Journal of
open research software

nd – A Framework for the Analysis of n-dimensional Earth Observation Data

SOFTWARE METAPAPER

JOHANNES N. HANSEN

]u[ubiquity press

ABSTRACT

`nd` is a software framework for the analysis of n-dimensional datacubes. Such data structures are common in Earth observation where the data are often multivariate on a spatio-temporal grid. The Python library `xarray` already exists to represent such data sets. This software builds on `xarray` by providing an interface with other parts of the Python ecosystem, such as `scikit-learn` and `rasterio`. Most of the functionality is accessible through an added accessor `xarray.Dataset.nd`. The software is maintained on GitHub and releases are published on the Python Package Index (PyPI) and Zenodo.

CORRESPONDING AUTHOR:

Johannes N. Hansen

School of Mathematics, University of Edinburgh, GB

johannes.hansen@ed.ac.uk

KEYWORDS:

python; xarray; earth observation; remote sensing; satellite data; netcdf; radar; synthetic aperture radar; datacube

TO CITE THIS ARTICLE:

Hansen JN 2022 `nd` – A Framework for the Analysis of n-dimensional Earth Observation Data. *Journal of Open Research Software,* 10: 3. DOI: *https://doi.org/10.5334/jors.377*

(1) OVERVIEW

INTRODUCTION

There is a current disconnect between the remote sensing and scientific computing communities. This is partly due to the discontinuity of the tools preferred in either community: the scientific computing community requires an API that integrates well with existing ecosystems for scientific data processing. In remote sensing, on the other hand, a large part of the processing is still done with graphical tools, e.g. SNAP or GIS software such as QGIS, ArcGIS, and ENVI. Most of these also provide a command line interface (CLI), as does the well-established tool GDAL. However, when it comes to providing an API for programming languages, these tools lack in features or usability. GDAL, for example, does offer a Python API, but it is clear that this API has been developed as a wrapper for the CLI rather than envisioned around concepts more native to the programming language. This shows e.g. in the fact that most operations that can be performed in the Python GDAL API still read one file and output another rather than passing around Python objects. The popular Python library rasterio [5] is a more Python-friendly wrapper around GDAL, but is still cumbersome when working with more than a single image. There is essentially no library out there in any programming language that natively implements e.g. pre-processing of SAR data without having to go through the CLI of some GUI based software. Also, operations such as reprojection, mosaicking and tiling are not easily available outside of dedicated command line tools. All of this means it is not easy for a scientist or developer to implement a new algorithm without worrying about IO, file formats, and preprocessing because there is no library that implements all of those steps in a sufficiently modular way to allow for the addition of new algorithms as modules (other than as plugins for GUI software). Python developers need libraries that neatly fit into the SciPy ecosystem [15] which provides the basis for all scientific computing in Python. Clearly, there is a need for a grand unified EO library.

Another issue with Earth observation data processing is the amount of data required. For applications dealing with global datasets it is often unfeasible to download all data locally for processing. Even if storage, memory, and compute resources are not an issue, bandwidth limitations mean that obtaining such datasets could be a matter of months or years. Projects like Google Earth Engine [6] or AWS Earth [1] aim to overcome this limitation by providing datasets that exist physically right next to their cloud computing services such that no data will need to be transferred via low bandwidth connections until the export of the final product. In most cases, the generated data product is much smaller than the sum of all the raw input data. It took Google Earth Engine more than three years to acquire the multi-petabyte dataset which is now available, partly because the Landsat archive had to be transferred from the long-term tape storage [6]. While the focus here is not on cloud services like this, a well-designed library with scalable tools would also be suitable to run on such a service.

