# pyddem Documentation

**Romain Hugonnet, Robert McNabb**

**Oct 18, 2020**

# Contents

This is the documentation for pyddem, a set of tools for the processing of time series of DEMs.

CHAPTER 1

# Installation and Setup

The following is a (non-exhaustive) set of instructions for getting setup to run pyddem on your own machine.

As this is a (non-exhaustive) set of instructions, it may not work 100% with your particular setup. We are happy to try to provide guidance/support, **but we make no promises**.

## 1.1 Optional: Preparing a python environment

If you like, you can set up a dedicated python environment for your pyddem needs, or install into an existing environment. Setting up a dedicated environment can be handy, in case any packages required by pymmaster clash with packages in your default environment. Our personal preference is conda, but your preferences/experience may differ.

The git repository has a file, pyddem_environment.yml, which provides a working environment for pyddem and conda. Once you have conda installed, simply run:

```
conda env create -f pyddem_environment.yml
```

This will create a new conda environment, called pyddem_environment, which will have all of the various python packages necessary to run pyddem_environment. To activate the new environment, type:

```
conda activate pyddem_environment
```

And you should be ready to go. Note that you will have to activate this environment any time you wish to run pyddem, if it is not already activated in your terminal.

## 1.2 Installing from PyPI

As of October 2020, pyddem v0.1 is available via the Python Package Index (PyPI). As such, it can be installed directly using pip:

```
pip install pyddem
```

This will install version 0.1 of the package. As time allows, we will continue to update the various functions, but this will install a working version that you can use to stack and fit time series of DEMs, as well as perform various statistical analysis on the DEMs and dDEMs.

## 1.3 Installing a development version

If you would prefer to install a version that you can update yourself, you can do so as well.

First, clone the git repository:

```
git clone https://github.com/iamdonovan/pyddem.git
```

Next, use **pip** to install the scripts and python modules:

```
pip install -e pyddem
```

Note: the *-e* allows you to make changes to the code (for example, from git updates or through your own tinkering), that will then be updated within your python install. If you run *pip install* without this option, it will install a static version of the package, and any changes/updates will have to be explictly re-installed.

## 1.4 Checking the installation

Assuming that you haven't run into any errors, you should be set up and ready to go. You can verify this by running:

```
stack_dems.py -h
```

From the command line (in a non-Windows environment; *Windows instructions coming soon-ish*).

You should see the following output (or something very similar):

```
usage: fit_stack.py [-h] [-b INC_MASK] [-n NPROC] [-t TIME_RANGE TIME_RANGE]
                    [-o OUTFILE] [-c]
                    stack

Fit time series of elevation data using Gaussian Process.

positional arguments:
  stack                 NetCDF file of stacked DEMs to fit.

optional arguments:
  -h, --help            show this help message and exit
  -b INC_MASK, --inc_mask INC_MASK
                        inclusion mask. Areas outside of this mask (i.e.,
                        water) will be omitted from the fitting. [None]
  -n NPROC, --nproc NPROC
                        number of processors to use [1].
  -t TIME_RANGE TIME_RANGE, --time_range TIME_RANGE TIME_RANGE
                        Start and end dates to fit time series to (default is
                        read from input file).
  -o OUTFILE, --outfile OUTFILE
                        File to save results to. [fitted_stack.nc]
  -c, --clobber         Clobber existing outfile [False].
```

If you don't see this, feel free to ask for help by sending an e-mail, though it can also be helpful to google around for some solutions first. If you do send us an e-mail, be sure to include the specific error messages that you receive. Screenshots are also helpful.

Good luck!

CHAPTER 2

# Creating a stack of DEMs with pyddem

The basic procedure for stacking DEMs involves using either `pyddem.stack_tools.create_mmaster_stack()` or *stack_dems.py* from the command line.

The first step is to have a number of DEMs available to stack. For this example, we'll assume that these are in the same folder as we are trying to create the stack, and that they all have a filename of the form **AST*.tif**. Note that any file format that can be read by GDAL will work here, but the date of acquisition should be parse-able by **pybob.GeoImg**.

## 2.1 Using a reference DEM

If the DEMs are not already co-registered to a reference DEM, you can do that within the stacking process.

To do this, you need to have a shapefile (or any file format that can be opened with geopandas) with the footprint(st) of the reference DEM. At the moment, this is set up assuming that the tiles are stored in various 'subfolder's of a single 'path'. So, each feature of the features in the shapefile should have attributes that look something like this:

| ID | filename | subfolder | path |
|---|---|---|---|
| 0 | 59_25_1_2_30m_v3.0_reg_dem.tif | NorthAsia | ~/data/ArcticDEM/v3.0/Mosaic |

With this, `pyddem.stack_tools.create_mmaster_stack()` will create a VRT from all tiles that intersect the boundary of the stack, and use this to co-register each individual DEM in turn. You can also pass filenames for an exclusion mask (i.e., glaciers) to identify terrain that should not be included in the co-registration, as well as an inclusion mask (i.e., land).

## 2.2 Running from a script

The following example demonstrates how you can run `pyddem.stack_tools.create_mmaster_stack()` from your own script. `pyddem.stack_tools.create_mmaster_stack()` will load each of the DEMs in turn and insert it into the stack.

```
from glob import glob
from pyddem.stack_tools import create_mmaster_stack

filelist_dems = glob('AST*.tif')
fn_out = 'my_stack.nc'
fn_glacmask = '~/data/RGI/v6.0/10_rgi60_NorthAsia/10_rgi60_NorthAsia.shp'
fn_reftiles = '~/data/ArcticDEM/v3.0/Mosaic/ArcticDEM_Tiles.shp'

my_stack = create_mmaster_stack(filelist_dems,
                                res=100,
                                outfile=fn_out,
                                exc_mask=fn_glacmask,
                                mst_tiles=fn_reftiles,
                                clobber=True,
                                add_ref=True,
                                coreg=True)
```

This will create an output file called **my_stack.nc**. Note that by not specifying an extent, we automatically take the extent of the first DEM (chronologically). In the stack file, each of the **AST*.tif** DEMs are re-sampled to 100 m (**res=100**) and added to the stack in chronological order. It will co-register each of the DEMs to the reference DEM (**coreg=True**), overwrite any existing file (**clobber=True**), and add the reference DEM to the stack (**add_ref**), with any terrain falling within the RGI v6.0 glacier outlines (**exc_mask=fn_glacmask**) ignored for co-registration.

## 2.3 Running from the command line

We can do the same thing as above using the command-line script provided in **bin/stack_dems.py**:

```
stack_dems.py AST*.tif -res 100 -o my_stack.nc -exc_mask ~/data/RGI/v6.0/10_rgi60_
↪NorthAsia/10_rgi60_NorthAsia.shp
       -ref_tiles '~/data/ArcticDEM/v3.0/Mosaic/ArcticDEM_Tiles.shp' -c -do_coreg -
↪add_ref
```

Once you have a stack of DEMs, you can run various *fitting* routines on the stack to fit a time series of elevation.

CHAPTER 3

# Fitting a time series of DEMs with pyddem

Once you have created a stack of DEMs, `pyddem.fit_tools` has a number of functions to fit a time series of DEMs. The main function, which will take a stack of DEMs and produce a fitted time series, is `pyddem.fit_tools.fit_stack()`, which can also be run using the command-line tool *fit_stack.py*.

This assumes that you have first created a stack of DEMs following the instructions provided here.

## 3.1 Least Squares fitting

`pyddem.fit_tools` offers two options for fitting a linear time series of elevation: ordinary and weighted least-squares. These options work very well for areas with relatively sparse data, or where we don't see many non-linear changes (i.e., no surges or rapid thinning events).

The following two examples show how to run weighted least squares (WLS) fitting using `pyddem.fit_tools.fit_stack()`. The procedure for running ordinary least squares fitting is very much the same, simply swap out **method='wls'** for **method='ols'**.

