COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

Analysis and Visualization toolkit for plaNetary Inferences

Software User Manual

AVNI

Version 0.1.0

(generated October 24, 2023)

Pritwiraj (Raj) Moulik

with contributions by (in alphabetical order):
Rene Gassmoeller, Chris Havlin, Ross Maguire

Website: avni.globalseismology.org

©Copyright 2021, Pritwiraj (Raj) Moulik, AVNI Project and Regents of the University of California

CONTENTS

This guide is here to help you start creating interactive 3D plots with AVNI with the help of our examples and tutorials.

CHAPTER

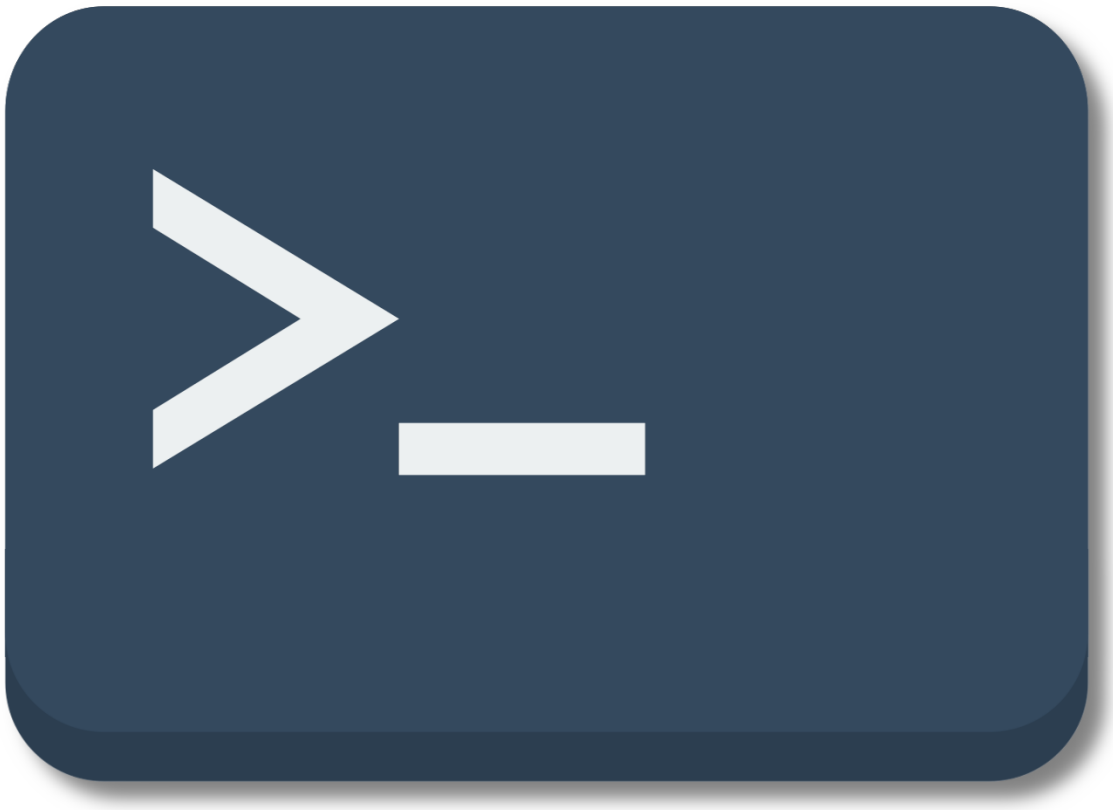
ONE

INSTALLATION

The first prerequisite for installing AVNI is Python itself. If you don't have Python yet and want the simplest way to get started, we recommend you use the Anaconda Distribution. The second prerequisite is the gfortran Fortran compiler. Once Fortran is installed, please set the environment variable *F90* in your command line shell to *gfortran*.

Install via `pip`

For Advanced Users



Already familiar with Python? Follow our advanced setup instructions for `pip`!

Setup Instructions

CHAPTER

TWO

SUPPORT

For general questions about the project, its applications, or about software usage, please create a discussion in AVNI Forum where the community can collectively address your questions. You are also send one of the developers an email. The project support team can be reached at avni@globalseismology.org.

CHAPTER

THREE

CITING AVNI

If you are using AVNI in your scientific research, please help our scientific visibility by citing our work! Head over to cite page to learn more about citing AVNI.

CHAPTER

FOUR

DOCUMENTATION OVERVIEW

Note: If you haven't already installed AVNI, please take a look at our installation guides. Please also kindly find some resources for learn_python if you need to.

The documentation for AVNI is divided into three main sections:

1. The *API reference* provides documentation for the classes, functions and methods in the AVNI codebase. This is the same information that is rendered when running `help(avni.<function_name>)` in an interactive Python session, or when typing `avni.<function_name>?` in an IPython session or Jupyter notebook.
2. The frequenty asked questions (FAQ) provides answers to several common questions about the AVNI codebase. Feel feel free to reach out to us to get *more help*

Important: Checkout the *project governance*, *design philosophy* and *code of conduct* pages for a broad overview of the project expectations and outlook.

4.1 Project Governance

The purpose of this document is to formalize the governance process used by the AVNI project and to clarify how decisions are made and how the various elements of our community interact. This document elucidates the relationship between open source collaborative development and work that may be funded by entities that require private development.

4.1.1 The Project

The AVNI software ecosystem (The Project) is designed to be an open source software project. The goal of The Project is to develop open source software for analysis of geoscience data in Python. The Project is released under the GNU GPL v3 (or similar) license, and is hosted publicly under the `globalseismology` GitHub organization.

The Project is developed by a team of distributed developers, called Contributors. Contributors are individuals who have contributed code, documentation, designs, or other work to the Project.

Anyone can be a Contributor. Contributors can be affiliated with any legal entity or none. Contributors participate in the project by submitting, reviewing, and discussing GitHub Pull Requests and Issues and participating in open and public Project discussions on GitHub and other channels. The foundation of Project participation is openness and transparency.

The Project Community consists of all Contributors and Users of the Project. Contributors work on behalf of and are responsible to the larger Project Community and we strive to keep the barrier between Contributors and Users as low as possible.

The Project is not a legal entity, nor does it currently have any formal relationships with legal entities.

4.1.2 Community Effort

AVNI software ecosystem (The Project) is part of the wider community effort to create a three-dimensional reference Earth model REM3D. The Project aims to provide tools that reduce the barrier of entry for the adoption of Earth models and datasets. A description of the REM3D project is provided in[1][2], and a complete list of relevant papers is provided below:

BibTeX for REM3D

```
@article{MoulikEtAl2021,
        doi = {10.1093/gji/ggab418},
        issn = {0956-540X},
        journal = {Geophysical Journal International},
        month = {10},
        number = {3},
        pages = {1808-1849},
        title = {{Global reference seismological data sets: multimode surface␣
↪wave dispersion}},
        volume = {228},
        year = {2021},
        author = {Moulik, P and Lekic, V and Romanowicz, B and Ma, Z and␣
↪Schaeffer, A and Ho, T and Beucler, E and Debayle, E and Deuss, A and Durand,␣
↪S and Ekström, G and Lebedev, S and Masters, G and Priestley, K and Ritsema, J␣
↪and Sigloch, K and Trampert, J and Dziewonski, A M}}

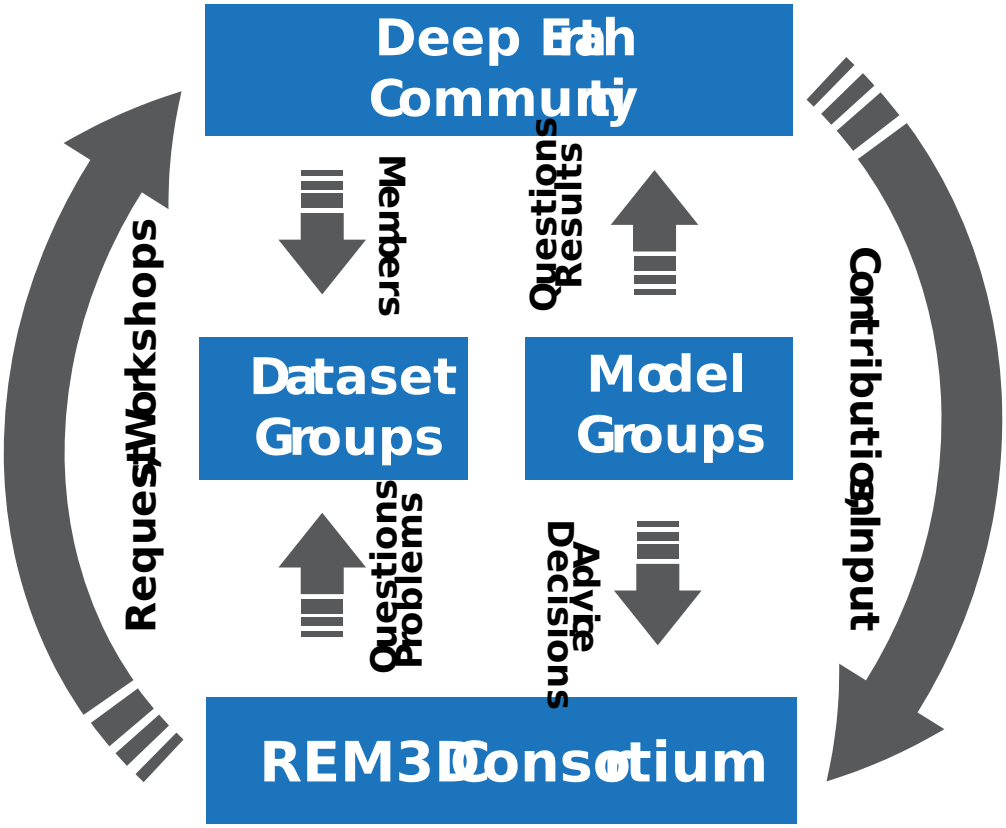

@misc{REM3D_AGU_2022,
        abstract = {{Reconciliation of techniques, models and data has emerged␣
↪as a frontier area for deep Earth exploration. Past results have proven␣
```

[1] P Moulik, V Lekic, B Romanowicz, Z Ma, A Schaeffer, T Ho, E Beucler, E Debayle, A Deuss, S Durand, G Ekström, S Lebedev, G Masters, K Priestley, J Ritsema, K Sigloch, J Trampert, and A M Dziewonski. Global reference seismological data sets: multimode surface wave dispersion. *Geophysical Journal International*, 228(3):1808–1849, 10 2021. doi:10.1093/gji/ggab418.

[2] P Moulik and The 3D Reference Earth Model (REM3D) Consortium. Three-dimensional Reference Earth Model Project: Data, Techniques, Models & Tools. American Geophysical Union (AGU) Fall Meeting, Chicago, IL, USA, 2022. doi:10.5281/zenodo.7883683.

```
↪indispensable for assessing earthquake hazard, characterizing plate tectonics,
↪ elucidating material properties under extreme conditions, imaging interior␣
↪structure, and as a general reference in other fields. We present advancements␣
↪on a three-dimensional reference Earth model (REM3D) that captures the␣
↪consensus view of heterogeneity in the mantle. Progress in modeling the␣
↪Earth's interior is driven by diverse data, ranging from astronomic-geodetic␣
↪constraints to full seismic waveforms and derivative measurements of body␣
↪waves (~ 1 - 20s), surface waves (~ 20 - 300s) and normal modes (~ 250 -␣
↪3000s). Reconciliation of data involves retrieving the missing metadata,␣
↪archiving in scalable storage formats, documenting outliers indicative of␣
↪the limitations in some techniques, and quantifying summary reference data␣
↪with uncertainties. Building on our recent work on reference surface-wave␣
↪dispersion datasets, arrival times of primary, diffracted, and reflected␣
↪phases from the transition-zone and deeper discontinuities are reconciled for␣
↪a body-wave reference dataset. This procedure involves revised techniques and␣
↪archival formats for the processing of frequency-dependent arrival times. A␣
↪revised dataset of normal-mode eigenfrequencies, quality factors and splitting␣
↪is reconciled with updated uncertainties based on inter-catalog consistencies.␣
↪Full-spectrum tomography uses these diverse observations to constrain physical␣
↪properties - seismic velocity, anisotropy, density, attenuation and the␣
↪topography of discontinuities - in variable spatial resolution. This technique␣
↪is expanded to include long-wavelength geoid as an additional constraint on␣
↪density variations. All geoscience workflows typically involve querying data␣
↪or models in order to make inferences. Analysis and Visualization toolkit␣
↪for plaNetary Inferences (AVNI) is a web-based software environment powered␣
↪by Python that facilitates these computational workflows. AVNI tools are␣
↪web-based so that the shared resources are accessed by authenticated users␣
↪through Application Programming Interfaces (APIs), without the overhead of␣
↪storing data, compiling and running intensive codes.}},
       author       = {Moulik, P and
                       The 3D Reference Earth Model (REM3D) Consortium},
       title        = {{Three-dimensional Reference Earth Model Project:
                        Data, Techniques, Models \& Tools}},
       howpublished = {American Geophysical Union (AGU) Fall Meeting, Chicago,␣
↪IL, USA},
       date-added = {2022-06-07 18:05:07 -0400},
       date-modified = {2022-06-07 18:07:23 -0400},
       doi          = {10.5281/zenodo.7883683},
       year = {2022}}
```

While AVNI is a distinct project from REM3D, the avenues of community pariticipation are similar. In order to maximize the likelihood of success and utility to the broader deep Earth community, the REM3D project receives input from two advisory working groups, one focused on the reference dataset, and the other on the reference model. Both working groups provide feedback and data to the REM3D project theough various channels.



4.1.3 Governance model

This section describes the governance and leadership model of The Project.

The foundations of Project governance are:

- openness and transparency
- active contribution
- institutional neutrality

Traditionally, Project leadership is provided by a subset of Contributors, informally called Core Developers, whose active and consistent contributions were rewarded by granting them "commit rights" to the Project GitHub repositories. In general, all Project decisions are made through consensus among the Core Developers with input from the Community.

4.1.4 Document history

https://github.com/globalseismology/avni/commits/main/docs/overview/governance.rst

4.1.5 License

To the extent possible under law, the authors have waived all copyright and related or neighboring rights to the AVNI project governance document, as per the terms of our license.

4.2 Design philosophy

4.2.1 Interactive versus scripted analysis

AVNI has some great interactive plotting abilities that can help you explore your data, and there are a few GUI-like interactive plotting commands (like browsing through the raw data and clicking to mark bad channels, or click-and-dragging to annotate bad temporal spans). But in general it is not possible to use AVNI to mouse-click your way to a finished, publishable analysis. AVNI works best when you assemble your analysis pipeline into one or more Python scripts. On the plus side, your scripts act as a record of everything you did in your analysis, making it easy to tweak your analysis later and/or share it with others (including your future self).

4.2.2 Integration with the scientific python stack

AVNI also integrates well with other standard scientific Python libraries. For example, AVNI objects underlyingly store their data in NumPy arrays, making it easy to apply custom algorithms or pass your data into one of scikit-learn's machine learning pipelines. AVNI's 2-D plotting functions also return matplotlib `Figure` objects, so you can customize your AVNI plots using any of matplotlib or AVNI's plotting commands. The intent is that AVNI will get most geoscientist 90% of the way to their desired analysis goal, and other packages can get them over the finish line.

4.2.3 Submodule-based organization

A useful-to-know organizing principle is that AVNI objects and functions are separated into submodules. This can help you discover related functions if you're using an editor that supports tab-completion. For example, you can type `avni.tools.<TAB>` to see all the functions in the tools submodule; similarly for model functions (*avni.models*), functions for reading and writing data (`avni.io`), mapping (*avni.mapping*), etc. This also helps save keystrokes — instead of:

```
import avni
avni.tools.bases.eval_vbspl(...)
avni.tools.bases.eval_splrem(...)
```

you can import submodules directly, and use just the submodule name to access its functions:

```
from avni.tools import bases
bases.eval_vbspl(...)
bases.eval_splrem(...)
```

4.2.4 (Mostly) unified API

Whenever possible, we've tried to provide a unified API for the different data classes. For example, the `Reference1D` and `Model3D` classes all have a `plot()` method that can typically be called with no parameters specified and still yield an informative plot of the data. Similarly, they all have the methods like `copy()` with similar or identical method signatures.

4.2.5 In-place operation

Because gescience datasets can be quite large, AVNI tries very hard to avoid making unnecessary copies of your data behind-the-scenes. To further improve memory efficiency, many object methods operate in-place (and silently return their object to allow method chaining). In-place operation may lead you to frequent use of the `copy()` method during interactive, exploratory analysis — so you can try out different preprocessing approaches or parameter settings without having to re-load the data each time — but it can also be a big memory-saver when applying a finished script to different models or datasets.

4.3 Community Code of Conduct

4.3.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors, users and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion (or lack thereof), sexual identity and orientation, or technology choices.

4.3.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Conduct which could reasonably be considered inappropriate in a professional setting

4.3.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful. We see all of these actions as last resorts. Our goal is to maintain a friendly and inclusive community where everyone feels welcome to participate and express themselves, but conduct by individuals that jeopardizes the harmony of the project will need to be addressed.

4.3.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

4.3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting AVNI maintainers at avni@globalseismology.org, or CIG leadership at private@geodynamics.org. CIG leadership will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. CIG leadership is obligated to maintain confidentiality with regard to the reporter of an incident, although confidentiality cannot be promised in all situations under the Title IX law of the United States. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

4.3.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant homepage, version 1.4, available here and draws on the Community Code of Conduct of the Computational Infrastructure for Geodynamics, available here.

CHAPTER

FIVE

PYTHON API REFERENCE

This is the reference for classes (`CamelCase` names) and functions (`underscore_case` names) of AVNI, grouped thematically by analysis stage. Functions and classes that are not below a module heading are found in the `avni` namespace.

avni:

5.1 avni.api package

5.1.1 Submodules

avni.api.applet_support module

api support for web_applets. Not recommended for general use.

class avni.api.applet_support.**App**(*client*)

Bases: `object`

generic app support class

class avni.api.applet_support.**SW**(*client*)

Bases: *avni.api.applet_support.App*

surface wave specific app support

buildCommonData(*self*, *args) for SW app. builds events common to two SW data for user selections returns filename (file stored in cache*)

args={ 'grp_1': group 1 abbreviation 'grp_2': group 2 abbrevation 'period': period to compare at. string, e.g., '100.0' 'overtone':overtone to compare at. string, e.g., 0 'wave':wave type 'R' or 'L' 'orbit': orbit 1, 2, 3. concanenated with wave to get R1, R2, etcself. 'dat_type': 'processed' or 'raw' }

selections must exist in HDF keys.

filterCommonData(*self*, *args) for SW app: args includes plotly/dash graphical selections. See avni-applets. returns filename (file stored in cache*)

avni.api.client module

client for accessing avni api

class avni.api.client.**Client**(*api_key=''*, *timeout=300*, *api_init_file=None*)

Bases: object

checkConnection()

checks if the avni api is accessible

call(*path*, *parameters=None*, *time_out=None*)

general form for api calls, returns json as python dictionary

checkUserStats()

checks stats for a user

searchForApiKey(*api_init_file=None*)

searches for API Key Input: ——

api_init_file: the api.ini file full path. If None, will default to looking in avni/config/api.ini

api_key: the api key. Empty string if not found

setApiConfig(*api_key=''*, *api_init_file=None*)

builds the api.ini file Input: ——

api_key: the api key to store. Will pull from self.key if not specified.

api_init_file: the file to store key (subsequents calls to api will need to specify the directory if not using the default). Default is avni/config/api.ini

no output

avni.api.cmt module

global cmt api class

class avni.api.cmt.**CMT**(*client*)

Bases: object

fetchCMTevents(*cmt_filters={}*)

fetches events from CMT, returns pandas dataframe. Inputs:

cmt_filters = dictionary for filtering: keys can be: 'date_start','date_end','depth_min_km', 'depth_max_km','mag_min' or 'mag_max' date keys are strings: 'YYYY-MM-DD' or 'YYYY-MM-DD HH:MM:SS' other keys are integer or float values. API defaults mag_min to 5, no other defaults.

avni.api.f2py module

class avni.api.f2py.**f2pyWrapper**(*client*)

Bases: object

f2py wrapper

listf2py()

lists available f2py functions in config/f2py.ini

args: None

result: dictionary of avaialable f2py functions & arguments

callf2py(*function='f2pyfun'*, *args={}*)

Abstract function for calling f2py functions in a general way

function: the function to call (string, required) args: kwdict for f2py function (required if function has input)

result: result of the f2py function call

jsonF2pyArgs(*args*, *function*)

converts f2py arguments into json-able form, stores f2py positional order and expected type. returns json of the dict.

formatResult(*result*, *function*)

returns f2py result in expected tuple, preserving positioning

avni.api.model module

model api class

class avni.api.model.**Model**(*client*)

Bases: object

listModels(*args={}*)

Fetch a list of available models.

none

result: dictionary containing list of models and model info.

addConfigDescriptions(*ModelList={}*)

loads kernel, model descriptions from config object

evaluate_points(*args_in={}*)

Evaluates list of lat/lon/depth points for a given parameter ('vs')

args_in: dictionary of arguments:

required args: 'lat': latitude in degrees 'lon': longitude in degrees 'depth': depth in km

lat, lon and depth are scalars, lists or numpy arrays. arrays must be 1d, all must be the same size.

optional args (default): 'model': model to use, string ('S362ANI+M') 'kernel': model to use, string ('BOX25km_PIX1X1') 'parameter': parameter to fetch, string ('vs') 'interpolated': 1/0 for interpolation with KdTree (1)

model+kernel must match a model file. (see listModels() method)

result: dictionary with scalar/list/numpy array of same size as input

`depthProfile(`*args_in={}*, *return_numpy=True*`)`

Evaluates a depth profile centered on provided lat/lon for a given parameter ('vs')

args_in: dictionary of arguments:

required args: 'lat': latitude in degrees 'lon': longitude in degrees

lat, lon and depth are scalars

optional args (default): 'N_depth': depth in km, integer (100) 'depthMin': min depth, float (0.) 'depthMax': max depth, float (2890) 'model': model to use, string ('S362ANI+M') 'kernel': model to use, string ('BOX25km_PIX1X1') 'parameter': parameter to fetch, string ('vs') 'interpolated': 1/0 for interpolation with KdTree (1)

model+kernel must match a model file. (see listModels() method)

result: dictionary with numpy array of parameter, depth

`crossSection(`*args={}*`)`

Interpolation of cross-section along a great circle path

args: dictionary of arguments:

required args: 'lat': starting latitude in degrees, float, 'lon': starting longitude in degrees, float, 'azimuth': direction to move from starting lat/lon

(degrees clockwise from North, 90=East, 270=West)

'gcdelta': great circle distance (degrees) to move along azimuth

optional args (default): 'model': model to use, string ('S362ANI+M') 'kernel': model to use, string ('BOX25km_PIX1X1') 'parameter': parameter to fetch, string ('vs') 'interpolated': 1/0 for interpolation with KdTree (1) 'quickInterp': 1/0 for quick interpolation (0) 'includeTopo': 1/0 for incuding topography along transect (0)

model+kernel must match a model file. (see listModels() method)

result: dictionary of results, including numpy arrays

if args['parameter']='vs',

result={'parameter':string with parameter name, e.g., 'vs' 'vs': 2d numpy array, 'depth': 1d array,depth in km, 'lat': 1d array, latitude of surface points, 'lon': 1d array, longitude of surface points, 'theta': 1d array,

angular distance along transect [degrees] 'topo': 1d array, topography along transect if includeTopo }

`fixedDepth(`*args={}*`)`

Interpolation at a fixed depth.

args: dictionary of arguments:

required args: 'depth': the constant depth to use in km

optional args: 'lat1': starting latitude in degrees, float (-90.) 'lon1': starting longitude in degrees, float (0.) 'lat2': end latitude in degrees, float (90.) 'lon2': end longitude in degrees, float (360.) if using any of the optional above args, you must use all of them. The result will be values in a grid formed by taking start and end coordinates are the opposing points of a box with the vertex coordinates (lat2,lon1), (lat2,lon2), (lat1,lon1), (lat1,lon2). If not specifying coordinates, will extract values at a fixed depth for the whole earth.

