Tropical Cyclone Risk Model User Guide

Release 1.0.2

Geoscience Australia

05 November 2015

The **Tropical Cyclone Risk Model** is a stochastic tropical cyclone model developed by Geoscience Australia for estimating the wind hazard from tropical cyclones.

Due to the relatively short record of quality-controlled, consistent tropical cyclone observations, it is difficult to estimate average recurrence interval wind speeds due to tropical cyclones. To overcome the restriction of observed data, TCRM uses an autoregressive model to generate thousands of years of events that are statistically similar to the historical record. To translate these events to estimated wind speeds, TCRM applies a parametric windfield and boundary layer model to each event. Finally an extreme value distribution is fitted to the aggregated windfields at each grid point in the model domain to provide ARI wind speed estimates.

Part I

Features

- **Multi-platform**: TCRM can run on desktop machines through to massively-parallel systems (tested on Windows XP/Vista/7, *NIX);
- **Multiple options for wind field & boundary layer models**: A number of radial profiles and simple boundary layer models have been included to allow users to test sensitivity to these options.
- **Globally applicable**: Users can set up a domain in any TC basin in the globe. The model is not tuned to any one region of the globe. Rather, the model is designed to draw sufficient information from best-track archives or TC databases;
- **Evaluation metrics**: Offers capability to run objective evaluation of track model metrics (e.g. landfall rates);
- **Single scenarios**: Users can run a single TC event (e.g. using a b-deck format track file) at high temporal resolution and extract time series data at chosen locations;

Part II

Releases

Latest releases can be downloaded from the Geoscience Australia GitHub repository.

Bleeding edge versions are accessible here.

Contributions are welcome – create a fork or clone the repo. Read the page on contributing to TCRM for more details.

Part III

Contents

CHAPTER

ONE

LICENSE INFORMATION

Geoscience Australia has tried to make the information in this product as accurate as possible. However, it does not guarantee that the information produced by this product is totally accurate or complete. Therefore, users should use information from this product as an indicative guide of the regional wind speed only, and should not solely rely on this information when making a decision.

This work was produced at Geoscience Australia, funded by the Commonwealth of Australia. Neither the Australian Government, Geoscience Australia nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Australian Government or Geoscience Australia. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Australian Government or Geoscience Australia, and shall not be used for advertising or product endorsement purposes.

This document does not convey a warranty, express or implied, of merchantability or fitness for a particular purpose.

1.1 User Manual

1.2 Software

CHAPTER

TWO

INTRODUCTION

Tropical cyclones are one of the more frequent natural disasters globally. The observed record is limited, so it is difficult to evaluate what the 'worst possible' cyclone would look like. Based on these historic records, we can statistically 'extrapolate' the record and use a stochastic model to generate plausible, synthetic TC events.

TCRM was borne out of a desire to explore the epistemic uncertainty around parameterisations used in tropical cyclone hazard models. There are a wide range of idealised radial profiles and boundary layer models available in the literature, and the choice of profile or boundary layer model will influence the resulting hazard (and risk) produced the model. TCRM allows users to run with a range of profiles or boundary layer models and evaluate the resulting differences, within a single modelling system. Users can also implement their own models into the code base (which can then be shared with the user community).

2.1 Model outline

TCRM has 5 main modules to generate hazard information: `DataProcess`, `StatInterface`, `TrackGenerator`, `wind` and `hazard`. Supporting *`Utilities`* and plotting routines (`PlotInterface`) are included.

The `DataProcess` module reads the input track database and extracts parameters from the data, namely intensity and location information, speed, bearings and genesis locations. The data are stored in text files to allow users to examine the data in another analysis tool. There are also a number of plotting routines to display some of the basic relationships in the data.

`StatInterface` uses the processed files generated by `DataProcess` to evaluate the statistical relations between the parameters, including the means, variance and autocorrelations on a coarse grid across the model domain. PDFs of the initial values of parameters (initial speed, bearing and intensity) and genesis probability are also calculated.

`TrackGenerator` is the stochastic engine of the model. This module samples from the distributions of genesis location and initial parameter values to start a new TC event, then steps forward in time using the autoregressive nature of TC behaviour. This allows users to generate thousands of random events that share the same statistical properties as the input track dataset.

The `wind` module calculates the maximum wind speed around each of the synthetic TC events generated by the `TrackGenerator`. A 2-dimensional parametric profile is used to calculate the wind field to enable large numbers of events to be processed efficiently, but at high spatial resolution (e.g. a grid spacing of around 2 km). Users can select from a number of radial profiles and combine with one of three boundary layer models to incorporate asymmetries associated with forward motion of the TC vortex.

`hazard` uses the wind fields calculated in the `wind` module to fit a generalised extreme value (GEV) distribution at each grid point across the model domain to determine return period wind speeds. The fitting routine uses the method of L-moments [1] to determine the location, shape and scale parameters of the GEV.

[1] Hosking, J. R. M. (1990): L-moments: Analysis and Estimation of Distributions using Linear Combinations of Order Statistics. *Journal of the Royal Statistical Society*, **52**, 105–124.

2.2 Software

The software is made available as an open-source package. Users can add new components to the model and are encouraged to submit them back to the project. Where possible, the code has been modularised to ease the process of adding new methods (such as radial profiles or boundary layer models).

2.3 References

CHAPTER

THREE

INSTALLATION

Installing TCRM is intended to be a simple process, requiring only a small amount of compilation and basic understanding of command line operations. TCRM has been installed and (lightly) tested on a range of unix-based systems, Windows and Mac OS/X systems.

3.1 Downloading

The TCRM code can be downloaded from the Geoscience Australia GitHub page.

For users wanting to only run the code, a zip file or gzipped tar file of the latest releases can be downloaded from the Releases page.

Those wanting to contribute to development can fork the repository. Submit a pull request to have your changes integrated into TCRM.

3.2 Setting the environment

To enable TCRM to run flawlessly, you may need to change some environment settings. The important variable to set is the `PYTHONPATH` variable. This should be set to the path where you have extracted the contents of the zip file. In the examples below, change `/path/to/tcrm` to the location where you extracted the TCRM files.

A complete discussion on environment variables in Python is given in the Python documentation.

3.2.1 Windows

The Python documentation contains some simple instructions for setting environment variables on Windows systems here. See this link for setting the variables on different Windows systems.

3.2.2 BASH shell

```
export PYTHONPATH=$PYTHONPATH:/path/to/tcrm:/path/to/tcrm/Utilities
```

3.2.3 CSH/TCSH shell

```
setenv PYTHONPATH $PYTHONPATH:/path/to/tcrm:/path/to/tcrm/Utilities
```

3.3 Dependencies

TCRM relies on a number of additional libraries that are not part of the standard library. There are several ways to obtain the required libraries – using Python's recommended tool pip, installing a distribution such as Python(x,y) package (for Windows environments) or Anaconda (cross-platform), or installing the libraries from source or binary installers (pre-compiled binary Windows installer versions for all the libraries (both 32-bit and 64-bit) can be obtained here).

For detailed instructions on installation of these dependencies, please see the documentation for each individual library.

- Python - v2.7 preferred
- Numpy - v1.6 or later
- Scipy - v0.12 or later
- Matplotlib v1.2 or later.
- Basemap
- netcdf4-python - version 1.0.8 or later
- Shapely - v1.2.15 or later
- Parallel execution in multi-processor environments (with MPI installed) requires Pypar

3.3.1 Using pip

If you have pip installed, the required modules can be installed using the following command, executed in the main TCRM directory:

```
pip -v install -r requirements.txt
```

This will automatically build the required libraries (listed in the `requirements.txt` file) and any dependencies. `pip` must be on the `$PATH` for this to work.

3.4 Compiling the extensions

3.4.1 Unix

The model requires a number of C extensions to be compiled before execution. These can be built using Python's inbuilt `distutils` module. Copy the required files from the *installer* directory to the base directory and then execute the build process:

```
python intaller/setup.py build_ext -i
```

3.4.2 Windows

For Windows users, the code includes the `compile.cmd` script in the main TCRM diretory that will build these extensions in place.

3.5 Testing the installation

The model code includes a suite of unit tests that ensure elements of the code base will work as expected, even if a user makes modificaitons to the code.

The test suite can be run from the main directory. On Windows, run the `run_test_all.cmd` script from the main TCRM directory. On Unix, use the command:

```
python ./tests/run.py
```

This should report no errors or failures.

3.5.1 Special note for Windows systems

On a Windows system, `tests.test_files.testModulePath()` may fail due to the different path separators (`/` versus `\\`) used by the Windows system. This test failure will appear as:

```
======================================================================
FAIL: testModulePath (tests.test_files.TestModuleUtilities)
Test flModulePath returns correct path, base & extension
----------------------------------------------------------------------
Traceback (most recent call last):
  File "tcrm\tests\test_files.py", line 22, in testModulePath
    self.assertEqual(self.path, p)
AssertionError: 'tcrm/tests' != 'tcrm\\tests'

----------------------------------------------------------------------
Ran 111 tests in 92.513s

FAILED (failures=1)
```

Such an error will not affect model execution.

CHAPTER

FOUR

SETTING UP THE MODEL

Execution of TCRM is controlled by reading the simulation settings from a configuration file. The configuration file is a text file, and can be edited in any text editor (e.g. Notepad, Wordpad, vi, emacs, gedit). An example configuration file is provided in the examples folder to give users a starting point.

4.1 The configuration file

The TCRM configuration file is divided into a series of sections, each with a set of option/value pairs. Most options have default values and may not need to be specified in the configuration file. One value that has no default is the Region gridLimit option. This defines the model domain and must be set in any configuration file used.

4.1.1 Actions

This section defines which components of TCRM will be executed. The options are:

- *DownloadData* - download input datasets (defaults are included)
- *DataProcess* - process the input TC track database
- *ExecuteStat* - calculate the TC statistics over the model domain
- *ExecuteTrackGenerator* - generate a set of stochastic TC tracks
- *ExecuteWindfield* - Calculate the wind field around a set of TC tracks
- *ExecuteHazard* - Calculate the return period wind speeds from a set of wind field files
- *PlotHazard* - Plot the return period wind speed maps and return period curves for locations in the model domain
- *PlotData* - Plot some basic statistical analyses of the input TC track database
- *ExecuteEvaluate* - Evaluate a set of stochastic TC tracks, comparing to the input TC track database.

All options are boolean (i.e. `True` or `False`).

```
[Actions]
DataProcess = True
ExecuteStat = True
ExecuteTrackGenerator = True
ExecuteWindfield = True
ExecuteHazard = True
PlotHazard = True
PlotData = False
ExecuteEvaluate = False
DownloadData = True
```

4.1.2 Region

This section defines the model domain and the size of the grid over which statistics are calculated. The model domain (`gridLimit`) is specified as a Python dict with keys of `xMin`, `xMax`, `yMin` and `yMax`. This sets the domain over which the wind fields and hazard will be calculated. Stochastic tracks are generated over a broader domain. The `gridSpace` option controls the size of the grid cells, which are used for calculating statistics. At this time, the values here must be integer values, but can be different in the `x` (east-west) and `y` (north-south) directions. The `gridInc` option control the incremental increase in grid cell size when insufficient observations are located within a grid cell (see the `StatInterface` description):

```
[Region]
gridLimit = {'xMin': 113.0, 'xMax': 124.0, 'yMin': -24.0, 'yMax': -13.0}
gridSpace = {'x':1.0,'y':1.0}
gridInc = {'x':1.0,'y':0.5}
```

4.1.3 DataProcess

This section controls aspects of the processing of the input track database. Firstly, the `InputFile` option specifies the file to be processed. A relative or absolute path can be used. If no path name is included (as in the example below), then TCRM assumes the file is stored in the `input` path. If using an automatically downloaded dataset, then this file name must match the name specified in the appropriate dataset section (which is named by the `Source` option in this section) of the configuration file (further details below).

The `Source` option is a string value that acts as a pointer to a subsequent section in the configuration file, that holds details of the input track file structure.

The `StartSeason` and `FilterSeason` options control what years of the input track database are used in calibrating the model. In the default case, only data from 1981 onwards is used for model calibration. If `FilterSeasons = False`, no season filtering is performed and the full input track database is used.

```
[DataProcess]
InputFile = Allstorms.ibtracs_wmo.v03r05.csv
StartSeason = 1981
FilterSeasons = True
Source = IBTRACS
```

4.1.4 StatInterface

The `StatInterface` section controls the methods used to calculate distributions of TC parameters from the input track database.

`kdeType` and `kde2DType` specify the kernel used in the kernel density estimation method for creating probability density functions that are used in selecting initial values for the stochastic TC events (e.g. longitude, latitude, initial pressure, speed and bearing). `kdeStep` defines the increment in the generated probability density functions and cumulative distribution functions.

`minSamplesCell` sets the minimum number of valid observations in each grid cell that are required for calculating the distributions, variances and autocorrelations used in the `TrackGenerator` module. If there are insufficient valid observations, then the bounds of the grid cell are incrementally increased (in steps as specified by the `gridInc` values) until sufficient observations are found.

```
[StatInterface]
kdeType = Gaussian
kde2DType = Gaussian
kdeStep = 0.2
minSamplesCell = 100
```

4.1.5 TrackGenerator

The `TrackGenerator` section controls the stochastic track generation module. It is here that users can control the number of events and the number of years generated.

The `NumSimulations` option sets the number of TC event sets that will be generated. Any integer number of events (up to 1,000,000) is possible. `YearsPerSimulation` sets the number of simulated years that will be generated for each event set. For evaluating hazard, the value should be set to 1, as the extreme value distribution fitting process assumes annual maxima. The annual frequency of events is based on a Poisson distribution around the mean annual frequency, which is determined from the input track database.

For track model evaluations, it is recommended to set `YearsPerSimulation` to a similar number to the number of years in the input track database. For example, in our testing that used data from 1981–2013, we set the value to 30.

`NumTimeSteps` controls the maximum lifetime an event can exist for. `TimeStep` sets the time interval (in hours) for the track generator. `SeasonSeed` and `TrackSeed` are used to fix the random number generator on parallel systems to ensure truly random numbers on each individual processor.

```
[TrackGenerator]
NumSimulations = 500
YearsPerSimulation = 1
NumTimeSteps = 360
TimeStep = 1.0
SeasonSeed = 1
TrackSeed = 1
```

4.1.6 WindfieldInterface

The `WindfieldInterface` section controls how the wind fields from each track in the simulated tracks are calculated. There are two main components to the wind field – the radial profile and the boundary layer model.

The `profileType` option sets the radial profile used. Valid values are:

- `holland` – the radial profile of Holland (1980) [1]
- `powell` – Similar to the Holland profile, but uses a variable beta parameter that is a function of latitude and size. [2]
- `schloemer` – From Schloemer (1954) – essentially the Holland profile with a beta value of 1 [3]
- `willoughby` – From Willoughby and Rahn (2004). Again, the Holland profile, with beta a function of the maximum wind speed, radius to maximum wind and latitude [4]
- `jelesnianski` – From Jelesnianski (1966). [5]
- `doubleHolland` – A double exponential profile from McConochie *et al.* (2004) [6]

The `windFieldType` value selects the boundary layer model used. Three boundary layer models have been implemented:

- `kepert` – the linearised boundary layer model of Kepert (2001) [7]

[1] Holland, G. J. (1980): An Analytic Model of the Wind and Pressure Profiles in Hurricanes. *Monthly Weather Review*, **108**

[2] Powell, M., G. Soukup, S. Cocke, S. Gulati, N. Morisseau-Leroy, S. Hamid, N. Dorst, and L. Axe (2005): State of Florida hurricane loss projection model: Atmospheric science component. *Journal of Wind Engineering and Industrial Aerodynamics*, **93**, 651–674

[3] Schloemer, R. W. (1954): Analysis and synthesis of hurricane wind patterns over Lake Okeechobee. *NOAA Hydrometeorology Report* **31**, 1954

[4] Willoughby, H. E. and M. E. Rahn (2004): Parametric Representation of the Primary Hurricane Vortex. Part I: Observations and Evaluation of the Holland (1980) Model. *Monthly Weather Review*, **132**, 3033–3048

[5] Jelesnianski, C. P. (1966): Numerical Computations of Storm Surges without Bottom Stress. *Monthly Weather Review*, **94**, 379–394

[6] McConochie, J. D., T. A. Hardy, and L. B. Mason (2004): Modelling tropical cyclone over-water wind and pressure fields. *Ocean Engineering*, **31**, 1757–1782

[7] Kepert, J. D. (2001): The Dynamics of Boundary Layer Jets within the Tropical Cyclone Core. Part I: Linear Theory. *J. Atmos. Sci.*, **58**, 2469–2484

- `hubbert` – a vector addition of forward speed and tangential wind speed from Hubbert *et al.* (1994) [8]
- `mcconochie` – a second vector addition model, from McConochie *et al.* (2004) [6]

The `beta` option specifies the β parameter used in the Holland wind profile. The additional β options (`beta1` and `beta2`) are used in the `doubleHolland` wind profile, which is a double exponential profile, therefore requiring two β parameters.

