ocbpy Documentation

Release 0.2.1-beta

Angeline G. Burrell; Gareth Chisham; Jone Reistad

Jun 25, 2021

Contents

This documentation describes the Open Closed field line Boundary (OCB) gridding process and provides examples for usage.

CHAPTER 1

Overview

One of the challenges of working in the polar Magnetosphere-Ionosphere-Thermosphere (MIT) system is choosing an appropriate coordinate system. The ocbpy module converts between the Altitude Adjusted Corrected GeoMagnetic (AACGM) coordinates and a grid that is constructed relative to the Open Closed field line Boundary (OCB). This is particulary useful for statistical studies of the poles, where gridding relative to a fixed magnetic coordinate system would cause averaging of different physical regions, such as auroral and polar cap measurements. This coordinate system is described in the articles listed in *Citation Guide*

CHAPTER 2

Citation Guide

When publishing work that uses OCBpy, please cite both the package and the boundary data set. Specifying which version of OCBpy used will also improve the reproducibility of your presented results.

2.1 OCBpy

- Burrell, A. G., et al. (2020). aburrell/ocbpy: Version 0.2.0. Zenodo. doi:10.5281/zenodo.1217177.

```
@Misc{ocbpy,
      author = {Burrell, A. G. and Chisham, G. and Reistad, J. P.},
      title  = {aburrell/ocbpy: Version 0.2.0},
      year   = {2020},
      date   = {2020-06-10},
      doi    = {10.5281/zenodo.1179230},
      url    = {http://doi.org/10.5281/zenodo.1217177},
      }
```

This package was first described in the python in heliophysics over article, which may also be cited if a description of the package is desired.

- Burrell, A. G., et al. (2018). Snakes on a spaceship — An overview of Python in heliophysics. Journal of Geophysical Research: Space Physics, 123, 10,384–10,402. doi:10.1029/2018JA025877.

2.2 Authors

- Angeline G. Burrell - https://github.com/aburrell
- Gareth Chisham
- Jone P. Reistad - https://github.com/jpreistad

2.3 Contributors

- Dominic Jodoin - https://github.com/cotsog

2.4 IMAGE FUV Boundaries

Please cite both the papers discussing both the instrument and the boundary retrieval method.

- **SI12/SI13**: Mende, S., et al. Space Science Reviews (2000) 91: 287-318. http://doi.org/10.1023/A:1005292301251.
- **WIC**: Mende, S., et al. Space Science Reviews (2000) 91: 271-285. http://doi.org/10.1023/A:1005227915363.
- **OCB**: Chisham, G. (2017) A new methodology for the development of high-latitude ionospheric climatologies and empirical models, J. Geophys. Res. Space Physics, 122, 932–947, http://doi.org/10.1002/2016JA023235.
- **OCB**: Chisham, G. (2017) Auroral Boundary Derived from IMAGE Satellite Mission Data (May 2000 - Oct 2002), Version 1.1, Polar Data Centre, Natural Environment Research Council, UK. http://doi.org/10.5285/75aa66c1-47b4-4344-ab5d-52ff2913a61e.

2.5 AMPERE Boundaries

Please follow the AMPERE data usage requirements provided by APL and cite the R1/R2 FAC boundary retrieval method and the OCB correction method.

- **FAC**: Milan, S. E. (2019): AMPERE R1/R2 FAC radii. figshare. Dataset. https://doi.org/10.25392/leicester.data.11294861.v1
- **OCB**: Burrell, A. G., et al. (2020): AMPERE Polar Cap Boundaries, Ann. Geophys., 38, 481-490, http://doi.org/10.5194/angeo-38-481-2020

2.6 DMSP SSJ Boundaries

The DMSP SSJ boundaries are retrieved using ssj_auroral_boundary. Please follow the citation guidelines on their page. The general reference for the DMSP SSJ boundary data set is provided below.

- **SSJ Auroral Boundaries (2010-2014)**: Kilcommons, L., et al. (2019). Defense Meteorology Satellite Program (DMSP) Electron Precipitation (SSJ) Auroral Boundaries, 2010-2014 (Version 1.0.0) [Data set]. Zenodo. http://doi.org/10.5281/zenodo.3373812

CHAPTER 3

OCB Data Sets

OCBs must be obtained from observations for this coordinate transformation. The standard OCB data sets can be found in `ocbpy/boundaries`, though this location may also found using *`ocbpy.boundaries.files.get_boundary_directory()`*. Currently, three different boundary data set types are available. Some data sets also include the locations of the Equatorward Auroral oval Boundary (EAB). Routines to retrieve boundary filenames from specific instruments, time periods, hemispheres, and boundary types are provided in the *`ocbpy.boundaries.files`* sub-module.

3.1 IMAGE

Data from three auroral instruments provide northern hemisphere OCB locations for 3 May 2000 03:01:42 UT - 22 Aug 2002 00:01:28, though not all of the times included in these files contain high-quality estimations of the OCB. Recommended selection criteria are included as defaults in the *`OCBoundary`* class. You can read more about the OCB determination and this selection criteria, as well as the three auroral instruments (IMAGE Wideband Imaging Camera (WIC) and FUV Spectrographic Imagers SI12 and SI13) in the articles listed in *IMAGE FUV Boundaries*.

3.2 AMPERE

OCB data sets can also be obtained from AMPERE (Active Magnetosphere and Planetary Electrodynamics Response Experiment) R1/R2 Field-Aligned Current (FAC) boundary data. This data is provided for both hemispheres between 2010-2016, inclusive. Because there is an offset between the R1/R2 FAC boundary and the OCB, a correction is required. This correction can be implemented using the routines in *`ocbpy.ocb_correction`*. More information about the method behind the identification of these boundaries and their offset can be found in the articles listed in *AMPERE Boundaries*. Recommended selection criteria are included as defaults in the `OCBoundary` class.

3.3 DMSP SSJ

DMSP particle precipitation instruments make it possible to identify the poleward and equatorward boundaries of the auroral oval along the satellite orbit. Details about this identification process can be found in the references listed in *DMSP SSJ Boundaries*. Routines to download and proceess the DMSP boundary files are provided in the `ocbpy.boundaries.dmsp_ssj_files` sub-module.

3.4 Boundary File Module

Provide desired boundary file names

`ocbpy.boundaries.files.`**`get_boundary_directory`**`()`
Get the OCBpy boundary directory

Returns **boundary_dir** – Directory holding the boundary files included in OCBpy

Return type str

`ocbpy.boundaries.files.`**`get_boundary_files`**`(`*bound='ocb'*`)`
Get boundary filenames and their spatiotemporal ranges

Parameters **bound** (*str*) – String specifying which boundary is desired (OCB or EAB) (default='ocb')

Returns **boundary_files** – Dict with keys of boundary files containing dicts specifying the hemisphere, instrument, file start time, and file end time

Return type dict

Notes

IMAGE instruments are separated into WIC, SI12, and SI13 Unknown instruments should have filenames of the format, instrument_hemisphere_%Y%m%d_%Y%m%d.boundary

`ocbpy.boundaries.files.`**`get_default_file`**`(`*stime*, *etime*, *hemisphere*, *instrument=''*, *bound='ocb'*`)`
Get the default file for a specified time and hemisphere

Parameters

- **stime** (*dt.datetime or NoneType*) – Starting time for which the file is desired; if None, will prioritize IMAGE data for the northern and AMERE data for the southern hemisphere
- **etime** (*dt.datetime or NoneType*) – Ending time for which the file is desired; if None, will prioritize IMAGE data for the northern and AMPERE data for the southern hemisphere
- **hemisphere** (*int*) – Hemisphere for which the file is desired (1=north, -1=south)
- **instrument** (*str*) – Instrument that provides the data. This will override the starting and ending times. Accepts 'ampere', 'amp', 'image', 'si12', 'si13', 'wic', 'dmsp-ssj', and '' (to accept instrument defaults based on time range). Will also accept the instrument name for any instrument whose boundary file follows the naming convention INST_HEMI_YYYYMMDD_YYYYMMDD_*.BBB, where: INST = instrument name HEMI = north or south YYYYMMDD = starting and ending year, month, day BBB = ocb or eab (default='')

- **bound** (*str*) – String specifying which boundary is desired (OCB or EAB) (default='ocb')

Returns

- **default_file** (*str or NoneType*) – Default filename with full path defined or None if no file was available for the specified input constraints
- **instrument** (*str*) – Instrument for the default file (either 'ampere', 'image', or 'dmsp-ssj')

3.5 DMSP SSJ File Module

CHAPTER 4

OCB Gridding

OCB gridding is performed by matching observations and OCBs in Universal Time (UT) and then normalising the AACGM magnetic coordinates of the observation to OCB coordinates. This is done by determining the observation's location relative to the current OCB and placing it in the same location relative to a typical OCB that has a magnetic latitude of 74 degrees. Data matching is performed by *ocbpy.ocboundary.match_data_ocb()*. Coordinate normalisation, as well as OCB loading and data cycling is done within *ocbpy.ocboundary.OCBoundary*. These classes and functions make up the *ocbpy.ocboundary* module.

For observations that depend on the cross polar cap potential, it is also important to scale the magnitude. This ensures that the magnitudes from different sized polar caps compare to the *typical* polar cap the OCB gridding produces. For vector data, the local polar north and east components may also change. Magnitude scaling is performed by *ocbpy.ocb_scaling.normal_evar()* or *ocbpy.ocb_scaling.normal_curl_evar()*. Vector scaling, re-orientation, and OCB coordinate normalisation are performed within the class *VectorData*. These classes and functions make up the *ocbpy.ocb_scaling* module.

