windmultipliers Documentation

Release 2.0

Geoscience Australia

December 09, 2015

CONTENTS

CHAPTER

ONE

OVERVIEW

This package is used to produce wind terrain, shielding and topographic multipliers for national coverage using the input of National Dynamic Land Cover Dataset of Australia Version 2.0 (http://www.ga.gov.au/metadata-gateway/metadata/record/gcat_83868) and 1 second Shuttle Radar Topography Mission (SRTM) Smoothed Digital Elevation Models (DEM-S) Version 1.0 (http://www.ga.gov.au/metadata-gateway/metadata/record/gcat_72759). The output is based on tiles with dimension about 1 by 1 degree in netCDF format. It includes terrain, shielding and topographic multiplier respectively. Each multiplier further contains 8 directions.

CHAPTER

TWO

DEPENDENCIES

Python 2.7, NumPy, SciPy, NetCDF4 and GDAL are needed.

CHAPTER

THREE

PACKAGE STRUCTURES

The script for deriving terrain, shielding and topographic multipliers is **all_multipliers.py** that can be run in parallel using MPI. It links four modules:

1. terrain;
2. shielding;
3. topographic; and
4. utilities

Terrain module includes:

- **terrain_mult.py:** produce the terrain multiplier for a given tile

Shielding module includes:

- **shield_mult.py:** produce the shielding multiplier for a given tile

Topographic module includes:

- **topo_mult.py: produce the topographic multiplier for a given tile**
  - make_path.py: generate indices of a data line depending on the direction
  - **multiplier_calc.py: calculate the multipliers for a data line extracted from the dataset:**
    - mh.py: calculate Mh
    - findpeaks.py: get the indices of the ridges in a data line Directory

Utilities module includes supporting tools such as:

- blrb.py;
- files.py;
- get_pixel_size_grid.py;
- meta.py;
- nctools.py;
- value_lookup.py;
- vincenty.py.

Note: Before running **all_multipliers.py** to produce terrain, shielding and topographic multipliers, the configuration file named **multiplier_conf.cfg** needs to be configured. There are some variables to be pre-defined:

- **root:** the working directory of the task.
- **upwind_length:** the upwind buffer distance

Then copy the input files (dem and terrain classes) into the input folder (created beforehand manually) under **root**, and start to run **all_multipliers.py**. The results are respectively located under output folder (created automatically during the process) under **root**.

CHAPTER

FOUR

BACKGROUND

Wind multipliers are factors that transform regional wind speeds to local wind speeds considering local effects of land cover and topographic influences. It includes terrain, shielding, topographic and direction multipliers. Except the direction multiplier whose value can be defined specifically by the Australian wind loading standard AS/NZS 1170.2, terrain, shielding and topographic multipliers are calculated using this software package based on the principles and formulae defined in the AS/NZS 1170.2. The wind multipliers are primarily used for assessment of wind hazard at individual building locations. Further details on wind multipliers can be found in Geoscience Australia record: *Local Wind Assessment in Australia: Computation Methodology for Wind Multipliers*, which is available via http://www.ga.gov.au/metadata-gateway/metadata/record/75299.

CHAPTER

FIVE

ISSUES

Issues for this project are currently being tracked through Github.

CHAPTER

SIX

CODE DOCUMENTATION

6.1 All_multipliers

6.1.1 all_multipliers.py

`all_multipliers` - Calculate terrain, shielding & topographic multipliers

This module can be run in parallel using MPI if the pypar library is found and all_multipliers is run using the mpirun command. For example, to run with 8 processors:

```
mpirun -n 8 python all_multipliers.py
```

moduleauthor Tina Yang <tina.yang@ga.gov.au>

class `all_multipliers.`**`Multipliers`**(*landcover*, *dem*)
Computing multipliers parallelly based on tiles.

`clip_dataset`(*extent*, *dst_filename*)
Clip the DEM using an extent and save the clipped to a new file.

Parameters

- **`extent`** – *tuple* the input tile extent with buffer
- **`dst_filename`** (*str*) – destination filename.