'Nlat': number of latitude points to extract, int (100) 'Nlon': number of longitude points to extract, int (200) 'model': model to use, string ('S362ANI+M') 'kernel': model to use, string ('BOX25km_PIX1X1') 'parameter': parameter to fetch, string ('vs') 'interpolated': 1/0 for interpolation with KdTree (1) 'quickInterp': 1/0 for quick interpolation (0)

model+kernel must match a model file. (see listModels() method)

result: dictionary of results, including numpy arrays

if args['parameter']='vs',

result={'parameter':string with parameter name, e.g., 'vs' 'vs': 2d numpy array with shape (len(lat),len(lon)), 'lat': 1d array, latitude 'lon': 1d array, longitude }

avni.api.traveltimes module

travel time api class

`class avni.api.traveltimes.TT(`*client*`)`

Bases: `object`

`listModels(`*args={}*`)`

Fetch a list of available models.

Parameters

none

`predictPaths(`*args*`)`

predicts travel times for the specified model and paths.

Parameters

args dictionary, with required and optional keys as follows:

for model tracing, user must provide one of the following three

(1) 'start_lat','start_lon','end_lat','end_lon' [start and end] coordinates in degrees

(2) 'start_lat','start_lon','azimuth','dist_deg': start coordinates with azimuth and great circle distance in degrees.

(3) 'dist_deg' great circle distance in degrees (FOR 1D tracing ONLY)

additional required keys:

'source_depth_km' source depth in km

Additional optional parameters

'type' tracing type, '1D' or '3D' (default is '1D'), string

'component' tracing component, default 'PSV', string

'model' the model to use, default is 'PREM_tt_table.h5'. Use listModels() to see available models

'phase' seismic phase to trace, default 'PP'. Use listPhases() to see available phases

`processPathArgs`(*self*, *args*)

processes path arguments in args dict, called by predictPaths()

`listPhases`()

lists available phases for component of a given model. Use listModels() to see list of available models.

Parameters

args dictionary with optional keys: 'type' '1D' or '3D', default '1D' 'model' model to check for phases, default is 'PREM_tt_table.h5'

5.2 avni.data package

5.2.1 Submodules

avni.data.CMT module

`avni.data.CMT.update_gcmt_nbn`(*standard='allorder.nbn'*, *quick='qcmt.nbn'*)

Updates the NBN file containing moment tensors from the Global CMT Project.

standard : file used in the standard catalogs of published CMTs

quick : file containing quick and unpublished CMTs from recent earthquakes

avni.data.CMT.**load_gcmt_nbn**(*choice=1*, *standard='allorder.nbn'*, *quick='qcmt.nbn'*)

Updates the NBN file containing moment tensors from the Global CMT Project.

choice: 1 for both standard/quick (default), 2 for standard, 3 for quick

standard : file used in the standard catalogs of published CMTs

quick : file containing quick and unpublished CMTs from recent earthquakes

avni.data.CMT.**get_gcmt_info**(*cmtname*, *prefixes=['J', 'C']*)

Updates the NBN file containing moment tensors from the Global CMT Project.

cmtname : CMTNAME of the earthquake in the Global CMT catalog.

prefix : Some prefixes that are also checked if cmtname is not found.

avni.data.NM module

avni.data.NM.**read_rts_catalog**(*infile*, *base_units=True*)

reads a mode catalog in bimary rts format

base_units: convert from native units to base units in constants

avni.data.NM.**write_modes_hdf**(*infile*, *table_name*, *absorption_band='None'*)

writes an hdf5 file based on the output of getgennomo(?)

params: infile <str>: path to mode catalog produced by getgennomo table_name <str>: name of hdf5 modes table that will be written

avni.data.NM.**get_mode_attribute**(*table*, *mode_type*, *radial_order*, *angular_order*, *attribute*)

returns the eigenfrequency of a single mode

params: table <str>: path to hdf5 format modes table mode_type <str>: either "spheroidal", "toroidal", or "radial" radial_order <int>: radial order (ie. overtone number) angular_order <int>: angular order attribute: characteristics of a mode such as pvel,gvel,omega

returns: value: value of attribute queried

avni.data.NM.**get_mode_freq**(*table*, *mode_type*, *radial_order*, *angular_order*, *freq_units='mhz'*)

returns the eigenfrequency of a single mode

params: table <str>: path to hdf5 format modes table mode_type <str>: either "spheroidal", "toroidal", or "radial" radial_order <int>: radial order (ie. overtone number) angular_order <int>: angular order

returns: freq: eigen frequency of the mode in freq_units

avni.data.SW module

avni.data.SW.**get_travel_times1D**(*table*, *distance_in_degree*, *period*, *output='pvel'*, *overtone=0*, *phase=None*)

Return the arrival time in seconds of a surface-wave phase or mode type/overtone. When phase is queries, mode_type/overtone/arc are ignored

table <str>: path to hdf5 format modes table distance_in_degree <float>: great circle distance, can be minor or major arc period <float>: period in second mode_type <str>: either “spheroidal”, “toroidal”, or “radial” overtone <int>: overtone number (defaults to 0, ie. fundamental mode)

avni.data.SW.**get_velocity**(*table*, *period*, *overtone*, *mode_type*, *output='pvel'*)

Return the arrival time in seconds of a surface-wave phase or mode type/overtone. When phase is queries, mode_type/overtone/arc are ignored

table <str>: path to hdf5 format modes table distance_in_degree <float>: great circle distance, can be minor or major arc period <float>: period in second mode_type <str>: either “spheroidal”, “toroidal”, or “radial” overtone <int>: overtone number (defaults to 0, ie. fundamental mode)

avni.data.SW.**get_dispersion_curve**(*table*, *mode_type*, *output='pvel'*, *overtone=0*, *freq_units='mhz'*)

return a dispersion curve for

params: table <str>: path to hdf5 format modes table mode_type <str>: either “spheroidal”, “toroidal”, or “radial” output <str>: either “gvel” or “pvel” for group or phase velocity overtone <int>: overtone number (defaults to 0, ie. fundamental mode) freq_units <str>: units of frequency axis. either “rad/s”,”hz”,or “mhz”

returns: freq,vel: frequency (in freq_units) and (group or phase) velocity in km/s

avni.data.SW.**readSWascii**(*file*, *delim='-'*, *required=None*, *warning=False*)

Reads the AVNI format for analysis and plotting.

file : input file in default AVNI format

delim [delimiter that combines fields into a joint field e.g. network-station] seperate out during I/O.

required [fields needed as comments (e.g. #CITE: Author et al., YEAR) in the file]

defaults: ‘CITE’, ‘SHORTCITE’, ‘REFERENCE MODEL’, ‘PVEL’, ‘CRUST’, ‘MODEL3D’, ‘SIGMATYPE’, ‘WEITYPE’,’EQTYPE’, ‘STATTYPE’, ‘FORMAT’, ‘WRITE’, ‘FIELDS’

SWdata : dictionary with fields data, metadata and comments

avni.data.SW.**writeSWascii**(*SWdata*, *filename*, *iflagthreshold=None*, *delim='-'*, *writeheader=True*, *writedata=True*, *verbose=True*)

Writes the AVNI format for analysis and plotting

SWdata : dictionary of SW data with metadata and data fields

filename : output file name

delim [delimiter that combines fields into a joint field e.g. network-station] seperate out during I/O.

iflagthreshold [threshold for iflag which corresponds to the processing level] that was cleared

avni.data.SW.**SWasciitohdf5**(*files*, *hdffile='Summary.SW.data.h5'*, *datatype='summary'*, *delim='-'*)

Read a list of files to hdf5 container format.

files: a panda list of ascii files to append

hdffile: output hdf5 file to append fields to

datatype: type of data to group the data with

delim [delimiter that combines fields into a joint field e.g. network-station] seperate out during I/O.

avni.data.SW.**SWhdf5toascii**(*query='0/25.0/L1/REM3D'*, *hdffile='Summary.SW.data.h5'*, *iflag=0*, *datatype='summary'*, *delim='-'*, *outfile=None*, *model3d=None*, *refmodel=None*, *crust=None*, *weitype=None*, *sigmatype=None*, *stattype=None*, *eqtype=None*)

write hdf field to a file. None is the default i.e. the values in metadata

query: key query to the hdf5 file till the group level e.g. 0/25.0/L1/GDM52

hdffile: output hdf5 file to append fields to

datatype: type of data to group the data with e.g. raw, processed, summary

delim [delimiter that combines fields into a joint field e.g. network-station] seperate out during I/O.

iflag : select the flag to sub-select data

outfile : output ascii file. If None, decided based on query

model3d, refmodel,crust,weitype : Fields used to query various groups and attributes sigmatype,stattype ,eqtype : If None, use the values in the hdffile metadata

ASCII file with the values selected

avni.data.SW.**readSWhdf5**(*query='0/25.0/L1/REM3D'*, *hdffile='Summary.SW.data.h5'*, *iflag=0*, *datatype='summary'*, *delim='-'*, *model3d=None*, *refmodel=None*, *crust=None*, *weitype=None*, *sigmatype=None*, *stattype=None*, *eqtype=None*)

Read a list of files to hdf5 container format.

query: key query to the hdf5 file till the group level e.g. 0/25.0/L1/GDM52

hdffile: output hdf5 file to append fields to

datatype: type of data to group the data with e.g. raw, processed, summary

delim [delimiter that combines fields into a joint field e.g. network-station] seperate out during I/O.

iflag : select the flag to sub-select data

model3d, refmodel,crust,weitype : Fields used to query various groups and attributes sigmatype,stattype ,eqtype : If None, use the values in the hdffile metadata

SWdata [dictionary of SW data with metadata and data fields consistent with] readSWascii and writeSWascii

avni.data.TT module

`avni.data.TT.readTTascii(`*file*, *delim='-'*, *required=None*, *warning=False*`)`

Reads the AVNI format for analysis and plotting.

file : input file in default AVNI format

delim [delimiter that combines fields into a joint field e.g. network-station] seperate out during I/O.

required [fields needed as comments (e.g. #CITE: Author et al., YEAR) in the file]

defaults: 'CITE', 'SHORTCITE', 'REFERENCE MODEL', 'PVEL', 'CRUST', 'MODEL3D', 'SIGMATYPE', 'WEITYPE','EQTYPE', 'STATTYPE', 'FORMAT', 'WRITE', 'FIELDS'

TTdata : dictionary with fields data, metadata and comments

`avni.data.TT.writeTTascii(`*TTdata*, *filename*, *iflagthreshold=None*, *delim='-'*`)`

Writes the AVNI format for analysis and plotting

TTdata : dictionary of SW data with metadata and data fields

filename : output file name

delim [delimiter that combines fields into a joint field e.g. network-station] seperate out during I/O.

iflagthreshold [threshold for iflag which corresponds to the processing level] that was cleared

`avni.data.TT.writetablehdf5(`*cagc_rays_outdir*, *tt_table_name*, *source_depth*, *component*, *verbose=True*`)`

writes an hdf5 file based on the output of CAGCRAYS

params: cagc_rays <str>: path to output directory of CAGCRAYS tt_table_name <str>: name of hdf5 file to create or add to source_depth <float>: source depth in km component <str>: 'PSV', or 'SH' verbose <bool>: print extra output, helpful for debugging

`avni.data.TT.get_travel_times1D(`*table*, *distance_in_degree*, *source_depth_in_km*, *phase*, *component='PSV'*, *branch=None*`)`

Get 1D travel travel times from a lookup table

params: table <str>: path to hdf5 travel time table distance_in_degree <float>: source/receiver distance in degrees source_depth_in_km <float>: source depth in km phase

<str>: seismic phase component <str>: 'PSV' or 'SH'. defaults to 'PSV'(for now) branch <str>: branch (doesn't do anything yet)

returns time_in_s <float>: travel time in seconds.

avni.data.common module

This script/module contains routines that are used to analyze data and files that contain them.

`avni.data.common.creation_date(`*path_to_file: str*`)`

Try to get the date that a file was created, falling back to when it was last modified if that isn't possible.

Parameters

path_to_file [str] full path to a file

Returns

datetime datetime stamp in UTC as AVNI server stores datetime in UTC

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2020.01.06 11.00

`avni.data.common.update_file(`*file*, *folder=None*, *baseurl=None*, *subdirectory=None*`)`

If the AVNI server contain a downloadable resource that is newer, download it locally.

Parameters

file [str] full path and name of the file to sync with AVNI servers

folder [str] folder where local files are store, by default as output from tools.get_filedir()

baseurl: str public URL from where the public downloads can take place, by default as specified as *downloadpage* in *constants.py*

subdirectory: str subdirectory inside the baseurl where the file should be synced

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.01.06 11.00

5.3 avni.models package

5.3.1 Submodules

avni.models.common module

This script/module contains routines that are used to Earth models

avni.models.common.**readepixfile**(*filename: str*) → Tuple[numpy.ndarray, dict, list]

Read a local file in extended pixel (.epix) format.

Parameters

filename [str] Name of the file containing four columns: (*latitude*, *longitude*, *pixel_size*, *value*)

Returns

tp.Tuple[np.ndarray, dict, list] First element is an array containing (*latitude*, *longitude*, *pixel_size*, *value*).

Second element are metadata from input fields if specified.

Third element are all other comments except lines containing metadata.

Raises

IOError File not found in local directory

Notes

This function assumes cell-centered values within a pixel of size *pixel_size*. In order to interpolate in arbitrary locations, this assumption needs to be used.

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

Examples

```
>>> epixarr,metadata,comments = readepixfile('file.epix')
```

avni.models.common.**writeepixfile**(*filename: str*, *epixarr: numpy.ndarray*, *metadata: dict = {'BASIS': 'PIX', 'FORMAT': '50'}*, *comments: Optional[list] = None*) → None

Write named numpy array to extended pixel format (.epix) file.

Parameters

filename [str] Name of the file containing four columns: (*latitude*, *longitude*, *pixel_size*, *value*)

epixarr [np.ndarray] array containing (*latitude*, *longitude*, *pixel_size*, *value*)

metadata [dict, optional] metadata from input fields if specified, by default {'BASIS':'PIX','FORMAT':'50'}

comments [list, optional] all other comments except lines containing metadata, by default None

Raises

IOError File cannot be written in local directory

Notes

This function assumes cell-centered values within a pixel of size *pixel_size*. In order to interpolate in arbitrary locations, this assumption needs to be used.

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

Examples

```
>>> writeepixfile('file.epix',epixarr,metadata,comments)
```

`avni.models.common.read3dmodelfile`(*modelfile: str*) → dict

Reads a standard 3D model file into a *model3d* dictionary containing *data* and *metadata*. The *data* field contains the basis coefficients of the parameterization employed in the 3D model file. The *metadata* fields contain various parameters relevant for evaluating the 3D model at various locations.

Parameters

modelfile [str] Text file describing the 3D model

Returns

dict A *model3d* dictionary containing *data* and *metadata*

Raises

IOError File not found in local directory

ValueError Parameterization type has not been implemented yet. Current ones include spherical harmonics, spherical splines and pixels.

Notes

This function has several fields in the *data* and *matadata* fields. The details depend on what kind of parameterization is employed. In general, *maxkern* is the maximum number of radial kernels and *maxcoeff* is the maximum number of corresponding lateral basis functions, *resolution* and *realization* are the indices for the resolution level and the realization from a model ensemble (usually 0 if it is a single file).

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

Examples

```
>>> model3d = read3dmodelfile('S362ANI+M')
```

avni.models.common.**epix2xarray**(*model_dir: str = '.', setup_file: str = 'setup.cfg', output_dir: str = '.', n_hpar: int = 1, buffer: bool = True*) → xarray.core.dataset.Dataset

Convert a set of extended pixel format (.epix) files in a directory to a netCDF4 file in AVNI format using xarray.

Parameters

model_dir [str, optional] path to directory containing epix layer files, by default '.'

setup_file [str, optional] setup file containing metadata for the model, by default 'setup.cfg'

output_dir [str, optional] path to output directory, by default '.'

n_hpar [int, optional] number of unique horizontal parameterizations, by default 1

buffer [bool, optional] write to buffer instead of file for the intermediate step of the ascii file, by default True

Returns

xr.Dataset An xarray Dataset that stores the values from a 3D model.

Raises

IOError File not found in local directory

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.models.common.**checksetup**(*parser*)

Check the arguments in the model setup file and populate the default options

Parameters

parser [A ConfigObj object]

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.02.16 5.00

`avni.models.common.epix2ascii(`*model_dir: str = '.'*, *setup_file: str = 'setup.cfg'*, *output_dir: str = '.'*, *n_hpar: int = 1*, *checks: bool = True*, *buffer: bool = True*, *onlyheaders: bool = False*`)`

Write AVNI formatted ASCII file from a directory containing extended pixel format (.epix) files

Parameters

model_dir [str, optional] path to directory containing epix layer files, by default '.'

setup_file [str, optional] setup file containing metadata for the model, by default 'setup.cfg'

output_dir [str, optional] path to output directory, by default '.'

n_hpar [int, optional] number of horizontal parameterizations (currently only handles models with 1 horizontal parameterization, and with a constant pixel width), by default 1

checks [bool, optional] checks if the metadata in *setup_file* is consistent with epix files, by default True

buffer [bool, optional] write to buffer instead of file for the intermediate step of the ascii file, by default True

onlyheaders [bool, optional] write only headers, not the coefficients, by default False

Returns

buffer or file name The memory buffer containing the file output (buffer=True) or the output file name on disk.

Raises

IOError Some epix files have not been found in the directory

AssertionError Checks for compatibility across epix files are not satisfied

ValueError Invalid values are found for some metadata fields

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.models.common.ascii2xarray(`*asciioutput*, *outfile=None*, *model_dir: str = '.'*, *setup_file: str = 'setup.cfg'*, *complevel: int = 9*, *engine: str = 'netcdf4'*, *writenc4: bool = False*`)`

Create an xarray Dataset from AVNI formatted ASCII file

Parameters

asciioutput [buffer or str] A filename string or buffer that will be read

outfile [str, optional] Output file in netCDF4 format, by default None

model_dir [str, optional] _description_, by default '.'

setup_file [str, optional] setup file containing metadata for the model, by default 'setup.cfg'

complevel [int, optional] options for compression in netcdf file, by default 9

engine [str, optional] options in netcdf file, by default 'netcdf4'

writenc4 [bool, optional] write a netcdf4 file, by default False

Returns

ds: xarray Dataset Contains the model description in a new xarray Dataset

Raises

IOError Configuration file has not been found in the directory

AssertionError Checks for compatibility across files are not satisfied

ValueError Only pixels are allowed as parameterization for netCDF files

warnings.warn Checks for number of pixels not satisfied

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.models.common.getLU2symmetric(`*insparse*`)`

Get the full symmetric matrix from LU matrix

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.models.common.readResCov(`*infile: str*, *onlymetadata: bool = False*`)`

Reads Resolution or Covariance matrix created by invwdata_pm64 with option -r. R=inv(ATA+DTD)ATA and the name of file is typically outmodel.Resolution.bin

Parameters

infile [str] Binary file containing Resolution or Covariance matrix

onlymetadata [bool, optional] Only read the metadata and ignore the elements of the matrix, by default False

Returns

refmdl [str] Reference 1D model

kerstr [str] Kernel signifying the basis sets comprising radial and horizontal parameterization

ntot [int] Number of parameters or basis coefficients in the model

indexrad1,indexrad2,indexhor1,indexhor2 [np.ndarray] Arrays describing the radial and horizontal basis sets that each ATA element in *out* correponds to

out [np.ndarray] Elements of the sparse Resolution or Covariance matrix

Raises

IOError Input file has not been found in the directory

ValueError Number of bytes in binary file does not match expected

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.models.kernel_set module

This script/module contains routines that are used to analyze/visualize the data sets in the standard AVNI format.

class avni.models.kernel_set.**Kernel_set**(*dictionary*)

Bases: object

A class for kernel sets that define the G matrix for relating data d to model m, d=Gm

initialize(*dictionary*, *required=None*, *optional=None*)

property name

property keys

property scaling

get_attributes(*dictionary*, *parserfile='/home/pm5113/Github/avni-private/avni/config/attributes.ini'*)

Reads configuration file and get the basis attributes like knot depth for each parameter in modelarray list of coefficients. parser is the output from parser = ConfigObj(config) where config is the configuration *.ini file.

extract_lateral(*dictionary*)

extract_radial(*dictionary*)

find_radial(*parameter*)

find radial indices of a given physical parameter

search_radial(*search*, *unique=False*)

evaluate_bases(*parameter*, *latitude=None*, *longitude=None*, *depth_in_km=None*)

depth_in_km [depth in km where the projection matrix is needed.] If None, returns the projection matrix for the lat/lon and radial basis as a dirac delta.

pixeldepths(*parameter*)

avni.models.lateral_basis module

This script/module contains routines that are used to analyze/visualize the data sets in the standard AVNI format.

class avni.models.lateral_basis.**Lateral_basis**(*name*, *types*, *metadata=None*)

Bases: object

A class for radial bases that defines a unique combination of parameters, their radial parameterization and any scaling that is used.

property type

property name

property keys

check()

Checks that object contains all attributes required for evaluating a particular basis set.

addtypes(*names*, *types*)

types = ['ylm','sh','wavelet','slepians']

eval_lateral(*lat*, *lon*, *store=False*)

Evaluate radial basis at a depth interpolated from existing projection matrices.

avni.models.model3d module

This script/module contains routines that are used to analyze/visualize the data sets in the standard AVNI format.

class avni.models.model3d.**Model3D**(*file=None*, ***kwargs*)