### 3.1.1 Running WLS fitting from a script

This example shows how you can run `pyddem.fit_tools.fit_stack()` to fit a linear trend using weighted least-squares to a stack of DEMs. It will filter the raw elevations using a reference DEM and a spatial filter, fit only pixels that fall within the provided land mask using 2 cores. For more information on the various parameters, see `pyddem.fit_tools.fit_stack()`.

```
import numpy as np
from pyddem.fit_tools import fit_stack


ref_dem = 'arcticdem_mosaic_100m_v3.0.tif'
ref_dem_date = np.datetime64('2013-08-01')
fn_landmask = '~/data/Alaska_Coastline.shp',
```

(continues on next page)

(continued from previous page)

```
fit_stack('my_stack.nc',
          fn_ref_dem=ref_dem,
          ref_dem_date=ref_dem_date,
          filt_ref='min_max',
          inc_mask=fn_landmask,
          nproc=2,
          method='wls')
```

The resulting fit will be written to a number of geotiffs:

- **fit_dh.tif** - the fitted linear rate of elevation change
- **fit_interc.tif** - the intercept of the linear fit
- **fit_err.tif** - slope error of the linear fit
- **fit_nb.tif** - number of valid observations per pixel
- **fit_dmin.tif** - first date with valid observation per pixel
- **fit_dmax.tif** - final date with valid observation per pixel

### 3.1.2 Running WLS fitting from the command line

This example will do the same as above, but using the command-line tool *fit_stack.py*.

```
fit_stack.py my_stack.nc -ref_dem arcticdem_mosaic_100m_v3.0.tif -ref_dem_date 2013-
↪08-01 -f min_max
    -inc_mask ~/data/Alaska_Coastline.shp -n 2 -m wls
```

## 3.2 Gaussian Process Regression (GPR)

In addition to ordinary least squares and weighted least squares linear fitting, `pyddem.fit_tools.fit_stack()` also models a time series of elevation using gaussian process regression (GPR). This kind of fitting allows us to capture some of the nonlinear elevation changes seen over glaciers, for example where large surges have taken place, or where thinning has accelerated due to dynamic processes.

Below are two example GIFs showing the fitted annual rate of elevation change, and the fitted cumulative elevation change over Mýrdalsjökull, Iceland, between 1 January 2000 and 31 December 2019. It was created with an average of 68 observations (ASTER DEMs and ArcticDEM[1],[2] strips) per pixel.

Gaussian Process Regression takes as input a **kernel**, or a model of the variance $\sigma_h(x, y, \Delta t)^2$ of the data. Here, we

[1] ArcticDEM DEMs provided by the Polar Geospatial Center under NSF-OPP awards 1043681, 1559691, and 1542736.

[2] Porter, Claire; Morin, Paul; Howat, Ian; Noh, Myoung-Jon; Bates, Brian; Peterman, Kenneth; Keesey, Scott; Schlenk, Matthew; Gardiner, Judith; Tomko, Karen; Willis, Michael; Kelleher, Cole; Cloutier, Michael; Husby, Eric; Foga, Steven; Nakamura, Hitomi; Platson, Melisa; Wethington, Michael, Jr.; Williamson, Cathleen; Bauer, Gregory; Enos, Jeremy; Arnold, Galen; Kramer, William; Becker, Peter; Doshi, Abhijit; D'Souza, Cristelle; Cummens, Pat; Laurier, Fabien; Bojesen, Mikkel, 2018, *ArcticDEM*, https://doi.org/10.7910/DVN/OHHUKH, Harvard Dataverse, V1

have explicitly programmed a kernel that is a combination of the following kernel functions:

$$\sigma_h(x, y, \Delta t)^2 = PL(x, y, \Delta t) + ESS(\phi_{per}, \sigma_{per}^2, \Delta t) + RBF(\Delta t_{loc}, \sigma_{loc}^2, \Delta t) + RQ(\Delta t_{nl}, \sigma_{nl}, \Delta t) * PL(x, y, \Delta t) + \sigma_h(t, x, y)^2$$

with:

- PL a pairwise linear kernel, representing the long-term elevation trend of the pixel
- ESS a periodic exponential sine-squared kernel, representing the seasonality of the elevation changes
- RBF a local radial basis function kernel, showing how close elevation changes are to each other with varying time differences
- RQ a rational quadratic kernel multiplied by a linear kernel, to capture the long-term non-linear trends.
- white noise representing the average of the measurement errors at time *t*, $\sigma_h(t, x, y)^2$

When running `pyddem.fit_tools.fit_stack()` from a script, it is possible to program your own kernel, in order to model the variance of whatever elevation changes you might be looking for. See the scikit-learn docs for more information.

### 3.2.1 Running GPR fitting from a script

This example shows how you can run `pyddem.fit_tools.fit_stack()` to fit a trend using GPR to a stack of DEMs. It will filter the raw elevations using a reference DEM and both a spatial and temporal filter, fit only pixels that fall within the provided land mask using 4 cores.

Here, we will use the default kernel (see above), but running from a script or via the python interpreter, it is possible to use your own kernel (parameter **kernel=**). For more information on the other parameters, see `pyddem.fit_tools.fit_stack()`.

```
import numpy as np
from pyddem.fit_tools import fit_stack


ref_dem = 'arcticdem_mosaic_100m_v3.0.tif'
ref_dem_date = np.datetime64('2013-08-01')
fn_landmask = '~/data/Alaska_Coastline.shp',

fit_stack('my_stack.nc',
          fn_ref_dem=ref_dem,
          ref_dem_date=ref_dem_date,
          filt_ref='min_max',
          inc_mask=fn_landmask,
          nproc=2,
          method='gpr')
```

### 3.2.2 Running GPR fitting from the command line

The process for running GPR fitting using `pyddem.fit_tools.fit_stack()` works very similar to the example for WLS fitting. Note that from the command-line, it is not currently possible to use your own kernel for the fitting.

```
fit_stack.py my_stack.nc -ref_dem arcticdem_mosaic_100m_v3.0.tif -ref_dem_date 2013-
↪08-01 -f min_max
    -inc_mask ~/data/Alaska_Coastline.shp -n 2 -m gpr
```

Once the fit has run, it will create an output file called **fit.nc**, which contains variables for the fitted elevation and confidence interval (1-$\sigma$) at each time step.

That's it! The last thing to do is to open up the netCDF file and check the results. After that, you can use `pyddem.volint_tools` to calculate volume changes from your fitted elevation changes. Good luck!

## 3.3 Notes

# CHAPTER 4

# pyddem

pyddem python modules and shell scripts

## 4.1 scripts

### 4.1.1 fit_stack.py

Fit time series of elevation data to a stack of DEMs.

```
usage: fit_stack.py [-h] [-te EXTENT EXTENT EXTENT EXTENT] [-ref_dem REF_DEM]
                    [-ref_date REF_DATE] [-f FILT_REF]
                    [-filt_thresh FILT_THRESH] [-inc_mask INC_MASK]
                    [-exc_mask EXC_MASK] [-n NPROC] [-m METHOD] [-opt_gpr]
                    [-filt_ls] [-ci CI] [-t TLIM TLIM] [-ts TSTEP]
                    [-o OUTFILE] [-wf] [-c] [--merge_dates] [-d]
                    stack
```

#### Positional Arguments

| | |
|---|---|
| **stack** | NetCDF file of stacked DEMs to fit. |

#### Named Arguments

| | |
|---|---|
| **-te, --extent** | Extent over which to limit fit, given as [xmin xmax ymin ymax] |
| **-ref_dem** | Filename for input reference DEM. |
| **-ref_date** | Date of reference DEM. |
| **-f, --filt_ref** | Type of filtering to do. One of min_max, time, or both<br>Default: "min_max" |

**-filt_thresh** Maximum dh/dt from reference DEM for time filtering.

**-inc_mask** Filename of optional inclusion mask (i.e., land).

**-exc_mask** Filename of optional exclusion mask (i.e., glaciers).

**-n, --nproc** number of processors to use [1].

Default: 1

**-m, --method** Fitting method. One of Gaussian Process Regression (gpr, default),Ordinary Least Squares (ols), or Weighted Least Squares (wls)

Default: "gpr"

**-opt_gpr** Run learning optimization in the GPR Fitting [False]

Default: False

**-filt_ls** Filter least squares with a first fit [False]

Default: False

**-ci** Confidence Interval to filter least squares fit [0.99]

Default: 0.99

**-t, --tlim** Start and end years to fit time series to (default is read from input file).

**-ts, --tstep** Temporal step (in years) for fitted stack [0.25]

Default: 0.25

**-o, --outfile** File to save results to. [fit.nc]

Default: "fit.nc"