`thetaMax` is used in the McConochie and Hubbert boundary layer models to specify the azimuthal location of the maximum wind speed under the translating storm.

`Margin` defines the spatial extent over which the wind field is calculated and is in units of degrees. A margin of 5 is recommended for hazard models, to ensure low wind speeds from distant TCs are incorporated into the fitting procedure.

`Resolution` is the horizontal resolution (in degrees) of the wind fields. Values should be no larger than 0.05 degrees, as the absolute peak of the radial profile may not be adequately resolved, leading to an underestimation of the maximum wind speeds.

```
[WindfieldInterface]
profileType = holland
windFieldType = kepert
beta = 1.3
beta1 = 1.3
beta2 = 1.3
thetaMax = 70.0
Margin = 2
Resolution = 0.05
```

4.1.7 Hazard

The `Hazard` section controls how the model calculates the return period wind speeds, and whether to calculate confidence ranges.

The `Years` option is a comma separated list of integer values that specifies the return periods for which wind speeds will be calculated. `MinimumRecords` sets the minimum number of values required for performing the fitting procedure at a given grid point.

`CalculateCI` sets whether the `hazard` module will calculate confidence ranges using a bootstrap resampling method. If `True`, the module will run the fitting process multiple times and calculate upper and lower percentile values of the resulting return period wind speeds. The `PercentileRange` option sets the range – for a value of 90, the module will calculatae the 5th and 95th percentile values. `SampleSize` sets the number of randomly selected values that will be used in each realisation of the extreme value fitting procedure for calculating the confidence range.

```
[Hazard]
Years = 2,5,10,20,25,50,100,200,250,500,1000
MinimumRecords = 50
CalculateCI = True
PercentileRange = 90
SampleSize = 50
PlotSpeedUnits = mps
```

4.1.8 RMW

The `RMW` section contains a single option: `GetRMWDistFromInputData`. Set this value to `True` if the input track database has reliable data on the radius to maximum winds.

[8] Hubbert, G. D., G. J. Holland, L. M. Leslie and M. J. Manton (1991): A Real-Time System for Forecasting Tropical Cyclone Storm Surges. *Weather and Forecasting*, **6**, 86–97

```
[RMW]
GetRMWDistFromInputData = False
```

4.1.9 Input

The `Input` section sets the source of some supplementary data, as well as the datasets to be automatically downloaded. The `LandMask` option specifies the path to a netcdf file (supplied) that contains a land/sea mask. The `MSLPFile` option specifies the path to a netcdf file (downloaded) that contains daily long-term mean sea level pressure data (e.g. from a NCEP/NCAR reanalysis products).

The `Datasest` option is a comma separated list of values indicating the data that should be downloaded on first execution. For each value in the list, there must be a corresponding section in the configuration file, that has options of `URL` (the URL of the data to be downloaded), `path` (where to store the data once it has been downloaded) and `filename` (the filename to give to the data once downloaded).

In the example below, for the `IBTRACS` dataset, there are additional options that describe the format of the track database with the same name. This is a legitimate approach, so long as there are no duplicate options.

Note that the `filename` option in the `IBTRACS` section matches the `InputFile` option in the `DataProcess` section, and the `filename` in the `LTMSLP` section matches the `MSLPFile` in the `Input` section.

The `CoastlineGates` option specifies the path to a comma-delimited text file that holds the points of a series of coastline gates that are used in the `Evaluate.landfallRates` module.

```
[Input]
LandMask = input/landmask.nc
MSLPFile = MSLP/slp.day.ltm.nc
Datasets = IBTRACS,LTMSLP
CoastlineGates = input/gates.csv

[IBTRACS]
URL = ftp://eclipse.ncdc.noaa.gov/pub/ibtracs/v03r05/wmo/csv/Allstorms.ibtracs_wmo.v03r05.csv.gz
path = input
filename = Allstorms.ibtracs_wmo.v03r05.csv
Columns = tcserialno,season,num,skip,skip,skip,date,skip,lat,lon,skip,pressure
FieldDelimiter = ,
NumberOfHeadingLines = 3
PressureUnits = hPa
LengthUnits = km
DateFormat = %Y-%m-%d %H:%M:%S
SpeedUnits = kph

[LTMSLP]
URL = ftp://ftp.cdc.noaa.gov/Datasets/ncep.reanalysis.derived/surface/slp.day.1981-2010.ltm.nc
path = MSLP
filename = slp.day.ltm.nc
```

4.1.10 Output

The `Output` section defines the destination of the model output. Set the `Path` option to the directory where you wish to store the data. Paths can be relative or absolute. By default, output is stored in a subdirectory of the working directory named `output`.

```
[Output]
Path = output
```

4.1.11 Logging

The `Logging` section controls how the model records progress to file (and optionally STDOUT). `LogFile` option specifies the name of the log file. If no path is given, then the log file will be stored in the current working directory. For parallel execution, a separate log file is created for each thread, with the rank of the process appended to the name of the file.

The `LogLevel` is one of the `Logging` levels. Default is `INFO`. The `Verbose` option allows users to print all logging messages to the standard output. This can be useful when attempting to identify problems with execution. For parallel execution, this is set to `False` (to prevent repeated messages being printed to the screen). Setting the `ProgressBar` option to `True` will display a simple progress bar on the screen to indicate the status of the model execution. This will be turned off if TCRM is executed on a parallel system, or if it is run in batch mode.

```
[Logging]
LogFile = main.log
LogLevel = INFO
Verbose = False
ProgressBar = False
```

4.1.12 Source format options

For the input data source specified in the *DataProcess* → *Source* option, there must be a corresponding section of the given name. In this example case, the source is specified as `IBTRACS` (the same as one of the `Dataset` options). The `IBTRACS` section therefore controls both the download dataset options, and specifies the textural format of the input track database.

The options that relate to the dataset download are `URL`, `path` and `filename`. `URL` specifies the location of the data to be downloaded. The `path` option specifies the path name for the storage location of the dataset. The `filename` option gives the name of the file to be saved (this can be different from the name of the dataset).

The remaining options relate to the format of the track database. `Columns` is a comma-separated list of the column names in the input database. If a column is to be ignored, it should be named `skip`. The `FieldDelimiter` is the delimiter used in the input track database (it's assumed that the input file is a text format file!). The `NumberOfHeadingLines` indicates the number of text lines at the top of the file that should be ignored (usually this is column headers – due to the multiple lines used in some track databases, TCRM does not attempt to decipher the column names from the header. `PressureUnits`, `LengthUnits` and `SpeedUnits` specify the units the numerical values of pressure, distance and speed (respectively) used in the input track database. The `DateFormat` option is a string represenation of the date format used in the track database. The format should use Python's datetime formats.

```
[IBTRACS]
URL=ftp://eclipse.ncdc.noaa.gov/pub/ibtracs/v03r05/wmo/csv/Allstorms.ibtracs_wmo.v03r05.csv.gz
path=input
filename=Allstorms.ibtracs_wmo.v03r05.csv
Columns=tcserialno,season,num,skip,skip,skip,date,skip,lat,lon,skip,pressure
FieldDelimiter=,
NumberOfHeadingLines=3
PressureUnits=hPa
LengthUnits=km
DateFormat=%Y-%m-%d %H:%M:%S
SpeedUnits=kph
```

4.1.13 References

CHAPTER

FIVE

RUNNING THE MODEL

The primary way to run TCRM is at the command line (A graphical interface is under development). Command line arguments are passed to the main `tcrm.py` script for defining the path to the configuration file, as well as enabling verbose output and/or debugging.

5.1 Command line arguments

-c file, --config file	Path to a configuration file
-v, --verbose	If given, then logging messages will be printed to the console
-d, --debug	In the case that execution results in an exception, allow the Python stack to call into the stack trace (through implementation of a custom hook script).

5.2 Examples

Make sure `python` is in your system path, then from the base directory, call the `tcrm.py` script, with the configuration file option included. For example, to run the example simulation:

```
python tcrm.py -c example/port_hedland.ini
```

The model will print a simple progress indicator to the console to show that the model is working.

If the `-v` option is included, then all logging messages will be printed to the console. The level of logging detail is set in the configuration file.

5.3 Running on a parallel system

As a stochastic model, TCRM can generate massive numbers of synthetic events, which implies run times can become very long. If TCRM is installed on a multiprocessor system (either a shared memory or distributed memory system), then the workload can be shared across the workload. TCRM uses the `pypar` module to enable multi-threaded processing. Instructions for installing `pypar` are given on the Pypar website.

Pypar is built around the MPI library, and so uses the `mpirun` command to execute the model across multiple processors. As an example the following command will execute across 16 processors:

```
mpirun -np 16 python tcrm.py -c example/port_hedland.ini
```

Running across multiple processors means that logging messages from each individual processor can get mixed up with others. To avoid this, a separate log file is created for each thread, and output to the console is suppressed (even if the `-v` or `--verbose` option is given at the command line).

CHAPTER

SIX

EXAMPLES

6.1 Full TCRM hazard simulation

An example configuration file is provided with the code to provide users a starting point for testing the installation, and for setting up their own simulation. The example simulation is centred on Port Hedland, Australia (118.6E, 20.3S), which has experienced numerous severe tropical cyclones.

Once the model has been *installed* and tested, the example simulation can be run as follows:

```
$ python tcrm.py -c example/port_hedland.ini
```

The model will automatically create output folders to store the results in, as well as a log file that provides details on the progress of the model. The simulation will process the input track database (IBTrACS v03r05), run the statistical interface, generate 1000 simulated years of TC events, the wind swaths associated with those simulated years, calculate the hazard (return period wind speeds) for the region and plot the output as maps and a return period curve for Port Hedland. The hazard data are also stored in netCDF (version 4) files.

6.1.1 Running in an MPI environment

For a multiprocessor environment using MPI and Pypar, the example can be run with:

```
$ mpirun -np 16 python tcrm.py -c example/port_hedland.ini
```

This will execute TCRM across 16 CPUs (if available).

TCRM has been tested on systems up to 256 CPUs. Testing with moderate event sets (4000 events) indicate > 23 times speedup when run across 32 CPUs.

6.2 Scenario simulation

An example scenario is also included with the code to demonstrate an individual event simulation. This uses Tropical Cyclone *Tracy*, which impacted Darwin, Australia in 1974. The example simulation uses the best track data captured in the IBTrACS dataset, with the radius to maximum winds sourced from JTWC. TC *Tracy* struck Darwin early on Christmas Morning (1974), and resulted in 68 deaths and the largest peacetime evacuation of a city in Australia's history. A more detailed simulation of TC Tracy using TCRM is given in Schofield *et al.* (2009).

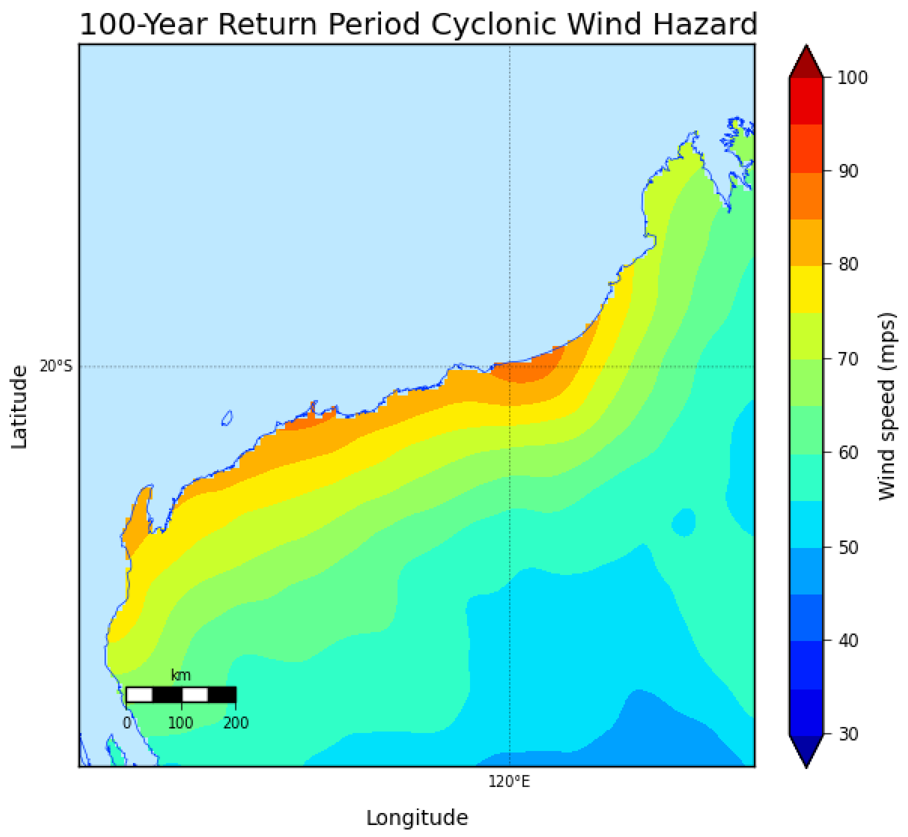

Fig. 6.1: 100-year return period wind speed near Port Hedland, Australia. Wind speeds represent a 3-second gust wind speed, 10 metres above ground level in open, flat terrain.

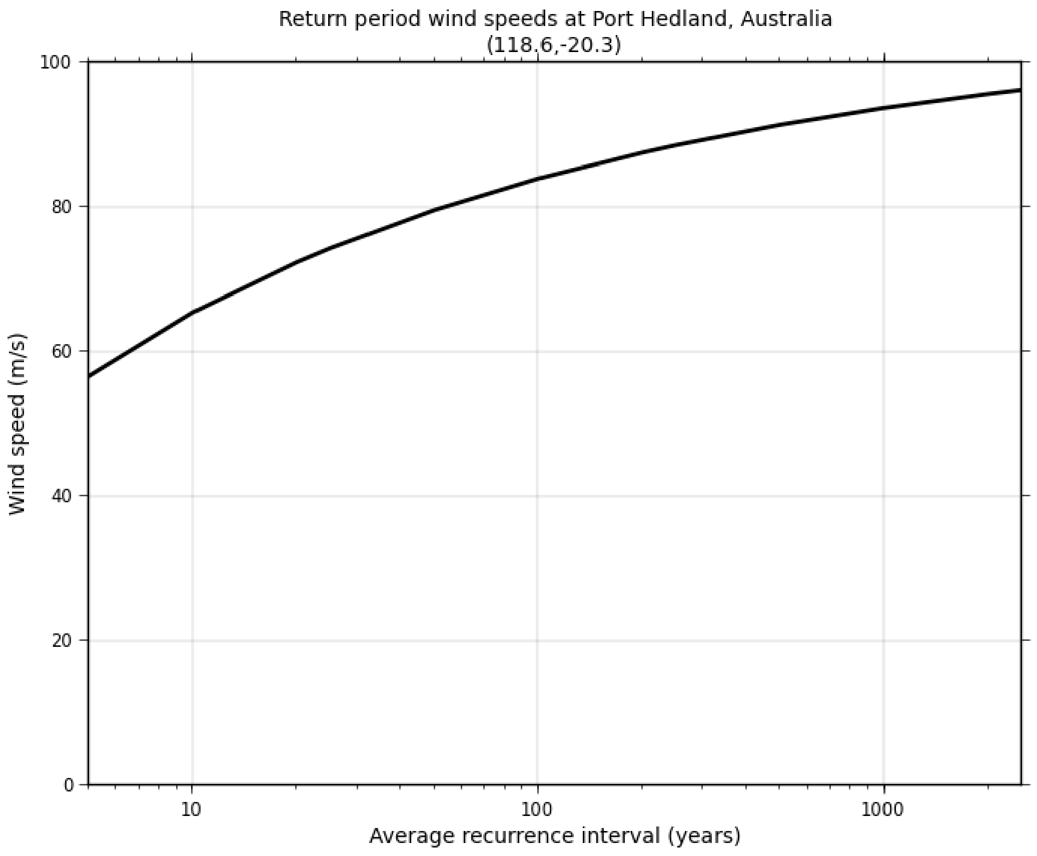

Fig. 6.2: Hazard curve for Port Hedland, Australia, based on 1000 years of synthetic tropical cyclone events. The grey shading indicates the 90th percentile range, based on a bootstrap resampling procedure.

