swarmtoolkit

Release 1.2.2

Mikael Toresen

August 31, 2016

Contents

1 Documentation

swarmtoolkit is a toolbox intended to provide simple and quick access to the Swarm L1 and L2 products for Python3.

Main features:

- reading and introspection of CDF (Common Data Format) files and the containing parameters to *numpy.ndarray*'s, whether the CDF's are stored locally, within zip-files or on an ftp-server.
- shift parameters with respect to each other in time, both using a user-defined time shift and by finding a best fit within a range using a minimizer. Parameters can also be aligned through interpolation to be evaluated at the same time values.
- Compute the magnetic field and its derivative in the NEC frame from SHC ascii files.
- Convenience functions to visualize plots using *matplotlib* and *mpl_toolkits.basemap*.

The demo notebook, found under the demo directory gives some instructional examples on how this package may be used, and is a good way to quickly get started. The documentation provides a more comprehensive look at the features.

Documentation shown below may also be found for each function:

- by using the `help` function:

```
>>> help(swarmtoolkit.getCDFparams)
```

- in *ipython* shell by typing `?` after the object:

```
>>> swarmtoolkit.getCDFparams?
```

For ease of use, the abbreviation `st` is suggested for *swarmtoolkit*:

```
>>> import swarmtoolkit as st
>>> st.plot([0,1,2],[2,1,0])
```

`swarmtoolkit.`**`debug_info`**(*activate=1*)

Set the verbosity level of the logger.

Parameters **`activate`** (*int*) – Possible values: - `activate>0` : logger set to DEBUG (default) - `activate<0` : logger set to CRITICAL (virtually no logging) - `activate=0` : logger set to INFO (nominal value)

`swarmtoolkit.`**`concatenate_values`**(**an*, *axis=None*)

Concatenate nD array's along *axis*.

Concatenate along last axis if none specified.

`swarmtoolkit.`**`dl_ftp`**(*url='swarm-diss.eo.esa.int'*, ***kwargs*)

Download files from ftp server.

Download files from a specified ftp url using filtering and/or interactive specification, if not already downloaded to given location.

Parameters **`url`** (*str*) – url of filename(s) or filename directory used to search for cdf file.

Keyword Arguments

- **user** (*str*) – ftp server username
- **pw** (*str*) – ftp server password
- **dst** (*str*) – location to download to
- **cdfsuffix** – see *getCDFparamlist*
- **use_filtering** – see *getCDFlist*
- **param0** – see *getCDFparams*
- **sat** – see *getCDFlist*
- **period** – see *getCDFlist*
- **use_passive_mode** (*bool, optional*) – Try to set this to `False` if having difficulties with connecting to ftp server (default `True`).
- **use_current** (*bool, optional*) – ignore directories named 'Previous' (default `True`).
- **use_color** (*bool, optional*) – allow coloring in listing of files/directories (default `True`).

Returns Paths to files specified for download. If only one file, a string will be returned.

Return type str or list of str

Raises `OSError` – if unable to connect to ftp server

`swarmtoolkit.`**`extract_parameter`**(*cdflist*, *parameter*, ***kwargs*)

Extract given parameter from cdf file.

Extract a parameter's values, unit and name from a list of cdf files and concatenate values along a given axis to a *numpy.ndarray*.

Parameters

- **cdflist** (*str or list of str*) – Path(s) of cdf file(s).
- **parameter** (*str*) – Name of parameter within cdf file. If name is not known, parameters within cdf files may be shown using *getCDFparamlist*.

Keyword Arguments

- **cat** (*bool*) – concatenate parameter values (default `True`)
- **axis** (*int*) – concatenate along specified axis. If `axis=None` it will concatenate along last axis. Ie. in a 5x4x2 array it will set axis to 3 (default `None`).

Returns list of parameter lists, which includes values, units and names of parameters. If only one parameter, it will not be contained within a list.

Return type list

Raises `CDFError` – if unable to decode any cdf file specified

`swarmtoolkit.`**`getCDFlist`**(*src=None*, *dst='.'*, *sort_by_t=False*, ***kwargs*)

Get a list of cdf's from a given location if zip or cdf.

Parameters

- **src** (*str, optional*) – Path or url of filename(s) or filename directory used to search for cdf file. If set to `None`, use current directory (default `None`).
- **dst** (*str, optional*) – Directory path of output files if input includes zip files or remote files. If directory does not exist it will be created. If set to `None` use same location as src (default `None`).
- **sort_by_t** (*bool, optional*) – Sort cdf filelist by start time of product (requires filenames to follow standard ESA Swarm naming convention) (default `False`).

Keyword Arguments

- **cdfsuffix** – see *getCDFparamlist*.
- **temp** (*bool*) – Specify whether to store cdf files extracted from zip temporarily or not (default `False`).
- **includezip** (*bool*) – Traverse zip files if cdf files already in directory. Otherwise will only open zip files if no cdf files present. (default `False`).
- **use_ftp** (*bool*) – specify use of ftp server to locate cdf files. Will be overruled (implicitly set to True) if *user* is specified (see *dl_ftp*)(default `False`).
- **filter_param** (*bool*) – Specify whether *param0* (if specified, see *getCDFparams*) will be used for filtering files based on filenames or not (default `False`).
- **sat** (*str or list of str*) – Filter files based on satelite in filename, eg. `sat=['A','C']` will only include files from Alpha or Charlie, while `sat='AC'` will only accept the joint Alpha-Charlie product files (default `None`).
- **period** (list of two *datetime.datetime* objects) – If a product is not (partially) within time window specified, product will be filtered away based on filename. Can alternatively be set using two of *start_t*, *end_t* and *duration* (default `None`).
- **start_t, end_t** (*str, scalar or datetime.datetime*) – Start/end time of filter. Scalars will be interpreted as fractional days since MJD2000 and strings can follow the formats

'yyyy-mm-dd', 'yyyymmdd', 'yyyymmddThhmmss' or 'yyyymmddhhmmss' (default None).