**-wf, --write_filt** Write filtered stack to file [False]

Default: False

**-c, --clobber** Clobber existing outfile [False].

Default: False

**--merge_dates** Merge any DEMs with same acquisition date [False]

Default: False

**-d, --dask_parallel** Run with dask parallel tools [False]

Default: False

### 4.1.2 stack_dems.py

Create a NetCDF stack of DEMs.

```
usage: stack_dems.py [-h] [-extent xmin xmax ymin ymax] [-res RES]
                     [-epsg EPSG] [-o OUTFILE] [-c] [-u] [-do_coreg]
                     [-r REF_TILES] [-inc_mask INC_MASK] [-exc_mask EXC_MASK]
                     [-outdir OUTDIR] [-filt_dem FILT_DEM] [-add_ref]
                     [-add_corr] [-nd LATLONTILE_NODATA] [-filt_mm_corr] [-z]
                     [-y YEAR0] [-t TMPTAG]
                     filelist [filelist ...]
```

### Positional Arguments

**filelist** List of DEM files to read and stack.

### Named Arguments

**-extent** Extent of output DEMs to write.

**-res** pixel resolution (in meters)

**-epsg** Target EPSG code. Default is taken from first (chronological) DEM.

**-o, --outfile** Output NetCDF file to create [mmaster_stack.nc]

Default: "mmaster_stack.nc"

**-c, --clobber** Overwrite any existing file [False]

Default: False

**-u, --uncert** Read stable terrain statistics for each DEM from [filename].txt

Default: False

**-do_coreg** Co-register DEMs to a reference DEM before adding to stack [False]

Default: False

**-r, --ref_tiles** Path to shapefile of reference DEM tiles to use for co-registration [None]

**-inc_mask** Filename of inclusion mask (i.e., land) for use in co-registration.

**-exc_mask** Filename of exclusion mask (i.e., glaciers) for use in co-registration.

**-outdir** Output directory for temporary files [tmp].

Default: "tmp"

**-filt_dem**

**-add_ref** Add reference DEM as a stack variable [False]

Default: False

**-add_corr** Add correlation masks as a stack variable [False]

Default: False

**-nd, --latlontile_nodata** Apply nodata for a lat/lon tile footprint to avoid overlapping and simplify xarray merging.

**-filt_mm_corr** Filter MMASTER DEM with correlation mask when stacking.

Default: False

**-z, --l1a_zipped** Use if DEMs are zipped.

Default: False

**-y, --year0** Year 0 to reference time variable to [1900]

Default: 1900.0

**-t, --tmptag** Tag to append to temporary filenames [None]

## 4.2 modules

### 4.2.1 fit_tools

pyddem.fit_tools provides tools to derive filter and interpolate DEM stacks into elevation time series

`pyddem.fit_tools.`**`constrain_var_slope_corr`**(*ds, ds_arr, ds_corr, t_vals, uncert, fn_ref_dem=None, nproc=1*)

Given a data cube of stacked DEMs, a data cube of stereo-correlations and a per-DEM vector of co-registration RMSE: estimate a median slope and compute a data cube of elevation errors. (error dependance on the slope and stereo-correlation is defined independently) #TODO: allow for a user-defined function of error?

**Parameters**

- **ds** – xarray Dataset of datacube
- **ds_arr** – Data cube of elevations
- **ds_corr** – Data cube of stereo-correlations
- **t_vals** – Time vector of data cube
- **uncert** – Vector of co-registration RMSEs of the stacked DEMs
- **fn_ref_dem** – Filename for input reference DEM
- **nproc** – Number of cores for multiprocessing [1]

**Returns** Data cube of errors, Raster of median slope values

`pyddem.fit_tools.`**`cube_to_stack`**(*ds, out_cube, y0, nice_fit_t, outfile, slope_arr=None, ci=True, clobber=False, filt_bool=False*)

Write data cube to netcdf with new time axis, confidence interval, slope, etc… #TODO: this could be simplified quite a bit…

**Parameters**

- **ds** – xarray Dataset of data cube
- **out_cube** – Data cube of time series of elevations
- **y0** – Origin of time vector
- **nice_fit_t** – Time vector relative to the origin
- **outfile** – Filename of output netcdf
- **slope_arr** – Slope raster to write as concomitant DataArray
- **ci** – Boolean to look for confidence interval in the out_cube
- **clobber** – Boolean to replace output file
- **filt_bool** – Boolean to force a boolean datacube as output (e.g., to the mask data cube of the multi-step filtering)

**Returns**

`pyddem.fit_tools.`**`dask_time_filter_ref`**(*ds, ds_arr, ref_dem, t_vals, ref_date, max_dhdt=[-50, 50], base_thresh=100.0, nproc=1*)

Dask wrapper of temporal filtering of stacked DEMs with a reference DEM: removes all elevations outside a positive/negative linear elevation trend from the reference, with a base threshold.

**Parameters**

- **ds_arr** – Data cube of elevations

- **ds_arr** – Data cube of elevations
- **ref_dem** – GeoImg of reference DEM
- **t_vals** – Time vector of data cube
- **ref_date** – Date of reference DEM
- **max_dhdt** – Maximum positive/negative elevation change rate per year from reference elevations [-50 m,50 m]
- **base_thresh** – Base vertical threshold for the temporal filter (avoid elevations close to reference) [100 m]
- **nproc** – Number of cores for multiprocessing [1]

**Returns** Filtered data cube

`pyddem.fit_tools.`**`estimate_var`**(*dh*, *mask_cube*, *arr_mask=None*, *cube_mask=None*, *nsamp=100000*)

Estimation of elevation variance from data cube of difference to reference elevations

**Parameters**

- **dh** – Data cube of difference between stack of DEMs and reference DEM
- **mask_cube** – Mask where sampling must occur
- **arr_mask** – Add a raster mask (no time dimension)
- **cube_mask** – Add a data cube mask
- **nsamp** – Number of samples to draw

**Returns** NMAD, number of samples, STD

`pyddem.fit_tools.`**`estimate_vgm`**(*ds*, *ds_arr*, *inc_mask=None*, *exc_mask=None*, *rast_mask=None*, *nsamp=10000*, *tstep=0.25*, *lag_cutoff=None*, *min_obs=8*, *nproc=1*, *pack_size=50*)

Aggregate 1D temporal variograms for multiple stack pixels: random sampling within a inclusion mask

**Parameters**

- **ds** – xarray Dataset of data cube
- **ds_arr** – Data cube of elevations
- **inc_mask** – Filename of inclusion shapefile
- **exc_mask** – Filename of exclusion shapefile
- **rast_mask** – Additional mask (boolean raster)
- **nsamp** – Number of pixels sampled in time
- **tstep** – Time interval for aggregating pairwise time lags
- **lag_cutoff** – Maximum time lag to sample
- **min_obs** – Minimum of valid observation per pixel for sampling
- **nproc** – Number of cores for multiprocessing [1]

**Returns** Lags, Variance, Variance dispersion

pyddem.fit_tools.**fit_stack**(*fn_stack, fit_extent=None, fn_ref_dem=None, ref_dem_date=None, filt_ref='min_max', time_filt_thresh=[-30, 30], inc_mask=None, exc_mask=None, nproc=1, method='gpr', opt_gpr=False, kernel=None, filt_ls=False, conf_filt_ls=0.99, tlim=None, tstep=0.25, outfile='fit.nc', write_filt=False, clobber=False, merge_date=False, dask_parallel=False*)

Given a netcdf stack of DEMs, perform filtering and temporal fitting of elevation with uncertainty propagation