4.1 OCBoundary Module

Hold, manipulate, and load the open-closed field line boundary data

References

class ocbpy.ocboundary.**OCBoundary**(*filename='default'*, *instrument=''*, *hemisphere=1*, *boundary_lat=74.0*, *stime=None*, *etime=None*, *rfunc=None*, *rfunc_kwargs=None*)

Object containing open-closed field-line boundary (OCB) data

Parameters

- **filename** (*str or NoneType*) – File containing the required open-closed circle boundary data sorted by time. If NoneType, no file is loaded. If 'default', *ocbpy.boundaries.files.get_default_file* is called. (default='default')

- **instrument** (*str*) – Instrument providing the OCBoundaries. Requires 'image', 'ampere', or 'dmsp-ssj' if a file is provided. If using filename='default', also accepts 'amp', 'si12', 'si13', 'wic', and ''. (default='')
- **hemisphere** (*int*) – Integer (+/- 1) denoting northern/southern hemisphere (default=1)
- **boundary_lat** (*float*) – Typical OCBoundary latitude in AACGM coordinates. Hemisphere will give this boundary the desired sign. (default=74.0)
- **stime** (*dt.datetime or NoneType*) – First time to load data or beginning of file. If specifying time, be sure to start before the time of the data to allow the best match within the allowable time tolerance to be found. (default=None)
- **etime** (*dt.datetime or NoneType*) – Last time to load data or ending of file. If specifying time, be sure to end after the last data point you wish to match to, to ensure the best match within the allowable time tolerance is made. (default=None)
- **rfunc** (*numpy.ndarray, function, or NoneType*) – OCB radius correction function, if None will use instrument default. Function must have AACGM MLT (in hours) as argument input. To allow the boundary shape to change with univeral time, each temporal instance may have a different function (array input). If a single function is provided, will recast as an array that specifies this function for all times. (default=None)
- **rfunc_kwargs** (*numpy.ndarray, dict, or NoneType*) – Optional keyword arguements for *rfunc*. If None is specified, uses function defaults. If dict is specified, recasts as an array of this dict for all times. Array must be an array of dicts. (default=None)

records
Number of OCB records (default=0)

Type int

rec_ind
Current OCB record index (default=0; initialised=-1)

Type int

dtime
Numpy array of OCB datetimes (default=None)

Type numpy.ndarray or NoneType

phi_cent
Numpy array of floats that give the angle from AACGM midnight of the OCB pole in degrees (default=None)

Type numpy.ndarray or NoneType

r_cent
Numpy array of floats that give the AACGM co-latitude of the OCB pole in degrees (default=None)

Type numpy.ndarray or NoneType

r
Numpy array of floats that give the radius of the OCBoundary in degrees (default=None)

Type numpy.ndarray or NoneType

min_fom
Minimum acceptable figure of merit for data (default=0)

Type float

x, y, j_mag, etc.
Numpy array of floats that hold the remaining values held in *filename*

Type numpy.ndarray or NoneType

Raises `ValueError` – Incorrect or incompatible input

get_aacgm_boundary_lat(*aacgm_mlt*, *rec_ind=None*, *overwrite=False*, *set_lon=True*)
Get the OCB latitude in AACGM coordinates at specified longitudes

Parameters

- **aacgm_mlt** (*int, float, or array-like*) – AACGM longitude location(s) (in degrees) for which the OCB latitude will be calculated.
- **rec_ind** (*int, array-like, or NoneType*) – Record index for which the OCB AACGM latitude will be calculated, or None to calculate all boundary locations (default=None).
- **overwrite** (*bool*) – Overwrite previous boundary locations if this time already has calculated boundary latitudes for a different set of input longitudes (default=False).
- **set_lon** (*bool*) – Calculate the AACGM longitude of the OCB alongside the MLT (default=True).

Notes

Updates OCBoundary object with list attributes. If no boundary value is calculated at a certain time, the list is padded with None. If a boundary latitude cannot be calculated at that time and longitude, that time and longitude is filled with NaN.

aacgm_boundary_lat contains the AACGM latitude location(s) of the OCB (in degrees) for each requested time[3].

aacgm_boundary_mlt holds the aacgm_mlt input for each requested time. The requested MLT may differ from time to time, to allow easy comparison with satellite passes[3].

aacgm_boundary_lon holds the aacgm_lon input for each requested time. This is calculated from *aacgm_boundary_mlt* by default[3].

get_next_good_ocb_ind(*min_sectors=7*, *rcent_dev=8.0*, *max_r=23.0*, *min_r=10.0*)
Read in the next usuable OCB record from the data file.

Parameters

- **min_sectors** (*int*) – Minimum number of MLT sectors required for good OCB. (default=7)
- **rcent_dev** (*float*) – Maximum number of degrees between the new centre and the AACGM pole (default=8.0)
- **max_r** (*float*) – Maximum radius for open-closed field line boundary in degrees. (default=23.0)
- **min_r** (*float*) – Minimum radius for open-closed field line boundary in degrees (default=10.0)

[3] 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.

Notes

Updates self.rec_ind to the index of next good OCB record or a value greater than self.records if there aren't any more good records available after the starting point

IMAGE FUV checks that: - more than 6 MLT boundary values have contributed to OCB circle - the OCB 'pole' is with 8 degrees of the AACGM pole - the OCB 'radius' is greater than 10 and less than 23 degrees AMPERE/DMSP-SSJ checks that: - the Figure of Merit is greater than or equal to the specified minimum

inst_defaults()
Get the information needed to load an OCB file using instrument specific formatting

Returns

- **hlines** (*int*) – Number of header lines
- **ocb_cols** (*str*) – String containing the names for each data column
- **datetime_fmt** (*str*) – String containing the datetime format

Notes

Updates the min_fom attribute for AMPERE and DMSP-SSJ

load(*hlines=0*, *ocb_cols='year soy num_sectors phi_cent r_cent r a r_err'*, *datetime_fmt=''*, *stime=None*, *etime=None*)
Load the data from the specified Open-Closed Boundary file

Parameters

- **ocb_cols** (*str*) – String specifying format of OCB file. All but the first two columns must be included in the string, additional data values will be ignored. If 'year soy' aren't used, expects 'date time' in 'YYYY-MM-DD HH:MM:SS' format. (default='year soy num_sectors phi_cent r_cent r a r_err')
- **hlines** (*int*) – Number of header lines preceeding data in the OCB file (default=0)
- **datetime_fmt** (*str*) – A string used to read in 'date time' data. Not used if 'year soy' is specified. (default='')
- **stime** (*dt.datetime or NoneType*) – Time to start loading data or None to start at beginning of file. (default=None)
- **etime** (*datetime or NoneType*) – Time to stop loading data or None to end at the end of the file. (default=None)

normal_coord(*lat*, *lt*, *coords='magnetic'*, *height=350.0*, *method='ALLOWTRACE'*)
Converts position(s) to normalised co-ordinates relative to the OCB

Parameters

- **lat** (*float or array-like*) – Input latitude (degrees), must be geographic, geodetic, or AACGMV2
- **lt** (*float or array-like*) – Input local time (hours), must be solar or AACGMV2 magnetic
- **coords** (*str*) – Input coordiate system. Accepts 'magnetic', 'geocentric', or 'geodetic' (default='magnetic')
- **height** (*float or array-like*) – Height (km) at which AACGMV2 coordinates will be calculated, if geographic coordinates are provided (default=350.0)

- **method** (*str*) – String denoting which type(s) of conversion to perform, if geographic coordinates are provided. Expects either 'TRACE' or 'ALLOWTRACE'. See AACGMV2 for details[2]. (default='ALLOWTRACE')

Returns

- **ocb_lat** (*float or array-like*) – Magnetic latitude relative to OCB (degrees)
- **ocb_mlt** (*float or array-like*) – Magnetic local time relative to OCB (hours)
- **r_corr** (*float or array-like*) – Radius correction to OCB (degrees)

Notes

Approximation - Conversion assumes a planar surface

See also:

`aacgmv2()`

revert_coord(*ocb_lat*, *ocb_mlt*, *r_corr=0.0*, *coords='magnetic'*, *height=350.0*, *method='ALLOWTRACE'*)

Converts the position of a measurement in OCB into AACGM co-ordinates

Parameters

- **ocb_lat** (*float or array-like*) – Input OCB latitude in degrees
- **ocb_mlt** (*float or array-like*) – Input OCB local time in hours
- **r_corr** (*float or array-like*) – Input OCB radial correction in degrees, may be a function of AACGM MLT (default=0.0)
- **coords** (*str*) – Output coordiate system. Accepts 'magnetic', 'geocentric', or 'geodetic' (default='magnetic')
- **height** (*float or array-like*) – Geocentric height above sea level (km) at which AACGMV2 coordinates will be calculated, if geographic coordinates are desired (default=350.0)
- **method** (*str*) – String denoting which type(s) of conversion to perform, if geographic coordinates are provided. Expects either 'TRACE' or 'ALLOWTRACE'. See AACGMV2 for details[2]. (default='ALLOWTRACE')

Returns

- **lat** (*float or array-like*) – latitude (degrees)
- **lt** (*float or array-like*) – local time (hours)

Notes

Approximation - Conversion assumes a planar surface

See also:

`aacgmv2()`

ocbpy.ocboundary.**match_data_ocb**(*ocb*, *dat_dtime*, *idat=0*, *max_tol=600*, *min_sectors=7*, *rcent_dev=8.0*, *max_r=23.0*, *min_r=10.0*)

Matches data records with OCB records, locating the closest values within a specified tolerance

[2] Angeline Burrell, Christer van der Meeren, & Karl M. Laundal. (2020). aburrell/aacgmv2 (All Versions). Zenodo. doi:10.5281/zenodo.1212694.