- **duration** (*scalar*) – Duration after *start_t* or before *end_t* in fractional days (default None).
- **param0** – see *getCDFparams* .

Returns **cdfl** – list of strings of absolute paths to cdf files found in *src*.

Return type list

Notes

The *src* argument will be overwritten by any *src* entry in *kwargs*.

If *start_t*. *end_t* and *duration* are all specified, *duration* will be ignored.

swarmtoolkit.**getCDFparamlist**(*cdflist, cdfsuffix=['DBL', 'CDF'], unzip=False, verbose=True, **kwargs*)

List parameters in the given files if unique.

Prints a list of parameters for each unique product type based on filename, and returns a dictionary with corresponding information.

Parameters

- **cdflist** (*str or list of str*) – Path(s) of cdf file(s). If the only input path is a directory, its content will be explored.
- **cdfsuffix** (*str or list of str, optional*) – case-insensitive list of strings of file extensions to accept as valid cdf files. If none found, zip files will be explored for the same file suffixes (default ['DBL','CDF']).
- **unzip** (*bool, optional*) – unzip any zip file in *cdflist* (default False).

Returns dictionary of product parameter lists. If only one product type is found, only a list of parameter is returned.

Return type dict or list

Raises CDFError – if unable to decode any cdf file specified.

swarmtoolkit.**getCDFparams**(*src*, **params*, ***kwargs*)

Extract parameters from cdf input as a list.

See *getCDFlist* for a list of keyword arguments not specified here.

Parameters

- **src** (*str*) – Path or url of filename(s) or filename directory used to search for cdf file.
- **params** (*str, optional*) – Names of parameters to be extracted from *src*. If names are not known, parameters within cdf files may be accessed using *getCDFparamlist*. Multiple parameters should be given as separate, comma-separated strings. If no parameters are provided, all parameters from the first cdf found in *src* will be used.

Keyword Arguments

- **param0** (*str*) – specify case-insensitive parameter name to use for filtering of files. If none specified first parameter in *params* is assumed. Will not be used unless filter_param=True is specified. Only supports values included in the dictionary swarmtoolkit.aux.PRODUCT_DIC.
- **asdict** (*bool*) – return parameters as dictionaries instead of lists(default False).

Returns *Parameter*'s as ordered in *params*. If only one parameter is specified, *Parameter* will not be contained within a list or dict. Note that ordering does not apply to dict.

Return type list or dict

Notes

Other keyword arguments are passed on to *getCDFlist*, *extract_parameter* and *dl_ftp* where applicable.

See also:

`getCDFlist()`, `extract_parameter()`, `dl_ftp()`

swarmtoolkit.**param_peek**(*in_arr_cdfl*, *parameter=None*, *n_show=5*, *axis=0*, *cataxis=None*)

Show some values contained in given cdf(s)/array.

Print values from a cdf file, *Parameter*, list of cdf files or *numpy.ndarray*:

- cdf input or *Parameter* only: parameter name and units
- 1D-array only : first and last *n_show* values
- all : shape, max, min, mean, median and population standard deviation values. number of zeros, NaN's, and largest jumps along given axis

Parameters

- **in_arr_cdfl** (str or list of str or *numpy.ndarray* or *Parameter*) – path to cdf file(s), or array.
- **parameter** (*str, optional*) – if input is cdf file(s), parameter name should be specified. If names are not known, parameters within cdf files may be shown using *getCDFparamlist*. (default `None`).
- **n_show** (*int, optional*) – number of values to show from start and end of 1D-array (default `5`).
- **axis** (*int*) – axis to view largest jumps over (default `0`).
- **cataxis** (*int*) – axis to concatenate over. If `cataxis=None` it will concatenate along last axis, e.g. in a 5x4x2 array it will set axis to 3 (default `None`).

swarmtoolkit.**read_mma**(*cdf_fn*, *source='internal'*, *frame='geo'*)

Read Swarm MMA final products

Extract the Gauss coefficients, time values and degree and order as a dictionary from magnetospheric MMA_SHA_2F products. Output can be directly inserted into `swtoolkit.get_Bnec`.

Parameters

- **cdf_fn** (*str*) – path to cdf file.
- **source** (*{ 'internal' | 'external' }, optional*) – specify internal or external model coefficients, ie. $g_{l,m}$ and $h_{l,m}$ or $q_{l,m}$ and $s_{l,m}$ (default `'internal'`).
- **frame** (*{ 'geo' | 'sm' }, optional*) – specify frame, either Geographic, or Solar Magnetic frame (default `'geo'`)

Returns

- dictionary with numpy.ndarrays with keys‘‘'coeff', 't'‘‘ and
- `'lm'`.

swarmtoolkit.**read_sp3**(*fname*, *doctype=2*, *SI_units=True*)

Read SP3 ascii files to array.

Read orbital format 'Standard Product # 3' (SP3) to numpy array of two SP3 document types as shown by example 1 and example 2 in https://igscb.jpl.nasa.gov/igscb/data/format/sp3_docu.txt . Output may be converted to SI units.

Parameters

- **fname** (*str*) – Path of SP3 file.
- **doctype** (*int*) – Two SP3 document formats:

  1. only position at given timestamps
  2. position and velocity at given timestamps with rate-of-change of clock correction

  doctype corresponds to these two options (default `2`).
- **SI_units** (*bool*) – Convert to SI-units from SP3 units (default `True`).

Returns `[x, y, z, t, header]` for `doctype=1`, `[x, y, z, vx, vy, vz, dt, t, header]` for `doctype=2`, where `header` is the first 22 lines of the SP3 document as a string.