**Parameters**

- **fn_stack** – Filename for input netcdf file
- **fit_extent** – extent over which to limit fit, as [xmin, xmax, ymin, ymax]
- **fn_ref_dem** – Filename for input reference DEM
- **ref_dem_date** – Date of reference DEM
- **filt_ref** – Type of filtering. One of 'min_max', 'time', or 'both'. Requires a reference DEM.
- **time_filt_thresh** – Maximum dh/dt from reference DEM for time filtering
- **inc_mask** – Optional inclusion mask. Pixels outside of the mask will not be fit.
- **exc_mask** – Optional exclusion mask. Pixels inside of the mask will not be fit.
- **nproc** – Number of cores for multiprocessing [1]
- **method** – Fitting method, currently supported: Gaussian Process Regression "gpr", Ordinary Least Squares "ols" and Weighted Least Squares "wls" ["gpr"]
- **opt_gpr** – Run learning optimization in the GPR fitting [False]
- **kernel** – Kernel
- **filt_ls** – Filter least square with a first fit [False]
- **conf_filt_ls** – Confidence interval to filter least square fit [99%]
- **tlim** – time range [t_min t_max] over which to fit. Default is full range of stack.
- **tstep** – Temporal step for fitted stack [0.25 year]
- **outfile** – Path to outfile
- **write_filt** – Write filtered stack to file
- **clobber** – Overwrite existing output files
- **merge_date** – Merge any DEMs which have the same acquisition date
- **dask_parallel** – run with dask parallel tools

**Returns**

pyddem.fit_tools.**get_dem_date**(*ds, ds_filt, t, outname*)

Extract a DEM from an elevation time series, with elevation error and time lag to the closest observation

**Parameters**

- **ds** – xarray Dataset of elevation time series
- **ds_filt** – xarray dataset of boolean data cube of valid observation used to generate the time series

;param t: Date of extraction :outname: Filename for output rasters

**Returns**

pyddem.fit_tools.**get_var_by_corr_slope_bins**(*ds*, *ds_arr*, *arr_slope*, *bins_slope*, *cube_corr*, *bins_corr*, *outfile*, *inc_mask=None*, *exc_mask=None*, *nproc=1*)

Sampling method for elevation measurement error from stacks of DEMs: binning of external variables and sampling of variance

**Parameters**

- **ds** – xarray Dataset of data cube
- **ds_arr** – Data cube of elevations
- **arr_slope** – Slope raster
- **bins_slope** – Bins to sample categories of slope
- **cube_corr** – Data cube of stereo-correlations
- **bins_corr** – Bins to sample categories of stereo-correlations
- **outfile** – Filename for output files
- **inc_mask** – Filename of inclusion shapefile for the sampling of variance
- **exc_mask** – Filename of exclusion shapefile for the sampling of variance
- **nproc** – Number of cores for multiprocessing [1]

**Returns**

pyddem.fit_tools.**gpr**(*data*, *t_vals*, *uncert*, *t_pred*, *opt=False*, *kernel=None*, *not_stable=True*, *detrend_ls=False*, *loop_detrend=False*)

Gaussian Process regression wrapper

**Parameters**

- **data** – Data cube of elevations
- **t_vals** – Time vector of data cube
- **uncert** – Data cube of errors
- **t_pred** – Time vector to predict
- **opt** – Boolean, run kernel optimization
- **kernel** – Provide kernel object
- **not_stable** – Mask of unstable terrain (to apply different kernels)
- **detrend_ls** – Boolean, remove linear trend before applying GP
- **loop_detrend** – Boolean, remove linear trend at each iteration before applying GP

**Returns** Data cube of fitted elevations, Data cube of propagated 1-sigma elevation errors, Data cube mask of valid data

pyddem.fit_tools.**isel_maskout**(*ds*, *inc_mask*)

Mask out values out of shapefile and minimize spatial extent, remove void time steps (to decrease computing time)

**Parameters** **ds** – xarray Dataset of datacube

**Returns** ds: reduced Dataset

pyddem.fit_tools.**isel_merge_dupl_dates**(*ds*)

Merge duplicate dates of a xarray dataset by taking the mean of similar dates (e.g., overlapping same-date ASTER DEMs)

**Parameters** **ds** – xarray Dataset of datacube

**Returns** ds: merged Dataset

`pyddem.fit_tools.ls`(*subarr*, *t_vals*, *err*, *weigh*, *filt_ls=False*, *conf_filt=0.99*)

Ordinary or Weighted Least-Squares wrapper

**Parameters**

- **subarr** – Data cube of elevations
- **t_vals** – Time vector of data cube
- **err** – Data cube of errors
- **weigh** – Boolean, use weighted least-squares
- **filt_ls** – Boolean, use a first CI filtering before fitting again
- **conf_filt** – Confidence interval of the filtering fit

**Returns** Raster concatenation (LS slope, LS intercept, slope 1-sigma error, number of samples, date of first sample, date of last sample), Data cube mask of valid data

`pyddem.fit_tools.make_dh_animation`(*ds*, *fn_shp=None*, *month_a_year=None*, *rates=False*, *figsize=(10, 10)*, *t0=None*, *t1=None*, *dh_max=20*, *var='z'*, *cmap='RdYlBu'*, *xlbl='easting (km)'*, *ylbl='northing (km)'*)

Generate a GIF from elevation time series

**Parameters**

- **ds** – xarray Dataset of elevation time series
- **month_a_year** – Numerical month to keep only one month per year in the animation
- **rates** – Display rates instead of cumulative change
- **figsize** – Tuple of figure size
- **t0** – Starting date
- **t1** – End date
- **dh_max** – Max scale for colorbar
- **var** – Variable to display in the dataset
- **cmap** – Colormap
- **xlbl** – xlabel for animation
- **ylbl** – ylabel for animation

**Returns** Figure, List of images and annotations for animation

`pyddem.fit_tools.manual_refine_sampl_temporal_vgm`(*fn_stack, fn_ref_dem, out_dir, filt_ref='both', max_dhdt=[-50, 50], ref_dem_date=<MagicMock name='mock()' id='140049883343280'>, inc_mask=None, gla_mask=None, nproc=1*)

Full sampling method of temporal variogram from stacks of DEMs: pre-filtering of data, binning of external variables and sampling of temporal variograms

**Parameters**

- **fn_stack** – Filename of netcdf stack of DEMs
- **fn_ref_dem** – Filename of reference DEM
- **out_dir** – Output directory
- **filt_ref** – Filtering method
- **max_dhdt** – Maximum positive/negative elevation change rate per year from reference elevations [-50 m,50 m]
- **ref_dem_date** – Date of reference DEM
- **inc_mask** – Filename of inclusion shapefile for the stack
- **gla_mask** – Filename of inclusion shapefile for the sampling of variogram
- **nproc** – Number of cores for multiprocessing [1]

**Returns**

`pyddem.fit_tools.`**`ols_matrix`**(*X*, *Y*, *conf_interv=0.68*, *conf_slope=0.68*)

Ordinary Least-Squares (matrix, for optimized processing time)

**Parameters**

- **X** – Data cube of x
- **Y** – Data cube of y
- **conf_interv** – Confidence interval for y output
- **conf_slope** – Confidence interval for WLS slope output

**Returns** Slope, Intercept, Slope CI, Y lower CB, Y upper CB

`pyddem.fit_tools.`**`prefilter_stack`**(*ds, ds_arr, fn_ref_dem, t_vals, filt_ref='min_max', ref_dem_date=None, max_dhdt=[-50, 50], nproc=1*)

Given a data cube of stacked DEMs and a reference DEM: condition a first-order spatial and temporal filtering of outliers.

**Parameters**

- **ds** – xarray Dataset of datacube
- **ds_arr** – Data cube of elevations
- **t_vals** – Time vector of data cube
- **filt_ref** – Method of filtering: spatial "min_max", temporal "time" or "both" ["min_max"]
- **fn_ref_dem** – Filename for input reference DEM
- **ref_dem_date** – Date of reference DEM
- **max_dhdt** – Maximum positive/negative elevation change rate per year from reference elevations [-50 m,50 m]
- **nproc** – Number of cores for multiprocessing [1]

**Returns** Filtered data cube

pyddem.fit_tools.**robust_wls_ref_filter_stack**(*ds, ds_arr, err_arr, t_vals, fn_ref_dem, ref_dem_date=<MagicMock name='mock()' id='140049875237072'>, max_dhdt=[-50, 50], nproc=1, cutoff_kern_size=1000, max_deltat_ref=2.0, base_thresh=100.0*)

Given a data cube of stacked DEMs, of elevation errors and a reference DEM: condition a spatial and temporal filtering of outliers based on WLS linear trends.

