

Documentation of pargopy

version

HERRY Thomas

May 22, 2018

Contents

Welcome to Pargopy's documentation!	1
pargopy package	1
Submodules	1
argodb.py	1
argotools.py	1
atlas.py	3
check_decreasing_pressure.py	3
decorator.py	3
general_tools.py	4
interpolation_tools.py	4
param.py	5
research_tools.py	5
stats.py	6
task_giver.py	7
task_giver_stats.py	7
tile.py	8
atlas_tools.py	8
mouseprofile.py	8
interp.py	8
eape.py	9
variable_selector.py	9
netCDF_form.py	9
test_stats.py	9
manual_check_tools.py	9
variable_selector.py	10
Module contents	10
Indices and tables	10
Index	11
Python Module Index	15

Welcome to Pargopy's documentation!

pargopy package

Submodules

argodb.py

Created on Mon Mar 12 13:10:24 2018

File creating the summary of ARGO used to generate the atlas

`pargopy.argodb.get_all_wmos ()`

Return a dictionary of all wmo (list of int) with dac (string) as keys HAVING a *_prof.nc file
A few wmo have no *_prof.nc (352 exactly) because ... they have actually no profile reported

Return type: dic

`pargopy.argodb.get_header_of_all_profiles (wmostats)`

Build argodb from the infos in wmostats

Once it is created it is more efficient to read it from the disk using 'read_argodb()'

Return type: dic

`pargopy.argodb.get_header_of_all_wmos (wmodic)`

Get the header of all wmo

Return type: dic

`pargopy.argodb.main ()`

Main function of argodb.py

`pargopy.argodb.propagate_flag_backward (argodb, subargodb, verbose=True)`

Update argodb FLAG using subargodb :rtype: None

`pargopy.argodb.update_wmodic ()`

Read the full argodb database and update argodb.pkl

Return type: None

argotools.py

Created on Mon Mar 12 13:10:24 2018

@author: herry

Tools used by all the python files to generate the atlas

`pargopy.argotools.conversion_gregd_juld (year, month, day)`

Method converting gregorian day into julian day

Return type: float

`pargopy.argotools.conversion_juld_gregd (juld)`

Method converting julian day into gregorian day

Return type: list of int

`pargopy.argotools.count_profiles_in_database (wmostats)`

Count the total number of profiles in database :rtype: int

`pargopy.argotools.count_wmos (wmodic)`

Count the total number of wmo in the Argo database base :rtype: list of wmo

`pargopy.argotools.dac_from_wmo (wmodic, wmo)`

Retrieve the dac of a wmo :rtype: list of dac

Welcome to Pargopy's documentation!

`pargopy.argotools.extract_idx_from_argodb(argodb, idx)`

Return a argodb type dictionary that is a subset of argodb and containing only entries given in idx (list)

Return type: dic

`pargopy.argotools.extract_idx_from_wmostats(wmostats, idx)`

Return a wmostats type dictionary that is a subset of wmostats and containing only entries given in idx (list)

Return type: dic

`pargopy.argotools.extract_idx_inside_tile(res, argodb)`

Extract from 'argodb' the list of profiles that are inside the tile The tile limits are given by 'res'

rtype: dic

`pargopy.argotools.fix_flag_latlonf(argodb)`

Set a flag error for profiles having bad masked positions

Bad masked position yield value of 99999. This fix only concerns a few profiles from dac='jma'

Return type: None

`pargopy.argotools.flag_argodb(argodb, wmodic)`

Add the flag to argodb

Return type: dic

`pargopy.argotools.get_datamode(data)`

Return the data mode of the profile

Return type: np.array

`pargopy.argotools.get_idx_from_list_wmo(argodb, wmos)`

Get the list of profile indices present in argodb that correspond to the list of wmos

Return type: list of int

`pargopy.argotools.get_profile_file_path(dac, wmo)`

Return the file path to the *_prof.nc data file

Return type: string

`pargopy.argotools.get_tag(kdac, wmo, kprof)`

Compute the tag number of a profile

The inverse of get_tag() is retrieve_infos_from_tag()

