Apex Python library

Release 1.0.1

Feb 28, 2018

Contents

CHAPTER 1

Overview

This is a Python wrapper for the Apex fortran library by Emmert et al. [2010][1], which allows converting between geodetic, modified apex, and quasi-dipole coordinates as well as getting modified apex and quasi-dipole base vectors (Richmond [1995][2]). MLT calculations are also included. The package is free software (MIT license).

1.1 Quick start

Install (requires NumPy):

```
pip install apexpy
```

Conversion is done by creating an `Apex` object and using its methods to perform the desired calculations. Some simple examples:

```
>>> from apexpy import Apex
>>> from __future__ import print_function
>>> A = Apex(date=2015.3)  # datetime objects are also supported
>>> # geo to apex, scalar input
>>> mlat, mlon = A.convert(60, 15, 'geo', 'apex', height=300)
>>> print("{:.12f}, {:.12f}".format(mlat, mlon))
57.469573974609, 93.633583068848
>>> # apex to geo, array input
>>> glat, glon = A.convert([90, -90], 0, 'apex', 'geo', height=0)
>>> print(["{:.12f}, {:.12f}".format(ll, glon[i]) for i,ll in enumerate(glat)])
['83.099594116211, -84.594589233398', '-74.388267517090, 125.714927673340']
>>> # geo to MLT
>>> import datetime as dt
```

[1] Emmert, J. T., A. D. Richmond, and D. P. Drob (2010), A computationally compact representation of Magnetic-Apex and Quasi-Dipole coordinates with smooth base vectors, J. Geophys. Res., 115(A8), A08322, doi:10.1029/2010JA015326.

[2] Richmond, A. D. (1995), Ionospheric Electrodynamics Using Magnetic Apex Coordinates, Journal of geomagnetism and geoelectricity, 47(2), 191–212, doi:10.5636/jgg.47.191.

```
>>> mlat, mlt = A.convert(60, 15, 'geo', 'mlt', datetime=dt.datetime(2015, 2, 10, 18,␣
↪0, 0))
>>> print("{:.12f}, {:.12f}".format(mlat, mlt))
56.590423583984, 19.108103879293
>>> # can also convert magnetic longitude to mlt
>>> mlt = A.mlon2mlt(120, dt.datetime(2015, 2, 10, 18, 0, 0))
>>> print("{:.2f}".format(mlt))
20.89
```

If you don't know or use Python, you can also use the command line. See details in the full documentation.

1.2 Documentation

https://apexpy.readthedocs.org/

1.3 References

1.4 Badges

docs	
tests	
package	

CHAPTER 2

Installation

This package requires NumPy, which you can install alone or as a part of SciPy. Some Python distributions come with NumPy/SciPy pre-installed. For Python distributions without NumPy/SciPy, Windows/Mac users should install pre-compiled binaries of NumPy/SciPy, and Linux users may have NumPy/SciPy available in their repositories.

When you have NumPy, install this package at the command line using `pip`[1]:

```
pip install apexpy
```

On Linux this will build apexpy from source, which requires a fortran compiler such as gfortran.

The package has been tested with the following setups (others might work, too):

- Windows (32/64 bit Python), Linux (64 bit), and Mac
- Python 2.7, 3.3, 3.4 (and 3.5 on Linux/Mac[2])

[1] pip is included with Python 2 from v2.7.9 and Python 3 from v3.4. If you don't have pip, get it here.

[2] I do not know of any way to compile the Fortran extension on Windows in a manner that is compatible with Python 3.5. If you get it working, let me know!

