Software paper for submission to the Journal of Open Research Software

Please submit the completed paper to: editor.jors@ubiquitypress.com

---

(1) Overview

Title
A grid for multidimensional and multivariate spatial modelling and processing.

Paper Authors
1. Stål, Tobias 2. Anya M. Reading

Paper Author Roles and Affiliations
1, 2. School of Natural Sciences and Institute for Marine and Antarctic Studies, University of Tasmania, Hobart, Australia

Abstract
Complex spatial numerical models depend on multivariate data in various dimensionality, resolution and formats. It can be challenging to combine datasets, and it is time consuming to set up a robust spatial modelling frame.
We share a Python module containing basic functions to import various data sources to a defined multidimensional regular grid. The module also contains methods for visualisation and export. To facilitate reproducibility, the grid can point to original data sources. The module is written in intelligible high level Python, and uses well documented libraries as numpy, xarray and rasterio.

Keywords
Spatial processing, Python, Regular grid, Geophysics, Oceanography, Meteorology

Introduction
Models represent a target system in a defined part of the world. It can be both spatially and temporally constrained and represent data or phenomena [7]. Grids are used for discretization of parameter space of physical properties in the model. Each cell in a grid can represent a scalar or a vector from measurements or a modelled value. Cells can also be assigned a value by interpolation of nearby cells. Regular grids are build from congruent parallelepipeds and can be rectilinear or Cartesian, the later is a special case where the cells are unit squares or unit cubes. Smaller cells can sample higher frequencies, but too densely located cells can be a problem for computing. With higher dimensionality, the number of operations and memory allocation grows exponentially [34]. Gridded data are used for e.g. finite element analysis or finite difference computations. Standard tools include signal processing as frequency filtering, conversions, statistical analysis, classification and regression. Grids that represent part of Earth, must also be associated with a geodetic datum for reference to the physical world. A geographic coordinate system is also needed for conversion of grid indices to geographical coordinates.

Scientists use an increasing amount of time and effort to build software [38]. Tools to produce regular grids exist in many general-purpose programming languages, but there are a number of caveats in producing working multidimensional and multivariate models [34]. The input data often comes in different formats, as rasters, points in an irregular grid or vector data as polygons and lines. The resolution can vary over a large range. Dimensionality also varies from scalar constants to multidimensional vectors.

With increasing availability of data, organizing and distributing relevant datasets becomes important. Recently there have been a number of successful attempts to connect various data sources with potential users and provide open source modelling frames. The Quantarctica project by The Norwegian Polar Institute [30] makes datasets from various sources easily accessible for scientists and students. Quantarctica depends on a Geographical Information System (GIS) application [24], and even with some 3D functionality in recent upgrades, GIS is predominantly a 2D frame. Another relevant project is the multidimensional DataCube [16, 17]. DataCube preprocess and present remote sensing geographical and geophysical attributes for researchers and the broader public. DataCube is mainly targeted for changes in Landsat raster data over time, but has a broad range of applications. GemPy [3] is a recently developed full open-source method for geological modelling that facilitates stochastic geomodeling and probabilistic programming. The package uses the linear algebra compiler Theano [1] for efficient computation.

We share agrid, a framework to produce a regular grid for multidimensional and multivariate spatial modelling and processing. The scope for agrid is different from e.g. DataCube and GemPy in the sense that agrid is smaller, more general and more flexible. We provide methods of incorporating various formats of data, but the code is relatively light and the dependencies is kept to a minimum. Following the principles of Wilson et al. (2014), the code is written in highest possible language level and made readable and intelligible. In a research project, agrid can point directly to the original data sources. This simplifies the workflow, as development can be done in low resolution or small extent, but larger grids can be used to render results. agrid facilitates access to array operations in the spatial domains, as projected grid cells. Preprocessing and visualisation is moved from third part software or stand-alone applications to a condensed workflow that provides overview, reproducibility and flexibility to the research. The code is easy to modify to fit various needs.

agrid is written in Python 3, a high-level, general-purpose programming language [8]. Python is growing in popularity [28] and has become a standard tool for scientific computing [22]. Python is equipped with libraries for fast array operations in high level language [21, 35], basic statistics [18], signal processing and other scientific tools [4], machine learning [23], visualisation [37, 25] and discipline specific libraries for e.g. seismology [2, 19], astronomy[29] and GIS [10, 9, 14]. Python also provides interfaces for other languages as R, C and Fortran.