Return type list of *numpy.ndarray*

Raises `EOFError` – If the specified SP3 file is empty.

swarmtoolkit.**read_EFI_prov_txt**(*fname*, **params*, *filter_nominal=False*)

Read provisional EFI products.

Read an ascii file containing provisional EFI products.

Parameters

- **fname** (*str*) – Path and filename of provisional EFI ascii file.
- **params** – Names of parameters to be extracted from *fname*. Possible values: - 'timestamp' - 'latitude' - 'longitude' - 'radius' - 'n' - 't_elec' - 'u_sc' - 'flag'

  If no parameters are specified, all will be returned in a dictionary. Multiple parameters should be given as separate, comma-separated strings.
- **filter_nominal** (*bool*) – Only extract data where `Flag=1` (default `False`).

Returns If parameters specified, returns list of *numpy.ndarray*'s; otherwise it will return dictionary of all parameter *numpy.ndarray*'s.

Return type List or dictionary

Notes

This function has not been subject to any optimization, and is slow. Multiple file support or data concatenation is as of yet not implemented.

swarmtoolkit.**unzip_file**(*input_file*, *output_loc*)

Unzip file to location if files in zip are not already present.

Parameters

- **input_file** (*str*) – path of zip file to extract content from.
- **output_loc** (*str*) – output path to extract content to.

Returns z

Return type `zipfile.ZipFile` object

class `swarmtoolkit.`**`Parameter`**(*values*, *unit=''*, *name=''*)

Container for a single parameter.

Has 3 attributes:

- values
- name
- unit

values can also be accessed by calling the parameter (eg. `myparam()`) or by accessing its indices (eg. `myparam[2]`)

`swarmtoolkit.`**`align_param`**(*p1*, *p2*, *t1*, *t2*, *k=3*, *align_to=False*)

Interpolate parameters such that time values overlap.

Given parameter arrays $p_1(t_1)$ and $p_2(t_2)$ with their corresponding time arrays one can downsample the most frequently sampled array, and align the arrays wrt. time (using interpolation) such that only one time array t is required for p_1 and p_2. Only overlapping temporal regions will be utilized. Upsampling may be forced using the *align_to* argument.

Parameters

- **`p1`** (*`1D numpy.ndarray`*) – First parameter to align.
- **`p2`** (*`1D numpy.ndarray`*) – Second parameter to align.
- **`t1`** (*`1D numpy.ndarray of datetime.datetime objects`*) – time values of *p1*. Should have same length as *p1*. If `align_to=True`, *t1* will set the output frequency.
- **`t2`** (*`1D numpy.ndarray of datetime.datetime objects`*) – time values of *p2*. Should have same length as *p2*.
- **`k`** (*`int, optional`*) – Degree of the smoothing spline. Must be `1 <= k <= 5` (default `3`).
- **`align_to`** (*`bool, optional`*) – Align *p2* to *p1* independent of respective frequency, instead of downsampling to lowest frequency of the two. This may be useful for aligning multiple parameters to a specific frequency, for upsampling, or for handling datasets where one of the parameters is not uniformly sampled (default `False`).

Returns (p1',p2',t) where *p1'* and *p2'* are sampled at *t* instead of at *t1* or *t2*. p1',p2' and t are *numpy.ndarray's*. As only the temporal overlap is utilized, the temporal span of the array will in general be less than or equal to the smallest temporal span of the two.

Return type tuple

Raises `ValueError`

- if respective p,t pairs are not of same length.
- if length of t array is less than spline order.
- if time arrays are not ordered ascending.
- if unable to interpolate arrays.

Notes

To be able to interpolate, the spline order must be less than the total number of time steps. This funtion assumes uniform sampling, and the sampling times are currently solely determined by the step length between the first two timesteps. If one of the parameters is uniformly sampled while the other is not, or one of the arrays contains non-finite numbers, `align_to=True` may be used with the finite, uniformly sampled parameter as *p1*.

See also:

plot_align()

swarmtoolkit.**shift_param**(*p1*, *p2*, *t1*, *t2*, *delta_t=0*, *dt_lim=(-20, 20)*, *v=1*, *spline_points=10000000.0*, *eval_width=None*, *k=3*, *auto=False*, *useminos=True*, *imincall=10000.0*, *bins=1000.0*, *return_delta=False*, *show=False*, *ext=2*)

Return values of *p1* and *p2* shifted by *delta_t*.

Shift a parameter $p_1(t_1)$ wrt. a second parameter $p_2(t_2)$ by a time step Δt. The shift can also be done automatically to find best fit by using a minimizer based on 'SEAL Minuit'(*iminuit*) with interpolated values.