CHAPTER 3

Usage examples

3.1 Python library

For full documentation of the functions, see the reference for *apexpy.Apex*.

```
>>> from apexpy import Apex
>>> from __future__ import print_function
>>> A = Apex(date=2015.3)  # datetime objects are also supported
>>> # geo to apex, scalar input
>>> mlat, mlon = A.convert(60, 15, 'geo', 'apex', height=300)
>>> print("{:.12f}, {:.12f}".format(mlat, mlon))
57.469573974609, 93.633583068848
>>> # apex to geo, array input
>>> glat, glon = A.convert([90, -90], 0, 'apex', 'geo', height=0)
>>> print(["{:.12f}, {:.12f}".format(ll, glon[i]) for i,ll in enumerate(glat)])
['83.099594116211, -84.594589233398', '-74.388267517090, 125.714927673340']
>>> # geo to MLT
>>> import datetime as dt
>>> mlat, mlt = A.convert(60, 15, 'geo', 'mlt', datetime=dt.datetime(2015, 2, 10, 18,␣
↪0, 0))
>>> print("{:.12f}, {:.12f}".format(mlat, mlt))
56.590423583984, 19.108103879293
>>> # can also convert magnetic longitude to mlt
>>> mlt = A.mlon2mlt(120, dt.datetime(2015, 2, 10, 18, 0, 0))
>>> print("{:.2f}".format(mlt))
20.89
```

3.2 Command-line interface

The Python package also installs a command called `apexpy` which allows using the *convert()* method from the command line. The command-line interface allows you to make use of the Python library even if you don't know or

use Python. See the reference for *Command-line interface* for a list of arguments to the commands. Below are some simple usage examples.

Produce a file called e.g. `input.txt` with the input latitudes and longitudes on each row separated by whitespace:

```
# lat lon
# comment lines like these are ignored
60 15
61 15
62 15
```

To convert this to apex using the magnetic field model for the date 2015-01-01 using a height of 300 km, run the command `apexpy geo apex 20150101 --height 300 -i input.txt -o output.txt` (in this specific example you could also just use `2015` or `201501` for the date). The output file will look like this:

```
57.46954727 93.63981628
58.52270126 94.04476166
59.57146454 94.47725677
```

Alternatively, you can skip the files and just use command-line piping:

```
$ echo 60 15 | apexpy geo apex 2015 --height 300
57.46954727 93.63981628
```

MLT conversion works in much the same way, but requires both date and time (`YYYYMMDDHHMMSS`). For example, if you want to find the MLT (and the apex latitude) at geodetic coordinates (60, 15) for midnight on the day 2015-01-01, run `echo 60 15 | apexpy geo mlt 20150101000000`. The output will look like this:

```
56.59033585 1.03556417
```

CHAPTER 4

Reference

4.1 apexpy.Apex

The `Apex` class is used for all the main functionality (converting between coordinate systems, field line mapping, and calculating base vectors).

class apexpy.**Apex**(*date=None*, *refh=0*, *datafile=None*)

Performs coordinate conversions, field-line mapping and base vector calculations.

Parameters

- **date** (float (decimal year) or `datetime.date` or `datetime.datetime`, optional) – Determines which IGRF coefficients are used in conversions. Uses current date as default.
- **refh** (*float, optional*) – Reference height in km for apex coordinates (the field lines are mapped to this height)
- **datafile** (*str, optional*) – Path to custom coefficient file

Variables

- **year** (*float*) – Decimal year used for the IGRF model
- **refh** (*float*) – Reference height in km for apex coordinates
- **datafile** (*str*) – Path to coefficient file

Notes

The calculations use IGRF-12 with coefficients from 1900 to 2020[1].

[1] Thébault, E. et al. (2015), International Geomagnetic Reference Field: the 12th generation, Earth, Planets and Space, 67(1), 79, doi:10.1186/s40623-015-0228-9.

References

convert (*lat*, *lon*, *source*, *dest*, *height=0*, *datetime=None*, *precision=1e-10*, *ssheight=318550*)
Converts between geodetic, modified apex, quasi-dipole and MLT.

Parameters

- **lat** (*array_like*) – Latitude
- **lon** (*array_like*) – Longitude/MLT
- **source** (*{'geo', 'apex', 'qd', 'mlt'}*) – Input coordinate system
- **dest** (*{'geo', 'apex', 'qd', 'mlt'}*) – Output coordinate system
- **height** (*array_like, optional*) – Altitude in km
- **datetime** (`datetime.datetime`) – Date and time for MLT conversions (required for MLT conversions)
- **precision** (*float, optional*) – Precision of output (degrees) when converting to geo. A negative value of this argument produces a low-precision calculation of geodetic lat/lon based only on their spherical harmonic representation. A positive value causes the underlying Fortran routine to iterate until feeding the output geo lat/lon into geo2qd (APXG2Q) reproduces the input QD lat/lon to within the specified precision (all coordinates being converted to geo are converted to QD first and passed through APXG2Q).
- **ssheight** (*float, optional*) – Altitude in km to use for converting the subsolar point from geographic to magnetic coordinates. A high altitude is used to ensure the subsolar point is mapped to high latitudes, which prevents the South-Atlantic Anomaly (SAA) from influencing the MLT.

