AACGM-v2 Python library

Release 2.6.3

Angeline G. Burrell, et al.

Feb 08, 2023

CONTENTS

CHAPTER

ONE

OVERVIEW

This is a Python wrapper for the AACGM-v2 C library, which allows converting between geographic and magnetic coordinates. The currently included version of the C library is 2.6. The package is free software (MIT license). When referencing this package, please cite both the package DOI and the AACGM-v2 journal article:

Shepherd, S. G. (2014), Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations, Journal of Geophysical Research: Space Physics, 119, 7501–7521, doi:10.1002/2014JA020264.

1.1 Quick start

Install (requires NumPy):

```
pip install aacgmv2
```

Convert between AACGM and geographic coordinates:

```
>>> import aacgmv2
>>> import datetime as dt
>>> import numpy as np
>>> np.set_printoptions(formatter={'float_kind': lambda x:'{:.4f}'.format(x)})
>>> # geo to AACGM, single numbers
>>> dtime = dt.datetime(2013, 11, 3)
>>> np.array(aacgmv2.get_aacgm_coord(60, 15, 300, dtime))
array([57.4736, 93.6111, 1.4816])
>>> # AACGM to geo, mix arrays/numbers
>>> np.array2string(np.array(aacgmv2.convert_latlon_arr([90, -90], 0, 0, dtime, method_
↪code="A2G"))).replace('\n', '')
'[[82.9686 -74.3390] [-84.6501 125.8476] [14.1246 12.8772]]'
```

Convert between AACGM and MLT:

```
>>> import aacgmv2
>>> import datetime as dt
>>> import numpy as np
>>> np.set_printoptions(formatter={'float_kind': lambda x:'{:.4f}'.format(x)})
>>> # MLT to AACGM
>>> dtime = dt.datetime(2013, 11, 3, 0, 0, 0)
>>> np.array(aacgmv2.convert_mlt([1.4822189, 12], dtime, m2a=True))
array([93.6203, -108.6130])
```

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://aacgmv2.readthedocs.io/en/latest/

http://superdarn.thayer.dartmouth.edu/aacgm.html

1.3 Badges

docs	
tests	
package	

CHAPTER

TWO

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, the operating system package managers can be used to install `numpy`. However, this step may be unnecessary, as PyPi should install `numpy` along with *aacgmv2* if it is missing.

We recommend installing this package at the command line using `pip`:

```
pip install aacgmv2
```

2.1 Tested Setups

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

- Mac (64 bit), Windows (64 bit), and Linux (64 bit)
- Python 3.7, 3.8, 3.9, and 3.10

2.2 Known Problems and Solutions

There is a known issue using *aacgmv2* on Windows with pycharm. After a successful installation, running the code in Powershell or pycharm won't work because, even though the environment variables appear to be set successfully, the C code can't access them. This may cause Python to crash. To fix this issue, simply manually set the environment variables before starting your Python session. To find out what you need to set, run the following code in Python:

```
import aacgmv2
import os

print(os.getenv("AACGM_v2_DAT_PREFIX"))
print(os.getenv("IGRF_COEFFS"))
```

CHAPTER

THREE

USAGE EXAMPLES

3.1 Python library

For full documentation of the functions, see *Reference → aacgmv2*.

For simple examples on converting between AACGM and geographic coordinates and AACGM and MLT, see *Overview → Quick Start*.

3.2 Command-line interface

The Python package also installs a command called `aacgmv2` with several sub-commands that allow conversion between geographic/geodetic and AACGM-v2 magnetic coordinates (mlat, mlon, and mlt). The Command-Line Interface (CLI) allows you to make use of the Python library even if you don't know or use Python. See *Reference → Command-line interface* for a list of arguments to the commands. Below are some simple usage examples.

3.2.1 Convert geographic/magnetic coordinates

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

```
# lat lon alt
# comment lines like these are ignored
60 15 300
61 15 300
62 15 300
```

To convert this to AACGM-v2 for the date 2015-02-24, run the command `python -m aacgmv2 convert -i input.txt -o output.txt -d 20150224`. The output file will look like this:

```
57.47612194 93.55719875 1.04566346
58.53323704 93.96069212 1.04561304
59.58522105 94.38968625 1.04556369
```

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

```
$ echo 60 15 300 | python -m aacgmv2 convert -d 20150224
57.47612194 93.55719875 1.04566346
```

3.2.2 Convert MLT

This works in much the same way as calling the CLI with the `convert` flag. The file should only contain a single column of numbers (MLTs or magnetic longitudes, depending on which way you're converting):

```
1
12
23
```