CHAPTER

SEVEN

EVALUATION

TCRM provides functionality to evaluate the performance of the track model over the model domain through the `Evaluate` module. By running a number of simulations, each generating a similar number of years of events to the input track database, we can compare the synthetic events with the observations. Metrics include track density, genesis distribution, minimum pressure distributions, landfall rates and longitude crossing rates.

7.1 Setup

An example configuration for executing an evaluation. Other sections of the configuration file should remain unchanged.

```
[Actions]
DataProcess=True
ExecuteStat=True
ExecuteTrackGenerator=True
ExecuteWindfield=False
ExecuteHazard=False
PlotHazard=False
PlotData=False
ExecuteEvaluate=False
DownloadData=True

[Input]
LandMask=input/landmask.nc
MSLPFile=MSLP/slp.day.ltm.nc
CoastlineGates=input/gates.txt

[TrackGenerator]
NumSimulations=500
YearsPerSimulation=30
NumTimeSteps=360
TimeStep=1.0
SeasonSeed=1
TrackSeed=1
```

Note that the `YearsPerSimulation` option is now set to 30, so TCRM will generate 500 event sets, each with the equivalent of 30 years of TCs. The annual frequency of events is based on a Poisson distribution around the mean annual frequency, which is determined from the input track database.

A csv-format file containing a set of gates around the coastline is required for `Evaluate.landfallRates`. Gates are defined as a series of points around the coastline in number, longitude, latitude format. Below are the first 15 points in the sample gates provided in the code base.

```
0.00, 114.78, -34.70
1.00, 115.20, -32.95
2.00, 114.90, -31.17
3.00, 114.49, -29.42
4.00, 113.66, -27.83
```

```
5.00, 112.73, -26.28
6.00, 112.96, -24.50
7.00, 113.22, -22.72
8.00, 114.47, -21.43
9.00, 115.98, -20.44
10.00, 117.74, -20.08
11.00, 119.46, -19.55
12.00, 121.12, -18.84
13.00, 121.74, -17.15
14.00, 123.04, -15.91
15.00, 124.49, -14.85
```

7.2 Output

The `Evaluate` module generates a set of figures that compare the synthetic event set to the input track database. For the synthetic events, the module calculates a mean value and upper and lower percentiles of the distribution (commonly the 5th and 95th percentile values).

Some of the results are also saved to netCDF files (track density, pressure distributions and longitude crossing rates) for users to import into other graphics packages.

CHAPTER

EIGHT

USING DIFFERENT TC DATABASES - SOURCE FORMATS

There are a wide variety of formats of tropical cyclone track databases available, both observational and model generated (e.g. we have used tropical cyclone-like vortices derived from regional and global climate models). To introduce users to the model, we include the format for the International Best Tracks Archive for Climate Stewardship (IBTrACS-WMO format). The default action for TCRM is to download the current version (v03r05 at August 2014) and use this to calibrate the model.

8.1 Required fields

As a bare minimum, TCRM requires data on the date/time, longitude, latitude, central pressure of each cyclone observation. It also requires a field that indicates each unique TC event in the database. This can be a `tcserialno` - a unique string for each event; a TC `num` which is an integer value for each event in a season; or an `indicator` field, which is set to `1` for the first observation of a new TC, and `0` otherwise.

For individual scenario modelling, an additional field containing the radius to maximum winds (`rmax`) must be included.

8.2 Database format

The first few lines of the IBTrACS database are shown below.

```
IBTrACS WMO: International Best Tracks Archive for Climate Stewardship -- WMO DATA ONLY -- Versio
Serial_Num,Season,Num,Basin,Sub_basin,Name,ISO_time,Nature,Latitude,Longitude,Wind(WMO),Pres(WMO)
N/A,Year,#,BB,BB,N/A,YYYY-MM-DD HH:MM:SS,N/A,deg_north,deg_east,kt,mb,N/A,%,%,N/A
1848011S09080,1848,02, SI, MM,XXXX848003,1848-01-11 06:00:00, NR, -8.60,  79.80,  0.0,    0.0,reu
1848011S09080,1848,02, SI, MM,XXXX848003,1848-01-12 06:00:00, NR, -9.00,  78.90,  0.0,    0.0,reu
1848011S09080,1848,02, SI, MM,XXXX848003,1848-01-13 06:00:00, NR,-10.40,  73.20,  0.0,    0.0,reu
1848011S09080,1848,02, SI, MM,XXXX848003,1848-01-14 06:00:00, NR,-12.80,  69.90,  0.0,    0.0,reu
1848011S09080,1848,02, SI, MM,XXXX848003,1848-01-15 06:00:00, NR,-13.90,  68.90,  0.0,    0.0,reu
1848011S09080,1848,02, SI, MM,XXXX848003,1848-01-16 06:00:00, NR,-15.30,  67.70,  0.0,    0.0,reu
1848011S09080,1848,02, SI, MM,XXXX848003,1848-01-17 06:00:00, NR,-16.50,  67.00,  0.0,    0.0,reu
1848011S09080,1848,02, SI, MM,XXXX848003,1848-01-18 06:00:00, NR,-18.00,  67.40,  0.0,    0.0,reu
```

8.3 Setting the configuration options

The columns arrangement can be determined from looking at the second line of the database: `Serial_Num,Season,Num,Basin,Sub_basin,Name,ISO_time,Nature,Latitude,Longitude,Wind(WMO),Percentile,Pres(WMO) Percentile,Track_type`. The important variables required for TCRM are the `Serial_Num`, `ISO_time`, `Latitude`, `Longitude` and `Pres(WMO)` columns. All other variables can be ignored. Based on this, the `Columns` option can be set as:

```
Columns = tcserialno,season,num,skip,skip,skip,date,skip,lat,lon,skip,pressure
```

Notice that there are only 12 columns specified, but the database contains 15 columns. The last three columns are not used, so since they are not specified, they are automatically ignored.

The data are comma-delimited, so the `FieldDelimiter` option is set to a comma – it does not need to be wrapped in quotes.

```
FieldDelimiter = ,
```

The first three lines are metadata and information that describe the data (version, column names, units). These lines are ignored when the `NumberOfHeadingLines` option is set to 3.

```
NumberOfHeadingLines = 3
```

The format of the date field needs to be specified in the configuration file. Format specification is controlled by Python's standard `datetime` module.

```
DateFormat = %Y-%m-%d %H:%M:%S
```

The remaining three options – `PressureUnits`, `LengthUnits` and `SpeedUnits` – set the units of the pressure values, any length units and the speed units (of motion of the storms). Notice that the third line in the database indicates the wind speed is in units of `kt` (knots) - this refers to the observed maximum wind speed. The `SpeedUnits` option controls the units for the forward motion speed of storms - a field that is not included in the IBTrACS database.

```
PressureUnits = hPa
LengthUnits = km
SpeedUnits = kmh
```

The full section for using IBTrACS is shown below.

```
[IBTRACS]
; Input data file settings
URL = ftp://eclipse.ncdc.noaa.gov/pub/ibtracs/v03r05/wmo/csv/Allstorms.ibtracs_wmo.v03r05.csv.gz
Path = input
Filename = Allstorms.ibtracs_wmo.v03r05.csv
NumberOfHeadingLines = 3
Columns = tcserialno,season,num,skip,skip,skip,date,skip,lat,lon,skip,pressure
FieldDelimiter = ,
DateFormat = %Y-%m-%d %H:%M:%S
PressureUnits = hPa
LengthUnits = km
SpeedUnits = kph
```

CHAPTER

NINE

SCENARIO MODELLING

TCRM can also be used to simulate the wind field from an individual event. Given the track of a tropical cyclone, users can run the `wind` module only, and generate the maximum wind swath from a TC. The *tcevent.py* script enables users to efficiently run a scenario simulation, including performing temporal interpolation of the track positions to generate a realistic representation of the wind field. This can be useful for exploring synthetic events in more detail, or for analysing the wind field from an historical tropical cyclone.

TCRM does not currently account for local landscape effects on wind speed (e.g. topographic enhancement or changes in surface roughness due to vegetation or built environments). Users should determine the best way to include these effects for their own purposes.

As TCRM uses a parametric profile, the primary vortex is axisymmetric, and asymmetry in the surface winds arises solely due to the forward motion of the cyclone and the (uniform) surface friction. For a complete and accurate representation of the winds from an actual tropical cyclone, the simulated winds from TCRM should be combined with observed maximum wind speeds in the vicinity of the cyclone using spatial interpolation methods (e.g. kriging).

9.1 Setting up a scenario

9.1.1 Track file requirements

Simulating an individual event requires details of the track of the cyclone. As a minimum, a track file must contain the following fields: `index`, `indicator` or `tcserialno` (to uniquely identify an event); date/time information (either as a single field, or as individual components); latitudel; longitude; central pressure and radius to maximum winds.

The `index` field (or the alternatives) is required as the software uses the same methods to read the source data for both individual events and best-track databases that contain many events. The example configuration below uses an `index` field, with `1` in the first row and `0` in all subsequent rows.

The radius to maximum wind is required in individual events, as it cannot be artificially generated, as is the case in the generation of synthetic events.

9.1.2 Configuration

Users will need to manually edit a configuration file to execute a scenario simulation. Many of the options in the main TCRM configuration file are not required for scenario simulation, so it is recommended to create a reduced configuration file that contains only the required sections and options.

A basic configuration file for a scenario simulation would look like this:

```
[DataProcess]
InputFile = scenario.csv
Source = NRL
FilterSeasons = False
```

```
[WindfieldInterface]
Margin = 3
Resolution = 0.02
profileType = powell
windFieldType = kepert

[Timeseries]
Extract = True
StationFile = ./input/stationlist.shp
StationID = WMO

[Input]
landmask = input/landmask.nc
mslpfile = MSLP/slp.day.ltm.nc

[Output]
Path = ./output/scenario

[NRL]
Columns = index,skip,skip,date,lat,lon,skip,skip,pressure,rmax
FieldDelimiter = ,
NumberOfHeadingLines = 0
PressureUnits = hPa
SpeedUnits = kts
LengthUnits = km
DateFormat = %Y%m%d %H:%M

[Logging]
LogFile = output/scenario/log/scenario.log
LogLevel = INFO
Verbose = True
NewLog = True
DateStamp = True
```

9.2 Running the scenario

The *tcevent.py* script loads the track file, performs the temporal interpolation and then passes the interpolated track file to the `wind` module. Since the `Region` section has been removed, the `wind.run()` method will set the grid domain to cover the entire extent of the track.

Make sure `python` is in your system path, then from the base directory, call the `tcevent.py` script, with the configuration file option included. For example, to run the example scenario:

```
python tcevent.py -c example/scenario.ini
```

If the `-v` option is included, then all logging messages will be printed to the console. The level of logging detail is set in the configuration file.

Note *tcevent.py* cannot be executed in parallel.

9.3 Command line arguments

-c file, --config file	Path to a configuration file.
-v, --verbose	If given, logging messages will be printed to the console.
-d, --debug	In the case that execution results in an exception, allow the Python stack to call into the stack trace (through implementation of a custom hook script) and start the Python debugger (`pdb`).

9.4 Extract time series data

When running a scenario, it is possible to extract a time series of the wind speed and sea level pressure values from the grid at selected locations. The locations are defined in a user-supplied point shape file (a location database is planned for inclusion in future versions to better facilitate this feature). The shape file should contain a field with a unique identifier, otherwise station output is numbered sequentially through the locations.

- Locations must be provided in geographic coordinates (longitude, latitude coordinates). No reprojection is performed.
- Output is a 'regional' wind speed – that is, the wind speed at that location, excluding local topographic or landscape effects. These effects can be incorporated offline (i.e. outside the TCRM framework).

The data is stored in a separate csv file for each location, and data is plotted on a simple figure for visual inspection.

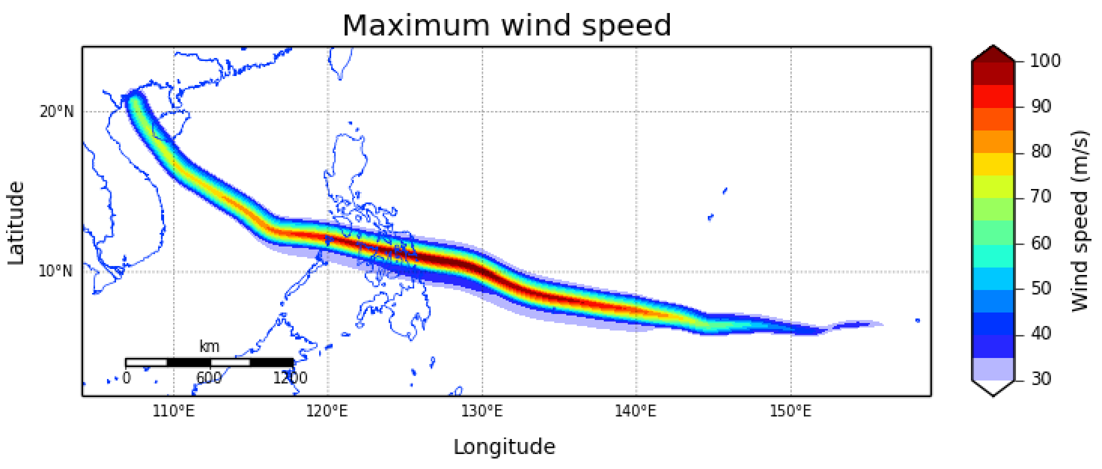

Fig. 9.1: Estimated maximum wind speed swath of Super Typhoon *Haiyan* (2013) across the Philippine archipelago. This simulation used the best track estimate from the Joint Typhoon Warning Center to establish the intensity and radius to maximum winds of the typhoon. No attempt is made to fit the radial profile to defined wind radii (e.g. radius of 46-, 50- or 34-knots).

Note The double labels on the secondary (right-hand) y-axis require Matplotlib version 1.3 or later.

9.5 Troubleshooting

Some common errors when running a scenario.

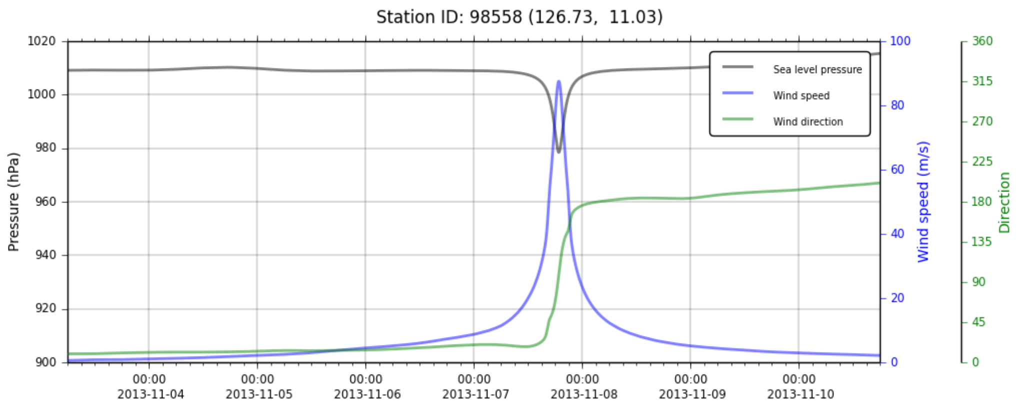

Fig. 9.2: Time series data for Super Typhoon *Haiyan* at Guiuan, Samar, Philippines.

9.5.1 `MemoryError`

In isolated cases, `tcevent.py` may fail and report a message that ends with `MemoryError`. This arises when the size of arrays in Python exceed 2GB (on 32-bit systems). Conditions that lead to this error are not clear. To resolve the problem, it is recommended to reduce the domain of the wind field by adding a `Region` section to the configuration file.:

```
[Region]
gridLimit = {'xMin':118., 'xMax':122., 'yMin':-23., 'yMax':-17.}
```

This will restrict calculation of the wind field to the defined domain. The `gridLimit` value is described in the *Region* section.

