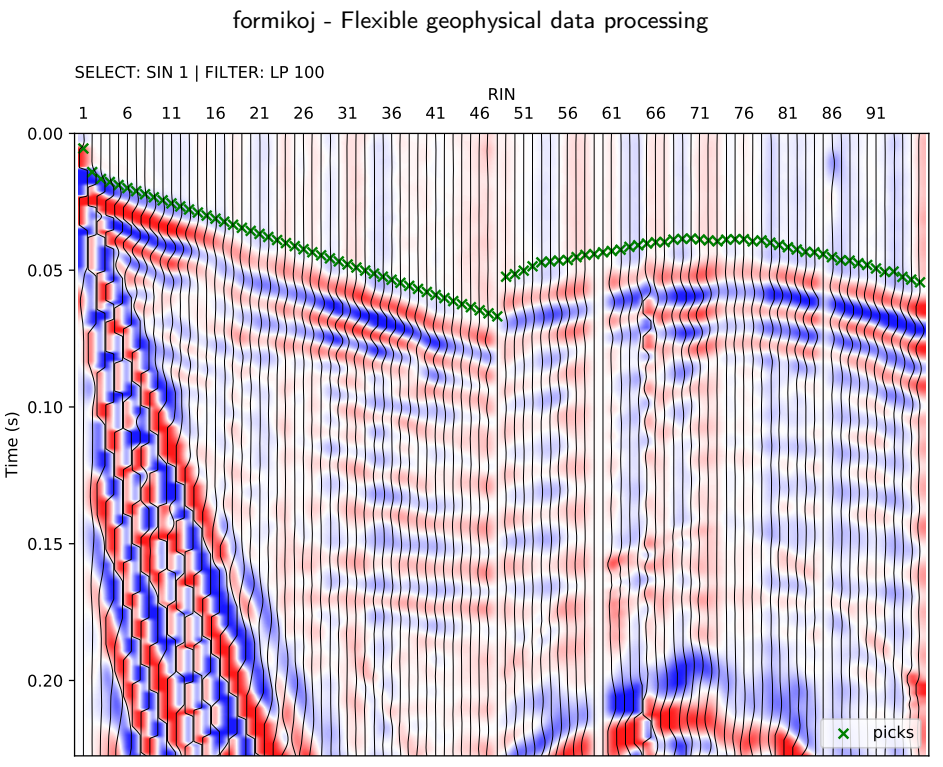


Figure 13: Examplary seismic waveforms from the soda lakes dataset shown for shot index number (SIN) 1 with a 100 Hz lowpass filter applied to suppress high frequency noise. The recorded seismic waveforms clearly reflect the geometry of the geophones with RIN 1 to 48 deployed along the direction of wave propagation, while RIN 49 to 96 are deployed perpendicular to the propagating wavefront.

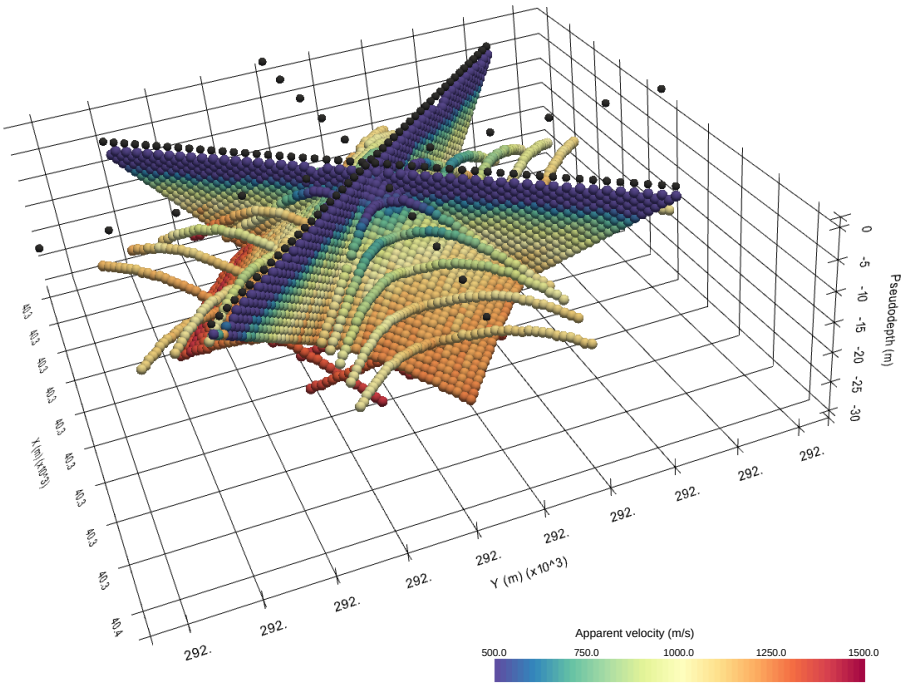


Figure 14: 3D pseudosection showing the apparent seismic velocities determined from the first break traveltimes obtained from the soda lakes dataset and the corresponding absolute offset between the shot and receiver stations. The apparent velocity for each shot-receiver pair is illustrated at the corresponding 2D midpoint and pseudodepth (1/3 of the absolute offset), thus yielding a 3D representation.