Return type: int

`pargopy.argotools.plot_location_profiles(argodb)`

Plot a scatter plot of profiles in argodb

argodb can be the full database or any subset

Return type: None

`pargopy.argotools.plot_wmo_data(dac, wmo)`

Plot raw 'TEMP' data for dac, wmo

Return type: None

`pargopy.argotools.plot_wmos_stats(wmostats)`

Plot the histogram of number of profiles per number of levels

Return type: None

`pargopy.argotools.read_dic(name, path_localdata)`

Function used to read each dic used to create .pkl files Regroups : - read_wmodic, read_wmstats, read_argodb from argodb.py - read_argo_filter from research_tools.py - read_tile from tile.py

Return type: dict

`pargopy.argotools.read_profile` (dac, wmo, iprof=None, header=False, data=False, headerqc=False, dataqc=False, verbose=True)

Basic driver to read the *_prof.nc data file

The output is a dictionary of vectors - read one or all profiles read the header (lat, lon, juld) or not - read the data or not always return IDAC, WMO, N_PROF, N_LEVELS - and DATA_UPDATE (all 5 are int)

Return type: dic

`pargopy.argotools.retrieve_infos_from_tag` (argodb, tag)

Retrieve idac, wmo and iprof from tag (array of int)

It is the inverse of `get_tag()`

Return type: dic

`pargopy.argotools.test_tiles` (argo, i)

Test that tile 'i' is correctly defined with all profiles positions within the tile limits, encompassing the margins

Return type: None

```
pargopy.argotools.tile_definition()
Define the tiles coordinates, in the form of a vector of lon and lat + their margins
The tile indexing is
|-----+-----+-----+-----+-----| | 280 | 281 | 282 | ... | 299 | |-----+-----+-----+-----+-----| | ... | ... | ... | ... | ... |
|-----+-----+-----+-----+-----| | 20 | 21 | 22 | ... | 39 | |-----+-----+-----+-----+-----| | 0 | 1 | 2 | ... | 19 |
|-----+-----+-----+-----+-----|
rtype: float, float, int, int, float, float

pargopy.argotools.write_dic(name, dic, path_localdata)
Function used to write each dic used to create .pkl files Regroups : - write_wmodic, write_wmstats, write_argodb
from argodb.py - write_argo_filter from research_tools.py - write_tile from tile.py
Return type: None
```

Created on Mon Mar 12 13:10:24 2018 File creating the atlas of stats from netCDF4 import Dataset import numpy as np import os import argotools as argotools import param as param

```
pargopy.atlas.atlas_filename(diratlas, reso, year, mode, typestat)
pargopy.atlas.get_glo_grid(reso)
pargopy.atlas.glue_tiles(reso)
    Glue the stats tiles together into a global 3D atlas
pargopy.atlas.gridindex2lonlat(ix, iy)
pargopy.atlas.ij2tile(i, j)
```

Created on Mon Mar 12 13:10:24 2018

File used to avoid values error for pressure, temperature and salinity

```
pargopy.check_decreasing_pressure.check_pressure (p)
```

```
pargopy.check_decreasing_pressure.try to remove duplicate pressure (p)
```

```
Created on Thu Apr 26 07:16:12 2018
@author: therry
paragopy.decorator.call_results(filename)
```

Function used to know the number of call and the mean time of execution of each function from a chosen file Do not forget to import the file you want to analyse before using this function !

```
class pargopy.decorator.exec_time (fonc)
```

Bases: **object**

Decorator used to know the number of call for a function and the Mean time for its execution

```
results ()
```

Print the result : - Number of call - Mean time for execution

general_tools.py

```
pargopy.general_tools.compute_weight (x, y, lon, lat, reso)
```

Compute the weight between points (x, y) and point (lon, lat) with a gaussian filter

```
pargopy.general_tools.cubiccoef (z0, zs)
```

Weights for cubic interpolation at z0 given the four depths in zs

```
pargopy.general_tools.deg_to_rad (angle)
```

Degree to radians

```
pargopy.general_tools.dist_sphe (x, y, lon, lat)
```

Compute the spherical arc between two points on the unit sphere

```
pargopy.general_tools.fixqcarray (qc)
```

Transform a Argo 2D array of qc flags (string) into a int array

```
pargopy.general_tools.insitu_to_absolute (Tis, SP, p, lon, lat, zref)
```

Transform in situ variables to TEOS10 variables

```
pargopy.general_tools.interp_at_zref (CT, SA, z, zref)
```

Interpolate CT and SA from their native depths z to zref

```
pargopy.general_tools.lincoef (z0, zs)
```

Weights for linear interpolation at z0 given the two depths in zs

```
pargopy.general_tools.npa2ma (x)
```

Convert a numpy array into a masked array set mask=True on NaN

```
pargopy.general_tools.raw_to_interpolate (temp, sal, pres, temp_qc, sal_qc, pres_qc, lon, lat, zref)
```

Interpolate in situ data on zref depths ierr = 0: no pb ierr > 0: pb

```
pargopy.general_tools.remove_bad_qc (temp, sal, pres, temp_qc, sal_qc, pres_qc)
```

Return the index list of data for which the three qc's are 1 and the error flag ierr ierr = 0 : no pb ierr = 1 : too few data in the profile

```
pargopy.general_tools.select_depth (zref, z)
```

Return the number of data points we have between successive zref. This is used to decide which interpolation is the best: none, linear or cubic. For endpoints (zref=0) and bottom(zref=2000) use linear extrapolation

interpolation_tools.py

Created on Wed Mar 14 14:37:02 2018

@author: herry

Tools used for the interpolation of the values from ARGO

```
pargopy.interpolation_tools.insitu_to_absolute (Tis, SP, p, lon, lat, zref)
```

Transform in situ variables to TEOS10 variables

Return type: float, float, float

```
pargopy.interpolation_tools.interp_at_zref (CT, SA, z, zref)
```

Interpolate CT, SA, dCT/dz and dSA/dz from their native depths z to zref

Method: we use piecewise Lagrange polynomial interpolation

For each `zref[k]`, we select a list of `z[j]` that are close to `zref[k]`, imposing to have `z[j]` that are above and below `zref[k]` (except near the boundaries)
If only two `z[j]` are found then the result is a linear interpolation
If n `z[j]` are found then the result is a n -th order interpolation.
For interior points we may go up to 6-th order
For the surface level (`zref==0`), we do extrapolation
For the bottom level (`zref=2000`), we do either extrapolation or interpolation if data deeper than 2000 are available.

Return type: float, float, float, float

`pargopy.interpolation_tools.interpolate_profiles` (subargodb, wmodic)
Interpolate the profiles in subargodb

Return type: dic

`pargopy.interpolation_tools.lagrangepoly` (`x0`, `xi`)
Weights for polynomial interpolation at `x0` given a list of `xi` return both the weights for function (`cs`) and its first derivative (`ds`)
Example: `lagrangepoly(0.25, [0, 1]) >>> [0.75, 0.25,], [1, -1]`

Return type: float, float

`pargopy.interpolation_tools.raw_to_interpolate` (`temp`, `sal`, `pres`, `temp_qc`, `sal_qc`, `pres_qc`, `lon`, `lat`, `zref`)
Interpolate in situ data on `zref` depths
`ierr = 0`: no pb
`ierr > 0`: pb

Return type: float, float, float, float, float, float, float, int

`pargopy.interpolation_tools.remove_bad_qc` (`temp`, `sal`, `pres`, `temp_qc`, `sal_qc`, `pres_qc`)
Return the index list of data for which the three qc's are 1 and the error flag `ierr`
`ierr = 0` : no pb
`ierr = 1` : too few data in the profile

Return type: list, int

`pargopy.interpolation_tools.select_depth` (`zref`, `z`)
Return the number of data points we have between successive `zref`.
for each interval k , we select the `z_j` such that
 $zref[k] \leq z_j < zref[k+1]$, for $k=0 \dots nref-2$
 $zref[nref-1] \leq z_j < zextra$, for $k=nref-1$
and return
`nbperintervale[k]` = number of `z_j`
`kperint[k]` = list of `j`'s
with `zextra = 2*zref[-1] - zref[-2]`

Return type: int, list

param.py

Created on Wed Apr 11 11:01:25 2018

@author: herry

File defining the pathes to use this program

research_tools.py

Created on Tue Mar 20 15:24:20 2018

@author: herry

Tools used to generate the filters that are used after to generate the tiles

`pargopy.research_tools.creating_tiles` ()
Giving values to the variables

Return type: None

`pargopy.research_tools.extract_idx_from_argodb(argodb, idx)`

Return a argodb type dictionary that is a subset of argodb and containing only entries given in idx (list)

Return type: dic

`pargopy.research_tools.extract_idx_from_wmostats(wmostats, idx)`

Return a wmostats type dictionary that is a subset of wmostats and containing only entries given in idx (list)

Return type: dic

`pargopy.research_tools.get_idx_from_list_wmo(argodb, wmos)`

Get the list of profile indices present in argodb that correspond to the list of wmos

Return type: list of int

`pargopy.research_tools.get_idx_from_tiles_lim(res, argodb)`

Get the list of profile indices present in argodb that correspond to the list of wmos

Return type: dic

`pargopy.research_tools.main()`

Main function of stats.py

`pargopy.research_tools.mbox(x1, x2, y1, y2, dlat, dlon, col, itile, m)`

`pargopy.research_tools.plot_map()`

`pargopy.research_tools.test_tiles(argo_extract, i)`

Test to know if the tiles are correctly done with the lat and lon limits

Return type: None

stats.py

Compute statistics on one tile

`pargopy.stats.compute_mean_at_zref(itile, reso_deg, mode, date)`

Compute the mean at depths zref

Return type: dict

`pargopy.stats.compute_stats_at_zref(mode, date, grid_lon, grid_lat, reso_deg)`

compute statistics on a small grid defined at grid_lon x grid_lat the small grid should fit inside one tile

`pargopy.stats.compute_std_at_zref(itile, reso_deg, timeflag, mode, date, verbose=False)`

Compute the standard deviations at depths zref

Return type: dict

`pargopy.stats.create_stat_file(itile, typestat, reso, timeflag, date, mode)`

Create statistics netcdf file

Return type: None

`pargopy.stats.date_mode_filter(mode, date, itile)`

Make the tile filter to choose keep only the chosen mode ('R', 'A', 'D', 'AD' or 'RAD') and the profiles under the chosen date (year, month, day) Return the tile_extract according to the filters used.

Return type: dic

`pargopy.stats.generate_filename(itile, typestat, reso, timeflag, date, mode)`

Generates the filename of the netCDF stat file

Return type: str

`pargopy.stats.grid_coordinate(itile, reso)`

Returns the coordinates of each point of the grid for a given tile

coordinates are round multiples of reso_deg reso sets the grid resolution, typically 0.5deg

Return type: numpy.ndarray, numpy.ndarray

`pargopy.stats.main(itile, typestat, reso, timeflag, date, mode)`
Main function of stats.py

`pargopy.stats.read_stat_file(itile, typestat, reso, timeflag, date, mode, var_choice)`
Read statistics into a netcdf file

Return type: dict

`pargopy.stats.retrieve_tile_from_position(lon0, lat0)`
Return the tile index in which (lon0, lat0) sits

Return type: list

`pargopy.stats.write_stat_file(itile, typestat, reso, timeflag, date, mode, stats_mode)`
Write statistics into a netcdf file

Return type: None

task_giver.py

Masternslave used for the tiles creation

`pargopy.task_giver.getavailableslave(slavestate)`
Return the index of a slave that is awaiting a task. A busy slave has a state == 0. If all slaves are busy then wait until a msg is received, the msg is sent upon task completion by a slave. Then determin who sent the msg. The msg is collected in the answer array. By scanning it, we determine who sent the message.

`pargopy.task_giver.master_work_blocking(nslaves)`
Master organizes the work using blocking communications with slaves

`pargopy.task_giver.master_work_nonblocking(nslaves)`
Main program for master
Master basically supervises things but does no work

`pargopy.task_giver.ordering_tasks(tasks)`
Sort the tasks according to their workload workload is proportional to size of the tile file

Return type: list of int

`pargopy.task_giver.slave_work_blocking(islave)`
Very simple function for slaves based on blocking communications with master

`pargopy.task_giver.slave_work_nonblocking(islave)`
Main function for slaves.
Slaves enter an infinite loop: keep receiving messages from the master until reception of 'done'. Each messages describes the task to be done. When a new task is received slave treats it. At the end of it the slave sends a message to the master saying that he is over, and that he is available for a new task.

task_giver_stats.py

Masternslave used to create the stats files

`pargopy.task_giver_stats.getavailableslave(slavestate)`
Return the index of a slave that is awaiting a task. A busy slave has a state == 0. If all slaves are busy then wait until a msg is received, the msg is sent upon task completion by a slave. Then determin who sent the msg. The msg is collected in the answer array. By scanning it, we determine who sent the message.

`pargopy.task_giver_stats.master_work_nonblocking(nslaves)`
Main program for master
Master basically supervises things but does no work

`pargopy.task_giver_stats.ordering_tasks(tasks)`
Sort the tasks according to their workload workload is proportional to size of the tile file

Return type: list of int

```
pargopy.task_giver_stats.slave_work_nonblocking (islave)
```

Main function for slaves.

Slaves enter an infinite loop: keep receiving messages from the master until reception of 'done'. Each messages describes the task to be done. When a new task is received slave treats it. At the end of it the slave sends a message to the master saying that he is over, and that he is available for a new task.

tile.py

Created on Wed Apr 4 10:07:26 2018

@author: herry

File used to generate the different tiles of the atlas

```
pargopy.tile.generate_argotiles ()
```

Generate the argodb dictionary for each tile and save it in 'argo%003i'

```
pargopy.tile.generate_tile (i)
```

Interpolate all Argo profiles in tile 'i' onto 'zref' depths. Save the result in the 'tile%003i.pkl' file

```
pargopy.tile.main (itile)
```

Main function of tile.py

```
pargopy.tile.plot_tile (i)
```

Plots the tiles values (Ti, Si, Ri) with the values non interpolate

Return type: None

atlas_tools.py

Created on Mon Mar 12 13:10:24 2018

File used to create the on-click tools for the atlas

```
pargopy.atlas_tools.compute_stats_near_point (lon0, lat0)
```

```
pargopy.atlas_tools.lonlatstr (lon, lat)
```

```
pargopy.atlas_tools.onclick (event)
```

```
pargopy.atlas_tools.retrieve_tile_from_position (lon0, lat0)
```

Return the tile index in which (lon0, lat0) sits

Return type: list

```
pargopy.atlas_tools.select_profiles_near_point (lon0, lat0)
```

Return the list of profiles at a distance 'reso' from (lon0, lat0)

Return type: None

mouseprofile.py

```
class pargopy.mouseprofile.MouseProfile (line, tag, filename)
```

Bases: **object**

```
connect ()
```

```
on_press (event)
```

first click: highlight the line and indicate the tag number second click: confirm action on this profile

```
on_release (event)
```

on release we reset the press data

interp.py

Welcome to Pargopy's documentation!

```
class pargopy.interp.PLagrange(x, y, order=2, extralims=None)
    Bases: scipy.interpolate.interpolate.PPoly

pargopy.interp.f(x)
```

eape.py

```
pargopy.eape.compute_eape(z0, r0, cs, rho)
    Compute the eape of rho(...,z0) based on reference profile r0(z0) with sound_speed cs(z0)

pargopy.eape.integrate_K(zref, z, K)
    compute  $\int_0^z K(z')dz'$ 

pargopy.eape.wherenotnan(x)

pargopy.eape.wherenotnanxy(x, y)
```

variable_selector.py

Created on Wed May 9 12:57:38 2018

@author: therry

```
pargopy.variable_selector.compute_at_zref(itile, reso_deg, mode, date, block_choice,
tile_dict=None)
    Compute the variables at depths zref

    Return type: dict
```

netCDF_form.py

Created on Wed Apr 25 12:20:04 2018

@author: therry

```
pargopy.netCDF_form.create_dim(filename, zref, nlat, nlon, mode, date)

pargopy.netCDF_form.create_var(filename, var_list)
    Create the netcdf file 'filename' for the list of variables 'var_list' the dimensions '(zref, lat, lon)' and the attributes of
    each variable is retrieved from the json file.
    Warning 'filename' should be created first

    Return type: None

pargopy.netCDF_form.read_var(filename, var_list)
    Read the list of variables 'var_list' from netcdf file 'filename' Return the result as a dictionary of numpy arrays

    Return type: dict
```

```
pargopy.netCDF_form.write_var(filename, var_list, var_dic)
    Write the list of variables 'var_list' in the netcdf file 'filename'. Variables are transfered in the form of a dictionary
    'var_dic', where each entry is a numpy array.
```

test_stats.py

manual_check_tools.py

Created on Wed May 16 07:30:13 2018

@author: therry

```
pargopy.manual_check_tools.retrieve_coords_from_tag(tag)
    Retrieve the coords (lon, lat) of a given tag of a profile

    Return type: list[int, int]
```

Indices and tables

`pargopy.manual_check_tools.retrieve_itile_from_coords(xlat, xlon)`

Retrieve the tile which contains the profile with its lat and lon

Return type: int

`pargopy.manual_check_tools.update_argoextract(itile, tag)`

Returns an `argo_extract` dict where the profile flagged as wrong with the given tag is updated (flag updated with value 404)

Return type: dict

`pargopy.manual_check_tools.update_stats(itile, tile)`

Returns the stats without the profile checked as wrong

Return type: dict

`pargopy.manual_check_tools.update_tile(itile, tag)`

Returns the tile updated without the tag checked as wrong

Return type: dict

variable_selector.py

Created on Wed May 9 12:57:38 2018

@author: therry

`pargopy.variable_selector.compute_at_zref(itile, reso_deg, mode, date, block_choice, tile_dict=None)`

Compute the variables at depths zref

Return type: dict

Module contents

Indices and tables

- `genindex`
- `modindex`
- `search`

Index

A

[atlas_filename\(\)](#) (in module [pargopy.atlas](#))

C

[call_results\(\)](#) (in module [pargopy.decorator](#))

[check_pressure\(\)](#) (in module [pargopy.check_decreasing_pressure](#))

[compute_at_zref\(\)](#) (in module [pargopy.variable_selector](#)) [1]

[compute_eape\(\)](#) (in module [pargopy.eape](#))

[compute_mean_at_zref\(\)](#) (in module [pargopy.stats](#))

[compute_stats_at_zref\(\)](#) (in module [pargopy.stats](#))

[compute_stats_near_point\(\)](#) (in module [pargopy.atlas_tools](#))

[compute_std_at_zref\(\)](#) (in module [pargopy.stats](#))

[compute_weight\(\)](#) (in module [pargopy.general_tools](#))

[connect\(\)](#) ([pargopy.mouseprofile.MouseProfile](#) method)

[conversion_gregd_juld\(\)](#) (in module [pargopy.argotools](#))

[conversion_juld_gregd\(\)](#) (in module [pargopy.argotools](#))

[count_profiles_in_database\(\)](#) (in module [pargopy.argotools](#))

[count_wmos\(\)](#) (in module [pargopy.argotools](#))

[create_dim\(\)](#) (in module [pargopy.netCDF_form](#))

[create_stat_file\(\)](#) (in module [pargopy.stats](#))

[create_var\(\)](#) (in module [pargopy.netCDF_form](#))

[creating_tiles\(\)](#) (in module [pargopy.research_tools](#))

[cubiccoef\(\)](#) (in module [pargopy.general_tools](#))

D

[dac_from_wmo\(\)](#) (in module [pargopy.argotools](#))

[date_mode_filter\(\)](#) (in module [pargopy.stats](#))

[deg_to_rad\(\)](#) (in module [pargopy.general_tools](#))

[dist_sphe\(\)](#) (in module [pargopy.general_tools](#))

E

[exec_time](#) (class in [pargopy.decorator](#))

[extract_idx_from_argodb\(\)](#) (in module [pargopy.argotools](#))

(in module [pargopy.research_tools](#))

[extract_idx_from_wmostats\(\)](#) (in module [pargopy.argotools](#))

(in module [pargopy.research_tools](#))

[extract_idx_inside_tile\(\)](#) (in module [pargopy.argotools](#))

F

[f\(\)](#) (in module [pargopy.interp](#))

[fix_flag_latlonf\(\)](#) (in module [pargopy.argotools](#))

[fixqccarray\(\)](#) (in module [pargopy.general_tools](#))

[flag_argodb\(\)](#) (in module [pargopy.argotools](#))

G

[generate_argotiles\(\)](#) (in module [pargopy.tile](#))

[generate_filename\(\)](#) (in module [pargopy.stats](#))

[generate_tile\(\)](#) (in module [pargopy.tile](#))

[get_all_wmos\(\)](#) (in module [pargopy.argodb](#))

[get_datamode\(\)](#) (in module [pargopy.argotools](#))

[get_glo_grid\(\)](#) (in module [pargopy.atlas](#))

[get_header_of_all_profiles\(\)](#) (in module [pargopy.argodb](#))

[get_header_of_all_wmos\(\)](#) (in module [pargopy.argodb](#))

[get_idx_from_list_wmo\(\)](#) (in module [pargopy.argotools](#))

(in module [pargopy.research_tools](#))

[get_idx_from_tiles_lim\(\)](#) (in module [pargopy.research_tools](#))

[get_profile_file_path\(\)](#) (in module [pargopy.argotools](#))

[get_tag\(\)](#) (in module [pargopy.argotools](#))

[getavailableslave\(\)](#) (in module [pargopy.task_giver](#))

(in module [pargopy.task_giver_stats](#))

[glue_tiles\(\)](#) (in module [pargopy.atlas](#))

[grid_coordinate\(\)](#) (in module [pargopy.stats](#))

[gridindex2lonlat\(\)](#) (in module [pargopy.atlas](#))

I

[ij2tile\(\)](#) (in module [pargopy.atlas](#))

[insitu_to_absolute\(\)](#) (in module [pargopy.general_tools](#))

(in module [pargopy.interpolation_tools](#))

[integrate_K\(\)](#) (in module [pargopy.eape](#))

[interp_at_zref\(\)](#) (in module [pargopy.general_tools](#))

(in module [pargopy.interpolation_tools](#))

[interpolate_profiles\(\)](#) (in module [pargopy.interpolation_tools](#))

L

[lagrangepoly\(\)](#) (in module [pargopy.interpolation_tools](#))

[lincoef\(\)](#) (in module [pargopy.general_tools](#))

[lonlatstr\(\)](#) (in module [pargopy.atlas_tools](#))

M

main() (in module `pargopy.argodb`)
(in module `pargopy.research_tools`)
(in module `pargopy.stats`)
(in module `pargopy.tile`)
master_work_blocking() (in module `pargopy.task_giver`)
master_work_nonblocking() (in module `pargopy.task_giver`)
(in module `pargopy.task_giver_stats`)
mbox() (in module `pargopy.research_tools`)
MouseProfile (class in `pargopy.mouseprofile`)

N

npa2ma() (in module `pargopy.general_tools`)

O

on_press() (`pargopy.mouseprofile.MouseProfile` method)
on_release() (`pargopy.mouseprofile.MouseProfile` method)
onclick() (in module `pargopy.atlas_tools`)
ordering_tasks() (in module `pargopy.task_giver`)
(in module `pargopy.task_giver_stats`)