**Parameters**

- **ds** – xarray Dataset of data cube
- **ds_arr** – Data cube of elevations
- **err_arr** – Data cube of elevation errors
- **t_vals** – Time vector of data cube
- **fn_ref_dem** – Filename for input reference DEM
- **ref_dem_date** – Date of reference DEM
- **max_dhdt** – Maximum positive/negative elevation change rate per year from reference elevations [-50 m,50 m]
- **nproc** – Number of cores for multiprocessing [1]
- **cutoff_kern_size** – Radius size for kernel spatial filtering [1000 m]
- **max_deltat_ref** – Maximum time lag between acquisition of reference DEM and date provided [2 yr]
- **base_thresh** – Base vertical threshold for the temporal filter (avoid elevations close to reference) [100 m]

**Returns** Filtered data cube

pyddem.fit_tools.**spat_filter_ref**(*ds_arr, ref_dem, cutoff_kern_size=500, cutoff_thr=20.0, nproc=1*)

Spatial filtering of stacked DEMs with a reference DEM: removes all elevations smaller than the minimum or larger than the maximum reference elevation found within a circle of certain size, with a base threshold.

**Parameters**

- **ds_arr** – Data cube of elevations
- **ref_dem** – GeoImg of reference DEM
- **cutoff_kern_size** – Radius size for kernel spatial filtering [500 m]
- **cutoff_thr** – Base vertical threshold for the spatial filter (avoid elevations close to reference)

**Returns** ds_arr: Filtered data cube

pyddem.fit_tools.**time_filter_ref**(*ds_arr, ref_arr, t_vals, ref_date, max_dhdt=[-50, 50], base_thresh=100.0*)

**Temporal filtering of stacked DEMs with a reference DEM: removes all elevations outside a positive/negative linear** elevation trend from the reference, with a base threshold.

**Parameters**

- **ds_arr** – Data cube of elevations
- **ref_arr** – Raster of reference DEM

- **t_vals** – Time vector of data cube
- **ref_date** – Date of reference DEM
- **max_dhdt** – Maximum positive/negative elevation change rate per year from reference elevations [-50 m,50 m]
- **base_thresh** – Base vertical threshold for the temporal filter (avoid elevations close to reference) [100 m]

**Returns** ds_arr: Filtered data cube

pyddem.fit_tools.**vgm_1d**(*argsin*)

1D variogram sampling

**Parameters** **argsin** – tuple of x values, y values, lag cutoff value and interval of x sampling (for easier parallel proc)

**Returns** semivariance

pyddem.fit_tools.**vgm_1d_med**(*argsin*)

1D median variogram sampling

**Parameters** **argsin** – tuple of x values, y values, lag cutoff value and interval of x sampling (for easier parallel proc)

**Returns** semivariance

pyddem.fit_tools.**wls_matrix**(*x*, *y*, *w*, *conf_interv=0.68*, *conf_slope=0.68*)

Weighted Least-Squares (matrix, for optimized processing time)

**Parameters**

- **x** – Data cube of x
- **y** – Data cube of y
- **w** – Data cube of weights
- **conf_interv** – Confidence interval for y output
- **conf_slope** – Confidence interval for WLS slope output

**Returns** Slope, Intercept, Slope CI, Y lower CB, Y upper CB

### 4.2.2 spstats_tools

pyddem.spstats_tools provides tools to derive spatial statistics for elevation change data.

pyddem.spstats_tools.**aggregate_tinterpcorr**(*infile, cutoffs=[10000, 100000, 1000000]*)

Weighted aggregation of empirical spatial variograms between different regions, with varying time lags and sampling ranges, accounting for the number of pairwise samples drawn

**Parameters**

- **infile** – Filename of sampled variograms
- **cutoffs** – Maximum successive ranges for sampling variogram

**Returns**

pyddem.spstats_tools.**cov**(*h*, *crange*, *model='Sph'*, *psill=1.0*, *kappa=0.5*, *nugget=0*)

Compute covariance based on variogram function

**Parameters**

- **h** – Spatial lag
- **crange** – Correlation range
- **model** – Model
- **psill** – Partial sill
- **kappa** – Smoothing parameter for Exp Class
- **nugget** – Nugget

**Returns** Variogram function

pyddem.spstats_tools.**double_sum_covar**(*list_tuple_errs*, *corr_ranges*, *list_area_tot*, *list_lat*, *list_lon*, *nproc=1*)

Double sum of covariances for propagating multi-range correlated errors for disconnected spatial ensembles

**Parameters**

- **list_tuple_errs** – List of tuples of correlated errors by range, by ensemble
- **corr_ranges** – List of correlation ranges
- **list_area_tot** – List of areas of ensembles
- **list_lat** – Center latitude of ensembles
- **list_lon** – Center longitude of ensembles
- **nproc** – Number of cores to use for multiprocessing [1]

**Returns**

pyddem.spstats_tools.**get_tinterpcorr**(*df, outfile, cutoffs=[10000, 100000, 1000000], nlags=100, nproc=1, nmax=10000*)

Sample empirical spatial variograms with time lags to observation

**Parameters**

- **df** – DataFrame of differences between ICESat and GP data aggregated for all regions
- **outfile** – Filename of csv for outputs
- **cutoffs** – Maximum successive ranges for sampling variogram
- **nlags** – Number of lags to sample up to cutoff
- **nproc** – Number of cores to use for multiprocessing [1]
- **nmax** – Maximum number of observations to use for pairwise sampling (drawn randomly)

**Returns**

pyddem.spstats_tools.**neff_circ**(*area*, *list_vgm*)

Effective number of samples numerically integrated for a sum of variogram functions over an area: circular approximation of Rolstad et al. (2009)

**Parameters**

- **area** – Area
- **list_vgm** – List of variogram function to sum

**Returns** Number of effective samples

`pyddem.spstats_tools.neff_rect(area, width, crange1, psill1, model1='Sph', crange2=None, psill2=None, model2=None)`

Effective number of samples numerically integrated for a sum of 2 variogram functions over a rectangular area: rectangular approximation of Hugonnet et al. (TBD)

**Parameters**

- **area** – Area
- **width** – Width of rectangular area
- **crange1** – Correlation range of first variogram
- **psill1** – Partial sill of first variogram
- **model1** – Model of first variogram
- **crange2** – Correlation range of second variogram
- **psill2** – Partial sill of second variogram
- **model2** – Model of second variogram

**Returns** Number of effective samples

`pyddem.spstats_tools.vgm(h, crange, model='Sph', psill=1.0, kappa=0.5, nugget=0)`

Compute variogram model function (Spherical, Exponential, Gaussian or Exponential Class)

**Parameters**

- **h** – Spatial lag
- **crange** – Correlation range
- **model** – Model
- **psill** – Partial sill
- **kappa** – Smoothing parameter for Exp Class
- **nugget** – Nugget

**Returns** Variogram function

### 4.2.3 stack_tools

pymmaster.stack_tools provides tools to create stacks of DEM data, usually MMASTER DEMs.

`pyddem.stack_tools.create_crs_variable(epsg, nco=None)`

Given an EPSG code, create a CRS variable for a NetCDF file. Return collections.OrderedDict object if nco is None

**Parameters**

- **epsg** (*int*) – EPSG code for chosen CRS.
- **nco** (*netCDF4.Dataset*) – NetCDF file to create CRS variable for.

**Returns crso** NetCDF variable representing the CRS.

pyddem.stack_tools.**create_mmaster_stack**(*filelist*, *extent=None*, *res=None*, *epsg=None*, *outfile='mmaster_stack.nc'*, *clobber=False*, *uncert=False*, *coreg=False*, *ref_tiles=None*, *exc_mask=None*, *inc_mask=None*, *outdir='tmp'*, *filt_dem=None*, *add_ref=False*, *add_corr=False*, *latlontile_nodata=None*, *filt_mm_corr=False*, *l1a_zipped=False*, *y0=1900*, *tmptag=None*)

Given a list of DEM files, create a stacked NetCDF file.