Returns

- **lat** (*ndarray or float*) – Converted latitude (if converting to MLT, output latitude is apex)
- **lat** (*ndarray or float*) – Converted longitude/MLT

geo2apex (*glat*, *glon*, *height*)
Converts geodetic to modified apex coordinates.

Parameters

- **glat** (*array_like*) – Geodetic latitude
- **glon** (*array_like*) – Geodetic longitude
- **height** (*array_like*) – Altitude in km

Returns

- **alat** (*ndarray or float*) – Modified apex latitude
- **alon** (*ndarray or float*) – Modified apex longitude

apex2geo (*alat*, *alon*, *height*, *precision=1e-10*)
Converts modified apex to geodetic coordinates.

Parameters

- **alat** (*array_like*) – Modified apex latitude
- **alon** (*array_like*) – Modified apex longitude
- **height** (*array_like*) – Altitude in km

- **precision** (*float, optional*) – Precision of output (degrees). A negative value of this argument produces a low-precision calculation of geodetic lat/lon based only on their spherical harmonic representation. A positive value causes the underlying Fortran routine to iterate until feeding the output geo lat/lon into geo2qd (APXG2Q) reproduces the input QD lat/lon to within the specified precision.

Returns

- **glat** (*ndarray or float*) – Geodetic latitude
- **glon** (*ndarray or float*) – Geodetic longitude
- **error** (*ndarray or float*) – The angular difference (degrees) between the input QD coordinates and the qlat/qlon produced by feeding the output glat and glon into geo2qd (APXG2Q)

geo2qd(*glat*, *glon*, *height*)
Converts geodetic to quasi-dipole coordinates.

Parameters

- **glat** (*array_like*) – Geodetic latitude
- **glon** (*array_like*) – Geodetic longitude
- **height** (*array_like*) – Altitude in km

Returns

- **qlat** (*ndarray or float*) – Quasi-dipole latitude
- **qlon** (*ndarray or float*) – Quasi-dipole longitude

qd2geo(*qlat*, *qlon*, *height*, *precision=1e-10*)
Converts quasi-dipole to geodetic coordinates.

Parameters

- **qlat** (*array_like*) – Quasi-dipole latitude
- **qlon** (*array_like*) – Quasi-dipole longitude
- **height** (*array_like*) – Altitude in km
- **precision** (*float, optional*) – Precision of output (degrees). A negative value of this argument produces a low-precision calculation of geodetic lat/lon based only on their spherical harmonic representation. A positive value causes the underlying Fortran routine to iterate until feeding the output geo lat/lon into geo2qd (APXG2Q) reproduces the input QD lat/lon to within the specified precision.

Returns

- **glat** (*ndarray or float*) – Geodetic latitude
- **glon** (*ndarray or float*) – Geodetic longitude
- **error** (*ndarray or float*) – The angular difference (degrees) between the input QD coordinates and the qlat/qlon produced by feeding the output glat and glon into geo2qd (APXG2Q)

apex2qd(*alat*, *alon*, *height*)
Converts modified apex to quasi-dipole coordinates.

Parameters

- **alat** (*array_like*) – Modified apex latitude

- **alon** (*array_like*) – Modified apex longitude
- **height** (*array_like*) – Altitude in km

Returns

- **qlat** (*ndarray or float*) – Quasi-dipole latitude
- **qlon** (*ndarray or float*) – Quasi-dipole longitude

Raises `ApexHeightError` – if *height* > apex height

qd2apex (*qlat*, *qlon*, *height*)
Converts quasi-dipole to modified apex coordinates.