Perhaps the largest barrier is the lack of a common standard file format. While GeoTiff is the de-facto standard for most applications exchanging geospatial raster data, it has a number of crucial drawbacks. Firstly, it isn't capable of storing data in more than two dimensions and as such is unsuitable for datacube structures. Secondly, while GeoTiff files do contain geocoding information (that is what makes it a GeoTiff rather than a Tiff), they are far from self-descriptive: they don't support storing band names, coordinate system information, etc. When software such as QGIS opens a GeoTiff, they will immediately create an additional XML file storing metadata such as projection information that is gathered from user input. Clearly, GeoTiff is not fully up to the task of geospatial data processing if it requires a separate metadata file to work properly. Rather than inventing yet another file format or even a new Python data structure, this project builds on the xarray library [8] which interfaces neatly with the NetCDF file format and extends pandas to n-dimensional data. xarray has been developed by the climate data community and is now officially recommended by pandas in favor of the deprecated pandas Panel [13].

NetCDF for Earth Observation

NetCDF (specifically NetCDF-4) is a highly efficient file format that was built on top of HDF5. It is capable of random access which ties in with indexing and slicing in numpy. Because slices of a large dataset can be accessed independently, it becomes feasible to handle larger-than-memory file sizes. NetCDF-4 also supports data compression using zlib. Random access capability for compressed data is maintained through data chunking. Furthermore, NetCDF is designed to be fully self-descriptive. Crucially, it has a concept of named dimensions and coordinates, can store units and arbitrary metadata. While NetCDF has been the file format of choice in climate modeling for a long time, the Earth observation community is only recently adopting it. To be convinced of the slow but steady adoption of NetCDF it is revealing to look at the evolution of file formats used by ESA for the Copernicus missions. Sentinel-1 and Sentinel-2 still use the proprietary file format SAFE which is essentially a collection of GeoTiff files put together in a folder with metadata. Sentinel-3 uses the SEN3 format, which works similarly to SAFE, except now it is a collection of NetCDF files put together in a folder with metadata. Finally, with Sentinel-5p, ESA has made the move to a single NetCDF file for each product. In order to facilitate the work with existing file formats, the framework aims to provide an interface to read and write file formats to/from NetCDF via its Python cousin xarray.

The compatibility layer between other file formats and NetCDF is mostly fed by GDAL, which offers good support for a long list of geospatial file formats [4].

Purpose

The main goal of this library is to generalize methods that work in lower dimensions to higher-dimensional data.

Multi-dimensional data often arise as spatio-temporal datacubes, e.g. climate data or time series of geospatial satellite data. Many data analysis methods are designed to work on single images or time series at a single point. `nd` makes it easy to broadcast these methods across a whole dataset, adding additional features such as automatic parallelization.

Examples include

- pixelwise change detection algorithms
- reprojection between coordinate systems
- machine learning algorithms

The software was produced during PhD research. It has been used to generate the results in [7].

IMPLEMENTATION AND ARCHITECTURE

`nd` is built on `xarray`. Internally, all data are passed around as `xarray` Datasets and all provided methods expect this format as inputs. An `xarray.Dataset` is essentially a Python representation of the NetCDF file format and as such easily reads/writes NetCDF files.

`nd` is making heavy use of the `xarray` and rasterio libraries. The GDAL library is only used via `rasterio` as a compatibility layer to enable reading supported file formats. `nd.open_dataset` may be used to read any NetCDF file or any GDAL-readable file into an `xarray.Dataset`.

The popular machine learning library `scikit-learn` is designed to work with tabular data. The `nd.classify` module implements an interface to automatically apply `scikit-learn` estimators to `xarray` objects of arbitrary dimensions. Dataset dimensions may be treated as data dimensions (independent data points) or feature dimensions (used for training and prediction). This makes it easy to train a classifier on an entire time series, for example.

Similarly, reprojecting data using GDAL (via `rasterio`) is restricted to individual images. `nd` makes the integration with `xarray` easy by automatically applying the reprojection operation to the spatial data dimensions and broadcasting across all extra dimensions.