Bases: object

A class for 3D reference Earth models used in tomography

property name

property num_resolutions

property num_realizations

read(*file*, ***kwargs*)

plot()

add_realization(*coef=None*, *name=None*, *resolution=None*)

Added a set of realizations to the object. resolution is the tesselation level at which the instance is update with name and coeff (default: None, last resolution).

add_resolution(*metadata=None*)

Added a resolution level to the object. num_realizations is the number of realizations of model coefficients within that object.

readhdf5(*hf*, *query=None*)

Reads a standard 3D model file from a hdf5 file

query [if None, use the model available if only one is included.] Choose from query hf.keys() if multiple ones are available

hf: hdf5 handle from h5py

writerealization(*resolution=0*, *realization=0*, *outfile=None*)

writehdf5(*outfile=None*, *overwrite=False*)

Writes the model object to hdf5 file

buildtree3D(*resolution=0*, *dbs_path='/home/pm5113/Github/avni-private/avni/files'*)

Build a KDtree interpolant based on the metadata

ifwithinregion(*latitude*, *longitude*, *depth_in_km=None*, *resolution=0*)

average_in_polygon(*depth_in_km*, *parameter*, *polygon_latitude*, *polygon_longitude*, *num_cores=1*, *orientation='anti-clockwise'*, *threshold=1e-06*, *grid=1.0*, *outside=False*, *mask=None*, ***kwargs*)

Get the average values within a polygon

parameter: variable whose value will be returned

depth_in_km: depth where values are being queried at (km)

polygon_latitude,polygon_longitude: closed points that define the polygon. First and last points need to be the same.

grid: fineness of the grid to use for averaging (in degrees)

num_cores: Number of cores to use for the calculations

orientation: clockwise or anti-clockwise orientation of points specified above

threshold: limit to which the sum of azimuth check to (-)360 degrees is permitted to be defined as within the polygon.

outside: give average values outside polygon

average: average value within the polygon of interest

tree: if kwarg argument to evaluate_at_location has interpolated=True and tree==None, returns the tree data structure.

check_unit(*units*, *parameter*, *resolution*)

evaluate_unit(*parameter*, *values*, *units*, *depth_in_km=None*, *add_reference=True*, *resolution=0*)

get_reference(*parameter*, *depth_in_km*, *resolution=0*, *interpolation='linear'*, *dbs_path='/home/pm5113/Github/avni-private/avni/files'*)

evaluate_slice(*parameter*, *data=None*, *grid=10.0*, *depth_in_km=None*, ***kwargs*)

checks whether the data input is a DataArray and the coordinates are compatible

Parameters

parameter: 2D or 3D parameter

data [xarray Dataset or a dictionary with values]

grid [size of pixel in degrees, ignore if data is Dataset]

depth_in_km: list of depths in km for a 3D parameter

Returns

data: new or updated xarray Dataset

evaluate_at_location(*parameter*, *latitude*, *longitude*, *depth_in_km=None*, *resolution=0*, *realization=0*, *grid=False*, *interpolated=False*, *tree=None*, *nearest=None*, *units=None*, *add_reference=True*, *dbs_path='/home/pm5113/Github/avni-private/avni/files'*)

Evaluate the mode at a location (latitude, longitude,depth)

parameter: variable whose value will be returned

depth_in_km: depth where values are being queried at (km)

interpolated: If True, use KDTree from a predefined grid. If False, evaluated exactly using kernel_set instance.

nearest: number of points to interpolate. If None, defaults to values based on self[resolution][‘interpolant’]

grid: make a grid by unraveling (depth_in_km,latitude,longitude)

units: if ‘absolute’ try converting to absolute units of the parameter. If ‘default’, give the default units from model instance. If None, simply return the sparse array

add_reference: add the reference paramter value to perturbation, only valid if units queried is ‘absolute’

getpixeldepths(*resolution*, *parameter*)

coeff2modelarr(*resolution=0*, *realization=0*, *parameter=None*)

Convert the coeffiecient matrix from the file to a sparse model array. Add up all the resolutions if a list is provided.

realization : index of the set of coefficients in an ensemble. Default is 0 as there is only one set of coefficients when read from a model file.

resolution : list of resolutions to include the the modelarray

parameter: parameters to select. Default is to include all available.

readprojbinary(*lateral_basis*)

Reads Projection matrix created by plot_3dmod_pm. lateral_basis can be M362 or pixel1

get_projection(*parameter*, *latitude*, *longitude*, *depth_in_km=None*, *resolution=0*, *grid=False*)

Get the projection matrix from a lateral basis to another and for particular depths

depth_in_km [depth in km where the projection matrix is needed.] If None, returns the projection matrix for the lat/lon and radial basis as a dirac delta.

grid: make a grid by repeating (latitude,longitude) by number of depth_in_km

projslices(*projection*, *variable*, *depth*, *resolution=0*, *realization=0*)

Projection matrix multiplied by model ensemble. Choses the nearest depth available for the projection.

checktree3D(*tree*, *parameter*, *resolution=0*)

reparameterize(*model3d*, *resolution=0*, *realization=0*, *interpolated=False*, *tree=None*, *nearest=1*, *dbs_path='/home/pm5113/Github/avni-private/avni/files'*)

Inverts for new coefficients in self.data from the coefficients in model3d class

write : output the reparamterized model as a text file

interpolated: assume that it won't be a simple interpolation

to_profiles(*grid=10.0*, *type='pixel'*, *resolution=0*, *realization=0*, *model_dir='.'*, *interpolated=True*)

converts a model3d class to profiles class

grid: either a grid size of a grid file

interpolant: interpolant type either in pixel or nearest

model_dir: directory to find reference 1D model

get_resolution(*rescovfile=None*, *LU2symmetric=True*, *resolution=0*, *realization=0*)

Reads Resolution or Covariance matrix created by invwdata_pm64 with option -r. R=inv(ATA+DTD)ATA and the name of file is typically outmodel.Resolution.bin First read the matrix and model file, perform checks and create a sparse matrix.

printsplinefiles()

Prints out the splines knots into a file.

Parameters

model3d [The model object from read3dmodelfile]

avni.models.profiles module

This script/module contains routines that are used to analyze/visualize the data sets in the standard AVNI format.

class avni.models.profiles.**Profiles**(*file=None*, ***kwargs*)

Bases: object

A class for the profiles from a laterally heteroegeneous model

property name

property interpolant

read(*file*, ***kwargs*)

read_setup(*file='setup.cfg'*)

Try reading a folder containing ascii files for every location on the surface

setup_file: setup file containing metadata for the model

writehdf5(*outfile=None*, *overwrite=False*)

writes profile class to an hdf5 container

readhdf5(*hf*, *interpolant=None*, *only_metadata=False*)

Reads a standard 3D model file from a hdf5 file

interpolant: group to load

only_metadata: do not return the pandas dataframe if True

find_index(*latitude*, *longitude*)

finds the nearest point in self.data['index']

evaluate_at_location(*latitude*, *longitude*, *depth_in_km*, *parameter*, ***kwargs*)

evaluate the profiles at a particular point within the domain

get_profile(*index*, *parameters*)

Use evaluate_at_location at depths defined in reference1D

avni.models.radial_basis module

This script/module contains routines that are used to analyze/visualize the data sets in the standard AVNI format.

class avni.models.radial_basis.**Radial_basis**(*name*, *types*, *metadata=None*)

Bases: object

A class for radial bases that defines a unique combination of parameters, their radial parameterization and any scaling that is used.

Parameters

object [_type_] object.data - Contains the following fields that describe the radial basis. depths_in_km: depth array in km vercof: value of the bases evaluated at those depths dvercof: gradient of the bases evaluated at those depths object.metadata: Contains metadata for various calculations. name: to store a name for the radial_basis type: type of radial basis e.g. vbspl attributes: a dictionary containing variables used to define this particular type

e.g. knots for vbspl. Checked that these are defined using self.check.

property type

property name

property keys

add_attribute(*key*, *value*)

Add attributes needed by the radial basis

key: string key name

value: values corresponding to the key

check()

Checks that object contains all attributes required for evaluating a particular basis set.

eval_radial(*depths_in_km*, *store=False*)

Evaluates the radial bases at various depths.

depths_in_km: depths where the radial parameteriation needs to be evaluated.

store: store them in data

avni.models.realization module

This script/module contains routines that are used to analyze/visualize the data sets in the standard AVNI format.

class avni.models.realization.Realization(*file=None*)

Bases: `object`

A class for the realization of a 3D model

property type

property name

property refmodel

property keys

read(*file*)

Try reading the file into resolution/realization either as ascii, hdf5 or nc4

decode_units()

decode_symbols(*searchstr*, *unique=False*)

scale_coefficients()

scale model coefficients based on scaling

decode_scaling()

decode_mapping()

readascii(*modelfile*)

Reads a standard 3D model file. maxkern is the maximum number of radial kernels and maxcoeff is the maximum number of corresponding lateral basis functions. resolution and realization are the indices for the resolution level and the realization from a model ensemble (usually 0 if a single file)

readnc4(*ds*)

Read netCDF4 file into a resolution and realization of model3D class.

to_xarray(*outfile=None*, *complevel=9*, *engine='netcdf4'*, *writenc4=False*)

write an xarrary dataset from a avni formatted ascii file

ascii_file: path to avni format output file

outfile: output netcdf file

setup_file: setup file containing metadata for the model

complevel, engine: options for compression in netcdf file

writenc4: write a netcdf4 file, if True

to_harmonics(*lmax=40*, *variables=None*)

avni.models.reference1d module

This script/module contains routines that are used to analyze/visualize the data sets in the standard AVNI format.

class avni.models.reference1d.**Reference1D**(*file=None*)

Bases: object

A class for 1D reference Earth models used in tomography

property name

derive()

read(*file*)

Read a card deck file used in OBANI. Other formats not ready yet

plot()

read_bases_coefficients(*file*)

coefficients_to_cards(*base_units=True*)

evaluates bases coefficients at prescribed depth levels to get a card deck file

base_units: convert from native units to base units in constants

write_mineos_cards(*file*)

read_mineos_cards(*file*, *header=3*)

get_Love_elastic()

Get the Love parameters and Voigt averaged elastic properties with depth

A,C,N,L,F: anisotropy elastic Love parameters

kappa: bulk modulus

mu: shear modulus

vphi: bulk sound velocity

xi: shear anisotropy ratio

phi: P anisotropy ratio

Zs, Zp: S and P impedances

get_mineralogical()

Get the Love parameters and Voigt averaged elastic properties with depth

gravity: gavity at each depth

Brunt-Vaisala Frequency: Used for Bullen's parameter

Bullen: Bullen's parameter

pressure: pressure at each depth

poisson: Poisson's ratio

if_discontinuity(*depth_in_km*)

Returns whether a depth is a discontinuity.

get_discontinuity()

Get values, average values and contrasts at discontinuities

Returns a structure self.metadata['disc'] that has three arrays:

delta: containing absolute difference in parameters between smaller/larger radii

average: containing absolute average parameters between smaller/larger radii

contrasts: containing contrast in parameters (in %)

get_custom_parameter(*parameters*)

Get the arrays of custom parameters defined in various Earth models

evaluate_at_depth(*depth_in_km*, *parameter='vsh'*, *boundary='+'*, *interpolation='linear'*)

Get the values of a parameter at a given depth

boundary: + for value at larger radius at a discontinuity

to_mineoscards(*directory='.'*, *fmt='cards'*)

Writes a model file that is compatible with MINEOS.

to_TauPmodel(*directory='.'*, *fmt='tvel'*)

Writes a model file that is compatible with TauP. file format options 'tvel' and 'nd'.

Note: TauP can't handle zero shear velocity in the ocean layer... To work around this, zero values an ocean layer will be written as 1e-4.

to_axisem(*directory='.'*, *anelastic=True*, *anisotropic=True*, *fmt='bm'*)

Write 1D model to be used as an external model in axisem

5.4 avni.mapping package

5.4.1 Submodules

avni.mapping.common module

avni.mapping.common.**interp_weights**(*xyz*, *uvw*, *d: int = 3*)

Get weights for interpolation

First, a call to `scipy.spatial.qhull.Delaunay()` is made to triangulate the irregular grid coordinates. Second, for each point in the new grid, the triangulation is searched to find in which triangle (actually, in which simplex, which in your 3D case will be in which tetrahedron) does it lay. Third, barycentric coordinates of each new grid point with respect to the vertices of the enclosing simplex are computed.

From: http://stackoverflow.com/questions/20915502/speedup-scipy-griddata-for-multiple-interpolations-be

Parameters

xyz Locations of an irregular grid

uvw Locations to find

d [int, optional] Some integer that probably denotes dimensions, by default 3

Returns

vertices, weights Vertices of the simplex and weights to use

avni.mapping.common.**interpolate**(*values*, *vtx*, *wts*, *fill_value=nan*)

Interpolate based on values, faster form of `scipy.interpolate.griddata()`

An interpolated values is computed for that grid point, using the barycentric coordinates, and the values of the function at the vertices of the enclosing simplex.

From: http://stackoverflow.com/questions/20915502/speedup-scipy-griddata-for-multiple-interpolations-be

Parameters

values _description_

vtx [_type_] _description_

wts [_type_] _description_

fill_value [_type_, optional] _description_, by default np.nan

Returns

__type__ _description_

avni.mapping.ellipsoidal module

avni.mapping.ellipsoidal.**get_distaz**(*eplat: Union[float, list, tuple, numpy.ndarray]*, *eplon: Union[float, list, tuple, numpy.ndarray]*, *stlat: Union[float, list, tuple, numpy.ndarray]*, *stlon: Union[float, list, tuple, numpy.ndarray]*, *num_cores: int = 1*)

Get the distance and azimuths between pairs of positions in geographic coordinates

This function first converts the queried station and source locations to geocentric coordinates. In practice, we projects all points from an ellipsoid of flatness (f) to a sphere of equatorial radius (a_e) though the geocentric conversion factor (W or geoco below). This is also called the parametric or reduced latitude conversion, introduced by Legendre and Bessel who solved problems for geodesics on the ellipsoid by transforming them to an equivalent problem for spherical geodesics by using this smaller geocentric latitude.

The spherical law of cosines formula is then used to calculate distances on this spherical geodesic of radius a_e. This procedure stretches the angular distances between adjacent geographic latitudes nearer to the poles and equator, which imitates the behavior in an ellipsoid where adjacent latitudes become finely spaced. The equatorial radius is used instead of mean radius (R) for conversion of distances to km since this conversion is strictly valid for a sphere of radius a_e and in order to obtain accurate distances near the equator.

Parameters

eplat [tp.Union[float,list,tuple,np.ndarray]] Latitudes of source location(s) in Geographic Coordinates

eplon [tp.Union[float,list,tuple,np.ndarray]] Longitudes of source location(s) in Geographic Coordinates

stlat [tp.Union[float,list,tuple,np.ndarray]] Latitudes of station location(s) in Geographic Coordinates

stlon [tp.Union[float,list,tuple,np.ndarray]] Longitudes of station location(s) in Geographic Coordinates

num_cores [int, optional] Number of cores to use in the calculation, by default 1

Returns

delta,azep,azst A tuple with elements as distance in degrees, azimuth from source(s), and backazimuth from station(s).

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.mapping.ellipsoidal.**delazgc_helper**(*args*)

A helper function to parallelize ddelazgc

avni.mapping.ellipsoidal.**geographic_to_geocentric**(*latin: float*) → float

Convert a geographic latitude to geocentric latitude

Parameters

latin [float] Input latitude in geographic coordinate

Returns

float Output geocentric latitude

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.mapping.ellipsoidal.**geocentric_to_geographic**(*latin: float*) → float

Convert a geocentric coordinate to geographic coordinate

Parameters

latin [float] Input latitude in geocentric coordinate

Returns

float Output geographic latitude

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.mapping.ellipsoidal.**inpolygon**(*latitude: Union[float, list, tuple, numpy.ndarray]*, *longitude: Union[float, list, tuple, numpy.ndarray]*, *polygon_latitude: Union[list, tuple, numpy.ndarray]*, *polygon_longitude: Union[list, tuple, numpy.ndarray]*, *num_cores: int = 1*, *orientation: str = 'anti-clockwise'*, *threshold: float = 1e-06*) → Union[numpy.ndarray, bool]

Finds whether a (set of) point(s) is(are) within a closed polygon

Parameters

latitude,longitude [tp.Union[float,list,tuple,np.ndarray]] Set of queried locations in Geographic Coordinates

polygon_latitude,polygon_longitude [tp.Union[list,tuple,np.ndarray]] Closed points that define the polygon. First and last points need to be the same.

num_cores [int, optional] Number of cores to use for the calculations, by default 1

orientation [str, optional] clockwise or anti-clockwise orientation of points specified above, by default 'anti-clockwise'

threshold [float, optional] Limit to which the sum of azimuth check to (-)360 degrees is permitted to be defined as within the polygon, by default 1E-6

Returns

tp.Union[np.ndarray,bool] Logical array of bool(s) containing the same number of elements as latitude/longitude

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.mapping.geodesy module

`avni.mapping.geodesy.getplanetconstants(`*planet: Union[None, str] = None*, *configfile: Union[None, str] = None*, *option=None*`)`

Load the astronomic-geodetic constraints for a planet from a configuration file.

Parameters

planet [tp.Union[None,str], optional] _description_, by default None so `constants.planetpreferred()`

configfile [tp.Union[None,str], optional] all the planet configurations are in this file., by default None so read from so `get_configdir/constants.planetpreferred()`

option GRS option for constants, by default None so use the default one

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.02.16 5.00

`avni.mapping.geodesy.evaluate_grs(`*GM: Union[None, float] = None*, *f: Union[None, float] = None*, *a_e: Union[None, float] = None*, *omega: Union[None, float] = None*, *R: Union[None, float] = None*, *nzo: int = 10*, *store: bool = False*`)`

Calculate geopotential constants in a reference earth model.

All the following page numbers and equation numbers refer to the book Physical Geodesy by Hofmann-wellenhof and Moritz [HWM06]

Parameters

GM [tp.Union[None,float], optional] Gravitational constant times mass reference, by default None

f [tp.Union[None,float], optional] Flattening, by default None

a_e [tp.Union[None,float], optional] Semi-major axis, by default None

omega [tp.Union[None,float], optional] Angular velocity, by default None

R [tp.Union[None,float], optional] _description_, by default None

nzo [int, optional] Number of zonal harmonics (2,4,... 2*nzo), by default 10

store [bool, optional] Store in constants or return as output if False, by default False

Returns

barC2n Normalized even zonal harmonics of the corresponding Somigliana-Pizzetti normal field. barC2n(:,1): normalized zonal harmonics barC2n(:,2): degree of the zonal harmonic [2 4 ... 2*nzo]

geqt Normal gravity at the equator

gpol Normal gravity at the pole

U0 Normal potential at the ellipsoid

m omega^2*a^2*b/(GM)

ecc First eccentricity

eccp Second eccentricity

a_p Semi-minor axis

E Linear eccentricity

c Polar radius of curvature

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.mapping.spherical module

avni.mapping.spherical.**intersection**(*path1start: Union[list, numpy.ndarray]*, *path1brngEnd: Union[float, int, list, numpy.ndarray]*, *path2start: Union[list, numpy.ndarray]*, *path2brngEnd: Union[float, int, list, numpy.ndarray]*)

Get the intersection of two great circle paths. Input can either be point and bearings or two sets of points.

If c1 & c2 are great circles through start and end points (or defined by start point + bearing), then candidate intersections are simply c1 X c2 & c2 x c1 most of the work is deciding correct

intersection point to select! If bearing is given, that determines which intersection, if both paths are defined by start/end points, take closer intersection https://www.movable-type.co.uk/scripts/latlong-vectors.html#intersection

Parameters

path1start [tp.Union[list,np.ndarray]] Location of start point of the first curve in [latitude, longitude]

path1brngEnd [tp.Union[float,int,list,np.ndarray]] End point of first curve either in terms of a bearing (float/int) or location [latitude, longitude]

path2start [tp.Union[float,int,list,np.ndarray]] Location of start point of the second curve in [latitude, longitude]

path2brngEnd [tp.Union[float,int,list,np.ndarray]] End point of second curve either in terms of a bearing (float/int) or location [latitude, longitude]

Returns

intersection Preferred intersection of the two curves based on the input

antipode Antipode of the intersection where the great-circle curves meet as well

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.mapping.spherical.midpoint`(*lat1: Union[int, float]*, *lon1: Union[int, float]*, *lat2: Union[int, float]*, *lon2: Union[int, float]*)

Get the mid-point from positions in geographic coordinates. Input values as degrees

`avni.mapping.spherical.cart2spher`(*xyz: Union[list, numpy.ndarray]*) → numpy.ndarray

Convert from cartesian to spherical coordinates.

Parameters

xyz [tp.Union[list,np.ndarray]] Data containing columns for x,y,z locations in cartesian coordinates

Returns

np.ndarray Output containing radius, latitude and longitude in spherical coordinates

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.mapping.spherical.spher2cart`(*rlatlon: Union[list, numpy.ndarray]*) → numpy.ndarray

Convert from spherical to cartesian coordinates.

Parameters

rlatlon [tp.Union[list,np.ndarray]] Data containing columns for x,y,z locations in spherical coordinates

Returns

np.ndarray Output containing x,y,z in spherical coordinates

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.mapping.spherical.polar2cart(`*rtheta: Union[list, numpy.ndarray]*`)` → numpy.ndarray

Convert from polar to cartesian coordinates

Parameters

rtheta [tp.Union[list,np.ndarray]] A single r,theta or an array of these for conversion

Returns

np.ndarray Output containing x,y in cartesian coordinates

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.mapping.spherical.cart2polar(`*xy: Union[list, numpy.ndarray]*`)` → numpy.ndarray

Convert from polar to cartesian coordinates

Parameters

xy [tp.Union[list,np.ndarray]] A single x,y or an array of these ffor conversion

Returns

np.ndarray Output containing r,theta in polar coordinates

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.mapping.spherical.getDestination(`*lat: Union[int, float]*, *lon: Union[int, float]*, *azimuth: Union[int, float]*, *distance*`)` → list

Returns the location of destination point given the start lat, long, aziuth, and distance (in meters).

Parameters

lat, lon [tp.Union[int,float]] Start point latitude and longitude

azimuth [tp.Union[int,float]] Azimuth to destination

distance Distance to destination (in meters)

Returns

list List containing latitude and longitude of destination point

avni.mapping.spherical.**calculateBearing**(*lat1: Union[int, float]*, *lon1: Union[int, float]*, *lat2: Union[int, float]*, *lon2: Union[int, float]*)

Calculates the azimuth in degrees from start point to end point.

