REPORT
doi:10.26443/seismica.v3i2.1184

DASCore: a Python Library for Distributed Fiber Optic Sensing

Derrick Chambers [*1], **Ge Jin** [2], **Ahmad Tourei** [2], **Abdul Hafiz Saeed Issah** [4], **Ariel Lellouch** [3], **Eileen R. Martin** [2,4], **Donglin Zhu** [2], **Aaron J. Girard** [2], **Shihao Yuan** [2], **Thomas Cullison** [6], **Tomas Snyder** [2], **Seunghoo Kim** [2,6], **Nicholas Danes** [7], **Nikhil Punithan** [2], **M. Shawn Boltz** [1], **Manuel M. Mendoza** [5]

[1]Spokane Mining Research Division, National Institute for Occupational Safety and Health, Spokane, USA, [2]Department of Geophysics, Colorado School of Mines, Golden, USA, [3]Geophysics Department, Tel Aviv University, Tel-Aviv Jaffa, Israel, [4]Department of Applied Math and Statistics, Colorado School of Mines, Golden, USA, [5]Cooperative Institute for Research in Environmental Sciences and Department of Geological Sciences, University of Colorado Boulder, Boulder, CO, USA, [6]Department of Geophysics, Stanford University, Stanford, USA, [7]Cyber Infrastructure and Advanced Research Computing, Colorado School of Mines, Golden, USA

Abstract In the past decade, distributed acoustic sensing (DAS) has enabled many new monitoring applications in diverse fields including hydrocarbon exploration and extraction; induced, local, regional, and global seismology; infrastructure and urban monitoring; and several others. However, to date, the open-source software ecosystem for handling DAS data is relatively immature. Here we introduce DASCore, a Python library for analyzing, visualizing, and managing DAS data. DASCore implements an object-oriented interface for performing common data processing and transformations, reading and writing various DAS file types, creating simple visualizations, and managing file system-based DAS archives. DASCore also integrates with other Python-based tools which enable the processing of massive data sets in cloud environments. DASCore is the foundational package for the broader DAS data analysis ecosystem (DASDAE), and as such its main goal is to facilitate the development of other DAS libraries and applications.

Production Editor: Gareth Funning
Handling Editor: Bradley Lipovsky
Copy & Layout Editor: Kirsty Bayliss

Received: January 17, 2024
Accepted: June 12, 2024
Published: July 30, 2024

1 Introduction

Over the past decade, off-the-shelf distributed acoustic sensing (DAS) units have become more reliable and economical, resulting in their broad adoption for a wide range of geophysical applications. DAS consists of an interrogator unit (i.e. a collection of opto-electronic components) connected to a fiber-optic cable. The interrogator unit uses rapid laser pulses to measure the average axial strain, strain rate, or effective velocity across different segments of the cable known as channels. DAS tends to be used in scenarios requiring thousands of channels, and for continuous or time-lapse monitoring, and often leads to many-terabyte datasets (Spica et al., 2023). Analysis of DAS data has been hindered by lack of free, readily available, easy-to-use software for reading, processing, visualizing, and analyzing large-scale DAS data recorded in a wide variety of formats (Lindsey and Martin, 2021).

Here we present DASCore: an open-source Python library for processing, visualizing, and managing DAS data. Although the Python programming language is certainly not the most performant, its open-source license, approachable design, ability to easily interface with more efficient languages, and vibrant community have made it one of the most popular language choices. Today, Python plays an important role in most computation-dependent fields including astronomy, biology, machine learning, etc. In seismology, stalwart Python libraries such as ObsPy (Krischer et al., 2015) and Pyrocko (Heimann et al., 2017) have enticed many researchers, sometimes with initial reservations, to embrace the scientific Python ecosystem. They have also enabled the creation of a variety of impactful applications and libraries for tasks such as template matching for earthquake detection (Chamberlain et al., 2018), complex ray-tracing (Bloch and Audet, 2023), seismic signal classification (Bueno et al., 2020), ambient noise correlations (Jiang and Denolle, 2020), seismic utility packages (Chambers et al., 2021) and many others. In this same spirit, DASCore aims to facilitate the rapid development of DAS-based research workflows and provide a robust foundation on which other projects can build.