To convert these MLTs to magnetic longitudes at 2015-02-24 14:00:15, run e.g. `python aacgmv2 convert_mlt 20150224140015 -i input.txt -o output.txt -v` (note that the date/time is a required parameter). The output file will then look like this:

```
-120.34354125
44.65645875
-150.34354125
```

As with the `convert` flag, you can use stdin/stdout instead of input and output files:

```
$ echo 12 | python -m aacgmv2 convert_mlt 20150224140015 -v
44.65645875
```

3.3 Module Function Examples

The purpose of *aacgmv2* is to convert between geographic or geodetic coordinates and magnetic coordinates at a known location. It does not perform field-line tracing from one location for another, since this requires making assumptions about the shape of the magnetic field lines. We recommend using `apexpy` for this purpose.

3.3.1 Convert coordinates

There are several functions that convert between geographic/detic and magnetic coordinates. Which one to use on what you want. To convert latitude and longitude for a single value, use *convert_latlon()*. For multiple values at the same time, use *convert_latlon_arr()*. The times can't be vectorized because the C code needs to be re-initialized for every new time. You can see how these two functions perform differently using the `timeit` package:

```
import timeit

# The array version takes less than a second per run: ~0.99892
array_command = "".join(["import aacgmv2; import datetime as dt; ",
                         "import numpy as np; rando_lon = ",
                         "np.random.uniform(low=-180, high=180, size=100);",
                         "rando_lat = np.random.uniform(low=-90, high=90, ",
                         "size=100); aacgmv2.convert_latlon_arr(rando_lat, ",
                         "rando_lon, 300.0, dt.datetime(2015, 5, 5), ",
                         "method_code='G2A')"])
timeit.timeit(array_command, number=1000)

# The single value version run using list compression takes longer: ~2.36
# It also raises a warning every time there is a measurement near the
# magnetic equator (where AACGMV2 coordinates are undefined).
list_command = "".join(["import aacgmv2; import datetime as dt; ",
```

(continues on next page)

(continued from previous page)

```
                          "import numpy as np; rando_lon = ",
                          "np.random.uniform(low=-180, high=180, size=100);",
                          "rando_lat = np.random.uniform(low=-90, high=90, ",
                          "size=100); [aacgmv2.convert_latlon(rando_lat[i], ",
                          "lon, 300.0, dt.datetime(2015, 5, 5), ",
                          "method_code='G2A') for i, lon in ",
                          "enumerate(rando_lon)]"])
timeit.timeit(list_command, number=1000)
```

To convert between magnetic longitude and local time, use *convert_mlt()*. This function examines the data and uses different C wrappers for array or single valued inputs.:

```
import aacgmv2
import datetime as dt

# Convert MLT to longitude and back again
dtime = dt.datetime(2020, 1, 1)
mlon = aacgmv2.convert_mlt(24.0, dtime, m2a=True)
mlt = aacgmv2.convert_mlt(mlon, dtime, m2a=False)

# This yields: 78.405 E = 24.000 h
print("{:.3f} E = {:.3f} h".format(mlon[0], mlt[0]))
```

If you want magnetic latitude, longitude, and local time at a given location, you can use *get_aacgm_coord()* for a single location or *get_aacgm_coord_arr()* for several locations at a given time. These functions combine the latitude, longitude, and local time conversion functions to allow the user to access all magnetic coordinates in a single call. However, they do not allow the user to convert from magnetic to geodetic coordinates.:

```
import aacgmv2
import datetime as dt

dtime = dt.datetime(2020, 1, 1)
mlat, mlon, mlt = aacgmv2.get_aacgm_coord(45.0, 0.0, 300.0, dtime,
                                          method='ALLOWTRACE')

# This yeilds: 40.749 E, 76.177 N, 23.851 h
print("{:.3f} E, {:.3f} N, {:.3f} h".format(mlat, mlon, mlt))
```

3.3.2 Utilities

There are additional utilities available in *aacgmv2.utils* that may prove useful to users. The example below demonstrates how to convert between geographic and geodetic coordinates.:

```
import aacgmv2

# This will yield a geodetic lat of 45.192 degrees
gd_lat = aacgmv2.utils.gc2gd_lat(45.0)
print("{:.3f}".format(gd_lat))
```

Another utility provides the subsolar point in geocentric coordinates.:

```
import aacgmv2

# This will yield geocentric values of: -179.233 E, -23.059 N
ss_gc_lon, ss_gc_lat = aacgmv2.utils.subsol(2020, 1, 1)
print("{:.3f} E, {:.3f} N".format(ss_gc_lon, ss_gc_lat))
```

Finally, you can retrieve a Cartesian unit vector that points to the dipolar International Geomagnetic Reference Field (IGRF) northern pole.:

```
import aacgmv2
import datetime as dt

# For IGRF-13 this will yield an array with values of:
# array([ 0.04867761, -0.1560909 ,  0.98654251])
aacgmv2.utils.igrf_dipole_axis(dt.datetime(2020, 1, 1))
```

CHAPTER

FOUR

REFERENCE

4.1 aacgmv2

These functions are available when you `import aacgmv2`.