Parameters

lat1,lon1 [tp.Union[int,float]] Start point latitude and longitude

lat2,lon2 [tp.Union[int,float]] End point latitude and longitude

Returns

bearing Bearing to second point

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.mapping.spherical.**getIntermediate**(*lat1: Union[int, float]*, *lon1: Union[int, float]*, *azimuth: Union[int, float]*, *distance: Union[int, float]*, *interval: Union[int, float]*) → list

Returns intermediate coordinates between two coordinate pairs given a desired interval

Parameters

lat1,lon1 [tp.Union[int,float]] Start point latitude and longitude

azimuth [tp.Union[int,float]] Azimuth to destination

distance Distance to destination (in meters)

interval [tp.Union[int,float]] Interval for intermediate points (in meters)

Returns

list Coordinates (lat,lon) of intermdiate points

avni.mapping.spherical.**calculateDistance**(*lat1: Union[int, float]*, *lon1: Union[int, float]*, *lat2: Union[int, float]*, *lon2: Union[int, float]*, *final_units='m'*) → float

Calculates distance between two coordinates

Parameters

lat1,lon1 [tp.Union[int,float]] Start point latitude and longitude

lat2,lon2 [tp.Union[int,float]] End point latitude and longitude

final_units [str, optional] Output distance in km/m/deg, by default 'm'

Returns

float Distance between coordinate pairs in m, km or deg

5.5 avni.plots package

5.5.1 Submodules

avni.plots.common module

This module contains the various subroutines used for plotting

`avni.plots.common.updatefont`(*fontsize: int = 15*, *fontname: str = 'sans-serif'*, *ax=None*)

Updates the font type and sizes globally or for a particular axis handle

Parameters

fontsize [int, optional] Size of font, by default 15

fontname [str, optional] Name of font, by default 'sans-serif'

ax [matplotlib.axes.Axes, optional] Axes handle, by default None

Returns

ax Updated axes handle if ax is not None

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.common.initializecolor`(*name: str*, ***kwargs*)

Initialize a color palette instance.

This can be from standard Python palettes (e.g. jet), those in `constants()` or downloadable from server.

Parameters

name [str] Name of color palette. Can have *_r* appended to standard ones for reversed color scales e.g. *jet_r*.

****kwargs** [dict] Optional arguments for Basemap

Returns

cpalette Output color palette

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.common.standardcolorpalette`(*name: str = 'avni'*)

Register a custom AVNI color palette from `constants()`

Parameters

name [str, optional] Color palette name that will be used elsewhere, by default 'avni'. If name ends in '_r', uses the reversed color scale

Returns

cpalette Output color palette

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.plots.common.**get_colors**(*val: float*, *xmin: float = - 1.0*, *xmax: float = 1.0*, *palette: str = 'coolwarm'*, *colorcontour: int = 20*) → tuple

Gets the value of color for a given palette

Parameters

val [float] Value to query

xmin [float, optional] Minimum value or the color scale, by default -1.

xmax [float, optional] Maximum value or the color scale, by default 1.

palette [str, optional] Color palette to query, by default 'coolwarm'

colorcontour [int, optional] Number of color contours to use in dividing up the color palette, by default 20

Returns

tuple Tuple of (r, g, b, a) scalars.

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.plots.common.**grayify_cmap**(*cmap*)

Return a grayscale version of the colormap

Parameters

cmap Input color palette

Returns

cpalette Output color palette

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.plots.common.**make_colormap**(*seq*, *name: str = 'CustomMap'*)

Return a LinearSegmentedColormap for a sequence of colors

Parameters

seq A sequence of floats and RGB-tuples. The floats should be increasing and in the interval (0,1).

name [str, optional] Name to give to this color palette, by default 'CustomMap'

Returns

cpalette Output color palette

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.common.getcolorlist(`*cptfile: str*, *type='avni'*`)` → list

Get a list of color tuples from a color palette (.cpt) file

Parameters

cptfile [str] A color palette file

type [str, optional] Either avni format or standard per GMT project, by default 'avni'

Returns

list A list of colors tuples (r, g, b)

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.common.readstandardcpt(`*cptfile: str*`)` → list

Read a GMT color map from a color palette (.cpt) file

Parameters

cptfile [str] color palette file

Returns

list A list of colors tuples (r, g, b)

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.common.customcolorpalette(`*name: str = 'bk'*, *cptfolder: Union[None, str] = None*, *colormax: float = 2.0*, *middlelimit: float = 0.5*, *ifgraytest: int = 0*`)`

Used to return preset color palettes from `constants.cptfolder()`

Parameters

name [str, optional] Name of the color palette, by default 'bk'

cptfolder [tp.Union[None, str], optional] Location of the color palette (.cpt) files, by default None so uses `constants.cptfolder()`

colormax [float, optional] Limits of the colorbar (-colormax,colormax), by default 2.

middlelimit [float, optional] Limit to which the middle color (e.g. grey) will extend on either side of color mid point, by default 0.5

ifgraytest [int, optional] Tests how the figure looks in gray scale, by default 0

Returns

cpalette Output color palette

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.plots.models module

`avni.plots.models.plot_gcpaths(`*m*, *stlon: Union[float, numpy.ndarray]*, *stlat: Union[float, numpy.ndarray]*, *eplon: Union[float, numpy.ndarray]*, *eplat: Union[float, numpy.ndarray]*, *ifglobal: bool = False*, ***kwargs*`)`

Plots great-circle paths from longitude and latitude arrays.

Parameters

m An instance of *mpl_toolkits.basemap* Class

stlon [tp.Union[float,np.ndarray]] Longitudes of station location(s)

stlat [tp.Union[float,np.ndarray]] Latitudes of station location(s)

eplon [tp.Union[float,np.ndarray]] Longitudes of station location(s)

eplat [tp.Union[float,np.ndarray]] Latitudes of station location(s)

ifglobal [bool, optional] Set extent to be global, by default False

****kwargs** [dict] Optional arguments for Basemap

Returns

m Updated instance of *mpl_toolkits.basemap* Class

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.plot_hotspots(`*m*, *dbs_path: Union[None, str] = None*, *lon360: bool = False*, ***kwargs*`)`

Reads hotspots.pkl from *dbs_path* and plots on to map instance

Earlier, the data was in pickle format, cross-platform compatibility required json # hotspots = pickle.load(open('%s/hotspots.pkl' % (dbs_path), 'rb')) # tools.writejson(hotspots,'%s/hotspots.json' % (dbs_path))

Parameters

m An instance of *mpl_toolkits.basemap* Class

dbs_path [tp.Union[None,str], optional] path specified by user where hotspots.json is located. If not found, defaults to downloading the file from the AVNI server, by default None

lon360 [bool, optional] False if the no longitude above 180 is permitted and is wrapped around, by default False

****kwargs** [dict] Optional arguments for Basemap

Returns

m Updated instance of *mpl_toolkits.basemap* Class

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.plot_plates`(*m*, *dbs_path: Union[None, str] = None*, *lon360: bool = False*, *boundtypes: list = ['ridge', 'transform', 'trench']*, ***kwargs*)

Plots different types of tectonic plates on to a map object

Parameters

m An instance of *mpl_toolkits.basemap* Class

dbs_path [tp.Union[None,str], optional] path specified by user where hotspots.json is located. If not found, defaults to downloading the file from the AVNI server, by default None

lon360 [bool, optional] False if the no longitude above 180 is permitted and is wrapped around, by default False

boundtypes [list, optional] Types of boundaries to plot, by default [‘ridge’, ‘transform’, ‘trench’]

****kwargs** [dict] Optional arguments for Basemap

Returns

m Updated instance of *mpl_toolkits.basemap* Class

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.plots.models.**globalmap**(*ax*, *valarray: Union[list, numpy.ndarray]*, *vmin: Union[float, int]*, *vmax: Union[float, int]*, *dbs_path: Union[None, str] = None*, *colorlabel: Union[None, str] = None*, *colorticks: bool = True*, *ticklabels: Union[None, list, numpy.ndarray] = None*, *colorpalette: str = 'avni'*, *colorcontour=21*, *hotspots: bool = False*, *grid: Union[list, numpy.ndarray] = [30.0, 90.0]*, *gridwidth: int = 0*, *shading: bool = False*, *model: Union[None, str] = None*, *resolution: str = 'l'*, *field: str = 'z'*, ***kwargs*)

Plots a 2-D cross-section of a 3D model on a predefined axis

Parameters

ax Axis handle number

valarray [tp.Union[list,np.ndarray]] a named numpy array containing latitudes (lat), longitudes (lon) and values (val). Can be initialized from three numpy arrays lat, lon and val $ data = np.vstack((lat,lon,val)).transpose() $ dt = {'names':['latitude', 'longitude', 'value'], 'formats':[float, float, float]} $ valarray = np.zeros(len(data), dtype=dt) $ valarray['latitude'] = data[:,0]; valarray['longitude'] = data[:,1]; valarray['value'] = data[:,2]

vmin,vmax [tp.Union[float,int]] Minimum and maximum value of the color scale

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

colorlabel [tp.Union[None,str], optional] Label to use for the colorbar. If None, no colorbar is plotted, by default None

colorticks [bool, optional] Label and draw the ticks in the colorbar, by default True

ticklabels [tp.Union[None,list,np.ndarray], optional] Labels for ticks on the colorbar, by default None

colorpalette [str, optional] Matplotlib color scales or the AVNI one, by default 'avni'

colorcontour [int, optional] Number of contours for colors in the plot. Maximum is 520 and odd values are preferred so that mid value is at white/yellow or other neutral colors, by default 21

hotspots [bool, optional] Plot hotspots, by default False

grid [tp.Union[list,np.ndarray], optional] Grid spacing in latitude and longitude, by default [30.,90.]

gridwidth [int, optional] Width of the grid lines, by default 0

shading [bool, optional] Shade the plot based on topography, by default False

model [tp.Union[None,str], optional] Name of the topography file in NETCDF4 format, by default None so use `constants.topography()`

resolution [str, optional] Resolution of boundary database to use in Basemap. Can be c (crude), l (low), i (intermediate), h (high), f (full), by default 'l'

field [str, optional] Field name in the NETCDF4 file to use, by default 'z'

****kwargs** [dict] Optional arguments for Basemap

Returns

m Updated instance of `mpl_toolkits.basemap()` Class

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.backgroundmap(`*ax*, *dbs_path: Union[None, str] = None*, *plates: str = 'r'*, *oceans: str = 'w'*, *continents: str = 'darkgray'*, *boundary: str = 'k'*, ***kwargs*`)`

Plots a background map of a 3D model on an axis handle.

Parameters

ax Axis handle number

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

plates [str, optional] Color of tectonic plates, by default 'r'

oceans [str, optional] Color of oceans, by default 'w'

continents [str, optional] Color of continents, by default 'darkgray'

boundary [str, optional] Color of background around the map, by default 'k'

****kwargs** [dict] Optional arguments for Basemap

Returns

m Updated instance of `mpl_toolkits.basemap()` Class

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.insetgcpathmap(`*ax*, *lat1: Union[int, float]*, *lon1: Union[int, float]*, *azimuth: Union[int, float]*, *gcdelta: Union[int, float]*, *projection: str = 'ortho'*, *width: float = 50.0*, *height: float = 50.0*, *dbs_path: Union[None, str] = None*, *numdegticks: int = 7*, *hotspots: bool = False*`)`

Plots the great-circle path based on azimuth and delta from initial location.

Takes width/heght arguments in degrees if projection is Mercator, etc.

Parameters

ax Axis handle number

lat1 [tp.Union[int,float]] Initial location latitude

lon1 [tp.Union[int,float]] Initial location longitude

azimuth [tp.Union[int,float]] Azimuth to final location

gcdelta [tp.Union[int,float]] Distance in degrees to final location

projection [str, optional] Map projection, by default 'ortho'

width [float, optional] Width of the inset map, by default 50.

height [float, optional] Height of the inset map, by default 50.

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

numdegticks [int, optional] Number of ticks along great-circle path, by default 7

hotspots [bool, optional] Plot hotspots, by default False

Returns

m Updated instance of `mpl_toolkits.basemap()` Class

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.setup_axes`(*fig*, *rect*, *theta: Union[list, numpy.ndarray]*, *radius: Union[list, numpy.ndarray]*, *numdegticks: int = 7*, *r_locs: list = [3480.0, 3871.0, 4371.0, 4871.0, 5371.0, 5871.0, 6346.6]*, *r_labels: list = ['CMB', ' ', '2000', ' ', '1000', ' ', 'Moho']*, *fontsize: int = 12*)

Setup the polar axis for a section plot

Parameters

fig A figure hand from `plt.figure()`

rect A 3-digit number for axis on a plot. Obtained as axis handle from `gridspec.GridSpec()` instance. $gs = gridspec.GridSpec(1, 2) $rect = gs[1]

theta [tp.Union[list,np.ndarray]] Range of degrees to plot

radius [tp.Union[list,np.ndarray]] Range of radius to plot

numdegticks [int, optional] Number of ticks along great-circle path, by default 7

r_locs [list, optional] Radius locations to plot as curves, by default [3480.,3871.,4371.,4871.,5371.,5871.,6346.6]

r_labels [list, optional] Labels for the radius locations, by default ['CMB',' ','2000',' ','1000',' ','Moho']

fontsize [int, optional] Tick font size, by default 12

Returns

ax1, aux_ax Axis and auxillary axis where the polar axis plot has been made

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.gettopotransect`(*lat1: Union[int, float]*, *lon1: Union[int, float]*, *azimuth: Union[int, float]*, *gcdelta: Union[int, float]*, *model: Union[None, str] = None*, *tree=None*, *dbs_path: Union[None, str] = None*, *numeval: int = 50*, *resolution: str = 'l'*, *nearest: int = 1*)

Get the topography transect based on azimuth and delta from initial location.

Parameters

lat1 [tp.Union[int,float]] Initial location latitude

lon1 [tp.Union[int,float]] Initial location longitude

azimuth [tp.Union[int,float]] Azimuth to final location

gcdelta [tp.Union[int,float]] Distance in degrees to final location

model [tp.Union[None,str], optional] Name of the topography file in NETCDF4 format, by default None so `constants.topography()`

tree A `scipy.spatial.cKDTree()` read from an earlier run, by default None

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

numeval [int, optional] Number of evaluations of topo/bathymetry along the transect, by default 50

resolution [str, optional] Resolution of boundary database to use in Basemap. Can be c (crude), l (low), i (intermediate), h (high), f (full), by default 'l'

nearest [int, optional] Number of nearest values in the KD-tree to interpolated from, by default 1 so nearest

Returns

valselect,model,tree Values along selected transect, topography model values, and a `scipy.spatial.cKDTree()`

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.plottopotransect(`*ax*, *theta_range: numpy.ndarray*, *elev*, *vexaggerate: int = 150*`)`

Plot a topographic transect on an axis

Parameters

ax Axis handle number

theta_range [np.ndarray] Range of angles

elev [_type_] Elevation from topgraphy file, usually in `sparse.csc_matrix()` format.

vexaggerate [int, optional] Vertical exxageration to make the plot visible, by default 150

Returns

ax Axis handle where the plot has been made

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.getmodeltransect(`*lat1: Union[int, float]*, *lon1: Union[int, float]*, *azimuth: Union[int, float]*, *gcdelta: Union[int, float]*, *model: str = 'S362ANI+M.BOX25km_PIX1X1.avni.nc4'*, *tree=None*, *parameter: str = 'vs'*, *radii: list = [3480.0, 6346.6]*, *dbs_path: Union[None, str] = None*, *numevalx: int = 200*, *numevalz: int = 200*, *distnearthreshold: float = 500.0*, *nearest: int = 10*`)`

Get the tomography slice from a AVNI NETCDF4 file

Parameters

lat1 [tp.Union[int,float]] Initial location latitude

lon1 [tp.Union[int,float]] Initial location longitude

azimuth [tp.Union[int,float]] Azimuth to final location

gcdelta [tp.Union[int,float]] Distance in degrees to final location

model [str, optional] Name of the tomographic model file in NETCDF4 format, by default 'S362ANI+M.BOX25km_PIX1X1.avni.nc4'

tree A `scipy.spatial.cKDTree()` read from an earlier run, by default None

parameter [str, optional] Physical parameter field in the NETCDF4 file, by default 'vs'

radii [list, optional] Range of radius to plot on the slice, by default [3480.,6346.6]

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

numevalx [int, optional] Number of model evaluations along the horizontal direction, by default 200

numevalz [int, optional] Number of model evaluations along the vertical direction, by default 200

distnearthreshold [float, optional] Threshold points that are up to a distance away [NOT IMPLEMENTED], by default 500.

nearest [int, optional] Number of nearest values in the KD-tree to interpolated from, by default 10 so averages nearest 10 points

Returns

xsec,model,tree Values along selected section, toography model values, and a `scipy.spatial.cKDTree()`

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.section`(*fig*, *lat1: Union[int, float]*, *lon1: Union[int, float]*, *azimuth: Union[int, float]*, *gcdelta: Union[int, float]*, *model: str*, *parameter: str*, *vmin: Union[float, int]*, *vmax: Union[float, int]*, *dbs_path: Union[None, str] = None*, *modeltree=None*, *colorlabel: Union[None, str] = None*, *colorpalette: str = 'avni'*, *colorcontour: int = 20*, *nelevinter: int = 100*, *radii: list = [3480.0, 6346.6]*, *vexaggerate: int = 50*, *width_ratios: list = [1, 3]*, *numevalx: int = 200*, *numevalz: int = 300*, *nearest: int = 10*, *topo: Union[None, str] = None*, *resolution: str = 'l'*, *topotree=None*, *hotspots: bool = False*, *xsec_data=None*)

Plot one section across a pair of points based on azimuth and delta from initial location.

Parameters

fig A figure hand from `plt.figure()`

lat1 [tp.Union[int,float]] Initial location latitude

lon1 [tp.Union[int,float]] Initial location longitude

azimuth [tp.Union[int,float]] Azimuth to final location

gcdelta [tp.Union[int,float]] Distance in degrees to final location

model [str] Name of the tomographic model file in NETCDF4 format e.g. 'S362ANI+M.BOX25km_PIX1X1.avni.nc4'

parameter [str] Physical parameter field in the NETCDF4 file, by default 'vs'

vmin,vmax [tp.Union[float,int]] Minimum and maximum value of the color scale

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

modeltree, optional A `scipy.spatial.cKDTree()` of tomograhic model read from an earlier run, by default None

colorlabel [tp.Union[None,str], optional] Label to use for the colorbar. If None, no colorbar is plotted, by default None

colorpalette [str, optional] Matplotlib color scales or the AVNI one, by default 'avni'

colorcontour [int, optional] Number of contours for colors in the plot. Maximum is 520 and odd values are preferred so that mid value is at white/yellow or other neutral colors, by default 20

nelevinter [int, optional] Number of evaluations of topo/bathymetry along the transect, by default 100

radii [list, optional] Range of radius to plot on the slice, by default [3480.,6346.6]

vexaggerate [int, optional] Vertical exxageration to make the plot visible, by default 50

width_ratios [list, optional] Width ratios of the great circle and section subplots, by default [1,3]

numevalx [int, optional] Number of model evaluations along the horizontal direction, by default 200

numevalz [int, optional] Number of model evaluations along the vertical direction, by default 300

nearest [int, optional] Number of nearest values in the KD-tree to interpolated from, by default 10 so averages nearest 10 points

topo [tp.Union[None,str], optional] Name of the topography file in NETCDF4 format, by default None so `constants.topography()`

resolution [str, optional] Resolution of boundary database to use in Basemap. Can be c (crude), l (low), i (intermediate), h (high), f (full), by default 'l'

topotree [_type_, optional] A `scipy.spatial.cKDTree()` of topography read from an earlier run, by default None

hotspots [bool, optional] Plot hotspots on top of the plot [NOT IMPLEMENTED], by default False

xsec_data, optional Interpolated data along a section found from an earlier run, by default None

Returns

fig,topo,topotree,model,modeltree Figure handle, topography values, tomographic model values and the corresponding `scipy.spatial.cKDTree()`

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.plots.models.plot1section`(*latitude: Union[int, float]*, *longitude: Union[int, float]*, *azimuth: Union[int, float]*, *gcdelta: Union[int, float]*, *model: str*, *parameter: str*, *vmin: Union[float, int]*, *vmax: Union[float, int]*, *figuresize: Union[list, numpy.ndarray] = [8, 4]*, *outfile: Optional[str] = None*, ***kwargs*)

Plot one section across a pair of points based on azimuth and delta from initial location..