Parameters

- **qlat** (*array_like*) – Quasi-dipole latitude
- **qlon** (*array_like*) – Quasi-dipole longitude
- **height** (*array_like*) – Altitude in km

Returns

- **alat** (*ndarray or float*) – Modified apex latitude
- **alon** (*ndarray or float*) – Modified apex longitude

Raises `ApexHeightError` – if apex height < reference height

mlon2mlt (*mlon*, *datetime*, *ssheight=318550*)
Computes the magnetic local time at the specified magnetic longitude and UT.

Parameters

- **mlon** (*array_like*) – Magnetic longitude (apex and quasi-dipole longitude are always equal)
- **datetime** (`datetime.datetime`) – Date and time
- **ssheight** (*float, optional*) – Altitude in km to use for converting the subsolar point from geographic to magnetic coordinates. A high altitude is used to ensure the subsolar point is mapped to high latitudes, which prevents the South-Atlantic Anomaly (SAA) from influencing the MLT.

Returns **mlt** (*ndarray or float*) – Magnetic local time [0, 24)

Notes

To compute the MLT, we find the apex longitude of the subsolar point at the given time. Then the MLT of the given point will be computed from the separation in magnetic longitude from this point (1 hour = 15 degrees).

mlt2mlon (*mlt*, *datetime*, *ssheight=318550*)
Computes the magnetic longitude at the specified magnetic local time and UT.

Parameters

- **mlt** (*array_like*) – Magnetic local time
- **datetime** (`datetime.datetime`) – Date and time

- **ssheight** (*float, optional*) – Altitude in km to use for converting the subsolar point from geographic to magnetic coordinates. A high altitude is used to ensure the subsolar point is mapped to high latitudes, which prevents the South-Atlantic Anomaly (SAA) from influencing the MLT.

Returns **mlon** (*ndarray or float*) – Magnetic longitude [0, 360) (apex and quasi-dipole longitude are always equal)

Notes

To compute the magnetic longitude, we find the apex longitude of the subsolar point at the given time. Then the magnetic longitude of the given point will be computed from the separation in magnetic local time from this point (1 hour = 15 degrees).

map_to_height (*glat*, *glon*, *height*, *newheight*, *conjugate=False*, *precision=1e-10*)

Performs mapping of points along the magnetic field to the closest or conjugate hemisphere.

Parameters

- **glat** (*array_like*) – Geodetic latitude
- **glon** (*array_like*) – Geodetic longitude
- **height** (*array_like*) – Source altitude in km
- **newheight** (*array_like*) – Destination altitude in km
- **conjugate** (*bool, optional*) – Map to *newheight* in the conjugate hemisphere instead of the closest hemisphere
- **precision** (*float, optional*) – Precision of output (degrees). A negative value of this argument produces a low-precision calculation of geodetic lat/lon based only on their spherical harmonic representation. A positive value causes the underlying Fortran routine to iterate until feeding the output geo lat/lon into geo2qd (APXG2Q) reproduces the input QD lat/lon to within the specified precision.

Returns

- **newglat** (*ndarray or float*) – Geodetic latitude of mapped point
- **newglon** (*ndarray or float*) – Geodetic longitude of mapped point
- **error** (*ndarray or float*) – The angular difference (degrees) between the input QD coordinates and the qlat/qlon produced by feeding the output glat and glon into geo2qd (APXG2Q)

Notes

The mapping is done by converting glat/glon/height to modified apex lat/lon, and converting back to geographic using newheight (if conjugate, use negative apex latitude when converting back)

map_E_to_height (*alat*, *alon*, *height*, *newheight*, *E*)

Performs mapping of electric field along the magnetic field.

It is assumed that the electric field is perpendicular to B.

Parameters

- **alat** (*(N,) array_like or float*) – Modified apex latitude
- **alon** (*(N,) array_like or float*) – Modified apex longitude

- **height** (*(N,) array_like or float*) – Source altitude in km
- **newheight** (*(N,) array_like or float*) – Destination altitude in km
- **E** (*(3,) or (3, N) array_like*) – Electric field (at *alat*, *alon*, *height*) in geodetic east, north, and up components

Returns **E** (*(3, N) or (3,) ndarray*) – The electric field at *newheight* (geodetic east, north, and up components)

map_V_to_height(*alat*, *alon*, *height*, *newheight*, *V*)

Performs mapping of electric drift velocity along the magnetic field.