Conversion functions between geo-graphic/detic and AACGM-V2 magnetic coords.

`aacgmv2.logger`

Logger handle

Type
(logger)

`aacgmv2.high_alt_coeff`

Upper altitude limit for using coefficients in km

Type
(float)

`aacgmv2.high_alt_trace`

Upper altitude limit for using field-line tracing in km

Type
(float)

`aacgmv2.AACGM_V2_DAT_PREFIX`

Location of AACGM-V2 coefficient files with the file prefix

Type
(str)

`aacgmv2.IGRF_COEFFS`

Filename, with directory, of IGRF coefficients

Type
(str)

4.1.1 aacgmv2.wrapper

Pythonic wrappers for AACGM-V2 C functions.

aacgmv2.wrapper.**convert_bool_to_bit**(*a2g=False*, *trace=False*, *allowtrace=False*, *badidea=False*, *geocentric=False*)

Convert boolian flags to bit code specification.

Parameters

- **a2g** (*bool*) – True for AACGM-v2 to geographic (geodetic), False otherwise (default=False)
- **trace** (*bool*) – If True, use field-line tracing, not coefficients (default=False)
- **allowtrace** (*bool*) – If True, use trace only above 2000 km (default=False)
- **badidea** (*bool*) – If True, use coefficients above 2000 km (default=False)
- **geocentric** (*bool*) – True for geodetic, False for geocentric w/RE=6371.2 (default=False)

Returns

bit_code – code specification in bits

Return type

int

aacgmv2.wrapper.**convert_latlon**(*in_lat*, *in_lon*, *height*, *dtime*, *method_code='G2A'*)

Convert between geomagnetic coordinates and AACGM coordinates.

Parameters

- **in_lat** (*float*) – Input latitude in degrees N (code specifies type of latitude)
- **in_lon** (*float*) – Input longitude in degrees E (code specifies type of longitude)
- **height** (*float*) – Altitude above the surface of the earth in km
- **dtime** (*dt.datetime*) – Datetime for magnetic field
- **method_code** (*str or int*) – Bit code or string denoting which type(s) of conversion to perform (default="G2A")

  G2A
  Geographic (geodetic) to AACGM-v2

  A2G
  AACGM-v2 to geographic (geodetic)

  TRACE
  Use field-line tracing, not coefficients

  ALLOWTRACE
  Use trace only above 2000 km

  BADIDEA
  Use coefficients above 2000 km

  GEOCENTRIC
  Assume inputs are geocentric w/ RE=6371.2

Returns

- **out_lat** (*float*) – Output latitude in degrees N
- **out_lon** (*float*) – Output longitude in degrees E

- **out_r** (*float*) – Geocentric radial distance (R_Earth) or altitude above the surface of the Earth (km)

Raises

- **ValueError** – If input is incorrect.
- **RuntimeError** – If unable to set AACGMV2 datetime.

aacgmv2.wrapper.**convert_latlon_arr**(*in_lat*, *in_lon*, *height*, *dtime*, *method_code='G2A'*)

Convert between geomagnetic coordinates and AACGM coordinates.

Parameters

- **in_lat** (*np.ndarray, list, or float*) – Input latitude in degrees N (method_code specifies type of latitude)
- **in_lon** (*np.ndarray or list or float*) – Input longitude in degrees E (method_code specifies type of longitude)
- **height** (*np.ndarray or list or float*) – Altitude above the surface of the earth in km
- **dtime** (*dt.datetime*) – Single datetime object for magnetic field
- **method_code** (*int or str*) – Bit code or string denoting which type(s) of conversion to perform (default="G2A")

  G2A
  Geographic (geodetic) to AACGM-v2

  A2G
  AACGM-v2 to geographic (geodetic)

  TRACE
  Use field-line tracing, not coefficients

  ALLOWTRACE
  Use trace only above 2000 km

  BADIDEA
  Use coefficients above 2000 km

  GEOCENTRIC
  Assume inputs are geocentric w/ RE=6371.2

Returns

- **out_lat** (*np.ndarray*) – Output latitudes in degrees N
- **out_lon** (*np.ndarray*) – Output longitudes in degrees E
- **out_r** (*np.ndarray*) – Geocentric radial distance (R_Earth) or altitude above the surface of the Earth (km)

Raises

- **ValueError** – If input is incorrect.
- **RuntimeError** – If unable to set AACGMV2 datetime.