Parameters

latitude [tp.Union[int,float]] Initial location latitude

longitude [tp.Union[int,float]] Initial location longitude

azimuth [tp.Union[int,float]] Azimuth to final location

gcdelta [tp.Union[int,float]] Distance in degrees to final location

model [str] Name of the tomographic model file in NETCDF4 format e.g. 'S362ANI+M.BOX25km_PIX1X1.avni.nc4'

parameter [str] Physical parameter field in the NETCDF4 file, by default 'vs'

vmin,vmax [tp.Union[float,int]] Minimum and maximum value of the color scale

figuresize [tp.Union[list, np.ndarray], optional] Figure size, by default [8,4]

outfile [str, optional] Output file to use in `fig.savefig()`, by default None

****kwargs** [dict] Optional arguments for Basemap

Returns

topo,topotree,model,modeltree Topography values, tomographic model values and the corresponding `scipy.spatial.cKDTree()`

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.plots.models.**plot1globalmap**(*epixarr: numpy.ndarray*, *vmin: Union[float, int]*, *vmax: Union[float, int]*, *dbs_path: Union[None, str] = None*, *colorpalette: str = 'rainbow2'*, *projection: str = 'robin'*, *colorlabel: str = 'Anomaly (%)'*, *lat_0: Union[int, float] = 0*, *lon_0: Union[int, float] = 150*, *outfile: Optional[str] = None*, *shading: bool = False*)

Plot one global map

Parameters

epixarr [np.ndarray] Array containing (*latitude*, *longitude*, *pixel_size*, *value*)

vmin,vmax [tp.Union[float,int]] Minimum and maximum value of the color scale

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

colorpalette [str, optional] Matplotlib color scales or the AVNI one, by default 'rainbow2'

projection [str, optional] Map projection, by default 'robin'

colorlabel [str, optional] Label to use for the colorbar. If None, no colorbar is plotted, by default "Anomaly (%)"

lat_0 [tp.Union[int,float], optional] Center latitude for the plot, by default 0

lon_0 [tp.Union[int,float], optional] Center longitude for the plot, by default 150

outfile [str, optional] Output file to use in `fig.savefig()`, by default None

shading [bool, optional] Shade the plot based on topography, by default False

avni.plots.models.**plot1hitmap**(*hitfile: str*, *dbs_path: Union[None, str] = None*, *projection: str = 'robin'*, *lat_0: Union[int, float] = 0*, *lon_0: Union[int, float] = 150*, *colorcontour: list = [0, 25, 100, 250, 400, 600, 800, 1000, 1500, 2500, 5000, 7500, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000]*, *colorpalette: str = 'Blues'*, *outformat: str = '.pdf'*, *ifshow: bool = True*)

Plot one hit count map

Parameters

hitfile [str] A file containg named columns - "latitude", "longitude", "value"

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

projection [str, optional] Map projection, by default 'robin'

lat_0 [tp.Union[int,float], optional] Center latitude for the plot, by default 0

lon_0 [tp.Union[int,float], optional] Center longitude for the plot, by default 150

colorcontour [list, optional] Number of contours for colors in the plot, by default [0,25,100,250,400,600,800,1000,1500,2500,5000,7500,10000,15000,20000,25000,30000,35000,4000

colorpalette [str, optional] Matplotlib color scales or the AVNI one, by default 'Blues'

outformat [str, optional] Output file format, by default '.pdf'

ifshow [bool, optional] Display the plot before writing a file, by default True

avni.plots.models.**plotreference1d**(*ref1d*, *figuresize: list = [7, 12]*, *height_ratios: list = [2, 2, 1]*, *ifshow: bool = True*, *format: str = '.eps'*, *isotropic: bool = False*, *zoomdepth: list = [0.0, 1000.0]*)

Plot the ref1d object array in a PREM like plot

Parameters

ref1d An instance of the `Reference1D` class.

figuresize [list, optional] Figure size, by default [7,12]

height_ratios [list, optional] Height ratios of the three subplots, by default [2, 2, 1]

ifshow [bool, optional] Display the plot before writing a file, by default True

format [str, optional] Output file format, by default '.eps'

isotropic [bool, optional] Whether model is isotropic so seperate curves for Vsh and Vsv, by default False

zoomdepth [list, optional] Zoom into a depth extent in km, by default [0.,1000.]

avni.plots.models.**plotmodel3d**(*model3d*, *dbs_path: Union[None, str] = None*, *x: int = 0*, *percent_or_km: str = '%'*, *colormin: Union[int, float] = -6.0*, *colormax: Union[int, float] = 6.0*, *depth: Union[None, list, numpy.ndarray] = None*, *resolution: int = 0*, *realization: int = 0*)

Plots interactively a model slice of a variable at a given depth till an invalid depth is input by the user

Parameters

model3d An instance of the `Model3D` class.

dbs_path [tp.Union[None,str], optional] Path specified by user where database containing hotpot locations, coastlines is located. If not found, defaults to downloading the files from the AVNI server, by default None so uses `tools.get_filedir()`.

x [int, optional] Index for variable to plot, by default 0

percent_or_km [str, optional] Plot in percent (relative) or km/s (absolute) [NOT IMPLEMENTED], by default '%'

colormin, colormax [tp.Union[int,float], optional] Minimum and maximum value of the color scale, by default -6 and 6.

depth [tp.Union[None,list,np.ndarray], optional] Depth to plot, by default None

resolution [int, optional] Resolution index in the `Model3D` instance, by default 0

realization [int, optional] Realization index in the `Model3D` instance, by default 0

5.6 avni.tools package

5.6.1 Submodules

avni.tools.bases module

`avni.tools.bases.eval_vbspl(`*radius: Union[list, tuple, numpy.ndarray]*, *knots: Union[str, list, tuple, float, numpy.int64, numpy.ndarray, bool]*`)`

Evaluate the cubic spline knot with second derivative as 0 at end points. In contrast to *eval_splrem()*, this function distributes the spline knots unevenly at the specified depths or radii.

Parameters

radius [tp.Union[list,tuple,np.ndarray]] A radius value or array of radii (or alternatively depths) queried in km

knots [tp.Union[str,list,tuple,float,np.int64,np.ndarray,bool]] A numpy array or list of radii (or depths) of spline knots in km

Returns

vercof, dvercof: float or np.ndarray value of the polynomial coefficients at each depth and derivative. Both arrays have size (Nradius, Nsplines).

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.bases.eval_splrem(`*radius: Union[list, tuple, numpy.ndarray]*, *radius_range: Union[list, tuple, numpy.ndarray]*, *nsplines: int*`)`

Evaluate the cubic spline knot with second derivative as 0 at end points. In contrast to *eval_vbspl()*, this function distributes the spline knots evenly within the radius range.

Parameters

radius [tp.Union[list,tuple,np.ndarray]] A radius value or array of radii (or alternatively depths) queried in km

radius_range [tp.Union[list,tuple,np.ndarray]] Limits of the radius (or depths) limits of the region

nsplines [int] number of splines within the range

Returns

vercof, dvercof: float or np.ndarray value of the polynomial coefficients at each depth and derivative. Both arrays have size (Nradius, Nsplines).

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.bases.**eval_polynomial**(*radius: Union[list, tuple, numpy.ndarray]*, *radius_range: Union[list, tuple, numpy.ndarray]*, *rnorm: float*, *types: list = ['CONSTANT', 'LINEAR']*)

Evaluate a set of polynomial functions within a range of radii.

Parameters

radius [tp.Union[list,tuple,np.ndarray]] A radius value or array of radii queried in km

radius_range [tp.Union[list,tuple,np.ndarray]] Limits of the radius limits of the region

rnorm [float] normalization for radius, usually the radius of the planet

types [list, optional] polynomial coefficients to be used for calculation. Options are : TOP,BOTTOM, CONSTANT, LINEAR, QUADRATIC, CUBIC, by default ['CONSTANT','LINEAR']

Returns

vercof, dvercof: float or np.ndarray value of the polynomial coefficients and derivative at each depth size (Nradius)

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.bases.**eval_splcon**(*latitude: Union[list, tuple, numpy.ndarray]*, *longitude: Union[list, tuple, numpy.ndarray]*, *xlaspl: numpy.ndarray*, *xlospl: numpy.ndarray*, *xraspl: numpy.ndarray*)

Evaluate spherical splines at a set of locations.

Parameters

latitude [tp.Union[list,tuple,np.ndarray]] Latitudes of locations queried

longitude [tp.Union[list,tuple,np.ndarray]] Longitudes of locations queried

xlaspl [np.ndarray] Latitude locations of splines

xlospl [np.ndarray] Longitude locations of splines

xraspl [np.ndarray] Radius of splines

Returns

horcof [np.ndarray] Value of the horizontal coefficents at each location. Size of numpy array is [len(latitude) X ncoefhor]

`avni.tools.bases.splcon(`*lat: float*, *lon: float*, *ncoefhor: int*, *xlaspl: numpy.ndarray*, *xlospl: numpy.ndarray*, *xraspl: numpy.ndarray*`)`

Evaluate spherical splines at a given location. Modified from the Fortran code splcon.f that is based on Wang and Dahlen (1995) [WD95].

Parameters

lat [float] latitude query

lon [float] longitude query

ncoefhor [int] number of splines

xlaspl [np.ndarray] spline latitudes

xlospl [np.ndarray] spline longitudes

xraspl [np.ndarray] spline radii

Returns

ncon,icon,con number of splines, spline index, and value at each location

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.bases.eval_ylm(`*latitude: Union[list, tuple, numpy.ndarray]*, *longitude: Union[list, tuple, numpy.ndarray]*, *lmaxhor: int*, *weights: Union[None, list, tuple, numpy.ndarray] = None*, *grid: bool = False*, *norm: str = 'ylm'*`)`

Evaluate spherical harmonics with a specific normalization.

Parameters

latitude [tp.Union[list,tuple,np.ndarray]] Latitudes of locations queried

longitude [tp.Union[list,tuple,np.ndarray]] Longitudes of locations queried

lmaxhor [int] Maximum spherical harmonic degree

weights [tp.Union[None,list,tuple,np.ndarray], optional] Weights to multiply the bases with i.e. coefficients, by default None

grid [bool, optional] Create a grid based on a combination of latitudes and longitudes, by default False

norm [str, optional] Spherical harmonic normalization, by default 'ylm'

Returns

horcof Value of the horizontal coefficents at each location. Size of numpy array is [len(latitude) X ((lmaxhor+1)^2)]

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.bases.eval_pixel`(*latitude: Union[list, tuple, numpy.ndarray]*, *longitude: Union[list, tuple, numpy.ndarray]*, *xlapix: Union[list, tuple, numpy.ndarray]*, *xlopix: Union[list, tuple, numpy.ndarray]*, *xsipix: Union[list, tuple, numpy.ndarray]*)

Evaluate pixel values at a set of locations.

Parameters

latitude [tp.Union[list,tuple,np.ndarray]] Latitudes of locations queried

longitude [tp.Union[list,tuple,np.ndarray]] Longitudes of locations queried

xlapix [tp.Union[list,tuple,np.ndarray]] Pixel center latitudes

xlopix [tp.Union[list,tuple,np.ndarray]] Pixel center longitudes

xsipix [tp.Union[list,tuple,np.ndarray]] Pixel sizes in degrees

Returns

horcof Value of the horizontal coefficents at each location. Size of numpy array is [len(latitude) X len(xsipix)]

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common module

avni.tools.common.**stage**(*file: str*, *overwrite: bool = False*)

Stages a file in the AVNI file directories for testing

Parameters

file [str] file name to be staged

overwrite [bool, optional] overwite any existing symlink or file, by default False

Raises

IOError File not found or a symlink already exists

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**parse_line**(*line: str*, *rx_dict: dict*)

Function used to parse line with key word from rx_dict

Parameters

line [str] Line to search

rx_dict [dict] Do a regex search against all defined regexes

Returns

key,match Result of the first matching regex, None if not found

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**ifwithindepth**(*start_depths: numpy.ndarray*, *end_depths: numpy.ndarray*, *depth_in_km: numpy.ndarray*)

Check if a set of depths are within a range of depths

Parameters

start_depths [np.ndarray] Starting depth for the range

end_depths [np.ndarray] End depth for the range

depth_in_km [np.ndarray] Depths to check for

Returns

output index of depth range that each *depth_in_km* belongs to

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**makegrid**(*latitude: Union[list, tuple, numpy.ndarray]*, *longitude: Union[list, tuple, numpy.ndarray]*, *depth_in_km: Union[None, list, tuple, numpy.ndarray] = None*)

Make a 2D or 3D grid out of input locations and depths.

Parameters

latitude [tp.Union[list,tuple,np.ndarray]] Latitudes of locations queried

longitude [tp.Union[list,tuple,np.ndarray]] Longitudes of locations queried

depth_in_km [tp.Union[None,list,tuple,np.ndarray], optional] Depths of locations queried, by default None

Returns

nrows,latitude,longitude,[optional: depth_in_km] A 2D or 3D grid with *nrows* rows found by unraveling (depth_in_km,latitude,longitude)

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**convert2nparray**(*value: Union[str, list, tuple, float, numpy.int64, numpy.ndarray, bool]*, *int2float: bool = True*, *allowstrings: bool = True*) → numpy.ndarray

Converts input value to a float numpy array. Boolean are returned as Boolean arrays.

Parameters

value [tp.Union[str,list,tuple,float,np.int64,np.ndarray,bool]] A single value or a set of values.

int2float [bool, optional] Convert integer to floats, by default True

allowstrings [bool, optional] Check if value has strings, by default True

Returns

np.ndarray Output values as a numpy array.

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**precision_and_scale**(*x: float*, *max_digits: int = 14*) → Tuple[int, int]

Returns precision and scale of a float

Parameters

x [float] A floating point number

max_digits [int, optional] Maximum digits or magnitude, by default 14

Returns

tp.Tuple[int, int] Returns precision and scale of a float

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.common.alphanum_key`(*s: str*) → list

Helper tool to sort lists in ascending numerical order (natural sorting), rather than lexicographic sorting

Parameters

s [str] string to sort

Returns

list Sorted list

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.common.diffdict`(*first_dict: dict*, *second_dict: dict*) → dict

Helper tool to get difference in two dictionaries

Parameters

first_dict [dict] First dictionary

second_dict [dict] Second dictionary

Returns

dict Difference

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.common.df2nparray`(*dataframe: pandas.core.frame.DataFrame*) → numpy.ndarray

Helper tool to return the named numpy array of the pandas dataframe

Parameters

dataframe [pd.DataFrame] Input Pandas Dataframe

Returns

np.ndarray Equivalent named numpy array

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**krunge**(*n: int*, *x: float*, *h: Union[int, float]*, *y: Union[list, tuple, numpy.ndarray]*, *f: Union[list, tuple, numpy.ndarray]*, *m: int = 0*, *phi: numpy.ndarray = array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])*, *savey: numpy.ndarray = array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])*)

Some sort of integration or interpolation? x is incremented on second and fourth calls. Resets itself after 5'th call.

Parameters

n [int] number of points

x [float] indepent variable for points

h [tp.Union[int,float]] step size

y [tp.Union[list,tuple,np.ndarray]] function evaluated at each point

f [tp.Union[list,tuple,np.ndarray]] function evaluated at each point

m [int, optional] call number, by default 0

phi [np.ndarray, optional] Intermediate variable, by default np.zeros(6)

savey [np.ndarray, optional] Intermediate variable, by default np.zeros(6)

avni.tools.common.**firstnonspaceindex**(*string: str*) → Tuple[int, int]

Gets the first and last non-space index of a string

Parameters

string [str] String to search

Returns

tp.Tuple[int,int] First and last non-space index

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**get_fullpath**(*path: str*) → str

Provides the full path by replacing . and ~ in path

Parameters

path [str] Input file or folder path

Returns

str Full path after replacing . and ~ in path

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**listfolders**(*path: str*) → list

Return a list of directories in a path

Parameters

path [str] Input file or folder path

Returns

list List of directories

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**get_installdir**(*module: str = 'avni'*, *checkwrite: bool = True*, *checkenv: bool = True*) → str

Get the installation directory for any module in the current machine.

Parameters

module [str, optional] Module to search for, by default 'avni'

checkwrite [bool, optional] Checks for write access to the files, by default True

checkenv [bool, optional] Checks if the directory is specified as an environment variable, by default True

Returns

str Path to installation directory

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**get_filedir**(*module: str = 'avni'*, *subdirectory: Union[None, str] = None*, *checkwrite: bool = True*, *makedir: bool = True*) → str

Get the local files directory for any module in the current machine.

Parameters

module [str, optional] Module to search for, by default 'avni'

subdirectory [tp.Union[None,str], optional] A directory inside the main directory, by default None

checkwrite [bool, optional] Checks for write access to the files, by default True

makedir [bool, optional] Make a new directory if it doesn't exist, by default True

Returns

str Path to local files directory

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**get_cptdir**(*module: str = 'avni'*, *checkwrite: bool = True*, *makedir: bool = True*) → str

Get the directory with color palettes for any module in the current machine.

Parameters

module [str, optional] Module to search for, by default 'avni'

checkwrite [bool, optional] Checks for write access to the files, by default True

makedir [bool, optional] Make a new directory if it doesn't exist, by default True

Returns

str Path to local CPT color palette directory

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**get_configdir**(*module: str = 'avni'*, *checkwrite: bool = True*, *makedir: bool = True*) → str

Get the directory containing configuration files.

Parameters

module [str, optional] Module to search for, by default 'avni'

checkwrite [bool, optional] Checks for write access to the files, by default True

makedir [bool, optional] Make a new directory if it doesn't exist, by default True

Returns

str Path to local config directory as specified in `constants.configfolder()`

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**get_projections**(*checkwrite: bool = True*, *makedir: bool = True*, *types: str = 'radial'*) → Tuple[str, bool]

Get the file containing projection matrices.

Parameters

checkwrite [bool, optional] Checks for write access to the files, by default True

makedir [bool, optional] Make a new directory if doesn't exist, by default True

types [str, optional] Type can be radial or lateral, by default 'radial'

Returns

tp.Tuple[str,bool] First entry is location of projection file Second end is whether this file already exists in the file system

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.common.writejson`(*nparray: numpy.ndarray*, *filename: str*, *encoding: str = 'utf-8'*)

Writes a json file from a numpy array

Parameters

nparray [np.ndarray] Input numpy array to write to JSON

filename [str] Output JSON file name

encoding [str, optional] File encoding, by default 'utf-8'

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.02.16 5.00

`avni.tools.common.readjson`(*filename: str*, *encoding: str = 'utf-8'*) → numpy.ndarray

Reading from a JSON file to a numpy array

Parameters

filename [str] Input JSON file name

encoding [str, optional] File encoding, by default 'utf-8'

Returns

np.ndarray Output numpy array

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.common.uniquenumpyrow`(*a: numpy.ndarray*)

Gets the unique rows from a numpy array and the indices. e.g. to get unique lat-lon values

Parameters

a [np.ndarray] A numpy aray with multiple rows that needs to searched

Returns

unique_a,idx Unique rows and their indices

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**sanitised_input**(*prompt: str*, *type_=None*, *min_=None*, *max_=None*, *range_=None*)

Provide a user prompt with values between min-max or range of values

For specific values: user_input = sanitised_input("Replace(r)/Ignore(i) this datum?", str.lower, range_=('r', 'i') For a range: age = sanitised_input("Enter your age: ", int, range_=xrange(100))

Parameters

prompt [str] Prompt to the user

type_ [optional] type of input needed, by default None

min_ [optional] Minimum value, by default None

max_ [optional] Maximum value, by default None

range_ [optional] Range of values, by default None

Returns

ui User input

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.common.**appendunits**(*ureg=None*, *system: str = 'mks'*, *unitsfile: str = '/home/pm5113/Github/avni-private/avni/config/units.ini'*)

Append the custom units from unitsfile to *ureg* registry

Parameters

ureg [_type_, optional] Input unit registry, by default constants.ureg

system [str, optional] Default unit system; if not the same as ureg changes it, by default 'mks'

unitsfile [str, optional] additional definitions to add to *ureg*, by default get_configdir()+'/'+constants.customunits

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.02.16 5.00

avni.tools.common.**convert2units**(*valstring: str*)

Returns the value with units from a string that has units.

Parameters

valstring [str] A string to convert. Only space allowed is that between value and unit.

Returns

vals Value with units

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.common.decimals`(*value: Union[list, tuple, numpy.ndarray]*) → numpy.ndarray

Returns the number of decimals in a set of floating point numbers.

Parameters

value [tp.Union[list,tuple,np.ndarray]] Set of floating point numbers

Returns

np.ndarray Number of decimals in each float

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.harmonics module

`avni.tools.harmonics.getdepthsfolder`(*folder: str = '.'*, *extension: str = '.epix'*, *delimiter: str = '.'*) → list

Get list of depths from filenames to iterate through

Parameters

folder [str, optional] Folder to search, by default '.'

extension [str, optional] File extension to search, by default '.epix'

delimiter [str, optional] delimiter in file names, by default '.'

Returns

list List of depths

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.harmonics.rdswpsh`(*filename: str*) → Tuple[numpy.ndarray, dict, list]

Code to get spherical harmonic coefficients in the *ylm* normalization.

Parameters

filename [str] Input file with spherical harmonic coefficients

Returns

tp.Tuple[np.ndarray, dict, list] First element is a numpy array with coefficients in the *ylm* normalization.

Second element are metadata from input fields if specified.

Third element are all other comments except lines containing metadata.

Raises

IOError File not found in the file system

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.harmonics.swp_to_epix(`*infile: str*, *grid: int = 1*, *lmax: Union[None, int] = None*`)` → Tuple[numpy.ndarray, dict, list]

Convert spherical harmonics coefficients to extended pixel (.epix) format.

Parameters

infile [str] Input spherical harmonic coefficient file

grid [int, optional] Grid size for pixels, by default 1

lmax [tp.Union[None,int], optional] Maximum angular degree, by default None which results in the maximum specified on file

Returns

tp.Tuple[np.ndarray, dict, list] First element is an array containing (*latitude*, *longitude*, *pixel_size*, *value*).

Second element are metadata from input fields if specified.

Third element are all other comments except lines containing metadata.

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.harmonics.wrswpsh(`*filename: str*, *shmatrix: numpy.ndarray*, *metadata: dict = {'FORMAT': '0'}*, *comments: Optional[list] = None*, *lmax: Union[None, int] = None*`)`

Write spherical harmonic coefficients from the ylm normalization to a file.

Parameters

filename [str] Output file name

shmatrix [np.ndarray] Numpy array with coefficients in the *ylm* normalization

metadata [_type_, optional] Metadata from input fields if specified., by default {'FORMAT':'0'}

comments [list, optional] All other comments except lines containing metadata., by default None

lmax [tp.Union[None,int], optional] Maximum angular degree, by default None which results in the maximum specified on file

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.02.16 5.00

avni.tools.harmonics.**get_coefficients**(*shmatrix: numpy.ndarray*, *lmin: Union[None, int] = None*, *lmax: Union[None, int] = None*) → numpy.ndarray

Read the spherical harmonic coefficients into a named numpy array

Parameters

shmatrix [np.ndarray] Numpy array with coefficients in the *ylm* normalization

lmin [tp.Union[None,int], optional] Minimum angular degree, by default None which results in the minimum specified on file

lmax [tp.Union[None,int], optional] Maximum angular degree, by default None which results in the maximum specified on file

Returns

———-

np.ndarray Named numpy array of spherical harmonic coefficients

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.02.16 5.00

avni.tools.harmonics.**swp_to_xarray**(*shmatrix: numpy.ndarray*, *grid: int = 10*, *lmax: Union[None, int] = None*) → xarray.core.dataarray.DataArray

Convert from spherical harmonic coefficients in *ylm* normalization to a multi-dimensional pixel grid.

Parameters

shmatrix [np.ndarray] Numpy array with coefficients in the *ylm* normalization

grid [int, optional] Grid size in degrees for pixels, by default 10

lmax [tp.Union[None,int], optional] Maximum angular degree, by default None which results in the maximum specified on file

Returns

xr.DataArray A multi-dimensional xarray DataArray containing the pixel grid

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.harmonics.**convert_to_swp**(*data: Union[numpy.ndarray, xarray.core.dataarray.DataArray]*, *lmax: int*) → numpy.ndarray

Convert multi-dimensional pixel grid to spherical harmonic coefficients in *ylm* normalization.

Parameters

data [tp.Union[np.ndarray,xr.DataArray]] A multi-dimensional xarray DataArray or named numpy array containing the pixel grid

lmax [int] Maximum angular degree to evaluate coefficients to

Returns

np.ndarray Numpy array with coefficients in the *ylm* normalization

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.harmonics.**calcshpar2**(*shmatrix: numpy.ndarray*, *lmax: Union[None, int] = None*) → Tuple[float, float, float, numpy.ndarray]

This calculates the mean, roughness, RMS and power of spherical harmonic coefficients in *ylm* normalization

Parameters

shmatrix [np.ndarray] Numpy array with coefficients in the *ylm* normalization

lmax [tp.Union[None,int], optional] Maximum angular degree, by default None which results in the maximum specified on file

Returns

tp.Tuple[float,float,float,np.ndarray] First element the globally averaged value on the unit sphere.

Second and third elements are RMS and roughness values, respectively.

Fourth element is a numpy array containing the power at each degree.

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.harmonics.**swp_correlation**(*shmatrix1: numpy.ndarray*, *shmatrix2: numpy.ndarray*, *lmax: Union[None, int] = None*) → Tuple[float, float, numpy.ndarray, numpy.ndarray]

Calculate RMS and correlation between two sets of spherical harmonic coefficients.

