nc5ng-python Documentation

Release 0.0.4

Andrey Shmakov

Aug 12, 2018

Contents

`nc5ng-python` is a project providing python libraries for import, analysis, generation, plotting, export, filitering, and manipulation of geodetic datum transformation data.

The source data for transformations come from the United States National Geodetic Survey *NADCON5* data and `Fortran` processing programs linked into python in the nc5ng.org project `nadcon5-ng` (Home Page) . Transformation grids can be provided from either source or built yourself (See: *Install Conversion Data*)

CHAPTER 1

Installation

Installation instructions for nc5ng-python

1.1 System Installation

`nc5ng` offers common installation options for python distributions

Note: It is recommended to use a python virtual environment (`virtualenv`) to install this package and dependencies. Installing directly to a system python distribution can break certain system packages.

1.1.1 Requirements

- Python 3.x - `pip` and `virtualenv` (*Recommended*)
- Fortran (`gfortran`)
- GMT (4.x-6.x)
- GMT/Python (*Optional*) - GMT6.0 Required for ‘‘GMT/Python‘‘(*Development Release Only*)

1.1.2 Python Package Index

Released versions of `nc5ng` can be installed directly from the PyPi by using *pip*

```
pip install nc5ng
```

Specific versions can be installed by specifying `nc5ng==VERSION`

1.1.3 Development Versions

`pip` can be used to install development versions of python packages by specifying the git repository, including branch or commit

```
pip install -U git+git://github.com/nc5ng/nc5ng-python@stable
pip install -U git+git://github.com/nc5ng/nc5ng-
↪python@8f482ba1ce6484ab18e6ccd88d6e251655cd61f2
```

Local Copies can be installed in editable mode for development

```
git clone git@github.com/nc5ng/nc5ng-python
pip install -U -e ./nc5ng-python
```

1.2 Docker Deployment

Docker images are provided to use nc5ng through container virtualization.

Pre-configured images are available from the nc5ng-docker project (Dockerhub), and base images with gmt tools are available from the gmt-docker project (Dockerhub).

For convenience, the docker image `nc5ng/nc5ng` contains a pre-compiled conversion for testing.

For more information on deployment and use please see the docker project pages.

1.2.1 Requirements

- `docker`
- `docker-compose` (*Optional*)

1.3 Install Conversion Data

Raw Source data is provided directly from the *nc5ng-core* package. However conversion data must either be built or downloaded seperately.

Conversion data are released publically on the nadcon5-ng github or upstream via National Geodetic Survey. For compilation please see the source library docs , project homepage, and github

Conversion data can be placed anywhere and is referenced by configration arguments used to load data.

CHAPTER 2

Reference Manual

2.1 *nc5ng.nc5data* Conversion Data API

PyPi Package: `nc5ng-core`

Data Wrapper API for *nadcon5-ng* source and output data

2.1.1 Conversions

Conversions are used to aggregate the `NADCON5.0` source and output data

class nc5ng.nc5data.**Conversion**(*region*, *old_datum*, *new_datum*, *grid_spacing='900'*, ***kwargs*)

A Conversion agregates all parts of a NADCON5 Datum Conversion and serves as the primary user interface into the dataset

A Conversion is created based on region, source, target datum, and gridspacing the same parameters that go into the data build pipeline

Conversions maintain a large set of data in memory (when loaded) and have accessors for data

Creating a conversion is simple

```
c = Conversion('conus', 'ussd', 'nad27')``
```

The output data of a conversion can be accessed directly by output prefix

```
v1 = c.output_data['vmacdlat']
v2 = c.output_data['vmacdlon']
```

Output data is indexed by PID, to extract all PID's with lat and lon conversions in this data set

```
shared_pids = [ i for i in v1 if i in v2 ]
```

All point data for a single point can be examined directly, including its source data

```
point = v[shared_pids[0]]
point.source
```

class ConversionInput(*region*, *old_datum*, *new_datum*, ***kwargs*)
ConversionInput holds all input data associated with a conversion

On construction attempts to load all source data for a given conversion set

class ConversionOutput(*region*, *old_datum*, *new_datum*, *grid_spacing*, *load_all=False*, ***kwargs*)
ConversionOutput holds all output data associated with a conversion

On construction attempts to load all known valid output files by iterating through known combinations of output types

Par region The `RegionData` index to plot (e.g. `'conus'`)

Par old_datum The source datum for conversion (e.g. `'ussd'`)

Par new_datum The target datum for conversion (e.g. `'nad27'`)

Par grid_spacing The conversion grid spacing (e.g. `900`)