**Parameters**

- **filelist** (*array-like*) – List of DEM filenames to stack.
- **extent** (*array-like*) – Spatial extent of DEMs to limit stack to [xmin, xmax, ymin, ymax].
- **res** (*float*) – Output spatial resolution of DEMs.
- **epsg** (*int*) – EPSG code of output CRS.
- **outfile** (*str*) – Filename for output NetCDF file.
- **clobber** (*bool*) – clobber existing dataset when creating NetCDF file.
- **uncert** (*bool*) – Include uncertainty variable in the output NetCDF.
- **coreg** (*bool*) – Co-register DEMs to an input DEM (given by a shapefile of tiles).
- **ref_tiles** (*str*) – Filename of input reference DEM tiles.
- **exc_mask** (*str*) – Filename of exclusion mask (i.e., glaciers) to use in co-registration
- **inc_mask** (*str*) – Filename of inclusion mask (i.e., land) to use in co-registration.
- **outdir** (*str*) – Output directory for temporary files.
- **filt_dem** (*str*) – Filename of DEM to filter elevation differences to.
- **add_ref** (*bool*) – Add reference DEM as a stack variable
- **add_corr** (*bool*) – Add correlation masks as a stack variable
- **latlontile_nodata** (*str*) – Apply nodata for a lat/lon tile footprint to avoid overlapping and simplify xarray merging
- **filt_mm_corr** (*bool*) – Filter MMASTER DEM with correlation mask out of mmaster_tools when stacking (disk space),
- **l1a_zipped** (*bool*) – Use if files have been zipped to save on space.
- **y0** (*float*) – Year 0 to reference NetCDF time variable to.
- **tmptag** (*str*) – string to append to temporary files.

**Returns nco** NetCDF Dataset of stacked DEMs.

pyddem.stack_tools.**create_nc**(*img*, *outfile='mmaster_stack.nc'*, *clobber=False*, *t0=None*)

Create a NetCDF dataset with x, y, and time variables.

**Parameters**

- **img** (*pybob.GeoImg*) – Input GeoImg to base shape of x, y variables on.
- **outfile** (*str*) – Filename for output NetCDF file.
- **clobber** (*bool*) – clobber existing dataset when creating NetCDF file.
- **t0** (*np.datetime64*) – Initial time for creation of time variable. Default is 01 Jan 1900.

**Returns nco, to, xo, yo** output NetCDF dataset, time, x, and y variables.

`pyddem.stack_tools.`**`get_footprints`**(*filelist*, *proj4=None*)

Get a list of footprints, given a filelist of DEMs.

**Parameters**

- **`filelist`** (*`array-like`*) – List of DEMs to create footprints for.
- **`proj4`** (*`str, int`*) – proj4 representation of output CRS. If None, the CRS is chosen from the first DEM loaded. Can also supply an EPSG code as an integer.

**Returns fprints, this_crs** A list of footprints and a proj4 string (or dict) representing the output CRS.

`pyddem.stack_tools.`**`make_geoimg`**(*ds*, *band=0*, *var='z'*)

Create a GeoImg representation of a given band from an xarray dataset.

**Parameters**

- **`ds`** (*`xarray.Dataset`*) – xarray dataset to read shape, extent, CRS values from.
- **`band`** (*`int`*) – band number of xarray dataset to use
- **`var`** (*`string`*) – variable of xarray dataset to use

**Returns geoimg** GeoImg representation of the given band.

### 4.2.4 tdem_tools

pymmaster.tdem_tools provides tools to post-process DEM stacks: volume integration over specific outlines, comparison to point data, spatial aggregation…

`pyddem.tdem_tools.`**`add_base_to_int`**(*df*, *fn_base*, *reg*)

Use "base" RGI file to check whether all glaciers are covered before aggregation, and to remove nominal glaciers and CL2 glacier estimates

**Parameters**

- **`df`** – DataFrame of integrated volume time series
- **`fn_base`** – Filename of "base" RGI file
- **`reg`** – RGI region number: 20 corresponds to Alaska and WNA combined, 21 corresponds to HMA combined (necessary for some uncertainty calculations as those are contiguous)

**Returns** DataFrame of integrated volume time series with nominal, CL2 glaciers removed and not-covered glaciers added with nodata

`pyddem.tdem_tools.`**`aggregate_all_to_period`**(*df*, *list_tlim=None*, *mult_ann=None*, *fn_tarea=None*, *frac_area=None*, *list_tag=None*)

Temporal integration of volume change time series into rates for a specific period and error propagation For regions, we can account for time-varying areas by approximated per-region annual areas (Zemp et al. (2019))

**Parameters**

- **`df`** – DataFrame of volume time series
- **`list_tlim`** – List of tuples or early and late date to integrate over
- **`mult_ann`** – Ignore tuples, integrate over all successive multi-annual periods of this length (e.g., four successive 5-year rates)
- **`fn_tarea`** – Filename of DataFrame with temporally-varying areas

- **frac_area** – Use only a corresponding fraction of the regional area to estimate time-varying areas (e.g., subset of Greenland, or other)
- **list_tag** – Tag a naming to each integration period

**Returns** DataFrame of volume change rates for all successive periods fitting the year length

pyddem.tdem_tools.**aggregate_df_int_time**(*infile*, *tlim=None*, *rate=False*)

Temporal integration over any period, choice between cumulative or rate

**Parameters**

- **infile** – Filename of DataFrame with volume change time series
- **tlim** – Time limits for integration
- **rate** – Boolean, add rates

**Returns**

pyddem.tdem_tools.**aggregate_indep_regions**(*df_p*)

Aggregate regions with independent assumptions for uncertainty propagation :param df_p: :return:

pyddem.tdem_tools.**aggregate_int_to_all**(*df*, *nproc=1*, *get_corr_err=True*)

Aggregate spatially per-glaciers volume changes time series and propagate errors

**Parameters**

- **df** – DataFrame of integrated volume time series
- **nproc** – Number of cores used for multiprocessing [1]
- **get_corr_err** – Derive correlated error (long): currently forced to annual only, because seasonal biases are not accounted for and can be complex

**Returns** DataFrame of aggregated volume time series

pyddem.tdem_tools.**comp_stacks_icebridge**(*list_fn_stack*, *fn_icebridge*, *gla_mask=None*, *inc_mask=None*, *exc_mask=None*, *nproc=1*, *read_filt=False*)

Compare IceBridge data with elevation time series

**Parameters**

- **list_fn_stack** – List of filename of netCDF to compare to
- **fn_icebridge** – Filename of IceBridge post-processed into .csv
- **gla_mask** – Filename of glacier shapefile
- **inc_mask** – Filename of inclusion shapefile (e.g. land)
- **exc_mask** – Filename of exclusion shapeifle (e.g. ocean)
- **nproc** – Number of cores used for multiprocessing [1]
- **read_filt** – Boolean, read boolean data cube of valid observation

**Returns** Arranged output data (see_comp_stacks_icesat)

pyddem.tdem_tools.**comp_stacks_icesat**(*list_fn_stack*, *fn_icesat*, *gla_mask=None*, *inc_mask=None*, *exc_mask=None*, *nproc=1*, *read_filt=False*, *shift=None*)

Compare ICESat data with elevation time series

**Parameters**

- **list_fn_stack** – List of filename of netCDF to compare to

- **fn_icesat** – Filename of ICESat HDF5 file
- **gla_mask** – Filename of glacier shapefile
- **inc_mask** – Filename of inclusion shapefile (e.g. land)
- **exc_mask** – Filename of exclusion shapeifle (e.g. ocean)
- **nproc** – Number of cores used for multiprocessing [1]
- **read_filt** – Boolean, read boolean data cube of valid observation

**Returns** Arranged output data (see_comp_stacks_icesat)

pyddem.tdem_tools.**df_all_base_to_tile**(*list_fn_int_base*, *fn_base*, *tile_size=1*, *nproc=1*, *sort_tw=True*)

Integrate all per-glacier volume change time series on a global tiling of certain size, for all possible 1-,2-,4-,5-,10- and 20-year periods, with error propagation

**Parameters**

- **list_fn_int_base** – List of filename of DataFrame of per-glacier volume change time series with base RGIID info added
- **fn_base** – Filename of "base" RGI file
- **tile_size** – Size of tiling in lat/lon degrees
- **nproc** – Number of cores used for multiprocessing [1]
- **sort_tw** – Sort between marine-terminating and land-terminating glaciers

**Returns**

pyddem.tdem_tools.**df_int_to_base**(*infile*, *fn_base=None*)

Wrapper for adding base RGI info to DataFrame per glacier to all regions separately and write to .csv