Parameters

- **ocb** (*ocbpy.OCBoundary*) – Class containing the open-close field line boundary data
- **dat_dtime** ((*list or numpy array of datetime objects*)) – Times where data exists
- **idat** (*int*) – Current data index (default=0)
- **max_tol** (*int*) – maximum seconds between OCB and data record in sec (default=600)
- **min_sectors** (*int*) – Minimum number of MLT sectors required for good OCB. (default=7)
- **rcent_dev** (*float*) – Maximum number of degrees between the new centre and the AACGM pole (default=8.0)
- **max_r** (*float*) – Maximum radius for open-closed field line boundary in degrees (default=23.0)
- **min_r** (*float*) – Minimum radius for open-closed field line boundary in degrees (default=10.0)

Returns **idat** – Data index for match value, None if all of the data have been searched

Return type int or NoneType

Notes

Updates OCBoundary.rec_ind for matched value. None if all of the boundaries have been searched.

ocbpy.ocboundary.**retrieve_all_good_indices**(*ocb*)

Retrieve all good indices from the ocb structure

Parameters **ocb** (*ocbpy.OCBoundary*) – Class containing the open-close field line boundary data

Returns **good_ind** – List of indices containing good OCBs

Return type list

4.2 OCB Scaling Module

Scale data affected by magnetic field direction or electric field

References

class ocbpy.ocb_scaling.**VectorData**(*dat_ind*, *ocb_ind*, *aacgm_lat*, *aacgm_mlt*, *ocb_lat=nan*, *ocb_mlt=nan*, *r_corr=nan*, *aacgm_n=0.0*, *aacgm_e=0.0*, *aacgm_z=0.0*, *aacgm_mag=nan*, *dat_name=None*, *dat_units=None*, *scale_func=None*)

Object containing a vector data point

Parameters

- **dat_ind** (*int or array-like*) – Data index (zero offset)
- **ocb_ind** (*int or array-like*) – OCBoundary record index matched to this data index (zero offset)

- **aacgm_lat** (*float or array-like*) – Vector AACGM latitude (degrees)
- **aacgm_mlt** (*float or array-like*) – Vector AACGM MLT (hours)
- **ocb_lat** (*float or array-like*) – Vector OCB latitude (degrees) (default=np.nan)
- **ocb_mlt** (*float or array-like*) – Vector OCB MLT (hours) (default=np.nan)
- **aacgm_n** (*float or array-like*) – AACGM North pointing vector (positive towards North) (default=0.0)
- **aacgm_e** (*float or array-like*) – AACGM East pointing vector (completes right-handed coordinate system (default = 0.0)
- **aacgm_z** (*float or array-like*) – AACGM Vertical pointing vector (positive down) (default=0.0)
- **aacgm_mag** (*float or array-like*) – Vector magnitude (default=np.nan)
- **dat_name** (*str*) – Data name (default=None)
- **dat_units** (*str*) – Data units (default=None)
- **scale_func** (*function*) – Function for scaling AACGM magnitude with arguements: [measurement value, mesurement AACGM latitude (degrees), mesurement OCB latitude (degrees)] (default=None)

unscaled_r
Radius of polar cap in degrees

Type float or array-like

scaled_r
Radius of normalised OCB polar cap in degrees

Type float or array-like

ocb_n
OCB north component of data vector (default=np.nan)

Type float or array-like

ocb_e
OCB east component of data vector (default=np.nan)

Type float or array-like

ocb_z
OCB vertical component of data vector (default=np.nan)

Type float or array-like

ocb_mag
OCB magnitude of data vector (default=np.nan)

Type float or array-like

ocb_quad
AACGM quadrant of OCB pole (default=0)

Type int or array-like

vec_quad
AACGM quadrant of Vector (default=0)

Type int or array-like

pole_angle
Angle at vector location appended by AACGM and OCB poles in degrees (default=np.nan)

Type float or array-like

aacgm_naz
AACGM north azimuth of data vector in degrees (default=np.nan)

Type float or array-like

ocb_aacgm_lat
AACGM latitude of OCB pole in degrees (default=np.nan)

Type float or array-like

ocb_aacgm_mlt
AACGM MLT of OCB pole in hours (default=np.nan)

Type float or array-like

Notes

May only handle one data type, so scale_func cannot be an array

calc_ocb_polar_angle()
Calculate the OCB north azimuth angle

Returns **ocb_naz** – Angle between measurement vector and OCB pole in degrees

Return type float or array-like

Raises ValueError – If the required input is undefined

Notes

Requires *ocb_quad*, *vec_quad*, *aacgm_naz*, and *pole_angle*

calc_ocb_vec_sign(*north=False*, *east=False*, *quads={}*)
Get the sign of the North and East components

Parameters

- **north** (*bool*) – Get the sign of the north component(s) (default=False)
- **east** (*bool*) – Get the sign of the east component(s) (default=False)
- **quads** (*dict*) – Dictionary of boolean values or arrays of boolean values for OCB and Vector quadrants. (default=dict())

Returns **vsigns** – Dictionary with keys 'north' and 'east' containing the desired signs

Return type dict

Raises ValueError – If the required input is undefined

Notes

Requires *ocb_quad*, *vec_quad*, *aacgm_naz*, and *pole_angle*

calc_vec_pole_angle()
Calculate the angle between the AACGM pole, a measurement, and the OCB pole using spherical trigonometry

Raises ValueError – If the input is undefined or inappropriately sized arrays

Notes

Requires *aacgm_mlt*, *aacgm_lat*, *ocb_aacgm_mlt*, and *ocb_aacgm_lat*. Updates *pole_angle*.

define_quadrants()
Find the MLT quadrants (in AACGM coordinates) for the OCB pole and data vector

Notes

North (N) and East (E) are defined by the AACGM directions centred on the data vector location, assuming vertical is positive downwards Quadrants: 1 [N, E]; 2 [N, W]; 3 [S, W]; 4 [S, E]

Requires *ocb_aacgm_mlt*, *aacgm_mlt*, and *pole_angle*. Updates *ocb_quad* and *vec_quad*

Raises ValueError – If the required input is undefined

scale_vector()
Normalise a variable proportional to the curl of the electric field.

Raises ValueError – If the required input is not defined

Notes

Requires *ocb_lat*, *ocb_mlt*, *ocb_aacgm_mlt*, and *pole_angle*. Updates *ocb_n*, *ocb_e*, *ocb_z*, and *ocb_mag*

set_ocb(*ocb*, *scale_func=None*)
Set the OCBoundary values for provided data (updates all attributes)

Parameters

- **ocb** (*ocbpy.OCBoundary*) – Open Closed Boundary class object
- **scale_func** (*function*) – Function for scaling AACGM magnitude with arguements: [measurement value, mesurement AACGM latitude (degrees), mesurement OCB latitude (degrees)] Not necessary if defined earlier or no scaling is needed. (default=None)

ocbpy.ocb_scaling.**archav**(*hav*)
Formula for the inverse haversine

Parameters **hav** (*float or array-like*) – Haversine of an angle

Returns **alpha** – Angle in radians

Return type float or array-like

Notes

The input must be positive. However, any number with a magnitude below 10-16 will be rounded to zero. More negative numbers will return NaN.

ocbpy.ocb_scaling.**hav**(*alpha*)
Formula for haversine

Parameters **alpha** (*float or array-like*) – Angle in radians

Returns **hav_alpha** – Haversine of alpha, equal to the square of the sine of half-alpha

Return type float or array-like

ocbpy.ocb_scaling.**normal_curl_evar**(*curl_evar*, *unscaled_r*, *scaled_r*)
Normalise a variable proportional to the curl of the electric field

Parameters

- **curl_evar** (*float or array*) – Variable related to electric field (e.g. vorticity)
- **unscaled_r** (*float or array*) – Radius of polar cap in degrees
- **scaled_r** (*float or array*) – Radius of normalised OCB polar cap in degrees

Returns **nvar** – Normalised variable

Return type float or array

Notes

Assumes that the cross polar cap potential is fixed across the polar cap regardless of the radius of the Open Closed field line Boundary. This is commonly assumed when looking at statistical patterns that control the IMF (which accounts for dayside reconnection) and assume that the nightside reconnection influence is averaged out over the averaged period[1].

ocbpy.ocb_scaling.**normal_evar**(*evar*, *unscaled_r*, *scaled_r*)
Normalise a variable proportional to the electric field

Parameters

- **evar** (*float or array*) – Variable related to electric field (e.g. velocity)
- **unscaled_r** (*float or array*) – Radius of polar cap in degrees
- **scaled_r** (*float or array*) – Radius of normalised OCB polar cap in degrees

Returns **nvar** – Normalised variable

Return type float or array

Notes

Assumes that the cross polar cap potential is fixed across the polar cap regardless of the radius of the Open Closed field line Boundary. This is commonly assumed when looking at statistical patterns that control the IMF (which accounts for dayside reconnection) and assume that the nightside reconnection influence is averaged out over the averaged period[1].

[1] Chisham, G. (2017), A new methodology for the development of high-latitude ionospheric climatologies and empirical models, Journal of Geophysical Research: Space Physics, 122, doi:10.1002/2016JA023235.

CHAPTER 5

OCB Boundary Correction

Many high-latitude boundaries are related to each other. Both the poleward edge of the auroral oval and the R1/R2 current boundary have been successfully related to the OCB. If you have a data set of boundaries that can be related to the OCB, OCBpy is capable of applying this correction as a function of MLT. These corrections are applied using the `rfunc` and `rfunc_kwargs` keyword arguments in `ocbpy.OCBoundary` object. Several correction functions are provided as a part of *`ocbpy.ocb_correction`* module.

5.1 OCB Correction Module

Functions that specify the boundary location as a function of MLT

References

`ocbpy.ocb_correction.`**`circular`**(*mlt*, *r_add=0.0*)

Return a circular boundary correction

Parameters

- **`mlt`** (*`float or array-like`*) – Magnetic local time in hours (not actually used)
- **`r_add`** (*`float`*) – Offset added to default radius in degrees. Positive values shift the boundary equatorward, whilst negative values shift the boundary poleward. (default=0.0)

Returns **r_corr** – Radius correction in degrees at this MLT

Return type float or array-like

`ocbpy.ocb_correction.`**`elliptical`**(*mlt*, *instrument='ampere'*, *method='median'*)

Return the results of an elliptical correction to the data boundary[4]

Parameters

- **`mlt`** (*`float or array-like`*) – Magnetic local time in hours

[4] Burrell, A. G. et al.: AMPERE Polar Cap Boundaries, Ann. Geophys., 38, 481-490, doi:10.5194/angeo-38-481-2020, 2020.