It is assumed that the electric field is perpendicular to B.

Parameters

- **alat** (*(N,) array_like or float*) – Modified apex latitude
- **alon** (*(N,) array_like or float*) – Modified apex longitude
- **height** (*(N,) array_like or float*) – Source altitude in km
- **newheight** (*(N,) array_like or float*) – Destination altitude in km
- **V** (*(3,) or (3, N) array_like*) – Electric drift velocity (at *alat*, *alon*, *height*) in geodetic east, north, and up components

Returns **V** (*(3, N) or (3,) ndarray*) – The electric drift velocity at *newheight* (geodetic east, north, and up components)

basevectors_qd(*lat*, *lon*, *height*, *coords='geo'*, *precision=1e-10*)

Returns quasi-dipole base vectors f1 and f2 at the specified coordinates.

The vectors are described by Richmond [1995][2] and Emmert et al. [2010][3]. The vector components are geodetic east and north.

Parameters

- **lat** (*(N,) array_like or float*) – Latitude
- **lon** (*(N,) array_like or float*) – Longitude
- **height** (*(N,) array_like or float*) – Altitude in km
- **coords** (*{'geo', 'apex', 'qd'}, optional*) – Input coordinate system
- **precision** (*float, optional*) – Precision of output (degrees) when converting to geo. A negative value of this argument produces a low-precision calculation of geodetic lat/lon based only on their spherical harmonic representation. A positive value causes the underlying Fortran routine to iterate until feeding the output geo lat/lon into geo2qd (APXG2Q) reproduces the input QD lat/lon to within the specified precision (all coordinates being converted to geo are converted to QD first and passed through APXG2Q).

Returns

- **f1** (*(2, N) or (2,) ndarray*)
- **f2** (*(2, N) or (2,) ndarray*)

[2] Richmond, A. D. (1995), Ionospheric Electrodynamics Using Magnetic Apex Coordinates, Journal of geomagnetism and geoelectricity, 47(2), 191–212, doi:10.5636/jgg.47.191.

[3] Emmert, J. T., A. D. Richmond, and D. P. Drob (2010), A computationally compact representation of Magnetic-Apex and Quasi-Dipole coordinates with smooth base vectors, J. Geophys. Res., 115(A8), A08322, doi:10.1029/2010JA015326.

References

basevectors_apex (*lat*, *lon*, *height*, *coords='geo'*, *precision=1e-10*)

Returns base vectors in quasi-dipole and apex coordinates.

The vectors are described by Richmond [1995][4] and Emmert et al. [2010][5]. The vector components are geodetic east, north, and up (only east and north for *f1* and *f2*).

Parameters

- **lat, lon** (*(N,) array_like or float*) – Latitude
- **lat** (*(N,) array_like or float*) – Longitude
- **height** (*(N,) array_like or float*) – Altitude in km
- **coords** (*{'geo', 'apex', 'qd'}, optional*) – Input coordinate system
- **return_all** (*bool, optional*) – Will also return f3, g1, g2, and g3, and f1 and f2 have 3 components (the last component is zero). Requires *lat*, *lon*, and *height* to be broadcast to 1D (at least one of the parameters must be 1D and the other two parameters must be 1D or 0D).
- **precision** (*float, optional*) – Precision of output (degrees) when converting to geo. A negative value of this argument produces a low-precision calculation of geodetic lat/lon based only on their spherical harmonic representation. A positive value causes the underlying Fortran routine to iterate until feeding the output geo lat/lon into geo2qd (APXG2Q) reproduces the input QD lat/lon to within the specified precision (all coordinates being converted to geo are converted to QD first and passed through APXG2Q).

Returns

- **f1, f2** (*(2, N) or (2,) ndarray*)
- **f3, g1, g2, g3, d1, d2, d3, e1, e2, e3** (*(3, N) or (3,) ndarray*)

Note: *f3*, *g1*, *g2*, and *g3* are not part of the Fortran code by Emmert et al. [2010][5]. They are calculated by this Python library according to the following equations in Richmond [1995][4]:

- *g1*: Eqn. 6.3
- *g2*: Eqn. 6.4
- *g3*: Eqn. 6.5
- *f3*: Eqn. 6.8

References

get_apex (*alat*)

Computes the field line apex for a given modified apex latitude.