Figure 1 shows an architecture for a framework that might achieve this. In essence, there is a data compatibility layer which handles the ingestion of geospatial data formats into xarray, via rasterio and GDAL. As mentioned previously, the library does not aim

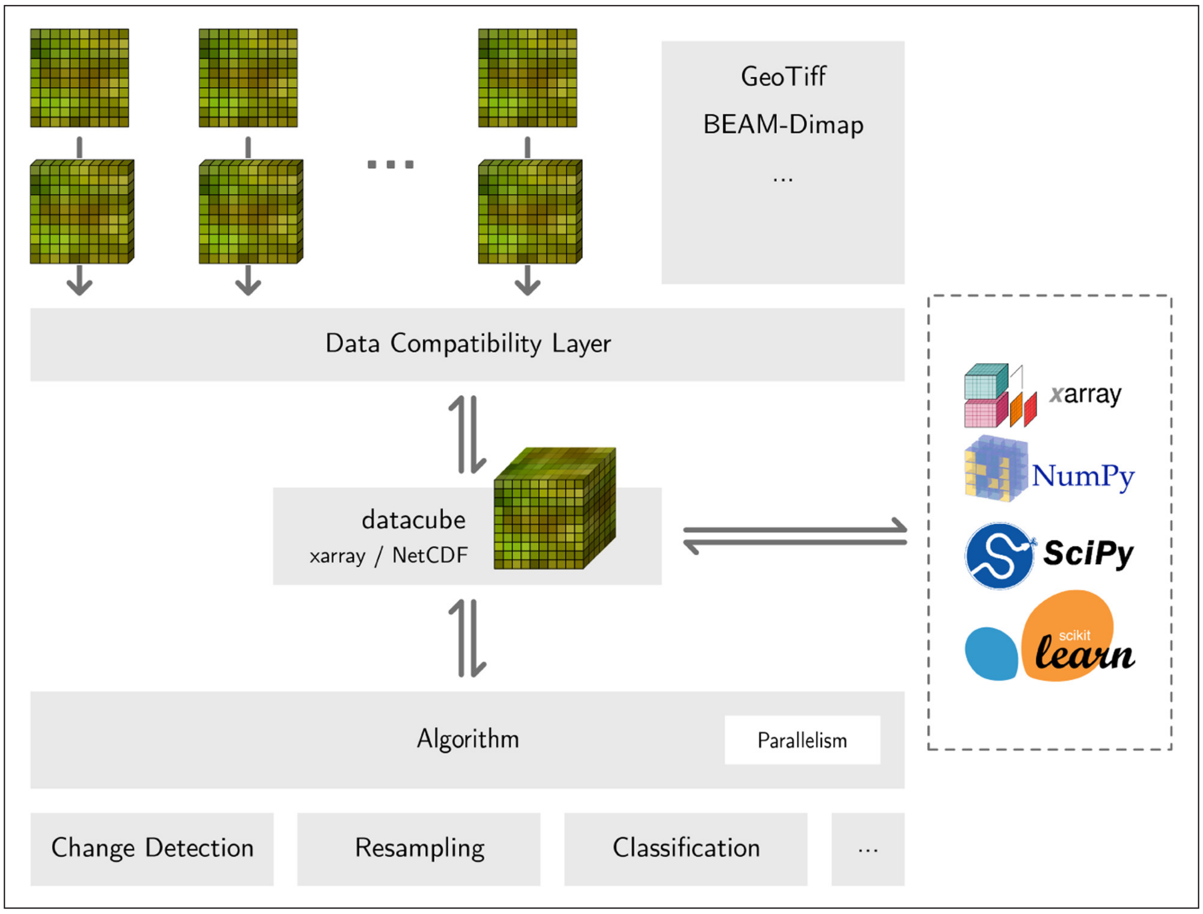

Figure 1 This diagram illustrates the principle ideas behind the software architecture.

at creating yet another data structure, but rather builds around the multivariate datacube as represented by an xarray Dataset.