`cut_dem`(*tile_info*)
Cut from the input DEM for a tile

Parameters **`tile_info`** – *tuple* the input tile info

Returns *file* the output dem for a tile

`multipliers_calculate`(*temp_tile_dem*, *tile_info*)
Calculate the multiplier values for a specific tile

Parameters

- **`temp_tile_dem`** – *file* the input DEM tile
- **`tile_info`** – *tuple* the input tile info

`open_dem`()
Open the DEM file

`parallelise_on_tiles`(*tiles*, *progress_callback=None*)
Iterate over tiles to calculate the wind multipliers

Parameters **`tiles`** – *generator* that yields tuples of tile dimensions.

class all_multipliers.**TileGrid**(*upwind_length*, *raster_ds*)
Tiling to minimise MemoryErrors and enable parallelisation.

get_gridlimit(*k*)
Return the limits without buffer for tile *k*. x-indices correspond to the east-west coordinate, y-indices correspond to the north-south coordinate.

Parameters **k** – *int* tile number

Returns minimum, maximum x-index and y-index for tile *k*

get_gridlimit_buffer(*k*)
Return the limits with buffer for tile *k*. x-indices correspond to the east-west coordinate, y-indices correspond to the north-south coordinate.

Parameters **k** – *int* tile number

Returns minimum, maximum x-index and y-index for tile *k*

get_startcord(*k*)
Return starting longitude and latitude value of the tile without buffer

Parameters **k** – *int* tile number

Returns *float* starting x and y coordinate of a tile without buffer

get_tile_extent(*k*)
Return the exntent without buffer for tile *k*. x corresponds to the east-west coordinate, y corresponds to the north-south coordinate.

Parameters **k** – *int* tile number

Returns minimum, maximum x and y coordinate for tile *k*

get_tile_extent_buffer(*k*)
Return the exntent for tile *k*. x corresponds to the east-west coordinate, y corresponds to the north-south coordinate.

Parameters **k** – *int* tile number

Returns minimum, maximum x and y coordinate for tile *k*

get_tilename(*k*)
Return the name of a tile

Parameters **k** – *int* tile number

Returns *string* name of a tile composing of starting coordinates

tile_grid()
Defines the indices required to subset a 2D array into smaller rectangular 2D arrays (of dimension x_step * y_step plus buffer size for each side if available).

all_multipliers.**attempt_parallel**()
Attempt to load Pypar globally as *pp*. If pypar cannot be loaded then a dummy *pp* is created.

all_multipliers.**balance**(*nn*)
Compute p'th interval when nn is distributed over s bins

all_multipliers.**balanced**(*iterable*)
Balance an iterator across processors.

This partitions the work evenly across processors. However, it requires the iterator to have been generated on all processors before hand. This is only some magical slicing of the iterator, i.e., a poor man version of scattering.

all_multipliers.**disable_on_workers**(*f*)
Disable function calculation on workers. Function will only be evaluated on the master.

all_multipliers.**do_output_directory_creation**(**args*, ***kwargs*)
Create all the necessary output folders.

Parameters **root** – *string* Name of root directory

Raises OSError If the directory tree cannot be created.

all_multipliers.**get_tileinfo**(*tilegrid*, *tilenums*)
Generate a list of tuples of the name and extent of a tile

Parameters

- **tilegrid** – *TileGrid* instance
- **tilenums** – list of tile numbers (must be sequential)

Returns tileinfo: list of tuples of tile names and extents

all_multipliers.**get_tiles**(*tilegrid*)
Helper to obtain a generator that yields tile numbers

Parameters **tilegrid** – *TileGrid* instance

all_multipliers.**reproject_dataset**(**args*, ***kwargs*)
Clip and reproject a source dataset to match the projection of another dataset and save the projected dataset to a new file.

Parameters

- **src_filename** – Filename of the source raster dataset, or an open `gdal.Dataset`
- **match_filename** – Filename of the dataset to match to, or an open `gdal.Dataset`
- **dst_filename** (*str*) – Destination filename.
- **resampling_method** – Resampling method. Default is bilinear interpolation.
- **match_projection** – Projection of the output

all_multipliers.**run**(**args*, ***kwargs*)
Run the wind multiplier calculations.

This will attempt to run the calculation in parallel by tiling the domain, but also provides a sane fallback mechanism to execute in serial.

all_multipliers.**timer**(*f*)
Basic timing functions for entire process

6.2 terrain

6.2.1 __init__.py

6.2.2 terrain.py

`terrain` – Calculate terrain multiplier

This module is called by the module all_multipliers to calculate the terrain multiplier for an input tile for 8 directions and output as NetCDF format.