- **instrument** (*str*) – Data set's instrument name (default='ampere')
- **method** (*str*) – Method used to calculate the elliptical correction, accepts 'median' or 'gaussian'. (default='median')

Returns **r_corr** – Radius correction in degrees at this MLT

Return type float or array-like

ocbpy.ocb_correction.**harmonic**(*mlt*, *instrument='ampere'*, *method='median'*)
Return the results of a harmonic fit correction to the data boundary[4]

Parameters

- **mlt** (*float or array-like*) – Magnetic local time in hours
- **instrument** (*str*) – Data set's instrument name (default='ampere')
- **method** (*str*) – Method used to determine coefficients; accepts 'median' or 'gaussian' (default='median')

Returns **r_corr** – Radius correction in degrees at this MLT

Return type float or array-like

CHAPTER 6

Supported Instrument Data Sets

Currently, support is included for files from the following sources:

1. SuperMAG (*ocbpy.instruments.supermag*)
2. SuperDARN Vorticity (*ocbpy.instruments.vort*)
3. pysat (ocbpy.instruments.pysat_instruments)

These routines may be used as a guide to write routines for other data sets. A *ocbpy.instruments.general* loading sub-module is also provided for ASCII files. All the non-boundary data routines are part of the ocbpy.instruments module. Support for time-handling that may be useful for specific data sets are provided in *ocbpy.ocb_time*.

6.1 General Instrument Module

General loading routines for data files

ocbpy.instruments.general.**load_ascii_data**(*filename*, *hlines*, *gft_kwargs={}*, *hsplit=None*, *datetime_cols=None*, *datetime_fmt=None*, *int_cols=None*, *str_cols=None*, *max_str_length=50*, *header=None*)

Load an ascii data file into a dict of numpy array

Parameters

- **filename** (*str*) – data file name
- **hlines** (*int*) – number of lines in header. If zero, must include header.
- **gft_kwargs** (*dict*) – Dictionary holding optional keyword arguments for the numpy genfromtxt routine (default=dict())
- **hsplit** (*str, NoneType*) – character seperating data labels in header. None splits on all whitespace characters. (default=None)

- **datetime_cols** (*list, NoneType*) – If there are date strings or values that should be converted to a datetime object, list them in order here. Not processed as floats. NoneType produces an empty list. (default=None)
- **datetime_fmt** (*str, NoneType*) – Format needed to convert the datetime_cols entries into a datetime object. Special formats permitted are: 'YEAR SOY', 'SOD'. 'YEAR SOY' must be used together; 'SOD' indicates seconds of day, and may be used with any date format (default=None)
- **int_cols** (*list, NoneType*) – Data that should be processed as integers, not floats. NoneType produces an empty list. (default=None)
- **str_cols** (*list, NoneType*) – Data that should be processed as strings, not floats. NoneType produces an empty list. (default=None)
- **max_str_length** (*int*) – Maximum allowed string length. (default=50)
- **header** (*list, NoneType*) – Header string(s) where the last line contains whitespace separated data names. NoneType produces an empty list. (default=None)

Returns

- **header** (*list of strings*) – Contains all specified header lines
- **out** (*dict of numpy.arrays*) – The dict keys are specified by the header data line, the data for each key are stored in the numpy array

Notes

Data is assumed to be float unless otherwise stated.

ocbpy.instruments.general.**test_file**(*filename*)

Test to ensure the file is small enough to read in

Parameters **filename** (*str*) – Filename to test

Returns **good_flag** – True if good, bad if false

Return type bool

Notes

Python can only allocate 2GB of data without crashing

6.2 SuperMAG Instrument Module

Perform OCB gridding for SuperMAG data

Notes

SuperMAG data available at: http://supermag.jhuapl.edu/

ocbpy.instruments.supermag.**load_supermag_ascii_data**(*filename*)

Load a SuperMAG ASCII data file

Parameters **filename** (*str*) – SuperMAG ASCI data file name

Returns **out** – The dict keys are specified by the header data line, the data for each key are stored in the numpy array

Return type dict

ocbpy.instruments.supermag.**supermag2ascii_ocb**(*smagfile*, *outfile*, *hemisphere=0*, *ocb=None*, *ocbfile='default'*, *instrument=''*, *max_sdiff=600*, *min_sectors=7*, *rcent_dev=8.0*, *max_r=23.0*, *min_r=10.0*)

Coverts and scales the SuperMAG data into OCB coordinates

Parameters

- **smagfile** (*str*) – file containing the required SuperMAG file sorted by time
- **outfile** (*str*) – filename for the output data
- **hemisphere** (*int*) – Hemisphere to process (can only do one at a time). 1=Northern, -1=Southern, 0=Determine from data (default=0)
- **ocb** (*ocbpy.OCBoundary or NoneType*) – OCBoundary object with data loaded from an OC boundary data file. If None, looks to ocbfile
- **ocbfile** (*str*) – file containing the required OC boundary data sorted by time, or 'default' to load default file for time and hemisphere. Only used if no OCBoundary object is supplied (default='default')
- **instrument** (*str*) – Instrument providing the OCBoundaries. Requires 'image' or 'ampere' if a file is provided. If using filename='default', also accepts 'amp', 'si12', 'si13', 'wic', and ''. (default='')
- **max_sdiff** (*int*) – maximum seconds between OCB and data record in sec (default=600)
- **min_sectors** (*int*) – Minimum number of MLT sectors required for good OCB (default=7).
- **rcent_dev** (*float*) – Maximum number of degrees between the new centre and the AACGM pole (default=8.0)
- **max_r** (*float*) – Maximum radius for open-closed field line boundary in degrees default=23.0)
- **min_r** (*float*) – Minimum radius for open-closed field line boundary in degrees (default=10.0)

Raises IOError – If unable to open the input or output file

Notes

May only process one hemisphere at a time. Scales the magnetic field observations using *ocbpy.ocb_scale.normal_curl_evar*.

See also:

ocbpy.ocb_scale.normal_curl_evar()

6.3 SuperDARN Vorticity Instrument Module

Perform OCB gridding for SuperDARN vorticity data

Notes

Specialised SuperDARN data product, available from: gchi@bas.ac.uk

ocbpy.instruments.vort.**load_vorticity_ascii_data**(*vortfile*, *save_all=False*)
Load SuperDARN vorticity data files.

Parameters

- **vortfile** (*str*) – SuperDARN vorticity file in block format
- **save_all** (*bool*) – Save all data from the file (True), or only data needed to calculate the OCB coordinates and normalised vorticity (False). (default=False)

Returns **vdata** – Dictionary of numpy arrays

Return type dict

ocbpy.instruments.vort.**vort2ascii_ocb**(*vortfile*, *outfile*, *hemisphere=0*, *ocb=None*, *ocbfile='default'*, *instrument=''*, *max_sdiff=600*, *save_all=False*, *min_sectors=7*, *rcent_dev=8.0*, *max_r=23.0*, *min_r=10.0*)
Coverts the location of vorticity data from AACGM to OCB coordinates

Parameters

- **vortfile** (*str*) – file containing the required vorticity file sorted by time
- **outfile** (*str*) – filename for the output data
- **hemisphere** (*int*) – Hemisphere to process (can only do one at a time). 1=Northern, -1=Southern, 0=Determine from data (default=0)
- **ocb** (*ocbpy.ocboundary.OCBoundary or NoneType*) – Object containing open closed boundary data or None to load from file
- **ocbfile** (*str*) – file containing the required OC boundary data sorted by time, ignorned if OCBoundary object supplied. (default='default')
- **instrument** (*str*) – Instrument providing the OCBoundaries. Requires 'image' or 'ampere' if a file is provided. If using filename='default', also accepts 'amp', 'si12', 'si13', 'wic', and ''. (default='')
- **max_sdiff** (*int*) – maximum seconds between OCB and data record in sec (default=600)
- **save_all** (*bool*) – Save all data (True), or only that needed to calcuate OCB and vorticity (False). (default=False)
- **min_sectors** (*int*) – Minimum number of MLT sectors required for good OCB. (default=7)
- **rcent_dev** (*float*) – Maximum number of degrees between the new centre and the AACGM pole (default=8.0).
- **max_r** (*float*) – Maximum radius for open-closed field line boundary in degrees. (default=23.0)
- **min_r** (*float*) – Minimum radius for open-closed field line boundary in degrees (default=10.0)

Raises

- IOError – If unable to open input or output file
- ValueError – If unable to retrieve all necessary data from the input file

Notes

Input header or col_names must include the names in the default string.