Parameters

- **p1** (*1D numpy.ndarray*) – Parameter to be shifted by *delta_t*
- **p2** (*1D numpy.ndarray*) – parameter to shift *p1* with respect to.
- **t1** (*1D numpy.ndarray of datetime.datetime objects*) – time values of *p1*. Should have same length as *p1*.
- **t2** (*1D numpy.ndarray of datetime.datetime objects*) – time values of *p2*. Should have same length as *p2*.
- **delta_t** (*int,float, optional*) – time shift in seconds to shift *p1* by. If used together with the argument *auto=True*, this value will be used as a first guess to the best fit. It can then be set to *None* if *dt_lim* is set. A middle value will then be assumed (default `0`).
- **dt_lim** (*int/float list_like of length 2 or int/float.*) – Maximum and minimum value of *delta_t* in seconds allowed for minimizing algorithm. If *dt_lim* is a number, symmetry round *delta_t* will be assumed, eg `[delta_t-dt_lim,delta_t+dt_lim]`. *int* or *float* must be non-negative. If `dt_lim is None` it will be set to `dt_lim=(delta_t - ((1-eval_ratio)/2)*abs(delta_t), delta_t + ((1-eval_ratio)/2)*abs(delta_t))`, where `eval_ratio=eval_width/len(p1)` (default `(-20,20)`).
- **v** (*int, optional*) – Verbosity level of function. `0 <= v <= 2` (default `1`).
- **spline_points** (*int, optional*) – Number of points used to make a spline fit of p1 with. Number will be reduced if p1 has fewer points. Float values will be truncated (default `1e7`).
- **eval_width** (*int, optional*) – Number of points in time to compare *p1* and *p2* values, centered around the value of `t1+delta_t`. Number will be reduced by increasing span of dt_lim to accommodate for all possible values of *delta_t*. If set to `eval_width=None` a width corresponding to 60% of the length of p2 will be used (default `None`).
- **k** (*int, optional*) – Degree of the smoothing spline. Must be `1 <= k <= 5`.
- **auto** (*bool, optional*) – Use minimizer to find best fit for *delta_t* (default `False`).
- **useminos** (*bool*) – If `auto=True`,run minos (default `True`)

- **imincall** (*int*) – If `auto=True`, number of calls to migrad/minos. Float values will be truncated (default `1e4`)
- **bins** (*int*) – If *auto=True*, number of bins for profile of solution space (if no solution is found from initial *delta_t*, divide dt_lim into *bins*, and find best solution out of these). Also applicable for when visualizing profile using `show=True`. Float values will be truncated (default `1e3`).
- **return_delta** (*bool*) – return *delta_t* as output (default `False`).
- **show** (*False*) – show solution profile in a plot (see iminuit draw_profile)(default `False`).
- **ext** (*int*) – handling of values outside interpolation region:
  - extrapolation = 0
  - set to zero = 1
  - raise error = 2
  - set to constant(equal to boundary) = 3

Returns

- a tuple of numpy.ndarray's `(p1,p2,t1+delta_t,t2)` are returned.
- *Only values with temporal overlap are returned. Output will be of*
- equal length. If *return_delta=True*, a tuple
- `(p1,p2,t1+delta_t,t2,delta_t)` will be returned, with delta_t as
- *float.*

Raises

- `ValueError`
  - if length's are incompatible
  - if *eval_width*>length of p2
  - if neither *delta_t* nor *dt_lim* are provided.
  - if `delta_t=None` and *dt_lim* is a number.
  - if *dt_lim* is negative
- `IndexError` – if *dt_lim* has length less than 2.

Notes

This function assumes uniform sampling rate, and may not give desired results if this is not the case. As minimizing functions can be non-trivial, some tweaking of arguments may be necessary to get optimal results.

See also:

align_param(), *where_overlap()*

`swarmtoolkit.`**where_overlap**(*t1*, *t2*, *delta_t=0*)

Find overlap between two datetime arrays, where one array may be shifted by *delta_t*.

This is essentially a convenience function to access `spacepy.toolbox.tOverlap(t1+delta_t,t2,presort=True)`.

`swarmtoolkit.`**fourier_transform**(*param*, *dt_t*, *norm=None*)

Fourier transformation of 1d array with corresponding dt information.

Parameters

- **param** (*array_like*) – input parameter
- **dt_t** (*float, datetime.timedelta or numpy.ndarray of datetime.datetime*) – sample time or array of parameter sampling times.
- **norm** (*{None,'ortho'}*) – None : no scaling 'ortho': direct fourier transform scaled by `1/sqrt(n)`, with `n` being the length of *param* (default `None`).

Returns *numpy.ndarray* of fourier transform of param, and *numpy.ndarray* of corresponding frequencies.

Return type tuple

Raises `TypeError` – if *dt_t* is array and content is not *datetime.datetime* objects

Notes

Requires uniform temporal sampling.

swarmtoolkit.**cyclic2rising**(*a, lim=[-90, 90]*)

Return array with monotonic rising values, from array with cyclic values (requires first indices to be rising, and assumes approx. equidistant points).

Parameters

- **a** (*array_like*) – array of smooth cyclic values to be made monotonic rising.
- **lim** (*list*) – list of extremal(min,max) values within which *a* is cyclic (default `[-90,90]`).

Returns array of monotonic rising values

Return type *numpy.ndarray*

swarmtoolkit.**rising2cyclic**(*a, lim=[-90, 90]*)

Returns an array of cyclic values between two extremal values (requires first indices to be rising)

Parameters

- **a** (*array_like*) – array of monotonic rising values to be made cyclic
- **lim** (*list*) – list of extremal(min,max) values to make array cyclic within (default `[-90,90]`).

Returns array of cyclic values

Return type *numpy.ndarray*

swarmtoolkit.**interpolate2d_sphere**(*lat*, *lon*, *param*, *radians=True*, ***kwargs*)

Interpolate on rectangular mesh on sphere

Convenience function to call RectSphereBivariateSpline

Parameters

- **lat_rad** (*array_like*) – 1-D array of latitude coordinates in strictly ascending order. Coordinates must be given in radians, and lie within `(0, pi)`.
- **lon_rad** (*array_like*) – 1-D array of longitude coordinates in strictly ascending order. Coordinates must be given in radians and lie within the interval `(0, 2*pi)`.
- **param** (*array_like*) – 2-D array of parameter with shape `(lat_rad.size, lon_rad.size)`

- **radians** (*bool, optional*) –

Returns Spline function to be used for evaluation of interpolation

Return type *scipy.interpolate.RectSphereBivariateSpline*

Notes

Keyword arguments passed on to RectSphereBivariateSpline.

swarmtoolkit.**where_diff**(*values, atol=None, rtol=None, pdiff=[75, 25], axis=0, no_jump=False*)

Get indices of values which are significantly different from the preceding values.