References Yang, T., Nadimpalli, K. & Cechet, R.P. 2014. Local wind assessment in Australia: computation methodology for wind multipliers. Record 2014/33. Geoscience Australia, Canberra.

moduleauthor Tina Yang <tina.yang@ga.gov.au>

terrain_mult.**convo**(*one_dir*, *data*, *avg_width*, *lag_width*)
Convolute the initial terrain multplier to final values for one of the eight directions

Parameters

- **one_dir** – *str* the direction
- **data** – numpy.ndarray the initial terrain multiplier values
- **avg_width** – :*int* the number of cells within the upwind buffer
- **lag_width** – :*int* the number of cells within the lag distance

Returns numpy.ndarray the final terrain multiplier value

terrain_mult.**terrain**(*temp_tile*, *tile_extents_nobuffer*)
Performs core calculations to derive the terrain multiplier

Parameters

- **temp_tile** – *file* the image file of the input tile of the land cover
- **tile_extents_nobuffer** – *tuple* the input tile extent without buffer

terrain_mult.**terrain_class2mz_orig**(*data*)
Transfer the landsat classified image into original terrain multiplier

Parameters **data** – numpy.ndarray the input terrain class values

Returns numpy.ndarray the initial terrain multiplier value

6.3 shielding

6.3.1 __init__.py

6.3.2 shielding.py

shielding – Calculate shielding multiplier

This module is called by the module all_multipliers to calculate the shielding multiplier for an input tile for 8 directions and output as NetCDF format.

References Yang, T., Nadimpalli, K. & Cechet, R.P. 2014. Local wind assessment in Australia: computation methodology for wind multipliers. Record 2014/33. Geoscience Australia, Canberra.

moduleauthor Tina Yang <tina.yang@ga.gov.au>

shield_mult.**blur_image**(*im*, *kernel*, *mode='constant'*)
Blurs the image by convolving with a kernel (e.g. mean or gaussian) of typical size.

Parameters

- **im** – numpy.ndarray input data of initial shielding values
- **kernel** – numpy.ndarray the kernel used for convolution

Returns numpy.ndarray the output data afer convolution

shield_mult.**combine**(*ms_orig_array*, *slope_array*, *aspect_array*, *one_dir*)
Used for each direction to derive the shielding multipliers by considering slope and aspect after covolution in the previous step. It will also remove the conservatism.

Parameters

- **ms_orig_array** – numpy.ndarray convoluted shielding values
- **slope_array** – numpy.ndarray the input slope values
- **aspect_array_reclassify** – numpy.ndarray input aspect values
- **one_dir** – *str* the direction of wind

Returns numpy.ndarray the output shielding mutipler values

shield_mult.**convo_combine**(*ms_orig*, *slope_array*, *aspect_array*, *tile_extents_nobuffer*)
Apply convolution to the orginal shielding factor for each direction and call the combine module to consider the slope and aspect and remove conservitism to get final shielding multiplier values

Parameters

- **ms_orig** – *file* the original shidelding factor map
- **slope_array** – numpy.ndarray the input slope values
- **aspect_array** – numpy.ndarray the input aspect values
- **tile_extents_nobuffer** – *tuple* the input tile extent without buffer

shield_mult.**get_slope_aspect**(*input_dem*)
Calculate the slope and aspect from the input DEM

Parameters **input_dem** – *file* the input DEM

Returns numpy.ndarray the output slope values

Returns numpy.ndarray the output aspect values

shield_mult.**init_kern**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is north direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**init_kern_diag**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is south west direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**kern_e**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is east direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**kern_n**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is north direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**kern_ne**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is north-east direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**kern_nw**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is north-west direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**kern_s**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is south direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**kern_se**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is south-east direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**kern_sw**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is south-west direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**kern_w**(*size*)
Returns a mean kernel for convolutions, with dimensions (2*size+1, 2*size+1), it is west direction

Parameters **size** – *int* the buffer size of the convolution

Returns numpy.ndarray the output kernel used for convolution

shield_mult.**reclassify_aspect**(*data*)
Reclassify the aspect valus from 0 ~ 360 to 1 ~ 9

Parameters **data** – numpy.ndarray the input aspect values 0 ~ 360

Returns numpy.ndarray the output aspect values 1 ~ 9

shield_mult.**shield**(*terrain*, *input_dem*, *tile_extents_nobuffer*)
Performs core calculations to derive the shielding multiplier