Notes

At least one of in_lat, in_lon, and height must be a list or array.

If errors are encountered, NaN or Inf will be included in the input so that all successful calculations are returned. To select only good values use a function like *np.isfinite*.

Multi-dimensional arrays are not allowed.

aacgmv2.wrapper.**convert_mlt**(*arr*, *dtime*, *m2a=False*)

Converts between magnetic local time (MLT) and AACGM-v2 longitude.

Parameters

- **arr** (*array-like or float*) – Magnetic longitudes (degrees E) or MLTs (hours) to convert
- **dtime** (*array-like or dt.datetime*) – Date and time for MLT conversion in Universal Time (UT).
- **m2a** (*bool*) – Convert MLT to AACGM-v2 longitude (True) or magnetic longitude to MLT (False). (default=False)

Returns

out – Converted coordinates/MLT in degrees E or hours (as appropriate)

Return type

np.ndarray

Notes

This routine previously based on Laundal et al. 2016, but now uses the improved calculation available in AACGM-V2.4.

aacgmv2.wrapper.**convert_str_to_bit**(*method_code*)

Convert string code specification to bit code specification.

Parameters

method_code (*str*) – Bitwise code for passing options into converter:

G2A
Geographic (geodetic) to AACGM-v2

A2G
AACGM-v2 to geographic (geodetic)

TRACE
Use field-line tracing, not coefficients

ALLOWTRACE
Use trace only above 2000 km

BADIDEA
Use coefficients above 2000 km

GEOCENTRIC
Assume inputs are geocentric w/ RE=6371.2

Returns

bit_code – Method code specification in bits

Return type
int

Notes

Multiple codes should be seperated by pipes |. Invalid parts of the code are ignored and no code defaults to 'G2A'.

aacgmv2.wrapper.**get_aacgm_coord**(*glat*, *glon*, *height*, *dtime*, *method='ALLOWTRACE'*)

Get AACGM latitude, longitude, and magnetic local time.

Parameters

- **glat** (*float*) – Geodetic latitude in degrees N
- **glon** (*float*) – Geodetic longitude in degrees E
- **height** (*float*) – Altitude above the surface of the earth in km
- **dtime** (*dt.datetime*) – Date and time to calculate magnetic location
- **method** (*str*) – The type(s) of conversion to perform (default="ALLOWTRACE")

  TRACE
  Use field-line tracing, not coefficients

  ALLOWTRACE
  Use trace only above 2000 km

  BADIDEA
  Use coefficients above 2000 km

  GEOCENTRIC
  Assume inputs are geocentric w/ RE=6371.2

Returns

- **mlat** (*float*) – Magnetic latitude in degrees N
- **mlon** (*float*) – Magnetic longitude in degrees E
- **mlt** (*float*) – Magnetic local time in hours

aacgmv2.wrapper.**get_aacgm_coord_arr**(*glat*, *glon*, *height*, *dtime*, *method='ALLOWTRACE'*)

Get AACGM latitude, longitude, and magnetic local time.

Parameters

- **glat** (*np.array or list*) – Geodetic latitude in degrees N
- **glon** (*np.array or list*) – Geodetic longitude in degrees E
- **height** (*np.array or list*) – Altitude above the surface of the earth in km
- **dtime** (*dt.datetime*) – Date and time to calculate magnetic location
- **method** (*str*) – The type(s) of conversion to perform (default="ALLOWTRACE")

  TRACE
  Use field-line tracing, not coefficients

  ALLOWTRACE
  Use trace only above 2000 km

BADIDEA
Use coefficients above 2000 km

GEOCENTRIC
Assume inputs are geocentric w/ RE=6371.2

Returns

- **mlat** (*float*) – Magnetic latitude in degrees N
- **mlon** (*float*) – Magnetic longitude in degrees E
- **mlt** (*float*) – Magnetic local time in hours

aacgmv2.wrapper.**set_coeff_path**(*igrf_file=False*, *coeff_prefix=False*)

Set the IGRF_COEFF and AACGMV_V2_DAT_PREFIX environment variables.