Function to find discontinuities using absolute tolerance, relative tolerance and percentile differences over an array.

Parameters

- **values** (*array_like*) – input array to be evaluated
- **atol** (*float, optional*) – absolute tolerance such that where the difference between any value and its preceeding value is larger than *atol* will be flagged as a discontinuity. May be combined with *rtol* to only flag intersection of *atol* and *rtol* (default `None`).
- **rtol** (*float, optional*) – relative tolerance such that where the difference between any value and its preceeding value divided by its value is larger than *rtol* , it will be flagged as a discontinuity. May be combined with *atol* to only flag intersection of *atol* and *rtol* (default `None`).
- **pdiff** (*list of float of length 2, optional*) – Two values between 0 and 100. The percentile difference such that where the difference between any value and its preceeding value is larger than the difference between the values of the two percentiles of the data, it will be flagged as a discontinuity (default `[75,25]`).
- **axis** (*int*) – Axis in array over which to evaluate (default `0`).
- **no_jump** (*bool*) – Flag continuities instead of discontinuities (default `False`).

Returns Indices of flagged values

Return type ndarray or tuple of *numpy.ndarrays*

swarmtoolkit.**get_Bnec**(*shc_fn_dict*, *latitude*, *longitude*, *cols='all'*, *lmax=-1*, *lmin=-1*, *lmin_file=1*, *r=1*, *h=0*, *t_out=[]*, *k=-1*, *dB=False*, *gradient=''*, *source='internal'*, *ext=2*)

Compute magnetic field components in NEC-frame from SHC ascii file.

Get computation of the magnetic field components or its derivative for given latitude and longitude in the North-East-Center reference system given gaussian spherical harmonics coefficients file.

Parameters

- **shc_fn_dict** (*str*) – Path of input SHC ascii file or dictionary containing fields `'coeff'``(gauss coefficents)`,'t''(time[float]) and ''lm'(degree and order pairs) and optionally 'k'(spline order,default 3, will be overwritten if *k* is specified as keyword argument).
- **latitude** (*array_like*) – latitude values to evaluate magnetic field at
- **longitude** (*array_like*) – longitude values to evaluate magnetic field at
- **cols** (*list_like, optional*) – List of columns to read from file. This should correspond to the columns the different times values coefficients will be read from. In a standard SHC file the first two columns (0 and 1) correspond to the degree (`l`) and order (`m`) of the

harmonic and should not be included in *cols*. As such the default value `cols='all'` corresponds to `cols=range(2,2+N_times)`, where *N_times* is the number of time snapshots in the file.

- **lmin** (*lmax,*) – Maximum and minimum degree of harmonics l_{max} (l_{min}). If non-positive, suitable values will be set based on the number of coefficients (default `-1`).
- **lmin_file** (*int, optional*) – Lowest value of degree in SHC file (default `1`).
- **r** (*float, optional*) – Fractional radius at which to evaluate magnetic field. This is the radius divided by the reference radius 6371.2 km (see also *h*)(default `1`).
- **h** (*float, optional*) – Height(in km) above reference radius 6371.2 km at which to evaluate magnetic field. A non-zero value of *h* will overwrite any value of *r* (default `0`).
- **t_out** (*datetime.datetime, scalar or datetime/scalar list, optional*) – Times at which to evaluate magnetic field. Float values should correspond to fractional years. If left empty, times will be taken from the SHC file (default `[]`).
- **k** (*int, optional*) – Spline order for temporal interpolation. If not set, spline_order will be taken from SHC file. If *k* is greater than or equal to number of temporal snapshots, *k* will be reduced (default `-1`, implying set by SHC file).
- **dB** (*bool*) – Return interpolated magnetic field derivative dB/dt instead of magnetic field (default `False`).
- **gradient** (*{ '' | 'X' | 'Y' | 'Z'}*) – Compute the gradient of one of the magnetic field components. Note that solutions are numerically unstable at the poles. This will stack with *dB*, such that if `bool=True` and `gradient='Y'`, the time derivatives of the gradient of the east component will be computed. `''` implies no gradient (default `''`).
- **source** (*{ 'internal' | 'external' }*) – determine whether to interpret the SHC file as containing the internal or the external Gaussian coefficients (default `'internal'`).
- **ext** (*{ 0 | 1 | 2 | 3 }*) – If interpolation is performed, determine the behaviour when extrapolating:

  0 : return extrapolated value 1 : return 0 2 : raise error (default) 3 : return boundary value

Returns array with shape `(N_times,3,latitude,longtitude)`

Return type numpy.ndarray

See also:

get_l_maxmin(), *read_shc()*, *read_mma()*

swarmtoolkit.**get_Bparameter**(*B*, *outp='FDI'*)

Get Intensity, declination or inclination of magnetic field

Parameters

- **B** (*np.ndarray of floats*) – numpy.ndarray of magnetic vector components. Components are assumed to be B_N = B[0], B_E = B[1], B_C=B[2]. Expected shape of array is `(Ntimes,3,dim1,dim2)` or `(3,?,?)` where `?` signifies optional dimensions.
- **outp** (*str or list, optional*) – Output parameter. Must be `'F'``(intensity), ``'D'` (declination) or `'I'``(Inclination)` `(default ``'FDI'`).

Returns array with shape `(N_times,len(outp),latitude,longtitude)`

Return type numpy.ndarray

swarmtoolkit.**get_index**(*l*, *m*, *lmin=1*, *mmax=-1*)

Get index of a Gauss coefficient in an array, from degree and order

Order *m* needs to be less or equal to degree *l*.