agrid was initially developed for studies of the Antarctic lithosphere [32, 33] and preprocessing of geophysical data for visualisation purposes [20, 26], but with updates as presented here, it can be used in any discipline, geographical region, projection, dimensionality and any resolution. We hope this contribution will be useful for col-

leagues, and hopefully encourage more scientists to embrace reproducible coding. The code is here presented in version 0.2.0. Subsequent versions will include additional functions for conversion and visualisation, support hexagonal 2D grids, curvilinear grid and increased polar and spherical functionality. We are happy to discuss further development and assist users if contacted by email, or directly via the github interface and pull requests. Python and libraries are also supported by large online communities.

Implementation and architecture

agrid is provided as a Python module that imports all dependencies and defines an agrid class object Grid() when imported. When calling Grid(), an object is created that represents the spatial extent of the model. The grid is initiated with projection, extent, initial resolution and initial depths. The parameters are used to define a default set of coordinates. Grid cells can be selected by geographical coordinates as well as index numbers in the grid.

When the instance is created, an xarray dataset is populated with the coordinates. A number of attributes are defined to describe the grid. Dimensions includes, but are not limited to space (X, Y, Z) and time (t). Some data contain additional dimensionality, e.g. satellite images can contain a number of frequency bands, models might include probability or likelihood. There is no upper limit of how many dimensions that can be used and a dimension does not need to have defined coordinates.

Additional coordinates with different resolution can be created and added to the object at any point. There is therefore no need to oversample low resolution data for computation with data with smaller cell size. Computations of grids of different resolution are achieved by creating an additional dimension of the low resolution grid in which the contained cells in the higher resolution are expanded. After the operation, the additional dimension is unfolded back to the higher resolution grid.

Extent is defined as left, right, up and down, and refers to the rectangular map view. At setup, there is an option of the grid representing the corners or the centre points of each cell. The default settings are a global grid in WGS84 (EPSG:4326), with a resolution of $1° \approx 111.1$km.

```
from agrid import *
km = 1000
ant = Grid(res = [10*km, 10*km],
        crs =3031,
        depths = [0*km, 10*km, 20*km, 50*km, 100*km],
        left = -3100*km,
        up  = 3100*km,
        right = 3100*km,
        down = -3100*km)
```

Listing 1: Initiation of a grid object.

Code 1 generates a frame of Antarctica, using WGS 84 / Antarctic Polar Stereographic projection and a lateral cell size of 10km × 10km. The Extent is defined