Parameters

- **terrain** – *file* the input tile of the terrain class map (landcover).
- **input_dem** – *file* the input tile of the DEM
- **tile_extents_nobuffer** – *tuple* the input tile extent without buffer

shield_mult.**terrain_class2ms_orig**(*terrain*)
Reclassify the terrain classes into initial shielding factors

Parameters **terrain** – *file* the input terrain class map

Returns *file* the output initial shielding value

6.4 topographic

6.4.1 __init__.py

6.4.2 findpeaks.py

findpeaks – Generate the indices of the ridges in a data line

This module is called by the module multiplier_calc

findpeaks.**findpeaks**(*y*)
Generate the indices of the peaks in a data line

Parameters **y** – numpy.ndarray the elevation of a line

Returns numpy.ndarray the index values of the ridges in the line

findpeaks.**findvalleys**(*y*)
Generate the indices of the valleys in a data line

Parameters **y** – numpy.ndarray the elevation of a line

Returns numpy.ndarray the index values of the valleys in the line

6.4.3 make_path.py

makepath – Returns a vector of array indices for a path

This module is called by the module topomult

make_path.**make_path**(*nr*, *nc*, *n*, *dire*)
Returns a vector of array indices for a path starting at index n in a matrix of size nr by nc and proceeding in direction dir, where dir is one of the 8 cardinal directions (n,s,e,w,ne,nw,se,sw). Note that the array indices are all 1-d indices.

Parameters

- **nr** – *int* number of rows of the input DEM
- **nc** – *int* number of columns of the input DEM
- **n** – *int* starting index
- **dire** – *string* firection of the path

Returns numpy.ndarray the indices of a path

6.4.4 mh.py

mh – Calculate the topographic multipliers

This module is called by the module multiplier_calc

mh.**escarpment_factor**(*profile*, *ridge*, *valley*, *data_spacing*)
Calculate escarpment factor

Parameters

- **profile** – numpy.ndarray the elevation of a line

- **ridge** – numpy.ndarray the indices of the ridges of a line
- **valley** – numpy.ndarray the indices of the valleys of a line
- **data_spacing** – *float* distance between neighbour points of a line

Returns *float* the escarpment factor

mh.**mh_calc**(*profile*, *ridge*, *valley*, *data_spacing*)
Calculate topographic multiplier

Parameters

- **profile** – numpy.ndarray the elevation of a line
- **ridge** – numpy.ndarray the indices of the ridges of a line
- **valley** – numpy.ndarray the indices of the valleys of a line
- **data_spacing** – *float* distance between neighbour points of a line

Returns numpy.ndarray the topogrpahic multiplier of the line

6.4.5 multiplier_calc.py

multiplier_calc – Computes the topographic multipliers for a data line

This module is called by the module topomult

multiplier_calc.**multiplier_calc**(*line*, *data_spacing*)
Computes the multipliers for a data line

Parameters

- **line** – numpy.ndarray the elevation of a line
- **data_spacing** – *float* the distance between the neighour points

Returns numpy.ndarray the topographic values of the line

6.4.6 topo_mult.py

topomult – Calculate topographic multiplier

This module is called by the module all_multipliers to calculate the topographic multiplier for an input tile for 8 directions and output as NetCDF format.

References Yang, T., Nadimpalli, K. & Cechet, R.P. 2014. Local wind assessment in Australia: computation methodology for wind multipliers. Record 2014/33. Geoscience Australia, Canberra.

moduleauthor Tina Yang <tina.yang@ga.gov.au> Histroical authors: Xunguo Lin, Chris Thomas, Wenping Jiang, Craig Arthur

topo_mult.**remove_conservatism**(*mh_in*)
Remvoe the conservatism as described in the reference GA record

Parameters **mh_in** – numpy.ndarray the input topographic multiplier

Returns numpy.ndarray the output topographic multiplier

topo_mult.**tasmania**(*mh_in*, *dem*)
Apply the Tasmania factor for the topographic multiplier

Parameters

- **mh_in** – numpy.ndarray the input topographic multiplier
- **dem** – numpy.ndarray the input DEM value

Returns numpy.ndarray the output topographic multiplier

topo_mult.**topomult**(*input_dem*, *tile_extents_nobuffer*)
Executes core topographic multiplier functionality

Parameters

- **input_dem** – *file* the input tile of the DEM
- **tile_extents_nobuffer** – *tuple* the input tile extent without buffer

6.5 utilities

These are tools or functions used to support the main computation.!!