Parameters

- **l** (*int*) – Degree of coefficient
- **m** (*int*) – Order of coefficient
- **lmin** (*int, optional*) – Lower bound of *l* in array (default 1).
- **mmax** (*int, optional*) – Upper bound of *m* in array. If *mmax* less than 0, then it is assumed to only be restricted by *l* (default -1).

Returns Index of coefficent. If invalid input parameters are provided, -1 will be returned.

Return type int

Notes

Index is given by: .. math:

l^2 - l_{ ext{min}}^2 + 2|m| , if m=0. l^2 - l_{ ext{min}}^2 + 2|m| - 1, if m>0.

If mmax is set an additional term $-(l - m_extmax + 1)(l - m_extmax + 2)$ is added.

swarmtoolkit.**get_l_maxmin**(*arr_len*, *lmax=0*, *lmin=0*, *suppress=False*)

Semi-brute force attempt to get a reasonable value of lmax (maximum degree) based on array length

Idea based on the fact that the array length will never exceed `lmax**2`, but `lmax` will never be larger than `array length/2`. The algorithm then favours solutions with lower `lmax` where the array length does not correspond to a unique `(lmax,lmin)` pair. `lmax` and/or `lmin` may be set. If no pair is found, an error is raised.‘‘suppress=True‘‘ suppresses logger output.

Assumes maximum order value is the same as maximum degree

swarmtoolkit.**read_shc**(*shc_fn*, *cols='all'*)

Read values of gaussian coefficients (g,h) from column(s) in file.

File should be ascii file obeying the SHC format.

Parameters

- **shc_fn** (*str*) – Path of input SHC ascii file
- **cols** (*list_like*) – List of columns to read from file. This should correspond to the columns the different times values coefficients will be read from. In a standard SHC file the first two columns (0 and 1) correspond to the degree (l) and order (m) of the harmonic and should not be included in *cols*. As such the default value `cols='all'` corresponds to `cols=range(2,2+N_times)`, where `N_times` is the number of time snapshots in the file.

Returns

Tuple with following values at given indices:

0. **numpy.ndarray of gaussian coefficients with such that** `myarray[0]` gives all coefficients at the first time point, given that there are multiple time snapshots. Otherwise `array[0]` will only contain the first coefficient.

1. **spline order *k* as an integer used to reconstruct model from** time snapshots.
2. number of columns as an integer.
3. **time of the temporal snapshots (in fractional years in the** standard SHC format) as 1D *numpy.ndarray*.

Return type Tuple

Notes

Missing data values marked as NaN are currently not handled.

`swarmtoolkit.`**`plot_align`**(*p1, p2, t1, t2, k=3, align_to=False, show=False, fmt_t=True, figsize=[8.0, 6.0], logx=False, logy=False, legends=[], lloc='best', lhide=False, colors=[], **plotkwargs*)

Convenience function which combines *align_param* with *plot*

Align p1 and p2 using interpolation such that values will be sampled on the same time steps. Output will be the same as for *plot*.

See *align_param* and *plot* for more information on arguments.

`swarmtoolkit.`**`plot`**(*x, y, *xy, show=False, fmt_t=True, figsize=[8.0, 6.0], logx=False, logy=False, legends=[], lloc='best', lhide=False, lbox=False, lfontsize=15, colors=[], **plotkwargs*)

Basic plot using *matplotlib*.

A convenience function to use matplotlib.pyplot.plot with some set parameters. Of particular note this function handles an x-axis with datetimes better than the default behaviour in matplotlib.

Parameters

- **`x`** (*`array_like`*) – Input x-values.
- **`y`** (*`array_like`*) – Input y-values.
- **`xy`** (*`optional`*) – Additional x- and y-values.
- **`show`** (*`bool, optional`*) – Show plot (default `False`).
- **`fmt_t`** (*`bool, optional`*) – Format datetime x-ticks (see matplotlib.figure.autofmt_xdate) (default `True`).
- **`figsize`** (*`tuple of length 2, optional`*) – Size of figure as tuple of width and height in inches (default `matplotlib.pyplot.rcParams["figure.figsize"]`).
- **`logx`** (*`bool, optional`*) – Set x-axis scale to log (default `False`).
- **`logy`** (*`bool, optional`*) – Set y-axis scale to log (default `False`).
- **`legends`** (*`list_like, optional`*) – Add legend(s)(default `[]`).
- **`lloc`** (*`str or int, optional`*) – Location of legend. Can be one of: 'best' : 0, (default) 'upper right' : 1, 'upper left' : 2, 'lower left' : 3, 'lower right' : 4, 'right' : 5, 'center left' : 6, 'center right' : 7, 'lower center' : 8, 'upper center' : 9, 'center' : 10
- **`lhide`** (*`bool, optional`*) – Do not show legends. Useful to combine legends with twinx legends (default `False`).
- **`lbox`** (*`bool`*) – box legends in semi-transparent box (default `False`)

- **lfontsize** (*scalar*) – fontsize of legend (default 15)
- **colors** (*list_like, optional*) – Color cycle to use in plot (eg. ['r','g','b'] will show plots in red, green and blue (see matplotlib.colors for more examples). Default will use colormap set in the rcParams. (default []).
- **plotkwargs** (*optional*) – Additional keyword arguments to pass on to matplotlib.pyplot.plot, these will be overwritten if conflicting with other values.

Returns matplotlib.figure.Figure and matplotlib.axes.Axes instances for plot

Return type tuple

See also:

plot_twinx(), *plot_align()*

swarmtoolkit.**plot_geo**(*lat, lon, param, ptype='scatter', figsize=[8.0, 6.0], cmap='jet', cbar=True, dark_map=False, show=False, contourlevels=15, log_contour=False, show_lat=True, show_lon=False, show_grid=True, **kwargs*)

Plot parameter on the globe using *mpl_toolkits.basemap.Basemap*.