*Corresponding author: derchambers@cdc.gov

2 Library Description

DASCore provides the following high-level features:

- An object-oriented interface for working with DAS data and metadata
- IO support for DAS files and file archives
- Processing and transformation routines
- Static visualizations

2.1 Data Structures

DASCore's application programming interface (API) was inspired by ObsPy and the multi-dimensional ar-

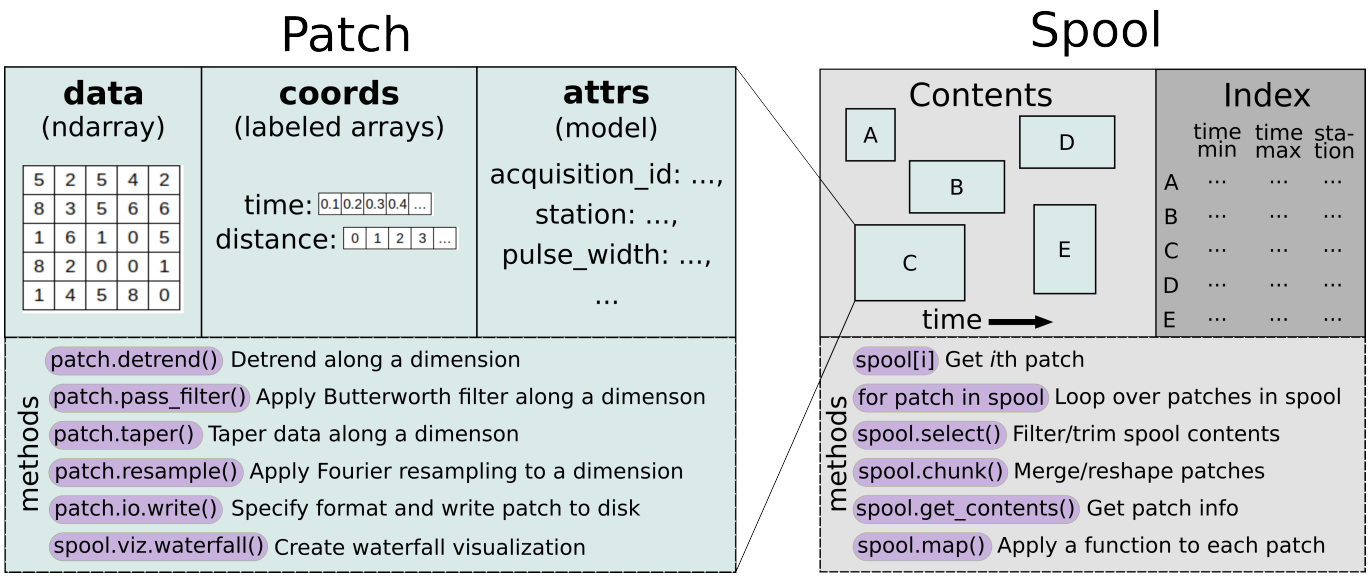

Figure 1 DASCore's `Patch` and `Spool` and some of their associated methods. See text for details.

ray library Xarray (Hoyer and Hamman, 2017). DASCore utilizes `Patch` and `Spool` data structures to facilitate data processing and management (Figure 1). For readers familiar with ObsPy, the `Patch` is analogous to ObsPy's `Trace` and is modeled after Xarray's `DataArray`. The `Spool` combines the functionality of ObsPy's `Stream` and `Client` and is focused on data querying and retrieval.

The `Patch` class has three main attributes: `coords` which specify the labels and values for each dimension, `attrs` which stores non-coordinate related metadata, and `data` which contains an n-dimensional array. The `Patch` is immutable, to the extent Python allows, so all of its methods return new objects. We chose to make the `Patch` and its sub-components immutable in order to minimize errors related to data mutation, allow safe sharing of `Patch` components, and make distributed computing simpler.