CHAPTER

TEN

CONTRIBUTING CODE

The preferred way to contribute to TCRM is to fork the main repository on GitHub:

1. Fork the project repository: click on the 'Fork' button near the top of the page. This creates a copy of the code under your account on the GitHub server.
2. Clone this copy to your local disk:

```
$ git clone git@github.com:YourLogin/tcrm.git
$ cd tcrm
```

3. Create a branch to hold your changes:

```
$ git checkout -b my-feature
```

   and start making changes. Never work in the `master` branch!

4. Work on this copy on your computer using Git to do the version control. When you're done editing, do:

```
$ git add modified_files
$ git commit
```

   to record your changes in Git, then push them to GitHub with:

```
$ git push -u origin my-feature
```

Finally, go to the web page of the your fork of the tcrm repo, and click 'Pull request' to send your changes to the maintainers for review. request. This will send an email to the committers.

(If any of the above seems like magic to you, then look up the Git documentation on the web.)

It is recommended to check that your contribution complies with the following rules before submitting a pull request:

- All public methods should have informative docstrings with sample usage presented as doctests when appropriate.
- When adding additional functionality, provide at least one example script in the `examples/` folder. Have a look at other examples for reference. Examples should demonstrate why the new functionality is useful in practice.
- At least one paragraph of narrative documentation with links to references in the literature (with PDF links when possible) and an example.

You can also check for common programming errors with the following tools:

- Code with good unittest coverage, check with:

```
$ pip install nose coverage
$ nosetests --with-coverage path/to/tests_for_package
```

- No pyflakes warnings, check with:

```
$ pip install pyflakes
$ pyflakes path/to/module.py
```

- No PEP8 warnings, check with:

```
$ pip install pep8
$ pep8 path/to/module.py
```

- AutoPEP8 can help you fix some of the easy redundant errors:

```
$ pip install autopep8
$ autopep8 path/to/pep8.py
```

10.1 Issues

A great way to start contributing to TCRM is to pick an item from the list of Issues in the issue tracker. (Well there are none there yet, but we will be putting some up soon!) Resolving these issues allow you to start contributing to the project without much prior knowledge. Your assistance in this area will be greatly appreciated by the more experienced developers as it helps free up their time to concentrate on other issues.

10.2 Documentation

We are in the process of creating sphinx based documentation for TCRM. Any help in setting this up will be gratefully accepted!

At present you will find the user_manual in the doc folder.

We are glad to accept any sort of documentation: function docstrings, reStructuredText documents (like this one), tutorials, etc. reStructuredText documents live in the source code repository under the docs/ directory.

When you are writing documentation, it is important to keep a good compromise between mathematical and algorithmic details, and give intuition to the reader on what the algorithm does. It is best to always start with a small paragraph with a hand-waving explanation of what the method does to the data and a figure (coming from an example) illustrating it.

CHAPTER

ELEVEN

UTILITIES PACKAGE

11.1 Module contents

11.2 Submodules

11.3 Utilities.Intersections module

11.3.1 `Intersections` – determine line intersections for polygons and lines

class **`Crossings`**

Determine if a a line intersects some other geometric feature (another line, a circle, a polygon).

`CircleLine`(*c*, *r*, *a1*, *a2*)

Determine the intersection points of a circle and a line segment.

Parameters

- **`c`** – `Point` object describing the centre of the circle.
- **`r`** – Eadius of the circle in map units.
- **`a1`** – `Point` object describing the start of the line segment.
- **`a2`** – `Point` object describing the end of the line segment.

Returns `Intersection` object with the `Intersection.status` updated. "Inside" means the line segment is wholly contained within the circle; "Outside" means the line segment is wholly outside the circle; "Intersection" means the line intersects the circle. In this final case, the `points` attribute of the `Intersection` object is populated with a `Point` object of the location of the intersection.

`CirclePolygon`(*c*, *r*, *points*)

Determine the intersection points of a circle and a polygon. Uses `Crossings.CircleLine()` on each line segment in the polygon to determine if the two features intersect.

Parameters

- **`c`** – `Point` representing the centre of the circle.
- **`r`** – Radius of the circle in map units.

Paran points Array of `Point` objects representing the polygon.

Returns `Intersection` object with updated status and points attributes.

`LineLine`(*a1*, *a2*, *b1*, *b2*)

Determine if two line segments intersect.

Parameters

- **`a1`** – Starting `Point` of line 1.

- **a2** – Ending `Point` of line 1.
- **b1** – Starting `Point` of line 2.
- **b2** – Ending `Point` of line 2.

Returns `Intersection` object with `status` set to `Intersection` if the lines intersect, `Coincident` if the lines overlap, `Parallel` if the lines are parallel or `No Intersection` if the lines do not intersect. If the lines intersect, then the `points` attribute is set to the location of the intersection.

`LinePolygon`(*a1*, *a2*, *points*)
Determine if a line intersects a polygon.

Parameters

- **a1** – Starting `Point` of the line.
- **a2** – Ending `Point` of the line.
- **points** – Collection of `Point` objects that represent the vertices of a polygon.

`lerp`(*a1*, *a2*, *u*)
Linear interpolation between two points:

Parameters

- **a1** – First `Point`.
- **a2** – Second `Point`.
- **u** – Fractional distance along the line between the two points.

Returns Coordinates of the interpolated point as a tuple.

class `Intersection`(*state=None*)
An Intersection object.

Parameters **state** (*str*) – Initial state of the `Intersection` (default `None`).

Parameters: None Members: status (string) and points (list) Methods: None Internal Methods: None

class `Point`(*x*, *y*)
Class representation for a Point Contains x and y members Created by Geoff Xu, 2006

`getX`()

`getY`()

`convert2vertex`(*a1*, *a2*)
Converts 2 1D arrays into a list of `Points`.

Parameters

- **a1** – ordinate
- **a2** – abscissa

Returns List of `Point` objects.

`inLand`(*P*, *V*)
Test to see if a point is within the list of vertices.

Parameters

- **P** – `Point` object.
- **V** – List of `Point` objects that represent vertices of the shape to be tested. Must be a closed set.

Returns `True` if the point lies inside the vertices, `False` otherwise.

Return type boolean

11.4 Utilities.colours module

11.4.1 `colours` - add custom colour scales

11.5 Utilities.columns module

11.5.1 `columns` – load delimited data files

`colReadCSV`(*configFile*, *dataFile*, *source*)

Loads a csv file containing 'column' data into a record (numpy) array with columns labelled by 'fields'. There must be a section in the `configFile` named `source` that sets out the format of the data file.

Parameters

- **`configFile`** (*str*) – Path to a configuration file that holds details of the input data.
- **`dataFile`** (*str*) – Path to the input file to load.
- **`source`** (*str*) – Name of the source format. There must be a corresponding section in the `configFile`.

Returns A `numpy.ndarray` that contains the input data.

11.6 Utilities.config module

11.6.1 `config` – reading configuration files

`ConfigParser`(**args*, ***kwargs*)

`cnfGetIniValue`(*configFile*, *section*, *option*, *default=None*)

Helper function to interface with code that uses the old config parser.

Parameters

- **`configFile`** (*str*) – path to the configuration file to read.
- **`section`** (*str*) – Section name to read.
- **`option`** (*str*) – Option name to read.
- **`default`** – Optional default value to use if the section/option pair is not present in the file.

Returns Value recorded in the section/option of the config file, if present; the default value otherwise (if defined).

Raises NoSectionError, NoOptionError if the section/option is not defined, and no default value given.

`formatList`(*lst*)

Convert a list into a comma-joined string.

Parameters **`lst`** (*list*) – Input list to join.

Returns A string comprised of the list elements joined by commas.

Return type str

`parseBool`(*txt*)

Parser for boolean options

Parameters **`txt`** (*str*) – String from config file to parse.

Returns `True` if the string is 'True', `False` otherwise.

Return type boolean

`parseList` (*txt*)

Parse a comma-separated line into a list.

Parameters **`txt`** (*str*) – String from config file to parse.

Returns List, based on the input string.

Return type list

`singleton` (*cls*)

11.7 Utilities.confjson module

class **`StrictConfigParser`** (*defaults=None, dict_type=<class 'collections.OrderedDict'>, allow_no_value=False*)

Bases: `ConfigParser.ConfigParser`

`dget` (*section, option, default=None, type=<type 'str'>*)

`flatten` (*config*)

11.8 Utilities.convolve module

11.8.1 `convolve` – convolve two 2-d fields

`convolve` (*im, direction, m='terrain', res=25.0, height=5.0*)

Smooth a 2D array im by convolving with a kernel of size n.

Parameters

- **`im`** (`numpy.ndarray`) – 2-d array of values to be smoothed.
- **`dir`** (*str*) – One of 'N','NE','E','SE','S','SW','W','NW' to define the direction of the site multiplier to evaluate.
- **`m`** (*str*) – Model type = either "terrain" or "shield".
- **`res`** (*float*) – Resolution of the input grid dataset.
- **`height`** (*float*) – nominal height of the buildings to be used in evaluating the shielding multiplier.

Returns 2-d array of convolved data (convolved with the appropriate kernel) The output array is the same size as the input array, with boundary values set to be filled to a value of 1.0

Return type `numpy.ndarray`

`getKernel` (*d, m, r=25.0, h=5.0*)

Define an appropriate kernel for smoothing the data

Parameters

- **`d`** (*str*) – Wind direction - one of ['N','NE','E','SE','S','SW','W','NW']
- **`m`** (*str*) – Multiplier type - either 'terrain' or 'shield'
- **`r`** (*float*) – Resolution of the dataset (in metres) (default is 25)
- **`h`** (*float*) – Nominal height of buildings, for use in evaluating the shielding multiplier (default is 5)

Returns kernel to apply to the raw multiplier values to generate directional multiplier values.

Return type 2-d `numpy.ndarray`

11.9 Utilities.datasets module

11.9.1 `datasets` – manage downloaded datasets

class **`DataSet`**(*name*, *url*, *path*, *filename=None*)

Bases: `object`

Download a dataset from a url, store in a specified path with a given filename.

Parameters

- **`name`** (*str*) – Name of the dataset to be downloaded.
- **`url`** (*str*) – URL of the dataset.
- **`path`** (*str*) – path name for the storage location of the dataset
- **`filename`** (*str or None*) – name of the file to be saved (can be different from the name of the dataset).

`download`(*callback=None*)

Execute the download of the dataset. If the dataset has already been downloaded (the *`isDownloaded`* will be set to True), then don't try to download.

Parameters **`callback`** (*function*) – Callback function (for reporting status to STDOUT).

Raises IOError Unable to download the dataset.

`isDownloaded`()

Determine if a file has already been downloaded

Returns *True* if the file exists, *False* otherwise.

`checkAndDownload`(*callback=None*)

Check the `DATASETS` list and download each, if it has not previously been downoladed.

Parameters **`callback`** (*function*) – Callback function (for reporting status to STDOUT).

`loadDatasets`(*configFile*)

Load the details of the datasets to be downloaded from the configuration settings. This updates the `DATASETS` list.

11.10 Utilities.files module

exception **`WindowsError`**

Bases: `exceptions.OSError`

`flConfigFile`(*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 = flConfigFile('.ini') Calling flConfigFile from /foo/bar/baz.py should return /foo/bar/baz.ini

`flGetStat` (*filename*, *CHUNK=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`** (*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 = flGetStat(filename)

`flLoadFile` (*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.

`flLogFatalError` (*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.

`flModDate` (*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 = flModDate('C:/foo/bar.csv' , dateformat='%Y-%m-%dT%H:%M:%S')

`flModuleName` (*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 `flModuleName`)

Returns Module name.

Return type str

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

`flModulePath` (*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 `flModulePath`)

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

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

`flProgramVersion` (*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)

`flSaveFile` (*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.

`flSize` (*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 = flSize('C:/foo/bar.csv')

`flStartLog` (*logFile*, *logLevel*, *verbose=False*, *datestamp=False*, *newlog=True*)

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

Parameters

- **`logFile`** (*str*) – Full path to log file.
- **`logLevel`** (*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: flStartLog('/home/user/log/app.log', 'INFO', verbose=True)

11.11 Utilities.grid module

11.11.1 `grid` – read, write and sample ascii grid files

Provide functions to read, write and sample ascii grid files. The ascii files are assumend to have and ArcGIS GIRD format:

```
ncols         nx
nrows         ny
xllcorner     xll
yllcorner     yll
cellsize      dx
NODATA_value -9999
data data data ...
 :    :    :
 :    :    :
```

class **`SampleGrid`**(*filename*)

Sample data from a gridded data file. The class is instantiated with a gridded data file (either an ascii file or a netcdf file with single 2-d variable), and the `SampleGrid.sampleGrid()` method returns the value of the grid point closest to the given longitude and latitude.

Parameters **`filename`** (*str*) – Path to a file containing gridded data.

Example:

```
    >>> grid = SampleGrid( '/foo/bar/grid.nc' )
    >>> value = grid.sampleGrid( 100., -25. )
```

`sampleGrid`(*lon*, *lat*)

Sample a value from the grid at the given cLon, cLat At this time, does not interplolate from the input grid to the given location.

Parameters

- **`lon`** (*float*) – Longitude of the point to sample.
- **`lat`** (*float*) – Latitude of the point to sample.

Returns Value of the nearest grid point to the given lon/lat point.

`grdRead`(*filename*, *delimiter=None*)

Read formatted data from an ascii grid format file. Returns the longitude and latitude of the grid and the data values

Parameters

- **`filename`** (*str*) – Path to an ascii grid format or netcdf file.
- **`delimiter`** – Delimiter for the ascii format file (optional).

Returns longitude, latitude, data

Return type `numpy.ndarray`

Usage: longitude, latitude, data = grdRead(filename, [delimiter])

`grdReadFromNetcdf`(*filename*)

Read formatted data from a netcdf file. Returns the longitude and latitude of the grid and the data values. Assumes that there is only one (non-coordinate) variable in the file, which is only 2 dimensional.

Parameters **`filename`** (*str*) – Path to a netcdf file to read.

Returns longitude, latitude and grid data.

Usage: longitude, latitude, data = grdReadFromNetcdf(filename)

grdSave (*filename, data, lon, lat, delta, delimiter=' ', nodata=-9999, fmt='%.10e', coords='latlon'*)

Save formatted data to an ascii grid format file. The files have 6 header lines describing the data, followed by the data in a gridded format.

Parameters

- **filename** (*str*) – Path to the file to be written.
- **data** – 2-d array of data values to store.
- **lon** – Array of longitudes corresponding to data points.
- **lat** – Array of latitudes corresponding to data points.
- **delta** (*float*) – Spacing between grid points.
- **delimiter** (*str*) – Delimiter to put between data points (default ' ').
- **nodata** (*float*) – Value to indicate missing values (default -9999).
- **fmt** (*str*) – String format statement.
- **coords** (*str*) – Optionally store the data in UTM coordinates. Default is to store the data in geographic coordinates (`coords='latlon'`). If `coords='UTM'`, then the latitude & longitudes are converted to the local UTM coordinate system.

Raises ValueError If the `filename` is not a string of file handle.

Usage:

```
>>> grdSave(filename, data, lon, lat, delta, delimiter=' ',
            nodata=-9999, fmt='%.10e, coords='latlon')
```

11.12 Utilities.interp3d module

11.12.1 `interp3d` – interpolate to a set of points in 3-dimensional space

Use `scipy.ndimage.interpolation.map-coordinates()` to interpolate data in three dimensions.

interp3d (*input_array, coords, scale=[360.0, 180.0, 365.0], offset=[0.0, -90.0, 0.0], prefilter=True*)

Wrapper to `scipy.ndimage.interpolation.map_coordinates()`, which converts coordinates of points to indices that correspond to the array. We assume that one is working with lon, lat, day data (i.e. initially designed to work with daily long term mean sea level pressure)

Parameters

- **input_array** – A 3-d array of data at regular intervals, representing the data to be evaluated.
- **coords** – A 3-by-n array of coordinates at which the data in `input_array` will be interpolated to.
- **scale** (*list*) – A (list of) scale factor(s) that reduces the `coords` values to the range of indices in `input_array`.
- **offset** (*list*) – A (list of) offset factor(s) that is subtracted from the `coords` values before adjusting the `scale` (above).
- **prefilter** (*boolean*) – If `True` (default), then apply a spline filter before interpolation (necessary for spline interpolation of order > 1). If `False`, it is assumed that the input is already filtered. Default is `True`.