Parameters

shmatrix1 [np.ndarray] First numpy array with coefficients in the *ylm* normalization

shmatrix2 [np.ndarray] Second numpy array with coefficients in the *ylm* normalization

lmax [tp.Union[None,int], optional] Maximum angular degree, by default None which results in the maximum specified on file

Returns

tp.Tuple[float,float,np.ndarray,np.ndarray] First and second elements are the RMS values for the two sets of coefficients.

Third element is the numpy array containing the correlation at each spherical harmonic degree.

Fourth element is a numpy array containing cumulative correlation up to each spherical harmonic degree.

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.io module

`avni.tools.io.close_h5py()`

Close all h5py files

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.io.store_sparse_hdf(`*h5f*, *varname: str*, *mat*, *compression: str = 'gzip'*`)`

Store a *csr* matrix in HDF5

Parameters

h5f HDF5 file handle

varname [str] node prefix in HDF5 hierarchy

mat [scipy.sparse.csr.csr_matrix] sparse matrix to be stored

compression [str, optional] Compression type in HDF5, by default "gzip"

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.02.16 5.00

`avni.tools.io.load_sparse_hdf(`*h5f*, *varname: str*`)`

Load a *csr* matrix from HDF5 file

Parameters

h5f HDF5 file handle

varname [str] node prefix in HDF5 hierarchy

Returns

scipy.sparse.csr.csr_matrix A sparse *csr* matrix

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.io.**store_numpy_hdf**(*h5f*, *varname: str*, *array: numpy.ndarray*, *compression: str = 'gzip'*, *compression_opts: int = 9*)

Store a named numpy array in HDF5 file

Parameters

h5f HDF5 file handle

varname [str] node prefix in HDF5 hierarchy

array [np.ndarray] Named numpy array

compression [str, optional] Compression type in HDF5, by default "gzip"

compression_opts [int, optional] Compression level opts, by default 9

:Authors: Raj Moulik (moulik@caa.columbia.edu)

:Last Modified: 2023.02.16 5.00

avni.tools.io.**load_numpy_hdf**(*h5f*, *varname: str*) → numpy.ndarray

Read a named numpy array from HDF5 file

Parameters

h5f HDF5 file handle

varname [str] node prefix in HDF5 hierarchy

Returns

np.ndarray Named numpy array

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.trigd module

This module has several trogonometric routines in degrees. Typically used to interface with fortran codes.

avni.tools.trigd.**cosd**(*x: Union[float, int]*) → float

avni.tools.trigd.**sind**(*x: Union[float, int]*) → float

avni.tools.trigd.**tand**(*x: Union[float, int]*) → float

avni.tools.trigd.**acosd**(*x: Union[float, int]*) → float

avni.tools.trigd.**asind**(*x: Union[float, int]*) → float

avni.tools.trigd.**atand**(*x: Union[float, int]*) → float

avni.tools.trigd.**atan2d**(*y: Union[float, int]*, *x: Union[float, int]*) → float

avni.tools.xarray module

avni.tools.xarray.**xarray_to_epix**(*data: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *latname: str = 'latitude'*, *lonname: str = 'longitude'*) → numpy.ndarray

Convert multi-dimensional pixel grid in xarray to extended pixel (.epix) format

Parameters

data [tp.Union[xr.DataArray,xr.Dataset]] Multi-dimensional pixel grid in xarray formats

latname [str, optional] Name to use for latitude column in named numpy array, by default 'latitude'

lonname [str, optional] Name to use for longitude column in named numpy array, by default 'longitude'

Returns

np.ndarray Array containing (*latitude*, *longitude*, *pixel_size*, *value*)

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.xarray.**epix_to_xarray**(*epixarr: numpy.ndarray*, *latname: str = 'latitude'*, *lonname: str = 'longitude'*) → xarray.core.dataarray.DataArray

Convert extended pixel (.epix) to multi-dimensional pixel grid in xarray format

Parameters

epixarr [np.ndarray] Array containing (*latitude*, *longitude*, *pixel_size*, *value*)

latname [str, optional] Name to use for latitude column in named numpy array, by default 'latitude'

lonname [str, optional] Name to use for longitude column in named numpy array, by default 'longitude'

Returns

xr.DataArray Multi-dimensional pixel grid in xarray formats

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.xarray.**tree3D**(*treefile: str*, *latitude: Union[None, list, tuple, numpy.ndarray] = None*, *longitude: Union[None, list, tuple, numpy.ndarray] = None*, *radius_in_km: Union[None, list, tuple, numpy.ndarray] = None*)

Build a KD-tree at specific locations

Parameters

treefile [str] Name of the file where tree is (or will be) stored

latitude [tp.Union[list,tuple,np.ndarray], optional] Latitudes of locations queried, by default None

longitude [tp.Union[list,tuple,np.ndarray], optional] Longitudes of locations queried, by default None

radius_in_km [tp.Union[list,tuple,np.ndarray], optional] Radii of locations queried, by default None

Returns

tree A `scipy.spatial.cKDTree()`

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.xarray.**querytree3D**(*tree*, *latitude: Union[list, tuple, numpy.ndarray]*, *longitude: Union[list, tuple, numpy.ndarray]*, *radius_in_km: Union[list, tuple, numpy.ndarray]*, *values: Union[None, scipy.sparse.csc.csc_matrix, scipy.sparse.csr.csr_matrix, list, tuple, numpy.ndarray] = None*, *nearest: int = 1*)

Query a KD-tree for values at specific locations

Parameters

tree A `scipy.spatial.cKDTree()`

latitude [tp.Union[list,tuple,np.ndarray], optional] Latitudes of locations queried, by default None

longitude [tp.Union[list,tuple,np.ndarray], optional] Longitudes of locations queried, by default None

radius_in_km [tp.Union[list,tuple,np.ndarray], optional] Radii of locations queried, by default None

values [tp.Union[None,sparse.csc_matrix,sparse.csr_matrix,list,tuple,np.ndarray], optional] Values at each KD-tree point, by default None

nearest [int, optional] Number of nearest values in the KD-tree to interpolated from, by default 1 so nearest

Returns

inds or interp, inds indices of the nearest points in te KD-tree and the interporlated value (if values is not None)

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.xarray.get_stride`(*resolution: str*) → int

Get the stride to use for various resolution of plotting. This dictates downsampling before interpolation.

Parameters

resolution [str] Resolution of boundary database to use in Basemap. Can be c (crude), l (low), i (intermediate), h (high), f (full)

Returns

int stride to use for resolution

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

`avni.tools.xarray.ncfile2tree3D`(*ncfile: str*, *treefile: str*, *lonlatdepth: list = ['longitude', 'latitude', 'depth']*, *resolution: str = 'h'*, *radius_in_km: Union[None, list, tuple, numpy.ndarray] = None*)

Read or write a pickle interpolant with KD-tree

Parameters

ncfile [str] Name of the topography file in NETCDF4 format

treefile [str] Name of the file where tree is (or will be) stored

lonlatdepth [list, optional] A list of variable names of the longitude, latitude, depth (in km) arrays, by default ['longitude','latitude','depth']

resolution [str, optional] Dictates downsampling before interpolation, by default 'h'

radius_in_km [tp.Union[None,list,tuple,np.ndarray], optional] Radius in kilometer when a 2D surface. Ignores the 3rd field in *lonlatdepth*, by default None

Returns

tree A `scipy.spatial.cKDTree()`

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.xarray.**readtopography**(*model: Union[None, str] = None*, *resolution: str = 'h'*, *field: str = 'z'*, *latitude: str = 'lat'*, *longitude: str = 'lon'*, *latitude_limits: list = [- 90, 90]*, *longitude_limits: list = [- 180, 180]*, *dbs_path: Union[None, str] = None*)

Read standard topography file in NETCDF4 format specified in `constants()`

Parameters

model [str, optional] Name of the topography file in NETCDF4 format, by default `constants.topography()`

resolution [str, optional] Dictates downsampling before interpolation, by default 'h'

field [str, optional] Field name in the NETCDF4 file to use, by default 'z'

latitude [str, optional] Name to use for latitude column in named numpy array, by default 'lat'

longitude [str, optional] Name to use for longitude column in named numpy array, by default 'lon'

latitude_limits [list, optional] Limit for restricting the domain for reading topography, by default [-90,90]

longitude_limits [list, optional] Limit for restricting the domain for reading topography, by default [-180,180]

dbs_path [tp.Union[None,str], optional] Database path to the folder contianing topography file, by default None so takes the value in `constants.topofolder()`

Returns

xr.Dataset Multi-dimensional pixel grid in xarray formats

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.xarray.**checkxarray**(*data: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *latname: str = 'latitude'*, *lonname: str = 'longitude'*, *Dataset=True*) → Tuple[list, float, tuple]

Checks whether the data input is a DataArray and the coordinates are compatible

Parameters

data [tp.Union[xr.DataArray,xr.Dataset]] Multi-dimensional pixel grid in xarray formats

latname [str, optional] Name to use for latitude column in named numpy array, by default 'latitude'

lonname [str, optional] Name to use for longitude column in named numpy array, by default 'longitude'

Dataset [bool, optional] Allow Dataset or not, by default True

Returns

tp.Tuple[list,float,tuple] First element is a list of error warnings while performing checks

Second element is size of the uniform pixel.

Third element is a tuple containing the shape of the grid

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.xarray.**areaxarray**(*data: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *latname: str = 'latitude'*, *lonname: str = 'longitude'*, *pix_width: Union[None, numpy.ndarray] = None*) → xarray.core.dataarray.DataArray

Calculate area for multi-dimensional pixel grid in xarray formats

Parameters

data [tp.Union[xr.DataArray,xr.Dataset]] Multi-dimensional pixel grid in xarray formats

latname [str, optional] Name to use for latitude column, by default 'latitude'

lonname [str, optional] Name to use for longitude column, by default 'longitude'

pix_width [tp.Union[None,np.ndarray], optional] Width of pixels if not the default derived from data, by default None so is derived

Returns

xr.DataArray A DataArray object with area of each pixel

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

avni.tools.xarray.**meanxarray**(*data: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *area: Union[None, xarray.core.dataarray.DataArray] = None*, *latname: str = 'latitude'*, *lonname: str = 'longitude'*, *pix_width: Union[None, numpy.ndarray] = None*) → Tuple[float, xarray.core.dataarray.DataArray, float]

Calculate geographically weighted average of a multi-dimensional pixel grid in xarray formats

Parameters

data [tp.Union[xr.DataArray,xr.Dataset]] Multi-dimensional pixel grid in xarray formats to average over

area [tp.Union[None,xr.DataArray], optional] Area of each pixel, by default None so calculated on the fly

latname [str, optional] Name to use for latitude column, by default 'latitude'

lonname [str, optional] Name to use for longitude column, by default 'longitude'

pix_width [tp.Union[None,np.ndarray], optional] Width of pixels if not the default derived from data, by default None so is derived

Returns

tp.Tuple[float,xr.DataArray,float] First element is the global average Second element is a DataArray containing area weights for each pixel Third element is the percentage of global area covered by this basis set

Authors Raj Moulik (moulik@caa.columbia.edu)

Last Modified 2023.02.16 5.00

5.7 avni.constants module

This module has several constants that are accessed by AVNI functions. These include locations of databases and default options for certain calculations and file locations.

5.8 avni.f2py module

This module 'f2py' is auto-generated with f2py (version:2). Functions:

closfl(lufl,istat) openfl(lufl,namein,iap,ifile,irec,istat,lrec) bffi(lufl,ifbin,ibuf,nbytes,istat,nread,irec) getflpos(lufl,iposition,ierror) swap4of4(iword) acosd = acosd(arg) dsind = dsind(darg) atand = atand(arg) sind = sind(arg) cosd = cosd(arg) dcosd = dcosd(darg) dacosd = dacosd(darg) tand = tand(arg) asind = asind(arg) tadder(jy,jd,ih,im,fs,jy1,jd1,ih1,im1,fs1,secs) y,wk1,wk2,wk3 = ylm(xlat,xlon,lmax,ylen,wklen) lpyr = lpyr(iy) drspleder = drspleder(i1,i2,x,y,q,s,ilen=len(x)) x,xp,xcosec = legndr(theta,l,m,wklen) ddelaz(eplat,eplong,stlat,stlong,delta,azep,azst) delta,azep,azst = ddelazgc(eplat,eplong,stlat,stlong) pdaz(epla,eplo,azim,delta,xlat,xlon) delazgc(eplat,eplong,stlat,stlong,delta,azep,azst) monday(iy,jd,mo,id) julday = julday(iy,imo,idm) delaz(eplat,eplong,stlat,stlong,delta,azep,azst) yreal = shold(rlat,rlon,lold,numold) sub_indices(lmax,numcoe,indexl,indexm,indextyp) xp = xp(l,m,theta) factrl = factrl(n) gammln = gammln(xx) plgndr = plgndr(l,m,x) difftime(time1,time2,timediff) nread,ierror = loadnbn2memory(igcmtchoice,allorder,qcmt) getcmt5(ievt,catalog,iyear,month,iday,ihour,minute,fsec,catlat,catlon,catdep,xmb,xms,region,event,stamp,ela

ievt,iyear,month,iday,ihour,minute,fsec,elat,elon,edep,xmw,ierror = getcmtbyname(event) readnbn(lu,nread,ierror) splcon,splcond = vbspl(x,np,xarr) q,f = drspln(i1,i2,x,y) drsple = drsple(i1,i2,x,y,q,s) qtau(ilay,x,pray,isotropic,iraytype,qray) get_ishflag(phase,ishflag) reademfl(lu,nl,xb,xt,icm,ici,nlev,iflu,iani,ierr,ml=len(xb)) get_branch(imodel,phase,narr,delta,slowness,ddeltadp,branch,ierror) qtauall(ilay,x,pray,isotropic,iraytype,qray,vpv,vph,vsv,vsh,eta,qmu,qkappa,s1,s2,s3,s4,s5,r) evemb(ilay,rad,rho,grav,bigk,pres,vaisala,bullen,ierr) fdrays(delta,npps,p,t,d,dddp,q,turn,pr,tr,dddpr,qr,turac evem(ilay,rad,isotropic,rho,vpv,vph,vsv,vsh,eta,qmu,qka,ierr) dxt(ilay,x,pray,isotropic,iraytype,d,f) getptab(ianisoflag,ishflag,iss,isl,np,ns,id,pmin,pmax,nps,p,t,d,dddp,tsta drspledr = drspledr(i1,i2,x,y,q,s) evemdr(ilay,rad,isotropic,rho,vpv,vph,vsv,vsh,eta,qmu,qka) fqs(ilay,x,xnorm,iturn,fxtpp,dxtdp,pray,isotropic,iraytype,nvals,vals) prange(name,xbotlay,xtoplay,ianisoflag,ishflag,radso,isl,icm,ici,pmin,pmax,numlay=len(xbotlay)) evemell(rad,ell,eta2,ierr) qtauzero(ilay,a1,b1,pray,isotropic,iraytype,z) grav,vaisala,bullen,pressure = getbullen(emodel,maxlay,capom,capg) cagcrays_pm(lu,filename,epla,eplo,depth,stla,stlo,phase,ishflag,ianisoflag,itypcalc,delta,azep,azst,narr,arrival anivel(th,itp,vpv8,vph8,vsv8,vsh8,eta8,vpha,vgrp,phi) reset_ptab() reademb(lu,nl,xb,xt,nlev,iflu,iani,capom,capg,ierr,ml=len(xb)) krunge = krunge(n,y,f,x,h) readptab(lu,initpos,rad4,isl,ifanis,ierr) tt_predict(delstep,filename,iallpsv,iellip,epla,eplo,depth,phase_file) wgray(pray,ilay,xbot,xtop,numlev,isotropic,iraytype,delta,ttime,dddp,tstar,xturn,npts,runrad,rundel,runtim, conv2geocen(xlatin,eplaout) wgint(ilay,r1,r2,xnorm,iturn,fxtpp,dxtdp,pray,isotropic,iraytype,nint,deltainc,tti psvrayin(ia,ics,icm,ici,i670,i400,i220,np,ns,id,iss,irr,nn,ilay=len(np)) addray(nps,id,ic1,i,ic2=len(nps)) jchar = jchar(ia) rayseq(np,ns,iss,irr,icnsx,icnr,icn,iseq,idirseg,nseg,nn=len(np)) yarr,yarrp = dbsplrem(x,x0,xright,np)

COMMON blocks: /gioheader/ iopen(20),ipos(20),irw(20),isize(20),irecl(20) /plevel/ iprtlv /nbncommon/ nreadnbn,ivernbn(500000),cmtnamenbn(500000,16),stampnbn(500000,16),catalognbn(500000 /empar1/ emtitle(80) /empar2/ radlev(1000),rholev(1000),rhospl(3,1000),vpvlev(1000),vpvspl(3,1000),vsvlev /empar3/ topemreg(30),botemreg(30),itoplev(30),ibotlev(30),numemreg /ptablex/ nlayers,pfull(16000),tfull(16000,25,3),dfull(16000,25,3),dddpfull(16000,25,3),tstarfull(16000,25,3),xturnfull(16000

.

CHAPTER

SIX

GETTING HELP

There are several places to obtain help with avni software tools.

- The AVNI Forum is a good placed to go for both troubleshooting and general questions.
- The FAQ page has some troubleshooting tips, and is a good source of general information. There are also some troubleshooting tips built into the Python and AVNI installation pages.
- If you want to request new features or if you're confident that you have found a bug, please create a new issue on the GitHub issues page. When reporting bugs, please try to replicate the bug with sample data, and make every effort to simplify your example script to only the elements necessary to replicate the bug.

CHAPTER

SEVEN

AVNI DEVELOPMENT

Warning: Reminder: all contributors are expected to follow our *code of conduct*. All contributions are governed by the terms of our license.

AVNI is a community project that lives by the participation of its members — i.e. including you! It is our goal to build an inclusive and participatory community so we are happy that you are interested in participating!

To *report* bugs, *request* new features, or *ask about* confusing documentation, it's usually best to open a new issue on AVNI Forum first; you'll probably get help fastest that way, and it helps keep our GitHub issue tracker focused on things that we *know* will require changes to our software (as opposed to problems that can be fixed in the user's code). We may ultimately ask you to open an issue on GitHub too, but starting on the forum helps us keep things organized. For fastest results, be sure to include information about your operating system and AVNI version, and (if applicable) include a reproducible code sample that is as short as possible.

The project accepts contributions in the form of bug reports, fixes and feature additions. For *general troubleshooting* or *usage questions*, please consider posting your questions on our GitHub Discussions. The best way to start contributing is by raising GitHub Issues on our GitHub page to discuss ideas for changes or enhancements, or to tell us about behavior that you think might be a bug.

The goal of this page is to provide instructions and best practices to guide developers and contributing scientists.

Note: Please route your *general* questions to the AVNI Forum. The helps our GitHub issue tracker focused on things that we *know* will require changes to our software (as opposed to problems that can be fixed in the user's code).

7.1 Further reading

- Contributing guide: Contains details on the preferred contribution workflow and the recommended system configuration for a smooth contribution/development experience.
- Using Git for AVNI: The standard way of contributing to AVNI development is described here. This approach is recommended for most users, especially those not familiar with Git. These instructions are adequate for contributing to AVNI, but we recommend that all new users to Git attempt the tutorial below for more complete understanding of the workflow.
- Software Carpentry Git Tutorial: For those who are not familiar with Git, we recommend attempting this tutorial, or even better, attending an in-person tutorial session if available in your area.
- Best Practices: Accepted conventions for AVNI development are described here. To create a product that is maintainable over the long term, it is important that contributing scientists follow these conventions.
- Versioning Conventions: The conventions behind AVNI's version numbering system are explained here.

7.1.1 Contributing guide

Note: If you do not already have one, you will need to open a free account on github. You might also need to create an account on our Web site for API access. Please feel free to reaching out to us at avni@globalseismology.org.

Thanks for taking the time to contribute! AVNI is an open-source project sustained mostly by volunteer effort. We welcome contributions from anyone as long as they abide by our code of conduct.

There are several ways to contribute:

- Use the software, and when you find bugs, tell us about them!
- Fix bugs on your own.
- Tell AVNI Forum about parts of the documentation that you find confusing or unclear.
- Tell AVNI Forum about things you wish AVNI could do.
- Answer questions on the AVNI Forum.
- Fix mistakes or add notes in our function documentation strings.
- Improve existing tutorials or write new ones.
- Implement new features.

In order to *report* bugs, *request* new features, or *ask about* confusing documentation, it's best to open a new issue on the AVNI Forum. You will get help fastest that way, and it helps keep our GitHub issues page focused on things that we *know* will require changes to our software (as opposed

to problems that can be fixed in the user's code). We may ultimately ask you to open an issue on the GitHub issues page too, but starting on the forum helps us keep things organized. For fastest results, be sure to include information about your operating system and AVNI version, and (if applicable) include a reproducible code sample that is as short as possible.

If you want to *fix* bugs, *add* new features, or *improve* our docstrings/tutorials/website, those kinds of contributions are made through our GitHub repository. The rest of this page explains how to set up your workflow to make contributing via GitHub seamless.

Want an example to work through?

Feel free to just read through the rest of the page, but if you find it easier to "learn by doing", take a look at our GitHub issues marked "easy", pick one that looks interesting, and work through it while reading this guide!

Overview of the contribution process

> **Warning:** Reminder: all contributors are expected to follow our code of conduct.

Changes to AVNI are typically made by forking the AVNI GitHub repository, making changes to your fork (usually by cloning it to your personal computer, making the changes locally, and then pushing the local changes up to your fork on GitHub), and finally creating a pull request to incorporate your changes back into the shared "upstream" version of the codebase.

In general you'll be working with three different copies of the AVNI codebase: the official remote copy at https://github.com/globalseismology/avni (usually called `upstream`), your remote fork of the upstream repository (similar URL, but with your username in place of `globalseismology`, and usually called `origin`), and the local copy of the codebase on your computer. The typical contribution process is to:

1. synchronize your local copy with `upstream`
2. make changes to your local copy
3. push your changes to `origin` (your remote fork of the upstream)
4. submit a pull request from your fork into `upstream`

The sections *Basic git commands* and *Regular daily git usage* (below) describe this process in more detail.

Setting up your local development environment

Configuring git

Note: GitHub desktop is a GUI alternative to command line git that some users appreciate; it is available for Windows and MacOS.

To get set up for contributing, make sure you have git installed on your local computer:

- On Linux, the command `sudo apt install git` is usually sufficient; see the official Linux instructions for more options.
- On MacOS, download the .dmg installer; Atlassian also provides more detailed instructions and alternatives such as using MacPorts or Homebrew.
- On Windows, download and install git for Windows. With Git BASH it provides its own shell that includes many Linux-equivalent command line programs that are useful for development.