Rather than a concrete implementation, the `Spool` is an *interface* which unifies data access across a variety of sources. It enables DASCore code to treat an archive of DAS files in a file system, a single data file, or DAS data loaded into memory, in an identical manner. Additionally, using DASCore's plug-in system, external codes can add support for other file formats and data sources. For example, a separate package currently under development will provide `Spool` implementations to interact with a number of databases and remote resources. In addition to a unified interface, most `Spool` implementations lazily plan the selection, chunking, and merging of data sources through the `chunk` and `select` methods. A `Spool` implementation included in DASCore enables the selection and querying of large file archives using an HDF5-based index built on the table and in-kernel query features of PyTables (Alted and Fernández-Alonso, 2003). The index, which itself is an HDF5 file, currently tracks basic metadata such as instrument name, experiment identifier, sampling rate, start and end times, etc. It can be included with a dataset and shared among multiple users. Although initially creating the index does take some time, once it exists DASCore can query and retrieve data very efficiently. For example, we currently use DASCore to manage several datasets, one of the largest contains 230,000 DAS files comprising 50 TB of data. It took approximately 4 hours to index and we haven't observed any slowdowns in querying or data retrieval as compared to smaller (GB-scale) datasets. Currently, DASCore requires one of the `Patch` dimensions to be named "time" but this restriction may be lifted in future versions.

2.2 Processing and Transformations

Processing routines and transformations are implemented as `Patch` methods. These always return new `Patch` instances, leaving the original unaltered but sharing immutable components where possible. Most methods and transformations can be applied along any, or multiple, dimensions which are specified using Python's keyword arguments. Although certainly not exhaustive, a few of the commonly used `Patch` processing and transformation methods are shown in Figure 1.

2.3 Supported File Formats

DASCore provides four types of file support: `read` - the ability to load the file contents into one or more `Patch` objects; `write` - the ability to dump one or more `Patch` objects to a file or byte stream; `scan` - a fast extraction of the file's metadata; and `get_format` - the ability to automatically determine the format (and version) of a file. Table 1 shows DASCore's file support for the current version, 0.1.0, although we anticipate adding support for more formats in the future.

2.4 Visualization

Although visualization is not DASCore's primary focus, it currently provides two types of matplotlib-based plots; the waterfall plot and the wiggle plot. The waterfall plot is a simple 2D color image and scale bar that is commonly used to visualize dense seismic array data, including DAS data. The wiggle plot is more commonly

format & version	read	write	scan	get_format
DASDAE v1	✓	✓	✓	✓
DASHDF5 v1	✓		✓	✓
H5SIMPLE v1	✓		✓	✓
PICKLE	✓	✓		✓
PRODML v2.0	✓		✓	✓
PRODML v2.1	✓		✓	✓
RSF		✓		
SEGY v2.0	✓		✓	✓
TDMS v4713	✓		✓	✓
TERRA15 v4	✓		✓	✓
TERRA15 v5	✓		✓	✓
TERRA15 v6	✓		✓	✓
WAV		✓		

Table 1 Supported file formats for DASCore.