Parameters

- **lat** (*array_like*) – Latitude of *param*.
- **lon** (*array_like*) – Longitude of *param*.
- **param** (*array_like*) – Value of *param* at each (lat,lon)-coordinate.
- **ptype** ({'scatter'|'colormesh'|'contour'}, optional) – Set plot type (default 'scatter').
- **figsize** (*tuple, optional*) – Size of figure as tuple of width and height in inches (default matplotlib.pyplot.rcParams["figure.figsize"]).
- **cmap** (*matplotlib.colors.ColorMap or str*) – (Name of) colormap to be used in plot (default matplotlib.pyplot.rcParams["image.cmap"]).
- **cbar** (*bool, optional*) – use colorbar (default True).
- **dark_map** (*bool, optional*) – draw map with darker tones of gray (default False).
- **show** (*bool, optional*) – Show plot (default False).
- **contourlevels** (*int, optional*) – number of coutour levels to use in coutourplot (default 15).
- **log_contour** (*bool, optional*) – plot contour levels using logarithmic distances between lines.
- **show_lat** (*bool, optional*) – show labels for latitude(requires *show_grid*) (default True).
- **show_lon** (*bool, optional*) – show labels for longitude(requires *show_grid*) (default False).
- **show_grid** (*bool, optional*) – show gridlines(graticules) (default True).

Returns matplotlib.figure.Figure and mpl_toolkits.basemap.Basemap object for plot.

Return type tuple

Notes

See http://matplotlib.org/basemap/api/basemap_api.html for full set of possible keyword arguments. In particular the projection can be set with `projection`, which is by default set to 'moll' (Mollweide projection) in this funciton. In addition, depending on the value of *ptype*, the following values are used as default:

scatter:

See mpl_toolkits.basemap.scatter default values:

•linewidths : 0.0

•vmin : min(param)

•vmax : max(param)

colormesh:

See mpl_toolkits.basemap.pcolormesh Note that as *colormesh requires 2D arrays; providing ''latlon=True'* allows latitude and longitude to be converted to a 2d mesh properly from two 1D arrays. default values:

•shading : flat

•alpha : 0.8

contour:

See mpl_toolkits.basemap.pcolormesh.contour Note that as *colormesh requires 2D arrays; providing ''latlon=True'* allows latitude and longitude to be converted to a 2d mesh properly from two 1D arrays. default values :

•animated : True

`swarmtoolkit.`**`plot_scatter`**(*x, y, param, show=False, fmt_t=True, figsize=[8.0, 6.0], vmax=None, vmin=None, cmap='jet', cbar=True, **scatterkwargs*)

Scatterplot with colorbar using matplotlib.pyplot.scatter.

Parameters

- **`x`** (*`array_like`*) – x-coordinates of *param*.
- **`y`** (*`array_like`*) – y-coordinates of *param*.
- **`param`** (*`array_like`*) – value(determining colour) of *param* at each (x,y)-coordinate.
- **`show`** (*`bool, optional`*) – Show plot (default `False`).
- **`fmt_t`** (*`bool, optional`*) – Format datetime x-ticks (see matplotlib.figure.autofmt_xdate) (default `True`).
- **`figsize`** (*`tuple of length 2, optional`*) – Size of figure as tuple of width and height in inches (default `matplotlib.pyplot.rcParams["figure.figsize"]`).
- **`vmax`** (*`scalar, optional`*) – vmax sets the upper bound of the colour data. If either *vmin* or *vmax* are None, the min and max of the color array is used (default `None`).
- **`vmin`** (*`scalar, optional`*) – vmin sets the lower bound of the colour data. If either *vmin* or *vmax* are None, the min and max of the color array is used (default `None`).
- **`cmap`** (*`matplotlib.colors.ColorMap`*) – colormap to be used in plot (default `matplotlib.pyplot.rcParams["image.cmap"]`).
- **`cbar`** (*`bool, optional`*) – use colorbar (default `True`).

Keyword Arguments

- **s** (*scalar or array_like*) – (size of points)**2 (default `3`).
- **linewidths** (*scalar*) – (default `0.0`).
- **alpha** (*scalar*) – blending value between 0(transparent) and 1(opaque).

Returns matplotlib.figure.Figure and matplotlib.axes.Axes instances for plot as a tuple

Return type tuple

`swarmtoolkit.`**`plot_twinx`**(*x*, *y*, **xy*, *show=False*, *logy=False*, *legends=[]*, *lloc='best'*, *lall=True*, *lbox=False*, *lfontsize=15*, *ax=None*, *colors=[]*, ***plotkwargs*)

Overplot with a twin x-axis.

Share same x-axis as another plot, but with separate y-axis values. Should be used in conjunction with another plot function (eg. *plot*).

Parameters

- **x** (*array_like*) – Input x-values.
- **y** (*array_like*) – Input y-values.
- **xy** (*optional*) – Additional x- and y-values.
- **show** (*bool, optional*) – Show plot (default False).
- **logy** (*bool, optional*) – Set y-axis scale to log (default `False`).
- **legends** (*list_like, optional*) – Add legend(s) (default `[]`).
- **lloc** (*str or int, optional*) – Location of legend. Can be one of:: 'best' : 0 (default) 'upper right' : 1 'upper left' : 2 'lower left' : 3 'lower right' : 4 'right' : 5 'center left' : 6 'center right' : 7 'lower center' : 8 'upper center' : 9 'center' : 10
- **lall** (*bool, optional*) – Combine legends from *ax* with *legends* (default `True`).
- **lbox** (*bool*) – box legends in semi-transparent box (default `False`)
- **lfontsize** (*scalar*) – fontsize of legend (default `15`)
- **ax** (*matplotlib.axes.Axes*) – Axes instance of plot to share x-axis with. If `'ax=None'`, get current Axes instance (default None).
- **colors** (*list_like, optional*) – Color cycle to use in plot (eg. `['r','g','b']` will show plots in red, green and blue (see matplotlib.colors for more examples). Default will use colormap set in the rcParams. (default `[]`).
- **plotkwargs** (*optional*) – Additional keyword arguments to pass on to matplotlib.pyplot.plot, these will be overwritten if conflicting with other values.