in the unit of the projection. As the crs is given as an integer, it is interpreted as an EPSG code. The 2D grid is Cartesian, but the depths slices are increasing, as defined by the list.
Usually, coordinates in a Cartesian grid are given as X - Y, but due to the convention of indexing arrays as row - column and geographical coordinates as lat - lon, grid coordinates are also given as Y - X for consistency, as exemplified in Listing 2.
The data is stored as data frames in a xarray dataset [12, 11]. xarray is build on numpy [21, 35] and pandas [18] and provides high level functions for labelled multidimensional datasets. xarray has a structure similar to netCDF file format[27], and netCDF is also the native format to store grids.
The instance of `Grid()` class also contains a number of functions to import data, visualisation and export (Fig. 1).
Raster data, e.g. geoTiff, can be imported with rasterio [10] and the underlying gdal [36]. Rasters are warped to fit the extent and projection of the grid. Vector data are imported with geopandas [14] and fiona with options for rasterization and interpolation. Grids or data points can be read from a number of formats and interpolated to the grid. The class also contains functions for visualisation using matplotlib [13] and matplotlib basemap [37]. Map views with e.g. coast lines and coordinates can be produced directly form the class. mayavi [25] and the underlying VTK [31] are used for 3D visualisation. Data can be saved as netCDF, geoTiff or ascii. Selected data points or frames can also be saved as vector data (Fig. 1).
Listing 2 produces an example of generating a map from 2D raster, Fig. 2.

```
# Import raster
ant.ds['ICE'] = (('Y', 'X'),
    ant.read_raster('data/bedmap2_tiff/bedmap2_thickness.tif'))

no_data = 32767.
ant.ds['ICE'] = ant.ds['ICE'].where(ant.ds['ICE'] != no_data)

#Import polygons
dranage, int_map = ant.assign_shape(
    'data/GSFC_DrainageSystems.shp',
    'ID', map_to_int = True,
    return_map = True)

ant.ds['DRANAGE'] = (('Y', 'X'), dranage)
ant.map_grid(ant.ds['DRANAGE'], cmap='viridis', cbar=True)

#Use ID from dict to select segments of map.
poly = [int_map[str(_) + 'g'] for _ in range(8,20)]

 #Select part of ice extent by polygons
ant.ds['SEL_ICE'] = ant.ds['ICE']*ant.ds['DRANAGE'].isin(poly)

#Import DEM for use in 3D oblique figure
ant.ds['DEM'] = (('Y', 'X'),
```

```
    ant.read_raster(str(source[2])) )

ant.ds['DEM'] = ant.ds['DEM'].where(ant.ds['DEM'] != no_data)

#Make maps
ant.map_grid('DRANAGE',
    cmap='Spectral',
    save_name== 'fig_2a.pdf')

ant.map_grid('ICE',
    cmap='viridis', cbar=True,
    save_name = 'fig_2b.pdf')

ant.map_grid('SEL_ICE',
    cmap = 'viridis',
    save_name = 'fig_2c.pdf')

ant.oblique_view('DEM',
    vmin= 0, vmax = 4200,
    cmap = 'bone' ,
    azimuth = 180, roll = -90,
    save_name = 'fig_2d.pdf')

#Analyse. E.g:
vol = int(ant.ds['SEL_ICE'].sum()*np.prod(ant.res)/km**3)
```

Listing 2: Import and vizualisation. Bedmap[6] and Antarctic drainage systems, GSFC [39]. This code generates all figures for Fig. 2 and calculates the volume of the grounded part of some Antarctic drainage systems.

Quality control

The module is published with a number of tutorials to demonstrate the functionality with different data sources. The code has been tested for different scales and extent. There are some limitations in the visualisation functions for less common projections. In future, matplotlib basemap might be replaced with a different library. Only limited error handling is included, as the used dependencies often contains good error handling techniques. Development errors have been investigated by comparing results from GIS applications. 2D data that have been imported, processed and exported have been compared to similar processing in GIS applications QGIS and ArcGIS. Those test cases are also availble from the github repository.

(2) Availability

Operating system

The code is developed and tested in Ubuntu 18.04 and macOS High Sierra 10.13.6. It has also been tested on Windows.

Programming language
Python >= 3.6 (tested on Python 3.6 and Python 3.7). The code might work with some modifications on earlier Python versions, but functionality cannot be guaranteed.

Additional system requirements
Very low requirements for basic use, but can be scaled up for larger grids. The use of dask arrays relax the need for large RAM.

Dependencies
The class depends on a number of Python packages: Minimum dependencies: basemap geopandas matplotlib numpy pyproj rasterio scipy xarray
Additional dependencies: fiona, imageio mayavi shapely

List of contributors
Tobias Stål, Anya M. Reading

Software location:
https://github.com/TobbeTripitaka/agrid

Name: agrid
Persistent identifier: https://doi.org/10.5281/zenodo.2553966
Licence: MIT License
Publisher: Tobias Stål
Version published: 0.2
Date published: 31/03/2019

Code repository

Name: GitHub
Persistent identifier: https://github.com/TobbeTripitaka/agrid.git
Licence: MIT
Date published: 01/03/2019

Language
agrid was developed in English.

(3) Reuse potential
agrid is deliberately developed for reuse in a broad range of applications. The code is commented and explained to guide and advice modifications. The code could be useful for any spatial processing and analysis in areas as solid Earth geophysics, oceanography, ice sheet modelling or hydrogeology. For some uses the complete code might be useful, but for other applications copied snippets or functions can be incorporated. The MIT license allows for a broad reuse.
This release of the code is presented with six tutorial notebooks that shows the functions and some toy applications. This paper is built from a simple SConstruct script that is also availble for reproduction [15, 5].
The presented version 0.2.0 is tested and used, but further features will be added to later updates and added to same repository.