6.5.1 __init__.py

6.5.2 blrb.py

blrb – Functions for BiLinear Recursive Bisection (BLRB).

All shape references here follow the numpy convention (nrows, ncols), which makes some of the code harder to follow.
==========

moduleauthor Roger Edberg (roger.edberg@ga.gov.au)

blrb.**bilinear**(**args*, ***kwargs*)
Bilinear interpolation of four scalar values.

Parameters

- **shape** – Shape of interpolated grid (nrows, ncols).
- **f_ul** – Data value at upper-left (NW) corner.
- **f_ur** – Data value at upper-right (NE) corner.
- **f_lr** – Data value at lower-right (SE) corner.
- **f_ll** – Data value at lower-left (SW) corner.
- **dtype** – Data type (numpy I presume?).

Returns Array of data values interpolated between corners.

blrb.**indices**(**args*, ***kwargs*)
Generate corner indices for a grid block.

Parameters

- **origin** – Block origin (2-tuple).
- **shape** – Block shape (2-tuple: nrows, ncols).

Returns Corner indices: (xmin, xmax, ymin, ymax).

blrb.**interpolate_block**(**args*, ***kwargs*)
Interpolate a grid block.

Parameters

- **origin** – Block origin (2-tuple).
- **shape** – Block shape (nrows, ncols).
- **eval_func** (*callable; accepts grid indices i, j and returns a scalar value.*) – Evaluator function.
- **grid** (`numpy.array`.) – Grid array.

Returns Interpolated block array if grid argument is None. If grid argument is supplied its elements are modified in place and this function does not return a value.

blrb.**interpolate_grid**(**args*, ***kwargs*)
Interpolate a data grid.

Parameters

- **depth** (`int`) – Recursive bisection depth.
- **origin** (`tuple` of length 2.) – Block origin,
- **shape** (`tuple` of length 2 `(nrows, ncols)`.) – Block shape.
- **eval_func** (*callable; accepts grid indices i, j and returns a scalar value.*) – Evaluator function.
- **grid** (`numpy.array`.) – Grid array.

Todo Move arguments `eval_func` and `grid` to positions 1 and 2, and remove defaults (and the check that they are not `None` at the top of the function body).

blrb.**subdivide**(**args*, ***kwargs*)
Generate indices for grid sub-blocks.

Parameters

- **origin** – Block origin (2-tuple).
- **shape** – Block shape (nrows, ncols).

Returns

Dictionary containing sub-block corner indices:

{ 'UL': <list of 2-tuples>, 'UR': <list of 2-tuples>, 'LL': <list of 2-tuples>, 'LR': <list of 2-tuples> }

6.5.3 meta.py

Provides utilities for logging and meta programming.

class meta.**Singleton**
Metaclass for Singletons.

We could also keep the singletons in a dictionary in this class with keys of type class. I prefer, however, to keep them in the actual class.

meta.**create_arg_string**(*func*, **args*, ***kwargs*)
Constructs a string of the arguments passed to a function on a given invocation.

Parameters

- **func** – The function for which the string is to be constructed.
- **args** – The positional arguments passed in the call to `func`.
- **kwargs** – The keyword arguments passed in the call to `func`.

`meta.print_call`(*logger*)
Decorator which prints the call to a function, including all the arguments passed.

Parameters

- **func** – The function to be decorated.
- **logger** – Callable which will be passed the string representation of the function call. Then nologging is performed (the decorated is simply returned).

6.5.4 files.py

Provides utilities dealing with files.

`utilities.files.fl_config_file`(*extension='.ini'*, *prefix=''*, *level=None*)
Build a configuration filename (default extension .ini) based on the name and path of the function/module calling this function. Can also be useful for setting log file names automatically. If prefix is passed, this is preprended to the filename.

Parameters

- **extension** (*str*) – file extension to use (default '.ini'). The period ('.') must be included.
- **prefix** (*str*) – Optional prefix to the filename (default '').
- **level** – Optional level in the stack of the main script (default = maximum level in the stack).

Returns Full path of calling function/module, with the source file's extension replaced with extension, and optionally prefix inserted after the last path separator.