Returns

Return type matplotlib.axes.Axes

See also:

plot(), *plot_align()*

`swarmtoolkit.`**`save_raw`**(*fig_*, *fn='raw_img.png'*, *shape_ratio=None*, *dpi=1*)

Save content of figure to file without axes or padding

Parameters

- **fig** (*list or tuple*) – List with figure in first index. This corresponds to the output of the plotting functions.

- **fn** (*str, optional*) – Name of output file (default 'raw_img.png').
- **shape_ratio** (*list or tuple*) – width and height of image in relative units (matplotlib's "inches") , should be manually set to prevent padding (default None).
- **dpi** (*scalar,optional*) – the resolution of the image in dots per inch.

Examples

Printing straight from plot function:

```
>>> import swarmtoolkit as st
>>> st.save_raw(st.plot([0,1,2],[0,2,1]))
```

How to retrieve the image as a numpy.ndarray:

```
>>> import matplotlib.pyplot as plt
>>> img_as_array = plt.imread('raw_img.png')
```

Save plotted image normally in matplotlib:

```
>>> import matplotlib.pyplot as plt
>>> plt.savefig('myfilename.png')
```

Alternatively *figure* or *axes* object can be used (eg. fig.savefig)

2 Keyword arguments hierachy:

Several key functions in *swarmtoolkit* pass on keyword arguments to functions they call under the hood. Below is an overview to help keep track of this:

getCDFparams -> *getCDFparamlist* (if no parameter is provided)

getCDFparams -> *getCDFlist* -> *dl_ftp* (if use_ftp=True or user is provided)

getCDFparams -> *extract_parameter* -> *concatenate_values*

3 Installation requirements

swarmtoolkit requires:

```
- Python (>=3.2)
- Numpy (>=1.5)
- Scipy (>=0.14)
- matplotlib(>=1.5)
- basemap (>=1.0, from mpl_toolkits)
- spacepy (>=0.1.5)
- ftputil (>=3.0)
```

- iminuit (>=1.0) (for the function *shift_param*)
- numexpr (>=2.4) (for the function *shift_param*)
- astropy (>=1.0) (for the function *map_of_means*)

This *should* be all you need to do to get started with swarmtoolkit:

Install python, C-compile w/ python headers, which for ubuntu the following should suffice:

```
apt-get install build-essential python3-dev
```

then download miniconda and run the bash/exe installer.

install required packages:

```
conda install numpy scipy matplotlib spacepy basemap numexpr pip ipython \
  ipython-notebook

pip install ftputil iminuit
```

Then everything should be ready to be run. If you want to use `swarmtoolkit`, either type in:

```
python setup.py install
```

in the root folder of swarmtoolkit, or, manually or add `swarmtoolkit` to your pythonpath in a `.bash_profile` or `.bashrc` file eg:

```
export PYTHONPATH=$PYTHONPATH:/path/to/swarmtoolkit/directory
```

to use jupyter/ipython notebook just type `ipython notebook` in a terminal (optionally add a file path of a notebook file) and it should start up in a browser.

4 Testing

To test `swarmtoolkit` using *nose*, simply run `nosetests` (or alternatively `nose2`) in the `tests`-directory.

5 Remarks

Swarm Level0 data products *can* be read, but tools for this have been stored elsewhere as it is less flexible, and builds mainly upon the work of Stefano Mattia, and uses a different framework. To get this, or if there are other questions related to *swarmtoolkit*, you can contact Mikael Toresen mikael.toresen@gmail.com.

A demo of some of the functionality may be found under the *demo* folder.

6 Complementary packages

6.1 Converting between geographic and magnetic coordinates aacgmv2

aacgmv2 is a *pip*-installable wrapper to the AACGM-v2 C library (Altitude adjusted corrected geomagnetic) with computation required for conversion to the *magnetic local time*. From their github repository (taken 15.07.2016):

Convert between AACGM and geographic coordinates:

```
>>> from aacgmv2 import convert
>>> from datetime import date
>>> # geo to AACGM, single numbers
>>> mlat, mlon = convert(60, 15, 300, date(2013, 11, 3))
>>> mlat
array(57.47207691280528)
```

```
>>> mlon
array(93.62138045643167)
>>> # AACGM to geo, mix arrays/numbers
>>> glat, glon = convert([90, -90], 0, 0, date(2013, 11, 3), a2g=True)
>>> glat
array([ 82.96656071, -74.33854592])
>>> glon
array([ -84.66516034,  125.84014944])
```

Convert between AACGM and MLT:

::

```
    >>> from aacgmv2 import convert_mlt
    >>> from datetime import datetime
    >>> # MLT to AACGM
    >>> mlon = convert_mlt([0, 12], datetime(2013, 11, 3, 18, 0), m2a=True)
    >>> mlon
    array([ 159.10097421,  339.10097421])
```

Where the 1st and 2nd arguments of *convert* are the latitude and longitudes, the 3rd argument is the altitude, and the date will default to the current time. Switching between conversion and the inverse conversion is done using the *a2g* parameter which is *False* by default. latitude, longitude and altitude can be arrays or scalars.

convert_mlt takes the magnetic logitude(magnetic local time) and a datetime object and returns the magnetic local time(magnetic longitude) given that *m2a* is set to *False*‘(‘*True*). Magnetic longitude and magnetic local time can be arrays or scalars.

Index