  Windows 10 also offers the Windows subsystem for Linux *that offers similar functionality to git BASH, but has not been widely tested by AVNI developers yet and may still pose problems with graphical output (e.g. building the documentation)*

Once git is installed, the only absolutely necessary configuration step is identifying yourself and your contact info:

```
$ git config --global user.name "Your Name"
$ git config --global user.email you@yourdomain.example.com
```

Make sure that the same email address is associated with your GitHub account and with your local git configuration. It is possible to associate multiple emails with a GitHub account, so if you initially set them up with different emails, you can add the local email to the GitHub account.

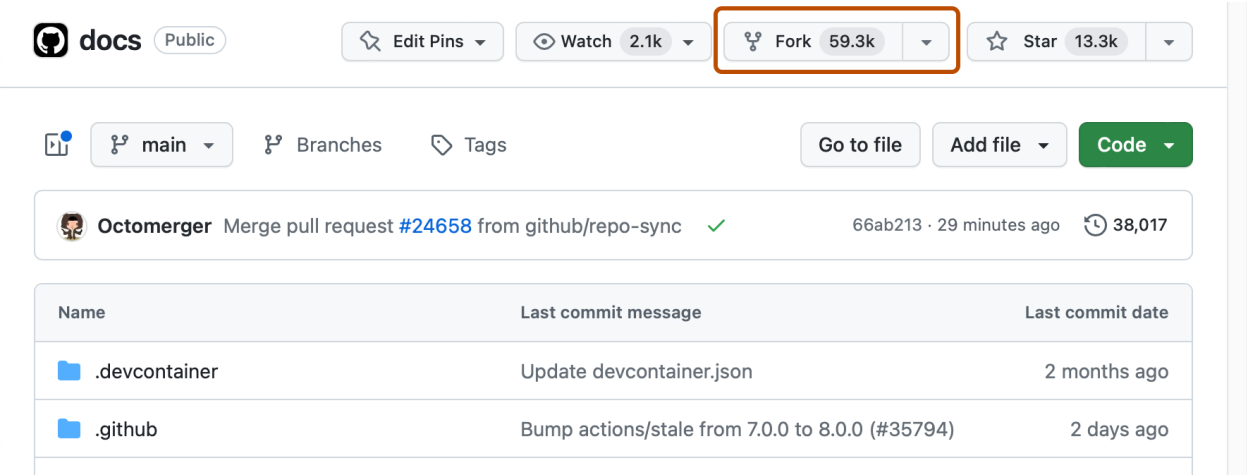
Sooner or later, git is going to ask you what text editor you want it to use when writing commit messages, so you might as well configure that now too:

```
$ git config --global core.editor vim    # or vim, or nano, or subl, or...
```

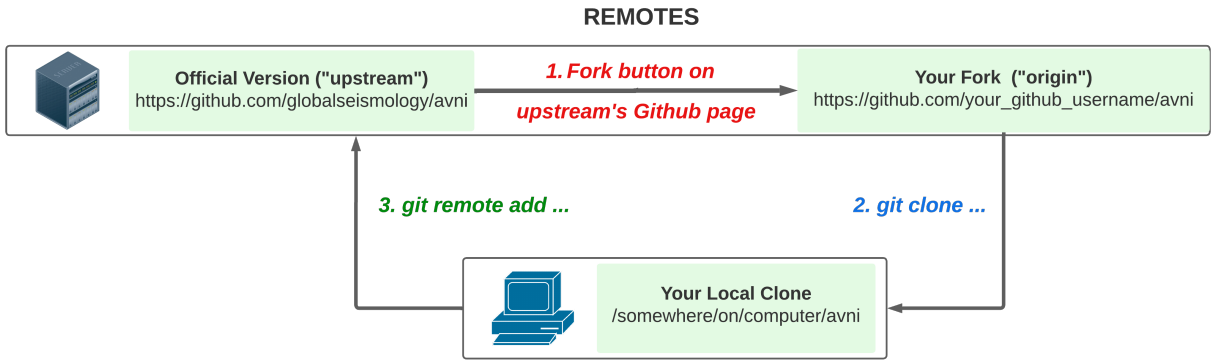
There are many other ways to customize git's behavior; see configuring git for more information.

Forking the AVNI repository

Once you have git installed and configured, and before creating your local copy of the codebase, go to the AVNI GitHub page and create a fork into your GitHub user account.



This will create a copy of the AVNI codebase inside your GitHub user account (this is called "your fork"). Changes you make to AVNI will eventually get "pushed" to your fork, and will be incorporated into the official version of AVNI (often called the "upstream version") through a "pull request". This process will be described in detail below; a summary of how that structure is set up is given here:



Creating the virtual environment

Note: Supported Python environments

We strongly recommend the Anaconda or Miniconda environment managers for Python. Other setups are possible but are not officially supported by the AVNI development team; see discussion here. These instructions use `conda` where possible; experts may replace those lines with some combination of `git` and `pip`.

These instructions will set up a Python environment that is separated from your system-level Python and any other managed Python environments on your computer. This lets you switch between different versions of Python (AVNI version 0.1.0 requires Python version 3.7 or higher) and also switch between the stable and development versions of AVNI (so you can, for example, use the same computer to analyze your data with the stable release, and also work with the latest

development version to fix bugs or add new features). Even if you've already followed the installation instructions for the stable version of AVNI, you should now repeat that process to create a new, separate environment for AVNI development (here we'll give it the name `avni-dev`):

```
$ curl --remote-name https://raw.githubusercontent.com/globalseismology/avni/
↪main/environment.yml
$ conda env create --file environment.yml --name avni-dev
$ conda activate avni-dev
```

Now you'll have *two* AVNI environments: `avni` (or whatever custom name you used when installing the stable version of AVNI) and `avni-dev` that we just created. At this point `avni-dev` also has the stable version of AVNI (that's what the `environment.yml` file installs), but we're about to remove the stable version from `avni-dev` and replace it with the development version. To do that, we'll clone the AVNI repository from your remote fork, and also connect the local copy to the `upstream` version of the codebase, so you can stay up-to-date with changes from other contributors. First, edit these two variables for your situation:

```
$ GITHUB_USERNAME="insert_your_actual_GitHub_username_here"
$ # pick where to put your local copy of AVNI development version:
$ INSTALL_LOCATION="/opt"
```

Note: On Windows, add `set` before the variable names (`set GITHUB_USERNAME=...`, etc.).

Then make a local clone of your remote fork (`origin`):

```
$ cd $INSTALL_LOCATION
$ git clone https://github.com/$GITHUB_USERNAME/avni.git
```

Finally, set up a link between your local clone and the official repository (`upstream`):

```
$ cd AVNI
$ git remote add upstream https://github.com/globalseismology/avni.git
$ git fetch --all
```

Now we'll remove the *stable* version of AVNI and replace it with the *development* version (the clone we just created with git). Make sure you're in the correct environment first (`conda activate avni-dev`), and then do:

```
$ cd $INSTALL_LOCATION/AVNI    # make sure we're in the right folder
$ conda remove --force avni  # the --force avoids dependency checking
$ pip install -e .
```

The command `pip install -e .` installs a python module into the current environment by creating a link to the source code directory (instead of copying the code to pip's `site_packages` directory, which is what normally happens). This means that any edits you make to the AVNI source code will be reflected the next time you open a Python interpreter and `import avni` (the `-e` flag of `pip` stands for an "editable" installation).

Finally, we'll add a few dependencies that are not needed for running AVNI, but are needed for locally running our test suite:

```
$ pip install -r requirements_testing.txt
```

And for building our documentation:

```
$ pip install -r requirements_doc.txt
$ conda install graphviz
```

Note: On Windows, if you installed graphviz using the conda command above but still get an error like this:

```
WARNING: dot command 'dot' cannot be run (needed for graphviz output),
check the graphviz_dot setting
```

try adding the graphviz folder to path:

```
$ PATH=$CONDA_PREFIX\\Library\\bin\\graphviz:$PATH
```

To build documentation, you will also require optipng:

- On Linux, use the command `sudo apt install optipng`.
- On MacOS, optipng can be installed using Homebrew.
- On Windows, unzip `optipng.exe` from the optipng for Windows archive into the `doc/` folder. This step is optional for Windows users.

You can also choose to install some optional linters for reStructuredText:

```
$ conda install -c conda-forge sphinx-autobuild doc8
```

Basic git commands

Learning to work with git can take a long time, because it is a complex and powerful tool for managing versions of files across multiple users, each of whom have multiple copies of the codebase. We've already seen in the setup commands above a few of the basic git commands useful to an AVNI developer:

- `git clone` *<URL_OF_REMOTE_REPO>* (make a local copy of a repository)
- `git remote add` *<NICKNAME_OF_REMOTE>* *<URL_OF_REMOTE_REPO>* (connect a local copy to an additional remote)
- `git fetch --all` (get the current state of connected remote repos)

Other commands that you will undoubtedly need relate to branches. Branches represent multiple copies of the codebase *within a local clone or remote repo*. Branches are typically used to experiment with new features while still keeping a clean, working copy of the original codebase that you

can switch back to at any time. The default branch of any repo is called `main`, and it is recommended that you reserve the `main` branch to be that clean copy of the working `upstream` codebase. Therefore, if you want to add a new feature, you should first synchronize your local `main` branch with the `upstream` repository, then create a new branch based off of `main` and check it out so that any changes you make will exist on that new branch (instead of on `main`):

```
$ git checkout main             # switch to local main branch
$ git fetch upstream              # get the current state of the remote upstream␣
↪repo
$ git merge upstream/main       # synchronize local main branch with remote␣
↪upstream main branch
$ git checkout -b new-feature-x  # create local branch "new-feature-x" and check␣
↪it out
```

Note: Alternative

You can save some typing by using `git pull upstream/main` to replace the `fetch` and `merge` lines above.

Now that you're on a new branch, you can fix a bug or add a new feature, add a test, update the documentation, etc. When you're done, it's time to organize your changes into a series of commits. Commits are like snapshots of the repository — actually, more like a description of what has to change to get from the most recent snapshot to the current snapshot.

Git knows that people often work on multiple changes in multiple files all at once, but that ultimately they should separate those changes into sets of related changes that are grouped together based on common goals (so that it's easier for their colleagues to understand and review the changes). For example, you might want to group all the code changes together in one commit, put new unit tests in another commit, and changes to the documentation in a third commit. Git makes this possible with something called the stage (or *staging area*). After you've made some changes to the codebase, you'll have what git calls "unstaged changes", which will show up with the status command:

```
$ git status    # see what state the local copy of the codebase is in
```

Those unstaged changes can be added to the stage one by one, by either adding a whole file's worth of changes, or by adding only certain lines interactively:

```
$ git add avni/some_file.py      # add all the changes you made to this file
$ git add avni/some_new_file.py  # add a completely new file in its entirety
$ # enter interactive staging mode, to add only portions of a file:
$ git add -p avni/viz/some_other_file.py
```

Once you've collected all the related changes together on the stage, the `git status` command will now refer to them as "changes staged for commit". You can commit them to the current branch with the commit command. If you just type `git commit` by itself, git will open the text editor you configured it to use so that you can write a *commit message* — a short description of the changes you've grouped together in this commit. You can bypass the text editor by passing a

commit message on the command line with the `-m` flag. For example, if your first commit adds a new feature, your commit message might be:

```
$ git commit -m 'ENH: adds feature X to the Epochs class'
```

Once you've made the commit, the stage is now empty, and you can repeat the cycle, adding the unit tests and documentation changes:

```
$ git add avni/tests/some_testing_file.py
$ git commit -m 'add test of new feature X of the Epochs class'
$ git add -p avni/some_file.py avni/viz/some_other_file.py
$ git commit -m 'DOC: update Epochs and BaseEpochs docstrings'
$ git add tutorials/new_tutorial_file.py
$ git commit -m 'DOC: adds new tutorial about feature X'
```

When you're done, it's time to run the test suite to make sure your changes haven't broken any existing functionality, and to make sure your new test covers the lines of code you've added (see *Building the documentation*, below). Once everything looks good, it's time to push your changes to your fork:

```
$ # push local changes to remote branch origin/new-feature-x
$ # (this will create the remote branch if it doesn't already exist)
$ git push origin new-feature-x
```

Finally, go to the AVNI GitHub page, click on the pull requests tab, click the "new pull request" button, and choose "compare across forks" to select your new branch (`new-feature-x`) as the "head repository". See the GitHub help page on creating a PR from a fork for more information about opening pull requests.

If any of the tests failed before you pushed your changes, try to fix them, then add and commit the changes that fixed the tests, and push to your fork. If you're stuck and can't figure out how to fix the tests, go ahead and push your commits to your fork anyway and open a pull request (as described above), then in the pull request you should describe how the tests are failing and ask for advice about how to fix them.

To learn more about git, check out the GitHub help website, the GitHub skills tutorial series, and the pro git book.

Regular daily git usage

A typical workflow on a given workda involves the following steps in sequence on your local machine:

- update your copy of the repository:

  ```
  $ git pull
  ```

- if you get conflicts when doing so (i.e. if local changes you have made conflict with changes made by others on the same line of the same file of the source code), a powerful way of

resolving them is to type this: (*meld* needs to be installed on your system; if it is not, you can install it with *apt-get install meld* or similar):

```
$ git mergetool --tool=meld
```

- make some changes to any file you want using your favorite editor (in the line below we use `vi` as an example):

```
$ vim some_file.f90
```

- commit your changes locally, adding a very short message (one line) explaining what you have changed; it is recommended to do a git pull right before that in order to make sure that your local copy is up-to-date:

```
$ git pull ; git commit -a -m "Explain your commit"
```

- if you get conflicts when committing your changes (i.e. if your changes conflict with changes made by others on the same line of the same file of the source code), a powerful way of resolving them is to type this: (meld needs to be installed on your system; if it is not, you can install it with *yum install meld* in Linux, download MacOS version from this webpage.):

```
$ git mergetool --tool=meld
```

- (optional) if you want to check what has changed (and thus what will be committed) before typing the `git commit` above, you can type one or both of these two commands:

```
$ git status -s
$ git diff
```

- push your changes to your GitHub fork; it is recommended to do a git pull right before that in order to make sure that your local copy is up-to-date:

```
$ git pull ; git push
```

- Create a pull-request to get your changes into the main repository (this is needed only once for each change; if you are fixing an existing change after receiving an error message from our BuildBot code-consistency checking system, you need the "git push" above again but you do NOT need to create a pull request a second time); it is recommended to do a git pull right before that in order to make sure that your local copy is up-to-date:

```
$ git pull ; git pull-request
```

Note (optional): It is not strictly necessary to create a pull request for *every* commit you make if you do not want to, you can safely submit pull requests after making a few commits instead if you prefer. However, it also does not hurt to do so.

Connecting to GitHub with SSH (optional)

One easy way to speed up development is to reduce the number of times you have to type your password. SSH (secure shell) allows authentication with pre-shared key pairs. The private half of your key pair is kept secret on your computer, while the public half of your key pair is added to your GitHub account; when you connect to GitHub from your computer, the local git client checks the remote (public) key against your local (private) key, and grants access your account only if the keys fit. GitHub has several help pages that guide you through the process. As of the date of writing this document,

Once you have set up GitHub to use SSH authentication, you should change the addresses of your AVNI GitHub remotes, from `https://` addresses to `git@` addresses, so that git knows to connect via SSH instead of HTTPS. For example:

```
$ git remote -v  # show existing remote addresses
$ git remote set-url origin git@github.com:$GITHUB_USERNAME/avni.git
$ git remote set-url upstream git@github.com:globalseismology/avni.git
```

AVNI coding conventions

General requirements

All new functionality must be documented

This includes thorough docstring descriptions for all public API changes, as well as how-to examples or longer tutorials for major contributions. Docstrings for private functions may be more sparse, but should usually not be omitted.

Avoid API changes when possible

Changes to the public API (e.g., class/function/method names and signatures) should not be made lightly, as they can break existing user scripts. Bug fixes (when something isn't doing what it says it will do) do not require a deprecation cycle.

Note that any new API elements should be added to the main reference; classes, functions, methods, and attributes cannot be cross-referenced unless they are included in the *Python API Reference* (`doc/python_reference.rst`).

Describe your changes in the changelog

Include in your changeset a brief description of the change in the *changelog* (`doc/changes/latest.inc`; this can be skipped for very minor changes like correcting typos in the documentation).

There are different sections of the changelog for each release, and separate **subsections for bug-fixes, new features, and changes to the public API.** Please be sure to add your entry to the appropriate subsection.

The styling and positioning of the entry depends on whether you are a first-time contributor or have been mentioned in the changelog before.

First-time contributors

Welcome to AVNI! We're very happy to have you here. And to ensure you get proper credit for your work, please add a changelog entry with the following pattern **at the top** of the respective subsection (bugs, enhancements, etc.):

```
Bugs
----

- Short description of the changes (:gh:`0000` by :newcontrib:`Firstname
  Lastname`)

- ...
```

where `0000` must be replaced with the respective GitHub pull request (PR) number, and `Firstname Lastname` must be replaced with your full name.

It is usually best to wait to add a line to the changelog until your PR is finalized, to avoid merge conflicts (since the changelog is updated with almost every PR).

Lastly, make sure that your name is included in the list of authors in `doc/changes/names.inc`, otherwise the documentation build will fail. To add an author name, append a line with the following pattern (note how the syntax is different from that used in the changelog):

```
.. _Your Name: https://www.your-website.com/
```

Many contributors opt to link to their GitHub profile that way. Have a look at the existing entries in the file to get some inspiration.

Recurring contributors

The changelog entry should follow the following patterns:

```
- Short description of the changes from one contributor (:gh:`0000` by
  `Contributor Name`_)
- Short description of the changes from several contributors (:gh:`0000` by
  `Contributor Name`_, `Second Contributor`_, and `Third Contributor`_)
```

where 0000 must be replaced with the respective GitHub pull request (PR) number. Mind the Oxford comma in the case of multiple contributors.

Sometimes, changes that shall appear as a single changelog entry are spread out across multiple PRs. In this case, name all relevant PRs, separated by commas:

```
- Short description of the changes from one contributor in multiple PRs
  (:gh:`0000`, :gh:`1111` by `Contributor Name`_)
- Short description of the changes from several contributors in multiple PRs
  (:gh:`0000`, :gh:`1111` by `Contributor Name`_, `Second Contributor`_, and
  `Third Contributor`_)
```

Test locally before opening pull requests (PRs)

AVNI uses continuous integration (CI) to ensure code quality and test across multiple installation targets. However, the CIs are often slower than testing locally, especially when other contributors also have open PRs (which is basically always the case). Therefore, do not rely on the CIs to catch bugs and style errors for you; run the tests locally instead before opening a new PR and before each time you push additional changes to an already-open PR.

Code style

Adhere to standard Python style guidelines

All contributions to AVNI are checked against style guidelines described in PEP 8. We also check for common coding errors (such as variables that are defined but never used). We allow very few exceptions to these guidelines, and use tools such as pep8, pyflakes, and flake8 to check code style automatically.

When modifying an existing file, try to maintain consistency with its original style. If the code you add looks drastically different from the original code, it may be difficult for readers to follow. Try to avoid this. Please give space for breathing by use 4 spaces instead of tabs:

Listing 7.1: good

```
dx = 0.5 \* fac \* (a - b)
```

Listing 7.2: bad

```
dx=1/2\ *fac*\ (a-b)
```

Use consistent variable naming

Classes should be named using `CamelCase`. Functions and instances/variables should use `snake_case` (`n_samples` rather than `nsamples`). Avoid single-character variable names, unless inside a comprehension or generator.

We (mostly) follow NumPy style for docstrings

In most cases you can look at existing AVNI docstrings to figure out how yours should be formatted. If you can't find a relevant example, consult the Numpy docstring style guidelines for examples of more complicated formatting such as embedding example code, citing references, or including rendered mathematics.

Note that we diverge from the NumPy docstring standard in a few ways:

1. We use a module called `sphinxcontrib-bibtex` to render citations. Search our source code (`git grep footcite` and `git grep footbibliography`) to see examples of how to add in-text citations and formatted references to your docstrings, examples, or tutorials. The structured bibliographic data lives in `doc/references.bib`; please follow the existing key scheme when adding new references (e.g., `Singleauthor2019`, `AuthoroneAuthortwo2020`, `FirstauthorEtAl2021a`, `FirstauthorEtAl2021b`).
2. We don't explicitly say "optional" for optional keyword parameters (because it's clear from the function or method signature which parameters have default values).
3. For parameters that may take multiple types, we use pipe characters instead of the word "or", like this: `param_name : str | None`.
4. We don't **always** include a `Raises` or `Warns` section describing errors/warnings that might occur. Whenever we do, these are only for errors that are non-obvious or have a large chance of getting raised.

Other style guidance

- Guidelines for formatting in Fortran is provided in the *Fortran Guidelines* page.
- Use single quotes whenever possible.
- Prefer generators or comprehensions over `filter()`, `map()` and other functional idioms.
- Use explicit functional constructors for builtin containers to improve readability (e.g., list(), dict(), set()).
- Avoid nested functions or class methods if possible — use private functions instead.

- Avoid `*args` and `**kwargs` in function/method signatures.

Code organization

Importing

Import modules in this order, preferably alphabetized within each subsection:

1. Python built-in (`copy`, `functools`, `os`, etc.)
2. NumPy (`numpy as np`) and, in test files, pytest (`pytest`)
3. AVNI imports (e.g., `from .pick import pick_types`)

When importing from other parts of AVNI, use relative imports in the main codebase and absolute imports in tests, tutorials, and how-to examples. Imports for `matplotlib`, `scipy`, and optional modules (`sklearn`, `pandas`, etc.) should be nested (i.e., within a function or method, not at the top of a file). This helps reduce import time and limit hard requirements for using AVNI.

Return types

Methods should modify inplace and return `self`, functions should return copies (where applicable). Docstrings should always give an informative name for the return value, even if the function or method's return value is never stored under that name in the code.

Building the documentation

Important: Latest documentation should generally be written in MyST format, unless the use case requires more complicated use of the reStructuredText format. Conversion between the two formats can be done using pandoc: `pandoc --from=rst --to=markdown --output=README.md README.rst`

Our documentation (including docstrings in code files) is in two formats that are built using Sphinx:

1. reStructuredText format that is built using Sphinx-Gallery
2. MyST format, a rich and extensible flavour of Markdown format, that is built using myst_parser

The easiest way to ensure that your contributions to the documentation are properly formatted is to follow the style guidelines on this page, imitate existing documentation examples, refer to the Sphinx and Sphinx-Gallery reference materials when unsure how to format your contributions, and build the docs locally to confirm that everything looks correct before submitting the changes in a pull request.

You can build the documentation locally using GNU Make with `doc/Makefile`. From within the `doc` directory, you can test formatting and linking by running:

```
$ make html_dev-noplot
```

This will build the documentation *except* it will format (but not execute) the tutorial and example files. If you have created or modified an example or tutorial, you should instead run `PATTERN=<REGEX_TO_SELECT_MY_TUTORIAL> make html_dev-pattern` to render all the documentation and additionally execute just your example or tutorial (so you can make sure it runs successfully and generates the output / figures you expect).