Returns 1-d array of values corresponding to the interpolated values at the points given in `coords`.

Example:

```
>>> vals = interp3d(data, coords, scale=[360., 180., 365.],
...                 offset=[0., -90., 0.])
```

11.13 Utilities.interpTrack module

11.13.1 `interpTrack` – interpolate TC track to a shorter time interval

`interpolateTrack`(*configFile*, *trackFile*, *source*, *delta=0.1*, *interpolation_type=None*)

Interpolate the data in a track file to the time interval delta hours.

Parameters

- **`configFile`** (*str*) – Configuration file that contains information on the source format of the track file.
- **`trackFile`** (*str*) – Path to csv format track file.
- **`source`** (*str*) – Name of the data source. There must be a corresponding section in the configuration file that contains the description of the data.
- **`delta`** (*float*) – Time interval in hours to interpolate to. Default is 0.1 hours
- **`interpolation_type`** (*str*) – Optionally use Akima or linear interpolation for the track positions. Default is linear 1-dimensional spline interpolation.

Returns 10 arrays (id, time, date, lon, lat, bearing, forward speed, central pressure, environmental pressure and radius to maximum wind) that describe the track at `delta` hours intervals.

11.14 Utilities.lat_long_UTM_conversion module

11.14.1 `lat_long_UTM_conversion` – convert between geographic & UTM coordinates

Usage:

```
>>> zone, easting, northing = LLToUTM(latitude, longitude,
...                                   ReferenceEllipsoid=23)
>>> latitude, longitude = UTMtoLL(northing, easting, zone,
...                               isSouthernHemisphere=True,
...                               ReferenceEllipsoid=23)
```

See http://www.pygps.org

`LLtoUTM`(*Lat*, *Long*, *ReferenceEllipsoid=23*, *ZoneNumber=None*)

Converts lat/long to UTM coords. Equations from USGS Bulletin 1532

Lat and Long are in decimal degrees Written by Chuck Gantz- chuck.gantz@globalstar.com

Parameters

- **`Lat`** (*float*) – Latitude. North latitudes are positive, south latitudes are negative.
- **`Lon`** (*float*) – Longitude. East longitudes are positive, west longitudes are negative.
- **`ReferenceEllipsoid`** (*int*) – Reference ellipsoid for the coordinate system (default=23 – WGS84).
- **`ZoneNumber`** (*str*) – UTM Zone for coordinates.

Returns Tuple of (ZoneNumber, UTMEasting, UTMNorthing)

UTMtoLL(*northing*, *easting*, *zone*, *isSouthernHemisphere=True*, *ReferenceEllipsoid=23*)

Converts UTM coords to lat/long. Equations from USGS Bulletin 1532 East Longitudes are positive, West longitudes are negative. North latitudes are positive, South latitudes are negative Lat and Long are in decimal degrees. Written by Chuck Gantz- chuck.gantz@globalstar.com Converted to Python by Russ Nelson <nelson@crynwr.com>

FIXME: This is set up to work for the Southern Hemisphere.

Using http://www.ga.gov.au/geodesy/datums/redfearn_geo_to_grid.jsp

```
Site Name:    GDA-MGA: (UTM with GRS80 ellipsoid)
Zone:   36
Easting:  511669.521  Northing: 19328195.112
Latitude:   84  0 ' 0.00000 ''  Longitude: 34  0 ' 0.00000 ''
Grid Convergence:  0  -59 ' 40.28 ''  Point Scale: 0.99960166
____________
Site Name:    GDA-MGA: (UTM with GRS80 ellipsoid)
Zone:   36
Easting:  519384.803  Northing: 1118247.585
Latitude:   -80  0 ' 0.00000 ''  Longitude: 34  0 ' 0.00000 ''
Grid Convergence:  0  59 ' 5.32 ''  Point Scale: 0.99960459
____________
Site Name:    GDA-MGA: (UTM with GRS80 ellipsoid)
Zone:   36
Easting:  611263.812  Northing: 10110547.106
Latitude:   1  0 ' 0.00000 ''  Longitude: 34  0 ' 0.00000 ''
Grid Convergence:  0  -1 ' 2.84 ''  Point Scale: 0.99975325
____________
Site Name:    GDA-MGA: (UTM with GRS80 ellipsoid)
Zone:   36
Easting:  611263.812  Northing: 9889452.894
Latitude:   -1  0 ' 0.00000 ''  Longitude: 34  0 ' 0.00000 ''
Grid Convergence:  0  1 ' 2.84 ''  Point Scale: 0.99975325
```

So this uses a false northing of 10000000 in the both hemispheres. ArcGIS used a false northing of 0 in the northern hem though. Therefore it is difficult to actually know what hemisphere you are in.

Parameters

- **northing** (*float*) – Northing coordinate in UTM coordinates.
- **easting** (*float*) – Easting coordinate in UTM coordinates
- **zone** (*int*) – UTM zone number.
- **isSouthernHemisphere** (*boolean*) – `True` if the location is in the southern hemisphere (default).
- **ReferenceEllipsoid** (*int*) – Reference ellipsoid for the coordinate system (default=23 – WGS84).

Returns Tuple pair of latitude, longitude for the point.

11.15 Utilities.lmomentFit module

11.15.1 `lmomentFit` – fit GEV functions

Two functions {pelgev, samlmu} from the LMOMENTS Fortran package ported to Python to fit a Generalised Extreme Value Distribution function. Original code developed by: J. R. M. HOSKING, IBM RESEARCH DIVISION, T. J. WATSON RESEARCH CENTER, YORKTOWN HEIGHTS, NEW YORK 10598, U.S.A.

Note: Permission to use, copy, modify and distribute this software for any purpose and without fee is hereby granted, provided that this copyright and permission notice appear on all copies of the software. The name of

---

pelgev (*XMOM*)

Parameter estimation via L-moments for the Generalised Extreme Value Distribution. For -0.8 <= TAU3 < 1., K is approximated by rational functions as in Donaldson (1996, Commun. Statist. Simul. Comput.). If TAU3 is outside this range, Newton-Raphson iteration is used.

Parameters **XMOM** (List or `numpy.ndarray`) – Array of length 3, containing the L-moments Lambda-1, Lambda-2 and TAU3.

Returns Location, scale and shape parameters of the GEV distribution.

Return type `numpy.ndarray`

pelgpa (*XMOM*)

samlmu (*X*, *NMOM*)

Sample L-moments for a data array.

Parameters

- **X** (`numpy.ndarray`) – Array of length N, containing the data in ascending order.
- **NMOM** – Number of L-moments to be found (maximum 100).

Returns The sample L-moments.

Return type `numpy.ndarray`

samlmu3 (*X*)

Functional equivalent to lmoments.samlmu(X, 3). Vectorised for speed but still about half speed of original fortran package.

Parameters **X** – Array of length N, containing the data in ascending order.

Returns First 3 L-moments.

Return type `numpy.ndarray`

11.16 Utilities.loadData module

11.16.1 `loadData` - load TC track data from formatted files

Load a formatted csv file that contains tropical cyclone track information and return a collection of `Track` objects.

The format of the files is defined in a named section of the input configuration file. See the *Source Formats* section of the User Guide for details on specifying the format.

TODO: Modify the source to be a dict containing the kwargs to `numpy.genfromtxt` to make reading additional formats possible (e.g. include converters for some data fields, as in B-deck format track files.

date2ymdh (*dates*, *datefmt='%Y-%m-%d %H:%M:%S'*)

Convert date strings to arrays for date components.

Parameters

- **dates** (`numpy.ndarray`) – Array of str objects that describe a date.
- **datefmt** (*str*) – Format string of the date values in *dates*, default='%Y-%m-%d %H:%M:%S'

Returns `numpy.ndarray` arrays containing the year, month, day, hour and minute of the dates contained in the input array *dates*.

`filterPressure` (*pressure*, *inputPressureUnits='hPa'*, *missingValue=2147483647*)
Filter pressure values to remove any non-physical values.

Parameters

- **`pressure`** (`numpy.ndarray`) – input pressure values to check.
- **`inputPressureUnits`** (*str*) – The units of the pressure values. Can be one of `hPa`, `Pa`, `kPa`, `Pascals` or `mmHg`.
- **`missingValue`** (int or float (default `sys.maxint`)) – replace all null values in the input data with this value.

Returns `numpy.ndarray` with only valid pressure values.

`getInitialPositions` (*data*)
Get the array indices corresponding to the initial position of TCs in the input dataset. This is done through examining the data for a number of specific fields to see when they change, or if the data has a field that indicates such an instance.

Parameters **`data`** (*dict*) – `dict` of arrays that contains the data loaded from the input file

Returns `numpy.ndarray` of indices that can be used to slice the observations and return those corresponding to an initial TC position.

Note: Using only the 'num' field will result in different results than using 'num' and 'season'. From experience, the 'num' field in best-track datasets refers to the sequential number of the storm for that season - i.e. it starts at 1 for each season and increments for each new storm. The use of 'num' only should be reserved for those situations where the dataset is known to have unique numbers for each storm (e.g in simulated data).

`getMaxWind` (*track*, *missingValue=2147483647*)
Determine the maximum wind speed of a `Track` instance

Parameters

- **`track`** – A `Track` instance
- **`missingValue`** (int or float (default `sys.maxint`)) – replace all null values in the input data with this value.

Returns `Track.trackMaxWind` attribute updated with calculated wind speed updated.

`getMinPressure` (*track*, *missingValue=2147483647*)
Determine the minimum pressure of a `Track` instance

Parameters

- **`track`** – A `Track` instance
- **`missingValue`** – Replace missing values with this value (default `sys.maxint`).

Returns `Track.trackMinPressure` attribute updated

`getSpeedBearing` (*index*, *lon*, *lat*, *deltatime*, *ieast=1*, *missingValue=2147483647*)
Calculate the speed and bearing of a TC.

Parameters

- **`index`** (`numpy.ndarray`) – Array of 0/1 indicating start of new TC (1)
- **`lon`** (`numpy.ndarray`) – Longitudes of TC positions.
- **`lat`** (`numpy.ndarray`) – Latitudes of TC positions.
- **`deltatime`** (`numpy.ndarray`) – Time difference (hours) between consecutive TC observations.

- **ieast** (*int*) – Indicate which direction has positive longitude. 1 = positive longitude eastwards -1 = positive longiture westwards.
- **missingValue** (int or float, default = *sys.maxint*) – Replace questionable values with *missingValue*.

Returns speed and bearing : `numpy.ndarray`

Example:

```
>>> speed, bearing = getSpeedBearing(index, lon, lat, deltatime)
```

getTime (*year*, *month*, *day*, *hour*, *minute*)
Calculate the number of days since 0001-01-01 00:00:00 UTC + 1

Parameters

- **year** (`numpy.ndarray` or list) – Year values.
- **month** (`numpy.ndarray` or list) – Month values.
- **day** (`numpy.ndarray` or list) – Day values.
- **hour** (`numpy.ndarray` or list) – Hour values.
- **minute** (`numpy.ndarray` or list) – Minute values.

Returns `numpy.ndarray` of days since 0001-01-01 00:00:00 UTC + 1

getTimeDelta (*year*, *month*, *day*, *hour*, *minute*)
Calculate the time difference between consecutive observations.

Parameters

- **year** – `numpy.ndarray` of the year of all observations.
- **month** – As for year, but for the month of observation.
- **day** – As for year, but for the day of observation.
- **hour** – As for year, but for the hour of the observation.
- **minutes** – As for year, but for the hour of the observation.
- **seconds** – As for year, but for the hour of the observation.

Returns `numpy.ndarray` of time difference between observations in hours.

getTimeElapsed (*indicator*, *year*, *month*, *day*, *hour*, *minute*)
Calculate the age in hours of each event.

Parameters

- **indicator** (`numpy.ndarray`) – Array (values of 1 or 0) indicating the beginning of a new TC in the input dataset.
- **year** – `numpy.ndarray` of the year of all observations.
- **month** – As for year, but for the month of observation.
- **day** – As for year, but for the day of observation.
- **hour** – As for year, but for the hour of the observation.
- **minutes** – As for year, but for the hour of the observation.

Returns `numpy.ndarray` of time since the initial observation for each TC (in hours).

julianDays (*year*, *month*, *day*, *hour*, *minute*)
Calculate the julian day (day of year) based on the known date/time information.

Parameters

- **year** – `numpy.ndarray` of the year of all observations.

- **month** – As for year, but for the month of observation.
- **day** – As for year, but for the day of observation.
- **hour** – As for year, but for the hour of the observation.
- **minute** – As for year, but for the hour of the observation.

Returns `numpy.ndarray` of julian day values for each observation.

loadTrackFile (*configFile*, *trackFile*, *source*, *missingValue=0*, *calculateWindSpeed=True*)
Load TC track data from the given input file, from a specified source. The configFile is a configuration file that contains a section called 'source' that describes the data. This returns a collection of `Track` objects that contains the details of the TC tracks in the input file.

Parameters

- **configFile** (*str*) – Configuration file with a section `source`.
- **trackFile** (*str*) – Path to a csv-formatted file containing TC data.
- **missingValue** – Replace all null values in the input data with this value (default=0).
- **calculateWindSpeed** (*boolean*) – Calculate maximum wind speed using a pressure-wind relation described in `maxWindSpeed()`

Pararm str source Name of the source format of the TC data. There *must* be a section in `configFile` matching this string, containing the details of the format of the data.

Returns A collection of `Track` objects. If any of the variables are not present in the input dataset, they are (where possible) calculated (date/time/windspeed), sampled from default datasets (e.g. environmental pressure) or set to the missing value.

Example:

```
>>> tracks = loadTrackFile('tcrm.ini', 'IBTRaCS.csv', 'IBTrACS' )
```

ltmPressure (*jdays*, *time*, *lon*, *lat*, *ncfile*)
Extract pressure value from a daily long-term mean SLP dataset at the given day of year and lon,lat position To use this function (and hence some form of daily LTM SLP data) requires knowledge of the day of year.

Parameters

- **jdays** (`numpy.ndarray`) – Julian day (day of year) values.
- **time** (`numpy.ndarray`) – Time of day for each observation (fraction of a day).
- **lon** (`numpy.ndarray`) – Longitude of TC position.
- **lat** (`numpy.ndarray`) – Latitude of TC position.
- **ncfile** (*str*) – Path to netCDF file containing daily long-term mean sea level pressure data.

Returns `numpy.ndarray` of long-term mean sea level pressure values at the day of year and positions given.

maxWindSpeed (*index*, *deltatime*, *lon*, *lat*, *pressure*, *penv*, *gustfactor=0.9524*)
Calculate the 10-minute-mean maximum wind speed from the central pressure deficit, using the method described in Holland et al. (2010).

Parameters

- **indicator** (`numpy.ndarray`) – Array (values of 1 or 0) indicating the beginning of a new TC in the input dataset.
- **deltatime** (`numpy.ndarray`) – Time difference (in hours) between each point in the record.
- **lon** (`numpy.ndarray`) – Longitudes of TC postions.
- **lat** (`numpy.ndarray`) – Latitudes of TC positions.

- **`pressure`** (`numpy.ndarray`) – Central pressure estimate of TCs (hPa).
- **`penv`** (`numpy.ndarray`) – Environmental pressure estimates for each TC postion (hPa).
- **`gf`** (*float*) – Gust factor - default value represents converting from a 1-minute sustained wind speed to a 10-minute mean wind speed. Based on Harper et al. 2010, WMO-TD1555.

Returns `numpy.ndarray` of estimated wind speed based on central pressure deficit.

Example:

```
>>> v = maxWindSpeed(indicator, dt, lon, lat, pressure, penv)
```

`parseAge` (*data*, *indicator*)
Parse the TC age information to get a proxy date record. Assumes every TC starts at 2000-01-01 00:00 and calculates year, month, day, hour and minute values based on the age field.

Assumes the age is given in hours since the initial timestep.

Parameters

- **`data`** (*dict*) – `dict` of arrays that contains the data loaded from the input file.
- **`indicator`** (`numpy.ndarray`) – Array (values of 1 or 0) indicating the beginning of a new TC in the input dataset.