6.4 pysat Instrument Module

6.5 Time Handling Module

Routines to convert from different file timekeeping methods to datetime

ocbpy.ocb_time.**convert_time**(*year=None*, *soy=None*, *yyddd=None*, *sod=None*, *date=None*, *tod=None*, *datetime_fmt='%Y-%m-%d %H:%M:%S'*)

Convert to datetime from multiple time formats

Parameters

- **year** (*int or NoneType*) – Year or None if not in year-soy format (default=None)
- **soy** (*int or NoneType*) – Seconds of year or None if not in year-soy format (default=None)
- **yyddd** (*str or NoneType*) – String containing years since 1900 and 3-digit day of year (default=None)
- **sod** (*int, float, or NoneType*) – Seconds of day or None if the time of day is not in this format (default=None)
- **date** (*str or NoneType*) – String containing date information or None if not in date-time format (default=None)
- **tod** (*str or NoneType*) – String containing time of day information or None if not in date-time format (default=None)
- **datetime_fmt** (*str*) – String with the date-time or date format (default='%Y-%m-%d %H:%M:%S')

Returns **dtime** – Datetime object

Return type dt.datetime

ocbpy.ocb_time.**datetime2hr**(*dtime*)

Calculate hours of day from datetime

Parameters **dtime** (*dt.datetime*) – Universal time as a timestamp

Returns **uth** – Hours of day, includes fractional hours

Return type float

ocbpy.ocb_time.**deg2hr**(*lon*)

Convert from degrees to hours

Parameters **lon** (*float or array-like*) – Longitude-like value in degrees

Returns **lt** – Local time-like value in hours

Return type float or array-like

ocbpy.ocb_time.**fix_range**(*values*, *min_val*, *max_val*, *val_range=None*)

Ensure cyclic values lie below the maximum and at or above the mininum

Parameters

- **values** (*int, float, or array-like*) – Values to adjust
- **min_val** (*int or float*) – Maximum that values may not meet or exceed
- **max_val** (*int or float*) – Minimum that values may not lie below
- **val_range** (*int, float, or NoneType*) – Value range or None to calculate from min and max (default=None)

Returns **fixed_vals** – Values adjusted to lie min_val <= fixed_vals < max_val

Return type int, float, or array-like

ocbpy.ocb_time.**get_datetime_fmt_len**(*datetime_fmt*)
Get the lenght of a string line needed for a specific datetime format

Parameters **datetime_fmt** (*str*) – Formatting string used to convert between datetime and string object

Returns **str_len** – Minimum length of a string needed to hold the specified data

Return type int

Notes

See the datetime documentation for meanings of the datetime directives

ocbpy.ocb_time.**glon2slt**(*glon*, *dtime*)
Convert from geographic longitude to solar local time

Parameters

- **glon** (*float or array-like*) – Geographic longitude in degrees
- **dtime** (*dt.datetime*) – Universal time as a timestamp

Returns **slt** – Solar local time in hours

Return type float or array-like

ocbpy.ocb_time.**hr2deg**(*lt*)
Convert from degrees to hours

Parameters **lt** (*float or array-like*) – Local time-like value in hours

Returns **lon** – Longitude-like value in degrees

Return type float or array-like

ocbpy.ocb_time.**hr2rad**(*lt*)
Convert from hours to radians

Parameters **lt** (*float or array-like*) – Local time-like value in hours

Returns **lon** – Longitude-like value in radians

Return type float or array-like

ocbpy.ocb_time.**rad2hr**(*lon*)
Convert from radians to hours

Parameters **lon** (*float or array-like*) – Longitude-like value in radians

Returns **lt** – Local time-like value in hours

Return type float or array-like

ocbpy.ocb_time.**slt2glon**(*slt*, *dtime*)
Convert from solar local time to geographic longitude

Parameters

- **slt** (*float or array-like*) – Solar local time in hours
- **dtime** (*dt.datetime*) – Universal time as a timestamp

Returns **glon** – Geographic longitude in degrees

Return type float or array-like

ocbpy.ocb_time.**year_soy_to_datetime**(*yyyy*, *soy*)
Converts year and soy to datetime

Parameters

- **yyyy** (*int*) – 4 digit year
- **soy** (*float*) – seconds of year

Returns **dtime** – datetime object

Return type dt.datetime

ocbpy.ocb_time.**yyddd_to_date**(*yyddd*)
Convert from years since 1900 and day of year to datetime

Parameters **yyddd** (*str*) – String containing years since 1900 and day of year (e.g. 100126 = 2000-05-5).

Returns **dtime** – Datetime object containing date information

Return type dt.datetime

CHAPTER 7

Examples

Here are some simple examples that will show how to initialise an `OCBoundary` object, find a trustworthy open-closed boundary, convert between AACGM and OCB coordinates, and more.

7.1 Initialise an OCBoundary object

Start a python or iPython session, and begin by importing ocbpy, numpy, matplotlib, and datetime.

```
import numpy as np
import datetime as dt
import matplotlib as mpl
import matplotlib.pyplot as plt
import ocbpy
```

Next, initialise an OCB class object. This uses the default IMAGE FUV file and will take a few minutes to load.

```
ocb = ocbpy.ocboundary.OCBoundary()
print(ocb)

Open-Closed Boundary file: ~/ocbpy/ocbpy/boundaries/si13_north_circle
Source instrument: IMAGE
Open-Closed Boundary reference latitude: 74.0 degrees

219927 records from 2000-05-05 11:35:27 to 2002-08-22 00:01:28

YYYY-MM-DD HH:MM:SS Phi_Centre R_Centre R
-----------------------------------------
2000-05-05 11:35:27 356.93 8.74 9.69
2000-05-05 11:37:23 202.97 13.23 22.23
2002-08-21 23:55:20 322.60 5.49 15.36
2002-08-22 00:01:28 179.02 2.32 19.52
```

7.2 Retrieve a good OCB record

Get the first good OCB record, which will be record index 27.

```
ocb.get_next_good_ocb_ind()
print(ocb.rec_ind)
```

To get the OCB record closest to a specified time, use *match_data_ocb()*

```
first_good_time = ocb.dtime[ocb.rec_ind]
test_times = [first_good_time + dt.timedelta(minutes=5 * (i + 1))
              for i in range(5)]
itest = ocbpy.match_data_ocb(ocb, test_times, idat=0)
print(itest, ocb.rec_ind, test_times[itest], ocb.dtime[ocb.rec_ind])

0 31 2000-05-05 13:45:30 2000-05-05 13:50:29
```

7.3 Convert between AACGM and OCB coordinates

We'll start by visualising the location of the OCB using the first good OCB in the default IMAGE FUV file.

```
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.set_theta_zero_location("S")
ax.xaxis.set_ticks([0, 0.5*np.pi, np.pi, 1.5*np.pi])
ax.xaxis.set_ticklabels(["00:00", "06:00", "12:00 MLT", "18:00"])
ax.set_rlim(0,25)
ax.set_rticks([5,10,15,20])
ax.yaxis.set_ticklabels(["85$^\circ$", "80$^\circ$", "75$^\circ$",
                         "70$^\circ$"]
```

Mark the location of the circle centre in AACGM coordinates

```
ocb.rec_ind = 27
ax.plot(np.radians(ocb.phi_cent[ocb.rec_ind]), ocb.r_cent[ocb.rec_ind],
        "mx", ms=10, label="OCB Pole")
```

Calculate at plot the location of the OCB in AACGM coordinates

```
mlt = np.linspace(0.0, 24.0, num=64)
ocb.get_aacgm_boundary_lat(mlt, rec_ind=ocb.rec_ind)
theta = ocbpy.ocb_time.hr2rad(mlt)
rad = 90.0-ocb.aacgm_boundary_lat[ocb.rec_ind]
ax.plot(theta, rad, "m-", linewidth=2, label="OCB")
ax.text(theta[35], rad[35] + 1.5, "74$^\circ$", fontsize="medium", color="m")
```

Add more reference labels for OCB coordinates. Since we know the location that we want to place these labels in OCB coordinates, the *revert_coord()* method can be used to get the location in AACGM coordinates.

```
lon_clock = list()
lat_clock = list()

for ocb_mlt in np.arange(0.0, 24.0, 6.0):
    aa,oo = ocb.revert_coord(74.0, ocb_mlt)
```

(continues on next page)

(continued from previous page)

```
    lon_clock.append(oo * np.pi / 12.0)
    lat_clock.append(90.0 - aa)

ax.plot(lon_clock, lat_clock, "m+")
ax.plot([lon_clock[0], lon_clock[2]], [lat_clock[0], lat_clock[2]], "-",
        color="lightpink", zorder=1)
ax.plot([lon_clock[1], lon_clock[3]], [lat_clock[1], lat_clock[3]], "-",
        color="lightpink", zorder=1)
ax.text(lon_clock[2] + .2, lat_clock[2] + 1.0, "12:00",fontsize="medium",
        color="m")
ax.text(lon[35], olat[35] + 1.5, "82$^\circ$", fontsize="medium", color="m")
```

Now add the location of a point in AACGM coordinates, calculate the location relative to the OCB, and output both coordinates in the legend

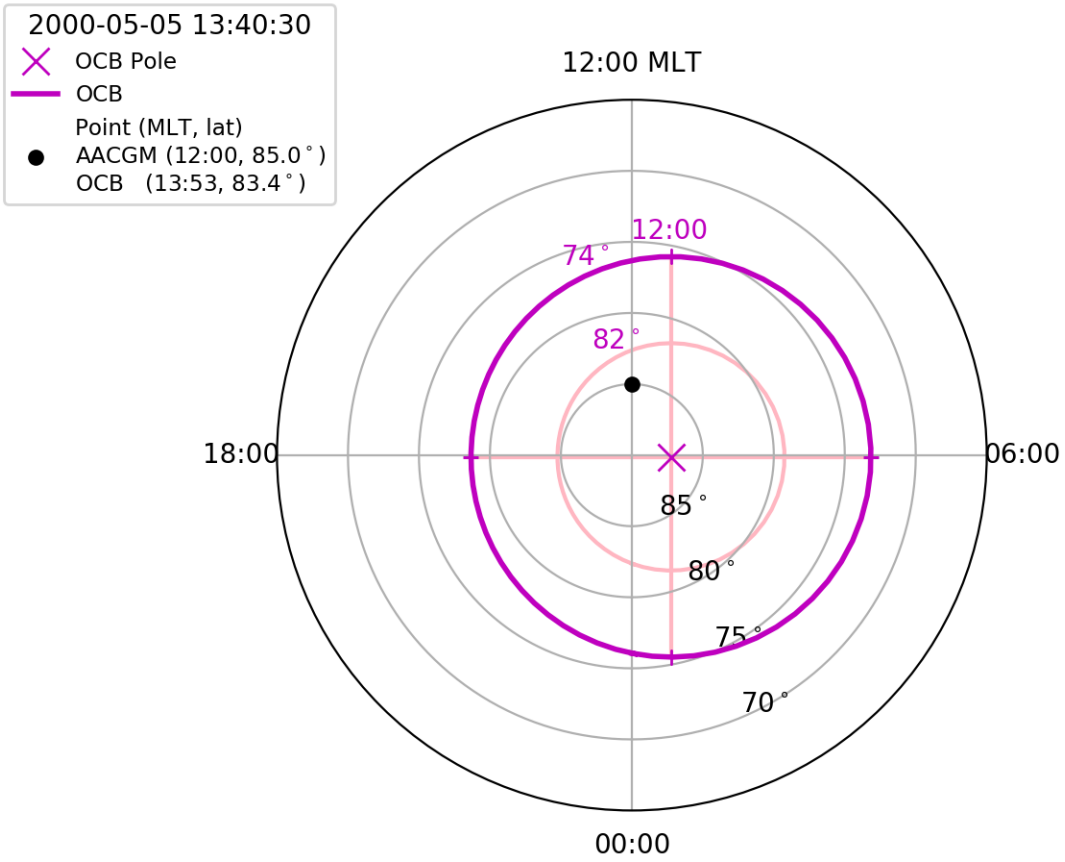
```
aacgm_lat = 85.0
aacgm_lon = np.pi
ocb_lat, ocb_mlt = ocb.normal_coord(aacgm_lat, aacgm_lon * 12.0 / np.pi)

plabel = "\n".join(["Point (MLT, lat)", "AACGM (12:00, 85.0$^\circ$)",
                    "OCB ({:.0f}:{:.0f},{:.1f}$^\circ$)".format(
                        np.floor(ocb_mlt),
                        (ocb_mlt - np.floor(ocb_mlt)) * 60.0, ocb_lat)])
ax.plot([aacgm_lon], [90.0-aacgm_lat], "ko", ms=5, label=plabel)
```

Find the location relative to the current OCB. Note that the AACGM coordinates must be in degrees latitude and hours of magnetic local time (MLT).

```
ocb_lat, ocb_mlt = ocb.normal_coord(aacgm_lat, aacgm_lon * 12.0 / np.pi)
ax.plot([ocb_mlt * np.pi / 12.0], [90.0 - ocb_lat], "mo", label="OCB Point")
```

Add a legend to finish the figure.

```
ax.legend(loc=2, fontsize="small", title="{:}".format(
    ocb.dtime[ocb.rec_ind]), bbox_to_anchor=(-0.4, 1.15))
```



Scaling of values dependent on the electric potential can be found in the *ocbpy.ocb_scaling* module.

7.4 Loading DMSP SSJ boundary files

Unlike the IMAGE and AMPERE boundaries, the DMSP SSJ boundaries are not included with the package. However, routines to obtain them are. To use them, you need to install the ssj_auroral_boundary package. Once installed, you can download DMSP SSJ data and obtain a boundary file for a specified time period using `ocbpy.boundaries.dmsp_ssj_files`. For this example, we'll use a single day. You can download the files into any directory, but this example will put them in the same directory as the other OCB files.

```
import datetime as dt
import ocbpy
import os

stime = dt.datetime(2010, 12, 31)
etime = stime + dt.datetime(days=1)
out_dir = os.path.join(os.path.split(ocbpy.__file__)[0], "boundaries")

bfiles = ocbpy.boundaries.dmsp_ssj_files.fetch_format_ssj_boundary_files(
    stime, etime, out_dir=out_dir, rm_temp=False)
```

By setting `rm_temp=False`, all of the different DMSP files will be kept in the specified output directory. You should have three CDF files (the data downloaded from each DMSP spacecraft), the CSV files (the boundaries calculated for each DMSP spacecraft) and four boundary files. The boundary files have an extention of `.eab` for the Equatorial

Auroral Boundary and `.ocb` for the Open-Closed field line Boundary. The files are separated by hemisphere, and also specify the date range. Because only one day was obtained, the start and end dates in the filename are identical. When `rm_temp=True`, the CDF and CSV files are removed.

You can now load the DMSP SSJ boundaries by specifying the desired filename, instrument, and hemisphere or merely the instrument and hemisphere.

```
# Load with filename, instrument, and hemisphere
south_file = os.path.join(out_dir,
                          "dmsp-ssj_south_20101231_20101231_v1.1.2.ocb")
ocb_south = ocbpy.ocboundary.OCBoundary(filename=south_file,
                                        instrument='dmsp-ssj', hemisphere=-1)
print(ocb_south)

Open-Closed Boundary file: ~/ocbpy/ocbpy/boundaries/dmsp-ssj_south_20101231_20101231_
↪v1.1.2.ocb
Source instrument: DMSP-SSJ
Open-Closed Boundary reference latitude: -74.0 degrees

21 records from 2010-12-31 00:27:23 to 2010-12-31 22:11:38

YYYY-MM-DD HH:MM:SS Phi_Centre R_Centre R
-------------------------------------------------------------------------------
2010-12-31 00:27:23 356.72 14.02 1.70
2010-12-31 12:27:56 324.82 0.86 0.65
2010-12-31 18:49:58 233.68 6.12 2.48
2010-12-31 22:11:38 318.60 4.64 4.26

Uses scaling function(s):
circular(**{})

# Load with date, instrument, and hemisphere
ocb_north = ocbpy.ocboundary.OCBoundary(stime=stime, instrument='dmsp-ssj',
                                        hemisphere=1)
print(ocb_north)

Open-Closed Boundary file: ~/ocbpy/ocbpy/boundaries/dmsp-ssj_north_20101231_20101231_
↪v1.1.2.ocb
Source instrument: DMSP-SSJ
Open-Closed Boundary reference latitude: 74.0 degrees

27 records from 2010-12-31 01:19:13 to 2010-12-31 23:02:48

YYYY-MM-DD HH:MM:SS Phi_Centre R_Centre R
-------------------------------------------------------------------------------
2010-12-31 01:19:13 191.07 10.69 0.54
2010-12-31 06:27:18 195.29 13.52 0.35
2010-12-31 21:21:32 259.27 2.73 2.03
2010-12-31 23:02:48 234.73 3.94 1.38

Uses scaling function(s):
circular(**{})
```

The circular scaling function with no input adds zero the the boundaries, and so performs no scaling. At this point in time, the EAO boundaries are not used, but future versions of this package will grid data relative to both the OCB and EAO boundary.

7.5 Load a general data file (DMSP SSIES)

DMSP SSIES provides commonly used polar data, which can be accessed from Madrigal, which also has a Python API called madrigalWeb. To run this example, follow the previous link(s) and download the ASCII file for F15 on 23 May 2000. Choosing the UT DMSP with quality flags for the best calcuations of ion drift and selecting ASCII instead of HDF5 will provide you with a file named **dms_ut_20000523_15.002.txt**. To load this file, use the following commands.

```
import datetime as dt
import numpy as np
import matplotlib as mlt
import matplotlib.pyplot as plt
import ocbpy

hh = ["YEAR     MONTH      DAY       HOUR      MIN       SEC       RECNO     KINDAT␣
↪   KINST       UT1_UNIX        UT2_UNIX       GDALT      GDLAT      GLON       MLAT␣
↪        MLT      ION_V_SAT_FOR ION_V_SAT_LEFT VERT_ION_V       NI           PO+     ␣
↪      PHE+         PH+        TI         TE     RPA_FLAG_UT IDM_FLAG_UT     RMS_X   ␣
↪     SIGMA_VY      SIGMA_VZ"]
dmsp_filename = "dms_ut_20000423_15.002.txt"
dmsp_head, dmsp_data = ocbpy.instruments.general.load_ascii_data(dmsp_filename, 1,␣
↪datetime_cols=[0, 1, 2, 3, 4, 5], header=hh, datetime_fmt="%Y %m %d %H %M %S", int_
↪cols=[6, 7, 8, 25, 26])

print(dmsp_data['TI'].shape, dmsp_data.keys())

(200060,), ['PH+', 'KINST', 'MIN', 'RPA_FLAG_UT', 'KINDAT', 'datetime', 'MLAT', 'UT2_
↪UNIX', 'ION_V_SAT_FOR', 'ION_V_SAT_LEFT', 'GDALT', 'UT1_UNIX', 'GDLAT', 'HOUR',
↪'PHE+', 'IDM_FLAG_UT', 'SIGMA_VZ', 'SIGMA_VY', 'SEC', 'RMS_X', 'TI', 'TE', 'DAY',
↪'GLON', 'NI', 'RECNO', 'PO+', 'MLT', 'YEAR', 'MONTH', 'VERT_ION_V']
```

In the call to *ocbpy.instruments.general.load_ascii_data()*, quality flags and number of points are saved as integers by specifying int_cols. The header may not need to be specified using `header`; have a go loading it without this keyword arguement. The results will be the same as long as there are no errors in the file header specification.

Before calculating the OCB coordinates, add space in the data dictionary for the OCB coordinates and find out which data have a good quality flag.

```
ram_key = 'ION_V_SAT_FOR'
rpa_key = 'RPA_FLAG_UT'
idm_key = 'IDM_FLAG_UT'
dmsp_data['OCB_MLT'] = np.full(shape=dmsp_data[ram_key].shape,
                               fill_value=np.nan)
dmsp_data['OCB_LAT'] = np.full(shape=dmsp_data[ram_key].shape,
                               fill_value=np.nan)
igood = np.where((dmsp_data[rpa_key] < 3) & (dmsp_data[idm_key] < 3))
print(len(igood[0]), dmsp_data[ram_key][igood].max(),
      dmsp_data[ram_key][igood].min())

7623 978.0 -2159.0
```

Now get the OCB coordinates for each location. This will not be possible everywhere, since IMAGE doesn't provide Southern Hemisphere data and only times with a good OCB established within the last 5 minutes will be used.

```
idmsp = 0
ndmsp = len(igood[0])
```

(continues on next page)

(continued from previous page)

```
ocb = ocbpy.ocboundary.OCBoundary()
ocb.get_next_good_ocb_ind()

print(idmsp, ndmsp, ocb.rec_ind, ocb.records)

0 7623 27 219927
```

This is the starting point for cycling through the records.

```
while idmsp < ndmsp and ocb.rec_ind < ocb.records:
    idmsp = ocbpy.match_data_ocb(ocb, dmsp_data['datetime'][igood],
                                 idat=idmsp, max_tol=600)
    if idmsp < ndmsp and ocb.rec_ind < ocb.records:
        nlat, nmlt, r_corr = ocb.normal_coord(
            dmsp_data['MLAT'][igood[0][idmsp]],
            dmsp_data['MLT'][igood[0][idmsp]])
        dmsp_data['OCB_LAT'][igood[0][idmsp]] = nlat
        dmsp_data['OCB_MLT'][igood[0][idmsp]] = nmlt
        idmsp += 1

igood = np.where(~np.isnan(dmsp_data['OCB_LAT']))
print(len(igood[0]), dmsp_data['OCB_LAT'][igood].max())

840 86.77820739248244
```

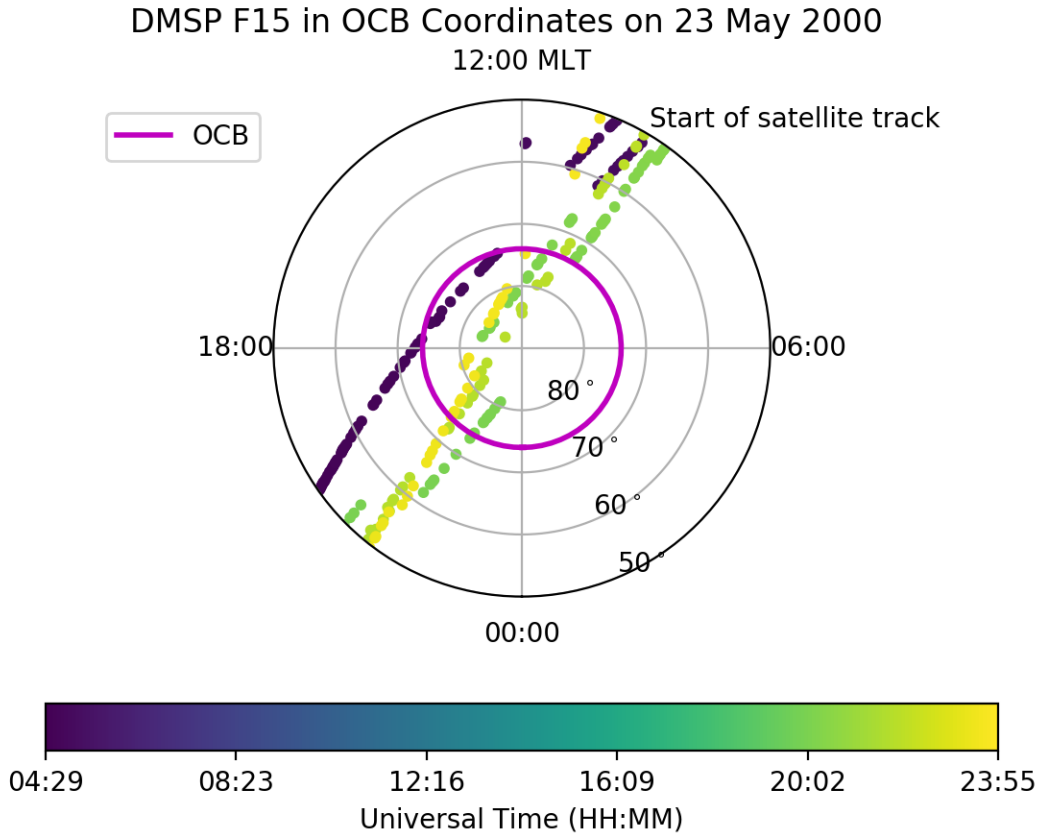
Now, let's plot the satellite track over the pole, relative to the OCB, with the location accouting for changes in the OCB at a 5 minute resolution. Note how the resolution results in apparent jumps in the satellite location. We aren't going to plot the ion velocity here, because it is provided in spacecraft coordinates rather than magnetic coordinates, adding an additional (and not intensive) level of processing.

```
fig = plt.figure()
fig.suptitle("DMSP F15 in OCB Coordinates on {:}".format(
    dmsp_data['datetime'][igood][0].strftime('%d %B %Y')))
ax = fig.add_subplot(111, projection="polar")
ax.set_theta_zero_location("S")
ax.xaxis.set_ticks([0, 0.5*np.pi, np.pi, 1.5*np.pi])
ax.xaxis.set_ticklabels(["00:00", "06:00", "12:00 MLT", "18:00"])
ax.set_rlim(0,40)
ax.set_rticks([10,20,30,40])
ax.yaxis.set_ticklabels(["80$^\circ$", "70$^\circ$", "60$^\circ$",
                         "50$^\circ$"])

lon = np.arange(0.0, 2.0 * np.pi + 0.1, 0.1)
lat = np.ones(shape=lon.shape) * (90.0 - ocb.boundary_lat)
ax.plot(lon, lat, "m-", linewidth=2, label="OCB")

dmsp_lon = dmsp_data['OCB_MLT'][igood] * np.pi / 12.0
dmsp_lat = 90.0 - dmsp_data['OCB_LAT'][igood]
dmsp_time = mpl.dates.date2num(dmsp_data['datetime'][igood])
ax.scatter(dmsp_lon, dmsp_lat, c=dmsp_time, cmap=mpl.cm.get_cmap("viridis"),
           marker="o", s=10)
ax.text(10 * np.pi / 12.0, 41, "Start of satellite track")

tticks = np.linspace(dmsp_time.min(), dmsp_time.max(), 6, endpoint=True)
dticks = ["{:s}".format(mpl.dates.num2date(tval).strftime("%H:%M"))
          for tval in tticks]
```

(continues on next page)

(continued from previous page)

```
cb = fig.colorbar(ax.collections[0], ax=ax, ticks=tticks,
                  orientation='horizontal')
cb.ax.set_xticklabels(dticks)
cb.set_label('Universal Time (HH:MM)')
ax.legend(fontsize='medium', bbox_to_anchor=(0.0,1.0))
```



7.6 Grid and scale vector data

Many space science observations, such as ion drifts, are vectors. The *ocbpy.ocb_scaling.VectorData* class ensures that the vector location, direction, and magnitude are gridded and scaled appropriately.

The example presented here uses SuperDARN data. The example file, **20010214.0100.00.pgr.grd** may be obtained by fitting and then gridding the rawacf file, available from any of the SuperDARN mirrors. FitACF v3.0 was used to create this file. See the Radar Software Toolkit for more information.

The SuperDARN data may be read in python using pydarn. To load this file (or any other grid file), use the following commands.

```
import datetime as dt
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import aacgmv2
import ocbpy
import pydarn

filename = '20010214.0100.00.pgr.grd'
sd_read = pydarn.SDarnRead(filename)
grd_data = sd_read.read_grid()
print(len(grd_data))

13
```

If you used the same file, there will be 13 grid records. Next, load the OCBs for the northern hemisphere (PGR is a Canadian radar) and the period of time available within this file.

```
stime = dt.datetime(grd_data[0]['start.year'], grd_data[0]['start.month'],
                    grd_data[0]['start.day'], grd_data[0]['start.hour'],
                    grd_data[0]['start.minute'],
                    int(np.floor(grd_data[0]['start.second'])))
etime = dt.datetime(grd_data[-1]['start.year'], grd_data[-1]['start.month'],
                    grd_data[-1]['start.day'], grd_data[-1]['start.hour'],
                    grd_data[-1]['start.minute'],
                    int(np.floor(grd_data[-1]['start.second'])))
ocb = ocbpy.ocboundary.OCBoundary(stime=stime, etime=etime)
print(ocb)

Open-Closed Boundary file: ~/ocbpy/ocbpy/boundaries/si13_north_circle.ocb
Source instrument: IMAGE
Open-Closed Boundary reference latitude: 74.0 degrees

12 records from 2001-02-14 01:33:24 to 2001-02-14 01:55:54

YYYY-MM-DD HH:MM:SS Phi_Centre R_Centre R
-----------------------------------------
2001-02-14 01:33:24 162.58 1.41 16.16
2001-02-14 01:35:26 176.90 1.77 16.16
2001-02-14 01:53:51 311.82 2.51 14.44
2001-02-14 01:55:54 193.16 2.67 15.63

Uses scaling function(s):
circular(**{})
```

To convert this vector into OCB coordinates, we first need to pair the grid record to an approriate boundary. In this instance, the first record in each case is appropriate.

```
ocb.get_next_good_ocb_ind()
print(ocb.dtime[ocb.rec_ind]-stime)

0:01:24
```

If you are using a different file, you can use *match_data_ocb()* to find an appropriate pairing, as illustrated in previous examples.

Now that the data are paired, we can initialize an *ocbpy.ocb_scaling.VectorData* object. To do this, however, we need the SuperDARN LoS velocity data in North-East-Vertical coordinates. SuperDARN grid files determine the median magnitude and direction of Line-of-Sight (LoS) Doppler velocities. For each period of time, the velocity is expressed in terms of a median vector magnitude and an angle off of magnetic north. To convert between the two,

use the following routine.

```
def kvect_to_ne(kvect, vmag):
    drifts_n = vmag * np.cos(np.radians(kvect))
    drifts_e = vmag * np.sin(np.radians(kvect))
    return drifts_e, drifts_n

# Calculate the drift components
drifts_e, drifts_n = kvect_to_ne(grd_data[0]['vector.kvect'],
                                 grd_data[0]['vector.vel.median'])

# Create an array of the data indices
dat_ind = np.arange(0, len(grd_data[0]['vector.kvect']))

# Calculate the magnetic local time from the magnetic longitude
mlt = aacgmv2.convert_mlt(grd_data[0]['vector.mlon'], stime)

# Initialize the vector data object
pgr_vect = ocbpy.ocb_scaling.VectorData(
    dat_ind, ocb.rec_ind, grd_data[0]['vector.mlat'], mlt,
    aacgm_n=drifts_n, aacgm_e=drifts_e,
    aacgm_mag=grd_data[0]['vector.vel.median'], dat_name='LoS Velocity',
    dat_units='m s$^{-1}$', scale_func=ocbpy.ocb_scaling.normal_curl_evar)

# Calculate the OCB coordinates of the vector data
pgr_vect.set_ocb(ocb)
```

Because there are 110 vectors at this time and location, printing `pgr_vect` will create a long string! Vector data does not require array input, but does allow it to reduce the time needed for calculating data observed at the same time. A better way to visualise the array of vector velocity data is to plot it. The following code will create a figure that shows the AACGMV2 velocities on the left and the OCB velocities on the right. Because data from only one radar is plotted, only a fraction of the polar region is plotted.