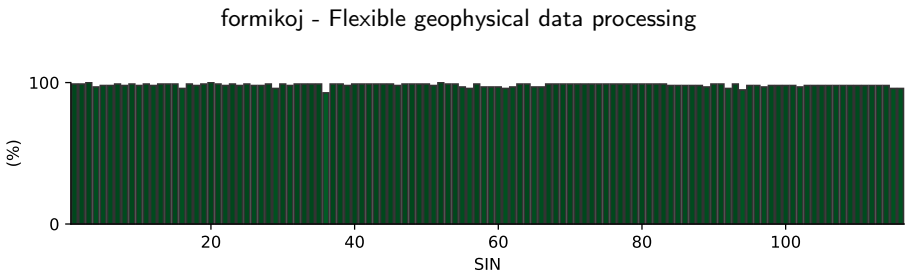


Figure 15: Picking percentage for each shot in the soda lakes dataset indicating a high signal-to-noise ratio allowing for the picking of first break traveltimes for nearly all shot-receiver pairs.

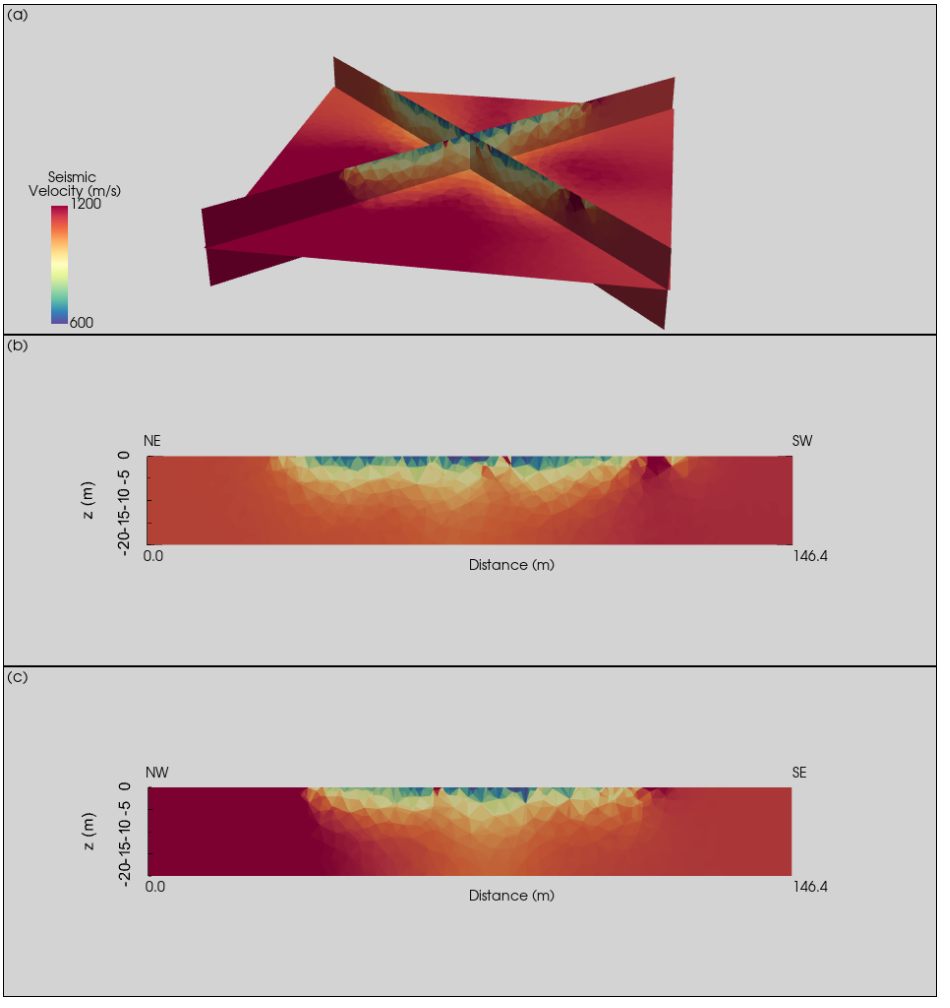


Figure 16: Subsurface model of the soda lake in terms of the seismic P-wave velocity. (a) 3D representation of orthogonal slices through the resolved subsurface model. (b) 2D representation of the NE–SW slice. (c) 2D representation of the NW–SE slice.