Example configFile = fl_config_file('.ini') Calling fl_config_file from /foo/bar/baz.py should return /foo/bar/baz.ini

`utilities.files.fl_get_stat`(*filename*, *chunk_whole=65536*)
Get basic statistics of filename - namely directory, name (excluding base path), md5sum and the last modified date. Useful for checking if a file has previously been processed.

Parameters

- **filename** (*str*) – Filename to check.
- **chunk_whole** (*int*) – (optional) chunk size (for md5sum calculation).

Returns path, name, md5sum, modification date for the file.

Raises

- **TypeError** – if the input file is not a string.
- **IOError** – if the file is not a valid file, or if the file cannot be opened.

Example dir, name, md5sum, moddate = fl_get_stat(filename)

`utilities.files.fl_load_file`(*filename*, *comments='%'*, *delimiter=','*, *skiprows=0*)
Load a delimited text file – uses `numpy.genfromtxt()`

Parameters

- **filename** (*file or str*) – File, filename, or generator to read

- **comments** (*str, optional*) – (default '%') indicator
- **delimiter** (*str, int or sequence, optional*) – The string used to separate values.

utilities.files.**fl_log_fatal_error**(*tblines*)
Log the error messages normally reported in a traceback so that all error messages can be caught, then exit. The input 'tblines' is created by calling `traceback.format_exc().splitlines()`.

Parameters **tblines** (*list*) – List of lines from the traceback.

utilities.files.**fl_mod_date**(*filename*, *dateformat='%Y-%m-%d %H:%M:%S'*)
Return the last modified date of the input file

Parameters

- **filename** (*str*) – file name (full path).
- **dateformat** (*str*) – Format string for the date (default '%Y-%m-%d %H:%M:%S')

Returns File modification date/time as a string

Return type str

Example modDate = fl_mod_date('C:/foo/bar.csv' , dateformat='%Y-%m-%dT%H:%M:%S')

utilities.files.**fl_module_name**(*level=1*)
Get the name of the module <level> levels above this function

Parameters **level** (*int*) – Level in the stack of the module calling this function (default = 1, function calling `fl_module_name`)

Returns Module name.

Return type str

Example mymodule = fl_module_name( ) Calling fl_module_name() from "/foo/bar/baz.py" returns "baz"

utilities.files.**fl_module_path**(*level=1*)
Get the path of the module <level> levels above this function

Parameters **level** (*int*) – level in the stack of the module calling this function (default = 1, function calling `fl_module_path`)

Returns path, basename and extension of the file containing the module

Example path, base, ext = fl_module_path( ), Calling fl_module_path() from "/foo/bar/baz.py" produces the result "/foo/bar", "baz", ".py"

utilities.files.**fl_program_version**(*level=None*)
Return the __version__ string from the top-level program, where defined.

If it is not defined, return an empty string.

Parameters **level** (*int*) – level in the stack of the main script (default = maximum level in the stack)

Returns version string (defined as the `__version__` global variable)

utilities.files.**fl_save_file**(*filename*, *data*, *header=''*, *delimiter=', '*, *fmt='%.18e'*)
Save data to a file.

Does some basic checks to ensure the path exists before attempting to write the file. Uses `numpy.savetxt` to save the data.

Parameters

- **filename** (*str*) – Path to the destination file.
- **data** – Array data to be written to file.
- **header** (*str*) – Column headers (optional).
- **delimiter** (*str*) – Field delimiter (default ‘,’).
- **fmt** (*str*) – Format statement for writing the data.

utilities.files.**fl_size**(*filename*)

Return the size of the input file in bytes

Parameters **filename** (*str*) – Full path to the file.

Returns File size in bytes.

Return type int

Example file_size = fl_size(‘C:/foo/bar.csv’)

utilities.files.**fl_start_log**(*log_file*, *log_level*, *verbose=False*, *datestamp=False*, *newlog=True*)

Start logging to log_file all messages of log_level and higher. Setting `verbose=True` will report all messages to STDOUT as well.

Parameters

- **log_file** (*str*) – Full path to log file.
- **log_level** (*str*) – String specifiying one of the standard Python logging levels (‘NOTSET’,’DEBUG’,’INFO’,’WARNING’,’ERROR’, ‘CRITICAL’)
- **verbose** (*boolean*) – `True` will echo all logging calls to STDOUT
- **datestamp** (*boolean*) – `True` will include a timestamp of the creation time in the filename.
- **newlog** (*boolean*) – `True` will create a new log file each time this function is called. `False` will append to the existing file.