The second important part of the library is the algorithms. The idea is to provide an abstract base class `Algorithm`, which contains the basic scaffold for any algorithm that may be added as a module. This includes basic parallelism and capabilities for distributed computing, e.g. using Python *dask* [3]. It further ensures that all algorithms follow the same basic structure in terms of parameters and outputs to ensure consistency across all modules and to reduce the learning curve for new developers and users. This is modeled after the way *scikit-learn* [14] implements its algorithms.

A number of basic algorithms will need to be implemented before the library becomes attractive for researchers. However, after reaching that threshold, the modular structure built on existing tools makes contributing easy and will invite algorithm contributions from other scientists.

The current implementation of this library is still in its early stages and has resulted from the very practical needs of applying multi-dimensional data analysis methods to Earth observation data. The code is hosted on GitHub [12] and also available from PyPI [11]. The full documentation is hosted on ReadThe Docs [10].

The package is currently structured according to the following submodules:

`nd.change` contains various change detection algorithms.
`nd.classify` provides an interface to apply `scikit-learn` classifiers to `xarray` Datasets.
`nd.filters` contains a selection of filters for *n*-dimensional datacubes, e.g. boxcar, Gaussian, arbitrary kernel convolutions, and non-local means.
`nd.io` collates all methods to read from and write to various file formats, largely wrapping `xarray` and `rasterio` methods.
`nd.testing` provides extra methods used by the unit tests.
`nd.tiling` allows for splitting of a dataset into tiles as well as merging tiles into a single dataset, including buffered tiles.
`nd.utils` is a collection of various utility functions used by other submodules.
`nd.vector` provides tools to work with vector data, in particular to rasterize such data to match a gridded reference dataset.
`nd.visualize` provides methods to create visualizations of xarray datasets, e.g. maps, RGB images, and video.
`nd.warp` contains functionality around reprojections between coordinate reference systems, resampling, etc.

The majority of the software functionality can be accessed via the xarray accessors `xarray.Dataset.nd` and `xarray.Dataset.filter`.

Several example use cases are shown in the tutorial notebooks provided with the repository, as well as in the documentation. A few of these examples are reproduced here.

Reprojection

Reprojecting a dataset into a new coordinate reference system can be easily achieved using `ds.nd.reproject()`. This function is a high-level wrapper around `rasterio`'s own reprojection methods and maintains the dataset structure including additional (non-spatial) dimensions. The following example demonstrates the reprojection on some GHRSST data [9].

```
ds_proj = ds.nd.reproject(
src_crs='epsg:4326',
dst_crs='epsg:2163')
```

Figure 2 shows the dataset before and after reprojection.

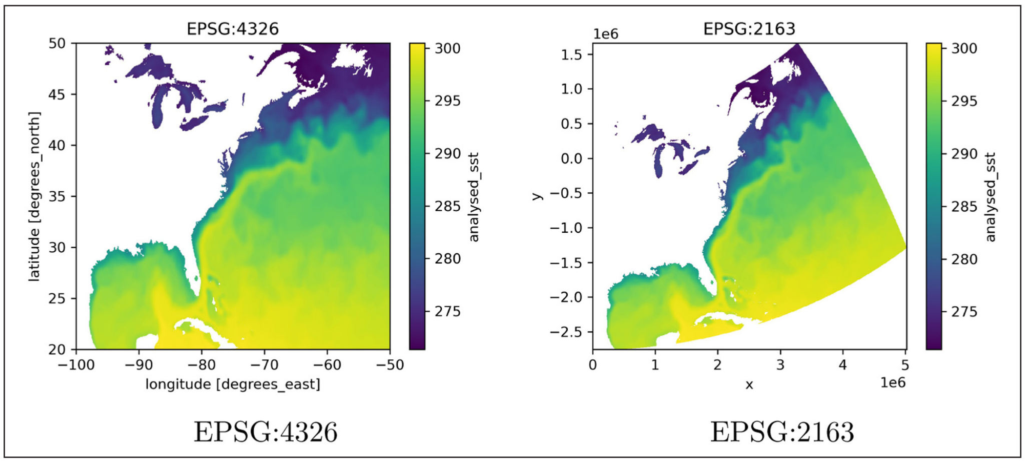

Figure 2 Reprojection of a subset of GHRSST data from EPSG:4326 (WGS84) to EPSG:2163 (US National Atlas Equal Area).