Returns `numpy.ndarray`'s of year, month, day, hour, minute and :class:`datetime.datetime` objects.

`parseDates` (*data*, *indicator*, *datefmt='%Y-%m-%d %H:%M:%S'*)
Parse the date/time information to extract year, month, day, hour and minute details for the input dataset.

Parameters

- **`data`** (*dict*) – `dict` of arrays that contains the data loaded from the input file.
- **`indicator`** (`numpy.ndarray`) – Array (values of 1 or 0) indicating the beginning of a new TC in the input dataset.
- **`datefmt`** (*str*) – Format string of the date values in *dates*, default='%Y-%m-%d %H:%M:%S'

Returns `numpy.ndarray`'s of year, month, day, hour, minute and :class:`datetime.datetime` objects.

11.17 Utilities.maputils module

11.17.1 `maputils` – mapping functions

Contains mapping functions that operate on arrays. Supercedes some of the functions lying around in some unusual places (like DataProcess).

`bear2LatLon` (*bearing*, *distance*, *oLon*, *oLat*)
Calculate the longitude and latitude of a new point from an origin point given a distance and bearing.

Parameters

- **`bearing`** – Direction to new position (degrees, +ve clockwise from north).
- **`distance`** – Distance to new position (km).
- **`oLon`** – Initial longitude.
- **`oLat`** – Initial latitude.

Returns new longitude and latitude (in degrees)

bearing2theta(*bearing*)

Converts bearing in azimuth coordinate system into theta in cartesian coordinate system. Assumes -2*pi <= bearing <= 2*pi

Parameters **bearing** – Bearing to convert (in radians) (+ve clockwise from north).

Returns Angle in cartesian coordinate system (+ve anticlockwise from east).

coriolis(*lat*)

Calculate the Coriolis factor (f) for a given latitude (degrees). If a list is passed, return a list, else return a single value.

Parameters **lat** (*Array-like.*) – Latitude (degrees).

Returns Coriolis factor.

dist2GC(*cLon1*, *cLat1*, *cLon2*, *cLat2*, *lonArray*, *latArray*, *units='km'*)

Calculate the distance between an array of points and the great circle joining two (other) points. All input values are in degrees. By default returns distance in km, other units specified by the 'units' kwarg.

Based on a cross-track error formulation from: http://williams.best.vwh.net/avform.htm#XTE

Parameters

- **cLon1** (*float*) – Longitude of first point.
- **cLat1** (*float*) – Latitude of first point.
- **cLon2** (*float*) – Longitude of second point.
- **cLat2** (*float*) – Latitude of second point.
- **lonArray** – numpy.ndarray of longitudes for which the distance to the line joining the two points will be calculated.
- **latArray** – numpy.ndarray of latitudes for which the distance to the line joining the two points will be calculated.

Returns 2-d array of distances between the array points and the line joining two points.

Return type numpy.ndarray

distGC(*lat*, *lon*)

Distance based on the great circle navigation between pairs of points.

Parameters

- **lat** – A pair of latitude values for the two points.
- **lon** – A pair of longitude values for the two points.

Returns Distance (in kilometres) between the two points, based on great circle navigation.

Example:

```
>>> dist = distGC([-20, -40],[120,190])
6914.42
```

find_index(*array*, *value*)

Find the index of 'array' with a value closest to 'value'

Parameters

- **array** (numpy.ndarray or *list.*) – array of data values.
- **value** – a value to search *array* for (or find the index of the nearest value to *value*).
- **idx** (*int*) – index of *array* that most closely matches *value*

Raises ValueError if *value* is a numpy.ndarray or a list

Example:

```
>>> find_index(np.arange(0., 100., 0.5), 15.25)
30
```

find_nearest (*array*, *value*)

Find the closest value in 'array' to 'value'

Parameters

- **array** (numpy.ndarray) – array of data values.
- **value** (*int or float*) – a value to search the array for (or find the index of the nearest value to 'value'

Returns array[idx], where idx is the index of array that corresponds to the value closest to 'value'.

Raises

- **ValueError** – If *value* is a numpy.ndarray or a list.
- **IndexError** – If the *value* cannot be found in the *array*

Example:

```
>>> n = find_nearest( np.arange(0,100.,0.5), 15.25 )
15.0
```

gridLatLonBear (*cLon*, *cLat*, *lonArray*, *latArray*)

Generate a grid containing the bearing of the points defined by (lonArray,latArray) from the point defined by (cLon,cLat). (lonArray,latArray) and (cLon,cLat) are in degrees. Returns bearing in radians.

Parameters

- **cLon** (*float*) – Longitude of the point to measure the distance from.
- **cLat** (*float*) – Latitude of the point to measure the distance from.
- **lonArray** – 1-d array of longitude values that will define the grid over which distances will be calculated.
- **latArray** – 1-d array of latitude values that will define the grid over which distances will be calculated.

Returns 2-d array containing the bearing (direction) of the points defined in `lonArray` and `latArray` from the point (`cLon`, `cLat`)

Example:

```
>>> from maputils import gridLatLonBear
>>> import numpy as np
>>> lonArray = np.arange(90.,100.,0.1)
>>> latArray = np.arange(-20.,-10.,0.1)
>>> gridLatLonBear( 105., -15., lonArray, latArray)
array([[-1.94475949, -1.94659552, -1.94845671, ..., -2.36416927,
        -2.37344337, -2.38290081],
       [-1.93835542, -1.94015859, -1.94198663, ..., -2.35390045,
        -2.36317282, -2.37263233],
       [-1.93192776, -1.93369762, -1.93549204, ..., -2.34343069,
        -2.35269718, -2.36215458],
        ...,
       [-1.29066433, -1.28850464, -1.28632113, ..., -0.84374983,
        -0.83405688, -0.82416555],
       [-1.28446304, -1.28227062, -1.28005406, ..., -0.83332654,
        -0.82361918, -0.813717  ],
       [-1.27828819, -1.27606348, -1.27381433, ..., -0.82310335,
        -0.81338586, -0.80347714]])
```

gridLatLonDist (*cLon*, *cLat*, *lonArray*, *latArray*, *units=None*)

Generate a grid containing the spherical earth distance of the points defined by (lonarray, latarray) from the point defined by (clon, clat). (lonarray,latarray) and (clon,clat) are in degrees. Returns distances in km by default, other units specified by the 'units' kwarg.

Based on m_lldist.m by Rich Pawlowicz (rich@ocgy.ubc.ca) Modified by Craig Arthur 2006-11-13

Parameters

- **cLon** (*float*) – Longitude of the point to measure the distance from.
- **cLat** (*float*) – Latitude of the point to measure the distance from.
- **lonArray** – 1-d array of longitude values that will define the grid over which distances will be calculated.
- **latArray** – 1-d array of latitude values that will define the grid over which distances will be calculated.
- **units** (*str*) – Units of distance to be returned (default is kilometre)

Returns 2-d array containing the distance of the points defined in `lonArray` and `latArray` from the point (`cLon`, `cLat`).

Example:

```
>>> lonArray = np.arange(90.,100.,0.1)
>>> latArray = np.arange(-20.,-10.,0.1)
>>> dist = gridLatLonDist( 105., -15., lonArray, latArray, 'km')
```

latLon2Azi (*lat*, *lon*, *ieast=1*, *azimuth=0*, *wantdeg=True*)

Returns the bearing and distance (in km) between consecutive members of the array pair (lat,lon).

Parameters

- **lat** – Latitudes of positions.
- **lon** – Longitudes of positions.
- **ieast** (*int*) – 1 for longitudes increasing towards East, -1 for longitudes increasing towards West (default 1).
- **azimuth** (*float*) – Local coordinate system constructed with origin at latr,lonr, X axis ('North') in direction of azimuth, and Y axis such that X x Y = Z(down) when going from (lat,lon) to (x,y) (default 0).
- **wantdeg** (*boolean*) – If `True` return bearings as degrees, not radians.

Returns azimuth (+ve clockwise from north) and distance (in km).

latLon2XY (*xr*, *yr*, *lat*, *lon*, *ieast=1*, *azimuth=0*)

Calculate the cartesian distance between consecutive lat,lon points. Will bomb at North and South Poles. Assumes geographical coordinates and azimuth in decimal degrees, local Cartesian coordinates in km.

Parameters

- **xr** – Reference longitude, normally 0.
- **yr** – Reference latitude, normally 0.
- **lat** – Array of latitudes.
- **lon** – Array of longitudes.
- **ieast** (*int*) – 1 if longitude increases toward the East (normal case), -1 if longitude increases toward the West.
- **azimuth** (*int*) – local coordinate system constructed with origin at latr,lonr, X axis ('North') in direction of azimuth, and Y axis such that X x Y = Z(down) when going from (lat,lon) to (x,y) scalar or array.

Returns Array of northward and eastward distances between consecutive points. use `xy2r()` to convert to a distance between consecutive points.

`makeGrid`(*cLon*, *cLat*, *margin=2*, *resolution=0.01*, *minLon=None*, *maxLon=None*, *minLat=None*, *maxLat=None*)

Generate a grid of the distance and angle of a grid of points surrounding a storm centre given the location of the storm. The grid margin and grid size can be set in configuration files. xMargin, yMargin and gridSize are in degrees.

Parameters

- **`cLon`** (*float*) – Reference longitude.
- **`cLat`** (*float*) – Reference latitude.
- **`margin`** (*float*) – Distance (in degrees) around the centre to fit the grid.
- **`resolution`** (*float*) – Resolution of the grid (in degrees).
- **`minLon`** (*float*) – Minimum longitude of points to include in the grid.
- **`maxLon`** (*float*) – Maximum longitude of points to include in the grid.
- **`minLat`** (*float*) – Minimum latitude of points to include in the grid.
- **`maxLat`** (*float*) – Maximum latitude of points to include in the grid.

Returns 2 2-d arrays containing the distance (km) and bearing (azimuthal) of all points in a grid from the `cLon`, `cLat`.

`makeGridDomain`(*cLon*, *cLat*, *minLon*, *maxLon*, *minLat*, *maxLat*, *margin=2*, *resolution=0.01*)

Generate a grid of the distance and angle of a grid of points across a complete model domain, given the location of the storm.

Parameters

- **`cLon`** (*float*) – Reference longitude.
- **`cLat`** (*float*) – Reference latitude.
- **`minLon`** (*float*) – Minimum longitude of points to include in the grid.
- **`maxLon`** (*float*) – Maximum longitude of points to include in the grid.
- **`minLat`** (*float*) – Minimum latitude of points to include in the grid.
- **`maxLat`** (*float*) – Maximum latitude of points to include in the grid.
- **`margin`** (*float*) – Distance (in degrees) around the centre to fit the grid.
- **`resolution`** (*float*) – Resolution of the grid (in degrees).

Returns 2 2-d arrays containing the distance (km) and bearing (azimuthal) of all points in a grid from the `cLon`, `cLat`, spanning the complete region.

`meshLatLon`(*cLon*, *cLat*, *margin=2*, *resolution=0.01*)

Create a meshgrid of the longitudes and latitudes of a grid.

Parameters

- **`cLon`** (*float*) – Longitude of centre of grid.
- **`cLat`** (*float*) – Latitude of centre of grid.
- **`margin`** (*float*) – Distance (in degrees) around the centre to build the grid.
- **`resolution`** (*float*) – Resolution of the grid (degrees).

Returns Coordinate matrices for the longitude and latitude vectors, covering the region within `margin` degrees of (`cLon`, `cLat`).

`meshLatLonDomain`(*minLon*, *maxLon*, *minLat*, *maxLat*, *margin=2*, *resolution=0.01*)

Create a meshgrid of the lon/lat grid across th full model domain.

Parameters

- **`minLon`** (*float*) – Minimum longitude of the domain.
- **`maxLon`** (*float*) – Maximum longitude of the domain.
- **`minLat`** (*float*) – Minimum latitude of the domain.
- **`maxLat`** (*float*) – Maximum latitude of the domain.
- **`margin`** (*float*) – Distance (in degrees) around the centre to build the grid.
- **`resolution`** (*float*) – Resolution of the grid (degrees).

Returns Coordinate matrices for the longitude and latitude vectors, covering the full domain, plus an additional margin of `margin` degrees.

`theta2bearing` (*theta*)
Converts a cartesian angle (in radians) to an azimuthal bearing (in radians). Assumes -2*pi <= theta <= 2*pi

Parameters **`theta`** – Angle in cartesian coordinate system (+ve anticlockwise from east).

Returns Bearing in azimuth coordinate system (+ve clockwise from north).

`xy2r` (*x*, *y*)
Given x and y arrays, returns the distance between consecutive elements.

Parameters

- **`x`** (`numpy.ndarray`) – x-coordinate of points.
- **`y`** (`numpy.ndarray`) – y-coordinate of points.

Returns Distance (in native units) between consecutive points.

Return type `numpy.ndarray`

11.18 Utilities.metutils module

11.18.1 `metutils` – perform basic meteorological calculations

`convert` (*value*, *input*, *output*)
Convert value from input units to output units.

Parameters

- **`value`** – Value to be converted
- **`input`** (*str*) – Input units.
- **`output`** (*str*) – Output units.

Returns Value converted to `output` units.

`coriolis` (*lat*)
Calculate the Coriolis factor (f) for a given latitude (degrees).

Parameters **`lat`** (`numpy.ndarray` or scalar float) – Latitude (degrees).

Returns Coriolis factor

Return type `numpy.ndarray` or scalar float

`dewPointToRH` (*t_dry*, *t_dew*)
Calculate relative humidity from dry bulb and dew point (in degrees Celsius)

Parameters

- **`t_dry`** (*float*) – Dry bulb temperature (degrees Celsius).
- **`t_dew`** (*float*) – Dew point temperature (degrees Celsius).

Returns Relative humidity (%)

Return type float

`dewPointToVapPr` (*t_dp*, *units_vp='hPa'*)
Calculate vapour pressure from dew point temperature.

Parameters

- **`t_dp`** (*float*) – Dew point temperature (degrees Celsius)
- **`units_vp`** (*str*) – Output units (default `gPressureUnits`).

Returns Vapour pressure of water content.

Return type float

`elevToAirPr` (*elev*, *units_ap='hPa'*)
Approximate air pressure in hectopascals (mb) at a given elevation.

Parameters

- **`elev`** (*float*) – Elevation above sea level (metres).
- **`units_ap`** (*str*) – Units of air pressure (default `"hPa"`).

Returns Approximate air pressure at the given elevation in the specified or default units.

Return type float

`genesisPotential` (*zeta*, *rh*, *vmax*, *shear*)
Calculate genesis potential index

`mixRatToVapPr` (*rat*, *prs*)
Calculate vapour pressure from mixing ratio.

Parameters

- **`rat`** (*float*) – Mixing ratio.
- **`prs`** (*float*) – Air pressure.

Returns Vapour pressure.

Return type float

`rHToDewPoint` (*rh*, *t_dry*)
Calculate dew point from relative humidity and dry bulb (in degrees Celsius).

Parameters

- **`rh`** (*float*) – Relative humidity (%).
- **`t_dry`** (*float*) – Dry bulb temperature (degrees Celsius).

Returns Dew point temperture (degrees Celsius).

Return type float

`rHToMixRat` (*rh*, *tmp*, *prs*, *tmp_units='C'*)
Calculate mixing ratio from relative humidity, temperature and pressure.

Parameters

- **`rh`** (*float*) – Relative humidity (%).
- **`tmp`** (*float*) – Temperature (any units, default degrees Celsius).
- **`prs`** (*float*) – Air pressure (hPa).
- **`tmp_units`** (*str*) – Air temperature units (default degrees Celsius).

Returns Mixing ratio (g/kg).

Return type float

satVapPr (*temp*, *units_vp='hPa'*)
Saturation vapour pressure from temperature in degrees celsius.

Parameters

- **temp** (*float*) – Temperature (degrees celsius).
- **units_vp** (*str*) – Units of the vapour pressure to return. Default is `gPressureUnits`.

Returns saturation vapour pressure in the specified or default units.

Note: Calculation is in kPa with a conversion at the end to the required units.

Example:

```
>>> from metutils import satVapPr
>>> satVapPr(25.)
31.697124349060619
```

spHumToMixRat (*q*, *units='gkg'*)
Calculate mixing ratio from specific humidity. Assumes the input specific humidity variable is in units of g/kg.