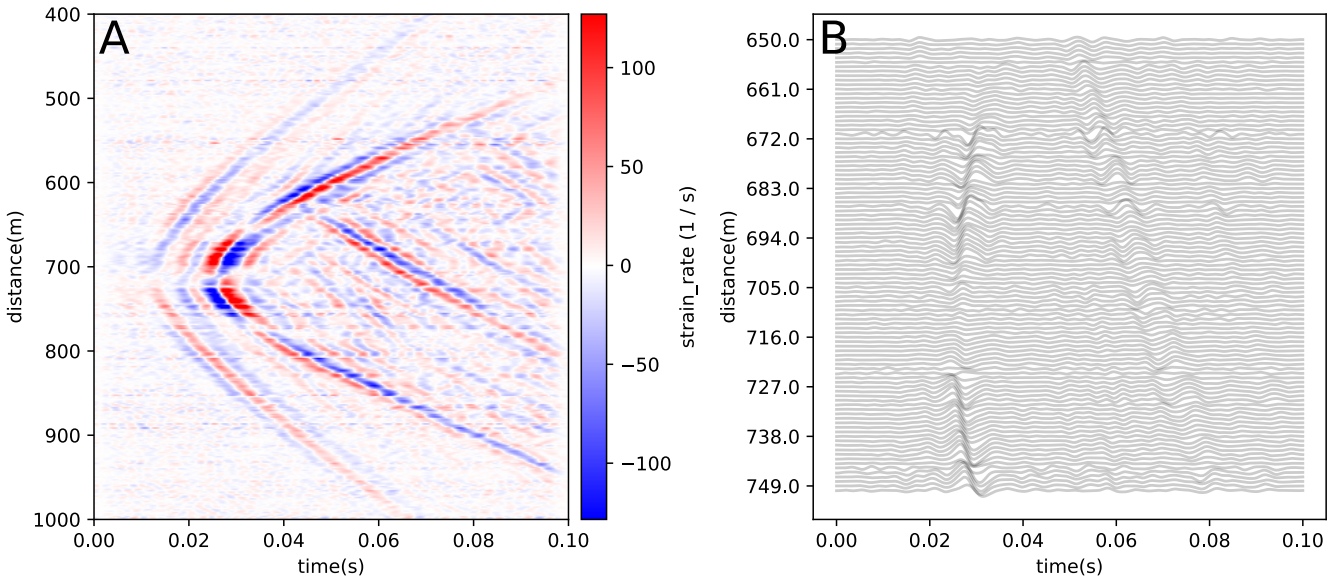

Figure 2 Waterfall (left) and wiggle (right) plots produced by Listing 2.

used in small or sparse seismic networks and displays a selection of channel traces offset along the y-axis and scaled according to some trace-specific metric, such as absolute value. For example, Figure 2, shows both a waterfall plot (A) and wiggle plot (B) of a microseismic event detailed in Staněk et al. (2022). The code to generate this figure is found in Listing 2.

2.5 Software Engineering

DASCore strives to implement best software engineering practices. It has an extensive test suite which includes more than 1800 automated tests which exercise over 99.5% of the codebase. DASCore also makes use of GitHub's features to enforce coding style rules, run the test suite before code can merge into the main branch, organize development planning, code review and issue tracking, and for packaging and automatic releases to the Python package index and conda-forge. DASCore has comprehensive documentation, which includes tutorials, recipes, API references, and notes, hosted at https://dascore.org. The documentation is built in a somewhat unique way among Python projects because it uses Quarto (Bauer and Landesvatter, 2023) rather than other popular documentation frameworks such as Sphinx or MkDocs. Although largely a personal preference, we find Quarto to be more user-friendly, feature complete, and easier to customize.

Listing 1 Example of `Patch` processing

```
import dascore as dc

patch = (
    # Load data from DASCore's data registry.
    dc.get_example_patch('example_event_1')
    # Taper the ends of the time dimension.
    .taper(time=0.05)
    # Apply a low-pass filter at 300 Hz.
    .pass_filter(time=(..., 300))
)

# Plot the result.
patch.viz.waterfall(show=True, scale=0.2)
```

Listing 2 Example of `Patch` visualization

```
import dascore as dc
import matplotlib.pyplot as plt

# Setup matplotlib figure/axis.
_, (ax1, ax2) = plt.subplots(
    1, 2, figsize=(10, 4.5)
)

# Load example patch.
patch = dc.get_example_patch(
    "example_event_2"
)

# Sub-select only center channels.
sub_patch = patch.select(
    distance=(650, 750)
)

# Plot waterfall.
patch.viz.waterfall(ax=ax1, scale=0.5)

# Plot wiggles.
sub_patch.viz.wiggle(ax=ax2, scale=0.5)

# Add subplot labels.
ax1.text(.01, .99, 'A', ha='left', va='top',
         transform=ax1.transAxes, size=24)
ax2.text(.01, .99, 'B', ha='left', va='top',
         transform=ax2.transAxes, size=24)

plt.tight_layout()
plt.show()
```

3 Additional Examples

3.1 Processing and visualization

DASCore processing (and transformation) methods are typically combined in a chain of method calls. Structuring code in this way provides a unique semantic meaning to each line, leaves room for comments, and avoids creating intermediary variables. Listing 1 provides an example of using DASCore to load, taper, filter, and visualize an example patch.