Acknowledgements
We are grateful for discussion and test cases with Eleri Evans, Shawn Hood, Dr. Peter E Morse and Dr. Joanne Whittaker.

Funding statement
This research was supported under Australian Research Council's Special Research Initiative for Antarctic Gateway Partnership (Project ID SR140300001).

Competing interests
The author has no competing interests to declare.

References

[1] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, volume 1, 2010.

[2] Moritz Beyreuther, Robert Barsch, Lion Krischer, Tobias Megies, Yannik Behr, and Joachim Wassermann. ObsPy: A Python toolbox for seismology. *Seismological Research Letters*, 81(3):530–533, 2010. ISSN 0895-0695. DOI: `http://dx.doi.org/10.1785/gssrl.81.3.530`.

[3] Miguel de la Varga, Alexander Schaaf, and Florian Wellmann. Gempy 1.0: open-source stochastic geological modeling and inversion. *Geoscientific Model Development*, 2019.

[4] Eric Jones and Travis Oliphant and Pearu Peterson and others. SciPy: Open source scientific tools for Python, 2001. URL `http://www.scipy.org/`. [Online; accessed 13 Dec 2018].

[5] Sergey Fomel. Revisiting SEP tour with Madagascar and SCons. *Journal open research software*, 2013.

[6] P. Fretwell, H. D. Pritchard, D. G. Vaughan, J. L. Bamber, N. E. Barrand, R. Bell, C. Bianchi, R. G. Bingham, D. D. Blankenship, G. Casassa, G. Catania, D. Callens, H. Conway, A. J. Cook, H. F. J. Corr, D. Damaske, V. Damm, F. Ferraccioli, R. Forsberg, S. Fujita, Y. Gim, P. Gogineni, J. A. Griggs, R. C. A. Hindmarsh, P. Holmlund, J. W. Holt, R. W. Jacobel, A. Jenkins, W. Jokat, T. Jordan, E. C. King, J. Kohler, W. Krabill, M. Riger-Kusk, K. A. Langley, G. Leitchenkov, C. Leuschen, B. P. Luyendyk, K. Matsuoka, J. Mouginot, F. O. Nitsche, Y. Nogi, O. A. Nost, S. V. Popov, E. Rignot, D. M. Rippin, A. Rivera, J. Roberts, N. Ross, M. J. Siegert, A. M. Smith, D. Steinhage, M. Studinger, B. Sun, B. K. Tinto, B. C. Welch, D. Wilson, D. A. Young, C. Xiangbin, and A. Zirizzotti. Bedmap2: improved ice bed, surface and thickness datasets for Antarctica. *The Cryosphere*, 7(1):375–393, 2013. DOI: `http://dx.doi.org/10.5194/tcd-6-4305-2012`.

[7] Roman Frigg and Stephan Hartmann. Models in science. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2018 edition, 2018.

[8] G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

[9] Sean Gillies. The Shapely user manual, 2013. URL `https://pypi.org/project/Shapely/`.

[10] Sean Gillies et al. Rasterio: geospatial raster I/O for Python programmers, 2018. URL `https://github.com/mapbox/rasterio`.

[11] S. Hoyer and J. Hamman. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. DOI: `http://dx.doi.org/10.5334/jors.148`.

[12] Stephan Hoyer, Clark Fitzgerald, Joe Hamman, Akleeman, Thomas Kluyver, Maximilian Roos, Jonathan J. Helmus, Markel, Pete Cable, Fabien Maussion, Alistair Miles, Takeshi Kanmae, Phillip Wolfram, Scott Sinclair, Benoit Bovy, Ebrevdo, Rafael Guedes, Ryan Abernathey, Filipe, Spencer Hill, Ned Richards, Antony Lee, Nikolay Koldunov, Mike Graham, Maciekswat, Jeffrey Gerard, Igor Babuschkin, Christoph Deil, Erik Welch, and Andreas Hilboll. xarray: v0.8.0, 2016.

[13] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, May 2007. ISSN 1521-9615. DOI: `http://dx.doi.org/10.1109/MCSE.2007.55`.