Parameters

- **igrf_file** (*str or bool*) – Full filename of IGRF coefficient file, True to use aacgmv2.IGRF_COEFFS, or False to leave as is. (default=False)
- **coeff_prefix** (*str or bool*) – Location and file prefix for aacgm coefficient files, True to use aacgmv2.AACGM_V2_DAT_PREFIX, or False to leave as is. (default=False)

aacgmv2.wrapper.**test_height**(*height*, *bit_code*)

Test the input height and ensure it is appropriate for the method.

Parameters

- **height** (*float*) – Height to test in km
- **bit_code** (*int*) – Code string denoting method to use

Returns
good_height – True if height and method are appropriate, False if not

Return type
bool

Notes

Appropriate altitude ranges for the different methods are explored in Shepherd (2014). Summarized, they are: 1. Coefficients: 0-2000 km 2. Tracing: 0-1 Earth Radius

Altitudes below zero will work, but will not provide a good representation of the magnetic field because it goes beyond the intended scope of these coordiantes.

If you use the 'BADIDEA' code, you can bypass all constraints, but it is a Bad Idea! If you include a high enough altiutde, the code may hang.

aacgmv2.wrapper.**test_time**(*dtime*)

Test the time input and ensure it is a dt.datetime object.

Parameters
dtime (*any*) – Time input in an untested format

Returns
dtime – Time as a datetime object

Return type
dt.datetime

Raises
ValueError – If time is not a dt.date or dt.datetime object

4.1.2 aacgmv2.utils

Utilities that support the AACGM-V2 C functions.

References

Laundal, K. M. and A. D. Richmond (2016), Magnetic Coordinate Systems, Space Sci. Rev., doi:10.1007/s11214-016-0275-y.

aacgmv2.utils.**gc2gd_lat**(*gc_lat*)
Convert geocentric latitude to geodetic latitude using WGS84.

Parameters
gc_lat (*array-like or float*) – Geocentric latitude in degrees N

Returns
gd_lat – Geodetic latitude in degrees N, same type as input *gc_lat*

Return type
array-like or float

aacgmv2.utils.**igrf_dipole_axis**(*date*)
Get Cartesian unit vector pointing at dipole pole in the north (IGRF).

Parameters
date (*dt.datetime*) – Date and time

Returns
m_0 – Cartesian 3 element unit vector pointing at dipole pole in the north (geocentric coords)

Return type
np.ndarray

Notes

IGRF coefficients are read from the igrf12coeffs.txt file. It should also work after IGRF updates. The dipole coefficients are interpolated to the date, or extrapolated if date > latest IGRF model

aacgmv2.utils.**subsol**(*year*, *doy*, *utime*)
Find subsolar geocentric longitude and latitude.

Parameters

- **year** (*int*) – Calendar year between 1601 and 2100
- **doy** (*int*) – Day of year between 1-365/366
- **utime** (*float*) – Seconds since midnight on the specified day

Returns

- **sbsllon** (*float*) – Subsolar longitude in degrees E for the given date/time
- **sbsllat** (*float*) – Subsolar latitude in degrees N for the given date/time

Raises
ValueError – If year is out of range

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).

References

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

4.2 aacgmv2._aacgmv2

This submodule contains the interface to the AACGM-v2 C library. For the user-friendly wrapper, see the functions in *aacgmv2.wrapper*.

Interface to the AACGM-v2 C library.

aacgmv2._aacgmv2.**convert**(*in_lat*, *in_lon*, *height*, *code*)

Converts between geographic/dedic and magnetic coordinates.

Parameters

- **in_lat** (*float*) – Input latitude in degrees N (code specifies type of latitude)
- **in_lon** (*float*) – Input longitude in degrees E (code specifies type of longitude)
- **height** (*float*) – Altitude above the surface of the earth in km
- **code** (*int*) – Bitwise code for passing options into converter (default=0)

  0 - G2A
  Geographic (geodetic) to AACGM-v2

  1 - A2G
  AACGM-v2 to geographic (geodetic)

  2 - TRACE
  Use field-line tracing, not coefficients

  4 - ALLOWTRACE
  Use trace only above 2000 km

  8 - BADIDEA
  Use coefficients above 2000 km

  16 - GEOCENTRIC
  Assume inputs are geocentric w/ RE=6371.2

Returns

- **out_lat** (*float*) – Output latitude in degrees
- **out_lon** (*float*) – Output longitude in degrees
- **out_r** (*float*) – Geocentric radial distance in Re

aacgmv2._aacgmv2.**convert_arr**(*in_lat*, *in_lon*, *height*, *code*)

Converts between geographic/dedic and magnetic coordinates.