Visualization functions are found in the patch `viz` namespace, and since they operate on matplotlib axes, they are highly customizable. For example, Listing 2 shows the code used to produce Figure 2.

3.2 Units

Most of DASCore's functionality provides support for units using the Pint library. Units improve the clarity of the code intent and can reduce errors. Listing 3 provides an example of importing a few common units such as meter (m), foot (ft), and millisecond (ms), filtering the selected portion of data, and converting Patch's distance units to feet.

3.3 Archive management

Listing 4 code shows how to index and query a collection of DAS files in a directory. The `Spool.update` method ensures any files with timestamps after the last update are appended to the index. With these concepts, DASCore can also process data associated with active acquisitions in near real-time.

Listing 3 Example of using units

```
import dascore as dc
from dascore.units import m, ft, ms

patch = (
    dc.get_example_patch()
    # Trim ten millisecond from patch start
       and end.
    .select(time=(10*ms, -10*ms), relative=
       True)
    # Apply spatial filter specifying
       wavelength.
    .pass_filter(distance=(3*m, 6*m))
    # Convert distance coord to ft.
    .convert_units(distance=ft)
)
```

Listing 4 Example for interacting with directory of DAS files

```
import dascore as dc
from dascore.units import second, minute

spool = dc.spool("data_path").update()

sub_spool = spool.select(
    # Select time range.
    time=("2021-01-01", "2021-01-01T12:00"),
    # Query supports posix style matching.
    tag="experiment_??",
)

# Chunk data into patches with overlap.
sub_chunked_spool = sub_spool.chunk(
    time=10*minute,
    overlap=30*second,
)

# Iterate the spool to get patches.
for patch in sub_chunked_spool:
    ...
```

3.4 Parallel processing

Since DASCore `Spool` instances act as a *sequence* of patches, many of the popular multi-process/distributed frameworks can use them to easily implement embarrassingly parallel workflows. For example, Listing 5 provides an example of using the `ProcessPoolExecutor` from Python's standard library to spread out a processing workflow to multiple processes. This can also be done with libraries like Dask (Rocklin, 2015) which provide an `Executor`-like interface (an object which has a `map` method).

3.5 Escape Hatches

Although DASCore's abstractions are useful for basic processing, users may want to work with DAS data in other ways. To that end, DASCore provides several "escape hatches" for accessing raw data directly or converting data contained in a `Patch` to another data struc-

Listing 5 Example of parallel processing

```
from concurrent.futures import
    ProcessPoolExecutor

import dascore as dc

def patch_function(patch):
    """A custom function for processing
        patches."""
    ...

spool = dc.get_example_spool("random_das")

executor = ProcessPoolExecutor()

results = spool.map(patch_function, client=
    executor)
...
```

Listing 6 Example of converting a `Patch` to data formats used by external libraries

```
import dascore as dc

patch = dc.get_example_patch()

# Access raw data and coordinates.
data = patch.data
time_array = patch.coords.get_array('time')
dist_array = patch.coords.get_array('distance
    ')

# Convert a patch to an obspy stream.
stream = patch.io.to_obspy()

# Convert a 2D patch to a pandas DataFrame.
df = patch.io.to_dataframe()

# Convert a patch to an Xarray DataArray.
dar = patch.io.to_xarray()
```

ture. Listing 6 shows several such examples. First, this example extracts the data of a patch, which is a 2D NumPy array. The two coordinate arrays can also be extracted, resulting in two 1D NumPy arrays. Additionally, the `to_obspy()`, `to_pandas()`, and `to_xarray()` methods can be applied to a patch to convert these to an ObsPy `Stream`, a Pandas `DataFrame`, or an Xarray `DataArray`, respectively. The latter is particularly useful as the Xarray/PanGeo ecosystem (Hamman et al., 2018) is well equipped for large-scale cloud computing.