Returns `logging.logger` object.

Example fl_start_log(‘/home/user/log/app.log’, ‘INFO’, verbose=True)

6.5.5 value_lookup.py

value_lookup – dictionaries relevant to terrain & shielding multipliers

Contains lookup dictionaries for classification, e.g.

```
>>> terrain_class_desc = dict([(1, 'City Buildings'),
>>>                            (2, 'Dense Forest'),
>>>                            (3, 'High Density Metro'),
>>>                            ...
>>>                            (14, 'orched/open forest'),
>>>                            (15, 'Mudflats/saltevaporators/sandy beaches')])
```

6.5.6 vincenty.py

class vincenty.**GreatCircle**(*rmajor*, *rminor*, *lon1*, *lat1*, *lon2*, *lat2*)

formula for perfect sphere from Ed Williams’ ‘Aviation Formulary’ (http://williams.best.vwh.net/avform.htm)

code for ellipsoid posted to GMT mailing list by Jim Leven in Dec 1999

Contact: Jeff Whitaker <jeffrey.s.whitaker@noaa.gov>

points (*npoints*)
compute arrays of npoints equally spaced intermediate points along the great circle.

Parameters **npoints** – the number of points to compute.

Returns lons, lats (lists with longitudes and latitudes of intermediate points in degrees).

Example npoints=10 will return arrays lons,lats of 10 equally spaced points along the great circle.

vincenty.**vinc_dist** (*f*, *a*, *phi1*, *lembda1*, *phi2*, *lembda2*)
Returns the distance between two geographic points on the ellipsoid and the forward and reverse azimuths between these points. lats, longs and azimuths are in radians, distance in metres

Parameters

- **f** – flattening
- **a** – equatorial radius (metres)
- **phi1** – latitude of first point
- **lembda1** – longitude of first point
- **phi2** – latitude of second point
- **lembda2** – longitude of second point

Returns (s, alpha12, alpha21) as a tuple

vincenty.**vinc_pt** (*f*, *a*, *phi1*, *lembda1*, *alpha12*, *s*)
Returns the lat and long of projected point and reverse azimuth given a reference point and a distance and azimuth to project.

Parameters lats, longs and azimuths passed in decimal degrees

Returns (phi2, lambda2, alpha21) as a tuple

6.5.7 get_pixel_size_grid.py

get_pixel_size_grid – calculate the image pixel size in meter

moduleauthor Alex Ip

class get_pixel_size_grid.**Earth**
Values relevant to earth.

get_pixel_size_grid.**get_pixel_size** (*dataset*, *xxx_todo_changeme*)
Returns X & Y sizes in metres of specified pixel as a tuple. N.B: Pixel ordinates are zero-based from top left

Parameters

- **dataset** – *file* the input dataset
- **xxx_todo_changeme** – *tuple* the input (x, y) point

Returns tuple of *float* the grid size at the input (x, y) point

get_pixel_size_grid.**get_pixel_size_grids** (*dataset*)
Returns two grids with interpolated X and Y pixel sizes for given datasets

Parameters **dataset** – *file* the input dataset

Returns tuple of `numpy.ndarray` grid sizes for input dataset

6.5.8 nctools.py

Tools used to produce output in netCDF format

`nctools.`**clip_array**(*data*, *x_left*, *y_upper*, *pixelwidth*, *pixelheight*, *extent*)
Return the clipped area of the input array according to an sub-extent

Parameters

- **data** – `numpy.ndarray` the input array
- **x_left** – *float* the left-most longitude vlaue
- **y_upper** – *float* the upper-most latitude values
- **pixelwidth** – *float* the pixel width
- **pixelheight** – *float* the pixel height
- **extent** – *tuple* the clipping extent

Returns `numpy.ndarray` the clipped array

`nctools.`**get_lat_lon**(*extent*, *pixelwidth*, *pixelheight*)
Return the longitude and latitude values that lie within the given extent

Parameters

- **extent** – *tuple* the clipping extent
- **pixelwidth** – *float* the pixel width
- **pixelheight** – *float* the pixel height

Returns lon: `numpy.ndarray` containing longitude values

Returns lat: `numpy.ndarray` containing latitude values

`nctools.`**nc_create_dim**(*ncobj*, *name*, *values*, *dtype*, *atts=None*)
Create a *dimension* instance in a `netcdf4.Dataset` or `netcdf4.Group` instance.