Parameters

- **q** (*float*) – Specific humidity (any units, assumed g/kg).
- **units** (*str*) – Units of specific humidity, default g/kg.

Returns Mixing ratio (kg/kg).

Return type float

spHumToRH (*q*, *tmp*, *prs*)
Calculate relative humidity from specific humidity, temperature and pressure.

Parameters

- **q** (*float*) – Specific humidity (g/kg).
- **tmp** (*float*) – Temperature (degrees Celsius).
- **prs** (*float*) – Air pressure (hPa).

Returns Relative humidity (%)

Return type float

vapPrToDewPoint (*vp*, *units_vp='hPa'*)
Calculate dew point from vapour pressure (in kPa)

Parameters

- **vp** (*float*) – Input vapour pressure.
- **units_vp** (*str*) – Input units (default `gPressureUnits`)

Returns dew point temperature (in degrees Kelvin)

Return type float

vapPrToMixRat (*es*, *prs*)
Calculate mixing ratio from vapour pressure In this function, we (mis)use the symbol es for vapour pressure, when it correctly represents saturation vapour pressure. The function can be used for both.

Parameters

- **es** (*float*) – Vapour pressure.
- **prs** (*float*) – Air pressure.

Returns Mixing ratio.

Return type float

vapPrToRH (*vp*, *sat_vp*)
Calculate relative humidity from vapour pressure and saturated vapour pressure.

Parameters

- **vp** (*float*) – Vapour pressure (hPa)
- **sat_vp** (*float*) – Saturation vapour pressure (hPa)

Returns Relative humidity (%)

Example:

```
>>> from metutils import vapPrToRH
>>> vapPrToRH(10., 30.)
33.33333333333
```

vapPrToSpHum (*es*, *prs*)
Convert vapour pressure to specific humidity.

Parameters

- **es** (*float*) – Vapour pressure.
- **prs** (*float*) – Air pressure.

Returs Specific humidity.

Return type float

vapour (*temp*)
Determine saturation vapour pressure, given temperature).

Parameters **temp** (*float*) – Air temperature (degrees Celsius)

Returns Saturation vapour pressure (kPa).

Return type float

wetBulbGlobeTemp (*t_dp*, *temp*)
Calculate Wet Bulb Globe Temperature from Dew Point Temperature and Dry Bulb Temperature (same as air temperature).

Parameters

- **t_dp** (*float*) – Dew point temperature (degees Celsius).
- **temp** (*float*) – Air temperature (degrees Celsius).

Returns Wet bulb globe temperature (degrees Celsius).

Return type float

wetBulbToDewPoint (*db*, *wb*, *elev=0*)
Calculate Dew Point from dry bulb and wet bulb temperatures.

Parameters

- **db** (*float*) – Dry bulb temperature (degrees Celsius).
- **wb** (*float*) – Wet bulb temperature (degrees Celsius).
- **elev** (*float*) – Optional elevation of the observation (metres).

Returns Dew point temperature (degrees Kelvin).

Return type float

wetBulbToRH (*t_db*, *t_wb*, *elev*)
Calculate relative humidity from dry bulb and wet bulb temperatures, and optional elevation.

Parameters

- **t_db** (*float*) – Dry bulb temperature (degrees Celsius).
- **t_wb** (*float*) – Wet bulb temperature (degrees Celsius).
- **elev** (*float*) – Elevation (metres).

Returns Relative humidity (%).

Example:

```
>>> from metutils import wetBulbToRH
>>> wetBulbToRH(25., 10., 0)
6.747686
>>> wetBulbToRH(25., 20., 0)
63.01954
```

wetBulbToVapPr (*db*, *wb*, *elev*, *units_vp='hPa'*)

Calculate vapour pressure from dry bulb and wet bulb temperatures, and optional elevation.

Parameters

- **db** (*float*) – Dry bulb temperature (degrees Celsius).
- **wb** (*float*) – Wet bulb temperature (degrees Celsius).
- **elev** (*float*) – Elevation (metres).
- **units_vp** (*str*) – Output units (default `gPressureUnits`)

Returns Vapour pressure.

Return type float

11.19 Utilities.nctools module

11.19.1 `nctools` – NetCDF utility functions

ncCreateDim (*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

ncCreateVar (*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`

`ncFileInfo` (*filename*, *group=None*, *variable=None*, *dimension=None*)

Print summary information about a netCDF file.

Based on ncinfo (https://code.google.com/p/netcdf4-python/source/browse/trunk/utils/ncinfo)

Parameters

- **`filename`** (*str*) – Path to valid netCDF file.
- **`group`** (*str*) – Name of a *netCDF4.Group* instance to describe.
- **`variable`** (*str*) – Name of a *netCDF4.Variable* instance to describe.
- **`dimension`** (*str*) – Name of a *netCDF4.Dimension* instance to describe.

`ncGetData` (*ncobj*, *var*)

Extract data values from a variable in a netCDF file.

Note that the variable object is a better way to manipulate the variables, as the object includes all attributes (e.g. units, range, long_name, etc) - use *ncGetVar* for that purpose.

Parameters

- **`ncobj`** (`NetCDF4.Dataset`) – `NetCDF4.Dataset` object.
- **`var`** (*str*) – Name of the variable in the dataset to extract.

Returns *numpy.masked_array* containing the data of the variable, with missing values masked.

Return type *numpy.ndarray*

Raises KeyError If variable does not exist in the given file.

`ncGetDims` (*ncobj*, *dim*, *dtype=<type 'float'>*)

Extract the value of a dimension from a netCDF file. This function assumes the file is written following the CF convention, with the values of the dimension stored in a 1-d variable with the same name.

Parameters

- **`ncobj`** (`netCDF4.Dataset`) – `netCDF4.Dataset` object
- **`dim`** (*str*) – Name of the desired dimension.
- **`dtype`** (`numpy.dtype`) – Data type of the dimension

Returns `numpy.ndarray` of the requested dimension.

`ncGetTimes` (*ncobj*, *name='time'*)

Get the time data from a netcdf file.

Parameters

- **`ncobj`** – `netCDF4.Dataset` or `netCDF4.Group` instance.
- **`name`** (*str*) – Name of the time variable.

Return times Array of time dimension values as `datetime` objects.

Return type `numpy.ndarray` of `datetime` objects

`ncGetVar` (*ncobj*, *name*)

Return a *netCDF4.variable* object.

Parameters

- **ncobj** (`netCDF.Group` or `netCDF.Dataset`) – `netCDF.Group` or `netCDF.Dataset` instance.
- **name** (*str*) – Name of the desired variable.

Return varobj `netCDF.Variable` instance

Return type `netCDF.Variable`

ncLoadFile (*filename*)
Load a netCDF file and return a `netCDF4.Dataset` object.

Parameters **filename** (*str*) – Path to the netCDF file to open.

Returns `netCDF4.Dataset` object

Return type `netCDF4.Dataset`

ncSaveGrid (*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.

11.20 Utilities.parallel module

11.20.1 `parallel` – base parallel processing functions

attemptParallel()

Attempt to load Pypar globally as *pp*. If Pypar loads successfully, then a call to *pypar.finalize* is registered to be called at exit of the Python interpreter. This is to ensure that MPI exits cleanly.

If pypar cannot be loaded then a dummy *pp* is created.

Returns A pypar object - either a dummy or the real thing

disableOnWorkers(*f*)

Decorator to disable function *f* calculation on workers. The function will only be evaluated on the master thread.

Parameters **f** (*function*) – Function to be wrapped

11.21 Utilities.pathLocator module

11.21.1 `pathLocator` – determine directory one level above the Utilites folder

getRootDirectory()

Return the name of the path one level above the directory of this current file.

Returns Path name one level above this.

Return type str

is_frozen()

Determine if modules have been built into the interpreter, e.g. by py2exe.

Returns *True* if the modules are frozen, *False* otherwise.

11.22 Utilities.process module

11.22.1 `process` – control processing of files

`pAlreadyProcessed`(*directory*, *filename*, *attribute*, *value*)
Determine if a file has already been processed (i.e. it is stored in gProcessedFiles)

Parameters

- **`directory`** (*str*) – Base path of the file being checked.
- **`filename`** (*str*) – Name of the file.
- **`attribute`** (*str*) – Attribute name to be checked.
- **`value`** (*str*) – Value of the attribute to be tested.

Returns True if the value matches that stored in gProcessedFiles, False otherwise.

Return type boolean

`pArchiveDateFormat`(*date_format=None*)
Set or get archive date format. Archived files can optionally have a date string inserted into the filename to retain all files with the same name, but different timestamps.

Parameters **`date_format`** (*str*) – archive date format (if setting it)

Returns archive date format

Return type str

`pArchiveDir`(*archive_dir=None*)
Set or get the archive directory. If setting the directory, its existence is checked and the directory is created.

Parameters **`archive_dir`** (*str*) – Archive directory (if setting it).

Returns The archive directory.

Return type str

Raises OSError if the directory cannot be created

`pArchiveFile`(*filename*)
Move the file to the archive directory (if specified), inserting a timestamp in the name.

Parameters **`filename`** (*str*) – Full path of the file to be archived.

Returns *True* if the file is successfully moved, *False* otherwise.

Return type bool

`pArchiveTimestamp`(*timestamp=False*)
Set or get archive timstamp flag. If the flag is *True*, then files that are to be archived will have a timestamp inserted into the file name.

Parameters **`timestamp`** (*bool*) – *True* or *False* (if setting it)

Returns The value of *g_archive_timestamp*

Return type bool

`pDeleteDatFile`()
Delete the existing data file - defined in the *gDatFile* variable (list of previously-processed files).

Returns True if existing dat file successfully deleted, False otherwise

Return type bool

`pGetProcessedEntry`(*directory*, *filename*, *attribute*)
Retrieve the value of an attribute for a file from the gProcessedFiles dictionary.

Parameters

- **directory** (*str*) – Path name of the file.
- **filename** (*str*) – File name to retrieve the details of.
- **attribute** (*str*) – Attribute to retrieve.

Returns Attribute value

pGetProcessedFiles (*datFileName=None*)
Retrieve a list of processed files from a dat file. This will also set the global gDatFile.

Parameters **datFileName** (*str*) – Name of a data file to read from.

Returns True if successfully read the data file, False otherwise.

Return type bool

pMoveFile (*origin*, *destination*)
Move a single file to an archive directory.

Parameters

- **origin** (*str*) – Full path of the file to be moved.
- **destination** (*str*) – Full path of the file destination.

Returns *True* if the file is successfully moved, *False* otherwise.

Return type bool

pSetProcessedEntry (*directory*, *filename*, *attribute*, *value*)
Update the attribute of the given file with the given value.

Parameters

- **directory** (*str*) – Base directory of the file.
- **filename** (*str*) – File name.
- **attribute** (*str*) – Name of the file attribute to be updated.
- **value** (*str*) – Attribute value.

pWriteProcessedFile (*filename*)
Write the various attributes of the given file to *gDatFile*

Parameters **filename** (*str*) – Name of file that has been processed.

Returns True if the attributes of the file are successfully stored in gProcessedFiles and written to gDatFile, False otherwise.

Return type bool

11.23 Utilities.progressbar module

11.23.1 `progressbar` – display progress on STDOUT

class ProgressBar (*modname*, *showbar=True*)
Bases: `object`

update (*progress*, *startPos=0*, *endPos=1*)

class SimpleProgressBar (*modname*, *showbar=True*)
Bases: *Utilities.progressbar.ProgressBar*

update (*progress*, *startPos=0*, *endPos=1*, *incr=5.0*)

11.24 Utilities.shptools module

11.24.1 `shptools` - helper functions for manipulating shape files

`parseData`(*data*)

Parse a dict of dicts to generate a list of lists describing the fields, and an array of the corresponding data records

Parameters **`data`** (*dict*) – a dict of dicts with field names as keys, and each sub-dict containing keys of 'Type', 'Length', 'Precision' and 'Data'.

Returns fields, records

Return type list

`shpCreateFile`(*outputFile*, *shapes*, *data*, *shpType*)

Create a shapefile of the give type, containing the given fields.

Parameters

- **`outputFile`** (*str*) – full path (excluding extension!) to the shapefile to create.
- **`shapes`** (`shapefile._Shape`) – Collection of shape objects representing the geometry of features.
- **`shptype`** (*int*) – `shapefile` object type (these are integer values, but you can also use the shapelib.SHPT value).
- **`fields`** (*dict*) – a dictionary of dictionaries with field names as keys, and each sub-dictionary containing keys of 'Type', 'Length','Precision' and 'Data': 'Type' must be one of the following integer values: 0 - strings 1 - integers 2 - doubles 4 - Invalid

Raises `shapefile.ShapefileException` if unable to write the file.

`shpGetField`(*shape_file*, *field_name*, *dtype=<type 'float'>*)

Extract from the records the value of the field corresponding to fieldname.

Parameters

- **`shpFile`** (*str*) – path to a valid shape file
- **`fieldname`** (*str*) – name of a field in the attribute table of the shape file (.dbf)
- **`dtype`** (*dtype*) – type of values in the requested field. Default is float

Returns the value of the given field for each feature

Return type array or list (if *dtype* is a string)

`shpGetVertices`(*shape_file*, *key_name=None*)

Returns a dictionary of arrays containing coordinate pairs representing vertices contained in the shapefile.

Dictionary keys can either be a field contained within the corresponding dbf file (through the optional kwarg keyName), or if no key name is provided, the object id number (i.e. the record number) is used.

WARNING: If any records share the same value for the chosen key, then only one record will be retained in the returned values.

Input: :type shape_file: str :param shape_file: path to a shape file, excluding the extension

Parameters **`key_name`** (*optional str*) – name of a field in the shape file that acts as a key for the dictionary of returned vertices

Returns dict keyed by object id number or optionally a field name, with values being arrays of vertices of the corresponding shape object.

Return type dict

Example: vertices = shpGetVertices('/foo/bar/baz/shp', 'FIELD1')

This function is retained for backwards compatibility. We recommend using the shapefile interface directly for extracting shapes and records.

`shpReadShapeFile` (*shape_file*)

Return the vertices and records for the given shape file

Parameters **`shape_file`** (*str*) – path of input shape file.

Returns vertices

Return type dict

Returns records

Return type dict

`shpSaveTrackFile` (*outputFile*, *tracks*, *fmt='point'*)

Save track data to shapefile. The fields are sorted using the same function as in shpCreateFile, so the fields should be in the correct order.

Parameters

- **`outputFile`** (*str*) – name for the output shapefile, excluding extension.
- **`tracks`** (`Track` object) – collection of track features.
- **`format`** – Type of features to save. "point" will save each record as a single point. "lines" will save each individual TC track as a single (POLYLINE) feature. "segments" will save each segment of a track to a line feature between consecutive observations.

`shpWriteShapeFile` (*outputFile*, *shpType*, *fields*, *shapes*, *records*)

Save data to a shapefile. The fields are sorted using the same function as in shpCreateFile, so the fields should be in the correct order.

Parameters

- **`outputFile`** – A dbf file object created by shpCreateFile
- **`shpType`** – The type of features to be created.
- **`data`** (*dict*) – A dictionary of dictionaries with field names as keys, and each sub-dictionary containing keys of 'Type', 'Length','Precision' and 'Data' 'Type' must be one of the following integer values: 0 - strings 1 - integers 2 - doubles 4 - Invalid

Raises `shapefile.ShapefileException` if unable to write the file.

`tracks2line` (*tracks*, *outputFile*, *dissolve=False*)

Writes tracks to a shapefile as a collection of line features

If dissolve==True, then each track feature is written as a single polyline feature, otherwise each track segment is stored as a separate feature.

Parameters

- **`tracks`** (list of `Track` objects) – `Track` features to store in a shape file
- **`outputFile`** (*str*) – Path to output file destination
- **`dissolve`** (*boolean*) – Store track features or track segments.

`tracks2point` (*tracks*, *outputFile*)

Writes tracks to a shapefile as a collection of point features

Parameters

- **`tracks`** (list of `Track` objects) – `Track` features to store in a shape file
- **`outputFile`** (*str*) – Path to output file destination

11.25 Utilities.smooth module

11.25.1 `smooth` – smooth a 2D field using a Gaussian filter

`gaussKern` (*size*)
Calculate a normalised Gaussian kernel to apply as a smoothing function.

Parameters **`size`** (*int*) – the size of the kernel to use (how many points will be used in the smoothing operation).