Parameters

- **in_lat** (*list*) – Input latitudes in degrees N (code specifies type of latitude)
- **in_lon** (*list*) – Input longitudes in degrees E (code specifies type of longitude)
- **height** (*list*) – Altitudes above the surface of the earth in km
- **code** (*int*) – Bitwise code for passing options into converter (default=0)

  0 - G2A
  Geographic (geodetic) to AACGM-v2

  1 - A2G
  AACGM-v2 to geographic (geodetic)

  2 - TRACE
  Use field-line tracing, not coefficients

  4 - ALLOWTRACE
  Use trace only above 2000 km

  8 - BADIDEA
  Use coefficients above 2000 km

  16 - GEOCENTRIC
  Assume inputs are geocentric w/ RE=6371.2

Returns

- **out_lat** (*list*) – Output latitudes in degrees
- **out_lon** (*list*) – Output longitudes in degrees
- **out_r** (*list*) – Geocentric radial distances in Re
- **out_bad** (*list*) – Indices at or greater than zero indicate filler data in previous outputs

Notes

Return values of -666 are used as filler values for lat/lon/r, while filler values of -1 are used in out_bad if the output in out_lat/lon/r is good

aacgmv2._aacgmv2.**inv_mlt_convert**(*yr*, *mo*, *dy*, *hr*, *mt*, *sc*, *mlt*)

Converts from universal time and magnetic local time to magnetic longitude.

Parameters

- **yr** (*int*) – 4 digit integer year (1590-2025)
- **mo** (*int*) – Month of year (1-12)
- **dy** (*int*) – Day of month (1-31)
- **hr** (*int*) – hours of day (0-23)
- **mt** (*int*) – Minutes of hour (0-59)
- **sc** (*int*) – Seconds of minute (0-59)
- **mlt** (*float*) – Magnetic local time

Returns
mlon – Magnetic longitude (degrees)

Return type
float

aacgmv2._aacgmv2.**inv_mlt_convert_arr**(*yr*, *mo*, *dy*, *hr*, *mt*, *sc*, *mlt*)

Converts from universal time and magnetic local time to magnetic longitude.

Parameters

- **yr** (*list*) – 4 digit integer year (1590-2025)
- **mo** (*list*) – Month of year (1-12)
- **dy** (*list*) – Day of month (1-31)
- **hr** (*list*) – hours of day (0-23)
- **mt** (*list*) – Minutes of hour (0-59)
- **sc** (*list*) – Seconds of minute (0-59)
- **mlt** (*list*) – Magnetic local time

Returns
mlon – Magnetic longitude (degrees)

Return type
list

aacgmv2._aacgmv2.**inv_mlt_convert_yrsec**(*yr*, *yr_sec*, *mlt*)

Converts from universal time and magnetic local time to magnetic longitude.

Parameters

- **yr** (*int*) – 4 digit integer year (1590-2025)
- **yr_sec** (*int*) – Seconds of year (0-31622400)
- **mlt** (*float*) – Magnetic local time

Returns
mlon – Magnetic longitude (degrees)

Return type
float

aacgmv2._aacgmv2.**mlt_convert**(*yr*, *mo*, *dy*, *hr*, *mt*, *sc*, *mlon*)

Converts from universal time to magnetic local time.

Parameters

- **yr** (*int*) – 4 digit integer year (1590-2025)
- **mo** (*int*) – Month of year (1-12)
- **dy** (*int*) – Day of month (1-31)
- **hr** (*int*) – hours of day (0-23)
- **mt** (*int*) – Minutes of hour (0-59)
- **sc** (*int*) – Seconds of minute (0-59)
- **mlon** (*float*) – Magnetic longitude

Returns
mlt – Magnetic local time (hours)

Return type
float

aacgmv2._aacgmv2.**mlt_convert_arr**(*yr*, *mo*, *dy*, *hr*, *mt*, *sc*, *mlon*)

Converts from universal time to magnetic local time.

Parameters

- **yr** (*list*) – 4 digit integer year (1590-2025)
- **mo** (*list*) – Month of year (1-12)
- **dy** (*list*) – Day of month (1-31)
- **hr** (*list*) – hours of day (0-23)
- **mt** (*list*) – Minutes of hour (0-59)
- **sc** (*list*) – Seconds of minute (0-59)
- **mlon** (*list*) – Magnetic longitude

Returns
mlt – Magnetic local time (hours)

Return type
list

aacgmv2._aacgmv2.**mlt_convert_yrsec**(*yr*, *yr_sec*, *mlon*)

Converts from universal time to magnetic local time.