Parameters **alat** (*array_like*) – Modified apex latitude

Returns **apex** (*ndarray or float*) – Field line apex in km

[4] Richmond, A. D. (1995), Ionospheric Electrodynamics Using Magnetic Apex Coordinates, Journal of geomagnetism and geoelectricity, 47(2), 191–212, doi:10.5636/jgg.47.191.

[5] Emmert, J. T., A. D. Richmond, and D. P. Drob (2010), A computationally compact representation of Magnetic-Apex and Quasi-Dipole coordinates with smooth base vectors, J. Geophys. Res., 115(A8), A08322, doi:10.1029/2010JA015326.

set_epoch(*year*)
Updates the epoch for all subsequent conversions.

Parameters **year** (*float*) – Decimal year

set_refh(*refh*)
Updates the apex reference height for all subsequent conversions.

Parameters **refh** (*float*) – Apex reference height in km

Notes

The reference height is the height to which field lines will be mapped, and is only relevant for conversions involving apex (not quasi-dipole).

4.2 apexpy.helpers

This module contains helper functions used by `Apex`.

apexpy.helpers.**checklat**(*lat*, *name='lat'*)
Makes sure the latitude is inside [-90, 90], clipping close values (tolerance 1e-4).

Parameters

- **lat** (*array_like*) – latitude
- **name** (*str, optional*) – parameter name to use in the exception message

Returns **lat** (*ndarray or float*) – Same as input where values just outside the range have been clipped to [-90, 90]

Raises `ValueError` – if any values are too far outside the range [-90, 90]

apexpy.helpers.**getsinIm**(*alat*)
Computes sinIm from modified apex latitude.

Parameters **alat** (*array_like*) – Modified apex latitude

Returns **sinIm** (*ndarray or float*)

apexpy.helpers.**getcosIm**(*alat*)
Computes cosIm from modified apex latitude.

Parameters **alat** (*array_like*) – Modified apex latitude

Returns **cosIm** (*ndarray or float*)

apexpy.helpers.**toYearFraction**(*date*)
Converts `datetime.date` or `datetime.datetime` to decimal year.

Parameters **date** (`datetime.date` or `datetime.datetime`)

Returns **year** (*float*) – Decimal year

Notes

The algorithm is taken from http://stackoverflow.com/a/6451892/2978652

apexpy.helpers.**gc2gdlat**(*gclat*)
Converts geocentric latitude to geodetic latitude using WGS84.

Parameters **gclat** (*array_like*) – Geocentric latitude

Returns **gdlat** (*ndarray or float*) – Geodetic latitude

`apexpy.helpers.`**`subsol`**(*datetime*)

Finds subsolar geocentric latitude and longitude.

Parameters **datetime** (`datetime.datetime`)

Returns

- **sbsllat** (*float*) – Latitude of subsolar point
- **sbsllon** (*float*) – Longitude of subsolar point

Notes

Based on formulas in Astronomical Almanac for the year 1996, p. C24. (U.S. Government Printing Office, 1994). Usable for years 1601-2100, inclusive. According to the Almanac, results are good to at least 0.01 degree latitude and 0.025 degrees longitude between years 1950 and 2050. Accuracy for other years has not been tested. Every day is assumed to have exactly 86400 seconds; thus leap seconds that sometimes occur on December 31 are ignored (their effect is below the accuracy threshold of the algorithm).

After Fortran code by A. D. Richmond, NCAR. Translated from IDL by K. Laundal.

4.3 apexpy.fortranapex

This module is the interface to the apex Fortran library by Emmert et al. [2010][1]. The interface is not documented. Use *`apexpy.Apex`* for all conversions and calculations. You can find some documentation of the actual Fortran library in the source file apexsh.f90.

4.4 Command-line interface

When you install this package you will get a command called `apexpy`, which is an interface to the *`convert()`* method. See the documentation for this method for a more thorough explanation of arguments and behaviour.