Applying a function across arbitrary dimensions

The following code snippet demonstrates how a function of arbitrary signature acting on any subset of dimensions can be easily mapped across a dataset using `ds.nd.apply()`. In this example we are reducing the time dimension by mapping the function `np.argmax` to find the hottest month for each pixel. While this example is quite trivial and could be solved without using `ds.nd.apply()`, the method is very flexible and can efficiently map functions of any signature over arbitrary dimensions.

```
hottest = ds.nd.apply(
np.argmax,
signature='(time)->()')
```

Figure 3 shows the result for this function call, displaying the hottest month per pixel for the GHRSST dataset.

Integration with *scikit-learn*

A shapefile containing class labels may be rasterized to match a given dataset as follows:

```
labels = nd.vector.rasterize('labels.shp',
ds)
```

To apply a scikit-learn classifier to an xarray dataset we can simply wrap it in a `Classifier` provided by `nd.classify`:

```
from nd.classify import Classifier
from sklearn.ensemble import
RandomForestClassifier
clf = Classifier(RandomForestClassifier(n_
estimators=10))
pred = clf.fit(ds, labels).predict(ds)
```

The outcome is demonstrated in ***Figure 4***.

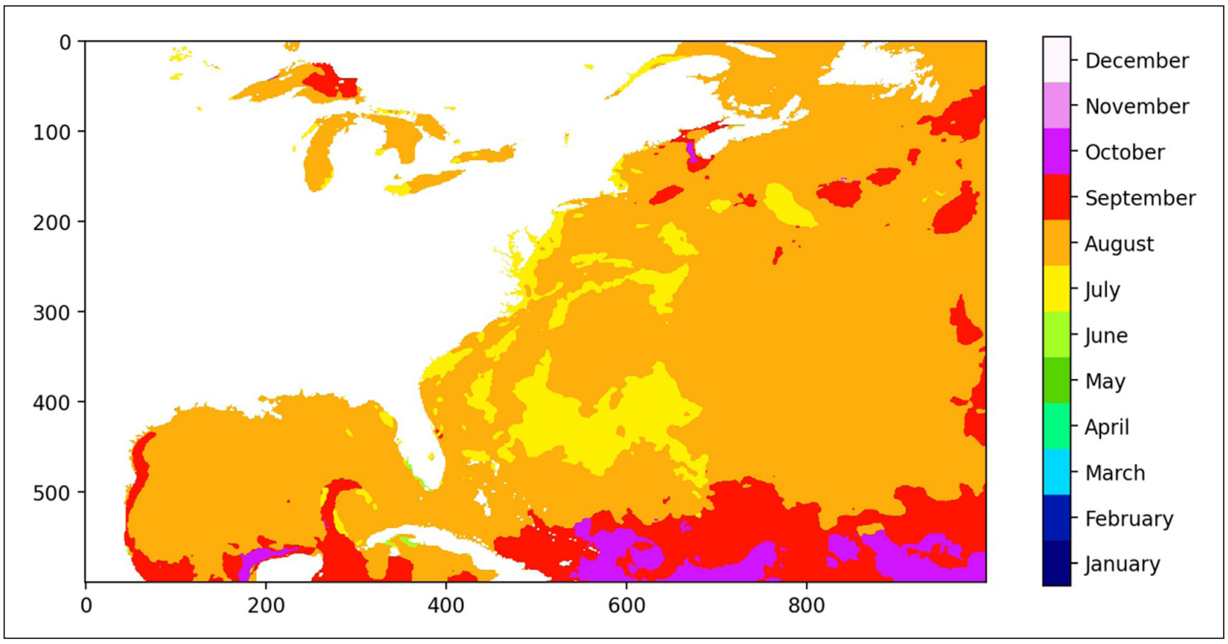

Figure 3 This figure shows the hottest month for each pixel in this subset of the GHRSST data. The result was obtained by mapping `np.argmax` over the dataset using `ds.nd.apply()`.