2.1.2 Static NADCON Data Services

Services to access certain static nadcon input data e.g. Grid Bound points

2.1.3 *nadcon5-ng* Data Types

DataPoint

DataPoint is the base DataPointType hierarchy for nadcon5-ng conversions

Other parts of this library may use a different hierarchy, but share a type generator

class nc5ng.nc5data.nadcon5_types.**DataPoint**(**args*, ***kwargs*)
Base Data Point Type for *nadcon5-ng* conversion data

Type Mixins

class nc5ng.nc5data.nadcon5_types.**MetaMixin**
Mixin base type for standard meta information

class nc5ng.nc5data.nadcon5_types.**DataContainerMixin**(**args*, ***kwargs*)
Mixin type for containers of DataPoints

class nc5ng.nc5data.nadcon5_types.**GMTMixin**

class nc5ng.nc5data.nadcon5_types.**GMTMetaMixin**

2.1.4 *nadcon5-ng* File Parsers

File Parsers for import nadcon5-ng source and output data

FileParers implementing the BaseFileParser API use a calling convention to

class nc5ng.nc5data.nadcon5_files.**ControlFileParser**(*control_dir='/home/docs/checkouts/readthedocs.org/user_packages/nc5ng/nc5data/data/Control'*)

class nc5ng.nc5data.nadcon5_files.**CoverageFileParser**(***kwargs*)

class nc5ng.nc5data.nadcon5_files.**GridParamFileParser**(*grid_file='/home/docs/checkouts/readthedocs.org/user packages/nc5ng/nc5data/data/Data/grid.parameters'*, **regions*)

class nc5ng.nc5data.nadcon5_files.**InFileParser**(*fdir='/home/docs/checkouts/readthedocs.org/user_builds/nc5ng/ packages/nc5ng/nc5data/data/InFiles'*, *ffile=None*)

class nc5ng.nc5data.nadcon5_files.**VectorFileParser**(***kwargs*)

class nc5ng.nc5data.nadcon5_files.**WorkEditsFileParser**(*ffile='/home/docs/checkouts/readthedocs.org/user_bui packages/nc5ng/nc5data/data/Work/workedits'*)

2.2 *nc5ng.gmt* GMT Wrapper API

PyPi Package: `nc5ng-common`

GMT Wrapper Library and Convenience Methods for `nc5ng`

2.2.1 GMT Options

GMTOptions form a loose wrapper on the GMT command line arguments and GMT/Python shorthands

It is a subtype of `dict` but has additional properties for creating appropirate GMT arguments

class nc5ng.gmt.**GMTOptions**(*projection='M10.0i', region=[240, 190, 30, 80], frame=True, insert=None, border=None, scale=None, dir_rose=None, mag_rose=None, logo=False, area_thresh=None, lakes=None, resolution='c', land=None, rivers=None, borders=None, water=None, shorelines=1, linear_lines=False, cpt=None, offset=None, errors=None, color=None, symbol=None, pen=None, basemap=None, coast=None, plot=None, **kwargs*)

GMTOptions are decorated dictionaries to wrap GMT/Python Keyword options

GMTOptions takes keywords or other dictionary to construct gmt plot options for *basemap*, *coast*, *plot* via properties *GMTOptions.basemap*, *.coast .plot*

GMT Options can be combined (copy-combine) by calling one with the other

p1 = GMTOptions(lakes=0) p2 = GMTOptions(**PLOT_OPTS['default'])

p3 = p2(p1) # override default by turning off lakes

> **Warning:** GMTOptions **do not track single letter GMT Arguments**, because ambiguous double-mappings exist (e.g. *-A* means different things to *coast* vs *plot*).
>
> Instead, single letter options can be overriden as keyword arguments to GMTPlotter

2.2.2 GMT Plotter

GMTPlotter forms a loose wrapper around `gmt.Figure` object, that constructs plots from embedded gmt options inside `nc5ng` objects (meta-api, `object.gmt_meta`)

class nc5ng.gmt.plotter.**GMTPlotter**(*base_plot_options={'__class__': <class 'nc5ng.gmt.options.GMTOptions'>, 'area_thresh': 1200, 'basemap': None, 'border': None, 'borders': ['1', '2'], 'coast': None, 'color': None, 'cpt': None, 'dir_rose': None, 'errors': None, 'frame': True, 'insert': None, 'kwargs': {}, 'lakes': None, 'land': None, 'linear_lines': False, 'logo': False, 'mag_rose': None, 'offset': None, 'pen': None, 'plot': None, 'projection': 'M10.0i', 'region': [240, 190, 30, 80], 'resolution': 'fine', 'rivers': None, 'scale': None, 'self': {'plot': None, '__class__': <class 'nc5ng.gmt.options.GMTOptions'>, 'color': None, 'cpt': None, 'scale': None, 'borders': ['1', '2'], 'land': None, 'border': None, 'rivers': None, 'symbol': None, 'region': [240, 190, 30, 80], 'logo': False, 'kwargs': {}, 'coast': None, 'linear_lines': False, 'dir_rose': None, 'lakes': None, 'area_thresh': 1200, 'basemap': None, 'errors': None, 'self': {...}, 'insert': None, 'pen': None, 'mag_rose': None, 'resolution': 'fine', 'frame': True, 'water': 'lightblue', 'offset': None, 'shorelines': 1, 'projection': 'M10.0i'}, 'shorelines': 1, 'symbol': None, 'water': 'lightblue'}*)