Parameters

- **ncobj** – `netCDF4.Dataset` or `netCDF4.Group` instance.
- **name** (*str*) – Name of the dimension.
- **values** (*numpy.ndarray*) – Dimension values.
- **dtype** (*numpy.dtype*) – Data type of the dimension.
- **atts** (*dict or None*) – Attributes to assign to the dimension instance

`nctools.`**nc_create_var**(*ncobj*, *name*, *dimensions*, *dtype*, *data=None*, *atts=None*, ***kwargs*)
Create a *Variable* instance in a `netCDF4.Dataset` or `netCDF4.Group` instance.

Parameters

- **ncobj** (`netCDF4.Dataset` or `netCDF4.Group`) – `netCDF4.Dataset` or `netCDF4.Group` instance where the variable will be stored.
- **name** (*str*) – Name of the variable to be created.
- **dimensions** (*tuple*) – dimension names that define the structure of the variable.

- **dtype** (`numpy.dtype`) – `numpy.dtype` data type.
- **data** (`numpy.ndarray` or None.) – `numpy.ndarray` Array holding the data to be stored.
- **atts** (*dict*) – Dict of attributes to assign to the variable.
- **kwargs** – additional keyword args passed directly to the `netCDF4.Variable` constructor

Returns `netCDF4.Variable` instance

Return type `netCDF4.Variable`

`nctools.`**nc_save_grid**(*filename*, *dimensions*, *variables*, *nodata=-9999*, *datatitle=None*, *gatts={}*, *writedata=True*, *keepfileopen=False*, *zlib=True*, *complevel=4*, *lsd=None*)

Save a gridded dataset to a netCDF file using NetCDF4.

Parameters

- **filename** (*str*) – Full path to the file to write to.
- **dimensions** – `dict` The input dict 'dimensions' has a strict structure, to permit insertion of multiple dimensions. The dimensions should be keyed with the slowest varying dimension as dimension 0.

```
dimesions = {0:{'name':
                'values':
                'dtype':
                'atts':{'long_name':
                        'units':  ...} },
             1:{'name':
                'values':
                'type':
                'atts':{'long_name':
                        'units':  ...} },
                      ...}
```

- **variables** – `dict` The input dict 'variables' similarly requires a strict structure:

```
variables = {0:{'name':
                'dims':
                'values':
                'dtype':
                'atts':{'long_name':
                        'units':
                        ...} },
             1:{'name':
                'dims':
                'values':
                'dtype':
                'atts':{'long_name':
                        'units':
                        ...} },
                 ...}
```

  The value for the 'dims' key must be a tuple that is a subset of the dimensions specified above.

- **nodata** (*float*) – Value to assign to missing data, default is -9999.
- **datatitle** (*str*) – Optional title to give the stored dataset.

- **gatts** (*dict* or None) – Optional dictionary of global attributes to include in the file.
- **dtype** (numpy.dtype) – The data type of the missing value. If not given, infer from other input arguments.
- **writedata** (*bool*) – If true, then the function will write the provided data (passed in via the variables dict) to the file. Otherwise, no data is written.
- **keepfileopen** (*bool*) – If True, return a netcdf object and keep the file open, so that data can be written by the calling program. Otherwise, flush data to disk and close the file.
- **zlib** (*bool*) – If true, compresses data in variables using gzip compression.
- **complevel** (*integer*) – Value between 1 and 9, describing level of compression desired. Ignored if zlib=False.
- **lsd** (*integer*) – Variable data will be truncated to this number of significant digits.

Returns *netCDF4.Dataset* object (if keepfileopen=True)

Return type netCDF4.Dataset

Raises

- **KeyError** – If input dimension or variable dicts do not have required keys.
- **IOError** – If output file cannot be created.
- **ValueError** – if there is a mismatch between dimensions and shape of values to write.

nctools.**save_multiplier**(*multiplier_name*, *multiplier_values*, *lat*, *lon*, *nc_name*)

Save multiplier data to a netCDF file.

Parameters

- **multiplier_name** – *string* the multiplier name
- **multiplier_values** – numpy.ndarray the multiplier values
- **lat** – numpy.ndarray containing latitude values
- **lon** – numpy.ndarray containing longitude values
- **nc_name** – *string* the netcdf file name

CHAPTER

SEVEN

MODULE INDEX

- genindex
- modindex
- search