[14] K Jordahl. GeoPandas: Python tools for geographic data. 2014. URL `https://github.com/geopandas/geopandas`.

[15] Steven Knight. Scons user guide. *Python Software Foundation*, 2010.

[16] Adam Lewis, Leo Lymburner, Matthew BJ Purss, Brendan Brooke, Ben Evans, Alex Ip, Arnold G Dekker, James R Irons, Stuart Minchin, Norman Mueller, et al. Rapid, high-resolution detection of environmental change over continental scales from satellite data–the Earth Observation Data Cube. *International Journal of Digital Earth*, 9(1):106–111, 2016.

[17] Adam Lewis, Simon Oliver, Leo Lymburner, Ben Evans, Lesley Wyborn, Norman Mueller, Gregory Raevksi, Jeremy Hooke, Rob Woodcock, Joshua Sixsmith, Wenjun Wu, Peter Tan, Fuqin Li, Brian Killough, Stuart Minchin, Dale Roberts, Damien Ayers, Biswajit Bala, John Dwyer, Arnold Dekker, Trevor Dhu, Andrew Hicks, Alex Ip, Matt Purss, Clare Richards, Stephen Sagar, Claire Trenham, Peter Wang, and Lan-Wei Wang. The Australian Geoscience Data Cube — Foundations and lessons learned. *Remote Sensing of Environment*, 202:276–292, December 2017. DOI: `http://dx.doi.org/10.1016/j.rse.2017.03.015`.

[18] Wes McKinney. pandas: a Python data analysis library. 2015. URL `http://pandas.pydata.org`.

[19] Tobias Megies, Moritz Beyreuther, Robert Barsch, Lion Krischer, and Joachim Wassermann. ObsPy - what can it do for data centers and observatories? *Annals of Geophysics*, 54(1):47–58, 2011. ISSN 1593-5213. DOI: `http://dx.doi.org/10.4401/ag-4838`.

[20] Peter Morse, Anya Reading, and Christopher Lueg. Animated analysis of geoscientific datasets: An interactive graphical application. *Computers & Geosciences*, 109:87–94, December 2017. DOI: `http://dx.doi.org/10.1016/j.cageo.2017.07.006`.

[21] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[22] Travis E. Oliphant. Python for Scientific Computing. *Computing in Science & Engineering*, 9(3):10–20, 2007. DOI: `http://dx.doi.org/10.1109/mcse.2007.58`.

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[24] QGIS. QGIS geographic information system. *Open Source Geospatial Foundation Project.*, 2015. By development team and community.

[25] Prabhu Ramachandran and Gael Varoquaux. Mayavi: 3D visualization of scientific data. *Computing in Science and Engineering*, 13(2):40–51, 2011. ISSN 1521-9615. DOI: `http://dx.doi.org/10.1109/MCSE.2011.35`.

[26] A. M. Reading, P. E. Morse, and T. Staal. Visualising uncertainty: interpreting quantified geoscientific inversion outputs for a diverse user community. *AGU Fall Meeting Abstracts*, December 2017.

[27] Rew, Russ and Davis, Glenn. NetCDF: An Interface for Scientific Data Access. *IEEE Computer Graphics and Applications*, 10(4):76–82, 1990. ISSN 0272-1716. DOI: `http://dx.doi.org/10.1109/38.56302`.

[28] David Robinson. The incredible growth of Python. *Stack Overflow Blog*, September 2017. URL `https://stackoverflow.blog/2017/09/06/incredible-growth-python/`. Retrieved on Jan 1st 2019.

[29] Thomas P Robitaille, Erik J Tollerud, Perry Greenfield, Michael Droettboom, Erik Bray, Tom Aldcroft, Matt Davis, Adam Ginsburg, Adrian M Price-Whelan, Wolfgang E Kerzendorf, et al. Astropy: A community python package for astronomy. *Astronomy & Astrophysics*, 558:A33, 2013.

[30] George Roth, Kenichi Matsuoka, Anders Skoglund, Yngve Melvær, and Stein Tronstad. Quantarctica: A unique, open, standalone GIS package for Antarctic research and education, February 2018. URL `http://quantarctica.npolar.no`.