Parameters

- **yr** (*int*) – 4 digit integer year (1590-2025)
- **yr_sec** (*int*) – Seconds of year (0-31622400)
- **mlon** (*float*) – Magnetic longitude

Returns
mlt – Magnetic local time (hours)

Return type
float

aacgmv2._aacgmv2.**set_datetime**(*year*, *month*, *day*, *hour*, *minute*, *second*)

Set the date and time for the IGRF magnetic field.

Parameters

- **year** (*int*) – Four digit year starting from 1590, ending 2025
- **month** (*int*) – Month of year ranging from 1-12
- **day** (*int*) – Day of month (1-31)
- **hour** (*int*) – Hour of day (0-23)
- **minute** (*int*) – Minute of hour (0-59)
- **second** (*int*) – Seconds of minute (0-59)

Returns

Return type
Void

4.3 Command-line interface

When you install this package you will get a command called `aacgmv2`. It has two subcommands, `convert` and `convert_mlt`, which correspond to the functions `aacgmv2.convert_latlon_arr()` and `aacgmv2.convert_mlt()`. See the documentation for these functions for a more thorough explanation of arguments and behaviour.

You can get help on the two commands by running `python aacgmv2 convert -h` and `python aacgmv2 convert_mlt -h`.

4.3.1 convert

```
$ python aacgmv2 convert -h
usage: aacgmv2 convert [-h] [-i FILE_IN] [-o FILE_OUT] [-d YYYYMMDD] [-v] [-t]
                       [-a] [-b] [-g]

optional arguments:
  -h, --help            show this help message and exit
  -i FILE_IN, --input FILE_IN
                        input file (stdin if none specified)
  -o FILE_OUT, --output FILE_OUT
                        output file (stdout if none specified)
  -d YYYYMMDD, --date YYYYMMDD
                        date for magnetic field model (1900-2020, default:
                        today)
  -v, --a2g             invert - convert AACGM to geographic instead of
                        geographic to AACGM
  -t, --trace           use field-line tracing instead of coefficients
  -a, --allowtrace      automatically use field-line tracing above 2000 km
  -b, --badidea         allow use of coefficients above 2000 km (bad idea!)
  -g, --geocentric      assume inputs are geocentric with Earth radius 6371.2
                        km
```

4.3.2 convert_mlt

```
$ python aacgmv2 convert_mlt -h
usage: aacgmv2 convert_mlt [-h] [-i FILE_IN] [-o FILE_OUT] [-v] YYYYMMDDHHMMSS

positional arguments:
  YYYYMMDDHHMMSS        date and time for conversion

optional arguments:
  -h, --help            show this help message and exit
  -i FILE_IN, --input FILE_IN
                        input file (stdin if none specified)
  -o FILE_OUT, --output FILE_OUT
```

(continues on next page)

(continued from previous page)

```
                        output file (stdout if none specified)
  -v, --m2a             invert - convert MLT to AACGM longitude instead of
                        AACGM longitude to MLT
```

CHAPTER

FIVE

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.

You can also contribute by testing pull request or performing code reviews. If you help out in ways that don't involve hacking the code, please add your name under the **Thanks** header in the `AUTHORS.rst` file. We appreciate the time you have given to improve this project.

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 *aacgmv2* for local development:

1. Fork aacgmv2 on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/aacgmv2.git
```

3. Build the local code to allow for local Python development:

```
pip install -e .
```

4. 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 `tests/test_py_aacgmv2.py` (for the wrapper), `test_c_aacgmv2.py` (for the C extension), or `tests/test_cmd_aacgmv2.py` (for the command-line interface). `tests/test_dep_aacgmv2.py` includes tests for deprecated functions. The tests are run with `pytest` and can be written as normal functions (starting with `test_`) containing a standard `assert` statement for testing output, or use the numpy testing suite.

5. When you're done making changes, run the local unit tests using pytest:

```
python -m pytest
```

6. 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
```

7. Submit a pull request through the GitHub website. Pull requests should be made to the `develop` branch.

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 (run `tox`)[1]
2. Update/add documentation if relevant
3. Add a note to `CHANGELOG.rst` about the changes
4. Add yourself to the end of `AUTHORS.rst` and the `.zenodo.json` files

[1] If you don't have all the necessary Python versions available locally or have trouble building all the testing environments, you can rely on GitHub Actions - it will run the tests for each change you add in the pull request.