4 Future Plans and DASDAE

DASCore is the foundational library of a broader initiative known as the distributed acoustic sensing data analysis ecosystem (DASDAE), and is distributed through the DASDAE GitHub organization. Similar to the Astropy project (Price-Whelan et al., 2022), DASDAE aims to foster a community of compatible open-source packages for a variety of fiber-optic sensing projects. We aim for DASCore to have a relatively light set of dependencies, to contain tools useful for many DAS projects across application domains, and to maintain high standards of testing and documentation. Other DASDAE packages are expected to be compatible with DASCore, but may focus on processing workflows with more dependencies, specific applications such as near-surface imaging, low-frequency DAS processing, machine learning, etc. More information can be found at https://dasdae.org.

In addition to expanding the processing and IO support, we plan to implement a class for managing experiment metadata, tentatively called an `Inventory`. The `Inventory` will be a Pydantic model hierarchy which extends the metadata included in the `Patch`. It will be capable of capturing general information about DAS acquisitions including fiber type, splice locations, interrogator configuration, tap-tests, etc. The `Inventory` will be based on the community-driven DAS meta-data standards developed by the DAS research coordination network (RCN) and the ProdML standard used in the oil and gas industry (Shipley et al., 2008).

Acknowledgements

Although the information in this paper is correct at the time of submission, the most up-to-date details can be found at DASCore's website (https://dascore.org). DASCore development efforts are supported in part by the NSF Geoinformatics Program, under grant #2148614, NSF Earth Science Postdoctoral Fellowship award #2053085, and sponsors of the Center for Wave Phenomena. In addition to the scientific libraries already mentioned, DASCore makes use of the following python libraries: Pooch (Uieda et al., 2020), h5py (Collette et al., 2021), SciPy (Virtanen et al., 2020), Matplotlib (Hunter, 2007), Pydantic (https://docs.pydantic.dev/latest/), Rich (https://github.com/Textualize/rich), PyTables (Alted and Fernández-Alonso, 2003), Pint (https://pint.readthedocs.io/en/stable/), SegyIO (https://segyio.readthedocs.io/en/latest/), and NumPy (Harris et al., 2020). We would also like to thank two anonymous reviewers whose comments helped improve this work.

The findings and conclusions in this paper are those of the authors and do not necessarily represent the official position of the National Institute for Occupational Safety and Health, Centers for Disease Control and Prevention or any other affiliated organization of the authors.

Data and code availability

DASCore's documentation is found at https://dascore.org. The code repository, issue tracker, and general development occur on https://github.com/dasdae/dascore. As of the submission of this manuscript, the latest version of DASCore is 0.1.0 which was archived on 2024-01-11 with the following DOI: 10.5281/zenodo.10494398.

Competing interests

The authors declare no competing interests and have fully disclosed all financial support for this project.

References

Alted, F. and Fernández-Alonso, M. PyTables: processing and analyzing extremely large amounts of data in Python. *PyCon2003. April*, pages 1–9, 2003.

Bauer, P. C. and Landesvatter, C. Writing a reproducible paper with RStudio and Quarto. 2023. doi: https://doi.org/10.31219/osf.io/ur4xn.

Bloch, W. and Audet, P. PyRaysum: Software for Modeling Ray-theoretical Plane Body-wave Propagation in Dipping Anisotropic Media. *Seismica*, 2(1), 2023. doi: https://doi.org/10.26443/seismica.v2i1.220.

Bueno, A., Zuccarello, L., Díaz-Moreno, A., Woollam, J., Titos, M., Benítez, C., Álvarez, I., Prudencio, J., and De Angelis, S. PICOSS: Python interface for the classification of seismic signals. *Computers & geosciences*, 142:104531, 2020. doi: https://doi.org/10.1016/j.cageo.2020.104531.

Chamberlain, C. J., Hopp, C. J., Boese, C. M., Warren-Smith, E., Chambers, D., Chu, S. X., Michailos, K., and Townend, J. EQcorrscan: Repeating and near-repeating earthquake detection and analysis in Python. *Seismological Research Letters*, 89(1): 173–181, 2018. doi: https://doi.org/10.1785/0220170151.