[31] J Schöberl, Ken Martin, and Bill Lorensen. The Visualization Toolkit. Kitware,. Technical report, ISBN 978-1-930934-19-1, 2006. (VTK).

[32] Tobias Ståal, Anya M. Reading, Jacqueline Halpin, and Joanne Whittaker. A multivariate approach for mapping lithospheric domain boundaries in East Antarctica. *Manuscript submitted for publication.*, 2018.

[33] Tobias Ståal, Anya M. Reading, Jacqueline Halpin, and Joanne Whittaker. The Antarctic crust and lithosphere: A 3D model and framework for interdisciplinary research. *Manuscript in preparation.*, 2019.

[34] Joe F Thompson, Bharat K Soni, and Nigel P Weatherill. *Handbook of grid generation*. CRC press, 1998.

[35] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. DOI: `http://dx.doi.org/10.1109/mcse.2011.37`.

[36] Frank Warmerdam and GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation, 2018. URL `http://gdal.org`.

[37] J Whitaker. The matplotlib basemap toolkit user's guide. *Matplotlib Basemap Toolkit documentation, February*, 2011.

[38] Greg Wilson, D A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven H D Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, and Others. Best practices for scientific computing. *PLoS biology*, 12(1):e1001745, 2014. DOI: `http://dx.doi.org/10.1371/journal.pbio.1001745`.

[39] H Jay Zwally, Mario B Giovinetto, Matthew A Beckley, and Jack L Saba. Antarctic and greenland drainage systems, gsfc cryospheric sciences laboratory. *Available at icesat4. gsfc. nasa. gov/cryo_data/ant_grn_drainage_systems. php. Accessed March*, 1:2015, 2012.

Authors are able to enter into separate, additional contractual arrangements for the non-exclusive distribution of the journal's published version of the work (e.g., post it to an institutional repository or publish it in a book), with an acknowledgement of its initial publication in this journal.

By submitting this paper you agree to the terms of this Copyright Notice, which will apply to this submission if and when it is published by this journal.

Figures

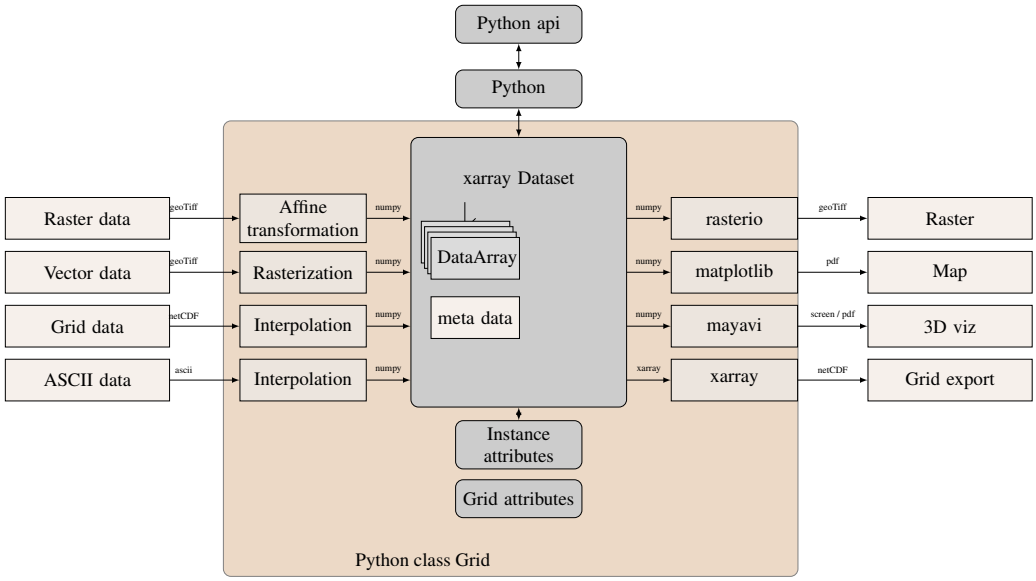

Figure 1: Structure of the code. A class object (brown) contains functions for e.g. import and export. It also contains a xarray dataset (gray) and attributes. Various data formats (left) are converted to numpy arrays and interoperated as data arrays in an xarray dataset. Each data array can be associated to coordinates. The dataset also contains metadata. Data can be exported or visualized (right). The class object, it's content as the dataset and data array can be reached by Python (top).