P

`pargopy` (module)
`pargopy.argodb` (module)
`pargopy.argotools` (module)
`pargopy.atlas` (module)
`pargopy.atlas_tools` (module)
`pargopy.check_decreasing_pressure` (module)
`pargopy.decorator` (module)
`pargopy.eape` (module)
`pargopy.general_tools` (module)
`pargopy.interp` (module)
`pargopy.interpolation_tools` (module)
`pargopy.manual_check_tools` (module)
`pargopy.mouseprofile` (module)
`pargopy.netCDF_form` (module)
`pargopy.param` (module)
`pargopy.research_tools` (module)
`pargopy.stats` (module)
`pargopy.task_giver` (module)
`pargopy.task_giver_stats` (module)

`pargopy.test_stats` (module)
`pargopy.tile` (module)
`pargopy.variable_selector` (module) [1]
`PLagrange` (class in `pargopy.interp`)
`plot_location_profiles()` (in module `pargopy.argotools`)
`plot_map()` (in module `pargopy.research_tools`)
`plot_tile()` (in module `pargopy.tile`)
`plot_wmo_data()` (in module `pargopy.argotools`)
`plot_wmos_stats()` (in module `pargopy.argotools`)
`propagate_flag_backward()` (in module `pargopy.argodb`)

R

`raw_to_interpolate()` (in module `pargopy.general_tools`)
(in module `pargopy.interpolation_tools`)
`read_dic()` (in module `pargopy.argotools`)
`read_profile()` (in module `pargopy.argotools`)
`read_stat_file()` (in module `pargopy.stats`)
`read_var()` (in module `pargopy.netCDF_form`)
`remove_bad_qc()` (in module `pargopy.general_tools`)
(in module `pargopy.interpolation_tools`)
`results()` (`pargopy.decorator.exec_time` method)
`retrieve_coords_from_tag()` (in module `pargopy.manual_check_tools`)
`retrieve_infos_from_tag()` (in module `pargopy.argotools`)
`retrieve_itile_from_coords()` (in module `pargopy.manual_check_tools`)
`retrieve_tile_from_position()` (in module `pargopy.atlas_tools`)
(in module `pargopy.stats`)

S

`select_depth()` (in module `pargopy.general_tools`)
(in module `pargopy.interpolation_tools`)
`select_profiles_near_point()` (in module `pargopy.atlas_tools`)
`slave_work_blocking()` (in module `pargopy.task_giver`)
`slave_work_nonblocking()` (in module `pargopy.task_giver`)
(in module `pargopy.task_giver_stats`)

T

`test_tiles()` (in module `pargopy.argotools`)
(in module `pargopy.research_tools`)

`tile_definition()` (in module `pargopy.argotools`)

`try_to_remove_duplicate_pressure()` (in module `pargopy.check_decreasing_pressure`)

U

`update_argoextract()` (in module `pargopy.manual_check_tools`)

`update_stats()` (in module `pargopy.manual_check_tools`)

`update_tile()` (in module `pargopy.manual_check_tools`)

`update_wmodic()` (in module `pargopy.argodb`)

W

`wherenotnan()` (in module `pargopy.eape`)

`wherenotnanxy()` (in module `pargopy.eape`)

`write_dic()` (in module `pargopy.argotools`)

`write_stat_file()` (in module `pargopy.stats`)

`write_var()` (in module `pargopy.netCDF_form`)

Python Module Index

p

- `pargopy`
- `pargopy.argodb`
- `pargopy.argotools`
- `pargopy.atlas`
- `pargopy.atlas_tools`
- `pargopy.check_decreasing_pressure`
- `pargopy.decorator`
- `pargopy.eape`
- `pargopy.general_tools`
- `pargopy.interp`
- `pargopy.interpolation_tools`
- `pargopy.manual_check_tools`
- `pargopy.mouseprofile`
- `pargopy.netCDF_form`
- `pargopy.param`
- `pargopy.research_tools`
- `pargopy.stats`
- `pargopy.task_giver`
- `pargopy.task_giver_stats`
- `pargopy.test_stats`
- `pargopy.tile`
- `pargopy.variable_selector`