Chambers, D. J., Boltz, M. S., and Chamberlain, C. J. ObsPlus: A Pandas-centric ObsPy expansion pack. *Journal of open source software*, 6(60):2696, 2021. doi: https://doi.org/10.21105/joss.02696.

Collette, A., Kluyver, T., Caswell, T. A., Tocknell, J., Kieffer, J., Scopatz, A., Dale, D., Jelenak, A., VINCENT, T., Sciarelli, P., et al. h5py/h5py: 3.1. 0. *Zenodo*, 2021. doi: https://doi.org/10.5281/zenodo.7568214.

Hamman, J., Rocklin, M., and Abernathy, R. Pangeo: a big-data ecosystem for scalable earth system science. In *EGU General Assembly Conference Abstracts*, page 12146, 2018.

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., et al. Array programming with NumPy. *Nature*, 585(7825): 357–362, 2020. doi: https://doi.org/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., et al. Pyrocko-An open-source seismology toolbox and library. 2017. doi: https://doi.org/10.5880/GFZ.2.1.2017.001.

Hoyer, S. and Hamman, J. Xarray: ND labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. doi: https://doi.org/10.5334/jors.148.

Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007. doi: https://doi.org/10.1109/MCSE.2007.55.

Jiang, C. and Denolle, M. A. NoisePy: A new high-performance python tool for ambient-noise seismology. *Seismological Research Letters*, 91(3):1853–1866, 2020. doi: https://doi.org/10.1785/0220190364.

Krischer, L., Megies, T., Barsch, R., Beyreuther, M., Lecocq, T., Caudron, C., and Wassermann, J. ObsPy: A bridge for seismology into the scientific Python ecosystem. *Computational Science & Discovery*, 8(1):014003, 2015. doi: https://doi.org/10.1088/1749-4699/8/1/014003.

Lindsey, N. J. and Martin, E. R. Fiber-optic seismology. *Annual Review of Earth and Planetary Sciences*, 49:309–336, 2021. doi: https://doi.org/10.1146/annurev-earth-072420-065213.

Price-Whelan, A. M., Lim, P. L., Earl, N., Starkman, N., Bradley, L., Shupe, D. L., Patil, A. A., Corrales, L., Brasseur, C., Nöthe, M., et al. The Astropy Project: sustaining and growing a community-oriented open-source project and the latest major release (v5. 0) of the core package. *The Astrophysical Journal*, 935(2):167, 2022. doi: https://doi.org/10.3847/1538-4357/ac7c74.

Rocklin, M. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, volume 130, page 136. SciPy Austin, TX, 2015. doi: https://doi.org/10.25080/Majora-7b98e3ed-013.

Shipley, D., Weltevrede, B., Doniger, A., Klumpen, H. E., and Ormerod, L. Production data standards: The PRODML business case and evolution. In *SPE Intelligent Energy International Conference and Exhibition*, pages SPE–112259. SPE, 2008. doi: https://doi.org/10.2118/112259-MS.

Spica, Z. J., Ajo-Franklin, J., Beroza, G. C., Biondi, B., Cheng, F., Gaite, B., Luo, B., Martin, E., Shen, J., Thurber, C., Viens, L., Wang, H., Wuestefeld, A., Xiao, H., and Zhu, T. PubDAS: A PUBlic Distributed Acoustic Sensing Datasets Repository for Geosciences. *Seismological Research Letters*, 94:983–998, 2023. doi: https://doi.org/10.1785/0220220279.

Staněk, F., Jin, G., and Simmons, J. Fracture imaging using DAS-recorded microseismic events. *Frontiers in Earth Science*, 10: 907749, 2022. doi: https://doi.org/10.3389/feart.2022.907749.

Uieda, L., Soler, S. R., Rampin, R., Van Kemenade, H., Turk, M., Shapero, D., Banihirwe, A., and Leeman, J. Pooch: A friend to fetch your data files. *Journal of Open Source Software*, 5(45): 1943, 2020. doi: https://doi.org/10.21105/joss.01943.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, 17(3):261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.