**Parameters**

- **infile** – DataFrame of integrated volume time series
- **fn_base** – Filename of "base" RGI file

**Returns**

pyddem.tdem_tools.**df_int_to_reg**(*infile*, *nproc=1*)

Wrapper for integrating volume change time series per region (based on RGI number) and write to .csv

**Parameters**

- **infile** – DataFrame of integrated volume time series
- **nproc** – Number of cores used for multiprocessing [1]

**Returns**

pyddem.tdem_tools.**df_region_to_multann**(*infile_reg*, *fn_tarea=None*, *frac_area=None*)

Wrapper for temporal integration of regional volume change time series for all possible 1-,2-,4-,5-,10- and 20-year periods and write to .csv

**Parameters**

- **infile_reg** – DataFrame of regional, integrated volume time series
- **fn_tarea** – Filename of DataFrame with temporally-varying areas
- **frac_area** – Use only a corresponding fraction of the regional area to estimate time-varying areas (e.g., subset of Greenland, or other)

**Returns**

pyddem.tdem_tools.**get_base_df_inventory**(*dir_shp*, *outfile*)

Create a "base" dataframe which contains characteristics of RGIIds features (this is done separately to avoid duplicating that info in GBs of data into elevation time series, as those are 2D DataFrames) :param dir_shp: Directory containing RGI shapefiles :param outfile: Output .csv file containing dataframe

pyddem.tdem_tools.**get_dt_closest_valid**(*ds_filt*, *dates*)

Derive data cube of time lag to closest valid observation based on the date vector of the time series and a boolean data cube of valid observation at original dates :param ds_filt: Boolean data cube of valid observations :param dates: Time vector of time series

**Returns** Data cube indexed to time vector with time lags to observation, Data cube with count of valid observation between time index, Data cube with yearly count of valid observation at the first yearly time index

pyddem.tdem_tools.**hypsocheat_postproc_stacks_tvol**(*list_fn_stack*, *fn_shp*, *feat_id='RGIId'*, *tlim=None*, *nproc=1*, *outfile='int_dh.csv'*)

Hypsometric cheat volume integration of elevation time series The objective of the "cheat" is to highly decrease processing time by allowing to combine parts of shapefile features (e.g., glaciers) which are projected in different georeferencing systems in the stacks (projected in UTM), without having to merge the stacks into the same projection Because integration is hypsometric (based on reference elevation of the glacier), we do not need the information of spatial structure of the glacier

**Parameters**

- **list_fn_stack** – List of filenames of netCDF elevation time series
- **fn_shp** – Filename of shapefile
- **feat_id** – Feature name of interest
- **tlim** – Time limits for integration
- **nproc** – Number of cores used for multiprocessing [1]
- **outfile** – Filename of output .csv

**Returns**

pyddem.tdem_tools.**icesat_comp_wrapper**(*argsin*)

Multiprocessing wrapper to compare ICESat data with elevation time series

**Parameters** **argsin** – Tuple of arranged data (see comp_stacks_icesat)

**Returns** Arranged output data (see_comp_stacks_icesat)

pyddem.tdem_tools.**int_dc**(*dc*, *mask*, ***kwargs*)

Wrapper to for classic integration of elevation time series into volume change time series

**Parameters**

- **dc** – xarray Dataset (subset)
- **mask** – Raster boolean mask

**Returns** DataFrame of integrated elevation change

pyddem.tdem_tools.**inters_feat_shp_stacks**(*fn_shp*, *list_fn_stack*, *feat_field_name*)

Find all intersecting stacks for features of a shapefile, for a list of netCDF stacks (tiles for example)

**Parameters**

- **fn_shp** – Filename of shapefile

- **list_fn_stack** – List of filenames of netCDF stacks
- **feat_field_name** – Name of shapefile feature to sort by (e.g., RGIIDs for RGI glaciers)

**Returns** List of all features found, Corresponding list of intersecting stacks per feature

pyddem.tdem_tools.**saga_aspect_slope_curvature**(*dem_in*, *topo_out*, *method_nb=5*)

Derive maximum curvature using SAGA (command line)

**Parameters**

- **dem_in** – dem input
- **topo_out** – raster out (3 bands: aspect, slope, curvature)
- **method_nb** – algorithm used, see function description

**Returns**

requirement: SAGA 2.X with X>3

#ref:http://www.saga-gis.org/saga_tool_doc/2.2.3/ta_morphometry_0.html

aspect, slope, curvature methods methods number [0] maximum slope (Travis et al. 1975) [1] maximum triangle slope (Tarboton 1997) [2] least squares fitted plane (Horn 1981, Costa-Cabral & Burgess 1996) [3] 6 parameter 2nd order polynom (Evans 1979) [4] 6 parameter 2nd order polynom (Heerdegen & Beran 1982) [5] 6 parameter 2nd order polynom (Bauer, Rohdenburg, Bork 1985) [6] 9 parameter 2nd order polynom (Zevenbergen & Thorne 1987) [7] 10 parameter 3rd order polynom (Haralick 1983)

unit slope 0=rad, 1=deg, 2=percent unit aspect 0=rad, 1=deg

pyddem.tdem_tools.**sel_dc**(*ds*, *tlim*, *mask*)

Subset datacube in space and time with a spatial mask and time limits

**Parameters**

- **ds** – xarray Dataset
- **tlim** – Time limits
- **mask** – Raster boolean mask

**Returns** Subset Dataset, Subset boolean mask, Spatial index slices (to identify subset position in the original datacube, for instance)

pyddem.tdem_tools.**sel_int_hypsocheat**(*argsin*)

Multiprocessing wrapper by shapefile feature for "hypsometric cheat" volume integration of elevation time series

**Parameters** **argsin** – Tuple with: list of filename of netCDF stacks, shapefile parameters (see hypsocheat_postproc_stacks_tvol), time limits, processing index

**Returns** Three dataframes of volume change (see hypso_dc in volint_tools.py)

### 4.2.5 vector_tools

pymmaster.vector_tools provides tools to manipulate tilings (naming, extents) and vectors (rasterize, buffer, transform, ...)

pyddem.vector_tools.**SRTMGL1_naming_to_latlon**(*tile_name*)

Convert widely used 1x1° tile naming convention to lat, lon (originally SRTMGL1)

**Parameters** **tile_name** – naming convention of southwestern 1x1° corner

**Returns** lat, lon of southwestern corner

pyddem.vector_tools.**coord_trans**(*is_src_wkt*, *proj_src*, *is_tgt_wkt*, *proj_tgt*)

Create a OGR Transform object to reproject between different coordinate systems

**Parameters**

- **is_src_wkt** – Boolean, is the provided projection source in WKT, if not it needs to be EPSG code (int)
- **proj_src** – WKT or EPSG code for projection source
- **is_tgt_wkt** – Boolean, is the provided projection target in WKT, if not it needs to be EPSG code (int)
- **proj_tgt** – WKT or EPSG code for projection target

**Returns** OGR Transform object

pyddem.vector_tools.**create_mem_raster_on_geoimg**(*geoimg*)

Create raster in memory with the same extent that a GeoImg

**Parameters** **geoimg** – GeoImg

**Returns** GDAL DataSet

pyddem.vector_tools.**create_mem_shp**(*geom*, *srs*, *layer_name='NA'*, *layer_type=<MagicMock id='140049875311640'>*, *field_id='ID'*, *field_val='1'*, *field_type=<MagicMock id='140049875282184'>*)

Create shapefile object in memory

**Parameters**

- **geom** – Geometry object
- **srs** – Projection
- **layer_name** – Shapefile layer name
- **layer_type** – Shapefile layer type (polygon by default)
- **field_id** – Basic field id (field necessary by default)
- **field_val** – Basic field value
- **field_type** – Basic field type

**Returns** OGR DataSet

pyddem.vector_tools.**epsg_from_utm**(*utm_zone*)

Get EPSG code from UTM zone naming

**Parameters** **utm_zone** – UTM zone naming

**Returns** epsg

pyddem.vector_tools.**extent_from_poly**(*poly*)

Get extent of polygon

**Parameters** **poly** – OGR polygon

**Returns** extent

pyddem.vector_tools.**extent_rast**(*raster_in*)