CHAPTER

SIX

AUTHORS

- Angeline G. Burrell - https://github.com/aburrell
- Christer van der Meeren - https://github.com/cmeeren
- Karl M. Laundal - https://github.com/klaundal
- Hugo van Kemenade - https://github.com/hugovk

CHAPTER

SEVEN

THANKS

- Marina Shmidt - Testing and code review
- Bill Rideout - Testing
- dinsmoro - Testing - https://github.com/dinsmoro

CHAPTER

EIGHT

CHANGELOG

8.1 2.6.3 (2023-02-08)

- Moved CI support to GitHub actions
- Added a pyproject.toml for PyPi support
- Updated unit tests to current pytest standards
- Updated links in the documentation
- Improved the documentation style and added docstring tests
- Fixed memory leak in the array C wrappers
- Fixed longitude output format for *convert_mlt*
- Added more examples to the documentation

8.2 2.6.2 (2020-01-13)

- Drop support for EOL Python 2.7 and added testing for Python 3.9
- Added a .zenodo.json file
- Fixed AppVeyor environment

8.3 2.6.1 (2020-09-11)

- Moved formerly deprecated utilities from *deprecated.py* to *utils.py*
- Removed allowance for deprecated kwarg *code* from *convert_latlon* and *convert_latlon_arr*, as scheduled
- Updated CI to include Python 3.8 everywhere
- Moved all configuration information to setup.cfg
- Fixed coveralls implementation
- Fixed broken links in the documentation
- Removed unused code analysis tools
- Improved unit test coverage
- Make PEP8 changes

8.4 2.6.0 (2020-01-06)

- Updated AACGM-v2 coefficients derived using the IGRF13 model
- Updated IGRF and GUFM1 coefficients using the IGRF13 model
- Added additional checks to the C code for reading the IGRF13 coefficient file
- Removed *convert* routine in *deprecated.py*
- Pushed back keyword argument deprecation of *code*
- Scheduled deprecation for remaining routines in *deprecated.py*
- Parametrized several unit tests
- Updated *README.md* examples
- Updated CI to include python 3.8

8.5 2.5.3 (2019-12-23)

- Changed log warning about array functions to info
- Changed default method from *TRACE* to *ALLOWTRACE*
- Added C wrappers for list input, removing inefficient use of *np.vectorize*
- Fixed documentation for use of *method_code*
- Added FutureWarning for deprecated use of *code* keyword argument
- Updated previous version's changelog to encompass all changes
- Improved docstrings to make documentation easier to read
- Removed failing twine commands from *appveyor.yml*
- Removed *RuntimeWarning* filter from *tox.ini*

8.6 2.5.2 (2019-08-27)

- Added FutureWarning to deprecated functions
- Updated names in licenses
- Moved module structure routine tests to their own class
- Added high altitude limit to avoid while-loop hanging
- Changed version support to 2.7, 3.6, and 3.7
- Removed logbook dependency
- Added logic to avoid reseting environment variables if not necessary
- Added copyright and license disclaimer to module-specific program files
- Changed keyword argument *code* to *method_code*

8.7 2.5.1 (2018-10-19)

- Commented out debug statement in C code
- Updated environment variable warning to output to stderr instead of stdout
- Added templates for pull requests, issues, and a code of conduct

8.8 2.5.0 (2018-08-08)

- Updated C code and coefficients to version 2.5. Changes in python code reflect changes in C code (includes going back to using environment variables instead of strings for coefficient file locations)
- Added decorators to some of the test functions
- Specified AppVeyor Visual Studio version, since it was defaulting to 2010 and that version doesn't work with python 3

8.9 2.4.2 (2018-05-21)

- Fixed bug in convert_mlt that caused all time inputs to occur at 00:00:00 UT
- Fixed year of last two updates in changelog

8.10 2.4.1 (2018-04-04)

- Fix bug in installation that caused files to be placed in the wrong directory
- Added DOI

8.11 2.4.0 (2018-03-21)

- Update to use AACGM-v2.4, which includes changes to the inverse MLT and dipole tilt functions and some minor bug fixes
- Updated file structure
- Updated methods, retaining old methods in deprecated module
- Added testing for python 3.6
- Updated dependencies, removing support for python 3.3
- Tested on Mac OSX
- Updated comments to include units for input and output

8.12 2.0.0 (2016-11-03)

- Change method of calculating MLT, see documentation of convert_mlt for details

8.13 1.0.13 (2015-10-30)

- Correctly convert output of subsol() to geodetic coordinates (the error in MLT/mlon conversion was not large, typically two decimal places and below)

8.14 1.0.12 (2015-10-26)

- Return nan in forbidden region instead of throwing exception

8.15 1.0.11 (2015-10-26)

- Fix bug in subsolar/MLT conversion

8.16 1.0.10 (2015-10-08)

- No code changes, debugged automatic build/upload process and needed new version numbers along the way

8.17 1.0.0 (2015-10-07)

- Initial release

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX