Python Magnetic Vectors Documentation

Release 0.5.3

Russell Stoneback

Jun 23, 2020

Contents

CHAPTER 1

API

Supporting routines for coordinate conversions as well as vector operations and transformations used in Space Science.

OMMBV._core.**apex_distance_after_footpoint_step**(*glats*, *glons*, *alts*, *dates*, *direction*, *vector_direction*, *step_size=None*, *max_steps=None*, *steps=None*, *edge_length=25.0*, *edge_steps=5*, *ecef_input=False*)

Calculates the distance between apex locations after stepping along vector_direction.

Using the input location, the footpoint location is calculated. From here, a step along both the positive and negative vector_directions is taken, and the apex locations for those points are calculated. The difference in position between these apex locations is the total centered distance between magnetic field lines at the magnetic apex when starting from the footpoints with a field line half distance of edge_length.

Parameters

- **glats** (*list-like of floats (degrees)*) – Geodetic (WGS84) latitude
- **glons** (*list-like of floats (degrees)*) – Geodetic (WGS84) longitude
- **alts** (*list-like of floats (km)*) – Geodetic (WGS84) altitude, height above surface
- **dates** (*list-like of datetimes*) – Date and time for determination of scalars
- **direction** (*string*) – 'north' or 'south' for tracing through northern or southern footpoint locations
- **vector_direction** (*string*) – 'meridional' or 'zonal' unit vector directions
- **step_size** (*float (km)*) – Step size (km) used for field line integration
- **max_steps** (*int*) – Number of steps taken for field line integration
- **steps** (*np.array*) – Integration steps array passed to full_field_line, np.arange(max_steps+1)

- **edge_length** (*float (km)*) – Half of total edge length (step) taken at footpoint location. edge_length step in both positive and negative directions.
- **edge_steps** (*int*) – Number of steps taken from footpoint towards new field line in a given direction (positive/negative) along unit vector
- **ecef_input** (*bool (False)*) – If True, latitude, longitude, and altitude are treated as ECEF positions (km).

Returns A closed loop field line path through input location and footpoint in northern/southern hemisphere and back is taken. The return edge length through input location is provided.

Return type np.array,

Note: vector direction refers to the magnetic unit vector direction

---

OMMBV._core.**apex_distance_after_local_step**(*glats*, *glons*, *alts*, *dates*, *vector_direction*, *edge_length=25.0*, *edge_steps=5*, *ecef_input=False*, *return_geodetic=False*)

Calculates the distance between apex locations mapping to the input location.

Using the input location, the apex location is calculated. Also from the input location, a step along both the positive and negative vector_directions is taken, and the apex locations for those points are calculated. The difference in position between these apex locations is the total centered distance between magnetic field lines at the magnetic apex when starting locally with a field line half distance of edge_length.

Parameters

- **glats** (*list-like of floats (degrees)*) – Geodetic (WGS84) latitude
- **glons** (*list-like of floats (degrees)*) – Geodetic (WGS84) longitude
- **alts** (*list-like of floats (km)*) – Geodetic (WGS84) altitude, height above surface
- **dates** (*list-like of datetimes*) – Date and time for determination of scalars
- **vector_direction** (*string*) – 'meridional' or 'zonal' unit vector directions
- **edge_length** (*float (km)*) – Half of total edge length (step) taken at footpoint location. edge_length step in both positive and negative directions.
- **edge_steps** (*int*) – Number of steps taken from footpoint towards new field line in a given direction (positive/negative) along unit vector

Returns The change in field line apex locations.

Return type np.array

Note: vector direction refers to the magnetic unit vector direction

---

OMMBV._core.**apex_location_info**(*glats*, *glons*, *alts*, *dates*, *step_size=100.0*, *fine_step_size=1e-05*, *fine_max_steps=5*, *return_geodetic=False*, *ecef_input=False*)

Determine apex location for the field line passing through input point.

Employs a two stage method. A broad step (step_size) field line trace spanning Northern/Southern footpoints is used to find the location with the largest geodetic (WGS84) height. A binary search higher resolution trace (goal fine_step_size) is then used to get a better fix on this location. Each loop, step_size halved. Greatest geodetic height is once again selected once the step_size is below fine_step_size.

Parameters

- **glats** (*list-like of floats (degrees)*) – Geodetic (WGS84) latitude
- **glons** (*list-like of floats (degrees)*) – Geodetic (WGS84) longitude
- **alts** (*list-like of floats (km)*) – Geodetic (WGS84) altitude, height above surface
- **dates** (*list-like of datetimes*) – Date and time for determination of scalars
- **step_size** (*float (100. km)*) – Step size (km) used for tracing coarse field line
- **fine_step_size** (*float (1.E-5 km)*) – Fine step size for refining apex location height
- **fine_max_steps** (*int (1.E-5 km)*) – Fine number of steps passed along to full_field_trace. Do not change unless youknow exactly what you are doing.
- **return_geodetic** (*bool*) – If True, also return location in geodetic coordinates
- **ecef_input** (*bool*) – If True, glats, glons, and alts are treated as x, y, z (ECEF).

Returns

- *(float, float, float, float, float, float)* – ECEF X (km), ECEF Y (km), ECEF Z (km),
- *if return_geodetic, also includes* – Geodetic Latitude (degrees), Geodetic Longitude (degrees), Geodetic Altitude (km)

OMMBV._core.**calculate_geomagnetic_basis**(*latitude*, *longitude*, *altitude*, *datetimes*)

Calculates local geomagnetic basis vectors and mapping scalars.

Thin wrapper around calculate_mag_drift_unit_vectors_ecef set to default parameters and with more organization of the outputs.

Parameters

- **latitude** (*array-like of floats (degrees) [-90., 90]*) – Latitude of location, degrees, WGS84
- **longitude** (*array-like of floats (degrees) [-180., 360.]*) – Longitude of location, degrees, WGS84
- **altitude** (*array-like of floats (km)*) – Altitude of location, height above surface, WGS84
- **datetimes** (*array-like of datetimes*) – Time to calculate vectors

Returns

zon_x (y,z): zonal unit vector along ECEF X, Y, and Z directions fa_x (y,z): field-aligned unit vector along ECEF X, Y, and Z directions mer_x (y,z): meridional unit vector along ECEF X, Y, and Z directions

d_zon_mag: D zonal vector magnitude d_fa_mag: D field-aligned vector magnitude d_mer_mag: D meridional vector magnitude

d_zon_x (y,z) : D zonal vector components along ECEF X, Y, and Z directions d_mer_x (y,z) : D meridional vector components along ECEF X, Y, and Z directions d_fa_x (y,z) : D field aligned vector components along ECEF X, Y, and Z directions

e_zon_mag: E zonal vector magnitude e_fa_mag: E field-aligned vector magnitude e_mer_mag: E meridional vector magnitude

e_zon_x (y,z) : E zonal vector components along ECEF X, Y, and Z directions e_mer_x (y,z) : E meridional vector components along ECEF X, Y, and Z directions e_fa_x (y,z) : E field aligned vector components along ECEF X, Y, and Z directions

Return type dict

OMMBV._core.**calculate_integrated_mag_drift_unit_vectors_ecef**(*latitude*, *longitude*, *altitude*, *datetimes*, *steps=None*, *max_steps=1000*, *step_size=100.0*, *ref_height=120.0*, *filter_zonal=True*)

Calculates field line integrated geomagnetic basis vectors.

Unit vectors are expressed in ECEF coordinates.

Parameters

- **latitude** (*array-like of floats (degrees)*) – Latitude of location, degrees, WGS84
- **longitude** (*array-like of floats (degrees)*) – Longitude of location, degrees, WGS84
- **altitude** (*array-like of floats (km)*) – Altitude of location, height above surface, WGS84
- **datetimes** (*array-like of datetimes*) – Time to calculate vectors
- **max_steps** (*int*) – Maximum number of steps allowed for field line tracing
- **step_size** (*float*) – Maximum step size (km) allowed when field line tracing
- **ref_height** (*float*) – Altitude used as cutoff for labeling a field line location a footpoint
- **filter_zonal** (*bool*) – If True, removes any field aligned component from the calculated zonal unit vector. Resulting coordinate system is not-orthogonal.

Returns

Return type zon_x, zon_y, zon_z, fa_x, fa_y, fa_z, mer_x, mer_y, mer_z

---

Note: The zonal vector is calculated by field-line tracing from the input locations toward the footpoint locations at ref_height. The cross product of these two vectors is taken to define the plane of the magnetic field. This vector is not always orthogonal with the local field-aligned vector (IGRF), thus any component of the zonal vector with the field-aligned direction is removed (optional). The meridional unit vector is defined via the cross product of the zonal and field-aligned directions.

OMMBV._core.**calculate_mag_drift_unit_vectors_ecef**(*latitude*, *longitude*, *altitude*, *datetimes*, *step_size=2.0*, *tol=0.0001*, *tol_zonal_apex=0.0001*, *max_loops=100*, *ecef_input=False*, *centered_diff=True*, *full_output=False*, *include_debug=False*, *scalar=1.0*, *edge_steps=1*, *dstep_size=2.0*, *max_steps=None*, *ref_height=None*, *steps=None*)

Calculates local geomagnetic basis vectors and mapping scalars.

Zonal - Generally Eastward (+East); lies along a surface of constant apex height Field Aligned - Generally Northward (+North); points along geomagnetic field Meridional - Generally Vertical (+Up); points along the gradient in apex height

The apex height is the geodetic height of the field line at its highest point. Unit vectors are expressed in ECEF coordinates.

Parameters

- **latitude** (*array-like of floats (degrees) [-90., 90]*) – Latitude of location, degrees, WGS84
- **longitude** (*array-like of floats (degrees) [-180., 360.]*) – Longitude of location, degrees, WGS84
- **altitude** (*array-like of floats (km)*) – Altitude of location, height above surface, WGS84
- **datetimes** (*array-like of datetimes*) – Time to calculate vectors
- **step_size** (*float*) – Step size (km) to use when calculating changes in apex height
- **tol** (*float*) – Tolerance goal for the magnitude of the change in unit vectors per loop
- **tol_zonal_apex** (*Maximum allowed change in apex height along*) – zonal direction
- **max_loops** (*int*) – Maximum number of iterations
- **ecef_input** (*bool (False)*) – If True, inputs latitude, longitude, altitude are interpreted as x, y, and z in ECEF coordinates (km).
- **full_output** (*bool (False)*) – If True, return an additional dictionary with the E and D mapping vectors
- **include_deubg** (*bool (False)*) – If True, include stats about iterative process in optional dictionary. Requires full_output=True
- **centered_diff** (*bool (True)*) – If True, a symmetric centered difference is used when calculating the change in apex height along the zonal direction, used within the zonal unit vector calculation
- **scalar** (*int*) – Used to modify unit magnetic field within algorithm. Generally speaking, this should not be modified
- **edge_steps** (*int (1)*) – Number of steps taken when moving across field lines and calculating the change in apex location. This parameter impacts both runtime and accuracy of the D, E vectors.
- **dstep_size** (*float (016 km)*) – Step size (km) used when calculting the expansion of field line surfaces. Generally, this should be the same as step_size.
- **max_steps** (*int*) – Deprecated
- **ref_height** (*float*) – Deprecated
- **steps** (*list-like*) – Deprecated

Returns

- *zon_x, zon_y, zon_z, fa_x, fa_y, fa_z, mer_x, mer_y, mer_z, (optional dictionary)*
- *Optional output dictionary*
- ———————

- *Full Output Parameters*
- **d_zon_x (y,z)** (*D zonal vector components along ECEF X, Y, and Z directions*)
- **d_mer_x (y,z)** (*D meridional vector components along ECEF X, Y, and Z directions*)
- **d_fa_x (y,z)** (*D field aligned vector components along ECEF X, Y, and Z directions*)
- **e_zon_x (y,z)** (*E zonal vector components along ECEF X, Y, and Z directions*)
- **e_mer_x (y,z)** (*E meridional vector components along ECEF X, Y, and Z directions*)
- **e_fa_x (y,z)** (*E field aligned vector components along ECEF X, Y, and Z directions*)

Debug Parameters

diff_mer_apex : rate of change in apex height (km) along meridional vector diff_mer_vec : magnitude of vector change for last loop diff_zonal_apex : rate of change in apex height (km) along zonal vector diff_zonal_vec : magnitude of vector change for last loop loops : Number of loops vector_seed_type : Initial vector used for starting calculation (deprecated)

Note: The zonal and meridional vectors are calculated by using the observed apex-height gradient to rotate a pair of vectors orthogonal to eachother and the geomagnetic field such that one points along no change in apex height (zonal), the other along the max (meridional). The rotation angle theta is given by

Tan(theta) = apex_height_diff_zonal/apex_height_diff_meridional

The method terminates when successive updates to both the zonal and meridional unit vectors differ (magnitude of difference) by less than tol, and the change in apex_height from input location is less than tol_zonal_apex.

OMMBV._core.**cross_product**(*x1*, *y1*, *z1*, *x2*, *y2*, *z2*)

Cross product of two vectors, v1 x v2

Parameters

- **x1** (*float or array-like*) – X component of vector 1
- **y1** (*float or array-like*) – Y component of vector 1
- **z1** (*float or array-like*) – Z component of vector 1
- **x2** (*float or array-like*) – X component of vector 2
- **y2** (*float or array-like*) – Y component of vector 2
- **z2** (*float or array-like*) – Z component of vector 2

Returns Unit vector x,y,z components

Return type x, y, z

OMMBV._core.**ecef_to_enu_vector**(*x*, *y*, *z*, *glat*, *glong*)

Converts vector from ECEF X,Y,Z components to East, North, Up

Position of vector in geospace may be specified in either geocentric or geodetic coordinates, with corresponding expression of the vector using radial or ellipsoidal unit vectors.

Parameters

- **x** (*float or array-like*) – ECEF-X component of vector
- **y** (*float or array-like*) – ECEF-Y component of vector
- **z** (*float or array-like*) – ECEF-Z component of vector
- **latitude** (*float or array_like*) – Geodetic or geocentric latitude (degrees)

- **longitude** (*float or array_like*) – Geodetic or geocentric longitude (degrees)

Returns Vector components along east, north, and up directions

Return type east, north, up

OMMBV._core.**ecef_to_geocentric**(*x*, *y*, *z*, *ref_height=None*)
Convert ECEF into geocentric coordinates

Parameters

- **x** (*float or array_like*) – ECEF-X in km
- **y** (*float or array_like*) – ECEF-Y in km
- **z** (*float or array_like*) – ECEF-Z in km
- **ref_height** (*float or array_like*) – Reference radius used for calculating height. Defaults to average radius of 6371 km

Returns numpy arrays of locations in degrees, degrees, and km

Return type latitude, longitude, altitude

OMMBV._core.**enu_to_ecef_vector**(*east*, *north*, *up*, *glat*, *glong*)
Converts vector from East, North, Up components to ECEF

Position of vector in geospace may be specified in either geocentric or geodetic coordinates, with corresponding expression of the vector using radial or ellipsoidal unit vectors.

Parameters

- **east** (*float or array-like*) – Eastward component of vector
- **north** (*float or array-like*) – Northward component of vector
- **up** (*float or array-like*) – Upward component of vector
- **latitude** (*float or array_like*) – Geodetic or geocentric latitude (degrees)
- **longitude** (*float or array_like*) – Geodetic or geocentric longitude (degrees)

Returns Vector components along ECEF x, y, and z directions

Return type x, y, z

OMMBV._core.**field_line_trace**(*init*, *date*, *direction*, *height*, *steps=None*, *max_steps=10000.0*, *step_size=10.0*, *recursive_loop_count=None*, *recurse=True*, *min_check_flag=False*)
Perform field line tracing using IGRF and scipy.integrate.odeint.

Parameters

- **init** (*array-like of floats*) – Position to begin field line tracing from in ECEF (x,y,z) km
- **date** (*datetime or float*) – Date to perform tracing on (year + day/365 + hours/24. + etc.) Accounts for leap year if datetime provided.
- **direction** (*int*) – 1 : field aligned, generally south to north. -1 : anti-field aligned, generally north to south.
- **height** (*float*) – Altitude to terminate trace, geodetic WGS84 (km)
- **steps** (*array-like of ints or floats*) – Number of steps along field line when field line trace positions should be reported. By default, each step is reported; steps=np.arange(max_steps).

- **max_steps** (*float*) – Maximum number of steps along field line that should be taken
- **step_size** (*float*) – Distance in km for each large integration step. Multiple substeps are taken as determined by scipy.integrate.odeint

Returns 2D array. [0,:] has the x,y,z location for initial point [:,0] is the x positions over the integration. Positions are reported in ECEF (km).

Return type numpy array

OMMBV._core.**footpoint_location_info**(*glats*, *glons*, *alts*, *dates*, *step_size=100.0*, *num_steps=1000*, *return_geodetic=False*, *ecef_input=False*)

Return ECEF location of footpoints in Northern/Southern hemisphere

Parameters

- **glats** (*list-like of floats (degrees)*) – Geodetic (WGS84) latitude
- **glons** (*list-like of floats (degrees)*) – Geodetic (WGS84) longitude
- **alts** (*list-like of floats (km)*) – Geodetic (WGS84) altitude, height above surface
- **dates** (*list-like of datetimes*) – Date and time for determination of scalars
- **step_size** (*float (100. km)*) – Step size (km) used for tracing coarse field line
- **num_steps** (*int (1.E-5 km)*) – Number of steps passed along to field_line_trace as max_steps.
- **ecef_input** (*bool*) – If True, glats, glons, and alts are treated as x, y, z (ECEF).
- **return_geodetic** (*bool*) – If True, footpoint locations returned as lat, long, alt.

Returns Northern and Southern ECEF X,Y,Z locations

Return type array(len(glats), 3), array(len(glats), 3)

OMMBV._core.**full_field_line**(*init*, *date*, *height*, *step_size=100.0*, *max_steps=1000*, *steps=None*, ***kwargs*)

Perform field line tracing using IGRF and scipy.integrate.odeint.

Parameters

- **init** (*array-like of floats*) – Position to begin field line tracing from in ECEF (x,y,z) km
- **date** (*datetime or float*) – Date to perform tracing on (year + day/365 + hours/24. + etc.) Accounts for leap year if datetime provided.
- **height** (*float*) – Altitude to terminate trace, geodetic WGS84 (km)
- **max_steps** (*float*) – Maximum number of steps along each direction that should be taken
- **step_size** (*float*) – Distance in km for each large integration step. Multiple substeps are taken as determined by scipy.integrate.odeint
- **steps** (*array-like of ints or floats*) – Number of steps along field line when field line trace positions should be reported. By default, each step is reported, plus origin;

  steps=np.arange(max_steps+1).

  Two traces are made, one north, the other south, thus the output array could have double max_steps, or more via recursion.

Returns 2D array. [0,:] has the x,y,z location for southern footpoint [:,0] is the x positions over the integration. Positions are reported in ECEF (km).

Return type numpy array

OMMBV._core.**geocentric_to_ecef**(*latitude*, *longitude*, *altitude*)
Convert geocentric coordinates into ECEF

Parameters

- **latitude** (*float or array_like*) – Geocentric latitude (degrees)
- **longitude** (*float or array_like*) – Geocentric longitude (degrees)
- **altitude** (*float or array_like*) – Height (km) above presumed spherical Earth with radius 6371 km.

Returns numpy arrays of x, y, z locations in km

Return type x, y, z

OMMBV._core.**geodetic_to_ecef**(*latitude*, *longitude*, *altitude*)
Convert WGS84 geodetic coordinates into ECEF

Parameters

- **latitude** (*float or array_like*) – Geodetic latitude (degrees)
- **longitude** (*float or array_like*) – Geodetic longitude (degrees)
- **altitude** (*float or array_like*) – Geodetic Height (km) above WGS84 reference ellipsoid.

Returns numpy arrays of x, y, z locations in km

Return type x, y, z

OMMBV._core.**heritage_scalars_for_mapping_ion_drifts**(*glats*, *glons*, *alts*, *dates*, *step_size=None*, *max_steps=None*, *e_field_scaling_only=False*, *edge_length=25.0*, *edge_steps=1*, ***kwargs*)
Heritage technique for mapping ion drifts and electric fields.

Use scalars_for_mapping_ion_drifts instead.

Parameters

- **glats** (*list-like of floats (degrees)*) – Geodetic (WGS84) latitude
- **glons** (*list-like of floats (degrees)*) – Geodetic (WGS84) longitude
- **alts** (*list-like of floats (km)*) – Geodetic (WGS84) altitude, height above surface
- **dates** (*list-like of datetimes*) – Date and time for determination of scalars
- **e_field_scaling_only** (*boolean (False)*) – If True, method only calculates the electric field scalar, ignoring changes in magnitude of B. Note ion velocity related to E/B.

Returns array-like of scalars for translating ion drifts. Keys are, 'north_zonal_drifts_scalar', 'north_mer_drifts_scalar', and similarly for southern locations. 'equator_mer_drifts_scalar' and 'equator_zonal_drifts_scalar' cover the mappings to the equator.

Return type dict

Note: Directions refer to the ion motion direction e.g. the zonal scalar applies to zonal ion motions (meridional E field assuming ExB ion motion)

OMMBV._core.**magnetic_vector**(*x*, *y*, *z*, *dates*, *normalize=False*)
Uses IGRF to calculate geomagnetic field.

Parameters

- **x** (*array-like*) – Position in ECEF (km), X
- **y** (*array-like*) – Position in ECEF (km), Y
- **z** (*array-like*) – Position in ECEF (km), Z
- **normalize** (*bool (False)*) – If True, return unit vector

Returns Magnetic field along ECEF directions

Return type array, array, array

OMMBV._core.**normalize_vector**(*x*, *y*, *z*)
Normalizes vector to produce a unit vector.

Parameters

- **x** (*float or array-like*) – X component of vector
- **y** (*float or array-like*) – Y component of vector
- **z** (*float or array-like*) – Z component of vector

Returns Unit vector x,y,z components

Return type x, y, z

OMMBV._core.**project_ecef_vector_onto_basis**(*x*, *y*, *z*, *xx*, *xy*, *xz*, *yx*, *yy*, *yz*, *zx*, *zy*, *zz*)
Projects vector in ecef onto different basis, with components also expressed in ECEF

Parameters

- **x** (*float or array-like*) – ECEF-X component of vector
- **y** (*float or array-like*) – ECEF-Y component of vector
- **z** (*float or array-like*) – ECEF-Z component of vector
- **xx** (*float or array-like*) – ECEF-X component of the x unit vector of new basis
- **xy** (*float or array-like*) – ECEF-Y component of the x unit vector of new basis
- **xz** (*float or array-like*) – ECEF-Z component of the x unit vector of new basis

Returns Vector projected onto new basis

Return type x, y, z

OMMBV._core.**python_ecef_to_geodetic**(*x*, *y*, *z*, *method=None*)
Convert ECEF into Geodetic WGS84 coordinates

Parameters

- **x** (*float or array_like*) – ECEF-X in km
- **y** (*float or array_like*) – ECEF-Y in km
- **z** (*float or array_like*) – ECEF-Z in km

- **method** (*'iterative' or 'closed' ('closed' is deafult)*) – String selects method of conversion. Closed for mathematical solution (http://www.epsg.org/Portals/0/373-07-2.pdf , page 96 section 2.2.1) or iterative (http://www.oc.nps.edu/oc2902w/coord/coordcvt.pdf).

Returns numpy arrays of locations in degrees, degrees, and km

Return type latitude, longitude, altitude

OMMBV._core.**scalars_for_mapping_ion_drifts**(*glats*, *glons*, *alts*, *dates*, *max_steps=None*, *e_field_scaling_only=None*, *edge_length=None*, *edge_steps=None*, ***kwargs*)

Translates ion drifts and electric fields to equator and footpoints.

All inputs are assumed to be 1D arrays.

Parameters

- **glats** (*list-like of floats (degrees)*) – Geodetic (WGS84) latitude
- **glons** (*list-like of floats (degrees)*) – Geodetic (WGS84) longitude
- **alts** (*list-like of floats (km)*) – Geodetic (WGS84) altitude, height above surface
- **dates** (*list-like of datetimes*) – Date and time for determination of scalars
- **e_field_scaling_only** (*Deprecated*) –
- **max_steps** (*Deprecated*) –
- **edge_length** (*Deprecated*) –
- **edge_steps** (*Deprecated*) –

Returns array-like of scalars for translating ion drifts. Keys are, 'north_zonal_drifts_scalar', 'north_mer_drifts_scalar', and similarly for southern locations. 'equator_mer_drifts_scalar' and 'equator_zonal_drifts_scalar' cover the mappings to the equator.

Return type dict

OMMBV._core.**step_along_mag_unit_vector**(*x*, *y*, *z*, *date*, *direction=None*, *num_steps=1.0*, *step_size=25.0*, *scalar=1*)

Move along 'lines' formed by following the magnetic unit vector directions.

Moving along the field is effectively the same as a field line trace though extended movement along a field should use the specific field_line_trace method.

Parameters

- **x** (*ECEF-x (km)*) – Location to step from in ECEF (km).
- **y** (*ECEF-y (km)*) – Location to step from in ECEF (km).
- **z** (*ECEF-z (km)*) – Location to step from in ECEF (km).
- **date** (*list-like of datetimes*) – Date and time for magnetic field
- **direction** (*string*) – String identifier for which unit vector direction to move along. Supported inputs, 'meridional', 'zonal', 'aligned'
- **num_steps** (*int*) – Number of steps to take along unit vector direction
- **= float** (*step_size*) – Distance taken for each step (km)

- **scalar** (*int*) – Scalar modifier for step size distance. Input a -1 to move along negative unit vector direction.

Returns [x, y, z] of ECEF location after taking num_steps along direction, each step_size long.

Return type np.array

Notes

centered_diff=True is passed along to calculate_mag_drift_unit_vectors_ecef when direction='meridional', while centered_diff=False is used for the 'zonal' direction. This ensures that when moving along the zonal direction there is a minimal change in apex height.

OMMBV.satellite.**add_footpoint_and_equatorial_drifts**(*inst*, *equ_mer_scalar='equ_mer_drifts_scalar'*, *equ_zonal_scalar='equ_zon_drifts_scalar'*, *north_mer_scalar='north_footpoint_mer_drifts_scalar'*, *north_zon_scalar='north_footpoint_zon_drifts_scalar'*, *south_mer_scalar='south_footpoint_mer_drifts_scalar'*, *south_zon_scalar='south_footpoint_zon_drifts_scalar'*, *mer_drift='iv_mer'*, *zon_drift='iv_zon'*)

Translates geomagnetic ion velocities to those at footpoints and magnetic equator. .. note:

```
Presumes scalar values for mapping ion velocities are already in the inst, labeled
by north_footpoint_zon_drifts_scalar, north_footpoint_mer_drifts_scalar,
equ_mer_drifts_scalar, equ_zon_drifts_scalar.

Also presumes that ion motions in the geomagnetic system are present and labeled
as 'iv_mer' and 'iv_zon' for meridional and zonal ion motions.

This naming scheme is used by the other pysat oriented routines
in this package.
```

Parameters

- **inst** (*pysat.Instrument*) –
- **equ_mer_scalar** (*string*) – Label used to identify equatorial scalar for meridional ion drift
- **equ_zon_scalar** (*string*) – Label used to identify equatorial scalar for zonal ion drift
- **north_mer_scalar** (*string*) – Label used to identify northern footpoint scalar for meridional ion drift
- **north_zon_scalar** (*string*) – Label used to identify northern footpoint scalar for zonal ion drift
- **south_mer_scalar** (*string*) – Label used to identify northern footpoint scalar for meridional ion drift
- **south_zon_scalar** (*string*) – Label used to identify southern footpoint scalar for zonal ion drift
- **mer_drift** (*string*) – Label used to identify meridional ion drifts within inst
- **zon_drift** (*string*) – Label used to identify zonal ion drifts within inst

Returns Modifies pysat.Instrument object in place. Drifts mapped to the magnetic equator are labeled 'equ_mer_drift' and 'equ_zon_drift'. Mappings to the northern and southern footpoints are labeled 'south_footpoint_mer_drift' and 'south_footpoint_zon_drift'. Similarly for the northern hemisphere.