Returns `numpy.ndarray` normalised 2D kernel array for use in convolutions

`smooth` (*im*, *n=15*)
Smooth a 2D array *im* by convolving with a Gaussian kernel of size *n*.

Parameters

- **`im`** (`numpy.ndarray`) – Array of values to be smoothed
- **`n`** (*int*) – Number of points to include in the smoothing.

Returns smoothed array (same dimensions as the input array)

11.26 Utilities.stats module

11.26.1 `stats` – helper functions for statistical methods

`between` (*value*, *minval*, *maxval*, *fuzz=2*, *inclusive=True*)
Test whether a value is within some range with some fuzziness at the edges to allow for floating point noise.

The fuzziness is implemented by expanding the range at each end *fuzz* steps using the numpy.nextafter function. For example, with the inputs minval = 1, maxval = 2, and fuzz = 2; the range would be expanded to minval = 0.99999999999999978 and maxval = 2.0000000000000009 before doing comparisons.

Parameters

- **`val`** (*float*) – Value being tested.
- **`minval`** (*float*) – Lower bound of range. Must be lower than *maxval*.
- **`maxval`** (*float*) – Upper bound of range. Must be higher than *minval*.
- **`fuzz`** (*int*) – Number of times to expand bounds using *numpy.nextafter*.
- **`inclusive`** (*boolean*) – Set whether endpoints are within the range.

Returns True if *val* is between *minval* and *maxval*, false otherwise.

From http://penandpants.com/category/python/numpy/

`cdf` (*x*, *y*)
Cumulative Density Function extracted from cdf_lin.m

`cdf2d` (*x*, *y*, *z*)
2D Cumulative Density Function extracted from cdf2d.m Assumes the grid is uniformly defined, i.e. dx and dy are constant.

`circmean` (*samples*, *high=6.283185307179586*, *low=0*)
Compute the circular mean for samples assumed to be in the range [low to high]

`circstd` (*samples*, *high=6.283185307179586*, *low=0*)
Compute the circular standard deviation for samples assumed to be in the range [low to high]

`circvar` (*samples*, *high=6.283185307179586*, *low=0*)
Compute the circular variance for samples assumed to be in the range [low to high]

getCellLonLat (*cellNum*, *gridLimit*, *gridSpace*)
Return the lon/lat of a given cell, based on gridLimit and gridSpace

getCellNum (*lon*, *lat*, *gridLimit*, *gridSpace*)
Return the cell number given longitude and latititude

getOccurence (*occurList*, *indList*)
Returns an array of indices corresponding to cyclone observations that

logger = <logging.RootLogger object at 0x030D3030>
Functions:

cdf(x,y) [1D array of float] Cumulative Density Function extracted from cdf_lin.m

cdf2d(x,y,z): 2D array of float 2D Cumulative Density Function extracted from cdf2d.m

getCellNum(lon, lat, gridLimit, gridSpace): int Determine the cell number based on the lat/lon, the grid bounds and the grid spacing.

getCellLonLat(cellNum, gridLimit, gridSpace): 2D float Determine the lat/lon of the northwestern corner of cellNum

validCellNum(cellNum, gridLimit, gridSpace): boolean Determine whether the given cell number can exist within the bounds of the region defined by gridSpace and gridLimit

maxCellNum(gridLimit, gridSpace): int Determine maximum cell number based on grid limits and spacing.

maxCellNum (*gridLimit*, *gridSpace*)
Get maximum cell number based on grid and grid space

probability (*return_period*)
Return an annual probability given a return period

rMaxDist (*mean*, *sig*, *maxrad=200.*)
Based on the logarithmic distribution reported by Willoughby & Rahn (2004)

statCellFraction (*gridLimit*, *gridSpace*, *valueFile*)
Calculate the fractional value of each grid cell, based on the values stored in valueFile. :param dict gridLimit: Dictionary of bounds of the grid. :param dict gridSpace: Resolution of the grid to calculate values. :param str valueFile: Path to the ascii grid file containing values to sample.

Returns `numpy.ndarray` of fractional values, with length equal to the number of cells

Notes: Still need to include bounds checking to ensure the valueFile data actually covers the gridLimits.

statMaxRange (*minval*, *maxval*, *step*)
Returns the maximum value in a range used for arranging the cells to be evenly spaced

statMinRange (*minval*, *maxval*, *step*)
Returns the minimum value in a range Used for arranging the cells to be evenly spaced

statRemoveNum (*a*, *Num=2147483647*)
Remove all elements in an array for which value is Num

validCellNum (*cellNum*, *gridLimit*, *gridSpace*)
Checks whether the given cell number is valid

11.27 Utilities.tcrandom module

11.27.1 `tcrandom` – extended version of Python's `random` library

class Random (*value=None*)
Bases: `random.Random`

An extension of the standard `random` library to allow sampling from additional distributions.

cauchyvariate(*x0*, *gamma*)
Random variate from the Cauchy distribution.

Parameters

- **x0** (*float*) – Location parameter.
- **gamma** (*float*) – Scale parameter ($\gamma > 0$)

Returns A random variate from the Cauchy distribution.

logisticvariate(*mu*, *sigma*)
Random variate from the logistic distribution.

Parameters

- **mu** (*float*) – Location parameter (μ real).
- **sigma** (*float*) – Scale parameter ($\sigma > 0$).

Returns A random variate from the logistic distribution.

11.28 Utilities.template module

11.28.1 `template` – replace strings with another value

replace(*infile*, *outfile*, *replacements*)
Replace all instances of the keys with values in infile and w write to outfile.

In the input file, keywords to be replaced should be written as '{keyword}'.

e.g. if infile has a line containing a formatted keyword:

```
Input = {PATH}
```

and `replacements = dict('PATH': '/foo/baz')`, then the output file will be written as:

```
Input = /foo/baz
```

Parameters

- **infile** (*str*) – Path to an input file.
- **outfile** (*str*) – Destination file to be written.
- **replacements** (*dict*) – A set of key-value pairs that dictate the replacements to occur in the input file.

11.29 Utilities.timeseries module

11.29.1 `timeseries` - Extract timeseries from each timestep of a simulation

Extract station timeseries from each timestep of a simulation. This samples the regional wind speed, not the site-specific wind speed. To include site-specific effects, you will first need to include the multiplier values for each site in the station file, then run tsmultipliers.py to apply said multipliers to the output.

class Station(*stationid*, *longitude*, *latitude*)
Bases: `object`

insideGrid(*gridx*, *gridy*)
Determine if a point is within the defined grid

class **`Timeseries`** (*configFile*)

Bases: `object`

Timeseries:

Description:

Parameters: Members: Methods: Internal methods:

`extract` (*dt*, *spd*, *uu*, *vv*, *prs*, *gridx*, *gridy*)

Extract data from the grid at the given locations. Data is stored in a dictionary, with keys as the station id's.

Parameters

- **`tstep`** (*float*) – time step being evaluated, as a float (output from matplotlib.num2date)
- **`spd`** – `numpy.ndarray` of speed values.
- **`uu`** – `numpy.ndarray` of eastward wind speed values.
- **`vv`** – `numpy.ndarray` of northward wind speed values.
- **`prs`** – `numpy.ndarray` of pressure values.
- **`gridx`** – `numpy.ndarray` of grid longitudes.
- **`gridy`** – `numpy.ndarray` of grid latitudes.

`sample` (*lon*, *lat*, *spd*, *uu*, *vv*, *prs*, *gridx*, *gridy*)

Extract values from 2-dimensional grids at the given lat/lon.

Parameters

- **`lon`** (*float*) – Longitude of the point to extract.
- **`lat`** (*float*) – Latitude of the point to extract.
- **`spd`** – `numpy.ndarray` of speed values.
- **`uu`** – `numpy.ndarray` of eastward wind speed values.
- **`vv`** – `numpy.ndarray` of northward wind speed values.
- **`prs`** – `numpy.ndarray` of pressure values.
- **`gridx`** – `numpy.ndarray` of grid longitudes.
- **`gridy`** – `numpy.ndarray` of grid latitudes.

Returns speed, esatward and northward wind components, and pressure values at the given location

Return type tuple

`shutdown` ()

Write the data to file, each station to a separate file.

11.30 Utilities.track module

Track-related attributes

class **`Track`** (*data*)

Bases: `object`

A single tropical cyclone track.

The object exposes the track data through the object attributes. For example, If *data* contains the tropical cyclone track data (*numpy.array*) loaded with the `readTrackData()` function, then the central pressure column can be printed out with the code:

```
t = Track(data)
print(t.CentralPressure)
```

Parameters **`data`** (*numpy.ndarray*) – the tropical cyclone track data.

`inRegion`(*gridLimit*)
Check if the tropical cyclone track falls within a region.

Parameters **`gridLimit`** (`dict`) – the region to check. The `dict` should contain the keys `xMin`, `xMax`, `yMin` and `yMax`. The *x* variable bounds the latitude and the *y* variable bounds the longitude.

11.31 Utilities.tracks2shp module

11.31.1 `tracks2shp` – save a track instance as a shape file

`recdropfields`(*rec*, *names*)
Return a new numpy record array with fields in *names* dropped. From http://matplotlib.org/api/mlab_api.html#matplotlib.mlab.rec_drop_fields

Parameters

- **`rec`** – `numpy.recarray` containing a number of fields.
- **`names`** (*list*) – List of names to drop from the record array.

Returns A new `numpy.recarray` with the fields in *names* dropped.

`tracks2line`(*tracks*, *outputFile*, *dissolve=False*)
Writes tracks to a shapefile as a collection of line features

If dissolve==True, then each track feature is written as a single polyline feature, otherwise each track segment is stored as a separate feature.

Parameters

- **`tracks`** (list of `Track` objects) – `Track` features to store in a shape file
- **`outputFile`** (*str*) – Path to output file destination
- **`dissolve`** (*boolean*) – Store track features or track segments.

Raises `shapefile.ShapefileException` if there is an error when attempting to save the file.

`tracks2point`(*tracks*, *outputFile*)
Writes tracks to a shapefile as a collection of point features.

Parameters

- **`tracks`** (list of `Track` objects) – `Track` features to store in a shape file
- **`outputFile`** (*str*) – Path to output file destination

Raises `shapefile.ShapefileException` if there is an error when attempting to save the file.

11.32 Utilities.tsmultipliers module

11.32.1 `tsmultipliers` – apply site-exposure multipliers to time series output

`process_timeseries`(*config_file*)

Process a set of timeseries files to include the multiplier values.

The combined multiplier values are stored in a shape file as fields, and records are keyed by the same code that is used to select stations for sampling.

Parameters **`config_file`** (*str*) – Path to a configuration file.

`startup`()

Parse command line arguments and call the `main()` function.

`tsmultiply`(*inputFile*, *multipliers*, *outputFile*)

Apply multipliers to a single file. Values are combined then written back to the source file.

Parameters

- **`inputFile`** (*str*) – Path to the input timeseries file. This will need to contain the values of the three multipliers (topography, terrain and shielding) for each of eight directions.
- **`multipliers`** (*tuple*) – The eight combined multiplier values for the location.
- **`outputFile`** (*str*) – Destination for the processed file.

Returns The records corresponding to the maximum (localised) wind speed and minimum pressure.

11.33 Utilities.version module

11.33.1 `version` – provide details of software version

`git`(*command*)

Execute the given command with git

Parameters **`command`** (*str*) – A valid git command.

Returns Output from the given command.

Return type str

`status`()

Check status TCRM of code.

Returns A message containing a listing of recent changes to the model code.

Return type str

`version`()

Check version of TCRM code.

Returns Current git commit hash and the date/time of the commit.

Return type str

Raises `subprocess.CalledProcessError` if the git command fails.

Note: This requires `git` to be installed to execute.

11.34 Utilities.vorticity module

11.34.1 `vorticity` – calculate vorticity of a vector field

`absolute`(*u*, *v*, *lon*, *lat*)

Calculates the absolute vorticity (f + curl(u,v)) of a wind field using a basic centred difference scheme. The centred differencing means the returned array is reduced in size by 2 elements in each dimension.

Parameters

- **`u`** (`numpy.ndarray`) – 2-d array of eastward vector component.
- **`v`** (`numpy.ndarray`) – 2-d array of northward vector component.
- **`lon`** (`numpy.ndarray`) – 1-d array of longitudes of grid that defines the vector field.
- **`lat`** (`numpy.ndarray`) – 1-d array of latitudes of grid that defines the vector field.

Returns 2-d `numpy.ndarray` of absolute vorticity values.

`relative`(*u*, *v*, *lon*, *lat*)

Calculates the relative vorticity (curl(u,v)) of a wind field using a basic centred difference scheme. The centred differencing means the returned array is reduced in size by 2 elements in each dimension.

Parameters

- **`u`** (`numpy.ndarray`) – 2-d array of eastward vector component.
- **`v`** (`numpy.ndarray`) – 2-d array of northward vector component.
- **`lon`** (`numpy.ndarray`) – 1-d array of longitudes of grid that defines the vector field.
- **`lat`** (`numpy.ndarray`) – 1-d array of latitudes of grid that defines the vector field.

Returns 2-d `numpy.ndarray` of relative vorticity values.

CHAPTER

TWELVE

GLOSSARY

Annual Exceedance Probability The probability of a threshold being exceeded once (or more) in any year.

Genesis Formation of a tropical cyclone event.

Hazard The magnitude of wind speed. TCRM generates a 3-second gust wind speed, at 10 metres above ground over open, flat terrain (an aerodynamic roughness length of 0.02 m).

Kernel density estimation A non-parametric method to estimate the probability density function of a random variable.

Lysis Termination of a tropical cyclone event. Can be caused by the tropical cyclone weakening through stochastic process, landfall decay or exiting the defined domain.

mpirun mpirun is a runtime environment for the Message Passing Interface (MPI). See http://www.mcs.anl.gov/research/projects/mpi/

Pypar PyPar is a python library that provides efficient and scalable parallelism using the message passing interface (MPI) to handle big data and highly computational problems. See http://github.com/daleroberts/pypar

Return period The average time between exceedances of a given threshold. Compare to Annual Exceedance Probability.

Tropical cyclone A tropical cyclone is a rapidly-rotating storm system characterized by a low-pressure center, strong winds, and a spiral arrangement of thunderstorms that produce heavy rain.

CHAPTER

THIRTEEN

GNU GENERAL PUBLIC LICENSE

13.1 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: **(1)** assert copyright on the software, and **(2)** offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

13.2 TERMS AND CONDITIONS

13.2.1 0. Definitions

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that **(1)** displays an appropriate copyright notice, and **(2)** tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

13.2.2 1. Source Code

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that **(a)** is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and **(b)** serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

13.2.3 2. Basic Permissions

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

13.2.4 3. Protecting Users' Legal Rights From Anti-Circumvention Law

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

13.2.5 4. Conveying Verbatim Copies

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

13.2.6 5. Conveying Modified Source Versions

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- **a)** The work must carry prominent notices stating that you modified it, and giving a relevant date.
- **b)** The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- **c)** You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- **d)** If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

13.2.7 6. Conveying Non-Source Forms

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- **a)** Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- **b)** Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either **(1)** a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or **(2)** access to copy the Corresponding Source from a network server at no charge.
- **c)** Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- **d)** Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- **e)** Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either **(1)** a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or **(2)** anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

13.2.8 7. Additional Terms

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- **a)** Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- **b)** Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- **c)** Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- **d)** Limiting the use for publicity purposes of names of licensors or authors of the material; or
- **e)** Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- **f)** Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

13.2.9 8. Termination

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated **(a)** provisionally, unless and until the copyright holder explicitly and finally terminates your license, and **(b)** permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

13.2.10 9. Acceptance Not Required for Having Copies

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

13.2.11 10. Automatic Licensing of Downstream Recipients

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

13.2.12 11. Patents

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either **(1)** cause the Corresponding Source to be so available, or **(2)** arrange to deprive yourself of the benefit of the patent license for this particular work, or **(3)** arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license **(a)** in connection with copies of the covered work conveyed by you (or copies made from those copies), or **(b)** primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

13.2.13 12. No Surrender of Others' Freedom

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13.2.14 13. Use with the GNU Affero General Public License

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

13.2.15 14. Revised Versions of this License

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

13.2.16 15. Disclaimer of Warranty

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13.2.17 16. Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

13.2.18 17. Interpretation of Sections 15 and 16

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

13.3 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

- genindex
- modindex
- search

PYTHON MODULE INDEX