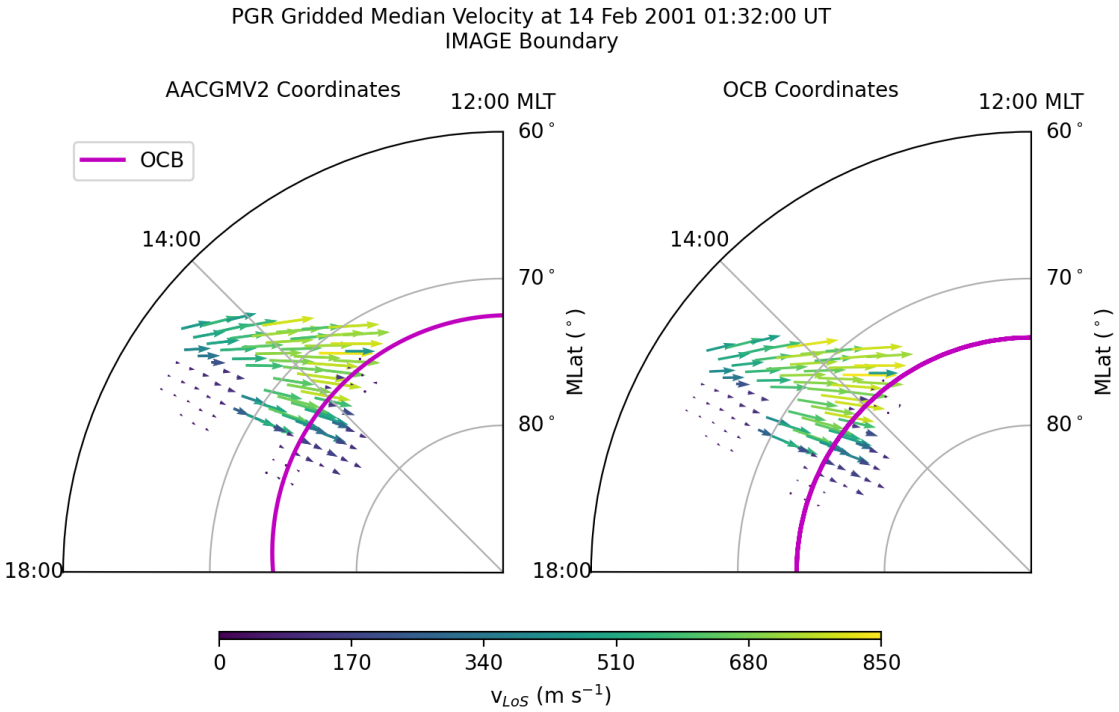
```
# Initialize the figure and axes
fig = plt.figure(figsize([8.36, 4.8]))
fig.subplots_adjust(wspace=.2, top=.95, bottom=.05)
axa = fig.add_subplot(1,2,1, projection='polar')
axo = fig.add_subplot(1,2,2, projection='polar')

# Format the axes
xticks = np.linspace(0, 2.0 * np.pi, 9)
for aa in [axa, axo]:
    aa.set_theta_zero_location('S')
    aa.xaxis.set_ticks(xticks)
    aa.xaxis.set_ticklabels(["{:02d}:00{:s}".format(int(tt), ' MLT'
                                                    if tt == 12.0 else '')
                             for tt in ocbpy.ocb_time.rad2hr(xticks)])
    aa.set_rlim(0, 30)
    aa.set_rticks([10, 20, 30])
    aa.yaxis.set_ticklabels(["80$^\circ$", "70$^\circ$", "60$^\circ$"])
    aa.set_thetamin(180)
    aa.set_thetamax(270)
    aa.set_ylabel('MLat ($^\circ$)', labelpad=30)
    aa.yaxis.set_label_position('right')

fig.suptitle(
    'PGR Gridded Median Velocity at {:} UT\n{:s} Boundary'.format(
        stime.strftime('%d %b %Y %H:%M:%S'), ocb.instrument.upper()),
```

(continues on next page)

(continued from previous page)

```
    fontsize='medium')
axa.set_title('AACGMV2 Coordinates', fontsize='medium')
axo.set_title('OCB Coordinates', fontsize='medium')

# Get and plot the OCB
xmlt = np.arange(0.0, 24.1, .1)
blat, bmlt = ocb.revert_coord(ocb.boundary_lat, xmlt)
axa.plot(ocbpy.ocb_time.hr2rad(bmlt), 90.0-blat , 'm-', lw=2, label='OCB')
axo.plot(xmlt, 90.0-np.full(shape=xmlt.shape, fill_value=ocb.boundary_lat),
         'm-', lw=2, label='OCB')

# Get and plot the gridded LoS velocities.  The quiver plot requires these
# in Cartesian coordinates
def ne_to_xy(mlt, vect_n, vect_e):
    theta = ocbpy.ocb_time.hr2rad(mlt) - 0.5 * np.pi
    drifts_x = -vect_n * np.cos(theta) - vect_e * np.sin(theta)
    drifts_y = -vect_n * np.sin(theta) + vect_e * np.cos(theta)
    return drifts_x, drifts_y

adrift_x, adrift_y = ne_to_xy(mlt, drifts_n, drifts_e)
odrift_x, odrift_y = ne_to_xy(pgr_vect.ocb_mlt, pgr_vect.ocb_n,
                              pgr_vect.ocb_e)

vmin = 0.0
vmax = 850.0
vnorm = mpl.colors.Normalize(vmin, vmax)

axa.quiver(ocbpy.ocb_time.hr2rad(mlt), 90.0 - grd_data[i]['vector.mlat'],
           adrift_x, adrift_y, grd_data[i]['vector.vel.median'], norm=vnorm)
axo.quiver(ocbpy.ocb_time.hr2rad(pgr_vect.ocb_mlt), 90.0 - pgr_vect.ocb_lat,
           odrift_x, odrift_y, pgr_vect.ocb_mag, norm=vnorm)

# Add a colour bar
cax = fig.add_axes([.25, .1, .53, .01])
cb = fig.colorbar(axa.collections[0], cax=cax,
                  ticks=np.linspace(vmin, vmax, 6, endpoint=True),
                  orientation='horizontal')
cb.set_label('v$_{LoS}$ (m s$^{-1}$)')
```

After displaying or saving this file, the results shoud look like the figure shown below. Note how the velocities increase as the beam directions align more closely with the direction of convection. However, across all beams the speeds inside the OCB are slow while those outside (in the auroral oval) are fast. The location and direction of the vectors have only shifted to maintain their position relative to the OCB. The magnitude has also been scaled, but the influence is small.


PGR Gridded Median Velocity at 14 Feb 2001 01:32:00 UT
IMAGE Boundary
AACGMV2 Coordinates
OCB Coordinates
12:00 MLT
14:00
18:00
60°
70°
80°
MLat (°)
OCB
0
170
340
510
680
850
v_{LoS} (m s^{-1})

CHAPTER 8

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.

8.1 Short version

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

8.2 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

8.3 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 :)

8.4 Development

To set up *ocbpy* for local development:

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

```
git clone git@github.com:your_name_here/ocbpy.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 into the `ocbpy/tests/` directory, either in the appropriately named file (for changes to an existing file) or in a new file (that should share the name of the new file, prepended by `test_`. The tests use unittest. Changes or additions to the documentation (located in `docs`) should also be made at this time.

4. When you're done making changes, run the tests locally before submitting a pull request
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.

8.4.1 Pull Request Guidelines

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

Do not merge any pull requests, the local maintainer is in charge of merging until this project grows.

8.4.2 Tips

To run a subset of tests from the test directory for a specific environment:

```
python test_name.py
```

To run all the tests for a specific environment:

```
python setup.py test
```

8.5 Contributor Covenant Code of Conduct

8.5.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors 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 sexual identity and orientation.

8.5.2 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
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.5.3 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.

8.5.4 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.

8.5.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at angeline.burrell@nrl.navy.mil. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. 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.

8.5.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant [homepage], version [1.4].

8.6 Changelog

Summary of all changes made since the first stable release

8.6.1 0.2.X (XX-XX-2021)

- REL: Added a .zenodo.json file
- DOC: Improved the PEP8 compliance in the documentation examples
- DOC: Improved the docstring numpydoc compliance
- DOC: Updated cross-referencing and added missing API sections
- BUG: Fixed header initialization error general instrument loading routine
- ENH: Added a setup configuration file
- ENH: Changed class *__repr__* to produce a string *eval* can use as input
- MAINT: Removed support for Python 2.7 and 3.5
- MAINT: Improved PEP8 compliance
- MAINT: Updated pysat routines to v3.0.0 standards
- TST: Integrated Requires.io
- TST: Added flake8 tests to Travis CI
- TST: Moved all configurations to setup.cfg, removing .coveragecfg
- TST: Added pysat xarray tests to the pysat test suite
- TST: Added new unit tests for *__repr__* methods

8.6.2 0.2.1 (11-24-2020)

- DOC: Updated examples in README
- BUG: Fixed an error in determining the sign and direction of OCB vectors
- STY: Changed a ValueError in VectorData to logger warning

8.6.3 0.2.0 (10-08-2020)

- First stable release

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Bibliography

[homepage] http://contributor-covenant.org

[1.4] http://contributor-covenant.org/version/1/4/

Python Module Index

o

Index