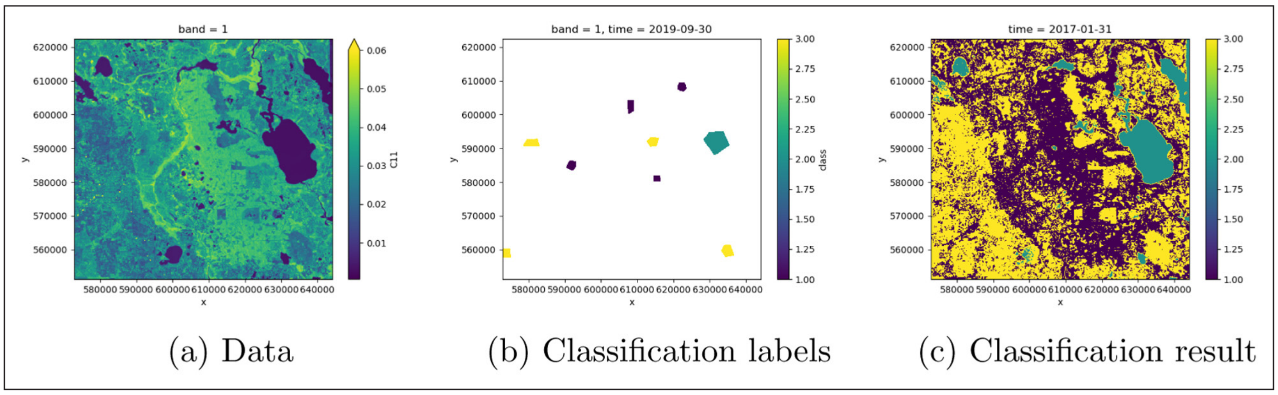

Figure 4 This figure demonstrates a sample classification outcome using Sentinel-1 data **(a)** with polygon labels **(b)**. The classification outcome is shown in **(c)**.

QUALITY CONTROL

An extensive suite of unit tests is provided with the software. The preferred way to run the tests is by using `pytest`. The tests are also run automatically on TravisCI at *https://travis-ci.com/github/jnhansen/nd* [16] for a variety of Python versions on Linux. The test coverage can be checked on Codecov at *https://codecov.io/gh/jnhansen/nd* [2].

The user may further choose to run the code in the provided sample notebooks in `nd/examples/` to test the software functionality.

(2) AVAILABILITY

OPERATING SYSTEM

Linux and Mac OS. Support for Windows is experimental.

PROGRAMMING LANGUAGE

Python, version 3.5 and later.

ADDITIONAL SYSTEM REQUIREMENTS

None.

DEPENDENCIES

nd is implemented in Python, with some extension modules written in C.

- `xarray`
- `scikit-learn`
- `GDAL`
- `libgsl-dev` (optional, for change detection)

LIST OF CONTRIBUTORS

1. Johannes N Hansen, sole contributor (University of Edinburgh)

SOFTWARE LOCATION

Archive

Name: Zenodo
Persistent identifier: *https://doi.org/10.5281/zenodo.4282546*
Licence: MIT
Publisher: Johannes Hansen
Version published: 0.2
Date published: 20/11/2020

Code repository

Name: GitHub
Persistent identifier: *https://github.com/jnhansen/nd*
Licence: MIT
Date published: 20/11/2020

The software is also available from the Python Package Index (PyPI) [11].

LANGUAGE

English.