Return type None

OMMBV.satellite.**add_mag_drift_unit_vectors**(*inst*, *lat_label='latitude'*, *long_label='longitude'*, *alt_label='altitude'*, ***kwargs*)

Add unit vectors expressing the ion drift coordinate system organized by the geomagnetic field. Unit vectors are expressed in S/C coordinates.

Interally, routine calls add_mag_drift_unit_vectors_ecef. See function for input parameter description. Requires the orientation of the S/C basis vectors in ECEF using naming, 'sc_xhat_x' where *hat (*=x,y,z) is the S/C basis vector and* _ (*=x,y,z) is the ECEF direction.

Parameters

- **inst** (*pysat.Instrument object*) – Instrument object to be modified
- **max_steps** (*int*) – Maximum number of steps taken for field line integration
- ****kwargs** – Passed along to calculate_mag_drift_unit_vectors_ecef

Returns Modifies instrument object in place. Adds 'unit_zon_*' where * = x,y,z 'unit_fa_*' and 'unit_mer_*' for zonal, field aligned, and meridional directions. Note that vector components are expressed in the S/C basis.

Return type None

OMMBV.satellite.**add_mag_drift_unit_vectors_ecef**(*inst*, *lat_label='latitude'*, *long_label='longitude'*, *alt_label='altitude'*, ***kwargs*)

Adds unit vectors expressing the ion drift coordinate system organized by the geomagnetic field. Unit vectors are expressed in ECEF coordinates.

Parameters

- **inst** (*pysat.Instrument*) – Instrument object that will get unit vectors
- ****kwargs** – Passed along to calculate_mag_drift_unit_vectors_ecef

Returns

unit vectors are added to the passed Instrument object with a naming scheme:

'unit_zon_ecef_*' : unit zonal vector, component along ECEF-(X,Y,or Z)
'unit_fa_ecef_*' : unit field-aligned vector, component along ECEF-(X,Y,or Z)
'unit_mer_ecef_*' : unit meridional vector, component along ECEF-(X,Y,or Z)

Return type None

OMMBV.satellite.**add_mag_drifts**(*inst*)

Adds ion drifts in magnetic coordinates using ion drifts in S/C coordinates along with pre-calculated unit vectors for magnetic coordinates.

Note: Requires ion drifts under labels 'iv_*' where * = (x,y,z) along with unit vectors labels 'unit_zonal_*', 'unit_fa_*', and 'unit_mer_*', where the unit vectors are expressed in S/C coordinates. These vectors are calculated by add_mag_drift_unit_vectors.

Parameters **inst** (*pysat.Instrument*) – Instrument object will be modified to include new ion drift magnitudes

Returns Instrument object modified in place

Return type None

CHAPTER 2

Contributing

Bug reports, feature suggestions and other contributions are greatly appreciated! Pysat is a community-driven project and welcomes both feedback and contributions.

CHAPTER 3

Short version

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

CHAPTER 4

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

CHAPTER 5

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 :)

CHAPTER 6

Development

To set up *pysat* for local development:

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

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

3. Create a branch for local development:

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

   Now you can make your changes locally. Tests should be added to the appropriately named file in `OMMBV/tests`.

4. When you're done making changes, run all the checks to ensure that nothing is broken on your local system:

```
nosetests -vs pysatMagVect
```

5. Update/add documentation (in `docs`), if relevant

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.

6.1 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 an example for use
2. Add a note to `CHANGELOG.md` about the changes
3. Ensure that all checks passed (current checks include Scrutinizer, Travis-CI, and Coveralls)[1]

[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 Travis to run the tests for each change you add in the pull request. Because testing here will delay tests by other developers, please ensure that the code passes all tests on your local system first.

Python Module Index

O

Index