You can get help on the command by running `apexpy -h`.

```
$ apexpy -h
usage: apexpy [-h] [--height HEIGHT] [--refh REFH] [-i FILE_IN] [-o FILE_OUT]
              SOURCE DEST DATE

Converts between geodetic, modified apex, quasi-dipole and MLT

positional arguments:
  SOURCE                Convert from {geo, apex, qd, mlt}
  DEST                  Convert to {geo, apex, qd, mlt}
  DATE                  YYYY[MM[DD[HHMMSS]]] date/time for IGRF coefficients,
                        time part required for MLT calculations

optional arguments:
  -h, --help            show this help message and exit
```

[1] Emmert, J. T., A. D. Richmond, and D. P. Drob (2010), A computationally compact representation of Magnetic-Apex and Quasi-Dipole coordinates with smooth base vectors, J. Geophys. Res., 115(A8), A08322, doi:10.1029/2010JA015326.

```
  --height HEIGHT       height for conversion
  --refh REFH           reference height for modified apex coordinates
  -i FILE_IN, --input FILE_IN
                        input file (stdin if none specified)
  -o FILE_OUT, --output FILE_OUT
                        output file (stdout if none specified)
```

CHAPTER 5

Contributing

Bug reports, feature suggestions and other contributions are greatly appreciated! While I can't promise to implement everything, I will always try to respond in a timely manner.

5.1 Short version

- Submit bug reports and feature requests at GitHub
- Make pull requests to the `develop` branch

5.2 Bug reports

When reporting a bug please include:

- Your operating system name and version
- Any details about your local setup that might be helpful in troubleshooting
- Detailed steps to reproduce the bug

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at GitHub.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *apexpy* for local development:

1. Fork apexpy on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/apexpy.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally. Add tests for bugs and new features in the relevant test file in the `tests` directory. The tests are run with `py.test` and can be written as normal functions (starting with `test_`) containing a standard `assert` statement for testing output.

4. When you're done making changes, run `py.test` locally if you can:

```
py.test
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Brief description of your changes"
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website. Pull requests should be made to the `develop` branch. The continuous integration (CI) testing servers will automatically test the whole codebase, including your changes, for multiple versions of Python on both Windows and Linux.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code, just make a pull request.

For merging, you should:

1. Include passing tests for your changes
2. Update/add documentation if relevant
3. Add a note to `CHANGELOG.rst` about the changes
4. Add yourself to `AUTHORS.rst`

CHAPTER 6

Authors

This python wrapper is made by:

- Karl M. Laundal
- Christer van der Meeren
- Angeline G. Burrell (maintainer)

Fortran code by Emmert et al. [2010][1]. Quasi-dipole and modified apex coordinates are defined by Richmond [1995][2]. The code uses IGRF-12 with coefficients valid through 2020 [Thébault et al., 2015][3].

[1] Emmert, J. T., A. D. Richmond, and D. P. Drob (2010), A computationally compact representation of Magnetic-Apex and Quasi-Dipole coordinates with smooth base vectors, J. Geophys. Res., 115(A8), A08322, doi:10.1029/2010JA015326.

[2] Richmond, A. D. (1995), Ionospheric Electrodynamics Using Magnetic Apex Coordinates, Journal of geomagnetism and geoelectricity, 47(2), 191–212, doi:10.5636/jgg.47.191.

[3] Thébault, E. et al. (2015), International Geomagnetic Reference Field: the 12th generation, Earth, Planets and Space, 67(1), 79, doi:10.1186/s40623-015-0228-9.

CHAPTER 7

Changelog

7.1 1.0.2 (2018-02-27)

- Extend character limit for allowable data file path, and update documentation to reflect a change in maintainers. Also updated testing implimentation, reduced fortran compiler warnings, and improved PEP8 compliance.

7.2 1.0.1 (2016-03-10)

- Remove geocentric to geodetic conversion of subsolar point based on feedback from Art Richmond. (The subsolar point is the same in geocentric and geodetic coordinates.) The helper function *gc2gdlat* have been kept to preserve backwards compatibility.

7.3 1.0.0 (2015-11-30)

- Initial release

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index