(3) REUSE POTENTIAL

`xarray` deliberately does not provide any domain specific functionality and instead encourages the development of third-party libraries for this purpose. nd is trying to be this library for Earth Observation.

The software architecture follows a modular architecture that is easy to reuse and extend. An abstract base class `nd.Algorithm` is provided that may be subclassed by additional algorithms. Algorithms implemented in this way will benefit from features such as automatic parallelization and some automated unit testing.

Bug reports and feature requests should be issued via the GitHub repository. Contributors are welcome to submit pull requests on GitHub.

The software documentation is available at *https://nd.readthedocs.io/* [10].

ACKNOWLEDGEMENTS

The research during which this software was developed was supervised by Dr Stuart King and Professor Edward Mitchard at the University of Edinburgh. I am grateful for their support.

FUNDING INFORMATION

The software was developed as part of PhD studies funded by The Data Lab as well as the School of Mathematics and the School of GeoSciences at the University of Edinburgh.

COMPETING INTERESTS

The author has no competing interests to declare.

AUTHOR AFFILIATION

Johannes N. Hansen *orcid.org/0000-0003-0743-1332*
School of Mathematics, University of Edinburgh, GB

REFERENCES

1. **AWS Earth.** URL: *https://aws.amazon.com/earth/* (visited on 04/09/2021).
2. **Codecov report for nd.** URL: *https://codecov.io/gh/jnhansen/nd* (visited on 12/02/2020).

3. **Dask – Dask natively scales Python.** URL: *https://dask.org/* (visited on 12/02/2020).
4. **GDAL documentation – Raster drivers.** URL: *https://gdal.org/drivers/raster/index.html* (visited on 12/02/2020).
5. **Gillies S,** et al. *Rasterio: geospatial raster I/O for Python programmers.* Map-box, 2013-. URL: *https://github.com/mapbox/rasterio*.
6. **Gorelick N,** et al. “Google Earth Engine: Planetary-scale geospatial analysis for everyone”. In: *Remote Sensing of Environment* (2017). DOI: *https://doi.org/10.1016/j.rse.2017.06.031*
7. **Hansen JN, Mitchard ETA, King S.** “Assessing Forest/Non-Forest Separability Using Sentinel-1 C-Band Synthetic Aperture Radar”. In: *Remote Sensing*. 12.11 2020; 1899. DOI: *https://doi.org/10.3390/rs12111899*
8. **Hoyer S, Hamman JJ.** “xarray: N-D labeled Arrays and Datasets in Python”. In: *Journal of Open Research Software*. 5 (Apr. 2017); issn: 2049–9647. DOI: *https://doi.org/10.5334/jors.148*
9. **JPL OurOcean.** “GHRSST Level 4 G1SST Global Foundation Sea Surface Temperature Analysis”. In: *NASA Physical Oceanography DAAC* (2010). DOI: *https://doi.org/10.5067/GHG1S-4FP01*
10. **nd documentation.** URL: *https://nd.readthedocs.io/* (visited on 12/02/2020).
11. **nd package on PyPI.** URL: *https://pypi.org/project/nd/* (visited on 12/02/2020).
12. **nd repository on GitHub.** URL: *https://github.com/jnhansen/nd* (visited on 12/02/2020).
13. **Pandas Panel.** URL: *https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.Panel.html* (visited on 12/02/2020).
14. **scikit-learn – Machine Learning in Python.** URL: *http://scikit-learn.org/* (visited on 12/02/2020).
15. **SciPy.** URL: *https://scipy.org/* (visited on 04/09/2021).
16. **Travis CI.** URL: *https://travis-ci.org* (visited on 12/02/2020).

TO CITE THIS ARTICLE:
Hansen JN 2022 nd – A Framework for the Analysis of n-dimensional Earth Observation Data. *Journal of Open Research Software,* 10: 3. DOI: *https://doi.org/10.5334/jors.377*

Submitted: 15 April 2021 Accepted: 03 March 2022 Published: 11 March 2022