Wrapper for GMT/Python Plotter

static plot_conversion(*conversion*, *coverage='all'*, *vector='all'*, *plotter=None*, ***kwargs*)

Static Method to Plot an `nc5ng.nc5data.Conversion`

Plotting options are Applied in the order

1. GMTPlotter configured or default options
2. Conversion configured options `Conversion.gmt_meta`
3. Data Set Options `PointData.gmt_meta`
4. Keyword overrides (full option name)
5. Keyowrd overrides (single letter GMT Style arguments)

Parameters

- **conversion** – Conversion data
- **coverage** – Coverage files (no file extension), to plot. Can be list of names (`coverage = ['cvacdlat','cvacdlon',]`), single file name (`coverage = 'cvacdlat'`), `None`, or `'all'`
- **vector** – Vector files (no file extension) to plot. Can be list of names, single file name, `None`, or `'all'`.
- **plotter** – plotter to use, if `None` one will be created and returned

2.3 *nc5ng.types* Core Types

PyPi Package: `nc5ng-common`

Common Base Types, Type Generators and Abstract Base Types for `nc5ng`

2.3.1 DataPoint Types

Base Datapoint Types

`DataPointType` Metaclass

class nc5ng.types.**DataPointType**(*name*, *bases*, *namespace*)

Metaclass for DataPoints, defines class creation and class/hierarchy member variables

The meta-class in part, hides some of the more rote requirements of our datapoint type from the actual object hierarchy

This allows some magic like run-time casting to the correct datapoint without knowing the type specifically, and a persisitent library-wide memory backed database for quick retrieval and to minimize replication

To understand the meta class, there are a number of tutorials available online, roughly speaking this class is what is used to "create" the DataPoint class, and allows us to manipulate the class without needing any implementation details.

Defining a new DataPointType Hierarchy simply requires using this class as the metaclass

class NewDataPoint(metaclass=DataPointType): pass

By creating a new DataPointType, the following changes will be done to the final class

- The type will have a database (dictionary) of types registered with the base class
- Each type and subtype will have a point container (default set) created
- The shorthand name will be generated from the class name
- Instance Creation will be overidden and allow creation of any other data type by specifting the type shorthand as an argument

Class Configuration:

Each new type has some meta-configuration available

1. To override data point registration - Create a *@classmethod __register__(cls, point)* to overide how a new point is registered/saved - Create a class member *_point_store* to change the underlying storage type (from set)
2. Override shorthand name by specific '_type_shorthand' explicitly in the class

Any class that uses this type as a metaclass will be registered

point_database
Return the Root Database of all DataPoints in this Hierarchy

point_store
Return the type-specific Point Buffer

type_shorthand
Get the class shorthand name

2.3.2 Data Parsers

Base Parser Types for *nc5ng* submodules

class nc5ng.types.parsers.**BaseFileParser**(*parser=None*, *fdir=None*, *ffile=None*)

Base Class for File Parsers

class nc5ng.types.parsers.**FortranFormatFileParser**(*fformat=None*, *ffilter=None*)
FileParser for Fortran Fixed Format Files

Parameters

- **fformat** – fortran format string
- **ffilter** – file pre-filter (exclude/include lines)

class nc5ng.types.parsers.**IndexedFortranFormatFileParser**(**args*, ***kwargs*)
Extentsion for FortranFileParser that allows classes to switch between pre-registered formats by index

Parameters

- **args** – Either list of `[format1,filter1],[format2,filter2],...` or serial list `format1, filter1, format2, filter2, ...`
- **kwargs** – Keyword argument dictionary `index:[format,filter]`, dictionary key used for indexing file parser

CHAPTER 3

Useful Links

3.1 Information

- *nc5ng.org* Organization Homepage : https://www.nc5ng.org
- `nc5ng-python` Project Homepage : https://www.nc5ng.org/projects/nc5ng-python

3.2 Related Projects

- `nadcon5-ng` Project Homepage : https://www.nc5ng.org/projects/nadcon5-ng
- `nc5ng-python-common` Project Homepage : https://www.nc5g.org/projects/nc5ng-python-common

3.3 Development

- `nc5ng` Organization Github : https://github.com/nc5ng
- `nc5ng-python` Project Github : https://github.com/nc5ng/nc5ng-python

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index