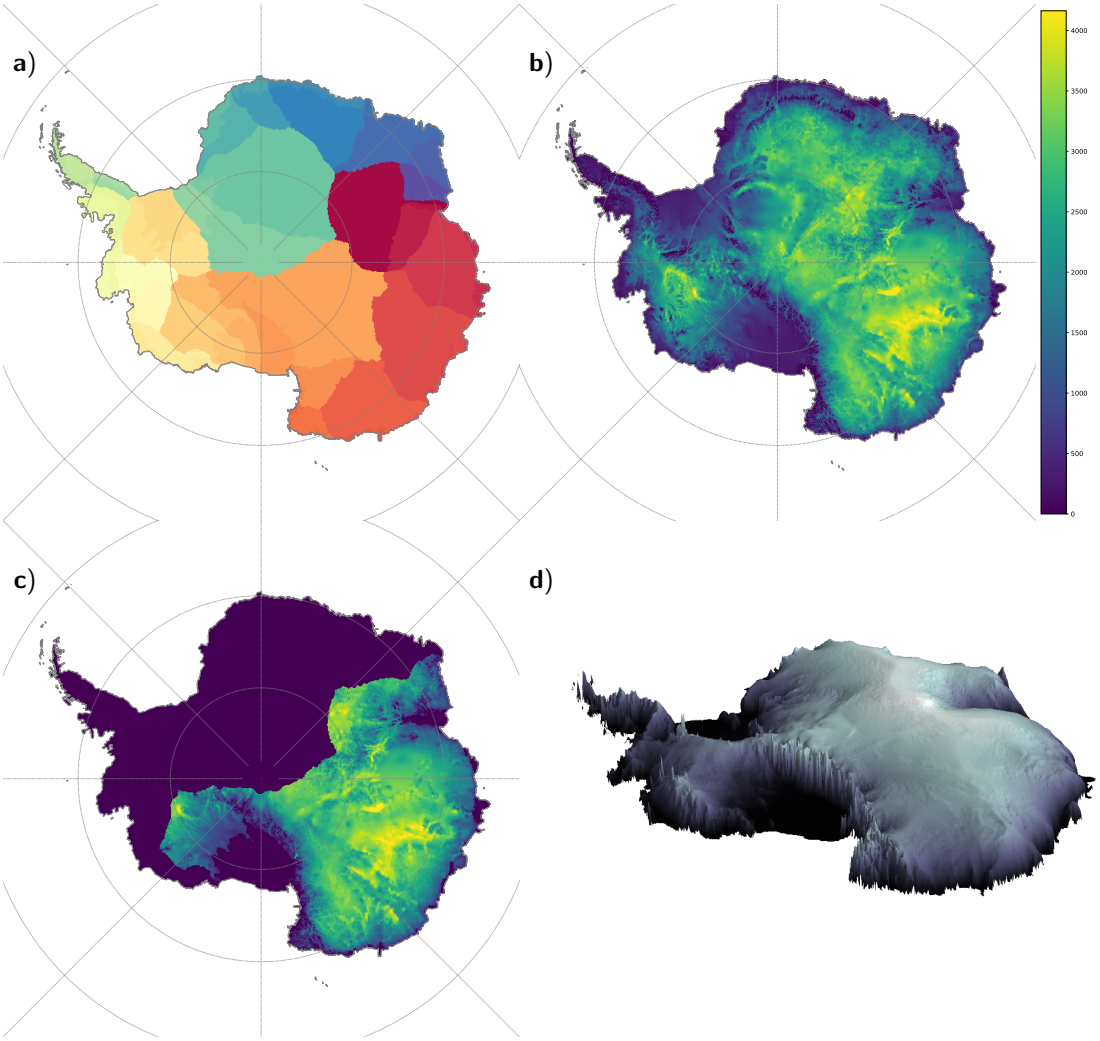

Figure 2: Map examples generated by code listing 1 and 2. (a) vector polygon data (drainage systems [39]). (b) raster data (subglacial elevation [6]). (c) Polygons used to select part of raster data. (d) Example of 3D rendering. See supplied tutorials and SCons script for more details.