Get raster extent

**Parameters** **raster_in** – Filename of raster

**Returns** extent

pyddem.vector_tools.**extract_odl_astL1A**(*fn*)
Extract data from ASTER L1A metadata (old ODL format)

**Parameters** **fn** – Filename of L1A .met file

**Returns** Arrays and DataFrames with metadata

pyddem.vector_tools.**geoimg_mask_on_feat_shp_ds**(*shp_ds*, *geoimg*, *layer_name='NA'*, *feat_id='ID'*, *feat_val='1'*, ***kwargs*)
Create mask of a shapefile feature on a GeoImg

**Parameters**

- **shp_ds** – GDAL DataSet of input shapefile
- **geoimg** – GeoImg
- **layer_name** – Layer name for shapefile
- **feat_id** – Feature of interest
- **feat_val** – Value of the feature to be masked

**Returns** GDAL DataSet

pyddem.vector_tools.**get_buffered_area_ratio**(*geom*, *proj_in*, *buff*)
Get buffered area ratio (for error estimation): reproject in a meter system centered on polygon, buffer of a certain distance, and look

**Parameters**

- **geom** – OGR polygon
- **proj_in** – Projection source
- **buff** – Buffer distance

**Returns** Area buffered to area ratio in percent, Area of polygon

pyddem.vector_tools.**inters_list_poly_with_poly**(*list_poly*, *poly*)
Find intersection between a list of polygon and another polygon

**Parameters**

- **list_poly** – List of OGR polygons
- **poly** – OGR polygon

**Returns** List of intersecting OGR polygons

pyddem.vector_tools.**l1astrip_polygon**(*l1a_subdir*)
Compute the merged polygon of stitched ASTER L1A granules from the metadata of the original granules

**Parameters** **l1a_subdir** – Directory containing several L1A .met files

**Returns** OGR polygon

pyddem.vector_tools.**latlon_to_SRTMGL1_naming**(*lat*, *lon*)
Convert lat, lon to widely used 1x1° tile naming (originally SRTMGL1)

**Parameters**

- **lat** – latitude of southwestern corner
- **lon** – longitude of southwestern corner

**Returns** tile naming

pyddem.vector_tools.**latlon_to_UTM**(*lat*, *lon*)
Get UTM zone, and corresponding EPSG given lat,lon coordinates

**Parameters**

- **lat** – latitude
- **lon** – longitude

**Returns** epsg, UTM zone naming

pyddem.vector_tools.**latlontile_nodatamask**(*geoimg*, *tile_name*)
Create mask on the extent of a 1x1° tile (used to avoid sampling twice neighbouring tiles in final calculations, although those need to have overlapping pixels to avoid resampling issues)

**Parameters**

- **geoimg** – GeoImg
- **tile_name** – Tile naming

**Returns** Mask

pyddem.vector_tools.**list_shp_field_inters_extent**(*fn_shp*, *field_name*, *extent*, *proj_ext*)
List features intersecting an extent in a shapefile

**Parameters**

- **fn_shp** – Filename of shapefile
- **field_name** – Field name to list for the features
- **extent** – extent
- **proj_ext** – projection of extent

**Returns** List of field feature values (e.g., RGIIds)

pyddem.vector_tools.**niceextent_utm_latlontile**(*tile_name*, *utm_zone*, *gsd*)
Create overlapping tile extents to avoid resampling effects at the edges

**Parameters**

- **tile_name** – tile naming
- **utm_zone** – UTM zone naming
- **gsd** – ground sampling distance (pixel size)

**Returns** extent

pyddem.vector_tools.**poly_from_coords**(*list_coord*)
Create polygon from list of coordinates

**Parameters** **list_coord** – List of tuples, need to repeat the first/last in order to close polygon

**Returns** OGR polygon

pyddem.vector_tools.**poly_from_extent**(*extent*)
Create polygon based on extent

**Parameters** **extent** – extent (xmin, ymin, xmax, ymax)

**Returns** OGR polygon

pyddem.vector_tools.**poly_utm_latlontile**(*tile_name*, *utm_zone*)
Create polygon of a 1x1° lat, lon tile projected in a specific UTM zone

**Parameters**

- **tile_name** – tile naming
- **utm_zone** – UTM zone naming

**Returns** OGR polygon

`pyddem.vector_tools.utm_from_epsg(epsg)`

Get UTM zone naming from EPSG code

**Parameters** **epsg** – EPSG code

**Returns** UTM zone naming

### 4.2.6 volint_tools

pymmaster.volint_tools provides tools to integrate elevation change into volume time series, adapted from McNabb et al. (2019).

`pyddem.volint_tools.double_sum_covar_hypso(tot_err, slope_bin, elev_bin, area_tot, crange, kernel=<function kernel_exclass>)`

1D hypsometric propagation of correlated error through double sum of covariance

**Parameters**

- **tot_err** – Error per hypsometric bin
- **slope_bin** – Slope per bin (to assess horizontal distance)
- **elev_bin** – Reference elevation per bin
- **area_tot** – Area per bin
- **crange** – Correlation range
- **kernel** – Form of kernel [Exponential Class]

**Returns** y interpolated, error propagated, error of interpolation

`pyddem.volint_tools.hypso_dc(dh_dc, err_dc, ref_elev, dt, tvals, mask, gsd, slope=None, bin_type='fixed', bin_val=100.0, filt_bin='5NMAD', method='linear')`

Hypsometric spatial integration of elevation time series with NMAD filtering and with propagation of correlated errors depending on time lag to closest observation currently written for the "hypsometric cheat" volume integration routine (see tdem tools), the x/y dimension are thus flattened in 1D and we use only hypsometry for integration for now, the function defining the long-range sum of variogram is defined directly in the script based empirical sampling and least-squares fits to ICESat done separately

**Parameters**

- **dh_dc** – Data cube of elevation change (flattened in x/y)
- **err_dc** – Data cube of elevation change errors (flattened in x/y)
- **ref_elev** – Raster of reference elevation (flattened in x/y)
- **dt** – Data cube of time lag to closest observation (flattened in x/y)
- **tvals** – Date vector
- **mask** – Mask of valid values (flattened in x/y)
- **gsd** – Ground sampling distance in meters
- **slope** – Slope, only used to propagate correlated hypometric errors
- **bin_type** – Type of elevation binning (percentage of elevation range, or flat value)

- **bin_val** – Elevation bin flat value (if that is used)
- **filt_bin** – Filtering of outliers per elevation bin [currently 5NMAD of full time range]
- **method** – Interpolation/extrapolation method for void elevatio bins

**Returns** Three dataframes with volume change time series

pyddem.volint_tools.**interp_linear**(*xp*, *yp*, *errp*, *acc_y*, *loo=False*)

Piece-wise linear interpolation with linear extrapolation on the edges and basic error propagation (where to interpolate is given by NaN values in the vectors)

**Parameters**

- **xp** – x
- **yp** – y
- **errp** – error
- **acc_y** – prior knowledge of maximum acceleration for assessing possible interpolation error
- **loo** – perform a first leave-one-out interpolation, remove data outliers (outside error bars)

**Returns** y interpolated, error propagated, error of interpolation

pyddem.volint_tools.**interp_lowess**(*xp*, *yp*, *errp*, *acc_y*, *crange*, *kernel=<function kernel_exclass>*)

LOWESS interpolation with error propagation (where to interpolate is given by NaN values in the vectors)

**Parameters**

- **xp** – x
- **yp** – y
- **errp** – error
- **acc_y** – prior knowledge of maximum acceleration for assessing possible interpolation error
- **rang** – Range of kernel used for local linear regression
- **kernel** – Form of kernel [Exponential Class]

**Returns** y interpolated, error propagated, error of interpolation

pyddem.volint_tools.**lowess_homemade_kern**(*x*, *y*, *w*, *a1*, *kernel='Exp'*)

#inspired by: https://xavierbourretsicotte.github.io/loess.html homebaked lowess with variogram kernel + heteroscedasticity of observations with error

:param x:x :param y:y :param w: heteroscedastic weights (inverse of variance) :param a1: range of the kernel (in variogram terms) :param kernel: kernel function :return:

CHAPTER 5

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## p

# Index

## H



## I



## L



## M



## N



## O



## P



## R



## S



## T



## U



## V



## W