Note: If you are using a *Windows command shell*, to use the pattern approach, use the following two lines:

```
> set PATTERN=<REGEX_TO_SELECT_MY_TUTORIAL> > make html_dev-pattern
```

If you are on Windows but using the git BASH shell, use the same two commands but replace `set` with `export`.

After either of these commands completes, `make show` will open the locally-rendered documentation site in your browser. If you see many warnings that seem unrelated to your contributions, it might be that your output folder for the documentation build contains old, now irrelevant, files. Running `make clean` will clean those up. Additional `make` recipes are available; run `make help` from the `doc` directory or consult the Sphinx-Gallery documentation for additional details.

GitHub workflow

Nearly everyone in the community of AVNI contributors and maintainers is a working scientist, engineer, or student who contributes to AVNI in their spare time. For that reason, a set of best practices have been adopted to streamline the collaboration and review process. Most of these practices are common to many open-source software projects, so learning to follow them while working on AVNI will bear fruit when you contribute to other projects down the road. Here are the guidelines:

- Search the GitHub issues page (both open and closed issues) in case someone else has already started work on the same bugfix or feature. If you don't find anything, open a new issue to discuss changes with maintainers before starting work on your proposed changes.
- Implement only one new feature or bugfix per pull request (PR). Occasionally it may make sense to fix a few related bugs at once, but this makes PRs harder to review and test, so check with AVNI maintainers first before doing this. Avoid purely cosmetic changes to the code; they make PRs harder to review.
- It is usually better to make PRs *from* branches other than your main branch, so that you can use your main branch to easily get back to a working state of the code if needed (e.g., if you're working on multiple changes at once, or need to pull in recent changes from someone else to get your new feature to work properly).
- In most cases you should make PRs *into* the upstream's main branch, unless you are specifically asked by a maintainer to PR into another branch (e.g., for backports or maintenance

bugfixes to the current stable version).

- Don't forget to include in your PR a brief description of the change in the *changelog* (`doc/whats_new.rst`).
- Our community uses the following commit tags and conventions:
  - Work-in-progress PRs should be created as draft PRs and the PR title should begin with `WIP`.
  - When you believe a PR is ready to be reviewed and merged, convert it from a draft PR to a normal PR, change its title to begin with `MRG`, and add a comment to the PR asking for reviews (changing the title does not automatically notify maintainers).
  - PRs that only affect documentation should additionally be labelled `DOC`, bugfixes should be labelled `FIX`, and new features should be labelled `ENH` (for "enhancement"). `STY` is used for style changes (i.e., improving docstring consistency or formatting without changing its content).
  - the following commit tags are used to interact with our continuous integration (CI) providers. Use them judiciously; *do not skip tests simply because they are failing*:
    - `[skip actions]` Skip our GitHub Actions, which test installation and execution on Linux and macOS systems.
    - `[ci skip]` is an alias for `[skip actions][skip azp][skip circle]`. Notice that `[skip ci]` is not a valid tag.
    - `[circle full]` triggers a "full" documentation build, i.e., all code in tutorials and how-to examples will be *executed* (instead of just nicely formatted) and the resulting output and figures will be rendered as part of the tutorial/example.

7.1.2 What's new

Version 0.1.0 (2023-10-22)

- Initial release (initial commit: 2017-06-28).
- This version contains basic API codebase with functionalities to query locally hosted models and data. Some public (non-authenticated) queries from the AVNI server have also been implemented.
- Model and Surface Wave Reference Data Formats (RSDF) in both ASCII and HDF5 formats have been updated.

Authors

The core committer list for this release is the following:

- new contributor Raj Moulik
- new contributor Ross Maguire
- new contributor Rene Gassmoeller
- new contributor Chris Havlin

7.1.3 MyST Quick reference

Note: Everything in this file will look like garbage in a regular markdown viewer, like if you're viewing this on github. Viewing it on readthedocs will render everything properly.

Also: numref will not work in this file since its not part of the main filetree. But it will work in anything thats a part of a toctree. :::

Text

```
Put single asterisks around text you *want to italicize*, but double asterisks␣
↪around text you **want to bold**.
```

Put single asterisks around text you *want to italicize*, but double asterisks around text you **want to bold**.

Headings

```
# Heading 1
## Heading 2
### Heading 3
#### Heading 4
```

becomes:

Heading 1

Heading 2

Heading 3

Heading 4

All standalone .md files must start with a level 1 header, regardless of whether it is a chapter, section, or subsection. Subheaders within the same file can start with ## and so on.

For a section to be referenced, it must have a header tag like so:

```
(sec:myst-quickref)=
# MyST Quick reference
```

To refer to the section:

```
Include a section reference to {ref}`sec:myst-quickref` in a sentence like so.
```

Include a section reference to *MyST Quick reference* in a sentence like so.

Admonitions

```
:::{admonition} General admonition as warning
:class: warning

Text goes here.
```

Attention: This is an `attention` admonition.

Danger: This is a `danger` admonition.

Error: This is an `error` admonition.

Important: This is an `important` admonition.

Note: This is a `note` admonition.

Tip: This is a `tip` admonition.

Warning: This is a `warning` admonition.

See also:

This is a `seealso` admonition.

TODO

This is a custom `TODO` admonition.

```
Becomes:

:::{admonition} General admonition as warning :class: warning

Text goes here. :::

:::{attention} This is an `attention` admonition. :::

:::{danger} This is a `danger` admonition. :::

:::{error} This is an `error` admonition. :::

:::{important} This is an `important` admonition. :::

:::{note} This is a `note` admonition. :::

:::{tip} This is a `tip` admonition. :::

:::{warning} This is a `warning` admonition. :::

:::{seealso} This is a `seealso` admonition. :::

:::{admonition} TODO :class: error

This is a custom `TODO` admonition. :::

## Lists

### Itemized lists

```md
* Level 1
  * Level 2
    * Level 3
```
```

- Level 1
  - Level 2
    - Level 3

Definition lists

```
Term 1
: Definition of term 1

Term 2
: Definition of term 2
```

becomes:

Term 1 : Definition of term 1

Term 2 : Definition of term 2

Code blocks

C++

````
```{code-block} c++
---
caption: C++ code block.
emphasize-lines: 3-4
---
int
main(int argc, char* argv[]) {
    // Emphasized lines corresponding to body of main().
    return 0;
}
```
````

becomes:

Listing 7.3: C++ code block.

```c++
int
main(int argc, char* argv[]) {
    // Emphasized lines corresponding to body of main().
    return 0;
}
```

python

````
```{code-block} python
---
caption: Python code block.
---
def square(x):
    return x**2
```
````

becomes:

Listing 7.4: Python code block.

```python
def square(x):
    return x**2
```

Console

````
```{code-block} console
---
caption: Interactive shell.
---
$ ls
a b c
```
````

becomes:

Listing 7.5: Interactive shell.

```
$ ls
a b c
```

Bash

````
```{code-block} bash
---
caption: Bash code block.
---
# Comment
for i in "a b c"; do
  print $i
````

(continues on next page)
````

(continued from previous page)

````
done
```
````

becomes:

Listing 7.6: Bash code block.

```bash
# Comment
for i in "a b c"; do
  print $i
done
```

cfg

````
```{code-block} cfg
---
caption: Config code block.
---
# Comment
[pylithapp]
journal.info.problem = 1

[pylithapp.petsc]
ksp_rtol = 1.0e-3
```
````
````

Listing 7.7: Config code block.

```
# Comment
[pylithapp]
journal.info.problem = 1

[pylithapp.petsc]
ksp_rtol = 1.0e-3
```

Captions are optional.

Tables

````
```{table} Table caption
:name: tab:quickref
|             Header 1 |   Header 2    | Header 3       |
| -------------------: | :-----------: | :------------- |
|        right aligned |   centered    | left aligned   |
| more data, more data | yet more data | even more data |
```
````

becomes:

Table 7.1: Table caption

Header 1	Header 2	Header 3
right aligned	centered	left aligned
more data, more data	yet more data	even more data

Refer to it with the numref tag, like so:

```
Please see {numref}`tab:quickref`.
```

Please see Table 7.1.

Figure labels cannot contain more than one dash, so we use colons instead. This is likely a bug.

Note: The numref command doesn't work in this particular file because its not a part of the main file structure. It does work in the manual, as long as the file is in a toctree. :::

Figures

Format a figure like this:

```
```{figure-md} fig:quickref
<img src="../_static/logos/logo_avni_color_withname.svg" alt="Screenshot"  width=
↪"100%"/>

This is the figure caption.
```
```

to get this:

Fig. 7.1: This is the figure caption.

Put the filename and relative path in "img src", but leave the file extension as `.*` instead of .png or whatever. Allowable formats for figures include SVGs, PNGs, and JPGs, but not PDFs unfortunately.

Refer to it like:

```
Please see {numref}`fig:quickref`.
```

Please see Fig. 7.1.

Figure labels cannot contain more than one dash, so we use colons instead. This is likely a bug.

Sphinx does not support the use of subfigures. Combine any sub-figures you want to include into one file. We recommend inkscape for this purpose.

Figures need to have a caption or they will not be rendered. If you want to include figures without captions you can instead do:

```
```{image} ../_static/logos/logo_avni_color_withname.png
:alt: AVNI Logo
:width: 80%
:align: center
```
```

Math

Inline math works just like in latex. Use dollar signs to put stuff like $\eta_r(z)$ in a sentence. That's:

```
Use dollar signs to put stuff like $\eta_r(z)$ in a sentence.
```

Be aware that not all latex packages are known by Sphinx, so some commands will need to be reformatted (i.e., \num{4e5} needs to be rewritten as 4\times 10^{5}.)

For math blocks:

```
```{math}
:label: eqn:one:two
F = m a
```
```

becomes

$$F = ma \tag{7.1}$$

```
Refer to the labeled equation like {math:numref}`eqn:one:two`.
```

Refer to the labeled equation like (7.1). (The label is not necessary if the equation doesn't have one, in which case it would be formatted like):

```
```{math}
F = ma
```
```

The split environment is built into the math directive; which allows you to use one align operator (&).

```
```{math}
:label: eq:aligned
  -\nabla \cdot \left[2\eta
   \left(\varepsilon(\mathbf u) - \frac{1}{3}(\nabla \cdot \mathbf u)\mathbf 1\
↪right)\right]
   + \nabla p' &=  -\bar \alpha \bar\rho T' \mathbf g \qquad \textrm{in $\Omega$}
↪,  \\
  \nabla \cdot (\bar\rho \mathbf u) &= 0  \qquad  \textrm{in $\Omega$}.
```
```

becomes:

$$\begin{aligned} -\nabla \cdot \left[2\eta \left(\varepsilon(\mathbf{u}) - \frac{1}{3}(\nabla \cdot \mathbf{u})\mathbf{1}\right)\right] + \nabla p' &= -\bar{\alpha}\bar{\rho}T'\mathbf{g} \qquad \text{in } \Omega, \\ \nabla \cdot (\bar{\rho}\mathbf{u}) &= 0 \qquad \text{in } \Omega. \end{aligned} \tag{7.2}$$

If you require more than one alignment character you need to start and end an additional `aligned` environment. Be sure to add the same number of align operators (&) to each line and use \\ to denote a new line.

```
```{math}
:label: eq:aligned
\begin{aligned}
  -\nabla \cdot \left[2\eta
   \left(\varepsilon(\mathbf u) - \frac{1}{3}(\nabla \cdot \mathbf u)\mathbf 1\
↪right)\right]
   + \nabla p' &=  -\bar \alpha \bar\rho T' \mathbf g & \qquad  & \textrm{in $\
↪Omega$},  \\
  \nabla \cdot (\bar\rho \mathbf u) &= 0  & \qquad  & \textrm{in $\Omega$}.
\end{aligned}
```
```

becomes:

$$\begin{aligned} -\nabla \cdot \left[2\eta \left(\varepsilon(\mathbf{u}) - \frac{1}{3}(\nabla \cdot \mathbf{u})\mathbf{1}\right)\right] + \nabla p' &= -\bar{\alpha}\bar{\rho}T'\mathbf{g} && \text{in } \Omega, \\ \nabla \cdot (\bar{\rho}\mathbf{u}) &= 0 && \text{in } \Omega. \end{aligned} \tag{7.3}$$

Links

```
[Put a link in text](https://en.wikipedia.org/wiki/Gibbs_free_energy) or just␣
↪post the full link:
<https://en.wikipedia.org/wiki/Gibbs_free_energy>
```

becomes:

Put a link in text or just post the full link: https://en.wikipedia.org/wiki/Gibbs_free_energy

Citations

```
Traditional citation {cite}`Moulik:2014cr`.
Citation as noun {cite:t}`Moulik:2014cr`.
Use a comma for multiple, like {cite:t}`Moulik:2014cr,MoulikEtAl2021`.
```

becomes

Traditional citation [ME14]. Citation as noun Moulik and Ekström [ME14]. Use a comma for multiple, like Moulik and Ekström [ME14], Moulik *et al.* [MLR+21].

If the paper you wish to cite is not already in references.bib, you will need to add its bibtex entry. Please format the citation tag as:

- `firstauthor:year` for one author
- `firstauthor:secondauthor:year` for two authors
- `firstauthor:etal:year` for more than two authors

where `firstauthor` and `secondauthor` refer to the *last* names of the authors.

```
Here's how you put a footnote[^footnote1] in a sentence.
```

Here's how you put a footnote[1] in a sentence (see the bottom of this file).

See here for more

https://myst-parser.readthedocs.io/en/latest/syntax/syntax.html or https://jupyterbook.org/en/stable/reference/cheatsheet.html

footnote (again)

```
[^footnote1]: And here's what the footnote says (put this at the very bottom of␣
↪the document)
```

7.1.4 Fortran Guidelines

Coding style

When modifying an existing file, try to maintain consistency with its original style. If the code you add looks drastically different from the original code, it may be difficult for readers to follow. Try to avoid this. As a general guideline, we recommend the following code formatting style:

give space for breathing:

good

```
  dx = 0.5 * fac * (a - b)
```

bad

```
  dx=1/2*fac*(a-b)
```

Note that in performance critical sections, please use multiplication by 0.5 rather than divide by 2 for floating-points.

use consistent 2-space indents:

good

```
  if (i == 1) then
    print *,'great'
  endif
```

[1] And here's what the footnote says (put this at the very bottom of the document)

bad

```
  if(i == 1)then
        print *,'not so great'
  endif
```

start your code with an indent:

good

```
  subroutine vbspl()
  implicit none
  ..
```

bad

```
subroutine vbspl
  implicit none
  ..
```

The line beginning should only be used for *very important* sections, as it makes the line *very prominent* to read. For example, only use it for function descriptions, important comments, or file headers. For comments, see also next point...

exception, module definitions start at beginning:

good

```
module models
  integer :: count
end module
```

bad

```
  module models
  integer :: count
  end module
```

comment, comment, comment your code:

good

```
  ! gets associated values
  fg = vbspl(4,2)

  ! find values
  gt = fg
```

bad

```
  fg = vbspl(4,2)
```

Note we prefer indenting the comments as well to make it easier for reading the code, e.g., when inside multiple if-then statements. Putting the comment at the beginning breaks the flow.

comment, comment, comment your functions:

good

```
  subroutine blah_blah_function()

! calculates TI gradient based on a conjugate gradient method
!
! based on: Tarantola, inverse problem theory, 2005.
!                  section 6.22.7 conjugate directions, page 217.
!                  formula for alpha_n based on Polak & Ribiere (1969)
!
! note: we use a preconditioner F_0 = 1, thus lambda_n = gamma_n in (6.322)
!          and use gamma_n as the smoothed kernel (for bulk_c, bulk_betav,..).

  ..
```

bad

```
  subroutine blah_blah_function()

! computation step

  ..
```

Note that we haven't been very strict in adopting a doxygen-readable function declaration.

use double-colons for parameter declarations:

good

```
  integer :: i,j,k
```

bad

```
  integer i,j,k
```

use separators between subroutines:

good

```
  ..
  end subroutine

!
```

(continues on next page)

(continued from previous page)

```
!-------------------------------------------------------------
!

  subroutine get_color(icolor)
  ..
```

bad

```
  ..
end subroutine

subroutine get_color(icolor)
  ..
```

F2PY Troubleshooting

Warning: AVNI used F2PY to wrap legacy Python code by building from source code using the `numpy.distutils`. This requires restricting the versions to `numpy<=1.22` and `setuptools<60.0` in `setup.py` following this link. Ultimately, AVNI will be migrated to use the latest version of the `setuptools` package. :::

The purpose of the `F2PY` – Fortran to Python interface generator– utility is to provide a connection between Python and Fortran. `F2PY` is a part of NumPy (`numpy.f2py`) and also available as a standalone command line tool. We store legacy Fortran code in the `avni/f2py` folder. A way to troubleshoot `F2PY` issues involves the following steps:

1. Copy all of the source code to a single directory
2. Make sure that the code will actually compile without f2py (either with a makefile or with a simple script)
3. Use f2py to create a python signature file (`.pyf` file).

See the makefile attached below which will compile all of the code, and generate the file `avni_forward.pyf`. The `.pyf` file can be used to figure out what problems are going on. In our case, there were a number of subroutines that were not being properly translated to the signature file. These subroutines are called "unknown_subroutine" in the signature file. For example, the signature file complained about a subroutine in `rayseq.f`:

Listing 7.8: Fortran code block giving an unknown tag issue.

```
subroutine unknown_subroutine ! in rayseq.f
    integer :: iprtlv
    common /plevel/ iprtlv
end subroutine unknown_subroutine
```

Since it wasn't really giving us any reasons why the subroutine was unknown, it was a little hard to figure out how to fix. It turns out the problem (in this case) was that the subroutine definition in rayseq.f exceeded the fortran character limit of 72. So to fix it, we just added a continuation line to the source code:

```
diff rayseq.f ../ALLCODES/rayseq.f
1c1,2
<       SUBROUTINE RAYSEQ (NP,NS,NN,ISS,IRR,ICNSx,ICNR,ICN,ISEQ,idirseg,NSEG)
---
>       SUBROUTINE RAYSEQ (NP,NS,NN,ISS,IRR,ICNSx,ICNR,ICN,ISEQ,
>      #idirseg,NSEG)
```

There were a number of other source files that we had to modify, but as far as we remember, the only changes we made were to fix character limit issues. Once there were no more problems in the python signature file, we copied the source codes back to their respective directories, and everything worked fine with `pip`. We can't say that this method will work for troubleshooting other `F2PY` issues.

```
# makefile
#LOBJ = ./objects
LOBJ = ./
FFLAGS = -ffixed-line-length-none -fPIC -O3 -fbounds-check
SRCS = tt_predict.f acosd.f asind.f atand.f bffi.f bullen.f cagcrays_pm.f␣
↪closfl.f conv2geocen.f cosd.f dacosd.f dbsplrem.f dcosd.f ddelaz.f ddelazgc.
↪f \
    delaz.f delazgc.f DiffTime.f drspleder.f drspledr.f drsple.f drspln.f␣
↪dsind.f dxt.f evemb.f evemdr.f evemell.f evem.f fdrays.f fqs.f \
    get_branch.f getcmt5.f getcmtbyname.f getcmtdatetime.f getflpos.f get_
↪ishflag.f getptab.f julday.f legndr.f lpyr.f monday.f openfl.f \
    pdaz.f prange.f psvrayin.f qtauall.f qtau.f qtauzero.f rayseq.f rademb.f␣
↪rademfl.f readnbn.f readptab.f sind.f swap4of4.f tadder.f \
    tand.f vbspl.f wgint.f wgray.f ylm.f
OBJS = tt_predict.o acosd.o asind.o atand.o bffi.o bullen.o cagcrays_pm.o␣
↪closfl.o conv2geocen.o cosd.o dacosd.o dbsplrem.o dcosd.o ddelaz.o ddelazgc.
↪o \
    delaz.o delazgc.o DiffTime.o drspleder.o drspledr.o drsple.o drspln.o␣
↪dsind.o dxt.o evemb.o evemdr.o evemell.o evem.o fdrays.o fqs.o \
    get_branch.o getcmt5.o getcmtbyname.o getcmtdatetime.o getflpos.o get_
↪ishflag.o getptab.o julday.o legndr.o lpyr.o monday.o openfl.o \
    pdaz.o prange.o psvrayin.o qtauall.o qtau.o qtauzero.o rayseq.o rademb.o␣
↪rademfl.o readnbn.o readptab.o sind.o swap4of4.o tadder.o \
    tand.o vbspl.o wgint.o wgray.o ylm.o
```

```
F2PY = f2py
F77 = gfortran
F90 = fortran
CC = gcc
avni_forward.so: $(OBJS) avni_forward.pyf
    $(F2PY) -c avni_forward.pyf $(OBJS)
avni_forward.pyf: $(SRCS)
    $(F2PY) --overwrite-signature -m avni_forward -h avni_forward.pyf $(SRCS)
clean:
    $(RM) $(LOBJ)/*.o $(LOBJ)/*.pyf ./avni_forward.so
$(LOBJ)/%.o: %.f
    $(F77) $(FFLAGS) -c $*.f -o $(LOBJ)/$*.o
```

7.1.5 Versioning conventions

Version tagging is based on a `major.minor.patch` numbering scheme following Semantic Versioning Specification (SemVer).

- The *major* number is incremented after extensive rewrites or substantial overhauls that may significantly affect workflow integration or user experience. Major updates undergo extensive testing, but subsequent patch updates may still be needed.
- The *minor* number is incremented whenever modifications to the source code are significant enough to require updates to parameter files or other input files.
- The *patch* number is used to denote bug fixes or other minor revisions. Updates to parameter files or other input files are not required.

Tagged releases are fairly infrequent because the associated testing is very time consuming.

AVNI provides free web-based and backend code access to tools, techniques, models and data related to global Earth sciences. It is a software ecosystem for analyzing and interpreting planetary models and data sets that were initially designed for the three-dimensional reference Earth model project REM3D Website. The codes are primarily written in Python with interfaces to legacy routines in C and Fortran. Some installation files as well as applets and API access require registration on the Register AVNI link.

- Open-source Python package with APIs to handle intensive queries License GNU GPL v3
- Introduce HDF5 storage formats for planetary models and processed seismic data
- Interactive web-based visualization tools for data and model exploration
- Formulate and benchmark solvers for rapid data validation of models

Share this project on Twitter: Tweet

AVNI is a CIG affiliated project 


```
import avni
print(avni.__version__)
```

```
0.1.0
```

CHAPTER

EIGHT

LINKS

Website	AVNI Website	AVNI GitHub
Deployment	pypi v0.1.0	
License	License GNU GPL v3	
Community	GitHub Discussions	GitHub Issues

CHAPTER

NINE

APPLICATIONS

PYTHON MODULE INDEX

a

INDEX